INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
SECCIÓN DE ESTUDIOS DE POSTGRADO E INVESTIGACIÓN
"CULHUACAN"

"DESARROLLO DE LOS ELEMENTOS DEL SISTEMA DE DIAGNÓSTICO DE UNA TURBINA DE GAS"

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS DE INGENIERÍA EN SISTEMAS ENERGÉTICOS

PRESENTA:

ING. JESÚS MARTÍNEZ LÓPEZ

ASESOR DE TESIS:

DR. IGOR LOBODA

México, D.F. Mayo del 2011

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D. F. siendo las 13:00 horas del día 09 del mes de mayo del 2011 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de SEPI-ESIME-CULHUACAN para examinar la tesis titulada:

"Desarrollo de los Elementos del Sistema de Diagnóstico de una Turbina de Gas"

Presentada por el alumno:

Martínez	**López**	**Jesús**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: A 0 9 0 2 8 8

aspirante de:

MAESTRÍA EN CIENCIAS DE INGENIERÍA EN SISTEMAS ENERGÉTICOS

Después de intercambiar opiniones, los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director(a) de tesis

Dr. Igor Loboda

Dr. Miguel Ángel Olivares Robles

Dr. Gabriel Sánchez Pérez

Dr. Georgiy Polupan

Dr. Pedro Guevara López

INSTITUTO POLITECNICO NACIONAL ESTADOS UNIDOS MEXICANOS S.E.P. SECCION DE ESTUDIOS DE POSGRADO E INVESTIGACION ESIME CULHUACAN

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Gonzalo Isaac Duchen Sánchez

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de _México, Distrito Federal_ el día _13_ del mes _Mayo_ del año _2011_, el (la) que suscribe _C. Jesús Martínez López_ alumno (a) del Programa de _Maestría en Ciencias de Ingeniería de Sistemas Energéticos_ con número de registro _A090288_, adscrito a _Sección de Estudios de Postgrado e Investigación ESIME-CULHUACAN_, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de _Dr. Igor Loboda_ y cede los derechos del trabajo intitulado _Desarrollo de los Elementos del Sistema de Diagnóstico de una Turbina de Gas_, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección _loboda@calmecac.esimecu.ipn.mx_. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ing. Jesús Martínez López
Nombre y firma

Agradecimientos

A mis padres: Cucando Martínez Díaz y Susana López Calleja

Gracias a su infinito amor, apoyo, confianza y comprensión han hecho de mí una persona triunfadora, por ser las personas ejemplares y mas maravillosas que me alientan a seguir adelante.

A mis abuelos: Jesús López Reyes y María del Refugio Calleja Portas

Gracias a sus consejos, regaños y alientos he aprendido a agradecer cada momento que dios me da y sobre todo a ser feliz. Por ser unos seres humanos tan rectos y admirables.

A mi hermano: Cucando Martínez López

Gracias por ser parte de mi vida, por soportarme en los momentos difíciles y compartir los gratos momentos.

Al amor de mi vida: Roxana Carolina Cruz García

Gracias por el apoyo incondicional, por compartir tantos momentos buenos y malos que me han hecho una persona madura. Por querer compartir una vida, Te amo, hermosa.

A mi asesor de tesis: Dr. Igor Loboda

Gracias por los conocimientos compartidos, por ser ejemplo de trabajo, y por su paciencia durante el desarrollo y culminación de esta tesis.

Al Instituto Politécnico Nacional

Por haberme brindado la oportunidad de ser parte de una gran institución, así como darme una formación profesional. Gracias por los recursos invertidos en mi formación.

Al CONACYT

Por los apoyos económicos recibidos para la culminación de esta tesis.

¡Gracias a Dios! *sobre todas las cosas por ponerme en este camino y dotarme de bendiciones. Para tí mi respeto y devoción.*

Jesús Martínez López

ÍNDICE

Justificación

Los motores de turbina de gas se han presentado un desarrollo muy importante para el país. Del mismo modo ha sufrido avances tecnológicos muy importantes. Sin embargo por ser maquinas que poseen una gran variedad y cantidad de componentes, esto hace que su confiabilidad se vea afectada, además de que sus elementos son demasiados costos y que una falla grave puede causar un desastre mayor. Para contrarrestar este efecto, al mismo tiempo se han desarrollado y mejorado sistemas de monitores para incrementar su confiabilidad.

El anisáis del conducto de flujo es una técnica muy utilizada para poder hacer un diagnóstico de las condiciones de las turbinas de gas. Esta técnica comprende principalmente tres etapas; el monitoreo de las turbina de gas, el diagnóstico detallado (reconocimiento de las fallas de motor) y el pronóstico de la vida del motor. A estas tres etapas le antecede una preliminar que es de suma importancia, el cálculo de las desviaciones.

Al sistema de monitoreo con el afán de hacerlo más eficiente se han aplicado las herramientas de las Redes Neuronales Artificiales (ANN). Las ANNs han tenido un crecimiento muy importante en diversas aéreas y se han manifestado en el diagnóstico paramétrico de turbinas de gas como una herramienta muy poderosa. El Perceptrón Multicapa ha sido la configuración mas implementada en lo que se refiere a este tema.

El objetivo del presente trabajo de tesis es desarrollar los elementos del sistema de diagnóstico de una turbina de gas, es decir crear los algoritmos y software del reconocimiento de fallas del conducto de flujo y el cálculo de las desviaciones. Este software se crea primero en el lenguaje de programación Matlab y después en el lenguaje Fortran. La última resulta en un modelo ejecutable que funciona independientemente de cualquier ambiente. La idea consiste en la creación de un algoritmo listo para integrarse en un sistema real de monitoreo de turbinas de gas.

Resumen

En los años recientes el uso de las turbinas de gas ha tenido un crecimiento enorme, debido a que son parte importante en el desarrollo económico del país. Son consideradas como sistemas complejos de ingeniería debido a que continuamente sufren cambios tecnológicamente más sofisticados. Lo que la hace más susceptibles a que se pueda afectar su confiabilidad, esto se puede observar en el incremento de accidentes dentro de la industria. Las fallas mecánicas ocupan un porcentaje elevado de las causas de estos accidentes. Para asegurar una confiabilidad necesaria de turbinas de gas se han implementado técnicas y herramientas de diagnóstico para tratar de tener un mejor control de las condiciones de las Turbinas de Gas.

Es muy amplia la variedad de técnicas de monitoreo de las condiciones de maquinas que se han mejorado en los últimos años como son: monitoreo de las vibraciones, análisis de los residuos del aceite, inspección visual, monitoreo de ruido y monitoreo de la contaminación del ambiente. Estas permiten determinar las condiciones de operación de la maquinaria en la industria. Siendo estas también empleadas para el monitoreo de las condiciones de turbinas de gas.

Los sistemas automatizados de monitoreo de las turbinas de gas están basados en la medición y recolección de variables. Por lo cual tales sistemas no necesitan el paro y desarmado del motor. Es decir que estos sistemas operan en tiempo real y proveen un análisis de diagnóstico en línea. Dentro del sistema de monitoreo existe una técnica específica para las turbinas de gas, llamada análisis del conducto de flujo (GPA - Gas Path Analysis). El análisis del conducto de flujo ha sido elegido como un enfoque representativo para el diagnóstico de las turbinas de gas.

El monitoreo de las condiciones de la turbina de gas basado en los parámetros medidos del conducto de flujo del motor, incluye las etapas de detección de problemas, identificación de fallas y pronostico, además de requerir de una operación preliminar de desviaciones computadas entre mediciones y un motor de referencia.

El presente trabajo nos enfocamos principalmente al desarrollo de los elementos del sistema de diagnostico de una turbina de gas, aplicadas a dos etapas del monitoreo, en la etapa preliminar que es las que se encarga de calcular las desviaciones y la etapa dos que se encarga de clasificar o identificar las fallas. Para ambos casos emplearemos una herramienta muy importante y utilizada en los años más recientes, las Redes Neuronales Artificiales (ANN – Artificial Neural Network). Que han demostrado ser herramienta muy compleja y muy útil en lo que se refiere al diagnóstico de turbinas de gas, considerando el crecimiento que han sufrido desde su aparición.

Aunque Matlab tiene varias funciones para realizar diferentes redes neuronales, en esta tesis hemos desarrollado nuestros propios algoritmos. Después ellos mismos hemos programado en Fortran. Como resultados tenemos módulos ejecutables listos para incluir en un sistema real de monitoreo.

Abstract

In recent years the use of gas turbines has grown tremendously, because they are an important part in national economic development. They are considered like complex engineering systems due to continual technological changes. Thus, gas turbine can be affected; this can be seen in the increase of accidents in industry. Mechanical faults occupy a high percentage of the causes of these accidents. To ensure a necessary reliability of gas turbines, many techniques and diagnostics tools have been implemented. They ensure control of the condition of gas turbines.

A wide variety of condition monitoring techniques of machines has been improved in the recent years such as: vibration monitoring, oil debris, visual inspection, monitoring noise, and monitoring environmental pollution. These help better determine the machinery operation condition in industry. These also are used for gas turbine condition monitoring.

Automated systems of gas turbine monitoring are based on parameters measurement and recording. For this operation such systems need not shutdown and disassembly of the engine. Therefore these systems work in real time and provide a diagnostic analysis online. Into this monitoring system there is a specific technique for gas turbines, called Gas Path Analysis (GPA). The gas path analysis has been chosen like a representative approach for gas turbine diagnosis.

Gas turbine condition monitoring, based on parameters measurement of engine gas path, includes stages of problem detection, fault identification and prognosis. It also requires one preliminary stage of computing deviations between measurements and a baseline.

In the present work we focus mainly on development of gas turbine diagnostic system elements, for two stages of monitoring, the preliminary stage of computing deviations and the second stage of faults identification. For both cases an important tool of Artificial Neuronal Networks (ANN) is widely used in recent years. The ANN`s have proved to be very complex tool and very useful for gas turbine diagnosis, considering the growth that has been suffer since they appeared.

Although Matlab have several functions to perform different neuronal network, in this thesis we have developed our own algorithms. Then one of them was realized in Fortran After. As a result, we have executable program modules ready to include in a real monitoring system.

Capítulo 1.- El Monitoreo de los Sistemas de Ingeniería (Maquinaria)

1.1. El monitoreo y su necesidad en la industria

1.1.1. Introducción

Mantenimiento es la dirección de control, ejecución y calidad de todas las actividades, las cuales aseguraremos que su nivel óptimo de disponibilidad y total rendimiento en la planta sean alcanzados, de tal forma que cumpla con los objetivos de la misma.

La productividad manufacturera es ampliamente influenciada por:

1) Grandes disposiciones de recursos físicos
2) Mejoras en la calidad de los recursos humanos
3) Mejoras en los métodos y técnicas de manufactura

Se ha demostrado que el uso apropiado del Monitoreo de las Condiciones y la administración de mantenimiento pueden ofrecer mejoras significantes en la eficiencia y directamente en la rentabilidad.

El objetivo del Monitoreo de las Condiciones y diagnóstico de Maquinaria es la estandarización de procedimientos, procesos y equipos requeridos únicamente relacionados a las actividades técnicas del monitoreo de las condiciones y diagnóstico de maquinaria en el que seleccione parámetros físicos asociados con una maquina operando estando

periódicamente o continuamente censada, medida y registrada con el fin de reducir, analizar, comparar y mostrar los datos e informar lo obtenido y por ultimo usar este resultado provisional para soportar las decisiones relacionadas a la operación y mantenimiento de la maquinaria [1].

Se ha encontrado que en la industria no ha existido aplicación alguna de planes de mantenimiento adecuados, lo cual ha provocado grandes pérdidas tanto monetarias como grandes riesgos a los operadores o trabajadores. Entre estas y otras razones se puede decir que la función del mantenimiento tiene un impacto muy grande en la eficiencia de cualquier industria.

Muchas naciones avanzadas y emergentes están justamente haciendo mejor uso de tecnología de mantenimiento existente. El monitoreo de las condiciones toman un papel muy importante en lograr mantener la continua prosperidad de la industria, debido a esto no es posible decir que los accidentes ocurrirán, si los podemos evitar.

Alrededor del mundo se han visto grandes desastres, ocasionados por la falta de un sistema de mantenimiento apropiado, los cuales han llevado consigo grandes pérdidas tanto del medio ambiente como decesos de trabajadores. Es aquí cuando nos damos cuenta de la necesidad de contar, implementar y mantener un sistema de monitoreo de las condiciones adecuado y una administración correcta del mismo.

Es muy importante conocer cada vez más sobre la tendencia del comportamiento de nuestra maquinaria, es decir tener todos los parámetros necesarios para poder determinar su comportamiento real de trabajo. El monitoreo, control de vibraciones y ruido en maquinaria de altas velocidades, sistemas hidráulicos y neumáticos, ventiladores, etc. están ganando la atención.

En la última década los fabricantes de la gran variedad de maquinaria se han visto forzados a ofrecer sus equipos con una amplia gama de paquetes encaminados a un mejor control en el sentido de monitoreo, en lugar de solo ofrecer el equipo básico o estándar. Además los clientes están siendo muy demandantes en que los equipos cumplan y tengan todos los paquetes de seguridad pertinentes, así como su manual de entrenamiento, programación y mantenimiento.

Para la industria la información es muy importante debido a que con esta ellos pueden mejorar desde sus materiales de construcción, procesos de manufactura y hasta su mercadotecnia. Es por ello que conforme ha avanzado el tiempo se ha tenido que recurriera a nuevas tecnologías como es el uso de una PC que conforme a pasado el tiempo también ha sufrido significativos cambios enfocados a mejorar la calidad de la información que hemos de analizar. Así mejorando nuestro monitoreo de las condiciones.

El monitoreo de las condiciones no solo se ha desarrollado por la seguridad de la maquinaria. Es evidente que lo que se trata de hacer es evitar los riesgos pero sin embargo también se tiene que ver que los recursos humanos estén bien implementados. Es decir que estén las personas que tengan que estar en el lugar que deban estar, esto implica tanto conocimientos, habilidades y actitud; esto para un mejor desempeño de todo el conjunto.

1.1.2. Monitoreo de las Condiciones en la industria

Cualquier planta, equipo o proceso puede llegar a fallar al operar durante su vida útil por uno o varios factores en combinación, intensidad, cambios de rango y duración. En nuestra planta, equipo o proceso tenemos parámetros como vibraciones, ruido, calentamiento, enfriamiento, corrosión, humedad, etc. las cuales nos ayudan a determinar las condiciones de nuestra planta, equipo o proceso. Para esto se cuenta con una amplia gama de sensores que detectan y monitorean rápidamente señales de cambio eléctrico, mecánico, neumático y la información del sistema de falla ayudando a diagnosticar y estableciendo un efectivo mantenimiento.

El mantenimiento basado en las condiciones provee una evaluación basada en la obtención, conteo y la correcta interpretación de los datos de la maquinaria dirigido a un plazo de tiempo y requerimientos anteriores al mantenimiento para predecir las fallas. El monitoreo de las condiciones se puede definir como: la continua o periódica medición e interpretación de un parámetro determinando la necesidad de mantenimiento.

Las herramientas y las técnicas utilizadas en la obtención de datos en el campo del monitoreo de las condiciones, así como la tecnología empleada para el diagnóstico entre otras son:

Tecnología de los sensores

La selección del mejor o el correcto sensor es la llave de un buen monitoreo debido a que estos son los encargados de medir, así como de enviar las señales a las áreas requeridas. El tiempo en que estas envían las señales es muy importante. Por lo anterior los sensores necesitan ser; muy rápidos, exactos y deben de contar con la capacidad para auto regularse en operación real. Además deben de contar con un diseño simple pero con una gran capacidad de resistencia en su construcción, confiabilidad alta y que no produzca incrementos en el sistema complejo, competitivo, etc. [2].

En el mercado existe una gran variedad de sensores efectivos en el monitoreo y control:

1) Transductores mecánicos como: desplazamiento, transductores de localización o posición, transductores de esfuerzo, transductores de momento, transductores de presión y transductores de flujo.
2) Transductores ópticos como: foto detectores, pirómetros, láser, fibras ópticas, transductores de estado sólido (LEDs, LCDS, CCDs).
3) Transductores térmicos como: termopares, termistores.
4) Sensores ambientales como: espectrómetros, indicadores d PH, monitores de contaminación de aire/agua/sólidos.

Monitoreo de las Condiciones y tecnología del diagnóstico

El Monitoreo de las Condiciones y la tecnología del diagnóstico es aplicada a un gran número de industrias como la petroquímica, metalurgia, ingeniería nuclear, aeronáutica, plantas generadoras. Debemos considerarlo como una área multidisciplinaria debido a que debemos aplicar una serie de enfoques, por lo cual no podemos aislarla y ponerla como una sola disciplina separada de la competitividad económica.

El funcionamiento del monitoreo del sistema de control requiere dos precondiciones para asegurar una aplicación satisfactoria:

1) El sistema se debe mantener estable en condiciones normales de operación, esto lo confirmamos con la estabilidad de los parámetros bajo observación.
2) Tomamos las mediciones de los instrumentos manual o automáticamente.

Como ya sabemos estas condiciones son ya conocidas y cualquier cambio en cualquier parámetro hará un monitoreo más sencillo, así como el cambio de la tendencia. Con tal monitoreo podemos prevenir una falla mayor.

Los métodos del monitoreo cubren las siguientes aéreas:

a) Medimos la variación en/y/ò de los valores absolutos de las salidas del sistema en términos de calidad y cantidad.
b) Medimos la relación de entrada y salidas del sistema.
c) Medimos y simultáneamente comparamos dos parámetros de salida con los parámetros de condición de operación estándar.

Contar con un buen proceso de control nos garantiza que podemos contar con una efectiva operación de nuestra planta de procesos. Sin embargo conforme avanza el tiempo la tecnología se adelanta demasiado por ello es necesario implementar las nuevas técnicas de control que van surgiendo para de esta forma poder afrontar los problemas que surjan.

Una de las últimas tecnologías es la aplicación de las Redes Neuronales Artificiales (ANNs), las cuales estas pensadas y diseñadas como las neuronas humanas. Una neurona es como un switch con un cierto número de salidas. Si las salidas de la neurona exceden, los valores del umbral pre-establecido, la neurona se activa autónomamente. Cada neurona puede tener un número establecido de entradas cada una conectada con otra neurona de la red. La neurona se puede apagar si considera que las entradas son insuficientes.

1.2. La importancia del monitoreo de las condiciones

Como ya se ha mencionado la implementación de sensores es muy importante, esto con la finalidad de detectar las condiciones o los cambios de funcionamiento de nuestra planta, proceso o maquinaria, para posteriormente verificar si las decisiones que se toman se pueden llevar a cabo o no y así al mismo tiempo tenemos que ver que el impacto financiero de nuestras decisiones no se incremente.

Para esto primero debemos detectar las fallas implementando el uso de sensores que se escuchen, que se sientan y si es posible que se puedan oler. En segundo lugar debemos de analizar las decisiones que se puedan tomar ya que estas pueden ser las que hagan la diferencia. Es muy importante este punto debido a que depende directamente del punto de vista de cada persona y esto puede significar la pronta localización de un riesgo futuro, para poder ser valorada con anticipación y ver el valor de este riesgo [3].

Monitoreo de las condiciones es la selección de los parámetros a medir de nuestra planta, proceso o maquinaria, dentro de los cuales verifiquemos los cambios del buen funcionamiento o condiciones de nuestro equipo, o algún otro valor que cambie significativamente. Por lo tanto nosotros tenemos que regular los parámetros monitoreados

y observar los cambios así de esta forma cada cambio observado estará detectando lo que podemos hacer con mayor detalle en el análisis de los parámetros medidos encaminado a la predicción de cuál es el problema.

Un análisis del monitoreo de las condiciones tienen forzosamente dos partes que se deben de cumplir, por un lado las cuestiones técnicas de las mediciones y el análisis, y por otro lado las cuestiones humana y organizacionales.

El Monitoreo de las condiciones tienen que llegar a ser parte importante de las estrategias operacionales de cualquier empresa, las cual se debe de reflejar en la gran producción de salida de la empresa así como en la reducción de presupuesto para mantenimiento. Sin embargo, etas cuestiones organizacionales son procesos humanos y son el resultado de las necesidades del crecimiento y la optimización en todos los niveles de la empresa [4].

Existen una variedad de tipos de mediciones comúnmente utilizadas para determinar el monitoreo de las condiciones y los requerimientos técnicos son muy variados para cada una de ellas. Y se ha llegado que el desarrollo de las herramientas del monitoreo de las condiciones sea por especialización, es decir que cada empresas se ha desarrollado y se ha especializado en una área en específico, y así obteniéndose que varias tecnologías se puedan emplear en el sistemas del monitoreo de las condiciones [5]. Esto se ha visto mas como una necesidad el tener que integrar las diferentes tecnologías, como pueden ser:

– *Vibración:* El análisis de vibraciones es la técnica más comúnmente empleada y la más tangible. Además si consideramos que en general todo tipo de maquinaria vibra y además es una medición fácil de hacer y de interpretar. Un transductor puede fácilmente estar ligado a la base de una maquina, el cual está siempre acompañado de un fuerte imán o un conector rápido, por lo cual la obtención de datos es rápida y efectiva.

– *Análisis de lubricación:* El análisis de lubricante es la segunda prueba técnica más común dentro del sistema de monitorea. Esta técnica nos ayuda más que nada en ver si el lubricante contiene partículas que puedan causar problemas.

– *Termografía:* Este método se realiza con el uso de una cama de imágenes térmicas la cual obtiene el mapa de distribución de las temperaturas a través de los paneles eléctricos los cuales se observan por unos puertos calientes de unas conexiones sueltas. Para la interpretación de los datos se requiere un pequeño entrenamiento relacionado con otras técnicas.

– *Ultrasonido: Emisiones Acústicas, Vibraciones de alta frecuencia*. Estas técnicas son utilizadas para la detección de fricción y de ráfagas de energía generadas por los defectos en los rodamientos, ya que este es donde los rodamientos pueden ser impactados a cierta velocidad creando choques y picos de energía.

Conforme hemos avanzado han surgido nuevas tecnologías y cada vez más se desarrolla maquinaria mas sofisticada, que sin duda ayudan a mejorar significativamente la recolección de datos para el sistema de mantenimiento predictivo. El uso de la computadora ha sido sin duda una gran ayuda para obtener, organizar e interpretar eficientemente los datos obtenidos. Como se ha discutido hasta este punto, la información es muy importante debido a que puede permitir manejar o dirigir de mejor manera la empresa o fabrica, por lo cual la aplicación del software puede tener un enorme impacto.

Desde las primeras aplicaciones del mantenimiento predictivo, así como sus diversas formas de aplicarlas, se ha visto que para los inversionistas es primordial o de vital importancia que sus ganancias se incrementen rápidamente y esto no ha sido posible, debido a las diferentes maneras en que las estrategias del mantenimiento predictivo pueden pagarse. Es decir el énfasis recae justo en el punto en que las inversiones se recuperen rápidamente, dejando a un lado o mejor dicho olvidando los demás beneficios que esto nos ofrece. El problema es que la justificación esta siempre hecha en los beneficios, los cuales no resultan en corto tiempo. Las ganancias son mayores y fácilmente realizables y en el caso conveniente puede ser preparado para la inversión. El punto para un mantenimiento predictivo es que este te da un regreso en inversiones.

Los principales ejecutores de las herramientas del monitoreo de las condiciones llegan a ser expertos a través de la experiencia que se va adquiriendo y es con esto que la consistencia en la obtención de los datos se incrementa y la habilidad para detectar problemas en la maquinaria puede llegar a ser mejor. Debido a esto podemos predecir algún riesgo de algún problema y así atacarlo antes que la falla suceda. Un programa de mantenimiento predictivo puede darnos las actividades correspondientes en las cuales nos debemos enfocar para que no se realicen con anterioridad.

Es importante reconocer al mantenimiento de las condiciones no como una técnica, si no como una unión de gente y organización para que realmente pueda dar los resultados deseados. El conocimiento de esta herramienta se puede dirigir hacia el mantenimiento predictivo entregando grandes efectos, pero también es muy importante que cuente con una gran organización, así como una gran actitud, tan buena como su tecnología para llegar a obtener los objetivos deseados.

El Mantenimiento Preventivo ocupa una posición muy fuerte en la administración estratégica para la maquinaria de una gran mayoría de compañías, por lo cual es recomendado como la manera hacia un mejoramiento para esas maquinarias de las compañías que operan bajo condiciones críticas. También se adapta perfectamente la administración, ingeniería y técnicas del mantenimiento. Lo importante de todo esto es el beneficio de poder tener la mejor información disponible y más rápida con el menor esfuerzo así como poder hacer la toma de decisiones sobre los problemas de la maquinaria con disponibilidad y fiabilidad.

Los cambios en el mantenimiento preventivo han ocurrido principalmente en lo organizacional y la técnica. En lo organizacional se ha mejorado en la tendencia para que se produzca ciclo de mejora continua y la progresiva eliminación de los enlaces débiles en la cadena. En cuanto a los cambios, en la mejora de la técnica, la tendencia llega en pasos, como los nuevos métodos que son elaborados o como las nuevas tecnologías que han sido convertidas en herramientas comercialmente disponibles.

Para los cambios de lo organizacional lo más importante e indiscutiblemente la última tendencia es el costo y el ahorro logrado, y por lo tanto el retorno de la inversión se estará logrando por el equipo. Los grandes cambios de la técnica son el desarrollo de la radiofonía y otros métodos de transmisión de señales del transductor, datos e información computacional de salida desde la maquinaria hacia la gente que necesita la información, eliminando el uso de caros cableados. Una muy importante consideración que afecta el

costo de la tecnología del mantenimiento preventivo es que el volumen de los instrumentos especializados que son producidos cada año sigue una tendencia a cada vez más pequeña.

Cada vez más la tendencia está siendo dirigida a utilizar hardware y sistemas operativos utilizados en volúmenes de mercado como una PCs portátil para proveer el funcionamiento requerido de los instrumentos. Hoy en día hay algunos grupos que están viendo por la siguiente generación de herramientas autómatas de diagnóstico, pero mientras las ideas están teniendo seguimiento, no hay una respuesta firme a este viejo problema.

1.3. Monitoreo de vibraciones

La vibración se define como un movimiento periódico sobre una posición de equilibrio. Como sabemos cualquier sistema que posee dentro de sus características ciertas propiedades de inercia y rigidez oscilan bajo su posición de equilibrio cuando esta es perturbada por una fuerza extraña al sistema. La vibración normalmente surge por el rozamiento entre dos superficies en contacto o movimiento entre rodamientos o engranes, etc.

La duración así como la magnitud de la vibración dependen principalmente del grado de amortiguación que posee el material así como de la relación entre la fuerza de excitación y la respuesta del sistema. La vibración que se genera es transmitida a través de la estructura o por medio de un componente a otro. Otro aspecto importante de la vibración es la resonancia. Este estado ocurre cuando la frecuencia natural del sistema corresponde a la frecuencia de excitación. Cuando la vibración reacciona de forma inesperada, es decir pasa los niveles aceptables, esta puede interrumpir o destruir los procesos, la cual puede desencadenar una serie de fallas lo cual no sería deseable.

Causas	Característica de la Frecuencia de Vibración	Observaciones
Desbalanceo	1/revolución (1/R)	La vibración debe ser alta cuando la velocidad de rotación coincide con una velocidad citica del sistema del rotor. El cambio de la fase de vibración significativa ocurrirá cuando pasa atreves de la velocidad crítica. Cuando una mezcla de niveles de velocidad son constantes.
Des-alineamiento	2/R, 1/R, 3/R	Varios tipos y causas para des-alineación. Los componentes de vibración axiales pueden ser tan significantes como los componentes radiales, particularmente para sistemas orientados.
Desgate de rodamientos – *Rodamientos de película de aceite*	40-50% de la primer velocidad critica del rotor, o en una velocidad critica del rotor	Claro excesivo debido a desgaste, daño o un mal ensamble pueden ser causas de desestabilidad de la película de aceite del rodamiento forzado para actuar en el rotor. Niveles de vibración inestables y pueden rápidamente encontrar elevadas magnitudes.
– *Rodamientos de elementos rodantes*	Varias frecuencia, particularmente elevados ordenes de frecuencia de velocidad de funcionamiento	La vibración tiende a localizarse en las regiones de defecto de los rodamientos, las vibraciones registradas usualmente inestables y se incrementan con el tiempo.
Rigidez disimétrica	2/R	Picos de vibración cuando 2/R estímulos coincide con una velocidad de rotación crítica. En un rotor de velocidad fija, los niveles de vibración constante. Surcos de compensación se utilizan en maquinarias largas para minimizar los estímulos.

Rotor curveado	1/R, 2/R	Si el rotor se curveo cerca del acoplamiento, elevados frecuencia de vibración axial 2/R es observada. En una velocidad fija los niveles de vibración del rotor constante.
Componentes aflojados	1/R y armónicos de frecuencia de velocidad en funcionamiento	Lo niveles de vibración pueden ser erráticos e inconsistente entre ciclos sucesivos de arranque y paro. A veces frecuencias sub armónico también se observa.
Cojinetes excéntricos o no circulares	1/R y para cojinetes no circulares en armónicos de frecuencia de velocidad en operación	Los niveles de vibración pueden ser anormales o excesivos en velocidades de rotación bajas también como en velocidades criticas de rotación. En una velocidad de rotación fija, niveles de vibración constante.
Disimetría térmica	1/R	Causado por ventilación de rotor no uniforme, bobinas eléctricas en corto y opresión no uniforme de partes. El rotor causa el arco con características de vibración como desbalanceo.
Defectos de sellos	Frecuencia demasiado alta correspondiente a múltiplos de la frecuencia de paso de sellos	La detección requiere transductores con alta respuesta de frecuencia.
Resonancia	En frecuencia de excitación como cuando la velocidad del rotor iguala una frecuencia natural en el rotor/soporte del sistema	La magnificación de la vibración en cada velocidad de resonancia de maquinaria y largos cambios de ángulo de fase en la 1/R responde como el rotor pasa atreves de la velocidad crítica. El rotor desbalanceado es el estimulo más común con el cual puede producir resonancia del sistema no-rotativo de la maquinaria. En maquinas eléctricas, el otro estimulo más grande es 2/R de la fuerza electromagnética que el rotor induce en el estator.
Estímulos eléctricos		
– *Estado estacionario*	La vibración responde más comúnmente al primero y segundo armónico de frecuencia del sistema eléctrico.	Produce vibración torsional del rotor y vibración de flexión de los alabes en la turbina. Vibraciones puede llegar a ser excesivas debido a desbalanceas actuales anormales en la red eléctrica conectados a la maquinaria o de transitorios eléctricos largos.
– *Transitorios*	La vibraciones responden en la frecuencia natural de torsión del sistema del rotor y armónicos de frecuencia de sistemas eléctricos	

Tabla 1.1 Causas más comunes de vibración de maquinaria y resultados de la característica

Causas	*Amplitud*	*Frecuencia*	*Observaciones*
Desbalanceo	Proporcional al desbalanceo	1 x rev/min	Para
Des-alineamiento	Largo en direcciones axiales: 50% o más de vibración radial	Usualmente 1 x rev/min; a veces 2 y 3 x rev/min	Mejor se encuentra por la aparición de vibraciones axiales largas; indicadores de línea de prueba deberían verificar el resultado. Des-alineación puedes ser confirmada por el análisis de fase.
Rodamientos de elementos rodantes	Inestable a menos que conste de picos de energía	Varios tiempos de rev/min pero probablemente no un evento múltiple de rev/min	En muchos casos varias amplitudes de vibraciones altas en un numero de frecuencia elevada. La vibración usualmente no se transmite a otra parte de la maquina, los rodamientos defectuosos son por lo tanto usualmente los más cercanos a la parte donde ocurre la amplitudes más altas. El uso de instalaciones de picos de energía provee lecturas confiables constantes.
Cojinetes, cojinetes limpiado		½ o 1 x rev/min	
Rotación del aceite	Vertical inusualmente alta comparada con horizontal	Un poco menos (5% o 8%) que ½ x rev/min	

Lubricación		Frecuencia elevada no como para ser múltiples de rev/min	
Jornadas excéntricas	Usualmente no son largas	1 x rev/min	En engranes, mayores vibraciones en línea con centros de engranes. En motor o generador eléctrico, las vibraciones desaparecen cuando la potencia es desconectada, En bombas o escape de gas, intento para balancear.
Defectos de engranes	Baja	Muy altas: diente de engrane x rev/min	
Holguras mecánicas	Amplitudes de 2 x rev/min mayores que ½ en 1 x rev/min	2 x rev/min	Pernos de montaje sueltos o tornillos que sujetan hacia abajo.
Correas defectuosas unidas	Erráticas o pulsaciones	1, 2, 3 y 4 x cinturón rev/min	Puede resultar en amplitudes mayores en dirección paralela a la tensión del cinturón.
Defectos eléctricos	Desaparecen cuando la potencia es apagada	1 x rev/min o 1 o 2 x frecuencia síncrona	Causada por defectos eléctricos resultando en campo magnético desigual.
Fuerzas aerodinámicas o hidrodinámicas		1 x rev/min o numero de alabes en el rotos x rev/min	Cavitación, recirculación o flujo turbulento indicado por vibración aleatorio probablemente sobre un rango de frecuencia ancho.
Fuerza reciprocante		1, 2 y mayores ordenes x rev/min	Golpe de cocción del motor ICE tiene que ser conocido; vibraciones mayores en otros múltiples de rev/min pueden ocurrir con reflejar el numero de disparos impulsos/revoluciones.

Tabla 1.2 Guía para las causas de las vibraciones

Con un buen monitoreo de vibraciones para una planta de producción o maquinaria y además con un análisis de las señales de vibración es posible evitar grandes gastos y evitables rupturas. Algunas de las causas más comunes de vibración en maquinaria, así como la característica del resultado de vibración se muestran en la tabla 1. La frecuencia es la llave para identificar la fuente de vibración de la maquinaria. La tabla 2 muestra una guía de las principales causas de vibración. Algunos de los parámetros más importantes dentro del análisis de las vibraciones, así como algunas técnicas útiles para el mismo son enlistados, además que las vibraciones pueden ser medidas en término de los siguientes parámetros:

a) Medimos la variación en/y/ò de los valores absolutos de las salidas del sistema en términos de calidad y cantidad.
b) Medimos la relación de entrada y salidas del sistema.
c) Medimos y simultáneamente comparamos dos parámetros de salida con los parámetros de condición de operación estándar.
d) Desplazamiento (m)
e) Velocidad (m/s)
f) Aceleración (m/s^2)
g) Frecuencia (Hz)
h) Banda ancha (hZ)
i) Picos de energía (gSE)
j) Densidad del espectro de potencia
k) Frecuencia (ms)
l) Valor máximo
m) Cuadrático medio (RMS)
n) Factor de cresta (CF)

o) Media aritmética (AM)
p) Media geométrica (GM)
q) Desviación estándar (SD)
r) Curtosis (K)
s) Asimetría
t) Fase

Como sabemos los parámetros de desplazamiento, velocidad y aceleración están relacionadas unos con otros. Por otro lado los picos de energía son causados por defectos en las superficies de los rodamientos o elemento en movimiento, para esta técnica medimos los picos de aceleración en un rango de 5-50 KHz, en unidades gSE. Debido al movimiento relativo de los rodamientos sobre su interior, se generan pulsos de alta frecuencia en forma de pulsos de ultrasonido en rangos de microsegundos.

Las señales de vibraciones para cualquier maquinaria son señales complejas, las cuales se componen de una mezcla se curvas sinusoidales con diferente amplitud, frecuencia y diferentes fases, todas ellas relacionadas con la velocidad de rotación. Para analizar la distribución de frecuencia o espectro es necesario transformar la señal de vibración de un tiempo de dominio dentro de una frecuencia de dominio. Esto se puede realizar con la ayuda de la rápida transformada de Fourier (FFT). Por otro lado la Densidad del espectro de potencia expresa la energía contenida en la señal con una banda ancha dada.

Un Cepstrum es un anagrama de un espectro, mientras que el análisis de Cepstral es el espectro de un espectro. Ahora si la transformada de Fourier para un logaritmo de la densidad media cuadrada es tomada, obtendremos que es denominado tanto como el cepstrum como una función de la variable independiente de frecuencia teniendo una dimensión de tiempo. Es importante señalar que el uso del análisis de cepstral es que el periodo armónico siempre puede ser detectado cuando este envuelto por un nivel alto de ruido.

La utilidad de cepstrum es su habilidad para predecir el espectro mientras resta intensidad al paso de la transmisión. Este análisis trabaja perfectamente para señales puras las cuales no se obscurecen por otra señal, pero sin embargo más de dos señales pueden ser examinadas en el dominio de tiempo, por otro lado cepstral puede no resolver el problema o identificar los componentes individuales.

Usar el espectro de Walsh tiene consigo una ventaja muy importante y es que reduce el tiempo de cálculo por sobre una magnitud de orden. Otra herramienta muy útil es el Conjunto de Análisis Tiempo- Frecuencia (siglas en ingles JTFA), la cual analiza una señal en dominio de tiempo y dominio de frecuencia simultáneamente. Como desventaja de esta herramienta es que solo algunas aplicaciones están disponibles para JTFA, el más común es el espectro de corto tiempo de la transformada de Fourier (siglas en ingles STFT), pero este espectro tiene también sus limitaciones.

Las limitaciones del STFT son sustituidas por el nuevo espectrograma Gabor este aplica dos técnicas conocidas como la expansión de Gabor y el pseudo Wigner – Ville de distribución (siglas en ingles PWVD). Esta herramienta computarizando es extremadamente eficiente mientras que por otro lado tenemos el espectro de Gabor, el cual

puede computarizar más rápido arriba de 6 tiempo en comparación con el STFT, además que el espectro Gabor produce mejor señal al ruido en la intensidad de salida de impresión.

La Kurtosis estática es el cuarto momento de una probabilidad de distribución y se puede calcular usando la ecuación (1.1).

$$K = \frac{(\sum {A_K}^4)/(n-1)}{(\sum {A_K}^4)^2/(n-1)} \tag{1.1}$$

Un buen rodamiento debe indicar un valor de Kourtosis de 3 (± 8%). Como hemos visto, los rodamientos llegan a desgastarse, lo cual hace que el número y la magnitud de los pulsos en la señal de vibración se incrementes. Esta técnica es muy sensible a los picos de la distribución del valor de la amplitud en una señal de vibración. Los altos valore de K son un índice de daños en los rodamientos. Esta técnica es muy adecuada para la tendencia del monitoreo además nos da una fiable y efectiva manera de detectar fallas en los elementos móviles.

El método de choque de pulsos del monitoreo de las condiciones de los rodamientos no mide sus vibraciones, pero si la ola de choques o pulsos del cual se originan. Esta ola de choques es causado por el impacto de dos cuerpos. Este impacto establece las olas de choque con magnitud dependiendo a la velocidad del impacto, resultando en olas de compresión o pulsos de choches a través de la caja de rodamientos, esta es detectada por un transductor, el cual es mecánicamente sintonizado. Así una frecuencia muy discreta es utilizada, además un filtro eléctricamente está encargado de asegurar que la frecuencia natural de 32 KHz del transductor se mantenga y no pueda ser influenciado por otra fuente. Las condiciones de los rodamientos se miden por el crecimiento en el nivel del valor de choque (SVL) de un buen rodamiento. Un crecimiento hasta 35 dB indica desarrollo del daño, 35-50 dB indica daño visible y 50-60 dB indica peligro de falla. Este método es tan sensible que es capaz de medir la película de lubricante entre los elementos móviles.

Los siguientes sensores son empleados en el campo de las condiciones de monitores de vibración:

a) Sensor de desplazamiento como son LVDT, RVDT, switches de proximidad, extensómetros, etc.
b) Sensor de velocidad
c) Sensor de aceleración como son acelerómetros
d) Sensores de fuerza, presión y flujo

El sensor comúnmente utilizado en el campo del monitoreo de las condiciones de vibraciones es del tipo que integra piezoeléctrico y acelerómetro el cual convierte las señales de aceleración en señales de velocidad o desplazamiento. Normalmente los acelerómetros se montan en lugares previamente bien seleccionado en una estructura fija de la maquinaria. Además es muy importante tomar en cuenta que los sensores deben estar montados en una posición la cual ofrezca o garantice la máxima respuesta a las vibraciones así como también nos pueda dar una rápida indicación de las probables fallas.

La técnica del análisis del elemento finito está siendo utilizada para determinar el modo de las formas de las vibraciones de varios elementos estructurales. El análisis de la forma de

deflexión está evaluando como una maquina es movida bajo las condiciones de la resonancia. La excentricidad del pico a pico o la medida excéntrica del lento giro está siendo particularmente aplicado a grandes turbinas de vapor y algunas turbinas de gas industriales.

El análisis de vibraciones también está siendo cuidadosamente analizado utilizando los planos Bode y Polar los cuales desplazan amplitudes y ángulos de fase de la vibración fundamental. Del plano Bode el factor de amplificación o magnitud Q es calculado dividiendo la velocidad de resonancia por la diferencia entre la velocidad de la maquina al punto de amplitud de –3 dB. Un valor elevado de Q significara una pequeña amortiguación del sistema mientras un valor pequeño de Q generalmente implica una alta amortiguación del sistema. Por otro lado el plano polar es normalmente utilizado para hacer una identificación precisa de la velocidad de balance de resonancia de la flecha, resonancia estructural, modo de la forma del rotor y el factor de amplificación del rotor/sistema de rodamientos.

Como podremos darnos cuenta, es necesario monitorear las condiciones de las vibraciones de las maquinas rotatoria durante su operación de aceleración o desaceleración, además como la velocidad en las maquinas fluctúa la frecuencia del espectro, los componentes importantes también se alteran. La información requerida está trazada en un plano de espectro de 3 dimensiones, también conocido como cascada o plano de cascada. Las ventajas de este método son que las frecuencias generadas por la maquinaria pueden ser producidas bajo condiciones normales de operación y la frecuencia de resonancia puede ser rápidamente diferenciada de la velocidad dependiente de la frecuencia.

1.4. Tecnología del diagnóstico de la caja de engranes

Desafortunadamente la estimación del monitoreo de las condiciones y el pronóstico ha permanecido por mucho tiempo y sigue siendo una ciencia inexacta. Sin embargo se han desarrollado grandes avances así como los diversos enfoques al monitoreo de la caja de engranes de acuerdo al tipo y por supuesto a las aplicaciones de estas.

Las estrategias del monitoreo incluyen una serie de actividades o procesos los cuales son muy importantes para desarrollar esta parte de la mejor manera. Estas son hasta el fallo (siglas en ingles RTF), tiempo basado en la revisión (necesario o no), revisión basada en la fiabilidad de la predicción y/o en la observación de las caja de engranes (fiabilidad centrada en el monitores (siglas en ingles RCM)), monitoreo periódico, monitoreo continuo y la evaluación del estado (Mantenimiento Basado en las Condiciones (siglas en ingles CBM)), etc.

En las elecciones de la estrategia para cualquier aplicación dada hay que hacerse dos preguntas previas las cuales son:

1) ¿Cuál es la consecuencia del desgaste o fallas en la caja de engranes?
2) ¿Cuál es el costo para cada enfoque a monitorear?

Para poder dar respuesta a estas preguntas, como podemos observar la primera envuelve ciertos términos como son: el costo, seguridad y el impacto del producto o proceso del desgaste o las fallas de las cajas de engranes. Como en todo el paso principal es el producir

un análisis realista de las consecuencias. Esto por un lado, por otro lado llegamos a la parte en la que se ve reflejados los efectos, los cuales incluyen por ejemplo la seguridad de los riesgos, degradación de la calidad de los productos, paridad de producción, y el costo del daño colateral, así con el costo de reparación o remplazo de las fallas de las cajas de engranes.

El desgaste y las fallas de las cajas de engranes normalmente son el resultado del desgaste y la fallas de los elementos principales como pueden ser la flecha, los engranes y baleros. Estas partes están sujetas normalmente a partes metálicas y el contacto es metal con metal durante el desgaste y también durante la operación por lo cual nos resulta un prolongado desgaste hasta el avance de la falla, por lo regular estas partes también están sujetas a cargas cíclicas las cuales nos generan fatiga en la superficie y en la estructura así hasta llegar a la ruptura.

Algunas de estas podrían ser las causas del daño en las cajas de engranes, sin embargo existen otras probables fallas como son: el abuso de operación, la falta de mantenimiento preventivo y crecimiento de problemas. Estos problemas con frecuencia se pueden observar o detectar con el incremento de ruido y vibración, generación anormal de tamaño y cantidades de desecho de metal e incremento de temperatura

La pérdida de potencia en la caja de engranes es un evidente síntoma de que hay una operación no adecuada de la maquinaria, está perdida de potencia se ve reflejada con la aparición de vibraciones y calentamiento de la misma. Por otro lado tenemos que la energía de vibración está relacionada con la red de engranes y las propiedades de los soportes de los rodamientos a la carga y la velocidad de rotación de la flecha de la caja de engranes. Ahora por otro lado sabemos que la resultante de la energía de vibración generada es transferida a la cubierta de la caja de engranes a través de los soportes de los rodamientos.

El análisis tecnológico de la detección de fallas en las cajas de engranes esta normalmente relacionado con los cambios de la característica de vibración de esta. Estos cambios pueden ser reflejados en cambios de amplitud de vibración, cambios en la amplitud de ciertas vibraciones de frecuencia, o en la forma de la señal de vibración.

Hora la vibración puede ser detectada como un caso de desplazamiento, velocidad o aceleración de los engranes que existen en la detección local. La vibración que se genera ya sea por una red de engranes o rodamiento o cualquiera de sus componentes es alterada por la característica de transmisión de vibración de los soportes de los rodamientos, y la función de transferencia mecánica de los soportes de la estructura y la cubierta de la caja de engranes, antes la vibración generada reactiva los sensores locales.

El éxito del sistema de monitoreo de las vibraciones de la caja de engranes es determinada por qué tan bueno es este factor, dirigido a proveer la sensibilidad de detección de fallas necesaria para una aplicación de monitoreo dado. Para esto se necesita de varias herramientas como son:

a) Acelerómetros
b) Sensores de velocidad
c) Sensores de desplazamiento
d) Detección de desechos del aceite

e) Detección de temperatura y termo grafía

Como sabemos la tendencia de los usuarios es tener todas las herramientas tecnológicas para la caja de engranes para así asegurar de cierta forma que las fallas no ocurran, sin embargo estas son fiables y el desgaste y las fallas ocurren después de miles y cientos de horas de operación. Del cual se ha recopilado información de vibraciones, desechos en el aceite, temperatura, la cual puede seguirse acumulando. Ahora veamos que, la conversión de los datos a información, análisis de la información y retención de los resultados es el objetivo del diagnóstico de la caja de engranes.

1.5. Monitoreo de los residuos del aceite

El oscurecimiento en apariencia del aceite de un motor utilizado no solo proporciona la idea de la necesidad de cambiarlo, también produce una inquietud sobre las condiciones del motor. El fluido de la maquinaria contiene la evidencia de muchos aspectos de las condiciones de la maquinaria deteriorada, solo esperando para ser probada y analizada.

El aceite en una maquina tiene una gran variedad de propósitos, este puede tener una función principal como lubricación o fluido de potencia, pero este también servirá de ayuda para enfriar el sistema, otra razón es que limpia y remueve desechos de las regiones criticas. Hay muchos tipos diferentes para describir el desgaste, una muy sencilla se muestra en la tabla 1.3.

Tipo de desgaste	Descripción	Comentarios
Adhesivo	Ocurre cuando dos superficies son forzadas bajo carga, y después se deslizan sobre la otra.	Incremento con carga y distancia de deslizamiento. Disminución con dureza de superficie.
Abrasivo	Ocurre cuando entre dos superficies que se deslizan entre sí, existe alguna partícula.	Partículas pueden rayar la superficie o puede causar desprendimiento de metal de la superficie de este.
Fatiga	Ocurre cuando impactos entre superficies	Esto puede pasar debido a impactos directos de rodamientos o deslizamientos produciendo un esfuerzo alternaciones repetidas. Los impactos pueden ser debido a cavitación.
Químico	Ocurre debido a la presencia de un químico en el aceite o atmosfera el cual causa deterioración del metal o de la superficie	Los otros tres tipos de desgaste pueden también ser implicados.

Tabla 1.3 Tipos de desgaste.

Como se puede observar la ausencia de cualquier aceite en estos procesos puede ser considerable y agravarse con la presencia de temperaturas elevadas. La presencia de aceite no solo reduce la temperatura, pero este separa las superficies de contacto para reducir los niveles de desgaste. Cualquier monitoreo del desgaste necesita ser consciente del rango del desgaste aceptable y los cambios en que el rango del desgaste indica un problema.

Existen muy poco aceites simples utilizados para lubricar o como fluidos de potencia. Cada fabricante de aceite cuenta con un gran número de aditivos en el paquete de aceites. Invariablemente de cual se utilice estos pueden llegar a ser muy efectivos cuando el aceite

es el primero en ser bombeado dentro del sistema, pero depende de la calidad del aditivo. En la tabla 1.4 se mencionan algunas tipos de aditivos.

Aditivo de aceite	Propósito	Comentarios
Anti-oxidación	Para frenar la formación de oxidación del aceite el cual produce lacas.	Compuestos inactivos se forman en lugar, pero los aditivos son agotados en el proceso.
Anti-desgaste	Para mejorar el contacto del metal, lo cual reduce el desgaste.	Una película absorbente es producida en la superficie del metal.
Inhibidor de corrosión	Para neutralizar el desarrollo de ácidos.	Una alcalinidad elevada es agregada al aceite, pero esto reduce gradualmente su eficacia.
Demulsificante	Para separar el agua del aceite.	Este agota en proporción la presencia de agua.
Agente de presión extrema	Para mejorar la superficie del metal bajo presión.	Un reactivo químico es causado por el aditivo.
Mejorar el índice de viscosidad	Para reducir el cambio de viscosidad con la temperatura.	Polímeros de cadena larga los cuales se abren con la temperatura. Una ambiente elevado cortante gradualmente rompe el polímero.

Tabla 1.4 Algunos aditivos de aceite típicos.

Numeras técnicas de monitoreo de aceite en línea están disponibles. Ninguna de ellas es capaz de analizar completamente el aceite en todas las formas, por lo cual es importante censar lo que es requerido para el monitoreo. La aplicación de las diferentes técnicas depende del tipo de partículas que se pretenden encontrar, por lo cual se dividen en general por el tipo de partículas a considerar como son: ferrosos, no ferrosos, no metálicos y todas las particulares.

– *Desechos ferrosos:* Como su nombre lo dice el desecho ferroso son todos aquellos metales ferro-magnéticos y paramagnéticos. En otras palabras son todos aquellos que pueden ser detectados magnéticamente. Este es un parámetro muy útil debido a que la fuerza magnética atrae las partículas y puede retenerse para su análisis.

El tipo de monitoreo más viejo es el del tapón magnético simple. Este es ahora el más efectivo, no solo porque mejoran los magnetos que se han desarrollado, pero porque algunos han sido diseñados con la posibilidad de perfeccionar:

- Puede ser removido de un sistema en operación.
- Puede tener un circuito eléctrico de conexión los cuales cortan cuando el desecho está presente.
- Puede ser equipado en una cámara ciclónica la cual genera su propio flujo turbulento.
- Puede ser censado por un cambio en el flujo magnético del monitoreo.

– *Desechos no ferrosos:* Una gran variedad de técnicas de monitoreo en línea relacionadas a la detección de partículas, como el paso de estas a través de una bobina inductiva. Esta técnica puede censar ambos metales los ferrosos y no ferrosos, pero en una forma rigurosa.

– *Desechos no metálicos:* Los materiales no metálicos incluyen todos los desechos que pueden ser adquiridos del ambiente, como pueden ser los materiales exóticos utilizados en maquinaria, desde plásticos suaves hasta cerámicas duras. Invariablemente estas tienen que ser

monitoreadas después de los metales ferrosos y no ferrosos ha sido removido de la muestra. Esto tiende a ser un tanto peligroso.

– *Todas las particulares:* La ventaja de censar todas las particulares es que pueden ser utilizadas en varias situaciones, desde censar un metal ferroso hasta el requerimiento de conocer que el desecho está siendo extraído del medio ambiente. el mismo monitoreo puede rendir en ambas funciones.

– Como un monitoreo tiene que ser aplicado con cuidado, es detectar un tamaño, o una distribución de tamaños, independientemente de la fuente. Debería de haber un cambio completamente gradual o puede ser que un repentino nacimiento indicando una falla está siendo desarrollada o se trata del desarrollo. La detección y análisis de todas las partículas en línea es realizado de varias maneras.

Las técnicas ópticas buscan en todo los desechos con efectos de luz. Existen muchos tipos diferentes de monitoreo óptico basado en una variedad de características luminosas:

- Reflexión hacia adelante
- Difracción de Fraunhofer
- Obscurecimiento de luz
- Dispersión de luz
- Dispersión fotométrica
- Espectroscopio de correlación de fotón
- Tiempo de transición
- Turbiedad

Entre estos cubren un amplio rango de tamaños. Ellos están limitados donde hay una densidad óptica elevada o con diferentes fluidos. Aire en el fluido puede también ser contado como partícula.

Técnicas aplicadas fuera de línea concierne a la exanimación de los desechos bajo el microscopio después de extraer los desechos del aceite. Esto puede realizarse automáticamente por el principio de análisis de imágenes usando un microscopio, video cámara, software computacional digitalizado y confiable.

Otra técnica de monitoreo la cual es más fuerte y disponible para hacer frente con los aspectos prácticos del sistema real, es basado en la técnica de obstrucción de filtro. En este caso las partículas con detectadas en diferentes mallas o rejillas, como el aceite es forzado a pasar atreves de esta. Esta detección es realizada por el principio de censar la caída de presión que ocurre.

1.6. Inspección con boroscopio

Cuando se utiliza en conjunto con las técnicas de análisis del conducto de flujo, vibraciones y tendencias, la inspección con boroscopio normalmente prevé el paso final en el proceso de identificar un problema interno [6]. La inspección con boroscopio es muy útil en proporcionar una vista general de las condiciones de componentes críticos, pero este está limitado por el diseño de la turbina de gas, el diseño del boroscopio y la capacidad del inspector.

La inspección con boroscopio es una exanimación visual sujeta a la experiencia del inspector y la calidad del instrumento que se está utilizando. El inspector necesita determinar el diámetro y la longitud adecuada del boroscopio para usar en cada puerto de inspección. También es muy importante determinar para cada localización a inspeccionar la fuente de iluminación del boroscopio. Dependiendo a la habilidad y experiencia del inspector, un boroscopio propiamente seleccionado deberá satisfacer todos los requerimientos necesarios.

Los baroscopios rígidos son una herramienta muy útil, ver figura 1.1, en lugares donde el acceso al campo de visión es atreves de un camino recto, sin embargo los baroscopios más versátiles son los flexibles de fibra óptica, ver figura 1.2. Estos están disponibles en rangos de diámetros desde 0.3 milímetros hasta 13 milímetros y longitudes de trabajo de aproximadamente 250 milímetros a 6,000 milímetros. Los boroscopio también están disponibles con articulaciones de 2 vías (arriba y abajo) y 4 vías (arriba, abajo, derecha e izquierda).

Un boroscopio es un instrumento óptico que consiste de un lente fijo, una serie de lentes transmisores y un lente ocular. En adición a esto un sistema de iluminación es también indispensable. Se debe tener sumo cuidado en la selección de la fuente de iluminación. El tamaño del puerto de entrada determina el diámetro máximo de la sonda del boroscopio que puede ser usada. La longitud de la sonda del boroscopio es determinada por el tamaño de la turbina de gas y la distancia desde el observador hasta el objeto a ser visto. Cuando utilizamos un boroscopio rígido es muy importante considerar la localización del área a ser inspeccionada con respecto a los puertos de acceso, los boroscopio flexibles pueden acomodarse casi en cualquier ángulo de vista.

Una consideración mayor en la selección de un boroscopio es el campo de vista, ya que este entra en tres categorías, estrecha, normal y ángulo amplio. Donde: campo de vista estrecho es de 10° a 40°, campo de vista normal es de 45° y campo de vista de ángulo amplio es de 50° a 80°.

La profundidad de campo es la distancia mínima y máxima del lente del objeto en el cual el objeto esta en enfoque nítido y es una función del campo de visión. Con un boroscopio es posible ver rápidamente señales de oxidación, erosión, corrosión, agrietamiento, y el resultado del daño de un objeto foranes sin desarmar la turbina de gas.

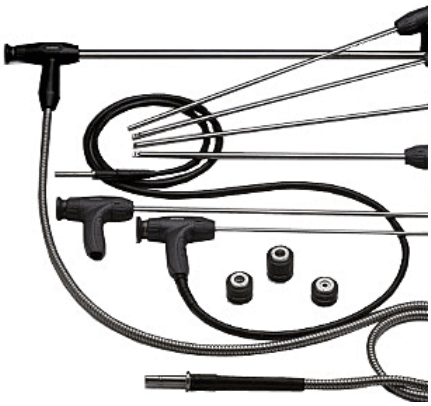


Figura 1.1 Boroscopio rígido.

Figura 1.2 Boroscopio flexible

Mientras que una inspección con boroscopio puede realizarse con frecuencia, es más efectiva cuando es utilizada para confirmar un problema destacado por medio del análisis de vibración o por el análisis de rendimiento del conducto de flujo. Al realizar una inspección con boroscopio a la turbina de gas puede ayudar a identificar áreas de problemas.

Por convencía en la evaluación de la condición del motor, algunos fabricantes instalan puertos para boroscopio en partes estratégicas atreves de sus motores. Sin embargo el inspector tiene que usar cada orificio disponibles o en algunos casos generar aberturas que son fácilmente de crear mediante remover componentes del motor (como son boquillas de combustibles, sonda de presión, sonda temperatura, etc.). En general los puertos para el boroscopio son localizados en la cámara de combustión y en la turbina.

El puerto para el boroscopio de la cámara de combustión puede ser utilizado para observar las condiciones de las boquillas del combustible, carcasa interior de la cámara de combustión, las boquillas de la primera etapa de la turbina y los alabes de la primera etapa de la turbina. Los puertos para el boroscopio de la mitad de la turbina puedes ser utilizados para observar los alabes de la turbina en ambas direcciones hacia adelante y hacia atrás de la posición del boroscopio.

Los puertos para el boroscopio son raramente proporcionados con la sección de compresor de la turbina de gas. Sin embargo, las venas de las guías de entrada al compresor y la primera etapa del compresor son accesibles por medio del compartimiento del aire de limpieza inmediatamente delante de la turbina de gas.

1.7. Monitoreo de las condiciones en una industria manufacturera y de producción

De acuerdo a la operación de producción, las compañías industriales están agrupadas en dos categorías: compañías manufactureras e industrias de procesos. Las compañías manufactureras son aquellas que producen productos discretos como carros, computadoras y herramientas mecánicas, mientras las industrias de procesos son representadas por industrias químicas, industria de producción de petróleos, industria de formación del acero, etc. [7].

Una compañía manufacturera puede organizar su producción como producción por lote, producción en masa o producción de trabajo-venta. En la producción por lote, los equipos y métodos generalmente propuestos son usados para producir cantidades pequeñas a medianas de productos, sus especificaciones pueden ser muy diferentes de un lote a otro. En la producción en masa, la producción en línea total es con frecuencia dedicada a solo un producto en particular con altos volúmenes de producción y por lo tanto con equipo y métodos especializados. Son usados para lograr costos de producción bajos. La producción de trabajo-venta es para producción de volúmenes bajos, muchas veces uno de un tipo para conocer especificaciones del pedido del cliente.

Una de las principales funciones de la ingeniería de la manufactura es planear la producción y control. De tal manera que planee y controle efectivamente la actividad manufacturera, la ingeniería de la manufactura requiere ciertos tipos de información: los productos a producir, datos y cantidades de entrega, y recursos humanos y físicos disponibles [8]. Con

este conocimiento ellos podrán determinar los requerimientos necesarios para cumplir el objetivo.

Cualquier producción comprende procesos físicos y flujo de información. Los procesos físicos abarcan los equipos mecánicos que deben ser tan buenos como los materiales que se procesan. Para que los procesos físicos funciones como se espera, no solo se requieren instrucciones, también se necesita que se monitoreen los procesos. Esto significa la colección de datos de la operación de la producción; la información obtenida de estos datos serán compartidos entre los diferentes departamentos funcionales de la compañía.

La información de los proceso define el estatus y el rendimiento de los procesos y envuelve las entradas y salidas de los proseos. La información relaciona el estatus de operación de la maquina y sus condiciones. Esto ayuda para calcular del proceso y de la información de la maquina, el nivel de utilización de la misma, para diseñar la optima cedula de cambio de la herramienta o para monitorear las averías de la maquina y diagnosticar sus causas.

Una porción del proceso y de la información de la maquina es utilizada para el control del proceso y de la maquina. Este control es realizado en tiempo real, muchas veces repetitiva y cíclicamente, y producen información en un rango mayor. Otro aspecto de la información de la producción es requerida para reportar resoluciones y tienden a envolver un gran cantidad de datos el cual es producido en un rango pequeño.

El monitoreo de las condiciones es confinado a los niveles de maquina o equipo, por lo tanto es evidente que reside en los niveles de procesos de equipo. Sin embargo, como la información de las condiciones generadas en muchas veces se deja pasar al ordenador de segundo nivel, de tal forma que se provee un reporte en el estatus de operación de la maquina en la planta. Esto puede sostener que el monitoreo de las condiciones también concierne al nivel del ordenador del supervisor.

El monitoreo de las condiciones es solo una parte del control de la producción pero este es un parte muy importante debido a nuestro deseo de automatizar el sistema de manufactura. Evidentemente el control de la producción incluye el monitoreo de los procesos de producción por lo tanto que el estatus de su rendimiento pueda ser establecido. El control de la producción envuelve las actividades:

- Censar
- Colección de datos
- Comunicación

Los sistemas de control de la producción modernos son implementados como un distribuidor de sistemas de computación debido a que los ordenadores están situados en diferentes partes y están unidos todos juntos por medio de algún tipo de redes de comunicación [9]. Todas estas actividades son controladas por el programa corriendo en diferentes computadoras en red.

El monitoreo y el control de la producción son actividades muy complejas; y los paquetes de software están disponibles en el mercado para permitir que la ingeniería de manufactura realice su trabajo más fácilmente. Básicamente tales sistemas proveen una dirección y supervisión de las herramientas que monitorean el rendimiento de la producción además de

suministrar información en tiempo real del entorno de producción, con los datos recolectados de la planta después se transforman en la información significativa.

La parte del software del sistema provee información de la producción a partir de los datos recabados, típicamente estos incluyen los siguientes parámetros:

- Utilización y eficiencia de la maquina
- Tiempos de operación y de paros, y análisis de tiempos establecidos.
- Conteo de la producción
- Análisis de rechazos o defectos
- Velocidad de la maquina
- Análisis de detecciones de paros automáticos
- Revisión y reportes de fin de trabajo
- Material utilizado y requerido
- Cambio de reporte y reporte de excepción
- Identificación del producto y seguimiento de la producción
- Mantenimiento mas optimo
- Gestión de residuos
- Colección de datos electrónicamente para sistemas de planeación
- Análisis de costo de automatización.

Cuando seleccionamos la parte del proceso para el monitoreo de las condiciones, las consideraciones deberían ser regidas por los requerimientos financieros, organizacional y de seguridad. Los criterios de selección puede ser basada en las consecuencias de falla convertidas en un gasto.

1.8. Monitoreo de las condiciones de una planta de potencia

Avances rápidos en tecnología de plantas de potencia junto con una competitividad mayor y un entorno desregulado han creado una necesidad para avanzar los sistemas de monitoreo de las condiciones, especialmente para criticas turbo maquinarias y equipos auxiliares. Con una nueva generación de elevadas temperaturas y elevadas capacidades de salida de los motores de turbina de gas, que están siendo aplicados en grandes combinaciones de plantas de ciclos de potencia, con el objetivo de alcanzar una mayor disponibilidad y limitar la degradaciones de máxima importancia [10].

Grandes cantidades de hidrocarburos alimentando plantas de potencia y turbinas de gas basadas en ciclos combinados serán el pilar de la generación de potencia para las décadas siguientes. Estas plantas son caracterizadas por grandes inversiones de capital y elevados costos de combustible creando una necesidad crítica para detectar, identificar y diagnosticar los problemas de la maquinaria.

Los estímulos por establecer un programa de monitoreo de las condiciones de la maquinaria es basado en la necesidad para evaluar y la tendencia de las condiciones de operación del equipo de tal manera que se minimice el riesgo y el impacto económico de un paro o falla inesperada. El monitoreo de las condiciones ha demostrado rápidamente su valor cuando ha sido implementado en equipos donde el tiempo muerto interrumpe la generación de

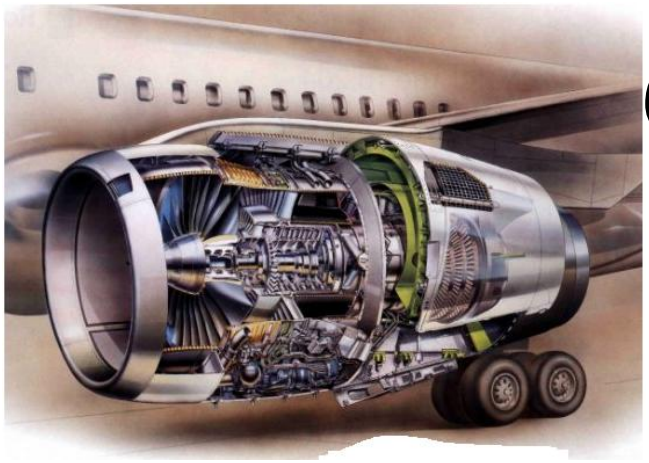
potencia [11]. Los principales objetivos de un sistema de monitoreo de las condiciones completo son:

- Monitorear eficientemente para mantener el mejor rango de calor del ciclo y limitar la deterioración de la turbina y el ciclo.
- Extender la seguridad del ciclo entre los intervalos de revisión.
- Minimizar el numero de actividades de revisión "destape, inspección y reparaciones necesarias".
- Mejorar la eficiencia del mantenimiento para dirigir reparaciones y acciones de revisiones hacia problemas incipientes.
- Ayuda a la planeación de la mano de obra y requerimientos de partes durante las revisiones.

Las aéreas de los problemas en una planta de potencia térmica son:

- Fallas en tubería
- Escoria y suciedad
- Chimenea de implosiones y explosiones
- Problemas de control
- Exfoliación
- Turbogenerador
  - Fallas en alabes
  - Problemas en el eje de la turbina
  - Fallas de rodamientos
  - Inducción de agua
  - Erosión de partículas solidas
  - Problemas de control de la turbina
- Problemas del condensador
- Bomba de alimentación de la caldera/bomba de condensados
- Generador – fallas de retenes de anillo, ranura de cuña, fallas de flecha.

Capítulo 2.- Turbinas de Gas y el Monitoreo de sus Condiciones



2.1. Turbinas de gas de potencia

Una turbina de gas es un motor diseñado para convertir la energía de un combustible en alguna forma de energía útil, por ejemplo: potencia mecánica (en un eje) o el impulso a alta velocidad de un reactor. Las turbinas de gas es una planta de potencia la cual produce una gran cantidad de energía para su tamaño y peso. Estas están formadas básicamente por una sección generadora del gas y una unidad para la conversión de la energía. La sección generadora del gas consta de un compresor, una cámara de combustión y una turbina, la cual solamente extrae energía suficiente para impulsar el compresor. De este modo el compresor origina un gas a temperatura y presión elevadas a la salida de la turbina, ver figura 2.1 y figura 2.2.

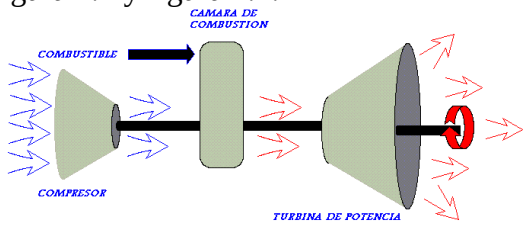


Figura 2.1 Generador de gas

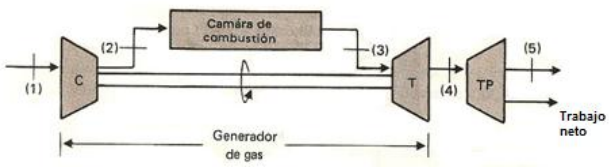


Figura 2.2 Motor básico de turbina de gas

Las turbinas de gas industriales se emplean en una gran variedad de aplicaciones, tales como: generación de electricidad, impulsores de bombas, compresores de combustible líquido o gaseoso, etc. En la figura 2.3 se muestran las partes que conforman una turbina de gas [12].

Figura 2.3 Motor de turbina de gas con sus componentes mostrados por separado.

En la figura 2.2 se muestra el motor básico (el más sencillo) de una turbina de gas. La turbina de gas generalmente opera en un ciclo abierto. El aire del medio ambiente se introduce dentro del compresor donde se comprime de forma adiabática, es decir su temperatura y presión se eleva. El aire de alta presión sigue hacia la cámara de combustión donde el combustible se quema a presión constante y se alcanza la temperatura máxima del ciclo, que se origina en la etapa 3. Luego los gases de alta temperatura que resultan de la combustión entran a la turbina, donde se expanden en forma adiabática en la turbina, de tal forma que producen potencia, se usa una parte del trabajo que se desarrolla en la turbina para impulsar el compresor y se entrega el resto al equipo que está afuera de la turbina de gas. Los gases de escape que salen de la turbina se expulsan hacia el medio ambiente.

En un ciclo de aire estándar se supone que el aire es el único fluido de trabajo, y la cámara de combustión se sustituye por un proceso de adición de calor. El ciclo que se considera como un ciclo cerrado se complementa con un proceso de rechazo de calor a presión constante hacia el medio ambiente. En el ciclo básico ideal de aire estándar que el fluido de trabajo experimenta en es el ciclo Brayton, el cual está integrado por cuatro procesos internamente reversibles:

- 1-2 Compresión isoentrópica (en el compresor)
- 2-3 Adición de calor a presión constante
- 3-4 Expansión isoentrópica (en la turbina)
- 4-1 Rechazo de calor a presión constante

Se supone que los procesos de compresión y expansión son adiabático y reversibles (isoentrópica); que no hay caída de presión durante el proceso de adición de calor y que la presión que sale de la turbina es igual a la presión que entra en el compresor [13]. Los diagramas *T-s* y *P-v* del ciclo Brayton ideal se muestran en las figuras 2.4 y 2.5 respectivamente.

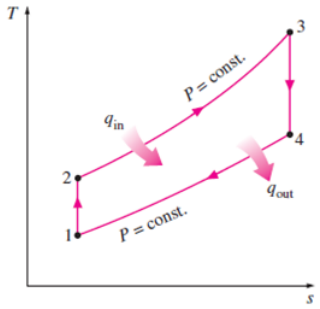


Figura 2.4 Diagrama *T-s*

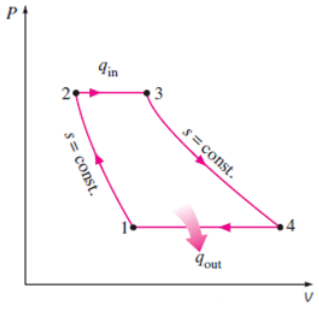


Figura 2.5 Diagrama *P-v*

Las turbinas de gas han experimentado un proceso y crecimiento fenomenal desde su aparición en 1930. Las primeras turbinas de gas construidas en 1940 y aun en 1950 tenían una eficiencia del ciclo sencillo alrededor del 17% debido a las bajas eficiencias del compresor y de la turbina, además de las bajas temperaturas de entrada a la turbina dadas las limitaciones de la metalurgia de aquellos tiempos. Por lo cual se ha tratado desde entonces mejorar la eficiencia del ciclo implementado varias modificaciones tanto al ciclo como a la turbina. Las modificaciones a las cuales se ha tenido que recurrir para mejorar la eficiencia del ciclo han sido incorporando elementos o componentes como son el enfriamiento, regeneración y recalentamiento, el cual se hace dependiendo la aplicación y en muchos caso se aplican hasta los tres tipos.

Turbina de Gas con Regenerador

La única mejora que se produce al aumentar en su totalidad la eficiencia térmica es cuando se incluye un dispositivo que transfiere energía (un intercambiador de calor) desde el gas caliente de descarga de la turbina hacia el aire que sale del compresor. En la figura 2.6 se ilustra el diagrama de flujo correspondiente al ciclo regenerativo. Hay dos tipos principales de regeneradores que se utilizan en la actualidad. El recuperador y el regenerador de matriz giratoria.

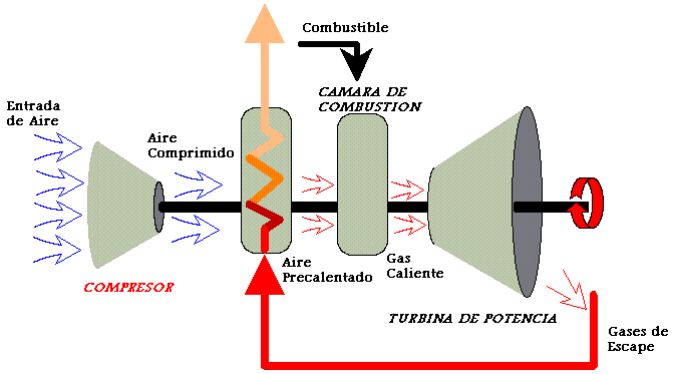


Figura 2.6 Diagrama de flujo de un motor de turbina de gas regenerativa.

La aplicación específica de una turbina de gas determinara que tipo de intercambiador de calor debe emplearse. Deberá recordarse que el aumento con la eficiencia cíclica que puede lograrse mediante el empleo de un regenerador o de un recuperador, debe compararse con

la desventaja ocasionada por un aumento en los problemas de servicio, costo, tamaño y en el peso.

Turbina de Gas con Enfriamiento Interno

El enfriamiento interno disminuye el trabajo del compresor sin alterar el trabajo de la turbina mientras. Puede lograrse que el proceso de compresión se aproxime al proceso de compresión isotérmica mediante el enfriamiento interno, el cual implica el empleo de dos o más compresores. En la figura 2.7 se supone que la turbina generadora de gas (TG) impulsa ambos compresores, entregándose la potencia generada por la turbina de potencia a un aparato externo o a otra turbina.

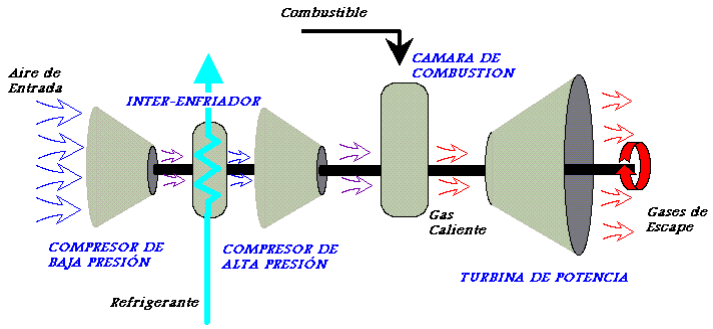


Figura 2.7 Motor de turbina de gas con enfriamiento interno.

El primer compresor (C_1) comprime aire desde la presión ambiente hasta una presión intermedia. El aire que sale del primer compresor entra en el enfriador interno, en donde se elimina el calor. Idealmente no tiene lugar ninguna caída de presión en el enfriador interno y la temperatura del aire cuando sale del enfriador (estado 1.5) es la misma que la temperatura a la entrada del primer compresor (estado 1). El segundo compresor (C_2) completa el proceso de compresión hasta la presión final deseada.

Debido al costo y complejidad que representa agregar un enfriador interno, no se utiliza más de uno, en el caso del diseño de un motor de turbina de gas, el agregar un enfriador interno se reduce el trabajo requerido para impulsar el compresor y solo cambia levemente el trabajo total de la turbina aumentara el trabajo neto correspondiente al ciclo.

Turbina de Gas con Recalentamiento

Recalentamiento aumenta el trabajo de la turbina sin alterar el trabajo del compresor. El trabajo que puede obtenerse de una turbina que opera entre presiones fijas de entrada y salida puede aumentarse al construir una turbina que opere isotérmicamente. En la práctica esto casi se obtiene al permitir que los gases se expandan desde la presión máxima del ciclo, a una cierta presión intermedia, recalentando a presión constante hasta la temperatura máxima del ciclo y después expandiendo los gases en una segunda turbina hasta la presión mínima del ciclo.

En la figura 2.8 aparece un diagrama esquemático correspondiente a un motor de turbina de gas que opera con ciclo de recalentamiento. Como se puede observar en la figura el motor tiene una turbina TG que impulsa al compresor y la turbina de potencia crea el trabajo neto.

Al agregar el recalentamiento a un motor de turbina de gas aumenta el trabajo de la turbina sin cambiar el trabajo del compresor.

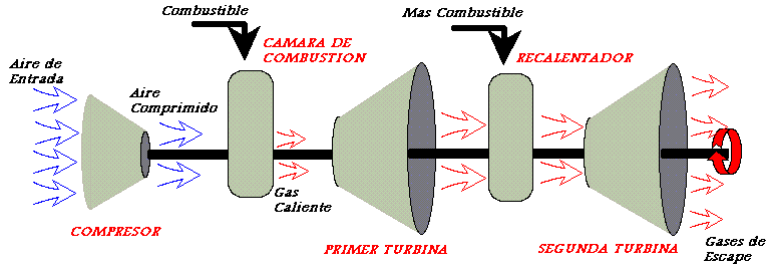


Figura 2.8 Diagrama esquemático de una turbina de gas con recalentamiento.

Turbina de gas con Enfriamiento Interno, Recalentamiento y Regeneración

El enfriamiento interno y el recalentamiento, al usarse aisladamente, disminuye la eficiencia térmica del ciclo; por tal motivo rara vez se utilizan solos. En la figura 2.9 se muestra un diagrama esquemático de un ciclo con recalentamiento – regeneración – enfriamiento interno que habitualmente se utiliza cuando se aplica recalentamiento y enfriamiento interno. El motor que se ilustra en esta figura utiliza la turbina T2 (TAP) para impulsar el compresor C2 (CAP) y la turbina T1 (TBP) para impulsar el compresor C1 (CBP). La turbina de potencia (TP) entrega el trabajo a un equipo externo.

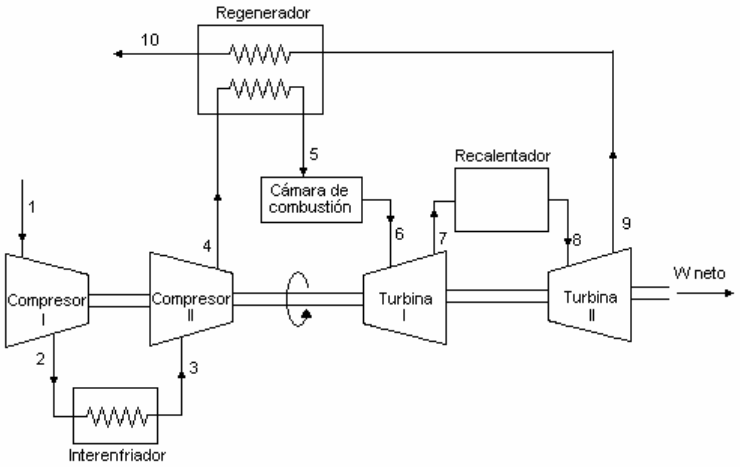


Figura. 2.9 Diagrama esquemático de un ciclo de turbina de gas con enfriamiento interno, recalentamiento y regeneración.

Planta de Potencia de Ciclo Combinado

La continua búsqueda de eficiencia térmica más altas ha originado modificaciones innovadoras en las plantas de potencia convencionales. El ciclo de vapor binario es una de ellas. Una aun más popular incluye un ciclo de potencia de gas que remata a un ciclo de potencia de vapor, que se denomina ciclo combinado de gas-vapor o solo ciclo combinado. El ciclo combinado de mayor interés es el ciclo Brayton de una turbina de gas, que remata al ciclo Rankine de una turbina de vapor, con una eficiencia térmica más alta que cualquiera de los ciclos ejecutados por separado.

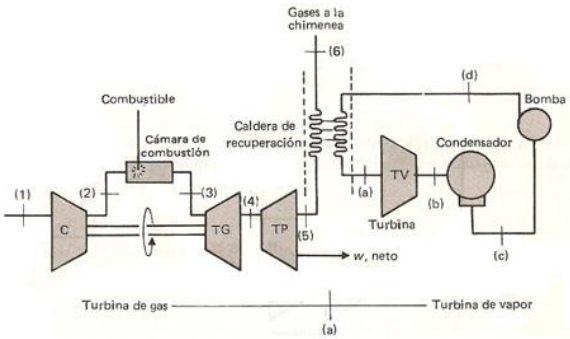


Figura 2.10 Ciclo combinado sin combustión suplementaria.

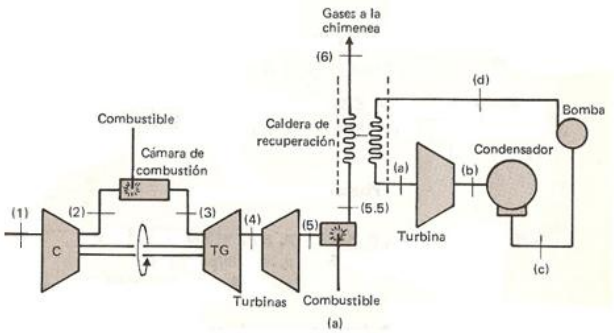


Figura 2.11 Ciclo combinado con cámara adicional de combustión.

Con los estudios se han demostrado que un sistema de ciclo combinado que utiliza turbina de gas y de vapor puede constituir una planta óptima en las aéreas de carga intermedia. La planta de ciclo combinado está formada habitualmente por una o más turbinas de gas descargando sobre una caldera de recuperación de calor. Algunas de estas unidades cuentan con un combustible suplementario agregado a los gases de escape, en tanto que otras no utilizan ningún combustible adicional [14]. En las figuras 2.10, 2.11 y 2.12 se muestran tres configuraciones posibles, aplicables a una planta de potencia de ciclo combinado.

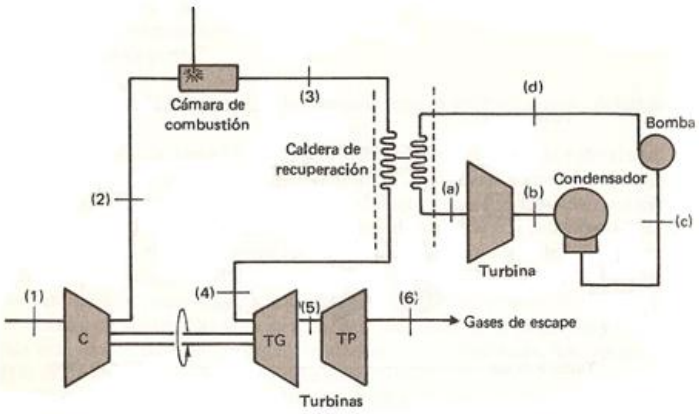


Figura 2.12 Ciclo combinado con caldera de recuperación de calor entre el compresor de la turbina de gas y las turbinas

2.2. Turbinas de gas para la propulsión de aviones

Turborreactor (Aire Estándar)

Son varios los tipos de motores de turbina de gas que se utilizan en la propulsión de los aviones. Comúnmente se dividen en las categorías de turborreactor y turbohélice. Todos se pueden construir con base en el mismo generador de gas. La figura 2.13 muestra un diagrama esquemático correspondiente a un motor turborreactor sin recalentador. El aire ambiente entra al difusor, donde disminuye la velocidad y aumenta la presión estática. Los estados de entrada y salida del compresor, de la cámara de combustión y de la turbina que se utilizan al definir las eficiencias y caídas de presión, son por lo general las presiones totales (estancamiento).

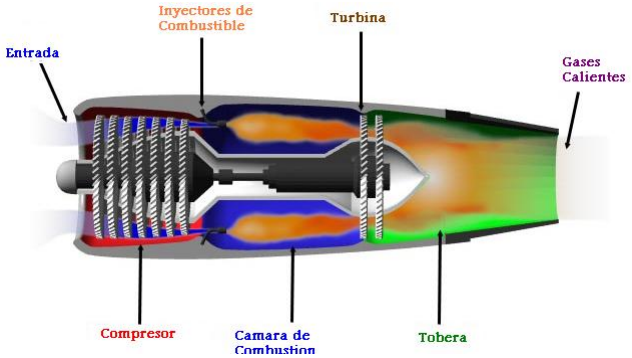


Figura 2.13 Diagrama esquemático general de un motor turborreactor sin recalentador.

Como el trabajo (potencia) realizado por la turbina es igual al trabajo requerido para impulsar el compresor, la presión a la salida de la turbina es elevada. El aire se expande a continuación, pasando por la tobera, donde aumenta la velocidad con una consiguiente disminución en la presión. Idealmente la tobera opera de forma isoentrópica. En la realidad opera adiabáticamente, pero en forma irreversible.

Motor Turboventilador

La eficiencia de la propulsión de un motor turborreactor es sumamente baja, excepto a velocidades elevadas de vuelo. Esto resulta evidente por la gran velocidad de flujo de gases que sale del motor. Para aumentar la eficiencia en la propulsión, debe disminuirse la velocidad de salida de la tobera. Esto puede lograrse extrayendo más potencia en la turbina, sin aumentar la potencia requerida para impulsar el compresor. Este aumento de potencia se puede utilizar para comprimir aire adicional, el cual se envía por fuera de la cámara de combustión, aumentando así la masa de aire comprimido sin aumentar la cantidad de combustible suministrado al motor. El tipo de motor que realiza lo anterior se denomina motor de turboventilador.

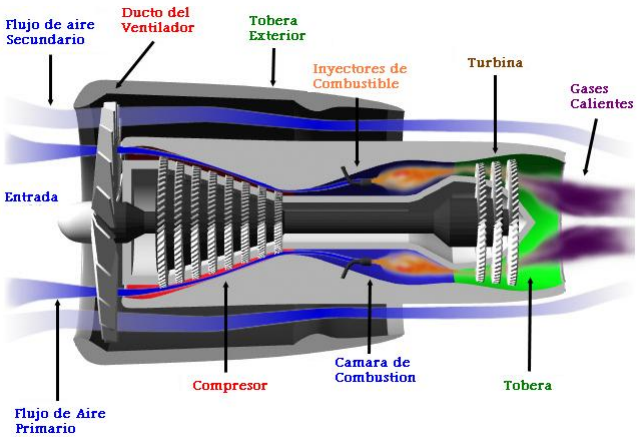


Figura 2.14 Disposición general de un motor turborreactor

Un motor turboventilador es básicamente un motor turborreactor en el que se han eliminado algunas etapas (escalonamiento) de la parte delantera del compresor, sustituyéndolos con etapas de mayor diámetro, habitualmente denominadas ventiladores. En la figura 2.14 aparece representado.

El turboventilador tiene la ventaja de que puede lograrse un gran aumento en impulso agregando un ventilador a un turborreactor ya existente. Esto reduce la velocidad de salida, aumenta la eficiencia de la propulsión. El motor turboventilador como no tiene aumento en el flujo de combustible a consecuencia de este aumento en el impulso tiene más impulso por masa de aire que entra al generador de gas y por lo tanto un consumo bajo de impulso – consumo especifico de combustible.

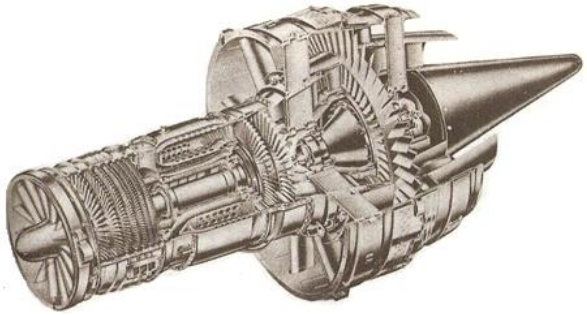


Fig. 2.15 Corte de un Motor turboventilador.

En el turboventilador sin mezcla el aire que pasa a través del ventilador entra a un ducto para después circular por su propia tobera. En el motor turboventilador con flujo mezclado el aire que pasa a través del ventilador de envía por un ducto alrededor de la cámara de combustión y se mezcla con los gases del generador de gas atrás de la turbina, la presión estática en el punto de mezcla debe ser la misma para los dos flujos, los flujos mezclados pasan a continuación por una tobera común. En la figura 2.15 se muestra un motor turboventilador.

Motor Turbohélice

Un tercer tipo de turbina de gas utilizada para la propulsión de aviones es la de motor turbohélice. La propulsión de un motor turbohélice se lleva a cabo mediante la acción combinada de una hélice situada adelante del motor y el impulso producido por los gases de escape procedentes de la turbina de gas. Un motor turborreactor puede convertirse en un motor turbohélice agregando una turbina adicional que impulse la hélice por medio de un sistema de engranajes reductores de la velocidad. En la figura 2.16 se presenta un esquema de este tipo de motor.

Un motor turbohélice combina las ventajas del motor turborreactor y la eficiencia en la propulsión de una hélice. El motor turborreactor deriva su impulso de un gran cambio de momento en una masa relativamente reducida de aire, en tanto que el motor turbohélice desarrolla su fuerza de propulsión generando un cambio de momento reducido aplicado a una masa relativamente grande de aire. La turbina de un motor turbohélice está diseñada no solo para absorber la potencia requerida para impulsar el compresor y los accesorios sino para dar también al eje de la hélice el máximo par de torsión posible.

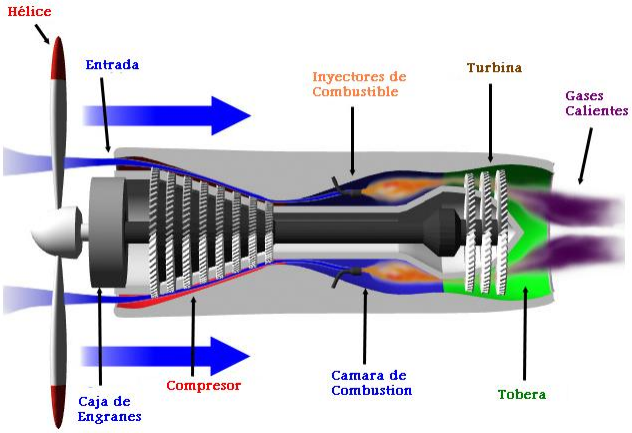


Figura 2.16 Diagrama esquemático de un motor de turbina de gas turbohélice.

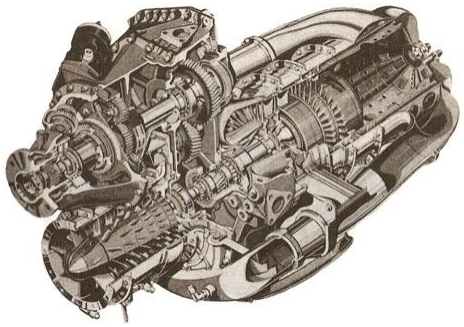


Figura 2.17 Corte de un motor de turbohélice.

El motor turbohélice tiene un impulso – consumo especifico de combustible más reducido durante la elevación y a velocidades bajas a subsónicas moderadas que los motores turborreactor y turboventilador, disminuyendo esta ventaja a medida que aumenta la altitud y la velocidad. En la figura 2.17 se observa el corte de un motor de turbohélice.

2.3. Compresores

La compresión de grandes volúmenes de aire es esencial para el buen funcionamiento de un motor de turbina de gas. Se ha logrado mediante dos tipos de compresores: el de flujo axial y el de flujo centrífugo o flujo radial.

Los compresores deben tener buena eficiencia sobre un amplio margen de puntos operativos. El objetivo de un buen diseño de un compresor consiste en obtener la mayor cantidad de aire a través de un compresor de un diámetro determinado, con un mínimo de etapas, en tanto se retienen elevadas eficiencias y estabilidad aerodinámica a lo largo del margen de operación. La libertad del diseño por lo general está limitada por razones mecánicas, geométricas, de costo y de tiempo. Otra razón a considerar es la compatibilidad entre la velocidad del eje de compresión y la de una buena turbina.

En la figura 2.18 se ilustra las trayectorias típicas de un flujo de los componentes de flujo axial y de flujo centrifugo. Las trayectorias del flujo en un compresor de flujo axial es paralelo en lo esencial al eje de giro, cada etapa incluye una hilera de alabes giratorios donde se agrega energía al fluido, este rotor va seguido por una hilera de alabes fijos

conocidos habitualmente como estator, se requieren varias etapas en un compresor de flujo axial para obtener las elevadas relaciones de presión que se desean.

En un compresor de flujo centrifugo el fluido entra por el centro (ojo) del compresor y gira radialmente hacia afuera, el componente de giro del compresor es seguido por un pasaje difusor que puede contar o no con aletas o alabes estacionarios.

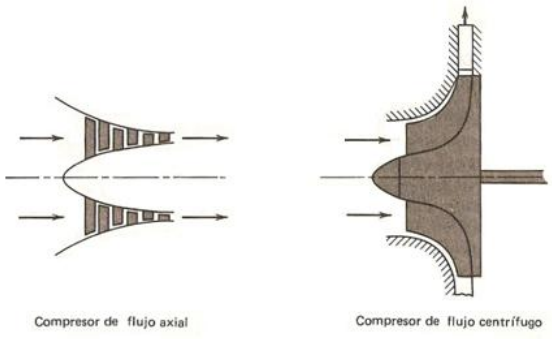


Fig. 2.18 Trayectorias del flujo de los compresores de flujo axial y de flujo centrífugo.

Las ventajas del compresor de flujo axial sobre el compresor de flujo centrífugo son las siguientes:

1. Área frontal más reducida para una determinada masa de flujo.
2. La dirección de flujo y la descarga son más adecuadas para las etapas múltiples.
3. Se puede aplicar investigación experimental en cascada en los componentes en desarrollo.
4. Eficiencia más elevada en ciertas medidas con altas relaciones de presión.

En la figura 2.19 se muestran las primeras tres etapas de un compresor de flujo axial. Muchos motores cuentan con alabes guías a la entrada, situados delante de la primera hilera de alabes giratorios (rotor). El propósito de la guía de entrada consiste en dirigir el fluido hacia la primera hilera de alabes rotores. Cada hilera de alabes rotores esta seguida por una hilera de alabes estacionarios (estator). Una etapa está formada por una hilera de alabes del rotor y una de alabes del estator. Todo el trabajo realizado en el fluido de trabajo se lleva a cabo en las hileras giratorias, convirtiendo los estatores en energía cinética del fluido en presión y dirigiendo el fluido hacia el rotor siguiente. Muchos motores de turbina de gas recientemente diseñados no cuentan con alabes guía a la entrada. Esto elimina una posible fuente de ruido.

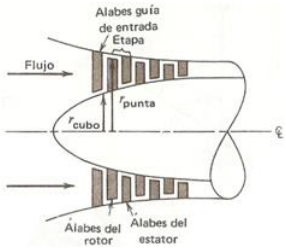


Figura 2.19 corte transversal de un compresor de flujo axial.

2.4. Turbinas

En una turbina la etapa consiste en un miembro estacionario y otro giratorio, es decir la hilera estacionaria llamada habitualmente tobera y que precede a la hilera giratoria (alabe).

Son dos los tipos generales de turbina: la turbina de flujo radial y la turbina de flujo axial. La primera se asemeja a un compresor de flujo centrífugo, excepto debido a que el flujo es hacia el interior en lugar de ser hacia afuera, en la figura 2.20 se muestran las vistas lateral y frontal de una turbina de flujo radial. Las turbinas de flujo radial se utilizan solo para potencias extremadamente bajas o cuando el tamaño compacto tiene más importancia que el rendimiento.

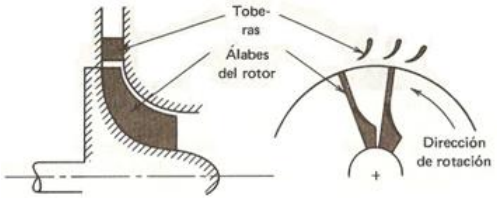


Figura 2.20 Turbina de flujo radial.

Las turbinas de flujo axial casi siempre se utilizan en los motores de turbinas de gas. Una turbina de flujo axial puede estar formada por una o más etapas, consistiendo cada una de ellas en una hilera de tobera y una de rotor. Las velocidades relativas en una turbina de flujo axial son en general substancialmente más elevadas que las que tienen lugar en un compresor de flujo axial, con un cambio mayor en entalpia por etapa. Además se requieren mucho menos etapas en una turbina que en un compresor, debido a que en el compresor de flujo axial el flujo se desacelera (se difunde) en los conductos con un aumento consiguiente en la presión, en tanto que el gas se acelera en una turbina [15].

2.5. Admisiones, cámara de combustión y toberas

Admisiones

Los objetivos de cualquier entrada al motor de turbina de gas para aviones consisten en proveer un suministro suficiente de aire al compresor, son una pérdida tan baja de presión como sea posible y de tal manera que el obstáculo que represente para el vuelo sea igualmente reducido. Su propósito consiste en ser un difusor que reduzca la velocidad del aire de admisión en la forma más eficiente posible.

- *Admisiones Subsónicas*

Existen distintos tipos de conducto de admisión de aire aplicados a los motores de turbinas de gas para la aviación, con los motores turbohélice el diseño de la admisión es complicado por la hélice y la caja de engranes en la entrada al motor. Los motores de aviación pueden situarse debajo de las alas del avión, en la base del estabilizador vertical o en el fuselaje del avión con su entrada ubicada en la raíz del ala o debajo del fuselaje. Cada una de estas puede ocasionar problemas relacionados con las admisiones subsónicas como pueden ser: distorsión a la entrada del compresor y pérdida en la presión total.

Las admisiones también pueden clasificarse como admisión simple o admisión dividida. La configuración de admisión dividida puede ocasionar distorsión en la entrada del motor.

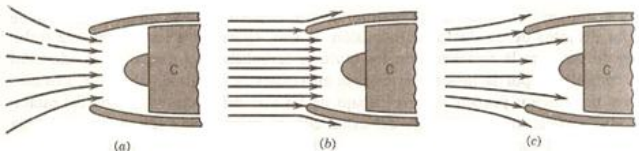


Figura 2.21 Patrón del flujo del aire para diferentes velocidades de avance. a) Velocidad estática, b) Velocidad de avance reducida y c) Velocidad de avance elevada

Las admisiones subsónicas son de un diseño fijo y la superficie interna de la admisión forma una de sección difusora antes del compresor. El patrón que sigue el aire de entrada con velocidad cero de avance del avión se muestra en la figura 2.21a; con la velocidad reducida en la figura 2.21b y con elevada velocidad de avance en la figura 2.21c. Es esencial que la entrada se diseñe de tal manera que no se produzca separación de la capa límite.

- *Admisiones Supersónicas*

Las admisiones supersónicas son mucho más difíciles de diseñar que las admisiones subsónicas. Las admisiones utilizadas en un avión supersónico representa el diseño que obtiene el funcionamiento óptimo aplicable a la misión para la cual se ha diseñado el avión. La admisión debe suministrar un adecuado funcionamiento subsónico, una buena distribución de presión a la entrada del compresor y relaciones de recuperación de presión elevadas y debe poder funcionar eficientemente con todas las presiones y temperaturas ambientales durante el despegue, el vuelo subsónico y en las condiciones de diseño supersónicas.

Las características de funcionamiento de la admisión, utilizadas para evaluar el funcionamiento de las admisiones supersónicas y que son las que tienen una influencia más importante en el funcionamiento del avión y en su campo de acción, son las siguientes:

1. Recuperación de la presión total.
2. Arrastre creado por la cubierta del motor.
3. Flujo de purga de la capa limite.
4. Relación de área de captura (relación de flujo de masa).
5. Peso.

Las admisiones supersónicas se clasifican habitualmente por su porcentaje de compresión interna, la cual se refiere a la cantidad de cambio de área supersónica que tiene lugar entre el reborde de la cubierta del motor y la garganta, esto se representa en la figura 2.22.

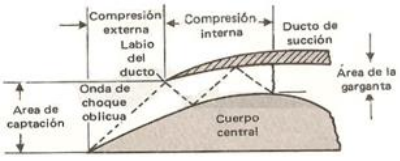


Figura 2.22 Admisión supersónica.

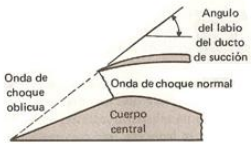


Figura 2.23 Admisión supersónica, toda la compresión externa.

Cámaras de Combustión

Los esfuerzos orientados para obtener cámaras de combustión eficientes, compactas y de baja emisión a fin de que se utilicen en los motores de turbina de gas, especialmente en sus aplicaciones de aviación, se ve dificultado por el amplio margen de condiciones operativas en que debe funcionar el sistema. El margen operativo debe incluir el arranque, el funcionamiento en vacio, la aceleración y la desaceleración, así como la operación a toda potencia, además que los motores de aviación deben operar en condiciones correspondientes al nivel del mar o a elevadas altitudes.

Los requerimientos aplicables a una de combustión incluyen lo siguiente:

1. Liberación de la energía química del combustible en el espacio más reducido posibles (longitud y diámetro)
2. Caída mínima de presión a todo lo largo del espacio de funcionamiento.
3. Operación estabilizada y eficiente a lo largo de una amplia gama de relaciones combustible – aire, altitudes, velocidades de vuelo o niveles de potencia.
4. Confiabilidad igual o superior a la vida sin revisión general del motor.
5. Capacidad para volver a encenderse en altitudes utilizada por los motores de avión.
6. Distribución correcta de la temperatura a la entrada del estator de la turbina.

Hay tres tipos de cámara: de recipiente, anular y una combinación de los dos anteriores.

La cámara de combustión de tipo *recipiente* está formada por un número de unidades de diámetro pequeño con revestimiento y cubiertas individuales. Las cámaras de este tipo se utilizan más frecuentemente en los motores de turbina de gas con compresor centrifugo, a medida que el aire sale del difusor del compresor se divide es enviado por ductos a los recipiente individuales. Este tipo de recipientes tienen una excelente resistencia estructural, sin peso exagerado, además de que los recipientes individuales pueden ser inspeccionados, eliminados y sustituirlos, lo cual resulta más fácil producir un cierto número de recipientes pequeños que uno solo de gran tamaño.

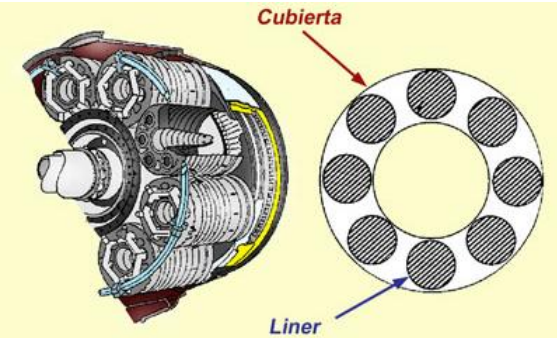


Figura 2.24 Cámara de combustión tipo recipiente

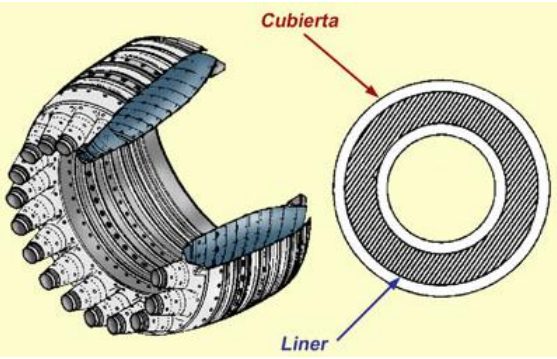


Figura 2.25 Cámara de combustión tipo anular

La cámara de combustión tipo *anular* figura 2.25 está formada por una o dos envolturas continuas, el combustible se introduce por conducto de toberas a la entrada de la envoltura y el aire secundario lo hace por perforaciones, este aire secundario mantiene la llama separada de la envoltura y diluye los gases de la cámara de combustión, dándoles la temperatura que se desea para la entrada de la turbina. Este tipo de cámara de combustión tiene la ventaja de utilizar eficientemente el espacio disponible y suministra una mezcla uniforme de gas a la entrada a las toberas de la turbina con la menor caída de presión. Una de sus desventajas principales consiste en que resulta imposible desarmar el recubrimiento del quemador sin desmontar totalmente el motor, además de su tendencia a torcerse y su debilidad estructural.

El tipo *mixto* de anular y recipiente figura 2.24 está formado por recipientes individuales colocados dentro de una cámara cilíndrica de combustión. Cada recipiente admite aire primario cerca del alabe de combustible y esta perforado de tal manera que pueda admitirse el aire de enfriamiento secundario corriente abajo en relación con la zona primaria, hay habitualmente un tubo de llama que une los recipientes permitiendo que la llama pase de uno a otro durante el arranque. Este tipo utiliza el área transversal con mayor efectividad que la de tipo recipiente y permite el empleo de un cierto número de llama que eliminan los problemas de desarrollo inherente a la cámara de combustión anular. Este diseño brinda una distribución razonablemente pareja de temperatura a la entrada de las toberas de las turbinas, con lo que se reduce el peligro de los puntos calientes.

Toberas de Escape

En el diseño de los aviones se ha utilizado muchos tipos de toberas entre las que sobresalen:

–*Toberas convergentes de área fija:* El sistema más simple de escape es la tobera convergente de área fija, este tipo de sistema de escape no tiene partes móviles no requiere mecanismos de control y por lo general se utiliza en los aviones comerciales subsónicos. La tobera de área fija se diseña para una relación de expansión y un flujo de masa.

–*Toberas convergentes – divergentes de área fija:* Esta tobera agrega peso, longitud y probablemente arrastre al sistema de escape, lo que puede anular cualquier impulso adicional que se hubiera logrado al utilizar una tobera convergente – divergente de área fija.

–*Toberas convergentes – divergentes de área variable:* En una tobera de área variable se requiere utilizar alguna tipo de mecanismo de diafragma iris con los controles necesarios para lograr la variación en el área.

–*Toberas de tapón:* Otro tipo de tobera que se ha investigado ampliamente cuando se requiere una tobera de área variable es la tobera tipo tapón, el área de la garganta puede variarse mediante el movimiento axial del tapo, del recubrimiento exterior o mediante un dispositivo de diafragma iris, la desventaja más importante de este tipo consiste en que requiere enfriamiento y que estructuralmente resulta débil.

–*Toberas bidimensionales:* Este tipo de toberas cuenta con la posibilidad de un tiro reducido en crucero y un impulso vectorial para lograr un mayor facilidad de manipulación, siendo por otra parte más simples de producir, sin embargo tienen un peso mayor y una eficiencia demás reducida a velocidades subsónicas.

Las toberas de escape de un motor de turbina de gas deberá cumplir con:

1. Estar acoplada a los demás componentes del motor para todas las condiciones operativas del mismo.
2. Brindar la relación optima de expansión.
3. Tener el mínimo de perdida al funcionamiento en condiciones de diseño y fuera de diseño.
4. Ofrecer arrastre bajo.
5. Suministrar impulso inverso si es necesario.
6. Ser capaz de incluir materiales que absorban el ruido.

2.6. Sistemas y accesorios

Dentro de los accesorios de la turbina de gas se encuentran básicamente el sistema de arranque, sistema de ignición, sistema de combustible, sistema de lubricación, sistema de enfriamiento del aire de entrada, sistema de inyección de agua o vapor y sistema de inyección de amoniaco. Los sistemas de accesorios son considerados como manejables cuando estos se encuentran conectados directamente a la flecha de la turbina de gas. Sin embargo usualmente solo uno o dos accesorios son conectados directamente. Los accesorios que se conectan indirectamente usualmente utilizan motores eléctricos, hidráulicos para potencia

Sistema de arranque

Los sistemas de arranque se dividen en dos categorías: aquellos que manejan el generador de gas directamente y aquellos que manejan el generador de gas a través de in caja de engranes intermedia. Los arrancadores pueden ser motores de diesel o gasolina, turbinas de gas, eléctricas, hidráulicas o neumáticos. Los sistemas de arranque satisface dos funciones independientes la primera es rotar el generador de gas hasta encontrar su velocidad para sostenerse y la segunda es para manejar el compresor del generador de gas para purgar el generador de gas y el ducto de escape de cualquier gas volátil antes de iniciar el ciclo de ignición.

La secuencia de arranque es:

- Arranque de enganche
- Purga de entrada y ducto de escape
- Energizar el encendedor
- Encendido de combustible

La función primaria del sistema de arranque es para acelerar el generador de gas desde el reposo hasta un punto justo más allá de la velocidad de auto-sustentación del generador de gas. El arrancador tiene que desarrollar el suficiente torque para superar el torque de arrastre del compresor y la turbina del generador de gas, así como cualquier carga adjunta incluyendo cargas de accesorios y resistencia de rodamientos.

Otra función del sistema de arranque es rotar el generador de gas después del paro, para acelerar el enfriamiento. Las funciones de la purga y del enfriamiento han llevado a la

utilización de dos velocidades de arranque. La velocidad baja es utilizada para la purga y el enfriamiento y la velocidad alta es utilizada para arrancar la unidad.

El generador de gas es arrancado por los siguientes modos:

1. Arrancador directamente conectado al eje del compresor.
2. Arrancador indirectamente conectado a la flecha del compresor por medio de la caja de engranes.
3. Choque de aire dirigido dentro del compresor o la turbina del compresor.

Los dispositivo utilizados para arrancar el generador de gas pueden ser motores eléctricos, motores neumáticos, motores hidráulicos, motores diesel o pequeñas turbinas de gas.

Sistema de ignición

La ignición es uno parte de los sistemas de la turbina de gas que a menudo se da por aceptado, hasta que un problema se desarrolla. Cada que surge un problema el sistema de ignición no es el primer sistema del que uno se preocupa. Una razón es que este sistema se ha perfeccionado dentro de uno de los más confiables sistemas en un paquete de turbina de gas. Otra razón es que la ignición solo es requerida durante el arranque. Una vez que el generador de gas ha sido acelerado hasta la velocidad de auto-sustentación el sistema puede y usualmente es des-energizada.

Es una práctica estándar el instalar dos encendedores en cada motor. Un encendedor es instalado en cada lado del motor, aunque normalmente no separados más de 180 grados. Este enfoque de diseño es aplicado para todos los diseños de cámaras de combustión. El sistema de ignición consta de dos unidades idénticas, unidad de ignición independiente (excitador) con una fuente de potencia eléctrica común.

Una vez que el generador de gas ha sido arrancado la función del encendedor es cumplida y cualquier exposición a los gases calientes de la combustión acortara su vida útil. Par eliminar esta exposición algunos diseños de encendedores incluyen mecanismo de resorte de retracción que permite al encendedor moverse hacia afuera del conducto de flujo como resultado del incremento de presión de la combustión.

Sistema de lubricación

El sistema de lubricación en las turbinas de gas tiene que cumplir dos requerimientos importantes principalmente: uno es proveer lubricante entre las superficies de los rodamientos rotatorios y estacionarios, y la otra es remover el calor de estas superficies. El diseño de los sistemas de lubricación tiene que considerar tipos de rodamientos, rpm, carga, temperatura y viscosidad del aceite. Los tipos de rodamientos entran en dos categorías principales: rodamientos hidrodinámicos y rodamientos anti-fricción.

El lubricante en un rodamiento hidrodinámico cubre la fricción de deslizamiento dentro de la fricción del fluido. Mientras la fricción del fluido es considerada baja, esto suele consumir potencia y producir calor. Por lo tanto, además provee una película de fluido que separa las superficies, el aceite lubricante también remueve el calor del área.

Lo rodamientos anti-fricción trabajan en el principio de fricción giratoria. La carga del eje es soportada directamente por los elementos rodantes y su pista en contacto metal con metal. Además, las altas unidades de presión entre lo elementos rodantes y las pistas interior y exterior impiden la formación de una película de aceite in-rompible. Por lo tanto, el trabajo del aceite lubricante es un poco diferente. El aceite lubricante suele requerirse para remover calor de las aéreas de los rodamientos y este reduce la fricción del deslizamiento entre los elementos rodantes y la pista que mantiene la posición de estos.

El servicio del motor y el tipo de rodamientos seleccionados por el diseñador del motor, determinan el tipo del aceite de lubricación utilizado en el sistema. En el marco de las turbinas de gas más pesadas, en las cuales incorporan rodamientos hidrodinámicos utilizan aceite mineral, mientras que en las turbinas de gas aeroderivativas en las cuales incorporan rodamientos anti-fricción utilizan aceites sintéticos.

2.7. Fallas de turbinas de gas

Los modos de fallas más comunes en una turbina de gas se presentan en la figura 2.26 y la tabla 2.1.

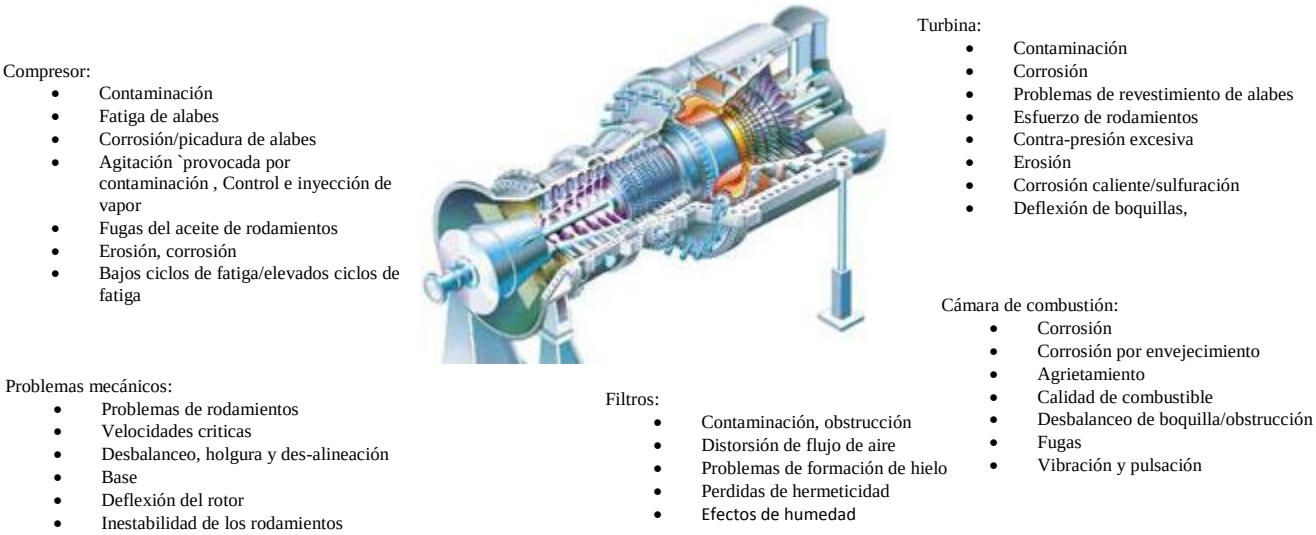


Figura 2.26 Problemas más comunes de la Turbina de Gas.

Compresor

Componente	Modo de falla	Causas
Alabes de rotor y estator	Fatiga (resonancia) Erosión, coque anticorrosivo, choque,	Vibración, distorsión del flujo de aire, agitación, polvo en el aire.
Disco	Fatiga- deslizamiento, desgaste, fricción	Efectos de temperatura de carga centrifuga.
Pernos de sujeción del compresor	Fatiga mecánica, desgaste por envejecimiento y por fricción	Puesta en marcha, trabajo cíclico, vibración.

Cámara de combustión

Componente	Modo de falla	Causas
Carcasa interior	Fatiga mecánica, desgate por fatiga térmica, distorsión y corrosión térmica	Puntos calientes, gradientes de temperatura, pulsaciones de presión dinámica.
Carcasa	Fatiga	Ciclos de presión.
Tubos de fuego cruzado	Desgaste, desgate por fatiga térmica	Pulsaciones y vibraciones.
Pieza de transición	Fatiga térmica, desgaste,	Pulsaciones y vibraciones dinámicas.

Sección de Turbina

Componente	Modo de falla	Causas
Alabes del rotor de la turbina	Fatiga de ciclos elevados, deslizamiento, corrosión, sulfuración, erosión	Esfuerzo centrifugo y temperatura, esfuerzo de vibraciones, ambientales, problemas de combustible, propagación de temperatura excesiva, problemas de enfriamiento.
Alabes del estator de la turbina	Corrosión por ruptura de deslizamiento, sulfuración, deflexión, fatiga, fatiga térmica.	Problemas de enfriamiento, perfile de temperatura inapropiado.
Disco del rotor de la turbina	Ruptura por deslizamiento, fatiga de frecuencia baja.	Espacio de enfriamiento inapropiado, esfuerzo térmico.

Tabla 2.1 Aéreas de problemas y modos de fallas en la turbina de gas.

Examinando el rango de los problemas, es claro que los sistemas de monitoreo de las condiciones completa debe abarcar e integrar una variedad de enfoques de monitoreo de las condiciones. Algunas de las fallas de las turbinas de gas se pueden observar en las figuras 2.27, 2.28, 2.29 y 2.30.

Figura 2.27 Destrucción de los alabes del compresor.

Figura 2.28 Erosión en los alabes de compresor.

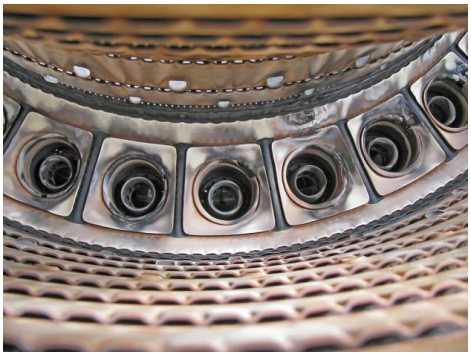


Figura 2.29 Daño por sobrecalentamiento de la cámara de combustión.

Figura 2.30 Daño en los inyectores de combustible de la cámara de combustión.

2.8. Diagnóstico paramétrico

Como ya se ha mencionado en la sección anterior un motor de turbina de gas puede ser considerado como una maquina compleja y cara. Es por esto que es de vital importancia que las turbinas de gas deben de contar con un sistema de monitoreo efectivo. [16, 17, 18]. Los sistemas de monitoreo de la turbina de gas están basado en variables medidas y registradas. Tales sistemas no necesitan parar ni desarmar el motor. Estos operan en tiempo real y proporcionan un análisis diagnóstico en línea y diagnóstico especiales basados en datos registrados. Con la base de datos, el análisis más profundo fuera de línea es realizado posteriormente.

El sistema puede utilizar toda la información disponible para un diagnóstico de la turbina de gas y cubrir un número máximo de sus subsistemas. Pero en bases teóricas para diagnosticar diferentes sistemas del motor puede ser común, cada uno de ellos requiere su propio algoritmo diagnóstico tomando en cuenta particularidades del subsistema. Hoy en día los diagnósticos paramétricos incluyen todos los principales subsistemas de la turbina de gas como son el conducto de flujo, transmisión, construcción de los elementos de las partes calientes, sistemas de mediciones, sistema de combustible, sistema de aceite, sistema de control, sistema de arranque, y sistema de geometría variable del compresor.

2.8.1 Clasificación de métodos

El sistema de monitoreo siempre incluye una técnica específica para turbinas de gas, el análisis del conducto de flujo (GPA- Gas Path Analysis). Este enfoque está basado en un buen desarrollo teórico de la turbina de gas y mediciones del conducto de flujo. El GPA puede considerarse como la parte principal de un sistema de monitoreo de la turbina de gas. El análisis del conducto de flujo ha sido elegido en esta tesis como un enfoque representativo de la diagnosis de la turbina de gas.

El GPA proporciona una amplia penetración al rendimiento de componentes, revelando mecanismos de degradación gradual y fallas abruptas. Además los defectos del conducto de flujo, mal funcionamiento de mediciones y sistema de control pueden ser detectados e identificados. Por otro lado permite estimar el rendimiento de partes principales del motor que no pueden ser medidas como son la potencia del eje, empuje, eficiencia global del

motor, consumo específico de combustible y margen de aumento del compresor. Una clasificación de las técnicas del análisis del conducto de flujo se ilustra en la figura 2.30

Como se puede observar en la figura 2.31, la técnica del análisis del conducto de flujo se puede dividir en tres grandes etapas, que son etapas del proceso de diagnóstico, bases teóricas y modelos utilizados, cada uno de ellos nos proporcionan una ayuda esencial en el análisis global. Estas tres etapas importantes e interconectadas estas son: monitoreo o detección de problemas, diagnóstico o identificación de fallas y pronostico, sin embargo como se ilustra estas etapas requieren de una etapa preliminar la cual se encarga de analizar datos reales y calcular los cambios en las variables monitoreadas a causa del envejecimiento o fallas del motor, a esta etapa se le conoce como validación de datos o desviaciones [19].

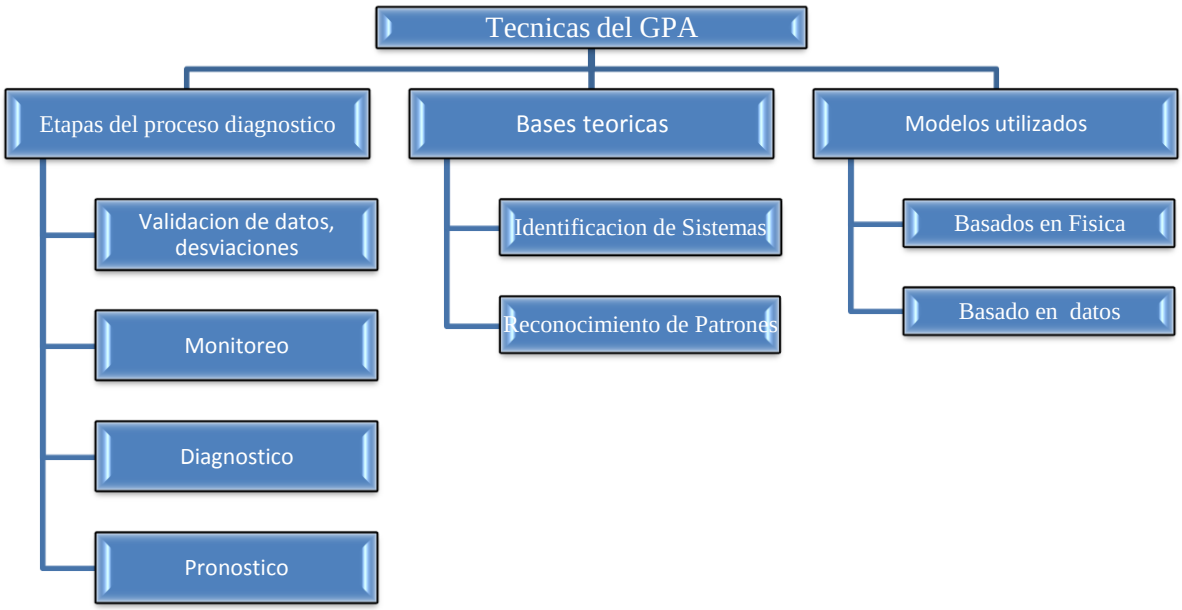


Figura 2.31 Clasificación de las técnicas del análisis del conducto de flujo

La eficiencia de un sistema de monitoreo automatizado depende de la calidad de las desviaciones, debido a que estas son las discrepancias entre las variables medidas de un motor en mal estado y las variables de un motor sin daño alguno (motor nuevo o reparado)

En lo que se refiere a los métodos del diagnóstico de turbinas de gas, estos pueden ser divididos en dos enfoques generales. El primer enfoque emplea técnicas de identificación de sistemas y en general es llamado modelo termodinámico. Los modelos aplicados relacionan las variables del conducto de flujo monitoreadas con parámetros de fallas especiales que permiten simular degradación de los componentes del motor. El objetivo de identificar la turbina de gas es encontrar los parámetros de fallas que minimicen la diferencia entre las variables del modelo generado y las variables monitoreadas medidas. La simplificación del proceso diagnóstico es logrado porque los parámetros determinados contienen información del estado técnico actual de cada componente. La principal limitación es que la inexactitud del modelo genera errores elevados en parámetros de fallas estimados.

El segundo enfoque es basado en la teoría de reconocimiento de patrones y principalmente utiliza los modelos basados en datos. La clasificación de fallas necesaria puede ser compuesta en la forma de patrones obtenidos para cada clase de falla apartar de datos reales. Cuando los patrones de cada clase de fallas están disponibles, las técnicas de reconocimiento basados en datos, como son las redes neuronales, pueden ser fácilmente entrenadas sin conocimiento detallado del sistema. Es por eso que este enfoque tiene una posibilidad teórica de excluir el modelo del proceso de diagnóstico.

Por otro lado tenemos los modelos empleados en el monitoreo de las condiciones y en particular en el GPA pueden ser agrupados en dos grande categorías basados en física y basados en datos. Los modelos basados en física (por ejemplo modelo termodinámico) requiere conocimiento detallado del sistema que se encuentra analizando (turbina de gas) y generalmente presentan un software complejo. Los modelos basados en datos dan una relación entre las variables de entrada y salida los cuales pueden ser obtenidos en base de datos reales disponibles sin la necesidad de conocer el sistema. Las técnicas del diagnóstico pueden clasificarse de la misma manera como; basados en física (basados en modelos) y basados en datos (empíricos) [20].

2.8.2. Modelo Polinomial del Estado Normal

Se puede representar las características climáticas y las de operación de una turbina de gas en un estado normal (nominal) curvas de operación, las cuales pueden ser descritas por polinomios y presentar el modelo de estado normal. Se ha demostrado en [19] que un polinomio completo de segundo orden proporciona la aproximación suficiente del rendimiento del motor (referencia). Para una variable monitoreada del conducto de flujo Y como una función de tres argumentos u_i (variable de operación), el polinomio se describe a continuación:

$$Y_0(\vec{U}) = a_0 + a_1u_1 + a_2u_2 + a_3u_3 + a_4u_1u_2 + a_5u_1u_3 + a_6u_2u_3 + a_7{u_1}^2 + a_8{u_2}^2 + a_9{u_3}^2 \qquad (2.1)$$

Los polinomios para todas las variables monitoreadas pueden estar dados por la siguiente expresión:

$$\overrightarrow{{Y_0}^T} = \overrightarrow{V^T}A \qquad (2.2)$$

Donde $\overrightarrow{{Y_0}^T}$ es un vector de $(1 \times m)$ de variables monitoreadas, $\overrightarrow{V^T}$ es una vector de $(1 \times k)$ de componentes $1, u_1, u_2, \dots, {u_1}^2, {u_2}^2$ y A representa una matriz de $(k \times m)$ de coeficientes desconocidos a_i para todas las variables monitoreadas m. Sin embargo las mediciones de un punto de operación en estado estacionario no son suficientes para calcular los coeficientes, datos recabados a n diferentes puntos de operación son incluidos dentro del conjunto de entrenamiento e involucrados dentro del cálculo. Con las nuevas matrices $Y(n \times m)$ y $V(n \times k)$ formadas a partir de estos datos, la ecuación anterior se transforma en un sistema lineal:

$$Y = VA \qquad (2.3)$$

En este modelo, hay que encontrar los coeficientes indeterminados a, para lo cual se crea una muestra de aprendizaje con una parte de los datos base y se aplica el método de

mínimos cuadrados. La función que se obtiene se verifica por promedios de la desviación obtenida en una muestra de prueba en la que se emplean todos los datos disponibles. Este modelo tiene la gran desventaja de que no es capaz de simular fallas.

Como se puede observar, el modelo de estado normal (función de referencia) ecuación 2.2 presenta un típico modelo basado en datos, debido a que solo los datos de las entradas y las salidas son utilizados para calcular los coeficientes del polinomio.

2.8.3. Modelo estático no lineal

Este modelo [20] nos permite calcular los parámetros $\vec{Y}$ de una turbina en diversos regímenes estacionarios y es el más exacto, ya que nos muestra el proceso más real. El vector $\vec{Y}$ de los parámetros del conducto de flujo, depende del vector $\vec{U}$ de los parámetros del régimen de motor y del vector $\vec{\theta}$ de los parámetros de estado del conducto de flujo. Su expresión matemática es la siguiente:

$$\vec{Y} = F(\vec{\theta}, \vec{U}) \tag{2.4}$$

Donde:

$\vec{Y} =$ Son las variables medidas, monitoreadas o diagnosticadas.

$\vec{\theta} =$ Es el conjunto de parámetros de estado. Los parámetros de estado nos sirven para definir el estado, no son el estado mismo. Los parámetros $\vec{\theta}$ son seleccionados con la intención de que puedan desplazar o deformar las características de los componentes del motor y ajustar el desempeño del modelo al desempeño real del motor de turbina de gas ó para simular fallas.

$\vec{U} =$ Es el conjunto de parámetros de control del motor y parámetros ambientales. Este vector depende del régimen de operación del motor. En el modelo tenemos la libertad de cambiar estos parámetros del conducto de flujo.

Este modelo nos permite realizar lo siguiente

1. Calcular los parámetros $\vec{Y}$ en cualquier régimen y para cualquier estado del motor.
2. Características de estrangulación (variando el consumo de combustible) y características climáticas (variando las características atmosféricas).
3. Identificar el estado normal de la turbina.
4. Simulación de fallas.

2.8.4. Modelo estático lineal

El modelo estático lineal presenta linealización de dependencia no lineal $\vec{Y} = F(\vec{\theta})$ entre las variables del conducto de flujo y los parámetros de falla determinados para una condición de operación fija $\vec{U}$. Es decir, este modelo relaciona las pequeñas desviaciones relativas de los parámetros de estado $\delta\vec{\theta}$ con la desviación relativa de las variables del

conducto de flujo $\delta\vec{Y}$, mediante la matriz de coeficientes *H*, alrededor de un régimen estacionario del motor (régimen diagnóstico).

La representación matemática de este modelo es la siguiente:

$$\delta\vec{Y} = H\delta\vec{\theta} \tag{2.5}$$

Ya que los errores de la linealización no son tan grandes, sobre algún uno por ciento, el modelo lineal puede ser satisfactoriamente aplicado para simular fallas en cualquier punto de operación fijo.

La matriz puede ser fácilmente calculada por medio del modelo termodinámico. Las variables del conducto de flujo $\vec{Y}$ son primeramente calculadas por el modelo para parámetros de fallas nominal $\overrightarrow{\Theta_0}$. Después, pequeñas variaciones son introducidas para cada turno en parámetros de fallas y el cálculo de las variables $\vec{Y}$ es repetido para cara parámetro corregido. Finalmente, para cada conjunto Y_i y Θ_j el correspondiente coeficiente de influencia es obtenido por la siguiente expresión:

$$H_{ij} = \frac{\delta Y_i}{\delta \Theta_j} = \frac{Y_i(\overrightarrow{\Theta_i}) - Y_i(\overrightarrow{\Theta_0})}{Y_i(\overrightarrow{\Theta_0})} \Big/ \frac{\Theta_j - \Theta_{0j}}{\Theta_{0j}} \tag{2.6}$$

2.8.5. Identificación del modelo no lineal

Debido a errores de simulación, sobre todo inexactitudes en las características de los componentes. Los valores de las mediciones reales $\overrightarrow{Y^*}$ difieren de los valores del modelo $\vec{Y}$, y el objetivo de la identificación del modelo es el de encontrar aquel estado de estimación de parámetros $\vec{\theta}$, en el cual, el nivel de error sea mínimo [21]. Por esta razón, la tarea de identificación del modelo de la ecuación 1.5 puede ser clasificada como un problema de optimización, en el que las estimaciones pueden ser encontradas mediante las siguientes expresiones matemáticas.

$$\hat{\vec{\theta}} = argmin \left\| \overrightarrow{Y^*} - \vec{Y} \right\| \tag{2.7}$$

$$\hat{\vec{\theta}} = argmin \left\| \overrightarrow{Y^*} - Y\left[\vec{\theta}, \vec{U}\right] \right\| \tag{2.8}$$

2.9. Proceso del diagnóstico paramétrico de turbinas de gas

Como se ha mencionado anteriormente el análisis diagnóstico del conducto de flujo comprende tres componentes generales e interrelacionados entre sí, los cuales son: el común monitoreo de las condiciones del motor, diagnóstico detallado y pronostico de la vida del motor.

Sin embargo como las fallas afectan las variables monitoreadas y registradas, el impacto de los cambios en las condiciones de operación es mucho más grave. Es por esto que la ejecución de los componentes antes mencionados siempre es procedida por una operación preliminar importante, la cual es el cálculo de las desviaciones [22, 23].

2.9.1. Cálculo de las desviaciones

La eficiencia de un sistema automatizado de monitoreo depende de la calidad de las desviaciones. Aunque los efectos de deterioro del motor afectan las mediciones y las variables monitoreadas del conducto de flujo, el impacto de los cambios en operación y las condiciones ambientales son mucho más fuertes. Sin embargo, si para una variable monitoreada Y_i substraemos un valor de referencia Y_{0i} de un valor medido actual Y_i^*, la diferencia $Y_i^* - Y_{0i}$ será prácticamente libre de influencia de estas condiciones, conservando un efecto de deterioro. Típicamente esta diferencia (desviación) es dada en una forma relativa. Por lo tanto una desviación relativa δY_i^* puede ser calculada de acuerdo a la expresión:

$$\delta Y_i^* = \frac{Y_i^* - Y_{0i}}{Y_{0i}} \tag{2.9}$$

Dado que variables monitoreadas en una turbina de gas dependen del punto de operación (régimen) del motor, los valores de referencia para una turbina de gas en operación estacionaria se presentan como: una función $\overrightarrow{Y_0} = f(\vec{U})$ de las variables de operación y las ambientales $\vec{U}$ [6]. Mencionando lo anterior escribimos la desviación en la forma siguiente:

$$\delta Y_i^* = \frac{Y_i^* - Y_{0i}(\vec{U})}{Y_{0i}(\vec{U})} \tag{2.10}$$

La calidad de las desviaciones calculadas conforme a la formula (2.10) será dependiente de la adecuación de la función de referencia.

2.9.2. Monitoreo

El primer componente mencionado dentro del análisis diagnóstico de las condiciones es el monitoreo, el cual tiene por objetivo solo determinar si el motor puede seguir trabajando bajo las circunstancias de operación actual o no puede seguir operando bajo estas condiciones. Para determinar lo anterior se aplican los siguientes métodos del monitoreo.

– Monitoreo por valores extremos: consideremos la figura 2.32, como se observa solo determinamos un valor máximo y un valor mínimo de cualquier variable monitoreada, es decir, para cada variable se determina el rango de operación en el cual el motor trabajara de forma aceptable. Con esto simplemente después se determina si la variable se encuentra en rango de operación para decir que si puede o no seguir trabajando el motor.

$$Y_{min} < Y^* < Y_{max} \tag{2.11}$$

– Monitoreo por las desviaciones de la norma (se usa la norma individual o de la serie): se determina la norma (ecuación 2.12) según la variables considerada, posteriormente de la misma manera que el método anterior solo se determina si es normal o no es normal este valor para que el motor siga trabajando.

$$\left|Y^* - Y_0(\vec{U})\right| < \Delta Y \tag{2.12}$$

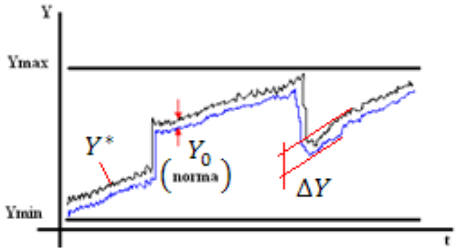


Figura 2.32 Tipos de monitoreo

– Monitoreo por los parámetros de los motores contiguos: normalmente se tiene trabajando varios motores de turbinas de gas de las mimas características, por lo que es normal y muy útil comparar sus variables monitoreadas y con esto se determina si es posible o no es posible que el motor continúe trabajando.

Según lo antes mencionado el valor Y de parámetro diagnóstico puede estar en uno de dos estados, D_0 (buen estado cuando $Y < Y_{lim}$) y D_1 (estado defectuoso cuando $Y > Y_{lim}$), donde Y_{lim} es el límite del buen estado del motor. Sin embargo, la decisión diagnóstica d_0 o d_1 se ejecuta por el parámetro medido Y^* que contiene los errores aleatorios

	D_0	D_1
d_0	$P(d_0/D_0)$	$P(d_0/D_1)$
d_1	$P(d_1/D_0)$	$P(d_1/D_1)$
	$P(d_0/D_0) + P(d_1/D_0)$ =1	$P(d_0/D_1) + P(d_1/D_1)$ =1

Tabla 2.1 Tabla de Probabilidad de monitoreo.

Por lo tanto hay dos probabilidad de error dentro de este monitoreo: la probabilidad del fallo falso $P_{ff} = P_{10} = P(d_1/D_0)$ y la del fallo omitido $P_{fo} = P_{01} = P(d_0/D_1)$. Podemos formar el criterio de errores según ecuación 2.13 y 2.14. El error del fallo omitido es muy grave.

$$C = C_{10}P_{10} + C_{01}P_{01} \tag{2.13}$$

$$\bar{C} = \frac{C}{C_{10}} = P_{10} + \frac{C_{01}}{C_{10}}P_{01} \tag{2.14}$$

2.9.3 Diagnóstico detallado

La siguiente etapa del análisis diagnóstico de las condiciones es la identificación (reconocimiento) de las fallas de turbinas de gas. Aquí damos una breve descripción de nuestro enfoque del reconocimiento.

Esta etapa del proceso total de monitoreo se basa comúnmente en la teoría de reconocimiento de patrones. Esta teoría supone una clasificación de los estados reconocidos de un sistema. En general se acepta la hipótesis de que el estado de un sistema puede pertenecer sólo a una de q clases.

$$D_1, D_2, ..., D_q \tag{2.15}$$

Predeterminadas de antemano. Aceptamos esta hipótesis para la clasificación de fallas de turbinas de gas. Existen varios principios para crear tal clasificación, el principio más común (2.15) dice que cada clase corresponde a uno de los componentes del motor (compresor, cámara de combustión, turbinas, etc.). Por lo tanto el esquema estructural del motor determina la estructura de clasificación de fallas.

Para simular las fallas del motor analizado, se aplica su modelo termodinámico del a ecuación 2.4, el cual calcula las variables monitoreadas como una función de las variables de operación y un vector de parámetros de falla $\vec{\Theta}$.

Los elementos del vector $\vec{\Theta}$ desplazan los mapas de características de los componentes en direcciones dadas, simulando fallas de estos componentes.

En el proceso de simulación de cada clase de fallas por el modelo termodinámico, una desviación normalizada Z_i^* para una variable monitoreada Y_i se compone por dos componentes: un componente sistemático Z_i y un componente aleatorio ε_i, p.e. $Z_i^* = Z_i + \varepsilon_i$. El componente sistemático $Z_i = \frac{Y_i - Y_{0i}}{Y_{0i} - \delta Y_{imax}}$ es inducido por fallas implantadas en el modelo. Cambio de cada parámetro particular de fallas produce una clase de fallas de severidad variable. El componente ε_i toma en cuenta los errores aleatorios en las desviaciones y se cambia dentro del intervalo (-1,1). Las desviaciones de todas las variables monitoreadas forman el vector $\vec{Z}^*$, el cual es un patrón para ser reconocido cuando se lleva a cabo el diagnóstico. En el espacio de las desviaciones, cada clase es representada estadísticamente por su propio conjunto de patrones $\vec{Z}^*$. La severidad de falla es dada por la distribución uniforme del parámetro correspondiente de falla en el intervalo (0, -5%), mientras que los errores ε_i son generados de acuerdo a la distribución Gaussiana.

2.9.4. Pronostico

El tercer componente del análisis diagnóstico de las condiciones es el pronóstico de la vida del motor, la predicción de la vida restante del motor es basado en el método del análisis de las tendencias. En general las desviaciones son calculadas con información real, datos de campo o pruebas. Subsecuentemente, aproximaciones de desviaciones lineales o no lineales dan funciones para las condiciones del motor y la predicción de la vida restante del motor, ver figura 2.34.

$$|Y_i - Y_{i-1}| < [\Delta Y] \tag{2.16}$$

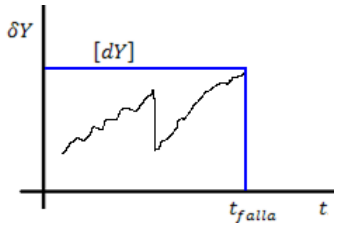
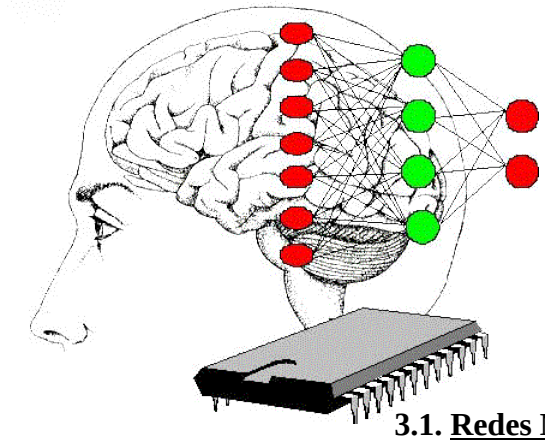


Figura. 2.33 Tipos de monitoreo

Capítulo 3.-

Redes Neuronales Aplicadas al Monitoreo



3.1. Redes Neuronales Artificiales

3.1.1. Introducción

El científico Santiago Ramón y Cajal fue el descubridor de la estructura neuronal del sistema nervioso, demostró en 1888 que el sistema nervioso estaba compuesto por una red de células individuales, las neuronas, las cuales se encontraban ampliamente conectadas entre ellas. Además estableció que la información fluye en la neurona desde las dendritas hacia el axón, atravesando el soma [24].

Las redes neuronales artificiales son sistemas, hardware o software, de procesamiento que copian esquemáticamente la estructura neuronal del cerebro para tratar de reproducir sus capacidades. Las redes son capaces de aprender de la experiencia a partir de las señales o datos provenientes del exterior, dentro de un marco de computación paralela y distribuida, fácilmente implementable en dispositivos hardware específicos.

Se estima que el sistema nervioso contiene aproximadamente cien mil millones de neuronas, aunque muchas de ellas presentan un aspecto similar muy peculiar, figura 3.1, con un cuerpo celular o soma, del cual surge un denso árbol de ramificaciones compuesto por las dendritas y del cual parte una fibra tubular denominada axón, el cual también se ramifica en su extremo final para ser conectado por otras neuronas. La unión entre dos neuronas se conoce como sinapsis, el contacto físico entre las neuronas es nulo en el tipo más común de sinapsis, además que esta es direccional, es decir que la información fluye siempre en un solo sentido.

Las Redes Neuronales Artificiales (Artificial Neural Networks – ANN's) se constituyeron inicialmente como una simulación abstracta de los sistemas nerviosos biológicos formados por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. Las conexiones de estos nodos se asemejan a las dendritas y axones de los sistemas nerviosos biológicos.

Por lo tanto las Redes Neuronales Artificiales (RNA) son una topología de varios elementos de procesamiento, los cuales efectúan su operación en forma paralela, sus elementos están conectados entre sí a través de un canal unidireccional. Se trata de un sistema de interconexión de neuronas en una red que colabora para producir un estímulo de salida. Las RNA son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso. Se trata de un sistema de interconexión de neuronas en una red que colabora para producir un estímulo de salida.

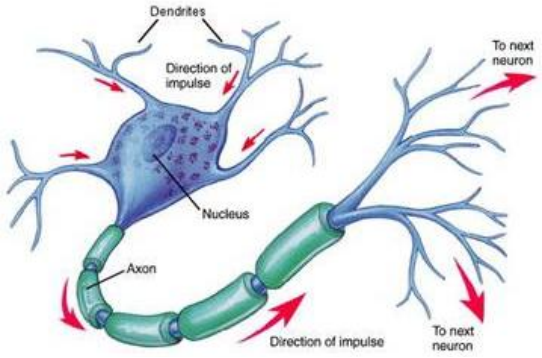


Figura 3.1 Estructura neuronal del ser humano.

Como ya se mencionó, las RNA imitan la estructura del sistema nervioso, con la intención de construir sistemas de procesamiento de la información paralelos, distribuidos y adaptativos, que puedan presentar un cierto comportamiento inteligente. Curiosamente las neuronas son mucho más simples, lentas y menos fiables que un computador, y a pesar de ello existen problemas difícilmente abordables mediante el computador convencional, que el cerebro resuelve eficazmente. Son tres conceptos claves de los sistemas nerviosos que se pretenden emular en los sistemas artificiales, paralelismo de cálculo, memoria distribuida y adaptabilidad, al entorno. De esta manera podemos hablar de las redes neuronales como sistemas paralelos, distribuidos y adaptativos.

Como ya se menciono antes los elementos básicos de un sistema neuronal biológico son las neuronas, las cuales se agrupan en millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia. Ahora en al realizar un sistema artificial puede establecerse una estructura jerárquica similar. Siendo el elemento esencial de partida la neurona artificial, que se organizara en capas, con varias capas construiremos una red neuronal, por ultimo una red neuronal, junto con las interfaces de entrada y salida, mas los módulos convencionales adicionales necesarios, constituirán el sistema global de proceso figura 3.2.

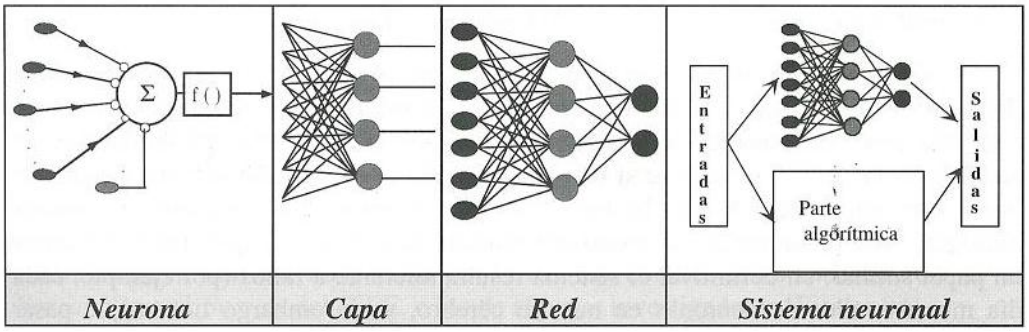


Figura 3.2 Estructura jerárquica de un sistema basado en ANS.

3.1.2. Modelo general de la neurona y su clasificación

Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que a partir de un vector de entrada procedente del exterior o de otras neuronas [24]. proporcionan una única respuesta o salida. Los elementos que constituyen la neurona *i* son los siguientes de acuerdo con la figura 3.3 [25].

- Conjunto de entradas, $x_j(t)$.
- Pesos sinápticos de la neurona *i*, w_{ij} que representan la intensidad de interacción entre neuronas pre-sináptica *j* y la neurona pos-sináptica *i*.
- Regla de propagación $\sigma(w_{ji}, x_j(t))$, que proporciona el valor del potencial pos-sináptico $h_i(t) = \sigma(w_{ji}, x_j(t))$ de la neurona *i* en función de sus pesos y entradas.
- Función de activación $f_i(a_i(t\text{-}1), h_i(t))$, que proporciona el estado de activación actual $a_i(t) = f_i(a_i(t\text{-}1), h_i(t))$ de la neurona *i*, en función de su estado anterior $a_i(t\text{-}1)$ y de su potencial pos-sináptico actual.
- Función de salida $F_i(a_i(t))$, que proporciona la salida actual $y_i(t) = F_i(a_i(t))$ de la neurona *i*en función de su estado de activación.

De este modo la operación de la neurona *i* puede expresarse como:

$$y_i(t) = F_i(f_i[a_i(t-1), \sigma_i(w_{ij}, x_j(t))]) \tag{3.1}$$

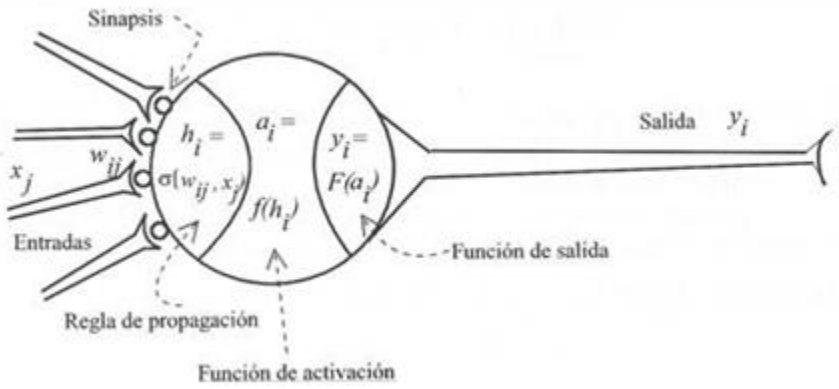


Figura 3.3 Modelo genérico de neurona artificial.

– *Entradas y Salidas*

Las variables de entrada y salida pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación.

– *Regla de propagación*

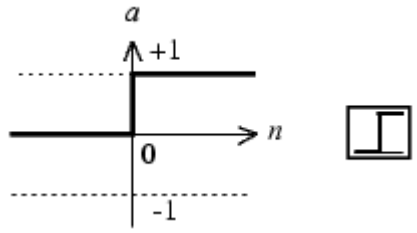
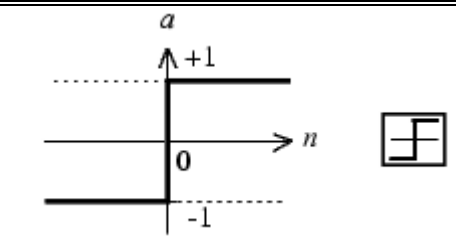
Permite obtener, a partir de las entradas y los pesos, el valor de la potencia pos-sináptico h_i de la neurona. La función más habitual es de tipo lineal y se basa en la suma ponderada de las entradas con los pesos sinápticos, que formalmente puede interpretarse como es el producto escalar de los vectores de entrada y pesos.

El peso sináptico w_{ij} define en este caso la intensidad de interacción entre la neurona pre-sináptica j y la pos-sináptica i. si el peso es positivo tendera a excitar a la neurona pos-sináptica, si el peso es negativo tendera a inhibirla.

– *Función de activación o función de transferencia*

La función de activación o de transferencia proporciona el estado de activación actual a partir del potencial pos-sináptico y del propio estado de activación anterior. La función de activación se suele considerar determinista y en la mayor parte de los modelos es monótona creciente y continua, como se observa habitualmente en las neuronas biológicas. La forma de las funciones de activación mas empleadas en las ANS se muestra en la tabla 3.1.

La función de transferencia es un modelo matemático de un sistema dinámico el cual se define como un conjunto de ecuaciones el cual representa la dinámica del sistema con precisión o por lo menos aproximada. En redes neuronales existen varias funciones de transferencia las cuales se presentan a continuación:

Función de Transferencia	Expresión Matemática	Aplicación
 **Hardlim**	$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	Para Redes Neuronales Fijas
 **Escalón**	$f(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$	Para Redes Neuronales Fijas

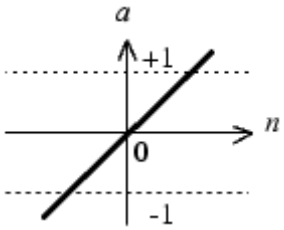
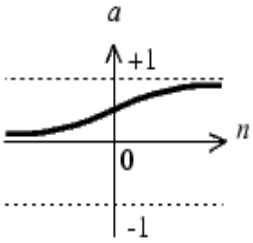
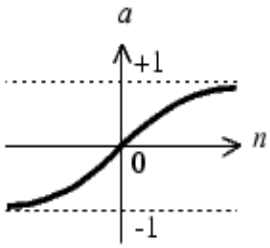
  **Pureline**	$y = f(x) = mx + b$	Nunca se utiliza para Redes Neuronales
  **Logsig**	$f(x) = \frac{1}{1 + e^{-x}}$	Se usa para Redes Neuronales Adaptables
  **Tansig**	$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$	Se usa para Redes Neuronales Adaptables

Tabla 3.1 Funciones de transferencias o activación.

Función de salida

Esta función proporciona la salida global de la neurona $y_i(t)$ en función de su estado de activación actual. Muy frecuentemente la función de salida es simplemente la identidad $F(x) = x$, de modo que el estado de activación de la neurona se considera como la propia salida [26].

Una clasificación de las redes es de acuerdo a su arquitectura, es decir a la topología, estructura o patrón de conexión de una red neuronal. En general, las neuronas se suelen agrupar en unidades estructurales a las cuales denominaremos capas, las neuronas de una capa pueden agruparse, formando de esta manera grupos neuronales. Dentro de un grupo o de una capa si no existe este tipo de agrupación, las neuronas suelen ser del mismo tipo. Finalmente al conjunto de una o más capas constituyen la red neuronal.

En la arquitectura de la red se distinguen tres capas principalmente de entrada, de salida y ocultas. Atendiendo a distintos conceptos, podemos establecer distintos tipos de

arquitecturas neuronales, así en relación a su estructura en capas, podemos hablar de redes monocapa y redes multicapa. Las redes monocapa son aquellas compuestas por una única capa de neuronas. Las redes multicapa son aquellas cuyas neuronas se organizan en varias capas.

Por otro lado, si vemos el flujo de datos en la red neuronal. Podemos hablar de redes unidireccionales y redes recurrentes. En las redes unidireccionales, la información circula en un único sentido, desde las neuronas de entrada hacia las de salida. En las redes recurrentes o realimentadas la información puede circular entre las capas en cualquier sentido, incluido el de salida-entrada.

Y por último, también podemos hablar de redes auto-asociativas y hetero-asosiativas. Con frecuencia se interpreta la operación de una red neuronal como la de una memoria asociativa, que ante un determinado patrón de entradas responde con un cierto patrón de salida.

Dependiendo del modelo de neurona concreto que se utilice, de la arquitectura o topología de conexión y del algoritmo de aprendizaje, surgirán distintos modelos de redes neuronales. De la gran cantidad de modelos y variantes que existen, se necesita una clasificación para podernos dar cuenta de los tipos así como estudiarlos de una forma más precisa, para lo cual adoptamos la propuesta de Simpson [27] ver figura 3.4.

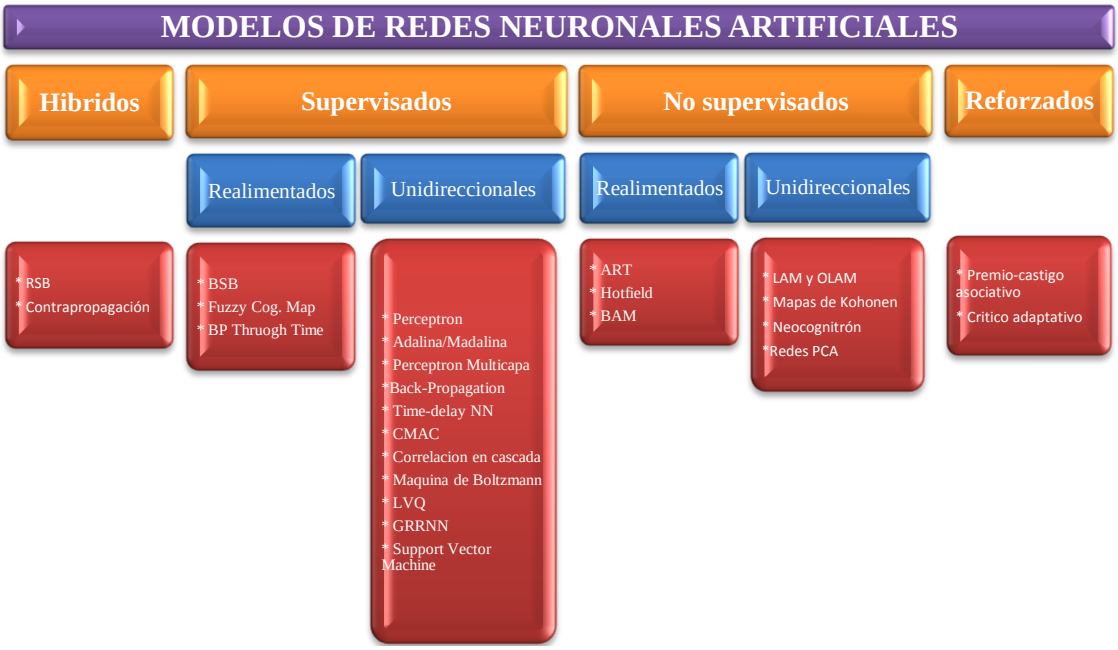


Figura 3.4 Clasificación de las RNA por el tipo de aprendizaje y la arquitectura.

3.2. Perceptrón Multicapa

Las redes unidireccionales organizadas en capas (feed-forward) y con entrenamiento supervisado, son empleadas como clasificadores de patrones y estimadores de funciones, sin embargo esta clasificación como ya se ha visto es muy amplia por lo cual nos centraremos en lo que es de nuestro interés para este trabajo, es decir en el perceptrón

multicapa. De este modo posteriormente veremos el popular algoritmo de aprendizaje denominado back-propagation (retro-propagación), el cual se emplea en este modelo de red.

El perceptrón multicapa con aprendizaje de retro-propagación es el modelo neuronal mas empleado en las aplicaciones prácticas, se estima que el 70% de los desarrollos con redes neuronales hacen uso de alguna de sus variantes [28].

Para introducir el modelo de perceptrón multicapa debemos ver un poco lo que es el perceptrón simple, el cual se inspira en las primeras etapas de procesamiento de los sistemas sensoriales de los animales, en los cuales la información va atravesando sucesivas capas de neuronas, que realizan un procesamiento progresivo de más alto nivel. El perceptrón simple es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entrada y otra de salida figura 3.5.

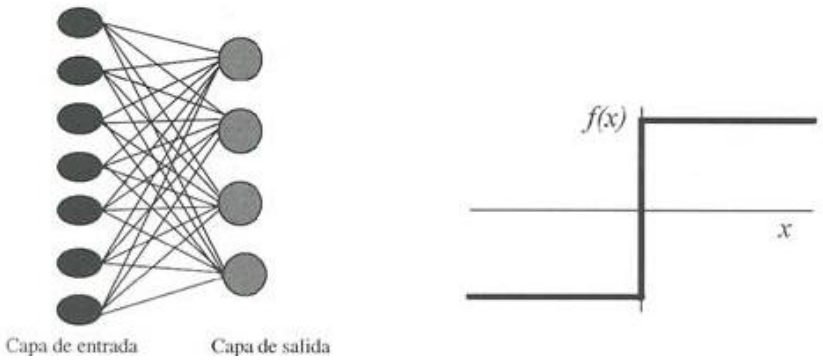


Figura 3.5 Perceptrón simple

Las neuronas de entrada no realizan ningún cálculo, únicamente envían información a las neuronas de salida. La función de activación de las neuronas de la capa de salida es de tipo escalón. Este modelo de perceptrón se puede aplicar como clasificador o como para la representación de funciones. Cada neurona del perceptrón representa una determinada clase, de modo que dado un vector de entrada, una cierta neurona responde 0 o 1 si pertenece a la clase que se representa y con lo opuesto sin es que pertenece a la clase. Una neurona de este tipo solo permite distinguir entre dos clases linealmente separables. Por lo tanto que las clases de funciones no separables linealmente no pueden ser representadas por un perceptrón simple.

La importancia del perceptrón radia en su carácter de dispositivo entrenable, puesto que el algoritmo de aprendizaje permite que el perceptrón determine automáticamente los pesos sinápticos que clasificad un determinado conjunto de patrones etiquetados.

El algoritmo de aprendizaje del perceptrón es de los denominados por corrección de pesos. Los algoritmos de este tipo ajustan los pesos en proporción a la diferencia existente entre la salida de la red y la salida deseada, con el objetivo de minimizar el error actual de la red.

Ahora si al perceptrón simple añadimos capas intermedias, es decir ocultas, obtendremos un perceptrón multicapas o MLP (Multi-layer Percptron). Esta arquitectura suele entrenarse

mediante el algoritmo de retropropagación (back-propagation) de errores o bien haciendo uso de alguna de sus variantes o derivados.

La estructura del MLP se presenta en la figura 3.6, en donde podemos observa la capa de la entrada de la red, las capa capas ocultas y la capa de salida, por otro lado cabe menciona que tenemos los pesos sinápticos de las capas ocultas con sus respectivo umbrales y los pesos sinápticos de la capa de salida con sus respectivos umbrales.

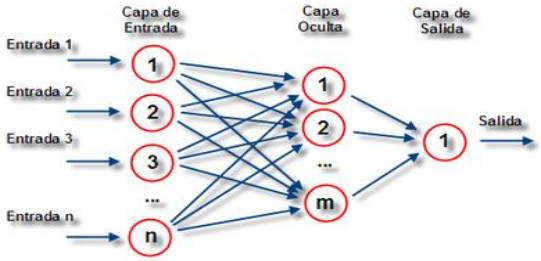


Figura 3.6 Perceptrón multicapa (MLP)

La arquitectura más común del MLP es con una capa de entrada, una capa oculta y una capa de salida, aunque existen numerosas variantes, como incluir neuronas no lineales en la capa de salida, introducir más capas ocultas, emplear diversas funciones de activación, limitar el número de conexiones entre neuronas y las de la capa siguiente, introducir dependencias temporales o arquitecturas recurrentes [29]

3.3. Algoritmo de la Red Neuronal MLP con aprendizaje de Retro-propagación

Una solución al problema de entrenar la arquitectura multicapa la proporciona el algoritmo de retropropagación de errores (backpropagation) [30]. Como ya se ha visto la red neuronal MLP con aprendizaje de retro-propagación es utilizada ampliamente por su alta capacidad de procesamiento aunado a la capacidad de aprender patrones. En su arquitectura más utilizada el MLP emplea una CAPA DE ENTRADA, una CAPA OCULTA y una CAPA DE SALIDA en su construcción más básica (figura 3.6).

Esta red neuronal artificial tiene varias entradas y una salida como se muestra en la figura. Los datos de entrada pasan al nodo central, pero solo podrán ser procesados de forma correcta aquellos valores de entradas que ya se halla aprendido la red, para eso antes se requiere un proceso al cual llamaremos aprendizaje o entrenamiento de la red, el cual consiste en pasar los datos de entrada como patrones fijos de aprendizaje, varias veces según requiera la red para aprenderse los valores de entrada. Para nuestro trabajo vamos a trabajar con una Red Neuronal tipo Multicapa (MLP) con entrenamiento de Retro-propagación.

El algoritmo de entrenamiento de retro-propagación se utiliza en capas para alimentar hacia adelante redes neuronales. Esto significa que neuronas artificiales están organizadas en capas, las cuales envían su señal hacia adelante, y posteriormente el error es propagado hacia atrás. La red recibe las entradas por una capa de entrada y la salida de la red está dada por la capa de salida. En la parte intermedia puede haber tantas capas ocultas como se necesiten. El algoritmo de retro-propagación es del tipo de aprendizaje supervisado, lo cual significa que nosotros proveemos al algoritmo con la entrada y la salida que deseemos que

calcule, para que posteriormente este calcule el error. La idea es reducir este error hasta que la red neuronal artificial se aprenda todos los datos de entrada y salida.

Por su forma de construcción la red de retro-propagación es capaz de resolver problemas que no son linealmente separables, lo cual la hace una herramienta muy útil. Sin embargo par que presente su mejor rendimiento esta red necesita ser entrenada de buena manera ya que de lo contrario esta no podrá extrapolar de forma correcta lo que nos significara que las salidas puedan ser imprecisas. Por otro lado es importante la existencia de mínimos locales en la función de error dificulta considerablemente el entrenamiento pues una vez alcanzando un mínimo el entrenamiento se puede estabilizar sin que hayamos alcanzando la tasa de convergencia fija.

Sea un MLP de tres capas, cuya arquitectura se presenta en la figura 3.7, con las entradas, las salidas, pesos y umbrales de las neuronas definidas a continuación:

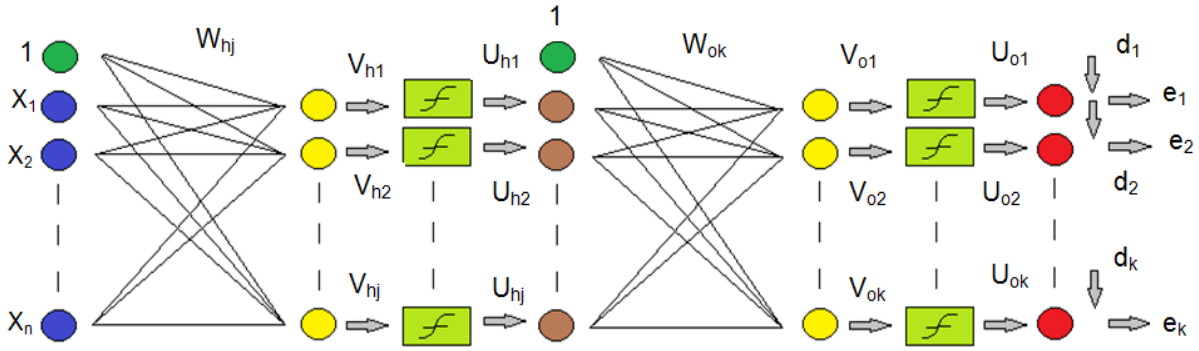


Figura 3.7 Esquema de la red de Retro-propagación.

- X_1, X_2, X_n son las entradas de la red.
- W_{hj}, matriz de pesos de la capa de entrada.
- W_{ok}, matriz de pesos de la capa de salida.
- U_{h1}, U_{h2}, U_{hj}, U_{o1}, U_{o2}, U_{ok}, salidas de las función de transferencia.
- d_1, d_2, d_k, valores deseados a la salida.
- e_1, e_2, e_k, error de la salida.

Como ya se ha explicado esta red como cualquier otra necesita dos etapas principalmente que son la de entrenamiento, la cual nos va a permitir que la red se aprenda un patrón y la de validación o la de prueba en la cual la red nos mostrara que tanto se aprendió o que tanto nos reconoce de lo que se aprendió.

Para la etapa de aprendizaje o entrenamiento tenemos dos fases; en la primera fase se presenta un patrón a la red el cual se utiliza durante todo el proceso de aprendizaje hasta que este llegue a la capa de salida. En la segunda fase ya que hemos calculado el error para ese patrón, dicho error lo debemos propagar o retroalimentar hacia atrás para que este sea utilizado para modificar los pesos de conexión para las n-capar intermedias, así como la capa de salida y entrada.

Las ecuaciones que se utilizan para el proceso de entrenamiento serán considerando que la red neuronal que utilizaremos será de la forma de la figura 3.7, en la cual cómo podemos

observar tiene una capa de entrada, una capa oculta y una capa de salida. Como se puede observar en la figura 3.7 tanto en la capa de entrada como en la capa oculta se agrega un "1" así como sus pesos, esto se hace para hacer una corrección de umbrales, y este se debe agregar si se tienen más capas.

Como se ha mencionado para esta red neuronal primero se propaga hacia adelante, para este paso tenemos que se presenta un vector de entrada $\overrightarrow{X_i}$ el cual se propaga multiplicándose por la matriz de pesos W_{hji} de donde se obtiene la sumatoria de sus elemento para obtener el vector $\overrightarrow{V_{hj}}$ el cual deberá pasar por la función de transferencia como se observa en las ecuaciones 3.2 y 3.3 respectivamente.

$$V_{hj} = \sum_{i=1}^{N_1} W_{hji} X_i + \theta_{hj} \tag{3.2}$$

$$U_{hj} = f(V_{hj}) \tag{3.3}$$

Ahora que se ha obtenido el vector de salida $\overrightarrow{U_{hj}}$ de la función de transferencia se procede casi de la misma manera que de la capa de entrada, ahora esta se multiplica por la matriz de pesos W_{okj} donde obtendremos la sumatoria de sus elementos para obtener el vector $\overrightarrow{V_{ok}}$, cual tendremos que pasa por la función de transferencia para obtener $\overrightarrow{U_{ok}}$, véase ecuaciones 3.4 y 3.5 respectivamente.

$$V_{ok} = \sum_{j=1}^{N_h} W_{okj} U_{hj} + \theta_{ok} \tag{3.4}$$

$$U_{ok} = f(V_{ok}) \tag{3.5}$$

Finalmente llegamos a la capa de salida en donde tendremos que calcular el error, para el cual contamos con un vector $\overrightarrow{d_k}$, el cual representa los valores deseados a los que pretendemos llegar y el vector $\overrightarrow{U_{ok}}$, el cual representa el valor que hemos calculado, con estos dos calculamos el error por cada patrón de entrada de la forma en que se muestra en la ecuación 3.6 y 3.7, el cual es un error cuadrático medio para hacer este error mucho más pequeño.

$$E_p = \frac{1}{2} \sum_{K=1}^{N_o} e_k{}^2 \tag{3.6}$$

$$E_p = \frac{1}{2} \sum_{K=1}^{N_o} (d_k - U_{ok})^2 \tag{3.7}$$

De aquí partimos para propagar este error hacia atrás de tal forma que modifiquemos las matrices de pesos primero la de la capa de salida, es decir la matriz W_{okj}, la cual la calculamos de cómo sigue aplicando las ecuaciones 3.8 y 3.9, las cuales representan con valores ya conocidos las derivadas de cada elemento desde la salida hasta los pesos de la matriz. También se aplica un coeficiente μ el cual representa el coeficiente de aprendizaje, es decir cuánto se va a modificar de la matriz.

$$W_{okj}(n) = W_{okj}(n-1) - \mu \frac{\partial E_p}{\partial W_{okj}} \tag{3.8}$$

$$\frac{\partial E_p}{\partial W_{okj}} = -(d_k - U_{ok})(1 - U_{ok})U_{ok} * U_{hj} \quad (3.9)$$

Hasta aquí solo se ha modificado la matriz de pesos de la capa de salida, es decir la matriz W_{okj}, pasaremos a modificar la matriz de la capa de entrada W_{hji} en la cual tendremos la influencia de las derivadas desde la salida hasta esta, las cuales expresamos en valores conocidos al igual que en los cálculos anteriores, para ello aplicamos las ecuaciones 3.10 y 3.11, y al de la misma manera que en la ecuación anterior se presenta α, el cual representa el coeficiente de aprendizaje, para modificar los pesos de la matriz de entrada W_{hji}.

$$W_{hji}(n) = W_{hji}(n-1) - \alpha\frac{\partial E_p}{\partial W_{hji}} \quad (3.10)$$

$$\frac{\partial E_p}{\partial W_{hji}} = -(1 + U_{hj})(1 - U_{hj})X_i * \sum_{k=1}^{N_o}(d_k - U_{ok})(1 - U_{ok})U_{ok} * W_{okj} \quad (3.11)$$

Con estas ecuaciones queda representado el algoritmo de la Red Neuronal de Retro-propagación, y estas se repiten cíclicamente es decir que una vez que se propaga el error entra un nuevo patrón a la red para ser procesado de tal manera que se propague un nuevo error y así sucesivamente hasta que terminen de pasar los $\overrightarrow{X_i}$ vectores de entrada, para que finalmente se complete una época. Para poder hacer este proceso de aprendizaje debemos tener en cuenta que entre mayor sean las épocas del entrenamiento mayor será el aprendizaje. Sin embargo falta obtener el error por cada época, la cual se calcula de igual manera que el error por patrón solo que como es un error medio cuadrático se tiene que sumas y dividirse entre el numero de patrones utilizados.

Como ya se ha mencionado es importante llegar a un error global no a un error local figura 3.8, ya que los errores locales no nos ayudan en el funcionamiento de la red y por ende en los resultados obtenidos con esta.

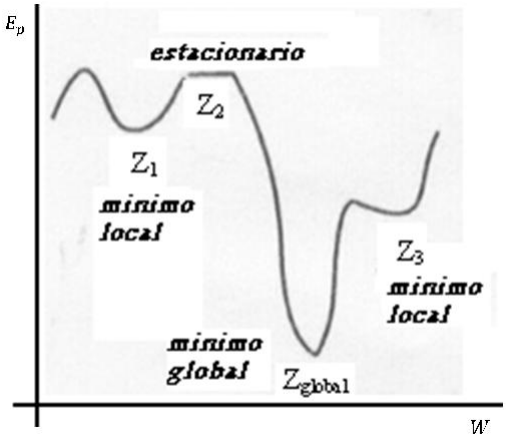


Figura 3.8 Grafica de los tipos de error.

Se debe comenzar siempre con pesos iniciales aleatorios ya que si se parte de pesos y umbrales iniciales nulos el aprendizaje no progresara. Existen dos maneras de presentar la actualización de los pesos estos son el aprendizaje por lotes y el aprendizaje en serie.

En el aprendizaje por lotes se lleva a cabo una fase de ejecución para todos y cada uno de los patrones del conjunto de entrenamiento, se calculó la variación en los pesos debida a cada patrón, se acumula y solamente entonces se procede a la actualización de los pesos, mientras que el aprendizaje en serie una variación común al algoritmo consiste en actualizar los pesos sinápticos tras la presentación de cada patrón.

En el aprendizaje en serie de debe tener en cuenta que el orden en la que se presentan los patrones debe ser aleatoria, puesto que si siempre se sigue un mismo orden el entrenamiento estaría viciado a favor del último patrón del conjunto de entrenamiento, cuyas actualizaciones, por ser la última, siempre predominara sobre las anteriores. Además esta aleatoriedad presenta una importante ventaja, puesto que puede permitir escapar de mínimos locales en determinadas ocasiones, por lo que al final del proceso puede alcanzarse un mínimo más profundo [31]

3.4. Matlab como herramienta en prototipos de redes neuronales

En el mercado existe una extensa variedad de sofisticadas herramientas de computación para resolver problemas matemáticos tales como Maple, Mathematica, MathCad, Matllab, etc. Sin embargo cada una de ellas tiene sus fortalezas y sus debilidades. Matlab en especifico tiene su principal fortaleza en cálculos que involucran matrices, de ahí su nombre debido a que es una abreviatura de Matrix Laboratory (Laboratorio de Matrices), la realización de cálculos matemáticos en el ambiente de Matlab sustituye la programación tradicional como C++ o Fortran, lo cual significa que el nivel de programación no necesita ser tan alto, con lo cual Matlab se convierte no solo en un programas más sino pasa a ser una muy útil herramienta estándar para ingenieros y científicos [32].

Los programas de alto nivel y Matlab destacan en el área de procesamiento de números es decir en programas que requieren cálculos repetitivos o el procesamiento de grandes cantidades de datos, pero usualmente se ejecutaran más rápido en C++ o Fortran, la única excepción a esto son los cálculos donde se involucran matrices, puesto que Matlab es optimo para matrices, por lo cual este tipo de necesidades se ejecutaran más rápido en Matlab que en cualquier otro programa de alto nivel.

Dentro del ambiente de MATLAB contamos con una caja de herramientas para poder estructurar la red con entrenamiento de retro-propagación que deseemos de acuerdo a lo que estemos buscando, es decir, MATLAB en esta opción nos proporciona una amplia gama de diferentes estructuras, como ya se ha mencionado el entrenamiento de retro-propagación ha sufrido diferentes variaciones con el objetivo de mejorar sus rendimientos, es importante recordar que la parte importante de una red es la parte de entrenamiento por lo que ya se ha mencionado anteriormente, y por ende la parte que se modifica esencialmente es esta, él como la red se aprende o hace la retro-propagación, dichas modificaciones se mencionan a continuación:

1. ***Trainlm:*** es una función de entrenamiento de redes la cual actualiza sus matrices de pesos y los valores del bias de acuerdo a la optimización de Levenberg-Marquardt., este algoritmo de retro-propagación es con frecuencia la más rápida en la caja de herramientas y es altamente recomendada como una primera opción de algoritmos supervisados, pero requiere de más memoria que cualquier otro algoritmo.

Este arreglo de retro-propagación es utilizado para calcular el Jacobiano jX de rendimiento de perforación con respecto a los pesos y la variable del bias X. Cada variable es ajustada de acuerdo a Levenberg-Marquardt como sigue:

jj = jX * jX
je = jX * E
do = -(jj+I*mu) \ je

Donde "**E**" son todos los errores e "**I**" es la matriz identidad. El valor adaptación "**mu**" es incrementado por "**mu_inc**" hasta el cambio por encima de los resultados en una reducción del valor de rendimiento. El cambio es posteriormente hecho por la red y "mu" disminuido por "**mu_dec**".

2. ***Traingd:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al gradiente descendiente, este algoritmo puede entrenar cualquier red tan larga como sus matrices de pesos, sus entradas y las funciones de transferencia tiene funciones derivativas. La retro-propagación es usada para calcular derivadas del rendimiento de perforación con respecto a las matrices de los pesos y las variables del bias X. cada variable es ajustada de acuerdo al gradiente descendiente como sigue:

   dX = lr * dperf/dX

3. ***Traingdm:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al gradiente descendiente con momento, es muy parecida a la anterior simplemente se le agrega un momento el cual altera el gradiente de cambio o mejor dicho el cambio previo de las matrices de pesos y el valor del bias X. cada variable es ajustada de acuerdo al gradiente descendiente con momento como sigue:

   do = mc*dXprev + or*(1-mc)*doper/do

   Donde "**dXprev**" es el cambio previo a los pesos o el bias.

4. ***Traingda:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al gradiente descendiente con tasa de aprendizaje adaptativa, la retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X. cada variable se ajusta de la siguiente manera:

   do = or*doper/do

   En cada época, si el rendimiento disminuye hacia el objetivo después la tasa de aprendizaje se aumenta por un factor "**lr_inc**", pero si el rendimiento aumenta más que el factor "**mas_perf_inc**", la tasa de aprendizaje es ajustado por un factor "**lr_dec**" y el cambio que aumenta el rendimiento no se hace.

5. ***Traingdx:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al gradiente descendiente con momento y con tasa de aprendizaje adaptativa, este tipo es una combinación de los dos últimas modificaciones, sin embargo esto no nos garantiza que esta funcione mejor o peor que las anteriores, depende de nuestra necesidad.

   La retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X. cada variable se ajusta de la siguiente manera:

   do = mc*dXprev + or*mc*doper/do

   Donde "**dXprev**" es el cambio previo a los pesos o al bias. En cada época, si el rendimiento disminuye hacia el objetivo después la tasa de aprendizaje se aumenta por un factor "**lr_inc**", pero si el rendimiento aumenta más que el factor "**mas_perf_inc**", la tasa de aprendizaje es ajustado por un factor "**lr_dec**" y el cambio que aumenta el rendimiento no se hace.

6. ***Trainrp:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al algoritmo de retro-propagación flexible (Rprop). La retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X. cada variable se ajusta de la siguiente manera:

   dX = deltaX.*sign(gX);

   Donde los elementos de "**deltaX**" son todos inicializados a "**delta0**" y "**gX**" es el gradiente. En cada iteración los elementos de "deltaX" son modificados. Si un elemento de "**gX**" cambia de signo de una iteración a otra, entonces le corresponde un elemento de "**deltaX**" el cual es disminuido por "**delta_dec**", ahora si un elemento "**gX**" mantiene el mismo signo de una iteración a otra entonces le corresponde un elemento "**deltaX**" el cual es de disminuido por un delta "**delta_inc**". De acuerdo a Riedmiller

7. ***Traincgf:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo a un gradiente conjugado de retro-propagación con actualizaciones Fletcher-Reeves, este arreglo ajusta cada variables como se muestra:

   X = X + a*dX;

   Donde "**dX**" es la dirección buscada. El parámetro "**a**" es escogida para minimizar el rendimiento a lo largo de la dirección buscada. La función de búsqueda lineal "**seachFcn**" es utilizada para localizar el punto mínimo. La primera dirección de búsqueda es la negativa del gradiente de rendimiento. En las siguientes iteraciones

la dirección de búsqueda es calculada del nuevo gradiente y las anteriores direcciones de búsqueda, de acuerdo a lo siguiente:

do = -gX + dX_old*Z;

Donde "gX" es el gradiente. El parámetro "Z" puede seo calculado en diferentes formas individuales. Por la variación the Fletcher-Reeves de gradiente conjugado, el cual es calculado de acuerdo a lo siguiente:

Z=normnew_sqr/norm_sqr;

Donde "**norm_sqr**" es la norma cuadrada de previos gradientes y "**normnew_sqr**" es la norma cuadrada del gradiente actual

8. ***Traincgp:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo a un gradiente conjugado de retro-propagación con actualizaciones Polak-Ribiére, este arreglo ajusta cada variables como se muestra

   X = X + a*dX;

   Donde "**dX**" es la dirección buscada. El parámetro "**a**" es escogida para minimizar el rendimiento a lo largo de la dirección buscada. La función de búsqueda lineal "**seachFcn**" es utilizada para localizar el punto mínimo. La primera dirección de búsqueda es la negativa del gradiente de rendimiento. En las siguientes iteraciones la dirección de búsqueda es calculada del nuevo gradiente y las anteriores direcciones de búsqueda, de acuerdo a lo siguiente:

   do = -gX + dX_old*Z;

   Donde "gX" es el gradiente. El parámetro "Z" puede seo calculado en diferentes formas individuales. Por la variación the Polak-Ribiére de gradiente conjugado, el cual es calculado de acuerdo a lo siguiente:

   Z = ((gX - gX_old)'*gX)/norm_sqr

   Donde "**norm_sqr**" es la norma cuadrada de previos gradientes y "**gX_old**" es el gradiente de la iteración previa.

9. ***Traincgb:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo a un gradiente conjugado de retro-propagación con reinicio Powell-Beale, este arreglo ajusta cada variables como se muestra

   X = X + a*dX;

Donde "**dX**" es la dirección buscada. El parámetro "**a**" es escogida para minimizar el rendimiento a lo largo de la dirección buscada. La función de búsqueda lineal "**seachFcn**" es utilizada para localizar el punto mínimo. La primera dirección de búsqueda es la negativa del gradiente de rendimiento. En las siguientes iteraciones la dirección de búsqueda es calculada del nuevo gradiente y las anteriores direcciones de búsqueda, de acuerdo a lo siguiente:

do = -gX + dX_old*Z;

Donde "gX" es el gradiente. El parámetro "Z" puede ser calculado en diferentes formas individuales. La variación the Powell-Beale de gradiente conjugado es distinguido por dos características. La primera, el algoritmo utiliza una prueba para determinar cuándo restablecer la dirección de búsqueda negativa del gradiente. Segundo la dirección de búsqueda es calculada del gradiente negativo, las dirección de búsqueda previas y la ultima dirección de búsqueda antes del reinicio previo.

10. ***Trainscq:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al método de escala del gradiente conjugado. La retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X.

    El algoritmo de escala de gradiente conjugado es basado en la dirección conjugada, como en traincgp, traincgf, y traincgb, pero este algoritmo no realiza una búsqueda lineal en cada iteración, según Moller.

11. ***Trainbfq:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al método BFGS casi Newton. La retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X. cada variable se ajusta de la siguiente manera:

    X = X + a*dX;

    Donde "**dX**" es la dirección buscada. El parámetro "**a**" es escogida para minimizar el rendimiento a lo largo de la dirección buscada. La función de búsqueda lineal "**seachFcn**" es utilizada para localizar el punto mínimo. La primera dirección de búsqueda es la negativa del gradiente de rendimiento. En las siguientes iteraciones la dirección de búsqueda es calculada de acuerdo a la siguiente fórmula:

    dX = -H\gX;

    Donde "**gX**" es el gradiente y "**H**" es una aproximación a la matriz Hesiana.

12. ***Trainoss:*** es una función de entrenamiento de redes la cual actualiza los valores de sus matrices de pesos y el bias de acuerdo al método de la secante de un solo paso. La retro-propagación es utilizada para calcular derivativas de rendimiento de perforación con respecto a los pesos y las variables del bias X.

X = X + a*dX;

Donde "**dX**" es la dirección buscada. El parámetro "**a**" es escogida para minimizar el rendimiento a lo largo de la dirección buscada. La función de búsqueda lineal "**seachFcn**" es utilizada para localizar el punto mínimo. La primera dirección de búsqueda es la negativa del gradiente de rendimiento. En las siguientes iteraciones la dirección de búsqueda es calculada de acuerdo a la siguiente fórmula:

do = -gX + Ac*X_step + Bc*dgX;

Donde "**gX**" es el gradiente, "**X_step**" es el cambio de los pesos en la iteración previa y "**dgX**" es el cambio en el gradiente de la última iteración, según Battiti

13. ***Trainbr:*** es una función de entrenamiento de redes la cual actualiza sus matrices de pesos y los valores del bias de acuerdo a la optimización de Levenberg-Marquardt. Este minimiza una combinación de errores cuadráticos y pesos, posteriormente determina la correcta combinación entonces como produce una red que generaliza bien, el proceso es llamado regularización Bayesiana.

    La regularización Bayesiana minimiza una combinación lineal de errores cuadráticos y pesos. Esto también modifica la combinación lineal de modo que al final del entrenamiento la red resultante tiene cualidades de generalización buena. Esta regularización Bayesiana tiene lugar con el algoritmo de Levenberg-Marquardt. Este arreglo de retro-propagación es utilizado para calcular el Jacobiano jX de rendimiento de perforación con respecto a los pesos y la variable del bias X. Cada variable es ajustada de acuerdo a Levenberg-Marquardt como sigue:

    jj = jX * jX
    je = jX * E
    dX = -(jj+I*mu) \ je

    Donde "**E**" son todos los errores e "**I**" es la matriz identidad. El valor adaptación "**mu**" es incrementado por "**mu_inc**" hasta que el cambio se muestre por encima de los resultados en una reducción del valor de rendimiento. El cambio es posteriormente hecho por la red y "**mu**" es disminuido por "**mu_dec**".

Estas estructuras de redes las podemos encontrar en el toolbox de Matlab. Como ya se ha mencionado uno de los principales campos de aplicación de las redes es el de reconocimiento de patrones, siendo este un campo de crecimiento en tiempos recientes, este tema se explica mejor en la siguiente sección.

3.5. Reconocimiento de patrones

El termino reconocimiento de patrones abarca una gran área de procesamiento de información, desde reconocimiento de voz o de escritura cursiva, hasta detección de fallas

en maquinaria o diagnóstico de enfermedades, que nos puede ser difícil comprender que tan compleja es esta tarea para las computadoras.

El proceso de reconocer cosas, como un sonido, comida, un depredador, una imagen, un amigo, un sabor, entre otras cosas, es algo que lo hacemos de forma inconsciente, pero que nos permite adaptarnos a nuestro entorno y puede ser clave en el momento de la supervivencia [33].

Esta habilidad natural que tiene prácticamente todo ser vivo nos lleva a varias preguntas que nos pueden llevar al terreno filosófico: ¿cómo identificamos cosas?, ¿qué proceso hace que nuestra mente resuelva fácilmente este tipo de problemas?, ¿cómo podemos abstraer conceptos?, o más globalmente, ¿cómo y por qué funciona nuestro cerebro?

Una de las principales cosas que observamos al tratar de resolver este tipo de problemas en una computadora, es que la solución general es muy complicada, dado que son muchos los factores que están involucrados en el proceso de reconocer. La descripción de cada una de las características (que llamaremos rasgos) que definen un objeto o situación, a su vez implica una descripción y que algunas pueden medirse para obtener un valor numérico (como las dimensiones) y otras no (como la belleza).

Por otro lado si tenemos presente algún otro objeto o situación que se parezca a la primera, ahora pensemos en las diferencias entre ambos casos. A continuación si tratamos de imaginar cómo resolver el problema de diferenciar (clasificar) entre estas dos, usando una computadora, seguramente nos percataremos de que hay rasgos que nos ayudan a diferenciar mejor a uno de otro. De estos procesos, que consisten en la extracción y selección de rasgos, depende en gran medida si podemos resolver o no el problema en la computadora.

Una vez que sabemos qué rasgos utilizar, necesitamos encontrar un método que pueda simularse en la computadora que de alguna forma compare el conjunto de rasgos (que llamaremos patrón) de un objeto a otro y es aquí donde nos topamos con otro problema: ¿qué método ocupar?, y ¿cómo le hacemos para, en una computadora, comparar cosas que no se pueden medir?

El problema se hace más complejo cuando nos damos cuenta de que el proceso que se utilicemos, tal vez ya no funcione si ahora también queremos diferenciar a un tercer objeto o situación de los primeros, y así podríamos enumerar un sinfín de problemas relacionados con objetos, seres vivos, conceptos e ideas abstractas, que están presentes en el entorno en que se desenvuelven los seres humanos.

La rama interdisciplinaria de las ciencias de la computación que se encarga de resolver este tipo de problemas se denomina Reconocimiento de Patrones, el cual incluye entre sus principales tareas a la clasificación y a la recuperación de patrones [34].

El diseño de un sistema de reconocimiento de patrones usualmente implica la repetición de un número de diferentes actividades: colección de datos, elección de las características, elección del modelo, entrenamiento y evaluación.

La colección de datos puede contar sorprendentemente gran parte del costo del desarrollo de un sistema de reconocimiento de patrones. Puede ser posible realizar un estudio viable preliminar con un pequeño conjunto de ejemplos típicos, pero mucho más datos usualmente serán necesitados para asumir un buen rendimiento en el sistema de envió.

La elección de distinguir las características es un paso crítico del diseño y depende de las características del dominio del problema. Tener acceso a ejemplos de datos, sin duda será valioso para elegir una función establecida. Sin embargo conocimientos previos también jugara un papel importante. Incorporar el conocimiento previo puede ser más sutil y difícil. En algunas aplicaciones el conocimiento en última instancia deriva de la información sobre la producción de los patrones. En otras aplicaciones el conocimiento puede ser sobre la forma de una categoría subyacente, o atributos específicos de los patrones, como son el hecho de que una cara tenga dos ojos, una nariz, etc.

En la selección o diseño de las características, obviamente nos gustaría encontrar que las características son simples de extraer, invariantes a la transformación irrelevante, insensibles al ruido y útiles para discriminación de patrones en diferentes categorías. En general, el proceso de uso de datos para determinar el clasificador está referido al entrenamiento del clasificador. La evaluación es importante tanto para medir el rendimiento del sistema y para identificar las necesidades para mejorarlo en sus componentes.

Un sistema complejo excesivamente puede permitir una clasificación perfecta de la muestra de entrenamiento, será poco probable un buen rendimiento en nuevos patrones. Esta situación es conocida como overfitting.

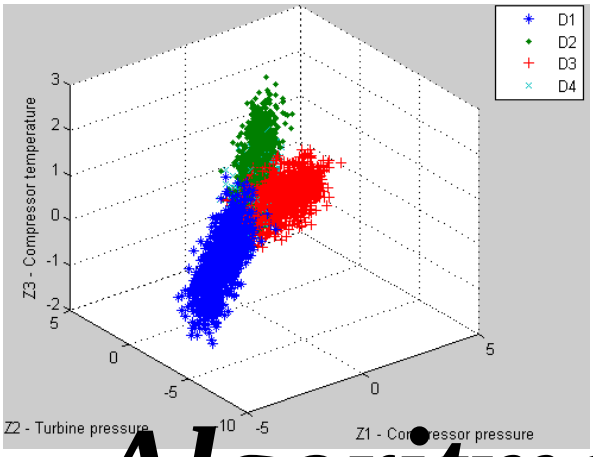
Los principales enfoques de Reconocimiento de Patrones son los siguientes:

1. Enfoque estadístico-probabilístico. Su principal clasificador está basado en la teoría de la probabilidad, y específicamente en el teorema de Bayes. Este método difiere del método geométrico en que las clases no están predefinidas, si no que se definen en términos de cualquier forma regular en la representación del espacio. Es históricamente el primer enfoque que existió y probablemente el más desarrollado.
2. Clasificadores basados en métricas. Usualmente se ubican dentro del enfoque estadístico y se basan en el concepto de métrica y espacios métricos para hacer la clasificación. Las clases están representadas por regiones en el espacio de representación.
3. Enfoque sintáctico-estructural. Se basa en la teoría de autómatas y lenguajes formales para hacer la clasificación. Se enfoca más en la estructura de las cosas a clasificar que en mediciones numéricas. En sintáctico o enfoque estructural, un patrón complejo se describe en términos de patrones de y su relación.
4. Enfoque lógico-combinatorio. Creado en la antigua Unión Soviética, tiene como principal característica el poder trabajar con variables de todo tipo, aunque sus

algoritmos suelen ser de complejidad alta. Los objetos son comparados directamente con algunos ejemplos almacenados o prototipos que son representativas de las clases subyacentes.

5. Enfoque neuronal. Se basa en modelos matemáticos de las neuronas biológicas, o dicho de otra forma, trata de emular la forma en cómo interactúan nuestras neuronas. Estas redes intento de aplicar los modelos de neuronales biológica sistemas para resolver problemas prácticos de reconocimiento de patrones. Este enfoque se ha vuelto tan popular que el uso de redes neuronales para la resolución de reconocimiento de patrón de problemas ya es común.
6. Enfoque Asociativo. Creado en el Centro de Investigación en Computación del IPN en 2002, utiliza los modelos de memorias asociativas para crear clasificadores robustos.

Es interesante comparar estos enfoques para el reconocimiento automático de modelos con las diversas formas el proceso de aprendizaje humano puede ser modelado: Simulación del sistema nervioso en sí, o simulación de los procesos en ese sistema, ya sea basados en información directa de los sentidos (como en la estadística y los planteamientos geométricos) o en un mayor nivel de información simbólica (como en el estructural-AP aproach). El procedimiento que se pongan en esta plantilla puede ser comparado con el aprendizaje mediante el almacenamiento de todos los hechos sin entenderlas.



Capítulo 4.- Algoritmos realizados en Matlab

Como ya se ha mencionado en capítulos anteriores la técnica del análisis del conducto de flujo se divide en tres etapas que son: monitoreo o detección de problemas, diagnóstico detallado y pronostico, además de incluir una etapa preliminar de validación de datos (cálculo de desviaciones). Así mismo se ha mencionado que esta tesis aplica el algoritmo de la Red neuronal de retro-propagación en estas etapas, principalmente trabajamos en dos de las etapas más importantes dentro del análisis, la etapa preliminar y la etapa del diagnóstico detallado.

Para realizar esta tesis, contamos con el programa de la respectiva etapa de monitoreo. Estos programas aplican como base funciones estándares de Matlab, para identificarlos, llamaremos algoritmo 1 al programa de la etapa preliminar, validación de datos (aproximación de función) y algoritmo 2 al diagnóstico detallado (clasificación de las fallas). Ambas aplicaciones se realizan para un motor de turbina de gas de dos ejes y de turbina libre de potencia

4.1. Programación de las redes y su entrenamiento

En el presente trabajo se programó en Matlab el algoritmo de diagnóstico con la red Neuronal de Retro-propagación, Se tiene como prototipo el algoritmo de diagnóstico basado en las funciones estándares de “Neural Networks Kit”. Para hacer un algoritmo no

dependiente de este Kit, en nuestro algoritmo las redes neuronales y su entrenamiento se programan “a mano”, sin funciones del Kit, Para confirmar que el algoritmo nuevo no tiene errores, este algoritmo se compara con el algoritmo 1 y 2 respectivamente (ver anexos).

4.1.1. Algoritmo 1: cálculo de desviaciones (versión 1.1)

4.1.1.1. Descripción

El algoritmo 1 se emplea para aproximar una función, lo que en nuestro lenguaje del diagnóstico paramétrico conocemos como cálculo de las desviaciones. Dicho algoritmo 1 está programado con funciones estándares de Matlab, por lo cual nosotros modificaremos esta parte, programaremos el algoritmo de la red de retro-propagación de acuerdo a las ecuaciones 3.2 a 3.11 explicadas en el capitulo anterior, de esta forma tendremos la versión 1.1 el algoritmo 1. Por tal razón la parte modificada al algoritmo 1 se observa en la figura 4.1.

Antes de entrara a la red es importante señalar que el algoritmo, por su naturaleza y por su aplicación, requiere que se normalicen los datos tanto de entrada como la salida deseada, se normalizan los datos de la muestra de entrenamiento así como también los datos de la muestra de validación, como se puede observar en la figura 4.2.

```
192      %  Network structure, training, and simulation
193 -    MiMa1=minmax(P1);
194 -    nn=12; nm=13; tgoal=0.000015;
195 -    if nm==1              % Training function
196 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
197 -       nepochs=1500;    nshow=50;
198 -       net.trainParam.mem_reduc=1;
199 -    elseif  nm==2
200 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
201 -       nepochs=50000;    nshow=100;
202 -       net.trainParam.lr=0.5;
203 -    elseif  nm==3
204 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
205 -       nepochs=50000;    nshow=100;
206 -       net.trainParam.lr=0.1;
207 -       net.trainParam.mc=0.5;
208 -    elseif  nm==4
209 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
210 -       nepochs=50000;    nshow=100;
211 -       net.trainParam.lr=0.1;
212 -       net.trainParam.lr_inc=1.10;
213 -    elseif  nm==5
214 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
215 -       nepochs=50000;    nshow=20;
216 -       net.trainParam.mc=0.5;
217 -       net.trainParam.lr=0.1;
218 -    elseif  nm==6
219 -       net1 =newff(MiMa1,[nn,ky],{'tansig','tansig'},'tr
220 -       nepochs=100;    nshow=20;
221 -    elseif  nm==7
222 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'tra
223 -       nepochs=5000;    nshow=100;
```

```
224 -    elseif  nm==8
225 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
226 -       nepochs=50000;    nshow=100;
227 -       net1.trainParam.delta=0.0001;
228 -    elseif  nm==9
229 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
230 -       nepochs=102400;    nshow=100;
231 -       net1.trainParam.delta=0.0001;
232 -    elseif  nm==10
233 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
234 -       nepochs=50000;    nshow=100;
235 -    elseif  nm==11
236 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
237 -       nepochs=102400;    nshow=200;
238 -    elseif  nm==12
239 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
240 -       nepochs=102400;    nshow=400;
241 -       net1.trainParam.min_grad=  1e-7;
242 -    elseif  nm==13
243 -       net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trai
244 -       nepochs=2000;    nshow=50;
245 -    end;
246 -    net1.trainParam.show=nshow;
247 -    net1.trainParam.epochs=nepochs;
248 -    net1.trainParam.goal=tgoal;
249 -    [net1,tr]=train(net1,P1,T1);              %   Training
250 -    A1=sim(net1,P1);  A2=sim(net1,P2);        %   Simulati
251 -    DA1T=(A1-T1)'; sser=norm(DA1T,'fro');
252 -    sser=sser*sser;      % sum of squared errors
253 -    mser=sser/(krb*ky); % mean squared error
```

Figura 4.1 Modificación al algoritmo prototipo 1.

```
%                       NEURAL NETWORKS
%   Normalized network inputs
kuu=ku;
nu=[1 2 3 4 6];
if idegr == 1                                   %*
   kuu=ku+1;
end;
for iu=1:kuu
   P1(:,iu)=U(:,nu(iu));
   P2(:,iu)=US(:,nu(iu));
end;
P1=P1'; P2=P2'; MiMaU=minmax(P1); DU=zeros(kuu,1); DU(:)=(MiMaU(:,2)-MiMaU(:,1))/2;
beta=1.0;
for iu=1:kuu
   for ir=1:krb
       P1(iu,ir)=beta*(P1(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
   for ir=1:krs
       P2(iu,ir)=beta*(P2(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
end;
%   Normalized targets
alf=0.9;
YY=[Y;YS];
MnY=min(YY);    MxY=max(YY);    DDY=(MxY-MnY)/2.0;
for iy=1:ky
   for ir=1:krb
      T1(iy,ir)=alf*(Y(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
   end;
   for ir=1:krs
      T2(iy,ir)=alf*(YS(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
   end;
end;
```

Figura 4.2 Normalización del conjunto de muestras de entrenamiento y validación.

Una vez teniendo los datos de entrada y salida listos, determinamos a la red con sus parámetros, es decir, número de neuronas de la capa entrada, oculta y salida, así también determinamos el valor del coeficiente de aprendizaje alfa, el número de iteraciones o épocas. Por otro lado es importante mencionar que para esta aplicación en particular, tendremos que el número de patrones de la muestra de entrenamiento es de 2608 y para la muestra de validación es de 4091. Con estos parámetros determinados procedemos a llamar a nuestra función de la red, tal y como se puede observar en la figura 4.3.

```
%%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%      ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%%

% VALORES DE ENTRADA
N = 5;  %input('Numero de neuronas de entrada: ');
L = 12; %input('Numero de neuronas ocultas: ');
M = 7; %input('Numero de neuronas de salida: ');
alfa = 0.0025; %input('Valor de alfa: ');
min = 0.000015;    %input('Minimo error: ');
iter = 1000; %input('Numero de iteraciones: ');
patrones_ent = 2608;   %input('Numero de patrones de entrenamiento: ');
patrones_sim = 4091;   %input('Numero de patrones de simulacion: ');

patrones = patrones_ent;

[matriz_Wh_ji, matriz_Wo_kj, vector_J] = ent_bpj (N, L, M, alfa, min, iter, patrones, P1, T1);
```

Figura 4.3. Parámetros que determinan el tamaño de la red.

Al llamar a nuestra función de la red de retro-propagación, lo primero que hace es establecer los valores iniciales de las matrices de pesos de la capa oculta y la capa de salida con números aleatorios, como lo muestra la figura 4.4

```
% INICIALIZACION DE LA MATRIZ DE PESOS W Y V
Wh = 0.2*rand(N+1,L)-.1;%0.2
Wo = 0.2*rand(L+1,M)-.1;%0.2
Wh_ji = Wh';
Wo_kj = Wo';
```

Figura 4.4 Estableciendo valores iniciales de las matrices de pesos.

Así mismo a los vectores de entrada de la red agregamos un 1 (el bias), el cual es un factor de corrección en los umbrales, tal y como se observa en la figura 4.5.

```
% Agregando el Bias a la entrada
B1 = [P1];
B = B1';
PB=[];
for g=1:patrones
    PB(g,:)=[1,B(g,:)];
end
T =T1';
```

Figura 4.5 Agregando el bias a los datos de entrada.

Con estos valores ya establecidos se procede a entrar a la fase de entrenamiento de la red por lo que se ingresan el vector a la entrada de la red, posteriormente pasara a la capa oculta, y a la capa de salida hasta obtener un error. Este recorrido lo tenemos en otra función la cual llamamos sim_j. Al obtener el error empezaremos a retro-propagarlo, de este modo se estará entrenado la red tal y como se observa en la figura 4.6

```
for kp = 1:iter
    %fprintf('\n ---- EPOCH: %d ---- \n',epoch);
    cont=cont+1;
    error=0;
    for ip = 1:patrones
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        d_k = E;
        jes_1 = d_k.*((1.+Uo_k).*(1.-Uo_k));
        delta1 = (-1).*jes_1*Uh_j;
        Wo_kj = Wo_kj - alfa.*delta1; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE SALIDA
        a=jes_1'*Wo_kj; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA DE ENTRADA
        jes_2 = (-1).*a.*((1.+Uh_j).*(1.-Uh_j));
        jes_2 = jes_2(2:length(jes_2));
        delta2 = jes_2'*Xi';
        Wh_ji = Wh_ji -alfa.*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE ENTRADA
        error = error + sum((d_k.*d_k)/M); %%%%% CALCULO DE ERRORRES
        %e = d_k.*d_k;
        %Error_p = (sum (e));
        %Error_mc (ip) = (Error_p'/M);
    end
    error = error/patrones; %%%% ERROR POR EPOCA
    vec_err(1,kp) = error;
    %fprintf(1,'Error = %4.9f ==> %d\n',error,kp);
    J(1,kp)=error;
    if error<min
        break;
    end
    %error;
end
```

Figura 4.6 Entrenamiento de la red de retro-propagación.

En la etapa de simulación, tenemos que pasar los datos de entrenamiento nuevamente y después los datos de validación, recordando que el numero de patrones o datos es diferente, debido a que la muestra de entrenamiento solo tiene los primeros 2608 puntos de mediciones sin alguna influencia, mientras que la muestra de validación cuenta con 4091

periodos de medición incluidos los periodos de ensuciamiento y el lavado del compresor. Por lo cual proseguimos como se observa en la figura 4.7

```
% Agregando el Bias a la entrada
B1_1 = [P1];
B_1 = B1_1';
PB_1=[];
for g = 1:patrones_ent
    PB11_1(g,:) = [1,B_1(g,:)];
end
PB1_1 = PB11_1';
PB = PB1_1;
T = T1;

for ip = 1:patrones_ent
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_1 = Uo_k;
    A1 (:,ip)= Uo_k_1';
end
```

```
% Agregando el Bias a la entrada
B1_2 = [P2];
B_2 = B1_2';
PB_2=[];
for g = 1:patrones_sim
    PB11_2(g,:) = [1,B_2(g,:)];
end
PB1_2 = PB11_2';
PB = PB1_2;
T = ones (M,patrones_sim);

for ip = 1:patrones_sim
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_2 = Uo_k;
    A2 (:,ip)= Uo_k_2';
end
```

Figura 4.7 Etapa de simulación de los datos.

Por último se simulan datos reales para verificar el funcionamiento de la red y de esta manera también observamos el comportamiento de la red cuando involucramos datos reales, como se ilustra en la figura 4.8.

```
%    Networks behaviour verification by their y(u)-plots
krv=2000;  nuv=4;      % points quantity and number of the first u-argument
nuf=3;  kuf=7;         % number and points quantity of the second u-argument
nyv=3;                 % y-function number
du=2.0/krv; duf=2.0/(kuf-1); tv=1:krv;
for iuf=1:kuf
   for ir=1:krv
      for iu=1:kuu
         P1V(iu,ir)=0.0;
      end;
      P1V(nuf,ir)=-1.0+(iuf-1)*duf;
      P1V(nuv,ir)=-1.0+du*ir;
   end;
   %%%%%%%%% SIMULACION DE P1V
   % Agregando el Bias a la entrada
   B1_1V = [P1V];
   B_1V = B1_1V';
   PB_2=[];
   for g = 1:ir
       PB11_1V(g,:) = [1,B_1V(g,:)];
   end
   PB1_1V = PB11_1V';
   PB = PB1_1V;
   T_2 = T2;
   T = T_2;
   aux_2 = 0;

   for ip = 1:ir
       [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
       Uo_k_1V = Uo_k;
       A1V (:,ip)= Uo_k_1V';
   end
   %A1V=sim(net1,P1V);
   for iy=1:ky
      for ir=1:krv
         YNV(ir,iy,iuf)=A1V(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
      end;
   end;
end;
```

Figura 4.8 Simulación de datos reales.

4.1.1.2. Resultados

Para calcular esta función y las desviaciones (ecuación 2.10), se programó el algoritmo descrito en las ecuaciones 3.2 a 3.11. Para entrenar y verificar la red de retro-propagación,

se usan los datos del motor escogido y registrados en campo. Estos datos fueron medidos por el sistema estándar de mediciones del motor y registrados en el sistema de monitoreo cada hora de la operación del motor. Por lo tanto, la estructura de las variables de los vectores $\vec{Y}$ y $\vec{U}$ corresponde al sistema estándar de mediciones.

Como hemos mencionado antes, el algoritmo de entrenamiento realizado en la versión 1.1 del algoritmo 1 funciona en el modo de serie. Cada vector $\vec{U}$ de los datos de entrenamiento entra a la red; la salida correspondiente de la red se compara con la salida deseada (tarea) $\vec{U}$ que se encuentra en los mismos datos de entrenamiento. El error (la diferencia entre la tarea y la salida actual) se usa para corregir los pesos de la red. Este ciclo se repite sucesivamente para todas 2608 secciones de medición escogidas para el entrenamiento (primera parte de los datos registrados disponibles) formando un ciclo interior del entrenamiento en serie. Puesto que los coeficientes de aprendizaje son relativamente pequeños, nunca es suficiente un ciclo interior. Por lo tanto, este ciclo se repite varias épocas en un ciclo exterior resultando en un número grande de las iteraciones totales. Al fin, la red entrenada se aplica a los datos de validación. La calidad de esta red se estima visualmente por medio de las gráficas de las desviaciones calculadas con la red en los datos de entrenamiento y los de validación.

Para ajustar este algoritmo de entrenamiento la versión 1.1 del algoritmo 1, probamos diferentes valores de los coeficientes de aprendizaje y del número total de las épocas. Los mejores resultados de la versión 1.1 obtenidos son con 3000 épocas totales, se describen abajo en comparación con los resultados del algoritmo 1.

Las figuras 4.9 y 4.10 ilustran las desviaciones computadas con los datos de entrenamiento por ambos programas, es decir para el algoritmo 1 y la versión 1.1 de este. Estas desviaciones dTt corresponden a la temperatura de gases atrás la turbina de alta presión y son presentadas contra el tiempo de operación del motor. El deterioro del motor analizado no influye aquí a las desviaciones, porque aquí ellas se calculan usando no la función de referencia, sino el modelo de motor deteriorado. Por consiguiente, estas desviaciones pueden ser consideradas como errores y deben ser minimizadas. Comparando las figuras, llegamos a la conclusión que la versión 1.1, aunque no produce errores graves, pierde contra el algoritmo 1: el nivel de desviaciones (errores) es notablemente mayor para la versión 1.1. Ahora vamos a ver si esta diferencia en la etapa de entrenamiento afecta a los resultados de validación.

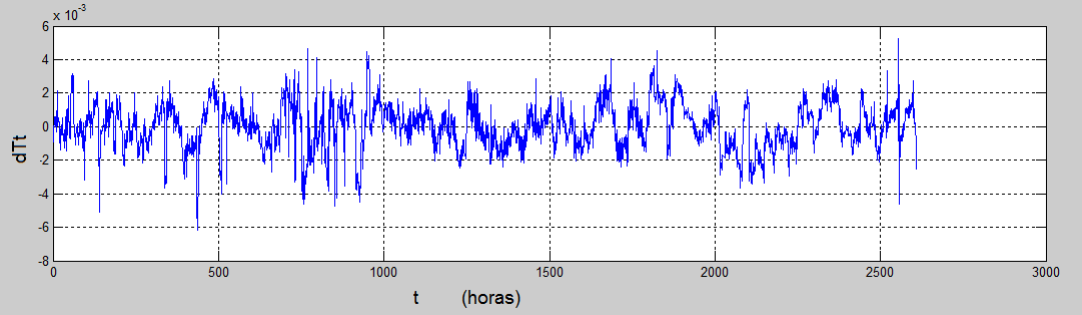


Figura 4.9 Desviaciones computadas con los datos de entrenamiento algoritmo 1.

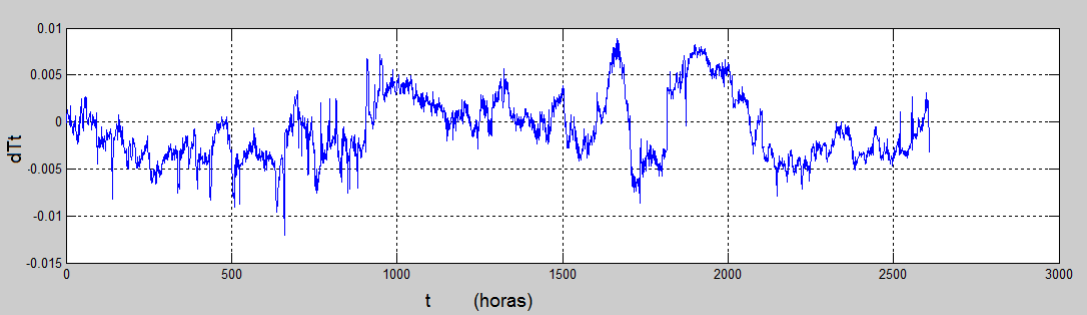


Figura 4.24 Desviaciones computadas con los datos de entrenamiento versión 1.1.

En la etapa de validación se utilizan todos los datos disponibles: los datos de entrenamiento más 1483 secciones adicionales de medición, 4091 secciones en total. Las figuras 4.11 y 4.12 que ilustran la validación tienen el mismo formato que las figuras previas. Las desviaciones se calculan aquí según la ecuación 2.10 donde la función de referencia $\overrightarrow{Y_0} = f(\vec{U})$ se define por una transformación simple del modelo de motor deteriorado, modelo determinado en la etapa de entrenamiento. Por la explicación anterior, las desviaciones calculadas en la validación incluyen: la influencia de deterioro del motor, en particular la contaminación de su compresor.

Como se puede observar en la figura 4.11, la grafica de desviaciones refleja correctamente el aumento gradual de la temperatura a causa de la contaminación antes y después del lavado del compresor en el punto t = 907 horas. Al menos eso se ve claramente en la parte izquierda de la gráfica, en tanto que en la parte derecha crecen las fluctuaciones (errores aleatorios).

En la figura 4.12 las tendencias sistemáticas en la parte izquierda de igual manera. Sin embargo la calidad de las desviaciones (relación entre el cambio sistemático y el ruido) parece peor aquí que en la figura 4.12. Parece que la inadecuación del modelo obtenido por el entrenamiento con la versión 1.1 ha resultado en una adecuación baja de la función de referencia, lo que causó una calidad peor de las desviaciones.

Así, a pesar de haber modificado varias veces tanto el número de épocas como los coeficientes de aprendizaje, hay que buscar todavía sus valores óptimos.

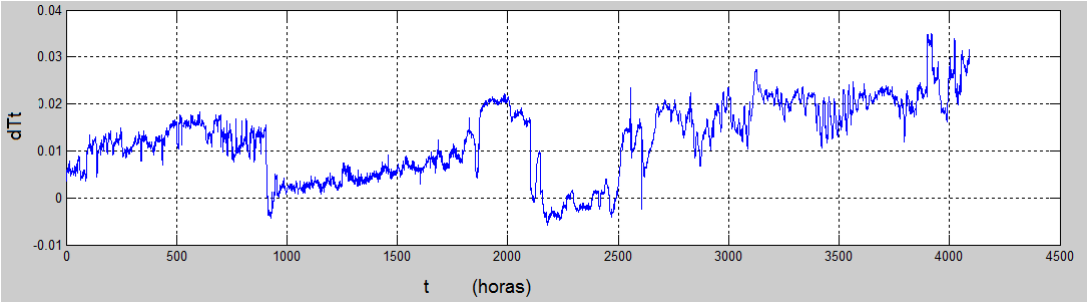


Figura 4.11 Desviaciones computadas con los datos de validación algoritmo 1.

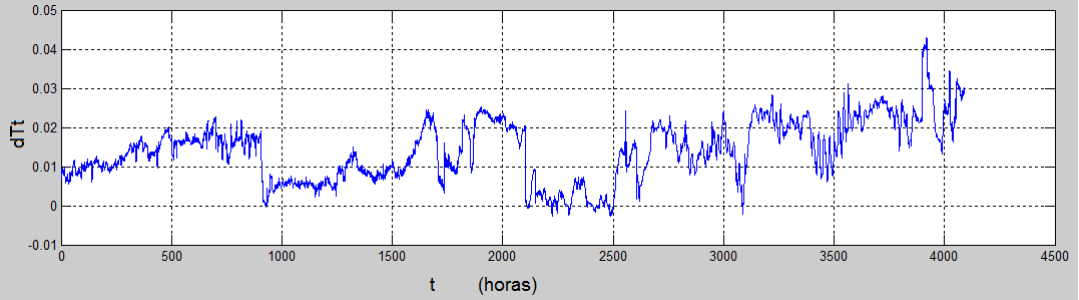


Figura 4.12 Desviaciones computadas con los datos de validación versión 1.1.

4.1.2. Algoritmo 2: cálculo de índices de confiabilidad del diagnóstico (versión 2.1)

4.1.2.1. Descripción

El algoritmo 2 calcula laos índices de confiabilidad del diagnóstico, es decir aplica la red neuronal de retro-propagación para reconocer que tan correcto es nuestro diagnóstico. Por lo cual asocia a cada vector de entrada a la red con un vector de salida, el cual corresponde a una clase de falla diferente.

En el algoritmo 2 la parte que modificaremos se muestra en la figura 4.13, de tal manera generaremos la versión 2.1 de este algoritmo.

```
% Net training
MiMa=minmax(P1);
nm=6; %Method number
if nm==2
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.epochs=100;
   net.trainParam.show=5;
   net.trainParam.mem_reduc=2;
elseif  nm==2
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.lr=0.1;
   net.trainParam.epochs=8000;
   net.trainParam.show=50;
elseif  nm==3
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.lr=0.1;
   net.trainParam.mc=0.5;
   net.trainParam.epochs=8000;
   net.trainParam.show=50;
elseif  nm==4
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.lr=0.1;
   net.trainParam.lr_inc=1.05;
   net.trainParam.epochs=1000;
   net.trainParam.show=50;
elseif  nm==5
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.lr=0.1;
   net.trainParam.mc=0.5;
   net.trainParam.lr_inc=1.02;
   net.trainParam.epochs=3000;
   net.trainParam.show=50;
elseif  nm==6
   net =newff(MiMa,[12,9],{'tansig','logsig'),'train
   net.trainParam.epochs=300;
   net.trainParam.show=20;
elseif  nm==7
   net =newff(MiMa,[12,9],{'tansig','logsig'),'traincgf');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==8
   net =newff(MiMa,[12,9],{'tansig','logsig'),'traincgp');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==9
   net =newff(MiMa,[12,9],{'tansig','logsig'),'traincgb');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==10
   net =newff(MiMa,[12,kdf],{'tansig','logsig'),'trainscg');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==11
   net =newff(MiMa,[12,9],{'tansig','logsig'),'trainbfg');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==12
   net =newff(MiMa,[12,9],{'tansig','logsig'),'trainoss');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
end;
net.trainParam.goal=0.026;
if i_EarlyStop==1
   val.P=P2;
   val.T=T2;
   [net,tr]=train(net,P1,T1,[],[],val);
else
   [net,tr]=train(net,P1,T1);
end;
A1=sim(net,P1);    %Simulation
A2=sim(net,P2);    %Simulation
```

Figura 4.13 Modificación al algoritmo prototipo 2.

Para hacer esta modificación en el algoritmo 2 procedemos a programar nuestra red neuronal de retro-propagación (versión 2.1). Para lo cual necesitamos primero hacer un

arreglo de la matriz de entrada o lo que es igual a nuestros vectores de entrada $\overrightarrow{X_t}$ de a cuerdo a las formulas 3.2 a 3.11, debido a que nuestro algoritmo trabajara de la forma en serie, mientras que el prototipo trabaja por lotes.

Para esto empezamos modificando la entrada, ver figura 4.14, consideramos que esta modificación mejora notablemente el funcionamiento de la red, debido a que presenta los vectores de entrada de tal forma que pasa un vector de cada tipo de falla (tabla 4.1 y 4.2), la cual nos queda:

```
124 -  for lp = 1:1000
125 -      for mp = 1:9
126 -          n = (lp-1)* kdf+mp;
127 -          P1 (:,n) = ZD1(:,lp,mp);
128 -          T1 (mp,n) = 1;
129 -      end
130 -  end
```

Figura 4.14 Modificación de los datos de entrada.

No.	Matriz P1								
1	-3.92652878	0.15548564	1.79067029	0.39771643	0.70575018	0.0815613	1.13647307	-0.47497617	-2.60911633
2	-4.4551225	0.61294425	-0.93229939	0.16925808	2.39718247	0.60821783	-0.08002938	0.54655246	-2.52911956
3	-0.5844782	-0.12324737	0.35500735	0.30266921	-0.07257992	-0.70033221	0.24732897	-0.53159654	0.1716696
4	-1.83691106	0.42483227	-1.34331759	0.18333328	2.31562664	-0.19677238	0.31083678	0.39870607	0.10406928
5	-1.27075839	0.93673043	-1.26648912	-0.40502193	0.84061632	0.9460642	0.94272913	-0.20532818	0.58429666
6	-3.91968731	0.29738879	-2.28387323	0.42849815	1.64254701	-0.50389979	-0.04218719	1.01458119	-1.92863171
	1	2	3	4	5	6	7	8	9

Tabla 4.1 Matriz de Entrada P1.

No.	Matriz T1								
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	1
clases	1	2	3	4	5	6	7	8	9

Tabla 4.2 Matriz del vector deseado T1.

Posteriormente sabemos del algoritmo 2 que en adición a la entrada P1 y salida deseada T1, primero tenemos que determinar la configuración de la red, debido a que hemos decidido poner como una función creada por nosotros mismo, por lo tanto iniciamos el cálculo de la red introduciendo valores importantes como se muestra en la figura 4.15, dichos valores son: el numero de neuronas de la capa entrada, numero de neuronas en la capa oculta y numero de neuronas de la capa de salida, además de parámetros que son necesarios para el cálculo, como son el valor de alfa, el cual determina la relación de aprendizaje de la red; el error mínimo, el cual se establece como objetivo para el error; el numero de iteraciones o épocas, el cual se establece para repetir los cálculos, este número es muy variable y muy importante debido a que es el que ayudara a tener un buen entrenamiento de la red; y por último es numero de patrones, que es el numero de datos que la red se debe de aprender.

```
118     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119
120     %%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%
121     %%%%%%%%%%%%%%%%%      ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%
122
123     % VALORES DE ENTRADA
124 -   N = 6;  %input('Numero de neuronas de entrada: ');
125 -   L = 12; %input('Numero de neuronas ocultas: ');
126 -   M = 9; %input('Numero de neuronas de salida: ');
127 -   alfa = 0.0025; %input('Valor de alfa: ');
128 -   min = 0.015;    %input('Minimo error: ');
129 -   iter = 5; %input('Numero de iteraciones: ');
130 -   patrones = 9000;   %input('Numero de patrones: ');
131
132 -   [matriz_Wh_ji, matriz_Wo_kj] = ent_bpj (N, L, M, alfa, min, iter, patrones, P1, T1)
133
```

Figura 4.15 Parámetros para determinar las características de la red.

Una vez determinados estos datos, entramos a la propia red con estos datos, sin embargo necesitamos acomodar los datos para que puedan ser procesados de buna manera y la red trabaje eficientemente, además de determinar los valores de las matrices de pesos Wh_jiy la Wo_kj, como se observan en las tablas 4.3 y 4.4 respectivamente, por lo cual hacemos los arreglos pertinentes como se muestra en la figura 4.16.

No.	Matriz de pesos Wh_ji											
1	0.06124591	-0.02112768	0.06074605	-0.06890569	-0.00800343	0.05889899	-0.09501186	-0.03560925	-0.00123151	0.04613836	0.0951466	0.00197174
2	0.00563334	-0.01101265	-0.01525772	-0.09343627	0.01614434	-0.04373463	0.0902671	-0.00652143	0.03808911	-0.03125636	0.00882429	0.03541482
3	0.0331683	0.01644945	-0.07806101	-0.09004385	-0.00674667	-0.0166913	-0.00715042	-0.08003782	-0.00497574	-0.02611101	-0.05333594	-0.08060826
4	0.04984334	-0.00181043	-0.00958876	-0.06891026	0.04621313	-0.02334571	0.00193071	0.0098436	-0.00871255	0.03787215	0.08883258	0.02454828
5	-0.01473364	-0.07357336	-0.05458445	0.09833388	0.03032575	-0.0432708	-0.01554567	0.03216861	-0.05786642	0.02001844	-0.07440827	-0.03368215
6	0.06307123	0.04489557	-0.03394035	-0.01530504	-0.05876597	0.06854718	-0.02774565	-0.0523722	0.00758481	-0.02709226	-0.04950686	-0.08525625
7	0.03333529	0.0967497	0.05199718	0.04924899	0.07965957	-0.07978841	-0.04528996	0.01493272	-0.03923551	0.05412068	-0.09575655	0.04671189
	1	2	3	4	5	6	7	8	9	10	11	12

Tabla 4.3 Matriz de pesos W_{hji}

No.	Matriz de pesos Wo_kj								
1	0.00960614	-0.02425649	-0.03461553	-0.07781107	0.09668307	0.09029518	-0.01694315	-0.03270567	0.07722098
2	-0.08405092	0.05018555	-0.05662764	-0.06999113	-0.08453737	0.08687293	-0.08976513	0.09968336	-0.04956429
3	0.04032582	0.05719098	0.02720002	-0.0040755	0.02647926	0.03604039	0.00611595	0.04470321	0.02815656
4	-0.0763791	0.06898239	-0.02248419	-0.09434995	-0.09374173	0.00028644	0.03552736	0.07283868	0.07880605
5	-0.06449681	-0.0428334	0.02343189	0.05158427	-0.01571719	-0.02810043	0.05409508	-0.0044061	0.0425281
6	-0.01441824	0.08660312	-0.03408775	0.0313944	-0.01236979	0.07733468	-0.0701334	-0.01734542	0.00430594
7	-0.07384193	0.03765804	-0.0940734	-0.01607755	-0.02665446	0.05463001	0.02666339	-0.04141995	-0.05069535
8	0.09052061	0.08282572	-0.04754947	0.0433847	-0.02257513	-0.02818675	-0.07475112	-0.0139325	-0.04370875
9	-0.03740072	0.09079805	-0.00862734	-0.048118	0.0201615	0.04310872	-0.08804312	-0.08619077	-0.02062949
10	-0.03596761	-0.05367471	0.04034959	0.05168227	0.04304367	-0.0217218	-0.03331984	0.02186423	0.08187672
11	-0.05609828	-0.0062006	-0.03589769	-0.09932313	0.04268198	0.01755804	-0.05486845	0.01011535	-0.02322552
12	-0.0693788	0.04130792	-0.05699417	0.06569683	0.01524432	0.04170873	-0.01330966	0.09560197	-0.04569471
13	-0.0623612	-0.01852468	-0.06700933	-0.06375745	0.079919	-0.08499363	0.0272716	0.0614827	0.05146977
	1	2	3	4	5	6	7	8	9

Tabla 4.4 Matriz de pesos W_{okj}

```
function [Wh_ji, Wo_kj] = ent_bpj (N, L, M, alfa, min, iter, patrones, P1, T1)
% INICIALIZACION DE LA MATRIZ DE PESOS W Y V
Wh = 0.2*rand(N+1,L)-.1;%0.2
Wo = 0.2*rand(L+1,M)-.1;%0.2
Wh_ji = Wh';
Wo_kj = Wo';
% Agregando el Bias a la entrada
B1 = [P1];
B = B1';
PB1=[];
for g=1:patrones
    PB1(g,:)=[1,B(g,:)];
end
PB_e = PB1';
PB = PB_e;
T1_e =T1;
T =T1_e;
% INICIA ENTRENAMIENTO
epoch = 1;
cont=0;
Z=0;
```

Figura 4.16 Declaración de las matrices Wh_j y la Wo_kj y arreglos de datos.

A continuación se empieza el cálculo de la red, pero como ya se ha dicho en capítulos anteriores, primero se pasa un vector de entrada, para procesarlo y obtener un valor calculado, de esta forma compararlo con el deseado y posteriormente propagar esta diferencia (error) hacia atrás en el proceso de aprendizaje. El vector de entrada tiene la forma de la tabla 4.5.

No.	Matriz de Entrada con vectores Xi								
1	1	1	1	1	1	1	1	1	1
2	-3.92652878	0.15548564	1.79067029	0.39771643	0.70575018	0.0815613	1.13647307	-0.47497617	-2.60911633
3	-4.4551225	0.61294425	-0.93229939	0.16925808	2.39718247	0.60821783	-0.08002938	0.54655246	-2.52911956
4	-0.5844782	-0.12324737	0.35500735	0.30266921	-0.07257992	-0.70033221	0.24732897	-0.53159654	0.1716696
5	-1.83691106	0.42483227	-1.34331759	0.18333328	2.31562664	-0.19677238	0.31083678	0.39870607	0.10406928
6	-1.27075839	0.93673043	-1.26648912	-0.40502193	0.84061632	0.9460642	0.94272913	-0.20532818	0.58429666
7	-3.91968731	0.29738879	-2.28387323	0.42849815	1.64254701	-0.50389979	-0.04218719	1.01458119	-1.92863171
	1	2	3	4	5	6	7	8	9

Tabla 4.5 Matriz de Entrada con Vectores X_i

Hemos incluido el proceso del cálculo del error en un función creada por nosotros mismo, dicha función se muestra en la figura 4.17.

```
function [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T)
Xi = PB(:,ip);     %%%%%%% INICIA VECTOR DE ENTRADA A LA RED
Vh_j = (Wh_ji*Xi); %%%%%%% PROPAGACION HACIA ADELANTE DE LOS PESOS
Uh = tansig(Vh_j); %%%%%%% SE UTILIZA LA FUNCION DE TRANSFERENCIA
Uh_j = [1.0,Uh'];
Uo_k = logsig(Wo_kj*Uh_j'); % SE UTILIZA LA FUNCION DE TRANSFERENCIA
E= T(:,ip)-Uo_k; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA DE SALIDA
```

Figura 4.17 Proceso del cálculo del error

Posteriormente después de haber calculado el error; de acuerdo a nuestra función creada el error es un parámetro de salida. Para empezar a retro-propagarlo, el cálculo de las derivadas

de los errores de los pesos $\frac{\partial E_p}{\partial W_{okj}}$ y $\frac{\partial E_p}{\partial W_{hji}}$, como se puede observar corresponden con las ecuaciones 3.9 y 3.11 del algoritmo de retro-propagación vistas anteriormente, dichas derivadas corresponden a "*delta1*" y "*delta2*" del programa como se observa en la figura 4.18, cabe mencionar que el tipo de error que se considera para el cálculo de la retro-propagación es un error cuadrático medio.

```
for kp = 1:iter
    cont=cont+1;
    error=0;
    for ip = 1:patrones
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        d_k = E;
        jes_1 = (d_k.*((1-Uo_k).*Uo_k));
        delta1 = (-1).*jes_1*Uh_j;
        Wo_kj = Wo_kj - alfa.*delta1; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE SALIDA
        a=jes_1'*Wo_kj; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA DE ENTRADA
        jes_2 = (-1).*a.*((1.+Uh_j).*(1.-Uh_j));
        jes_2 = jes_2(2:length(jes_2));
        delta2 = jes_2'*Xi';
        Wh_ji = Wh_ji -alfa.*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE ENTRADA
        error = error + sum((d_k.*d_k)/M); %%%%% CALCULO DE ERRORRES
    end
    error = error/patrones; %%%% ERROR POR EPOCA
    vec_err(1,kp) = error;
    J(1,kp)=error;
    if error<min
        break;
    end

end
```

Figura 4.18 Cálculo de retro-propagación de la red.

Este proceso se realiza *n* veces de acuerdo al parámetro de "*iter*" de nuestro programa para asegurar un buen entrenamiento de la red. Para hacer ver que tan bien se entreno a la red, hacemos un grafico en base al error, el cual nos indicara el nivel hasta el cual llego este, sin embargo este error que utilizamos es diferente al error que se propaga, debido a que este error es un error cuadrático promedio de cada época o iteración del cálculo, para lo cual se hace la grafica de acuerdo a la figura 4.19, y al final simplemente guardamos las matrices finales Wh_ji y la Wo_kj con los pesos corregidos. Los cuales se utilizaran en la simulación de los datos entrenados y los datos de validación.

```

figure(1);
plot(J);
xlabel('Epocas');
ylabel('Error Medio Cuadratico');
grid on;

save Wh_ji.tx Wh_ji -ascii
save Wo_kj.tx Wo_kj -ascii
save error J;
```

Figura 4.19 Programación de la grafica del error.

Después de entrenar la red, necesitamos simularla para saber cómo funciona y que tanto es lo que se aprendió, para esto primero reacomodamos las matrices de entrada tanto de la muestra de entrenamiento como la de validación como se observa en la figura 4.20.

```
156      % Formation of net entries
157 -    for idf=1:kdf
158 -        for itd=1:ktd
159 -            n=(idf-1)*ktd+itd;
160 -            P1_s(:,n)=ZD1(:,itd,idf);
161 -            T1_s(idf,n)=1;
162 -        end;
163 -    end;
164 -    for idf=1:kdf
165 -        for itd=1:ktd
166 -            n=(idf-1)*ktd+itd;
167 -            P2_s(:,n)=ZD2(:,itd,idf);
168 -            T2_s(idf,n)=1;
169 -        end;
170 -    end;
171
172 -    Wh_ji= matriz_Wh_ji;
173 -    Wo_kj= matriz_Wo_kj;
```

Figura 4.20 Acomodo de los datos de entrenamiento y validación para la etapa de simulación.

En la fase de validación utilizamos la matriz P1 y la matriz P2 cada una con sus respectivas salidas T1 y T2, con lo que obtendremos las matrices de salida correspondientes a cada matriz de entrada, ver figura 4.21, para que con estas podamos obtener las probabilidades de reconocimiento de cada matriz según lo que ya se aprendió la red neuronal.

```
178      % Agregando el Bias a la entrada
179 -    B1_1 = [P1_s];
180 -    B_1 = B1_1';
181 -    PB_1=[];
182 -    for g = 1:patrones
183 -        PB11_1(g,:) = [1,B_1(g,:)];
184 -    end
185 -    PB1_1 = PB11_1';
186 -    PB = PB1_1;
187 -    T = T1_s;
188 -    aux_1 = 0;
189
190 -    for ip = 1:patrones
191 -        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
192 -        Uo_k_1 = Uo_k;
193 -        A1 (:,ip)= Uo_k_1';
194 -    end
```

```
196      % Agregando el Bias a la entrada
197 -    B1_2 = [P2_s];
198 -    B_2 = B1_2';
199 -    PB_2=[];
200 -    for g = 1:patrones
201 -        PB11_2(g,:) = [1,B_2(g,:)];
202 -    end
203 -    PB1_2 = PB11_2';
204 -    PB = PB1_2;
205 -    T_2 = T2_s;
206 -    T = T2_s;
207 -    aux_2 = 0;
208
209 -    for ip = 1:patrones
210 -        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
211 -        Uo_k_2 = Uo_k;
212 -        A2 (:,ip)= Uo_k_2';
213 -    end
```

Figura 4.21 Etapa de simulación de la red neuronal.

4.1.2.1. Resultados

Con nuestro algoritmo del diagnóstico que hemos programado, realizamos diferentes cálculos para entrenar adecuadamente la red y comparar los resultados con los del algoritmo 2. Se realizaron varias pruebas cambiando el número de épocas para poder verificar si el cambio del error varía de forma drástica además de ver si podíamos tomar algún valor de épocas como aceptable antes de llegar al valor mínimo del error. Esto por un lado, y por otro se observo las probabilidades obtenidas para cada cambio de época, para concluir si podíamos aceptar o no estás probabilidades o si realmente necesitábamos entrenar a la red con más épocas para llegar a probabilidades más aceptables.

En dichas pruebas variamos el número de épocas, de tal forma que primero consideramos 100, 200, 300, 500 y 1000, obteniendo de cada una de ellas su grafica del error respectivamente. Para los valores 100 y 1000 las graficas del error están representadas por las figuras 4.22 y 4.23. La grafica correspondiente al prototipo está en la figura 4.24.

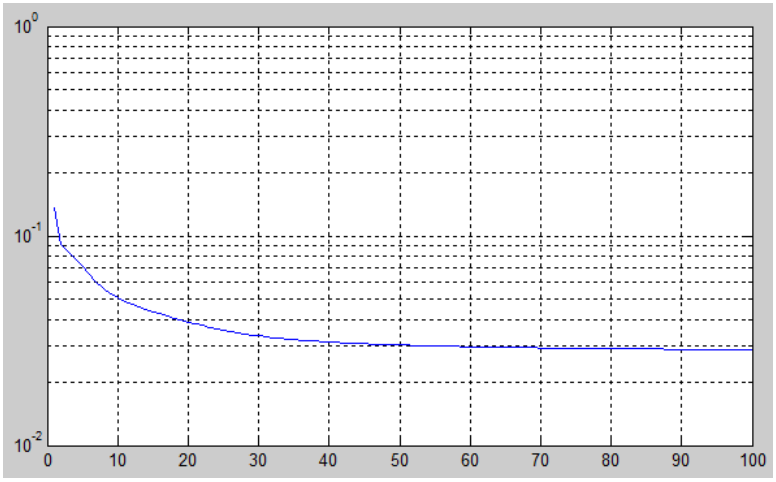


Figura 4.22 Grafica del Error para 100 épocas.

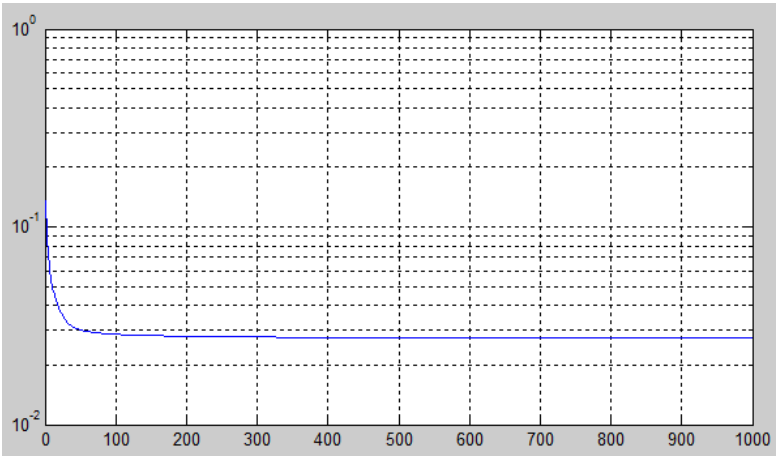


Figura 4.23 Grafica del Error para 1000 épocas.

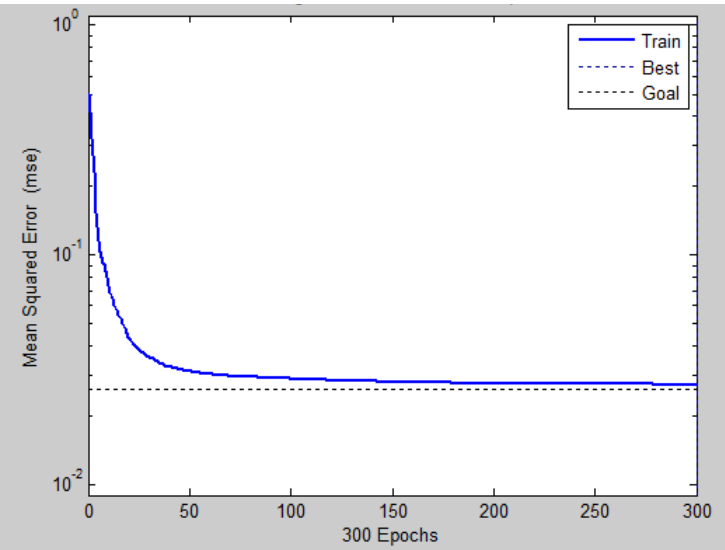


Figura 4.24 Grafica del Error del algoritmo prototipo.

Comparando las graficas del algoritmo y prototipo podemos llegar a la conclusión que el comportamiento de las graficas y errores alcanzados son prácticamente iguales. La tabla 4.6 incluye las probabilidades para los cálculos mencionados con los números de épocas 100, 200, 300, 500 y 1000. Se puede decir que para fines prácticos podemos considerar el de 300 épocas aceptables para poder hacer alguna prueba más.

Con ALFA = 0.0025		Clase1	Clase2	Clase3	Clase4	Clase5	Clase6	Clase7	Clase8	Clase9	Promedio
Epoch =100	PD1	0.8400	0.7340	0.8780	0.7220	0.8520	0.8280	0.8510	0.8350	0.8360	0.8196
	PD2	0.8430	0.7290	0.8900	0.7520	0.8470	0.7890	0.8720	0.8320	0.8440	0.8220
Epoch =200	PD1	0.8540	0.7420	0.8810	0.7250	0.8590	0.8310	0.8500	0.8330	0.8380	0.8237
	PD2	0.8490	0.7300	0.8840	0.7540	0.8480	0.7900	0.8690	0.8240	0.8340	0.8202
Epoch =300	PD1	0.8530	0.7470	0.8770	0.7250	0.8570	0.8380	0.8470	0.8330	0.8360	0.8237
	PD2	0.8480	0.7300	0.8830	0.7500	0.8510	0.7920	0.8670	0.8240	0.8320	0.8197
Epoch =500	PD1	0.8480	0.7530	0.8770	0.7240	0.8590	0.8440	0.8490	0.8290	0.8410	0.8249
	PD2	0.8450	0.7330	0.8830	0.7450	0.8530	0.7940	0.8630	0.8210	0.8330	0.8189
Epoch =1000	PD1	0.8490	0.7580	0.8780	0.7160	0.8630	0.8470	0.8450	0.8280	0.8460	0.8256
	PD2	0.8490	0.7400	0.8840	0.7430	0.8560	0.7980	0.8640	0.8130	0.8310	0.8198
Prototipo	PD1	0.8540	0.7350	0.8710	0.7460	0.8700	0.8520	0.8450	0.8270	0.8340	0.8260
	PD2	0.8530	0.7250	0.8860	0.7630	0.8660	0.8070	0.8660	0.8160	0.8230	0.8228

Tabla 4.6 Probabilidades para cada variación de épocas.

Después se hizo una prueba más variando el valor de alfa para observar si este podría mejorar el rendimiento de la red, para los cuales se tomo como base el experimento con 300 épocas el cual consideramos con aceptable de todas las épocas, además que el algoritmo prototipo aplica el mismo número de épocas, se vario el valor de alfa en 0.0020 y 0.0030 considerando que el del experimento es 0.0025, para dichos cálculos obtuvimos los siguientes resultados tanto error cuadrático promedio figuras 4.25 y 4.26 respectivamente, y tanto las probabilidades tabla 4.7.

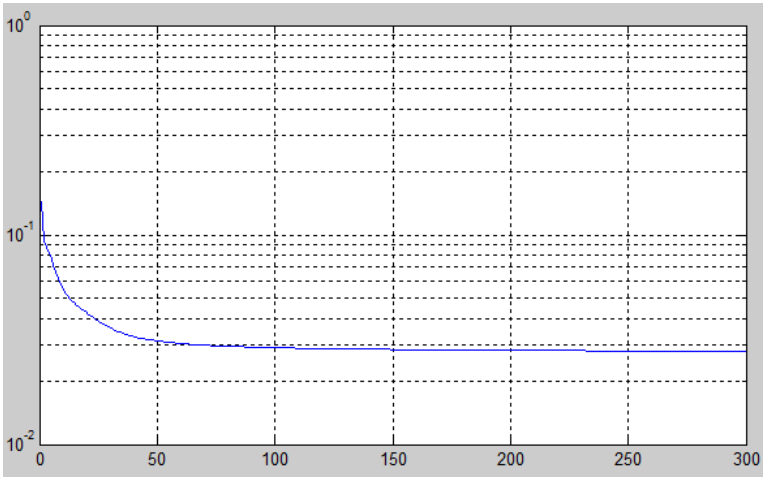


Figura 4.25 Grafica del Error para 300 épocas y con alfa = 0.0020.

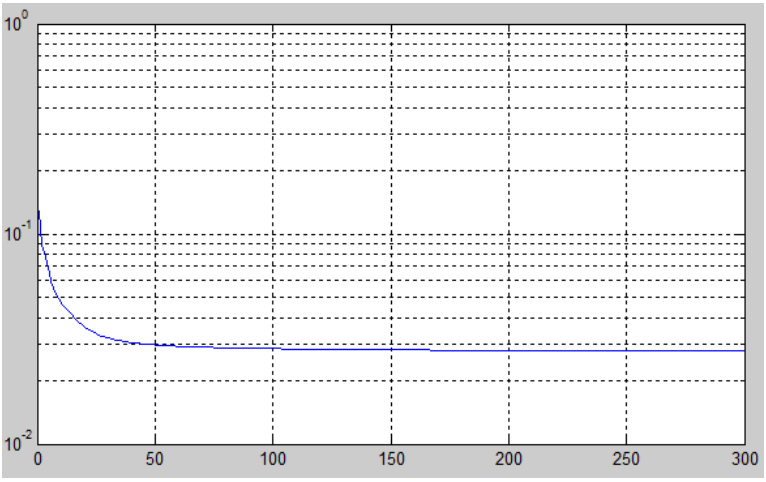


Figura 4.26 Grafica del Error para 300 épocas y con alfa = 0.0030.

Las probabilidades que se obtuvieron con estas modificaciones fueron las siguientes:

Con ALFA = variable		Clase1	Clase2	Clase3	Clase4	Clase5	Clase6	Clase7	Clase8	Clase9	Promedio
Prototipo	PD1	0.8540	0.7350	0.8710	0.7460	0.8700	0.8520	0.8450	0.8270	0.8340	0.8260
	PD2	0.8530	0.7250	0.8860	0.7630	0.8660	0.8070	0.8660	0.8160	0.8230	0.8228
Epoch =300 Alfa =0.0025	PD1	0.8530	0.7470	0.8770	0.7250	0.8570	0.8380	0.8470	0.8330	0.8360	0.8237
	PD2	0.8480	0.7300	0.8830	0.7500	0.8510	0.7920	0.8670	0.8240	0.8320	0.8197
Epoch =300 Alfa =0.0020	PD1	0.8540	0.7400	0.8760	0.7300	0.8570	0.8350	0.8490	0.8320	0.8370	0.8233
	PD2	0.8470	0.7270	0.8830	0.7560	0.8470	0.7900	0.8680	0.8250	0.8330	0.8196
Epoch =300 Alfa =0.0030	PD1	0.8520	0.7520	0.8780	0.7200	0.8580	0.8360	0.8480	0.8310	0.8360	0.8234
	PD2	0.8440	0.7320	0.8830	0.7460	0.8520	0.7920	0.8660	0.8220	0.8310	0.8191

Tabla 4.7. Probabilidades para cada variación de alfa (coeficiente de entrenamiento).

Como se puede ver tanto en la tabla 4.6 como en la tabla 4.7 las probabilidades que hemos obtenido están muy cercanas a las deseadas. Esto quiere decir que los valores de alfa alrededor de valores 0.0025 están aceptables y no hay que cambiar este valor.

4.2. Cálculo numérico de las derivadas (versión 2.2 del algoritmo 2)

4.2.1. Descripción

Habiendo desarrollado el algoritmo de la Red Neuronal con entrenamiento de Retro-propagación (ecuaciones 3.2 a la 3.11), deberíamos saber si es que el desarrollo del software funciona de la mejor manera, por lo cual tenemos que tener alguna referencia que nos indique este funcionamiento. El desarrollo del algoritmo de dicha red, como sabemos es un cálculo para encontrar las derivadas del error, siendo estas las principales fuentes para poder tener una referencia del funcionamiento de dicho software.

Nuestro siguiente trabajo consiste en determinar las derivadas de nuestra versión 2.1, tomando en cuenta que es la aplicación que obtuvimos mejores resultados fue en el diagnóstico detallado (reconocimiento de patrones), será con esta versión con la que trabajaremos, es decir la versión 2.1 del prototipo 2. Dicho trabajo se explica a continuación.

Para hacer esta comparación hemos decidido desarrollar otro algoritmo que nos calcule estas derivadas de forma numérica, es decir, haciendo la linealización obteniendo de esta forma la versión 2.2 del algoritmo 2. Sabemos que las derivadas se pueden calcular según la ecuación 4.1.

$$\frac{\partial E}{\partial W} \approx \frac{\Delta E}{\Delta W} \tag{4.1}$$

Por otro lado las derivadas pasan a formar una diferencia según la ecuación 4.2.

$$\frac{\Delta E}{\Delta W} = \frac{E_1 - E_0}{W_1 - W_0} \tag{4.2}$$

Donde vemos que tanto E_1 y E_0 son funciones de la matriz de pesos W y el cambio ΔW según la ecuación 4.3.

$$\frac{\Delta E}{\Delta W} = \frac{E_{1(W_0 - \Delta W)} - E_0(W_0)}{\Delta W} \tag{4.3}$$

De acuerdo al software de nuestra red neuronal con entrenamiento de retro-propagación, descrita en el capítulo 3, son dos las derivadas que se calculan en cada conjunto de la retro-propagación, una en el cálculo de la capa de salida y otra en la capa oculta, por lo que aplicaremos las ecuaciones anteriores a estas dos capas. Como ya se ha analizado en este capítulo el algoritmo completo, es decir el cálculo de la clasificación de fallas, nos enfocaremos a simplemente describir la parte de la red neuronal, básicamente en el entrenamiento y en el cálculo de la adaptación de los pesos de la red para la versión 2.2.

La idea de esta versión consiste en verificar lo más complejo en la versión 2.1, el cálculo de las derivadas. A diferencia de la versión 2.1, en esta versión las derivadas se calculan numéricamente.

Para la versión 2.2 las modificaciones fueron relativamente pocas en cuanto a programación se refiere, sin embargo en lo que se requiere a los cálculos matemáticos las modificaciones fueron muy significativas, como se menciono al inicio de esta sección para

este caso se trabajo con el cálculo numérico de las derivadas, mostrándose en general en las ecuaciones 4.1 a la 4.3.

Para emplear estas ecuaciones en nuestro cálculo, vamos a personalizarlas para identificarlas según corresponde con las ecuaciones 3.9 y 3.11, es decir que correspondan a las derivadas $\frac{\partial E_p}{\partial W_{okj}}$ y $\frac{\partial E_p}{\partial W_{hji}}$, que son de la capa de salida y la capa oculta respectivamente.

$$\frac{\partial E_p}{\partial W_{okj}} = \frac{d\_p1 - d\_p0}{\Delta W_{okj}} \tag{4.4}$$

$$\frac{\partial E_p}{\partial W_{hji}} = \frac{d\_p1 - d\_p0}{\Delta W_{hji}} \tag{4.5}$$

Cabe mencionar que esta forma del cálculo de las derivadas es un poco mas tardada, sin embargo lo que nos interesa de este cálculo es tener una referencia de funcionamiento y a pesar de que sabemos que es un cálculo aproximado, esperamos que las dos versiones ofrezcan un mismo resultado.

Para empezar a explicar esta modificación, primeramente vamos a ubicarnos dentro del algoritmo, por lo que consideremos la figura 4.7, partamos de esta para entender mejor lo que se realizara, como ya se explico en dicha figura tenemos que empezar por declarar las matrices de pesos Wh_jy la Wo_kj, así como hacer los arreglos pertinentes para que trabaje adecuadamente el algoritmo de la red. Una vez realizado esto el vector de entrada es introducido a la red para calcular un error de acuerdo a la figura 4.17.

A la salida de esta función que hemos creado tenemos un error de salida, el cual para esta versión se calcula un error medio cuadrático, a su vez también se declaran los cambios de ΔW_{okj} y ΔW_{hji}, como se observa en la figura 4.27, los cuales marcaran un cambio constante para cada patrón de entrada, es decir que variaran en un porciento de las matrices de pesos correspondientes.

```
27 -  for kp = 1:iter
28 -      cont=cont+1;
29 -      error=0;
30 -      for ip = 1:patrones
31 -          [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
32 -          e = E.*E;
33 -          d_k = (1/2)*(sum(e));
34 -          d_p0 = d_k;
35 -          AWo_kj = (1).*(abs (.01.*Wo_kj));
36 -          AWh_ji = (1).*(abs (.01.*Wh_ji));
37 -          Wo_0=Wo_kj;
```

Figura 4.27 Declaración de los cambios de las matrices de pesos

De una manera similar que para el cálculo de las derivadas de la versión 2.1 tenemos que formar una matriz de las derivadas. Sin embargo en este caso y por la forma del cálculo lo hacemos con ciclo de trabajo, y para cada ciclo debemos llamar a la función que hemos creado para calcular un error particular de ese patrón. Sin embargo cómo se puede observar en la figura 4.28 se tiene que variar tanto la matriz de pesos como el error calculado pero solo para ese patrón. Es decir, tenemos un patrón para calcular la variación de la matriz de pesos que le corresponde a ese patrón como se puede ver en la línea 40 de la figura 4.28,

para posteriormente pasar a la función que creamos, la cual nos entregara un error. Con este calculamos el error cuadrático medio el cual le denominamos "*d_p1*"según nuestro algoritmo. Ahora si tenemos todos los elementos necesarios para aplicar la ecuación 4.4, tomando en cuenta que "*d_p0*" es el error que nos resulta de la figura 4.28.

```
38 -        for k = 1:M
39 -            for j = 1:L+1
40 -                Wo_kj(k,j) = Wo_0(k,j) + AWo_kj(k,j);
41 -                [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
42 -                e = E.*E;
43 -                d_p1 = (1/2)*(sum(e));
44 -                delta1(k,j) = (d_p1-d_p0)/AWo_kj(k,j);
45 -                Wo_kj(k,j) = Wo_0(k,j);
46 -            end
47 -        end
48 -        Wo kj=Wo 0;
```

Figura 4.28 Cálculo de la matriz de derivadas de la capa de salida.

Como esto es en ciclos y este primer cálculo solo corresponde al valor en la posición (1,1) de la matriz de tamaño (k,j), inicializamos nuevamente los valores obtenidos en la figura 4.28, para posteriormente hacer los cambios según corresponden con lo programado en la figura 4.29. De esta manera al completar el cálculo de esta derivada, solo habremos pasado el primer patrón, no se ha tocado algún otro patrón.

Continuando con el mismo patrón de entrada, realizamos el mismo cálculo anterior pero ahora para la matriz de derivadas de la capa oculta por lo cual aplicamos la ecuación 4.5. El cálculo es relativamente lo mismo. Tómese en cuanta que para los cálculos de las derivadas se toman como valores base o no modificables tanto el error "*d_p0*" y las matrices de pesos que hemos asignado como "*Wo_0*" y "*Wh_0*".

```
49 -        Wh_0=Wh_ji;
50 -        for j = 1:L
51 -            for i = 1:N+1
52 -                Wh_ji(j,i) = Wh_0(j,i) + AWh_ji(j,i);
53 -                [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
54 -                e = E.*E;
55 -                d_p1 = (1/2)*(sum(e));
56 -                delta2(j,i) = (d_p1-d_p0)/AWh_ji(j,i);
57 -                Wh_ji(j,i) = Wh_0(j,i);
58 -            end
59 -        end
60 -        Wh_ji=Wh_0;
```

Figura 4.29 Cálculo de la matriz de derivadas de la capa oculta.

Una vez calculadas las matrices de las derivadas, calculamos la corrección de las matrices de pesos tanto para la matriz de la capa de salida como para la matriz de la capa oculta, según figura 4.30, los cuales guardamos para ser utilizados en el cálculo de validación, además como se menciona en la versión 2.2 es necesario tener un parámetro para considerar el rendimiento del algoritmo y al igual que en la versión 2.1, calculamos un error cuadrático promedio, el cual graficamos para considerar que tan bien se entrenó la red y si es que alcanza un error mínimo.

```
76 -            Wo_kj = Wo_kj - alfa*delta1; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE SALIDA
77 -            Wh_ji = Wh_ji - alfa*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ DE LA CAPA DE ENTRADA
78 -            error = error + sum((E.*E)/M); %%%%% CALCULO DE ERRORRES
79 -        end
80 -        error = error/patrones; %%%% ERROR POR EPOCA
81 -        vec_err(1,kp) = error;
82 -        J(1,kp)=error;
83 -        if error<min
84 -            break;
85 -        end
86 -    end
87
88 -    figure(1);
89 -    plot(J);
90 -    xlabel('Epocas');
91 -    ylabel('Error Medio Cuadratico');
92 -    grid on;
93
94 -    save Wh_ji.txt Wh_ji -ascii
95 -    save Wo_kj.txt Wo_kj -ascii
96 -    save error J
```

Figura 4.30 Cálculo de corrección de pesos.

4.2.2. Resultados

El cálculo de las derivadas se ha tomado en cuanta para comparar el rendimiento de ambos casos por lo cual presentamos, los diferentes cálculos de estas para las dos versiones del algoritmo. En primer caso tenemos la versión 2.1 en donde podemos observar en las tablas 4.8 y 4.9 las derivadas del error con respecto a las matrices de la capa de salida y la capa oculta respectivamente.

No.	Derivada de la capa de salida												
1	-0,134114713	0,041693984	0,04526088	-0,05251107	-0,04924719	0,04648919	-0,07426424	0,02354341	-0,03780784	-0,01693079	-0,00651006	-0,09061876	-0,0258002
2	0,123179823	-0,038294512	-0,04157059	0,04822964	0,04523188	-0,04269875	0,06820919	-0,02162382	0,03472522	0,01555036	0,00597927	0,08323026	0,02369661
3	0,117857587	-0,03663992	-0,03977445	0,04614578	0,04327754	-0,04085386	0,06526207	-0,02068952	0,03322484	0,01487848	0,00572092	0,07963413	0,02267275
4	0,119823313	-0,037251031	-0,04043784	0,04691544	0,04399936	-0,04153525	0,06635056	-0,0210346	0,03377899	0,01512663	0,00581634	0,08096233	0,0230509
5	0,131134164	-0,040767382	-0,04425501	0,05134407	0,04815273	-0,04545602	0,0726138	-0,02302018	0,0369676	0,01655453	0,00636538	0,08860486	0,02522682
6	0,129295991	-0,040195925	-0,04363466	0,05062436	0,04747775	-0,04481884	0,07159593	-0,0226975	0,03644941	0,01632247	0,00627615	0,08736284	0,0248732
7	0,128599976	-0,039979546	-0,04339977	0,05035184	0,04722217	-0,04457757	0,07121052	-0,02257532	0,03625319	0,01623461	0,00624237	0,08689256	0,02473931
8	0,124309977	-0,038645858	-0,04195199	0,04867214	0,04564687	-0,0430905	0,06883499	-0,02182222	0,03504381	0,01569303	0,00603413	0,08399389	0,02391402
9	0,130364245	-0,040528027	-0,04399518	0,05104262	0,04787001	-0,04518914	0,07218746	-0,02288503	0,03675055	0,01645733	0,00632801	0,08808464	0,02507871
No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabla 4.8 Cálculo de las derivadas del primer cálculo capa de salida versión 2.1.

No.	Derivadas de la capa oculta						
1	-0,0026749	0,01050306	0,011917	0,00156342	0,00491355	0,00339915	0,01048476
2	0,02004555	-0,07870943	-0,08930538	-0,01171619	-0,03682189	-0,02547305	-0,07857229
3	0,01378886	-0,05414237	-0,06143108	-0,00805929	-0,02532892	-0,01752232	-0,05404804
4	0,01595587	-0,06265118	-0,07108536	-0,00932586	-0,02930951	-0,02027606	-0,06254202
5	0,00900407	-0,03535475	-0,04011424	-0,00526268	-0,01653968	-0,011442	-0,03529315
6	-0,00237983	0,00934446	0,01060242	0,00139096	0,00437153	0,00302418	0,00932818
7	-0,02512872	0,09866865	0,11195153	0,01468719	0,04615923	0,03193253	0,09849673
8	-0,00662054	0,02599572	0,02949529	0,00386956	0,01216133	0,0084131	0,02595043
9	0,02072202	-0,0813656	-0,09231913	-0,01211157	-0,03806451	-0,02633268	-0,08122383
10	-0,01032452	0,04053953	0,04599701	0,00603446	0,01896523	0,01311997	0,0404689
11	0,01462525	-0,05742646	-0,06515727	-0,00854814	-0,02686528	-0,01858516	-0,0573264
12	0,00753612	-0,02959078	-0,03357432	-0,0044047	-0,01384318	-0,00957658	-0,02953922
No.	1	2	3	4	5	6	7

Tabla 4.9 Cálculo de las derivadas del primer cálculo capa oculta versión 2.1.

En lo que se refiere a la versión 2.2 del algoritmo, se han considerado varios cálculos para ver cómo se comporta esta versión con respecto a la variación de los pesos que hemos considerado, inicialmente consideramos variar en un porciento la variación de las matrices con signo positivo, sin embargo nos interesaba ver el comportamiento con signo negativo,

por lo que al hacer estos cálculos con un porciento positivo y un porciento negativo, las derivadas que se obtuvieron para un porciento positivo se muestran en las tablas 4.10 y 4.11 para las capas de salida y oculta respectivamente.

No.	Derivada de la capa de salida												
1	-0,134112263	0,041696055	0,04526205	-0,05250809	-0,04924497	0,04648963	-0,07425846	0,02354412	-0,03780708	-0,01693065	-0,00651003	-0,09061068	-0,02579961
2	0,123187614	-0,038292954	-0,04156849	0,04823304	0,04523373	-0,0426954	0,06821289	-0,021623	0,03472753	0,01555064	0,00597927	0,08323632	0,02369683
3	0,117869448	-0,036638044	-0,03977338	0,04614696	0,04327863	-0,04085246	0,06527195	-0,02068902	0,03322508	0,0148787	0,00572095	0,07964304	0,0226736
4	0,119849406	-0,037248762	-0,04043768	0,04692029	0,04400169	-0,04153399	0,06635222	-0,02103415	0,03378028	0,01512691	0,00581642	0,08097239	0,0230517
5	0,131160913	-0,04076512	-0,04425417	0,05134805	0,04815331	-0,04545561	0,07261606	-0,02301999	0,03696804	0,01655472	0,00636541	0,08860679	0,02522764
6	0,129322044	-0,040193502	-0,04363348	0,05062437	0,04747884	-0,04481616	0,07160077	-0,02269725	0,03645039	0,01632257	0,00627617	0,08736834	0,02487411
7	0,128604938	-0,039977005	-0,04339957	0,05035344	0,04722431	-0,04457511	0,07121292	-0,02257464	0,03625524	0,01623476	0,00624241	0,08689434	0,0247396
8	0,124320307	-0,038642814	-0,04195038	0,04867567	0,04564706	-0,04308984	0,068839	-0,02182208	0,03504598	0,01569314	0,00603414	0,08400767	0,02391474
9	0,130386005	-0,040526677	-0,04399427	0,05104602	0,04787163	-0,04518899	0,07219184	-0,02288465	0,03675102	0,0164577	0,00632802	0,08809052	0,02507924
No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabla 4.10 Cálculo de las derivadas del primer cálculo capa de salida +0.01 versión 2.2.

No.	Derivadas de la capa oculta						
1	-0,00277596	0,01090125	0,01237111	0,00162271	0,00509987	0,00352832	0,01088402
2	0,01993914	-0,07827397	-0,08880195	-0,01165309	-0,03660638	-0,0253309	-0,07804547
3	0,0139057	-0,0546227	-0,06203434	-0,00812934	-0,025557	-0,0176768	-0,05455318
4	0,01606664	-0,06318129	-0,0716953	-0,00939416	-0,02953851	-0,02042308	-0,06303337
5	0,00889433	-0,03491378	-0,03961925	-0,00519781	-0,01633389	-0,01129877	-0,03481791
6	-0,00224037	0,00881137	0,00999062	0,00131012	0,00411943	0,00284994	0,00880488
7	-0,02519355	0,09885858	0,11221981	0,01472304	0,04627006	0,0320089	0,09871188
8	-0,00652598	0,02563075	0,02911974	0,00381493	0,0119922	0,00829635	0,02558947
9	0,02076681	-0,08155366	-0,09252084	-0,01213782	-0,03815085	-0,02638989	-0,08141191
10	-0,01030653	0,04047631	0,04592496	0,00602439	0,01893389	0,01309826	0,04040967
11	0,01474847	-0,05795985	-0,06585042	-0,00862852	-0,02713304	-0,01876127	-0,05798878
12	0,00760294	-0,0298561	-0,03388047	-0,0044438	-0,01396655	-0,00966224	-0,02980499
No.	1	2	3	4	5	6	7

Tabla 4.11 Cálculo de las derivadas del primer cálculo capa oculta +0.01 versión 2.2.

Como se puede observar las derivadas con respecto al cálculo de la versión 2.1 tablas 4.8 y 4.9 es muy parecida, sin embargo como se menciono anteriormente nos interesa ver cómo se comporta la red si cambiamos el porcentaje de cambio de la matriz, para lo cual presentamos en la tabla 4.12 y 4.13 con un porciento negativo para ambas capas como se hizo anteriormente.

No.	Derivada de la capa de salida												
1	-0,134117163	0,041691912	0,04525971	-0,05251406	-0,04924941	0,04648875	-0,07427001	0,0235427	-0,03780859	-0,01693094	-0,00651009	-0,09062683	-0,02580079
2	0,123172031	-0,03829607	-0,04157268	0,04822624	0,04523002	-0,04270209	0,06820548	-0,02162464	0,0347229	0,01555009	0,00597926	0,0832242	0,02369639
3	0,117845724	-0,036641795	-0,03977551	0,0461446	0,04327646	-0,04085526	0,06525218	-0,02069003	0,03322461	0,01487826	0,00572089	0,07962521	0,0226719
4	0,11979721	-0,037253299	-0,04043799	0,04691059	0,04399703	-0,04153652	0,06634891	-0,02103505	0,03377771	0,01512636	0,00581626	0,08095227	0,02305011
5	0,131107392	-0,040769643	-0,04425584	0,05134009	0,04815214	-0,04545643	0,07261153	-0,02302038	0,03696716	0,01655434	0,00636535	0,08860293	0,025226
6	0,129269919	-0,040198348	-0,04363585	0,05062434	0,04747665	-0,04482152	0,0715911	-0,02269775	0,03644842	0,01632237	0,00627614	0,08735734	0,0248723
7	0,128595014	-0,039982086	-0,04339998	0,05035025	0,04722003	-0,04458004	0,07120813	-0,02257599	0,03625114	0,01623445	0,00624233	0,08689078	0,02473901
8	0,124299644	-0,038648901	-0,0419536	0,04866861	0,04564669	-0,04309116	0,06883098	-0,02182235	0,03504165	0,01569292	0,00603412	0,08398009	0,0239133
9	0,130342471	-0,040529378	-0,04399608	0,05103921	0,0478684	-0,04518928	0,07218308	-0,02288541	0,03675009	0,01645696	0,00632799	0,08807876	0,02507817
No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabla 4.12 Cálculo de las derivadas del primer cálculo capa de salida -0.01 versión 2.2.

No.	Derivadas de la capa oculta						
1	-0,00277647	0,01090052	0,01236561	0,00162257	0,00509945	0,00352747	0,01087974
2	0,01993619	-0,07829766	-0,08884751	-0,01165317	-0,03664102	-0,02534101	-0,07825288
3	0,01391123	-0,05460126	-0,06189315	-0,00812905	-0,02554021	-0,0176718	-0,05448038
4	0,01607432	-0,06302077	-0,07149614	-0,00939153	-0,02950154	-0,02042032	-0,06294906
5	0,00889376	-0,03493164	-0,03962885	-0,00519894	-0,01634123	-0,01130558	-0,0349057
6	-0,00224225	0,00878977	0,00998001	0,00130987	0,00411475	0,00284639	0,00876562
7	-0,02518663	0,09895995	0,11223015	0,01472309	0,04627388	0,03201216	0,09876256
8	-0,00652785	0,02562548	0,02903653	0,00381475	0,01198652	0,00829192	0,02557745
9	0,02076686	-0,08152938	-0,09251676	-0,0121377	-0,03814278	-0,02638938	-0,08138698
10	-0,01030757	0,04046552	0,04591335	0,0060241	0,01893238	0,01309728	0,04039105
11	0,01476697	-0,0579334	-0,06564458	-0,00862262	-0,02708422	-0,01874573	-0,0577027
12	0,00760296	-0,02985027	-0,03386339	-0,00444371	-0,01396534	-0,00966077	-0,02979732
No.	1	2	3	4	5	6	7

Tabla 4.13 Cálculo de las derivadas del primer cálculo capa oculta -0.01 versión 2.2.

Para un cambio de cinco por ciento positivo se muestra en las tablas 4.14 y 4.15 para las capas de salida y capa oculta respectivamente.

No.	Derivada de la capa de salida												
1	-0,134102461	0,041704335	0,04526674	-0,05249613	-0,0492361	0,0464914	-0,07423533	0,02354697	-0,03780404	-0,01693006	-0,00650989	-0,09057832	-0,02579726
2	0,123218766	-0,038286721	-0,04156012	0,04824662	0,04524115	-0,04268203	0,06822772	-0,02161972	0,0347368	0,01555173	0,00597929	0,08326054	0,02369771
3	0,117916874	-0,036630541	-0,03976914	0,04615169	0,04328296	-0,04084684	0,06531146	-0,02068701	0,03322602	0,01487958	0,00572107	0,07967869	0,022677
4	0,119953672	-0,037239683	-0,04043706	0,04693968	0,04401102	-0,04152893	0,06635883	-0,02103236	0,03378541	0,01512801	0,00581673	0,0810126	0,02305486
5	0,131267674	-0,040756068	-0,04425083	0,05136394	0,04815566	-0,04545396	0,0726251	-0,02301922	0,03696982	0,01655548	0,00636552	0,08861449	0,02523091
6	0,129426061	-0,040183802	-0,04362874	0,05062442	0,04748321	-0,04480542	0,07162009	-0,02269624	0,03645435	0,01632297	0,00627621	0,08739031	0,02487774
7	0,128624776	-0,039966836	-0,04339875	0,05035981	0,04723285	-0,04456523	0,07122249	-0,02257194	0,03626343	0,01623538	0,00624256	0,08690145	0,02474079
8	0,124361607	-0,038630634	-0,04194395	0,04868977	0,04564781	-0,04308721	0,06885505	-0,02182154	0,03505463	0,01569358	0,00603417	0,08406275	0,02391762
9	0,130472899	-0,040521274	-0,04399066	0,05105964	0,04787809	-0,04518841	0,07220936	-0,02288313	0,03675286	0,01645917	0,00632809	0,08811402	0,02508139
No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabla 4.14 Cálculo de las derivadas del primer cálculo capa de salida +0.05 versión 2.2.

No.	Derivadas de la capa oculta						
1	-0,00277492	0,01090269	0,01238179	0,00162299	0,00510069	0,00353	0,01089236
2	0,01994503	-0,07822651	-0,0887106	-0,01165292	-0,03653677	-0,02531062	-0,07762506
3	0,0138946	-0,05466547	-0,06231252	-0,00812994	-0,02559043	-0,01768677	-0,05469752
4	0,01605125	-0,06349792	-0,07208759	-0,0093994	-0,02961196	-0,02042858	-0,0632008
5	0,00889548	-0,03487802	-0,03960002	-0,00519554	-0,01631919	-0,01128514	-0,03464095
6	-0,0022366	0,00885466	0,01001186	0,00131064	0,00412879	0,00285704	0,00888372
7	-0,02520722	0,09864654	0,11219905	0,01472294	0,04626239	0,03200234	0,09860818
8	-0,00652225	0,02564128	0,02928446	0,00381528	0,01200356	0,00830519	0,02561347
9	0,02076671	-0,0816008	-0,09252897	-0,01213807	-0,03816666	-0,0263909	-0,08146025
10	-0,01030445	0,04049742	0,04594769	0,00602497	0,0189369	0,01310021	0,04044549
11	0,0147115	-0,05801278	-0,066263	-0,00864033	-0,0272308	-0,01879238	-0,05856283
12	0,0076029	-0,02986728	-0,03391102	-0,00444398	-0,01396893	-0,00966509	-0,02981949
No.	1	2	3	4	5	6	7

Tabla 4.15 Cálculo de las derivadas del primer cálculo capa oculta +0.05 versión 2.2.

Para un cambio de cinco por ciento negativos se muestra en las tablas 4.16 y 4.17 para las capas de salida y capa oculta respectivamente.

No.	Derivada de la capa de salida												
1	-0,134126959	0,041683618	0,04525502	-0,05252599	-0,04925827	0,04648698	-0,07429308	0,02353985	-0,03781162	-0,01693152	-0,00651023	-0,0906591	-0,02580314
2	0,123140851	-0,038302301	-0,04158105	0,04821265	0,0452226	-0,04271545	0,06819063	-0,02162792	0,03471362	0,01554899	0,00597925	0,08319996	0,02369551
3	0,117798252	-0,036649295	-0,03977975	0,04613988	0,04327213	-0,04086088	0,06521261	-0,02069203	0,03322367	0,01487737	0,00572078	0,07958952	0,0226685
4	0,119692692	-0,037262372	-0,04043861	0,04689117	0,04398769	-0,04154158	0,06634229	-0,02103684	0,03377258	0,01512525	0,00581595	0,080912	0,02304695
5	0,131000072	-0,040778682	-0,04425918	0,05132417	0,0481498	-0,04545808	0,07260248	-0,02302115	0,03696538	0,01655358	0,00636524	0,08859522	0,02522273
6	0,129165439	-0,040208036	-0,04364059	0,05062429	0,04747228	-0,04483224	0,07157174	-0,02269875	0,03644446	0,01632197	0,00627609	0,08733534	0,02486866
7	0,128575159	-0,039992242	-0,04340079	0,05034386	0,04721149	-0,04458991	0,07119855	-0,02257869	0,03624294	0,01623383	0,00624218	0,08688366	0,02473783
8	0,124258292	-0,038661067	-0,04196003	0,0486545	0,04564593	-0,04309379	0,06881493	-0,0218229	0,03503299	0,01569248	0,00603409	0,08392488	0,02391043
9	0,130255229	-0,040534777	-0,0439997	0,05102558	0,04786193	-0,04518987	0,07216554	-0,02288692	0,03674824	0,01645549	0,00632793	0,08805522	0,02507602
No.	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabla 4.16 Cálculo de las derivadas del primer cálculo capa de salida -0.05 versión 2.2.

No.	Derivadas de la capa oculta						
1	-0,00277748	0,01089906	0,01235426	0,00162228	0,00509861	0,00352574	0,01087094
2	0,01993029	-0,07834497	-0,08893838	-0,01165335	-0,03670996	-0,0253612	-0,0786619
3	0,01392227	-0,05455828	-0,06160668	-0,00812845	-0,0255065	-0,01766179	-0,05433354
4	0,01608964	-0,06269542	-0,07109203	-0,00938628	-0,0294271	-0,02041481	-0,06277923
5	0,00889261	-0,03496728	-0,03964804	-0,0052012	-0,01635589	-0,01131917	-0,03507981
6	-0,00224603	0,00874667	0,0099588	0,00130936	0,00410541	0,00283931	0,0086874
7	-0,02517262	0,09915328	0,11225074	0,01472319	0,0462815	0,03201866	0,09886158
8	-0,00653157	0,02561494	0,02886849	0,0038144	0,01197513	0,00828304	0,02555338
9	0,02076696	-0,0814794	-0,09250856	-0,01213746	-0,0381263	-0,02638836	-0,08133564
10	-0,01030962	0,04044346	0,04588965	0,00602352	0,01892933	0,01309531	0,04035238
11	0,01480401	-0,0578805	-0,06523379	-0,00861082	-0,0269867	-0,01871466	-0,05713257
12	0,007603	-0,02983814	-0,03382563	-0,00444353	-0,01396287	-0,00965774	-0,02978118
No.	1	2	3	4	5	6	7

Tabla 4.17 Cálculo de las derivadas del primer cálculo capa oculta -0.05 versión 2.2.

Si es verdad que nuestra algoritmo de la versión 2.2 trabaja de forma similar a la versión 2.1, es importante destacar que el nivel de variación de las matrices realmente no es muy significativa, tal y como se puede observar en las tablas 4.12 a la 4.17, comparándolas con

las tablas 4.10 y 4.11 según corresponda. Lo que si se debe de considerar es que el cambio no debe ser muy grande debido a que puede causar algún error.

Otro parámetro muy importante que podemos utilizar de comparación es la grafica del error cuadrático promedio que se ha calculado al final de cada versión y para cada prueba, sin embargo para que este sea representativa tenemos que realizar los cálculos más de una iteración o época, por lo tanto hemos considerado solo 5 épocas o iteraciones para realizar este cálculo, debido a que el cálculo de la versión 2.2 es muy lento.

El error promedio cuadrático en base logarítmica que se obtuvo para la versión 2.1 se muestra en la figura 4.31.

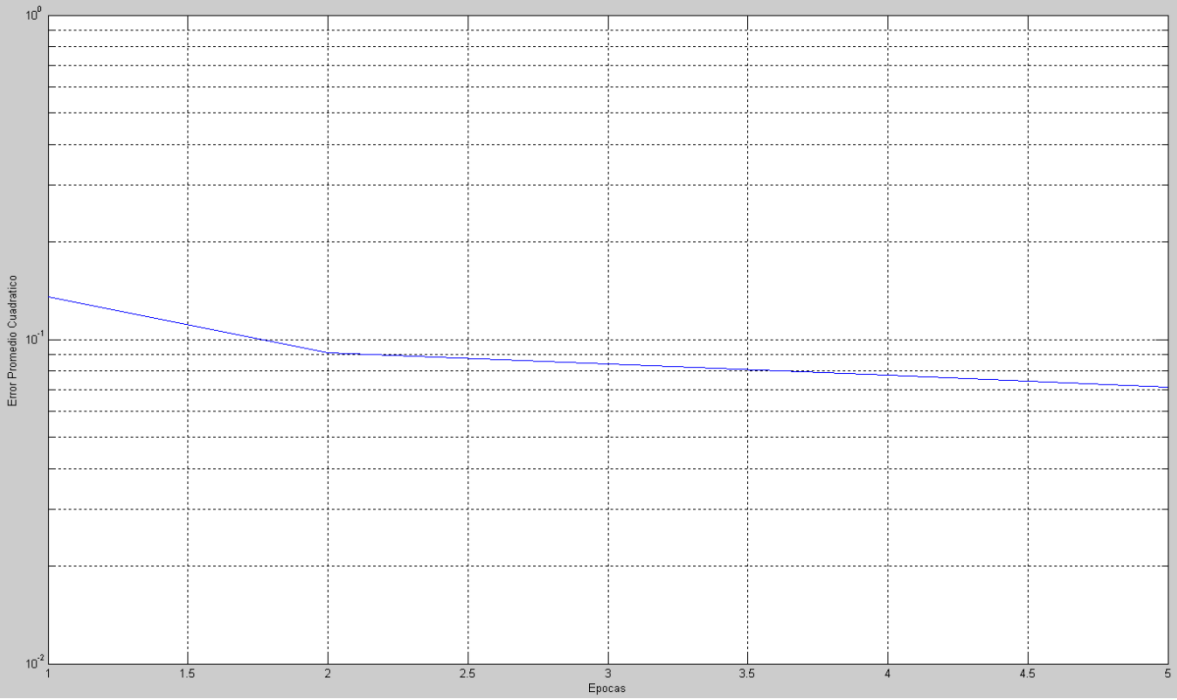


Figura 4.31 Grafica del error medio cuadrático versión 2.1.

Para obtener la grafica del error cuadrático promedio para la versión 2.2 se trabajo con la modificación de un porciento positivo, por lo que la grafica que muestra en la figura 4.32 es para la versión 2.2.

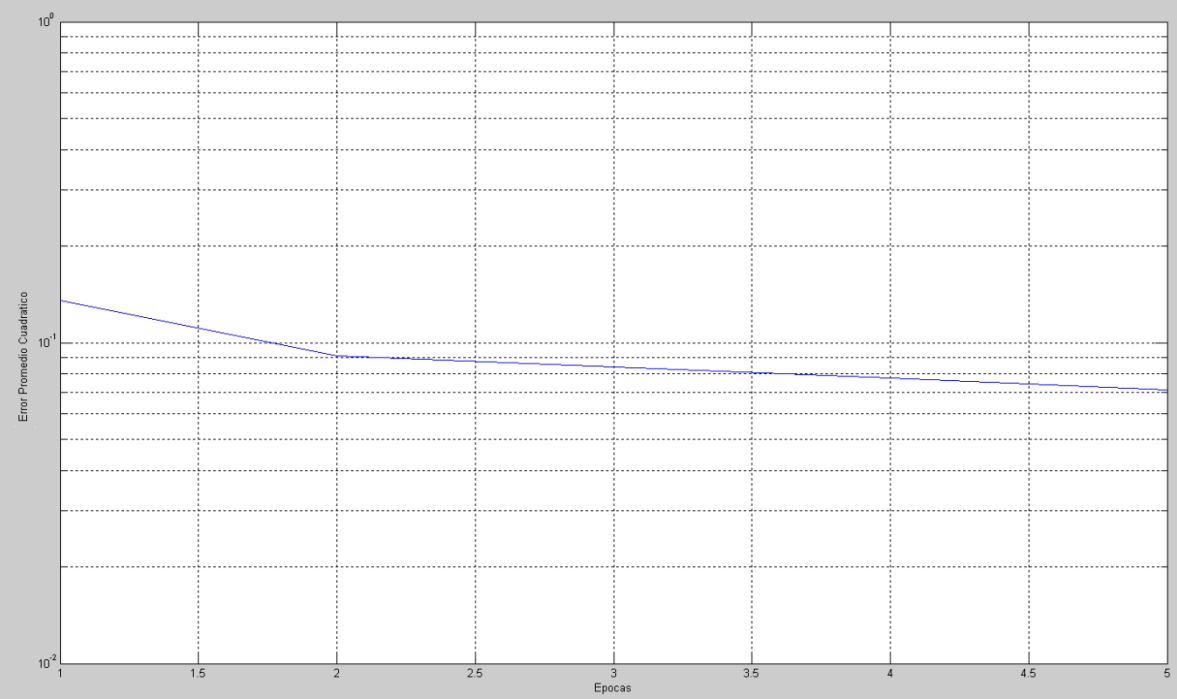


Figura 4.32 Grafica del error medio cuadrático versión 2.2.

Las figuras 4.31 y 4.32, a pesar de ser figuras un poco incompletas por el numero de iteraciones, son muy significativas, debido a que podemos observar que la tendencia del error es prácticamente la misma, se puede casi asegurar que están muy cercanas ambas graficas, lo que hace que aseguremos que realmente ambas versiones trabajan prácticamente de la misma forma, es decir correctamente.

Como se menciono al inicio de esta sección el objetivo de esta tarea es probar que el algoritmo de la versión 2.1 trabaja bien, al tener en cuenta los resultados de comparación de las mismas derivadas, así como las graficas de errores, podemos concluir que el cálculo de derivadas en la versión 2.1 es correcto.

Capítulo 5.-

Algoritmo en el lenguaje de programación Fortran

5.1. Introducción al lenguaje Fortran

Para un lenguaje de programación, Fortran ha existido desde hace mucho tiempo. Fue uno de los primeros lenguajes de alto nivel ampliamente utilizados, tan bien como el primer lenguaje de programación para ser estandarizado. Sigue siendo el primer lenguaje para aplicaciones computacionales para científicos e ingenieros.

El nombre de FORTRAN es una palabra compuesta derivada de *"The IBM Mathematical Formula Translatins System"*, este lenguaje de programación abarca un linaje de versiones para añadir extensiones al lenguaje.

En 1954 un proyecto se inicio bajo la dirección de John Backus en IBM para desarrollar un sistema de programación automática el cual convirtiera programas escritos en una notación matemática para ser instrucciones de máquina para la computadora IBM 704. Este proyecto logro desarrollar el primer compilador Fortran, que fue lanzado al mercado en 1957. La eficiencia de códigos generados por el compilador sorprendió incluso a sus autores.

Un año después de la inclusión del primer compilador Fortran, IBM introdujo Fortran II. Uno de los cambios más importantes en este nuevo software fue la adición de subrutinas que pudieran compilarse independientemente. Por lo tanto Fortran cambio substancialmente incluso durante su primer año. Ha estado cambiando desde entonces.

Hacia 1960, las versiones de Fortran estaban disponibles para las computadoras IBM 709, 650, 1620 y 7090. Significativamente, cada vez mayor la popularidad del Fortran estimuló a fabricantes de computadores de la competencia a proporcionar compiladores Fortran para sus máquinas, así que por 1963 existían más de 40 compiladores Fortran. Por estas razones, el Fortran es considerado el primer lenguaje de programación ampliamente usado soportado a través de una variedad de arquitecturas de computador. Desafortunadamente la profusión de dialectos de Fortran impidió que los programas escritos por un ordenador pudieran ser transportados a un ordenador diferente.

Al mismo tiempo la *"American Estándar Association (ASA)"*, posteriormente conocida como American National Estándar Institute (ANSI), empezó un proyecto de estandarizar muchos aspectos de procesamiento de datos. Algunos tuvieron la atrevida idea de estandarizar los lenguajes de programación. Un comité fue reunido para conformar y desarrollar un estándar para Fortran bajo la protección del Business Equipment Manufacturers Association (BEMA), posteriormente conocida como Computer and Business Equipment Manufacturers Association (CBEMA). Esta norma fue adaptada hasta 1966.[35]

El lenguaje continuo su crecimiento después de 1966, junto con el conocimiento en esta área de programación, hasta que en 1977 fue mejorada la versión que se tenia del Fortran 66, oficialmente publicada en 1978. Las características más significativas introducidas en esta versión fueron los tipos de datos carácter, la construcción de IF-THEN-ELSE y muchas nuevas facilidades de entradas y salidas, como los archivos de acceso directo y la declaración OPEN.

Tan pronto como las técnicas desarrolladas de Fortran 77 fueron completadas, ANSI X3J3 y la International Satandards Organization (ISO) WG5 cambio sus atención a la siguiente versión, la cual es llamada ahora como Fortran 90. El trabajo en Fortran 90 llego muy rápido después de la adopción de fortran 77 debido, a los contrarios pronunciamientos de algunos que Fortran estaba muerto.

Fortran 90 fue un avance mayo sobre Fortran 77. Este incluía: una gran forma de código fuente liberalizado, un conjunto completo de iteraciones y selección de estructuras de control, facilidades numéricas mejoradas, un comprensivo lenguaje de matrices de datos paralelos, estructura de datos, tipos y operadores definidos por el usuario, extensión de procedimientos, encapsulación de módulos, parámetros de tipo kind, objetos dinámicos y algunas extensiones de I/O.

Fortran 95 especificada por WG5 y producida por X3J3 represento una menor revisión para Fortran 90. La mayoría de los cambios fueron corregidos y clarificados según lo que había en fortran 90, sin embargo, pocos cambio significativos, como pura función y la construcción y declaración de FORALL fueron añadidas.

Las características más importantes introducidas en Fortran 2003 fueron: la interoperabilidad con el lenguaje de programación C, soporte para excepciones y aritmética IEEE, soporte para programación orientada a objetos, mejoramiento de tipos de datos, mejoramiento de entradas y salidas, soporte para uso internacional. [36]

El desarrollo del Fortran fue paralelo a la temprana evolución de la tecnología del compilador. De hecho, muchos avances en la teoría y el diseño de compiladores fueron motivados específicamente por la necesidad de generar código eficiente para los programas en Fortran.

Algunas versiones del compilador Fortran [37] son:

- FORTRAN IV
- FORTRAN 77
- FORTRAN 90
- FORTRAN 95
- FORTRAN 2003
- FORTRAN 2008
- FORTRAN 2010

Tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y los errores en la escritura de sólo un carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes.

Se debe tener en cuenta que la sintaxis de Fortran fue afinada para el uso en trabajos numéricos y científicos. Muchas de sus deficiencias han sido abordadas en revisiones recientes del lenguaje. Por ejemplo, Fortran 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos [38]. Esto no sólo mejora mucho la lectura del programa sino que además aporta información útil al compilador.

Por estas razones Fortran no es muy usado fuera de los campos de la informática y el análisis numérico, pero permanece como el lenguaje a escoger para desempeñar tareas de computación numérica de alto rendimiento.

5.2. Programación del algoritmo 2 (versión 2.3)

5.2.1. Descripción

Para el desarrollo del software en Fortran (versión 2.3), seguimos el mismo programa utilizado en Matlab, es decir, el programa al que hemos modificado cambiando el algoritmo del Perceptrón Multicapa con entrenamiento de Retro-propagación, como se ha mencionado la tarea asignada en este apartado es programar el algoritmo cambiando el lenguaje de programación de Matlab a Fortran, por lo cual se empieza de forma diferente la programación de dicho algoritmo. Este cambio de lenguaje nos permitirá crear el algoritmo de diagnóstico en forma de un modulo ejecutable que será independiente de Matlab como lo es Fortran.

En el algoritmo tenemos que generar números aleatorios para utilizarlos en diferentes cálculos. Como ambos lenguajes tienen sus propios comandos y sus propias formas de hacer los cálculos, encontramos que cada lenguaje de programación tiene su forma de generar estos números. Por eso será difícil de saber si trabajan de similar manera. Por lo tanto hemos considerado primero hacer una prueba tomando algunas matrices de Matlab y trabajar con ellas en Fortran, para posteriormente verificar si el resultado es igual y de esta forma determinar el funcionamiento del algoritmo programado en Fortran.

Para empezar la programación en Fortran necesitamos primero determinar las variables utilizadas inclusive el tipo de variables. La figura 5.1, la cual es en la primera parte del programa, determinar las variables necesarias.

```
program RNBP100P

implicit none
real time1
real time2
integer ::krg, nreg,kdf,kdl,kdf0,kdf1,idef2,kdf2,kpr0,ktd,nr1,nr2,Krdd,i_Erlystop
integer :: nn,i,x,y, irg,idf,idl, irdd,itd,idff,n,NE,LE,ME,iter,patrones,ip,je,rc,kpE,rc1,ixt
integer (4) ZZZ1,nT,k
INTEGER, ALLOCATABLE :: imax(:)
double precision DY (1,6),BT(6,2079), BB(2079,6), B(6,21,9,11),D(6,21,9,1),ZE(6,1), ZD_1(6,1)
double precision ZD_2(6,1),ld,ii,alfa,QJ,ZD1(6,1000,9), ZD2(6,1000,9),P1(6,9000),T1(9,9000)
double precision alfaE,Whji(12,7),Wokj(9,13),Z1(7,9000),B1(7,9000),Xi(7,1),Vnj(12,7),Vnjv(12,1)
double precision Z2(13,1),Uok(9,13),Uokv(9,1),de(9,1),E(9,1),delta1(9,13),aE(9,13),asE(1,13)
double precision jes1(1,12),delta2(12,7),EE(9,1),EES,vec_err(1,1000),P1S(6,9000),T1S(9,9000)
double precision B1S(7,9000),A1(9,9000),P2S(6,9000),T2S(9,9000),B2S(7,9000),A2(9,9000),PD1(9,9)
double precision amax,A1S(9,1),PDT1(9,1),psr1,PD1T(9,9),PD2(9,9),A2S(9,1),PDT2(9,1),psr2
double precision PD2T(9,9),t,ZD1r(9000,6),ZD2r(9000,6),ZD1b(6,9000),ZD2b(6,9000),Whjir(7,12)
double precision Wokjr(13,9)
```

Figura 5.1 Inicio del programa y declaración de las variables

Al igual que en Matlab determinamos las constante, como son la cantidad de regímenes, cantidad de clases, cantidad de puntos de interpolación, etc., como se muestra en la figura 5.2.

```
call cpu_time (time1)
! Defects: Gc, EFFc, Gpt, EFFpt, SIGcc, SIGin
! Parameters: Pc, Pt, Tc, Tt, Tpt, Gf
! Regimes n_hp_corr = 10700, 10600, 10500, 10400, 10300, 10200, 10100, 10000, 9900, 9800, 9700
! Defect development 0-5% (idñ = 1-21
krg = 11     ! regime quantity
nreg = 1
kdf = 9      ! class quantity
kdl = 21     ! interpolation points quantity for every class development
kdf0 = 9     ! maximal classes number
kdf1 = 9     ! single classes number
idef2 = 0; kdf2 = 4      ! multiple classes signal and number
kpr0 = 6     ! parameter quantity
ktd=1000     ! class simulationpoints
nr1 = 0; nr2 = 1     ! random generators indication
Krdd = krg*kdf0*kdl
!real DY (nreg,kpr0),BT(kpr0,krdd), BB(krdd,kpr0), B(kpr0,kdl,kdf,krg),D(kpr0,kdl,kdf,nreg)
!real ZE(kpr0,1)
DY = RESHAPE ((/.015, .015, .025, .015, .020, .020/),(/1,6/))
! parameter maximal errors

i_Erlystop = 0 ! Early Stop Option
```

Figura 5.2 Determinación de las constantes del algoritmo

Posteriormente abrimos el archivo que contiene los datos de campo registrados, dichos parámetros se mencionan en la figura 5.2, formamos la matriz BB con dichos datos además de dividirlos entre su error máximo para cada parámetro, como se observa en la figura 5.3

```
open (unit=8,file='DPAR.dat',action='read') !Abriendo el archivo de datos de entrada
read (8,*) BT    ! Leyendo los datos de entrada
close (8)
!write (*,*) BT

BB=transpose(BT)

do nn =1,kpr0
    do i = 1,krdd
    !BB(i,nn)= BT(nn,i)
    BB(i,nn)= BB(i,nn)/DY(1,nn)  ! Dividiendo entre el error maximo permitido
    end do
end do
```

Figura 5.3 Abriendo archivos de entrada y formación de la matriz BB.

Se crea la matriz multidimensional B la cual está determinada por 11 regímenes de operación de la turbina de gas, y cada régimen está conformado por: el número de parámetros recabados en campos que son 6, el número de interpolación que son 21 puntos para parámetro recabado y las clases de fallas que son 9. Tal y como se muestra en la figura 5.4.

Dentro de esta matriz multidimensional B solo trabajamos con un régimen de operación por lo cual seleccionamos solo el primer régimen dentro de esta matriz, de esta forma creamos solo la matriz con la que vamos a trabajar asignándole el nombre D, como se muestra en la figura 5.4.

```
do irg =1,krg
    do idf = 1,kdf0
        do idl = 1,kdl
            do nn =1,kpr0
            irdd= (irg-1)*kdf0*kdl+(idf-1)*kdl+idl
            B(nn,idl,idf,irg)=BB(irdd,nn)
            end do
        end do
    end do
end do
open (unit=9, file='B.sal', action='write');write (9,*) B;close(9)

!Determinacion del Regimen
do irg =1,nreg
    do idf = 1,kdf0
        do idl = 1,kdl
            do nn =1,kpr0
            D(nn,idl,idf,irg)=B(nn,idl,idf,irg)
            end do
        end do
    end do
end do
```

Figura 5.4 Matriz Multidimensional B y determinación del régimen de operación.

Se crean las matrices ZD1 y ZD2 de acuerdo a las figuras 5.8 y 5.9, la matriz ZD1 representa la muestra de prueba, mientras que la matriz ZD2 representa la muestra de validación, sin embargo mientras que en Matlab tenemos la opción de utilizar una función para generar números aleatorios con distribución normal, en Fortran necesitamos trabajar de diferente manera, por lo mismo tenemos que crear nuestra propia función que genere números aleatorios con distribución normal, para esto simplemente generamos la subrutina como se observa en la figura 5.5.

```
!!!!!!!!!!!!!!!!!GENERACION DE NUMEROS ALEATORIOS!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!DISTRIBUCION NORMAL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE GAUSS (ERR)
DOUBLE PRECISION ERR
PARAMETER (ONE=1.0,TWO=2.0)
SIG=1.0
AMO=0
10 CALL RANDOM_NUMBER (ZE1)
   CALL RANDOM_NUMBER (ZE2)
V1=TWO*ZE1-ONE
V2=TWO*ZE2-ONE
S=V1*V1 +V2*V2
IF(S.GT.ONE) GO TO 10
ERR=V1*SQRT(-TWO*ALOG(S)/S)*SIG+AMO
RETURN
END SUBROUTINE GAUSS
```

Figura 5.5 Subrutina que genera números aleatorios con distribución normal.

```
!LEARNING SAMPLES
k=1

do irg = 1,nreg
    do idff = 1,kdf
        do itd =1,ktd
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            do py = 1,6
                call gauss (ERR)
                ZE(py,1)=ERR
            end do
            ZE=ZE/3
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            call random_number(ld)!call random (ld);
            ld = ld*(kdl-1)!print *,'ld ', ld
            ii=floor(ld)+1
            alfa=ld-ii+1;!print *,'ii ', ii;print *,'alfa ', alfa
            do nn =1,kpr0
                ZD1(nn,itd,idff)=D(nn,ii,idff,irg)*(1-alfa)+D(nn,ii+1,idff,irg)*alfa+ZE(nn,1)
            end do
        end do
    end do
end do
open (unit=9, file='ZD1.sal', action='write');write (9,*) ZD1;close(9)
```

Figura 5.6 Generación de la matriz ZD1, muestra de prueba.

```
!TEST SAMPLES POINTS

do irg = 1,nreg
    do idff = 1,kdf
        do itd =1,ktd
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            do py = 1,6
                call gauss (ERR)
                ZE(py,1)=ERR
            end do
            ZE=ZE/3
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            call random_number(ld);
            ld = ld*(kdl-1);!print *,'ld ', ld
            ii=floor(ld)+1;alfa=ld-ii+1;!print *,'ii ', ii;print *,'alfa ', alfa
            do nn =1,kpr0
                ZD2(nn,itd,idff)=D(nn,ii,idff,irg)*(1-alfa)+D(nn,ii+1,idff,irg)*alfa+ZE(nn,1)
            end do
        end do
    end do
end do
open (unit=9, file='ZD2.sal', action='write');write (9,*) ZD2;close(9)
```

Figura 5.7 Generación de la matriz ZD2, muestra de validación.

A partir de la matriz ZD1, que es la muestra de prueba o entrenamiento, pasamos a formar la matriz P1, la cual representa la matriz de entrada a la red, así mismo conformamos la matriz T1 que representa las clases de fallas y nuestra salida deseada de la red. Ver figura 5.8.

```
do itd =1,ktd
    do  idff = 1,kdf
        n=(itd-1)*kdf+idff
        P1(:,n)=ZD1(:,itd,idff)
        T1(idff,n)=1
    end do
end do


open (unit=9, file='P1.sal', action='write');write (9,*) P1;close(9)
open (unit=9, file='T1.sal', action='write');write (9,*) T1;close(9)
```

Figura 5.8 Formación de las matrices P1 (entrada de la red) y T1 (target).

Determinamos las características de la red, es decir, numero de neuronas de entrada, numero de neuronas de la capa oculta y numero de neuronas de la capa de salida, así como el número de iteración que se entrenara a la red y los parámetros que se tendrá que aprender la red. Por otro lado determinamos los valores aleatorios de las matrices de la red, según la figura 5.9.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!RED BACKPROPAGATION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!VALORES DE ENTRADA

NE = 6  ! NUMERO DE NEUROPNAS DE ENTRADA
LE = 12 ! NUMERO DE NEURONAS OCULTAS
ME = 9  ! NUMERO DE NEURONAS DE SALIDA
alfaE = 0.0025  ! RELACION DE MODIFICACION DE PESOS
iter = 100  ! NUMERO DE ITERACIONES O EPOCAS
patrones = 9000 ! NUMERO DE PATRONES

! MATRICES DE PESOS W
call random_number(Whji);   call random_number(Wokj);
Whji = (Whji-1)*.2; Wokj = (Wokj -1)*0.2
open (unit=9, file='Whji.sal', action='write');write (9,*) Whji;close(9)
open (unit=9, file='Wokj.sal', action='write');write (9,*) Wokj;close(9)
```

Figura 5.9 Parámetros de la red.

Preparamos los datos de entrada como se observa en la figura 5.10, agregando el bias (1) que es una factor de corrección de umbrales que necesita le red. De tal forma que formamos la matriz B1, de este modo ya tendremos listos los datos para poderlos ingresar a la red neuronal.

```
!AGREGANDO EL BIAS A LA ENTRADA
Z1=(1)!(SIZE(Z1,1))-6
B1=Z1
!open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)

do ip = 1,patrones
    do nn = 1,NE
    B1(nn+1,ip)= P1(nn,ip)
    end do
end do
open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)
```

Figura 5.10 Formación de la matriz B1 (agregando el bias).

Con los datos ya listos para poder alimentar a la red, tendremos que pasar vector por vector hasta a completar los 9000 patrones que deseamos que se aprenda la red, como ya se ha mencionado en el análisis de la red neuronal con entrenamiento de retro-propagación, cada vector de entrada consta de su vector de salida deseado (target), por lo tanto en cada patrón obtendremos un error como se observa en la figura 5.11.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do kpE =1,iter

    EES=0

    do ip = 1,patrones
        Xi(:,1)=B1(:,ip)
        do je = 1,LE        !pasando el vector de entrada por la capa oculta
            Vnj(je,:)= Whji(je,:)*Xi(:,1)
        end do

        do je = 1,LE        !Vector de salida de la capa oculta
            Vnjv(je,1)= sum (Vnj(je,:))
        end do

        Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1 ! Pasando por la funcion de activacion
        Z2=(1)    !agregando el vias

        do je = 1,LE
            Z2(je+1,1)= Vnjv(je,1)
        end do

        do rc = 1,ME
            Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
        end do

        do rc = 1,ME        !Vector de salida de la capa oculta
            Uokv(rc,1)= sum (Uok(rc,:))
        end do

        Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1) ! Pasando por la funcion de activacion
        de(:,1)=T1(:,ip) !Estableciennmdo el error desceado
        E(:,1)= de(:,1)-Uokv(:,1) !Error de salida de la red
```

Figura 5.11 Entrada de la red y obtención del error.

Después de introducir los datos de entrada a la red, de pasar por la capa oculta y de la capa de salida, como se ha mencionado con cada patrón obtendremos un error, el cual tendremos que retro-propagar, con el objetivo de corregir los pesos de las matrices de la capa de oculta y la capa de salida para que de esta manera adapten y al final la red pueda reconocer satisfactoria mente los patrones aprendidos. Como se observa en la figura 5.12.

```
            do je = 1,LE+1
                  delta1 (:,je)= (-1)*(E(:,1)*((1-Uokv(:,1))*Uokv(:,1)))*(Z2(je,1))
            end do
            open (unit=9, file='delta1.sal', action='write');write (9,*) delta1;close(9)

            do je = 1,LE+1
                delta1 (:,je)= (alfaE)*delta1(:,je)
            end do

            Wokj = Wokj-delta1      !Se corrigen los pesos de la matriz de la capa de salida

            do je = 1,LE+1
                aE (:,je)= (E(:,1)*((1-Uokv(:,1))*Uokv(:,1)))*Wokj(:,je)
            end do

            do je = 1,LE+1
                asE(1,je)=sum(aE(:,je))
            end do

            do je = 1,LE+1
                asE(1,je)=(-1)*(asE(1,je))*((1+(Z2(je,1)))*(1-(Z2(je,1))))
            end do

            do je = 1,LE
                jes1(1,je) =asE(1,je+1)
            end do

            do je = 1,LE
                delta2 (je,:)= jes1(1,je)* Xi(:,1)
            end do
            open (unit=9, file='delta2.sal', action='write');write (9,*) delta2;close(9)

            do je = 1,LE
                delta2 (je,:)= alfaE* delta2(je,:)
            end do

            Whji = Whji - delta2
            EE(:,1)= (E(:,1))**2 !Calculo del Error cuadratico promedio
            EES = EES+(sum(EE))/ME

        end do
        EES = EES/patrones
        vec_err(1,kpE) = EES
        open (unit=9, file='vec_err.sal', action='write');write (9,*) vec_err;close(9)

    end do
```

Figura 5.12 Retro-propagación del error y corrección de los pesos de las Matrices.

Una vez que hemos entrenado la red lo suficiente, y hemos considerado que el error medio cuadrático es aceptable, tendremos que pasa a la etapa de simulación de la red con el objetivo de corroborar que tanto es lo que la red se aprendió, sin embargo esta etapa consta de dos parte una es simular los patrones aprendidos (datos de entrenamiento) ver figura 5.13 y otra es simular patrones diferentes (datos de validación) ver la figura 5.14.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!SIMULACION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!CONJUNTO DE ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!!!
do idff = 1,kdf
    do itd =1,ktd
        n=(idff-1)*ktd+itd
        P1S(:,n)=ZD1(:,itd,idff)
        T1S(idff,n)=1
    end do
end do
open (unit=9, file='P1S.sal', action='write');write (9,*) P1S;close(9)
open (unit=9, file='T1S.sal', action='write');write (9,*) T1S;close(9)
    !AGREGANDO EL BIAS A LA ENTRADA
Z1=(1) !(SIZE(Z1,1))-6
B1S=Z1

do ip = 1,patrones
    do nn = 1,NE
        B1S(nn+1,ip)= P1S(nn,ip)
    end do
end do
open (unit=9, file='B1S.sal', action='write');write (9,*) B1S;close(9)
B1=B1S
do ip = 1,patrones
    Xi(:,1)=B1(:,ip) !estableciendo el vector de entrada

    do je = 1,LE        !pasando el vector de entrada por la capa oculta
        Vnj(je,:)= Whji(je,:)*Xi(:,1)
    end do

    do je = 1,LE        !Vector de salida de la capa oculta
        Vnjv(je,1)= sum (Vnj(je,:))
    end do

    Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1  ! Pasando por la funcion de activacion
    Z2=(1)!(SIZE(Z2))-12    !agregando el vias

    do je = 1,LE
        Z2(je+1,1)= Vnjv(je,1)
    end do

    do rc = 1,ME
        Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
    end do
      do rc = 1,ME        !Vector de salida de la capa oculta
          Uokv(rc,1)= sum (Uok(rc,:))
      end do

      Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1) ! Pasando por la funcion de activacion
      A1(:,ip)=Uokv(:,1)
  end do
  open (unit=9, file='A1.sal', action='write');write (9,*) A1;close(9)
```

Figura 5.13 Simulación de los datos de entrenamiento.

En la salida de la red dentro de la fase de simulación creamos las matrices A1 y A2 correspondientes a los datos de entrenamiento y a los datos de validación respectivamente, debido a que estas dos matrices contienen los valores de reconocimiento de la red.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!CONJUNTO DE ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!!
do idff = 1,kdf
    do itd =1,ktd
        !do nn =1,kpr0
        n=(idff-1)*ktd+itd
        P2S(:,n)=ZD2(:,itd,idff)
        T2S(idff,n)=1
        !end do
    end do
end do
open (unit=9, file='P2S.sal', action='write');write (9,*) P2S;close(9)
open (unit=9, file='T2S.sal', action='write');write (9,*) T2S;close(9)
!AGREGANDO EL BIAS A LA ENTRADA
Z1=(1) !(SIZE(Z1,1))-6
B2S=Z1

do ip = 1,patrones
    do nn = 1,NE
    B2S(nn+1,ip)= P2S(nn,ip)
    end do
end do
open (unit=9, file='B1S.sal', action='write');write (9,*) B1S;close(9)
B1=B2S
do ip = 1,patrones
    Xi(:,1)=B1(:,ip) !estableciendo el vector de entrada

    do je = 1,LE        !pasando el vector de entrada por la capa oculta
        Vnj(je,:)= Whji(je,:)*Xi(:,1)
    end do

    do je = 1,LE        !Vector de salida de la capa oculta
        Vnjv(je,1)= sum (Vnj(je,:))
    end do

    Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1  ! Pasando por la funcion de activacion
    Z2=(1)!(SIZE(Z2))-12    !agregando el vias

    do je = 1,LE
        Z2(je+1,1)= Vnjv(je,1)
    end do

    do rc = 1,ME
        Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
    end do

    do rc = 1,ME        !Vector de salida de la capa oculta
        Uokv(rc,1)= sum (Uok(rc,:))
    end do

    Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1) ! Pasando por la funcion de activacion
    A2(:,ip)=Uokv(:,1)
end do
open (unit=9, file='A2.sal', action='write');write (9,*) A2;close(9)
```

Figura 5.14 Simulación de los datos de validación.

Para corroborar el funcionamiento de la red neuronal, así como el poder analizar que tan bien se entreno y cuanto es lo que reconoció la red neuronal calculamos las matrices de probabilidades del diagnóstico tanto para la muestra de entrenamiento, como para la muestra de validación, es decir, las matrices PD1 y PD2 respectivamente. Sacamos las probabilidades del diagnóstico correcto al extraes la diagonal de las matrices, cada elemento nos indicara la probabilidad con la que la red reconoce cada clase, tal y como se puede observar en la figura 5.15 y 5.16.

```
PD1=(0)!SIZE(PD1))-81
do rc = 1,ME
    do itd =1,ktd
        nT = (rc-1)*ktd+itd
        do rc1 = 1,ME
            A1S(rc1,1)= A1(rc1,nT)
        end do
        amax = max(A1(1,nT),A1(2,nT),A1(3,nT),A1(4,nT),A1(5,nT),A1(6,nT),A1(7,nT),A1(8,nT),A1(9,nT))
        ixt = SIZE(SHAPE(A1S))
        ALLOCATE ( imax(ixt))
        imax = MAXLOC (A1S)
        PD1(imax(1),rc)=PD1((imax(1)),rc)+1
        deallocate (imax)
    end do
end do

do rc = 1,ME
    do rc1 = 1,ME
        PD1(rc1,rc) = PD1(rc1,rc)/ktd
    end do
end do
open (unit=9, file='PD1.sal', action='write');write (9,*) PD1;close(9)

do rc = 1,ME
    PDT1(rc,1)= PD1(rc,rc)
end do
open (unit=9, file='PDT1.sal', action='write')
write(9,2000) (PDT1(rc1,1), rc1 = 1,ME)
2000 format (9F8.4, 10X)
close(9)

psr1=(sum(PDT1))/ME
PD1T=transpose(PD1)

open (unit=9, file='PD1.sal', action='write')
do 10 rc = 1,ME
    write(9,1000) (PD1(rc,rc1), rc1 = 1,ME)
10 continue
1000 format (9F8.4, 10X)
close (9)

open (unit=9, file='psr1.sal', action='write');write (9,*) psr1;close(9)
```

Figura 5.15 Obtención de la matriz de probabilidades del diagnóstico correcto PD1 (datos de entrenamiento).

Si bien ya se ha mencionado que las matrices de probabilidades nos ayudaran a determinar el rendimiento de la red, es importante mencionar, que ambas matrices deben estar muy cerca una de la otra, es decir que bien entrenada en una porción de datos, la red tiene que reconocer de la misma forma los datos nuevos.

```
PD2=(0)!SIZE(PD1))-81
do rc = 1,ME
    do itd =1,ktd
        nT = (rc-1)*ktd+itd
        do rc1 = 1,ME
            A2S(rc1,1)= A2(rc1,nT)
        end do
        amax = max(A2(1,nT),A2(2,nT),A2(3,nT),A2(4,nT),A2(5,nT),A2(6,nT),A2(7,nT),A2(8,nT),A2(9,nT))
        ixt = SIZE(SHAPE(A2S))
        ALLOCATE ( imax(ixt))
        imax = MAXLOC (A2S)
        PD2(imax(1),rc)=PD2((imax(1)),rc)+1
        deallocate (imax)
    end do
end do

do rc = 1,ME
    do rc1 = 1,ME
        PD2(rc1,rc) = PD2(rc1,rc)/ktd
    end do
end do
open (unit=9, file='PD2.sal', action='write');write (9,*) PD2;close(9)

do rc = 1,ME
    PDT2(rc,1)= PD2(rc,rc)
end do
open (unit=9, file='PDT2.sal', action='write')
write(9,2000) (PDT2(rc1,1), rc1 = 1,ME)
close(9)

psr2=(sum(PDT2))/ME
PD2T=transpose(PD2)

open (unit=9, file='PD2.sal', action='write')
do 20 rc = 1,ME
    write(9,1000) (PD2(rc,rc1), rc1 = 1,ME)
20 continue
close (9)
open (unit=9, file='psr2.sal', action='write');write (9,*) psr2;close(9)
call cpu_time (time2)
t = time2-time1
write (*,*) 'tiempo= ',t
open (unit=9, file='tiempo.sal', action='write');write (9,*) t;close(9)

stop
end program jes
```

Figura 5.16 Obtención de la matriz de probabilidades del diagnóstico correcto PD1 (datos de entrenamiento).

5.2.2 Pruebas 1 con números aleatorios generados en Matlab

Una vez de terminar de programar el algoritmo, es necesario poder corroborar que el programa trabaje de buena manera, por lo que notamos que solo hay dos punto en donde el algoritmo programado en Maltab y el de Fortran varían, no por ser un error, al contrario por la naturaleza de cada uno de los lenguajes de programación, el punto de cambio se debe a la generación de números aleatorios, tanto para distribución normal como para distribución uniforme.

Nuestra primer prueba fue extraer los valores de la matriz ZD1 y ZD2 de la versión 1 del algoritmo. Creamos desde Matlab los archivos llamados ZD1r.tx y ZD2r.tx, para poder utilizar estas matrices en nuestro algoritmo programado en Fortran y de esta forma poder trabajar con ellos, tal y como se observa en la figura 5.17. De esta manera lo que

pretendemos es llegar al mismo resultado para verificar que ambos algoritmos trabajan igual.

```
open (unit=8,file='ZD1r.tx',action='read');read (8,*) ZD1r;close (8)
open (unit=8,file='ZD2r.tx',action='read');read (8,*) ZD2r;close (8)

ZD1b=transpose(ZD1r)
ZD2b=transpose(ZD2r)

do idff = 1,kdf
    do itd =1,ktd
        irdd = (idff-1)*ktd + itd
        ZD1(:,itd,idff)=ZD1b(:,irdd)
    end do
end do


do idff = 1,kdf
    do itd =1,ktd
        irdd = (idff-1)*ktd + itd
        ZD2(:,itd,idff)=ZD2b(:,irdd)
    end do
end do
open (unit=9, file='ZD1.sal', action='write');write (9,*) ZD1;close(9)
open (unit=9, file='ZD2.sal', action='write');write (9,*) ZD2;close(9)
```

Figura 5.17 Modificación al algoritmo para comparar resultados con Matlab.

Por otro lado tenemos que generamos las matrices de pesos correspondientes a la capa oculta y a la capa de salida por medio de generación de números aleatorios de distribución normal. Al igual que procedimos anteriormente generamos desde Matlab los archivos Wh_ji.tx y Wo_kj.tx, con los que inicializamos estas matrices correspondiente en Fortran, como se puede observar en la figura 5.18.

```
open (unit=8,file='Wh_ji.tx',action='read');read (8,*) Whjir;close (8)
open (unit=8,file='Wo_kj.tx',action='read');read (8,*) Wokjr;close (8)

Whji=transpose(Whjir)
Wokj=transpose(Wokjr)
```

Figura 5.18 Matrices de pesos extraídas de Matlab.

Con esta prueba lo único que se pretende es corroborar el funcionamiento del algoritmo programado en el lenguaje Fortran, por lo cual es meramente ilustrativo. Considerando que ambos algoritmos se trabajan con los mismos parámetros de prueba, en este caso son 100 épocas o iteraciones de cálculo. Como es de pensarse el rendimiento que se ha obtenido en esta prueba con el lenguaje Fortran es similar al de Matlab, tal y como lo ilustran las figuras 5.19 y 5.20. Si comparamos las matrices de fortran figuras 5.19 y 5.20 con las figuras 5.21 y 5.22 que son de Matlab, las cuales representan las matrices de probabilidades de los datos de entrenamiento y de validación respectivamente, podemos decir que nuestro algoritmo programado en ambos lenguajes trabajan de una manera similar, ahora lo que falta por revisar es el funcionamiento de Fortran generando sus propios números aleatorio con ambas distribuciones, normal y uniforme.

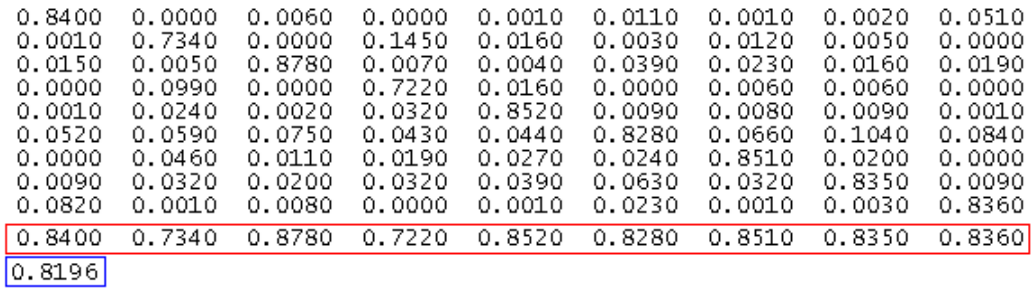


Figura 5.19 Matriz de probabilidades de diagnóstico PD1 generadas en Fortran.

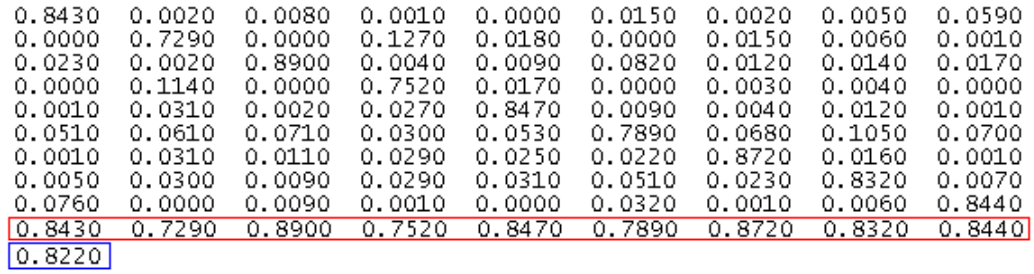


Figura 5.20 Matriz de probabilidades de diagnóstico PD2 generadas en Fortran.

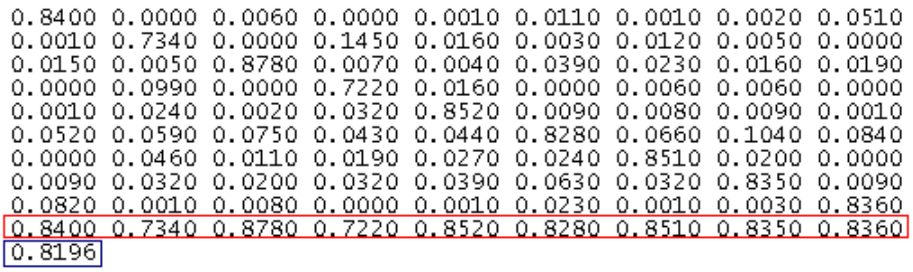


Figura 5.21 Matriz de probabilidades de diagnóstico PD1 generadas en Matlab.

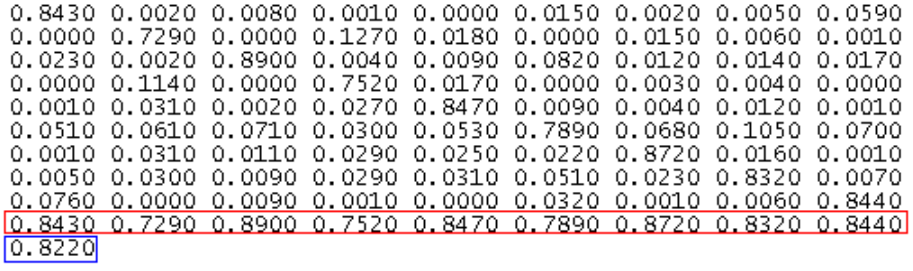


Figura 5.22 Matriz de probabilidades de diagnóstico PD2 generadas en Matlab.

4.2.2. Pruebas 2 con números aleatorios generados en Fortran

Después de realizar la prueba anterior quedamos convencidos que el algoritmo realizado en el lenguaje Fortran trabaja satisfactoriamente. Sin embargo, debemos dejar que el propio programa realice todas las operaciones por sí mismo. Por lo tanto tenemos que dejar que genere sus propios numero aleatorios con sus propias funciones como es el caso de la

generación de números aleatorios de distribución uniforme, así como creando nuestra propia función para generar números aleatorios con distribución normal. Como ya se ha explicado anteriormente.

Al permitir que el lenguaje de programación genere sus propios numero aleatorios, tendremos algunas dudas sobre el funcionamiento del mismo, sin embargo, el caso más crítico es el de generar números aleatorios con distribución normal, debido a que estos los generamos desde una subrutina creada por nosotros. Por lo que debemos revisar y determinar si esta distribución es correcta. Para poder realizar esta comprobación tenemos que obtener las matrices que se calculan con la generación de números aleatorios ZD1 y ZD2 que son los datos de entrenamiento y de validación respectivamente.

Para corroborar si estas matrices son relativamente correctas, debemos graficar algunas clases de estas matrices, tal y como se muestra en la figura 5.23, o podemos ver que la distribución de patrones alrededor de líneas de desarrollo de cada falla parece a la normal ya que es más densa en partes centrales y la periferia está dispersa. Si por otro lado la comparamos con la grafica de dispersión de la figura 5.24 generada por los cálculos de Matlab, podemos concluir que el algoritmo programado en Fortran genera patrones de modo muy similar, eso quiere decir que funciona correctamente.

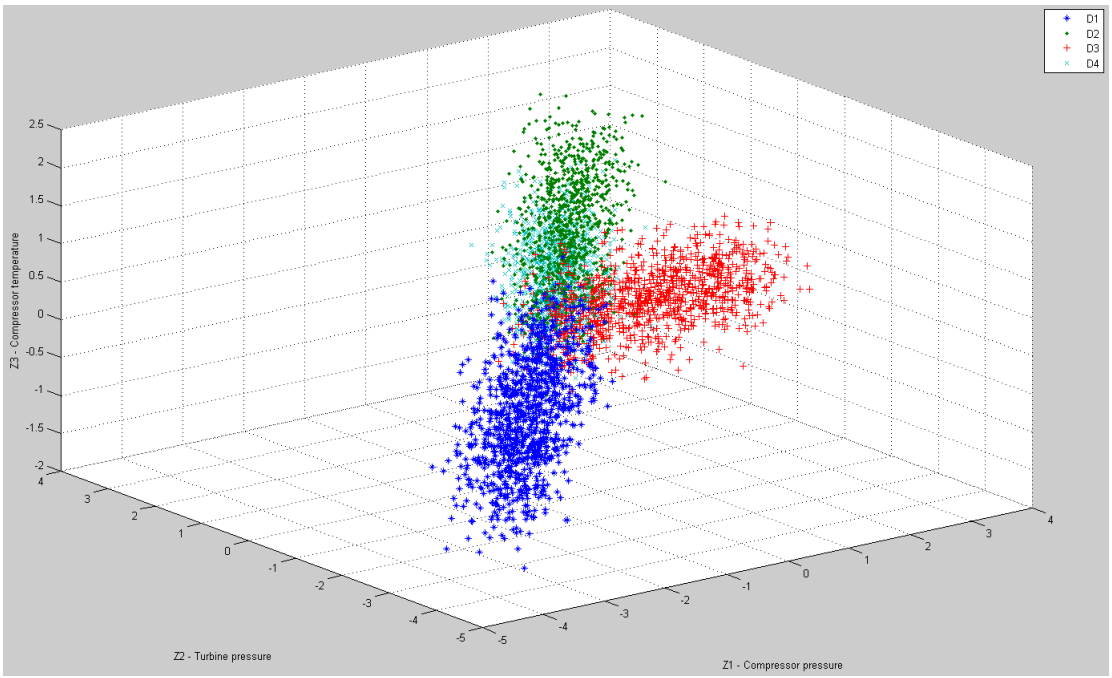


Figura 5.23 Dispersión de las clases generadas por Fortran.

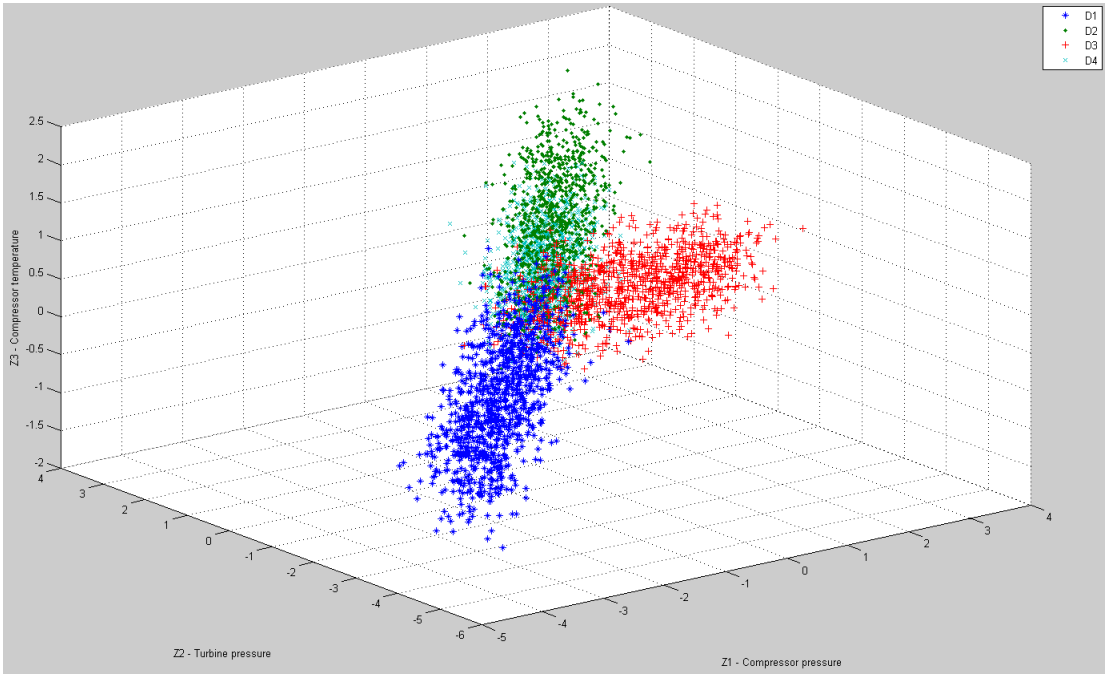


Figura 5.24 Dispersión de las clases generadas por Matlab.

Ahora vamos a comparar los resultados finales, las matrices de probabilidades, con este fin. Una de ellas es entrenando la red con 100 épocas de entrenamiento, de la cual obtuvimos las matrices de probabilidades del diagnóstico correcto, para los datos de entrenamiento, tal y como se ilustra la figura 5.25 y para los datos de validación según la figura 5.26.

0.8300	0.0010	0.0040	0.0000	0.0000	0.0070	0.0010	0.0010	0.0470
0.0000	0.7190	0.0010	0.1020	0.0080	0.0050	0.0150	0.0070	0.0000
0.0130	0.0040	0.8790	0.0020	0.0040	0.0400	0.0200	0.0130	0.0170
0.0000	0.1290	0.0000	0.7560	0.0110	0.0000	0.0060	0.0020	0.0000
0.0010	0.0270	0.0020	0.0260	0.8870	0.0160	0.0130	0.0120	0.0020
0.0620	0.0510	0.0820	0.0410	0.0470	0.8270	0.0950	0.1110	0.0760
0.0010	0.0370	0.0090	0.0390	0.0160	0.0250	0.8230	0.0230	0.0010
0.0050	0.0320	0.0190	0.0330	0.0250	0.0530	0.0270	0.8230	0.0100
0.0880	0.0000	0.0040	0.0010	0.0020	0.0270	0.0000	0.0080	0.8470
0.8300	0.7190	0.8790	0.7560	0.8870	0.8270	0.8230	0.8230	0.8470
0.8212								

Figura 5.25 Matriz de probabilidades de diagnóstico PD1 generadas en Fortran para 100 épocas.

0.8090	0.0010	0.0100	0.0000	0.0000	0.0090	0.0000	0.0000	0.0500
0.0010	0.6770	0.0000	0.1010	0.0140	0.0050	0.0120	0.0090	0.0010
0.0130	0.0050	0.8630	0.0030	0.0080	0.0350	0.0210	0.0120	0.0110
0.0000	0.1490	0.0000	0.7640	0.0130	0.0000	0.0030	0.0030	0.0000
0.0010	0.0260	0.0020	0.0340	0.8590	0.0110	0.0130	0.0160	0.0020
0.0510	0.0610	0.0910	0.0490	0.0420	0.8350	0.0880	0.1180	0.0920
0.0010	0.0400	0.0130	0.0190	0.0260	0.0230	0.8310	0.0170	0.0020
0.0070	0.0390	0.0170	0.0280	0.0360	0.0470	0.0310	0.8190	0.0170
0.1170	0.0020	0.0040	0.0020	0.0020	0.0350	0.0010	0.0060	0.8250
0.8090	0.6770	0.8630	0.7640	0.8590	0.8350	0.8310	0.8190	0.8250
0.8091								

Figura 5.26 Matriz de probabilidades de diagnóstico PD2 generadas en Fortran para 100 épocas.

En la segunda tarea entrenamos la red con 1000 épocas, obtuvimos las matrices de probabilidades del diagnóstico correcto según las figuras 5.27 y 5.28, para el caso de los datos de entrenamiento y para el caso de los datos de validación respectivamente como se observa. Al igual que en Matlab, en Fortran el entrenar la red neuronal con un mayor número de épocas mejora incondicionalmente las matrices de probabilidad, sin embargo no es un cambio que se intensifique, es decir, que no por tener un numero de épocas muy grande el cambio será significativo considerando que el tiempo también juega un papel muy importante, por esta razón no hemos considerado el entrenar la red neuronal con un numero de épocas mayor.

0.8430	0.0010	0.0050	0.0010	0.0000	0.0130	0.0010	0.0010	0.0510
0.0010	0.7430	0.0020	0.1140	0.0130	0.0080	0.0190	0.0070	0.0000
0.0100	0.0040	0.8750	0.0010	0.0030	0.0350	0.0170	0.0140	0.0130
0.0000	0.1190	0.0000	0.7510	0.0110	0.0000	0.0070	0.0030	0.0000
0.0000	0.0280	0.0020	0.0320	0.8860	0.0070	0.0150	0.0100	0.0010
0.0550	0.0520	0.0850	0.0410	0.0480	0.8330	0.0900	0.1070	0.0770
0.0020	0.0270	0.0100	0.0320	0.0130	0.0220	0.8210	0.0270	0.0010
0.0060	0.0250	0.0160	0.0280	0.0240	0.0560	0.0290	0.8210	0.0080
0.0830	0.0010	0.0050	0.0000	0.0020	0.0260	0.0010	0.0100	0.8490
0.8430	0.7430	0.8750	0.7510	0.8860	0.8330	0.8210	0.8210	0.8490
0.8247								

Figura 5.27 Matriz de probabilidades de diagnóstico PD1 generadas en Fortran para 1000 épocas.

0.8260	0.0020	0.0120	0.0010	0.0000	0.0210	0.0000	0.0000	0.0620
0.0000	0.7140	0.0000	0.1200	0.0160	0.0080	0.0150	0.0110	0.0000
0.0080	0.0040	0.8580	0.0030	0.0040	0.0420	0.0160	0.0100	0.0090
0.0000	0.1250	0.0000	0.7560	0.0130	0.0020	0.0070	0.0030	0.0000
0.0010	0.0250	0.0010	0.0280	0.8600	0.0080	0.0100	0.0170	0.0010
0.0450	0.0600	0.0910	0.0470	0.0480	0.8290	0.0900	0.1190	0.0890
0.0010	0.0300	0.0150	0.0180	0.0200	0.0190	0.8320	0.0160	0.0020
0.0060	0.0380	0.0180	0.0250	0.0370	0.0430	0.0290	0.8180	0.0170
0.1130	0.0020	0.0050	0.0020	0.0020	0.0280	0.0010	0.0060	0.8200
0.8430	0.7430	0.8750	0.7510	0.8860	0.8330	0.8210	0.8210	0.8490
0.8126								

Figura 5.28 Matriz de probabilidades de diagnóstico PD2 generadas en Fortran para 1000 épocas.

Si comparamos las matrices de probabilidades obtenidas en Fortran con las obtenidas en Matlab, nos podremos dar cuenta que la red neuronal programada en Fortran con mayor numero de épocas de entrenamiento aparentemente entrega probabilidades más adecuadas, considerando solamente que las probabilidades de entrenamiento deben estar muy cercanas a las probabilidades de validación.

Ahora bien si comparamos ambos lenguajes de programación, veremos que los resultados obtenidos para 100 épocas de entrenamiento, Fortran reconoce un poco mejor si solo nos basamos solo en los datos de entrenamiento de la red, observar y comparar, la figura 5.25 y la figura 5.29, sin embargo por otro lado si comparamos las figura 5.26 y 5.30 observaremos que Matlab puede reconocer mejor parámetros con los que no ha sido entrenada, es decir parámetros de validación.

Para 1000 épocas de entrenamiento, podemos decir que el comportamiento de Fortran es muy parecida al de MAtlab, debido a que si comparamos las matrices de probabilidades, tanto para los datos de entrenamiento ver figura 5.27 y 5.31 los resultados son muy parecidos, al igual que para los datos de validación ver figura 5.28 y 4.32 los resultados son muy cercanos, además ahora si comparamos las matrices de probabilidades de los datos de entrenamiento con los datos de validación observaremos que estas probabilidades son muy cercanas y no hay una variación muy grande, lo que nos da un parámetro para darnos cuenta que el algoritmo en Fortran trabaja de buena manera. Hay que comparar también los tiempos de ejecución para ambas tareas

```
0.8400 0.0000 0.0060 0.0000 0.0010 0.0110 0.0010 0.0020 0.0510
0.0010 0.7340 0.0000 0.1450 0.0160 0.0030 0.0120 0.0050 0.0000
0.0150 0.0050 0.8780 0.0070 0.0040 0.0390 0.0230 0.0160 0.0190
0.0000 0.0990 0.0000 0.7220 0.0160 0.0000 0.0060 0.0060 0.0000
0.0010 0.0240 0.0020 0.0320 0.8520 0.0090 0.0080 0.0090 0.0010
0.0520 0.0590 0.0750 0.0430 0.0440 0.8280 0.0660 0.1040 0.0840
0.0000 0.0460 0.0110 0.0190 0.0270 0.0240 0.8510 0.0200 0.0000
0.0090 0.0320 0.0200 0.0320 0.0390 0.0630 0.0320 0.8350 0.0090
0.0820 0.0010 0.0080 0.0000 0.0010 0.0230 0.0010 0.0030 0.8360
0.8400 0.7340 0.8780 0.7220 0.8520 0.8280 0.8510 0.8350 0.8360
0.8196
```

Figura 5.29 Matriz de probabilidades de diagnóstico PD1 generadas en Matlab para 100 épocas.

```
0.8430 0.0020 0.0080 0.0010 0.0000 0.0150 0.0020 0.0050 0.0590
0.0000 0.7290 0.0000 0.1270 0.0180 0.0000 0.0150 0.0060 0.0010
0.0230 0.0020 0.8900 0.0040 0.0090 0.0820 0.0120 0.0140 0.0170
0.0000 0.1140 0.0000 0.7520 0.0170 0.0000 0.0030 0.0040 0.0000
0.0010 0.0310 0.0020 0.0270 0.8470 0.0090 0.0040 0.0120 0.0010
0.0510 0.0610 0.0710 0.0300 0.0530 0.7890 0.0680 0.1050 0.0700
0.0010 0.0310 0.0110 0.0290 0.0250 0.0220 0.8720 0.0160 0.0010
0.0050 0.0300 0.0090 0.0290 0.0310 0.0510 0.0230 0.8320 0.0070
0.0760 0.0000 0.0090 0.0010 0.0000 0.0320 0.0010 0.0060 0.8440
0.8430 0.7290 0.8900 0.7520 0.8470 0.7890 0.8720 0.8320 0.8440
0.8220
```

Figura 5.30 Matriz de probabilidades de diagnóstico PD2 generadas en Matlab para 100 épocas.

```
0.8490 0.0000 0.0070 0.0010 0.0010 0.0090 0.0010 0.0030 0.0570
0.0000 0.7580 0.0000 0.1530 0.0160 0.0040 0.0130 0.0040 0.0000
0.0110 0.0040 0.8780 0.0060 0.0030 0.0340 0.0190 0.0120 0.0120
0.0010 0.0810 0.0000 0.7160 0.0080 0.0010 0.0060 0.0040 0.0000
0.0000 0.0240 0.0020 0.0350 0.8630 0.0060 0.0130 0.0110 0.0010
0.0530 0.0590 0.0800 0.0370 0.0470 0.8470 0.0730 0.1130 0.0770
0.0000 0.0410 0.0090 0.0170 0.0250 0.0220 0.8450 0.0210 0.0000
0.0080 0.0320 0.0180 0.0350 0.0370 0.0540 0.0290 0.8280 0.0070
0.0780 0.0010 0.0060 0.0000 0.0000 0.0230 0.0010 0.0040 0.8460
0.8490 0.7580 0.8780 0.7160 0.8630 0.8470 0.8450 0.8280 0.8460
0.8256
```

Figura 5.31 Matriz de probabilidades de diagnóstico PD1 generadas en Matlab para 1000 épocas.

0.8490	0.0020	0.0090	0.0010	0.0010	0.0200	0.0000	0.0050	0.0740
0.0000	0.7400	0.0000	0.1340	0.0150	0.0000	0.0140	0.0090	0.0010
0.0200	0.0020	0.8840	0.0040	0.0070	0.0790	0.0100	0.0090	0.0120
0.0000	0.1060	0.0000	0.7430	0.0110	0.0010	0.0020	0.0030	0.0000
0.0000	0.0340	0.0020	0.0340	0.8560	0.0130	0.0100	0.0140	0.0010
0.0500	0.0590	0.0780	0.0280	0.0590	0.7980	0.0770	0.1230	0.0760
0.0010	0.0270	0.0110	0.0260	0.0210	0.0200	0.8640	0.0170	0.0000
0.0050	0.0300	0.0120	0.0290	0.0300	0.0410	0.0210	0.8130	0.0050
0.0750	0.0000	0.0040	0.0010	0.0000	0.0280	0.0020	0.0070	0.8310
0.8490	0.7400	0.8840	0.7430	0.8560	0.7980	0.8640	0.8130	0.8310

0.8198

Figura 5.32 Matriz de probabilidades de diagnóstico PD1 generadas en Matlab para 1000 épocas.

Conclusiones Generales

Contar con un sistema de monitoreo confiable es de vital importancia, debido a que hoy en día las nuevas tecnologías hacen que la maquinaria y equipo industrial representen un impacto económico importante en el país, de este modo evitar que estos equipos puedan sufrir alguna falla que pueda ocasionar algún desastre que impacte no solo en lo económico si no que repercuta en vidas humanas. Siendo las turbinas de gas equipos muy sofisticados e implementados en diversas aplicaciones industriales de relevancia para el país, es de gran importancia contar con un sistema de diagnóstico capaz de proveer la información necesaria para determinar las condiciones de operación antes y después de una falla probable.

En este trabajo de investigación se presento a la Red Neuronal del tipo Perceptrón Multicapa con entrenamiento de Retro-propagación por su algoritmo matemático, para ser comparado con las funciones de Matlab que definen a esta misma configuración de red neuronal. Proponiéndose hacerse de esta manera debido a que se sabe que el algoritmo por su naturaleza siempre llegara un mínimo global esto por un lado, por el otro las funciones de Matlab trabajan en un sistema de lotes, lo que no genera algunas dudas del funcionamiento aplicando estas opción, mientras que el algoritmo de la red de retro-propagación trabaja con un sistema en serie lo que nos garantiza un mejor entrenamiento de la red.

El algoritmo de la red de retro-propagación se aplico al sistema de monitoreo, específicamente a la etapa preliminar de cálculo de desviaciones y a la etapa de diagnóstico detallado (reconocimiento de las fallas). Antes que otra cosa, cabe mencionar que del trabajo realizado debemos estar satisfechos de cierta forma debido a que los resultados obtenidos son muy cercanos a los objetivos que se esperaban alcanzar, teniendo en cuenta que el algoritmo programado por nosotros parece entrenarse de mejor manera de acuerdo a las graficas del error, sin embargo no se ve así en las matrices de probabilidades, además hay cosas que nos dejan duda, como el tiempo en que tarda el algoritmo en procesar la información es considerablemente demasiado comparado con el algoritmo del prototipo.

De acuerdo a los resultados obtenidos no es fácil poder determinar que el algoritmo de la red neuronal programado por nosotros solo pueda tener aplicación en reconocimiento de patrones y no así como aproximador de funciones en el caso del cálculo de las desviaciones, es cierto que presenta un mejor funcionamiento para reconocimiento de patrones, sin embargo los resultados obtenidos en aproximación de funciones es muy alentador no esta tan lejos de lo esperado.

Por otro lado si comparamos los lenguajes de programación Matlab y Fortran, observaremos que los resultados obtenidos en Fortran dependen seriamente del numero de épocas que es entrenada la red, debido a que entre mayor se a el numero de épocas los resultados se acercan más a los esperados, sin embargo, el tiempo que le toma a fortran para procesar los cálculos es casi el mismo que el de Matlab y es aquí donde esperábamos una notable diferencia a favor de Fortran. A pesar de esto debemos estar consientes que el nivel de programación puede tomar una gran importancia, es decir que tal vez con más tiempo y un nivel de programación más avanzado en Fortran pueda ser posible disminuir el tiempo que tarda el programa.

Referencias

[1] Rao, B.K.N.(1996). Handbook of Condition Monitoring & Maintenance Management in Industry. Elsevier Applied Science.

[2] Bela Liptak (1995). Instrument Engineering Handbook, third Edition, Butterworth/Heinemann.

[3] Handbook on Safety Related Maintenance (1993) International Atomic Energy Agency (IAEA) Vienna.

[4] Mobley, R.K.(1994). The Horizons of Maintenance Management, Maintenance Handbook, Fift Edition, McGrawHill, New York.

[5] Williams, J.H. Davies, A. & Drake, P.R. (1994). Condition-based Maintenance & Machine Diagnostics, Chapmann & Hall, UK

[6] Tony Giampaolo. The Gas turbine Handbook: Principles and Practices, Prentices Hall, 1997, USA.

[7] Personal computer monitoring system, BARCO PCMS, Technical leaflet, Barco Automation Inc., USA, 1995.

[8] Shop Floor Data Collection and Monitoring, Technical leaflet, Cambridge Monitoring Systenms Ltd, UK, 1995.

[9] B. Tinham, Where Maintenance Meets Management, Control and Instrumentation, June 1992.

[10] De Ruuijter, J. A. F., Schaafssma, C.T. Laagland, G. H. M., Donauer, P. (1995). Condition Monitoring of Thermal Power Plants. Power Gen Europe, Amsterdam.

[11] Rook, Sir Dennis. Condition Monitoring and Gas Fire Power Generation, Boyce Engineering International Inc. Seminar, UK. 1994.

[12] J Maclntyre, Condition Monitoring and Artificial Intelligence – A Literature Review, University of Sunderland, 1993.

[13] Kacprzynski Gregory J., Michel Gumina, Micheal J. Roemer, Daniel E. Caguiat, Thomas R. Galie, Jack J. McGroarty. A prognostic modeling approach for predicting recurring maintenance for shipboard propulsion system. IGTI/ASME Turbo Expo 2001, New Orlans, USA, 7p.

[14] Lars J. Kangas, Frank L. Greitzer. Automated health monitoring of M1A1/A2 battle tank. USAF Inaugural Engine Condition Monitoring (ECM) Workshop, 1997, San Diego, CA, USA.

[15] Fank L. Greitzer, Lars J. Kangas, Kristine M. Terrones, Melody A. Maynard, Bary W. Wilson, Ronald A. Pawlowski, Daniel R. Sisk and Newton B. Brown. Gas turbine engine health monitoring and prognostics. Internacional Society of Logistics Symposium, Las Vegas, Nevada, USA, 1999, 7p.

[16] Loboda, I., Yepifanov, S. and Feldstheyn, Ya.(2004). Deviation problem in gas turbine health monitoring, Proceedings of IASTED International Conference on Power an Energy Systems., 6p., Clearwater Beach, Florida, USA.

[17] Saravanamuttoo, H. I. H: and Maclsaac, B. D. (1983). Thermodynamic models for pipeline gas turbine diagnostics, ASME Journal of Engineering for Power, vol. 105, No. 10, pp. 875-884.

[18] Magnus Fast, Mohsen Assedi, Andrew Pike, Peter Breuhaus. Different condition monitoring models for gas turbines by means of artificial neural network. IGTI/ASME Turbo Expo 2009, June 8-12, 2009, Orlando, Florida, USA, 11p. ASME Paper GT2009-59364.
[19] William, W. Bathie. Fundamentos de Turbinas de Gas. Limusa, Mexico.
[20] Yunus A. Cengel, Michael A. Boles, Termodinámica, Mc Graw-Hill Interamericana, 2003.
[21] Boyce, M. P. Cogeneration and Combined Cycle Power Plants, ASME
[22] Boyce Meherwan P., Gas Turbine Engineering Handbook, Gulf Publishing Company, Houston Texas, July 1995.
[23] Loboda Igor. "An overview of Gas Turbine Healt Monitoring Methods", XI Congreso y Exposicion Latinoamericana de Trubomaquinaria, 14-17 Octubre 2008, Veracruz, Veracruz México, 10p.
[24] Hans R. Depold, Ravi Rajamani, William H. Morrison, Krishna R. Pattipati. A unified metric for fault detection and isolation in engines. IGTI/ASME Turbo Expo 2006, Barcelona, Spain, 7p
[25] Bonifacio Martin del brío, Alfredo Sanz Molina, *Redes Neuronales y Sistemas Difusos*, Alfaomega Grupo Editor, S.A. de C.V. 2005.
[26] James, A. Anderson. Redes Neuronales, Alfaomega Grupo Editores, 1ra edición, México, 2007.
[27] Simpson, P.K. *Artificial Neural Systems*, Pergamon Press,1989.
[28] Gedeon, T.D., WOng, P. M., Harris, D. *Balancing the bias and variance: Network topology and pattern set reduction techniques*. Proc. Int. Work on Artificial Neural Netyworks, IAWNN95, pp550-8, Torremolinos (España), junio, 1995.
[29] Werbos, P.J. *Back-propagation thruourh time: What it does and how to do it*. Proc of the IEEE, oct., pp. 1150-1560, 1990.
[30] Rumelhart, D. E., Hinton G. E., Willians R. J. *Learning representations by backpropagation errors*. Nature, 332, 533-6, 1986
[31] Bishop, C. M *Neural network and their applications*. Rev. Sci. Instrum., 65, 6, pp 1803-1832, 1994.
[32] Holly Moore, Matlab para Ingenieros, Pearson Prentice Hall, 1ra. Edición, 2007.
[33] Sánchez Garfias, F.A., Díaz de León Santiago, J.L. & Yáñez Márquez, C. (2003). *Reconocimiento automático de patrones: conceptos básicos*, IT-79, Serie Verde, ISBN 970-36-0044-1, CIC-IPN, México.
[34] Duda O. Richard, Hart E. Peter, Stork G. David, *Pattern Clasification*, A Wiley-Interscience Publication, 2nd ed., 2000.
[35] Adams, Brainerd, Martin, Smith and Wagner. Fortran 90 Handbook, 1992 McGraw Hill. ISBN 0-07-000406-4
[36] Adams, J.C., Brainerd, W.W., Hendrickson, R.A., Maine, R.E., Martin, J.T., Smith, B.T. The Fortran 2003 Handbook, The Complete Syntax, Features an Procedures, 2008, Spring Verlag. ISBN: 978-1-84628-378-9
[37] http://es.wikipedia.org/wiki/Fortran
[38] Walter, S. Brainerd. Guide to Fortran 2003 Programming, Springer, 2009.

APÉNDICE 1.

SOFTWARE DESARROLLADO

Algoritmo 1: Calculo de desviaciones

```
% NORMAL MODEL CALCULATION
%Version 1.0 - Matrixes, comments
%                MODEL COEFFICIENT  CALCULATION
%   Dimention parameters, model structure arrays
%Version 1.2
%   Universal "base-export" format and units
%   Possibility of temperature correction
%Version 1.3
%   Usage the operation time for plots
%Version 1.6
%   Argument change
%Version 1.7 (in the basis of the versions 1.2 & 1.6)
%   Universal: SMALL(simulated) base without time variable or GREAT(real)
base with time(lineal or cuadratic) [Change base or kcoef]
%   POLINOMIALS+NETWORKS
clear;
tcpu=cputime;
kuyt=12; ku=4; ky=7;
idegr=0;                % degradation variable
%ph, tv0, npt, gt,    nhp, pok, egt, tok, egp, tst, kw,    tnars
% 1   2    3   4       1    2    3    4    5    6    7
%sqrtt0=16.975;          % For temperature correction
tk=273.15; ktk=4; ntk=[2 7 8 10];

kcoef=17;                                              %*  16:(t)-member;
17:(t+t*t)-members
npol1=[5 1 2 3 4 1 1 1 2 2 3 1 2 3 4 6 6];             %*
```

```
npol2=[5 5 5 5 5 2 3 4 3 4 4 1 2 3 4 5 6];              %*

%              MODEL FORMATION
% File size
fid=fopen('base.txt','rt');
krb=-2;
while ne(1,feof(fid))
   tmpline=fgets(fid);
   krb=krb+1;
end;
fclose(fid);
if krb >= 1000
   idegr=1;          % data with degradation, time variable introduction
else
   kcoef=15;         % without time variable
end;
% Measured parameter input
fid=fopen('base.txt','rt');
tmpline=fgets(fid);  % skip line
tmpline=fgets(fid);  % skip line
UY=fscanf(fid,'%g',[kuyt,krb]);   % reading the U and Y information
fclose(fid);
UY=UY';
% Matrix formation
for k=1:ktk
   UY(:,ntk(k))=UY(:,ntk(k))+tk;
end;
%    Argument change
nu1=4;
nu2=4; DZ=[.004,.018,.015,.0025,.015,.02,.025];            % argument  gt
%nu2=5; DZ=[.03,.035,.01,.009,.03,.017,.06];                % argument
nhp
%nu2=6; DZ=[.0075,.025,.0275,.004,.001,.028,.01];           % argument
pok
%nu2=7; DZ=[.003,.05,.04,.012,.04,.013,.085];               % argument
egt
%nu2=8; DZ=[.005,.015,.02,.01,.015,.023,.015];              % argument
tok
%nu2=9; DZ=[.007,.001,.026,.002,.028,.028,.002];            % argument
egp
%nu2=10; DZ=[.01,.07,.015,.02,.06,.07,.13];                 % argument
tst
%nu2=11; DZ=[.006,.004,.025,.002,.003,.029,.01];            % argument  kW
TMPb=ones(krb,1);
TMPb(:)=UY(:,nu1);
UY(:,nu1)=UY(:,nu2);
UY(:,nu2)=TMPb(:);
%
for k=1:ku
   U(:,k)=UY(:,k);
end;
u1=ones(krb,1);
U(:,ku+1)=u1;
if idegr == 1                                          %*
   tf=UY(1,kuyt);                                     %*   Operating time of the
first point
```

```
   tfe=654.0;                                          %*   Equivalent time of
the first point
   krw=908; tw=7972.13;                                %*   Point number and
operating time after the washing
   for k=1:(krw-1)                                     %*
       UY(k,kuyt)=UY(k,kuyt)-tf+tfe;                 %*
   end;                                                %*
    for k=krw:krb                                      %*
       UY(k,kuyt)=UY(k,kuyt)-tw;                     %*
   end;                                                %*
   U(:,ku+2)=UY(:,kuyt);                               %*
end;                                                   %*
for k=1:kcoef
   A(:,k)=U(:,npol1(k)).*U(:,npol2(k));
end;
%
for k=1:ky
   Y(:,k)=UY(:,k+ku);
end;
% Model coefficients, model parameters and errors
for k=1:ky
   XX(:,k)=A\Y(:,k);
   YE(:,k)=A*XX(:,k);
   DY(:,k)=(Y(:,k)-YE(:,k))./YE(:,k);
end;
for j=1:kcoef
   X(j,:)=XX(j,:);
end;
[dymx,nymx]=max(DY);
[dymn,nymn]=min(DY);
sdy=std(DY,1,1);
SDYR=DZ.\sdy;
DZRT=sdy.\DZ;
SDYS=sqrt(7)/norm(DZRT,2);
SDYS1=norm(SDYR,2)/sqrt(7);
%               MODEL APPLICATION FOR  NEW DATA
% File size
fid=fopen('export.txt','rt');
krs=-2;
while ne(1,feof(fid))
   tmpline=fgets(fid);
   krs=krs+1;
end;
fclose(fid);
% Measured parameter input
fid=fopen('export.txt','rt');
tmpline=fgets(fid);  % skip line
tmpline=fgets(fid);  % skip line
UYS=fscanf(fid,'%g',[kuyt,krs]);   % reading the U and Y information
fclose(fid);
% Matrix formation
UYS=UYS';
for k=1:ktk
   UYS(:,ntk(k))=UYS(:,ntk(k))+tk;
end;
% Argument change
TMPs=ones(krs,1);
```

```
TMPs(:)=UYS(:,nu1);
UYS(:,nu1)=UYS(:,nu2);
UYS(:,nu2)=TMPs(:);
% Matrix formation
for k=1:ku
   US(:,k)=UYS(:,k);
end;
u1s=ones(krs,1);
US(:,ku+1)=u1s;
u0s=zeros(krs,1);
US(:,ku+2)=u0s;                              %*
for k=1:kcoef
   AS(:,k)=US(:,npol1(k)).*US(:,npol2(k));
end;
for k=1:ky
   YS(:,k)=UYS(:,k+ku);
end;
% Model  parameters and deviations
for k=1:ky
   YES(:,k)=AS*X(:,k);
   DYS(:,k)=(YS(:,k)-YES(:,k))./YES(:,k);
end;
%DSUMM=(DYS(:,2)-DYS(:,3)+DYS(:,4)+DYS(:,5)-DYS(:,6))/5;      %Integral
cryterion
DSUMM=(DYS(:,2)+DYS(:,3)+DYS(:,4)+DYS(:,5)-DYS(:,6))/5;      %Integral
cryterion
DSUMM1=(DYS(:,2)+1.2*DYS(:,5)-3.5*DYS(:,6))/3;      %Integral cryterion
[dymxs,nymxs]=max(DYS);
[dymns,nymns]=min(DYS);
sdys=std(DYS,1,1);
%                        NEURAL NETWORKS
%   Normalized network inputs
kuu=ku;
nu=[1 2 3 4 6];
if idegr == 1                                        %*
   kuu=ku+1;
end;
for iu=1:kuu
   P1(:,iu)=U(:,nu(iu));
   P2(:,iu)=US(:,nu(iu));
end;
P1=P1'; P2=P2'; MiMaU=minmax(P1); DU=zeros(kuu,1); DU(:)=(MiMaU(:,2)-
MiMaU(:,1))/2;
beta=1.0;
for iu=1:kuu
   for ir=1:krb
       P1(iu,ir)=beta*(P1(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
   for ir=1:krs
       P2(iu,ir)=beta*(P2(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
end;
%   Normalized targets
alf=0.9;
YY=[Y;YS];
MnY=min(YY);     MxY=max(YY);     DDY=(MxY-MnY)/2.0;
for iy=1:ky
```

```
    for ir=1:krb
       T1(iy,ir)=alf*(Y(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
    end;
    for ir=1:krs
       T2(iy,ir)=alf*(YS(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
    end;
end;
%  Network structure, training, and simulation
MiMa1=minmax(P1);
nn=12; nm=13; tgoal=0.000015;
if nm==1              % Training function
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainlm');
   nepochs=1500;   nshow=50;
   net.trainParam.mem_reduc=1;
elseif  nm==2
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traingd');
   nepochs=50000;   nshow=100;
   net.trainParam.lr=0.5;
elseif  nm==3
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traingdm');
   nepochs=50000;   nshow=100;
   net.trainParam.lr=0.1;
   net.trainParam.mc=0.5;
elseif  nm==4
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traingda');
   nepochs=50000;   nshow=100;
   net.trainParam.lr=0.1;
   net.trainParam.lr_inc=1.10;
elseif  nm==5
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traingdx');
   nepochs=50000;   nshow=20;
   net.trainParam.mc=0.5;
   net.trainParam.lr=0.1;
elseif  nm==6
   net1 =newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainrp');
   nepochs=100;   nshow=20;
elseif  nm==7
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traincgf');
   nepochs=5000;   nshow=100;
elseif  nm==8
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traincgp');
   nepochs=50000;   nshow=100;
   net1.trainParam.delta=0.0001;
elseif  nm==9
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'traincgb');
   nepochs=102400;   nshow=100;
   net1.trainParam.delta=0.0001;
elseif  nm==10
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainscg');
   nepochs=50000;   nshow=100;
elseif  nm==11
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainbfg');
   nepochs=102400;   nshow=200;
elseif  nm==12
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainoss');
   nepochs=102400;   nshow=400;
   net1.trainParam.min_grad=  1e-7;
```

```
elseif  nm==13
   net1=newff(MiMa1,[nn,ky],{'tansig','tansig'},'trainbr');
   nepochs=2000;   nshow=50;
end;
net1.trainParam.show=nshow; net1.trainParam.epochs=nepochs;
net1.trainParam.goal=tgoal;
[net1,tr]=train(net1,P1,T1);              %   Training
A1=sim(net1,P1);  A2=sim(net1,P2);        %   Simulation
DA1T=(A1-T1)'; sser=norm(DA1T,'fro'); sser=sser*sser;     % sum of
squared errors
mser=sser/(krb*ky);                                       % mean squared
error
%    Output denormalization. Networks deviations
for iy=1:ky
   for ir=1:krb
      YN(ir,iy)=A1(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
   end;
   for ir=1:krs
      YNS(ir,iy)=A2(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
   end;
   DYN(:,iy)=(Y(:,iy)-YN(:,iy))./YN(:,iy);
   DYNS(:,iy)=(YS(:,iy)-YNS(:,iy))./YNS(:,iy);
end;
sdyn=std(DYN,1,1);
sdysn=std(DYNS,1,1);

%    Networks behaviour verification by their y(u)-plots
krv=2000;  nuv=4;     % points quantity and number of the first u-
argument (nuv=5 - time variable for idegr=1 only)
nuf=3;  kuf=7;        % number and points quantity of the second u-
argument
nyv=3;                % y-function number
du=2.0/krv; duf=2.0/(kuf-1); tv=1:krv;
for iuf=1:kuf
   for ir=1:krv
      for iu=1:kuu
         P1V(iu,ir)=0.0;
      end;
      P1V(nuf,ir)=-1.0+(iuf-1)*duf;
      P1V(nuv,ir)=-1.0+du*ir;
   end;
   A1V=sim(net1,P1V);
   for iy=1:ky
      for ir=1:krv
         YNV(ir,iy,iuf)=A1V(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
      end;
   end;
end;
figure(5);
plot(tv,YNV(tv,nyv,1),tv,YNV(tv,nyv,2),tv,YNV(tv,nyv,3),tv,YNV(tv,nyv,4),
tv,YNV(tv,nyv,5),tv,YNV(tv,nyv,6),tv,YNV(tv,nyv,7));
grid on; legend('1','2','3','4','5','6','7');
%               COMMON PLOTS
t=1:krb;  ts=1:krs;
% figure(1);
% plot(t,DY(t,3),t,DY(t,4),t,DY(t,6)); grid on;
legend('egt','tok','tst');
```

```
% figure(2);
% plot(t,DYN(t,3),t,DYN(t,4),t,DYN(t,6)); grid on;
legend('egt','tok','tst');
% figure(3);
% plot(ts,DYS(ts,3),ts,DYS(ts,4),ts,DYS(ts,6)); grid on;
legend('egt','tok','tst');
% figure(4);
% plot(ts,DYNS(ts,3),ts,DYNS(ts,4),ts,DYNS(ts,6)); grid on;
legend('egt','tok','tst');
top=cputime-tcpu

figure(1);
subplot(6,1,1),plot(t,DY(t,3)); grid on; ylabel('dTtp');
subplot(6,1,2),plot(t,DYN(t,3)); grid on; ylabel('dTtn');
subplot(6,1,3),plot(t,DY(t,4)); grid on; ylabel('dTcp');
subplot(6,1,4),plot(t,DYN(t,4)); grid on; ylabel('dTcn');
subplot(6,1,5),plot(t,DY(t,6)); grid on; ylabel('dTptp');
subplot(6,1,6),plot(t,DYN(t,6)); grid on; xlabel('t'); ylabel('dTptn');
% subplot(2,1,1),plot(t,DY(t,3)); grid on; ylabel('dTtp');
% subplot(2,1,2),plot(t,DYN(t,3)); grid on; ylabel('dTtn');

figure(2);
% subplot(6,1,1),plot(ts,DYS(ts,3)); grid on; ylabel('dTtp');
% subplot(6,1,2),plot(ts,DYNS(ts,3)); grid on; ylabel('dTtn');
% subplot(6,1,3),plot(ts,DYS(ts,4)); grid on; ylabel('dTcp');
% subplot(6,1,4),plot(ts,DYNS(ts,4)); grid on; ylabel('dTcn');
% subplot(6,1,5),plot(ts,DYS(ts,6)); grid on; ylabel('dTptp');
% subplot(6,1,6),plot(ts,DYNS(ts,6)); grid on; xlabel('t');
ylabel('dTptn');
subplot(2,1,1),plot(ts,DYS(ts,3)); grid on; ylabel('dTtp');
subplot(2,1,2),plot(ts,DYNS(ts,3)); grid on; ylabel('dTtn');

% tt=ones(krs,1);
% tt(:)=UYS(:,kuyt);
% figure(4);
% plot(tt,DYS(:,3),tt,DYS(:,4),tt,DYS(:,6)); grid on;
legend('egt','tok','tst');
```

ALGORITMO 2: Cálculo de los índices de confiabilidad del diagnóstico

```
%   Trustworthiness "neural networks"
% Version 00
%   - 12 methods
% Version 01
%   - Early Stopping Option
% Version 02
%   - Multiple classes
%
% Defects:     Gc, EFFc, Gt, EFFt, Gpt, EFFpt, SIGcc, EFFcc, SIGin
% Parameters:  Pc, Pt,   Tc, Tt,   Tpt, Gf
% Regimes
n_hp_corr=10700,10600,10500,10400,10300,10200,10100,10000,9900,9800,9700
% Defect development 0-5% (idl=1-21)
clear;
krg=11;   % regime quantity
nreg=1;
kdf=9;    % class  quantity
kdl=21;   % interpolation points quantity for every class development
kdf0=9;    % maximal classes  number
kdf1=9;   % single classes  number
idef2=0; kdf2=4;                              % multiple classes signal and
number
kpr0=6;    % parameter quantity
ktd=1000; % class simulation points quantity
nr1=0; nr2=1;    % random generators initiation
DY=[0.015 0.015 0.025 0.015 0.020 0.020];   % parameter maximal errors
krdd=krg*kdf0*kdl;
i_EarlyStop=0; %Early Stopping Option

%                              LEARNING SAMPLES
% Reading from the file
fid=fopen('dpar.dat','rt');
BT=fscanf(fid,'%g',[kpr0,krdd]);
BB=BT';
for ipr=1:kpr0
   BB(:,ipr)=BB(:,ipr)/DY(ipr);
end;
% Formation of the multidimensional array of defect induced deviations
for irg=1:krg
   for idf=1:kdf0
      for idl=1:kdl
         irdd=(irg-1)*kdf0*kdl+(idf-1)*kdl+idl;
         B(:,idl,idf,irg)=BB(irdd,:);
      end;
   end;
end;
% Regime determination
D=B(:,:,:,nreg);

% Teaching sample
ZE=ones(kpr0,1);
ZD_1=ones(kpr0,1);
```

```
ZD_2=ones(kpr0,1);
rand('state',nr1);  randn('state',nr1);                %State fixing of
generators

if idef2==0
   kdf=kdf1;
else
   kdf=kdf2;
end;
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD1(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD1(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
      end;
   end;
end;

% Test sample points)
rand('state',nr2);  randn('state',nr2);                %State fixing of
generators
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      %dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD2(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD2(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
      end;
   end;
end;
```

```
% Formation of net entries
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P1(:,n)=ZD1(:,itd,idf);
      T1(idf,n)=1;
   end;
end;
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P2(:,n)=ZD2(:,itd,idf);
      T2(idf,n)=1;
   end;
end;


% Net training
MiMa=minmax(P1);
nm=6; %Method number
if nm==2
   net =newff(MiMa,[12,9],{'tansig','logsig'},'trainlm');
   net.trainParam.epochs=100;
   net.trainParam.show=5;
   net.trainParam.mem_reduc=2;
elseif  nm==2
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traingd');
   net.trainParam.lr=0.1;
   net.trainParam.epochs=8000;
   net.trainParam.show=50;
elseif  nm==3
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traingdm');
   net.trainParam.lr=0.1;
   net.trainParam.mc=0.5;
   net.trainParam.epochs=8000;
   net.trainParam.show=50;
elseif  nm==4
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traingda');
   net.trainParam.lr=0.1;
   net.trainParam.lr_inc=1.05;
   net.trainParam.epochs=1000;
   net.trainParam.show=50;
elseif  nm==5
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traingdx');
   net.trainParam.lr=0.1;
   net.trainParam.mc=0.5;
   net.trainParam.lr_inc=1.02;
   net.trainParam.epochs=3000;
   net.trainParam.show=50;
elseif  nm==6
   net =newff(MiMa,[12,9],{'tansig','logsig'},'trainrp');
   net.trainParam.epochs=300;
   net.trainParam.show=20;
elseif  nm==7
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traincgf');
   net.trainParam.epochs=200;
```

```
   net.trainParam.show=20;
elseif  nm==8
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traincgp');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==9
   net =newff(MiMa,[12,9],{'tansig','logsig'},'traincgb');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==10
   net =newff(MiMa,[12,kdf],{'tansig','logsig'},'trainscg');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==11
   net =newff(MiMa,[12,9],{'tansig','logsig'},'trainbfg');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
elseif  nm==12
   net =newff(MiMa,[12,9],{'tansig','logsig'},'trainoss');
   net.trainParam.epochs=200;
   net.trainParam.show=20;
end;
net.trainParam.goal=0.026;
if i_EarlyStop==1
   val.P=P2;
   val.T=T2;
   [net,tr]=train(net,P1,T1,[],[],val);
else
   [net,tr]=train(net,P1,T1);
end;
A1=sim(net,P1);     %Simulation
A2=sim(net,P2);     %Simulation

%                            TRUSTWORTHINESS CALCULATION
PD1=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A1(:,n));
      PD1(imax,idf)=PD1(imax,idf)+1;
   end;
end;
PD1=PD1/ktd;
PDT1=diag(PD1);
psr1=mean(PDT1);
PD1T=PD1';
%fw1=fopen('c:\_igor\matlab\Work\pd1.dat','wt');
fw1=fopen('pd1.dat','wt');
if idef2==0
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD1T,PDT1,psr1);
else
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f\n',PD1T,PDT1,psr1);
end;
fclose(fw1);
```

```
PD2=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A2(:,n));
      PD2(imax,idf)=PD2(imax,idf)+1;
   end;
end;
PD2=PD2/ktd;
PDT2=diag(PD2);
psr2=mean(PDT2);
PD2T=PD2';
fw2=fopen('pd2.dat','wt');
if idef2==0
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD2T,PDT2,psr2);
else
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f\n',PD2T,PDT2,psr2);
end;
fclose(fw2);

figure(2);
plot3(ZD1(1,:,1),ZD1(2,:,1),ZD1(3,:,1),'*',ZD1(1,:,2),ZD1(2,:,2),ZD1(3,:,
2),'.',ZD1(1,:,3),ZD1(2,:,3),ZD1(3,:,3),'+',ZD1(1,:,4),ZD1(2,:,4),ZD1(3,:
,4),'x'); grid on; legend('D1','D2','D3','D4');
xlabel('Z1 - Compressor pressure');
ylabel('Z2 - Turbine pressure');
zlabel('Z3 - Compressor temperature');
```

Versión 1.1 del algoritmo 1

```
tic;
% NORMAL MODEL CALCULATION
%Version 1.0 - Matrixes, comments
%               MODEL COEFFICIENT  CALCULATION
%   Dimention parameters, model structure arrays
%Version 1.2
%   Universal "base-export" format and units
%   Possibility of temperature correction
%Version 1.3
%   Usage the operation time for plots
%Version 1.6
%   Argument change
%Version 1.7 (in the basis of the versions 1.2 & 1.6)
%   Universal: SMALL(simulated) base without time variable or GREAT(real)
base with time(lineal or cuadratic) [Change base or kcoef]
%   POLINOMIALS+NETWORKS
clear;
clc;
tcpu=cputime;
kuyt=12; ku=4; ky=7;
idegr=0;                % degradation variable
%ph, tv0, npt, gt,    nhp, pok, egt, tok, egp, tst, kw,    tnars
% 1   2    3   4       1    2    3    4    5    6    7
%sqrtt0=16.975;          % For temperature correction
tk=273.15; ktk=4; ntk=[2 7 8 10];

kcoef=17;                                                  %*  16:(t)-member;
17:(t+t*t)-members
npol1=[5 1 2 3 4 1 1 1 2 2 3 1 2 3 4 6 6];                 %*
npol2=[5 5 5 5 5 2 3 4 3 4 4 1 2 3 4 5 6];                 %*

%              MODEL FORMATION
% File size
fid=fopen('base.txt','rt');
krb=-2;
while ne(1,feof(fid))
   tmpline=fgets(fid);
   krb=krb+1;
end;
fclose(fid);
if krb >= 1000
   idegr=1;         % data with degradation, time variable introduction
else
   kcoef=15;        % without time variable
end;
% Measured parameter input
fid=fopen('base.txt','rt');
tmpline=fgets(fid);  % skip line
tmpline=fgets(fid);  % skip line
UY=fscanf(fid,'%g',[kuyt,krb]);   % reading the U and Y information
fclose(fid);
UY=UY';
% Matrix formation
for k=1:ktk
```

```
   UY(:,ntk(k))=UY(:,ntk(k))+tk;
end;
%    Argument change
nu1=4;
nu2=4; DZ=[.004,.018,.015,.0025,.015,.02,.025];            % argument  gt
%nu2=5; DZ=[.03,.035,.01,.009,.03,.017,.06];                % argument
nhp
%nu2=6; DZ=[.0075,.025,.0275,.004,.001,.028,.01];           % argument
pok
%nu2=7; DZ=[.003,.05,.04,.012,.04,.013,.085];               % argument
egt
%nu2=8; DZ=[.005,.015,.02,.01,.015,.023,.015];              % argument
tok
%nu2=9; DZ=[.007,.001,.026,.002,.028,.028,.002];            % argument
egp
%nu2=10; DZ=[.01,.07,.015,.02,.06,.07,.13];                 % argument
tst
%nu2=11; DZ=[.006,.004,.025,.002,.003,.029,.01];            % argument  kW
TMPb=ones(krb,1);
TMPb(:)=UY(:,nu1);
UY(:,nu1)=UY(:,nu2);
UY(:,nu2)=TMPb(:);
%
for k=1:ku
   U(:,k)=UY(:,k);
end;
u1=ones(krb,1);
U(:,ku+1)=u1;
if idegr == 1                                              %*
   tf=UY(1,kuyt);                                         %*   Operating time of the
first point
   tfe=654.0;                                              %*   Equivalent time of
the first point
   krw=908; tw=7972.13;                                    %*   Point number and
operating time after the washing
   for k=1:(krw-1)                                         %*
       UY(k,kuyt)=UY(k,kuyt)-tf+tfe;                      %*
   end;                                                    %*
    for k=krw:krb                                          %*
       UY(k,kuyt)=UY(k,kuyt)-tw;                          %*
   end;                                                    %*
   U(:,ku+2)=UY(:,kuyt);                                  %*
end;                                                       %*
for k=1:kcoef
   A(:,k)=U(:,npol1(k)).*U(:,npol2(k));
end;
%
for k=1:ky
   Y(:,k)=UY(:,k+ku);
end;
% Model coefficients, model parameters and errors
for k=1:ky
   XX(:,k)=A\Y(:,k);
   YE(:,k)=A*XX(:,k);
   DY(:,k)=(Y(:,k)-YE(:,k))./YE(:,k);
end;
for j=1:kcoef
```

```
   X(j,:)=XX(j,:);
end;
[dymx,nymx]=max(DY);
[dymn,nymn]=min(DY);
sdy=std(DY,1,1);
SDYR=DZ.\sdy;
DZRT=sdy.\DZ;
SDYS=sqrt(7)/norm(DZRT,2);
SDYS1=norm(SDYR,2)/sqrt(7);
%                MODEL APPLICATION FOR  NEW DATA
% File size
fid=fopen('export.txt','rt');
krs=-2;
while ne(1,feof(fid))
   tmpline=fgets(fid);
   krs=krs+1;
end;
fclose(fid);
% Measured parameter input
fid=fopen('export.txt','rt');
tmpline=fgets(fid);  % skip line
tmpline=fgets(fid);  % skip line
UYS=fscanf(fid,'%g',[kuyt,krs]);   % reading the U and Y information
fclose(fid);
% Matrix formation
UYS=UYS';
for k=1:ktk
   UYS(:,ntk(k))=UYS(:,ntk(k))+tk;
end;
% Argument change
TMPs=ones(krs,1);
TMPs(:)=UYS(:,nu1);
UYS(:,nu1)=UYS(:,nu2);
UYS(:,nu2)=TMPs(:);
% Matrix formation
for k=1:ku
   US(:,k)=UYS(:,k);
end;
u1s=ones(krs,1);
US(:,ku+1)=u1s;
u0s=zeros(krs,1);
US(:,ku+2)=u0s;                          %*
for k=1:kcoef
   AS(:,k)=US(:,npol1(k)).*US(:,npol2(k));
end;
for k=1:ky
   YS(:,k)=UYS(:,k+ku);
end;
% Model  parameters and deviations
for k=1:ky
   YES(:,k)=AS*X(:,k);
   DYS(:,k)=(YS(:,k)-YES(:,k))./YES(:,k);
end;
%DSUMM=(DYS(:,2)-DYS(:,3)+DYS(:,4)+DYS(:,5)-DYS(:,6))/5;     %Integral
cryterion
DSUMM=(DYS(:,2)+DYS(:,3)+DYS(:,4)+DYS(:,5)-DYS(:,6))/5;     %Integral
cryterion
```

```
DSUMM1=(DYS(:,2)+1.2*DYS(:,5)-3.5*DYS(:,6))/3;     %Integral cryterion
[dymxs,nymxs]=max(DYS);
[dymns,nymns]=min(DYS);
sdys=std(DYS,1,1);
%                       NEURAL NETWORKS
%   Normalized network inputs
kuu=ku;
nu=[1 2 3 4 6];
if idegr == 1                                      %*
   kuu=ku+1;
end;
for iu=1:kuu
   P1(:,iu)=U(:,nu(iu));
   P2(:,iu)=US(:,nu(iu));
end;
P1=P1'; P2=P2'; MiMaU=minmax(P1); DU=zeros(kuu,1); DU(:)=(MiMaU(:,2)-
MiMaU(:,1))/2;
beta=1.0;
for iu=1:kuu
   for ir=1:krb
       P1(iu,ir)=beta*(P1(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
   for ir=1:krs
       P2(iu,ir)=beta*(P2(iu,ir)-MiMaU(iu,1)-DU(iu))/DU(iu);
   end;
end;
%   Normalized targets
alf=0.9;
YY=[Y;YS];
MnY=min(YY);    MxY=max(YY);    DDY=(MxY-MnY)/2.0;
for iy=1:ky
   for ir=1:krb
      T1(iy,ir)=alf*(Y(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
   end;
   for ir=1:krs
      T2(iy,ir)=alf*(YS(ir,iy)-MnY(iy)-DDY(iy))/DDY(iy);
   end;
end;
%  Network structure, training, and simulation


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%%

% VALORES DE ENTRADA
N = 5;  %input('Numero de neuronas de entrada: ');
L = 12; %input('Numero de neuronas ocultas: ');
M = 7; %input('Numero de neuronas de salida: ');
alfa = 0.0025; %input('Valor de alfa: ');
min = 0.000015;    %input('Minimo error: ');
iter = 1000; %input('Numero de iteraciones: ');
patrones_ent = 2608;   %input('Numero de patrones de entrenamiento: ');
patrones_sim = 4091;   %input('Numero de patrones de simulacion: ');
```

```
patrones = patrones_ent;

[matriz_Wh_ji, matriz_Wo_kj, vector_J] = ent_bpj (N, L, M, alfa, min,
iter, patrones, P1, T1);
J_new = sqrt((vector_J).*patrones);
%J_new = sqrt((vector_J.*vector_J).*patrones);
%Wh_ji = matriz_Wh_ji;
%Wo_kj = matriz_Wh_ji;

%figure(2);
%plot(J);
%xlabel('Epocas');
%ylabel('Error Medio Cuadratico interno');
%grid on;

figure(2);
plot(J_new);
xlabel('Epocas');
ylabel('Error Medio Interno');
grid on;

%save pesos2 Wh_ji Wo_kj
%save error J;

load pesos2 Wh_ji Wo_kj;
porcentaje=[];

% Agregando el Bias a la entrada
B1_1 = [P1];
B_1 = B1_1';
PB_1=[];
for g = 1:patrones_ent
    PB11_1(g,:) = [1,B_1(g,:)];
end
PB1_1 = PB11_1';
PB = PB1_1;
T = T1;

for ip = 1:patrones_ent
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_1 = Uo_k;
    A1 (:,ip)= Uo_k_1';
end

%patrones = patrones_sim;
% Agregando el Bias a la entrada
B1_2 = [P2];
B_2 = B1_2';
PB_2=[];
for g = 1:patrones_sim
    PB11_2(g,:) = [1,B_2(g,:)];
end
PB1_2 = PB11_2';
PB = PB1_2;
T = ones (M,patrones_sim);
```

```
for ip = 1:patrones_sim
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_2 = Uo_k;
    A2 (:,ip)= Uo_k_2';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DA1T=(A1-T1)'; sser=norm(DA1T,'fro'); sser=sser*sser;     % sum of
squared errors
mser=sser/(krb*ky);                                       % mean squared
error
%    Output denormalization. Networks deviations
for iy=1:ky
   for ir=1:krb
      YN(ir,iy)=A1(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
   end;
   for ir=1:krs
      YNS(ir,iy)=A2(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
   end;
   DYN(:,iy)=(Y(:,iy)-YN(:,iy))./YN(:,iy);
   DYNS(:,iy)=(YS(:,iy)-YNS(:,iy))./YNS(:,iy);
end;
sdyn=std(DYN,1,1);
sdysn=std(DYNS,1,1);

%    Networks behaviour verification by their y(u)-plots
krv=2000;  nuv=4;     % points quantity and number of the first u-
argument (nuv=5 - time variable for idegr=1 only)
nuf=3;  kuf=7;        % number and points quantity of the second u-
argument
nyv=3;                % y-function number
du=2.0/krv; duf=2.0/(kuf-1); tv=1:krv;
for iuf=1:kuf
   for ir=1:krv
      for iu=1:kuu
         P1V(iu,ir)=0.0;
      end;
      P1V(nuf,ir)=-1.0+(iuf-1)*duf;
      P1V(nuv,ir)=-1.0+du*ir;
   end;
   %%%%%%%%% SIMULACION DE P1V
   % Agregando el Bias a la entrada
   B1_1V = [P1V];
   B_1V = B1_1V';
   PB_2=[];
   for g = 1:ir
       PB11_1V(g,:) = [1,B_1V(g,:)];
   end
   PB1_1V = PB11_1V';
   PB = PB1_1V;
   T_2 = T2;
   T = T_2;
   aux_2 = 0;
```

```
    for ip = 1:ir
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        Uo_k_1V = Uo_k;
        A1V (:,ip)= Uo_k_1V';
    end
    %A1V=sim(net1,P1V);
    for iy=1:ky
       for ir=1:krv
          YNV(ir,iy,iuf)=A1V(iy,ir)*DDY(iy)/alf+MnY(iy)+DDY(iy);
       end;
    end;
end;
figure(7);
plot(tv,YNV(tv,nyv,1),tv,YNV(tv,nyv,2),tv,YNV(tv,nyv,3),tv,YNV(tv,nyv,4),
tv,YNV(tv,nyv,5),tv,YNV(tv,nyv,6),tv,YNV(tv,nyv,7));
grid on; legend('1','2','3','4','5','6','7');
%               COMMON PLOTS
t=1:krb;  ts=1:krs;
% figure(1);
% plot(t,DY(t,3),t,DY(t,4),t,DY(t,6)); grid on;
legend('egt','tok','tst');
% figure(2);
% plot(t,DYN(t,3),t,DYN(t,4),t,DYN(t,6)); grid on;
legend('egt','tok','tst');
% figure(3);
% plot(ts,DYS(ts,3),ts,DYS(ts,4),ts,DYS(ts,6)); grid on;
legend('egt','tok','tst');
% figure(4);
% plot(ts,DYNS(ts,3),ts,DYNS(ts,4),ts,DYNS(ts,6)); grid on;
legend('egt','tok','tst');
top=cputime-tcpu

figure(5);
subplot(6,1,1),plot(t,DY(t,3)); grid on; ylabel('dTtp');
subplot(6,1,2),plot(t,DYN(t,3)); grid on; ylabel('dTtn');
subplot(6,1,3),plot(t,DY(t,4)); grid on; ylabel('dTcp');
subplot(6,1,4),plot(t,DYN(t,4)); grid on; ylabel('dTcn');
subplot(6,1,5),plot(t,DY(t,6)); grid on; ylabel('dTptp');
subplot(6,1,6),plot(t,DYN(t,6)); grid on; xlabel('t'); ylabel('dTptn');
% subplot(2,1,1),plot(t,DY(t,3)); grid on; ylabel('dTtp');
% subplot(2,1,2),plot(t,DYN(t,3)); grid on; ylabel('dTtn');

figure(6);
% subplot(6,1,1),plot(ts,DYS(ts,3)); grid on; ylabel('dTtp');
% subplot(6,1,2),plot(ts,DYNS(ts,3)); grid on; ylabel('dTtn');
% subplot(6,1,3),plot(ts,DYS(ts,4)); grid on; ylabel('dTcp');
% subplot(6,1,4),plot(ts,DYNS(ts,4)); grid on; ylabel('dTcn');
% subplot(6,1,5),plot(ts,DYS(ts,6)); grid on; ylabel('dTptp');
% subplot(6,1,6),plot(ts,DYNS(ts,6)); grid on; xlabel('t');
ylabel('dTptn');
subplot(2,1,1),plot(ts,DYS(ts,3)); grid on; ylabel('dTtp');
subplot(2,1,2),plot(ts,DYNS(ts,3)); grid on; ylabel('dTtn');

% tt=ones(krs,1);
% tt(:)=UYS(:,kuyt);
```

```
% figure(4);
% plot(tt,DYS(:,3),tt,DYS(:,4),tt,DYS(:,6)); grid on;
legend('egt','tok','tst');
toc;
```

Función de entrenamiento: ent_bpj

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%
function [Wh_ji, Wo_kj, J] = ent_bpj (N, L, M, alfa, min, iter, patrones,
P1, T1)
% VALORES DE ENTRADA
%N = 6;  %input('Numero de neuronas de entrada: ');
%L = 12; %input('Numero de neuronas ocultas: ');
%M = 9; %input('Numero de neuronas de salida: ');
%alfa = 0.0025; %input('Valor de alfa: ');
%min = 0.015;    %input('Minimo error: ');
%iter = 100; %input('Numero de iteraciones: ');
%patrones = 9000;   %input('Numero de patrones: ');

% INICIALIZACION DE LA MATRIZ DE PESOS W Y V
Wh = 0.2*rand(N+1,L)-.1;%0.2
Wo = 0.2*rand(L+1,M)-.1;%0.2
Wh_ji = Wh';
Wo_kj = Wo';
% Agregando el Bias a la entrada
B1 = [P1];
B = B1';
PB1=[];
for g=1:patrones
    PB1(g,:)=[1,B(g,:)];
end
PB_e = PB1';
PB = PB_e;
T1_e =T1;
T =T1_e;
% INICIA ENTRENAMIENTO

epoch = 1;
cont=0;
Z=0;

for kp = 1:iter
    %fprintf('\n ---- EPOCH: %d ---- \n',epoch);
    cont=cont+1;
    error=0;
    for ip = 1:patrones
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        d_k = E;
        jes_1 = d_k.*((1.+Uo_k).*(1.-Uo_k));
        delta1 = (-1).*jes_1*Uh_j;
        Wo_kj = Wo_kj - alfa.*delta1; % SE CORRIGEN LOS PESOS DE LA
MATRIZ DE LA CAPA DE SALIDA
```

```
        a=jes_1'*Wo_kj; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA
CAPA DE ENTRADA
        jes_2 = (-1).*a.*((1.+Uh_j).*(1.-Uh_j));
        jes_2 = jes_2(2:length(jes_2));
        delta2 = jes_2'*Xi';
        Wh_ji = Wh_ji -alfa.*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ
DE LA CAPA DE ENTRADA
        error = error + sum((d_k.*d_k)/M); %%%%% CALCULO DE ERRORRES
        %e = d_k.*d_k;
        %Error_p = (sum (e));
        %Error_mc (ip) = (Error_p'/M);
    end
    error = error/patrones; %%%% ERROR POR EPOCA
    vec_err(1,kp) = error;
    %fprintf(1,'Error = %4.9f ==> %d\n',error,kp);
    J(1,kp)=error;
    if error<min
        break;
    end
    %error;
end

figure(1);
plot(J);
xlabel('Epocas');
ylabel('Error Cuadratico Promedio ');
grid on;

save Wh_ji.tx Wh_ji -ascii
save Wo_kj.tx Wo_kj -ascii
save error J;
```

Función de simulación: sim_j

```
function [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T)
Xi = PB(:,ip);     %%%%%%% INICIA VECTOR DE ENTRADA A LA RED
Vh_j = (Wh_ji*Xi); %%%%%%% PROPAGACION HACIA ADELANTE DE LOS PESOS
Uh =  (2./(exp((-2).*Vh_j)+1))-1;%((exp((2).*Vh_j)-
1))./(exp((2).*Vh_j)+1);
%Uh = tansig(Vh_j); %%%%%%% SE UTILIZA LA FUNCION DE TRANSFERENCIA
Uh_j = [1.0,Uh'];
Vo_k = Wo_kj*Uh_j'; %%%%%%%%%
Uo_k = (2./(exp((-2).*Vo_k)+1))-1;
%Uo_k = logsig(Wo_kj*Uh_j'); % SE UTILIZA LA FUNCION DE TRANSFERENCIA
E= T(:,ip)-Uo_k; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA
DE SALIDA
```

Versión 2.1 del algoritmo 2

```
clear;
clc;
close;
tic;
%   Trustworthiness "neural networks"
% Version 00
%   - 12 methods
% Version 01
%   - Early Stopping Option
% Version 02
%   - Multiple classes
%
% Defects:     Gc, EFFc, Gt, EFFt, Gpt, EFFpt, SIGcc, EFFcc, SIGin
% Parameters:  Pc, Pt,   Tc, Tt,   Tpt, Gf
% Regimes
n_hp_corr=10700,10600,10500,10400,10300,10200,10100,10000,9900,9800,9700
% Defect development 0-5% (idl=1-21)
clear;
krg=11;   % regime quantity
nreg=1;
kdf=9;    % class  quantity
kdl=21;   % interpolation points quantity for every class development
kdf0=9;    % maximal classes  number
kdf1=9;   % single classes  number
idef2=0; kdf2=4;                              % multiple classes signal and
number
kpr0=6;    % parameter quantity
ktd=1000; % class simulation points quantity
nr1=0; nr2=1;    % random generators initiation
DY=[0.015 0.015 0.025 0.015 0.020 0.020];   % parameter maximal errors
krdd=krg*kdf0*kdl;
i_EarlyStop=0; %Early Stopping Option

%                              LEARNING SAMPLES
% Reading from the file
fid=fopen('dpar.dat','rt');
BT=fscanf(fid,'%g',[kpr0,krdd]);
BB=BT';
for ipr=1:kpr0
   BB(:,ipr)=BB(:,ipr)/DY(ipr);
end;
% Formation of the multidimensional array of defect induced deviations
for irg=1:krg
   for idf=1:kdf0
      for idl=1:kdl
         irdd=(irg-1)*kdf0*kdl+(idf-1)*kdl+idl;
         B(:,idl,idf,irg)=BB(irdd,:);
      end;
   end;
end;
% Regime determination
D=B(:,:,:,nreg);
```

```
% Teaching sample
ZE=ones(kpr0,1);
ZD_1=ones(kpr0,1);
ZD_2=ones(kpr0,1);
rand('state',nr1);  randn('state',nr1);                %State fixing of
generators

if idef2==0
   kdf=kdf1;
else
   kdf=kdf2;
end;
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD1(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD1(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
      end;
   end;
end;

% Test sample points)
rand('state',nr2);  randn('state',nr2);                %State fixing of
generators
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      %dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD2(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD2(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
      end;
```

```
   end;
end;

for idf = 1:kdf
    for itd =1:ktd
        irdd = (idf-1)*ktd+itd;
        ZD1r (:,irdd)= ZD1(:,itd,idf);
    end
end

for idf = 1:kdf
    for itd =1:ktd
        irdd = (idf-1)*ktd+itd;
        ZD2r (:,irdd)= ZD2(:,itd,idf);
    end
end
save ZD1r.tx ZD1r -ascii
save ZD2r.tx ZD2r -ascii
%%%%%%%%%%%% $$$$$$$$$$$$$$$$$$$$

for lp = 1:1000
    for mp = 1:9
        n = (lp-1)* kdf+mp;
        P1 (:,n) = ZD1(:,lp,mp);
        T1 (mp,n) = 1;
    end
end

for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P1_s(:,n)=ZD1(:,itd,idf);
      T1_s(idf,n)=1;
   end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%

% VALORES DE ENTRADA
N = 6;  %input('Numero de neuronas de entrada: ');
L = 12; %input('Numero de neuronas ocultas: ');
M = 9; %input('Numero de neuronas de salida: ');
alfa = 0.0025; %input('Valor de alfa: ');
min = 0.015;    %input('Minimo error: ');
iter = 1000; %input('Numero de iteraciones: ');
patrones = 9000;   %input('Numero de patrones: ');

[matriz_Wh_ji, matriz_Wo_kj] = ent_bpj (N, L, M, alfa, min, iter,
patrones, P1, T1)

% Formation of net entries
for idf=1:kdf
```

```
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P1_s(:,n)=ZD1(:,itd,idf);
      T1_s(idf,n)=1;
   end;
end;
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P2_s(:,n)=ZD2(:,itd,idf);
      T2_s(idf,n)=1;
   end;
end;

Wh_ji= matriz_Wh_ji;
Wo_kj= matriz_Wo_kj;
%load pesos2 Wh_ji Wo_kj;

porcentaje=[];

% Agregando el Bias a la entrada
B1_1 = [P1_s];
B_1 = B1_1';
PB_1=[];
for g = 1:patrones
    PB11_1(g,:) = [1,B_1(g,:)];
end
PB1_1 = PB11_1';
PB = PB1_1;
T = T1_s;
aux_1 = 0;

for ip = 1:patrones
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_1 = Uo_k;
    A1 (:,ip)= Uo_k_1';
end

 % Agregando el Bias a la entrada
B1_2 = [P2_s];
B_2 = B1_2';
PB_2=[];
for g = 1:patrones
    PB11_2(g,:) = [1,B_2(g,:)];
end
PB1_2 = PB11_2';
PB = PB1_2;
T_2 = T2_s;
T = T2_s;
aux_2 = 0;

for ip = 1:patrones
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_2 = Uo_k;
    A2 (:,ip)= Uo_k_2';
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                              TRUSTWORTHINESS CALCULATION
PD1=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A1(:,n));
      PD1(imax,idf)=PD1(imax,idf)+1;
   end;
end;
PD1=PD1/ktd;
PDT1=diag(PD1);
psr1=mean(PDT1);
PD1T=PD1';
%fw1=fopen('c:\_igor\matlab\Work\pd1.dat','wt');
fw1=fopen('pd1.dat','wt');
if idef2==0
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD1T,PDT1,psr1);
else
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f\n',PD1T,PDT1,psr1);
end;
fclose(fw1);

PD2=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A2(:,n));
      PD2(imax,idf)=PD2(imax,idf)+1;
   end;
end;
PD2=PD2/ktd;
PDT2=diag(PD2);
psr2=mean(PDT2);
PD2T=PD2';
fw2=fopen('pd2.dat','wt');
if idef2==0
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD2T,PDT2,psr2);
else
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f\n',PD2T,PDT2,psr2);
end;
fclose(fw2);

figure(2);
plot3(ZD1(1,:,1),ZD1(2,:,1),ZD1(3,:,1),'*',ZD1(1,:,2),ZD1(2,:,2),ZD1(3,:,
2),'.',ZD1(1,:,3),ZD1(2,:,3),ZD1(3,:,3),'+',ZD1(1,:,4),ZD1(2,:,4),ZD1(3,:
,4),'x'); grid on; legend('D1','D2','D3','D4');
xlabel('Z1 - Compressor pressure');
ylabel('Z2 - Turbine pressure');
zlabel('Z3 - Compressor temperature');
```

```
save pesos PD1T PD2T psr1 psr2;
toc;
```

Función de entrenamiento: ent_bpj

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%%
function [Wh_ji, Wo_kj] = ent_bpj (N, L, M, alfa, min, iter, patrones,
P1, T1)
% VALORES DE ENTRADA
%N = 6;  %input('Numero de neuronas de entrada: ');
%L = 12; %input('Numero de neuronas ocultas: ');
%M = 9; %input('Numero de neuronas de salida: ');
%alfa = 0.0025; %input('Valor de alfa: ');
%min = 0.015;    %input('Minimo error: ');
%iter = 100; %input('Numero de iteraciones: ');
%patrones = 9000;   %input('Numero de patrones: ');

% INICIALIZACION DE LA MATRIZ DE PESOS W Y V
Wh = 0.2*rand(N+1,L)-.1;%0.2
Wo = 0.2*rand(L+1,M)-.1;%0.2
Wh_ji = Wh';
Wo_kj = Wo';

save Wh_ji.tx Wh_ji -ascii
save Wo_kj.tx Wo_kj -ascii
% Agregando el Bias a la entrada
B1 = [P1];
B = B1';
PB1=[];
for g=1:patrones
    PB1(g,:)=[1,B(g,:)];
end
PB_e = PB1';
PB = PB_e;
T1_e =T1;
T =T1_e;
% INICIA ENTRENAMIENTO

epoch = 1;
cont=0;
Z=0;

for kp = 1:iter
    %fprintf('\n ---- EPOCH: %d ---- \n',epoch);
    cont=cont+1;
    error=0;
    for ip = 1:patrones
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        d_k = E;
        jes_1 = (d_k.*((1-Uo_k).*Uo_k));
        delta1 = (-1).*jes_1*Uh_j;
```

```
        Wo_kj = Wo_kj - alfa.*delta1; % SE CORRIGEN LOS PESOS DE LA
MATRIZ DE LA CAPA DE SALIDA
        a=jes_1'*Wo_kj; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA
CAPA DE ENTRADA
        jes_2 = (-1).*a.*((1.+Uh_j).*(1.-Uh_j));
        jes_2 = jes_2(2:length(jes_2));
        delta2 = jes_2'*Xi';
        Wh_ji = Wh_ji -alfa.*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ
DE LA CAPA DE ENTRADA
        error = error + sum((d_k.*d_k)/M); %%%%% CALCULO DE ERRORRES
        %e = d_k.*d_k;
        %Error_p = (sum (e));
        %Error_mc (ip) = (Error_p'/M);
    end
    error = error/patrones; %%%% ERROR POR EPOCA
    vec_err(1,kp) = error;
    %fprintf(1,'Error = %4.9f ==> %d\n',error,kp);
    J(1,kp)=error;
    if error<min
        break;
    end
    %error;
end

figure(1);
plot(J);
xlabel('Epocas');
ylabel('Error Promedio Cuadratico');
grid on;

save Wh_ji.tx Wh_ji -ascii
save Wo_kj.tx Wo_kj -ascii
save error J;
```

Función de simulación: sim_j

```
function [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T)
Xi = PB(:,ip);      %%%%%%% INICIA VECTOR DE ENTRADA A LA RED
Vh_j = (Wh_ji*Xi); %%%%%%% PROPAGACION HACIA ADELANTE DE LOS PESOS
Uh =  (1-(exp((-2).*Vh_j))./(1+(exp((-2).*Vh_j)));%(2./(exp((-
2).*Vh_j)+1))-1;
%Uh = tansig(Vh_j); %%%%%%% SE UTILIZA LA FUNCION DE TRANSFERENCIA
Uh_j = [1.0,Uh'];
Vo_k = Wo_kj*Uh_j'; %%%%%%%%%
Uo_k = (1)./(exp(-Vo_k)+1); %
%Uo_k = logsig(Wo_kj*Uh_j'); % SE UTILIZA LA FUNCION DE TRANSFERENCIA
E= T(:,ip)-Uo_k; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA
DE SALIDA
```

Versión 2.2 del algoritmo 2

```
clear;
clc;
close;
tic;
%   Trustworthiness "neural networks"
% Version 00
%   - 12 methods
% Version 01
%   - Early Stopping Option
% Version 02
%   - Multiple classes
%
% Defects:     Gc, EFFc, Gt, EFFt, Gpt, EFFpt, SIGcc, EFFcc, SIGin
% Parameters:  Pc, Pt,   Tc, Tt,   Tpt, Gf
% Regimes
n_hp_corr=10700,10600,10500,10400,10300,10200,10100,10000,9900,9800,9700
% Defect development 0-5% (idl=1-21)
clear;
krg=11;   % regime quantity
nreg=1;
kdf=9;    % class  quantity
kdl=21;   % interpolation points quantity for every class development
kdf0=9;    % maximal classes  number
kdf1=9;   % single classes  number
idef2=0; kdf2=4;                              % multiple classes signal and
number
kpr0=6;    % parameter quantity
ktd=1000; % class simulation points quantity
nr1=0; nr2=1;    % random generators initiation
DY=[0.015 0.015 0.025 0.015 0.020 0.020];   % parameter maximal errors
krdd=krg*kdf0*kdl;
i_EarlyStop=0; %Early Stopping Option

%                              LEARNING SAMPLES
% Reading from the file
fid=fopen('dpar.dat','rt');
BT=fscanf(fid,'%g',[kpr0,krdd]);
BB=BT';
for ipr=1:kpr0
   BB(:,ipr)=BB(:,ipr)/DY(ipr);
end;
% Formation of the multidimensional array of defect induced deviations
for irg=1:krg
   for idf=1:kdf0
      for idl=1:kdl
         irdd=(irg-1)*kdf0*kdl+(idf-1)*kdl+idl;
         B(:,idl,idf,irg)=BB(irdd,:);
      end;
   end;
end;
% Regime determination
D=B(:,:,:,nreg);
```

```
% Teaching sample
ZE=ones(kpr0,1);
ZD_1=ones(kpr0,1);
ZD_2=ones(kpr0,1);
rand('state',nr1);  randn('state',nr1);              %State fixing of
generators

if idef2==0
   kdf=kdf1;
else
   kdf=kdf2;
end;
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD1(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD1(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
      end;
   end;
end;

% Test sample points)
rand('state',nr2);  randn('state',nr2);              %State fixing of
generators
for idf=1:kdf
   for itd=1:ktd
      ZE=randn(size(ZE))/3.0;
      %dzn=randn/3.0;
      if idef2==0
         ld=rand*(kdl-1);  ii=floor(ld)+1; alfa=ld-ii+1;
         ZD2(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
      else
         idf1=2*idf-1; idf2=2*idf;
         ld1=rand*(kdl-1); ld2=rand*(kdl-1);
         if (ld1+ld2)>(kdl-1)
            ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
         end;
         ii1=floor(ld1)+1; alfa1=ld1-ii1+1;  ii2=floor(ld2)+1; alfa2=ld2-
ii2+1;
         ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
         ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
         ZD2(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
```

```
      end;
   end;
end;


%%%%%%%%%%%% $$$$$$$$$$$$$$$$$$$$

for lp = 1:1000
    for mp = 1:9
        n = (lp-1)* kdf+mp;
        P1 (:,n) = ZD1(:,lp,mp);
        T1 (mp,n) = 1;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%

% VALORES DE ENTRADA
N = 6;  %input('Numero de neuronas de entrada: ');
L = 12; %input('Numero de neuronas ocultas: ');
M = 9; %input('Numero de neuronas de salida: ');
alfa = 0.0025; %input('Valor de alfa: ');
min = 0.015;    %input('Minimo error: ');
iter = 5; %input('Numero de iteraciones: ');
patrones = 9000;   %input('Numero de patrones: ');

[matriz_Wh_ji, matriz_Wo_kj] = ent_bpj (N, L, M, alfa, min, iter,
patrones, P1, T1)

% Formation of net entries
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P1_s(:,n)=ZD1(:,itd,idf);
      T1_s(idf,n)=1;
   end;
end;
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      P2_s(:,n)=ZD2(:,itd,idf);
      T2_s(idf,n)=1;
   end;
end;

load pesos2 Wh_ji Wo_kj;

porcentaje=[];

% Agregando el Bias a la entrada
B1_1 = [P1_s];
B_1 = B1_1';
```

```
PB_1=[];
for g = 1:patrones
    PB11_1(g,:) = [1,B_1(g,:)];
end
PB1_1 = PB11_1';
PB = PB1_1;
T = T1_s;
aux_1 = 0;

for ip = 1:patrones
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_1 = Uo_k;
    A1 (:,ip)= Uo_k_1';
end

 % Agregando el Bias a la entrada
B1_2 = [P2_s];
B_2 = B1_2';
PB_2=[];
for g = 1:patrones
    PB11_2(g,:) = [1,B_2(g,:)];
end
PB1_2 = PB11_2';
PB = PB1_2;
T_2 = T2_s;
T = T2_s;
aux_2 = 0;

for ip = 1:patrones
    [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
    Uo_k_2 = Uo_k;
    A2 (:,ip)= Uo_k_2';
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                              TRUSTWORTHINESS CALCULATION
PD1=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A1(:,n));
      PD1(imax,idf)=PD1(imax,idf)+1;
   end;
end;
PD1=PD1/ktd;
PDT1=diag(PD1);
psr1=mean(PDT1);
PD1T=PD1';
%fw1=fopen('c:\_igor\matlab\Work\pd1.dat','wt');
fw1=fopen('pd1.dat','wt');
if idef2==0
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD1T,PDT1,psr1);
```

```
else
   count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f\n',PD1T,PDT1,psr1);
end;
fclose(fw1);

PD2=zeros(kdf);
for idf=1:kdf
   for itd=1:ktd
      n=(idf-1)*ktd+itd;
      [amax,imax]=max(A2(:,n));
      PD2(imax,idf)=PD2(imax,idf)+1;
   end;
end;
PD2=PD2/ktd;
PDT2=diag(PD2);
psr2=mean(PDT2);
PD2T=PD2';
fw2=fopen('pd2.dat','wt');
if idef2==0
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD2T,PDT2,psr2);
else
   count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f\n',PD2T,PDT2,psr2);
end;
fclose(fw2);

figure(2);
plot3(ZD1(1,:,1),ZD1(2,:,1),ZD1(3,:,1),'*',ZD1(1,:,2),ZD1(2,:,2),ZD1(3,:,
2),'.',ZD1(1,:,3),ZD1(2,:,3),ZD1(3,:,3),'+',ZD1(1,:,4),ZD1(2,:,4),ZD1(3,:
,4),'x'); grid on; legend('D1','D2','D3','D4');
xlabel('Z1 - Compressor pressure');
ylabel('Z2 - Turbine pressure');
zlabel('Z3 - Compressor temperature');

save pesos PD1T PD2T psr1 psr2;
toc;
```

Función de entrenamiento: ent_bpj

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%  RED BACKPROPAGATION  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%     ENTRENAMIENTO     %%%%%%%%%%%%%%%%%%%%%%
function [Wh_ji, Wo_kj] = ent_bpj (N, L, M, alfa, min, iter, patrones,
P1, T1)
% VALORES DE ENTRADA
%N = 6;  %input('Numero de neuronas de entrada: ');
%L = 12; %input('Numero de neuronas ocultas: ');
%M = 9; %input('Numero de neuronas de salida: ');
%alfa = 0.0025; %input('Valor de alfa: ');
%min = 0.015;    %input('Minimo error: ');
%iter = 100; %input('Numero de iteraciones: ');
%patrones = 9000;   %input('Numero de patrones: ');
```

```
% INICIALIZACION DE LA MATRIZ DE PESOS W Y V
Wh = 0.2*rand(N+1,L)-.1;%0.2
Wo = 0.2*rand(L+1,M)-.1;%0.2
Wh_ji = Wh';
Wo_kj = Wo';
% Agregando el Bias a la entrada
B1 = [P1];
B = B1';
PB1=[];
for g=1:patrones
    PB1(g,:)=[1,B(g,:)];
end
PB_e = PB1';
PB = PB_e;
T1_e =T1;
T =T1_e;
% INICIA ENTRENAMIENTO

epoch = 1;
cont=0;
Z=0;

for kp = 1:iter
    %fprintf('\n ---- EPOCH: %d ---- \n',epoch);
    cont=cont+1;
    error=0;
    for ip = 1:patrones
        [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
        e = E.*E;
        d_k = (1/2)*(sum(e));
        d_p0 = d_k;
        %Wh_ji = (Wh_ji);
        %Wo_kj = (Wo_kj);
        AWo_kj = (1).*(abs (.01.*Wo_kj));
        AWh_ji = (1).*(abs (.01.*Wh_ji));

        Wo_0=Wo_kj;
        for k = 1:M
            for j = 1:L+1
                Wo_kj(k,j) = Wo_0(k,j) + AWo_kj(k,j);
                [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
                e = E.*E;
                d_p1 = (1/2)*(sum(e));
                delta1(k,j) = (d_p1-d_p0)/AWo_kj(k,j);
                Wo_kj(k,j) = Wo_0(k,j);
            end
        end

        Wo_kj=Wo_0;
        Wh_0=Wh_ji;
        for j = 1:L
            for i = 1:N+1
                Wh_ji(j,i) = Wh_0(j,i) + AWh_ji(j,i);
                [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T);
                e = E.*E;
```

```
                d_p1 = (1/2)*(sum(e));
                delta2(j,i) = (d_p1-d_p0)/AWh_ji(j,i);
                Wh_ji(j,i) = Wh_0(j,i);
            end
        end
        Wh_ji=Wh_0;
        Wo_kj = Wo_kj - alfa*delta1; % SE CORRIGEN LOS PESOS DE LA MATRIZ
DE LA CAPA DE SALIDA
        %delta2 = ((d_k.*(Wh_ji+AWh_ji))-(d_k.*AWh_ji))./AWh_ji;%%%%% SE
INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA DE ENTRADA
        %delta2=delta2(2:length(delta2));
        Wh_ji = Wh_ji - alfa*delta2; % SE CORRIGEN LOS PESOS DE LA MATRIZ
DE LA CAPA DE ENTRADA
        error = error + sum((E.*E)/M); %%%%% CALCULO DE ERRORRES
        %e = d_k.*d_k;
        %Error_p = (sum (e));
        %Error_mc (ip) = (Error_p'/M);
    end
    error = error/patrones; %%%% ERROR POR EPOCA
    vec_err(1,kp) = error;
    %fprintf(1,'Error = %4.9f ==> %d\n',error,kp);
    J(1,kp)=error;
    if error<min
        break;
    end
    %error;
end

figure(1);
plot(J);
xlabel('Epocas');
ylabel('Error Medio Cuadratico');
grid on;

save Wh_ji.txt Wh_ji -ascii
save Wo_kj.txt Wo_kj -ascii
save error J
```

Funcion de simulación: sim_j

```
function [E, Uo_k, Uh_j, Xi] = sim_j (PB, ip, Wh_ji, Wo_kj, T)
Xi = PB(:,ip);     %%%%%%% INICIA VECTOR DE ENTRADA A LA RED
Vh_j = (Wh_ji*Xi); %%%%%%% PROPAGACION HACIA ADELANTE DE LOS PESOS
Uh =  (1-(exp((-2).*Vh_j))./(1+(exp((-2).*Vh_j)));%(2./(exp((-
2).*Vh_j)+1))-1;
%Uh = tansig(Vh_j); %%%%%%% SE UTILIZA LA FUNCION DE TRANSFERENCIA
Uh_j = [1.0,Uh'];
Vo_k = Wo_kj*Uh_j'; %%%%%%%%%
Uo_k = (1)./(exp(-Vo_k)+1); %
%Uo_k = logsig(Wo_kj*Uh_j'); % SE UTILIZA LA FUNCION DE TRANSFERENCIA
E= T(:,ip)-Uo_k; %%%%% SE INICIA RETRO-PROPAGACION DEL ERROR EN LA CAPA
DE SALIDA
```

Version 2.3 del Algoritmo 2 (Fortran)

```
program prueba1
implicit none
real time1
real time2,R(6)
integer ::krg,
nreg,kdf,kdl,kdf0,kdf1,idef2,kdf2,kpr0,ktd,nr1,nr2,Krdd,i_Erlystop
integer :: nn,i,x,y, irg,idf,idl,
irdd,itd,idff,n,NE,LE,ME,iter,patrones,ip,je,rc,kpE,rc1,ixt
integer (4) ZZZ1,nT,k,NR
INTEGER, ALLOCATABLE :: imax(:)
double precision DY (1,6),BT(6,2079), BB(2079,6),
B(6,21,9,11),D(6,21,9,1),ZE(6,1), ZD_1(6,1), ZD_2(6,1),ld,ii,alfa,QJ!
double precision ZD1(6,1000,9),
ZD2(6,1000,9),P1(6,9000),T1(9,9000),alfaE,Whji(12,7),Wokj(9,13),Z1(7,9000
),B1(7,9000)
double precision Xi(7,1),Vnj(12,7),Vnjv(12,1),
Z2(13,1),Uok(9,13),Uokv(9,1),de(9,1),E(9,1),delta1(9,13),aE(9,13),asE(1,1
3),jes1(1,12),delta2(12,7)
double precision
EE(9,1),EES,vec_err(1,1000),P1S(6,9000),T1S(9,9000),B1S(7,9000),A1(9,9000
),P2S(6,9000),T2S(9,9000),B2S(7,9000),A2(9,9000)
double precision
amax,A1S(9,1),PDT1(9,1),psr1,PD1T(9,9),A2S(9,1),PDT2(9,1),psr2,PD2T(9,9),
t,ZD1r(9000,6),ZD2r(9000,6),ZD1b(6,9000),ZD2b(6,9000)
real PD1(9,9), PD2(9,9)
double precision Whjir(7,12),Wokjr(13,9),py ,ERR
!external ::GAUSS !RNNOA

call cpu_time (time1)
! Defects: Gc, EFFc, Gpt, EFFpt, SIGcc, SIGin
! Parameters: Pc, Pt, Tc, Tt, Tpt, Gf
! Regimes n_hp_corr = 10700, 10600, 10500, 10400, 10300, 10200, 10100,
10000, 9900, 9800, 9700
! Defect development 0-5% (idñ = 1-21
krg = 11    ! regime quantity
nreg = 1
kdf = 9           ! class quantity
kdl = 21    ! interpolation points quantity for every class development
kdf0 = 9    ! maximal classes number
kdf1 = 9    ! single classes number
idef2 = 0; kdf2 = 4          ! multiple classes signal and number
kpr0 = 6    ! parameter quantity
ktd=1000    ! class simulationpoints
nr1 = 0; nr2 = 1  ! random generators indication
Krdd = krg*kdf0*kdl
!real DY (nreg,kpr0),BT(kpr0,krdd), BB(krdd,kpr0),
B(kpr0,kdl,kdf,krg),D(kpr0,kdl,kdf,nreg)
!real ZE(kpr0,1)
DY = RESHAPE ((/.015, .015, .025, .015, .020, .020/),(/1,6/))
! parameter maximal errors

i_Erlystop = 0 ! Early Stop Option

open (unit=9, file='DATENT.sal', action='write')
```

```
write (9,*) 'krg:',krg
write (9,*) 'nreg:',nreg
write (9,*) 'kdf:',kdf
write (9,*) 'kdl:',kdl
write (9,*) 'kdf0:',kdf0
write (9,*) 'kpr0:',kpr0
write (9,*) 'ktd:',ktd
write (9,*) 'DY:',DY
write (9,*) 'Krdd:',Krdd
close(9)

open (unit=8,file='DPAR.dat',action='read') !Abriendo el archivo de datos
de entrada
read (8,*) BT     ! Leyendo los datos de entrada
close (8)
!write (*,*) BT

BB=transpose(BT)

do nn =1,kpr0
      do i = 1,krdd
      !BB(i,nn)= BT(nn,i)
      BB(i,nn)= BB(i,nn)/DY(1,nn) ! Dividiendo entre el error maximo
permitido
      end do
end do

do irg =1,krg
      do idf = 1,kdf0
            do idl = 1,kdl
                  do nn =1,kpr0
                  irdd= (irg-1)*kdf0*kdl+(idf-1)*kdl+idl
                  B(nn,idl,idf,irg)=BB(irdd,nn)
                  end do
            end do
      end do
end do
open (unit=9, file='B.sal', action='write');write (9,*) B;close(9)

!Determinacion del Regimen
do irg =1,nreg
      do idf = 1,kdf0
            do idl = 1,kdl
                  do nn =1,kpr0
                  D(nn,idl,idf,irg)=B(nn,idl,idf,irg)
                  end do
            end do
      end do
end do

open (unit=9, file='D.sal', action='write');write (9,*) D;close(9)

!Entrenamiento
ZE= (1)!reshape((/1,1,1,1,1,1/),(/6,1/))
ZD_1= (1) !reshape((/1,1,1,1,1,1/),(/6,1/))
ZD_2 = (1) !reshape((/1,1,1,1,1,1/),(/6,1/))
```

```
!write (*,*) ZE;write (*,*) ZD_1;write (*,*)ZD_2

!LEARNING SAMPLES
k=1

do irg = 1,nreg
      do idff = 1,kdf
            do itd =1,ktd
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          do py = 1,6
              call gauss (ERR)
              ZE(py,1)=ERR
          end do
          ZE=ZE/3

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          !call random_number(ZE);  ZE = ZE/3                !do   nn
=1,kpr0     !ZE(nn,1) = ZE(nn,1)/3!print *,'ZE ', ZE !end do
               call random_number(ld)!call random (ld);
               ld = ld*(kdl-1)!print *,'ld ', ld
               ii=floor(ld)+1
               alfa=ld-ii+1;!print *,'ii ', ii;print *,'alfa ', alfa
                 do nn =1,kpr0
                       ZD1(nn,itd,idff)=D(nn,ii,idff,irg)*(1-
alfa)+D(nn,ii+1,idff,irg)*alfa+ZE(nn,1)
                 end do
            end do
      end do
end do
!call faglStartWatch (ZD1, ZZZ1)
open (unit=9, file='ZD1.sal', action='write');write (9,*) ZD1;close(9)

!TEST SAMPLES POINTS

do irg = 1,nreg
      do idff = 1,kdf
            do itd =1,ktd
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          do py = 1,6
              call gauss (ERR)
              ZE(py,1)=ERR
          end do
          ZE=ZE/3
          !call gauss (ERR)
          !ld=ABS(ERR)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
               !call random_number(ZE);  ZE = ZE/3 !print *,'ZE ', ZE
               call random_number(ld);
               ld = ld*(kdl-1);!print *,'ld ', ld
               ii=floor(ld)+1;alfa=ld-ii+1;!print *,'ii ', ii;print
*,'alfa ', alfa
                 do nn =1,kpr0
                       ZD2(nn,itd,idff)=D(nn,ii,idff,irg)*(1-
alfa)+D(nn,ii+1,idff,irg)*alfa+ZE(nn,1)
                 end do
            end do
```

```
      end do
end do
open (unit=9, file='ZD2.sal', action='write');write (9,*) ZD2;close(9)


!open (unit=8,file='ZD1r.tx',action='read');read (8,*) ZD1r;close (8)
!open (unit=8,file='ZD2r.tx',action='read');read (8,*) ZD2r;close (8)

!ZD1b=transpose(ZD1r)
!ZD2b=transpose(ZD2r)

!do idff = 1,kdf
!     do itd =1,ktd
!                irdd = (idff-1)*ktd + itd
!           ZD1(:,itd,idff)=ZD1b(:,irdd)
!     end do
!end do


!do idff = 1,kdf
!     do itd =1,ktd
!           irdd = (idff-1)*ktd + itd
!           ZD2(:,itd,idff)=ZD2b(:,irdd)
!     end do
!end do
!open (unit=9, file='ZD1.sal', action='write');write (9,*) ZD1;close(9)
!open (unit=9, file='ZD2.sal', action='write');write (9,*) ZD2;close(9)

!FORMATION OF NET ENTRIES


do itd =1,ktd
      do  idff = 1,kdf
            n=(itd-1)*kdf+idff
            P1(:,n)=ZD1(:,itd,idff)
            T1(idff,n)=1
      end do
end do


open (unit=9, file='P1.sal', action='write');write (9,*) P1;close(9)
open (unit=9, file='T1.sal', action='write');write (9,*) T1;close(9)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!RED BACKPROPAGATION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!VALORES DE ENTRADA

NE = 6      ! NUMERO DE NEUROPNAS DE ENTRADA
LE = 12     ! NUMERO DE NEURONAS OCULTAS
ME = 9      ! NUMERO DE NEURONAS DE SALIDA
alfaE = 0.0025    ! RELACION DE MODIFICACION DE PESOS
iter = 1000 ! NUMERO DE ITERACIONES O EPOCAS
patrones = 9000   ! NUMERO DE PATRONES
```

```
! MATRICES DE PESOS W
call random_number(Whji);     call random_number(Wokj);
Whji = (Whji-1)*.2; Wokj = (Wokj -1)*0.2
open (unit=9, file='Whji.sal', action='write');write (9,*) Whji;close(9)
open (unit=9, file='Wokj.sal', action='write');write (9,*) Wokj;close(9)

!print *, Whji !(7,12)
!print *, Wokj !(13,9)=


!open (unit=8,file='Wh_ji.tx',action='read');read (8,*) Whjir;close (8)
!open (unit=8,file='Wo_kj.tx',action='read');read (8,*) Wokjr;close (8)

!Whji=transpose(Whjir)
!Wokj=transpose(Wokjr)

!AGREGANDO EL BIAS A LA ENTRADA
Z1=(1)!(SIZE(Z1,1))-6
B1=Z1
!open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)

do ip = 1,patrones
      do nn = 1,NE
      B1(nn+1,ip)= P1(nn,ip)
      end do
end do
open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do kpE =1,iter

      EES=0

      do ip = 1,patrones
            !do nn = 1,NE+1         !estableciendo el vector de entrada
                  Xi(:,1)=B1(:,ip)
            !end do

            do je = 1,LE            !pasando el vector de entrada por la
capa oculta
                  !do nn = 1,NE+1
                  Vnj(je,:)= Whji(je,:)*Xi(:,1)
                  !end do
            end do

            do je = 1,LE            !Vector de salida de la capa oculta
                  Vnjv(je,1)= sum (Vnj(je,:))
            end do

            !do je = 1,LE           ! Pasando por la funcion de activacion
                  Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1  !((exp(-
2*(Vnjv(je,1))))-1)/((exp(-2*(Vnjv(je,1))))+1)
            !end do
```

```
            Z2=(1)!(SIZE(Z2))-12     !agregando el vias

            do je = 1,LE
                  Z2(je+1,1)= Vnjv(je,1)
            end do

            do rc = 1,ME
                  !do je = 1,LE+1
                        Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
                  !end do
            end do

            do rc = 1,ME            !Vector de salida de la capa oculta
                  Uokv(rc,1)= sum (Uok(rc,:))
            end do

            !do rc = 1,ME           ! Pasando por la funcion de activacion
                  Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1)
            !end do

            !do rc = 1,ME           !Establecienmdo el error desceado
                  de(:,1)=T1(:,ip)
            !end do

            !do rc = 1,ME           !Error de salida de la red
                  E(:,1)= de(:,1)-Uokv(:,1)
            !end do

            do je = 1,LE+1
                  !do rc = 1,ME
                        delta1 (:,je)= (-1)*(E(:,1)*((1-
Uokv(:,1))*Uokv(:,1)))*(Z2(je,1))
                  !end do
            end do
            open (unit=9, file='delta1.sal', action='write');write (9,*)
delta1;close(9)


            do je = 1,LE+1
                  !do rc = 1,ME
                        delta1 (:,je)= (alfaE)*delta1(:,je)
                  !end do
            end do

            Wokj = Wokj-delta1            !Sec corrigen los pesos de la
matriz de la capa de salida

            do je = 1,LE+1
                  !do rc = 1,ME
                        aE (:,je)= (E(:,1)*((1-
Uokv(:,1))*Uokv(:,1)))*Wokj(:,je)
                  !end do
            end do

            do je = 1,LE+1
                  asE(1,je)=sum(aE(:,je))
            end do
```

```
            do je = 1,LE+1
                  asE(1,je)=(-1)*(asE(1,je))*((1+(Z2(je,1)))*(1-
(Z2(je,1))))
            end do

            do je = 1,LE
                  jes1(1,je) =asE(1,je+1)
            end do

            do je = 1,LE
                  !do nn = 1,NE+1
                        delta2 (je,:)= jes1(1,je)* Xi(:,1)
                  !end do
            end do
            open (unit=9, file='delta2.sal', action='write');write (9,*)
delta2;close(9)

            do je = 1,LE
                  !do nn = 1,NE+1
                        delta2 (je,:)= alfaE* delta2(je,:)
                  !end do
            end do

            Whji = Whji - delta2

            !do rc = 1,ME           !Calculo del Error cuadratico promedio
                  EE(:,1)= (E(:,1))**2
            !end do

            EES = EES+(sum(EE))/ME

      end do
      EES = EES/patrones

      vec_err(1,kpE) = EES
      open (unit=9, file='vec_err.sal', action='write');write (9,*)
vec_err;close(9)

end do

!call LoadPicture(vsc_err)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!SIMULACION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!CONJUNTO DE ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!!

do idff = 1,kdf
      do itd =1,ktd
            !do nn =1,kpr0
            n=(idff-1)*ktd+itd
            P1S(:,n)=ZD1(:,itd,idff)
            T1S(idff,n)=1
            !end do
```

```
      end do
end do

open (unit=9, file='P1S.sal', action='write');write (9,*) P1S;close(9)
open (unit=9, file='T1S.sal', action='write');write (9,*) T1S;close(9)

      !AGREGANDO EL BIAS A LA ENTRADA
Z1=(1) !(SIZE(Z1,1))-6
B1S=Z1
!open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)

do ip = 1,patrones
      do nn = 1,NE
      B1S(nn+1,ip)= P1S(nn,ip)
      end do
end do
open (unit=9, file='B1S.sal', action='write');write (9,*) B1S;close(9)

B1=B1S

do ip = 1,patrones
      !do nn = 1,NE+1          !estableciendo el vector de entrada
            Xi(:,1)=B1(:,ip)
      !end do

      do je = 1,LE            !pasando el vector de entrada por la capa
oculta
            !do nn = 1,NE+1
            Vnj(je,:)= Whji(je,:)*Xi(:,1)
            !end do
      end do

      do je = 1,LE            !Vector de salida de la capa oculta
            Vnjv(je,1)= sum (Vnj(je,:))
      end do

      !do je = 1,LE           ! Pasando por la funcion de activacion
            Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1  !((exp(-
2*(Vnjv(je,1))))-1)/((exp(-2*(Vnjv(je,1))))+1)
      !end do

      Z2=(1)!(SIZE(Z2))-12    !agregando el vias

      do je = 1,LE
            Z2(je+1,1)= Vnjv(je,1)
      end do

      do rc = 1,ME
            !do je = 1,LE+1
                  Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
            !end do
      end do

      do rc = 1,ME            !Vector de salida de la capa oculta
            Uokv(rc,1)= sum (Uok(rc,:))
      end do
```

```
      !do rc = 1,ME           ! Pasando por la funcion de activacion
            Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1)
      !end do

      !do rc = 1,ME
            A1(:,ip)=Uokv(:,1)
      !end do

end do
open (unit=9, file='A1.sal', action='write');write (9,*) A1;close(9)


!!!!!!!!!!!!!!!!!!!!!!!!!!!CONJUNTO DE ENTRENAMIENTO!!!!!!!!!!!!!!!!!!!!!

do idff = 1,kdf
      do itd =1,ktd
            !do nn =1,kpr0
            n=(idff-1)*ktd+itd
            P2S(:,n)=ZD2(:,itd,idff)
            T2S(idff,n)=1
            !end do
      end do
end do

open (unit=9, file='P2S.sal', action='write');write (9,*) P2S;close(9)
open (unit=9, file='T2S.sal', action='write');write (9,*) T2S;close(9)


!AGREGANDO EL BIAS A LA ENTRADA
Z1=(1) !(SIZE(Z1,1))-6
B2S=Z1
!open (unit=9, file='B1.sal', action='write');write (9,*) B1;close(9)

do ip = 1,patrones
      do nn = 1,NE
      B2S(nn+1,ip)= P2S(nn,ip)
      end do
end do
open (unit=9, file='B1S.sal', action='write');write (9,*) B1S;close(9)

B1=B2S

do ip = 1,patrones
      !do nn = 1,NE+1          !estableciendo el vector de entrada
            Xi(:,1)=B1(:,ip)
      !end do

      do je = 1,LE            !pasando el vector de entrada por la capa
oculta
            !do nn = 1,NE+1
            Vnj(je,:)= Whji(je,:)*Xi(:,1)
            !end do
      end do

      do je = 1,LE            !Vector de salida de la capa oculta
            Vnjv(je,1)= sum (Vnj(je,:))
      end do
```

```
      !do je = 1,LE           ! Pasando por la funcion de activacion
            Vnjv(:,1) = (2/(1+exp(-2*(Vnjv(:,1)))))-1  !((exp(-
2*(Vnjv(je,1))))-1)/((exp(-2*(Vnjv(je,1))))+1)
      !end do

      Z2=(1)!(SIZE(Z2))-12    !agregando el vias

      do je = 1,LE
            Z2(je+1,1)= Vnjv(je,1)
      end do

      do rc = 1,ME
            !do je = 1,LE+1
                  Uok (rc,:)= Wokj(rc,:)*Z2(:,1)
            !end do
      end do

      do rc = 1,ME            !Vector de salida de la capa oculta
            Uokv(rc,1)= sum (Uok(rc,:))
      end do

      !do rc = 1,ME           ! Pasando por la funcion de activacion
            Uokv(:,1) = (1)/((exp(-1*(Uokv(:,1))))+1)
      !end do

      !do rc = 1,ME
            A2(:,ip)=Uokv(:,1)
      !end do

end do
open (unit=9, file='A2.sal', action='write');write (9,*) A2;close(9)

PD1=(0)!SIZE(PD1))-81

do rc = 1,ME
      do itd =1,ktd
            nT = (rc-1)*ktd+itd
            do rc1 = 1,ME
                  A1S(rc1,1)= A1(rc1,nT)
            end do
            amax =
max(A1(1,nT),A1(2,nT),A1(3,nT),A1(4,nT),A1(5,nT),A1(6,nT),A1(7,nT),A1(8,n
T),A1(9,nT))
            !WRITE(*,*) MAXLOC(A1S), MAXVAL(A1S)
            ixt = SIZE(SHAPE(A1S))
            ALLOCATE ( imax(ixt))
            imax = MAXLOC (A1S)
            PD1(imax(1),rc)=PD1((imax(1)),rc)+1
            deallocate (imax)
      end do
end do

do rc = 1,ME
      do rc1 = 1,ME
            PD1(rc1,rc) = PD1(rc1,rc)/ktd
      end do
```

```
end do
open (unit=9, file='PD1.sal', action='write')
do 10 rc = 1,ME
    write(9,1000) (PD1(rc,rc1), rc1 = 1,ME)
10 continue
1000 format (9F8.4, 10X)
close (9)
!;write (9,*) PD1;close(9)

do rc = 1,ME
      PDT1(rc,1)= PD1(rc,rc)
end do
open (unit=9, file='PDT1.sal', action='write')
write(9,2000) (PDT1(rc1,1), rc1 = 1,ME)
2000 format (9F8.4, 10X)
close(9)

psr1=(sum(PDT1))/ME
PD1T=transpose(PD1)
open (unit=9, file='PD1T.sal', action='write');write (9,*) PD1T;close(9)
open (unit=9, file='psr1.sal', action='write');write (9,*) psr1;close(9)


PD2=(0)!SIZE(PD1))-81

do rc = 1,ME
      do itd =1,ktd
            nT = (rc-1)*ktd+itd
            do rc1 = 1,ME
                  A2S(rc1,1)= A2(rc1,nT)
            end do
            amax =
max(A2(1,nT),A2(2,nT),A2(3,nT),A2(4,nT),A2(5,nT),A2(6,nT),A2(7,nT),A2(8,n
T),A2(9,nT))!WRITE(*,*) MAXLOC(A1S), MAXVAL(A1S)
            ixt = SIZE(SHAPE(A2S))
            ALLOCATE ( imax(ixt))
            imax = MAXLOC (A2S)
            PD2(imax(1),rc)=PD2((imax(1)),rc)+1
            deallocate (imax)
      end do
end do

do rc = 1,ME
      do rc1 = 1,ME
            PD2(rc1,rc) = PD2(rc1,rc)/ktd
      end do
end do
open (unit=9, file='PD2.sal', action='write')
do 20 rc = 1,ME
    write(9,1000) (PD2(rc,rc1), rc1 = 1,ME)
20 continue
close (9)
!;write (9,*) PD2;close(9)

do rc = 1,ME
      PDT2(rc,1)= PD2(rc,rc)
end do
```

```
open (unit=9, file='PDT2.sal', action='write')
write(9,2000) (PDT2(rc1,1), rc1 = 1,ME)
close(9)

psr2=(sum(PDT2))/ME
PD2T=transpose(PD2)
open (unit=9, file='PD2T.sal', action='write');write (9,*) PD2T;close(9)
open (unit=9, file='psr2.sal', action='write');write (9,*) psr2;close(9)
call cpu_time (time2)
t = time2-time1
write (*,*) 'tiempo= ',t
open (unit=9, file='tiempo.sal', action='write');write (9,*) t;close(9)

!Tiempo para 1000 epocas = 20114.
stop
end program prueba1

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
!!!!!!!!!!!!!!!!!GENERACION DE NUMEROS
ALEATORIOS!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!DISTRIBUCION
NORMAL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE GAUSS (ERR)
!program gauss
DOUBLE PRECISION ERR
PARAMETER (ONE=1.0,TWO=2.0)
SIG=1.0
AMO=0
10 CALL RANDOM_NUMBER (ZE1)
   CALL RANDOM_NUMBER (ZE2)
V1=TWO*ZE1-ONE
V2=TWO*ZE2-ONE
S=V1*V1 +V2*V2
IF(S.GT.ONE) GO TO 10
ERR=V1*SQRT(-TWO*ALOG(S)/S)*SIG+AMO
!WRITE (*,*)GAUSS
RETURN
!end program gauss
END SUBROUTINE GAUSS
```

APÉNDICE 2.

PUBLICACIONES

INGENIERÍA MECÁNICA
ARTÍCULOS ACEPTADOS POR REFEREO

XII CONGRESO NACIONAL DE INGENIERÍA ELECTROMECÁNICA Y DE SISTEMAS

Software de la Red de Retro-propagación Aplicada al Diagnóstico de Turbina de Gas

Jesús Martínez López, Igor Loboda, Gabriel Sánchez Pérez
Instituto Politécnico Nacional, ESIME Culhuacán, Av. Santa Ana 1000
Col. San Francisco Culhuacán, México, D.F. 044300
Teléfono (55) 56242000 Ext. 73266 Fax (55)56262058 E-mail: squalo_jml@yahoo.com.mx

Resumen — **El monitoreo de las condiciones de la turbina de gas basado en los parámetros medidos del conducto de flujo del motor, incluye las etapas de detección de problemas, identificación de fallas y pronostico, además de requerir de una operación preliminar de desviaciones computadas entre mediciones y un motor de referencia. El presente articulo realiza una comparación entre redes neuronales; desarrolladas en Matlab, una con la ayudad del toolbox y otra desarrollada a partir de su algoritmo; ambas redes se aplican en especifico a la etapa preliminar del sistema de monitoreo y a la etapa de la clasificación de fallas, el objetivo del trabajo es mejorar el resultado de la red en las dos aplicaciones, se compara el desempeño de las redes a partir de las gràficas de los respectivos casos, sin embrago en la primera aplicación parase no tener un rendimiento aceptable, mientras que para la segunda aplicación el rendimiento es bastante alentador. El análisis de los resultados ha mostrado que para la aplicación de la red en el campo de reconocimiento de fallas, trabaja mucho mejor que para la aplicación de la función de referencia (desviaciones).**

Palabras Clave – **diagnostico, red neuronal, retro-propagación, clasificación de fallas, función de referencia**.

Abstract — Gas turbine health monitoring based on gas path measured parameter include stages of problems detection, fault identification and prognostics, also require a preliminary operation of monitoring deviation between measurements and an engine baseline. Present paper make a comparison between neuronal network development in Matlab, one of them whit toolbox help and other one from own algorithm, both networks are apply in specific preliminary stage of monitoring system and fault identification stage, the work objective is to get better network result in both applications, we compare network performed from plots of respective case, however in the firs application seems no to have acceptable performance, while in the second application the performance is to acceptable. The result analysis was shown that for the network application in the fault recognition area work much better than application of reference function (deviations).

Keywords — **diagnostic, neuronal network, back propagation, fault classification, reference function.**

I. Introducción

Las turbinas de gas conforman una parte importante en la economía del país. Son consideradas como sistemas complejos, siendo estos cada vez más sofisticados, lo cual puede afectar su confiabilidad. Por eso varias técnicas de diagnóstico han sido aplicadas en las recientes décadas, teniendo la capacidad de detectar e identificar fallas incipientes y reduciendo la intensidad de fallas graves [1,2,3].

Los sistemas de monitoreo de las turbinas de gas son ampliamente utilizados hoy en día, debido a que permiten mantener las condiciones buenas del motor.

Estos emplean información de los parámetros medidos de la turbina de gas, haciendo posible monitorear las condiciones actuales del motor, prediciendo sus futuros cambios e identificando algún problema así como su tipo y localización, a lo cual consideraremos como diagnóstico paramétrico. Tal diagnóstico es también basado en modelos matemáticos de parámetros medidos y monitoreados [4,5].

El monitoreo de las condiciones de turbinas de gas incluye tres etapas importantes e interconectadas las cuales son; detección de problemas, identificación de fallas y pronóstico. Sin embargo estas etapas requieren de una etapa preliminar la cual se encarga de analizar datos reales y computar los cambios (desviaciones) en las variables monitoreadas a causa del envejecimiento y fallas del motor.

Las redes neuronales en los recientes años se han hecho presentes en aplicaciones impensables, teniendo una gran aceptación por sus resultados. Se consideran también como una gran herramienta a la hora de hablar en el área del monitoreo de turbinas de gas. Teniendo en consideración algunos ejemplos [6,7,8], podemos concluir que en cualquier etapa del proceso total del monitoreo, es posible emplear Redes Neuronales, sin embargo la eficiencia de cada tipo de red depende de su aplicación en particular.

En Matlab se encuentra un toolbox de Redes Neuronales, el cual se han aplicado en nuestros trabajos previos para el desarrollo de los cálculos tanto; en las desviaciones, como para la identificación de fallas [6,7,9]. Sin embargo no estamos completamente seguros que estos métodos de creación o de trabajo de Redes Neuronales sean los más adecuados; Tampoco es claro si las funciones estándares de entrenamiento, las cuales funcionan en el modo de lote siempre alcancen el mínimo absoluto del error de red.

Se tiene el conocimiento de que el algoritmo de Retro-propagación en serie por su funcionamiento y en particular por la forma en que modifica sus pesos siempre alcanza el

El trabajo descrito en este artículo se desarrolló a través del apoyo del Instituto Politécnico Nacional (proyecto 20101199)

lo cual y de acuerdo con experiencias en estudios anteriores [6] hemos considerado que; la red presentara un mejor rendimiento si esta se constituye de una capa de entrada, una capa oculta con doce neuronas y una capa de salida.

Vamos a considerar primero como funciona una red de retro-propagación y luego describiremos el algoritmo de su entrenamiento. Como se ha mencionado la señal se propaga hacia adelante primero a través de la capa oculta, para lo cual necesitamos alimentar la red con un vector a la entrada $\vec{X}$, multiplicando variables X_i por los pesos y agregando a la suma un umbral de acuerdo con la ecuación (1). La suma deberá pasar por la función de transferencia como se observa en la ecuaciones (2).

$$Vh_j = \sum_{i=1}^{N_i} Wh_{ji} X_i + \theta h_j \quad (1)$$

$$Uh_j = f(Vh_j) \quad (2)$$

Obteniendo el vector $\overrightarrow{Uh}$, lo propagamos a la capa de salida según las ecuaciones (3) y (4):

$$Vo_k = \sum_{j=1}^{N_h} Wo_{kj} Uh_j + \theta o_k \quad (3)$$

$$Uo_k = f(Vo_k) \quad (4)$$

En la capa de salida se calcula el error medio cuadrático de acuerdo con las ecuaciones (5) y (6).

$$E_p = \frac{1}{2} \sum_{k=1}^{N_o} e_k^2 \quad (5)$$

$$E_p = \frac{1}{2} \sum_{k=1}^{N_o} (d_k - Uo_k)^2 \quad (6)$$

De aquí empezamos a describir el algoritmo mismo de entrenamiento por retro-propagación. El error se propaga hacia atrás de tal forma que modifiquemos las matrices de pesos; primero la de la capa de salida, como se especifica en las ecuaciones (7) y (8). Se aplica en las ecuaciones un coeficiente de aprendizaje μ para cambiar la velocidad de la modificación de los pesos.

$$Wo_{kj}(n) = Wo_{kj}(n-1) - \mu \frac{\partial E_p}{\partial Wo_{kj}} \quad (7)$$

$$\frac{\partial E_p}{\partial Wo_{kj}} = (d_k - Uo_k)(1 - Uo_k) Uo_k * Uh_j \quad (8)$$

Ahora se modifica la matriz de la capa oculta Wh de la misma manera que en las ecuaciones (7) y (8). En las ecuaciones nuevas (9) y (10) el coeficiente de aprendizaje se denomina α.

$$Wh_{ji}(n) = Wh_{ji}(n-1) - \alpha \frac{\partial E_p}{\partial Wh_{ji}} \quad (9)$$

$$\frac{\partial E_p}{\partial Wh_{ji}} = (1 + Uh_j)(1 - Uh_j) X_i * \sum_{k=1}^{N_o} (d_k - Uo_k)(1 - Uo_k) Uo_k * Wo_{kj} \quad (10)$$

Como ya se ha mencionado en el monitoreo de turbinas de gas, se tienen tres etapas, mas una etapa previa para el cálculo de las desviaciones. A continuación se describe la aplicación del algoritmo en la etapa preliminar que incluye: análisis de datos reales y cálculo de las desviaciones.

IV. Aplicación 1: Desviaciones

La eficiencia de un sistema automatizado de monitoreo depende de la calidad de las desviaciones. Estas son calculadas como discrepancias entre las variables medidas de un motor dañado y las variables de un motor sin daño alguno (motor nuevo o reparado) [4]. Aunque los efectos de deterioro del motor afectan las mediciones y las variables monitoreadas del conducto de flujo, el impacto de los cambios en operación y las condiciones ambientales son mucho más fuertes. Sin embargo, si para una variable monitoreada Y_i substraemos un valor de referencia Y_{ci} de un valor medido actual Y_i^*, la diferencia $Y_i^* - Y_{ci}$ será prácticamente libre de influencia de estas condiciones, conservando un efecto de deterioro. Típicamente esta diferencia (desviación) es dada en una forma relativa. Por lo tanto una desviación relativa δY_i^* puede ser calculada de acuerdo a la expresión:

$$\delta Y_i^* = \frac{Y_i - Y_{ci}}{Y_{ci}} \quad (11)$$

Dado que variables monitoreadas en una turbina de gas dependen del punto de operación (régimen) del motor, los valores de referencia para una turbina de gas en operación estacionaria se presentan como: una función $\vec{Y}_c = f(\vec{U})$ de las variables de operación y las ambientales $\vec{U}$ [6]. Mencionando lo anterior escribimos la desviación en la forma siguiente:

$$\delta Y_i^* = \frac{Y_i - Y_{ci}(\vec{U})}{Y_{ci}(\vec{U})} \quad (12)$$

La calidad de las desviaciones calculadas conforme a la formula (12) será dependiente de la adecuación de la función de referencia. Por lo cual pretendemos en este artículo obtener por él medio de la red de retro-propagación una función de referencia más adecuada a los datos reales.

Para calcular esta función y las desviaciones (12), dos versiones de software similares fueron diseñados. La diferencia consiste en la realización de la red de retro-propagación. La versión 1 usa las funciones estándares de Matlab para crear, entrenar y simular la red; mientras que en la versión 2 se programan los algoritmos descritos arriba. Para entrenar y verificar la red de retro-propagación en ambas versiones, se usan los datos del motor escogido (ver la sección I) registrados en campo. Estos datos fueron

medidos por el sistema estándar de mediciones del motor y registrados en el sistema de monitoreo cada hora de la operación del motor. Por lo tanto, la estructura de las variables de los vectores $\vec{Y}$ y $\vec{U}$ corresponde al sistema estándar de mediciones.

Como hemos mencionado antes, el algoritmo de entrenamiento realizado en la versión 2 de software funciona en el modo de serie. Cada vector $\vec{U}$ de los datos de entrenamiento entra a la red; la salida correspondiente de la red se compara con la salida deseada (tarea) $\vec{U}$ que se encuentra en los mismos datos de entrenamiento. El error (la diferencia entre la tarea y la salida actual) se usa para corregir los pesos de la red. Este ciclo se repite sucesivamente para todas 2608 secciones de medición escogidas para el entrenamiento (primera parte de los datos registrados disponibles) formando un ciclo interior del entrenamiento en serie. Puesto que los coeficientes de aprendizaje son relativamente pequeños, nunca es suficiente un ciclo interior. Por lo tanto, este ciclo se repite varias épocas en un ciclo exterior resultando en un número grande de las iteraciones totales. Al fin, la red entrenada se aplica a los datos de validación. La calidad de esta red se estima visualmente por el medio de las gráficas de las desviaciones, calculadas con la red en los datos de entrenamiento y los de validación.

Para ajustar este algoritmo de entrenamiento (versión 2 de software), probamos diferentes valores de los coeficientes de aprendizaje y del número total de las épocas. Los mejores resultados de la versión 2 obtenidos son con 3000 épocas totales, se describen abajo en comparación con los resultados de la versión 1 de software.

Las figuras 3 y 4 ilustran las desviaciones computadas con los datos de entrenamiento por ambas versiones de software. Estas desviaciones dTt corresponden a la temperatura de gases atrás la turbina de alta presión y son presentadas contra el tiempo de operación del motor. El deterioro del motor analizado no influye aquí a las desviaciones, porque aquí ellas se calculan usando no la función de referencia, sino el modelo de motor deteriorado [ver 7]. Por consiguiente, estas desviaciones pueden ser consideradas como errores y deben ser minimizadas. Comparando las figuras, llegamos a la conclusión que la versión 2 de software (software nuevo de entrenamiento), aunque no produce errores graves, pierde contra la versión 1 (función estándar de entrenamiento): el nivel de desviaciones (errores) es notablemente mayor para la versión 2. Ahora vamos a ver si esta diferencia en la etapa de entrenamiento afecta a los resultados de validación.

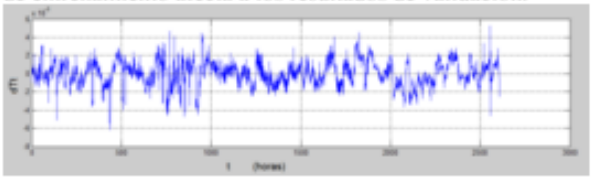


Fig. 3 Desviaciones computadas con los datos de entrenamiento (versión 1 de software)

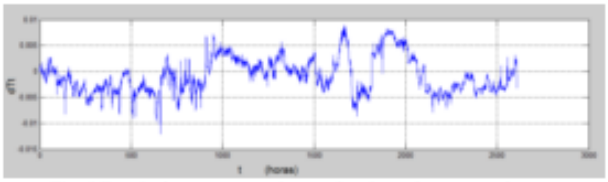


Fig. 4 Desviaciones computadas con los datos de entrenamiento (versión 2 de software)

En la etapa de validación se utilizan todos los datos disponibles: los datos de entrenamiento más 1483 secciones adicionales de medición, 4091 secciones en total. Las figuras 5 y 6 que ilustran la validación tienen el mismo formato que las figuras previas. Las desviaciones se calculan aquí según la fórmula (12) donde la función de referencia $\vec{Y_0} = f(\vec{U})$ se define por una transformación simple del modelo de motor deteriorado [ver 7], modelo determinado en la etapa de entrenamiento. Por la explicación anterior, las desviaciones calculadas en la validación incluyen: la influencia de deterioro del motor, en particular la contaminación de su compresor.

Como se puede observar en la figura 5, la grafica de desviaciones refleja correctamente el aumento gradual de la temperatura a causa de la contaminación antes y después del lavado del compresor en el punto t = 907 horas. Al menos eso se ve claramente en la parte izquierda de la gráfica, en tanto que en la parte derecha crecen las fluctuaciones (errores aleatorios).

En la figura 6 las tendencias sistemáticas en la parte izquierda de igual manera. Sin embargo la calidad de las desviaciones (relación entre el cambio sistemático y el ruido) parece peor aquí que en la figura 5. Parece que la inadecuación del modelo obtenido por el entrenamiento con la versión 2 de software ha resultado en una adecuación baja de la función de referencia, lo que causó una calidad peor de las desviaciones.

Así, a pesar de haber modificado varias veces tanto el número de épocas como los coeficientes de aprendizaje, hay que buscar todavía sus valores óptimos.

Fig. 5 Desviaciones computadas con los datos de validación (versión 1 de software)

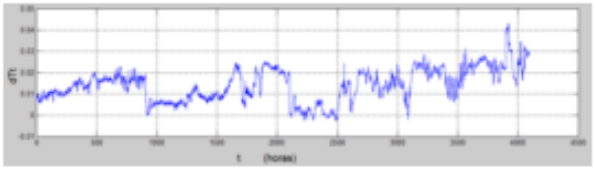


Fig. 6 Desviaciones computadas con los datos de validación

(versión 2 de software).

V. Aplicación 2: Identificación de las Fallas.

La siguiente aplicación del software nuevo es para la identificación (reconocimiento) de las fallas de turbinas de gas. Aquí damos una breve descripción de nuestro enfoque del reconocimiento. El lector puede consultar [4] para una explicación detallada.

Esta etapa del proceso total de monitoreo se basa comúnmente en la teoría de reconocimiento de patrones. Esta teoría supone una clasificación de los estados reconocidos de un sistema. En general se acepta la hipótesis de que el estado de un sistema puede pertenecer sólo a una de q clases. [4,9,10,11]

$$D_1, D_2, ..., D_q \qquad (13)$$

Predeterminadas de antemano. Aceptamos esta hipótesis para la clasificación de fallas de turbinas de gas. Existen varios principios para crear tal clasificación, el principio más común (13) dice que cada clase corresponde a uno de los componentes del motor (compresor, cámara de combustión, turbinas, etc.). Por lo tanto el esquema estructural del motor determina la estructura de clasificación de fallas.

Para simular las fallas del motor analizado, se aplica su modelo termodinámico, el cual calcula las variables monitoreadas como una función de las variables de operación y un vector de parámetros de falla $\vec{\Theta}$. Así, el modelo puede ser descrito por la siguiente expresión vectorial

$$\vec{Y} = F(\vec{U}, \vec{\Theta}) . \qquad (14)$$

Los elementos del vector $\vec{\Theta}$ desplazan los mapas de características de los componentes en direcciones dadas, simulando fallas de estos componentes.

En el proceso de simulación de cada clase de fallas por el modelo termodinámico, una desviación normalizada Z_i^* para una variable monitoreada Y_i se compone por dos componentes: un componente sistemático Z_i y un componente aleatorio ε_i, p.e. $Z_i^* = Z_i + \varepsilon_i$. El componente sistemático es inducido por fallas implantadas en el modelo. Cambio de cada parámetro particular de fallas produce una clase de fallas de severidad variable. El componente ε_i toma en cuenta los errores aleatorios en las desviaciones y se cambia dentro del intervalo (-1,1). Las desviaciones de todas las variables monitoreadas forman el vector $\vec{Z}^*$, el cual es un patrón para ser reconocido cuando se lleva a cabo el diagnóstico. En el espacio de las desviaciones, cada clase es representada estadísticamente por su propio conjunto de patrones $\vec{Z}^*$. La severidad de falla es dada por la distribución uniforme del parámetro correspondiente del intervalo (0, -5%), mientras que los errores ε_i so generados de acuerdo a la distribución Gaussiana.

La totalidad de patrones de todas las clases, llamad muestra de aprendizaje, es aplicada para entrenar la re neuronal. Para validar la red, un conjunto adicional llamado muestra de prueba – es formado justo como l muestra de aprendizaje. Cada patrón de conjunto de prueb entra en la red neuronal entrenada y esta hace la diagnosi correspondiente. Comparando la diagnosis con la clas actual, calculamos probabilidades de la diagnosis correct para cada clase y en promedio.

Estas probabilidades son empleadas como criterio principales para la comparación de dos versiones d software, que realizan en paralelo todo el enfoque descrit arriba. De la manera semejante a la de la sección IV, l versión 1 involucra las funciones estándares para entrenar utilizar la red de retro-propagación mientras que la versión usa el software nuevo al mismo fin. En adición a lo criterios probabilísticas, el error medio cuadrático de la re se emplea también para comparar estas versiones.

Para ajustar el algoritmo de entrenamiento en seri (versión 2 de software), se realizaron varias prueba cambiando el número de épocas y los coeficientes d aprendizaje. En estas pruebas variamos primero el núme de épocas considerando los valores 100, 200, 300, 500 1000. Al encontrar un valor aceptable, hacemos nueva pruebas variando los coeficientes de aprendizaje.

Vamos a analizar primero el proceso de entrenamient con ayuda de las gráficas del error mostradas en las figuras y 8 para las versiones 1 y 2 correspondientemente. Se pued ver que con el software nuevo (versión 2) hemos llegado resultados bastante alentadores: el error en la figura 8 baj hasta el mismo nivel que el error de la figura 7. Podemos ve también que para la versión 2 el error puede disminuir au más pero para eso necesitaríamos un gran número d épocas.

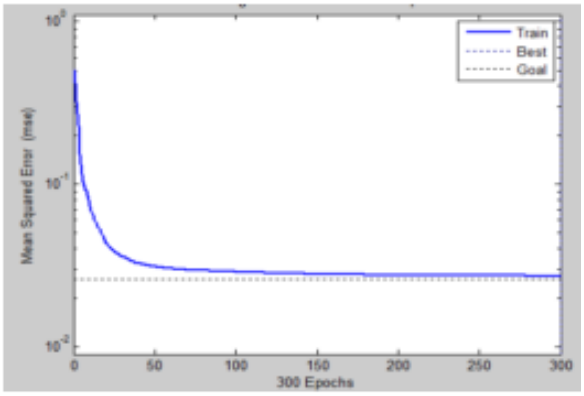


Fig. 7 Cambio del error medio cuadrático para la versión 1 de software. (300 épocas del entrenamiento)

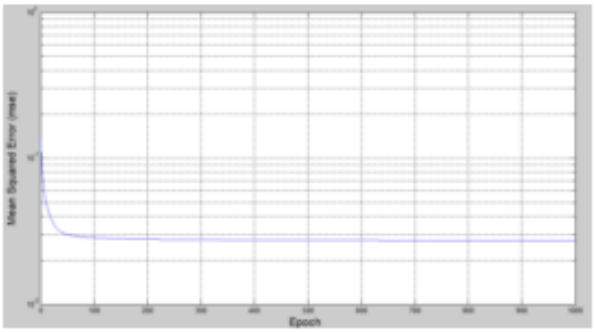


Fig. 8 Cambio del error medio cuadrático para la versión 2 de software (1000 épocas del entrenamiento).

Los resultados de la experimentación con números diferentes de épocas se dan en forma de las probabilidades de la diagnosis correcta en la Tabla 1. Observando las probabilidades promedio, podemos concluir que ellas no crecen con épocas adicionales. Al contrario, las probabilidades para la muestra de validación PD2 con 100 épocas de entrenamiento son las más altas y muy cercanas a las de la versión 1 de software.

De la misma manera la Tabla 2 presenta resultados de los experimentos con el coeficiente de aprendizaje α. Comparando las probabilidades PD2 promedio (última columna), podemos ver que el cambio de este coeficiente prácticamente no influye al resultado de la versión 2 de software: PD2=0.8196-0.8197. La versión 1 tiene aquí el resultado un poco mejor (PD2=0.8228) pero la diferencia no es muy notable.

Con ALFA = 0.0025		Clase1	Clase2	Clase3	Clase4	Clase5	Clase6	Clase7	Clase8	Clase9	Promedio
Epoch =100	PD1	0,8400	0,7340	0,8780	0,7220	0,8520	0,8280	0,8510	0,8350	0,8360	0,8196
	PD2	0,8430	0,7290	0,8900	0,7520	0,8470	0,7890	0,8720	0,8320	0,8440	0,8220
Epoch =200	PD1	0,8540	0,7420	0,8810	0,7250	0,8590	0,8310	0,8500	0,8330	0,8380	0,8237
	PD2	0,8490	0,7300	0,8840	0,7540	0,8480	0,7900	0,8690	0,8240	0,8340	0,8202
Epoch =300	PD1	0,8530	0,7470	0,8770	0,7250	0,8570	0,8380	0,8470	0,8330	0,8360	0,8237
	PD2	0,8480	0,7300	0,8830	0,7500	0,8510	0,7920	0,8670	0,8240	0,8320	0,8197
Epoch =500	PD1	0,8480	0,7530	0,8770	0,7240	0,8590	0,8440	0,8490	0,8290	0,8410	0,8249
	PD2	0,8450	0,7330	0,8830	0,7450	0,8530	0,7940	0,8630	0,8210	0,8330	0,8189
Epoch =1000	PD1	0,8490	0,7580	0,8780	0,7160	0,8630	0,8470	0,8450	0,8280	0,8460	0,8256
	PD2	0,8490	0,7400	0,8840	0,7430	0,8560	0,7980	0,8640	0,8130	0,8310	0,8198
Version 1	PD1	0,8540	0,7350	0,8710	0,7460	0,8700	0,8520	0,8450	0,8270	0,8340	0,8260
	PD2	0,8530	0,7250	0,8860	0,7630	0,8660	0,8070	0,8660	0,8160	0,8230	0,8228

Tabla 1. Probabilidades del diagnostico correcto variando el numero de épocas.

Con ALFA = variable		Clase1	Clase2	Clase3	Clase4	Clase5	Clase6	Clase7	Clase8	Clase9	Promedio
Version 1	PD1	0,8540	0,7350	0,8710	0,7460	0,8700	0,8520	0,8450	0,8270	0,8340	0,8260
	PD2	0,8530	0,7250	0,8860	0,7630	0,8660	0,8070	0,8660	0,8160	0,8230	0,8228
Epoch =300 Alfa =0.0025	PD1	0,8530	0,7470	0,8770	0,7250	0,8570	0,8380	0,8470	0,8330	0,8360	0,8237
	PD2	0,8480	0,7300	0,8830	0,7500	0,8510	0,7920	0,8670	0,8240	0,8320	0,8197
Epoch =300 Alfa =0.0020	PD1	0,8540	0,7400	0,8760	0,7300	0,8570	0,8350	0,8490	0,8320	0,8370	0,8233
	PD2	0,8470	0,7270	0,8830	0,7560	0,8470	0,7900	0,8680	0,8250	0,8330	0,8196
Epoch =300 Alfa =0.0030	PD1	0,8520	0,7520	0,8780	0,7200	0,8580	0,8360	0,8480	0,8310	0,8360	0,8234
	PD2	0,8470	0,7270	0,8830	0,7560	0,8470	0,7900	0,8680	0,8250	0,8330	0,8196

Tabla 2. Probabilidades del diagnostico correcto variando el coeficiente de aprendizaje.

VI. Conclusiones

Así, en este articulo hemos considerado, el análisis de las redes neuronales desde el punto de los comandos de Matlab en comparación con las mismas redes neuronales de retro-propagación pero, con sus algoritmos programados por nosotros mismo. Lo que nos motivo para realizar este estudio fue el hecho de saber que el algoritmo de la red de retro-propagación siempre llega a un mínimo global, ofreciéndonos la posibilidad de entregarnos resultados, con un rendimiento y procesamiento de la información mejores.

El algoritmo de la red de retro-propagación fue programado y adaptado para las dos aplicaciones antes señaladas y discutidas, una para la obtención de una función de regencia y otra para la clasificación de fallas. En ambos casos la red fue probada, entrenada y simulada considerando para cada aplicación los mejores parámetros de la red.

Después de numerosas modificaciones del cálculo y análisis de cada modificación, así como las opciones que se presentaron para mejorar el rendimiento de estos casos llegamos a la siguiente conclusión: del trabajo realizado antes que nada debemos estar satisfechos de cierta forma debido a que los resultados obtenidos son muy cercanos a los objetivos que se esperaban alcanzar. Sin embargo hay cosas que nos dejan duda, como el tiempo en que tarda el algoritmo en procesar la información es considerablemente demasiado; comparado con el algoritmo original. Eso por un lado, por otro lado nos deja pensando en la posibilidad de

mejorar el rendimiento del algoritmo de la red de retro-propagación.

Si bien para el reconocimiento de patrones (clasificación de fallas), presenta mejor rendimiento que para el cálculo de la función de referencia, esto no significa que solo se pueda utilizar para una aplicación y para la otra no, al contrario esto nos ayuda para seguir trabajando y ver que vamos en vías de obtener mejores resultados

REFERENCIAS

[1] Kacprzynski Gregory J., Michel Gumina, Micheal J. Roemer, Daniel E. Caguiat, Thomas R. Galie, Jack J. McGroarty. A prognostic modeling approach for predicting recurring maintenance for shipboard propulsion system. IGTI/ASME Turbo Expo 2001, New Orlans, USA, 7p.

[2] Lars J. Kangas, Frank L. Greitzer. Automated health monitoring of M1A1/A2 battle tank. USAF Inaugural Engine Condition Monitoring (ECM) Workshop, 1997, San Diego, CA, USA.

[3] Fank L. Greitzer, Lars J. Kangas, Kristine M. Terrones, Melody A. Maynard, Bary W. Wilson, Ronald A. Pawlowski, Daniel R. Sisk and Newton B. Brown. Gas turbine engine health monitoring and prognostics. Internacional Society of Logistics Symposium, Las Vegas, Nevada, USA, 1999, 7p.

[4] Loboda Igor. "An overview of Gas Turbine Healt Monitoring Methods", XI Congreso y Exposicion Latinoamericana de Trubomaquinaria, 14-17 Octubre 2008, Veracruz, Veracruz México, 10p.

[5] Hans R. Depold, Ravi Rajamani, William H. Morrison, Krishna R. Pattipati. A unified metric for fault detection and isolation in engines. IGTI/ASME Turbo Expo 2006, Barcelona, Spain, 7p

[6] Loboda Igor, Feldshteyn Yakov. "Polynomials and Networks for Gas Turbine Monitoring: a Comparative Study", Proceedings of GT2010: ASME Turbo Expo 2010: Powere for Land, Sea and Air, June 14-18, 2010, Glasgow, Scotland, Uk, 12p.

[7] Villarreal G. Claudia, Loboda Igor, Trahyn A. Iván, Análisis de una Red Neuronal en la Aplicación a la Función de Referencia de una Turbina de Gas, Memorias del Congreso Nacional de Instrumentación SOMI XXIV, Mérida, Yucatán, del 14 al 16 de octubre de 2009, 11p.

[8] Loboda Igor, Miyatake M. Nakano, Goryanchiy A. Gutiérrez Mojica E. M., González Aguilar J. E., Gas Turbine Fault Recognition by Artificial Neural Networks, Memorias del 4to Congreso Internacional de Ingeniería Electromecánica y de Sistemas, ESIME, IPN, México, 14-18 noviembre de 2005, ISBN: 970-36-0292-4, 6p.

[9] Magnus Fast, Mohsen Assedi, Andrew Pike, Peter Breuhaus. Different condition monitoring models for gas turbines by means of artificial neural network. IGTI/ASME Turbo Expo 2009, June 8-12, 2009, Orlando, Florida, USA, 11p. ASME Paper GT2009-59364.

[10] Duda R. O., Hart P. E. and Stork D. G. (2001) Pattern Classification, Wiley-Interscience, New York.

[11] Loboda I and Feldsheteyn Ya. (2007). A universal fault classification for gas turbine diagnosis under variable operating conditions. International Journal of Turbo & Jet Engine, Vol. 24. No. 1, pp 11-27, ISSN 0334-0082

[12] Loboda I, Yepifanov S. (2006). Gas turbine fault recognition trustworthiness, Cientifica ESIME-IPN, Mexico, Vol. 10, No. 2, pp 65-74, ISSN 1665-0654

Universidad Autónoma de Querétaro

FACULTAD DE INGENIERÍA Fi

6° CONGRESO INTERNACIONAL DE INGENIERÍA

Comparación de dos tipos de redes neuronales artificiales (retropropagación y base radial), utilizadas para el diagnóstico paramétrico de turbinas de gas

Comparison of two types of artificial neural networks (backpropagation and radial basis), used for parametric gas turbines diagnosis

Eulalio Torres García[2], Igor Loboda[1], Jesús Martínez López[2].
[1]Profesor en Instituto Politécnico Nacional, SEPI- Culhuacán, [2]Estudiante en Instituto Politécnico Nacional, SEPI- Culhuacán. Contacto: ingetorresg@gmail.com

RESUMEN: Este artículo está orientado a la comparación de dos tipos de redes Neuronales Artificiales (RNA), tipo Retropropagación y de Base Radial, aplicadas en el Diagnóstico de Turbinas de Gas. Se hace un análisis del comportamiento con 12 diferentes funciones de adaptación, combinaciones de 1 a 3 en los números llave para la iniciación de las vectores de comportamiento alrededor de clases identificadas, se utilizan 11 regímenes de operación de la turbina, todo ello controlado por una interface usuario que permite reducir el tiempo de procesamiento de datos. El resultado es arrojado a una probabilidad de aserción (calculada por estadística) de un diagnóstico correcto y el tiempo aproximado de entrenamiento. Comparado estos resultados, es posible determinar cuál es el tipo de red y el tipo de entrenamiento que debe tener una RNA, para ser utilizada en esta aplicación.

Palabras clave: Redes Neuronales Artificiales de Retropropagación (perceptron), Redes de Base Radial, Diagnóstico de turbinas de gas, Clasificación de fallas, Probabilidad de Diagnóstico Correcto.

- **INTRODUCCIÓN.**

Las turbinas de gas son sistemas muy complejos, que implica un método de diagnóstico de igual manera complejo. Entre mayor sea el número de componentes del sistema, hay mayor probabilidad de falla. Ello lleva a requerir una técnica que sea capaz de detectar las fallas con prelación en cada componente y así evitar pérdidas mayores. Las fallas y los procesos de deterioración afectan la fiabilidad, disponibilidad del equipo y los costos de operación. La aplicación de sistemas de diagnóstico y monitoreo, los cuales han llegado a ser una práctica estándar, permiten disminuir estos efectos negativos.

Actualmente el proceso de diagnóstico es un tema que implica conocer el motor y su comportamiento en estado normal, de este modo es posible identificar cuando tiene algún síntoma de falla. En el proceso de diagnóstico se aplican algoritmos basados en el estado normal del motor (estado inicial), algunas técnicas de reconocimiento de patrones y modelos avanzados de probabilidad que permiten identificar las desviaciones.

Una vez que se conoce el comportamiento del motor, es muy importante tomar en cuenta que no es correcto utilizar los parámetros directamente medidos, ya que el cambio de régimen puede esconder cambios inducidos por las fallas. En todos los métodos del diagnostico paramétrico se usan las desviaciones de los parámetros medidos

$$\delta Y = \frac{Y^* - Y_0}{Y_0} \quad (1)$$

en donde: Y^* es el parámetro medido actual del conducto de flujo; Y_0 es el parámetro del estado normal.

Loboda I (2003) menciona que el estado normal de una turbina de gas implica un estado fijo, que cumpla con los requisitos de una turbina, por ejemplo, el estado después de la fabricación. Este estado depende del régimen de funcionamiento de la turbina, por lo tanto, se puede modelar por medio de una función de estado normal, llamada modelo de estado normal.

Así podemos presentar en la forma general, un modelo de estado normal, como:

339-867

$$\vec{Y} = f_1(\vec{U}) \qquad (2)$$

en donde: $\vec{U}$- vector del régimen de turbina de gas y de las condiciones ambientales (temperatura y presión del aire).

Para formar la descripción completa de la turbina es necesario usar un modelo termodinámico no lineal del conducto de flujo, que se escribe por la siguiente fórmula:

$$\vec{Y} = f_2(\vec{U}, \vec{\theta}) \qquad (3)$$

en donde: $\vec{\theta}$ es el vector de los parámetros de estado, los cuales son capaces de corregir las características de los componentes de las turbinas de gas. Al poner en la formula (3) el vector fijo $\vec{\theta}_0$, que corresponde al estado normal transformamos el modelo termodinámico:

$$\vec{Y} = f_2(\vec{U}, \vec{\theta}_0) \qquad (4)$$

que corresponde al modelo normal (2).

Partiendo del modelo termodinámico, con una serie de datos obtenidos del motor, es posible generar patrones de comportamiento. En este patrón o vector de desviaciones ($\delta\vec{Y}$), cada puto representa una medición en el conducto de flujo. Es necesario considerar que a las desviaciones se deben agregar errores aleatorios ($\delta\vec{Y}^*$), que nos permitan introducir mediciones erróneas (ruido) y a pesar de ello generalizar el comportamiento. Ya que estos patrones que son utilizados para el entrenamiento, nos servirán para reconocer, clasificar y diagnosticar el estado del motor.

- **REDES NEURONALES ARTIFICIALES DE RETROPROPAGACIÓN.**

Retropropagación (backpropagation) es la generalización de la regla de aprendizaje Widrow-Hoff para redes neuronales multicapa y funciones de transferencia no diferenciables y no lineales. Los vectores de entrada (input) y los correspondientes vectores de salida (target) son utilizados para entrenar la red hasta que se pueden aproximar a una función. Esta está asociada con los vectores de entrada y los vectores específicos de salida, o clasificación de los vectores de entrada en el apropiado camino definido por el usuario *(Howard Demuth 2008)*.

La arquitectura base de cualquier RNA tipo Retropropagación y que a su vez es la base de nuestro estudio es la que se muestra en la Fig. 1.

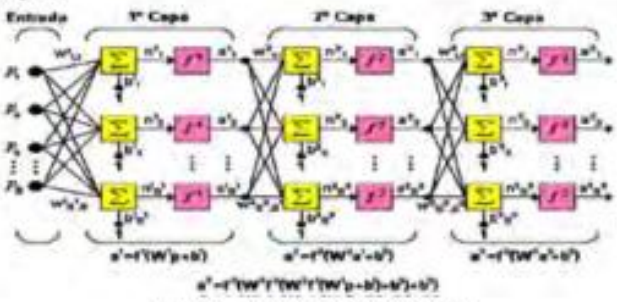


Figura 1. Arquitectura de la RNA

Existen diferentes caminos en los cuales se puede aumentar el desempeño del entrenamiento. Puede ser a través del control de las funciones de activación, que no es más que la forma en cómo se va a calcular los pesos en cada iteración, logaritmo tangencial, logaritmo sigmoidal o lineal, son las básicas. Otra manera de hacer la aproximación más exacta o en menor tiempo es el incremento del número de neuronas por capa o bien el numero de capas ocultas. En algunos casos este cambio aumenta la exactitud de la función y/o disminución del tiempo de entrenamiento. Cabe aclarar que esto no es regla, pues bien una vez mas es necesario recordar que dependiendo de la aplicación, es la función que darán mejores resultados.

Una función adaptativa es un ente formado por la neurona artificial y un algoritmo de corrección de pesos. La neurona artificial es modelada mediante una combinación lineal, esto es, produce una salida (en cada iteración) $y(n)$ que es el resultado de combinar linealmente las entradas $x(n)$, cada uno con su peso $w(n)$ característico de cada neurona. Los pesos son adaptables, y se modifican tras cada salida mediante un algoritmo de corrección. Dicho algoritmo, calculado como la diferencia entre la salida de la neurona y la salida deseada, que llamamos $d(n)$ y que es conocida. Así el método, no es más que n iteraciones para resolver la optimización. El problema se reduce a encontrar el valor de peso w_0 que minimiza el error y con ello alguna función dependiente del mismo que tome su valor mínimo en el mismo punto *(Manual de Redes neuronales - 2005)*.

En el toolbox de MATLAB existen 12 algoritmos de entrenamiento, trainlm, traingd, traingdm, trainda, traingdx, trainrp, traincgf, traincgp, traincgb, trainscg, trainbfg, trainoss. Estos algoritmos han demostrado ser buenos para diferentes aplicaciones, haciendo imposible decir que un algoritmo es mejor que otro. Lo que los hace diferentes es la forma en cómo manejan los pesos a la salida para la convergencia entre la salida deseada y la obtenida, por ejemplo, en dirección del gradiente, utilizando un grado de aprendizaje (lr-learning rate) que determinara el número de veces que se debe incrementar el gradiente. Otra forma es sumar al gradiente la tendencia reciente en la superficie del error. Existen algunos parámetros que permiten a las redes aprender más rápido o bien las hace oscilar y por ende se vuelven inestables, estas variables que permiten a las redes ser más o menos eficientes, son las que se buscan para encontrar las más eficientes en nuestra aplicación.

- **REDES NEURONALES ARTIFICIALES DE BASE RADIAL**

Una red de base radial (Radial Basis Function - RBF) está diseñada con neuronas en la capa oculta activadas mediante funciones radiales de carácter no lineal con sus centros gravitacionales propios. En la capa de salida se controla mediante funciones lineales. Aunque se reconoce que en la fase de entrenamiento es una red que presenta un alto desempeño, en general no se utiliza para altos volúmenes de datos, porque con lleva tiempo de entrenamiento muy largo. Lo cual hace que la red sea ineficiente para nuestro contexto, ya que si se desea utilizar para diagnostico en tiempo real, para el momento que esta alcance el tiempo entrenamiento, posiblemente la turbina ya habrá fallado. El entrenamiento a diferencia de Retropropagación es solo hacia adelante. La salida es influenciada por una transformación no lineal originada en la capa oculta a través de una función radial y una lineal en la capa de salida por una función lineal continúa *(Gutiérrez Mojica Eder M. 2005)*. En la Fig. II. se puede observar la topología general de una RBF.

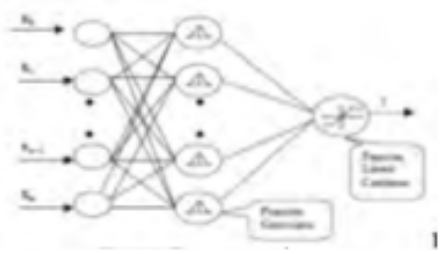


Fig. II. Topología de una red de base radial

341-867

- **ALGORITMO**

El algoritmo que se ha diseñado consta de 4 partes. La primera es la interface usuario, en ella se introducen todos los valores constantes, como el régimen de operación. Permitiendo tener una base de datos constante para más de un experimento, así disminuimos el tiempo de operación entre un experimento y el siguiente. Por otro lado al ser la misma base de operación los resultados pueden ser objeto de comparación. Además permite una rápida y eficiente manipulación de los datos de inicialización de la red.
La segunda es el ensamble de los datos; estos deben ser adecuados para que puedan ser leídos en forma correcta por el compilador. Esta parte del algoritmo tomara de la interface usuario las constantes, leerá los datos que fueron obtenidos de una turbina, preparara el arreglo multidimensional y proveerá el vector de aprendizaje. Para estos efectos se experimentó utilizando solo 9 clases diferentes de falla. En nuestro caso de estudio aceptamos la hipótesis de que el estado actual del motor pertenece solo a una clase, es decir no hay falla simultánea en más de un componente al mismo tiempo. Después se generó las nubes de puntos ($\delta\bar{Y}^*$) de manera estadística alrededor de nuestras clases identificadas. Esa generación de puntos se hace por medio de una función estadística de cercanía, llamada 3 sigma (3σ), la cual nos asegura que los puntos generados estén dentro de la clase adecuada (probabilidad 99.7%), incluyendo el espacio multidimensional. En la misma etapa se proporcionan las muestras de entrenamiento y validación para la posterior clasificación.
La tercera parte es el entrenamiento de la red, para ello tomamos de los datos previamente preparados. En cada tipo de red se debe especificar el número de épocas, error mínimo permitido, y valores propios del algoritmo de entrenamiento que se está utilizando. El proceso de clasificación toma una muestra de entrenamiento, y con una muestra de validación hace una comparación. Si el valor del error sobrepasa el deseado se regresa al entrenamiento de lo contrario regresa los vectores que forman parte de nuestra función ya entrenada.
La cuarta parte hace el cálculo de probabilidades de forma estadística. Se calcula una matriz de probabilidades y en pantalla solo se muestra el promedio de probabilidades de la segunda matriz, la cual fue de prueba. Se corta el tiempo de operación y también se envía a pantalla. Por último se grafican las clases, se muestran solo cuatro con el objetivo de hacerlas más visibles.

- **EXPERIMENTOS PRELIMINARES**

En primera instancia se definió cuál es el objetivo de la experimentación. Si bien el objetivo es comparación, tenemos que tener un marco de referencia, el objetivo primordial fue medir el tiempo de operación y la eficiencia de la red neuronal, siendo lo más importante la probabilidad de dar un diagnóstico correcto. El algoritmo fue diseñado para calcular dos matrices de probabilidades, como se menciono anteriormente. La primera no es mucho de nuestro interés ya que es la matriz obtenida en el entrenamiento, por ende al hacer el cálculo de probabilidad, el porcentaje de aserción será muy alto. La probabilidad que es calculada en la segunda muestra, es la matriz que se obtiene como resultado de prueba de la red, esta es de más importancia.
Etapas de experimentación; en la primera de ellas simplemente se introdujeron valores estándar dependientes de cada filtro adaptativo, como numero de épocas, grado de aprendizaje, coeficiente de momento, etc., con ello se observó el comportamiento de la red. Como se puede observar en la Fig. III., estos valores son definitorios del tiempo y el resultado obtenido, no debemos iniciar una red que necesita más de 2000 épocas con 100, pues ello dará probabilidades de aserción muy bajas, o viceversa, ello llevara al sobre entrenamiento, extensión del tiempo y resultados no satisfactorios.

Universidad Autónoma de Querétaro

INGENIERÍA Fi

6° CONGRESO INTERNACIONAL DE INGENIERÍA

metodo	1) trainlm	2) Traingd	3)Traingdm	4)traingda	5
Tiempo	10.53	15.41	15.20	1.59	
performance	0.0359	0.05	0.05	0.02814	
epocas	100	8000	8000	1000	
PD1	0.7451	0.715	0.7142	0.8274	
PD2	0.745	0.714	0.7136	0.8178	

Fig. III. Tabla inicial comparativa de datos sin optimización. Demuestran baja probabilidad (PD) y errores hasta del 5 % y tiempo de hasta 15 minutos

Con estos resultados se inicializa la experimentación, buscando en primera instancia aumentar la probabilidad obtenida para la muestra 2. Como se puede observar en la Fig. IV. en un principio la probabilidad era muy baja. La probabilidad 2 (PD2) está intimamente ligada con la gráfica de desempeño (performance), esta grafica es la suma de los errores. Si se reduce el valor en la gráfica de performance indica que estamos reduciendo el error y por ende aumentando la probabilidad de acierto.

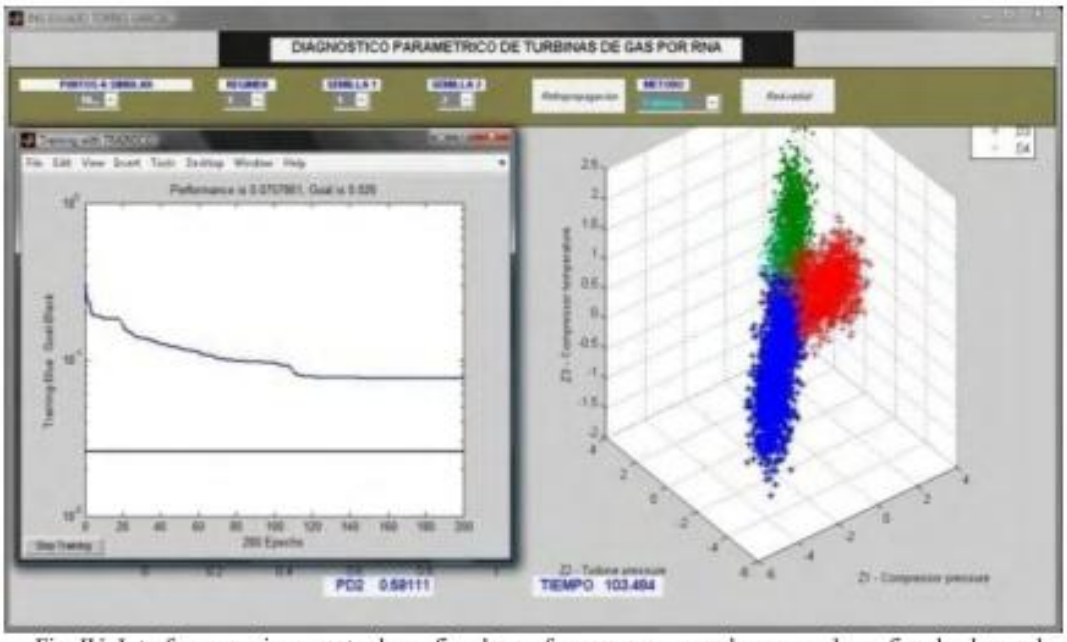


Fig. IV. Interface usuario, muestra la grafica de performance, o suma de errores, la grafica de clases, la probabilidad de acierto y el tiempo transcurrido

En la siguiente etapa, el parámetro a controlar es el tiempo de entrenamiento, pues bien de nada sirve tener una RNA que sea capaz de identificar el patrón de comportamiento después de horas. Si el diagnóstico se quiere hacer en tiempo real, pues después de horas es posible que el motor ya haya fallado, entonces el diagnóstico se vuelve inútil.

En la siguiente etapa se cambió el numero de épocas, disminuimos o aumentamos dependiendo del comportamiento de la grafica de performance, pues si la grafica permanecía estable por demasiadas épocas, no es necesario tuviese tantas épocas, entonces se reduce el valor hasta el mínimo para dar un valor de probabilidad aceptable.

Las nubes de puntos se generan aleatoriamente, cada una de estas nubes conforman una clase de falla, en cada una de ellas contendrá *n* puntos (numero que nosotros especificamos en el volumen al iniciar el cálculo), siendo estos puntos vectores de comportamiento del motor. Estas deben ser inicializadas con los mismos valores, de otra manera es imposible compara los resultados. Para la inicialización se utilizan números llave

(semillas), entre 1 y 3. Así la siguiente etapa es inicializar el programa con diferentes números llave para observar la generalización del entrenamiento.
Los datos provenientes del motor pertenecen a 11 regímenes de operación diferentes. Una vez que hemos logrado reducir el número de épocas, y hemos cambiando los otros valores propios de cada algoritmo de entrenamiento, se disminuye el tiempo de convergencia de la función, en general se ha optimizado la red. Se procede a cambiar el régimen de operación, para observar que estos valores sean validos para diferentes regímenes de operación.
La siguiente etapa es referente a el número de puntos de simulación (volumen de datos), en primera instancia se utilizaron 500 puntos por clase de simulación, en donde cada punto representa un vector (patrón $\delta\vec{V}^*$), se hace el mismo procedimiento pero ahora para 1000, 1500 y 2000 puntos de simulación, para cada clase de cada muestra, con ello observamos si la red sigue arrojando los mismos resultados. Por ende es estable para la utilización con diferentes volúmenes de datos.
La última etapa es hacer el mismo procedimiento para la comparación con redes de tipo radial, haciendo diferentes cambios en los valores propios de la función para lograr un resultado comparable con retropropagación. También se aumento el número de capas ocultas y el número de neuronas por capa para observar el comportamiento de la red, con ello de igual manera redundamos en el objetivo. Por último se compara con los resultados obtenidos de la experimentación con redes de retropropagación.

- **RESULTADOS**

El primer paso, fue analizar las redes sin ningún tipo de optimización eligiendo el número de neuronas al azar y observando el comportamiento, en la Fig. V., curva superior, se puede apreciar la dispersión en los resultados, aunque algunos se acercaban al resultado deseado en cuanto a performance. Ajustando algunos parámetros, como número de neuronas o funciones de activación, se obtuvieron errores aproximadamente iguales, como se puede observar la dispersión de los datos al iniciar la experimentación es demasiado grande, al hacerla optimización se logran resultados aproximadamente iguales (curva inferior Fig. V.), por lo menos en lo la referencia de error. En referencia a tiempo los resultados siguen siendo dispersos.

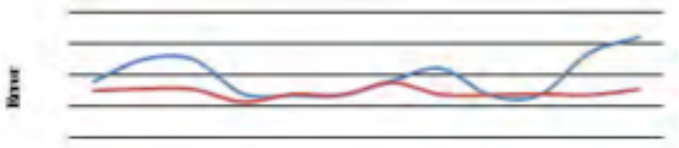


Fig. V. Grafica de desempeño de las diferentes redes neuronales, e base a la suma de errores, sin optimización (curva superior), con optimización (curva inferior).

Al comparar las graficas de desempeño (Fig. V) y probabilidad de aserción (Fig. VI) es obvio la relación inversamente proporcional que tiene error calculado en la red con la probabilidad de aserción en el diagnóstico. Durante la primera etapa de experimentación el volumen de datos fue con solo 500 puntos de operación. Y el primer régimen de operación. Una vez que se ha hecho la optimización de cada procedimiento es posible decir que todas las redes, son capaces de alcanzar la misma probabilidad de aserción (curva superior Fig. VI.).

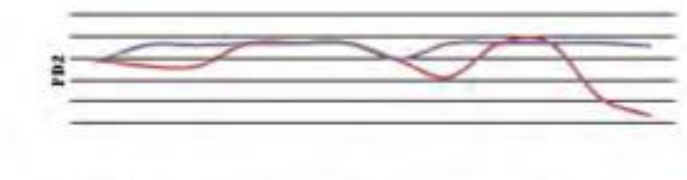


Fig. VI. Grafica de probabilidades, grafica inferior sin optimización, superior con optimización.

Sin embargo, cuando nos referimos a el tiempo de operación, siendo las funciones de clasificación que se muestran en Fig. VII. las que fueron capaces de entrenar la red en menor tiempo esto es en un rango de 30 a 150 segundos. Y como se puede observar el error esta en el rango de 2.5 y 4 %. Hubo algunas redes que tomaban hasta 30 minutos en el entrenamiento solo con 500 puntos por clase.

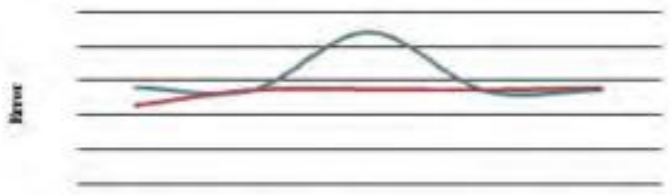


Fig. VII. Redes neuronales con comportamiento estable y error cercano. Régimen de operación 1. Curva inferior con optimización.

Al cambiar el régimen de operación es posible observar y eliminar las redes que tiene comportamiento inestable, como en el caso de traincgb, la cual se comporto excelente con 500 puntos de operación en el primer régimen (Fig. VIII.), y no presenta problemas en los regímenes subsecuentes. Pero sin embargo, presenta problemas al hacer el aumento del volumen de datos, aumenta el error hasta el 35%, el resto de ellos fueron eliminados por mostrar errores de hasta el 40%.

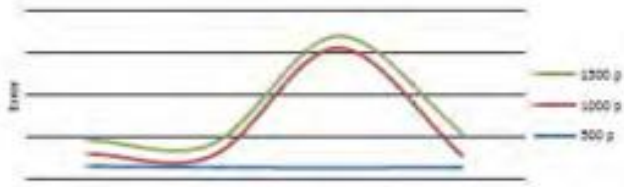


Fig. VIII. Comportamiento con aumento de densidad de datos de análisis. Régimen de operación 3.

345-867

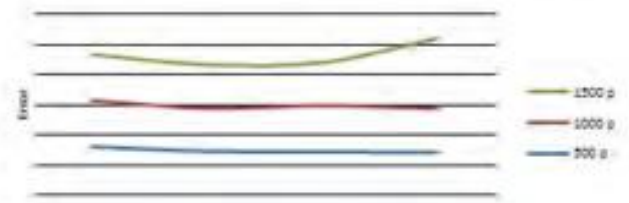


Fig. IX. Comportamiento de error con aumento de densidad de datos de análisis. Régimen de operación 5

Al hacer el aumento del volumen de datos, como se puede apreciar en la Fig. IX., se observo que el error aumenta directamente proporcional al número de datos analizados, aun así no sobrepasando en las redes seleccionadas el 12%. Una vez mas es interesante ver que con el cambio de régimen la función traincgb se comporta de manera estable.

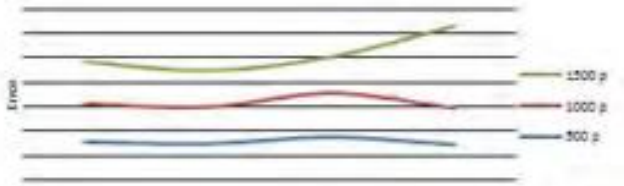


Fig. X. Comportamiento de error con aumento de densidad de datos de análisis. Régimen de operación 11

En los regimenes subsecuentes el comportamiento de las redes es muy parecido. La red entrenada se mantiene estable y aumenta el error directamente proporcional al numero de datos. El tiempo de calculo tambien incrementa en forma considerable. Las redes entrenadas con funciones traincgb y trainscg en algunos regimenes de operación son muy inestables, aumentando el error. También presenta un incremento del error de manera no lineal en relacion con el numero de puntos utilizados para la simulacion. Como se puede observar en la Fig. X el error no necesariamente esta dado por una funcion lineal, por ejemplo en la función trainscg el error es proporcional en el primer aumentode datos, pero al hacer el experimento con 1500 puntos el error aumenta practicamente el doble del calculo anterior.

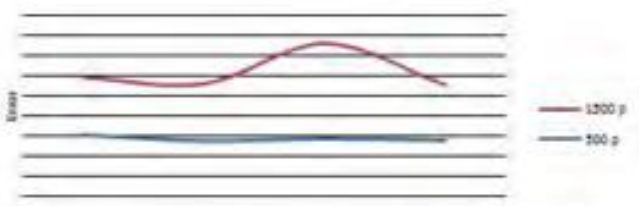


Fig. XI. Comportamiento de error con aumento de densidad de datos y cambio de semilla. Régimen de operación 9.

Como se muestra en la Fig. XI. al analizar el cambio de llave de inicialización de datos es posible observar, que la mayoría de redes siguen teniendo la misma tendencia. El resultado no depende del número de inicialización (semilla), sino del tipo de entrenamiento que esté recibiendo la red, siendo una vez más traincgb la que presenta una fluctuación mayor en comparación con los otros tipos de entrenamiento.

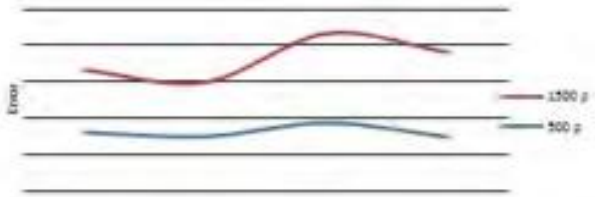


Fig. XII. Comportamiento de error con aumento de densidad de datos y cambio de semilla. Régimen de operación 11

En la Fig. XII, demostramos que el cambio de regimen no tiene impacto en el entrenamiento de la red, una vez mas es posible observar que el comportamiento del error depende enteramente del numeros de puntos de operación.

as redes neuronales de tipo retropropagacion demostraron un comportamiento bastante confiable, y en cuanto a tiempo demostraron ser capaces de dar un diagnóstico en tiempo muy corto, pues el tiempo en la fase de entrenamiento no va mas alla de los 2 minutos.
Con las redes del tipo radial, al hacer la configracion basica demostraron dos tipos de comportamiento. Primero, si no se especifica el numero de neuronas de la red, esta toma el tamaño de la matriz y utiliza el numero de columnas, asi pues, esta no tiene la capacidad de hacer calculos con grandes volumenes de datos. Segundo, si se reducen el numero de puntos es posible que esta reduzca de igual manera el numero de neuronas, y aunque es capaz de realizar el calculo, no tiene una probabilidad adecuada. Como se muestra en la Fig. XIII., se reduce el error a aproximadamente 7 % (SSE), pero se presenta el fenomeno de sobre entrenamiento, al hacer la prueba esta solo tiene una probabilidad de dar un diagnóstico correcto del 30%, y el tiempo de entrenamiento se ve severamente castigado, tomando aproximadamente 78 horas para el entrenamiento.

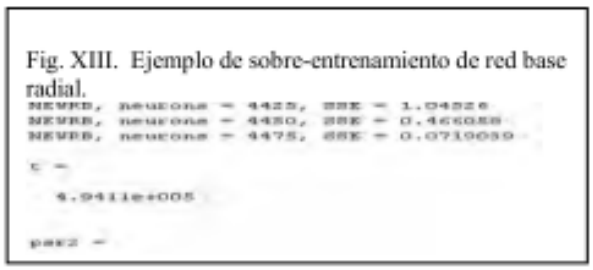


Fig. XIII. Ejemplo de sobre-entrenamiento de red base radial.

Mas sin embargo, es posible disminuir el tiempo de operación y de igual manera aumentar ella probabilidad de aserción, incluso al aumentar el número de puntos de simulación la red neuronal es capaz de dar una probabilidad de acierto considerable, la cual ya es comparable con los resultados arrojados por las redes de retropropagación. Trabajando con número de neuronas en los rangos de 15 y 20 se pudo alcanzar el tiempo y la probabilidad deseada para ser comparada con retropropagación. Como se puede observar en la Fig. XIV. La red de base radial tiene pequeñas diferencias en cuanto a probabilidad, aún así ya es capaz de acertar en el diagnóstico y el tiempo de entrenamiento de la red es comparable con el de las redes de tipo retropropagación.

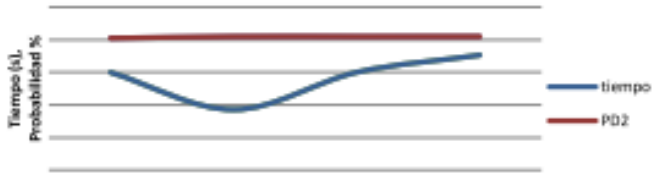


Fig. XIV. Comparación en tiempos de entrenamiento (s) y probabilidad de aserción de diagnostico (%) de diferentes tipos de redes neuronales.

- **CONCLUSIONES** En este articulo hemos descrito la experimentación de un algoritmo utilizado para el diagnóstico paramétrico de turbinas de gas basado en redes neuronales artificiales. Se utilizó un modelo termodinámico no lineal para simular los modos de degradación de la turbina de gas y las fallas de clasificación. Obtenemos y verificamos los índices de diagnóstico adecuados que permiten decir si nuestra turbina puede ser diagnosticada correctamente. Es importante recordar que las mediciones que fueron tomadas no se utilizan directamente sino que utilizamos desviaciones para simular fallas reales, con ello introducimos al modelo diferentes fallas y se considera el ruido en patrones simulados. Se genera la nube de puntos de cada clase, en donde cada punto representa un vector de desviaciones inducidas por fallas simuladas (patrón).
-

La experimentación en este tipo de algoritmos puede llegar a ser muy larga, dependiendo del tipo de procesador se puede aumentar o disminuir los tiempos de operación. Siendo estos experimentos los que conforman la base de investigaciones posteriores, pues ahora somos capaces de decir cuál es la red neuronal que tiene mejor comportamiento para el tipo de datos que se manejan, y el volumen de las bases de datos, sabemos que son datos que no pueden tener mucha variación, también sabemos que este tipo de redes

neuronales son capaces de eliminar el ruido dentro de los datos medidos, así podemos utilizar estas como interface para poder diagnosticar en tiempo real una turbina en operación.
Hay cuatro funciones (trainrp, traiscgb, traiscg, newrb) que presentan un comportamiento similar en el entrenamiento. De ellas, tres fueron funciones de entrenamiento de perceptrón y una es la red con RBF por base radial en su forma básica. La diferencia principal es en el tiempo de entrenamiento (Fig. XIV), pues mientras una fue capaz de analizar, clasificar y diagnosticar en solo 38 segundos, otra tomo 70 segundos, siendo esta diferencia un tanto indiferente (considerando solo 500 puntos de operación). En lo referente a la probabilidad de diagnostico correcto, estas mostraron un comportamiento estable, tanto en el cambio de régimen, como en el aumento de número de puntos de simulación. Siendo esta probabilidad entre 81 y 83%.

Agradecimientos

- Al Instituto Politécnico Nacional, por el apoyo económico a través del proyecto de investigación No. 20091273.

Referencias

- Gutiérrez Mojica Eder Martin, 2005.-Tesis: "Diagnóstico y monitoreo de los patrones de una turbina de gas mediante redes neuronales artificiales", Instituto Politécnico Nacional, México.
- Howard Demuth 2008, Martin Hagan, Mark Beale, Neuronal Network Toolbox 6, *Mathworks 2008.*
- Loboda I. 2001, E. L. Santiago, "Problems of gas turbine diagnostic model identification on maintenance data", Memorias del 6 Congreso Nacional de Ingeniería Electromecánica y de Sistemas, ESIME-Zacatenco, México, 2001.
- Loboda I. 2003, Trustworthiness of gas turbine diagnosing on transient regimes. Memorias del XVIII congreso de Instrumentación, SOMI, Ciudad de México D.F, México, 2003.
- Manual de redes neuronales, 2005, Universidad tecnológica de Pereira, http://ohm.utp.edu.co/neuronales, consultado el 25 agosto 2005.