Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

PropSim 0.1

TITULO DEL TFG: Modular numerical tool for gas turbine simulation

TITULACIÓN: Grado en Ingeniería de Aeronavegación.

AUTOR: Rodrigo Sampedro Casis.

DIRECTORES: Fernando Mellibovsky.

DATA: 20 de Octubre del 2015.

Título: Simulador modular de turbinas PropSim 0.1.

Autor: Rodrigo Sampedro Casis.
Directores: Fernando Mellibovsky.

Fecha: 20 Octubre de 2015.

Resumen

En este trabajo se ha implementado una herramienta de simulación gratuita sobre turbopropulsores, capaz de simular con diferentes grados de modelación termodinámica o complejidad, componentes de un motor a reacción, así como la combinación de los mismos para formar un motor completo.

La principal característica de este software es su compatibilidad, código abierto y licencia GNU, inexistente en el mercado actual. Además, se ha realizado una flexibilidad y un entorno de trabajo más adecuado a las necesidades del usuario. Para la realización del mismo se ha utilizado Java, sus librerías graficas Swing y librerías matemáticas GNU como JBLAS (librerías BLAS en Fortran accesibles desde Java) todo ello gestionado automáticamente por la herramienta de apache MAVEN usando el entorno de desarrollo Eclipse.

La flexibilidad del programa permite a un usuario simular e interconectar diferentes componentes existentes en las librerías, sin la necesidad de conocer, modificar y compilar código, ya sea a través de un pseudo lenguaje o de la interfaz gráfica.

Por otra parte, se ha implementado una fuerte y compleja estructura jerárquica de objetos basados en herencia, que permiten expandir mejorar o añadir nuevos módulos a las librerías de componentes, mediante nuevas clases heredadas, sin la necesidad de un conocimiento profundo del código y evitando la necesidad de modificar clases del programa original.

El software predispone de varios modelos termodinámicos principalmente basados en las asignaturas de Termodinámica y Propulsión EETAC-UPC. Así como la implementación de Soluciones Numéricas Discretas, por un solver de ecuaciones no lineales, propio con diferentes métodos numéricos descritos en el documento. Para la realización del mismo ha sido necesario un conocimiento profundo de los modelos implementados; así como de los métodos matemáticos para resolverlos, con el objetivo de optimizar la carga computacional de la solución.

Finalmente se hace énfasis en que este TFG es sólo la primera parte de un conjunto de 2 TFG realizados por el mismo autor. En este documento se describe la versión Alfa del software, fundamentos del mismo y cuyos resultados están enfocados a la mejora e incremento de realismo de la versión Beta del software que se está desarrollando durante curso 2015-16.

Title: Modular numerical gas turbine simulator PropSim 0.1

Author: Rodrigo Sampedro Casis

Directors: Fernando mellibovsky.

Date: October 20th 2015

Overview

In this work a free tool for the simulation of turboprops was implemented, capable of simulating the various components of a jet engine, separately or in conjunction, with different degrees of thermodynamic modelling or complexity, in order to simulate an entire jet engine.

The main characteristics of this software includes its compatibility, open code and GNU license, non-existing in today's market. Furthermore, the tool was designed with a greater flexibility and a more adapted work environment tailored to the needs of the users. To design this tool, Java was used, along with its graphical Swing libraries and mathematical GNU libraries, such as JBLAS (BLAS libraries in Fortran available from Java). This was all automatically managed with the apache tool MAVEN and using the Eclipse development environment.

On the other hand, a complex and robust hierarchical structure of inheritance based objects was implemented, which allows the expansion or the addition of new modules to the component libraries through new inherited classes. This is an interesting feature since a deep knowledge is not required to apply these changes, avoiding the need to modify the classes of the original program.

The software pre-disposes various thermodynamic models, based on the Thermodynamic and Propulsion subjects taught at EETAC-UPC. It also contains the implementation of Discrete Numerical Solutions, with a self-made nonlinear equations solver, based on different numerical methods described within this document. With the objective in mind of optimizing the computational load, a profound knowledge of the implemented models, as well as the mathematical methods for solving them, was required.

Finally, it is important to emphasize the fact that this final year project is only the first part of a set of two undergraduate theses done by the same author. This document can be considered the alpha version of the software, its basis and its results are focused on the increase and improvement of the beta version of the software's realistic conditions, which is being developed during the academic year 2015-2016.

Dedicado a todos mis compañeros y profesores durante estos últimos 5 años donde hemos compartidos buenos momento. Especialmente en las clases de termodinámica y propulsión.

Prólogo

Entre los años 2009 y 2014 he completado satisfactoriamente una simultaneidad de estudios en dos especialidades de Telecomunicaciones, Ing. Sistemas de Telecomunicación e Ing. Telemática, junto a la especialidad de Ing. Aeronavegación todas en la escuela EETAC-UPC en Castelldefels.

A finales de 2013 se me informó que ante la nueva planificación de estudios de dobles titulaciones que ha empezado a impartirse el curso 2015-16, se me permitían unificar en un solo TFG los respectivos proyectos a realizar para cada uno de los grados universitarios. Sin embargo, al no pertenecer a dicho plan de estudios, los créditos totales a matricular son 24x3 ECTS obteniendo como único beneficio la presentación de un documento y una única evaluación.

A comienzos de 2015, mientras desarrollaba mi trabajo final de carrera centrado en la temática GNSS y SBAS, tuve la percepción de enfocar únicamente dicho trabajo al ámbito de la telecomunicación, lo que me animó finalmente a realizar otro TFG donde plasmar aquellas ideas o conceptos que más me habían gustado durante la especialidad aeronáutica.

Fernando Millebovsky, me comentó la existencia de varios proyectos de Ing. Técnica donde desarrollaron varios software de turbomotores. Finalmente tras analizar la documentación previa he optado por realizar este TFG con el objetivo de desarrollar un software ambicioso, que fúndase las bases y contuviese los trabajos previos, con el objetivo de lograr un producto final realista, útil de cara al alumnado como investigadores basado en un producto GNU fácilmente extensible y personalizable.

Ante la dimensión y volumen del desarrollo del software, he decidido presentar este TFG (aeronavegación) como los fundamentos teóricos y conceptuales junto a una primera versión del software denominada 'Alfa'. Para posteriormente continuar con el desarrollo del software durante el año 2016 obteniendo la versión Beta como TFG de Telemática.

Por lo tanto este trabajo es la primera entrega, que explica las bases tanto teóricas y conceptuales aeronáuticas como los esquemas y estructuras básicos de la programación del software que van a condicionar la mejora del mismo; así como determinar su flexibilidad y eficiencia como producto final.

Los resultados han sido principalmente enfocados a la mejora de puntos críticos detectados durante el desarrollo del mismo, y a evaluar estrategias de cara a la implementación de modelos o técnicas que aumenten el realismo de las simulaciones sin incrementar exponencialmente los recursos computacionales necesarios durante la versión Beta.

Concluyo indicando que este TFG se centra en la visión aeronáutica del propio software, las estructuras y bases del mismo, que serán completadas con una segunda parte más especializada en pulir la eficiencia del software, así como sus interfaces humanas e incrementar el número de componentes disponibles.

Índice de Acrónimos

EES	Engeneering Equation Solver (Solucionador de ecuaciones de ingeniería)
FORTRAN	Formula Translating System
GNU	Licencia general pública
IDE	Integrated Development Environment (Entorno de desarrollo)
ISP	Impulso Especifico
JAVA	James Gosling, Arthur Van Hoff, yAndy Bechtolsheim (Lenguaje de programación)
JBLAS	Java BLAS library
JUnitTest	Java Unit Test
MAVEN	Herramienta de gestión de software de Apache
MVC	Modelo Vista Controlador
PFC	Proyecto Final de Carrera
POO	Programación Orienta a Objetos
SWING	Biblioteca Grafica de Java
TFG	Trabajo Final de Grado

Índice

Introducción

En la actualidad existen todo tipo de aeronaves como aviones, helicópteros y otros vehículos aéreos en el mercado, cuya tendencia mayoritaria, debido a la demanda globalizada de transporte rápido y eficiente, está en continuo crecimiento. Para poder propulsarse utilizan diferentes tipos de motores donde las principales tecnologías son la propulsión por hélice o por jet; las avionetas usan motores a pistón, los helicópteros turbinas, pero en ambos casos se basan en propulsión por hélice, por otra parte los aviones comerciales o militares utilizan tecnologías jet como turbofanes o turbojet.

Mientras que para los requisitos más bajos se utilizan propulsión basada en hélice con motores de combustión, cuando es necesario mejorar las velocidades o se necesitan potencias superiores como en los helicópteros se recurre a motores basados en turbinas de gas o turbo-ejes. Si los requisitos del motor están en velocidad transónica es necesario utilizar la tecnología jet por los problemas asociados a compresibilidad de las hélices, como son los turbofanes. Para velocidades supersónicas únicamente existen turbofanes especiales o turbojet. Finalmente a velocidades extremas se utilizan ramjets o scramjet o directamente motores cohete, con aplicaciones estrictamente militares o científicas.

Este trabajo se basa en el entendimiento, modelado y simulación de turbomotores, es decir, motores basados en una turbo máquina. Como ha quedado patente la mayor parte del transporte aéreo se sostiene bajo el uso de aplicaciones asociadas a una turbina además tiene otros usos extendidos como la generación de electricidad o potencia mecánica en el ámbito industrial.

Es de especial interés disponer de herramientas de simulación que permitan optimizar, diseñar y entender diversos tipos de turborreactores, así como los componentes que los forman.

Actualmente existen varias herramientas informáticas; sin embargo la gran mayoría está constituida por programas de licencias privadas, es decir, código cerrado y con un coste considerable. Por otra parte una gran variedad de este tipo de software no está en el mercado o no se puede adquirir al ser producción propia de los principales fabricantes.

En este trabajo se explica y modela el funcionamiento de los turborreactores y se desarrolla un software libre y gratuito que actualmente no está disponible en el mercado, con el fin de servir como base a programas customizados o como herramienta didáctica para la comprensión, estudio y análisis de turbo máquinas.

El documento posee la siguiente estructura:

- Capítulo 1: Objetivos y requisitos del software.
- Capítulo 2: Introducción e ideas básicas de turborreactores.
- Capítulo 3: Tipos de turborreactores y aplicaciones.
- Capítulo 4: Software estructura e ideas.

- Capítulo 5: Descripción y resumen de los resultados obtenidos.
- Capítulo 6: Conclusiones.
- Anexos: Los anexos A y B tratan sobre la teoría utilizada en este documento. En los anexos C y D explican al detalle la planificación y métodos de programación utilizados para su implementación.

Este trabajo se ha realizado, con la perspectiva de que sea útil para la continuidad del desarrollo del software, por lo que contiene continuas referencias a los anexos para un mayor entendimiento por parte del alumnado interesado en turborreactores, así como en la modificación del propio software para usos customizados.

Si el lector ya dispone de unos conocimientos amplios en la materia de sistemas termodinámicos y/o propulsión, se le recomienda realizar una lectura rápida de los capítulos 2 y 3, desechando los anexos A, B, C en los que se explican los fundamentos y definiciones teóricas utilizadas.

1. Objetivos y requisitos

En este capítulo se describen los objetivos y requisitos del trabajo desarrollado. Para visualizar la planificación y gestión del proyecto véase el Anexo D 1.1-3.

1.1 *Objetivo del TFG*

El objetivo de este trabajo es la elaboración de un software de simulación de turbo máquinas, orientado a los motores de reacción. La elaboración debe ser de creación propia, sin hacer uso de librerías o lenguajes propietarios, para que pueda ser difundido bajo licencia GNU.

El software en su versión minimalista debe permitir resolver motores o componentes de los motores, con igual o mayor realismo que los ejercicios de asignaturas de Termodinámica y Propulsión impartidas en la EETAC o ejercicios similares al Libro Termodinámica [1]. Con el propósito de permitir una modelación progresiva, se definen grados de modelización con mayor o menor realismo.

1.1 *Antecedentes*

La no existencia de software de simulación GNU y la escasez de software gratuito multiplataforma con modelos complejos ha permitido a alumnos previos de Ing. Técnica [2] bajo la tutela del mismo director de este trabajo, la realización conjunta de 3 PFC (Proyecto Final de Grado) independientes de Ingeniería Técnica Aeronáutica, con el fin de desarrollar una herramienta de análisis para turbinas de gas. Cada alumno se especializo en el modelado y programación de diferentes componentes, finalizando y presentado sus TFG (Trabajo Final de Grado) durante verano de 2012 [2].

Sin embargo, se obtuvo la creación de 3 herramientas independientes escritas en C++ en entorno de consola, únicamente funcionales bajo plataforma Windows y sin la integración de los diferentes componentes, necesaria para simular un turbojet o turbofan en su conjunto.

El software fue diseñado bajo programación orientada a objetos, no era flexible ni hacia uso de herencia, principal ventaja de la POO (Programación Orientada a Objetos). Únicamente permitía simular componentes concretos y carecía de una interfaz humana amigable. No permitía la interconexión de elementos sin actuación directa sobre el código y no se hizo pública la totalidad del código ni el software compilado tras la presentación de los TFC.

Finalmente ante mi disposición de realizar un simulador de propulsión se planteó la posibilidad de continuar el trabajo iniciado en 2012 o rehacerlo, eliminando los impedimentos de diseño para poder simular de manera flexible componentes o turborreactores completos, en una herramienta GNU cuyo principal uso fuese en Linux.

1.2 *Requisitos del Software*

Como feedback de las experiencias de los PFG de 2012 y los resultados obtenidos se plantearon los siguientes requisitos de cara a la creación de un nuevo simulador:

- El software ha de ser GNU, libre y gratuito.
- El software debe ser fácil de orientar a multiplataforma.
- Se desea una compatibilidad en los sistemas operativos mayoritarios, pero su principal uso será a través de sistemas Unix-Linux.
- El software debe contener un modo de simulación en consola y una interfaz visual que permita la creación y modificación de diseños fácilmente.
- Se valora ampliamente la eficiencia del mismo, por la gran cantidad de operaciones matemáticas para la realización de las simulaciones.
- El software debe contener un entorno grafico para el diseño o modificación de los proyectos.
- Se valora ampliamente la detección de errores humanos en la interfaz visual.
- El software debe estar escrito en POO, con abundantes comentarios y estructuras definidas que permitan una fácil y rápida comprensión del código.

1.3 *Desarrollo del Software*

Como decisiones escogidas para la realización del software y en base a los requisitos del mismo se ha optado por utilizar como lenguaje JAVA por su capacidad de multiplataforma y compatibilidad con la gran mayoría de sistemas operativos además de las librerías asociadas SWING (visual) o math son librerías GNU.

Para evitar el cuello de botella de la “ineficiencia matemática” de JAVA en comparación con otros lenguajes de programación compilados como C++ se dispone de las librerías JBLAS, que permiten ejecutar las operaciones matemáticas compiladas en FORTRAN desde una api en JAVA.

Para evitar futuros problemas y facilitar la traducción de JAVA hacia otros lenguajes como C++; no se han hecho uso de recursos inexistentes en C++ con el fin posibilitar la traducción del código CORE y CONSOLA a C++, manteniendo las interfaces visuales de JAVA.

JAVA es uno de los lenguajes de programación más orientados a POO, que junto al IDE Eclipse, permite una programación rápida y fluida. Auto gestiona la memoria, eliminado la posibilidad de errores en el uso de los recursos, de gran problemática en simuladores o programas iterativos y permite fácilmente la transformación en servicio web, online o bajo aplicación Android del software.

Finalmente se ha utilizado la implementación de JUnit test que permiten asegurar el correcto funcionamiento del software y facilitar el desarrollo del mismo sin la generación de nuevos bugs.

Como elemento de trasparencia y seguimiento se ha utilizado Github que sirve de plataforma para el control de versiones, servidor, reportes de bugs y expositor del software GNU desarrollado de manera pública y en tiempo real.

2. **Conceptos Básicos.**

Una turbo máquina se define como una máquina térmica, su principal función es extraer o trasferir energía y su principal elemento es un rotor giratorio, por el cual pasa un fluido de forma continua. Se trata de un diseño basado en un eje y un sistema de alabes de formas diversas donde se extrae o transmite energía a un fluido compresible, normalmente aire, empleando energía cinética, térmica o presión.

Se puede diferenciar un motor convencional de una turbo máquina en base a que la turbo máquina es capaz de extraer y trasmitir energía en un ratio potencia o empuje en relación a peso mucho mejor.

Comparando un motor a reacción de avión con un motor de pistón, la eficiencia y capacidad de producción de potencia de la turbina son muy superiores; así como un menor consumo-ratio. Sin embargo, los reactores necesitan de una complejidad mayor y tienen graves problemas de mantenimiento como son la lubricación y el control de temperatura.

2.1 ***Turborreactores***

La principal función de un propulsor es generar impulso, para ello los aviones hacen uso de un juego compresor-eje-turbina que permite impulsar grandes cantidades de aire y a gran velocidad (generan cantidad de movimiento).

El compresor es una turbo máquina capaz de trasferir energía a un fluido en forma de presión y temperatura, la turbina es una turbo máquina capaz de extraer energía de un fluido disminuyendo su temperatura y presión.

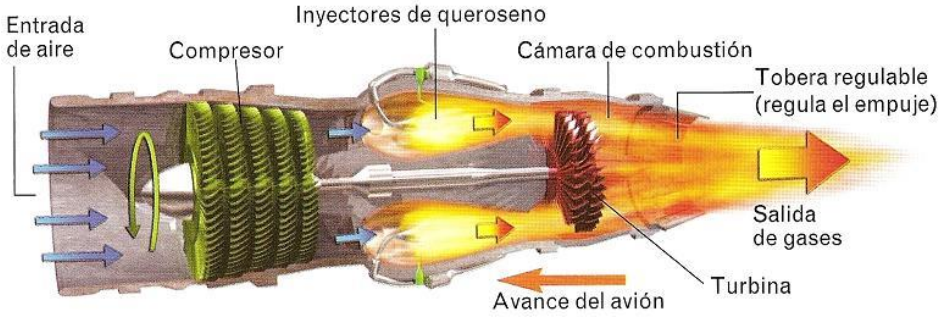


Fig. 1 Esquema básico Turborreactor [11].

Si se combinan ambas, uniendo sus rotores mediante un eje, se obtiene un elemento mecánico capaz de comprimir-descomprimir, calentar-enfriar el aire con balance trabajo neto cero. Sin embargo, cuando realizamos una combustión entre los dos elementos, se aporta energía en forma de calor al fluido. Este calor es transformado por la turbina, como potencia extra que es transmitida mediante el eje al compresor quien finalmente sí genera un remanente de presión en los gases de la combustión del que se obtiene el empuje del motor. Véase Fig. 1,

Por lo tanto, gran parte de la energía transmitida en forma de calor al fluido, se transforma en presión. Esta presión extra después de la turbina produce una fuerza neta que propulsa el aire fuera del motor e impulsa hacia delante el motor por reacción. Sin embargo, la aplicación de geometrías tales como toberas permite convertir presión en velocidad mejorando la conversión en energía cinética en formas de cantidad de movimiento. El uso de la tercera ley de newton (acción-reacción) para propulsarse permite denominar a los motores de aviones como “moto reactores” o “Propulso reactores” por basarse en la reacción del fluido que empujan en la dirección contraria a la de vuelo.

2.2 *Ciclo termodinámico*

Se recomienda una lectura del Anexo A, si se carece de conocimientos termodinámicos para entender bajo qué condiciones y definiciones se analiza termodinámicamente un reactor.

Un proceso termodinámico es un cambio de un estado de equilibrio a otro estado de equilibrio, experimentado por un sistema. Durante el cambio de estado se produce una absorción/extracción de calor/trabajo. Se denomina ciclo a la trayectoria descrita en un diagrama termodinámico, cuyo punto final y el inicial son comunes. El trabajo/calor del ciclo es el área del polígono formado durante la trayectoria en el diagrama (en las unidades adecuadas).

Podemos diferenciar los procesos básicos identificados en la figura 2 como:

- 1-2 Difusor-Compresión: Convierte la energía cinética del fluido en presión y transmite la energía del eje al fluido aumentando la presión, durante el proceso el fluido se calienta. Proceso Adiabático, se obtiene aire a alta presión.
- 2-3 Combustión: Se añade un combustible y se incendia, incrementando ampliamente la temperatura del fluido. Proceso Isobárico, se obtiene aire a alta presión y alta temperatura.
- 3-4 Expansión: La turbina reduce significativamente temperatura y presión para alimentar a través del eje al compresor. Proceso adiabático, se obtiene aire a presión media, temperatura media y movimiento en el eje.
- 4-1 Expulsión: El gas debido a la diferencia de presión produce un empuje. Normalmente se predispone de una tobera que continúa haciendo una expansión geométrica que acelera los gases transformando su energía en cinética, con el objetivo de que el aire salga a una presión similar a la exterior, una temperatura baja (pero más alta que la exterior) y una gran velocidad.

Se puede decir que un turborreactor encendido se autoalimenta y mientras más combustible se vierta en la cámara de combustión mayor es el empuje del mismo. No obstante, si está apagado, no es capaz de encenderse por sí mismo.

Además, el eje debe girar a altas revoluciones y toda la estructura se encuentra acosada por la presión y temperatura. Por ello, se puede deducir que aunque se trata de un tipo de maquina muy eficiente, necesita de complejos sistemas de

encendido, lubricación y refrigerado, por lo que no es práctico a pequeña escala, así mismo la potencia proporcionada tiende a ser estable sin grandes cambios.

“Muy práctico para un transporte de gran potencia y velocidad constante como es un avión, poco práctico para un automóvil genérico con cambios constantes de velocidad”.

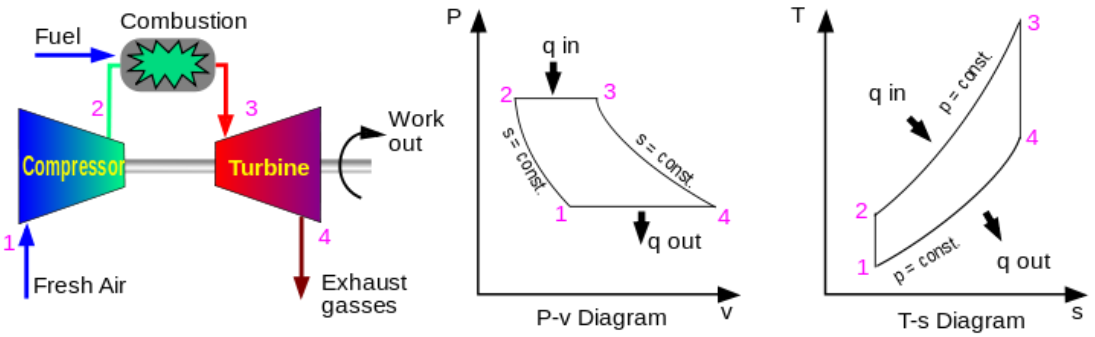


Fig. 2 Diagrama P-v T-S del cicló termodinámica de Brayton

Un turbo reactor, desde un punto de vista termodinámico, describe el denominado ciclo de Brayton. Para modelar de manera simplificada el proceso del ciclo se asume la idealidad teórica de los diversos componentes del motor, es decir, procesos adiabáticos (sin trasferencia de calor) o procesos reversibles (entropía constante) y asumiremos que todo existe dentro de un sistema abierto con flujos estacionarios.

En la Fig. 2 se puede observar las dos gráficas del proceso de un reactor, el diagrama Presión-Volumen específico o el diagrama Entalpia-Entropía. A veces se utiliza temperatura en vez de entalpia al asumir propiedades constantes y ser más sencillo de representar. En ellos se aprecia que los 4 pasos mencionados anteriormente se cumplen con las características mencionadas.

El área encerrada dentro del polígono es el trabajo realizado por ciclo para maximizarla, es decir, para aumentar el trabajo por unidad de masa, la única manera efectiva consiste en aumentar la entalpia (temperatura) y presión máxima soportable por la estructura, debido a que la presión atmosférica limita la línea baja 1- 4 en ambos ciclos.

La temperatura y la presión de la cámara de combustión limitan el trabajo capaz de desarrollar el motor. Concretamente la limitante es la temperatura ya que es la que afecta más negativamente a los materiales, degradando sus características o deformándolos por termo fluencia. Existen complejos sistemas de refrigeración, pero las mejoras extraídas son escasas en comparación con el aumento excesivo de la complejidad.

Por lo tanto, la potencia generada por un ciclo de Brayton queda limitada por la temperatura de combustión, que a su vez está limitada por la tecnología de materiales, la única manera de aumentar el trabajo producido sin la creación de nuevas tecnologías es aplicar un mismo ciclo a un volumen de masa mayor, es decir, dimensionar físicamente un motor más grande.

3. Tipos de motores y características

La gran mayoría de los motores a reacción tienen componentes y estructuras muy similares, pueden ser catalogados por sus características de uso que están muy relacionadas con su evolución histórica.

Existen motores basados en turbinas de gas, como Turbojet, Turbofan Turbopop y otros basados en otro tipo de ingestión de aire como son Pulsojet, Ramjet o Scramjet. Todos ellos contienen conceptualmente la misma idea aplicada a diferentes características del flujo de aire.

3.1 *Pulsojet*

El Pulsojet fue un experimento previo alemán (1920) a la invención de los motores a reacción basados en turbina. Se basa en un motor cohete, es decir, en la combustión y expansión de gases (Fig. 3 y 4).

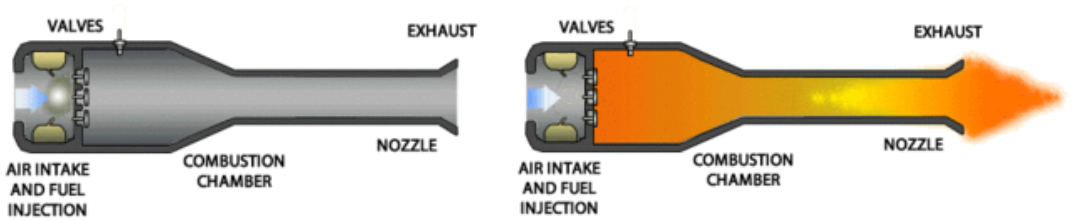


Fig. 3 Esquema de funcionamiento Pulsojet.

Su desarrollo prematuro se debe a su sencillez, carece de eje, turbina o compresor. Se instala una rejilla que únicamente deja pasar el aire en sentido difusor-tobera, se realiza una combustión interna, la expansión de los gases se expulsa a través de la tobera por que la rejilla impide su salida. La expansión acelera los gases a través del tubo y la tobera creando un vacío en la cámara de combustión. El vacío o presión negativa creada abre la rejilla, absorbe aire nuevo del difusor cerrando el ciclo.

Fig. 4 Pulsojet en un V2 [12].

Este tipo de motor, no sigue el ciclo de Brayton sino el ciclo de Lenoir y es mucho menos eficiente, disipando la mayor parte de la energía como pérdidas térmicas,

además tiene un alto consumo de combustible. Sin embargo, fue el motor empleado en aviones y los primeros misiles como la V2 (Fig.4), utilizada por los nazis durante la segunda guerra mundial.

A su favor, su carencia de elementos, lo hace un motor fácil de construir y de mantener; aunque actualmente, debido al alto precio del petróleo su uso es anecdótico. Inicialmente no se simula dicho tipo de motor en el software, pero es completamente factible la simulación del mismo con la creación de un módulo de conducto de expansión y el uso de una cámara de combustión convencional ya existente en el software.

3.2 *Turbojet*

Los motores turbojet, son la base de la aeronáutica a reacción, se inventaron y desarrollaron a finales de los años 30, principios de la segunda guerra mundial aunque su uso extendido no llego hasta principios de los años 50, debido a sus requisitos altos en tecnología de materiales.

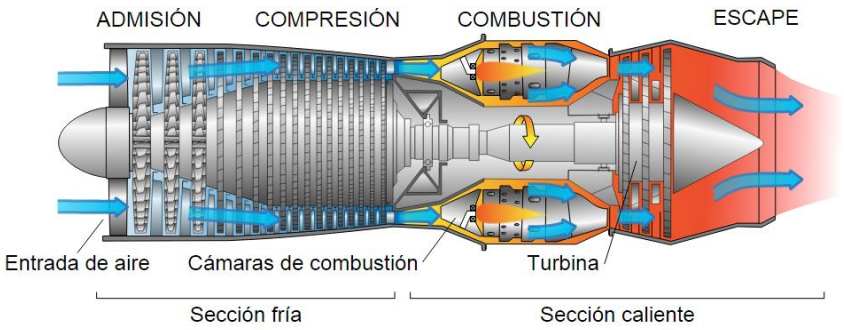


Fig. 5 Estructura de Turbojet Axial.

Se fundamente en el esquema más básico y utilizado como ejemplo durante la introducción y análisis termodinámico de motores a reacción (Fig. 5). Consta de un difusor, compresor, cámara de combustión, turbina, en algunos casos poscombustor y una tobera.

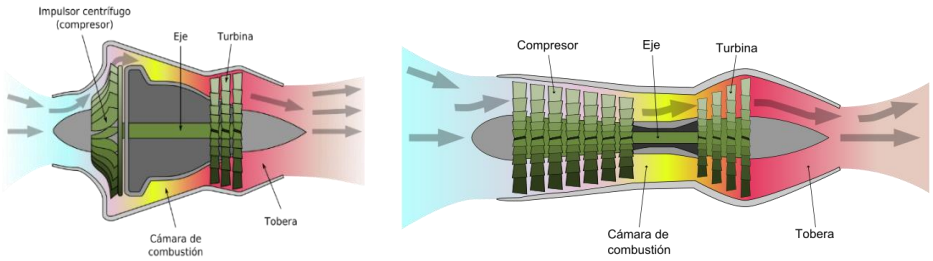


Fig. 6 Tecnología Centrípeta (izquierda), tecnología axial (derecha).

Desde sus inicios en la segunda guerra mundial, existen dos diseños diferentes para el juego compresor-turbina, basado en una tecnología de eje axial (alemana) u otra tecnología centrípeta (británica) véase Fig.6. Siendo la configuración Axial mejor para alcanzar altas velocidades en los gases expulsados, pero con una mayor complejidad en diseño y materiales utilizar.

Fue muy empleado durante la guerra fría ya que su reducido tamaño, gran potencia y alcance de velocidades hasta mach 2 (2 veces la velocidad del sonido) y propicio un gran desarrollo del diseño con aplicaciones en aviones comerciales a partir de 1949 hasta finales de la década de los 90.

Actualmente ambas configuraciones “puras” únicamente se encuentran disponibles en generadores mecánicos o eléctricos o en diseños o aviones obsoletos ya que tiene una eficiencia inferior y un consumo mayor a los motores más modernos. En aviones comerciales dejaron de utilizarse desde mediados de los 90, siendo anecdótica la visión “alargada” de los motores en los aviones anteriores al año 2000, véase Fig.7.

Fig. 7 Avión comercial basado en turbojet [13].

Es la principal implementación en el simulador, ya que permite evolucionar en el resto tipo de motores modificando o agregando nuevos módulos.

3.3 *Turbohélice o Turboprop*

Son la base de los actuales aviones regionales, su evolución se basa en la utilización de un turbojet como generador de rotación de una hélice y permite el objetivo de ahorrar combustible para velocidades subsónicas.

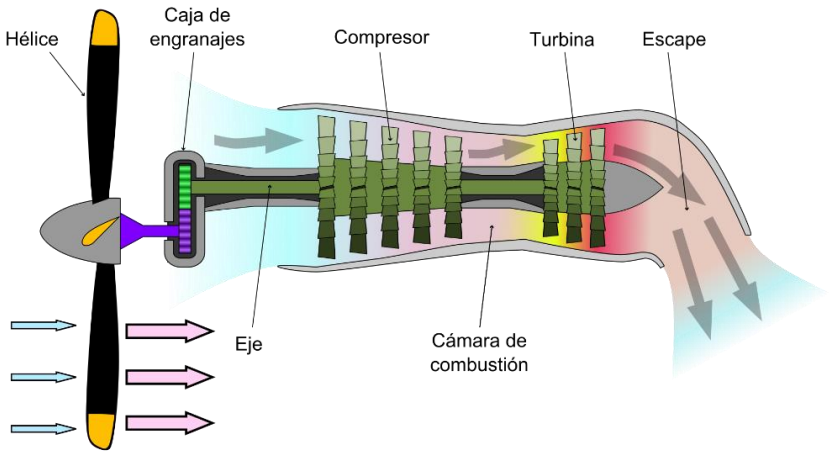


Fig. 8 Estructura de Turboprop o Turbohélix.

Se basa en implementar una caja de cambios que ralentiza las revoluciones del eje para mover una hélice. Permite transmitir la potencia a una hélice que es la que genera el 90% del impulso, mientras que el 10% restante es de los gases de escape. Su principal cambio es la utilización de una turbina que extrae toda la energía posible del fluido, ahorrando hasta un tercio de combustible y despreciando el aporte de empuje del mismo por escape de los gases.

Muy eficiente en consumo, está limitado a velocidades reducidas 400-600 km/horas y a alturas más reducidas, debido a la naturaleza y problemas a velocidad supersónica de las hélices. Por otra parte también genera problemas de ruido, son motores muy útiles para vuelos regionales o transporte de mercancía (Fig. 11) y se han popularizado desde finales de los 90.

Fig. 9 ATR-72, motor equipado con Turboprop [14].

3.4 *Turboejes*

Prácticamente idénticos a un turbohélice, posibilitan tener independiente el giro del rotor del motor, al eje de transmisión de potencia mecánica, permitiendo operar de una manera más independiente a la potencia requerida. Es la principal implementación en helicópteros o aviación amateur de aeródromos.

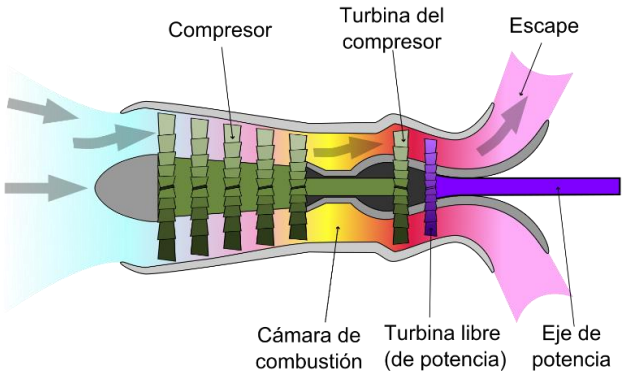


Fig. 10 Estructura de Turboeje.

Curiosamente su existo más extendido es como turbina de generadora de movimiento en barcos, locomotoras y hasta carros de combate. Lo que puede permitir otros usos de este mismo software de carácter no aeronáutico.

3.5 *Turbofan*

El turbofan es la evolución coherente del turbojet. Su principal característica es un menor consumo de combustible y un mayor empuje.

A diferencia del turbojet, cuya principal fuerza propulsora son los gases calientes a reacción, el turbofan define dos flujos de aire. Flujo minoritario (core) que pasa a través de un turbojet y un flujo mayoritario que es impulsado por un fan; un compresor de grandes dimensiones y pocas etapas que es hecho girar por el mismo eje que el turbojet.

Se puede decir que un turbofan no es más que un turbojet donde el eje del turbojet en vez de alimentar mecánicamente a una hélice genera un flujo frio a través de un fan; sin embargo, este cambio reestructura las funciones del turbojet.

El concepto de turbojet se basa en la reacción generada de una tasa de gases pequeña a alta velocidad y temperatura por lo que gran parte de la energía se pierde en temperatura o en el proceso de aceleración de los gases como fricción.

Sin embargo, el turbofan cambia el objetivo, en vez de mover masas relativamente pequeñas a alta velocidad, mueve grandes masas a baja velocidad. La eficiencia propulsiva del conjunto es mucho mayor que un turbojet, ya que se transfiere de una manera más eficiente la energía térmica a cinética y permite diseñar motores de mayor potencia. No obstante, estas mejoras tienen un coste, como la velocidad de salida de los gases es baja, únicamente permite volar a velocidades transónica o supersónicas de baja mach [0.6-1.5]. Por otra parte, las grandes dimensiones del Fan, hacen poco aerodinámico el motor generando un drag (resistencia aerodinámica) sustancial. Véase Fig. 8.

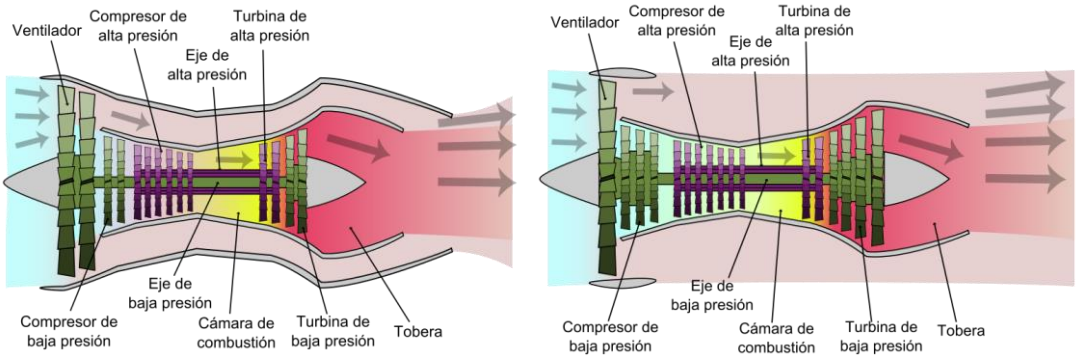


Fig. 11 Turbofan bajo (izquierda) y alto (derecha) bypass ratio.

Por ello hay dos tipos de turbofanes, aquellos con un gran bypass ratio, en los que el flujo principal es mucho mayor que el flujo que pasa por el core (turbojet), normalmente aviones comerciales. Y en los que para seguir alcanzando altas velocidades, tiene un bypass ratio bajo, donde ambos flujos son similares implementado mayoritariamente en aviones militares.

Otra característica importante es la utilización de varios ejes, permitiendo que existan juegos de turbina-compresor a diferentes velocidades de rotación maximizando su funcionamiento. A su vez, también existen diferentes implementaciones de la tobera, donde los flujos pueden mantenerse separados o unirse y salir por una tobera común.

Es el tipo de motor a reacción más utilizado en la actualidad y da su forma característica ancha y acabado en forma de cono de los motores actuales véase la figura 9.

Fig. 12 Avión comercial con Turbofan.

3.6 *Propfan*

Durante la crisis del petróleo en el año 79 y durante la década de los 80 se intensificó el desarrollo de motores con menor consumo, produciendo el cambio del turbojet al turbofan o turbohélix en la gran mayoría de aviones comerciales.

Sin embargo también se desarrollaron otros diseños que combinaban la eficiencia de los turbohélix con las limitaciones de velocidad y altura de los turbofanes, este tipo de motor de nuevo diseño se denominó Propfan o Ultra Hight ByPass debido a la gran diferencia entre el flujo externo y el interno

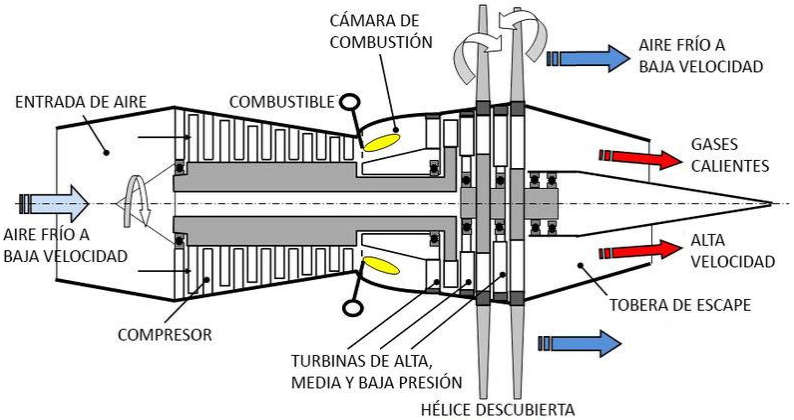


Fig. 13 Estructura de Propfan.

De manera similar al turbofan y al turbohélix, se agrega un conjunto de hélices solidario con el eje, pero en este caso es en la parte posterior del motor y con turbinas separadas con rotación inversa (Fig.13). Permite implementar las mejoras aerodinámicas de hélices y materiales para un vuelo a velocidad y altura

similar a un turbofan; sin embargo, generan problemas de contaminación por ruido.
Actualmente no existe ninguna implementación civil, debido a los problemas de ruido y las fuertes vibraciones que por cuestiones de fatiga descartan el uso de los mismos en la aviación civil.

Fig. 14 Motores Propfan, comercial (arriba) militar (abajo).

Si son usados en uso militar como el Antonov AN-70 y el nuevo avión de Airbus Militar A400M que se construye en Sevilla véase Fig. 15 (abajo).

Actualmente las nuevas tecnologías de materiales y diseños curvados, no descartan su uso debido a su bajo consumo de combustible como un futuro medio para transporte internacional.

3.7 *Estatorreactor o Ramjet*

Alcanzar altas velocidades mach, superiores a 2 veces la velocidad del sonido, es costoso en base a el diseño del turbojet-turbofan. Durante la guerra fría la creación de bombarderos supersónicos, se impulsó una carrera armamentística basada en el desarrollo de cazas y bombarderos más rápidos con acción inmediata en terreno enemigo. Para operar motores a varias veces la velocidad del sonido, manteniendo una velocidad subsónica en su interior es necesario disponer de un difusor característico, que disminuya la velocidad a subsónica, ejemplo de ello fueron los motores del Concorde que disponían de difusores especiales para hacer funcionar sus turbojet-turbofan.

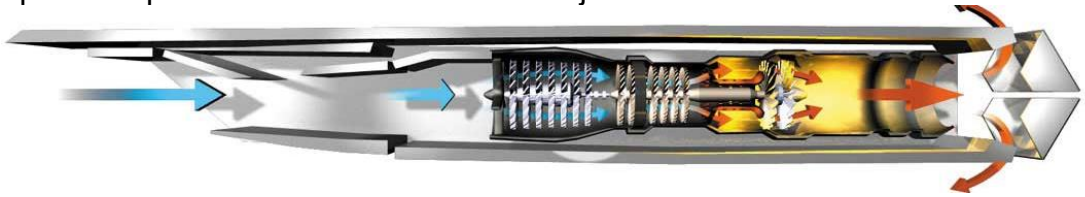


Fig. 15 Motor Concorde [15].

Sin embargo, a velocidades superiores a Mach 2, este proceso es complejo y únicamente optimizado a una velocidad concreta. La solución a este problema es la eliminación del compresor, eje y turbina.

El Ramjet o estatorreactor carente de elementos internos (Fig.15), se basan en complejos diseños y sistemas para el difusor, materiales resistentes a altas temperaturas y toberas de geometría variable. El resultado es un motor geométrico, adaptable a velocidades altas (Mach [1.5-3.5]) pero no valido para velocidades subsónicas.

Normalmente suele operar a altas altitudes (estratosfera) donde la temperatura y densidad del aire son muy inferiores a la troposfera. Muchos de los ramjets contienen un turbojet en su interior y un juego de tubos y compuertas que hacen pasar el flujo de aire a través del turbojet (Fig.16) para velocidades bajas o directamente pasar a la cámara de postcombustión funcionando como Ramjet. Esta configuración permite tener una hibridación de ambos motores, eliminando la necesidad de instalar motores adicionales para el despegue y obteniendo una eficiencia en el gasto de combustible mejor durante el ascenso y adquisición de velocidad.

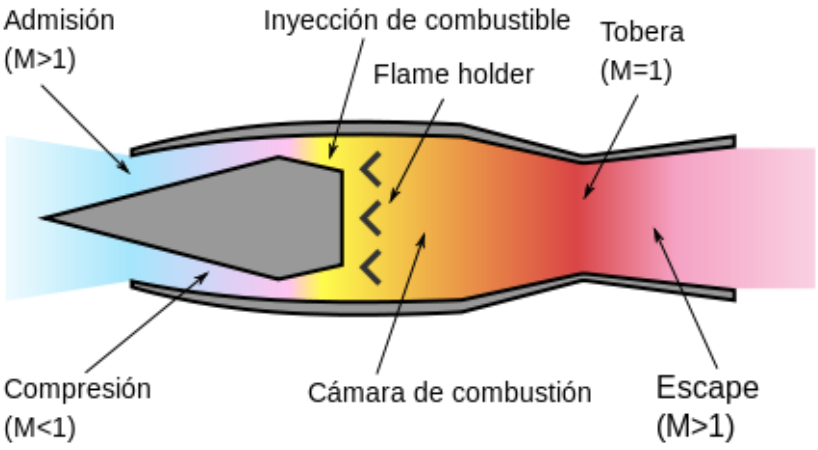


Fig. 16 Esquema básico Ramjet.

Únicamente tienen uso en aviones militares de finales de la guerra fría como el Black Bird (Locky SR-71) o misiles interceptores véase Fig.17.

Fig. 17 SR-71 y misil brahmos

3.8 *Scramjet*

Los motores Scramjet, son motores experimentales principalmente diseñados para el test de experimentos aerodinámicos o para vehículos no tripulados como misiles.

Su principal diferencia es el rango de velocidades de los mismos, superando ampliamente a los Ramjet. Permite alcanzar hasta Mach 10, por las características de la velocidad no es posible desacelerar los gases a través del difusor por lo que se basa en una combustión supersónica (Fig.18), es decir, los gases son comprimidos y desacelerados, pero todo el flujo a través del motor no baja de velocidades supersónicas incluyendo la combustión.

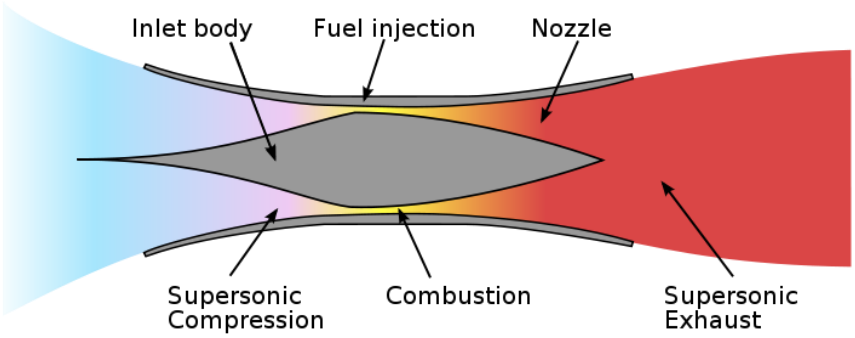


Fig. 18 Esquema de Scramjet.

Sus aplicaciones futuras son los vuelos suborbitales junto a misiles intercontinentales, pero actualmente solo se realizan motores con finalidades experimentales para testear modelos aerodinámicos complejos en vuelos suborbitales. No son soportados por los módulos de este software, pero se ha predispuesto la organización del mismo para incluirlos en un futuro.

3.9 *Selección del tipo de motor*

Para seleccionar que tipo de motor es el más adecuado para un avión es necesario conocer las condiciones de operación de dicho avión. La altura de vuelo, la potencia y la velocidad de vuelo condicionan qué tecnología es más eficiente para el rango de trabajo.

Actualmente la gran mayoría de aviones comerciales se basan en turbofanes, ya que los requisitos de altura (11.000 m) vuelo transónica (0.7-0.9 Mach) y un consumo de combustible minimizado, hacen del turbofan el diseño adecuado.

Sin embargo, cuando se reduce la altura de vuelo o se da más importancia al consumo específico que a la duración del viaje, otras opciones como turbohélix o propfan se imponen para viajes regionales o de carga véase Fig. 19.

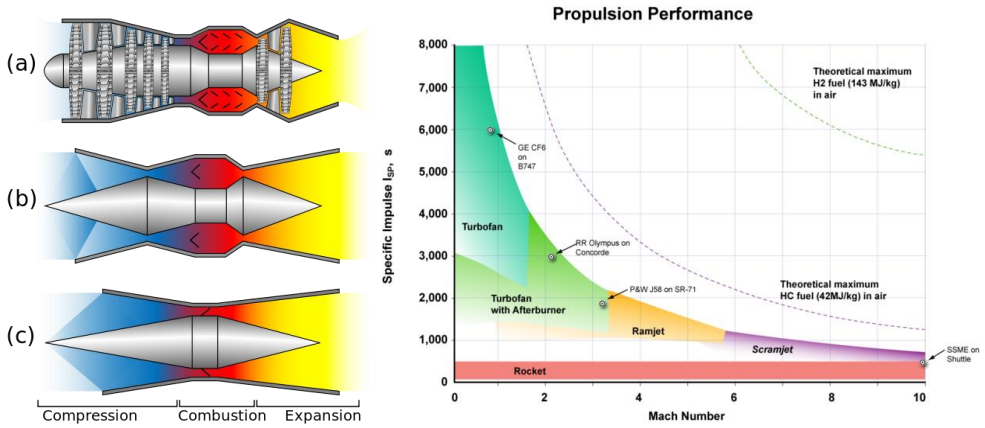


Fig. 19 Turbojet (a), Ramjet (b) y Scramjet (c), gráfica ISP - Mach tecnologías.

En la Fig. 19, se pueden ver las 3 configuraciones básicas de motores junto a una gráfica de eficiencia propulsiva (Véase anexo A 1.10 definiciones) donde se puede comparar en función del impulso especifico necesario y la velocidad del motor cuales la frontera de cada tecnología a utilizar.

Por otra parte, el análisis desde el punto de vista del software es que muchos de los elementos de cada motor se parecen o son comunes entre sí.

El enfoque adecuado del software debe ser la realización de bloques o apartados que permitan simular cada una de las etapas de los diferentes motores.

Esta configuración genera una ineficiencia a la hora de realizar los cálculos, relacionada con la interconexión de los módulos pero permite el reusó de módulos o su mejora vía herencia de POO.

Por ejemplo, se puede obtener un Turbofan a partir de la modificación del compresor de un Turbojet, un Scramjet mejorando el modelo de difusor, tobera y eliminando el eje, compresor y turbina de la simulación o permitiendo la implementación de modelos más complejos o detallistas con sangrados, datos tabulados o implementación de CFD dentro de un mismo bloque.

Finalmente, modificando completamente el modelo de ecuaciones a utilizar, pero reusando la estructura básica de bloques, propiedades y core matemático se puede utilizar para modelar motores Scramjet o Pulsojet que no están contemplados en esta versión del software.

4. Software PropSimulator

En esta primera versión únicamente se dispondrá de los principales turbopropulsores basado en turbojet, turbofan y la generación de motores personalizados o simulación individual de componentes. Es una versión Alfa del software, cuyo objetivo es sembrar las bases para luego ser mejorada y continuada con nuevos modelos más realistas o componentes específicos en mi TFG de telemática (2015-16) donde también puede contribuir futuros alumnos interesados en prácticas de empresas o TFG.

El software permite definir un turbo-motor y las características del mismo junto al modelo termodinámico a utilizar. Permite elegir tanto el modelo matemático como la complejidad de cada uno de los componentes de un turbo-motor. Por otra parte, es capaz de calcular los valores de cada una de las propiedades del fluido o de parámetros de los componentes a partir de las propiedades o parámetros definidas por el usuario.

Es decir, a partir de los datos iniciales se calcula el resto de valores del modelo, indicando en ciertos casos si son realistas o parten de un dimensionamiento erróneo, permitiendo resolver los problemas propuestos en las asignaturas de Termodinámica y Propulsión.

Por otra parte, se ha integrado y desarrollado en paralelo la estructura del software (Core) con sus posibles interfaces graficas o problemas que puede tener un usuario al ahora de definirlo, cimentando las bases de una eficiente interfaz gráfica futura.

De la misma manera, se ha hecho un merecido esfuerzo en un método de auto-test que permita un desarrollo customizado del software asegurando el correcto funcionamiento después de realizar modificaciones o mejoras por parte de futuros desarrolladores sin necesidad de un conocimiento profundo.

4.1 Definición del problema

El problema puede definirse en diversos pasos, donde un usuario con unos valores “input” o valores deseados en las características del motor debe decidir qué modelo utiliza en la simulación, ya que obtendrá el valor del resto de variables del modelo para el motor seleccionado.

Como el usuario puede definir cualquiera de las múltiples propiedades del fluido o parámetros de los componentes del motor, el software debe ser capar de obtener el valor final de las variables para cualquier input, para ello es necesario resolver el sistema de ecuaciones del modelo.

Estas ecuaciones difieren para cada tipo de motor y depende del grado de modelización termodinámico y complejidad de los componentes. Normalmente están formadas por ecuaciones no lineales en función de las características del fluido o de los bloques que interviene en la simulación. Por lo tanto, es necesario

crear un software capaz de resolver ecuaciones no lineales, similar a EES [3] proporcionado por el libro de Termodinámica [1] capaz de encontrar una solución al sistema. No es una programación trivial de la ecuaciones, sino la implementación de un solver genérico para aplicarlo a un conjunto de ecuaciones que varía en cada caso.

Este tipo de problemas, donde intervienen ecuaciones no lineales puede ser resuelto iterativamente con sucesivos sistemas lineales, que en cada iteración minimizan el error de la solución partiendo de un valor cercano (semilla), hasta obtener una solución tolerable o utilizando métodos heurísticos que obtienen soluciones cercanas a la deseada.

La principal complejidad y dificultad de este trabajo no se radica en la modelación y programación de un motor (como ya se realizó en 2012 [2]) sino se trata de la implementación genérica de una solución compleja, capaz de ser aplicada a diferentes tipos de motores utilizando bloques, es decir, con flexibilidad.

Esta flexibilidad de la elección de componentes, así como los modelos utilizados en los mismos, permite crear tus propios engines sin la necesidad de conocer o modificar el código, lo cual es el verdadero reto de este trabajo y la aportación sobre los anteriores.

Por otra parte, utilizando el concepto de POO y herencia, el software permite la creación de nuevos bloques customizados o nuevos modelos, sin la necesidad de modificar el código ya establecido, ganando en visión de futuro y solventando un problema grave detectado en las herramientas previas.

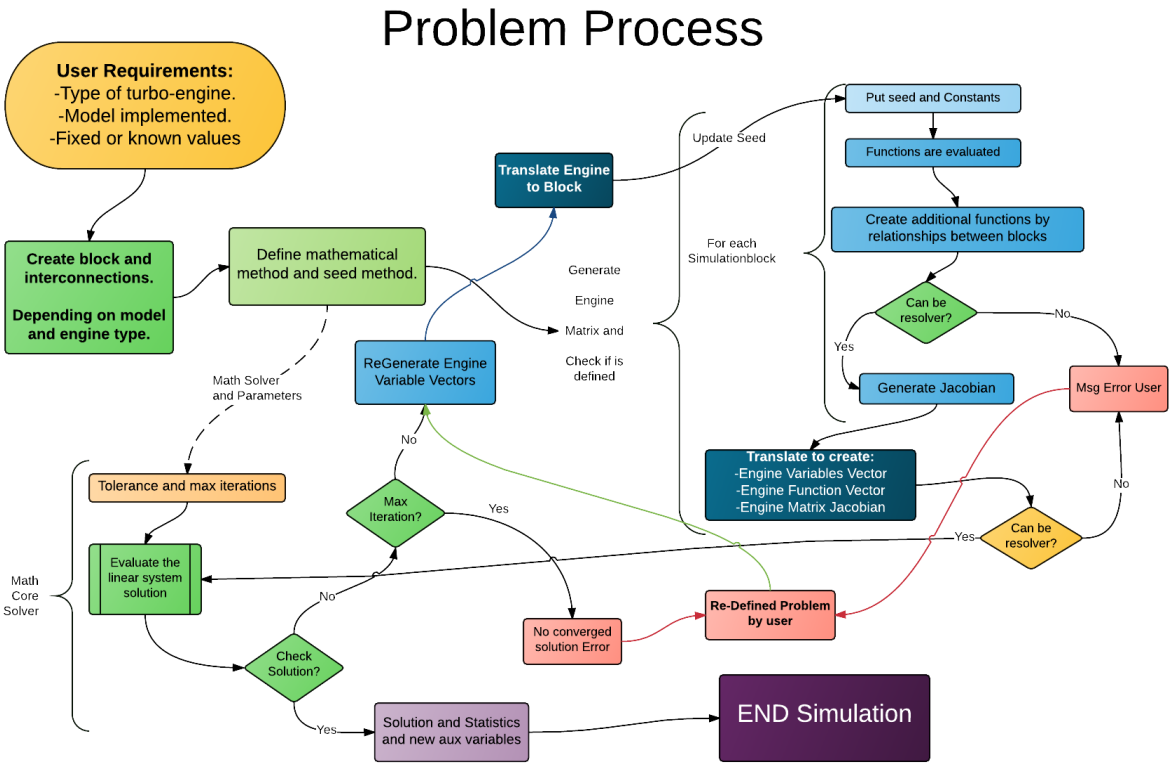


Fig. 20 Proceso principal del Simulador

En la figura 20 se puede observar el esquema adoptado con la finalidad de resolver el problema de la manera más flexible y genérica.

Debido a la necesaria flexibilidad, usando componentes independientes, las ecuaciones del modelo deben estar definidas a nivel de componente como Input-Output; así como interrelacionar los elementos de manera sencilla sin utilizar ecuaciones que relacionen directamente varios componentes o parámetros de los mismos. En otras palabras, únicamente se pueden definir las ecuaciones con las variables que contiene cada bloque. Es necesario relacionar las variables entre bloques con funciones extras, lo cual hace imposible enfocar las ecuaciones al conjunto entero de la simulación. Por lo tanto, es preciso calcular las partes por separado, relacionarlas y ver si el problema es resoluble.

Una vez definidas las ecuaciones y relaciones de cada bloque, se transforma la información individual en información del conjunto, se realiza una iteración matemática y se comprueba si cumple los requisitos de solución.

En la Fig.20 se resume el proceso realizado para cada simulación. Se obtiene la matriz del conjunto a partir de los componentes y se realizar una iteración cuya solución no cumple la tolerancia del residuo. Se transforma la información del conjunto a nivel individual, introduciendo las nuevas variable dentro de cada bloque, recalculando los residuos del modelo y jacobiano a nivel de bloque. Se vuelve a transformar la información individual a información del conjunto y se itera de nuevo hasta conseguir un resultado que sea aceptable (inferior a una tolerancia de residuo). Véase 4.9 para más detalles.

4.2 Soluciones previas y diferencias

En [2] se utilizó el enfoque POO para definir diversos bloques con sus ecuaciones características para cada modelo. En varios apartados se concatenaron varios bloques obteniendo una solución parcial al turbojet, sin embargo no se llegó a consolidar el motor en su conjunto. Este planteamiento es favorable porque el desarrollo de bloques con diferentes modelos permite mejorar modelos sin necesidad de reescribirlos desde cero.

Sin embargo, a diferencia de los PFC previos, en este software se ha optado por una concatenación jerarquizada de variables, funciones y características a lo largo de la POO no utilizada anteriormente; así como el uso de “interconexión” o relación de los componentes de forma dinámica no preestablecida por código.

Finalmente, se ha generado un entorno visual y un pseudo-lenguaje para definir los proyectos a simular, evitando la necesidad de modificar el código, compilarlo obteniendo un producto transparente al usuario final y multiplataforma. Véase 4.11.

4.3 Solución adoptada

Como no se dispone de la totalidad del código presentado en los PFC previos y la problemática asociada al mismo (herencia y difícil compresión del mismo) se

optó por reescribir desde cero todo el programa y aprovechar para cambiar de lenguaje de programación de C++ a JAVA.

La nueva solución ataca desde un plano más abstracto el problema, definiendo en clases jerarquizadas cada uno de los componentes y añadiendo complejidad a cada objeto conforme hereda.

Por otra parte, se han definido los objetos que contienen, relacionan o procesan a los objetos de la simulación permitiendo utilizar por ejemplo diversas implementaciones de métodos matemáticos de manera transparente al código.

La solución está enfocada de manera algebraica, se define un conjunto de matrices iniciales generadas por los bloques (componentes del motor) y se itera en un core matemático hasta obtener una solución convergente (Fig.20).

Para minimizar el coste computacional del mismo, se han realizado implementaciones alternativas a la solución genérica, que reducen los tiempos de cálculo o se ha mejorado el uso de los recursos utilizando librerías matemáticas compiladas en Fortran llamadas desde Java. El objetivo final es obtener una herramienta con las ventajas de Java minimizando sus carencias en bajo rendimiento por tratarse de un lenguaje que funciona sobre máquina virtual.

Por lo tanto el programa integra de una manera más flexible y genérica las características anteriores del software, solucionando los problemas estructurales detectados y partiendo de una visión más abstracta, que puede permitir un desarrollo futuro del mismo con cambios mínimos en la base del software. Así mismo, cumple con los requisitos GNU y de interoperabilidad sobre diversos sistemas que no consiguieron los softwares previos.

4.4 **Diseño del programa**

El programa se divide en 3 funcionalidades principales:

- Interfaz humana: Permite diseñar o modificar valores en un proyecto de manera visual o definiendo un archivo *.prop para su posterior simulación. Se puede utilizar tanto desde consola como ejecutando el modo gráfico. No se ha pulido una interfaz gráfica lograda con el usuario para esta versión, sino que se han creados los objetos, estructuras y métodos de comunicación necesarios para futuras versiones, implementado de una manera esquemática y funcional manera en forma de test para esta versión del software.

- Bloques elementos y conjuntos: Definen las matrices, evalúan los residuos y calcula la matriz jacobiana sobre los residuos. Cuando se habla de conjuntos hace referencia a proyectos prediseñados con los elementos colocados y conectados correctamente (turbojet o turbofan). Por lo tanto, son los objetos utilizados para la modelación y aquellos objetos que se encargan de gestionar la simulación. Son el denominado Model y

Controller del software con los modelos básicos implementados.

- Core matemático, realiza los cálculos matemáticos de cada iteración, es la parte computacionalmente compleja.

En la figura 21 podemos observar que el software puede identificarse con un framework MVC (Modelo Vista Controlador).

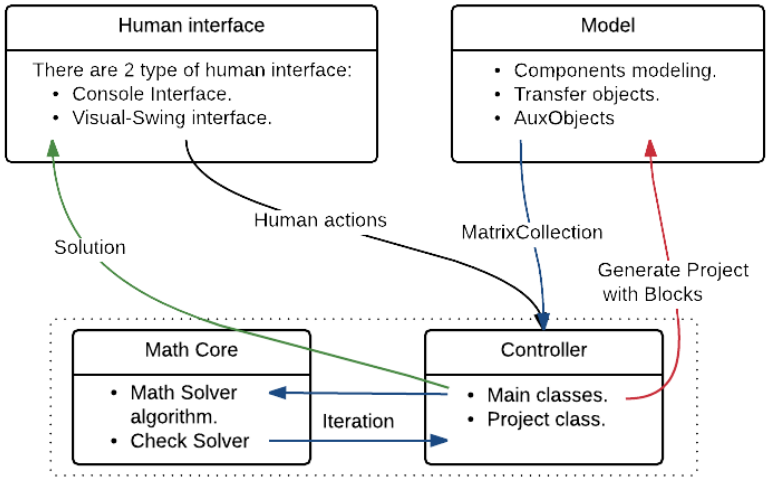


Fig. 21 Diagrama Estructura de PropSim

Con el objetivo de maximizar la compatibilidad, apariencia y programación orientada a objetos, se ha usado Java en las dos primeras funcionalidades. JAVA es un lenguaje muy abstracto orientado a Objetos, esto permite un desarrollo ágil y automatizado de los bloques, la funcionalidad principal de los mismos es la generación de las matrices que sirven para calcular los valores del modelo.

JAVA-SWING permite el desarrollo de complejas interfaces grafica compatibles con diferentes sistemas operativos (Windows, Linux, Mac OS) ya que es capaz de integrar diferentes librerías gráficas, dependiendo del entorno de simulación. Otra ventaja es que la librería SWING funciona puramente en formato de objetos, permitiendo diseñar ventanas y elementos a medidas, esto facilita utilizar directamente los objetos de transferencia e integrarlos en las ventanas, generando automáticamente los recursos a necesidad.

El único problema de Java es su falta de eficiencia computacional. Esta característica no es un problema cuando hablamos de acciones humanas o procesar formatos, ya que las diferencias son milisegundos imperceptibles.

Sin embargo, cuando se necesitan realizar cálculos reiterativos, es un verdadero problema que puede incrementar varias veces el tiempo de cálculo 3-5 veces más. Aunque verdaderamente esto únicamente significaría el equivalente a utilizar un ordenador de 1-2 años de antigüedad, se ha intentado minimizar este problema utilizando librerías externas con pasarela a JAVA. Este recurso es la librería JBLAS que es la interfaz entre las librerías BLAS de FORTRAN con JAVA.

Sin embargo, en la versión Alfa la evaluación de las funciones sigue siendo en JAVA así como la gestión y traducción de bloques a conjunto, por lo que en el apartado de resultado se han establecido las bases de la nueva versión Beta.

4.5 **Estructura del programa**

El programa se estructura siguiendo un modelo MVC, similar a framework actuales, se define una compleja jerarquía de objetos mediante herencia que aglutinan el modelo Fig.22.

Con el objetivo de diferenciar y organizar las diferentes partes del mismo se ha utilizado la estructura de package que provee JAVA como organización.

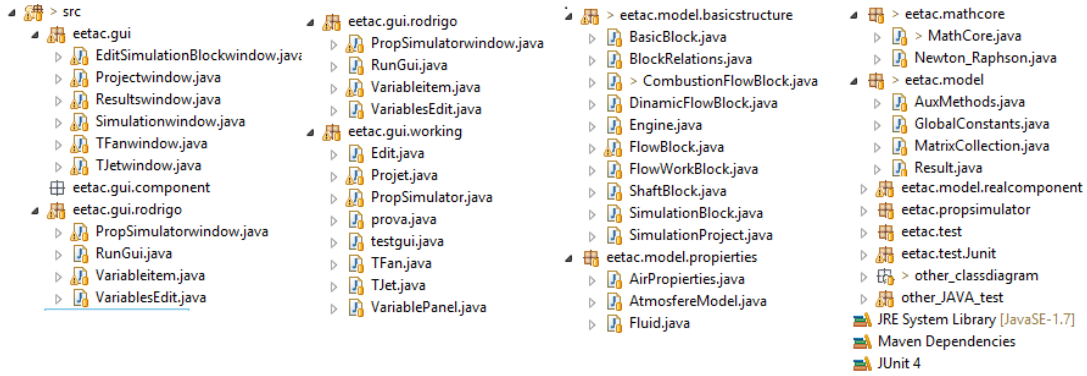


Fig. 22 Estructura de clases del software

La estructura utilizada es la siguiente: ‘eetac.subapartado.detalle’, entre las principales package son:

- ‘eetac.gui.* ’: Contiene todas las clases relacionadas con la interfaz visual.
- ‘eetac.mathcore.* ’: Contiene las clases principales del solver matemático.
- ‘eetac.model.* ’: Contiene las clases que modelan los elementos.
- ‘eetac.model.basicstructure.*’: Contiene las clases que modelan los elementos abstractos como FlowBlock, DynamicFlowBlock véase Fig.25.
- ‘eetac.model.propierties.*’: Contiene las clases auxiliares como propiedades del aire, fuel, modelo atmosférico etc...
- ‘eetac.model.realcomponent.* ’: Contiene los bloques básicos modelo 1.
- ‘eetac.model.realcomponent.X* ’: Contiene los bloques básicos modelo X.
- ‘eetac.propsumulator.* ’: Contiene la clase principal que arranca el modo consola o gráfico.
- ‘eetac.test.* ’: Contiene las clases que testean todo el modelo.

El uso de una estructura tan recurrente es debido a la gran cantidad de clases, que verdaderamente pueden perder a una persona desconocedora del proyecto, durante sus primeros días, si no se encuentran correctamente jerarquizadas y clasificadas.

4.6 Modelos implementados

Se han definido diferentes escalas de realidad en los modelos a simular. Aunque en cada escala se gana realismo y complejidad dentro del diseño del motor, se ha optado por incluir ciertos parámetros de modelos complejos en los objetos del package model.basicstructure, con el objetivo de facilitar y reducir código cuando se generen los modelos más complejos. Estas variables son ignoradas por los modelos más sencillos y únicamente consumen cierta cantidad de recursos en RAM que actualmente son despreciables.

Otro punto es la sencillez matemática, si no es necesario obtener unos resultados muy acotados, se pueden despreciar ciertos factores poco importantes o aproximar por constantes valores que cambian relativamente poco como las propiedades de los gases.

Con el fin de minimizar la construcción de cada uno de los motores cuando se inicie un nuevo proyecto y para consensuar la semilla del cálculo inicial se han establecido tanto a nivel visual como a nivel de proyecto, objetos heredados de la clase engine tales como Turbojet o Turbofan que contienen los objetos y conexiones preestablecidas con valores típicos de los mismos.

4.6.1 Modelo 1

Se basa en el modelo utilizado durante la asignatura de Termodinámica en él se realizan las siguientes asunciones:

- Fluido ideal, gas ideal, sin pérdidas por rozamiento, ni turbulencia.
- Flujo unidimensional, únicamente importa el eje transversal del motor 1D, se asume que el resto del fluido es homogéneo al del eje.
- Propiedades estáticas, las propiedades de los gases no varían en función de la temperatura se mantienen constantes.
- No se tiene en cuenta ciertos parámetros reales como eficiencias en el funcionamiento de difusores o toberas.
- No se tiene en cuenta transferencias térmicas entre elementos de la estructura.
- No se realizan sangrados, ni métodos de refrigeración con el fluido extraído.
- Si se tiene en cuenta la realización de procesos reales generadores de entropía. Modelados mediante la eficiencia isentrópica de los componentes.
- Se tiene en cuenta el parámetro de eficiencia politrópica junto a la eficiencia isentrópica. Esto último es una mejor respecto a los problemas de termodinámica adoptada en la asignatura de Propulsión.
- Asunción de velocidad cercana a cero en el interior del motor, utilizando presiones y temperatura de estancamiento.

Información más detallada véase Anexo B.2.

4.6.2 **Modelo 2**

La principal diferencia respecto al modelo 1 es la implementación de propiedades en función de la temperatura. Se implementa una función capaz de aproximar la entalpia del fluido en función de las características del mismo, α el porcentaje de la mezcla y temperatura. Para ello sea utilizado el siguiente modelo:

$$\frac{Cp}{r} = \frac{\left(\frac{Cp}{r}\right)_{air} + \alpha\left(\frac{Cp}{r}\right)_{fuel}}{1+\alpha} \quad (1)$$

$$\frac{h}{r} = \int_0^T \frac{dh}{r} = \frac{\left(\frac{h}{r}\right)_{air} + \alpha\left(\frac{h}{r}\right)_{fuel}}{1+\alpha} \quad (2)$$

Donde podemos aproximar la curva característica del aire y de fuel como:

$$\left(\frac{Cp}{r}\right)_{air} \approx 3.5 - 2.8\cdot 10^{-5}\,T + 2.24\cdot 10^{-8}\,T^2 + \left(\frac{3090}{T}\right)^2 \cdot \frac{e^{\frac{3090}{T}}}{\left(e^{\frac{3090}{T}}-1\right)^2} \quad (3)$$

$$\left(\frac{Cp}{r}\right)_{fuel} \approx -149.054 + 4.47659\,T - 4.00997\cdot 10^{-3}\,T^2 - 6.12432\cdot 10^{-7}\,T^3 \quad (4)$$

$$\left(\frac{h}{r}\right)_{air} \approx 3.5\,T - 1.4\cdot 10^{-5}\,T^2 + 7.467\cdot 10^{-9}\,T^3 + \frac{3090}{e^{\frac{3090}{T}}-1} \quad (5)$$

$$\left(\frac{h}{r}\right)_{fuel} \approx -149.054T + 2.2383T^2 + 1.3366\cdot 10^{-3}\,T^3 - 1.53108\cdot 10^{-7}\,T^4 \quad (6)$$

Por otra parte, se puede calcular el pressure ratio como en función de la entalpia del fluido y la constante de los gases:

$$\Phi = \int_0^T \frac{dh}{rT} = \frac{\Phi_{\text{air}} + \alpha\cdot\Phi_{\text{fuel}}}{1+\alpha} \quad (7)$$

$$\Phi_{air} = 3.5\,ln\,(T) - 2.8\cdot 10^{-5}\,T + 1.12\cdot 10^{-8}\,T^2 + \frac{3090}{T\cdot e^{\frac{3090}{T}}-1} - \ln\left(\frac{e^{\frac{3090}{T}}-1}{e^{\frac{3090}{T}}}\right) \quad (8)$$

$$\Phi_{fuel} = 4.47659\,ln\,(T) - 8.01994\cdot 10^{-3}\,T + 9.18648\cdot 10^{-7}\,T^2 \quad (9)$$

Finalmente, para calcular la entalpia aportada por el fuel en función de la temperatura del mismo, es decir, eliminando su auto calentamiento:

$$\frac{h_f}{r} = \frac{h_{f_0}}{r} - \frac{\Delta h_{f_c}}{r} \quad (10)$$

$$h_{f_0} = 4.3095\cdot 10^7\,J/kg$$

$$\frac{\Delta h_{f_c}}{r} = -1607.2 + 4.47659\,T + 4.00997 \cdot 10^{-3}\,T^2 - 6.12432 \cdot 10^{-7}\,T^3 \quad (11)$$

Información más detallada véase Anexo B.3.

4.6.3 **Modelo 3**

En el modelo 3 se han interiorizado las relaciones geométricas del motor, dimensionando el mismo geométricamente con áreas In y áreas Out y parámetros como el flujo másico por sección de área o flujo reducido.

Por lo tanto, todos los componentes pasan a tener un área de entrada y un área de salida. Sin embargo la programación de este modelo y su test no ha sido finalizada debido a la detección de un error en el modelo 2.

Como dicho modelo se basa principalmente en el modelo 2, por lo tanto el tiempo invertido en arreglar el modelo 2 ha obligado a posponer el modelo 3 para la versión Beta del software.

Información más detallada véase Anexo B.4.

4.7 ***Objetos***

Todos los elementos del software están divididos en Objetos simulando módulos o bloques. La primera generación de objetos se diferencia entre 3 tipos de clases véase Fig. 23:

- Clases estáticas o aglutinadoras de métodos, se usan como lugar común para muchas variables definidas de manera global o para la aglutinación de funciones recurrentes, no es necesario instanciarlas como objeto. Tiene una función Auxiliar de los objetos funcionales.

- Objetos contenedores o auxiliares, son objetos que se encargan de estructurar la información y sirven como medio común estandarizado para transmitir los datos entre los diferentes objetos funcionales. El mejor ejemplo de ellos es la MatrixCollection que aglutina todas las variables, vectores y matrices matemáticas de un bloque de simulación.

- Basic objects, es el elemento primordial de los modelos a utilizar, identifica nombra y describe el bloque. El resto de Bloques funcionales son hijos de este objeto.

Por otra parte, podemos diferenciar en la segunda generación (por herencia) entre objetos que pertenecen a la simulación matemática como los componentes que pueden existir en un turbopropulsor, basados en las ecuaciones que relacionan sus propiedades Input y Output. Y los objetos que no participan en la simulación pero calculan propiedades para aquellos objetos que sí, un ejemplo es el modelo de atmosfera estándar que indica las propiedades del aire en

función de la altura y velocidad de vuelo sin entrar en la simulación matemática del problema.

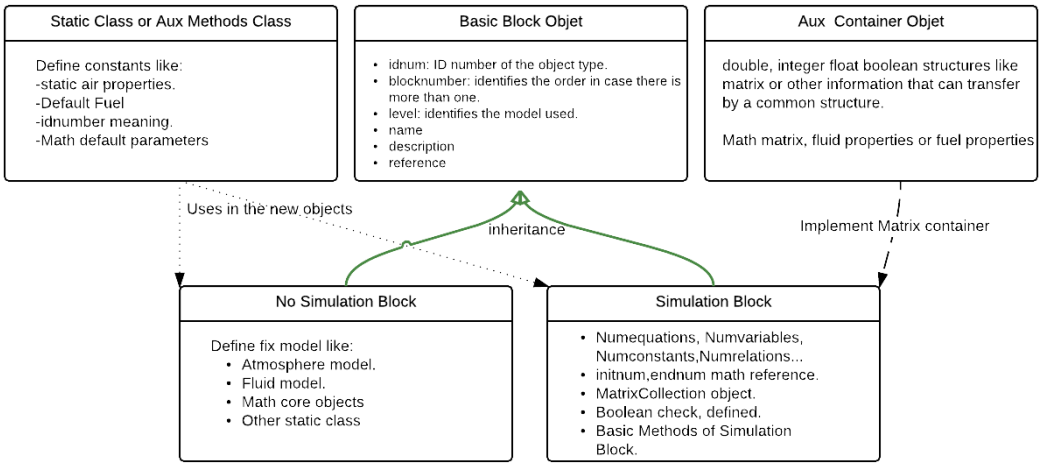


Fig. 23 Descripción principales tipos de Objetos PropSim

A partir del Simulación block, se definen en diferentes generaciones los elementos básicos de un turbo-engine, de manera jerárquica, como si las diferentes ramas de un árbol que comparten un tronco común. Cada uno de estos bloques contiene ecuaciones en forma de funciones, variables y matrices que permiten simular individualmente cada bloque véase Fig. 24.

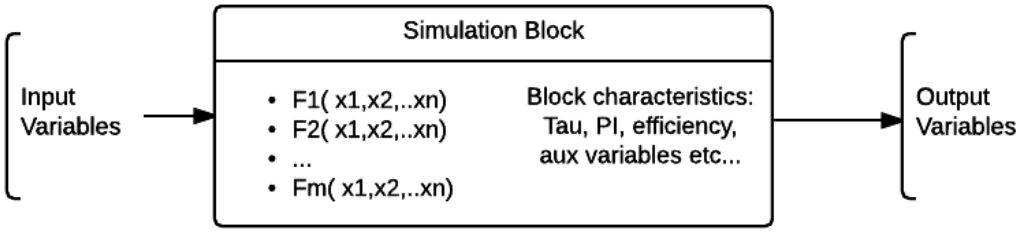


Fig. 24 Esquema Modelo aplicado sobre un componente.

La principal ventaja de este enfoque es la definición de los bloques básicos a partir de sus variables IN, características internas (variables del bloque) y variables OUT.

Por ello, podemos definir un FlowBlock como un componente cuyas variables IN y OUT son las propiedades del fluido presión, temperatura, flujo másico y las características geométricas del mismo como área, coeficiente de rozamiento de las paredes etc…

Si, por ejemplo queremos modelar un difusor o una tobera, únicamente es necesario heredar del Flowblock, añadir aquellos parámetros propios del elemento como eficiencia del difusor o diámetro de garganta e implementar las funciones que relacionan las propiedades IN, variables del módulo y propiedades OUT.

Si lo que se desea es, por ejemplo, modelar una cámara de combustión o poscombustor, es necesario añadir el flujo de fuel y características propias como eficiencia de combustión, aporte calorífico etc… así como rescribir las ecuaciones. Por otra parte si se necesita mejorar un modelo establecido digamos un compresor definido, únicamente es necesario heredar del compresor más simple, añadir las nuevas características del modelo y realizar un Overray de las funciones que relacionan las variables (Ecuaciones del modelo).

Lo verdaderamente curioso es que como todos los bloques heredan de un ancestro común, se puede aceptar el hecho de definir variables que afectan a modelos muy complejos como número de Reynolds del fluido, o características como la turbulencia del fluido en los bloques primigenios. Estas variables no toman valores (null) o no son utilizadas por los modelos más simples, lo que permite combinar modelos complejos con modelos sencillos dentro de un mismo motor, con el único coste adicional de un mayor uso de memoria RAM que dadas las características de los PC actuales es completamente despreciable.

Finalmente, cabe destacar que la existencia de los módulos o apartados independientes, permite la simulación concreta de dicho bloque, muy útil de cara a resolver problemas concretos donde el resto de componentes del motor pueden ser despreciados o simplemente cuando se está simulando otro elemento que no tiene que ver con un turbopropulsor, pero se compone del mismo componente.

Desde el punto de vista del desarrollador, si es necesario agregar nuevas variables o nuevas funcionalidades, únicamente es preciso modificar aquel objeto a partir del cual el resto de objetos adquieren dichas funcionalidades, modificando exclusivamente partes concretas del código. En el caso de que las nuevas funcionalidades solo sean afines a un único objeto, se parte del objeto de interés creando uno nuevo por herencia del mismo, al que si se le añaden las nuevas funcionalidades. Este modelo de desarrollo es especialmente útil ya que aunque implica un gasto extra en tiempo y estructuración en las primeras versiones del software como es esta (Alfa) implica una menor carga de trabajo o modificación del software en futuras modificaciones. Además, utilizando JUnit test, permite el desarrollo seguro del software sin un conocimiento completo del mismo véase anexo D 1.4.

4.8 **Jerarquía de objetos**

Los objetos simulados heredan todos de SimulationBlock y se diversifican en las “ramas” de los bloques que participan en un turbomotor véase Fig.25:

- FlowBlock: Son aquellos elementos cuya principal característica es el fluido que los atraviesa y los cambios en las propiedades del mismo.
- ShaftBlock: Son aquellos elementos cuya principal característica está relacionada con el rotor, o con un eje que gira.
- FutureSubsystemBlock: No implementado en esta versión del software es la rama dedicada a los subsistemas auxiliares del motor tales como lubricación, refrigeración etc... En esta versión del software aún no está definida su relación con los actuales bloques.

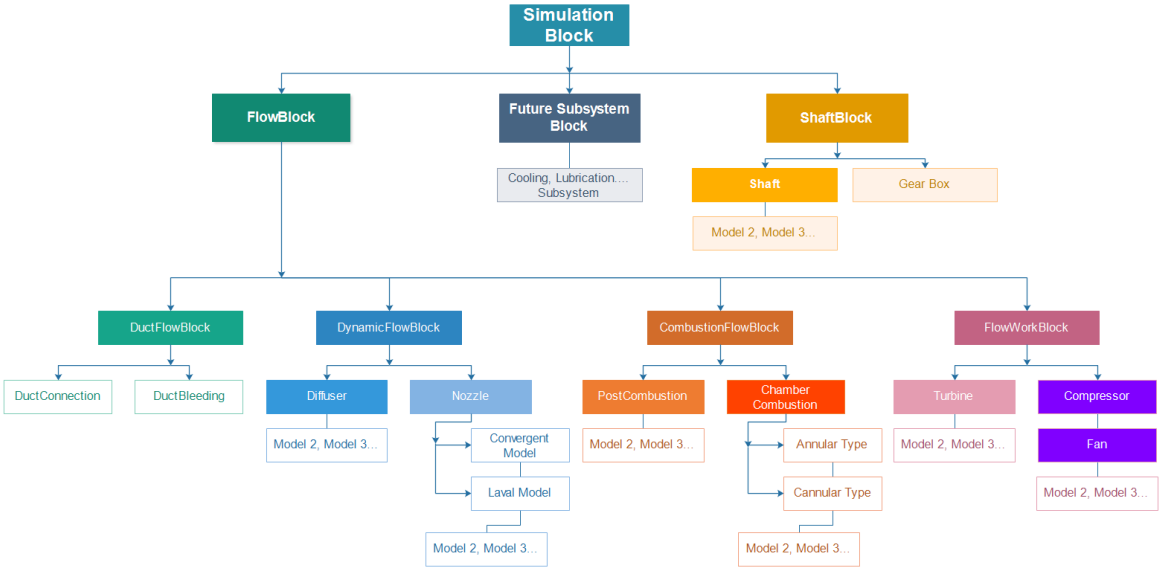


Fig. 25 Ramas principales de herencia de objetos.

Como sub-ramas de la rama principal FlowBlock (la verdaderamente desarrollada en esta versión del software), se encuentran:

- DuctBlock: Que genera los conductos de interconexión, sangrados de compresor etc… solo queda definida no es utilizada en los modelos implementados.
- DynamicFlowBlock: Se refiere a aquellos componentes cuya función es cambiar la velocidad del fluido en función de parámetros geométricos y sin transferencia de trabajo o calor. Es la clase padre de Diffuser y Nozzle.
- CombustionFlowBlock: Se refiere aquellos Flowblock donde se transfiere calor al fluido normalmente a través de la combustión de fuel. Es la clase padre de ChamberCombustion y PostCombustion.
- FlowWorkBlock: Se refiere aquellos Flowblock donde se extrae o transfiere energía del fluido en forma de trabajo. Es la clase padre de Compressor y Turbine.

Finalmente, como se ha explicado anteriormente, a cada elemento definido de un turbomotor como son compresor, turbina, difusor, tobera, cámara de combustión etc… Se puede especificar un modelo termodinámico más complejo, o para detallar las diferencias entre el uso de una u otra tecnología como son el caso de compresor-turbina axial-centrípeto o especificar los cambios según la estructura de la cámara de combustión usando el concepto de herencia en objetos (Fig.25).

Con la configuración actual del software, es posible diseñar nuevas sub-ramas que simulen los elementos de un Scramjet o Pulsojet a partir de las ya existentes. También permiten utilizar un desarrollo de subsistemas o Gearbox para permitir simular generadores, motores de helicóptero u otras herramientas que usan el concepto de turbina.

4.9 Método de Resolución

En base a en los valores conocidos por el usuario y el modelo elegido se obtiene un sistema de N ecuaciones algebraicas no lineales, M variables del sistema, C variables como constantes conocidas definidas por el usuario y R relaciones entre las variables como ecuaciones auxiliares definidas por el usuario.

A partir de la definición del engine se puede indicar que el sistema es resoluble si el número de ecuaciones del modelo más relaciones entre variables y constantes es igual al número de variables total del sistema. Hay que tener en cuenta la independencia de las variables suministradas por el usuario. Por ello se ha implementado un método de validación por bloque a la hora de suministrar la información véase 4.11.1. Sin embargo actualmente es posible suministrar información no coherente a varios módulos interconectados, por lo que el software únicamente detecta si el sistema es resoluble iterando hasta alcanzar un número máximo de iteraciones a partir del cual detecta que no tiene convergencia. Se ha planteado la idea de usar el determinante del jacobiano pero no ha quedado demostrada su efectividad para una evaluación discreta y sobre la semilla de las funciones.

$$|\nabla \bar{F}(\bar{x_\iota})| \neq 0 \quad (12)$$

Existen muchos métodos matemáticos para la resolución de ecuaciones no lineales, se han escogido el método de Newton-Raphson y optimizaciones del mismo para reducir la carga computacional.

4.9.1 Newton-Raphson

El principal método utilizado en este software se basa en el Método de Newton-Raphson o método de la tangente aplicado a N funciones con M dimensiones, donde M-N son las funciones que definen las constantes o las relaciones. Por lo que es necesario datos para que las matrices sean cuadradas y resolubles.

Definida una función continúa y partiendo de valores cercano a la solución el método converge cuadráticamente en la solución tal como se observa en la Fig. 26.

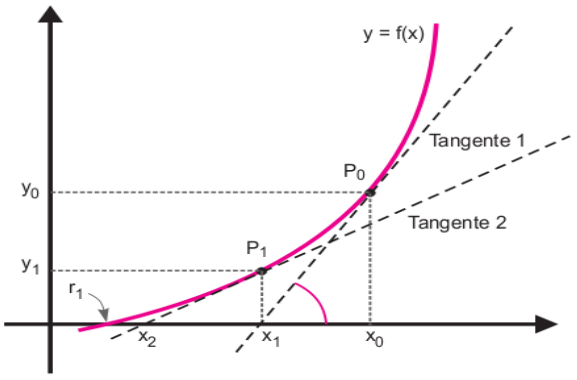


Fig. 26 Método Newton o tangente.

La implementación programada del método para una función unidimensional es:

$$X_{k+1} = X_k - \frac{f(X_k)}{f'(X_k)} \quad (13)$$

Donde:

X_{k+1}: Es el valor de la variable tras la iteración.
X_k: Es el valor actual de la variable.
$f(X_k)$: Es la función residuo del modelo.
$f'(X_k)$: Es la derivada del residuo para las variables actuales.
k: Es el número de iteración actual.

Sin embargo, en este caso el modelo se define M funciones de M variables. Para dimensionar de $R_1 \rightarrow R_M$ se puede demostrar que la siguiente implementación algebraica es equivalente a:

$$[\, X_{k+1} \,] = [\, X_k \,] - [\, J_F(x_k)]^{-1} [\, F(x_k) \,]$$

Expresado vectorialmente como

$$\bar{X}_{k+1} = \bar{X}_k - \bar{\bar{J}}_{\bar{F}}(\bar{x}_k)^{-1} \cdot \bar{F}(\bar{x}_k) \quad (14)$$

Donde:

$\overline{[\, X \,]} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}_{1XM}$	Es el vector vertical de M variables del modelo.
$\overline{[\, F(\bar{x}) \,]} = \begin{bmatrix} f_1 = F(\bar{x})_1 \\ \vdots \\ f_N = F(\bar{x})_N \end{bmatrix}_{1XM}$	Es el vector vertical de residuos de las N funciones + C constantes + R relaciones, donde M = N+C+R.
$\overline{\overline{[\, J_{\bar{F}}(\bar{x}_k)]}} = \begin{bmatrix} \frac{f_1}{\partial x_1} & \dots & \frac{f_1}{\partial x_M} \\ \vdots & \dots & \vdots \\ \frac{f_M}{\partial x_1} & \dots & \frac{f_M}{\partial x_M} \end{bmatrix}_{MXM} = \nabla \bar{F}(\bar{x})$	Es el Jacobiano sobre el vector $[\, F(x) \,]$, es decir, la matriz de derivadas parciales de cada una delas M funciones sobre cada una de las M variables.
k	Es la iteración actual.

De esta manera se puede calcular simultáneamente la resolución de las M funciones con M variables definiendo en los bloques las matrices $\bar{X}$, $\bar{F}(\bar{x})$ y $\bar{\bar{J}}_{\bar{f}}(\bar{x})$ y aplicando operaciones algebraicas en el core matemático del programa.

4.9.2 **Newton-Raphson Mejorado**

El principal problema de la utilización de este método para un espacio de N funciones con M variables es el cálculo de la matriz inversa del jacobiano sobre el vector Fx.

El cálculo de la matriz inversa es un proceso computacionalmente complejo, por lo tanto, es un problema a la hora de optimizar el código. Utilizando la librería JBLAS respecto a las propias herramientas de JAVA Math, se consigue disminuir el tiempo de cálculo casi 1/9.

Sin embargo cuanto mayor es la matriz a invertir mayor es el coste computacional. Para evitar futuros problemas con proyectos complejos o tiempo de espera no deseada se ha optimizado de 2 formas la obtención de la misma y se han buscado métodos alternativos que no requieran el cálculo de la matriz.

4.9.2.1 No recalcular JFx en cada iteración

En múltiples casos cuando la solución esta próxima, significa que el rango de valores sobre el que los que se está trabajando es relativamente pequeño, de la misma manera se puede entender que las derivadas parciales para puntos cercanos son similares.

Por lo tanto, se puede encontrar un punto donde el error producido por la reutilización de Jacobiano sobre Fx calculado en la anterior iteración sirve para alcanzar una nueva solución X' menos cercana a la solución final que si hubiésemos recalculado el jacobiano. Sin embargo, el tiempo ahorrado por el no calculo permite iterar más veces y por consiguiente alcanzar la solución final antes que calculando el jacobiano.

$$K_{iteraciones} * Tiempo_{jacobino} > K'_{iteraciones} * Tiempo'_{\frac{1}{2}jacobiano} \text{ donde } K' > K$$

Como este tipo de soluciones solo son ventajosas cuando nos encontramos cerca de la solución final únicamente puede ser utilizada cuando los residuos de las funciones han bajado de un valor umbral.

Sin embargo dado la característica de los modelos testeados (simples) se alcanza la solución final en pocas iteraciones, lo cual no permite por el momento demostrar y comparar esta solución alternativa. Que si se pueda observar para la versión Beta del programa.

4.9.2.2 No calcular la inversa

La idea es sencilla, si el problema principal es calcular la inversa de JFx la solución radica en encontrar un manera equivalente al problema con menor coste computacional. Partiendo de la ecuación: $\bar{X}_{k+1} = \bar{X}_k - \bar{\bar{J}}_{\bar{F}}(\bar{x}_k)^{-1} \cdot \bar{F}(\bar{x}_k)$ (15)

Se define $\bar{Y}_k$ tal que:

$$\bar{X}_{k+1} = \bar{X}_k + \bar{Y}_k \quad (16)$$

$$\bar{Y}_k = -\bar{\bar{J}}_{\bar{F}}(\bar{x}_k)^{-1} \cdot \bar{F}(\bar{x}_k) \quad (17)$$

Dicho de otra manera:

$$\nabla\bar{F}(\bar{x}_k) * \bar{Y}_k = -\bar{F}(\bar{x}_k) \quad (18)$$

Donde todos los elementos son conocidos a excepción de $\bar{Y}_k$, por lo tanto, se puede resolver el sistema de ecuaciones donde:

$$\begin{bmatrix} \frac{f_1}{\partial x_1} & \cdots & \frac{f_1}{\partial x_M} \\ \vdots & \cdots & \vdots \\ \frac{f_M}{\partial x_1} & \cdots & \frac{f_M}{\partial x_M} \end{bmatrix}_{MXM} * \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}_{1XM} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}_{1XM} \quad (19)$$

O lo que es lo mismo resolviendo por gauss la matriz:

$$\begin{bmatrix} \frac{f_1}{\partial x_1} & \cdots & \frac{f_1}{\partial x_1} & -f_1 \\ \vdots & \cdots & \vdots & \cdots \\ \frac{f_M}{\partial x_1} & \cdots & \frac{f_M}{\partial x_1} & -f_M \end{bmatrix}_{MX(M+1)} \xrightarrow{Gauss\ Method} \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix}_{1XM} \quad (20)$$

Se obtiene el mismo efecto que calculando la inversa del Jacobiano de Fx pero con un coste computacional inferior.

Se han estudiado otros métodos para la versión Beta del software que actualmente no se encuentran implementados. Todos ellos radican en aproximaciones del Jacobiano de Fx o de algoritmos o estimaciones similares al implementado en este punto son:

- Método de Quasi-Newton, utiliza el método anteriormente descrito en este puto para la primera iteración, después utiliza la matriz A como pseudo inversa del Jacobiano donde:

  $$\bar{\bar{A}}_k * (\bar{x}_{k-1} - \bar{x}_{k-2}) = \bar{F}(\bar{x}_{k-1}) - \bar{F}(\bar{x}_{k-2}) \quad (21)$$

  Teniendo que resolver el sistema de ecuaciones y perdiendo la convergencia cuadrática.

- Método del descenso rápido, se basa orientar un factor alfa en la dirección de máxima decrecimiento de las ecuaciones.

$$G(\bar{x}) = \sum_{i=1}^{N} F_i(\bar{x})^2 \quad [24]; \qquad \bar{X}_{k+1} = \bar{X}_k - \propto \cdot \nabla G(\bar{x}) \quad (22)$$

Donde el truco esta en encontrar la $\propto_{opt}$, para lo cual se suele utilizar tres valores $\propto_1 = 0$; $\propto_3 > \propto_1$; $\propto_2 = \frac{\propto_3}{2}$; y se escoge el mínimo de $\propto_{opt} \approx \propto_{min}$ talque para el polinomio de segundo orden basado en los puntos: $[\propto_1, h(\propto_1)], [\propto_2, h(\propto_2)], [\propto_3, h(\propto_3)]$ (23) definidos por la función:

$$H(\propto) = G(\bar{X}_k - \propto \cdot \nabla G(\bar{x}_k)))] \quad (24)$$

La principal ventaja de este método radica en que siempre converge aunque reduce su convergencia a lineal.

Se calcula el residuo en el actual y el residuo en el delta pequeño y la derivada, no usar punto intermedio, sino que lo se hace con dos punto y a derivada en el punto.

- Otros métodos de aproximación al jacobiano de Fx como el algoritmo de Broyden o similares.

4.9.2.3 Soluciones no convencionales

Se ha evaluado la implementación de métodos inexactos de newton (inexact newton) basados en sub-espacios de krylov para la versión Beta del software.

Su principal ventaja es la eliminación de la generación de jacobiano y las diferentes procesos traducción desde los bloques al conjunto, ya que estos métodos son capaces de aproximar el jacobiano sin pre calcularlo en el conjunto simulado.

Es la base de los métodos conocidos como GMRES (Generalized Minimun Residual), BICGSTAB(Biconjugate Gradient Stabilized), QMR(Quasi Minimal Residual), TFQMR(Transpose-free QMR) y MINRES (Minimal Residual).

4.10 **Auto test**

Una de las principales características del software es su complejidad. Aunque el uso reiterativo de herencia y la jerarquización del mismo permite un uso más adecuado y entendible para otros desarrolladores. La realidad es que el software contiene un número de clases con bastantes líneas cada una, aunque los comentarios ayudan a entender el funcionamiento del programa es posible no entender en toda su plenitud los mecanismos adecuados.

Para evitarle este problema a futuros desarrolladores así como evitar la generación de bugs en zonas estables del código se ha implementado un JUnit Test que es un test de valores y funcionamiento, que comprueba automáticamente si tras el nuevo código o las modificaciones del mismo el programa funciona correctamente.

Información más detallada véase anexo C 1.4.

4.11 **Interfaz Humana**

Para la interfaz visual de la versión Alfa se ha reducido a lo más esquemático y funcional posible. El menú principal se compone únicamente de los elementos o engines a elegir con los diferentes grados de modelización. Seguidamente se accede a la pantalla del proyecto donde se pueden modificar los valores de constantes a utilizar.

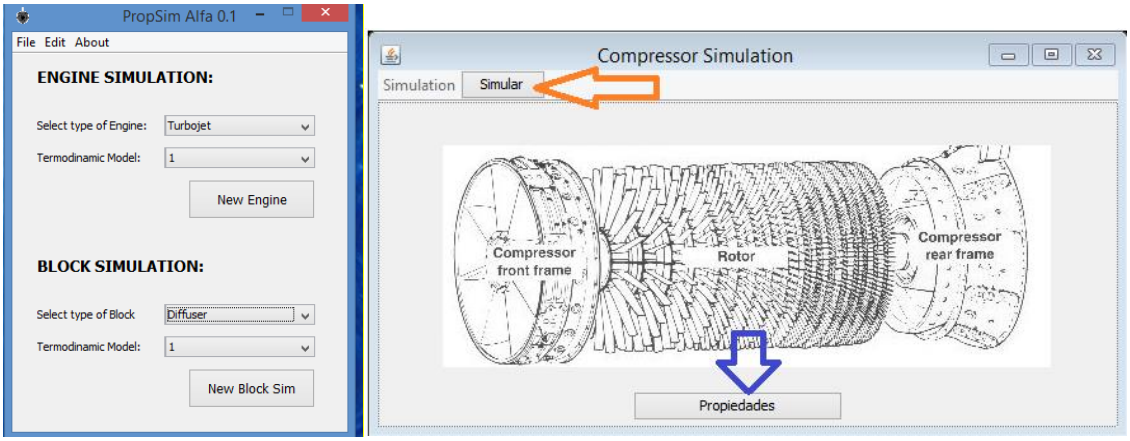


Fig. 27 Capturas de pantalla del entorno grafico

El simulationProyect o engine elegido constan de una ventana con una imagen de la simulación en cuestión y un botón para modificar los datos de los diferentes componentes.

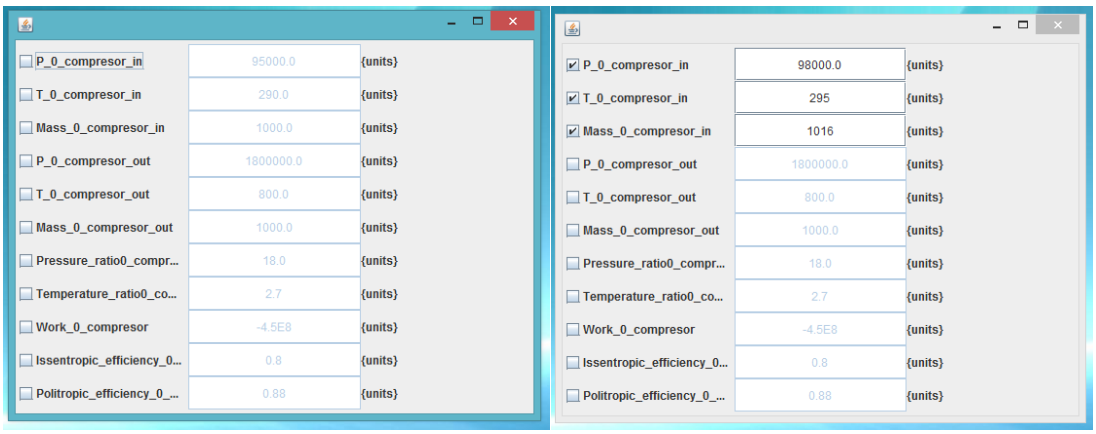


Fig. 28 Ventana de modificación de propiedades de componente

Si el SimulationProyect está definido, es decir, tiene el número adecuado de constantes y son independientes, se inicia la simulación devolviendo el resultado de la misma en un Messagebox.

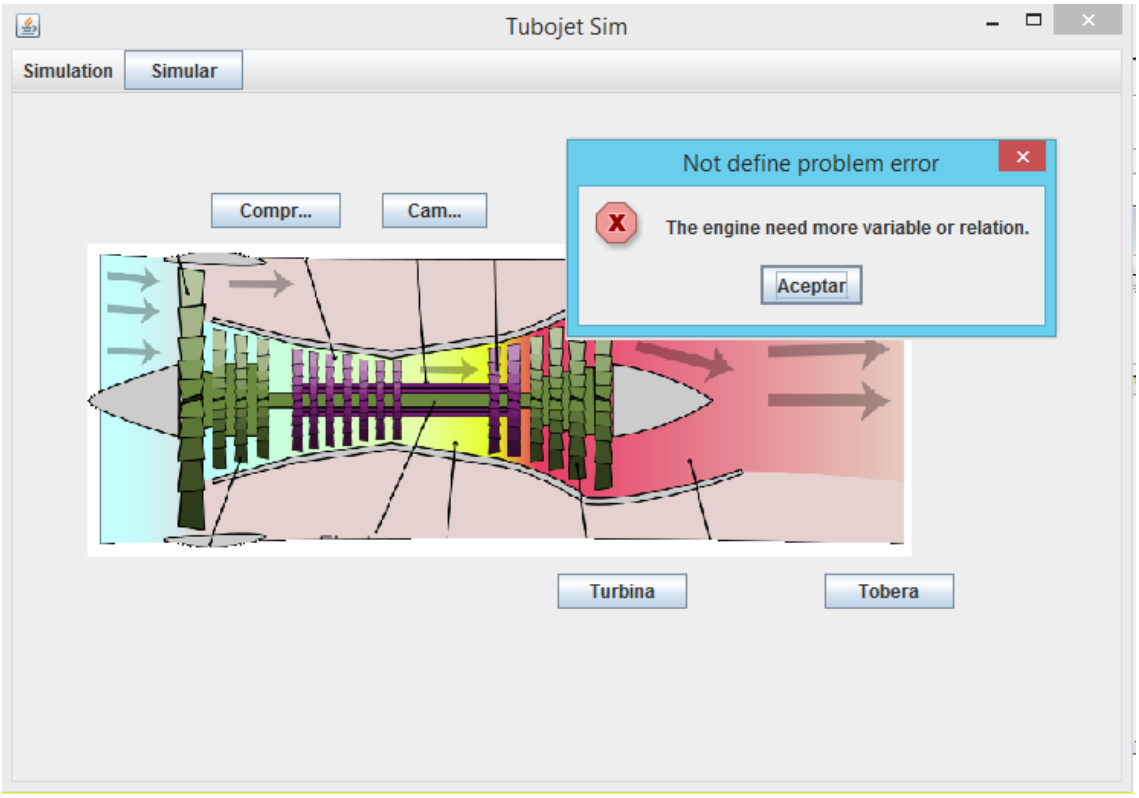


Fig. 29 Linux KDE Simulation window

Lo verdaderamente importante de la estructura son la ventana de modificación de variables, sus objetos estructura y eventos creados por POO e integrados con los objetos del programa.

4.11.1 Variables independientes

Desde la interfaz visual, cuándo se cliquea sobre un elemento para cambiar las variables del mismo hay que evitar que el usuario designe variables no independientes. El ejemplo típico es indicar las temperaturas IN o OUT y la relación de las mismas.

Este problema, que inicialmente parece sencillo no lo es ya que por ejemplo en el caso del compresor o turbina, las presiones y temperaturas están interrelacionadas con los parámetros del compresor, al igual que con el trabajo consumido o producido.

$$Compresor: \begin{bmatrix} \dot{m}_{in} \\ \dot{m}_{out} \end{bmatrix} \leftrightarrow \begin{bmatrix} cp \cdot \dot{m} \\ W \\ \Delta T \end{bmatrix} \leftrightarrow \begin{bmatrix} T_{in} \\ T_{out} \end{bmatrix} \leftrightarrow \tau \leftrightarrow \eta_i \, \eta_p \leftrightarrow \pi \leftrightarrow \begin{bmatrix} P_{in} \\ P_{out} \end{bmatrix}$$

La solución ha sido la implementación de una ecuación booleana que en función de los checkbox activados (variables proporcionadas) en la interfaz visual y el checkbox que el usuario intenta marcar. Si la variable que intenta marcar ya está indirectamente proporcionada por el resto de variable la ecuación devuelve true, no habilitando el checkbox ni el textbox donde introducir la variable y notificando al usuario del problema vía messagebox.

$$f_1: (P_{in} + P_{out}) \cdot (\pi + \tau + \Delta T) \cdot (P_{in} \cdot P_{out}) + (\pi \cdot \tau + \pi \cdot \Delta T + \tau \cdot \Delta T) \quad (25)$$

$$f_2: (T_{in} + T_{out}) \cdot (\pi + \tau + \Delta T) \cdot (T_{in} \cdot T_{out}) + (\pi \cdot \tau + \pi \cdot \Delta T + \tau \cdot \Delta T) \quad (26)$$

$$f_3: (\dot{m}_{in} + \dot{m}_{out} + W) \cdot (\dot{m}_{in} \cdot \dot{m}_{out} + W \cdot (\pi + \tau + P_{in} \cdot P_{out} + T_{in} \cdot T_{out}) \quad (27)$$

$$\Delta T = (\dot{m}_{in} + \dot{m}_{out}) \cdot W \quad (28)$$

$$aviable_{compresor} = f_1 + f_2 + f_3 \quad (50)$$

Todas son variables booleanas true si se ha proporcionado esa variable o false si no se ha proporcionado. Obviamente estas funciones pueden ser simplificadas ya que hay términos redundantes.

$$aviable_{compresor} = f(P_{in}, P_{out}, T_{in}, T_{out} \dots) => \begin{cases} 1 \rightarrow No\ habilita\ check\ \&\ textbox \\ 0 \rightarrow Si\ habilita\ check\ \&\ textbox \end{cases}$$

Donde hay que aplicar la misma regla para el resto de componentes existentes. Sin embargo, esta solución es local, y no imposibilita que se indique dos propiedades interrelacionadas entre componentes, el cual es un problema aún pendiente por resolver. En el apartado de resultados se ha sugerido una posible solución en estudio.

5. **Resultados**

El objetivo de este TFG es el desarrollo del software, sin embargo, los resultados del mismo no están enfocado al test o comparativa de los modelos con medidas reales, sino a la adecuación y mejora de este software respecto a previos, junto a la evaluación de nuevas estrategias o soluciones para la versión Beta en 2016.

Por lo tanto, se han obtenido varios resultados al realizar las comparativas con los anteriores predecesores, por otra parte se han analizado las mejoras computacionales implementadas. Finalmente los resultados más significativos son los puntos mejorables del software y la estrategia enfocada para la versión Beta del mismo.

5.1 **Grado de realización de los objetivos**

Se ha obtenido satisfactoriamente un software genérico que implementa un modelo de jerarquía incremental que permite simular un porcentaje significativo de los turbopropulsores actuales.

Se han implementado la simulación de bloques individuales de un motor, equivalente a los problemas de la asignatura de Termodinámica, por otra parte el auto test individuales en cada uno de los componentes ha contrastado el funcionamiento del mismo al calcular individualmente problemas conocidos.

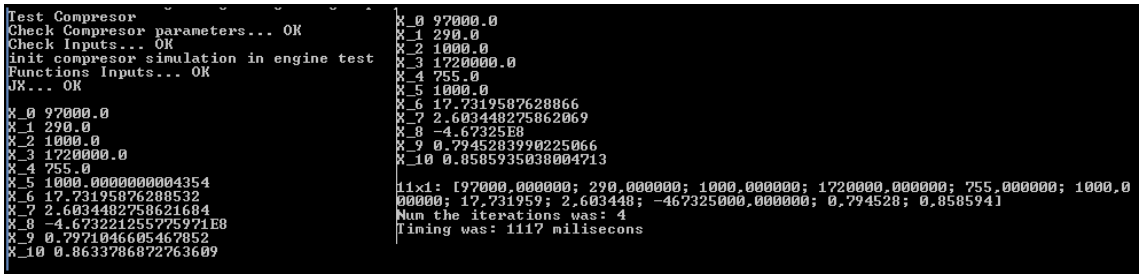


Fig. 30 Auto Test Compresor interfaz de Consola.

Para más información sobre los test implementados y el procedimiento del mismo véase anexo C 1.4. Para un mayor detalle sobre los objetivos conseguidos, los descartados y problemas detectados durante el desarrollo véase anexo D 1.5.

5.2 **Resultados Comparativas**

Las mejoras obtenidas respecto a los softwares anteriores, se basan en la realización flexible del mismo y no en los modelos o los resultados obtenidos.

Aunque no se puede realizar una comparativa real entre ambos por carencia propia de los anteriores, si queda esclarecido que los modelos teóricos implementados y testeados son similares o iguales a nivel de complejidad o ecuaciones termodinámicas utilizadas para el modelado de los mismos.

La carga computacional del mismo ha sido mucho más depurada y optimizada en este programa, sin embargo, ha sido cuidada en exceso ya que parte de una base diferente que es JAVA, la cual a cambio de sus ventajas tiene una capacidad matemática inferior que C++. Es decir, se ha obtenido un mejor resultado conceptualmente con propiedades y mejoras extras pero seguramente un resultado inferior, analizando exclusivamente el tiempo de cálculo.

Se ha conseguido establecer las bases de una estructura CORE de clases tanto de modelo como de mathcore, que integran una interfaz visual o por consola orientada a objetos. Este es un punto clave ya que mide no solo la flexibilidad del código, sino sus posibilidades de desarrollo futuras, que son mucho más abiertas y extensas que en anteriores proyectos.

Una de estas posibles extensiones es la traducción del Core de clases a c++, o de ciertas evaluaciones matemática que permitan utilizando la misma interfaz gráfica incluso poder elegir entre utilizar un core en c++ o en java véase 5.4.2.

Elección del uso de GitHub y JAVA con MAVEN, permite seguir el proyecto online desde el repositorio público, hacer tus propias versiones del código, notificar errores y autogestión las librerías necesarias manteniéndolas actualizadas a través de maven. Es decir, aporta la transparencia, el carácter GNU y la autogestión de librerías carente en los anteriores proyectos.

5.3 **Mejoras computacionales**

Durante la realización de las simulaciones para asegurar el correcto funcionamiento de los módulos programados se ha observado un aumento significativo del rendimiento de los algoritmos tras la utilización de la librería JBLAS.

Se puede observar que el tiempo promedio necesitado para realizar las simulaciones de bloques simples crece linealmente con el número de iteraciones como se ve en la ilustración 1.

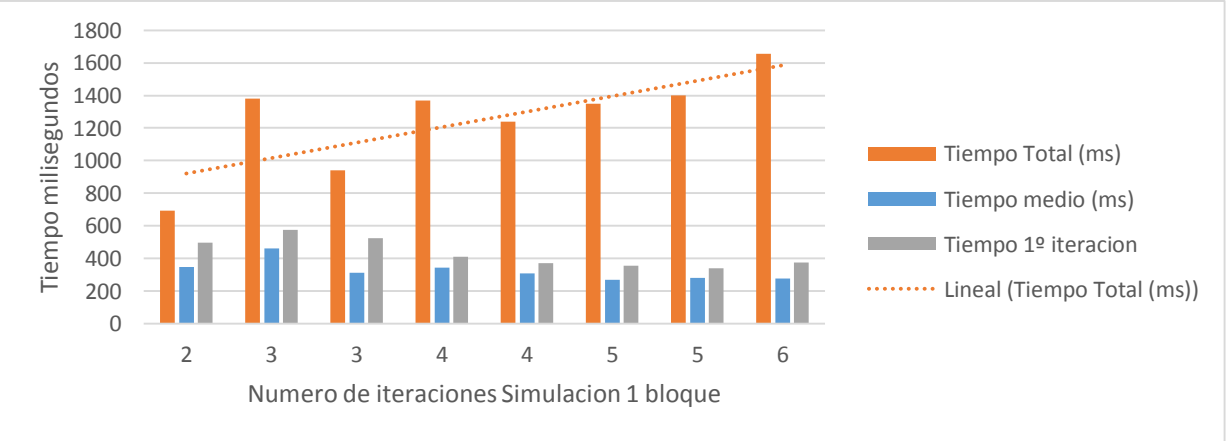


Ilustración 1 grafica de resultados Tiempo simulación – Iteraciones.

Uno de los comportamientos más significativos de mencionar es un retraso extra en el cálculo de las primeras iteraciones, así como en la generación de las matrices. Este aumento del tiempo de computación durante las primeras acciones está relacionado con el uso y predisposición de la máquina virtual de JAVA o la carga de la llamada a la librería JBLAS.

Otro de los puntos analizados es que la fluctuación del tiempo de respuesta depende del uso de la máquina virtual de java que puedan hacer otras aplicaciones de uso cotidiano como navegadores, o aplicaciones en la nube basadas en java (Dropbox, JDowloader, etc…).

Se ha concluido que aunque la resolución matemática se puede pulir un poco más, no interesa invertir fuertemente en ello. Sin embargo, si es interesante mejorar los valores iniciales de semilla, con el fin de partir de un valor inicial más cercano a la solución, poder converger en menos iteraciones y en consecuencia reducir el tiempo de cálculo. Estas estrategias parten de solucionar durante las primeras iteraciones un modelo más sencillo del motor, utilizar técnicas probabilísticas para la generación de varias semillas y escoger la que obtenga un menor residuo para realizar las iteraciones o la adecuación e interconexión de semillas de los módulos relacionados de una manera más acotada que la actual.

5.4 *Resultados para estrategias futuras*

Durante la realización de este trabajo ha quedado patenten la posibilidad de realizar una segmentación dinámica de los componentes de un turbopropulsor de manera flexible y jerarquizada que era la idea inicial del proyecto. Por otra parte, ha permitido implementar una interfaz humana transparente a aquel usuario desconocedor de conocimientos de programación pero con un conocimiento de los modelos de simulación eliminando la necesidad de modificar y conocer el código.

Se han reportado mejoras y estrategias a rediseñar de cara a la versión Beta, donde permiten no solo mejorar aquellas metas de este proyecto, sino arreglar aquellos inconvenientes detectados en el mismo.

5.4.1 **Adimensionalización de ecuaciones**

Uno de los principales puntos débiles estructurales del proyecto es la utilización del residuo de las ecuaciones como valor de check para la tolerancia de la solución. Dependiendo de las unidades y rangos de las mismas no es lo mismo tener un error de 0.01 Pa sobre varias atmosferas que 0.01º K sobre temperaturas en un rango de 200-900º K. Es decir, aunque la tolerancia utilizada para cumplir los requisitos de las ecuaciones es común a todas, el significado sobre cada una varía en la importancia del parámetro y sobretodo en el rango de valores del mismo.

$$F_i(x_1, x_2, \dots, x_n)\ [mag.] \rightarrow \frac{F_i(x_1, x_2, \dots, x_n)}{x_i}\ \ o\ \frac{F_i(x_1, x_2, \dots, x_n)}{G_i(x_i, x_j)}$$

Por lo tanto, una solución tangible pasa por la adimensionalización de dichas ecuaciones utilizando algunas de las variables o combinaciones matemáticas de las mismas con el objetivo de adimensionalizar el residuo y normalizarlo, permitiendo tener un parámetro más objetivo a la hora de evaluar la solución y por otra parte evitar iteraciones innecesarias ya que un criterio de 0.01 Pa en las ecuaciones de presión es más de 100 veces más preciso que un error de 0.01 en temperatura.

5.4.1 Ajuste de las semillas

Cuanto más cercana es la semilla a la solución menor es el número de iteraciones necesaria para la convergencia. En aquellos casos donde la semilla varia menos de un 5% sobre la solución final suele resolverse en 1-2 iteraciones, mientras aquellos casos donde la solución varía entre un 5-15% es necesario un número entre 4-6 iteraciones en el análisis de un bloque.

En consecuencia es muy importante que la semilla de la solución parte de valores cercanos a la misma para asegurar la convergencia del método. Por otra parte la cercanía de la misma a la solución disminuye en un grado no lineal el número de cálculos necesarios.

Por lo tanto, se pueden usar varias estrategias:

1. Definir un rango aceptable para las semillas, generar y analizar los residuos de un random de semillas en paralelo. Escoger la semilla con un residuo menor para realizar la simulación.
2. Solucionar Modelos matemáticos estáticos que aproximen la semilla a partir de valores claves del motor.
3. Iterar inicialmente modelos con complejidad inferior para acércanos a la solución final y utilizar el modelo complejo cuando el residuo ya está próximo a la solución.
4. Utilizar modelos o componentes con semillas inicializadas para el rango de interés del problema, es decir, predisposición al problema.

5.4.2 Eliminar traducción bloque-conjunto-bloque

Uno de los puntos menos eficiente del código es la evaluación de funciones bloque a bloque y la traducción de las mismas al conjunto. Es un punto crítico por dos cosas, obliga a hacer la evaluación de las funciones en JAVA Math que no es especialmente eficiente y obliga a realizar los cálculos de manera secuencial, ya que la traducción de componente al conjunto debe hacerse en un orden concreto que no permite la paralización del proceso.

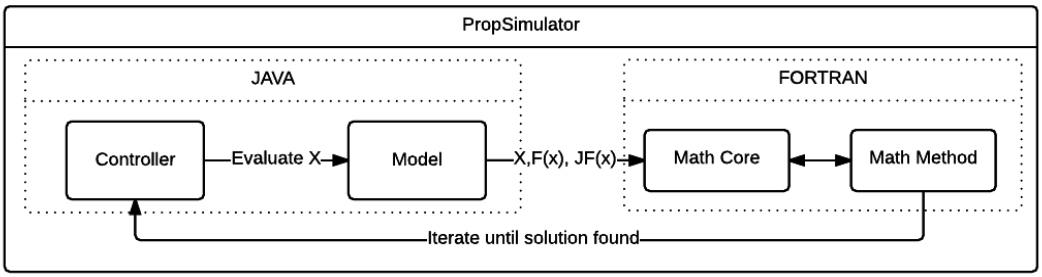


Fig. 31 Diagrama Actual del software

Para evitar este cuello de botella se ha ideado una nueva estrategia basada en la creación de “Funciones transferibles”, es decir, en vez de llamar al objeto para que evalué sus ecuaciones y devuelva la matriz asociada, se le solicita al objeto que nos transfiera sus funciones en forma de objeto.

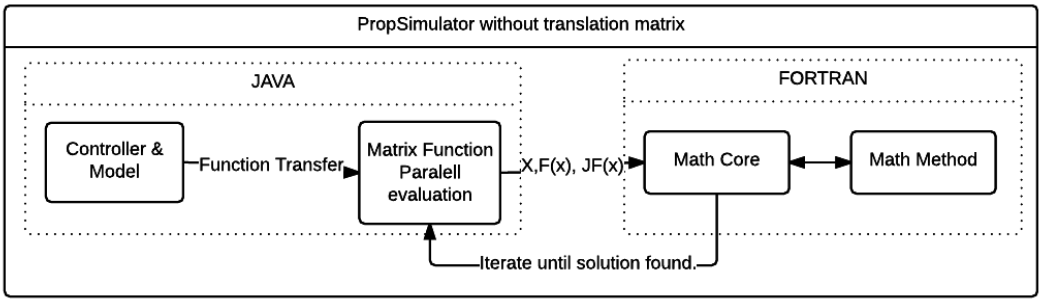


Fig. 32 Nueva estrategia para eliminar traducción de matrices

Las funciones y variables de todos los objetos son almacenadas en una matriz de objetos.

Finalmente el core matemático, evalúa de forma paralelizada las funciones sobre las variables creando directamente la matriz del conjunto y eliminando la necesidad de una traducción.

Otro de los puntos fuertes de esta idea es que esta capacidad de “enviar funciones” también existe en c++, lo cual puede permitir diseñar el software en JAVA con sus ventajas y abstracciones y paralelizar la evaluación de las funciones desde C++.

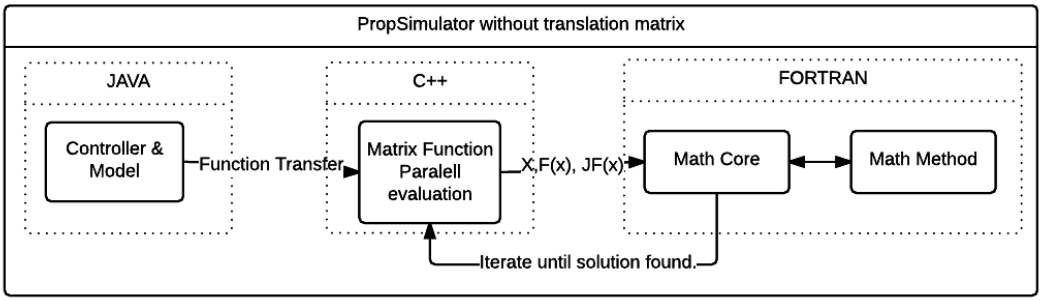


Fig. 33 Estrategia computacionalmente optimizada.

Es decir, permite evolución hacia un Core de evaluación y resolución matemática en C++ - FORTRAN, externo o compilado a través de JNI mientras que el planteamiento del engine, los modelos e interfaces graficas están soportados por JAVA con sus ventajas de abstracción e interoperabilidad entre sistemas.

5.4.3 **Valores iniciales no coherentes**

Actualmente se ha implementado un sistema proactivo en la corrección de variables input cuando el usuario proporciona las variables del bloque. Esto se ha conseguido a través de ecuaciones booleanas como se indica en 4.11.1. Sin embargo, cuando se interconectan varios módulos y se relacionan sigue existiendo la posibilidad de que las variables proporcionadas en ambos bloques sean incompatibles con las relaciones de los mismos no pudiendo obtener una solución al problema. Además, no es factible generar un sistema de ecuaciones booleanas estático para todo el conjunto de bloques, ya que la interconexión de los bloques depende de cada simulación que desee realizar el usuario.

Se ha planteado el uso un sistema de detección del problema mediante el cálculo del determinante de la matriz del jacobiano sobre las funciones. Evaluado sobre funciones continuas, indicaría si es resoluble por ser diferentes de cero. Sin embargo, es necesario evaluar que validez o equivalencia se puede encontrar para nuestro caso donde tenemos valores discretos en base a una semilla. Igualmente mantiene la problemática de detección del problema pero no indica al usuario el bloque o variable que genera el conflicto.

Una solución compleja parece ser la creación de un sistema booleano algebraico dinámico que obteniendo las f_i para cada conjunto de variables interrelacionadas, junto a la generación de nuevas ecuaciones booleanas a partir de las relaciones de cada variable sea capaz de generar un sistema algebraico booleano para el conjunto.

Finalmente la opción más interesante aunque menos investigada se basa en la modelización mediante de teoría de grafos para modelar la relaciones entre las variables a nivel de bloque, y a través de las relaciones de cada bloque permitir idear una especie de diagrama que estudiando sus propiedades nos indique, si existen dependencia o no en las variables introducidas.

6. **Conclusiones**

Durante la realización de este software y la documentación asociada, se ha conseguido consolidar los conocimientos adquiridos durante la carrera en materia termodinámica y propulsiva, con el objetivo de permitir la simulación de componentes o motores a través del software desarrollado en diferentes escalas de modelación. Permitiendo la resolución de la totalidad de los problemas relacionados con el cálculo de turbomotores o componentes en las diferentes asignaturas.

Se ha obtenido un software que cumple con los requisitos establecidos, así como se ha demostrado que es factible flexibilizar y dinamizar la elección de componentes y modelos dentro de una herramienta de simulación de turbomotores, mejorando o igualando otros softwares previos. Transformando el verdadero merito o valor en el "como" de la implementación, fundamentando una base común sobre la que generar modelos más complejos e interfaces visuales más cuidadas en versiones posteriores.

Se ha obtenido unos resultados valiosos que van a condicionar las decisiones de mejora o reestructuración del propio software para su versión Beta en 2016.

Entre ellas destaca la reestructuración de las clases hacia un modelo no recursivo durante el cálculo matemático y en la utilización de nuevos métodos de solución matemática no dependientes del Jacobiano. Y la adimensionalización de las ecuaciones con el fin de consensuar de una manera más homogénea el error de la solución final y reducir el número de iteraciones necesarias.

Finalmente, se concluye que la versión Alfa de este software, ha conseguido obtener casi la totalidad de las expectativas puestas en él, al rediseñar, mejorar y absorber las características principales de softwares previos. Por otra parte, ha permitido analizar nuevos problemas y las mejoras necesarias para resolverlos, así como detectar los verdaderos puntos críticos de la flexibilización y dinámica modular del software, con el fin de servir como base y contribuir para futuros desarrollos más estables de este software (versión Beta) u otros software relacionados interesados en la programación flexible a bloques.

Bibliografía

[1] Yunus A.Cengel, Michael A.Boles: “Thermodynamics“ “6 edition”, Mc Graw Hill edicion, 2009.

[2] Ferran Vilella Moreno, Jose Vigo Medina, David Garcia Soto “Gas Turbine Desing & Analysis Tool”, editorial PFG 2012.

[3] Michael A.Boles, “*EES software*”, North Carolina State university.

[4] Dixon, S. L., “Introduction: Dimensional Analysis: Similitude”, Cap. 1 en Fluid Mechanics, Thermodynamics of Turbomachinery, Elsevier Butterworth-Heinemann editorial, pág. 1-23, Oxford, 1998.

[5] Baskharone, E.A., Principles of Turbomachinery in Air Breathing Engines, Cambrige University Pres, Cambrige, 2006.

[6] Hoffman, J. D., Numerical Methods for Engineers and Scientists, Marcel Dekker. Inc., Basel, 2001.

[7] Transparencias Asigntura Proppulsion y Termodinamica.

[8] Internet MIT notes (20-06-2015) http://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/ .

[9] John A. Reed PDF - Paper Blog publish “ The Java Gas Turbine Simulator Applet and Documentation”.

[10] John A. Reed PDF - Paper Blog publish “Java Gas Turbine Simulator: Engine Component Mathematical Models”.

[11] https://tecnolleratoyeso.files.wordpress.com/2012/11/turborreactor.jpg (15-09-15) Image search by google.

[12] http://www.wikiwand.com/it/Argus_As_014 (28-09-15) Image search by google.

[13] http://www.sr692.com/fleet/20_dc832/index.html (02-10-15) Image search by google.

[14] http://nycaviation.com/newspage/wp-content/uploads/2009/11/airfrance-a380-roll-1024.jpg (17-09-15) Image search by google.

[15] http://bemil.chosun.com/nbrd/files/BEMIL091/upload/2006/07/an70(6-1).JPG (11-09-15) Image search by google.

[16] http://www.textoscientificos.com/imagenes/fisica/termodinamica-1.gif (11-10-15) Image search by google.

[17] http://public.lanl.gov/davidp/MSThesisHTML/2.html (11-10-15) Image search by google.

[18] http://public.lanl.gov/davidp/MSThesisHTML/2.html (11-10-15) Image search by google.

1. Anexo A: Termodinámica

En este anexo se dispone de una definición y explicación más detallada, con demostraciones o ejemplos de los conocimientos necesario para entender el análisis, ecuaciones y predicciones de cada uno de los componentes de un propulsor a reacción.

La Termodinámica puede definirse como la rama de la física que describe los estados en equilibrio a nivel macroscópico. Se basa en un método experimental de medición de magnitudes que junto a razonamientos deductivos permite describir y caracterizar como responden los sistemas a los cambios de su entorno, normalmente cambios de temperatura o fuerza (termo-dinámica).

Por lo tanto, se pueden usar los principios termodinámicos para analizar objetos de dimensiones 'visuales', introducir cambios en su entorno y calcular la evolución final (estable) de los cambios introducidos.

1.1 *Tipos de Sistemas*

Si definimos una línea imaginaria que aísla y permite el estudio de una parte del universo, estamos definiendo un sistema termodinámico.

Encontramos cuatro entidades, el sistema, la pared que envuelve al sistema y el entorno que envuelve a la pared, finalmente la suma del entorno y sistema es el universo en el que vivimos. Cuando indicamos que la pared aísla, nos referimos a que limita o define una frontera para el intercambio de masa, energía, volumen etc…

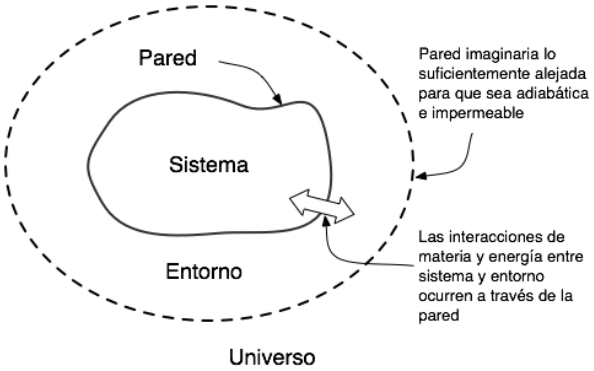


Fig. 34 Sistema termodinámico y universo [16].

Podemos definir tres tipos de sistemas principales:

- Sistema cerrado: Es aquel en el que la masa es constante.
- Sistema abierto: Es aquel en el que el volumen es constante pero la masa no.
- Sistema aislado: Es aquel en el que no se intercambiar energía térmica con el exterior, masa y el volumen es constante.

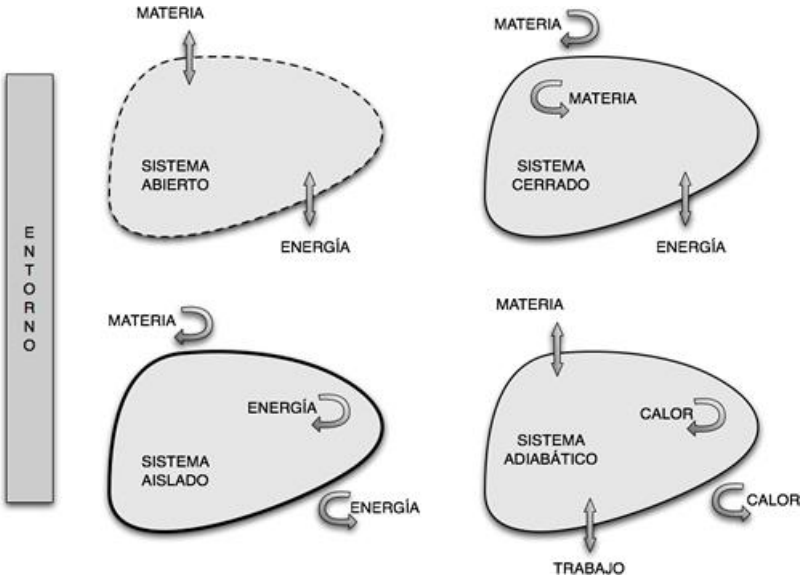


Fig. 35 Tipos de sistemas termodinámicos.

Una roca en el espacio es un sistema cerrado pues la masa del mismo es constante; el túnel de una carretera es un sistema abierto donde el volumen es constante pero la masa no; una nevera de playa ideal, es un sistema aislado ya que no permite el paso del calor hacia el interior.
El universo cósmico en su totalidad puede definirse como un sistema aislado.

De la misma manera podemos clasificar las paredes de los sistemas como:

- Adiabáticas: Impide el paso del calor, pero si permite el paso de energía en forma de trabajo.
- Diatérmanas o diatérmicas: Permiten el paso de energía en forma de calor no de trabajo.
- Móvil / Rígida: Permite o no permite transferir trabajo mecánico.

Cuando se analiza un motor de avión es posible identificar un sistema abierto a volumen constante (rígido), donde el aire entra y sale del mismo.

No obstante se puede hacer la aproximación de que la masa total de aire es constante ya que masa de aire que entra y sale y su presión de entrada y salida son aproximadamente las mismas. Otra manera de entender esta aproximación es definir el sistema incluyendo la atmosfera que rodea el avión, de esta manera asumimos que el aire expulsado se enfría y desacelera en la atmosfera y vuelve a entrar al motor.

Por lo tanto, es posible modelar un motor de avión ideal como si fuera un sistema cerrado, haciendo recircular el aire del mismo (ciclo termodinámico) o como sistema abierto asumiendo que siempre hay la misma cantidad de aire dentro del motor cuando queremos analizar al detalle las propiedades.

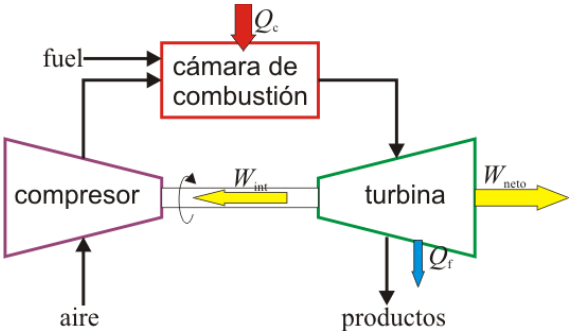


Fig. 36 Sistema abierto.

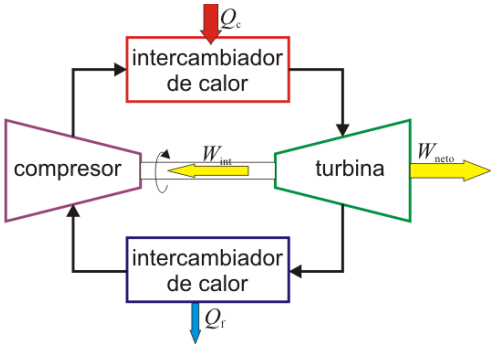


Fig. 37 Sistema Cerrado

1.2 *Tipos de variables*

El estado de un sistema termodinámico se determina mediante las variables o propiedades macroscópicas, son magnitudes medibles como fuerza, superficie, temperatura, volumen, masa etc…

Se dividen en dos grupos de variables:

- Propiedades extensivas: Son las que su valor depende de la medida o masa del sistema (volumen y masa).
- Propiedades intensivas: No depende de la masa del sistema (presión, temperatura, densidad).

Existen propiedades intensivas que se derivan de las extensivas cuando éstas últimas se definen por unidad de masa volumen, o cantidad de sustancia, por lo que reciben las denominaciones de variables específicas, volumétricas o molares, respectivamente. Una forma muy habitual de ver estas variables es dividida por unidad de tiempo, permitiendo obtener las unidades de las mismas por segundo de calor, trabajo y energía, muy útiles en flujos estacionarios como un motor a reacción.

Variables extensivas [unidades]	**Variables especificas** [unidades/masa]	**Especificas/tiempo** [unidades/ (masa·tiempo)]
$\boldsymbol{U, H, Q, W}$	u, h, q, w	$\dot{u}, \dot{h}, \dot{q}, \dot{w}$

1.3 *Tipos de procesos*

Se define con el término ‘proceso’ a cualquier cambio de un estado a otro, que experimenta un sistema. Los procesos pueden ser clasificados según dos categorías, por evolución o por propiedades.

Por evolución temporal y reversibilidad del proceso:

- Proceso Cíclico: El estado final es el mismo que el inicial.

- Proceso Cuasi estático (ideal): Es infinitamente lento, por ello el sistema solo pasa por estados de equilibrio, lo que lo hace reversible.
- Proceso No Cuasi estático: El sistema pasa por estados de no equilibrio, generándose entropía en el proceso, es irreversible.

Por las propiedades termodinámicas involucradas en el proceso:

- Proceso Isotermo: Se realiza a temperatura constante.
- Proceso Isobaro: Se realiza a presión contante.
- Proceso Isométrico: Se realiza a volumen constante.
- Proceso de flujo estacionario: Son procesos que pueden variar en el espacio pero no con el tiempo.
- Proceso Adiabático: Se realiza sin transferencia de calor.
- Proceso Politrópico: Se realiza un intercambio de energía tanto en el interior del sistema que contiene los gases, como una transferencia con el exterior del sistema.

Cuando se analiza un motor de avión, como un sistema estacionario, es decir, los procesos en el que tiene lugar son cuasi estáticos idealmente, con procesos de flujo estacionario.

Dentro de cada uno de los componentes del mismo se encuentran procesos adiabáticos idealmente (compresor y turbina), procesos isobaros (cámara de combustión) y procesos politrópicos.

La pertenencia a un tipo no excluye de otro; por lo tanto, es posible tener un proceso adiabático, politrópico, cuasi estático.

1.4 *Trabajo, Calores Específicos y Energías*

Para entender cómo cambian y bajo qué condiciones la presión, la temperatura y la energía del fluido, es necesario definir y relacionar los tipos de energías y constantes caloríficas del fluido.

1.4.1 **Energía interna**

La energía interna simbolizada como 'U' o su versión especifica 'u' se define como el reflejo a escala macroscópica de la energía de la suma de energías cinética, potencial interna u otros tipos de energía que almacenan las partículas individuales del sistema en estudio. No incluye la energía cinética de traslación o rotación del conjunto, ni la energía potencial gravitatoria o electroestática del conjunto. Se puede demostrar que la energía interna de un gas ideal únicamente depende del material y la temperatura del mismo.

Permite describir cuanta energía esta almacenada en la entidad con la siguiente ecuación [x], separando la energía del conjunto de la referida a la situación del conjunto de partículas (potencia y cinética).

$$Energia\ total = U_{intena} + E_{potencial} + E_{cinetica} \quad (30)$$

Un ejemplo sencillo de energía interna, es calentar una piedra que está en reposo, su velocidad y potencial se mantiene constante, la energía transmitida se almacena internamente como calor (o lo que es lo mismo energía cinética de vibración de las partículas).

Si en vez de calentar la piedra, se realiza algún tipo de trabajo sobre ella como una compresión, aumentando su densidad, esa energía en forma de trabajo es transmitida a energía interna.
Podemos asumir que si no hay variación de energía cinética o potencial (situación en reposo):

$$\Delta U = Q - W = m \cdot (u_2 - u_1) \quad (31)$$

O despejando el calor:

$$Q = \Delta U + W \quad (32)$$

U: Energía interna
Q: Calor
W: Trabajo aplicado sobre el sistema
m: Masa

1.4.2 **Entalpia**

Es una magnitud termodinámica expresada con la letra 'H' o su versión especifica 'h' que permite medir la cantidad de energía absorbida o cedida por un sistema termodinámico, es decir, la cantidad de energía intercambiada con el entorno. Se expresa como la energía del sistema y los cambios realizados al mismo en forma de trabajo o calor transferido. Si el proceso es isobárico la entalpia es el calor transferido.

$$H = U + W \quad (33)$$

1.4.3 **Trabajo**

El trabajo es una magnitud física que se representa con la letra 'W' y se expresa en unidades de energía. Se refiere a la energía necesaria transferida por una fuerza para realizar un desplazamiento. Aplicando el concepto a una fuerza ejercida por una presión, pasamos de integrar longitudinalmente a integrar respecto al volumen del sistema.

$$W = \int F\ dr = \int P\ dV \quad (34)$$

Para un sistema abierto o cerrado a presión constante, el trabajo de crear el sistema es:

$$W_{isobarico} = (P \cdot \Delta V) = P_{sistema} V_{sistema} \quad (35)$$

Si el volumen del sistema se mantiene constate, se asumen procesos ideales a temperatura constante (isotérmico) y recordando la ecuación estado del gas ideal:

$$PV = nRT = R_{gas}T = cte \quad (36)$$

$$W_{isentropico} = \int P\, dV = \int_A^B \frac{R_{gas}T}{V}\, dV = R_{gas}T \cdot ln\frac{V_A}{V_B} \quad (37)$$

En el caso anterior hemos asumido que el sistema está es un sistema cuasi estático y gases ideales. Sin embargo si el proceso se realiza "más rápido", de una manera más real, hablamos de procesos politrópicos, adiabáticos cuasi estáticos, donde presión, volumen y temperatura no son constantes y la relación entre presión y volumen puede definirse como:

$$PV^{\gamma} = \alpha = cte \quad (38)$$

$$W = \int P\, dV = \alpha \int_A^B \frac{1}{V^{\gamma}}\, dV = \left[\frac{\alpha}{1-\gamma}V^{1-\gamma}\right]_A^B =$$

$$= \frac{1}{1-\gamma}\cdot\left[\frac{\alpha}{V_B^{\gamma-1}} - \frac{\alpha}{V_A^{\gamma-1}}\right] = \frac{1}{1-\gamma}\cdot\left[\frac{P_B * V_B^{\gamma}}{V_B^{\gamma-1}} - \frac{P_A * V_A^{\gamma}}{V_A^{\gamma-1}}\right] = \frac{nR}{1-\gamma}\cdot[T_B - T_A]$$

$$W_{politropico} = m\frac{R_{gas}}{1-\gamma}\Delta T \quad (39)$$

Por lo tanto, el trabajo realizado por el sistema inicialmente depende del fluido y del cambio de temperaturas.

1.4.4 **Calor especifico.**

Se define el calor específico o capacidad calorífica específica 'c' como la magnitud física que expresa la cantidad de calor que hay que suministrar por unidad de masa a un sistema termodinámico para elevar su temperatura en una unidad.

$$c = \frac{Q}{m\Delta T} = \frac{1}{m}\cdot\frac{dQ}{dT} = \frac{dq}{dT} \quad (40)$$

c: Calor especifico.
m: Masa.
Q: Calor.
T: Temperatura.

Para los sistemas cerrados o de flujo constante, donde la masa no cambia es un parámetro importante ya que nos permite establecer la relación entre temperatura y energía almacenada en el sistema.

Para una presión constante se define una constante calorífica denominada c_P, para un sistema a volumen constante se define una constante calorífica c_V.

El valor de $c_P > c_V$ para un mismo fluido siempre será mayor, ya que a presión constante se permite que el sistema se expanda (trabajo del sistema).

Por la definición de energía interna a un volumen constante y su dependencia con la temperatura podemos expresar:

$$du = c_V(T)\, dT \quad (41)$$

A presión constante, la relación de entalpia y energía interna es:

$$dh = c_p(T)\, dT \quad (42)$$

Si recordamos que:

$$\left\{\begin{matrix} h = u + Pv \\ Pv = RT \end{matrix}\right\} \; h = u + RT \quad (43)$$

$$dh = du + RdT\,; \quad \frac{dh}{dT} = \frac{du}{dT} + R \quad (44)$$

$$C_P = C_V + R \quad (45)$$

$$\gamma = \frac{C_P}{C_V} \quad (46)$$

Se puede observar que según el tipo de proceso se obtendrán diferentes valores para la capacidad calorífica.

Tipo de Proceso	*Calor específico (c)*	*Entalpia específica (h)*	*Trabajo específico (w)*
Isotermo	∞	0	$R_{gas}T \cdot ln\frac{V_A}{V_B}$
Adiabático	0	0	$P \cdot \Delta V$
Isométrico	$C_V = \left(\frac{\partial u}{\partial T}\right)_{V=cte}$	$C_V \cdot T$	0
Isobaro	$C_P = \left(\frac{\partial h}{\partial T}\right)_{P=cte}$	$C_P \cdot T$	$P \cdot \Delta V$

1.5 *Leyes de la Termodinámica*

La termodinámica se basa en tres leyes fundamentales y un principio.

1.5.1 Principio Cero

Existe una propiedad denominada temperatura empírica (θ) que es común para todos los estados de equilibrio termodinámico, que se encuentran en equilibrio mutuo, es decir, si pones varios objetos en contacto con diferentes temperaturas, evolucionaran hasta una situación de equilibrio con una temperatura única para todos.

1.5.2 Primera Ley Termodinámica

Principio de la conservación de la energía. Establece que si se realiza un trabajo sobre el sistema o un intercambio de calor, la energía interna del sistema cambiara.

$$E_{entra} - E_{sale} = \Delta E_{sistema} \quad (48)$$

Simplificando el problema a dos fuentes de energía, en forma de trasferencia de calor o de trabajo:

$$\Delta U = Q - W \quad (49)$$

1.5.3 Segunda Ley Termodinámica

Define el concepto de irreversibilidad y entropía. He indica que la entropía de un sistema siempre aumenta o se mantiene constante.

$$\frac{dS}{dt} \geq 0 \quad (50)$$

1.5.3.1 Entropía

Se puede definir la entropía ('S') como una magnitud física que permite determinar que parte de la energía no puede utilizarse para producir trabajo.

Una manera más conceptual define la entropía como una medida de la distribución sobre el orden de un sistema. Si el sistema está muy desordenado, existe cierta cantidad de energía 'bloqueada', para ordenar el sistema ante un cambio.

Los sucesos en la naturaleza además de ser conservativos (masa, energía etc…) tienden a la uniformidad, esto es debido a que en un sistema con distribución aleatoria, la probabilidad más alta es la homogénea (azar), mientras que una estructurar compleja e irregular es improbable y tiende a modificarse hacia un

sistema más probable y más regular. Esta inclinación en las probabilidades de un suceso está determinada por la propensión del universo a uniformizar la energía. La entropía al ser una magnitud que está relacionada con el orden del sistema queda maximizada cuanto mayor sea la uniformidad del sistema.

Por lo tanto para saber cómo determina el sentido del proceso únicamente hay que ver en qué estado el sistema maximiza la entropía del mismo. Así, se define procesos no reversibles ya que no es posible volver al estado inicial una vez cambiado ya que parte de la energía queda 'bloqueada' sin utilidad y no puede ser transformada para volver al estado inicial.

Aunque inicialmente se modelaron todos los procesos como ideales, (entropía constante) la realidad es que en cualquier proceso se está produciendo una pérdida de la energía por irreversibilidad, está perdida normalmente se denota como eficiencia o rendimiento térmico.

$$\eta_{termica} = \frac{E_{producida}}{E_{suministrada}} = \frac{E_{salida}}{E_{entrada}} \quad (51)$$

1.5.4 **Tercera Ley Termodinámica**

Define como temperatura mínima 0º K o lo que es lo mismo -273.15 º C y que es imposible alcanzar dicha temperatura mediante un numero finito de pasos. Por lo tanto, la entropía es nula a cero grados kelvin.

1.6 ***Ciclo termodinámico***

Podemos visualizar el ciclo termodinámico de un motor a reacción utilizando los diagramas Presión-Volumen específico y Entalpia-Entropía (normalmente simplificada la entalpia como temperatura).

La trayectoria realizada por un turbojet es el denominado ciclo de Brayton. En ella podemos visualizar una compresión adiabática (1-2), incremento de temperatura isobárico (2-3), expansión adiabática (3-4) y enfriamiento a presión constante (4-1).

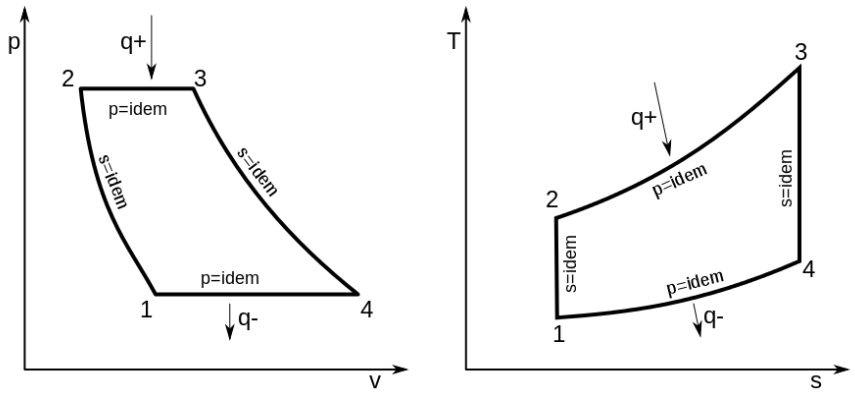


Fig. 38 Ciclo termodinámico P-V R-S

El trabajo realizado por el ciclo es el área entro del polígono definido por la trayectoria del fluido a través del motor. Y la eficiencia del ciclo puede ser definida como:

$$\eta_{ciclo} = 1 - \frac{q_+}{q_-} = 1 - \frac{h_4 - h_1}{h_3 - h_2} = 1 - \frac{T_4 - T_1}{T_3 - T_2} \approx 1 - \frac{T_1}{T_2} \quad (52)$$

De donde podemos deducir que cuanta mayor sea la temperatura después de la combustión o cuanto menor sea la temperatura después de la compresión más eficiente será el motor. Por otra parte si optimizamos la expansión de los gases a presión atmosférica las pérdidas de temperatura de T4 a T1 se minimizan.

Si analizamos los procesos más realistamente, con compresiones y expansiones no irreversibles se genera entropía:

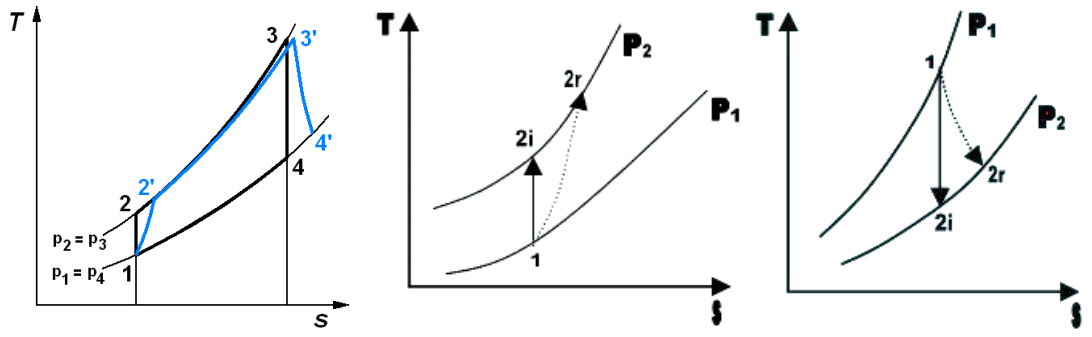


Fig. 39 Ciclo termodinámico ideal vs real

La generación de entropía tiene como resultado que para una compresión de P1 a P2 es necesario invertir más energía en el proceso, obteniendo una temperatura sensiblemente mayor en T2. Y de manera inversa al realizar una expansión se extrae menos energía, por lo que el fluido termina a una temperatura mayor.
Otro detalle del ciclo realista se basa en el diseño de la cama de combustión existe un pequeño diferencial de presión, direccionando el flujo hacia la turbina por ello la trayectoria seguida durante el proceso real alcanza la misma temperatura que el ideal pero a una presión ligeramente inferior.

1.7 *Velocidad del sonido*

Se define velocidad del sonido, como a la velocidad máxima a la que una onda de presión puede viajar a través del medio.

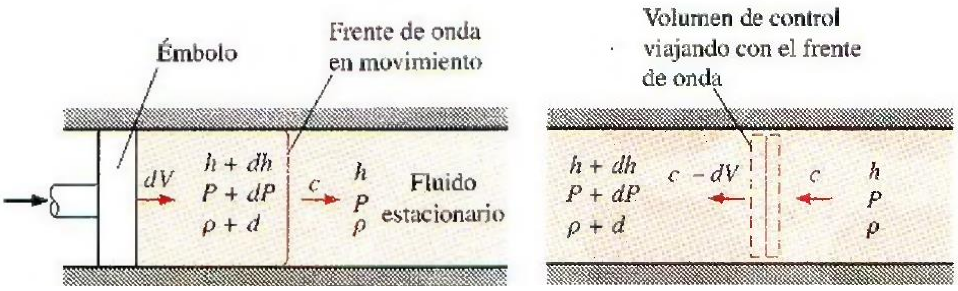


Fig. 40 Análisis de velocidad sónica.

Aplicamos conservación de masa y energía en el volumen de control, calculando las ecuaciones diferenciales despreciando los términos de segundo orden.

$$\dot{m}_{derecha} = \dot{m}_{izquierda} \quad (53)$$

$$\rho AV = (\rho + d\rho)A(V_{sónica} - dv) \quad \rightarrow \quad V_{sónica}d\rho - \rho dV = 0 \quad (54)$$

No existe ni trabajo calor que traviese las fronteras del volumen de control.

$$h + \frac{V_{sónica}^2}{2} = h + dh + \frac{(V_{sónica} - dV)^2}{2} \rightarrow \quad dh - VdV = 0 \quad (55)$$

Se puede demostrar que la propagación de una onda de presión no solo es adiabática sino que aproximadamente es isentrópica permitiendo definir:

$$dh = \frac{dP}{\rho} \quad (56)$$

Combinando las anteriores ecuaciones se puede llegar a véase cap12 de [1]:

$$a^2 = V_{sónica}^2 = \left(\frac{dP}{d\rho}\right)_{S=cte} = \gamma \cdot \left(\frac{dP}{d\rho}\right)_r \quad (57)$$

Para un fluido que es un gas ideal $P = \rho RT$:

$$a = \sqrt{\gamma RT} \quad (58)$$

$$Ma = \frac{V}{a} \quad (59)$$

1.8 *Eficiencias*

Cuando definimos eficiencia, nos referimos a los resultados obtenidos en comparación con los resultados ideales esperados. Principalmente nos referimos en clave energética aunque se pueden definir pseudo eficiencias basándonos en

parámetros más directos de medir, aunque su abstracción siga siendo energética.

1.8.1 Eficiencia térmica

Define la eficiencia en función de la energía saliente respecto a la energía entrante y trabajo o calor transferido al fluido.

$$\eta_{th} = \frac{E_{out}}{E_{in}} \quad (60)$$

1.8.2 Eficiencia isentrópica

Define la eficiencia de un componente respecto al componente ideal, es decir, describe la entropía generada del elemento. Se aplica sobre elementos que extraen o transfieren trabajo.

$$\eta_i = \frac{W_{ideal}}{W_{real}}\Big]_{W>0} = \frac{W_{real}}{W_{ideal}}\Big]_{W<0} \quad (61)$$

Su definición depende de si el trabajo que realiza es positivo o negativo. Aquellos casos donde el trabajo es extraído, el real obtendrá un trabajo inferior al elemento ideal, mientras que en el caso de transferir trabajo al fluido el ideal necesitara más trabajo que el modelo ideal.

1.8.3 Eficiencia politrópica

Se refiere a la eficiencia individual de las diferentes etapas que podemos encontrar en una compresión o expansión. Conforme se aumenta en la línea T-S aunque las etapas tengan una eficiencia similar, la entropía generada por las etapas en una posición más alta de línea será mayor. Cuanto más caliente esta un gas, más difícil es comprimirlo de manera real, es decir, mayor es la entropía generada, por consiguiente la eficiencia politrópico define un rendimiento isotrópico de un escalonamiento elemental tal que se mantiene constante a lo largo del todo el proceso. Al igual que en la eficiencia isentrópica, el sentido del trabajo invierte las definiciones par compresor o turbina.

$$\eta_p = \frac{dT}{dT_s}\Big]_{expansion} = \frac{dT_s}{dT}\Big]_{compresion} = cte$$

$$\frac{dT}{T} = \frac{\gamma - 1}{\gamma} \cdot \frac{dP}{P}; \quad \left(\frac{T}{P}\right)^{\frac{\gamma-1}{\gamma}} = \alpha$$

$$\eta_{cp} = \frac{\gamma - 1}{\gamma} \cdot \frac{\ln\left(\frac{P_{out}}{P_{in}}\right)}{\ln\left(\frac{T_{out}}{T_{in}}\right)}; \ \eta_{tp} = \frac{\gamma}{\gamma - 1} \cdot \frac{\ln\left(\frac{T_{out}}{T_{in}}\right)}{\ln\left(\frac{P_{out}}{P_{in}}\right)}$$

$$\pi_c = \tau_c^{\frac{\gamma}{\gamma-1}*\eta_{cp}};\ \pi_t = \tau_t^{\frac{\gamma}{\gamma-1}*\frac{1}{\eta_{tb}}} \quad (62)$$

1.8.4 **Eficiencia mecánica**

Se refiere a aquellos elementos mecánicos como el eje, que sufren pérdidas energéticas mayoritariamente por fricción.

$$\eta_{mec} = \frac{W_{M_{trasnferido}}}{W_{M_{absorvido}}} \approx \frac{W_M - W_{fricción}}{W_M} \quad (63)$$

1.9 ***Variables y propiedades***

Con el fin de facilitar el cálculo o abstraer conclusiones energéticas de cada uno de los componentes, se defines variables auxiliares que hacen adimensional cálculos o permiten tener en cuenta los cambios de las propiedades del fluido referenciadas al fluido inicial.

1.9.1 **Propiedades de estancamiento**

Para poder calcular los diversos cambios que sufre el fluido energéticamente a lo largo del motor se define las denominadas presiones o temperaturas totales.

Estas propiedades reflejan el equivalente en presión y temperatura de un fluido a velocidad cero. Cuando el fluido atraviesa el motor, asumimos que su velocidad es cercana a cero, sin embargo este axioma no es cierto, el fluido no tiene velocidad sino que se acelera y decelera a lo largo del motor. Para poder trabajar con presiones o temperaturas independientes de estos cambios, se calcula propiedades totales que incluyen una transformación adiabática de la energía cinética en entalpia del fluido.

$$Q = (h_0 - h) + \frac{1}{2}(0 - V^2) = 0$$

$$h_0 = h + \frac{1}{2}V^2;\quad h = cp \cdot T;\ M = \frac{V}{a} = \frac{V}{\sqrt{\gamma RT}} \quad (64)$$

Despejamos la temperatura, sustituimos velocidad por mach, y aproximamos por Taylor de orden 1:

$$T_{t1} = F(T_1) = T_1 + \frac{V^2}{2 \cdot cp} \approx T_1 \left(1 + \frac{\gamma - 1}{2}M^2\right) \quad (65)$$

$$P_{t1} = P_1 \cdot \left(\frac{T_t}{T}\right)^{\frac{\gamma}{\gamma-1}} \quad (66)$$

$$M_1^+ = \frac{V_1}{\sqrt{\gamma R T_{t1}}} = \frac{M_1}{\sqrt{1 + \frac{\gamma-1}{2} M_1^2}} \quad (67)$$

1.9.2 **Propiedades adimensionalizadas**

Si hacemos adimensional las magnitudes, abstraemos los cambios y condiciones a una referencia común, permitiendo simplificar ecuaciones y trabajar con márgenes numéricos más sencillos que se interpretan independientes de las condiciones externas del motor.

$$\theta_i = \frac{T_{t_i}}{T_0}; \quad \delta_i = \frac{P_{t_i}}{P_0}; \quad \pi_i = \frac{P_{t_i}}{P_i}; \quad \tau_i = \frac{T_{t_i}}{T_i};$$

$$\delta_0 = \theta_0^{\frac{\gamma}{\gamma-1}}; \quad \pi_i = \tau_i^{\frac{\gamma}{\gamma-1}}$$

Un parámetro importante es la adimensionalización del flujo de masa, ya que relaciona el flujo másico de fluido que atraviesa la sección del motor con su área, presión y temperatura. Por otra parte permite diseñar el motor referido a parámetros que permiten su escalada según la potencia deseada.

$$\mathcal{M} = \frac{\dot{m}\sqrt{T_t}}{P_t A} = \sqrt{\frac{\gamma}{r}} M^+ \cdot \left(1 + \frac{\gamma-1}{2} M^{+2}\right)^{\frac{1}{\gamma-1}} \quad (68)$$

$$\wp = \frac{\dot{m}\sqrt{T_t/T_0}}{P_t/P_0} = \frac{\dot{m}\sqrt{\theta}}{\delta} \quad (69)$$

1.9.3 **Propiedades por estación**

Para indicar a que región del motor nos referimos se definen un conjunto de estaciones genéricas para todos los motores para presiones, temperaturas totales y relaciones de las mismas dentro de cada componente.

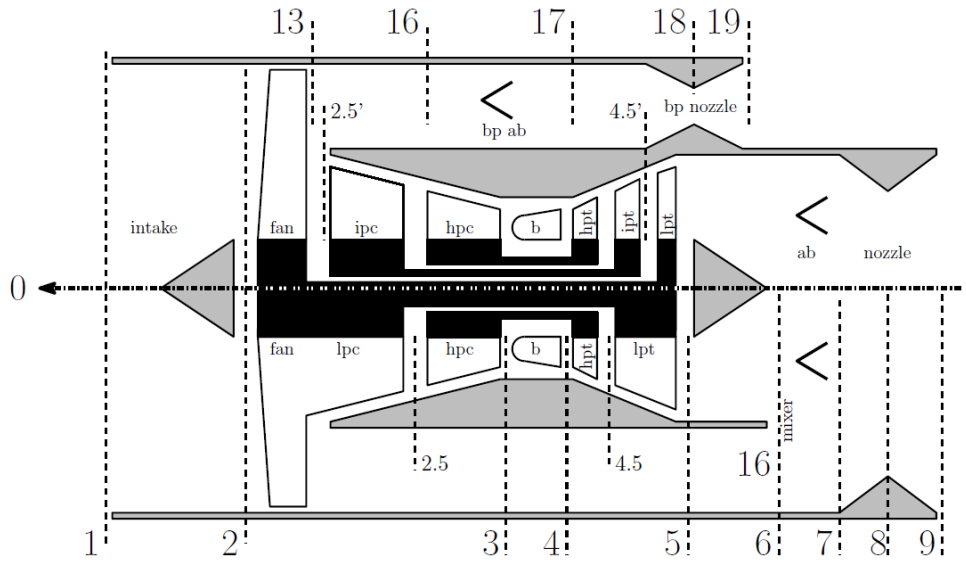


Fig. 41 Esquema de estaciones Turbomotor genérico.

component	T-ratio	p-ratio	eff. parameter
free stream	$\tau_{fs} = \theta_0$	$\pi_{fs} = \delta_0$	
intake	$\tau_i = T_{t_2}/T_{t_0}$	$\pi_i = p_{t_2}/p_{t_0}$	ε_i
compressor	$\tau_c = T_{t_3}/T_{t_2}$	$\pi_c = p_{t_3}/p_{t_2}$	η_c
burner	$\tau_b = T_{t_4}/T_{t_3}$	$\pi_b = p_{t_4}/p_{t_3}$	ε_b
turbine	$\tau_t = T_{t_5}/T_{t_4}$	$\pi_t = p_{t_5}/p_{t_4}$	η_t
afterburner	$\tau_{ab} = T_{t_7}/T_{t_5}$	$\pi_{ab} = p_{t_7}/p_{t_5}$	ε_{ab}
nozzle	$\tau_n = T_{t_9}/T_{t_7}$	$\pi_n = p_{t_9}/p_{t_7}$	ε_n
fan	$\tau_f = T_{t_{13}}/T_{t_2}$	$\pi_f = p_{t_{13}}/p_{t_2}$	η_f
bypass burner	$\tau_{bb} = T_{t_{17}}/T_{t_{13}}$	$\pi_{bb} = p_{t_{17}}/p_{t_{13}}$	ε_{bb}
bypass nozzle	$\tau_{bn} = T_{t_{19}}/T_{t_{17}}$	$\pi_{bn} = p_{t_{19}}/p_{t_{17}}$	ε_{bn}

1.10 *Definiciones propulsivas*

La función de un motorreactor es crear empuje, es decir, fuerza que propulse el avión en el que se encuentra instalado. Por consiguiente se definen definiciones propulsivas en base a la fuerza de empuje respectos a los parámetros del motor y se crean parámetros auxiliares que permiten evaluar la fuerza de un motor en referencia a su consumo, eficiencia, tamaño etc…

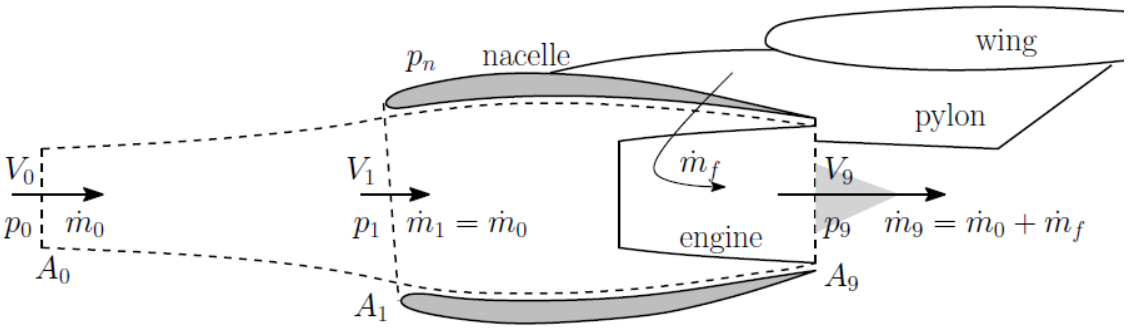
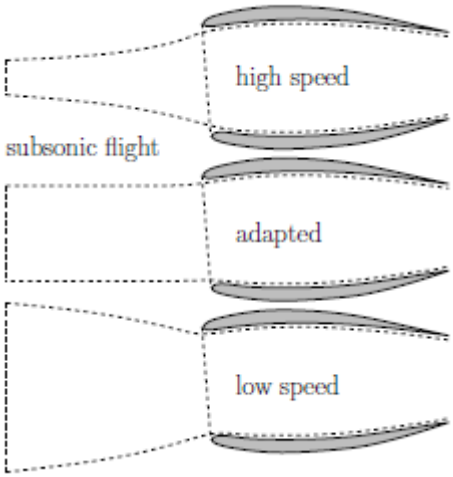


Fig. 42 Esquema básico de un turbomotor.

La fuerza generada por un motor, aplicando la tercera ley de newton queda definida de la siguiente manera siguiendo la figura 42.

$$F_{real} = \dot{m}_9 \cdot V_9 - \dot{m}_0 \cdot V_0 + (P_9 - P_0) \cdot A_9 - \left(D_a + D_{p_e} + D_{f_e}\right) \quad (70)$$

$$F_{tobera\ adaptada} \approx \dot{m}_9 \cdot V_9 - \dot{m}_0 \cdot V_0$$



Drag aditivo del difusor.

$$D_a = \int_0^1 P - P_0 \, dA \geq 0$$

Drag de la presión externa.

$$D_{p_e} = \int_1^9 P_n - P_0 \, dA$$

Drag de la fricción del motor.

$$D_{f_e} = \int_1^9 \tau_w \, P \, dx$$

Existen un conjunto de parámetros que analizan y caracterizan un motor:

- EPR (Engine Pressure Ratio): Relaciona la presión antes de la tobera con la presión después del difusor. Caracteriza la presión generada por el motor $\pi_{EPR} = \frac{P_{t7}}{P_{t2}}; \left(\propto \delta_7 = \frac{P_{t7}}{P_0}\right)$

- OPR (Overall Pressure Ratio): Relaciona presiones antes y después del compresor, caracteriza el trabajo del compresor. $\pi_{OPR} = \frac{P_{t3}}{P_{t2}}; \ \left(\propto \delta_3 = \frac{P_{t3}}{P_0}\right)$

- TET (Turbine Entry Temperature): Define la temperatura más alta del motor, la temperatura de la cámara de combustión.$TET = T_{t4}$
- EGT (Exhaust Gas Temperature): Define la temperatura de salida de los gases. $EGT = T_{t5}$
- Fuel/Air Ratio (richness): $\alpha = \frac{\dot{m}_{fuel}}{\dot{m}_0}$
- Specific Thrust or ISP: $\Psi = \frac{F}{\dot{m}_0}$
- TSFC o CTS (Trust-specific fuel consumption): $TSFC = C_{TS} = \frac{\dot{m}_{fuel}}{F}$
- Eficiencia Propulsiva: $\eta_{pr} = \frac{\dot{W}_F}{\dot{W}_S} = \frac{FV_0}{\dot{W}_S}$
- Eficiencia Total: $\eta_o = \frac{\dot{W}_F}{\dot{Q}_{in}} = \frac{F \cdot V_0}{\dot{m}_{fuel} \cdot h_f} = \eta_{th} \cdot \eta_{pr} = \frac{V_0}{C_{TS} \cdot h_f}$
- Fuerza útil: $P_F = F \cdot V_0 = \dot{m}_0 (V_9 - V_0) \cdot V_0 + \dot{m}_{fuel} \cdot V_9 \cdot V_0$
- Potencia desperdiciada: $P_W = \left(\dot{m}_0 + \dot{m}_{fuel}\right) \frac{(V_9 - V_0)^2}{2}$
- Potencia Total: $P_T = P_F + P_W = \left(\dot{m}_0 + \dot{m}_{fuel}\right) \cdot \frac{V_9^2 - V_0^2}{2}$

1. Anexo B: Componentes

En este anexo se detalla las definiciones y características básicas de los componentes de un motor a reacción.

Aunque a lo largo del anexo C se detalla los modelos utilizados para cada uno de los elementos es necesario conocer las definiciones de cada parámetro ya que en algunos casos como la eficiencia, se calculan de diferente forma para cada componente.

1.1 *Difusor.*

Se basa en un cuerpo geométrico cuya área aumenta, es un proceso adiabático donde el fluido se ve obligado a decelerarse, la energía cinética del mismo se transforma en presión y temperatura obteniendo finalmente un flujo a velocidad baja presión más alta.

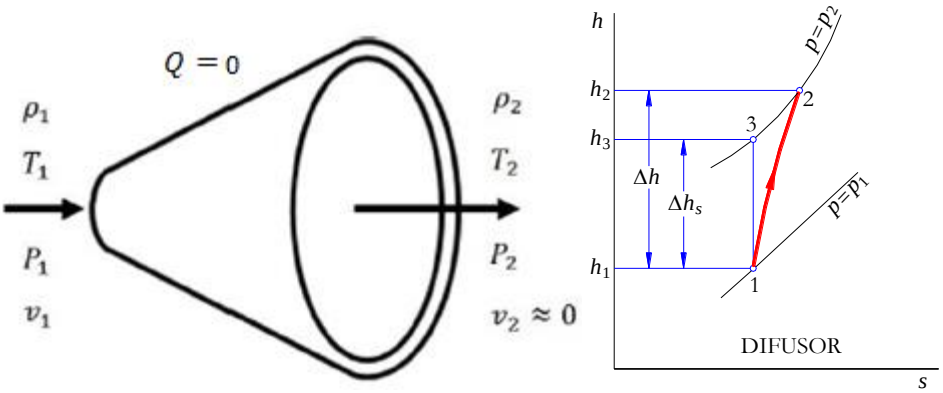


Fig. 43 Esquema de Difusor y diagrama entalpia-entropía

En el difusor se produce una compresión del fluido usando la energía cinética del mismo, el proceso puede definir una eficiencia del proceso en base a la entalpia total del fluido y la entalpia obtenida después del difusor. Sin embargo es poco práctica, normalmente la eficiencia del difusor depende de la velocidad de fluido, por lo tanto el resultado es un flujo con una presión total algo inferior a la esperada.

$$\eta_{difusor} = \frac{h_3 - h_1}{h_2 - h_1} = \frac{\Delta h_s}{\Delta h} \approx \frac{P_{t2} - P_1}{P_{t1} - P_1} \quad (71)$$

Sin embargo es un parámetro que no suele utilizarse salvo como análisis cualitativo del difusor normalmente usaremos, usaremos directamente las pérdidas de presión total al pasar por el difusor.

$$\epsilon_i = \frac{P_{t2}}{P_{t1}} \quad (72)$$

Durante la modelación de este componente no se han tenido en cuenta procesos complejos de compresión y ondas de choque que si se han realizado en la tobera.

La principal razón ha sido modelar un componentes sencillo que será completado para vuelo supersónico en las versión Beta al igual que en esta versión ya lo contempla para la tobera.

1.2 *Compresor.*

Se basa de un componente formado por alabes unidos a un eje, que comprimen el aires al ejercer sobre el mismo un trabajo y obligarlo a pasar por sección con un área decreciente en cada etapa.

Aunque existen compresores centrífugos y axiales este documento se centra principalmente es los axiales por su principal aplicación aeronáutica. Se pueden definir varias etapas de compresión, cada una con ejes independientes que permiten tener un compresor de alta y otro de baja presión.

Las eficiencias de un compresor se han definido en el punto 1.8 del Anexo A haciendo referencia a la eficiencia isentrópica y politrópico del mismo. El trabajo consumido por el compresor es positivo por definición (entra al sistema).

No se han asumido mapas de compresión, relación trabajo RPM, ni la entrada en perdida de los alavés. Estas características se sobre entienden en modelos más complejos a los implementados, por lo que únicamente se ha predispuesto de la estructura más genérica que permita implementarlos en un futuro cercano.

1.3 *Cámara Combustión.*

En la cámara de combustión se producen 2 tipos de procesos. El principal proceso es la combustión del fuel que aporta calor al fluido, el proceso secundario es la transmisión térmica del calor entre el propio fluido y la estructura en torno a él.

En esta primera versión del programa no se analizara al detalle la transmisión térmica del fluido ni los materiales que están en torno a él. Por lo tanto asumiremos una combustión y transmisión homogénea del calor a todo el fluido.

$$\eta_{fuel} = \frac{\Delta h_{real}}{\Delta \mathrm{h}_{\mathrm{ideal}}} \approx \frac{\dot{m}_{fuel\ quemado}}{\dot{m} fuel\ total} \qquad (73)$$

Sí que se modeliza la eficiencia de combustión como porcentaje de combustible que es quemado correctamente.

Aunque se asume proceso isobárico, existe un pequeño diferencial de presión en la cámara de combustión que expulsa los gases hace la turbina.

$$\pi_b = \frac{P_{t4}}{P_{t3}} \quad (74)$$

1.4 *Turbina.*

La turbina es un componente inverso al compresor, su estructura y ecuaciones son muy similares pero trabajando de manera inversa (expansión extracción de trabajo).

Uno de los puntos importante es la temperatura de turbina, ya que los materiales de la misma tienen unas limitaciones térmicas de funcionamiento que a su vez limitan la cantidad de energía en forma de combustible que se puede aportar al flujo sin riesgo de dañar la turbina.

En esta versión no se han implementados sangrados de compresor para realizar la refrigeración de la turbina ya que no se ha predispuesto de un modelo de intercambio de calor, que permitiese simular la temperatura de alabe, sin embargo la generación de sangrados o flujos de refrigeración se puede implementar fácilmente.

La eficiencia politrópico e isentrópica han quedado definidas en el punto 1.8 anexo A.

1.5 *Poscombustor.*

Se trata de una pseudo cámara de combustión, en aquellos motores que se desea tener un fuerte impulso durante un periodo corto, permite eyectar grandes cantidades de combustible (mezcla mayor que la cámara de combustión), aumentando drásticamente la temperatura del flujo y a través de la tobera la velocidad de los gases, es un proceso poco eficiente desde el punto de vista propulsivo pero muy útil para motores militares.

Se le aplican los mismos parámetros y modelos que a la cámara de combustión.

1.6 *Tobera.*

Al igual que el difusor se trata de un elemento geométrico adiabático, su principal función es la expansión de los gases. De manera muy similar al difusor se puede definir una eficiencia que asocie las perdidas por fricción o transferencias de calor reales que se transforman en una pérdida de temperatura o mejor representadas como una velocidad inferior para la misma presión de salida.

$$\eta_{tobera} = \frac{h_5 - h_9}{h_5 - h_{9\,ideal}} = \frac{\Delta \mathrm{h_s}}{\Delta h} \approx \frac{P_{t9} - P_9}{P_{t5} - P_9} \quad (75)$$

Que finalmente se aplica como:

$$\eta_{tobera} \approx \epsilon_n = \frac{P_{t9}}{P_{t5}} \quad (76)$$

1.6.1 **Ecuaciones principales**

La principal ecuación que gobierna una tobera se puede resumir en $\dot{m} = \rho \cdot A \cdot V$ donde el flujo másico es constante. A partir de ello podemos deducir derivando y descartando la energía potencial y un flujo isentrópica sin iteraciones de trabajo puede expresarse como:

$$\frac{d\rho}{\rho} + \frac{dA}{A} + \frac{dV}{V} = 0 \ (77); \qquad \frac{dP}{\rho} + VdV = 0 \quad (78)$$

Combinando ambas expresiones:

$$\frac{dA}{A} = \frac{dP}{\rho}\left(\frac{1}{V} - \frac{d\rho}{dP}\right) = \frac{dP}{\rho V^2}\ (1 - Ma^2) \quad (79)$$

Es decir, cuando el área se reduce el fluido se acelera, si el área aumenta el fluido se decelera, propiedad que es utilizada también por los difusores.

Es conveniente destacar que cuando el fluido alcanza velocidad supersónica invierte su tendencia, por lo tanto para acelerar más allá de la velocidad del sonido es necesario aumentar el área de la sección.

Recordando las propiedades de estancamiento 1.9.1 y aplicándolas a la Garganta de la tobera donde la velocidad es Ma=1 se obtiene:

$$\frac{T^*}{T_{t1}} = \frac{2}{\gamma - 1}; \quad \frac{P^*}{P_{t1}} = \left(\frac{2}{\gamma - 1}\right)^{\frac{\gamma}{\gamma-1}}; \quad \frac{\rho^*}{\rho_{t1}} = \left(\frac{2}{\gamma - 1}\right)^{\frac{1}{\gamma-1}} \quad (80)$$

Que define las propiedades del fluido en la garganta para un gas concreto, y por consiguiente al ser magnitudes de estancamiento son constantes a lo largo de toda la tobera (aplicando si es el caso ciertas eficiencias de la misma). Aplicando la ecuación de los gases ideales:

$$\dot{m} = \rho \cdot A \cdot V = \left(\frac{P}{RT}\right) \cdot A \cdot \left(Ma \cdot \sqrt{\gamma RT}\right) = P \cdot A \cdot Ma \cdot \sqrt{\frac{\gamma}{R \cdot T}} \quad (81)$$

Despejando T de la ecuación estancamiento (65):

$$\dot{m} = \frac{P_{t1} \cdot A \cdot Ma \cdot \sqrt{\frac{\gamma}{R \cdot T_{t1}}}}{\left(1 + \frac{\gamma - 1}{2} Ma_1^2\right)^{\frac{\gamma+1}{2(\gamma-1)}}} \quad (82)$$

Obtenemos el flujo másico en función de las características de estancamiento y la tobera por lo tanto evaluándolo en la garganta:

$$\dot{m} = A^* P_{t1} \sqrt{\frac{\gamma}{R \cdot T_{t1}}} \cdot \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{2\cdot(\gamma-1)}} \quad (83)$$

De donde podemos extraer los criterios para saber si la tobera está adaptada en el caso convergente y las diferentes posibilidades en la tobera convergente-divergente, a través de la combinación de las ecuaciones anteriores obteniendo:

$$\frac{A}{A^*} = \frac{1}{Ma}\left[\left(\frac{2}{\gamma+1}\right)\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{\frac{\gamma+1}{2\cdot(\gamma-1)}} \quad (84)$$

$$Ma^* = Ma\sqrt{\frac{\gamma+1}{2+(\gamma-1)Ma^2}} \quad (85)$$

1.6.2 **Convergente.**

Se basa en acelerar el fluido hasta la velocidad del sonido, para las primeras modelizaciones independientes del área de la garganta únicamente se trata de conocer si la tobera es capaz de acelerar dichos gases hasta velocidad supersónica. Podemos encontrar 3 tipos de casos:

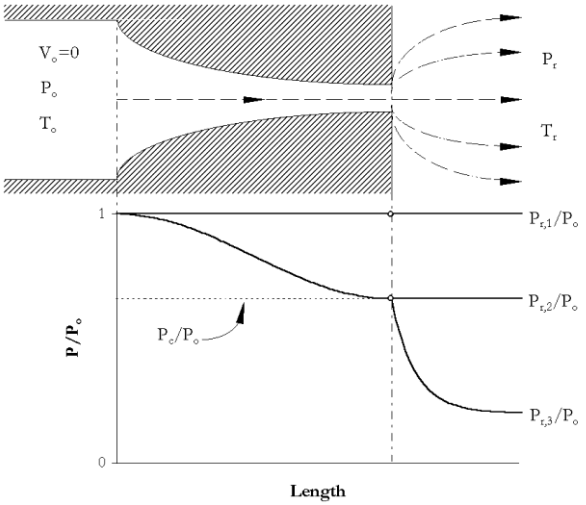


Fig. 44 Tobera Convergente casos [17].

- No hay suficiente presión para alcanzar velocidad supersónica, el fluido se acelera hasta que la presión del fluido es igual que la presión atmosférica pero la velocidad de salida es subsónica (Pr1 figura 44).

- El fluido se acelera hasta que la presión es igual a la presión atmosférica, y se alcanza velocidad supersónica en la salida (Pr2 figura 44).

- El fluido tiene demasiada presión, al tratarse de una tobera convergente no puede superar la velocidad del sonido, por lo tanto lo que ocurre es que la aceleración únicamente ocurres hasta alcanzar la velocidad del sonido, obteniendo una presión de salida mayor a la atmosférica. Como el flujo de masa está limitado solo lo podemos aumentar cambiando las presiones totales de etapas previas a la tobera (Pr3 figura 44).

1.6.3 **Convergente y Divergente.**

Cuando se analiza el caso de una tobera convergente divergente tenemos la existencia de múltiples casos que dependen tanto de las propiedades del fluido como de la geometría de la tobera y la presión externa. Como la geometría de una tobera suele ser fija únicamente tiene un conjunto de propiedades que la optimiza, mientras que en resto de los casos se hace un uso menos eficiente de la misma.

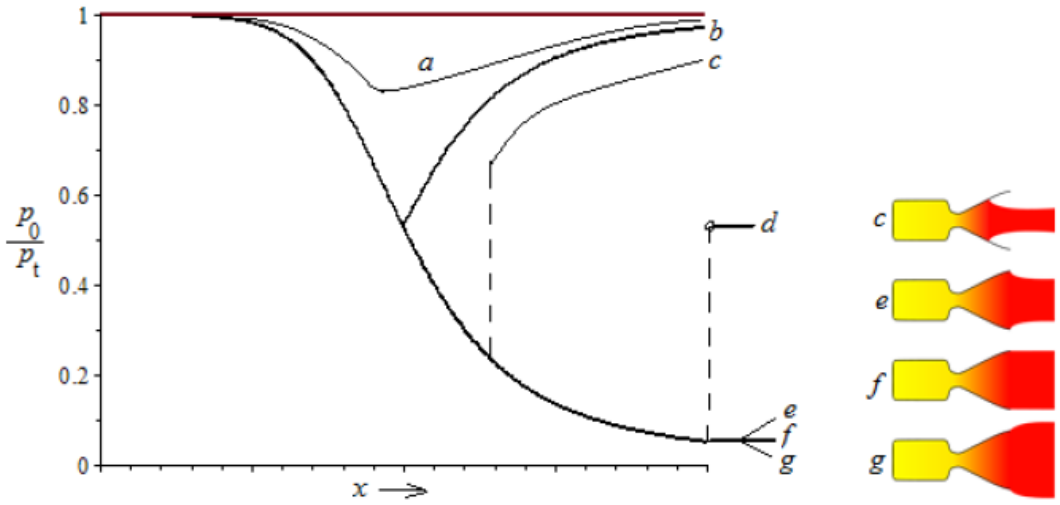


Fig. 45 Tobera Convergente Divergente casos [18].

- Flujo subsónico, si en la garganta no se alcanza la velocidad sónica, implica que la parte divergente realiza una función inversa a la deseada desacelerando el fluido (a) figura 45.

- Flujo supersónico en la garganta pero la presión es igual a la crítica, no superior por lo que en la parte divergente vuelve a desacelerar el fluido y aumentar su presión (b) figura 45.

- Flujo supersónico en la garganta de la tobera y presión superior a la crítica, durante la expansión de la parte divergente, la longitud de la tobera es excesiva, es decir, la presión de salida debe ser inferior a la presión atmosférica. Este efecto termina produciendo ondas de choque en el interior de la parte divergente, que no pueden asumirse como un proceso isentrópico, disminuyen la velocidad subsónica lo cual a la vez desacelera el flujo en el resto del tramo divergente valores entre (b) y (d) como (c) en figura 45.

- Flujo supersónico adaptado, la tobera tiene la longitud pero la presión de salida es sensiblemente diferente obteniendo una onda de choque en la salida (d) figura 45.

- Flujo supersónico con presión de salida igual a atmosférica, tobera adaptada es el óptimo funcionamiento de la tobera (f) en figura 45.

- Flujo supersónico sin embargo la tobera es “corta” por lo que cuando el fluido sale de ella aún tiene una presión superior a la atmosférica (g) en figura 45.

1.7 *Fan*

Aunque las características de un Fan pueden ser diferenciadas o concretadas respecto a un compresor, en esta primera versión del software un fan actúa como un tipo de compresor concreto o como parte de un compresor de dos flujos.

1.8 *Ejes y herramientas asociadas*

No se han modelado ni definido las características térmicas, mecánicas tales como inercia etc… en el modelado del os ejes. Para esta versión del software únicamente sirve como conexión energética entre los componentes.

Si se ha predispuesto de la estructura para su posterior diseño y encaje dentro del software así como las herramientas que dependan de un eje para su funcionamiento como generadores hidráulicos o eléctricos.

1.9 *Otros componentes*

Elementos concretos pertenecientes a Scramjet, Pulsojet, Turboprop, Turboejes o intercambiadores de motor subsónico-supersónico no han sido implementados.

La razón principal es que se sobre entiende que el modelo base es el Turbojet y por facilidad de implementar el Turbofan en el caso más sencillo Turbojet + fan.

Una vez quede estabilizado el desarrollo referente al turbojet, se crearan los módulos pertinentes a partir de los existentes para permitir la simulación del resto de tipos de motores mencionados en la introducción aeronáutica.

2. **Modelo 1**

El modelo 1 asume ciertos aspectos simplificados de los modelos ideales y otras características más realistas como la generación de entropía. Las principales simplificaciones es la utilización de propiedades constantes del fluido, gamma, constante del gas en un flujo 1D. Véase 4.6.1 TFG

2.1 ***Atmosfera Estándar.***

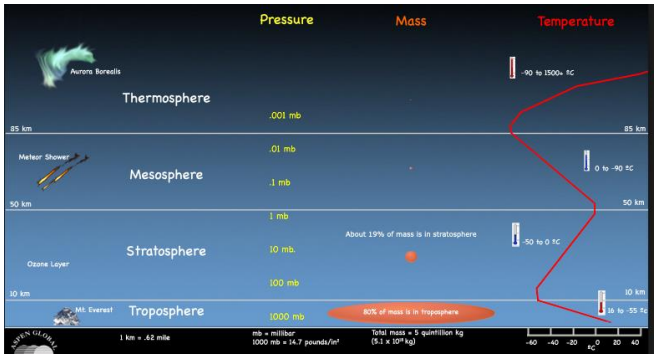
El modelo atmosférico, refleja las propiedades del aire de la atmosfera exterior entorno al motor. Estos valores son importantes para fijar tanto las propiedades del fluido de entrada, como el funcionamiento de la tobera en los gases de salida.

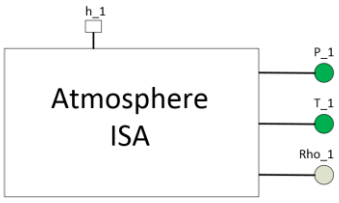
El modelo utilizado es el modelo atmosfera ISA, que se basa en modelizar la atmosfera desde sus capas más bajas hasta la troposfera de manera lineal. Es el principal modelo seguido en aplicaciones de carácter aeronáutico.

Definidos altura, velocidad del avión es capaz de calcular el resto de parámetros necesario.

Se basa en una aproximación para el intervalo 0-11km (troposfera) y 11-35 km mediante una aproximación de un modelo exponencial en presión y lineal en temperatura, impartido en la asignatura de Termodinámica.

Atmosfera ISA v1





Para $h_1 < h_{tropopause}$:

$$T_1 = T_{sealevel} - 0.0065 * h_1$$

$$P_{1\,static} = P_{sealevel} * \left(\frac{T_1}{T_{sealevel}}\right)^{-\frac{g_1}{R*a_i}}$$

Para $h_1 \geq h_{tropopause}$:

$$T_1 = T_{tropopause}$$

$$P_{1\,static} = P_{sealevel} * e^{-\frac{g_1}{R*T_1}(h_1 - h_{tropopause})}$$

$$P_1 = P_{1\,static} + P_{dynamic}$$

$$P_{dynamic} = \rho_1 * \frac{v_1^2}{2}$$

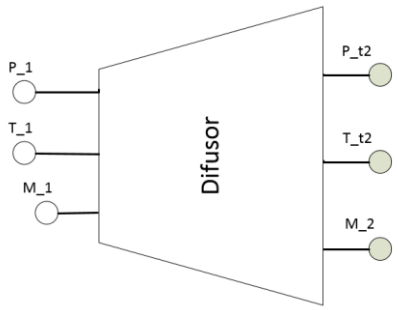
$$\rho_1 = \frac{P_{1\,static}}{R_{air} * T_1}$$

2.2 *Difusor.*

El modelo de difusor implementado en este modelo asume que la función del difusor es ideal y por lo tanto no existen perdidas en forma de entropía, así como una perfecta reducción de la velocidad de los gases a prácticamente cero.

Por otra parte el área del difusor no queda definida por lo que la única funcionalidad del difusor es pasar las propiedades atmosféricas del fluido a variables totales o de estancamiento.

Difusor v1



Variables matemáticas en simulación:

$$P_{t2} = P_1 * \left(1 + \frac{\gamma - 1}{2} Ma_1^2\right)^{\frac{\gamma}{\gamma - 1}}$$

$$T_{t2} = T_1 * (1 + \frac{\gamma - 1}{2} Ma_1^2)$$

$$\dot{m}_2 = \dot{m}_1$$

Variables locales útiles:

$$P_{t2} = P_{t0}$$

$$T_{t2} = T_{t0}$$

$$a_1 = \sqrt{\gamma_1 * R_{air} * T_1}$$

$$Ma_1 = \frac{v_1}{a_1}$$

$$\theta_2 = \frac{T_{t2}}{T_1} = \theta_0$$

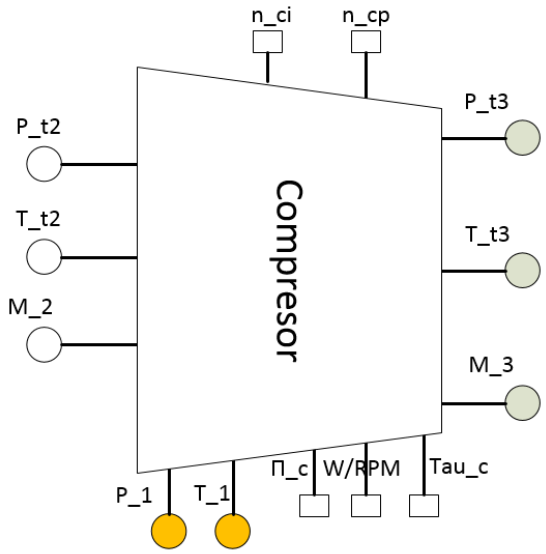
$$\delta_2 = \frac{P_{t2}}{P_1} = \delta_0$$

2.3 ***Compresor.***

Se parte del caso de compresor ideal, pero se incorpora en el la generación de entropía (proceso real) a través de la eficiencia politrópica y la eficiencia isentrópica.

No se definen área ni geometría, no se definen revoluciones del eje para su funcionamiento, estas funcionalidades se modelan únicamente como trabajo consumido y compresión realizada. No queda definida la tecnología usada (axial o centrifuga).

Compresor v1



Variables matemáticas en simulación:

$$P_{t3} = P_{t2} * \pi_c$$

$$T_{t3} = T_{t2} * \tau_c$$

$$\dot{m}_3 = \dot{m}_2$$

$$\pi_c = \tau_c^{\frac{\gamma}{\gamma-1}*\eta_{cp}}$$

$$\pi_c = \left(1 + \eta_{ci} * (\tau_c - 1)\right)^{\frac{\gamma}{\gamma-1}}$$

$$W = \dot{m}_2 * cp * (T_{t2} - T_{t3})$$

Variables locales útiles:

$$\theta_3 = \frac{T_{t3}}{T_1} = \theta_2 * \tau_c$$

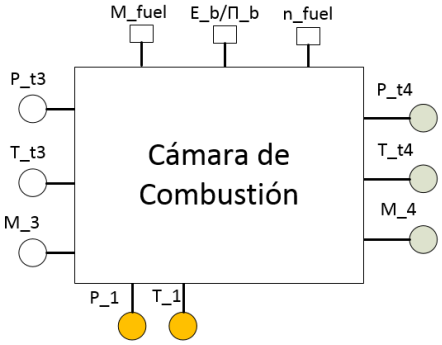
$$\delta_3 = \frac{P_{t3}}{P_1} = \delta_2 * \pi_c$$

2.4 *Cámara Combustión.*

Queda modelada como un conducto cuasi isobárico, donde se realiza una combustión homogénea.

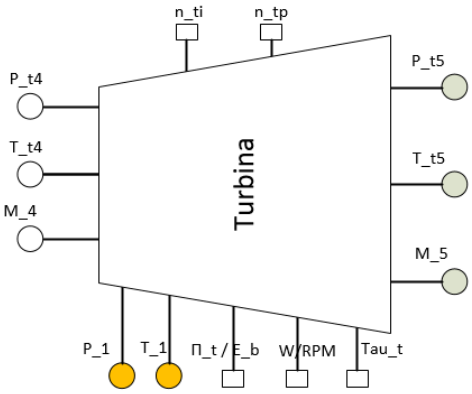
Se asume la difusión del calor de manera homogénea y se desprecia los intercambios estructurales de calor.

Si se modela la masa de fuel añadida al flujo principal. Se desprecia el auto calentamiento de la masa de combustible.

Cámara Combustión v1	
	
Variables matemáticas en simulación: $P_{t4} = P_{t3} * (1 - \varepsilon_b)$ $T_{t4} = \frac{\dot{m}_3}{\dot{m}_4}(T_{t3} + \frac{\alpha * h_f * \eta_{fuel}}{cp})$ $\tau_b = \frac{T_{t4}}{T_{t3}}$ $\dot{m}_4 = \dot{m}_3 + \dot{m}_{fuel}$	*Variables locales útiles:* $\alpha = \frac{\dot{m}_{fuel}}{\dot{m}_3}$ $\theta_4 = \frac{T_{t4}}{T_1} = \theta_3 * \tau_b$ $\delta_4 = \frac{P_{t4}}{P_1} = \delta_3 * \pi_b$ $\dot{Q} = \dot{m}_{fuel} * h_f * \eta_{fuel}$

2.5 ***Turbina.***

Se utiliza un modelo idéntico al compresor pero aplicado a una expansión.
No se tiene en cuenta problemas térmicos de la estructura ni refrigeración de la misma.

Turbina v1	
	
Variables matemáticas en simulación: $P_{t5} = P_{t4} * \pi_t$ $T_{t5} = T_{t4} * \tau_t$ $\dot{m}_5 = \dot{m}_4$ $\pi_t = \tau_t^{\frac{\gamma}{\gamma-1}*\frac{1}{\eta_{tb}}}$ $\pi_t = \left(1 + \frac{1}{\eta_{tbi}} * (\tau_t - 1)\right)^{\frac{\gamma}{\gamma-1}}$ $W = \dot{m}_4 * cp * (T_{t5} - T_{t4})$	*Variables locales útiles:* $\theta_5 = \frac{T_{t5}}{T_1} = \theta_4 * \tau_t$ $\delta_5 = \frac{P_{t5}}{P_1} = \delta_4 * \pi_t$

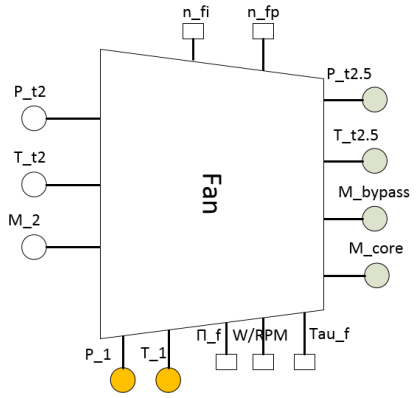
2.6 *Poscombustor.*

Mantiene el mismo modelo que la cámara de combustión. Únicamente se modela como una cámara de combustión cuya eficiencia de combustión es menor, y la propiedad del fluido tanto en entrada como en salida es la gamma fuel.

2.7 *Fan.*

Mantiene el mismo modelo que el compresor pero para un flujo fuera del core.

Fan v1



Variables matemáticas en simulación:

$$P_{t2.5} = P_{t2} * \pi_{fan}$$

$$T_{t2.5} = T_{t2} * \tau_{fan}$$

$$\dot{m}_{core} + \dot{m}_{bypass} = \dot{m}_2$$

$$\pi_{fan} = \tau_{fan}^{\frac{\gamma}{\gamma-1}*\eta_{fanp}}$$

$$\pi_{fan} = \left(1 + \eta_{fani} * (\tau_c - 1)\right)^{\frac{\gamma}{\gamma-1}}$$

$$W = \dot{m}_2 * cp * (T_{t2} - T_{t2.5})$$

Variables locales útiles:

$$\theta_{2.5} = \frac{T_{t2.5}}{T_1} = \theta_2 * \tau_{fan}$$

$$\delta_{2.5} = \frac{P_{t2.5}}{P_1} = \delta_2 * \pi_{fan}$$

O mediante una única implementación bajo el nombre Fan-Compresor que contiene a ambos elementos en un único bloque reduciendo el número de ecuaciones pero ligando el trabajo de ambos como:

$$W = \dot{m}_2 * cp * (T_{t2} - T_{t2.5}) + \dot{m}_{core} * cp * (T_{t2} - T_{t3})$$

2.8 *Tobera.*

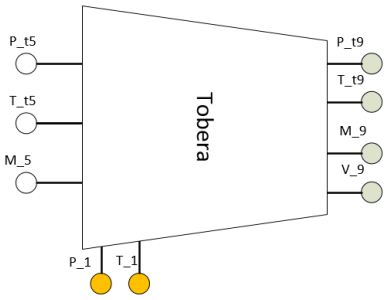
Se analizan dos casos por separados la tobera convergente o la tobera convergente – divergente. En todos los casos se parte de un modelo sencillo y bastante ideal que consigue resultados similares a la realidad sin gran complejidad.

2.8.1 **Convergente.**

Se analiza el ratio de presión entre la salida de la turbina o poscombustor, en referencia a la presión atmosférica.
Si el ratio permite una tobera adaptada expande idealmente los gases sin generación de entropía. Si el ratio no permite tener un tobera adaptada expande los grases hasta el mejor caso quedando una presión extra propulsora que no se convierte en velocidad.

Se asume que los gases en la entrada tienen velocidad cercana a cero al inicio de la tobera.

Tobera v1



Variables matemáticas en simulación:

$$P_9 = P_{t5} * \pi_{nCRR} \quad \text{si} \quad \pi_{nCRR} > \pi_{nR}$$

$$P_9 = P_{t5} * \pi_{nR} = P_1 \quad \text{si} \quad \pi_{nCRR} < \pi_{nR}$$

$$T_9 = T_5 * \left(\frac{P_9}{P_5}\right)^{\frac{\gamma-1}{\gamma}}$$

$$\dot{m}_9 = \dot{m}_5$$

$$V_9 = \frac{2}{\dot{m}_5} * cp * (T_{t5} - T_9)$$

Variables locales útiles:

$$\pi_{nR} = \frac{P_1}{P_{t5}}$$

$$\pi_{nCRR} = \frac{P^*}{P_{t5}} = \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}$$

$$\theta_9 = \frac{T_{t9}}{T_1} = \theta_5 * \tau_n$$

$$\delta_9 = \frac{P_{t9}}{P_1}$$

2.8.2 **Convergente y Divergente.**

Calcula si es posible realizar la adaptación de tobera, si no lo es conmuta a sistema tobera convergente.

Si es posible alcanzar Mach 1 en la garganta, asume una geometría flexible para adaptar la tobera a su geometría óptima.

Tobera Convergente-Divergente v1	
Variables matemáticas en simulación: Si $\pi_{nR} > \pi_{nCRR}$, es decir se dispone de una presión capaz de superar la atmosférica a Ma=1 en la garganta. Realizamos una expansión hasta alcanzar la presión atmosférica: $$P_9 = P_{atm}$$ $$\frac{P_1}{P_{t5}} = \pi_{nR} = \left(1 + \frac{\gamma - 1}{2} Ma_9\right)^{\frac{\gamma}{\gamma-1}}$$ $$T_9 = T_{t5} * \left(1 + \frac{\gamma - 1}{2} Ma_9\right)^{-1}$$ $$\dot{m}_9 = \dot{m}_5$$ $$V_9 = Ma_9 * \sqrt{\gamma R * T_9}$$	*Variables locales útiles:* $$\pi_{nR} = \frac{P_1}{P_{t5}}$$ $$\pi_{nCRR} = \frac{P^*}{P_{t5}} = \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma-1}}$$ $$\theta_9 = \frac{T_{t9}}{T_1} = \theta_5 * \tau_n$$ $$\delta_9 = \frac{P_{t9}}{P_1} = \delta_5$$

2.8.1 **Ejes.**

Durante la realización del modelo 1, no se contempla funcionalidad extra al eje que la interconexión de los diferentes elementos que extraen o aportan energía al fluido en forma de trabajo. Por lo tanto la única ecuación del mismo es:

$$W_{in} - W_{out} = 0 = W_{fan} + \sum W_{compresor} + \sum W_{turbina} + \sum W_{gearbox}$$

3. **Modelo 2**

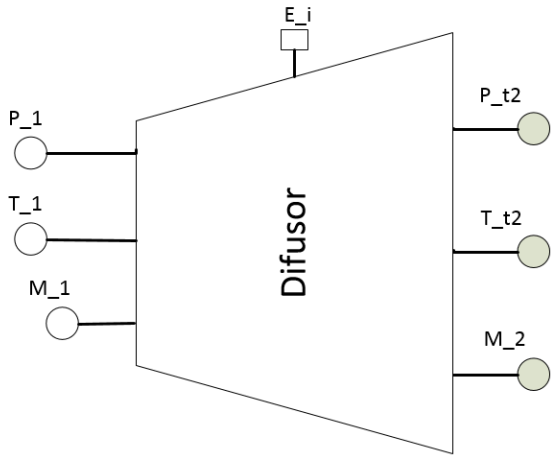
En el modelo 2 se han introducido cambios más precisos sobre el modelo 1, la principal característica es la variación de las propiedades del fluido con la temperatura. Por otra parte sean introducidas ciertas eficiencias no contempladas en el modelo 1 para dar realismo a aquellos componentes más idealizados. En el apartado 4.6.2 del TFG se especifican los principales cambios.

Recordemos que las funciones $\Phi(\mathrm{T},\alpha)\ \mathrm{y}\ \mathrm{h}(\mathrm{T},\alpha)$ dependen solo de la temperatura y propiedades del fluido (mezcla aire + fuel).

3.1 ***Difusor.***

Se introduce un término de eficiencia del difusor.

Difusor v2



Variables matemáticas en simulación:

$$P_{t2} = P_1 * (1 - \varepsilon_i) * \left(1 + \frac{\gamma - 1}{2} Ma_1^2\right)^{\frac{\gamma}{\gamma - 1}}$$

$$T_{t2} = T_1 * (1 + \frac{\gamma - 1}{2} Ma_1^2)$$

$$\dot{m}_2 = \dot{m}_1$$

Variables locales útiles:

$$P_{t2} = P_{t0} * (1 - \varepsilon_i)$$

$$T_{t2} = T_{t0}$$

$$a_1 = \sqrt{\gamma_1 * R_{air} * T_1}$$

$$Ma_1 = \frac{v_1}{a_1}$$

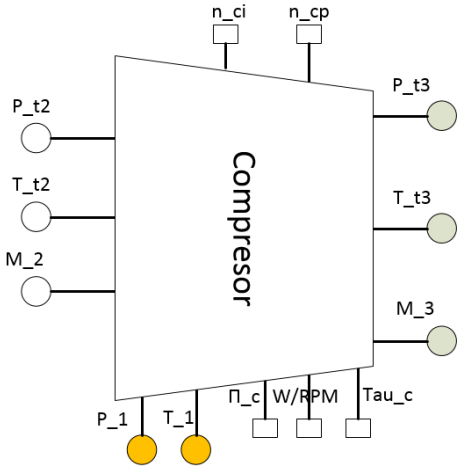
$$\theta_2 = \frac{T_{t2}}{T_1} = \theta_0$$

$$\delta_2 = \frac{P_{t2}}{P_1} = \delta_0 * (1 - \varepsilon_i)$$

3.2 ***Compresor.***

Se introducen los términos a partir de las funciones de Φ y h que son calculadas a partir de la temperatura y características del fluido.

Compresor v2



Variables matemáticas en simulación:

$$P_{t3} = P_{t2} * \pi_c$$

$$T_{t3} = T_{t2} * \tau_c$$

$$\dot{m}_3 = \dot{m}_2$$

$$\Phi_{t3} = \Phi_{t2} + \frac{1}{\eta_{cp}} \log \frac{P_{t3}}{P_{t2}}$$

$$\mathrm{h}_{t3} = \mathrm{h}_{t2} + \frac{1}{\eta_{ci}} (h_{t3i} - h_{t2})$$

$$W = \dot{m}_2 * (h_{t2} - h_{t3})$$

Variables locales útiles:

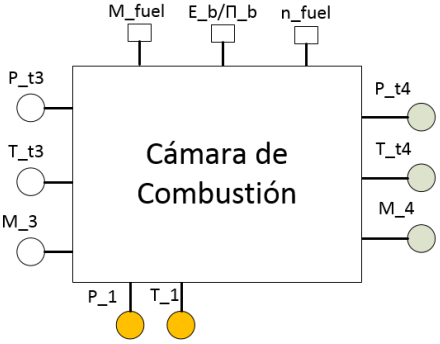
$$\theta_3 = \frac{T_{t3}}{T_1} = \theta_2 * \tau_c$$

$$\delta_3 = \frac{P_{t3}}{P_1} = \delta_2 * \pi_c$$

$$\Phi_{t3i} = \Phi_{t2} + \log \frac{P_{t3}}{P_{t2}}$$

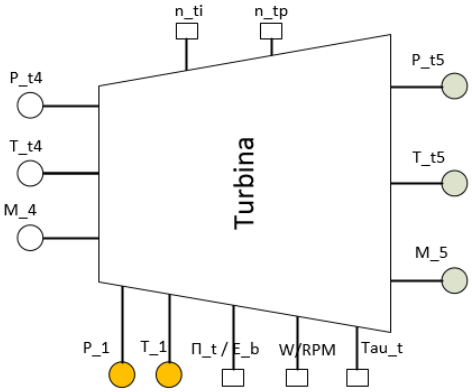
$$\alpha = 0$$

3.3 *Cámara Combustión.*

Cámara Combustión v1	
	
Variables matemáticas en simulación: $P_{t4} = P_{t3} * (1 - \varepsilon_b)$ $h_{t4} = \frac{h_{t3} + \alpha * h_f * \eta_{fuel}}{(1+\alpha)}$ $\tau_b = \frac{T_{t4}}{T_{t3}}$ $\dot{m}_4 = \dot{m}_3 + \dot{m}_{fuel}$	*Variables locales útiles:* $\alpha = \frac{\dot{m}_{fuel}}{\dot{m}_3}$ $\theta_4 = \frac{T_{t4}}{T_1} = \theta_3 * \tau_b$ $\delta_4 = \frac{P_{t4}}{P_1} = \delta_3 * \pi_b$ $\dot{Q} = \dot{m}_{fuel} * h_f * \eta_{fuel}$

3.4 ***Turbina.***

Turbina v2



Variables matemáticas en simulación:

$$P_{t5} = P_{t4} * \pi_t$$

$$T_{t5} = T_{t4} * \tau_t$$

$$\dot{m}_5 = \dot{m}_4$$

$$\Phi_{t5} = \Phi_{t4} + \eta_{tp} \log \frac{P_{t5}}{P_{t4}}$$

$$\mathrm{h}_{t5} = \mathrm{h}_{t4} + \eta_{ti}(h_{t5i} - h_{t4})$$

$$W = \dot{m}_4 * (h_{t4} - h_{t5})$$

Variables locales útiles:

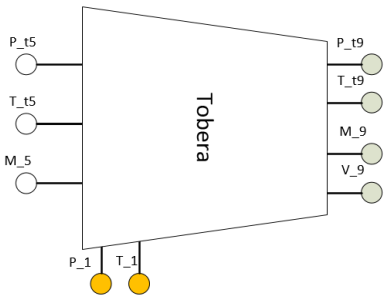
$$\theta_5 = \frac{T_{t5}}{T_1} = \theta_4 * \tau_t$$

$$\delta_5 = \frac{P_{t5}}{P_1} = \delta_4 * \pi_t$$

$$\alpha = \frac{\dot{m}_{fuel}}{\dot{m}_3}$$

3.5 *Tobera.*

3.5.1 **Convergente.**

Tobera v2	
	
Variables matemáticas en simulación: si $\pi_{nCRR} > \pi_{nR}$ $P_9 = P_{t5} * (1 - \varepsilon_n) * \pi_{nCRR}$ si $\pi_{nCRR} < \pi_{nR}$ $P_9 = P_{t5} * \pi_{nR} = P_1$ $T_9 = T_{t5} * \left(1 + \frac{\gamma - 1}{2}\left(\frac{V_9}{\sqrt{\gamma R T_9}}\right)^2\right)^{-1}$ $\dot{m}_9 = \dot{m}_5$ $V_9 = \frac{2}{\dot{m}_5} * (h_{t5} - h_9)$	*Variables locales útiles:* $\pi_{nR} = \frac{P_1}{P_{t5}(1 - \varepsilon_n)}$ $\pi_{nCRR} = \frac{P^*}{P_{t5}} = \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma - 1}}$ $\theta_9 = \frac{T_{t9}}{T_1} = \theta_5 * \tau_n$ $\delta_9 = \frac{P_{t9}}{P_1}$

3.5.2 **Convergente y Divergente.**

Tobera Convergente-Divergente v1	
Variables matemáticas en simulación: Si $\pi_{nR} > \pi_{nCRR}$, es decir se dispone de una presión capaz de superar la atmosférica a Ma=1 en la garganta. Realizamos una expansión hasta alcanzar la presión atmosférica: $$P_9 = P_{atm}$$ $$\pi_{nR} = \left(1 + \frac{\gamma - 1}{2} Ma_9\right)^{\frac{\gamma}{\gamma-1}}$$ $$T_9 = T_{t5} * \left(1 + \frac{\gamma - 1}{2} Ma_9\right)^{-1}$$ $$T_9 = T_1 * \pi_{nR}^{\frac{\gamma-1}{\gamma}}$$ $$\dot{m}_9 = \dot{m}_5$$ $$V_9 = Ma_9 * \sqrt{\gamma R * T_9}$$	*Variables locales útiles:* $$\pi_{nR} = \frac{P_1}{P_{t5}(1 - \varepsilon_n)}$$ $$\pi_{nCRR} = \frac{P^*}{P_{t5}} = \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma-1}}$$ $$\theta_9 = \frac{T_{t9}}{T_1} = \theta_5 * \tau_n$$ $$\delta_9 = \frac{P_{t9}}{P_1} = \delta_5$$

4. **Modelo 3**

El modelo 3 está definido teóricamente ya que únicamente define las áreas sección de cada uno de los elementos y el flujo másico reducido, es decir nos añade dos nuevos tipos de variable y una ecuación para relacionarlas con el resto. Sin embargo no se ha implementado en el código por que la versión previa del modelo 2 se detectó un error en la ecuación de la entalpia del fluido cuando estaba siendo optimizada.

Esto error, nos obliga a recalcular todos los test unitarios del modelo 2, lo que implica resolver manualmente todos los test, es decir, 1-2 semanas de trabajo, por lo tanto se ha propuesto su implementación después de que se solucione las modificaciones estructurales del core en la versión Beta, enunciadas en los resultados. Ya que una vez se reestructure el core con las mejoras en la versión Beta, también habrá que recalcular los test unitarios del modelo 1 y 2, debido a que una de las modificaciones afecta a la evaluación de los residuos.

Como conclusión no se ha recalculado un trabajo, que se volverá a recalcular de nuevo en la versión Beta y que condiciona 2 semanas de trabajo. Por otra parte no se puede avanzar en la implementación del modelo 3 sin tener testeado el 2, ya que sería costoso o imposible trazar los posibles errores en el código, al utilizar reiterativamente el concepto de herencia.

La principal diferencia del modelo 3 respecto al modelo 2 es la introducción del flujo másico reducido.

$$\mathcal{M} = \frac{\dot{m}\sqrt{T_t}}{P_t A} = \sqrt{\frac{\gamma}{r}} M^+ \cdot \left(1 + \frac{\gamma - 1}{2} M^{+2}\right)^{\frac{1}{\gamma - 1}}$$

$$M^+ = \frac{V}{\sqrt{\gamma R T_t}} = \frac{M}{\sqrt{1 + \frac{\gamma - 1}{2} M^2}}$$

El principio de conservación de masa, nos indica que para un volumen definido el flujo de masa a través de él es constante. De una manera similar si definimos el flujo de masa reducido como aquel flujo de masa adimensionalizadas por sección de área, y por la temperatura y presión total del fluido. Obtenemos una función del mismo que únicamente varía en función del Mac en esa sección del motor.

Por lo tanto este nuevo parámetro permite definir la sección área de los componentes y la velocidad mac relacionada con la misma. Que junto a las propiedades totales de la sección permiten conocer la velocidad real del fluido en dicha sección. Es decir, proporcionamos dimensión geométrica al motor y permite escalarlo.

1. Anexo C: Software

Este Anexo pretende ser un resumen más concreto de las ideas explicadas a lo largo de TFG. Una vez comprendidas las ideas básicas de la estructura del software expuestas en el capítulo 4, se describirán algunos de los objetos o variables generadas en las clases, así como las funciones o funcionalidades principales más complejas de entender.

1.1 *Objetos de Transferencia*

Java a diferencia de otros lenguajes únicamente permite el traspaso de parámetros como entidades, es decir, no es posible definir un parámetro de una función como un vector de diferentes variables. La norma suele ser definir listas o matrices en las que se insertan las variables, donde estas listas son una entidad.

Otra característica de Java es la existencia de interfaces, que permiten aglutinar características comunes a varios objetos sin la necesidad de herencia. Esta propiedad no solo permite trabajar con estos objetos con una estructura común, sino que además también permite procesarlos como si perteneciesen a una misma entidad. Esta característica no ha sido usada con el fin de lograr una compatibilidad completa de la traducción de partes del software para una posterior mejora del mismo computacionalmente hablando, bajo el uso de C++.

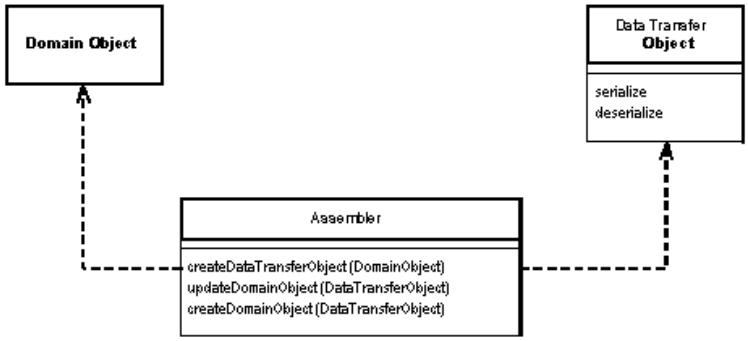


Fig. 46 Concepto objeto de transferencia o Assember

La necesidad de transferir grandes cantidades de variables y estructuradas, ha sido solucionada utilizando objetos de transferencia. Un objeto de transferencia es un objeto que aglutina variables (fig. 46), métodos de una manera estandarizada. De esta manera si creamos el objeto propiedades de un fluido contendrá todos los parámetros del mismo, las funciones para calcular ciertas propiedades en función de las anteriores o de input, permitiendo su uso mediante carga/descarga de cada uno de los componentes involucrados con fluidos.

1.2 *MatrixCollection*

Las variables más usadas son los vectores y matrices, X, Fx, JFx, que son necesarias en cada uno de los componentes del motor, deben ser transferidos a la simulación y al core matemático para realizar sus cálculos en cada iteración.

La solución adoptada (Fig.47), es el uso de MatrixCollection como objeto transferencia de las variables encargadas en los cálculos, por ello es un elemento común a una gran parte de objetos diferenciados SimulationBlock, SimulationProyect y MathCore y clases en herencia como Engine, Compressor etc...

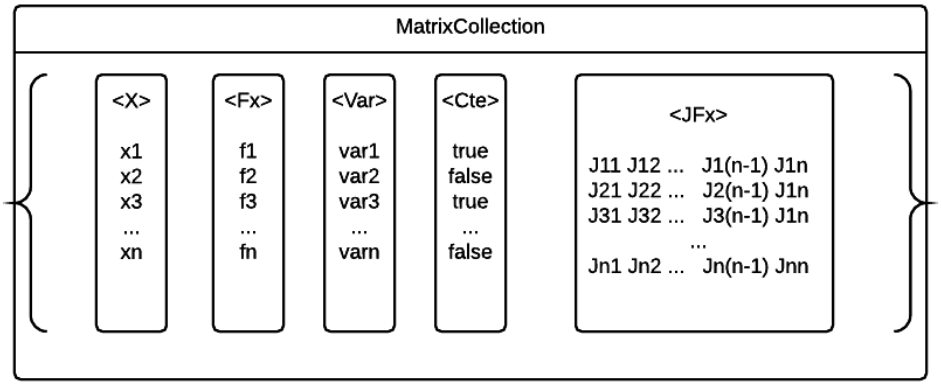


Fig. 47 MatrixCollection estructura.

En la Fig.48 podemos observar que tanto el Simulator Project como cada uno de los SimulationBlock utilizan la clase MatrixCollection como objeto de transferencia.

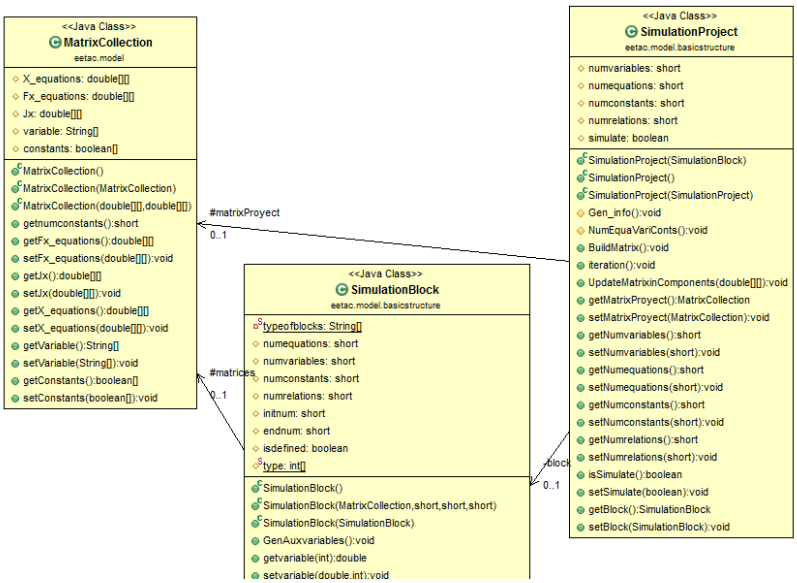


Fig. 48 diagrama de clases involucradas en MatrixCollection

Encargado de almacenar los vectores y matrices también indica si pertenecen a una constante definida por el usuario o no y el nombre de la misma.

Esto a su vez facilita que la interfaz gráfica Swing también pueda contener una MatrixCollection y crear los labels en función del vector de variables, los textbox con los valores de las variables y los checkbox y estados de los textbox (enable/disenable) según sean los datos proporcionados (constantes) o semillas.

1.3 *Traducción de matrices*

La característica actual basada en métodos newton-raphson hace necesario tener, un vector con las variables, los residuos de las funciones y las derivadas parciales de las ecuaciones, con el fin de calcular la nueva X', que iteración a iteración converge en la solución.

Para realizar este proceso es necesario previamente formar la MatrixCollection del conjunto. La traducción desde el componente al conjunto se basa en las MatrixCollection de los componentes y el orden de los mismos.

1.3.1 **Vector X, Fx y JFx en el conjunto**

Las variables tienen un índice local, dentro de cada componente, este índice suele estar relacionado con un orden coherente y común para los objetos que pertenecen a ramas concretas (Flowblock , Ejes, WorkFlowblock…).

La matriz de variables del conjunto es la concatenación de los vectores de cada uno de los elementos. Por lo tanto hay que establecer un orden o indexado general basado en la posición de los componentes.

Cada componente tiene un idnumber que indica el tipo de elemento, permite organizarlos por prioridad de idnumber, basándose en la función que tiene en el motor. En el caso de que existan múltiples componentes de un mismo tipo (dos compresores o dos turbinas) hay un parámetro idblock, que es asignado según el orden de colocación en el engine, permitiendo ordenar elementos de un mismo tipo.

El engine o Simulationproject es capaz de traducir componentes al conjunto (concatenando en orden) y del conjunto a componentes utilizando la posición y el indexado de la simulación véase figura 49.

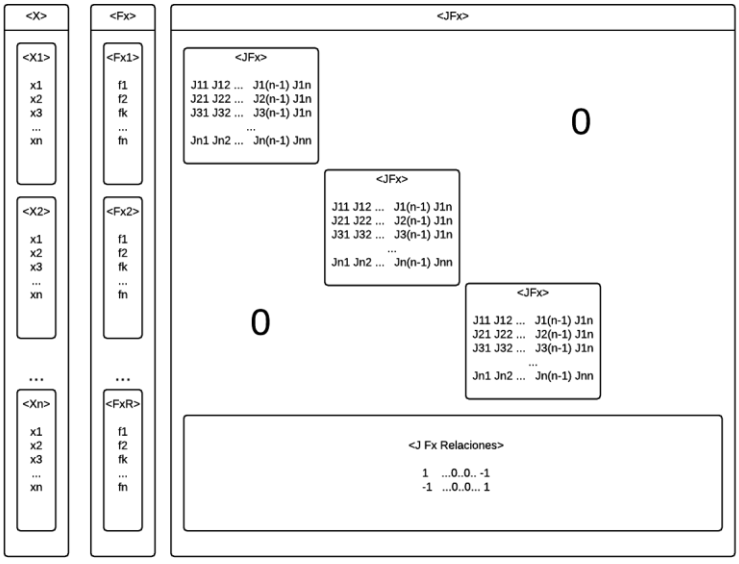


Fig. 49 Esquema de matriz del conjunto en base a los Componentes.

Para el caso de Fx es exactamente igual con una diferencia significativa, normalmente el número de funciones es inferior al número de variables, por ello el usuario designa constantes. Estas constantes son incluidas debajo de las funciones en la forma $F_i = Variableconstante - valorintroducido$ dentro de cada componente.

Por otra parte el número total de funciones seguirá siendo menor al de variables ya que una vez concatenadas las funciones más las constantes, normalmente hay que introducir las relaciones de variables entre bloques.

Estas relaciones son a escala del conjunto por lo que una vez concatenadas todas las funciones de los elementos se añaden las funciones artificiales que definen la relación entre variables de dos componentes (Fxr en la figura 49).

Cuando se trata de la matriz del jacobiano es un poco más complejo debido a la no homogeneidad del vector Fx. La matriz jacobiana proporcionada por el componente tiene una dimensión N variables por M funciones, donde las funciones son las del modelo utilizado más las constantes definidas por el usuario para el componente. Como cada componente tiene un número de variables, ecuaciones y constantes diferente, las matrices Jacobianas raramente serán iguales, ni cuadradas. Por lo tanto hay que colocarlas correctamente utilizando el indexado del conjunto, para hacer coincidir las variables concatenadas con las variables del jacobiano del elemento.

Una vez colocada en su lugar cada una de las matrices necesarias. Se han de añadir las derivadas parciales de las funciones de relaciones que se agregó al vector Fx como Fxr lo cual completara la matriz de jacobiano haciéndola cuadra.

Deben de tener el mismo indice para referirse a la misma función y variable. La derivada parcial de una función que es una relación de variables indicada como $F_r = Variable_a - Variable_b$ vale 1 cuando derivamos respecto a la primera y -1 respecto a la segunda. Por lo tanto por cada relación añadida al generar un residuo extra notado como Fxr, se genera una línea dentro del jacobiano del conjunto con valor 1 para la primera variable y -1 para la segunda notado en la ilustración como ‘JFx Relaciones’.

1.3.2 **Math Core**

El Math Core es un objeto capaz de realizar aproximaciones lineales hacia la solución final. Su objetivo es servir como base a herencia para la implementación de algoritmos más optimizados o rápidos sobrescribiendo sus métodos.

Sus principales características son un conjunto de matrices similar a la MatrixCollection donde se contienen las variables de la iteración anterior y los resultados de la nueva iteración.

Los parámetros de la iteración como número máximo de iteraciones, intentos realizados, tiempo consumido o tolerancia del residuo. Y las funciones

principales que serán sobrescritas por cada una de las implementaciones que hereden:

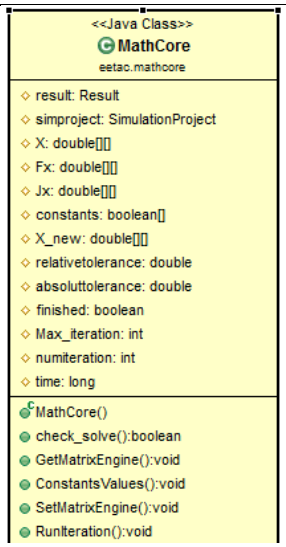


Fig. 50 Math Core Class

- Check_solve: comprueba si la solución obtenida en la iteración cumple con los requisitos de tolerancia.
- GetMatrixEngine: Se encarga de obtener las matrices necesarias para el Core matemático
- SetMatrixengine: Actualiza los valores del proyecto para recalcular una nueva iteración.
- RunIteration: Es la función principal, es donde se realiza la iteración matemática. Calcula las matrices necesarias y cálculos entre las mismas llamando a la librería JBLAS. Llama a la función Check_solve para comprobar

Para permitir la implantación de otros métodos matemáticos o derivados del mismo, se ha implementado un objeto genérico MathCore, que contiene las matrices básicas del modelo sobre el engine. A partir de estas matrices, se generan las matrices auxiliares pertinentes en cada método, obteniendo un vector de variables (soluciones a la iteración), que vuelve a ser iterado hasta llegar al máximo de iteraciones definido o a una solución tolerable.

En esta versión del software únicamente se ha implementado el método Newton-Raphson con diferentes variantes que reducen el consumo computacional del mismo. El core matemático está predispuesto a la generación de nuevos métodos de resolución incluso a la combinación de ambos para aquellos casos donde la semillas inicial no permita una convergencia asegurada como es el caso de métodos similares a Newton, para ello únicamente hay que heredar y sobrescribir los métodos principales añadiendo nuevas funcionalidades.

Recordando el punto 5.4.1 del TFG donde alude a la eliminación de la traducción entre bloque y conjunto y la generación de las matrices por parte del core matemático, es decir, eliminara la matriz jacobiana como requisito del math core y convertirá el MatrixCollection en un objeto de transferencia de índices y funciones con el nombre las variables y su estado.

1.4 *Método de Test*

Se han implementado test unitarios a nivel de componente y función que permiten asegurar el correcto funcionamiento del software en cada compilación realizada, para asegurar la no existencia de bugs durante el desarrollo o en modificaciones posteriores.

Los test implementados se basan en dos principales métodos de prueba:

- Prueba funcional, se testea unitariamente que cada una de las funciones del objeto si funciona correctamente, independientemente del carácter matemático o enunciativo dentro de la simulación.
- Prueba carga, se comprueban los valores iniciales de las variables y su carga correcta.
- Prueba matricial, se comprueban la generación de vectores y matrices para la primera iteración.
- Prueba engine, se integra el componente dentro de un SimulationProject o derivados para comprobar el correcto traspaso de matrices y funcionalidades dentro de la simulación.
- Prueba de MathCore, comprueba número de ecuaciones incógnitas etc…
- Prueba iterativa, se hace iterar hasta converger en la solución, se contrasta con la solución manual, insertada en el código.

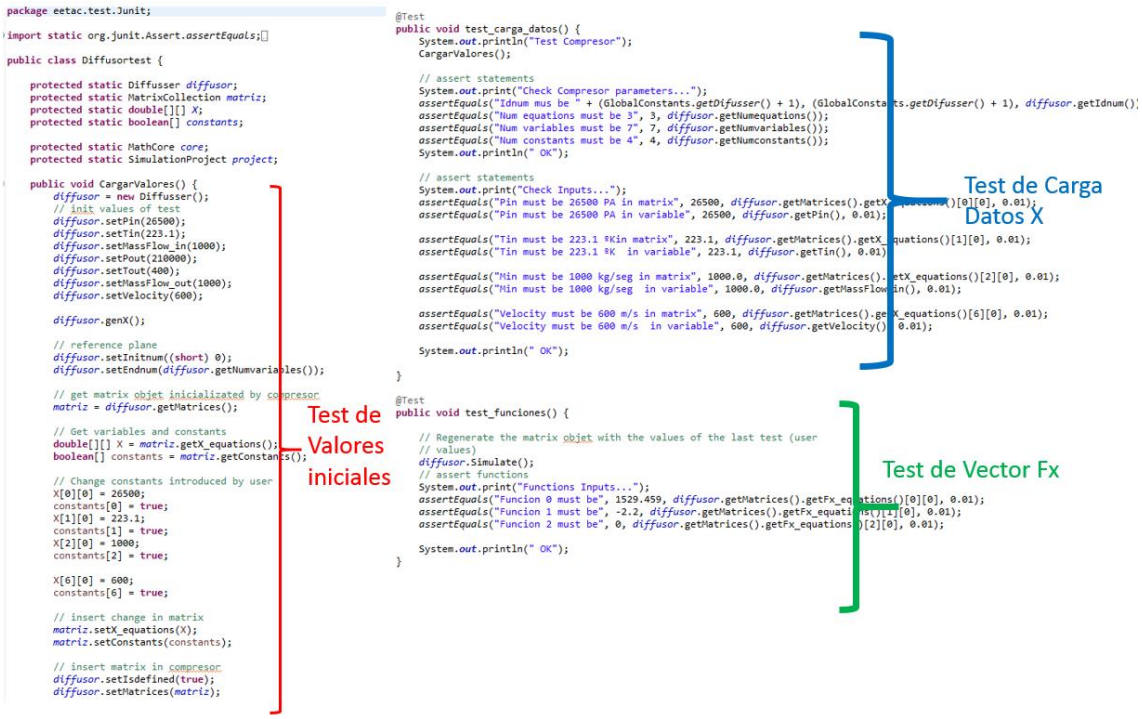


Fig. 51 Partes de código del test de Difusor modelo 1.

Por lo tanto para cada uno de los elementos descritos en este documento se ha realizado la siguiente dinámica véase figuras 51 y 52:

1. Búsqueda de un problema con valores realistas adecuado para el modelo usado.
2. Resolución manual del problema con el modelo propuesto.
3. Elección de variables constantes y no constantes.
4. Generación mediante Excel de X, Fx, JFx para la primera iteración.
5. Creación del test en el código, comprobar carga de valores y evaluaciones.
6. Comprobar vectores y matrices del objeto en la primera iteración.

7. Simular componentes hasta hallar solución, comparar solución con resolución manual.

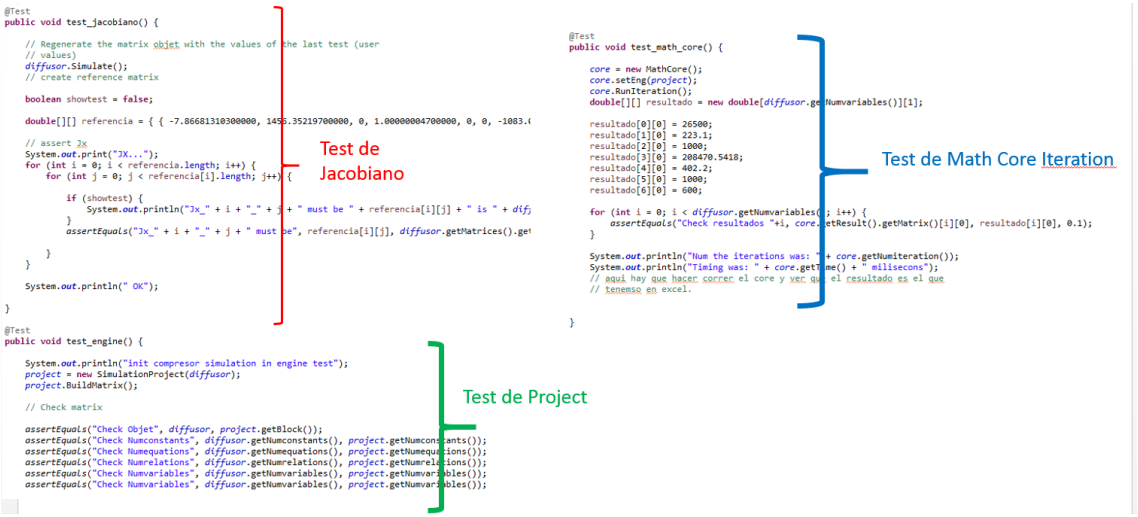
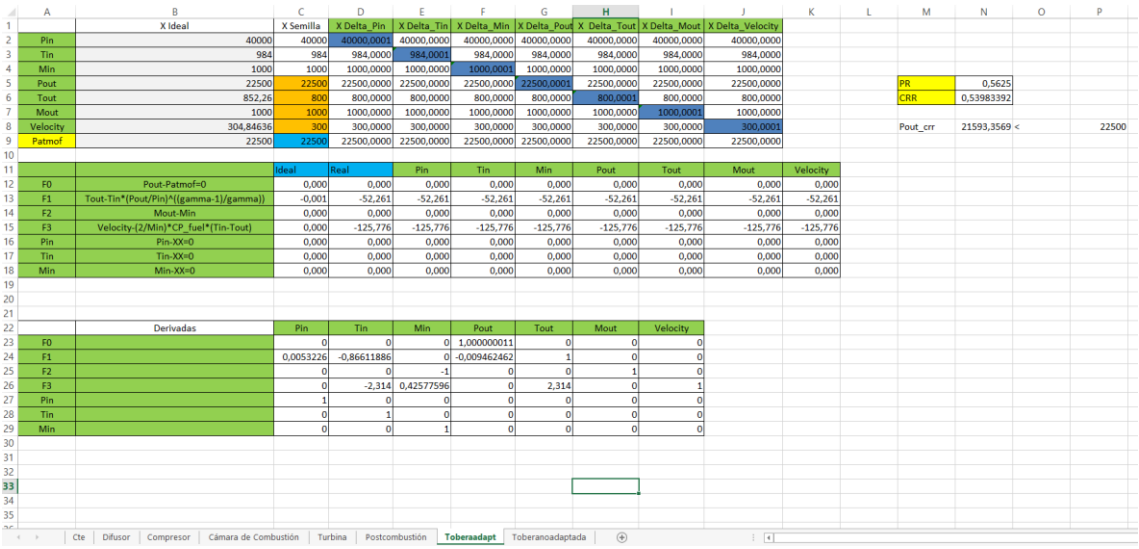


Fig. 52 Test código Difusor modelo 1.

En el caso del modelo 1 se ha evaluado en cada componente con un problema diferente, además con el mismo problema se ha simulado utilizando diferentes variables constates y otras como incógnitas obteniendo el mismo resultado.



	X Ideal	X Semilla	X Delta_Pin	X Delta_Tin	X Delta_Min	X Delta_Pout	X Delta_Tout	X Delta_Mout	X Delta_Velocity
Pin	40000	40000	40000,0001	40000,0000	40000,0000	40000,0000	40000,0000	40000,0000	40000,0000
Tin	984	984	984,0000	984,0001	984,0000	984,0000	984,0000	984,0000	984,0000
Min	1000	1000	1000,0000	1000,0000	1000,0001	1000,0000	1000,0000	1000,0000	1000,0000
Pout	22500	22500	22500,0000	22500,0000	22500,0000	22500,0001	22500,0000	22500,0000	22500,0000
Tout	852,26	800	800,0000	800,0000	800,0000	800,0000	800,0001	800,0000	800,0000
Mout	1000	1000	1000,0000	1000,0000	1000,0000	1000,0000	1000,0000	1000,0001	1000,0000
Velocity	304,84636	300	300,0000	300,0000	300,0000	300,0000	300,0000	300,0000	300,0001
Patmof	22500	22500	22500,0000	22500,0000	22500,0000	22500,0000	22500,0000	22500,0000	22500,0000

		Ideal	Real	Pin	Tin	Min	Pout	Tout	Mout	Velocity
F0	Pout-Patmof=0	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
F1	Tout-Tin*(Pout/Pin)^((gamma-1)/gamma))	-0,001	-52,261	-52,261	-52,261	-52,261	-52,261	-52,261	-52,261	-52,261
F2	Mout-Min	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
F3	Velocity-(2/Min)*CP_fuel*(Tin-Tout)	0,000	-125,776	-125,776	-125,776	-125,776	-125,776	-125,776	-125,776	-125,776
Pin	Pin-XX=0	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Tin	Tin-XX=0	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Min	Min-XX=0	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Derivadas	Pin	Tin	Min	Pout	Tout	Mout	Velocity
F0	0	0	0	1,000000011	0	0	0
F1	0,0053226	-0,86611886	0	-0,009462462	1	0	0
F2	0	0	-1	0	0	1	0
F3	0	-2,314	0,42577596	0	2,314	0	1
Pin	1	0	0	0	0	0	0
Tin	0	1	0	0	0	0	0
Min	0	0	1	0	0	0	0

PR	0,5625		
CRR	0,53983392		
Pout_crr	21593,3569	<	22500

Fig. 53 Excel de Problemas para generar X, Fx, JFX

En el caso del modelo 2 únicamente se tiene testeado parcialmente todos los módulos ya que ciertos elementos no tienen los 7 pasos realizados íntegramente. Esto es debido a la detección tardía de un error en la ecuación de cálculo de (h/r) del fuel que afecto a la totalidad de los test del modelo 2. Previamente habían funcionado con el modelo erróneo pero es necesario recalcular los problemas y las matrices de todos los test. Finalmente ante la existencia de implementar nuevas modificaciones en la fase Beta del software (véase Resultados), dentro de la definición de las funciones para adimensionalizar las, será necesario

recalcular TODOS los test de todos los modelos, por lo que se ha pospuesto este trabajo hasta la versión Beta.

1.5 *Archivo *.prop*

Cuando nos referimos a guardar un proyecto de simulación, únicamente hace referencia a la estructura y valores intrínsecos del sistema. Es decir a Componentes, relaciones, valores de semilla o valores indicados. No guarda simulaciones ni datos extra.

Aunque el programa actualmente aún está en una fase muy inicial del mismo, cuya mayor respuesta se realiza mediante clases test, o código manual por consola, se ha implementado las bases del futuro sistema de archivos, con el objetivo de que el usuario no necesite compilar código ni un conocimiento previo.

Esta implementación es simple y sencilla, ya que únicamente se presenta como test o prueba de la futura funcionalidad, es decir, actualmente no es práctica de cara a un usuario real, pero esto es debido a que es una versión Alfa del software.

Se han realizado 3 tipos de implementaciones, 2 independientes de los cambios y modificaciones del propio software que se auto adaptan. La tercera se basa en un lenguaje de comandos predefinido que es más práctico para la escritura humana y la futura simulación por consola, sin embargo es sensible a los cambios que se produzcan en la estructura y clases del software. Por ello se ha optado por un desarrollo de las dos principales, con un mero test de la tercera.

1.5.1 **Serialización de objetos.**

El método más eficaz y actualmente 100% funcional se basa en la serialización de java de objetos en archivos binarios.

Una vez definidos los componentes de la simulación todo se encuentra dentro del objeto principal SimulationProject. Por lo tanto serializando y guardando el objeto en un file, se puede cargar con posterioridad.

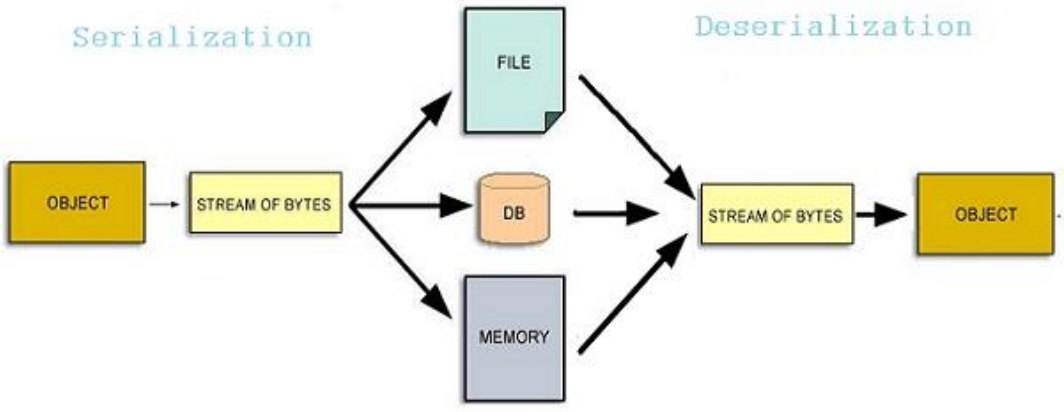


Fig. 54 Serialización de objetos Java.

Este tipo de método no solo puede utilizarse para almacenar el objeto como archivo sino para enviarlo por un socket a través de internet (conexión remota) o para guardarlo en algún tipo de base de datos Mysql o SQL.

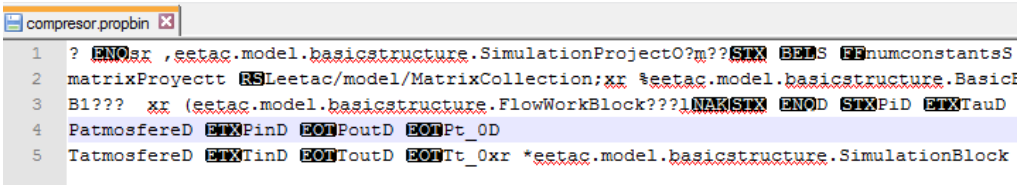


Fig. 55 Compresor Guardado binariamente.

La principal desventaja de este método es que si el usuario abre el dichero guardado, está en binario, es decir es inteligible y por lo tanto no se puede modificar manualmente.

1.5.2 JSon o GSon Objet

Se basa en las actuales serilizaciones de objetos webs. Permite obtener un resultado prácticamente idéntico al caso previo a través de JavaScript Object Notation.

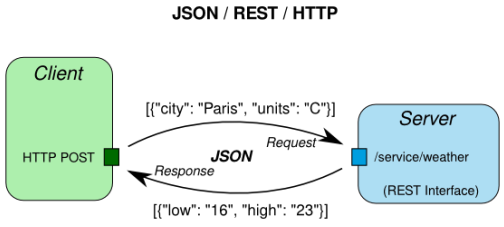


Fig. 56 JSon estructura y uso

Aunque ocupa más espacio y es más lento que el método anterior, permite abrir con un editor de textos el archivo modificando los parámetros del mismo.

Fig. 57 JSon codificacion en compressor.jprop

1.5.3 **Codificación Manual**

Se basa en la creación de un pseudo-lenguaje que el programa lee, separa línea a línea y va realizado los comandos que existan en el archivo.

Este método de guardar información es sensible a cambios del propio código por lo que se ha descartado su desarrollo hasta tener una versión estable del software, ya que todo trabajo realizado en el mismo puede ser modificado durante la versión Beta.

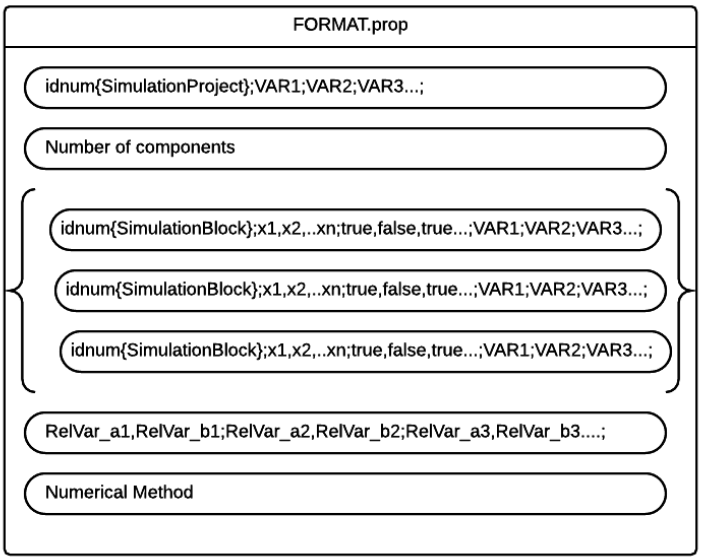


Fig. 58 Formato *.prop

Se ha establecido el siguiente código de comandos basico:

1. Línea, indica el tipo de proyecto simulado, se corresponde con el idnum del objeto que se desea instanciar normalmente un SimulationProyect o un Engine.
   Concatenado al idnumber el conjunto de variables del objeto instanciado en un orden predefinido. (Actualmente no carga las variables únicamente instancia el idnumber).

2. Número de componentes que contiene la simulación.

3. N líneas, una por elemento con la siguiente estructura:

   ‘Idnum;X;Cte;variables’
   Donde las variables siguen siendo concatenadas por ‘;’ y en el caso de ser vectores se concatenan con ‘c’.

4. Relaciones entre las variables de los elementos según el orden de indexado del conjunto. ‘variable_a1,variable_b1; Variable_a2,variable_b2;…’

5. Método numérico a usar. Actualmente implementado y funcional solo existe uno por lo tanto no es funcional.

1.6 *Resumen Diagramas de clases*

Los diagramas principales de clases son los siguientes:

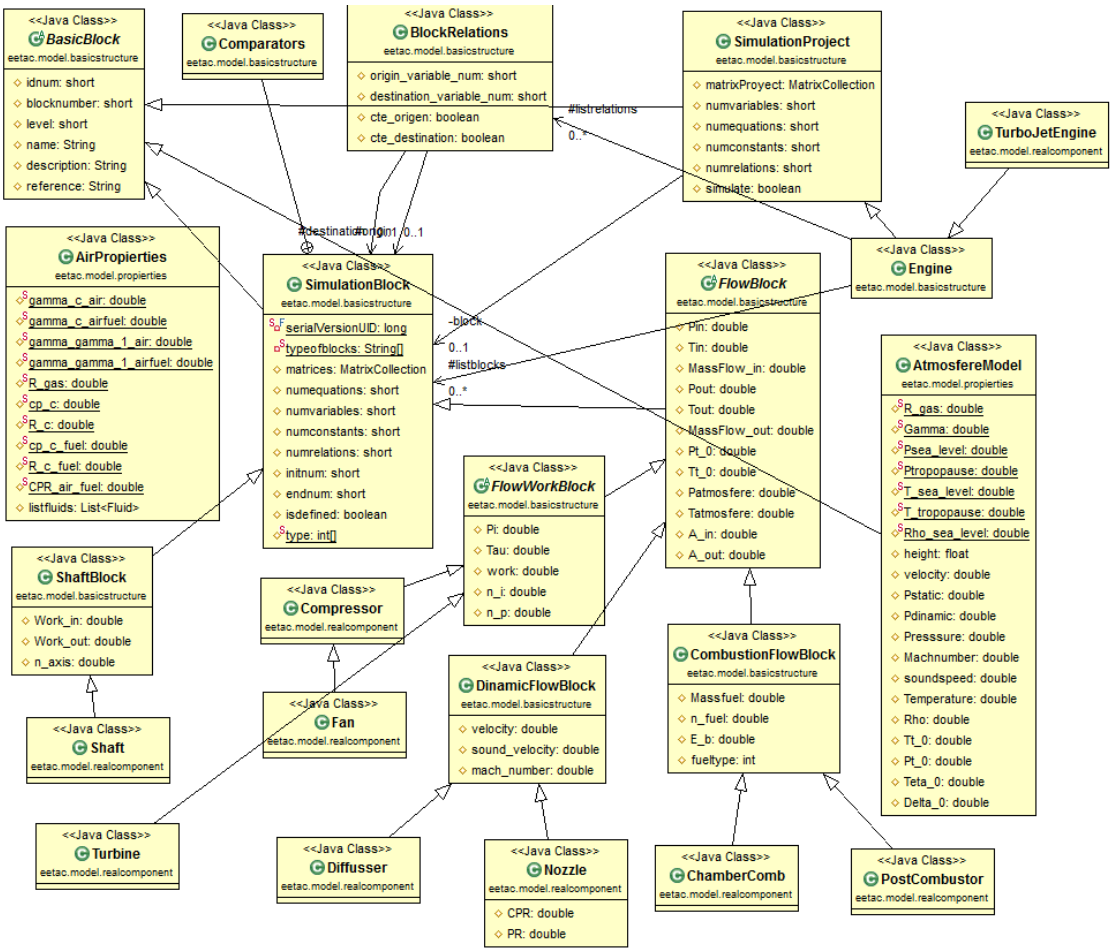


Fig. 59 Diagrama de clases principal

1. Anexo D: Desarrollo Proyecto

En este apartado también se aborda la planificación y gestión del desarrollo del software a lo largo del periodo que ha ocupado el TFG junto a los principales problemas encontrados durante el desarrollo del software.

1.1 *Planificación*

La planificación de este TFG se ha visto bastante condicionada por la limitación del periodo del mismo. Por cuestiones burocráticas he realizado en paralelo durante el semestre 2015-2 mi TFG de sistemas de telecomunicación y aeronavegación. Por consiguiente durante los primeros meses del semestre fue escasa o nula la productividad en el proyecto por la prioridad en importancia del otro TFG, lo cual fue fuertemente compensada con una carga de trabajo alta durante los meses de verano y septiembre.

1.1.1 **Principales etapas**

Durante el periodo Febrero-Abril únicamente se dedicó tiempo a la lectura de teórica, PFG previos, repaso de los temarios relacionados impartidos durante la carrera y la búsqueda de publicaciones relacionadas.

Entre Abril y Junio se definieron las bases y esquemas del software junto a los modelos seleccionados para el modelo 1 y 2.

Entre el 1 julio y el 1 de septiembre, realice un promedio de 4-6 horas diarias en la sala de becarios durante la semana, con respectivas jornadas similares durante el fin de semana consiguiendo la realización de la versión alfa del software en apenas un mes y medio. Las dos últimas semanas de agosto fueron principalmente test unitarios y fix de pequeños defectos encontrados en el software.

A partir de Septiembre la carga de trabajo principal estaba realizada y se descartaron varios objetivos por retrasos acumulados o tener expectativas demasiado extensas. Designándolos para la versión Beta.

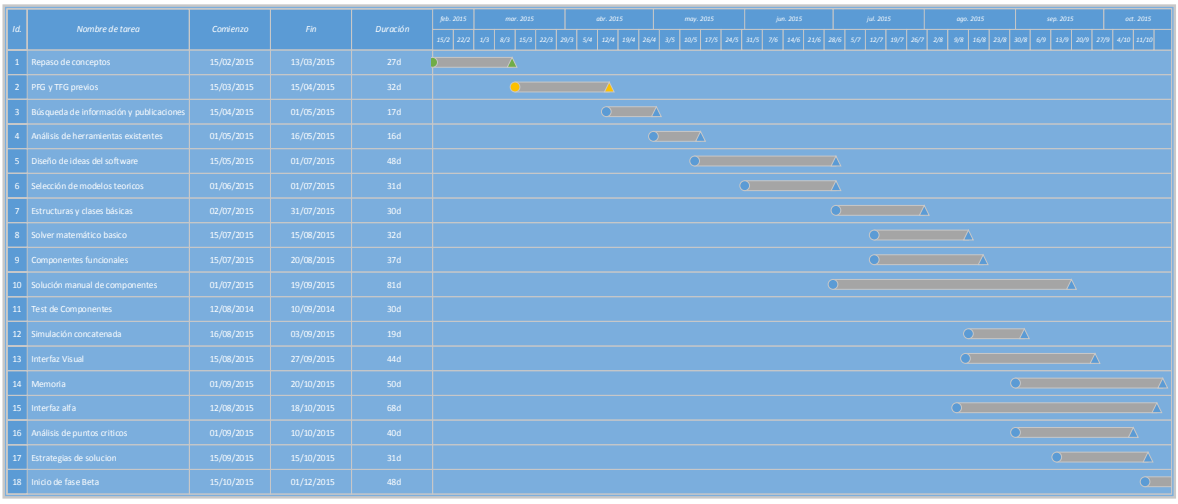
Entre 1 Septiembre y 1 Octubre se descartó finalizar a tiempo la interfaz visual avanzada y se optó por minimizar al máximo las funcionalidades o apariencia de la misma, mientras se establecían las bases de la memoria.

1 Octubre y 20 de Octubre, finalización de la memoria y desarrollo de ciertas soluciones orientadas hace la versión Beta del software. Detección de un fallo en las ecuaciones del modelo 2 que han dejado sin validación integral dicho modelo para la versión Alfa.

1.2 *Gant*

Los principales punto o hitos del TFG son:

- Repaso de Conceptos 15-Feb hasta 15 Marzo.
- TFG y PFG previos relacionados 15 Marzo – 15 Abril.
- Búsqueda de publicaciones e información adicional 15 Abril - 1 Mayo.
- Análisis de herramientas existentes y fundamentación de las bases del software. 1-15 Mayo.
- Diseño de ideas básicas del software 15-Mayo – 1 Junio.
- Selección de modelo 1 y modelo 2. 1-Junio – 1 Julio.
- 1 Julio – 15 Julio desarrollo de clases básicas y diseño del solver matemático en papel.
- 15 Julio – 1 Agosto, primeros componentes funcionales y test de solver.
- 1 Agosto – 15 Agosto math core integrado en programa y test de primeros componentes.
- 15 Agosto – 1 Septiembre, desarrollo de test de cada componente, simulación de varios componentes funcionalmente.
- 1 Septiembre – 8 Septiembre prueba de turbojet sin implementación de test.
- 15 Agosto – 20 Septiembre Desarrollo de interfaz visual avanzada.
- 25 Septiembre detección de fallo en ecuación y desestimada la interfaz avanzada.
- 5 Septiembre – 20 Octubre realización de memoria.
- 25 Septiembre – 15 Octubre realización interfaz Alfa.
- 20 Septiembre – 20 Octubre evaluación de puntos críticos y estimación de estrategias de resolución respecto a la versión Beta.



Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Repaso de conceptos	15/02/2015	13/03/2015	27d
2	PFG y TFG previos	15/03/2015	15/04/2015	32d
3	Búsqueda de información y publicaciones	15/04/2015	01/05/2015	17d
4	Análisis de herramientas existentes	01/05/2015	16/05/2015	16d
5	Diseño de ideas del software	15/05/2015	01/07/2015	48d
6	Selección de modelos teoricos	01/06/2015	01/07/2015	31d
7	Estructuras y clases básicas	02/07/2015	31/07/2015	30d
8	Solver matemático basico	15/07/2015	15/08/2015	32d
9	Componentes funcionales	15/07/2015	20/08/2015	37d
10	Solución manual de componentes	01/07/2015	19/09/2015	81d
11	Test de Componentes	12/08/2014	10/09/2014	30d
12	Simulación concatenada	16/08/2015	03/09/2015	19d
13	Interfaz Visual	15/08/2015	27/09/2015	44d
14	Memoria	01/09/2015	20/10/2015	50d
15	Interfaz alfa	12/08/2015	18/10/2015	68d
16	Análisis de puntos criticos	01/09/2015	10/10/2015	40d
17	Estrategias de solucion	15/09/2015	15/10/2015	31d
18	Inicio de fase Beta	15/10/2015	01/12/2015	48d

Fig. 60 Gant del proyecto

1.3 *Análisis Github*

El software esta público en la dirección:

https://github.com/ropnom/PropSimulator

Desde la cual se puede ver un seguimiento del trabajo del mismo proyecto así como la modificación y cambios realizados en el código.

El software actualmente cuenta con 7752 líneas de código de las cuales han seguido una tasa de ½ en reescritura. Es decir se han escrito 14.640 líneas de las cuales se han eliminado o reescrito o factorizado 7194. Este efecto es debido a los cambios en las estructuras jerárquicas de los objetos, cuando se encuentran características comunes a dos objetos, se integran en un objeto común ancestro, eliminando el código 2 o más clases y reagrupándolo de manera más genérica en un clase más abstracta en una jerarquía más alta.

Este proceso de mejora durante el desarrollo del software ha concluido con una gran variedad de funciones complejas implementadas de manera genérica y modular en los objetos de jerarquía más alta, que realmente son utilizadas por los objetos concretos, jerarquía baja, componentes reales como compresores o turbinas. Eliminando la necesidad de repetir ciertas parte del código comunes en funcionalidades o similares en apariencia.

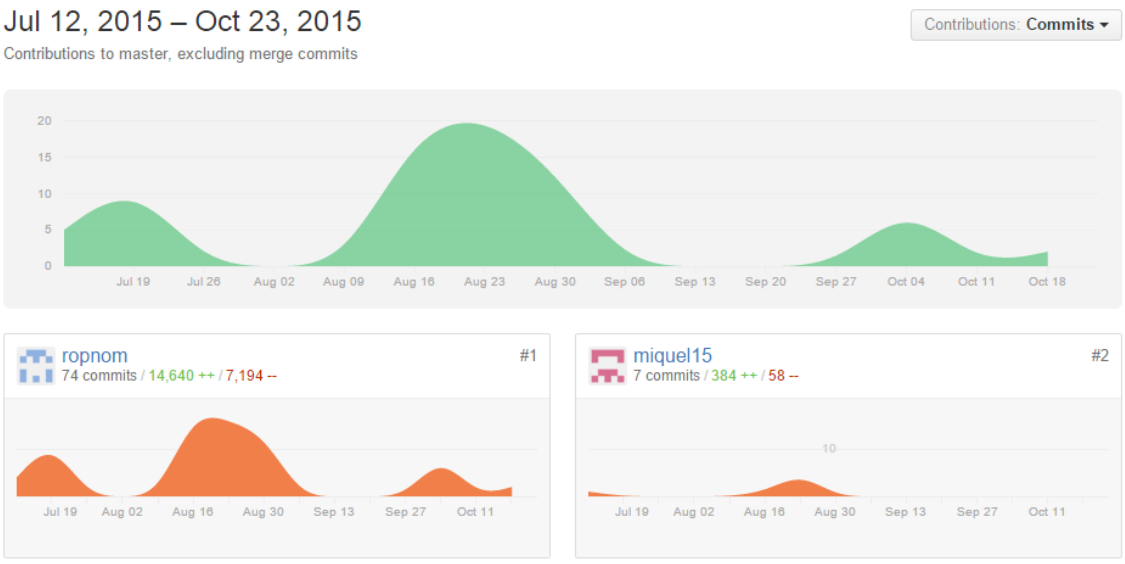


Fig. 61 Diagrama de trabajo y líneas aportadas al software 12-Julio hasta 23 Octubre

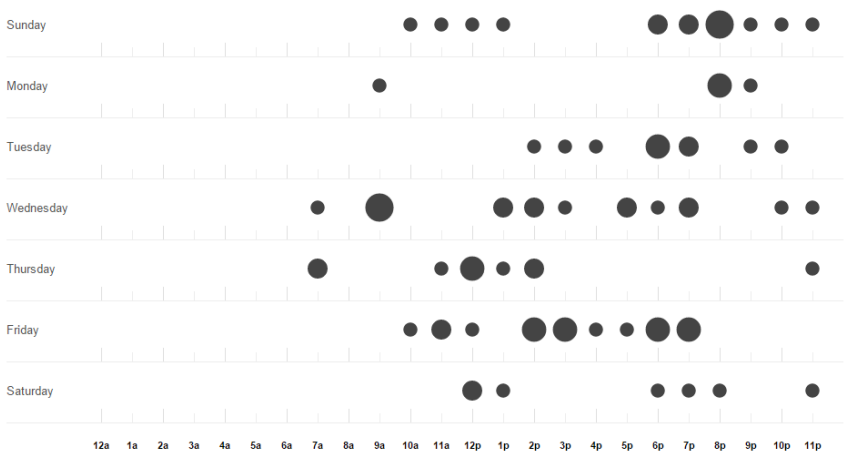


Fig. 62 Histograma semanal de producción en el desarrollo del software

En la figura 61 se puede apreciar la aportación e incrementos de líneas del software descrito en el apartado 1.2, donde se diferencian de forma clase 3 fases:

- Julio y la creación de las bases del software. 3000 líneas.
- Primeras semanas de agosto, aunque el volumen del software se triplico, la abstracción en clases jerarquizadas y generalización de las funciones realmente dio como resultado un incremento no excesivo del código, +4000 líneas, eliminadas -2200.
- Test e interfaz gráfica durante las dos última semanas de agosto se implementaron los test e inicios de la interfaz gráfica +4600 líneas -2600 líneas. En este periodo coincide con la colaboración de Miquel becario en prácticas, que colaboro en los test de modelación 2 con 384 líneas y 58 sobrescritas.
- Septiembre, tiene un vacío de desarrollo principalmente debido a la corrección de errores y desarrollo de la memoria.
- Octubre contiene gran parte del desarrollo comentado en los resultados enfocados directamente a la versión Beta. Como la interfaz gráfica avanzada o mejoras sustanciosas de la estructura del software.

En la figura 62 que de mostrada la veracidad de la planificación horaria y las horas desarrolladas durante el desarrollo del software con un total de 500-530 horas invertidas únicamente en programación.

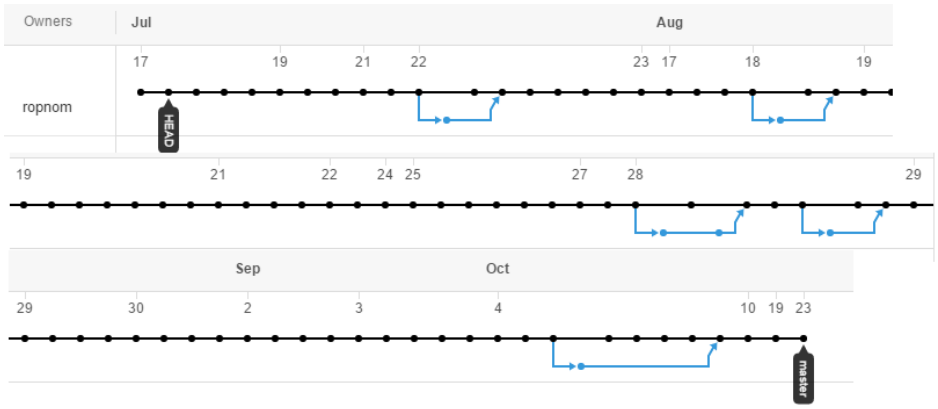


Fig. 63 Diagramas de Commits subidos al servidor público.

El desarrollo de la versión se ha hecho principalmente de manera secuencial incremental las funcionalidades de la misma (Fig.63), sin embargo si ha existido algún que otro problema entre cambios realizados que son las ramas alternativas señaladas en azul, que finalmente convergen en la oficial.

1.4 *Objetivos descartados y razones*

Durante el desarrollo del programa sean eliminados o pospuestos varios objetivos ambiciosos propuestos en el periodo Junio-Julio.

1.4.1 **Interfaz grafica**

El principal objetivo pospuesto es la interfaz gráfica, ya que no aporta información objetiva del programa pero consume un tiempo valioso. Una de las principales características es que la interfaz gráfica está diseñada con SWING lo que la hace compatible con todo sistema operativo con java, es decir, la casi totalidad de los sistemas actuales. Sin embargo no dispone de un IDE sencillo o grafico como otros entornos tales como Visual Estudio o c#, obligando al desarrollo de la misma a base de código, prueba y error.

Por otra parte se ha intentado integrarla interfaz gráfica en las estructuras jerárquicas del proyecto, es decir, en vez de diseñar por ejemplo una ventana para cada uno de los componentes independientes que se puedan simular, se ha diseñado una ventana que en función del objeto a mostrar, autogenerar los elementos necesarios en la interfaz gráfica. Para realizar esta acción ha sido necesario implementar componentes gráficos personalizados con integración de eventos. Cuando menciono integración de eventos, es referida a que acciones en la interfaz gráfica son directamente actualizados en el objeto por el propio elemento grafico sin necesidad de realizar una Carga/Guarda de información.

Estos apartados han sido desechados de cara a este TFG y serán publicados en los anexos de la versión Beta.

1.4.2 **Objetivos descartados por importancia no critica**

Ciertos objetivos no aportan estructura o conceptos nuevos, sino que mejoran o implementan con mayor detalle funcionalidades ya existentes.

Son los casos del modelo 3, que mejoran los resultados respecto al método 2 pero que no tenían mayor importancia que la complejidad del software.

El control de la independencia de variables aportada por el usuario, es un tema importante, pero no se ha conseguido esclarecer un método “sencillo” o rápido para la contención del problema, por lo que actualmente el problema está solucionado a nivel de bloque pero no a nivel de conjunto. Se ha pospuesto este objetivo para la versión Beta por falta de tiempo y porque la importancia del mismo no es crítica.

Mejora de los métodos matemáticos implementados, no tiene sentido hasta tener una versión estable de las clases a simular, por otra parte no se esperan mejoras superiores al 20% respecto a los métodos actuales, lo que a escala temporal de la simulaciones es totalmente despreciable. Por lo tanto se ha pospuesto para la versión Beta el desarrollo de los mismos.

Se ha detectado que un tema de mayor importancia que la mejora de los métodos matemático, es la generación más acertada o concreta de la semilla. Este objetivo tiene un beneficio mayor que la inversión de tiempo en el core matemático, sin embargo necesita ser planteado desde un punto de vista amplio lo cual no ha permitido plantearlo para esta versión del software Alfa, que únicamente cuenta con semillas estáticas predefinidas. Pero que seguramente será una de las principales características desarrolladas en la versión Beta.