Anexos

Anexo 1. Algoritmo de Tarjan

Vamos a explicar el algoritmo de Tarjan exhaustivamente*. De manera complementaria, hemos realizado el diagrama de flujo adjunto para facilitar su comprensión.

Comenzando desde un nodo cualquiera, se van recorriendo caminos por los nodos según la metodología *DFS,* de manera que los nodos conforme son visitados se van almacenando en una pila. Esta pila guarda los nodos del actual camino o desde la última marcha atrás (*backtracking*) ocurrida.

Los nodos recorridos por primera vez en el camino se guardan en una lista y una pila. Cuando se retrocede, los nodos son eliminados de la lista pero no de la pila. A su vez para cada nodo se guarda su posición en la pila que denominaremos *LowLink* del nodo.

El algoritmo consiste en una cantidad de pasos mayores que engloban una serie de pasos menores. Un paso mayor requiere comenzar poniendo en la pila y en la fila cualquier nodo que no ha estado en la pila en el anterior paso mayor. Entonces ejecuta una secuencia de pasos menores, cada uno de los cuales agranda la fila en uno o la reduce en uno si hay marcha atrás. El paso mayor termina cuando tanto la fila como la pila no contienen ningún elemento. Un paso menor comienza con la búsqueda de la arista saliendo del nodo *v* final de la fila, excluyendo aquellos que ya han sido visitados. Si una arista apunta a un nodo *w* cuyo *LowLink* es menor que el nodo *v* entonces el *LowLink* de *v* pasa a valer lo mismo que el de *w*. La búsqueda continúa desde el nodo al final de la fila hasta que se sucedan una de las siguientes posibilidades:

1) Una arista no apunta a un nodo que no está en la pila. En este caso el nodo es añadido a la pila y a la fila. Y el paso menor termina.

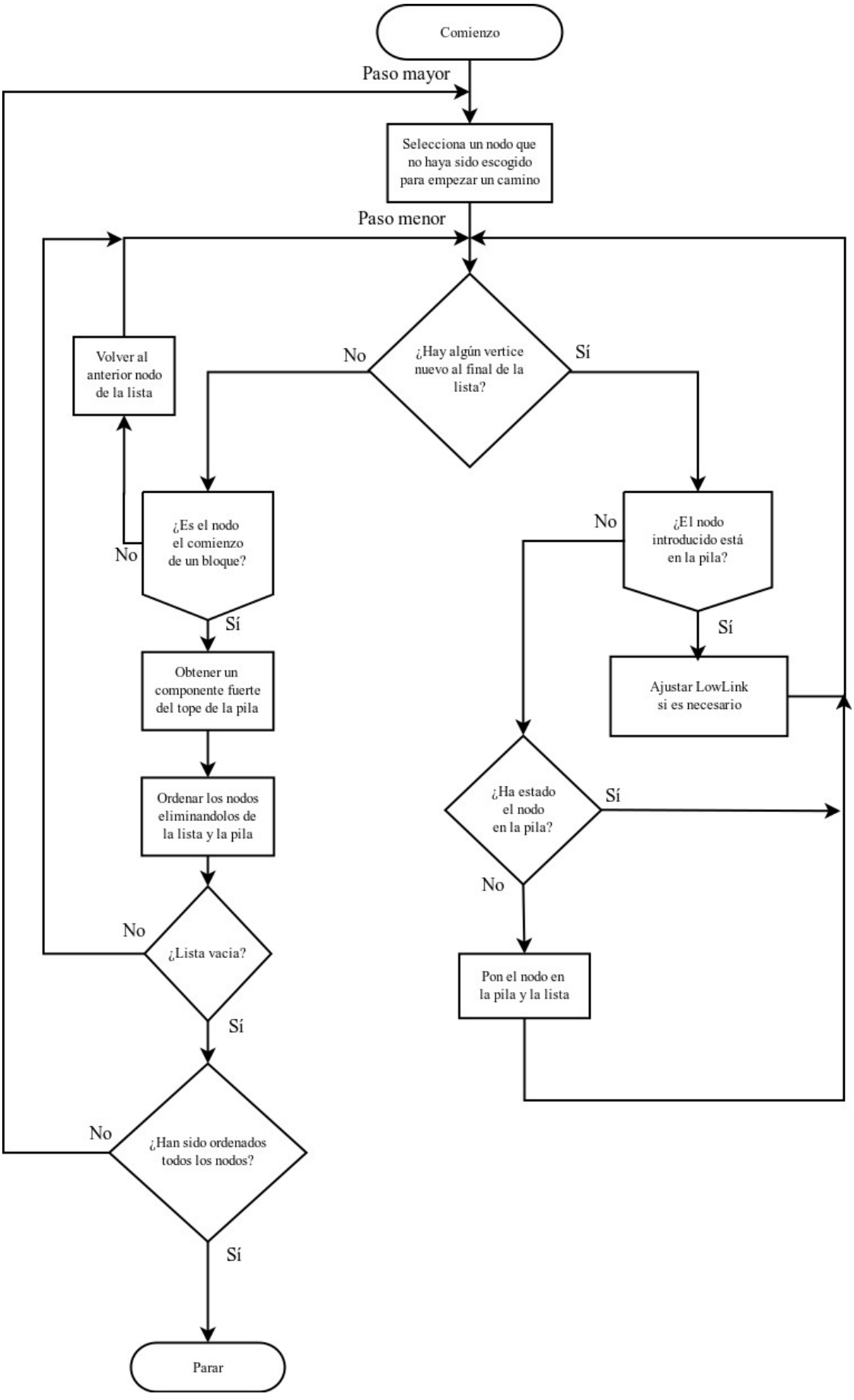
2) No quedan más aristas por visitar. Entonces analiza *Lowlink* tomando caminos divergentes:

   1. El *LowLink* del nodo al final de la fila puede ser el menor. En este caso este nodo es la raíz del componente fuerte. Por tanto *v* y todos los nodos por encima en la pila se eliminan de esta y forman un componente fuertemente conexo. Termina el paso menor haciendo marcha atrás al nodo anterior a *v* en la fila, a no ser que está y la pila estén vacías. En cuyo caso el paso mayor habría concluido.
   2. Puede haber un nodo con menor LowLink que el de *v* en la pila. En este caso se traslada al nodo *w* anterior a *v* en la lista. Al *LowLink* de *w* se le asigna el valor del de *v*.

El paso mayor termina cuando el paso menor hace que la pila no tenga valores en ella, una situación inevitable. En caso de que queden nodos sin ordenar, se produce el comienzo de otro paso mayor.

* Hemos utilizado los puntos 7 y 15 de la bibliografía en la realización de este apartado.

Lógicamente el algoritmo termina cuando todos los nodos han sido ordenados.



Anexo 2. Métodos de resolución numéricos

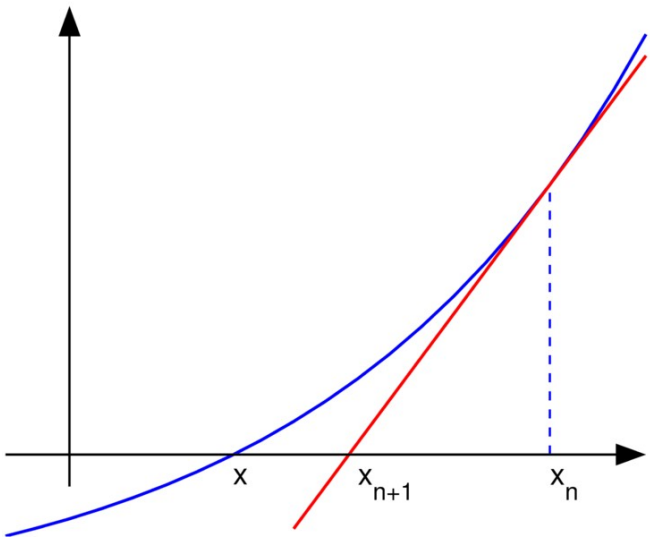
En este anexo vamos a dar una explicación de los distintos métodos numéricos que existen para comprender las razones que han llevado a la elección del método utilizado*:

1) Newton-Raphson

El método de Newton-Raphson es un método abierto, en el sentido de que su convergencia global no está garantizada. La única manera de alcanzar la convergencia es seleccionar un valor lo suficientemente cerca de la raíz buscada. Por tanto para que el algoritmo sea efectivo el punto inicial debe estar muy cerca del cero de la función. La cercanía del punto inicial a la solución depende de muchos factores, como por ejemplo puntos de inflexión, pendientes grandes, zonas de no existencia de la función, etc. Una vez hecho esto el algoritmo linealiza la función por la recta tangente en ese valor supuesto. La abscisa en el origen de dicha recta será, según el método, una mejor aproximación de la raíz. Se realizan iteraciones sucesivas hasta que converja.

Formulación del método de Newton-Raphson para una variable:

$$Xn+1 = Xn - \frac{f(Xn)}{f'(Xn)}$$



Método de Newton en un punto. Imagen extraída de la Wikipedia

Aunque es de convergencia local, converge muy rápidamente; de hecho, el orden de convergencia de esto método es cuadrática, a no ser que la multiplicidad algebraica sea mayor que uno, en cuyo caso la convergencia es lineal.

Aunque hemos tratado con casos y ejemplos de una sola variable se puede perfectamente extrapolar a funciones de varias variables.

$$Xk+1 = Xk - F(Xk) * J(Xk)^{-1}$$

* Hemos utilizado los puntos 1, 4 y 15 de la bibliografía de la memoria para realizar este apartado

Siendo F el vector de las funciones y J su Jacobiana.

Para este caso se sigue cumpliendo la convergencia local así como el orden de convergencia tan rápido. Sin embargo, tenemos un problema extra, aunque antes podía aparecer en el caso de que la derivada valiera cero, y es el caso de las Jacobianas singulares que se puedan dar. Además vemos que para cada iteración tenemos que resolver un sistema de ecuaciones lineal:

$$F(Xk)*J(Xk)^{-1}=A(Xk)$$

Resolver este sistema incrementa notablemente el número de operaciones que deben realizarse ya que no podemos utilizar métodos numéricos para resolver este sistema. Puesto que no podemos garantizar que se cumplan las propiedades que suelen requerir esos métodos, como por ejemplo que sea diagonal positiva dominante. Por lo que se requiere su resolución mediante métodos analíticos, como LU o QR, métodos que representan un coste computacional muy elevado, de $0(n^3)$.

2)Búsqueda lineal

Una manera de aumentar la convergencia del método de Newton-Raphson es aplicarle un *backtracking* a la hora de realizar una iteración. La idea consiste en que si el algoritmo va a realizar un salto muy grande que provoque que la función caiga fuera de la zona de aproximación o donde la función no exista, se reduzca este salto. Esto se debe a que la dirección del método de Newton siempre va hacia la solución del problema. Para ello se aplica un coeficiente λ a la fórmula antes mencionada:

$$Xk+1=Xk-\lambda*F(Xk)*J(Xk)^{-1}$$

Este coeficiente sera siempre menor o igual a uno. Cuando vale uno sera un paso de Newton normal, es decir el método habitual.

Sin embargo, el problema de este método es que si λ fuese muy pequeña siempre perderíamos el orden de convergencia cuadrático. Por tanto, nos interesa siempre intentar probar con $\lambda=1$ y si es necesario, reducir el valor.

Considerando que el programa trabaja con ecuaciones, elaboramos una única función objetivo para introducir en el método de optimización mediante la siguiente fórmula:

$$f=\frac{1}{2}*\sum F(x)^2$$

Es decir nuestra función objetivo es la suma de los cuadrados de las funciones multiplicado por un medio.

Por tanto, buscamos que cada iteración se pruebe con un valor de $\lambda=1$. Si tras aplicarlo no se cumple la siguiente inecuación

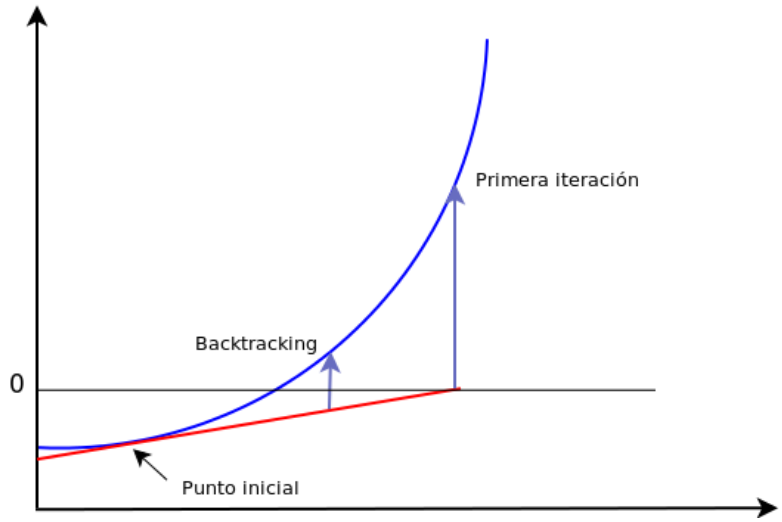
$$f(Xnew)\leq f(Xold)+\alpha\nabla f(Xnew-Xold)$$

Donde:

- $\alpha = 10^{-4}$
- $f(Xnew)$: f evaluado en el punto nuevo calculado en la iteración.
- $f(old)$: f calculado en el punto inicial.
- $\nabla f(Xnew - Xold)$: el gradiente de f evaluada en la diferencia entre el punto nuevo y el antiguo.

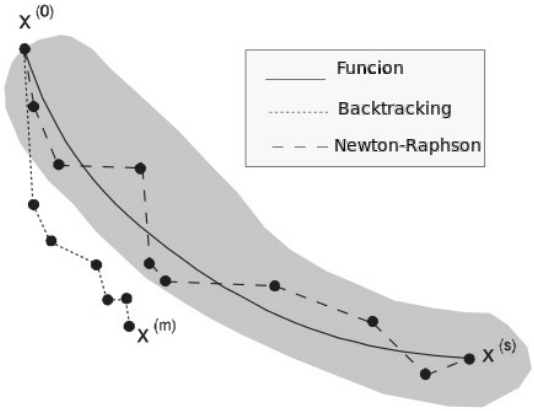
reduciremos el valor de λ hasta que lo supere, utilizando el método de búsqueda lineal para avanzar afianzadamente.

Es decir obligamos a que f siempre tenga que disminuir, de no ser así, reduciremos el paso.
Como podemos apreciar por la metodología, se van a realizar muchas más iteraciones de las que parecen imprescindibles en un primer momento. Esto se debe a que en cada iteración en la que tengamos que hacer *backtracking* porque no es bueno el punto nuevo, conlleva resolver un sistema de ecuaciones lineal. Como contrapartida, conseguiremos aumentar la convergencia del método de local a global.



Método de Newton en el que se muestra el resultado de un *backtracking*

Como podemos apreciar en la gráfica al hacer *backtracking* nos acercamos a la solución en vez de alejarnos de ella.
En la siguiente imagen mostramos como *backtracking* funciona cuando Newton-Raphson no. La zona gris es la zona donde sí existe la función físicamente. Como podemos apreciar, Newton se sale mientras *backtracking*, que se asemeja más a la función, se mantiene dentro de los límites.



Saltos que realizan los distintos métodos comparados con la representación de la función

3) Región de confianza

Lo que hacemos en este método es idear una función f como la descrita en el apartado 2, y creamos un modelo de f en el punto Xk , mk que satisfaga:

- $mk = f(Xk)$
- $\nabla mk(0) = \nabla f(Xk)$

Dado un modelo y un radio δk , obtendremos Sk , como la solución del problema de optimización:

- Minimiza: $mk(s)$
  - Sujeto a $\|s\| \leq \delta k$

Donde es la norma L2. Aceptaremos Sk y haremos $Xk+1 = Xk + Sk$ si mk cumple la siguiente condición:

$$\frac{f(Xk+Sk) - f(Xk)}{mk(Sk) - f(Xk)} \geq \alpha$$

Donde α es una constante muy pequeña.

Si se cumple, aceptamos Sk y tenemos que hacer δk más grande, introduciendo cualquier número dentro del siguiente intervalo:

$$[\|Sk\|, max(\delta k, \gamma * \|Sk\|)], \gamma > 1$$

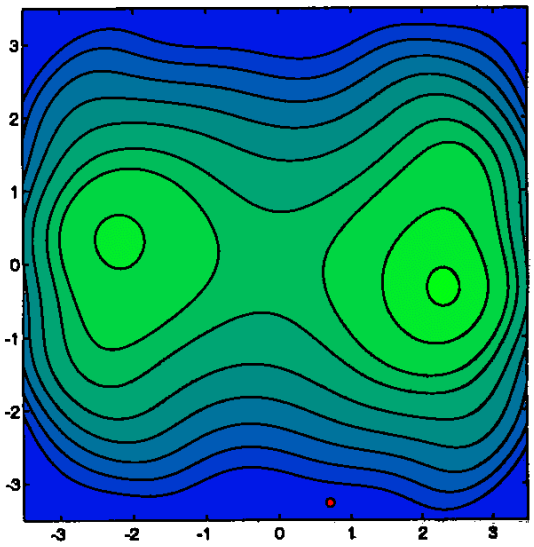
Vamos a ver un ejemplo para explicar mejor estas ideas:

$$f(X1, X2) = -10 * X1^2 + 10 * X2^2 + 4 * \sin(X1 * X2) - 2 * X1 + X1^4$$

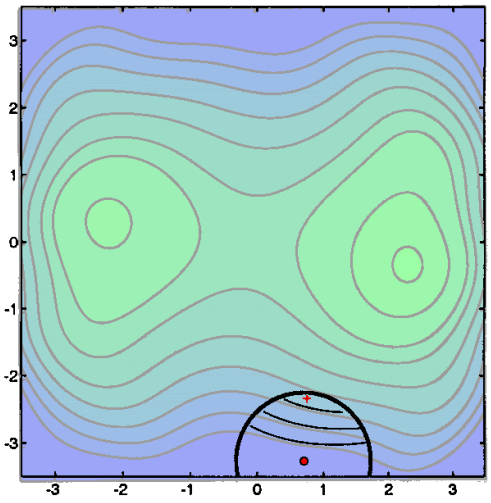
1)Partimos del punto inicial:

$X1 = 0{,}7 ; X2 = -3{,}3$ .El dibujo representa la función $f(X1, X2)$, donde el punto rojo es el punto inicial.*

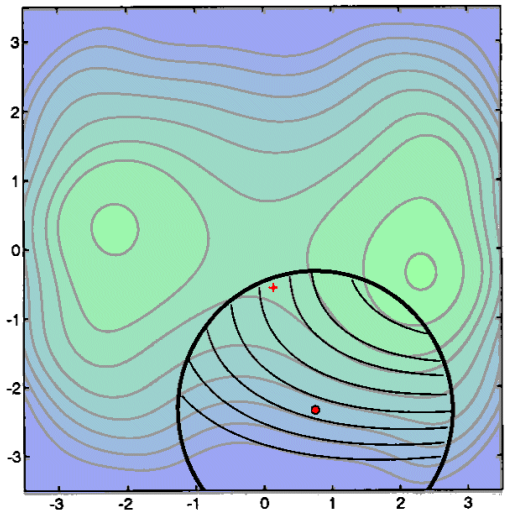
* El siguiente ejemplo junto con las imagenes han sido extraidos de la página www.applied-mathematics.net/



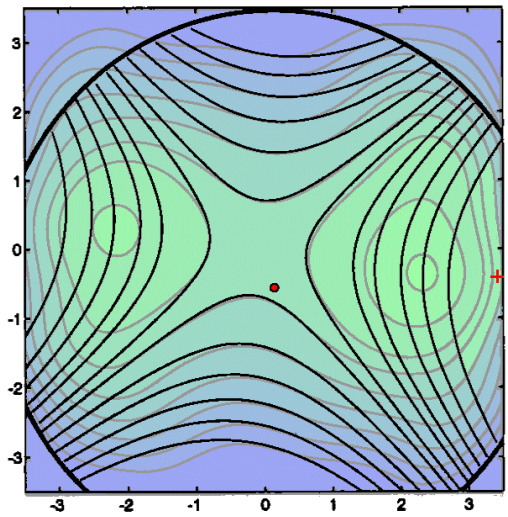
2) Probamos con un radio inicial. Dentro de ese radio buscamos el punto óptimo, indicado con la cruz de color rojo.



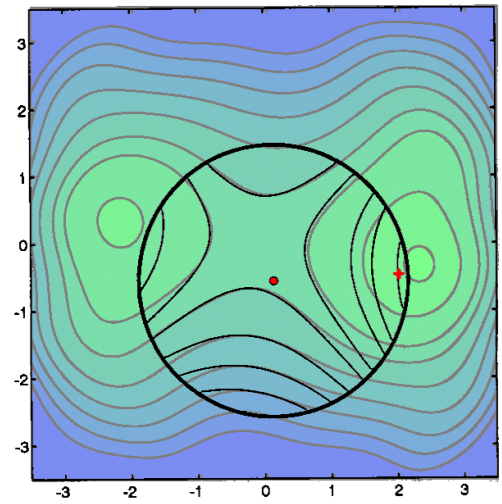
3) Si el valor de la función objetivo es menor, aumentamos el radio de confianza. Y la nueva iteración se hará desde ese nuevo punto.



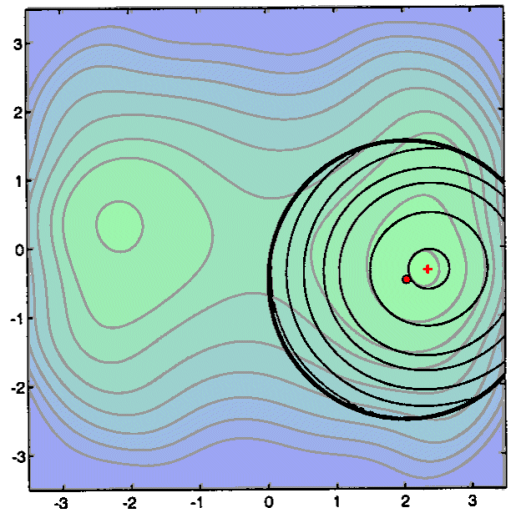
4) Si en la cruz roja el valor de la función no es menor, reducimos la región de confianza y volvemos a probar.



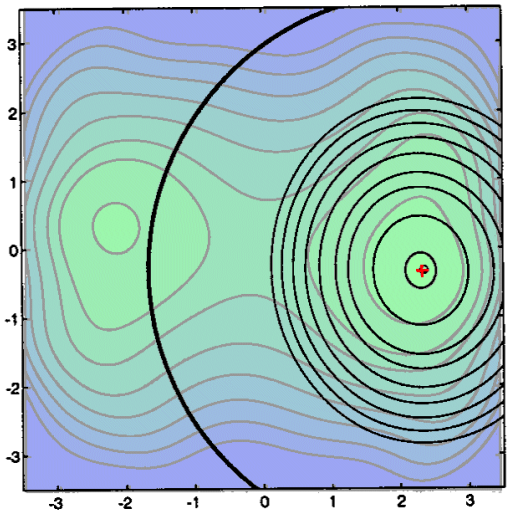
5) Ahora la nueva posición sí es mejor, de manera que nos trasladamos al nuevo punto.



6) En la siguiente imagen podemos ver como el algoritmo se repite muchas veces, reduciendo el radio, hasta que por fin encuentra un punto mejor.



7) Finalmente el radio va creciendo conforme nos aproximamos al punto final.



3.1) Levenberg-Marquardt

Este algoritmo interpola entre el método Gauss-Newton (en adelante, GNA) y el método del máximo descenso del gradiente. Levenberg-Marquardt (en adelante, LMA) es más robusto que GNA, lo que significa que este funcionará cuando el otro no lo haga, incluso si empieza en un punto muy lejano al final. Sin embargo, LMA es más lento que GNA. Básicamente, LMA es el algoritmo GNA pero utilizando aproximaciones de región de confianza.

Recordemos la importancia de la presencia de LMA, pues es un algoritmo muy extendido, podemos encontrarlo en muchas plataformas, por ejemplo en resolución de sistemas no lineales en Matlab.

Como habíamos comentado anteriormente, en los algoritmos de región de confianza se aproximaba una región para trabajar con ella por un modelo $mk(s)$. En el caso de LMA la aproximación es la siguiente:

$$mk(s)=\|F(Xk)+J(Xk)*s\|$$

Siendo F el vector con las funciones y J su respectiva Jacobiana. Sin obviar la norma L-2 y teniendo

en cuenta el cumplimiento de las condiciones de minimización comentadas en el apartado 3. Por tanto, el problema queda finalmente así:

$$s(\mu) = -[J(Xk)^T * J(Xk) + \mu I]^{-1} * J(Xk)^T * F(Xk)$$

Donde:

I = Matriz unidad.

μ = Parámetro de control.

S = Paso.

J = Jacobiana.

F = Funciones.

Utilizaremos el valor único valor no negativo $\mu \varepsilon \Re$ que cumpla $\|S(\mu)\| = \delta$, a no ser que $\|S(0)\| \leq \delta$ en cuyo caso utilizaremos el paso de Newton.

Depende de cómo controlemos el algoritmo, obtenemos un método u otro. En el caso de LMA modificamos el parámetro μ para controlar el algoritmo.

Este algoritmo ha demostrado que es potente y se comporta muy bien.

3.2) Hebden-Moré

Este método es igual que LMA salvo que varía el parámetro δk y calcula μ , es decir, al revés de como actúa LMA.

Hebden-Moré es más robusto que LMA, sin embargo, presenta el inconveniente de que hay que resolver el sistema para muchos valores de μ en cada iteración. Por lo que a pesar de ser más potente, resulta mucho más lento que LMA.

3.3) Double Dogleg

Para evitar la lentitud del método de Hebden-Moré, se creó este método cuya labor es la de aproximar la solución del problema de región de confianza limitando que $Xk + s$ debe pertenecer a la curva lineal que une Xk , el punto de Cauchy (el punto que minimiza la ecuación en la dirección de máximo descenso) y el punto de Newton (el punto que minimiza la ecuación sin limites). El problema con esta aproximación puede ser fácilmente resuelto y en la mayoría de los casos, el paso resultante es una buena aproximación del resultado.

Este método ha demostrado ser robusto. Sin embargo, no lo es tanto como LMA.

Anexo 3. QR/LU

En este anexo procedemos a explicar los métodos LU y QR* así como algunas características importantes que hemos tenido que tener en cuenta a lo largo del proyecto respecto a estos métodos.

Factorización LU:

La factorización LU de una matriz es una factorización, que resume el proceso de eliminación gaussiana aplicado a la matriz y que es conveniente en términos del número total de operaciones de punto flotante, cuando se desea calcular la inversa de una matriz o cuando se resolverá una serie de sistemas de ecuaciones con una misma matriz de coeficientes.

Primero vamos a explicar la factorización LU y luego la PLU, que es una pequeña modificación de la LU para mejorarla.

Suponga que una matriz A $m\,x\,n$ se puede escribir como el producto de dos matrices:

$$A = LU$$

donde L es una matriz triangular inferior $m\,x\,n$ y U es una matriz escalonada $m\,x\,n$. Entonces para resolver el sistema:

$$Ax = b$$

escribimos

$$Ax = (LU)x = L(Ux)$$

Una posible estrategia de solución consiste en tomar L para resolver:

$$Ly = b$$

Como la matriz L es triangular superior este sistema puede resolverse mediante sustitución hacia abajo, lo cual se hace fácilmente. Una vez con los valores encontrado de y , las incógnitas al sistema inicial se resuelve despejando x de $Ux = y$

Nuevamente, como U es escalonada, este sistema puede resolverse en caso de tener solución mediante sustitución hacia atrás. Estas observaciones nos dan la pauta para ver la conveniencia de una factorización como la anterior, es decir factorizar A como el producto de una matriz L triangular superior, por otra U escalonada. Esto es la descomposición LU.

El coste computacional de este algoritmo es de $O(n^3)$.

Factorización PLU

Muchas veces, no es posible escalonar una matriz solo con operaciones de eliminación, debido a que hay un cero en la diagonal o, aunque se puede escalonar en la diagonal, hay valores muy pequeños que pueden originar errores de redondeo ya que hay que dividir luego por los valores de

* Hemos utilizado el punto 15 de la bibliografía para redactar este anexo, así como los conocimientos de mi director.

la diagonal. Para estas matrices no funciona la factorización LU simple. Por tanto se realiza esta modificación la PLU donde P es una matriz de permutación que se obtiene cambiando las filas de la matriz identidad. En el caso de PLU se lleva a cabo una factorización LU pero se almacenan los cambios de filas en la matriz P.

Aunque hay matrices que requieren el cambio de filas para poder llevar a cabo LU, no suele surgir la necesidad de cambiar además columnas ya que con el cambio de filas se pueden suplir todos los problemas. Por lo que con PLU se pueden resolver sistemas de ecuaciones lineales si la matriz es cuadrada y no singular.

Factorización QR:

Es la descomposición de una matriz como el producto de una matriz ortogonal por una triangular superior. Suele usarse para resolver sistemas lineas no cuadrados y para sacar valores propios.

Si A es una matriz ($m x n$) con columnas linealmente independientes, entonces A puede factorizarse en la forma: $A = QR$

en la que Q es una matriz con columnas ortongonales ($m x m$) y R ($m x n$) es una matriz triangular superior.

En la resolución de sistemas de ecuaciones lineales la factorización QR tiene el mismo coste computacional que LU y tiene se aplica de una manera equivalente a este. La única diferencia en este aspecto es que QR puede usarse para resolver sistemas $m x n$. Además se puede hacer también el sistema de pivotaje parcial, siendo el nombre del algoritmo resultante PQR.

Anexo 4. Rendimiento del programa

Para evaluar el rendimiento del programa hemos procedido a resolver en distintos ordenadores el siguiente problema; hemos medido el consumo de memoria y el tiempo necesario de cálculo. También hemos realizado la misma prueba para el renderizado de ecuaciones.

El problema consta de 95 ecuaciones que exponemos a continuación:

M=81

L=1,86

G0=1366

LambdaPiel=0,45

CpPiel=2675

RhoPiel=1000

CpSg=3770

RhoSg=1060

EpsilonPiel=0,95

EpsilonRopa=0,37

Td1amb=18+273

Latitud=41,66

deltad1= (-1)*11,75

Taud1=Pi/4

Fd1a= 68

Td1ab=30 +273

Td1aPe=32,1+273

Td1aPi=33,8+273

T1=14,16

Fd1p1=128

Td1p1b=26,5+273

Td1p1Pe=33,4+273

Td1p1Pi=33,6+273

T2=14,08

Fd1p2=132

Td1p2b=25,9+273

Td1p2Pe=32,6+273

Td1p2Pi=32,7+273

T=(T1+T2)/2

v=(100/T)

Ey=(v*tan(20))^(2)*M/2*100/2,05

Ex=0,9*v^2*rhoair*Af/2*100

Wt=(Ex+Ey)/T

Af=2*r*l

As=0,202*M^(0,425)*L^(0,725)

2*pi*r*l+2*pi*r^2=As

Gdif=G0*6/100

Gdird1=50

Sin (gammad1)=cos(Latitud-deltad1)-2*cos(Latitud)*cos(deltad1)*sin(Taud1/2)^2

Gsold1=Gdird1+Gdif

Mopt=1/(sin(gammad1)+0,15*(3,885+gammad1)^(-1,253))

Pv=0,57*PsatAmb*76/10^5

Wtabla=2*Pv

sigma=5,67*10^(-8)

QRPD=(Tpiel^4*sigma-Gsold1-Td1amb^4*sigma)*0,9*As*31,9/100

QRPR=(TRopa^4*sigma-Gsold1-Td1amb^4*sigma)*0,7*As*68,1/100

Rpiel=Ln(r/(r-0,003))/(2*3,14*L*LambdaPiel)

Caudalsg=0

Tpiel=(Td1p1Pe+Td1p1Pi+Td1p2Pe+Td1p2Pi)/4

Rropa=0,155*0,57

Qropa=(Tpiel-TRopa)/Rropa*As*68,1/100

QRpielS=h*(Tropa-Td1Amb)*As*68,1/100

Qropa=QRPR+QrpielS

QDpielS=(Tpiel-Td1Amb)*h*As*21,6/100

Qpiel=QSudorPD+QDpielS+QRPD

muAir=0.00001815

rhoAir=1,218

LambdaAir=0.02498

Re=v*2*r*rhoAir/muAir

cpAir=1004

Pr=muAir*CpAir/LambdaAir

h*2*r/lambdaair=0,027*Re^(0,805)*Pr^(1/3)

Beta=2/(Tpiel+Td1amb)

Gr=9,81*Beta*(Tpiel-Td1amb)*(2*r)^3*rhoair^2/Muair^2

Mixta=Gr/Re^2

cpw=4183

h=k*(cpAir+0,009*cpw)

Tbh=12+273

Hlg=2500000

Psatamb=1389

WsatAmb=0,623*Psatamb/(101700-Psatamb)

Hamb=cpAir*(Tbh)+WsatAmb*(cpw*(Tbh)+Hlg)

PsatPiel=5012

Wsat=0,623*PsatPiel/(101700-PsatPiel)

Hi=cpAir*(TPiel)+Wsat*(cpw*(TPiel)+Hlg)

QSudorPD=k*(Hi-Hamb)*As*(10,3/100)

QSConvD=h*As*(10,3/100)*(Tpiel-Td1Amb)

Rvap=(894*Rropa)/0,36

Ktotal=(1/(k*As*21,3/100))+Rvap/(As*21,3/100)

QSudorPR=(wsat-wsatAmb)*Hlg/(Ktotal)

Latente=2272000

CaudalS*1000*Latente=QSudorPR+QSudorPD-QSConvD

Qtotal=Qropa+Qpiel+QSudorPR

Qtotal=QcondPiel+Qsudor+Qsangre

QcondPiel=(Tc-Tpiel)/Rpiel

Qsudor=CaudalS*1000*Cpw*(Tc-Tpiel)

Qsangre=CaudalSg*1000*Cpsg*(Tc-Tpiel)

Egenerada=Wt+Qtotal

Eta=Wt/(Egenerada)

Tcuerpo=Tc-273

TempRopa=Tropa-273

QconvTotal=QSConvD+QRpielS+QDpielS

QradTotal=QRPD+QRPR

QevaTotal=QSudorPR+QSudorPD-QSConvD

ExergiaDestruida=Qtotal-(1-Td1amb/Tc)*Qtotal

Consideraciones previas:

- Hemos contabilizado el tiempo de cálculo tras haber realizado una primera vez el proceso, ya que por el hecho de compilar el código en inicialmente, conlleva un retardo adicional.
- En el gasto de memoria tenemos dos datos, el naranja corresponde al tamaño que deja la

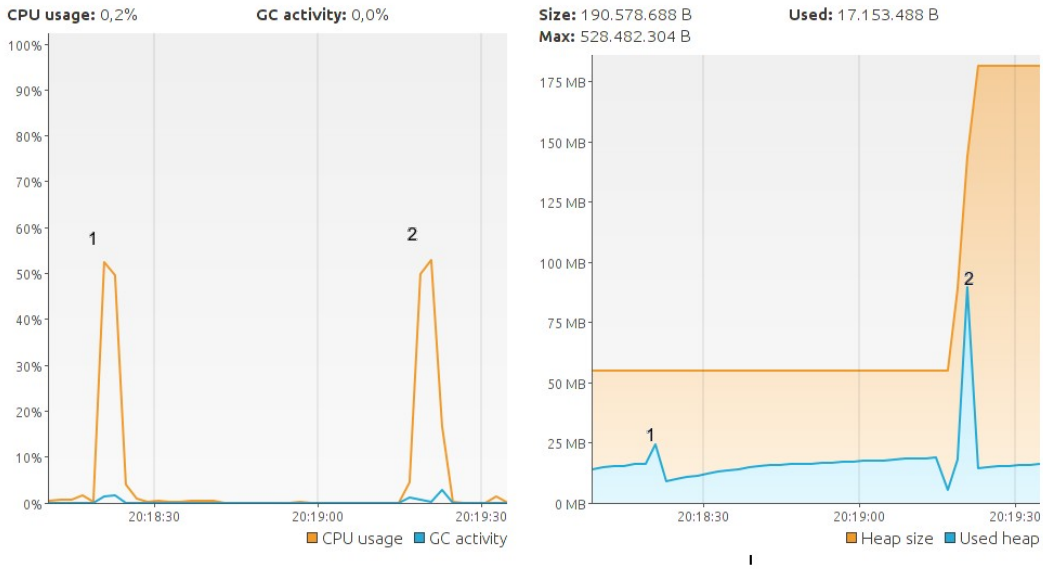
máquina virtual de Java para que gaste el programa y el azul, al consumo real de la aplicación.

- El gasto de procesador tiene a su vez dos colores, el azul que corresponde al consumo del recolector de basura de la máquina virtual, es el encargado de controlar el gasto de memoria, y el naranja que se refiere al consumo del procesador por parte de la aplicación.
- Mientras que el punto 1 en las gráficas es el cálculo de ecuaciones, el 2 corresponde al renderizado.
- El algoritmo utilizado para el cálculo ha sido el de Levenberg-Marquardt.

1) Primera prueba

Específicoaciones del ordenador (torre):

- AMD Phenon 4 núcleos a 2,1 GHz
- 2 Gigabytes de memoria RAM



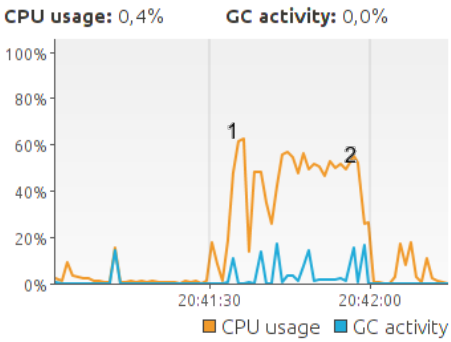
Tiempos:

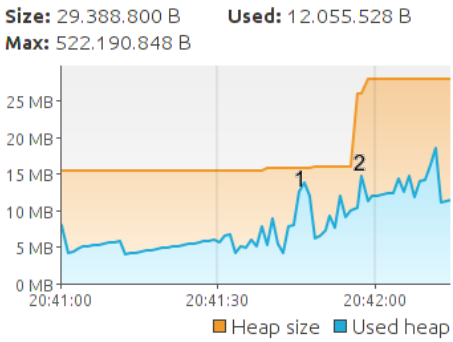
- Resolución: 374 ms
- Renderizado: 1360 ms

2) Segunda prueba

Especificaciones del ordenador (netbook):

- Intel atom 1,6 GHz
- 1 Gigabyte de memoria RAM





Tiempos

- Resolución: 1237 ms
- Renderizado: 4870 ms

Conclusiones:

Como podemos apreciar, el consumo de memoria depende de la plataforma sobre la que se ejecuta ya que en la primera prueba llega a gastar un máximo de 90 Mbytes, mientras que en la segunda no llega a 20. Lo mismo pasa con la memoria de la máquina virtual que varía de 200 Mbytes que alcanza como tope en la primera, a 27 de la segunda. Esta diferencia tan importante se debe a que en el primer caso solamente limpia la memoria, en dos momentos, los picos de las lineas azules en la gráfica del procesador, frente al segundo que hay más para controlar el gasto de RAM.

Un aspecto importante a tener en cuenta es el hecho de que los consumos máximos de memoria y tiempo de procesador son con el renderizado y no con la resolución que es lo primordial del programa, siendo en este segundo caso unos consumos de tiempo y memoria bastante buenos.

En conclusión, tenemos una plataforma que por el momento, se adapta al ordenador sobre el que se está ejecutando y cuyos consumos de memoria real relativa al máximo del ordenador son bajos, menos de un 5%. Además, los tiempos de cálculo para la resolución del ejemplo de 95 ecuaciones son aceptables en el *netbook* y buenos para el ordenador de torre.

Anexo 5. Código

En este anexo hemos introducido las partes del código más importantes a la hora de resolver sistemas los sistemas de ecuaciones.

Parte principal del código de traducción de MathEclipse a eSuite:
Esta sección hace el análisis por caracteres así como el análisis por cadenas.

```
	/**
	 * The input string can't contain comments, use before calling this method
	 * SolverGUI.cleanComments
	 * @param cadena
	 * @return A byte that means: 0 if there are no errors.
	 *                            1 if there is a illegal character error.
	 *                            2 if there is a syntaxes general error.
	 *                            3 If there are two operators followed.
	 *                            4 If there is a dot or comma after a letter.
	 *                            5 If there is not an equal sign
	 *                            6 If there are more open parenthesis than
	 *                              close parenthesis or vice versa
	 *                            7 If a special function is empty
	 *                            8 If behind a number is a variable and not
	 *                               an operator
	 *                            9 If an operator is at the end of the line
	 *                             or does not have a number or a variable
	 *                                 behind
	 *                            10 if a variable contains something
	 *                               different from a letter, number or _
	 *              Also returns a character
	 */
	public GramErr GramCheck(String cadena){

		String aux=new String("");
		char c = Espacio;
		char pc=Espacio;
		int NumC;
		int i=0;
		int j=0;
		boolean Change;
		boolean Equalsign=false;

		while (i!=cadena.length()){
			Change=false;
			c=cadena.charAt(i);
			c=Character.toLowerCase(c);
			NumC=(int) c;


//Comments are introduced with /* and close with */
//Only abcdefghijklmnopqrstuvwxyz1234567890 + - * / , . ^ ( )[ ] = and Tabulator are allowed
//This if is to check if a character is illegal. i.e:@ ; > < : | & { } " # ? ~ % \ $
//Note "_" it's only accepted because it it's translate to Gg
if (((NumC<=57)&(NumC>=48))|((NumC>=97)&(NumC<=122))(NumC==Igual)|(NumC==Plus)|(NumC==Menos)|(NumC==Slash)|(NumC==Espacio)|(NumC==Por)|(NumC==OpenP)|(NumC==CloseP)|(NumC==OpenC)|(NumC==CloseC)|(NumC==Dot)|(NumC==Comma)|(NumC==Elevado)/*|(NumC==Exclamacion)*/|(NumC==Barra)|(NumC==Tab)){/*Accepted characters*/
}else return (new GramErr((byte) 1,c));
```

```
            //At first erase tabs
            if(c==Tab) {
                  Change=true;
            }

//This is the way i decided to make a equation to look like this 0=something
from this
//something=other something
            if (c==Igual){
                  if (Equalsign) return (new GramErr((byte) 2,c));
                  aux+=SubsEqual;
                  Equalsign=true;
                  Change=true;
            }

//As matheclipse can't use "_" we make here a change to make it possible, later
we must
//translate a "Gg" to a _ to show in the results
            if(c==Barra){
                  aux+="Gg";
                  Change=true;
            }

            //To check if there are empty parenthesis
            if(c==CloseP|c==CloseC)
                  if(pc==OpenP|pc==OpenC) return (new GramErr((byte) 7,c));


            //Comma and dots can only be behind a number
            if (((c==Comma)|(c==Dot))&(!IsNumber(pc)))
                  return (new GramErr((byte) 2,c));

            /*//Behind a number can't be a letter
            if(IsLetter(c) & IsNumber(pc))
                  return (new GramErr((byte) 8,c));*/

      //This is to check that there are not two operators followed (except +,-)
            if ((CheckOperator(c))&(CheckOperator(pc))) {
                  if((c==Plus)|(c==Menos)){/*Ignore*/}
                  else
                  return (new GramErr((byte) 3,c));
            }

//This is to check that behind an operator there isn't a ) or a ]
            if ((CheckOperator(pc)) & ((c==CloseP) | (c == CloseC)))

                  return (new GramErr((byte) 9,pc));



//Looks for a missing operator or a dot or comma after a letter
            if ((IsLetter(pc))&((c==Comma)|(c==Dot)))
                  return (new GramErr((byte) 4,c));


//Because of the MathEclipse libraries does not use the commas, only dots, i
make this conversion to
      //allow both possibilities
            if(c==Comma){
                  Change=true;
                  aux+=".";
```

```
            }


            //Write the character only if there were no changes
            if (!Change) aux+=c;


            i++;

            if (i!=cadena.length()){
                  j=i;
            i=SkipSpaces(cadena,i);
            if(j!=i){i--;}
            }
            //Save previous character
            if(c!=Tab)
                  pc=c;
      }
//Checks if there is an operator at the end of the line for example x = 2 / or x
= 2 +
      if(CheckOperator(c))
            return (new GramErr((byte) 9,c));

      aux+=")";
      if (Equalsign){
            VarThisEquation=new VList();
            String aux2=new String("");
            String aux3=new String("");
            String PrevToken=new String(" ");
            StringTokenizer lector = new StringTokenizer(aux,"+/*-()[]{}
^=!",true);
while (lector.hasMoreTokens()){

      aux3=lector.nextToken();

      if(aux3.equals("ln")) aux3="log";//Translation of ln to log, thats if how
ME understand ln

//Now we check if the "variable" is not a special function like Sin or Pi, etc.
//This is to check if it is a number, every number start with a number. I mean
there is no dot
      //at the beginning , i.e:0.5

            if(IsFunction(aux3))
                  {/*Ignore this tokens*/}else{
                        if ((aux3.equals("pi"))|(aux3.equals("e"))|
(IsNumber(aux3.charAt(0)))){
                        if(!IsNumber(aux3)&IsNumber(aux3.charAt(0))) //This
check if the "number" is a
                        return (new GramErr((byte)
8,aux3/*aux3.charAt(0)*/));}// variable that start with
                        else{
                        //a number
                        if ((aux3.equals("+"))|(aux3.equals("-"))|
(aux3.equals("/"))|(aux3.equals("*"))|(aux3.equals(" "))|(aux3.equals("^"))|
(aux3.equals("!"))){//Ignore this tokens}
            else{if ((aux3.equals("("))|(aux3.equals(")"))|(aux3.equals("["))|
                                                      (aux3.equals("]"))){
                  try {
                        aux3=P2C(aux3,PrevToken);}
                        catch (Exception e){return new GramErr((byte) 6);}
                                                      }else
```

```
{if ((PrevToken.equals("["))&((aux3.equals("/"))|(aux3.equals("*"))
(aux3.equals("^")) |(aux3.equals("!")))){return new GramErr((byte) 2);}else{
//First i check the variable
if(!checkVariable(aux3))
                              return (new GramErr((byte) 10,aux3));

      //If it is not an operator, a special function or a number
      //that means that it must be a variable

      VarThisEquation.AddVar(aux3);
            }}}}}
                  //This checks if a special function is empty,unnecessary
                  //if ((PrevToken.equals("["))&(aux3.equals("]"))){
                  //    return (new GramErr((byte) 7));}


                  PrevToken=aux3;
//If the token is a function or pi or e i must translate it to the matheclipse
syntaxes
      if((IsFunction(aux3))|(aux3.equals("e"))|(aux3.equals("pi")))
{aux3=f2F(aux3);}
            aux2+=aux3;
                  }

            String n;
            for(int m=0 ;m < VarThisEquation.getSize(); m++){
                  n=VarThisEquation.getVar(m);
                  Var.addCountVar(n,1);
                  }
            Functions.add(new EqStorer(aux2,VarThisEquation));

            Pila p=new Pila();
      if (p.GetSize()!=-1){//If after reading the whole equation are non paired
parenthesis
                                                //then the size will be
different from -1
                  //p.ErasePila();
                  return (new GramErr((byte) 6));
                  }else{
                  //p.ErasePila();
                  return      (new GramErr((byte) 0));
                  }
            }//Check if the line is empty or is part of a comment line, that
means, comment =true

      else { if(emptyString(aux))
            return (new GramErr((byte) 0));
      else return (new GramErr((byte) 5));
      }

}
```

Métodos de disección de los sistemas de matrices:
El código aquí expuesto esta centrado en la división de los sistemas de ecuaciones en pequeños subsistemas, además se encarga de mandar resolver las ecuaciones cuando ya están bien seccionadas.

```
/**
 * This class is to subdivide the equation system in various equations system.
 * @author pablo salinas
 *
 */
public class PrepareMatrix{

/**
* The variables(columns) are positive int;
* The functions(rows) are negative int;
*/
private AdjacencyList _relations;
private LinkedList<NodoStorer> _Nodes = new LinkedList<NodoStorer>();
private ArrayList<ArrayList<Node>> SCC;
/*The nodes must be created first, so i will create two list of nodes*/
/** Variable nodes list*/
private LinkedList<Node> VarNodes = new LinkedList<Node>();
/**Functions node list*/
private LinkedList<Node> FuncNodes = new LinkedList<Node>();

public PrepareMatrix(){
      //Create the nodes
      for(int i = 1;i < CheckString.Var.getSize() + 1;i++){
            VarNodes.add(new Node(i));
            FuncNodes.add(new Node(-i));
      }
}

/**
* At first i save the initial positions of the variables in the list,
* and then i call compareTo, because i want to know the count of the variables
* to call the tarjan method properly
*/
public void PreNewton(){

      createAdjacencyList();
      //now i will call Tarjan with the AdjacencyList
      //and with the variable with the lowest appearance count
      Collections.sort(this._Nodes);
      int k = Integer.MAX_VALUE;
      Tarjan T;
      ListMatrix Relations;
      LinkedList<PotentialRelationMatrix> PRM = new
LinkedList<PotentialRelationMatrix>();
while(this._Nodes.size()>0){

int i=0;
      for(NodoStorer N:this._Nodes){//Now try with the variables with lowest
appearance
            if(i==0){
                  k = N.getCount();
                  i++;
                  }

            if(N.getCount() <= k){
                  T = new Tarjan();
```

```
                  this._relations.RestartNodes();
                  restartNodes();
                  //Call Tarjan

                  SCC = T.tarjan(/*this._Nodes.get(k-1).getNode()*/N.getNode(),
this._relations);
                  Relations = RelationMatrix(SCC);

            //if(Relations.checkDiagonal())//Store the value
            PRM.add(new PotentialRelationMatrix(Relations,SCC));
                  }
                  //k--;
                  //    k = N.getCount();
                  //i++;
       }



      Collections.sort(PRM);
      Relations = RelationMatrix(PRM.getFirst().SCC);

      PRM.clear();
      updateTerms(SCC);
      RelationMatrix2Newton(Relations,SCC);

      }


}

/**
* After one tarjan iteration, this._Nodes must be updated, erasing the nodes
* that were used to create the Boolean Matrix
* @param SCC
*/
private void  updateTerms(ArrayList<ArrayList<Node>> SCC){
      int aux ;
      Iterator<NodoStorer> it;
      for(ArrayList<Node> Lista: SCC){
            for(Node n:Lista){
                  aux = n.getName();
                  if(aux>0){//Its a column and then a variable

                  it = this._Nodes.listIterator();
            //Search in the list that variable, if founded, then removes it
                  while(it.hasNext()){
                        if(it.next().getNode().getName()==aux)
                              it.remove();
                                 }
                           }
            }}
}


/**

*
* @param SCC
* @return A Matrix with the relations between variables and functions;
* For example the equations x = 2 ; x + y =4 would be this way:
* {{1 , 0};{1 , 1}}
*/
private ListMatrix RelationMatrix(ArrayList<ArrayList<Node>> SCC){
```

```
LinkedList<String> var = new LinkedList<String>();
            LinkedList<Integer> func = new LinkedList<Integer>();
            int variable;
/*This splits the SCC in functions(negatives values) and variables(the positives
ones)*/
for(int i = 0;i < SCC.size(); i++){
      for(int j=0;j<SCC.get(i).size();j++){
            variable = SCC.get(i).get(j).getName();
            if(variable>0) var.add(CheckString.Var.getVar(variable1));
//previously i added 1
            else{
            //if(variable != 0)
            func.add(Math.abs(variable)-1);//previously i added 1
            }

      }
}

ListMatrix result = new ListMatrix(var.size());

Iterator<Integer> it = func.listIterator();
EqStorer function;
int row = 0;
while(it.hasNext()){
      int position = it.next();
      function = CheckString.Functions.get(position);

      Iterator<String> it2 = var.listIterator();
      String nombre;
      int col = 0;
      while(it2.hasNext()){
            nombre = it2.next();
      for(int j = 0;j<function.aux.size();j++)
            if(function.aux.get(j).GetVar().equalsIgnoreCase(nombre))
                  result.setValue(row, col, (byte) 1);

      col++;
      }

      row++;
      }

return result;
}



/**
* Creates an adjacency List of all the variables to use in Tarjan
*/
private void createAdjacencyList(){
this._relations = new AdjacencyList();
//int fila=-1,col=1;
int col = -1,fila;
EqStorer Eqaux;
Node Naux, nodo;
Iterator<EqStorer> it2=CheckString.Functions.iterator();
NodoNameCount NodoAux;
while(it2.hasNext()){
      Eqaux=it2.next();
      Naux = NodeOfTheList(FuncNodes , col);
```

```
      if(Naux.name != 0){
      Iterator<DerivEquation> it3 = Eqaux.aux.listIterator();
      fila = 0;
      while(it3.hasNext()){
            NodoAux = getVarPosition(it3.next().GetVar());
            nodo = NodeOfTheList(VarNodes , NodoAux.getName()+1);
            if(nodo.name != 0){
            //Store the from node
            //First i check if the node is already in the list
            boolean found = false;
            for(int k = 0;k<this._Nodes.size();k++){

      if(this._Nodes.get(k).getNode().getName()==nodo.getName())
                  found = true;
                  }
      if(!found)//add node
      this._Nodes.add(new NodoStorer(nodo,NodoAux.getCount()));


      _relations.addEdge(Naux,nodo , 1);
      _relations.addEdge(nodo, Naux, 1);
            fila++;
      }}

      col--;
      }
   }
}



/**
* Makes Jacobians matrix from the relation matrix and calls a method for solving
*them
* @param relations
*/
private void RelationMatrix2Newton(ListMatrix relations,
ArrayList<ArrayList<Node>> scc){
ArrayList<Byte> aux;
LinkedList<Integer> Variables= new LinkedList<Integer>();
int k = 0;
//At first i check the equations with one variable for solving them
//boolean found = false;
//Now i analyze the equations

while(relations.size()>0){
      Variables.clear();
      k = 0;

      aux = relations.getRow(0);
      Variables = ListMatrix.AnalizeRow(Variables, aux);

      //At first i check if we have a equation with one variable
      if(ListMatrix.numberVariables(aux) == 1){
            MakeJacobian(Variables,scc);
            relations.refresh(Variables);
            }else{
            while(ListMatrix.numberVariables(aux) !=  k + 1){
            k++;
            aux = relations.OperateRow(aux, relations.getRow(k));
            Variables = ListMatrix.AnalizeRow(Variables, aux);
            }
```

```
      MakeJacobian(Variables,scc);
      relations.refresh(Variables);
      }
      }

}

/**
*
* Makes two lists one with the functions and one with the variables. And then it
* calls to the
* solvers methods.
* @param Variables
* @param scc
*/
private void MakeJacobian(LinkedList<Integer>
Variables,ArrayList<ArrayList<Node>> scc){
Collections.sort(Variables);
Iterator<Integer> it = Variables.listIterator();
Iterator<ArrayList<Node>> nodo = scc.listIterator();
Iterator<Node> n;
LinkedList<Integer> Functions = new LinkedList<Integer>();
LinkedList<Integer> Vars = new LinkedList<Integer>();
int i = 0, size = Variables.size(), aux, j = 0;

int itvar=it.next();
while(nodo.hasNext()){
      n = nodo.next().listIterator();
      while(n.hasNext()){
            aux = n.next().name;

            if(aux < 0 & i<size) {
            Functions.add(Math.abs(aux)-1);//Add function position of the
CheckString.Functions
                  n.remove();
                  i++;
            }else{
                  if(j==itvar){
                        if(it.hasNext())
                              itvar=it.next();
                        Vars.add(aux-1);
                        n.remove();
                  }

                  j++;
            }
      }}
//Now i have two lists with the numbers of the functions and the variables

/*----------------------------------------SOLVER CALL----------------------*/
solver.LaunchOperations LO = new solver.LaunchOperations(Functions, Vars);
solver.OperationCounter OC = new solver.OperationCounter();
Thread N = new Thread(OC,"Clock");
Thread m = new Thread(LO,"Operations");
  N.start();
  m.start();
  while(m.isAlive())
            if(!N.isAlive())
                  m.interrupt();
```

```
}
/**
* Erase from the CheckString.Var the variables stored in
CheckString.OneEquationVar.
* Erase the solved variables from the checkString.Functions.
* Solves the equations that now have one variable.
*/

public static void PreTarjan(){

Iterator<VString> it;
boolean found = false;
VString auxV;

/*--Search equations with one variable, for solving them--*/
int pos = 0;

while(pos != -1){
      found = false;

      pos = searchOneVariableFunction();


      if(pos != -1){

/*--------ONE VARIABLE SOLVER CALL-------------------*/

solver.LaunchOperations LO = new
solver.LaunchOperations(CheckString.Functions.get(pos).getEquation()
,CheckString.Functions.get(pos).aux.get(0).GetVar());
      solver.OperationCounter OC = new solver.OperationCounter();
            Thread n = new Thread(OC,"Clock");
            Thread m = new Thread(LO,"Operations");
              n.start();
              m.start();
              while(m.isAlive())
                  if(!n.isAlive())
                        m.interrupt();



            //Change the function from Var to OneEquationVar
            it = CheckString.Var.Variables.listIterator();
            while(it.hasNext() & !found){
                  auxV = it.next();
            if(auxV.getVar().equalsIgnoreCase

      (CheckString.Functions.get(pos).aux.get(0).GetVar())){
                  CheckString.OneEquationVar.add(auxV);
                        it.remove();
                        found = true;
                  }
            }
updateFunctions(CheckString.Functions.get(pos).aux.get(0).GetVar());
      //Store the function
CheckString.FunctionsSolved.add(CheckString.Functions.get(pos));
      //Remove the function from functions
      CheckString.Functions.remove(pos);
      }
      }

}
```

Código para llamar a los métodos de resolución de ecuaciones:

Con este apartado nos comunicamos con las librerías NonLinear Optimization Java Package, creamos la función objetivo así como la Jacobiana, el gradiente y la Hessiana, solamente si son necesarias.

```
/**
* Creates the necessary gradient, hessian or Jacobian to call the solver methods
* @author Pablo Salinas
* This class must be static.
*/
public class PrepareUncmin{
/**
* Functions evaluated
*/
public static vector Fx = new vector();
/**
* Variables evaluated
*/
public static vector Xk = new vector();
/**
* Analytic Jacobian
*/
public static String[][] Jacobian;

/**
* Each element of the List is the Hessian of an equation;
* Example:
* Two functions:
* Cos(x)-1/2=0 ;x^2*y^3 = 25;
* Then this functions stores this:
* at the first position -> { {-cos(x) ,     0},
*                            {0    ,      0}}
* at the second position ->{ {2*Y^3 ,   6*x*y^2},
*                            {6*x*y^2 , 6*x^2*y}}
*/
public static LinkedList<String[][]> Hessians;

/**
*This constructor creates the f, that is a List with all the 1/2*Fx^2 values;
*The gradient values with is a Matrix(made with list) that contains the
*derivatives of 1/2*Fx^2
*The hessian values that contains all the diagonal and inferior partial second
*derivatives of the values of Fx^2
*@param Fx
*@param Xk
*/
public PrepareUncmin(LinkedList<Integer> Functions, LinkedList<Integer>
Variables){

clear();

Xk.Xvector(Variables);
Fx.Fvector(Functions);
//Hessians = new LinkedList<String[][]>();

CreateJacobianAndHessian();

}
/**
```

```
* This method is to use the Levenberg-marquard that does not need the hessian
* @param Equation
* @param Var
* @param bol
*/
public PrepareUncmin(LinkedList<Integer> Functions, LinkedList<Integer>
Variables,boolean b){

clear();

Xk.Xvector(Variables);
Fx.Fvector(Functions);

//Create Jacobian
/*Jacobian Matrix*/
Jacobian = matriz.JacobianLU(Xk, Fx);

}

/**
* This method is for the CheckString.OneEquationVar
* @param Equation
* @param Var
*/
public PrepareUncmin(String Equation,String Var){

clear();

Fx.vector.add(new nodo(Equation));//Fx stores the equation

Xk.vector.add(new nodo(Var));//Xk stores the variable
//Now i have to set the initial value of that variable
Xk.vector.get(0).SetValue(Config.DefaultInitialValue);
for(InitVal IV:Config.InitValue){
      if(Var.equalsIgnoreCase(IV.getVariable()))
            Xk.vector.get(0).SetValue(IV.getValue());
}
CreateJacobianAndHessian();

}

/**
* This method is to use the Levenberg-marquard that does not need the hessian
* @param Equation
* @param Var
* @param bol
*/
public PrepareUncmin(String Equation,String Var,boolean bol){

clear();

Fx.vector.add(new nodo(Equation));//Fx stores the equation

Xk.vector.add(new nodo(Var));//Xk stores the variable
//Now i have to set the initial value of that variable
Xk.vector.get(0).SetValue(Config.DefaultInitialValue);
for(InitVal IV:Config.InitValue){
      if(Var.equalsIgnoreCase(IV.getVariable()))
      Xk.vector.get(0).SetValue(IV.getValue());
}
//CreateJacobianAndHessian();
//Create Jacobian
```

```
/*Jacobian Matrix*/
Jacobian = matriz.JacobianLU(Xk, Fx);
}

/**
* Creates from Xk and Fx the analytic Jacobian and Hessian
*/
private void CreateJacobianAndHessian(){
Hessians = new LinkedList<String[][]>();


/*Jacobian Matrix*/
Jacobian = matriz.JacobianLU(Xk, Fx);

/*List with arrays of the second and partial derivatives*/
/*Each element of the List is the Hessian of an equation*/
String[][] Saux;
int i = 0;
for(int row = 0 ; row < Xk.getSize(); row++){

Saux = new String[Xk.getSize()][Xk.getSize()];

      for(int col = 0; col < Xk.getSize(); col++){
            i = 0;
            for(nodo n:Xk.vector){
            Saux[col][i] = evaluation.DiffAndEvaluator.diff(Jacobian[row][col],
n.GetCadena());
                  i++;
            }
      }
      Hessians.add(Saux);
}
}

/*
*This method sets this values and calls a method for solving the system equation
*
*@param  n         The number of arguments of the function to minimize
*@param  x         The initial estimate of the minimum point
*@param minclass   A class that implements the Uncmin_methods
*                  interface. Basically a method that evaluates
*                  the function the gradient and the Hesian
*@param typsiz     Typical size for each component of x
*@param fscale     Estimate of the scale of the objective function
*@param method     Algorithm to use to solve the minimization problem
*                    = 1  line search
*                    = 2  double dogleg
*                    = 3  More-Hebdon
*@param iexp       = 1 if the optimization function f_to_minimize
*                  is expensive to evaluate, = 0 otherwise.  If iexp = 1,
*                  then the Hessian will be evaluated by secant update
*                  rather than analytically or by finite differences.
*@param  msg       Message to inhibit certain automatic checks and output
*@param  ndigit    Number of good digits in the minimization function
*@param  itnlim    Maximum number of allowable iterations
*@param  iagflg    = 0 if an analytic gradient is not supplied
*@param  iahflg    = 0 if an analytic Hessian is not supplied
*@param  dlt       Trust region radius
*@param  gradtl    Tolerance at which the gradient is considered close enough
*                  to zero to terminate the algorithm
*@param  stepmx    Maximum allowable step size
*@param  steptl    Relative step size at which successive iterates
```

```
*                  are considered close enough to terminate the algorithm
*@param xpls       The final estimate of the minimum point
*@param fpls       The value of f_to_minimize at xpls
*@param gpls       The gradient at the local minimum xpls
*@param itrmcd     Termination code
*                      ITRMCD =  0:  Optimal solution found
*                      ITRMCD =  1:  Terminated with gradient small,
*                                    X is probably optimal
*                      ITRMCD =  2:  Terminated with stepsize small,
*                                    X is probably optimal
*                      ITRMCD =  3:  Lower point cannot be found,
*                                    X is probably optimal
*                      ITRMCD =  4:  Iteration limit (150) exceeded
*                      ITRMCD =  5:  Too many large steps,
*                                    function may be unbounded
*@param a          Workspace for the Hessian (or its estimate)
*                  and its Cholesky decomposition
*@param udiag      Workspace for the diagonal of the Hessian
*/
/** Solves the system of equations.
* @param EvaluationMethod   Changes the method for solving the equation method;
* 1 = Gauss-Newton
* 2 = Double Dogleg
* 3 = Hebden-More
*/
public static void Solve(int EvaluationMethod){

Uncmin_methods UM = new Uncmin_methods();
int tam = Fx.getSize();
double[] xpls = new double[tam+1];
double[] fpls = new double[tam+1];
double[] gpls = new double[tam+1];
double[][] a = new double[tam+1][tam+1];
double[] udiag = new double[tam+1];
double typsiz[] = new double[tam+1];
double fscale[] = new double[2];
int method[] = new int[2];//1 = Line-search, 2 = double dogleg,3 = More-Hebdon
int iexp[]= new int[2];
int msg[]= new int[2];
int[] itrmcd = new int[2];
int ndigit[]= new int[2];//Number of good digits in the minimization function
int itnlim[]= new int[2];//Number of maximun iterations
int iagflg[]= new int[2];//=0 analytic gradient, not supplied
int iahflg[]= new int[2];//=0 analytic hessian not supplied,
double dlt[]= new double[2];//dlt= trust region radius
double gradtl[] = new double[2];//tolerance at witch the gradient is considered
close enough to zero to finish
double stepmx[] = new double[2];//Maximum allowable stepsize
double steptl[] = new double[2];//Relative step size at wich is considered to
finish

//Set Values

for (int i = 1; i <= tam; i++) {

                typsiz[i] = 1.0;

            }

            fscale[1] = 1.0;

if( EvaluationMethod <= 3 & EvaluationMethod > 0)
```

```
method[1] =  EvaluationMethod;//1 line-search; 2 double dogleg, 3 hebden-more
      else
         method[1] = 2;

      iexp[1] = 0;
      msg[1] = 80;      //40 means no gradient and Hessian check, 80 the same
plus less comments
ndigit[1] = -1;
itnlim[1] = Config.MaxNumberOfIteration;
iagflg[1] = 1;
iahflg[1] = 1;
dlt[1] = Config.TrustRegionRadius;//-1 means that the first time will use the
Newton step length
// double epsm = Config.epsilon;
gradtl[1] = Config.GradientPrecision;
stepmx[1] = Config.MaxJump;
steptl[1] = Config.Precision;//Math.sqrt(epsm);// <-- This was the original

try{

if(Config.IterationAntiMinimum <= 0)
Uncmin_f77.optif9_f77(tam,Xk.Vector2Dogleg(),UM,typsiz,fscale,method,iexp,msg,
ndigit,itnlim,iagflg,iahflg,dlt,gradtl,stepmx,steptl,xpls,fpls,gpls,itrmcd,a,udi
ag);
else
      {     int MaxIterations = 0;
            vector Xkaux = Xk;
            double max = 0;
            do{


Uncmin_f77.optif9_f77(tam,Xkaux.Vector2Dogleg(),UM,typsiz,fscale,method,iexp,msg
,ndigit,itnlim,iagflg,iahflg,dlt,gradtl,stepmx,steptl,xpls,fpls,gpls,itrmcd,a,ud
iag);

      //check if the point is a good result
      if(itrmcd[1] == 1 | itrmcd[1] == 2){
      max = 0;
      for(double d:fpls)
      if(Math.abs(d) > Math.abs(max)) max = d;

      if (Math.abs(max) > Math.sqrt(Config.Precision)){
      Newton N = new Newton(Fx,Xk);
      Xkaux = N.Newtonsolver(Jacobian, Config.IterationAntiMinimum);

      }else{itrmcd[1] = 0;}
      }

            MaxIterations++;
            if(MaxIterations >= Config.IterationAntiMinimum)
               itrmcd[1] = 0;

            }while(itrmcd[1] == 1 | itrmcd[1] == 2 );

          }
         }catch(Exception e){
            e.printStackTrace();
          }finally{
                  clear();
          }

}
```

```java
/**
* Solves by the Levenberg-Marquard method
*/
public static void LMSolve(){
Lmder_fcn Lder = new Lmder_fcn();
int tam = Fx.getSize();
double x[] = new double[tam+1];
x=Xk.Vector2Dogleg();
double fvec[] = new double[tam+1];
double fjac[][] = new double[tam+1][tam+1];
double tol = Config.Precision;//Math.sqrt(Config.epsilon);
int info[] = new int[2];//This is a return value
int ipvt[] = new int[tam+1];//See Minpack_f77

try{  //Call the method
      Minpack_f77.lmder1_f77(Lder, tam, tam, x, fvec, fjac, tol, info, ipvt);
}catch(Exception e){e.printStackTrace();}
finally{clear();}
}
```

Clase de enlace con MathEclipse:
Este código lo utilizamos de enlace con MathEclipse, para que trabajar con el sea más fácil, tenemos por ejemplo métodos para derivar, para evaluar, para hacer código MathMl,...

```
/**
* This class is to communicate with matheclipse.
* You can evaluate a function, derivate it, get the variables, etc.
* @author Pablo Salinas
*
*/
public class DiffAndEvaluator {
/**
* true if there is an error while evaluating a function
*/
public static boolean SymbolicEvaluatorError=false;
/**
* If time limit for operations is reached then this will be true
*/
public static boolean TimeLimitExceeded = false;
/**
* This strings saves the equation which haven been the source of the evaluation
* error
*/
public static String StringErrorEvaluating;
/**
* Matheclipse class to communicate with
*/
static EvalUtilities util = new EvalUtilities();

/**
* This must be the first thing to start if you want matheclipse to work.
* But you only have to call it once.
*/
public static void PrepareME(){
F.initSymbols(null);
}
/**
* Introduces a String like this:A=4;B=5;C=8 into the matheclipse engine.
* @param Input string, A=4;B=5;C=8
*/
public static void IntroduceValues(String input) {
try {
      String[] values = input.split(";");
      for(String S:values)
      util.evaluate(S);
      // util.evaluate(s);

} catch (final Exception e) {
e.printStackTrace();
}
}
/**
* Evaluates a equation as a string in a point,
* the values must be introduced before using this method
*
* @param A string like this A*x/8-(1+c)
* @return A double of the equation evaluated
*/
public static double Evaluate(String s){
if(s!=null){
try {return  Double.parseDouble(util.evaluate(s).fullFormString());
```

```
}catch (DivisionByZero D){
      D.printStackTrace();
      StringErrorEvaluating = s;
      Config.ErrorFound = true;
      throw new RuntimeException();
      }
catch (NumberFormatException e) {
      e.printStackTrace();
      StringErrorEvaluating = s;
      Config.ErrorFound = true;
      throw new RuntimeException();

      }catch(EvaluationInterruptedException E){
      TimeLimitExceeded = true;
      E.printStackTrace();
      Config.ErrorFound = true;
      throw new RuntimeException();
      }catch(SyntaxError e){
      Config.ErrorFound = true;
//    throw new RuntimeException();
      return Double.NaN;
      }
catch(Exception e){
      e.printStackTrace();
      Config.ErrorFound = true;
//    throw new RuntimeException();
      return Double.NaN;
} }else {return Double.NaN;}
}



/**
 *
 * @param equation
 * @param var
 * @return The differentiation of the equation respect the variable
 */
public static String diff(String equation,String var) {
try {
if(equation.contains(var)){
            String input = "D["+equation+","+var+"]";
            return  util.evaluate(input).fullFormString();
      }else
            return "0";
} catch (final Exception e) {
e.printStackTrace();
}
return null;
}
/**
* Returns the variables of an equation
* @param equation
* @return Array of String with the variables
*/
public static String[] getVariables(String equation){

try {
      String input = "Variables["+equation+"]";
      input =  util.evaluate(input).fullFormString();
      input = input.substring(5, input.length()-1);
      input = input.replace(" ", "");
```

```
return input.split(",");
} catch (final Exception e) {
      e.printStackTrace();
}
return null;
}

/**
* A method to clean the values of the variables in the matheclipse engine
*/
public static void PurgeVar(){
@SuppressWarnings("unused")
IExpr result;
try {

for(VString VS : CheckString.Var.Variables)
      result = util.evaluate("ClearAll["+VS.getVar()+"]");

for(VString S: CheckString.OneEquationVar)
      result = util.evaluate("ClearAll["+S.getVar()+"]");

} catch (final Exception e) {
      e.printStackTrace();
      }
finally {
ComputerThreads.terminate();
}

}
/**
* Clear the value of the variable introduced
* @param var
*/
public static void PurgeVar(String var){
@SuppressWarnings("unused")
IExpr result;
try {
      result = util.evaluate("ClearAll["+var+"]");
} catch (Exception e) {
       e.printStackTrace();
}
}

/**
*This method insert the string in the matheclipse engine and returns the
*result.
* This method has control time so this is the method to use with the Mathematics
*section.
* @param In
* @returnThe result as a string
*/
public static String SymbolicEvaluator(String In){
      Counter c = new Counter();
      Thread n = new Thread(c,"Clock");
      CalculateThread t =new CalculateThread(In);
      Thread m = new Thread(t,"Evaluator");
      n.start();
      m.start();
      while(m.isAlive())
            if(!n.isAlive())
                  m.interrupt();
```

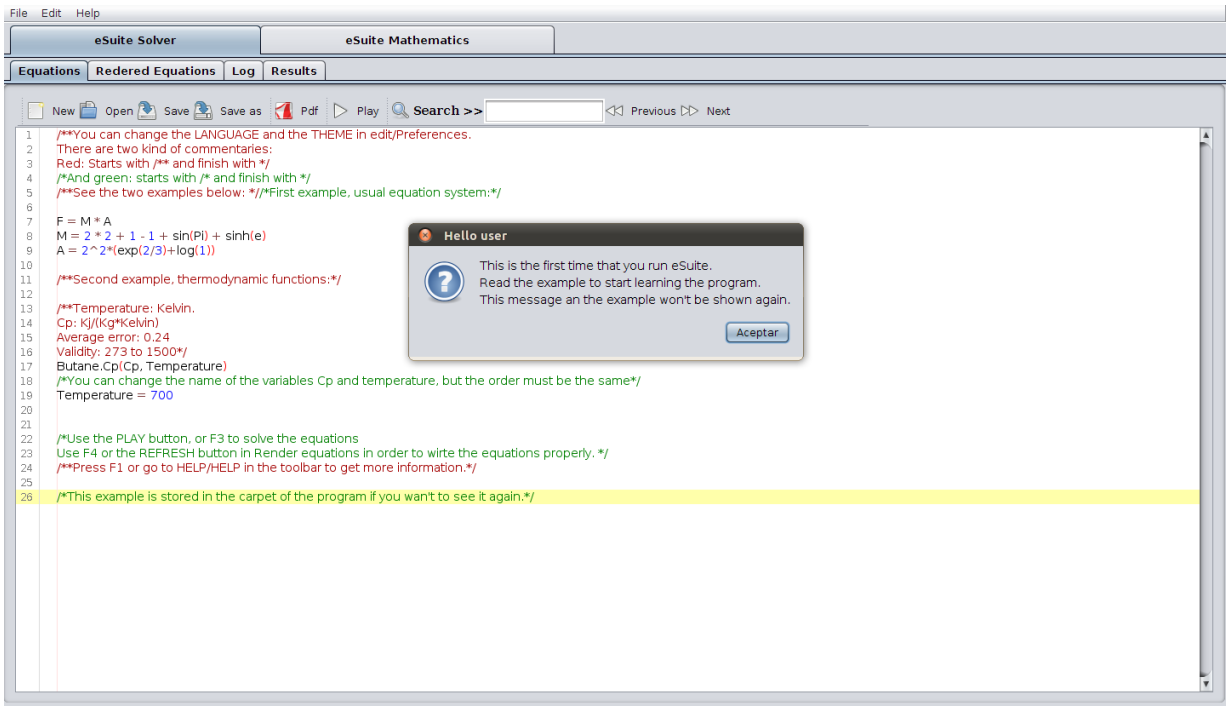
```
      return CalculateThread.getOutput();
  }

/**
* This method transform a string to mathML to later be used as a image to have a
* more elegant output
* @param equation
* @return The string wrote in mathML syntaxes
*/
public static String MathML(String cadena){
        EvalEngine EVAL=new EvalEngine();

/*-----------------KNOWN ERRORS ---------------*/
/*1Âº Sometimes appears something like this 1/10^(-(-2)),
*  it's true but it is not pretty.
*  This can be workaround  using / instead * :BAD->5,67*10^(-8) SOLUTION
->5,67/10^(8)*/

/*2Âº  * appears before a division and after a division.
* This can be workarounded  using / instead * :BAD->5,67*10^(-8) SOLUTION
->5,67/10^(8)*/
/*3Âº using x^-1 gives an error, but using x^(-1) doesn't,
 * the next version of matheclipse will have this issue fixed/
*/

//As * are ignored i translate it to Xx to later translate it back to *
      cadena=cadena.replace("*(", "(");
      cadena=cadena.replace(")*", ")");
      cadena=cadena.replace("*", " Xx ");
      MathMLUtilities math=new MathMLUtilities(EVAL,false);

      final StringBufferWriter buf = new StringBufferWriter();

      math.toMathML(cadena, buf);
      String aux=new String(buf.toString());

      /*--------Modifications to the MathML code--------*/
      //Now i translate back to *
      //aux2=aux2.replace("Dot(*", "*(");
      aux=aux.replace("Xx", "*");
      //Sometimes, the word Dot appears...so i erase it,
      //because upper and lower case cannot appear together
      aux=aux.replace("Dot", "");
      //To fix " (* " i erase the *
      aux=aux.replace("(</mo><mrow><mi>*", "(</mo><mrow><mi>");

return aux;
  }
}
```

Anexo 6. Contenido de la ayuda

Como uno de los objetivos imprescindibles del proyecto era crear un buen sistema de ayuda que convirtiera al programa en una herramienta realmente útil y asequible, hemos realizado diferentes tipos de ayuda que pasamos a explicar a continuación.

1) Primera ejecución

En la primera ejecución del programa se muestra una ventana emergente saludando al usuario y se facilita el primer acercamiento al contexto con un ejemplo que contiene información relacionada con el programa, como por ejemplo, los pasos a seguir para poder cambiar el idioma y el tema. Además, se adjuntan dos ejemplos concretos de la aplicación del programa: uno sobre ecuaciones y otro con el uso de propiedades termodinámicas. Aunque esta ventana emergente en inglés solo aparece la primera vez que se inicia el programa -recordemos que el programa suele detectar el ordenador que lo utiliza- el ejemplo queda registrado en el programa para que el usuario pueda consultarlo cuantas veces quiera. Esta información llega al usuario incluida en el último ejemplo de esta ventana emergente, como se aprecia en la foto que aparece a continuación.



2) Menú de ayuda

A continuación vamos a incluir la ayuda creada para el programa. Aparece en el mismo orden que en el programa y el titulo será el apartado que tiene dentro de la ayuda.

Solver:

eSuite Solver resuelve sistemas de ecuaciones no lineales utilizando algoritmos de región de confianza. Sus características fundamentales son:

1.- Las ecuaciones pueden escribirse sin ningún orden.

2.- Los comentarios empiezan con /* o con /** y terminan con */

3.- Caracteres permitidos: a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0 + - * / , . ^ ( ) [ ] = _ y el tabulador.

4.- No importa usar mayúsculas o minúsculas, así que x y X es lo mismo.

5.- Las variables deben empezar con una letra pero luego pueden contener más signos, por ejemplo, está permitido x23 pero no 3x2.

6.- Debe haber una única ecuación por línea.

7.- La única limitación para el número de ecuaciones es la memoria RAM o la complejidad de estas.

8.- El campo de búsqueda funciona en una sola dirección partiendo de la posición actual en el texto.

9.- El archivo guardado es un fichero en texto plano que puede ser abierto con el bloc de notas, con Gedit o similar.

Solver/FAQ:

1) Quiero introducir un logaritmo distinto al natural pero no me lo permite. ¿Qué hago?

En eSuite hay dos maneras de introducir algoritmos Ln o Log, pero ambas son iguales, Así que si quieres introducir uno distinto debes usar esta fórmula:

Log[a,m]=ln[m]/ln[a]

Por ejemplo, si quieres hacer un algoritmo con base 10:

Log(x*y)/Log(10).

2) El programa dice que hay un error al evaluar. He comprobado todas las ecuaciones pero están bien. ¿Cuál es el problema?

Tal vez al evaluar las ecuaciones en distintas iteraciones la Jacobiana llega a un punto singular, por tanto no sería culpa tuya. Puedes usar un algoritmo diferente o cambiar los valores iniciales.

3) En preferencias, modifico la fuente del texto pero solamente cambia en la ventana de ecuaciones.

Es algo normal, porque es necesario que en la ventana de resultados y en log haya una fuente monoespaciada para que los resultados salgan escritos en columnas.

4) ¿Por qué el tiempo total de cálculo es superior al que he puesto en preferencias como máximo?

Eso es porque el tiempo de preferencias en el tiempo máximo para el cálculo de un sistema. Si en tus ecuaciones hay más de un sistema de ecuaciones puede pasar que el tiempo total sea superior.

5) ¿Qué son los residuales y por qué deberían importarme?

Los residuales son el resultado de introducir los resultados finales en las ecuaciones. Por tanto, un valor elevado significa que los resultados no son buenos.

6) Muy bien, me has convencido. ¿Qué quieres decir con elevados?

Los residuales elevados dependen de la situación y la precisión que se requiera pero valores por encima de 0.01 son malos.

7) ¿Por qué *software* libre?

Porque con *software* libre el coste de desarrollo es mucho más bajo y mucho más ético.

8) ¿Puedo ayudar?

Toda la documentación interna está en inglés para que todo el mundo pueda colaborar, así que sí, ¡bienvenido!

Solver/Errores:

Tenemos varios tipos de errores:

1- Carácter ilegal encontrado: Solamente tienes que cambiar el carácter especificado.

2- Carácter inesperado. Esto puede deberse a dos signos igual en una ecuación. Elimínalo y solucionado.

3- Dos o más operadores contiguos: / * ^ no pueden ir seguidos entre ellos.

4- Punto o coma tras una letra: Las variables solamente pueden contener letras, números y _.

5- No hay signo igual: ¿Cómo te lo has podido dejar?

6- Hay más paréntesis o corchetes de apertura que de cierre. Busca en la ecuación con ayuda del remarcador de paréntesis o corchetes para encontrar el fallo.

7- Si una función trigonométrica, logarítmica, exponencial o hiperbólica esta vacía.

I.E. Sin( ) = 5.

8- Error si una variable empieza con número: Bórralo.

9- Un operador está vacío. Por ejemplo: x = 2/ or x/(= 3. Solución: Bórralo.

10- Una variable contiene algo diferente a un número, letra o _. Elimina el carácter.

11- Se ha sobrepasado el tiempo de cálculo por operaciones que se han interrumpido: Esto puede deberse a muchas cosas, pero generalmente es debido a que algún fallo hace que el algoritmo se introduzca en un bucle infinito. Lo mejor es cambiar el punto inicial o cambiar de algoritmo.

12- Error al evaluar: Esto quiere decir que el programa ha intentado evaluar una función y no ha obtenido un resultado por ejemplo 1/0. Prueba otro algoritmo, distinto punto inicial o las dos cosas.

13- Residuales elevados. Mira Log o FAQ para saber más sobre esto.

14- El número de ecuaciones y variables difiere. Mira el Log para saber el número de apariciones de cada variable.

15- Propiedad no encontrada: has llamado a una función termodinámica Sustancia.Propiedad(var1,var2) pero o la propiedad no existe o la has escrito mal.

16- Error inesperado: Este problema surge cuando el programa no es capaz de rescatar el origen del

error, mas sí detecta algún problema. Prueba las anteriores recomendaciones o reinicia el programa.

Solver/Log:

Esta sección muestra cómo fue el cálculo y el número de apariciones de las variables.

La primera sección, las cuentas, es útil cuando encontramos el error en que el número de variables y ecuaciones difiere, porque nos muestra si hay mal escrita alguna variable.

La sección de residuales es importante para saber el nivel de confianza de los resultados. Unos residuales elevados pueden indicar tres cosas:

1- El algoritmo ha caído en un mínimo. Prueba cambiando el algoritmo o el punto inicial. También puedes probar Double Dogleg, Line-Search o Hebden-More con el sistema de antimínimos activado. Consulta la sección Preferencias para más información.

2- El algoritmo no ha alcanzado una solución. Esto puede deberse a que las ecuaciones son muy complejas o a que el Jacobiano ha alcanzado un punto singular. Puedes probar cambiando el algoritmo o el punto inicial de nuevo.

3- Si el algoritmo queda interrumpido a mitad porque el tiempo de cálculo supera al permitido, entonces la solución estará a mitad y por tanto no será fiable.

Solver/Resultados:

Desde aquí puedes ver los resultados de las ecuaciones. Las variables se muestran por orden alfabético y sus valores están escritos en función de su valor:

Números pequeños: Si el valor absoluto de un número se encuentra entre 1 e-5, 0 el resultado se muestra así: 1.23456E-7 .

Números normales: Esto es para números que no son ni pequeños ni grandes. La codificación es así: 12345.1234567.

Números grandes: Para números mayores que 100000. El resultado se muestra así12345.12345E0.

Solver/Renderizado:

En esta sección las ecuaciones quedan dibujadas de una manera más próxima a como serían, si fuesen escritas en papel.

Para hacer esto eSuite utiliza la sintaxis MathML y jEuclid para mostrar el resultado.

Para convertir una ecuación a sintaxis MathML eSuite llama a MathEclipse para que haga la conversión. Sin embargo, MathEclipse borra los signos * así que hay que convertir * a una letra y luego deshacer el cambio. Por eso incorporamos el botón de ayuda que muestra algunos problemas de este cambio.

Solver/Ecuaciones:

Características:

1- Caracteres permitidos: a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0 + - * / , . ^ ( ) [ ] = _ y tabulador.

2- Coloreado de texto, remarcado de paréntesis.

3- Una ecuación por línea. Cada ecuación debe empezar y acabar en la misma línea.

4- Dos tipos de comentarios. Verdes con /* y rojos con /** ambos son multilínea y ambos cierran igual, con */.

5- No hay límite de filas o columnas. Sin embargo, al exportar a pdf o imprimir, las lineas que sean muy largas quedarán divididas en dos o más líneas, según convenga.

6- Puedes usar Deshacer y Rehacer desde el Menú contextual del botón derecho o desde el menú en Editar. No tienen una memoria específica para cada cambio, sino que recuerdan los cambios en bloques.

7- Las funciones de Cortar, Copiar y Pegar utilizan el portapapeles del sistema. Funcionan desde el Menú contextual del botón derecho o desde la barra de tareas en Editar, también funcionan los accesos directos desde el teclado.

8- eSuite puede trabajar con estas funciones: Sin, Cos, Tan, ArcSin, ArcTan, ArcCos, Sinh, Cosh, Tanh, Log, Ln (que es igual a log), Exp. Además, incorpora dos constantes: e y Pi.

Solver/Preferencias:

Este menú está dividido en tres secciones:

1) General

1- En trigonometría podremos escoger entre radianes y grados. Esto solamente afecta a eSuite Solver, no a Mathematics.

2- Fuente: Se puede modificar la fuente de ecuaciones y el tamaño para todas las ventanas.

3- Lenguaje: Puede seleccionarse el lenguaje de la aplicación.

4- Temas: Hay cuatro temas. El programa ha sido diseñado para trabajar con Nimbus, sin embargo, el tema que mejor se adapta a los sistemas operativos es System.

2) Ecuaciones

1- Precisión: Es un criterio de parada para los algoritmos. Valores más pequeños implicarán más tiempo de cálculo. La fórmula es: si precisión < ((Xnew-Xold)/(Xnew)) entonces parar.

2- Número de iteraciones: Si la solución no puede ser alcanzada, entonces el algoritmo parará al pasar un número de iteraciones.

3- Tiempo máximo para operaciones: Tiempo tras el cual las operaciones serán interrumpidas.

4- Métodos: Consulta la sección Algoritmos para saber más sobre esto.

5- Salto máximo: Double Dogleg, Hebden-More y Line-Search poseen otro criterio de parada que es el salto máximo entre dos iteraciones para evitar que el algoritmo de saltos muy elevados. Por defecto selección automática = 0.
6- Precisión del gradiente: Criterio de parada de Double Dogle, Hebden-More y Line-Search. El algoritmo parará si está muy cerca de un máximo o mínimo donde el gradiente vale cero.
7- Evitar mínimos: Si el algoritmo alcanza un mínimo parará en Double-Dogleg, Hebden-More y Line-Search puede pedir que se hagan iteraciones de Newton para escapar de ese mínimo. Esto puede ser inestable. Por defecto desactivado = 0;
8- Región de confianza: Para saber esto mire la sección de algoritmo. Por defecto -1 que significa que el programa escogerá por nosotros.

3) Matemáticas

1- Tiempo máximo: Tras un tiempo de cálculo, el motor de matemáticas será interrumpido.
2- Valor Máximo/Mínimo: Puede seleccionar el valor máximo y mínimo de la sección de gráficas.

Solver/Fórmulas termodinámicas:

Desde este menú se pueden añadir fórmulas termodinámicas a sus ecuaciones o incluir las suyas propias en la base de datos del programa.

Recomendaciones:

1- Usar nombres de variables en vez de letras. Por ejemplo: Temperatura en vez de T.
2- Las fórmulas deben añadirse como ecuaciones y con la sintaxis del programa. Por ejemplo: Presión * Volumen = mol * R * Temperatura.
3- Usar el área de información para incluir los límites de la fórmula, la precisión, etc.
4- Las secciones: Sustancia, Propiedad y Fórmula son obligatorias.
5- Se pueden eliminar sustancias y materiales por lo que hay que proceder con cautela.
6- Si se van a calcular la temperatura o presión de una formula de entalpía, entropía, cp, etc. probablemente habrá que darle un valor inicial a la propiedad que desea calcular.

Solver/Valores iniciales:

Desde aquí se puede escoger el valor inicial de todas las variables o solo para algunas. Incluso se puede poner que las últimas variables calculadas sean utilizadas como valores iniciales la próxima vez.
El valor inicial es crucial en la resolución de ecuaciones numéricamente. Estos algoritmos solamente buscan una solución por tanto ante el siguiente ejemplo x^2=1 hay dos soluciones:

x = -1 y x= 1.

Si el punto inicial es 2 el resultado será 1 pero si el punto inicial es -2 el resultado sera -1.

Esta es una razón pero hay más. Si el algoritmo empieza muy lejos de la solución tal vez este no sea capaz de llegar a resolverlo porque, por ejemplo, la función tenga muchos máximos por el camino y el algoritmo caiga en uno de ellos. O quizás llega a un punto donde la función no exista. Por ejemplo, la siguiente ecuación:

1/(x-1)= -2

Si evalúa en x = 1 El resultado es infinito y por tanto no podrá seguir trabajando.

1) Valor inicial

Esto cambia el valor inicial de las variables.

2) Guardar el último valor como punto inicial

Esto hará que el algoritmo trabaje mejor ya que en las sucesivas ocasiones, empezará directamente en la solución o si se han cambiado las ecuaciones, muy probablemente estará más cerca del punto final. Por defecto desactivado.

3) Poner punto inicial

Se puede escoger un punto inicial para cada variable, por separado. El botón de borrar elimina todos los valores de vez.

Solver/Ejemplos:

Todos estos ejemplos pueden ser resueltos por el programa.

Recomendaciones:

Dejar espacios entre operadores y entre líneas es un truco muy sencillo que conllevará un texto mucho más legible.

Ejemplo Nº 1

Cos(x) + 4 * X = 0

No importa si x es mayúscula o minúscula.

Ejemplo Nº 2

Log[x) / Log(10) = 1

X=10 es el resultado. Es la manera de hacer logaritmos con distinta base. Además, se puede apreciar que da igual usar corchetes que paréntesis.

Ejemplo Nº 3

2 * x + y = 1

x * y = 3

a + b = x

3 * a * b = 6

Se puede ver que hay un sistema de 4 x 4, Sin embargo, si se resuelve primero el de x e y puede resolverse mediante dos sistemas de 2 x 2. Esto es lo que hace eSuite.

Ejemplo Nº 4

1 / (x - 1) = 2

Si prueba resolver esta ecuación (sin cambiar el valor inicial) no lo resolverá. Esto sucede porque en x = 1 la función vale infinito. Pruebaremos con valor inicial = 2 usando Line-Search y la resolverá. Sin embargo con valores por debajo de 1 no podrá.

Ejemplo Nº 5

Estas ecuaciones las resuelve una a una:

F = m * a ^ 2

m = 3,4 * a

a = 2.3

eSuite/Algoritmos:

Esta sección contiene los mismos ejemplos que los del apartado 4.2 de la memoria y los del anexo 2 de región de confianza.

Matemáticas:

Características:

1- A diferencia de eSuite Solver cos, exp, sinh... deben estar seguidos de corchetes.

2- La sección de gráficas puede trabajar con varias funciones a la vez.

3- Internamente, el programa transforma las letras a mayúsculas, por eso todas las variables aparecerán en mayúsculas, de la misma manera que cuando falle algo al introducir un comando, este saldrá en mayúsculas también.

4- Las ecuaciones renderizadas no pueden ser copiadas.

5- Los comandos y ejemplos pueden verse escribiendo *help* o *quickhelp* en la línea de comandos.

3) Ayuda Mathematics

A esta ayuda se accede desde la sección de Mathematics, introduciendo el comando “QuickHelp” para una versión reducida o “Help” para la versión completa.

"Funciones comunes:" cos[], sin[], tan[], sqrt[], log[], arctan[], arcsin[], arccos[], pi, e,i for complex number, exp[],sinh[], cosh[],tanh[]

"Operadores:" + - * . / ^ ! () = > < ==

"Asignar valor a una variable:"Example.-> x=3/(2+1)

"Eliminar el valor asignado a una variable:" Clear[Variable] or ClearAll[Variable]

"Representación de una matriz:" {{vector},{vector}...}-> Example:{{1,2},{2,2}}

"Multiplicación de una matriz y un vector:" {array}.{vector}

"Derivada de una función:" Deriv[function,variable] Example-> Deriv[Cos[x],x]

"Integración de una función:" Integrate[function,variable] Example-> Integrate[Sin[x],x]

"Inversa de una matriz:" Inverse[{{1,2},{2,2}}]

"Integración numérica:" NIntegrate[Polynomial,{Variable,InitialPoint,FinalPoint}]

"Determinate de una matriz:" Det[{{1,2},{2,2}}]

"Descomposición LU de una matriz:" LUdecomposition[{{1,2},{2,2}}]

"Matriz Jacobiana:" JacobiMatrix[{x^2+y,2*y},{x,y}]->{{2*X,1},{0,2}

"Matriz identidad:" IdentityMatrix[Number]

"Valores propios de una matriz:" eigenvalues[Matrix]

"Vectores propios de una matriz:" eigenvectors[Matrix]

"Distancia euclidea:" EuclidianDistance[{vector},{vector}]; Example:EuclidianDistance[{1,2,3,4},{5,6,7,8}]->(8)

"Elevar una matriz:" MatrixPower[{{1,2},{3,4}},3] -> returns the matrix elevated to the number

"Matriz de Hilber:"HilbertMatrix[Number] -> returns the asociated HilbertMatrix

"Trace[]:"Returns the intermediate operations; Example: Trace[D[Cos[x],x]] -> {D[Cos[x],x],(-1)*Sin[x]*D[x,x],(-1)*Sin[x]*D[x,x],{D[x,x],1},(-1)*1*Sin[x],(-1)*Sin[x]}

"PrimeQ[Number]" -> Says if the number is prime or not

"NextPrime[Number]" -> Returns the following prime to that number

"Fibonacci[Number]" -> Returns the Number position of the fibonaccie sequence

"Binomial[Number1,Number2]" -> Binomial[9,4] returns {126}

"Busca la solución de una función no lineal con el metodo de Bisección:" Example: FindRoot[Exp[x]==Pi^3,{x,-1,10},Bisection]

"Busca la solución de una función no lineal con el metodo de Brent:" Example: FindRoot[Exp[x]==Pi^3,{x,-1,10},Brent]

"El McLaurin de una función:" Taylor[Function,{variable,0,number of terms}]

"Expandir:" Example. -> Expand[(-1+x)*(1+x)*(1+x^2)*(1+x^4)*(1+x^8)]->(-1+x^16)

"Factorizar:" Example. -> Factor[-1+x^16]->(-1+x)*(1+x)*(1+x^2)*(1+x^4)*(1+x^8)

"Absoluto:" Abs[Number]

"Parte real:" Re[Number]; Example: Re[1/3+i]->(1/3)

"Parte compleja:" Im[Number]; Example: Im[1/3+i]->(1)

"Número Catalan:" CatalanNumber[Number]

"Número Harmocio:" HarmonicNumber[Number]

"Factor de integración:" FactorInteger[Number]->{{Factor,exponent}};

Example:FactorInteger[12]->{{2,2},{3,1}}->(2^2)*(3^1)

"Comparación:" Example: Pi==E->False; Example: 4==8/2->True; Example: 2<9/4->True

"Sumatorio:" Sum[polynomial,{var,initial point,final point,step}]; Example:Sum[x,{x,0,4,2}] ->(6)

"Multiplicatorio:" Product[polynomial,{var,initial point,final point,step}]; Example:Product[x,

{x,1,4,1}] ->(24)

Anexo 7. Formulas termodinámicas

En este anexo incluimos todas las formulas termodinámicas que hemos introducido en el programa, con información sobre su rango de validez y unidades:

Material: Air;
Property: Cp;
Formula: Cp=4.184/28.88*(6.713+10^(-2)*0.04697*Temperature+10^-5*0.1147*Temperature^2-10^-9*0.4696*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.33
Validity: 273 to 1800 Kelvin

Material: Air;
Property: Cv;
Formula: cv=4.184/28.88*(6.713+10^(-2)*0.04697*temperature+10^-5*0.1147*temperature^2-10^-9*0.4696*temperature^3)-0.28704;
Variable: Cv, Temperature;
Note:Temperature: Kelvin.
Cv: Kj/(Kg*Kelvin)
Average error: 0.33
Validity: 273 to 1800 Kelvin

Material: Air;
Property: Enthalpy;
Formula: 12.074*temp^0+0.924502*temp^1+0.000115985*temp^2+-0.563569*10^-8*temp^3=enthalpy;
Variable: Enthalpy, Temp;
Note:Enthalpy = Kj/Kg
250 <= Temperature <= 2000 Kelvin
Maximum absolute differece: 0,27%

Material: Air;
Property: InternalEnergy;
Formula: 12.074*temp^0+0.924502*temp^1+0.000115985*temp^2+-0.563569*10^-8*temp^3-0.28704*temp=u;
Variable: Temp, U;
Note:Internal Energy U:Kj/Kg
250<= Temperature <= 2000 Kelvin
Maximum absolute difference= 0.40%

Material: Air;
Property: Entropy;
Formula: entropy=1.386989+0.18493*10^-3*temp+0.95*Log[temp];
Variable: Entropy, Temp;
Note:Entropy: Kj/Kg
250 <= Temperature <= 2000 Kelvin
Maximum absolute difference: 0.06%

Material: Air;
Property: Isentropic_Pressure_Function;
Formula: ipr=(1.386989+0.18493*10^-3*temp+0.95*Log[temp])/0.28704;
Variable: IPR, temp;
Note:IPR:Kelvin
250 <= Temperature <= 2000 Kelvin
Maximum absolute difference: 0.06%

Material: Air;
Property: Isentropic_Volume_Function;
Formula: ivr=Log[0.28704*temp]-((1.386989+0.18493*10^-3*temp+0.95*Log[temp])/0.28704);
Variable: IVR, Temp;
Note:IVR:kelvin
250 <= Temperature <= 2000 Kelvin
Maximum absolute difference:0.06%

Material: Air;
Property: Specific_Heat_Ratio;

Formula: g=1/(1-(0.28704/(4.184/28.88*(6.713+10^(-2)*0.04697*temperature+10^-5*0.1147*temperature^2-10^-9*0.4696*temperature^3))));
Variable: G, Temperature;
Note:Specific Heat Ratio G: Kelvin
250 <= Temperature <= 2000 Kelvin
Average error: 0.33%

Material: Air;
Property: Speed_of_Sound;
Formula: sos=31.558*((1/(1-(0.28704/(4.184/28.88*(6.713+10^(-2)*0.04697*temperature+10^-5*0.1147*temperature^2-10^-9*0.4696*temperature^3)))))0.28704*temperature)^0.5;
Variable: SoS, temperature;
Note:Speed of Sound SoS: m/s
250<= Temperature <= 2000 Kelvin
Average error:0.33%

Material: Argon;
Property: Cp;
Formula: cp=0.52034;
Variable: Cp;
Note:Cp(Kj/(Kg*Kelvin)) it's constant

Material: Argon;
Property: Enthalpy;
Formula: enthalpy=0.52034*temp;
Variable: Enthalpy, Temp;
Note:Ideal Gas
Enthalpy:Kj/Kg
Temperature: Kelvin

Material: Argon;
Property: Entropy;
Formula: entropy=0.52034*Log[temp];
Variable: Entropy, Temp;
Note:Entropy: Kj/(Kg*Kelvin)

Temperature: Kelvin
Ideal gas

Material: Butane;
Property: Cp;
Formula: Cp=4.184/58.08*(0,954+8.873*10^-2*Temperature-4.38*10^-5*Temperature^2+8.360*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.24
Validity: 273 to 1500

Material: Butane;
Property: Enthalpy;
Formula: 0.23665134/(1)*temp^1+5.10573*10^-3/(2)*temp^2+-4.16089*10^-7/(3)*temp^3+-1.1450804*10^-9/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal Gas
Enthalpy: Kj/Kg
280 <= Temperature <= 1080 Kelvin
Maximum difference: 0.1%

Material: Butane;
Property: Entropy;
Formula: 0.23665134*Log[temp]+5.10573*10^-3/(1)*temp^1+-4.16089*10^-7/(2)*temp^2+-1.1450804*10^-9/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Entropy: Kj/(Kg*K)
280 <= Temperature <= 1080 Kelvin
Maximum difference: 0.1%
Ideal gas

Material: CarbonMonoxide;
Property: Cp;

Formula: Cp=4.184/28,0101*(9.726+0.04001*10^-2*Temperature+0.1283*10^-5*Temperature^2-0.5307*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.30
Validity: 273 to 1800

Material: CarbonMonoxide;
Property: Enthalpy;
Formula: 1.020802/(1)*temp^1+3.82075*10^-4/(2)*temp^2+-2.4945*10^-6/(3)*temp^3+6.81145*10^-9/(4)*temp^4+-7.93722*10^-12/(5)*temp^5+4.291972*10^-15/(6)*temp^6+-8.903274*10^-19/(7)*temp^7=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
250 <= Temperature <= 1050 Kelvin
Maximum diference 0.1%

Material: CarbonMonoxide;
Property: Entropy;
Formula: 1.020802*Log[temp]+3.82075*10^-4/(1)*temp^1+-2.4945*10^-6/(2)*temp^2+6.81145*10^-9/(3)*temp^3+-7.93722*10^-12/(4)*temp^4+4.291972*10^-15/(5)*temp^5+-8.903274*10^-19/(6)*temp^6=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
250 <= Temperature <= 1050 Kelvin
Maximum diference 0.1%

Material: CarbonDioxide;
Property: Cp;
Formula: Cp=4.184/44.010*(5.316+1.4285*10^-2*Temperature-0.8362*10^-5*Temperature^2+1.784*10^-9*Temperature^3);
Variable: Cp, Temperature;

Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.22
Validity: 273 to 1800

Material: CarbonDioxide;
Property: Enthalpy;
Formula: 0.45386462/(1)*temp^1+1.5334795*10^-3/(2)*temp^2+-4.195556*10^-7/(3)*temp^3+-1.871946*10^-9/(4)*temp^4+2.862388*10^-12/(5)*temp^5+-1.6962*10^-15/(6)*temp^6+3.717285*10^-19/(7)*temp^7=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
200 <= Temperature <= 1000 Kelvin
Maximum diference 0.14%

Material: CarbonDioxide;
Property: Entropy;
Formula: 0.45386462*Log[temp]+1.5334795*10^-3/(1)*temp^1+-4.195556*10^-7/(2)*temp^2+-1.871946*10^-9/(3)*temp^3+2.862388*10^-12/(4)*temp^4+-1.6962*10^-15/(5)*temp^5+3.717285*10^-19/(6)*temp^6=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
200 <= Temperature <= 1000 Kelvin
Maximum diference 0.14%

Material: Ethane;
Property: Cp;
Formula: Cp=4.184/30,07*(1.648+10^-2*Temperature*4.124-1.53*10^-5*Temperature^2+1.740*10^-9*Temperature);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.28

Validity: 273 to 1500

Material: Ethane;
Property: Enthalpy;
Formula: 0.5319795/(1)*temp^1+3.755877*10^-3/(2)*temp^2+1.789289*10^-6/(3)*temp^3+-2.13225*10^-9/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
280 <= Temperature <= 1080 Kelvin
Maximum diference 0.9%

Material: Ethane;
Property: Entropy;
Formula: 0.5319795*Log[temp]+3.755877*10^-3/(1)*temp^1+1.789289*10^-6/(2)*temp^2+-2.13225*10^-9/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
280 <= Temperature <= 1080 Kelvin
Maximum diference 0.9%

Material: Hydrogen;
Property: Cp;
Formula: Cp=4.184/1.0079*(6.952-0.04576*10^-2*Temperature+0.09563*10^-5*Temperature^2-0.2079*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.79
Validity: 273 to 1800

Material: Hydrogen;
Property: Enthalpy_2;
Formula: 14.4947/(1)*temp^1=enthalpy;

Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
425 <= Temperature <= 490 Kelvin

Material: Hydrogen;
Property: Entropy_2;
Formula: 14.4947*Log[temp]=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
425 <= Temperature <= 490 Kelvin

Material: Hydrogen;
Property: Enthalpy_1;
Formula: 5.0066253/(1)*temp^1+1.01569422*10^-1/(2)*temp^2+-6.02891517*10^-4/(3)*temp^3+2.7375894*10^-6/(4)*temp^4+-8.4758275*10^-9/(5)*temp^5+1.43800374*10^-11/(6)*temp^6+-9.8072403*10^-15/(7)*temp^7=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
250 <= Temperature <= 425 Kelvin
Maximum diference 0.06%

Material: Hydrogen;
Property: Entropy_1;
Formula: 5.0066253*Log[temp]+1.01569422*10^-1/(1)*temp^1+-6.02891517*10^-4/(2)*temp^2+2.7375894*10^-6/(3)*temp^3+-8.4758275*10^-9/(4)*temp^4+1.43800374*10^-11/(5)*temp^5+-9.8072403*10^-15/(6)*temp^6=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
250 <= Temperature <= 425 Kelvin
Maximum diference 0.06%

Material: Helium;
Property: Cp;
Formula: 5.1931=cp;
Variable: Cp;
Note:Cp it's constant

Material: Helium;
Property: Enthalpy;
Formula: 5.1931*temp^1=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
250 <= Temperature <= 1050 Kelvin

Material: Helium;
Property: Entropy;
Formula: 5.1931*Log[temp]=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
250 <= Temperature <= 1050 Kelvin

Material: Methane;
Property: Cp;
Formula: Cp=4.184/16.043*(4.750+10^-2*Temperature*1.2+10^-5*Temperature^2*0.303-10^-9*2.63*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.57
Validity: 273 to 1500

Material: Methane;
Property: Enthalpy;
Formula: 1.9165258/(1)*temp^1+-1.09269*10^-3/(2)*temp^2+8.696605*10^-6/(3)*temp^3+-

5.2291144*10^-9/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
280 <= Temperature <= 1080 Kelvin
Maximum diference 0.9%

Material: Methane;
Property: Entropy;
Formula: 1.9165258*Log[temp]+-1.09269*10^-3/(1)*temp^1+8.696605*10^-6/(2)*temp^2+-5.2291144*10^-9/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
280 <= Temperature <= 1080 Kelvin
Maximum diference 0.9%

Material: Nitrogen;
Property: Cp;
Formula: Cp=4.184/14.0067*(6.903-0.03753*10^-2*Temperature+0.193*10^-5*Temperature^2-0.6861*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.34
Validity: 273 to 1800

Material: Nitrogen;
Property: Enthalpy;
Formula: 1.088047/(1)*temp^1+-3.55968*10^-4/(2)*temp^2+7.2907605*10^-7/(3)*temp^3+-2.8861556*10^-10/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
280 <= Temperature <= 1050 Kelvin

Maximum diference 0.2%

Material: Nitrogen;
Property: Entropy;
Formula: 1.088047*Log[temp]+-3.55968*10^-4/(1)*temp^1+7.2907605*10^-7/(2)*temp^2+-2.8861556*10^-10/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
280 <= Temperature <= 1050 Kelvin
Maximum diference 0.2%

Material: Oxygen;
Property: Cp;
Formula: Cp=4.184/15.9994*(6.085+0.3631*10^-2*Temperature-0.1709*10^-5*Temperature^2+0.3133*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.28
Validity: 273 to 1800

Material: Oxygen;
Property: Enthalpy;
Formula: 0.929247/(1)*temp^1+-3.220603*10^-4/(2)*temp^2+1.166523*10^-6/(3)*temp^3+-7.1157865*10^-10/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
250 <= Temperature <= 1050 Kelvin
Maximum diference 0.4%

Material: Oxygen;
Property: Entropy;
Formula: 0.929247*Log[temp]+-3.220603*10^-4/(1)*temp^1+1.166523*10^-6/(2)*temp^2+-

7.1157865*10^-10/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
250 <= Temperature <= 1050 Kelvin
Maximum diference 0.4%

Material: Propane;
Property: Cp;
Formula: Cp=4.184/44*(-0.966+7.279*10^-2*Temperature-3.755*10^-5*Temperature^2+7.850*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.12
Validity: 273 to 1500

Material: Propane;
Property: Enthalpy;
Formula: 8.41607*10^-2/(1)*temp^1+5.7701407*10^-3/(2)*temp^2+-1.292127*10^-6/(3)*temp^3+-6.9945925*10^-10/(4)*temp^4=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
280 <= Temperature <= 1080 Kelvin
Maximum diference 0.86%

Material: Propane;
Property: Entropy;
Formula: 8.41607*10^-2*Log[temp]+5.7701407*10^-3/(1)*temp^1+-1.292127*10^-6/(2)*temp^2+-6.9945925*10^-10/(3)*temp^3=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/Kg
280 <= Temperature <= 1050 Kelvin

Maximum diference 0.35%

Material: SteamSaturated;
Property: Cp;
Formula: Cp=4.184/18.01528*(7.7+0.04594*10^-2*Temperature+0.2521*10^-5*Temperature^2-0.8587*10^-9*Temperature^3);
Variable: Cp, Temperature;
Note:Temperature: Kelvin.
Cp: Kj/(Kg*Kelvin)
Average error: 0.24
Validity: 273 to 1800

Material: SteamSaturated;
Property: SaturationTemp_1;
Formula: temperature=0.426776*10^2+(-0.38927*10^4/(Log[pressure]-0.948654*10^1));
Variable: Pressure, Temperature;
Note:0,000611 <= Pressure < 12.33 MPa
273,16 <= Temperature < 600 Kelvin
Deviation: 0.08 %

Material: SteamSaturated;
Property: SaturationTemp_2;
Formula: temperature=-0.387592*10^3+(-0.125875*10^5/(Log[pressure]-0.152578*10^2));
Variable: Pressure, Temperature;
Note:12,33 <= Pressure<=22,1 MPa
600 <= Temperature <= 647,3 Kelvin
Deviation: 0,08 %

Material: SteamSaturated;
Property: Specific_Liquid_Volume;
Formula: 1.0-1.9153882(1-temp/647.3)^(1/3)+12.015186(1-temp/647.3)^(5/6)-7.8464025(1-temp/647.3)^(7/8)-3.888614(1-temp/647.3)^1+2.0582238(1-temp/647.3)^2-2.0829991(1-temp/647.3)^(3)+0.82180004(1-temp/647.3)^4+0.47549742(1-temp/647.3)^5=volume/3.155*10^(-3);
Variable: Temp, Volume;

Note:Volume: m^3
273,16 <= Temperature <= 647,3 K
Deviation: 0,1%

Material: SteamSaturated;
Property: Specific_Vapor_Volume;
Formula: 1.0+1.6351057(1-temp/647.3)^(1/3)+52.584599(1-temp/647.3)^(5/6)+-44.694653(1-temp/647.3)^(7/8)+-8.9751114(1-temp/647.3)^1+-0.4384553(1-temp/647.3)^2+-19.179576(1-temp/647.3)^3+36.765319(1-temp/647.3)^4+-19.462437(1-temp/647.3)^5+0.0(1-temp/647.3)^6+0.0(1-temp/647.3)^7=volume*pressure/(3.155*10^(-3)*22.089);
Variable: Pressure, Temp, Volume;
Note:Volume: m^3
Pressure: Mpa
273.16 <= Temperature <= 647.13 Kelvin
Deviation: 0.1%

Material: SteamSaturated;
Property: Saturated_Liquid_Enthalpy_1;
Formula: 624.698837(1-temp/647.3)^1+-2343.85369(1-temp/647.3)^2-9508.12101(1-temp/647.3)^3+71628.7928(1-temp/647.3)^4-163535.221(1-temp/647.3)^5+166531.1093(1-temp/647.3)^6+-64785.4585(1-temp/647.3)^7=(enthalpy/2099.3);
Variable: Enthalpy, Temp;
Note:Enthalpy: Kj/Kg
273,16 <= Temperature <= 300 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Saturated_Liquid_Enthalpy_2;
Formula: 0.8839230108-2.67172935(1-temp/647.3)^1+6.22640035(1-temp/647.3)^2-13.1789573(1-temp/647.3)^3-1.91322436(1-temp/647.3)^4+68.7937653(1-temp/647.3)^5+-124.819906(1-temp/647.3)^6+72.1435404(1-temp/647.3)^7=enthalpy/2099.3;
Variable: Enthalpy, Temp;
Note:Enthalpy : Kj/Kg
300 <= Temperature <= 600 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Saturated_Liquid_Enthalpy_3;
Formula: 1.0+-0.441057805(1-temp/647.3)^(1/3)+-5.52255517(1-temp/647.3)^(5/6)+6.43994847(1-temp/647.3)^(7/8)+-1.64578795(1-temp/647.3)^1+-1.30574143(1-temp/647.3)^2=enthalpy/(2099.3);
Variable: Enthalpy, Temp;
Note:Enthalpy: Kj/Kg
600 <= Temperature <= 647,3 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Latent_Heat_Vaporization;
Formula: 0.0+0.779221(1-temp/647.3)^(1/3)+4.62668(1-temp/647.3)^(5/6)+-1.07931(1-temp/647.3)^(7/8)+-3.87446(1-temp/647.3)^1+2.94553(1-temp/647.3)^2+-8.06395(1-temp/647.3)^3+11.5633(1-temp/647.3)^4+-6.02884(1-temp/647.3)^5=enthalpy/(2500.9);
Variable: Enthalpy, Temp;
Note:Enthalpy: Kj/Kg
273,16 <= Temperature <= 647,3 Kelvin
Deviation: 0.15%

Material: SteamSaturated;
Property: Saturated_Vapor_Enthalpy;
Formula: 1.0+0.457874342(1-temp/647.3)^(1/3)+5.08441288(1-temp/647.3)^(5/6)+-1.48513244(1-temp/647.3)^(7/8)+-4.81351884(1-temp/647.3)^1+2.69411792(1-temp/647.3)^2+-7.39064542(1-temp/647.3)^3+10.4961689(1-temp/647.3)^4+-5.46840036(1-temp/647.3)^5+0.0(1-temp/647.3)^6+0.0(1-temp/647.3)^7=enthalpy/(2099.3);
Variable: Enthalpy, Temp;
Note:Enthalpy: Kj/Kg
273,16 <= Temperature <= 647,13 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Saturated_Liquid_Entropy_1;
Formula: -1836.92956(1-temp/647.3)^1+14706.6352(1-temp/647.3)^2+-43146.6046(1-

temp/647.3)^3+48606.6733(1-temp/647.3)^4+7997.5096(1-temp/647.3)^5+-58333.9887(1-temp/647.3)^6+33140.0718(1-temp/647.3)^7=entropy/4.4289;
Variable: Entropy, Temp;
Note:Entropy: kj/(Kg*K)
273,16 <= Temperature <= 300 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Saturated_Liquid_Entropy_2;
Formula: 0.912762917-1.75702956(1-temp/647.3)^1+1.68754095(1-temp/647.3)^2+5.82215341(1-temp/647.3)^3+-63.33354786(1-temp/647.3)^4+188.076546(1-temp/647.3)^5+-252.3445312(1-temp/647.3)^6+128.058531(1-temp/647.3)^7=entropy/4.4289;
Variable: Entropy, Temp;
Note:Entropy: Kj/(Kg*Kelvin)
300 <= Temperature <= 600 Kelvin
Deviation: 0.05%

Material: SteamSaturated;
Property: Saturated_Liquid_Entropy_3;
Formula: 1.0+-0.32481765(1-temp/647.3)^(1/3)+-2.990556709(1-temp/647.3)^(5/6)+3.23419(1-temp/647.3)^(7/8)+-0.678067859(1-temp/647.3)^1-1.91910364(1-temp/647.3)^2=entropy/4.4289;
Variable: Entropy, Temp;
Note:Entropy:Kj/(Kg*Kelvin)
600 <= Temperature <= 647.3
Deviation: 0.02%

Material: SteamSaturated;
Property: Saturated_Vapor_Entropy;
Formula: 1.0+0.377391(1-temp/647.3)^(1/3)+-2.78368(1-temp/647.3)^(5/6)+6.93135(1-temp/647.3)^(7/8)+-4.34839(1-temp/647.3)^1+1.34672(1-temp/647.3)^2+1.75261(1-temp/647.3)^3+-6.22295(1-temp/647.3)^4+9.99004(1-temp/647.3)^5=entropy/4.4289;
Variable: Entropy, Temp;
Note:Entropy: Kj/(Kg*Kelvin)
273,16 <= Temperature <= 647,3 Kelvin
Deviation:0.05%

Material: SulfurDioxide;
Property: Cp;
Formula: 4.32805*10^-1*temp^0+5.9994156*10^-4*temp^1+4.593367*10^-7*temp^2+-1.433024*10^-9*temp^3+1.0409341*10^-12*temp^4+-2.5313735*10^-16*temp^5+0.0*temp^6=cp;
Variable: Cp, Temp;
Note:Ideal gas
Cp: Kj/(Kg*Kelvin)
300 <= Temperature <= 1100 Kelvin
Maximum diference 0.24%

Material: SulfurDioxide;
Property: Enthalpy;
Formula: 4.32805*10^-1/(1)*temp^1+5.9994156*10^-4/(2)*temp^2+4.593367*10^-7/(3)*temp^3+-1.433024*10^-9/(4)*temp^4+1.0409341*10^-12/(5)*temp^5+-2.5313735*10^-16/(6)*temp^6=enthalpy;
Variable: Enthalpy, Temp;
Note:Ideal gas
Enthalpy: Kj/Kg
300 <= Temperature <= 1100 Kelvin
Maximum diference 0.24%

Material: SulfurDioxide;
Property: Entropy;
Formula: 4.32805*10^-1*Log[temp]+5.9994156*10^-4/(1)*temp^1+4.593367*10^-7/(2)*temp^2+-1.433024*10^-9/(3)*temp^3+1.0409341*10^-12/(4)*temp^4+-2.5313735*10^-16/(5)*temp^5=entropy;
Variable: Entropy, Temp;
Note:Ideal gas
Entropy: Kj/(Kg*Kelvin)
300 <= Temperature <= 1100 Kelvin
Maximum diference 0.24%

Material: SteamSuperHeated;
Property: Specific_Volume_1;

Formula: volume=4.61631*10^-4*temp/press-0.0527993*Exp[-0.00375928*temp]
+0.1/press(0.022-Exp[-3.741378-4.7838281*10^-3*(42.6776+(-3892.7/(Log[press]-
9.48654)))^1+1.5923434*10^-5*(42.6776+(-3892.7/(Log[press]-9.48654)))^2])Exp[((42.6776+(-
3892.7/(Log[press]-9.48654)))-temp)/40];
Variable: Press, Temp, Volume;
Note:Volume: m^3
Temperature: Kelvin
0.001 <= Pressure <= 12 MPa
Maximum deviation: 1%

Material: SteamSuperHeated;
Property: Specific_Volume_2;
Formula: volume=4.61631*10^-4*temp/press-0.0527993*Exp[-0.00375928*temp]
+0.1/press(0.022-Exp[-3.741378+-4.7838281*10^-3*(-0.387592*10^3+(-0.125875*10^5/
(Log[press]-0.152578*10^2)))^1+1.5923434*10^-5*(-0.387592*10^3+(-0.125875*10^5/
(Log[press]-0.152578*10^2)))^2])Exp[((-0.387592*10^3+(-0.125875*10^5/(Log[press]-
0.152578*10^2)))-temp)/40];
Variable: Press, Temp, Volume;
Note:Volume: m^3
Temperature: Kelvin
12 < Pressure <= 22.1 MPa
Maximun deviation: 1%

Material: SteamSuperHeated;
Property: Enthalpy_1;
Formula: (2041.21+-40.40021*press+-
0.48095*press^2)+(1.610693+0.05472051*press+7.517537*10^-
4*press^2)*temp^1+(3.383117*10^-4+-1.975736*10^-5*press+-2.87409*10^-7*press^2)*temp^2-
(1707.82+-16.99419*(42.6776+(-3892.7/(Log[press]-9.48654)))+0.062746295*(42.6776+(-3892.7/
(Log[press]-9.48654)))^2+-1.0254259*10^-4*(42.6776+(-3892.7/(Log[press]-
9.48654)))^3+6.4561298*10^-8*(42.6776+(-3892.7/(Log[press]-9.48654)))^4)*Exp[((42.6776+(-
3892.7/(Log[press]-9.48654)))-temp)/45]=enthalpy;
Variable: Enthalpy, Press, Temp;
Note:Enthalpy: Kj/Kg
temperature: Kelvin

0.001 <= Pressure <= 12 MPa
Maximum deviation: 1%

Material: SteamSuperHeated;
Property: Enthalpy_2;
Formula: (2041.21+-40.40021*press+-0.48095*press^2)+(1.610693+0.05472051*press+7.517537*10^-4*press^2)*temp^1+(3.383117*10^-4+-1.975736*10^-5*press+-2.87409*10^-7*press^2)*temp^2-(1707.82+-16.99419*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))+0.062746295*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^2+-1.0254259*10^-4*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^3+6.4561298*10^-8*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^4)*Exp[((-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))-temp)/45]=enthalpy;
Variable: Enthalpy, Press, Temp;
Note:Enthalpy: Kj/Kg
Temperature: Kelvin
12 < Pressure <= 22.1 MPa
Maximum deviation: 1%

Material: SteamSuperHeated;
Property: Entropy_1;
Formula: 4.6162961+0.01039008*temp^1+-9.873085*10^-6*temp^2+5.43411*10^-9*temp^3+-1.170465*10^-12*temp^4+-0.4650306*Log[10*press+0.001]-(1.777804+-0.01802468*(42.6776+(-3892.7/(Log[press]-9.48654)))^1+6.854459*10^-5*(42.6776+(-3892.7/(Log[press]-9.48654)))^2+-1.184424*10^-7*(42.6776+(-3892.7/(Log[press]-9.48654)))^3+8.142201*10^-11*(42.6776+(-3892.7/(Log[press]-9.48654)))^4)*Exp[((42.6776+(-3892.7/(Log[press]-9.48654)))-temp)/85]=entropy;
Variable: Entropy, Press, Temp;
Note:Entropy: Kj/(Kg*Kelvin)
Temperature:Kelvin
0.001 <= Pressure <= 12 MPa
Maximum deviation: 1%

Material: SteamSuperHeated;

Property: Entropy_2;

Formula: 4.6162961+0.01039008*temp^1+-9.873085*10^-6*temp^2+5.43411*10^-9*temp^3+-1.170465*10^-12*temp^4+-0.4650306*Log[10*press+0.001]-(1.777804+-0.01802468*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^1+6.854459*10^-5*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^2+-1.184424*10^-7*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^3+8.142201*10^-11*(-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))^4)*Exp[((-0.387592*10^3+(-0.125875*10^5/(Log[press]-0.152578*10^2)))-temp)/85]=entropy;

Variable: Entropy, Press, Temp;

Note:Entropy:Kj/(Kg*Kelvin)

Temperature: Kelvin

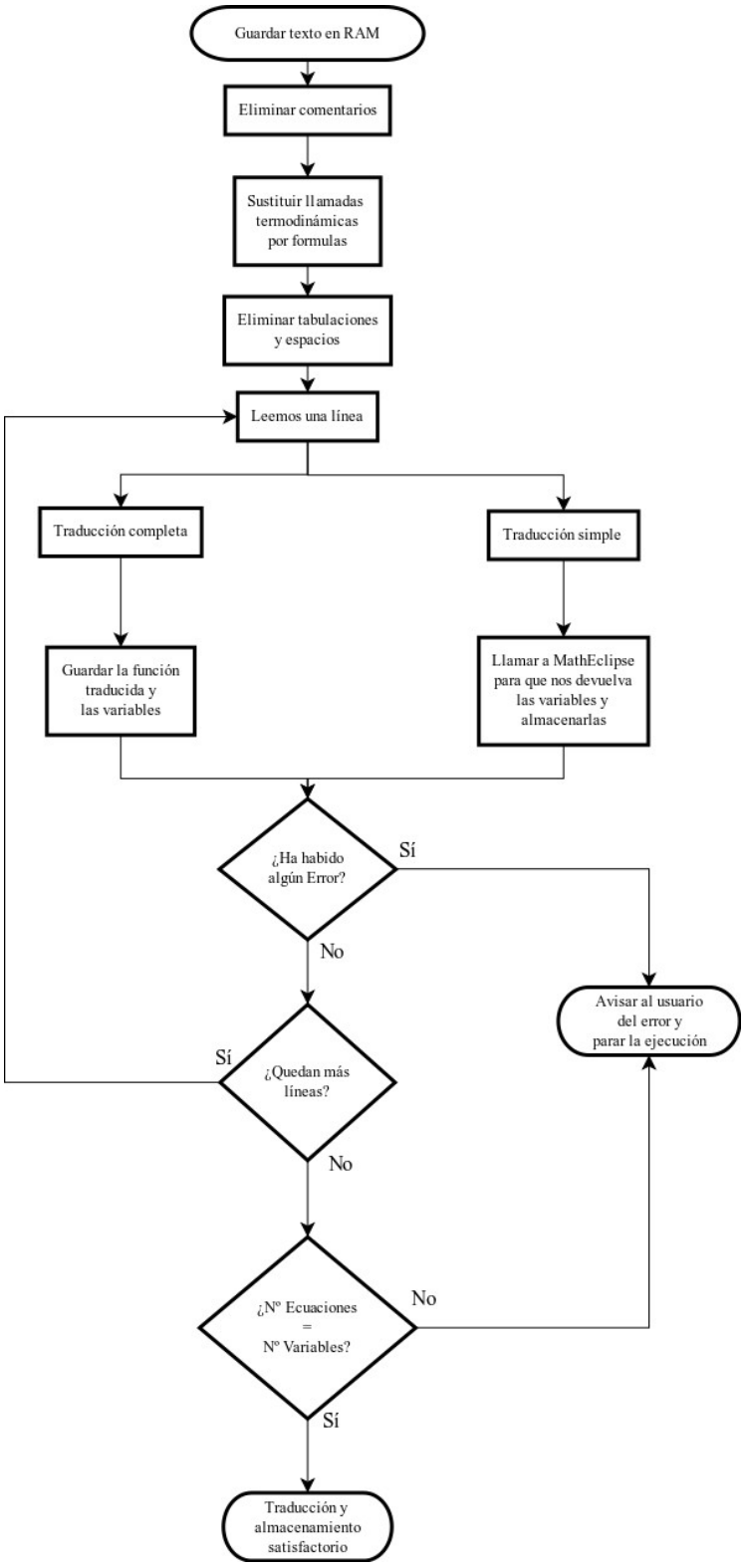
12 < Pressure <= 22.1

Maximum deviation: 1%

Anexo 8. Diagramas de flujo

Para facilitar la compresión de los algoritmos explicados en la memoria hemos incluido en este anexo unos diagramas de bloques que permiten entender más fácilmente los métodos del programa.

1)Traducción y almacenaje de funciones y variables.



2) Método implementado de reducción de sistemas de ecuaciones.