

9. Anexos.

9.1. Anexo A: La optimización en MATLAB.

Matlab es un entorno muy extenso de optimización matemática que incluye una completa biblioteca sobre Optimización que incluye funciones para resolver problemas como el planteado a comienzos de este Proyecto.

9.1.1. El Sistema.

El siguiente sistema es el resultado de aplicar la función de integridad sobre el sistema original:

$$\begin{aligned} \max \quad & \sum_{(i,j) \in A} c_{ij} f(x_{ij}) \\ \text{s.t.} \quad & X(j, N) - X(N, j) = \begin{cases} 1 & \text{if } j \in T \\ 0 & \text{if } j \notin T, j \neq e, \forall j \in N \\ -|T|+1 & \text{if } j = e \end{cases} \\ & x_{ij} \geq 0; \quad \forall (i, j) \in A \\ & x_{ij} \cdot x_{ji} = 0; \quad \forall (i, j) \in A \end{aligned}$$

9.1.2. Fmincon.

Dentro de las numerosas funciones incluidas en la *Optimization Toolbox* de Matlab, la que se ajusta a las exigencias del problema es *fmincon*.

Existe un manual detallado sobre esta función en el manual de ayuda que incorpora Matlab. Como este manual explica, *fmincon* admite todas las restricciones a las que está sujeto el problema. El resto de funciones en esta biblioteca tienen alguna limitación que no permite utilizarlas.

Fmincon requiere una serie de parámetros a su entrada, el formato de esta función es el siguiente:

$$[x, fval, exitflag, output, lambda, grad] = \text{fmincon}(\text{fun}, x0, A, b, Aeq, beq, lb, ub, nonlcon, P1, P2)$$

No es necesario incluir todos los parámetros de entrada ni todos los de salida en esta función, la expresión mínima de la misma es:

$$x = \text{fmincon}(\text{fun}, x0, A, b)$$

Sin embargo, para poder implementar todas las restricciones del problema se utilizarán la mayoría de los parámetros.

9.1.2.1. Parámetros de salida.

Los parámetros de salida proporcionados por *fmincon* y que son necesarios para comprobar la validez de la solución son:

- “*x*” - Vector que contiene el resultado.
- “*fval*” - Valor final de la función objetivo.
- “*exitflag*” - Describe la condición de salida:
 - >0 – La función converge a una solución *x*.
 - 0 – Ha sido alcanzado el máximo número de iteraciones o de evaluación de la función objetivo.
 - <0 – La función no ha convergido a una solución.
- *output* – Estructura que contiene información sobre la optimización. Los campos de la estructura son:
 - *iterations* – Número de iteraciones.
 - *funcCount* – Número de evaluaciones de la función.
 - *algorithm* – Algoritmo usado
 - *cgIterations* – Número de iteraciones PCG utilizadas (sólo de utilidad cuando usamos un método *Large-Scale*.
 - *stepsize* – Tamaño final de *step* (sólo tiene validez cuando utilizamos un *medium-scale* algorithm).

9.1.2.2. Parámetros de entrada

Los parámetros de entrada de *fmincon*, necesarios para implementar correctamente todas las restricciones y la función objetivo del problema son:

- *fun* – función objetivo. La función objetivo se especifica en un fichero *.m* aparte donde se especifica la función en sí. El fichero tiene que tener el mismo nombre que la función que implementa. A continuación se muestra el listado del fichero *fun_objetivo.m*:

```
function [f,g] = fun_objetivo(Xij,coste,NumTerminales)
Fij=((Xij-1).^2)./(Xij.^2+1);
%M=NumTerminales- 1;
%Fij=(Xij.^3-3.*Xij+2)./((M^2-3). *Xij+2);
Fij=Fij';
f = -(Fij*coste);
if nargout > 1
    g=2.*((1-Xij.^2)./(Xij.^2-1).^2);
    g=g';
    g=-(g*coste);
end
```

Como se puede observar, el nombre de la función es el mismo que el del fichero que la alberga. Además otro detalle importante es que la función objetivo aparece con un signo -, lo que indica que lo que se quiere es maximizar el resultado, no minimizarlo como normalmente hace *fmincon*.

También aparecen algunas líneas comentadas, eso es debido a que se utiliza o la primera o la segunda función de integridad, según se comenten unas líneas u otras.

Si el número de argumentos de salida que se le pide a la función es mayor que uno, entonces el gradiente de la función también se calcula. Este gradiente es necesario para conseguir mejores resultados puesto que si no se proporciona analíticamente como lo hacemos nosotros, Matlab lo calcula numéricamente y los resultados son peores.

A la función objetivo se le pasan como parámetros de entrada:

- x – Vector de entrada.
 - *coste* – Coste de los arcos.
 - *NumTerminales* – Necesario para la segunda función de integridad.
- x_0 – solución inicial. La solución inicial es necesaria para la correcta evolución de la optimización. En el problema, la solución inicial se ha hallado para cada problema especificado gracias a la implementación del algoritmo de Kruskal en Visual Basic. Se ha aprovechado que para la primera parte del Proyecto ya se había obtenido esta primera solución. Para ello, se ha exportado en cada ejecución del programa en Visual Basic cada una de estas primeras soluciones a un fichero en texto plano que puede ser leído por Matlab y por ello utilizado por *fmincon*.

Es necesario hacer hincapié en que la documentación de Matlab afirma que si esta primera solución es *no admisible*, el programa va a buscar una primera solución admisible que pueda ser utilizada para el arranque del algoritmo. Sin embargo debido a la complejidad del problema, sobre todo de las restricciones, Matlab no es capaz de encontrar una solución admisible si la primera solución proporcionada no lo es.

A continuación se muestra el fichero *PrimeraSol.txt* generado para el problema *steinb6.txt*:

steinb6.txt

```
6 12 1
9 36 1
10 17 1
12 5 1
16 8 1
17 36 2
18 22 1
19 28 1
21 5 1
```

25 5 1
 28 37 1
 34 29 1
 36 13 3
 37 22 2
 41 45 1
 42 27 1
 43 15 1
 8 5 2
 22 3 3
 27 38 2
 29 39 1
 38 44 3
 3 23 3
 5 24 6
 23 39 3
 39 31 5
 31 35 6
 35 11 6
 11 44 7
 44 24 11
 24 13 18
 13 15 21
 15 32 22
 32 4 22
 4 45 22
 45 26 24
 26 1 24

El primero y el segundo entero de cada línea son, respectivamente, el nodo inicial y el nodo final de cada arco, mientras el tercero indica el flujo que circula por ese arco. Para ello, se toma uno como origen y cada uno de los demás nodos terminales envían una unidad de tráfico a ese nodo origen. x_0 corresponde a la tercera columna del fichero de texto.

- A, b – Describen las restricciones que se puedan expresar como ecuaciones lineales. De modo que $A \cdot x = b$. Para nuestro problema A es una matriz $S \times M$, donde S es el número de restricciones del tipo:

$$X(j, N) - X(N, j) = \begin{cases} 1 & \text{if } j \in T \\ 0 & \text{if } j \notin T, j \neq e, \forall j \in N \\ -|T|+1 & \text{if } j = e \end{cases}$$

S por lo tanto es igual al número de nodos del sistema. Mientras que M debe coincidir con el número de componentes del vector x , es decir, el doble del número de arcos no-dirigidos del problema.

Las componentes de A serán 0, 1 ó -1. El vector x tiene el doble del número de arcos que se lee del fichero original, esto es así porque son arcos dirigidos. Para obtener el vector x dirigido a partir del no-dirigido se añade éste sobre sí mismo conmutando la primera y la segunda columna, que corresponden a los nodos inicial y final.

Por consiguiente, si el arco k comienza o termina en el nodo j , en el elemento (j, k) de A pondremos un 1 ó -1. Si j aparece en la primera columna, es nodo inicial, y se coloca un 1, si aparece en la segunda, es final y se coloca un -1.

b es un vector columna de dimensión $S \times 1$, y su valor depende únicamente de que el nodo j (elemento j de b) sea Terminal o no y que haya sido escogido como nodo origen.

- Aeq , beq – Describen las restricciones que podemos expresar como inecuaciones lineales. De modo que $Aeq \cdot x \leq beq$. En nuestro caso, existen dos posibilidades de expresar el conjunto de restricciones del tipo:

$$x_{ij} \geq 0; \quad \forall (i, j) \in A$$

Se pueden introducir en nuestro código en Matlab si se multiplican por -1 para obtener el signo adecuado en la inecuación.

$$-x_{ij} \leq 0; \quad \forall (i, j) \in A$$

Basta con tomar una matriz Aeq de dimensiones $M \times M$, donde M será igual al número de arcos no-dirigidos del problema. Aeq es una matriz diagonal con todos los elementos de la diagonal igual a -1.

beq es igual a un vector columna nulo de dimensiones $M \times 1$.

- lb , ub – *lower bound*, *upper bound*. Gracias a estos dos parámetros se pueden expresar las mismas restricciones que mediante Aeq y beq . Ya que se crean estos dos vectores (que tendrán tantas componentes como variables tengamos en el problema) haciendo lb nulo y ub igual a infinito.

NOTA: Aunque esta segunda forma de expresar las restricciones del tipo $x_{ij} \geq 0; \quad \forall (i, j) \in A$ ha sido también implementada, preferimos utilizar Aeq y beq debido a que en la documentación de `fmincon` proporcionada por Matlab existe, en el apartado referente a limitaciones, una referencia a que se consigue un mejor resultado numérico con estas últimas sobre todo cuando se utiliza Medium-Scale Algorithms.

- $nonlcon$ – Representa las restricciones no lineales del problema. Se pueden tener tantas restricciones no lineales como queramos tanto en forma de ecuación como de inecuación

no-lineal.

Las funciones no-lineales que expresan estas restricciones se introducen en un fichero aparte que debe tener el mismo nombre que la función que alberga. Por lo tanto se debe crear un fichero *nonlcon.m* en el que se introduzca el código. *nonlcon* devuelve cuatro vectores *C*, *Ceq*, *DC* y *DCeq*.

- *C* contiene las restricciones que se expresan como ecuaciones no-lineales
- *Ceq* contiene las que se expresan como inecuaciones no-lineales.
- *DC* es el gradiente de *C*. No es necesario proporcionar dicho gradiente, pero si recomendable puesto que el resultado es mejor si dicho gradiente se calcula analíticamente en vez de dejar que Matlab lo calcule numéricamente¹.
- *DCeq* es el gradiente de *Ceq*. Se calcula analíticamente por las mismas razones que *DC*.

A continuación se muestra el código de *nonlcon.m*.

```
function [C,Ceq,DC,DCeq] = nlincon(Xij, coste, NumTerminales)
```

```

C=[];
DC=[];
DCeq=[];
Ceq=zeros(size(Xij,1)/2,1);
aux=size(Ceq,1);
for i=1:1:aux
    Ceq(i)=Xij(i)*Xij(i+size(Xij,1)/2);
end
DCeq=zeros(size(Xij,1),size(Ceq,1));
for i=1:1:size(Ceq,1)
    DCeq(i,i)=Xij(i+size(Xij,1)/2);
end
for i=1:1:size(Ceq,1)
    DCeq(i+size(Xij,1)/2,i)=Xij(i);
end
```

Como se puede observar, tanto *C* como *DC* son nulas, esto es debido a que nuestras restricciones no lineales son ecuaciones, no inecuaciones.

$$x_{ij} \cdot x_{ji} = 0; \forall (i, j) \in A$$

El número de restricciones no lineales es igual al número de arcos no-dirigidos del problema.

DCeq es una matriz con el doble número de columnas que de filas puesto que tiene que tener la siguiente forma:

¹ Así se especifica en la documentación de *Matlab* sobre esta función.

$$\begin{pmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_2}{\partial x_1} & \dots & \frac{\partial c_N}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} & \frac{\partial c_2}{\partial x_2} & & \\ \vdots & \vdots & & \\ \frac{\partial c_1}{\partial x_M} & \dots & \dots & \frac{\partial c_N}{\partial x_M} \end{pmatrix}$$

El número de columnas es igual al número de restricciones no lineales en el sistema. Mientras que el número de filas es igual al número de arcos dirigidos. Se tiene, por tanto, el doble número de filas que de columnas para esta matriz.

El gradiente de estas restricciones se calcula de forma que para la componente correspondiente al arco dirigido (i,j) :

$$\frac{\partial (x_{ij} \cdot x_{ji})}{\partial x_{ij}} = x_{ji}$$

Por lo que en cada fila de *DCeq* únicamente existirá un elemento no nulo, mientras que en cada columna existirán dos.

- *P1, P2* – Parámetros adicionales que son necesarios en la función objetivo *fun_objetivo* y en la función no-lineal *nlincon*.

En primer lugar, estos parámetros se le pasan a la función objetivo porque el valor de la misma depende del coste asignado a cada arco y el valor de las restricciones dependen de si el nodo en cuestión es nodo terminal o no.

Otra alternativa a utilizar estos parámetros (alternativas que han sido implementadas pero sustituidas por *P1* y *P2*) consisten en realizar una lectura del fichero que contiene los datos iniciales del problema (*steinbX.txt*, *steincX.txt* ...) cada vez que se llama a *fun_objetivo* o *nlincon*.

Sin embargo , los resultados son similares y el número de accesos a este archivo ralentiza la ejecución de la optimización en Matlab debido a que el número de evaluaciones de la función en cada iteración suele ser del orden de 1000. Y en cada acceso al archivo tomaríamos uno de los datos, con lo cual serían necesarios dos accesos al archivo en cada iteración.

Fmincon permite la adicción de todos los parámetros que sean necesarios, estos parámetros se introducen después de *nlincon*, y deben ser los mismos para las dos funciones, es decir, para *nlincon.m* y para *fun_objetivo*. Aunque, como se comprobar, *nlincon* no hace uso de ninguno de ellos.

9.1.2.3. Opciones de *Fmincon*.

Además de introducir parámetros de entrada, *fmincon* soporta una serie de opciones de configuración tanto para controlar la evolución de la optimización como para indicar el método de

optimización que queremos usar.

Algunos de los parámetros se aplican a todos los algoritmos, otros son solo relevantes si se utilizan métodos del tipo *large-scale* y otros son sólo relevantes si se usan *medium-scale*.

Para introducir estas opciones se debe utilizar la función *optimset*, tanto para inicializar los valores de los mismos como para modificarlos.

- LargeScale – Usa un algoritmo del tipo *large-scale*, si es posible, cuando su valor es 'on'. Para usar algoritmos del tipo *medium-scale* hay que hacer que esta variable igual a 'off'.

Los siguientes parámetros tienen sentido cuando hablamos tanto de algoritmos *medium-scale* como del tipo *large-scale*:

- Diagnostics – Imprime a la salida una información de diagnóstico sobre la función a ser minimizada.
- Display – Nivel de información que se muestra a la salida. Su valor puede ser:
 - off – No muestra nada a la salida.
 - iter – Muestra una línea de información por cada iteración del algoritmo.
 - final – Muestra la salida final, con las estadísticas que reflejan el número total de iteraciones empleadas en el algoritmo, el número de evaluaciones de la función objetivo, la condición de salida, los multiplicadores de Lagrange de primer orden, etc.
- GradObj – Define si se va a proveer o no a *fmincon* del gradiente de la función objetivo calculado analíticamente o no. En caso de que el gradiente no sea proporcionado analíticamente *fmincon* va a calcularlo numéricamente. Sin embargo, la documentación que acompaña a Matlab asegura peores resultados si no se proporciona. De hecho, si se usa un método del tipo *largeScale* es obligatorio proporcionar este gradiente, no es así en *mediumScale* donde es opcional.
- MaxFunEvals – Máximo número de evaluaciones de la función objetivo.
- MaxIter – Máximo número de iteraciones del algoritmo utilizado.
- TolFun – Máxima tolerancia en el cálculo del valor de la función objetivo.
- TolCon – Máxima tolerancia en la violación de las restricciones.
- Tolx – Máxima tolerancia en el valor de la variable.

Los parámetros que se detallan a continuación sólo tienen sentido cuando hablamos de algoritmos del tipo *mediumScale*.

- DerivativeCheck – Compara el valor de las derivadas (gradiente de la función objetivo y de las restricciones no-lineales) proporcionadas por el usuario con las que calcula numéricamente (*finite-differencing derivatives*).
- DiffMaxChange – Cambio máximo en las variables cuando las derivadas son calculadas numéricamente.
- DiffMinChange – Mínimo cambio en las variables cuando las derivadas son calculadas numéricamente.

Los parámetros de configuración exclusivos de los algoritmos del tipo *largeScale* no van a ser comentados debido a que, como se explicará en la siguiente sección, una limitación de *fmincon*

obliga a usar algoritmos del tipo *mediumScale* aún siendo estos bastante peores.

9.1.2.4. Requerimientos.

Como hemos adelantado en secciones anteriores, los métodos de tipo *largeScale* proporcionan mejores soluciones que los de tipo *mediumScale*. Esto es debido a que para poder usar un método del tipo *largeScale*, debemos:

- Proveer el gradiente de la función objetivo. Esta no es una limitación, se puede proporcionar el gradiente de la función objetivo. De hecho, como es recomendable usarlo para métodos *mediumScale*, se calcula en *fun_objetivo*.
- Hacer que la opción *GradObj* esté a *on* en *options*.
- Especificar la región admisible para la solución utilizando uno de los siguientes tipos de restricciones (pero no las dos a la vez):

- O se usa un valor máximo y otro mínimo para el valor de las variables, es decir, para las restricciones del tipo :

$$x_{ij} \geq 0; \quad \forall (i, j) \in A$$

- O bien, las se expresa como ecuaciones lineales. En cuyo caso, necesitaremos usar la matriz *Aeq* (que además no puede tener más filas que columnas) y el vector columna *beq*.

Para nuestro problema, bastarían estos dos tipos de restricciones pero no se pueden utilizar a la vez si queremos usar *largeScale*.

- No se puede usar *A* y *b*, es decir, no se pueden tener restricciones de tipo inecuación. Siempre se pueden expresar los límites de una variable mediante *ub* y *lb*, pero no una inecuación más compleja. No es nuestro caso.

La función *fmincon* devuelve un “warning” si el gradiente **no** es proporcionado y además la opción *LargeScale* está a *off*, con lo cual, intentaría usar un método *LargeScale* sin gradiente. En nuestro caso, el mensaje que se obtiene si no hacemos *LargeScale* a *on*:

```
>> Warning: Large-scale (trust region) method does not currently solve this type of problem, switching to medium-scale (line search).
```

Si hacemos *LargeScale=off*, el mensaje desaparece. Esto es debido a que nuestro problema no cumple las restricciones necesarias para poder ejecutar un algoritmo de tipo *LargeScale*.

fmincon permite, sin embargo, que el gradiente sea aproximado numéricamente pero esta opción no es recomendable. El comportamiento numérico de la mayoría de los métodos de optimización es mejor cuando se proporciona el gradiente real de la función.

Si la solución inicial *x0* no es estrictamente admisible, *fmincon* busca una solución inicial por sí mismo. El problema resulta de las restricciones de nuestro sistema, *fmincon* no es capaz de encontrar una solución admisible inicial para nuestro problema, tal y como se muestran en las siguientes iteraciones del algoritmo. Se consigue que *fmincon* imprima una línea por cada iteración, haciendo la opción *Display* igual a *iter*.

2	19	-254.461	0.0008545	1	-0.221	28.2	Hessian modified twice; infeasible
3	21	-254.351	0.0004272	1	-0.11	28.2	Hessian modified twice; infeasible

4	23	-254.295	0.0002136	1	-0.0552	28.2	Hessian modified twice; infeasible
5	25	-254.268	0.0001068	1	-0.0276	28.2	Hessian modified twice; infeasible
6	27	-254.254	5.341e-005	1	-0.0138	28.2	Hessian modified twice; infeasible
7	29	-254.247	2.67e-005	1	-0.00686	28.2	Hessian modified twice; infeasible
8	31	-254.244	1.335e-005	1	-0.00342	28.2	Hessian modified twice; infeasible

Si las componentes de x no tienen límites superiores (o inferiores), entonces *fmincon* prefiere que los correspondientes parámetros *ub* (o *lb*) sean *Inf* (o *-Inf* para *lb*), en contraposición a cualquier otro número por muy alto que éste sea.

En los algoritmos del tipo *MediumScale* los resultados son bastante mejores si se especifican las restricciones utilizando *Aeq* y *beq*, en lugar de implícitamente usando *lb* y *ub*.

Si en nuestro problema están presentes restricciones de tipo ecuación dependientes entre sí, éstas son detectadas y eliminadas. Si esto ocurre, en la salida (solamente si tenemos la opción 'Display' igual a 'iter') mostrará en la primera iteración de nuestro problema, bajo la columna *Procedures*, la etiqueta *dependent* que indicará, tal y como se ha expresado anteriormente, que las restricciones del problema son dependientes entre sí.

Además, después o no de haber reducido el sistema, si Matlab encuentra que las restricciones forman un sistema de ecuaciones incompatible, muestra en las siguientes iteraciones la columna *Procedures* igual a *infeasible*. Es decir, el sistema de restricciones es no consistente.

9.1.3. El Código.

9.1.3.1. Las Restricciones.

Las siguientes líneas de código se encuentran en el fichero *Optima.m*, que albergan tanto la preparación de las restricciones, la llamada a la función *fmincon* y la comprobación de la fiabilidad de la solución.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% m-File : Algoritmo de Optimización mediante funciones
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% de integridad aplicado al problema de Steiner
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

%Obtencion de datos:
%N -> N° de nodos
%V -> N° de arcos(no dirigidos)
%NumTerminales -> N° de Terminales
%NodosTerminales -> Array que contiene cuales son los Nodos Terminales
%PrimeraSol -> Primera solucion(dirigida), obtenida mediante el algoritmo de
%      Kruskal en Visual Basic.

```

```

%Vector -> Array (Vx3) con arcos no dirigidos y su peso.
%origen -> Nodo Origen
[archivo]=textread('PrimeraSol.txt','%s',1);
archivo=str2mat(archivo);
[N,V]=textread(archivo,'%u %u',1);
[Vector(:,1),Vector(:,2),Vector(:,3)]=textread(archivo,'%u %u %u',V,'headerlines',1);
[NumTerminales]=textread(archivo,'%u',1,'headerlines',V+1);
[NodosTerminalesCell]=textread(archivo,'%s',1,'delimiter','\n','headerlines',V+2);
NodosTerminales=str2num(str2mat(NodosTerminalesCell));
[PrimeraSol(:,1),PrimeraSol(:,2),PrimeraSol(:,3)]=textread('PrimeraSol.txt','%u %u %u',V,'headerlines',1);
origen=NodosTerminales(1);

%MatAdy -> Matriz (NxN) con elemento ij igual al coste Cij
MatAdy=zeros(N);
for k=1:1:V;
    MatAdy(Vector(k,1),Vector(k,2))=Vector(k,3);
    MatAdy(Vector(k,2),Vector(k,1))=Vector(k,3);
end

%VectorDir -> Array (2*V x 3) con arcos dirigidos y peso
VectorDir=[Vector;Vector(:,2),Vector(:,1),Vector(:,3)];

%Xij0 -> Solucion Inicial( incluyendo arcos con trafico nulo)
Xij0=zeros(2*V,1);

for k=1:1:size(PrimeraSol,1)
    for j=1:1:size(VectorDir,1)
        if (PrimeraSol(k,1)==VectorDir(j,1) & PrimeraSol(k,2)==VectorDir(j,2))
            Xij0(j)=PrimeraSol(k,3);
        end
    end
end

%Aeq * x = beq -> Implementan restricciones X(j,N)-X(N,j)=...
Aeq=zeros(N,size(VectorDir,1));
beq=zeros(N,1);

```

```

for i=1:1:N
    for j=1:1:size(VectorDir,1)
        if VectorDir(j,1) == i
            Aeq(i,j)=1;
        elseif VectorDir(j,2) == i
            Aeq(i,j)=-1;
        end
    end
end

for j=1:1:size(Aeq,2)
    for k=1:1:NumTerminales
        if ((j==NodosTerminales(k)) & not(j == origen))
            beq(j)=1;
        elseif j==origen
            beq(j) = -NumTerminales + 1;
        end
    end
end

% lb < x < ub
%lb = zeros(size(VectorDir,1),1);
%ub = lb;
%for i=1:1:size(ub)
%    ub(i)=Inf;
%end

CosteSolucion=0;
error=0;

b=zeros(2*V,1);
A=zeros(2*V);
for i=1:1:size(A,2)
    A(i,i)=-1;
end

```


La salida de la optimización se escribe a un fichero, en nuestro caso *resultados.txt*, acumula una línea por cada uno de los problemas que se resuelvan. Así la salida muestra, tanto el nombre del fichero que contiene el problema, el nodo que se ha escogido como nodo origen, el valor de la función objetivo tras ser minimizada, el coste de la función objetivo tras la optimización, el coste óptimo de la misma y el error cometido.

9.1.3.3. Las funciones: función objetivo y función no lineal.

Las funciones *fun_objetivo* y *nlincon* deben estar almacenadas en los ficheros del mismo nombre y con extensión *.m* en el mismo directorio que el resto del código, desde el que se hace la llamada a estas funciones.

```
function [f,g] = fun_objetivo(Xij,coste,NumTerminales)
Fij=((Xij-1).^2)./(Xij.^2+1);
%M=NumTerminales- 1;
%Fij=(Xij.^3-3.*Xij+2)./((M^2-3). *Xij+2);
Fij=Fij';
f = -(Fij*coste);
if nargout > 1
    g=2.*((1-Xij.^2)./(Xij.^2+1).^2));
    %g=g';
    g=-(g.*coste);
end
```

Código. Función *fun_objetivo*.

```
function [C,Ceq,DC,DCEq] = nlincon(Xij,coste,NumTerminales)
C=[];
DC=[];
DCEq=[];
%aux1=Xij(1:size(Xij,1)/2);
%aux2=Xij(size(Xij,1)/2+1:size(Xij,1));
%CEq=aux1*aux2;
Ceq=zeros(size(Xij,1)/2,1);
aux=size(Ceq,1);
for i=1:aux
    Ceq(i)=Xij(i)*Xij(i+size(Xij,1)/2);
end
DCEq=zeros(size(Xij,1),size(Ceq,1));
for i=1:size(Ceq,1)
```

```

    DCeq(i,i)=Xij(i+size(Xij,1)/2);
end
for i=1:1:size(Ceq,1)
    DCeq(i+size(Xij,1)/2,i)=Xij(i);
end

```

Código. Función *nlincon*.

9.1.4. Ejemplo: Detalle de la optimización de *steinb1.txt*.

Vamos a seguir la optimización del problema *steinb1.txt* al detalle, analizando el valor de las variables y de los parámetros conoceremos si nuestro código produce tanto unas restricciones como una función objetivo como las que nosotros queríamos.

Por un lado, se estudiará tanto la salida que produce la ejecución de *fmincon*, como los parámetros que se calculan anteriormente y que dan lugar a las restricciones necesarias en el problema.

En primer lugar, se toma la solución inicial del fichero *PrimeraSol.txt*, el cual, ha sido generado tras la ejecución del programa en Visual Basic que también se ha implementado para este Proyecto. A continuación se muestra dicho fichero para el problema *steinb1.txt*:

```

steinb1.txt
7 9 1
9 3 1
10 8 1
11 12 1
13 12 1
3 1 1
8 4 2
12 6 2
1 5 1
5 6 2
6 4 5

```

Código. Primera Solución *PrimeraSol.txt*.

Se ha comprobado la validez de dicha solución como solución admisible del problema. De hecho, esta solución incluye a todos y cada uno de los nodos terminales del problema, sin la existencia de ninguna rama que no acabe en nodo terminal, y se ha comprobado que el cálculo del flujo que se

hace es el correcto. Más concretamente, el cálculo del flujo que circula por cada arco se ha realizado tomando que todos los nodos terminales, excepto uno, envían una unidad de flujo a ese nodo que acaba recibiendo $-|T|+1$ unidades de flujo.

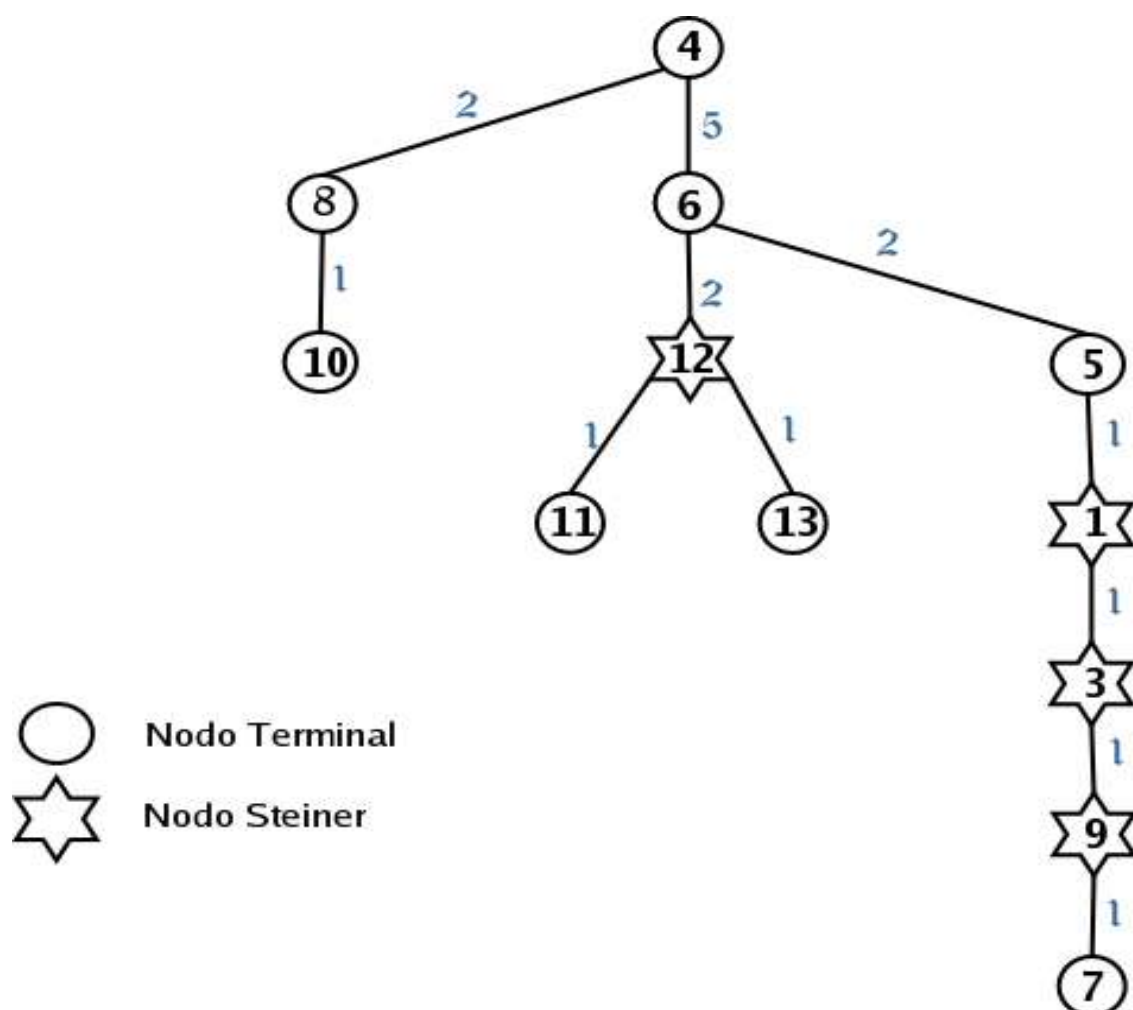


Figura 9.1.1 Primera solución (Resultado del Alg. de Kruskal) en *steinb1.txt*

En la Figura 9.1.1 podemos observar la Primera Solución que obtenemos en *steinb1.txt*, esta Primera Solución es resultado de ejecutar el Alg. de Kruskal y calcular el flujo (azul en la figura) que circula hacia un nodo origen, en nuestro caso el nodo 4.

Ésta es, sin duda, una solución admisible de nuestro problema, puesto que es un árbol que incluye a todos los Nodos Terminales.

<i>VectorDir</i>			<i>Xij0</i>
<i>Nodo Inicial</i>	<i>Nodo Final</i>	<i>Coste</i>	
1	5	7	1
3	9	1	0
3	1	7	1
4	8	2	0
6	4	2	5
6	5	2	0
6	3	8	0
8	10	4	0
9	2	8	0
9	7	5	0
10	2	6	0
11	1	8	0
11	2	14	0
12	6	8	2
12	10	9	0
12	11	2	0
12	13	7	0
12	7	15	0
13	9	11	0
5	1	7	0
9	3	1	1
1	3	7	0
8	4	2	2
4	6	2	0
5	6	2	2
3	6	8	0
10	8	4	1
2	9	8	0
7	9	5	1
2	10	6	0
1	11	8	0
2	11	14	0

<i>VectorDir</i>			<i>Xij0</i>
<i>Nodo Inicial</i>	<i>Nodo Final</i>	<i>Coste</i>	
6	12	8	0
10	12	9	0
11	12	2	1
13	12	7	1
7	12	15	0
9	13	11	0

Tabla 1 *VectorDir* contiene todos los arcos dirigidos del problema, mientras que *Xij0* contiene la solución inicial.

También se puede comprobar como esa primera solución se mapea en un array que concentra todos los arcos del problema. Tendrá la misma longitud que *VectorDir*, que es el vector mostrado en la anterior tabla en las tres primeras columnas y que incluye todos los arcos dirigidos del problema.

Una vez hecho esto, se calculan las matrices *Aeq* y *beq*, que representarán las restricciones lineales de nuestro problema, más concretamente las que se puedan poner en forma de igualdad. Las restricciones de este tipo son:

$$X(j, N) - X(N, j) = \begin{cases} 1 & \text{if } j \in T \\ 0 & \text{if } j \notin T, j \neq e, \forall j \in N \\ -|T| + 1 & \text{if } j = e \end{cases}$$

Por ello, *Aeq* tendrá dimensiones $N \times V$, donde N es el número total de nodos y V será igual al número de arcos dirigidos del problema. En *steinbl.txt*, $N=13$ y $V=38$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
1	1	0	-1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	-1	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	
3	0	1	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	-1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	1	1	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	0	0	0	0	0	0	1	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
8	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	
9	0	-1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	1	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	0	1	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1

Figura 9.1.2 Matriz *Aeq*

Los elementos de *Aeq* sólo pueden ser 1, 0 ó -1. El elemento (i,j) de *Aeq* será:

- 1, si existe el arco dirigido $i \rightarrow j$.
- 0, si no existe $i \rightarrow j$ y el $j \rightarrow i$. En realidad, si existe el uno, existe el otro y

viceversa. Esto es así debido a que se han extraído los arcos dirigidos de los no-dirigidos.

- -1, si existe el arco dirigido $i \rightarrow j$.

La fila correspondiente al nodo 6 está marcada en negrilla. Se puede comprobar que los elementos $(6,5)$, $(6,6)$, $(6,7)$ y $(6,33)$ son iguales a 1, esto quiere decir que en las posiciones 5,6,7 y 33 de *VectorDir* existe un arco con origen el nodo 6.

Sin embargo, los elementos $(6,14)$, $(6,24)$, $(6,25)$ y $(6,26)$ son iguales a -1, las posiciones 14,24,25 y 26 contienen arcos con destino el nodo 6.

El resto de elementos de *Aeq* son nulos.

A continuación se muestra el vector columna *beq*, que es el vector independiente de este sistema de ecuaciones. Tiene dimensiones $N \times 1$, donde N es el número total de nodos.

	1
1	0
2	0
3	0
4	-7
5	1
6	1
7	1
8	1
9	0
10	1
11	1
12	0
13	1

Figura 9.1.3
Elementos del
vector
columna *beq*.

Como se puede observar, los únicos elementos no nulos son aquellos correspondientes a los nodos terminales del problema: 4,5,6,7,8,10,11 y 13. Y de ellos, sólo el nodo 4 es distinto a 1, puesto que es el *nodo origen*, más concretamente su valor es 7, como debe ser.

En cuanto a las restricciones no lineales, se analizan los valores de *Ceq* y *DCeq*, que son los valores de dichas restricciones y sus derivadas respectivamente. Aunque *Ceq* se calcule en cada iteración siempre va a resultar el mismo vector, un vector todo-cero de dimensiones $V \times 1$, donde V es el número de arcos no-dirigidos del problema (en nuestro caso 19) y que se corresponde con el número de restricciones no-lineales del tipo:

$$x_{ij} \cdot x_{ji} = 0, \forall (i, j) \in A$$

Su gradiente es más complicado de entender, se calcula de la siguiente forma:

$$\begin{pmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_2}{\partial x_1} & \dots & \frac{\partial c_N}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} & \frac{\partial c_2}{\partial x_2} & & \\ \vdots & & & \\ \frac{\partial c_1}{\partial x_M} & \dots & \dots & \frac{\partial c_N}{\partial x_M} \end{pmatrix}$$

Por lo tanto se tiene una matriz NxM, con N igual al número de restricciones y M igual al número de variables, 19x38 en nuestro caso.

C_N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
X_M																			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

C _N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
X _M																			
33	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 2 Matriz DCeq

Vamos a analizar el gradiente de las restricciones no lineales para la condición número 5.

$$x_{64} \cdot x_{46} = 0; \forall (i, j) \in A$$

La columna sombreada es la derivada de esta condición. La solución resulta $x_{64}=5$ y $x_{46}=0$, será cuando se derive respecto a x_{46} cuando obtenga el único valor distinto de 0 en toda la columna.

En cuanto a la salida que ofrece la ejecución de la optimización en Matlab, ésta se muestra por la salida estándar y además se escriben los resultados más relevantes al archivo *resultados.txt*.

Lo primero que se obtiene a la salida es el *warning* de que no se puede resolver el problema y que se puede suprimir editando la opción *LargeScale* de *fmincon* a 'off'.

```
>> Warning: Large-scale (trust region) method does not currently solve this type of problem,
switching to medium-scale (line search).
```

Tras esto, si tenemos activada la opción '*Diagnostics*', se obtiene la información de diagnóstico previa a la ejecución de la optimización y que describe:

- El número de variables del problema.
- El método de optimización escogido.
- Como se calcula la matriz Hessiana.
- El número de restricciones de cada clase (ecuaciones e inecuaciones lineales y no lineales).

A continuación, se muestra dicho fragmento de la salida para el problema *steinbl.txt*.

```
%%%%%%%%%
%%%%%%%%%
Diagnostic Information

Number of variables: 38

Functions

Objective and gradient:      fun_objetivo
Hessian:                    finite-differencing (or Quasi-Newton)
Nonlinear constraints and gradient: nlincon
```

*Constraints**Number of nonlinear inequality constraints: 0**Number of nonlinear equality constraints: 19**Number of linear inequality constraints: 0**Number of linear equality constraints: 13**Number of lower bound constraints: 38**Number of upper bound constraints: 0**Algorithm selected**medium-scale*

%%%

End diagnostic information

Después de esto, y debido a que se tiene activada la opción '*DerivativeCheck*', se obtiene la desviación entre el gradiente que se ha proporcionado analíticamente y el gradiente que Matlab calcula numéricamente. Como se puede observar, esta desviación del error es muy pequeña por lo se concluye que el cálculo del gradiente de la función objetivo y de las restricciones no lineales se realiza de forma correcta.

*Function derivative**Maximum discrepancy between derivatives = 2.84217e-006**Constraint derivative**Maximum discrepancy between derivatives = 0*

Tras esto se muestra información sobre la única iteración que realiza en este problema:

	<i>max</i>	<i>Directional</i>	<i>First-order</i>
<i>Iter F-count</i>	<i>f(x)</i>	<i>constraint</i>	<i>Step-size derivative optimality Procedure</i>
<i>Maximum discrepancy between derivatives = 0</i>			
1	41	-208.631	0 1 0 32 dependent

Optimization terminated successfully:

First-order optimality measure less than options.TolFun and

maximum constraint violation is less than options.TolCon

La bondad de la solución inicial es tan buena (ya que el algoritmo de Kruskal, implementado en Visual Basic) y la pobreza del algoritmo al que Matlab está limitado para resolver este problema, hacen que Matlab se “conforme” con esta solución inicial, que no llega a ser la óptima (o cercana a la óptima).

De hecho, en el análisis que la salida proporciona tras la optimización se puede observar como la variable *exitflag* vale 1, con ello Matlab nos informa que el algoritmo utilizado ha convergido, además la Optimización no resulta en ningún error como muestra el anterior fragmento.

```
fval =

-208.6308

exitflag =

1

output =

    iterations: 1
    funcCount: 41
    stepsize: 1
    algorithm: 'medium-scale: SQP, Quasi-Newton, line-search'
firstorderopt: 1.9984e-014
cgiterations: []

lambda =

    lower: [38x1 double]
    upper: [38x1 double]
    eqlin: [13x1 double]
    eqnonlin: [19x1 double]
    ineqlin: [0x1 double]
    ineqnonlin: [0x1 double]
```

Además de este valor, *fmincon* presenta el valor de la función objetivo, el número de iteraciones para llegar al resultado, el número de evaluaciones de la función objetivo y el número de restricciones de cada tipo.

Más allá de los parámetros que proporciona *fmincon*, se calcula si la solución devuelta por la misma cumple todas las restricciones del problema.

En caso de que no cumpla alguna de ellas, se mostrará un mensaje de error en la pantalla,

que dependerá del tipo de restricción que se haya incumplido. En nuestro caso (*steinb1.txt*), no se muestra ningún mensaje de error, por lo que la solución es, al menos, admisible.

En la documentación sobre *fmincon* que acompaña a Matlab encontramos lo siguiente:

The minimization routine appears to enter an infinite loop or returns a solution that does not satisfy the problem constraints. Your objective (fun), constraint (nonlcon, seminfcon), or gradient (computed by fun) functions might be returning Inf, NaN, or complex values. The minimization routines expect only real numbers to be returned. Any other values can cause unexpected results. Insert some checking code into the user-supplied functions to verify that only real numbers are returned (use the function isfinite).²

Que traducido significa que si la rutina de minimización acaba entrando en un bucle infinito o devuelve un valor que no satisface las restricciones se debe comprobar que la función objetivo no esté devolviendo un valor entero, sino *Inf*, *NaN* o un valor complejo.

Sin embargo, este no es nuestro caso, puesto que se puede observar que el valor de la función objetivo es -208.63.

Este ha sido el seguimiento realizado de la optimización bajo Matlab del problema *steinb1.txt*, queda demostrado que las restricciones se calculan correctamente, así como la solución inicial y los gradientes de la función objetivo y de la función no lineal. La admisibilidad de la solución también ha sido comprobada.

9.2. Anexo B: El programa en Visual Basic.

En este apartado se describirá el funcionamiento de la interfaz del programa desarrollado en Visual Basic y haremos un listado del código que implementa las funciones utilizadas en el mismo.

9.2.1. La interfaz.

La interfaz del programa permite realizar las acciones de abrir un archivo, o varios a la vez, que contengan los datos correctos para poder ejecutar la optimización.

Los resultados obtenidos se muestran ordenadamente por pantalla a través de una hoja de cálculo de Microsoft Excel. La ventaja de mostrar los resultados a través de Excel da la facilidad de utilizar todas las funcionalidades de este programa a la hora de editar y salvar la tabla.

² © (1984-2005). The Mathworks, Inc.



Figura 9.2.1 Interfaz del programa

En concreto, se abren dos hojas Excel. Una de ellas con tiene la evolución del Coste de la Solución en cada una de las iteraciones del algoritmo, mientras que la otra únicamente muestra una fila por cada problema y muestra:

- El nombre del problema.
- El número de nodos del mismo.
- El número de arcos.
- El número de terminales.
- El tiempo total de la ejecución (en segundos).
- El tiempo empleado en converger a la solución final (en segundos).
- El número de iteraciones empleadas en encontrar la solución.
- El coste óptimo de la solución.
- El coste de la solución de la optimización implementada.
- El error (%) de la solución.

9.2.2. El Código.

9.2.2.1. El Bucle Principal.

Módulo: bucle principal.bas

```
Option Explicit
Option Base 1

Public Sub Bucle_Iterativo(ByRef Problema As String, ByRef IndiceFila As Integer)
    Dim VectorRecal() As Reg_Arcos
    Dim Solucion() As Reg_Arcos
    Dim CosteSolucion() As Long
    Dim Iteracion As Integer
    Dim CosteActual As Long
    Dim TiempoSim As Single
```

```

Dim TiempoSol As Single
Dim IteracionSol As Integer
Dim Origen As Integer
Dim ProfundidadSol As Integer

Origen = 1
TiempoSim = Timer
Solucion = Vector
VectorRecal = Vector
Calcular_MST Solucion()
Calcular_Pesos Solucion(), VectorRecal(), CosteActual
PrimeraSolucion Solucion(), Problema
Iteracion = 0
ReDim CosteSolucion(1)
CosteSolucion(1) = CosteActual
TiempoSol = Timer - TiempoSim
ProfundidadSol = 1

While CosteOptimo < CosteActual And Iteracion < 35 And ProfundidadSol < 10
    Iteracion = Iteracion + 1
    Hallar_CamMin_Dijkstra VectorRecal(), NumeroNodos, NodosOrigen(), Solucion(), Origen
    Depura_MST Solucion()
    Calcular_Pesos Solucion(), VectorRecal(), CosteActual
    ReDim Preserve CosteSolucion(UBound(CosteSolucion) + 1)

    If Iteracion = 1 Then
        TiempoSol = Timer - TiempoSim
        IteracionSol = 1
    ElseIf CosteActual <> CosteSolucion(Iteracion - 1) Then
        TiempoSol = Timer - TiempoSim
        IteracionSol = Iteracion
    End If

    CosteSolucion(UBound(CosteSolucion)) = CosteActual
    If CosteSolucion(UBound(CosteSolucion)) = CosteSolucion(UBound(CosteSolucion) - 1) Then
        ProfundidadSol = ProfundidadSol + 1
    Else

```

```

    ProfundidadSol = 1
End If
Wend

TiempoSim = Timer - TiempoSim
Anadir_Resul_Informe Problema, TiempoSim, IndiceFila, TiempoSol, IteracionSol, CosteSolucion
End Sub

```

9.2.2.2. La primera solución.

Módulo: Kruskal_MST.bas

```

Option Explicit
Option Base 1

Dim Conj_Mst() As Reg_Conj
Dim ContaConj As Long
Dim Tamano As Long

Public Sub Calcular_MST(ByRef Entrada() As Reg_Arcos)
    Dim i As Long
    Dim SolucionMST() As Reg_Arcos

    Tamano = 0
    ReDim SolucionMST(1)

    QuickSort Entrada, 1, UBound(Entrada)

    Cargar_Arco_Kruskal 1, Entrada(), SolucionMST()
    ContaConj = 1
    ReDim Conj_Mst(2)
    i = 1
    Conj_Mst(i).Nodo = Entrada(1).Origen
    Conj_Mst(i).Conjunto = ContaConj
    i = 2
    Conj_Mst(2).Nodo = Entrada(1).Destino
    Conj_Mst(2).Conjunto = ContaConj

```

```

For i = 2 To UBound(Entrada())
    Buscar_En_Conjunto i, Entrada(), SolucionMST()
Next i
Depura_MST SolucionMST()
Entrada = SolucionMST
End Sub

Private Function Partition(ByRef A() As Reg_Arcos, ByVal Lb As Long, ByVal Ub As Long) As Long
    Dim Temp As Reg_Arcos
    Dim pivot As Reg_Arcos
    Dim i As Long
    Dim j As Long
    Dim p As Long

    p = Lb + (Ub - Lb) \ 2
    pivot = A(p)
    A(p) = A(Lb)

    i = Lb + 1
    j = Ub
    Do
        Do While i < j
            If pivot.Longitud <= A(i).Longitud Then Exit Do
            i = i + 1
        Loop
        Do While j >= i
            If A(j).Longitud <= pivot.Longitud Then Exit Do
            j = j - 1
        Loop
        If i >= j Then Exit Do
        Temp = A(i)
        A(i) = A(j)
        A(j) = Temp
        j = j - 1
        i = i + 1
    Loop

```

```

    A(Lb) = A(j)
    A(j) = pivot
    Partition = j
End Function

Public Sub QuickSort(ByRef A() As Reg_Arcos, ByVal Lb As Long, ByVal Ub As Long)
    Dim M As Long

    Do While Lb < Ub
        M = Partition(A, Lb, Ub)

        If M - Lb <= Ub - M Then
            Call QuickSort(A, Lb, M - 1)
            Lb = M + 1
        Else
            Call QuickSort(A, M + 1, Ub)
            Ub = M - 1
        End If
    Loop
End Sub

Sub Cargar_Arco_Kruskal(ByVal i As Long, ByRef Entrada() As Reg_Arcos, ByRef SolucionMST() As Reg_Arcos)
    Tamano = Tamano + 1
    ReDim Preserve SolucionMST(Tamano)
    SolucionMST(Tamano) = Entrada(i)
End Sub

Sub Buscar_En_Conjunto(ByVal i As Long, ByRef Entrada() As Reg_Arcos, ByRef SolucionMST() As Reg_Arcos)
    Dim j As Long
    Dim k As Long
    Dim Indice As Long
    Dim J1 As Long, J2 As Long
    Dim numero As Long, Numero1 As Long, Numero2 As Long
    Dim Esta1 As Boolean, Esta2 As Boolean

```

```

Esta1 = False
Esta2 = False
j = 1

While (j <= UBound(Conj_Mst())) And (Esta1 = False Or Esta2 = False)
  If Entrada(i).Origen = Conj_Mst(j).Nodo Then
    Esta1 = True
    J1 = Conj_Mst(j).Conjunto
  End If
  If Entrada(i).Destino = Conj_Mst(j).Nodo Then
    Esta2 = True
    J2 = Conj_Mst(j).Conjunto
  End If
  j = j + 1
Wend

If Esta1 = False And Esta2 = False Then
  Indice = UBound(Conj_Mst())
  ReDim Preserve Conj_Mst(Indice + 2)
  ContaConj = ContaConj + 1
  Conj_Mst(Indice + 1).Nodo = Entrada(i).Origen
  Conj_Mst(Indice + 2).Nodo = Entrada(i).Destino
  Conj_Mst(Indice + 1).Conjunto = ContaConj
  Conj_Mst(Indice + 2).Conjunto = ContaConj
  Cargar_Arco_Kruskal i, Entrada(), SolucionMST()
Else
  If Esta1 = True And Esta2 = False Then
    Indice = UBound(Conj_Mst())
    ReDim Preserve Conj_Mst(Indice + 1)
    Conj_Mst(Indice + 1).Nodo = Entrada(i).Destino
    Conj_Mst(Indice + 1).Conjunto = J1
    Cargar_Arco_Kruskal i, Entrada(), SolucionMST()
  Else
    If Esta1 = False And Esta2 = True Then
      Indice = UBound(Conj_Mst())
      ReDim Preserve Conj_Mst(Indice + 1)
      Conj_Mst(Indice + 1).Nodo = Entrada(i).Origen

```

```

    Conj_Mst(Indice + 1).Conjunto = J2
    Cargar_Arco_Kruskal i, Entrada(), SolucionMST()
Else

    If J1 <> J2 Then
        For k = 1 To UBound(Conj_Mst())
            If Conj_Mst(k).Conjunto = J2 Then
                Conj_Mst(k).Conjunto = J1
            End If
        Next
        Cargar_Arco_Kruskal i, Entrada(), SolucionMST()
    End If
End If
End If
End If
End Sub

Public Sub Depura_MST(ByRef Solucion() As Reg_Arcos)
    Dim i As Integer, j As Integer, k As Integer
    Dim Tamano As Integer
    Dim NumAdyacentes As Integer
    Dim Salir As Boolean, Eliminar As Boolean
    Dim NodosAdyMST() As Lista_Ady
    Dim SolucionMST() As Reg_Arcos

    SolucionMST = Solucion
    Calcular_Adyacentes SolucionMST(), NodosAdyMST(), False

    Salir = False
    While Salir = False
        Salir = True
        For i = 1 To UBound(NodosAdyMST)

            If NodosAdyMST(i).numero = 1 Then
                Eliminar = True
                For k = 1 To UBound(NodosOrigen)
                    If i = NodosOrigen(k) Then

```

```

    Eliminar = False
End If
Next k
End If

If Eliminar = True Then
    Salir = False
    For k = 1 To UBound(NodosAdyMST(i).Mady)
        If NodosAdyMST(i).Mady(k).Nodo <> 0 Then
            For j = 1 To UBound(NodosAdyMST(NodosAdyMST(i).Mady(k).Nodo).Mady)
                If NodosAdyMST(NodosAdyMST(i).Mady(k).Nodo).Mady(j).Nodo = i Then
                    NodosAdyMST(NodosAdyMST(i).Mady(k).Nodo).Mady(j).Nodo = 0
                    NodosAdyMST(NodosAdyMST(i).Mady(k).Nodo).numero = NodosAdyMST(NodosAdyMST(i).Mady(k).
NodosAdyMST(i).Mady(k).Nodo).numero - 1
                End If
            Next j
        End If
    Next k
    NodosAdyMST(i).numero = 0
End If
Eliminar = False
Next i
Wend

For i = 1 To UBound(SolucionMST)
    For k = 1 To UBound(NodosAdyMST)
        If NodosAdyMST(k).numero = 0 Then
            If SolucionMST(i).Origen = k Or SolucionMST(i).Destino = k Then
                SolucionMST(i).Inactivo = True
            End If
        End If
    Next k
Next i

j = 1
Tamano = 1
ReDim Solucion(Tamano)
For i = 1 To UBound(SolucionMST)

```

```

If SolucionMST(i).Inactivo = False Then
  ReDim Preserve Solucion(Tamano)
  Tamano = Tamano + 1
  Solucion(j) = SolucionMST(i)
  j = j + 1
End If
Next i
End Sub

```

9.2.2.3. El algoritmo de Dijkstra.

Módulo: Dijkstra.bas

```

Option Explicit
Option Base 1

Private Type Reg_L
  Valor As Long
  Visto As Boolean
End Type

Dim Mat_C() As Long
Dim Mat_L() As Reg_L
Dim NodosAdy() As Lista_Ady

Public Sub Hallar_CamMin_Dijkstra(ByRef Vector() As Reg_Arcos, NumeroNodos As Long, NodosOrigen() As Long, Solucion() As Reg_Arcos, Origen As Integer)
  Dim i As Integer, j As Integer
  Dim D() As Single

  Calcular_Matriz_Pesos Vector, NumeroNodos, D()
  Calcular_Adyacentes Vector, NodosAdy, False

  Dijkstra NodosOrigen(Origen), NumeroNodos, D()
  Interpreta_MatrizAdy Vector, Solucion, D(), NodosOrigen(Origen)

  If Origen < UBound(NodosOrigen) Then

```

```

    Origen = Origen + 1
Else
    Origen = 1
End If

End Sub

Sub Calcular_Matriz_Pesos(ByRef Vector() As Reg_Arcos, ByVal NumeroNodos As Long, D() As Single)
    Dim i As Long, j As Long
    ReDim D(NumeroNodos, NumeroNodos, 3)
    For i = 1 To UBound(Vector())
        If Vector(i).Inactivo = False Then
            D(Vector(i).Origen, Vector(i).Destino, 1) = Vector(i).Longitud
            D(Vector(i).Destino, Vector(i).Origen, 1) = Vector(i).Longitud
            D(Vector(i).Origen, Vector(i).Destino, 3) = i
            D(Vector(i).Destino, Vector(i).Origen, 3) = i
        End If
    Next
    For i = 1 To NumeroNodos - 1
        For j = i + 1 To NumeroNodos
            If D(i, j, 3) = 0 Then
                D(i, j, 1) = Exp(16)
                D(j, i, 1) = Exp(16)
            End If
        Next
    Next
End Sub

Public Sub Calcular_Adyacentes(ByRef Vector() As Reg_Arcos, ByRef NodosAdy() As Lista_Ady, ByVal CalcularPosicion As Boolean)
    Dim j As Long

    ReDim NodosAdy(NumeroNodos)
    For j = 1 To UBound(Vector)
        NodosAdy(Vector(j).Origen).numero = NodosAdy(Vector(j).Origen).numero + 1
        ReDim Preserve NodosAdy(Vector(j).Origen).Mady(NodosAdy(Vector(j).Origen).numero)
        NodosAdy(Vector(j).Origen).Mady(NodosAdy(Vector(j).Origen).numero).Nodo = Vector(j).Destino
    Next
End Sub

```

```

NodosAdy(Vector(j).Destino).numero = NodosAdy(Vector(j).Destino).numero + 1
ReDim Preserve NodosAdy(Vector(j).Destino).Mady(NodosAdy(Vector(j).Destino).numero)
NodosAdy(Vector(j).Destino).Mady(NodosAdy(Vector(j).Destino).numero).Nodo = Vector(j).Origen

If CalcularPosicion = True Then
    NodosAdy(Vector(j).Origen).Mady(NodosAdy(Vector(j).Origen).numero).Posicion = j
    NodosAdy(Vector(j).Destino).Mady(NodosAdy(Vector(j).Destino).numero).Posicion = j
End If
Next
End Sub

Sub Dijkstra(ByVal Origen As Long, ByVal NumeroNodos As Long, ByRef D() As Single)
    Dim i As Integer, j As Integer
    Inicializa_Matrices Origen, NumeroNodos, D()
    For i = 1 To UBound(Mat_C)
        j = Elige_Menor(Origen)
        Iteracion_Algoritmo j, Origen, D()
    Next
End Sub

Sub Inicializa_Matrices(ByVal Origen As Long, ByVal NumeroNodos As Long, ByRef D() As Single)
    Dim l As Long
    Dim i As Integer, j As Integer, k As Integer

    ReDim Mat_C(NumeroNodos - 1)
    i = 1
    j = 1

    While i <= UBound(Mat_C())
        If j = Origen Then
            j = j + 1
            Mat_C(i) = j
        Else
            Mat_C(i) = j
        End If
        i = i + 1
    
```

```

j = j + 1
Wend

ReDim Mat_L(NúmeroNodos - 1)
For i = 1 To UBound(Mat_L())
    Mat_L(i).Valor = D(Origen, Mat_C(i), 1)
    Mat_L(i).Visto = False
    For j = 1 To NodosAdy(Mat_C(i)).numero
        For k = 1 To NodosAdy(NodosAdy(Mat_C(i)).Mady(j).Nodo).numero
            If NodosAdy(NodosAdy(Mat_C(i)).Mady(j).Nodo).Mady(k).Nodo = Mat_C(i) Then
                NodosAdy(NodosAdy(Mat_C(i)).Mady(j).Nodo).Mady(k).Posicion = i
            End If
        Next k
    Next j
Next i
End Sub

Function Elige_Menor(ByVal Origen As Long) As Long
    Dim i As Long
    Dim Minimo As Double
    Minimo = Exp(23)
    For i = 1 To UBound(Mat_L)
        If Mat_L(i).Visto = False And Mat_L(i).Valor < Minimo Then
            Minimo = Mat_L(i).Valor
            Elige_Menor = i
        End If
    Next
    Mat_L(Elige_Menor).Visto = True
End Function

Sub Iteracion_Algoritmo(ByVal Pos As Integer, ByVal Origen As Integer, ByRef D() As Single)
    Dim i As Integer
    Dim A As Integer, B As Double, C As Double
    For i = 1 To NodosAdy(Mat_C(Pos)).numero
        A = NodosAdy(Mat_C(Pos)).Mady(i).Nodo
        If A <> 0 Then

```

```

If A <> Origen Then
  If Mat_L(NodosAdy(Mat_C(Pos)).Mady(i).Posicion).Visto = False Then
    B = Mat_L(Pos).Valor
    C = Mat_L(NodosAdy(Mat_C(Pos)).Mady(i).Posicion).Valor
    If C >= B + D(Mat_C(Pos), Mat_C(NodosAdy(Mat_C(Pos)).Mady(i).Posicion), 1) Then
      Mat_L(NodosAdy(Mat_C(Pos)).Mady(i).Posicion).Valor = B + D(Mat_C(Pos), Mat_C(NodosAdy(Mat_C
(Pos)).Mady(i).Posicion), 1)
      D(Origen, Mat_C(NodosAdy(Mat_C(Pos)).Mady(i).Posicion), 2) = Mat_C(Pos)
      D(Origen, Mat_C(NodosAdy(Mat_C(Pos)).Mady(i).Posicion), 1) = Mat_L(NodosAdy(Mat_C(Pos)).Mady
(i).Posicion).Valor
    End If
  End If
End If
Else
  Exit Sub
End If
Next i
End Sub

Private Sub Interpreta_MatrizAdy(ByRef Vector() As Reg_Arcos, ByRef Solucion() As Reg_Arcos, ByRef D() As
Single, ByRef Origen As Long)
  Dim i As Integer, j As Integer
  Dim NodoIntermedio As Integer, NodoIntermedio2 As Integer

  ReDim Solucion(1)

  For j = 1 To NumeroNodos
    If j <> Origen Then
      NodoIntermedio = D(Origen, j, 2)
      If NodoIntermedio = 0 Then
        AnadirArco Vector, Solucion, Origen, j
        ' Solucion(UBound(Solucion)) = Vector(D(Origen, NodosOrigen(j), 3))
        ' ReDim Preserve Solucion(UBound(Solucion) + 1)
      Else
        AnadirArco Vector, Solucion, j, NodoIntermedio
        While NodoIntermedio <> 0
          NodoIntermedio2 = NodoIntermedio
          NodoIntermedio = D(Origen, NodoIntermedio2, 2)
        End While
        If NodoIntermedio <> 0 Then AnadirArco Vector, Solucion, NodoIntermedio, NodoIntermedio2
      End If
    End If
  Next j
End Sub

```

```

Wend
    AnadirArco Vector, Solucion, Origen, NodoIntermedio2
End If
End If
Next j
ReDim Preserve Solucion(UBound(Solucion) - 1)
End Sub

Private Sub AnadirArco(ByRef Vector() As Reg_Arcos, ByRef Solucion() As Reg_Arcos, ByVal Origen As Integer, ByVal Destino As Integer)
    Dim Encontrado As Boolean
    Dim i As Integer, j As Integer
    Dim Anadido As Boolean

    Anadido = False
    Encontrado = False
    i = 0
    While Encontrado = False
        i = i + 1
        If Vector(i).Origen = Origen And Vector(i).Destino = Destino Then Encontrado = True
        If Vector(i).Destino = Origen And Vector(i).Origen = Destino Then Encontrado = True
    Wend

    For j = 1 To UBound(Solucion)
        If Vector(i).Posicion = Solucion(j).Posicion Then Anadido = True
    Next j

    If Anadido = False Then
        Solucion(UBound(Solucion)) = Vector(i)
        ReDim Preserve Solucion(UBound(Solucion) + 1)
    End If
End Sub

```

9.2.2.4. Recalcular los Pesos.

Método: Recalcular_Pesos.bas

```
Option Explicit
```

Option Base 1

```
Public Sub Calcular_Pesos(ByRef Solucion() As Reg_Arcos, ByRef VectorRecal() As Reg_Arcos, ByRef CosteTotal As Long)
```

```
    Dim Xij() As Double
```

```
    Dim Fij() As Double
```

```
    Dim i As Integer
```

```
    Dim A As Long, M As Long
```

```
    Dim Minimo As Long
```

```
    Calcular_Flujo Solucion(), Xij()
```

```
    ReDim Fij(UBound(Xij))
```

```
    For i = 1 To UBound(Xij)
```

```
        If i = 1 Then
```

```
            Minimo = Xij(i)
```

```
        Else
```

```
            If Minimo > Xij(i) Then
```

```
                Minimo = Xij(i)
```

```
            End If
```

```
        End If
```

```
    Next i
```

```
    For i = 1 To UBound(Xij)
```

```
        Xij(i) = Xij(i) / Minimo
```

```
    Next i
```

```
    M = UBound(NodosOrigen) - 1
```

```
    A = (M ^ 2 - 3)
```

```
    For i = 1 To UBound(Xij)
```

```
        ' Fij(i) = 1 - (Xij(i) ^ 3 - 3 * Xij(i) + 2) / (A * Xij(i) + 2)
```

```
        Fij(i) = 1 - (Xij(i) - 1) ^ 2 / (Xij(i) ^ 2 + 1)
```

```
        ' Fij(i) = 1 - (Exp(Log(2) * (1 - Xij(i))) - 1) ^ 2
```

```
        Fij(i) = Solucion(i).Longitud * Fij(i)
```

```
        VectorRecal(Solucion(i).Posicion).Longitud = Fij(i)
```

```
    Next i
```

```
    CosteTotal = 0
```

```
    For i = 1 To UBound(Solucion)
```

```

CosteTotal = CosteTotal + Vector(Solucion(i).Posicion).Longitud
Next i
End Sub

Public Sub Calcular_Flujo(ByRef Solucion() As Reg_Arcos, ByRef Xij() As Double)
Dim NodosAdyReC() As Lista_Ady
Dim Trafico() As Long
Dim Salir As Boolean, Eliminar As Boolean, Encontrado As Boolean
Dim i As Integer, j As Integer, k As Integer, h As Integer
Dim Destino As Integer

ReDim Xij(UBound(Solucion))

For h = 1 To UBound(NodosOrigen) 'Este for lo he añadido para calcular tráfico con broadcasting

Calcular_Adyacentes Solucion(), NodosAdyReC(), True
ReDim Trafico(UBound(NodosAdyReC))
Salir = False

While Salir = False
Salir = True

For i = 1 To UBound(NodosAdyReC)
If NodosAdyReC(i).numero = 1 And i <> NodosOrigen(h) Then
Eliminar = True

For k = 1 To UBound(NodosOrigen)
If i = NodosOrigen(k) Then
Trafico(i) = Trafico(i) + 1
End If
Next k
End If

If Eliminar = True Then
For k = 1 To UBound(NodosAdyReC(i).Mady)

If NodosAdyReC(i).Mady(k).Nodo <> 0 Then

```

```

    Trafico(Nodos.AdyReC(i).Mady(k).Nodo) = Trafico(Nodos.AdyReC(i).Mady(k).Nodo) + Trafico(i)
    For j = 1 To UBound(Nodos.AdyReC(Nodos.AdyReC(i).Mady(k).Nodo).Mady)
        If Nodos.AdyReC(Nodos.AdyReC(i).Mady(k).Nodo).Mady(j).Nodo = i Then
            Nodos.AdyReC(Nodos.AdyReC(i).Mady(k).Nodo).numero = Nodos.AdyReC(Nodos.AdyReC(i).Mady(k).
Nodo).numero - 1
            Nodos.AdyReC(Nodos.AdyReC(i).Mady(k).Nodo).Mady(j).Nodo = 0
            Destino = Nodos.AdyReC(i).Mady(k).Nodo
        End If
    Next j
    Nodos.AdyReC(i).numero = Nodos.AdyReC(i).numero - 1

    Xij(Nodos.AdyReC(i).Mady(k).Posicion) = Xij(Nodos.AdyReC(i).Mady(k).Posicion) + Trafico(i)

End If
Next k
Salir = False
End If

Eliminar = False
Encontrado = False

Next i
Wend
Next h
End Sub

Public Sub PrimeraSolucion(ByRef Solucion() As Reg_Arcos, ByRef Problema As String)
    Dim Xij0() As Double
    Dim Xij() As Double
    Dim NodosAdyReC() As Lista_Ady
    Dim Salir As Boolean
    Dim i As Integer, j As Integer, k As Integer
    Dim Destino As Integer
    Dim indic As Integer
    Dim nFile As Integer
    Dim sFile As String
    sFile = "PrimeraSol.txt"
    nFile = FreeFile

```

```

indic = 1
ReDim Xij0(UBound(Solucion), 3)
ReDim Xij(UBound(Solucion))
Calcular_AdYacentes Solucion(), NodosAdyReC(), True
ReDim Trafico(UBound(NodosAdyReC))

Salir = False

While Salir = False
  Salir = True

  For i = 1 To UBound(NodosAdyReC)
    If NodosAdyReC(i).numero = 1 And i <> NodosOrigen(1) Then
      For k = 1 To UBound(NodosOrigen)
        If i = NodosOrigen(k) Then
          Trafico(i) = Trafico(i) + 1
        End If
      Next k

      For k = 1 To UBound(NodosAdyReC(i).Mady)

        If NodosAdyReC(i).Mady(k).Nodo <> 0 Then
          Trafico(NodosAdyReC(i).Mady(k).Nodo) = Trafico(NodosAdyReC(i).Mady(k).Nodo) + Trafico(i)
          For j = 1 To UBound(NodosAdyReC(NodosAdyReC(i).Mady(k).Nodo).Mady)
            If NodosAdyReC(NodosAdyReC(i).Mady(k).Nodo).Mady(j).Nodo = i Then
              NodosAdyReC(NodosAdyReC(i).Mady(k).Nodo).numero = NodosAdyReC(NodosAdyReC(i).Mady(k).
Nodo).numero - 1
              NodosAdyReC(NodosAdyReC(i).Mady(k).Nodo).Mady(j).Nodo = 0
              Destino = NodosAdyReC(i).Mady(k).Nodo
            End If
          Next j
          NodosAdyReC(i).numero = NodosAdyReC(i).numero - 1

          Xij(NodosAdyReC(i).Mady(k).Posicion) = Xij(NodosAdyReC(i).Mady(k).Posicion) + Trafico(i)

          Xij0(indic, 1) = i 'Xij0 solo se utiliza en MATLAB
          Xij0(indic, 2) = Destino
        End If
      Next k
    End If
  Next i
End While

```

```

Xij0(indic, 3) = Xij(NodosAdyReC(i).Mady(k).Posicion)
indic = indic + 1

End If
Next k
Salir = False
End If
Next i
Wend
Open sFile For Output As #nFile

Print #nFile, Problema

For i = 1 To UBound(Solucion)
Print #nFile, Xij0(i, 1) & " " & Xij0(i, 2) & " " & Xij0(i, 3)
Next i

Close #nFile
End Sub

```

9.2.2.5. Interpretar el archivo.

Método: Interpreta_archivo.bas

```

Option Explicit
Option Base 1

Public Sub AnalizaArchivo(sFile As String)
Dim i As Long
Dim nFile As Long
Dim lNumNodosOrigen As Long
Dim lNumArcos As Long
Dim sTemp As String
Dim lStartPos As Long
Dim lEndPos As Long

nFile = FreeFile

```

```

If bFileExists(sFile) = True Then
    Open sFile For Input As #nFile
    Input #1, NumeroNodos, lNumArcos
    ReDim Vector(lNumArcos)
    For i = 1 To lNumArcos
        Input #nFile, Vector(i).Origen, Vector(i).Destino, Vector(i).Longitud
        Vector(i).Posicion = i
    Next i
    Input #nFile, lNumNodosOrigen
    ReDim NodosOrigen(lNumNodosOrigen)
    Input #nFile, sTemp
    While Left$(sTemp, 1) = ESPACIO
        LTrim (sTemp)
    Wend

    lStartPos = 1
    For i = 1 To (lNumNodosOrigen - 1)
        lEndPos = InStr(lStartPos, sTemp, ESPACIO)
        NodosOrigen(i) = Val(Mid(sTemp, lStartPos, lEndPos - lStartPos))
        lStartPos = lEndPos + 1
    Next i
    NodosOrigen(i) = Val(Mid(sTemp, lStartPos))
    If Not EOF(nFile) Then
        Input #nFile, CosteOptimo
    Else
        CosteOptimo = -1
    End If
    Close #nFile
End If
End Sub

Public Function bFileExists(sFile As String) As Boolean
    If sFile = "" Then bFileExists = False Else bFileExists = True
End Function

Public Function GetFiles(Optional ByVal sTitle As String = "Open files...") As String

```

```

Dim sFilenames As String
Dim cdlOpen As Object
On Error GoTo ProcError
Set cdlOpen = CreateObject("MSComDlg.CommonDialog")
With cdlOpen
    .CancelError = True
    .Filter = "Documentos de Texto (*.txt) |*.txt|Todos los archivos (*.*) |*.*"
    .FilterIndex = 1
    .DialogTitle = sTitle
    .MaxFileSize = &H7FFF
    ' same as .Flags = cdlOFNHideReadOnly Or cdlOFNPathMustExist Or cdlOFNLongNames Or
    cdlOFNAllowMultiselect or cdlOFNExplorer
    .Flags = &H4 Or &H800 Or &H40000 Or &H200 Or &H80000
    .ShowOpen
    sFilenames = .FileName
End With
ProcExit:
    GetFiles = sFilenames
    Set cdlOpen = Nothing
    Exit Function
ProcError:
    If Err.Number = &H7FF3 Then Resume Next 'Cancel selected - Ignore
    MsgBox Err.Description & "(" & Err.Number & ")", vbExclamation, "Open error"
    sFilenames = ""
    Resume ProcExit
End Function

Public Sub ExtraerRutas(ByRef sFilenames As String, ByRef ListaArchivos() As String, ByRef Archivos() As String)
    Dim i As Integer
    Dim AuxString() As String

    AuxString = Split(sFilenames, Chr(0))

    If UBound(AuxString) <> -1 Then
        If UBound(AuxString) = 0 Then
            ReDim ListaArchivos(1)
            ListaArchivos(1) = AuxString(0)
        End If
    End If

```

```
Else
  ReDim ListaArchivos(UBound(AuxString))

  If UBound(AuxString) > 0 Then
    For i = 1 To UBound(AuxString)
      ListaArchivos(i) = AuxString(0) & "\" & AuxString(i)
    Next i
  End If
End If

AuxString = Split(sFilenames, Chr(Asc("\")))
sFilenames = AuxString(UBound(AuxString))

AuxString = Split(sFilenames, Chr(0))

If UBound(AuxString) = 0 Then
  ReDim Archivos(1)
  Archivos(1) = AuxString(0)
Else
  ReDim Archivos(UBound(AuxString))
  For i = 1 To UBound(AuxString)
    Archivos(i) = AuxString(i)
  Next i
End If
End If

End Sub
```

9.2.2.6. Generar la salida.

Método: GenerarInforme.bas.

```
Option Explicit
Option Base 1

Public Sub Generar_Informe(ByRef Archivos() As String)
```

```
Dim i As Integer
Set Inf_Principal = New Excel.Application
Inf_Principal.Visible = True
Inf_Principal.SheetsInNewWorkbook = 1
Inf_Principal.Workbooks.Add

Set Inf_Iteraciones = New Excel.Application
Inf_Iteraciones.Visible = True
Inf_Iteraciones.SheetsInNewWorkbook = 1
Inf_Iteraciones.Workbooks.Add

With Inf_Principal.ActiveSheet
    .Rows(1).AutoFormat 14
    .Rows(1).Font.Bold = True

    .Cells(1, 1).Value = "Problema"
    .Cells(1, 2).Value = "Num.Nodos"
    .Cells(1, 3).Value = "Num.Arcos"
    .Cells(1, 4).Value = "Num.Terminales"
    .Cells(1, 5).Value = "Tiempo Simul"
    .Cells(1, 6).Value = "Tiempo Sol"
    .Cells(1, 7).Value = "Iteraciones Sol"
    .Cells(1, 8).Value = "Coste Óptimo"
    .Cells(1, 9).Value = "Coste Solución"
    .Cells(1, 10).Value = "Error (%)"

    .Columns.AutoFit
End With

With Inf_Iteraciones.ActiveSheet
    .Rows(1).AutoFormat 14
    .Rows(1).Font.Bold = True

    For i = 1 To UBound(Archivos)
        .Cells(1, i).Value = Archivos(i)
```

```

.Columns.AutoFit
Next i
.Columns.AutoFit
End With

End Sub

Public Sub Anadir_Resul_Informe(ByRef Problema As String, ByRef TiempoSim As Single, ByRef Indice As Integer, ByRef TiempoSol As Single, ByRef IteracionSol As Integer, ByRef CosteSolucion() As Long)
    Dim i As Integer
    Dim HorasSim As Integer, MinSim As Integer, SegSim As Integer
    Dim HorasSol As Integer, MinSol As Integer, SegSol As Integer

    HorasSim = Int(TiempoSim / 3600)
    MinSim = Int((TiempoSim - HorasSim * 3600) / 60)
    SegSim = Int(TiempoSim - HorasSim * 3600 - MinSim * 60)

    HorasSol = Int(TiempoSol / 3600)
    MinSol = Int((TiempoSol - HorasSol * 3600) / 60)
    SegSol = Int(TiempoSol - HorasSol * 3600 - MinSol * 60)

    With Inf_Principal.ActiveSheet
        .Cells(Indice + 1, 1) = Problema
        .Cells(Indice + 1, 2) = NumeroNodos
        .Cells(Indice + 1, 3) = UBound(Vector)
        .Cells(Indice + 1, 4) = UBound(NodosOrigen)
        .Cells(Indice + 1, 5) = HorasSim & ":" & MinSim & ":" & SegSim
        .Cells(Indice + 1, 6) = HorasSol & ":" & MinSol & ":" & SegSol
        .Cells(Indice + 1, 7) = IteracionSol
        .Cells(Indice + 1, 8) = CosteOptimo
        .Cells(Indice + 1, 9) = CosteSolucion(UBound(CosteSolucion))
        .Cells(Indice + 1, 10) = ((CosteSolucion(UBound(CosteSolucion)) - CosteOptimo) / CosteOptimo) * 100

        .Columns.AutoFit
    End With

    With Inf_Iteraciones.ActiveSheet

```

```
For i = 1 To UBound(CosteSolucion)
    .Cells(i + 1, Indice) = CosteSolucion(i)
Next i
End With
End Sub
```

9.2.2.7. Definición de constantes y declaración de variables globales.

Método: Declaraciones.bas.

```
Option Explicit

Public Type Reg_Arcos
    Origen As Long
    Destino As Long
    Inactivo As Boolean
    Posicion As Long
    Longitud As Single
End Type

Public Type Reg_Conj
    Nodo As Long
    Conjunto As Long
End Type

Public Type MatrizAdy
    Nodo As Long
    Posicion As Long
End Type

Public Type Lista_Ady
    numero As Long
    Mady() As MatrizAdy
End Type

Public Const ESPACIO = " "
```

```
Public Vector() As Reg_Arcos  
Public NumeroNodos As Long  
Public NodosOrigen() As Long  
Public CosteOptimo As Long  
Public Inf_Principal As Excel.Application  
Public Inf_Iteraciones As Excel.Application
```