PRÁCTICAS DE AMPLIACION DE ROBÓTICA

MARTA ROSA FLORES

DANIEL TRIANO MORENO

Índice:

[EJERCICIO 1 3](#_Toc65138959)

[Solución 1.1: 3](#_Toc65138960)

# EJERCICIO 1

Se desea simular el movimiento de un vehículo con tracción diferencial con los siguientes parámetros: distancia entre ruedas 0.8 m, radio de las ruedas 0.1 m, velocidad máxima de las ruedas 15 rad/s. Para ello se tendrán que implantar las ecuaciones de movimiento así como el modelo dinámico simplificado de los dos actuadores. Se considerará una constante de tiempo de 0.12 s, ganancia unidad. Se pide:

1. Simular el movimiento en tiempo discreto con periodo de T = 25 ms partiendo desde parado y aplicando actuaciones constantes. Considerar las siguientes situaciones: trayectoria recta, giro a la izquierda, giro a la derecha y velocidad lineal nula.

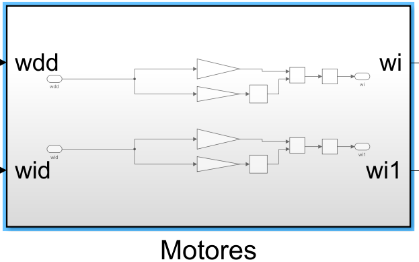
## Solución 1.1:

Antes de realizar alguna explicación de la solución obtenida, se van a enumerar las fórmulas necesarias para una correcta explicación:

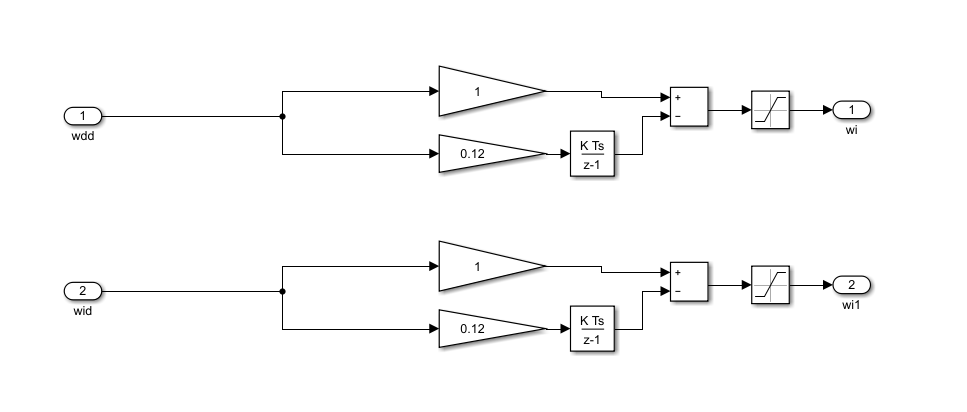
|  |  |
| --- | --- |
|  | (1) |
|  | (2) |
|  | (3) |

|  |  |
| --- | --- |
|  | (4) |
| 𝛥𝑠 = 𝑣 ⋅ 𝛥𝑡 | (5) |
| 𝛥𝜙 = 𝜔 ⋅ 𝛥𝑡 | (6) |
| 𝛥𝑥 = −𝑠𝑖𝑛(𝜙) ∙ 𝛥𝑠 | (7) |

Para poder resolver este apartado se han realizado distintos bloques. El primer de ellos serían los motores (ver **Figura 1**) y para ello se tratará de implementar las ecuaciones (3) y (4), tal y como muestra la **Figura 2**.



**Figura 1: Imagen del sub sistema “motores”**



**Figura 2: Interior del sub sistema “motores”**

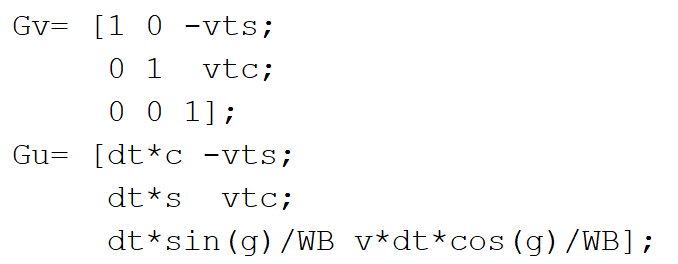
Tras una inspección superficial de las ecuaciones (3) y (4) se puede ver que son simétricas, donde simplemente varían las velocidades, izquierda para la primera y derecha para la segunda.

Para el caso de la rueda derecha se multiplica por una ganancia unitaria y se resta a la constante de tiempo discretizada. Se realiza el mismo procedimiento para ambos casos si bien al final del recorrido se le añade una saturación en 15 rad/s.

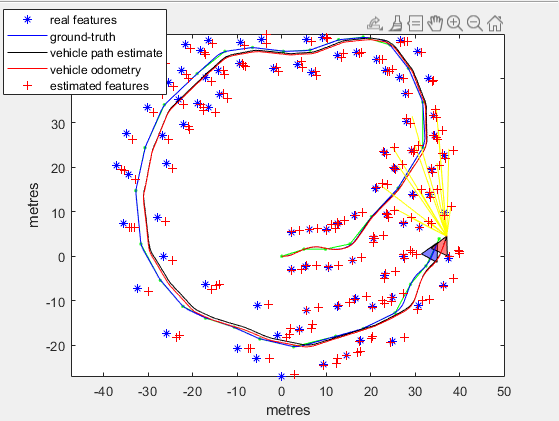
# EJERCICIO 2

1. **Completar la fase de predicción (función predict) con las matrices jacobianas correspondientes al modelo de conducción anterior.**

En primer lugar, se carga el mapa a recorrer tal y como se comenta en el enunciado. Si se acude a la función predict se puede observar como están definidas las drivadas parciales en función de u[V,G], tal y como muestra loa figura de abajo:



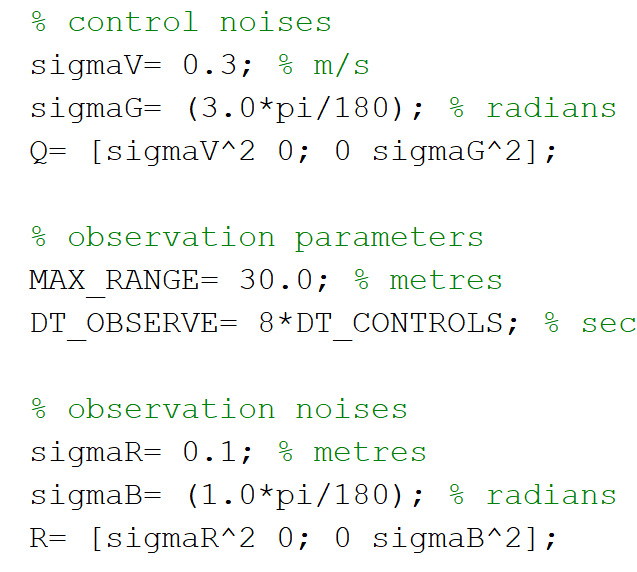
Tras ejecutar la función **ekfloc(lm,wp)** (ver figura de abajo), se analizan los siguientes resultados.



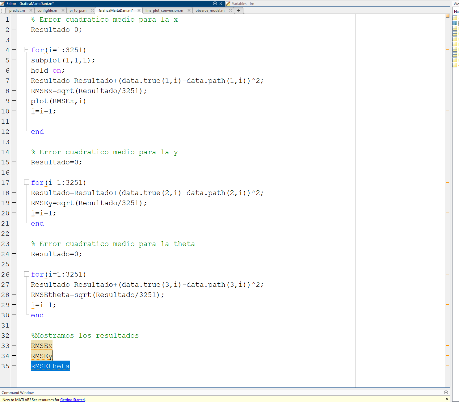
Se observa una cierta discrepancia entre la odometría y el camino recorrido, así como las marcas de entorno previstas y las reales.

1. **Calcular el error cuadrático medio en la distancia y ángulo de orientación a lo largo de la trayectoria, según la ecuación:**

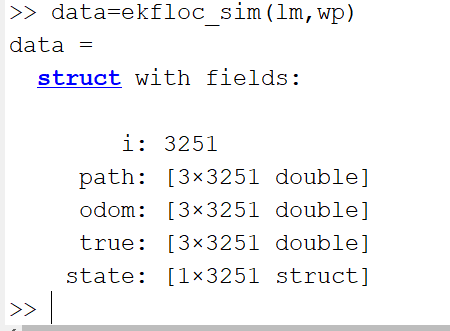
Si queremos analizar cómo afectas los distintos parámetros de ruido, ya sea de control o de observación, debemos acudir al archivo ***configfile,*** donde se modificar los parámetros correspondientes a las distintas sigmas (ver figura de abajo).



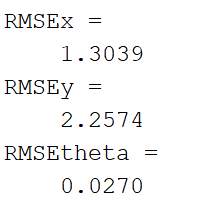
Para ver el error medio tanto en X, en Y como en Theta, se ha implementado la función llamada ***graficaMartaDani*** (ver figura de abajo),



Donde simplemente cogemos la columna correspondiente a cada variable para poder calcular su error. Se utiliza un bucle for de 3251 iteraciones dado que ese es el tamaño de nuestra función (ver figura de abajo):



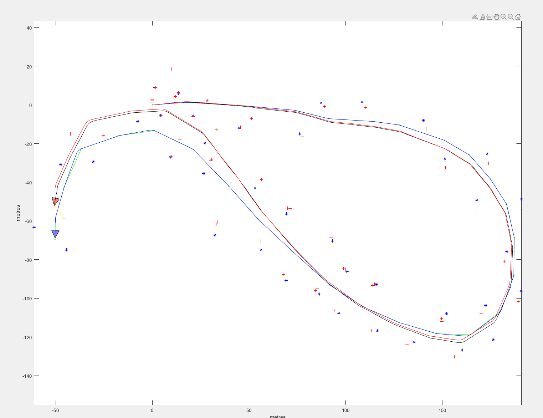
Así pues, se observa el error sin modificar ningún parámetro, obteniendo los siguientes resultados ver figura abajo:



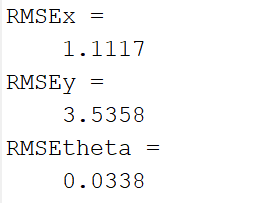
Se observa un mayor error en Y, y casi nulo en theta.

**3. Realizar simulaciones para distintos valores de incertidumbre en el control del vehículo y de la observación (al menos un cambio en cada uno de ellos). Modificar dichas incertidumbres en el fichero de parámetros (configfile). Estudiar su influencia en el resultado calculando el error cuadrático medio en cada caso y comentar las conclusiones.**

Si ahora se aumenta el valor de SigmaV por tres veces el valor anterior, se observa la figura de abajo:

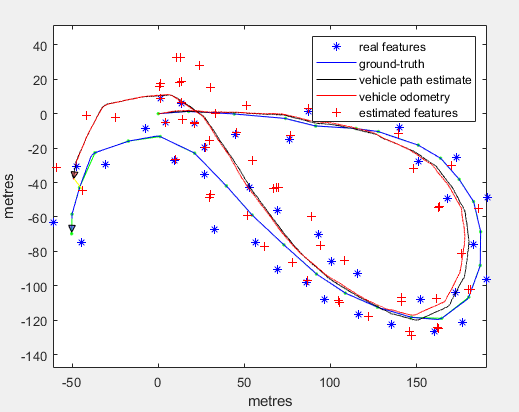


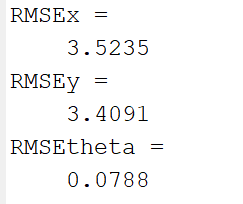
Se puede ver a simple vista y sin realizar cálculos que el error ha aumentado, sobre todo en el eje Y donde se dispersan más los puntos. Comprobamos eso en al calcularlo (ver figura de abajo):

pasamos de sigmaV 0.3 a 0.9 poner pie de foto

Confirmamos la previsión, el error en Y ha aumentado al aumentar el ruido de control SigmaV.

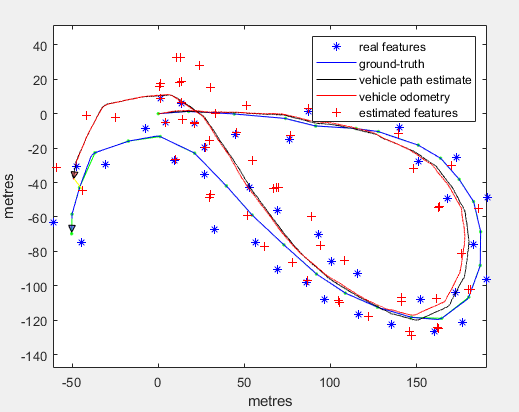
Si ahora modificamos SigmaG (ver foto de abajo), vemos que no solo aumenta el error en Y si no que también aumenta en X (ver figura de aun mas abajo xd).

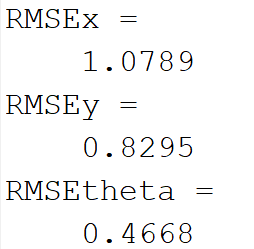
 sigma cambia de sigmaG= (3.0*pi/180); % radians POR sigmaG= (12.0*pi/180



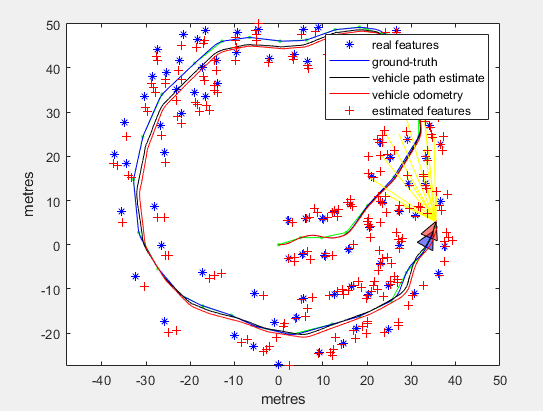
A su vez el error en theta duplica con respecto al anterior caso.

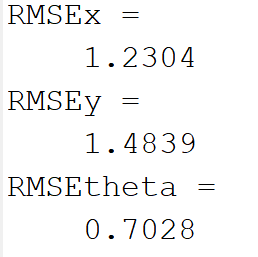
Modificamos a continuación los ruidos de observación, pasando de un SigmaR=0.1 a SigmaR=0.5 metros. Se observa la siguiente gráfica (ver abajo) donde claramente el error es menor respecto a las anteriores. Todo esto se confirma al ver el error real (figura de mas abajo)



resultados

Al realizar la ultima modificación, pasando a una sigmaB tres veces mayor, se observa un aumento del error en theta bastante notable. El error en X y en Y también aumenta pero en menor proporción. Todo esto se refleja en las foto de abajo



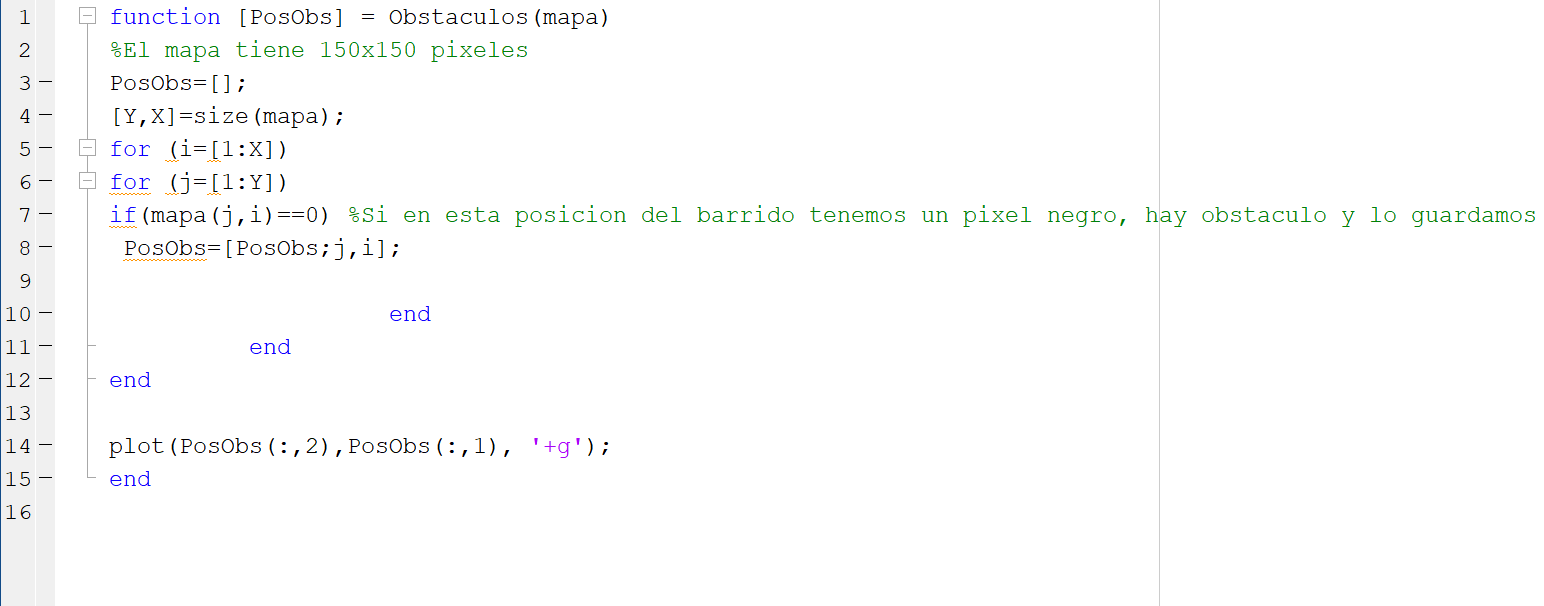


Como conclusión podemos ver que al aumentar el ruido la posición real del robot varía más que el camino que sigue la odometría. Si disminuimos el error se acercará por lo tanto al camino de la odometría, aumentando así la exactitud.

# EJERCICIO 3

**1. Completar el script denominado plantilla para conseguir la navegación reactiva del robot según el método de campos potenciales.**

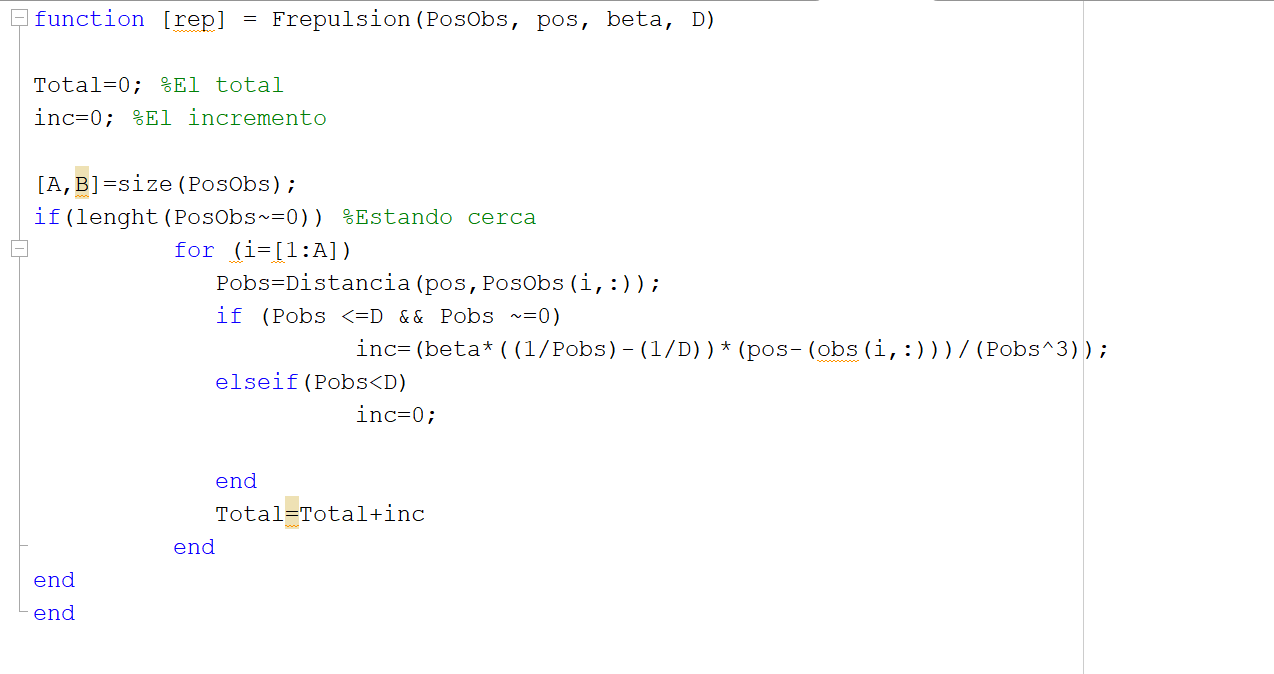
Se ha hecho una función que se encarga de detectar los obstáculos del mapa, para ello hemos hecho dos bucles for de una amplitud X e Y, que serán las proporciones del mapa seleccionado. Si se detecta un pixel de valor de valor 0, significará que se ha encontrado un pixel negro y por lo tanto un obstáculo (ver figura de abajo poner ref):



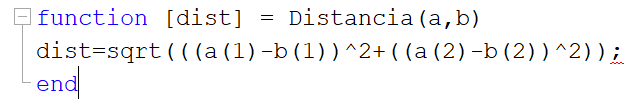
Mediante las siguientes ecuaciones, proporcionadas en las diapositivas de clase, se han implementado dos funciones, para calcular la fuerza de atracción y de repulsión respectivamente (ver ecuaciones 8 y 9)

|  |  |  |
| --- | --- | --- |
|  |  | (8) |
| = |  | (9) |

Dado que la fuerza de repulsión vale cero si no se cumple la condición, debemos emplear un condicionante IF, como se detalla en figura de abajo:



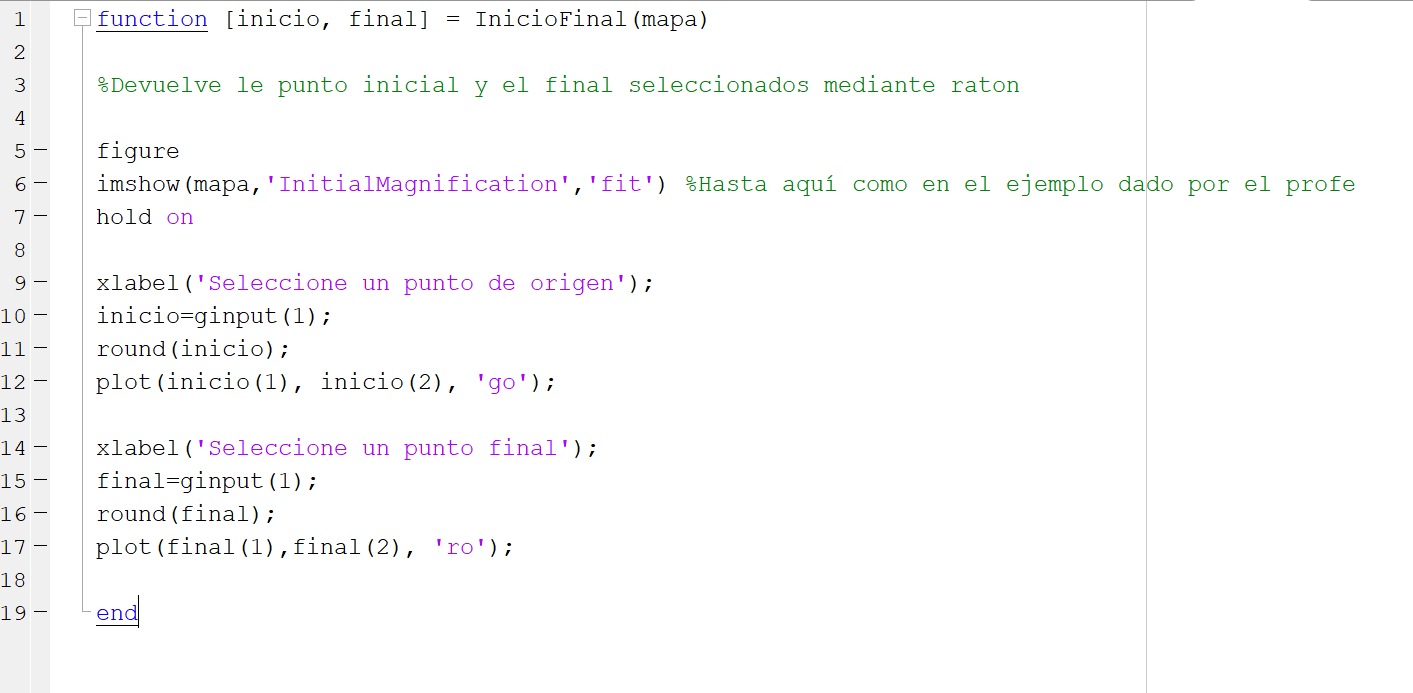
Se implementa una sencilla función llamada Distancia que calcula la distancia que hay entre dos puntos, como se puede observar en foto abajo:



NO RECUERDO COMO IMPLEMENTAMOS ESTO PARA QUE SE LEYERA BIEN

**2. Modificar adecuadamente el programa de Matlab anterior para permitir poder elegir cualquier origen y destino sobre el mapa, empleando para ello la función ginput().**

Dado que en este aparto se pide poder elegir de manera manual la posición de inicio y de final, lo que se ha realizado es una función que muestre en pantalla un mensaje para que el usuario elija un punto de inicio y al hacer click se guardará es punto como entrada. De la misma manera se hace con el punto final. Esto se puede ver desarrollado en la figura de abajo:



1. **Realizar varios experimentos con diferentes orígenes y destinos. Cambiar los parámetros del método, como α y β, o cualquier otro, para tratar de mejorar la situación. Poner el destino y origen en una situación de mínimo local (situación de trampa local), ¿es posible que el método pueda encontrar la solución sin modificarlo y cambiando solo los parámetros anteriormente mencionados?**
2. **Implementar el algoritmo de Dijkstra como una función de Matlab que devuelva el coste y la ruta óptima a partir de un origen y un destino pasados como parámetros, además del mapa topológico o grafo, que se le pasará a la función como una matriz NxN, que almacena el coste de llegar del nodo n1, como fila, al nodo n2, como columna. La función se debe implementar de forma que la llamada Dijkstra devuelva el coste de llegar desde el nodo origen al nodo destino, y un vector con la lista de nodos que componen la ruta (incluidos los nodos inicial y final).**

El algoritmo de Dijkstra es conocido como algoritmo de caminos mínimo porque consiste en encontrar el camino más optimo posible teniendo en cuenta los costes de desplazamiento.

Así pues, se van a realizar una serie de funciones que, trabajando de manera coordinada, consigan hallar tanto el coste total del desplazamiento, así como la ruta que seguirá dicho coste.

El mapa topológico seguido, el cual se detalla en la presentación es la siguiente VER FIG:

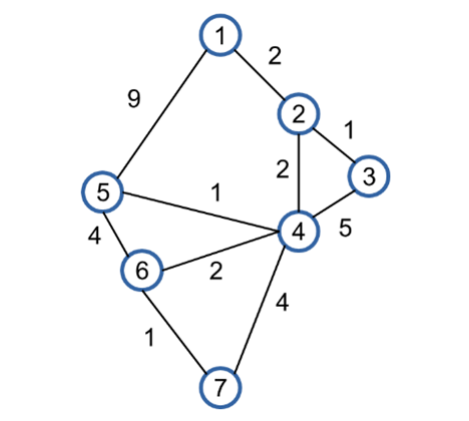
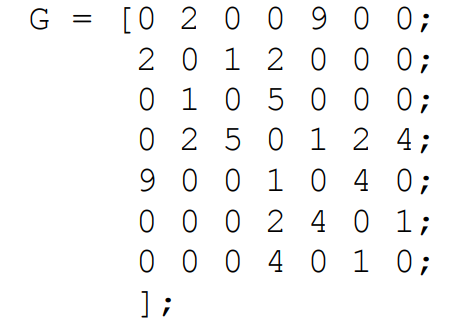


Figura 3: Mapa topológico de costes y nudos.

Si se representa dicho mapa en forma matricial, se consigue la siguiente matriz:



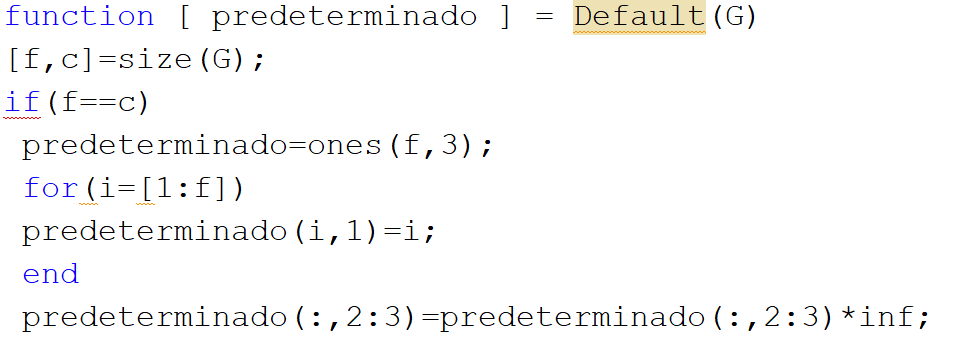
Mediante la manipulación de dicha matriz, es como se conseguirá el camino mínimo deseado.

Se implementa la función global *Dijkstra* VER FIGURA que a su vez está compuesta por otras sub funciones.

AQUÍ VA LA FUNCION GRANDE LA HEMOS LLAMADO DIJKSTRA

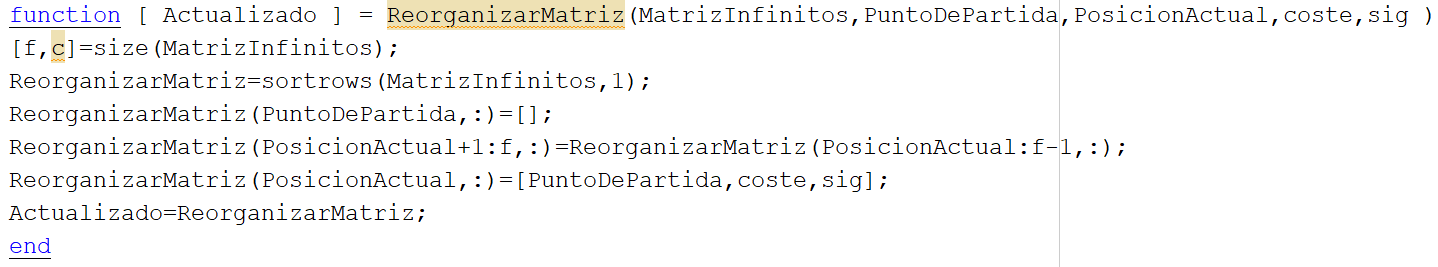
Antes de proceder a la explicación de dicha función y dado que utiliza otras sub funciones, se va a detallar en primer lugar el procedimiento de cada sub función y posteriormente el código general anteriormente mostrado.

La función Default (VER FIGURA), tiene como entrada la matriz topológica de costes. En primer lee y almacena el tamaño de matriz topológica de costes y se comprueba que la matriz es simétrica, es decir, tiene el mismo número de columnas. En caso contrario significaría que la matriz topológica de costes no proporciona los costes asociados a todos los caminos posibles.



En función del tamaño de la matriz simétrica se crea una matriz con tantas filas como nodos hay. Además, dicha matriz tiene tres columnas. La primera de ellas está enumerada del 1 hasta f, dado que f es el tamaño de la matriz que estamos analizando, pues recordemos que es simétrica. Dicha primera columna representará pues todos los nodos presentes en nuestro sistema. La segunda y la tercera, en la primera iteración se inicializarán en infinito, dado que no se sabe en dicha primera iteración si por ejemplo del punto uno se puede ir al punto 4 y su coste asociado. La tercera representa el punto al que se desplaza y la segunda el coste referido a desplazarse desde el punto marcado por la primera columna, hacia el nodo marcado por el de la tercera.

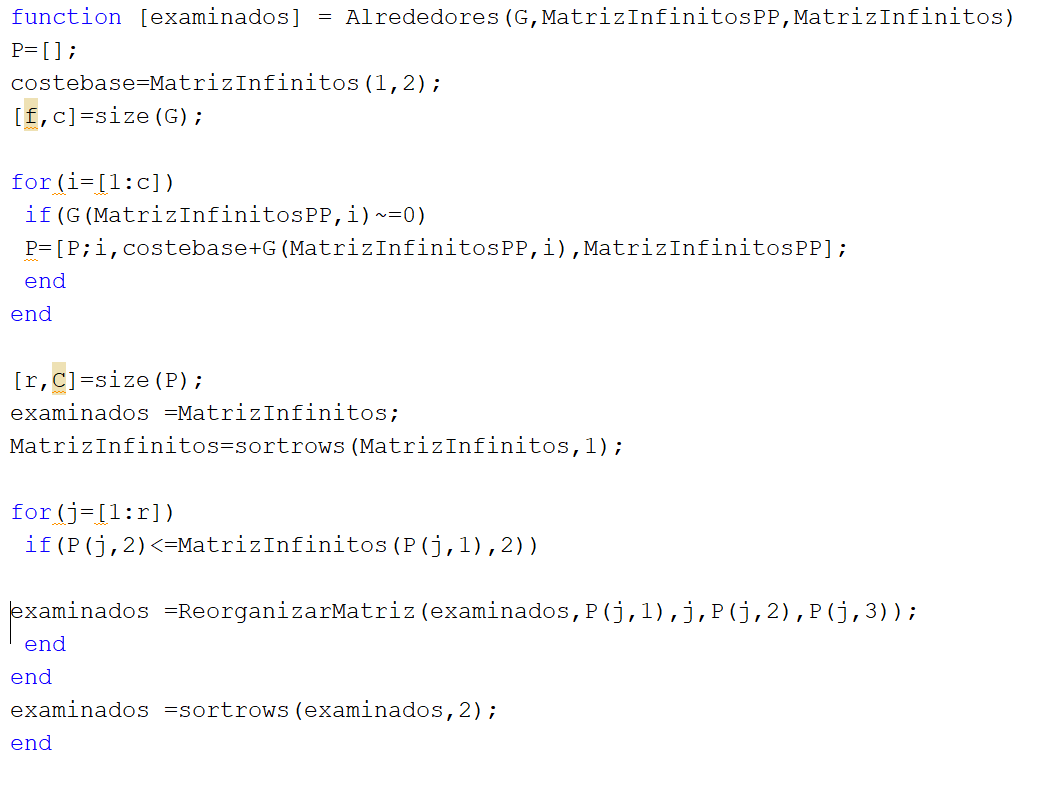
La función *ReorganizarMatriz* (Ver figura de abajo) tiene como entradas la matriz de infinitos previamente creada, el punto de origen dado al inicio, el de partida, la posición actual en la que se encuentra, el coste asociado a desplazarse al nodo siguiente y el nodo siguiente al que se quiere desplazar.



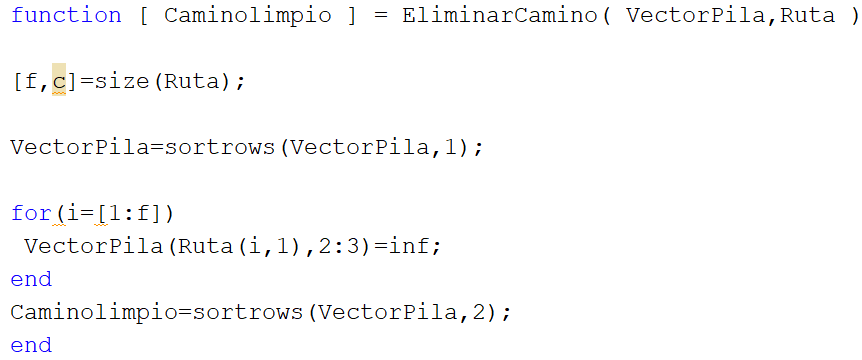
Se calcula nuevamente en esta función el tamaño de la matriz de infinitos, y se ordena toda la matriz en función de la primera columna. Se elimina el punto de partida de dicha matriz, pues no tiene lógica calcular el coste de desplazarte hacia donde ya estás.

Se reordena la matriz pues no nos interesa tener una fila solo de ceros, por lo tanto, si hemos partido, por ejemplo, del nodo cuatro eliminaremos la fila cuatro y la fila cinco pasará a estar en la fila cuatro mediante transformaciones elementales de matrices y así sucesivamente.

La función *Alrededores* VER FIGURA tiene como entradas la matriz topológica de costes, la primera posición de la matriz de infinitos donde se está trabajando y la propia matriz de infinitos. Como se ha detallado anteriormente, la segunda columna hace referencia al coste de desplazamientos y será esta la variable entorno a la cual se reorganizará todo. Mediante un bucle for se irán añadiendo los puntos a los que se puede desplazar partiendo del origen y posteriormente se reorganizará la matriz en función de los costes recién calculados.



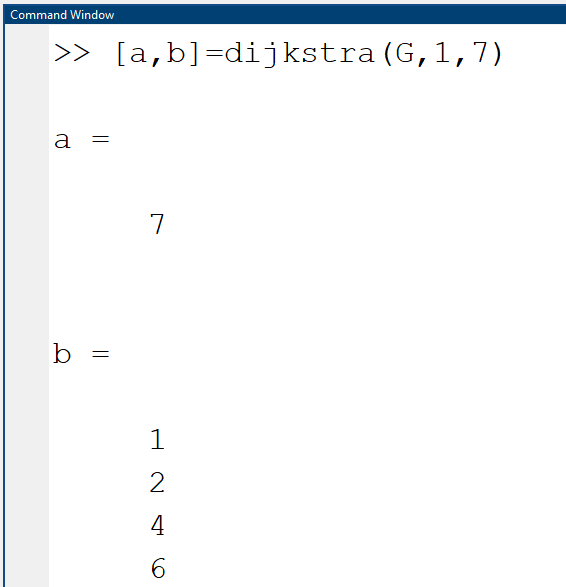
La función *EliminarCamino* VER FIGURA tiene como entradas la matriz de infinitos y la ruta almacenada hasta el momento. El procedimiento a seguir es muy sencillo vamos a eliminar de la ruta seguida los puntos por los que ya ha pasado poniéndolos a valor infinito.



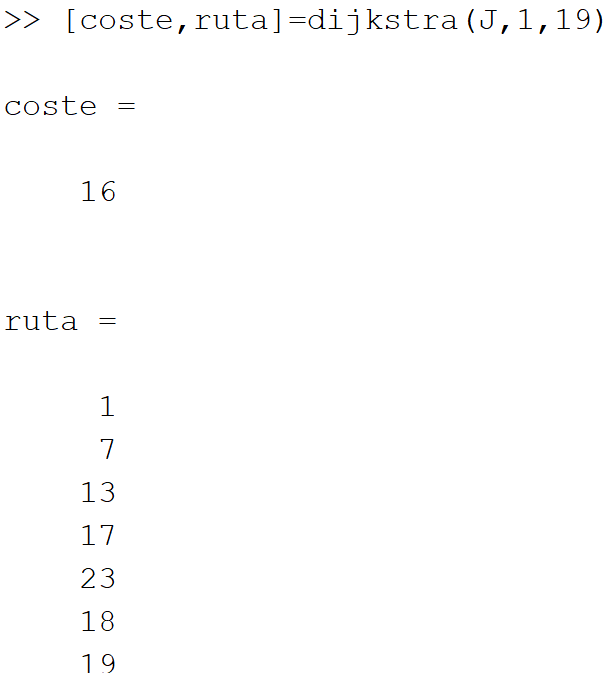
Por último, la función OptimizarRuta VER FOTO tiene como entradas la ruta seguida hasta el momento, el punto de origen y el destino, de formo que devolverá el menor coste entre dos puntos. Se hará chequeo entre el punto actual y el siguiente de manera secuencial.

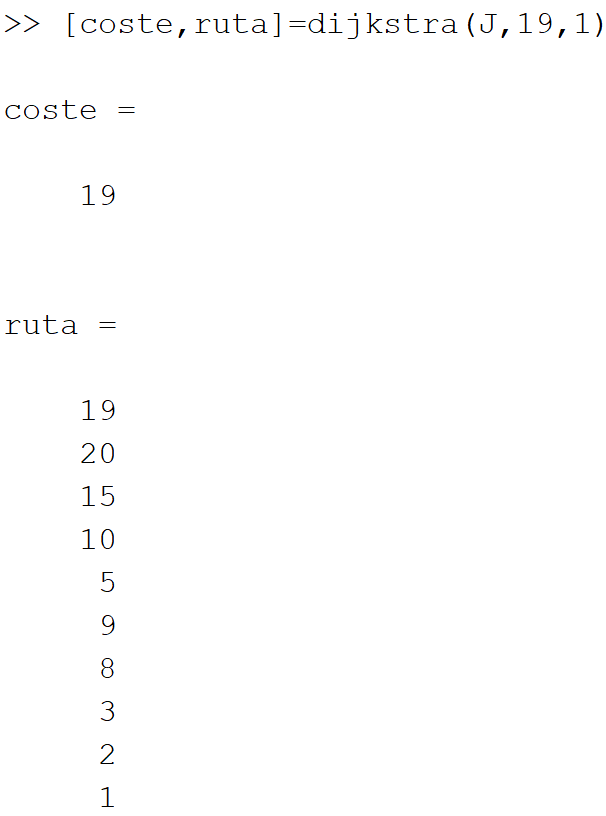
Así pues, tal y como se muestra en la PRIMERA FIGURA QUE HEMOS METIDO DONDE EL CODIGO GENERAL DE DIJSKTRA, se inicializarán las variables ruta y camino. Antes de entrar a realizar cualquier operación se analizará previamente si hay algún nodo incompleto o inalcanzable. Mientras no estemos en un punto aislado y no hayamos llegado al destino, se irá completando los distintos posibles caminos haciendo uso de las funciones nombradas anteriormente. Una vez se tengan todos los posibles caminos, se optimizará la ruta y se obtendrá tanto el coste total como la ruta empleada.

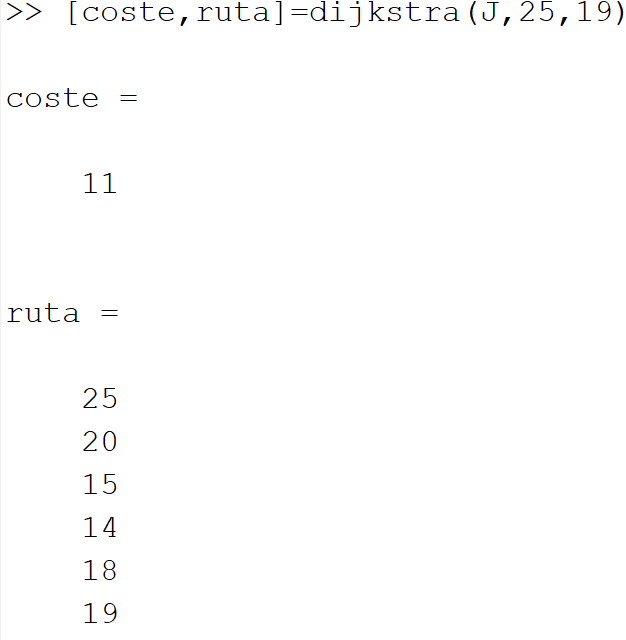
Se confirma la robustez del sistema mediante una serie de comprobaciones donde se obtienen los resultados esperados VER FIGURA DE ABAJO.

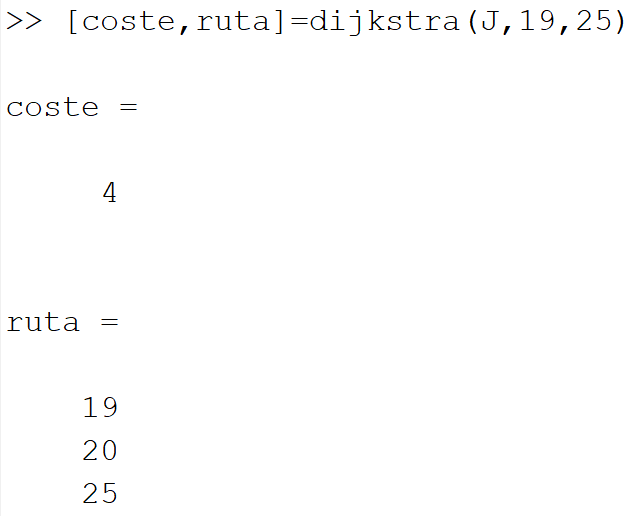


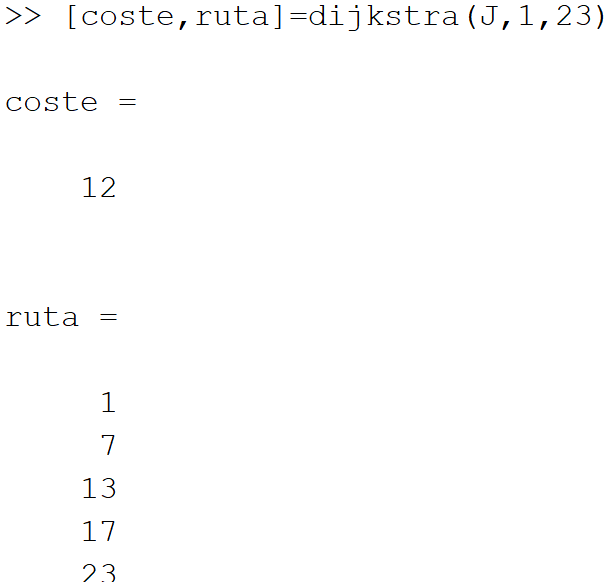
Se calculan los resultados para los siguientes nodos de partida y llegada:

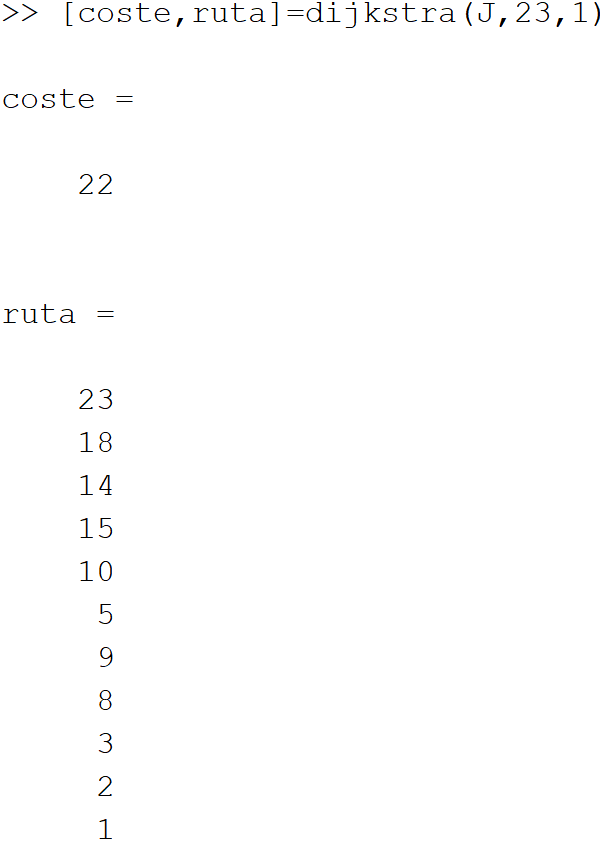


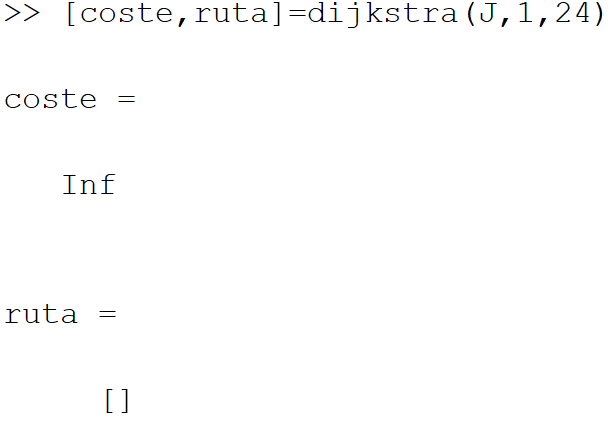


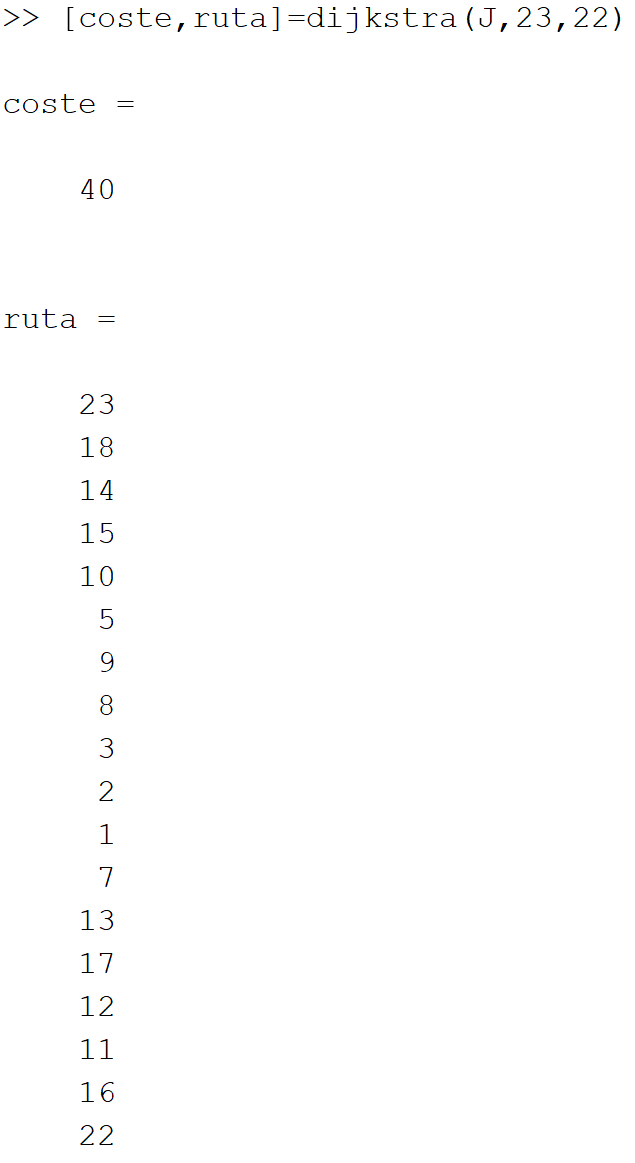












Tal y como se puede observar en el penúltimo de los resultados, dado que el nodo 24 no es alcanzable desde ningún otro nodo, el coste es infinito y no hay ruta mediante la cual se puede alcanzar dicho nodo. En cualquier otro caso, el sistema actúa correctamente según lo esperado.