

Títol: Resolució de models d'equilibri de trànsit usant Python

Volum: 1/1

Alumne: Olmeda San Miguel, Jorge

Director/Ponent: Codina Sancho, Esteve

Departament: EIO

DADES DEL PROJECTE

Títol del Projecte: Resolució de models d'equilibri de trànsit usant Python

Nom de l'estudiant: Olmeda San Miguel, Jorge

Titulació: Enginyeria Informàtica

Crèdits: 37.5

Director/Ponent: Esteve Codina Sancho

Departament: EIO

MEMBRES DEL TRIBUNAL (nom i signatura)

President: Montero Mercadé, Lidia

Vocal: Arias Vicente, Marta

Secretari: Codina Sancho, Esteve

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índice de contenido

Capítulo 1 :Análisis de redes de transporte urbano.....	7
1.1. Análisis de equilibrio de sistemas de transporte.....	9
1.1.1.Transporte urbano enfocado a sistemas.....	9
1.1.2.Equilibrio en los mercados.....	11
1.2 Representación de redes de transporte urbano.....	12
1.3.Equilibrio aplicado a redes de transporte urbano.....	15
1.4.Definición de equilibrio.....	15
1.4.1.Un ejemplo sencillo de equilibrio de usuarios.....	18
Capítulo 2 : Formulación del problema de asignación de flujos como un problema matemático.	22
2.1.Formulación matemática del problema de asignación de tráfico en equilibrio de usuarios.....	27
2.2.Técnicas de resolución del equilibrio de usuarios.....	28
2.3.Técnicas heurísticas para la búsqueda del equilibrio de usuarios.....	29
2.3.1.Capacity restraint.....	30
2.3.2.Incremental Assignment.....	33
2.4.Algoritmo de las combinaciones convexas para la búsqueda del equilibrio de usuarios.....	35
Capítulo 3: Minimización unidimensional.....	42
3.1.Métodos de reducción de intervalo.....	42
3.2.Métodos de ajuste de curvas.....	49
Capítulo 4 :Equilibrio de usuarios con demanda variable.....	53
4.1.Formulación matemática del problema de asignación de tráfico con demanda variable.....	54
4.2.Resolución del problema de asignación de tráfico con demanda variable.....	56
4.3.Resolución a través de la modificación de la representación de las redes de transporte.....	59
Capítulo 5: Especificación de la aplicación.....	62
5.1. Clase parametros.....	62
5.2.Clase dicParFunciones	62
5.3.Clase lgiros.....	63
5.4.Clase lassignacion.....	63
5.5.Clase Frank Wolfe.....	64
5.6.Fichero GoldenSection.....	65
5.7.Clase resultadosIteracion.....	66
5.8.Clase caminos.....	66
5.9.Clase criterioBLB.....	67
5.10.Clase arista.....	68
5.11.Clase aristaElastica.....	70
5.12.Clase Funcion.....	72
5.13.Clase funcionElastica.....	73
5.14.Fichero casoelastico.....	75
Capítulo 6: Características de Python.....	76
6.1.¿Qué es Python?.....	76
6.2.¿Por qué Python?.....	77
6.3.Instalación de Python.....	78
Capítulo 7: Formato de la información.....	80
7.1.Fichero de configuración del sistema.....	80

7.2. Parámetros de las funciones de rendimiento.....	81
7.3. Parámetros de las funciones de demanda variable.....	82
Capítulo 8 : Juegos de prueba.....	84
8.1. Juego de pruebas pequeño.....	84
8.2. Juego de pruebas mediano.....	92
8.3. Juego de pruebas con demanda elástica.....	95
Capítulo 9: Coste del proyecto.....	99
Bibliografía.....	101
Anexo A : MANUAL DE INSTALACIÓN.....	102

Capítulo 1 :Análisis de redes de transporte urbano

El coste de los viajes que tienen lugar en un momento dado en cualquier calle, intersección, o línea de tránsito en una zona urbana es el resultado de muchas decisiones personales.

Estas decisiones dependen, en parte, de la congestión del sistema de transporte y los puntos donde se concentra la congestión.

La congestión en cualquier punto del sistema de transporte, sin embargo, depende de la cantidad de viajes que pasan a través de ese punto.

Nuestro propósito es estudiar las interacciones entre la congestión y las decisiones de los usuarios para poder ser modelados y resueltos conjuntamente para así obtener el patrón de flujos de la red de transporte urbano.

Uno de los principales problemas a los que se enfrentan los ingenieros de transporte y planificadores urbanos es la de predecir el impacto sobre una red específica.

La parte analítica de este problema puede dividirse en dos fases.

En la primera fase, la red se especifica matemáticamente como un conjunto de inputs que se utilizan para predecir el patrón de flujo sobre esta red específica.

En la segunda fase, el patrón de flujo se utiliza para calcular una serie de medidas a aplicar sobre esta red.

Los inputs suelen incluir una descripción de lo siguiente:

- 1 - La infraestructura de transporte y servicios, incluyendo calles, intersecciones, y las líneas de tránsito.
- 2 - Las políticas de control y funcionamiento del sistema de transporte.
- 3 - La demanda de viajes, incluidos los patrones de actividad y uso de la infraestructura.

La primera fase del análisis utiliza estos inputs para calcular el flujo a través de cada uno de los componentes de la red urbana.

El flujo se mide en términos del número de unidades de viaje (vehículos, pasajeros, peatones), que atraviesan un determinado punto de la red en una unidad de tiempo. Esta parte del análisis se basa en las relaciones entre los flujos y congestión, así como las relaciones entre las decisiones de viaje y la congestión.

La segunda fase del análisis consiste en la redacción de una serie de medidas a partir del patrón de

flujos calculado en la primera fase.

Estas medidas deben incluir lo siguiente:

1. Medidas de nivel de servicio, como las que inciden en el coste y tiempo de los viajes. Estas medidas afectan a los usuarios del sistema de transporte directamente.
2. Las características de la operación, tales como los ingresos y beneficios. Principal preocupación de los operadores del sistema.
3. Flujo de los subproductos, como la contaminación y los cambios en el valor de la tierra. Estos afectan al medio ambiente.
4. Las medidas de bienestar, como la accesibilidad y la equidad. Estas pueden afectar indirectamente a los distintos grupos de población.

Nosotros nos centraremos en la primera fase del análisis, la de calcular el patrón de flujos a partir de los inputs.

Muchas de las medidas de la segunda fase pueden calcularse a partir de estos flujos de una manera directa.

Otros (como la contaminación o las medidas de equidad) pueden requerir sofisticadas técnicas y modelos complejos que no están dentro de nuestros objetivos. Estas medidas, sin embargo, utilizan el patrón de flujos como una de las principales aportaciones.

Además, un análisis completo de un determinado escenario puede incluir consideraciones de importancia que no se basan en el flujo, como los costes de construcción y los factores políticos.

En otras palabras, nosotros queremos conocer el patrón de flujos de la red de transportes a partir de las características del sistema. No tratamos de determinar la mejor configuración del sistema.

En muchos casos, sin embargo, el análisis está motivado por la necesidad de tomar decisiones sobre un determinado sistema. Estas decisiones (inversión, regulación, política) generan una serie de escenarios alternativos en el sistema.

Estos escenarios alternativos son formulados y resueltos con el fin de predecir el patrón de flujos. Que, a su vez, es usado para escoger entre las posibles medidas a aplicar.

El conjunto de medidas escogidas para cada alternativa son comparadas para estudiar su impacto en el sistema. Una técnica habitual es comparar el conjunto de medidas de un caso alternativo con el

conjunto de medidas de un caso base escogido por nosotros. Este caso base suele ser el estado actual del sistema o una proyección del funcionamiento del estado actual bajo la alternativa “no hacer nada”.

Estas comparaciones son usadas para tomar las mejores decisiones sobre el sistema.

Nuestro propósito es presentar las distintas hipótesis, modelos y algoritmos utilizados para calcular el patrón de flujo asociado a cada conjunto de inputs.

La idea principal es la interacción entre la congestión y las decisiones de los usuarios, que se traducen en un patrón de flujos.

Esta interacción puede ser modelada como un proceso de búsqueda del equilibrio entre las decisiones de los usuarios y la congestión.

1.1. Análisis de equilibrio de sistemas de transporte

Este capítulo introduce tres temas necesarios para el estudio de los sistemas de transporte.

- 1- La necesidad en el estudio del transporte urbano del enfoque basado en sistemas.
- 2- La idea general de equilibrio y de los diversos tipos de equilibrios
- 3- Las aplicaciones del equilibrio en el análisis de sistemas de transporte urbano

1.1.1. Transporte urbano enfocado a sistemas

El enfoque tradicional en muchas ingenierías es separar los elementos del sistema y estudiarlos de forma aislada. Este enfoque también es usado para estudiar el efecto de pequeños cambios en un sistema de transportes urbano. Por ejemplo, los tiempos en un semáforo, para ello sólo se toma en consideración el tráfico del cruce en estudio. Los efectos de los cambios en los cruces cercanos se suelen considerar insignificantes. Otros ejemplos de estudios aislados son la regulación del aparcamiento o el diseño de los cruces.

Si el impacto de una determinada política o diseño es muy pequeño, entonces el estudio de los elementos de forma aislada es una elección adecuada. Pero si los cambios en el objeto de estudio son más importantes, los cambios no sólo afectarán al elemento en estudio sino que se propagarán por el resto del sistema.

Un ejemplo de esto sería el efecto onda aplicado a una vía de comunicación saturada.

La autoridad del transporte metropolitano decide reducir la congestión de un tramo de la red saturado. Después de un estudio económico y medioambiental y teniendo en cuenta el flujo de ese tramo deciden que un carril más pueden solucionar el problema. Estudiando ese tramo de forma aislada se puede suponer que al haber un carril más, el tráfico de esa vía se repartirá, podrán ir más rápido y los retrasos disminuirán.

Los beneficios esperados con este cambio, estudiando el tramo de forma aislada, serán erróneos. Esto es debido a que no se ha tenido en cuenta las decisiones de los usuarios del sistema.

Por ejemplo, es posible que usuarios del sistema que no usaran el tramo antes decidan comenzar a usarlo debido a la ampliación de carriles. El tráfico en el tramo crecerá y la vía puede volver a saturarse. Así mismo, las vías conectadas con el tramo en estudio pueden resultar saturadas por el incremento de flujo a la entrada y salida del tramo ampliado. Del mismo modo, las vías con un recorrido parecido al tramo ampliado verán su tráfico reducido y sus usuarios mejorarán sus condiciones de viaje. Estas mejoras atraerán a usuarios del sistema que no utilizaban este tramo y modificarán su trayecto para usarla.

Cada cambio de trayecto provocará variaciones en la congestión de los tramos del sistema. Estas variaciones pueden provocar que algunos usuarios quieran cambiar su trayecto. A la larga las variaciones irán decreciendo, después de un corto espacio de tiempo (pueden ser horas o algunos días) el sistema se estabilizará en un nuevo estado de “equilibrio”, en este estado podemos decir que no hay variaciones significativas.

El ejemplo previo nos ha enseñado los cambios en el flujo de la red como resultado de un cambio en la infraestructura de la red de transporte. Cambios parecidos pueden ocurrir si modificamos la política de control de tráfico o si introducimos nuevos servicios de transporte. Por otra parte, los nuevos patrones de flujo pueden ser el resultado de cambios que no están directamente vinculados a los sistema de transporte, sino más bien cambios en las actividades generales en el área urbana. Estas nuevas actividades generan una necesidad y una mayor demanda en la red de transportes. Por ejemplo, la apertura de un centro comercial. El nuevo centro comercial atraerá los viajes orientados a ir de compras, una parte de ellos serán nuevos (gente que empieza a ir más de compras) y otra parte serán atraídos de los otros centros comerciales. Las calles cercanas al nuevo centro comercial se verán saturadas, esto provocará que los usuarios que usan estas vías para llegar a otros destinos decidan cambiar de ruta, cambiar el medio de desplazamiento o asumir que ahora tardarán

más. Estos nuevos cambios provocarán variaciones en el flujo de muchas vías del sistema, pudiendo causar más cambios en la selección de rutas de los usuarios.

En las proximidades de los otros centros comerciales también se producirán cambios, es probable que se vea una reducción del flujo de usuarios. Otra vez, después de un pequeño plazo y una serie de modificaciones de flujo, el sistema se estabilizará en un nuevo punto de equilibrio. En este punto, la frecuencia de viajes, el número de viajeros a cada destino, los modos de viaje escogidos y las rutas escogidas serán estables en todo el sistema.

El significado de equilibrio en el contexto de la ingeniería de transportes es semejante a la noción de estado de equilibrio en la física. El estado de equilibrio es el estado en el que no hay fuerzas resultantes que empujen al sistema a otro estado.

Además, cuando el sistema está en desequilibrio, hay fuerzas que empujan el sistema hacia un estado de equilibrio.

En nuestro caso, los flujos empujan al sistema hacia el equilibrio a través del cambio de ruta de los usuarios. En equilibrio, los usuarios no tienen motivación para cambiar de ruta y los flujos no cambiarán. Este es el estado en que nosotros nos centraremos.

Algo importante a resaltar, el patrón de flujos que determina el estado de equilibrio del sistema sólo puede ser encontrado estudiando todos los elementos de la red de transporte de forma simultánea. Tal y como hemos podido observar en los ejemplos usados antes.

1.1.2. Equilibrio en los mercados

La visión clásica de la economía de mercado (es decir, con competencia perfecta) para un determinado producto se compone de dos grupos que interaccionan : los productores y los consumidores.

El comportamiento de los productores es representado por una función de oferta y el comportamiento de los consumidores es representado por una función de demanda.

La función de oferta expresa la cantidad de producto X que los productores pondrán en el mercado para un determinado precio del producto X. Si los precios suben, los productores verán que es rentable aumentar su producción e incrementarán la cantidad de producto X que ponen en el mercado.

La función de demanda expresa la cantidad de producto X que los consumidores comprarán para un determinado precio del producto X. Si los precios suben, los consumidores comprarán menos cantidad del producto X.

El sistema se estabilizará en el punto de cruce de las dos funciones. Este punto nos dirá que si producimos Q^* unidades y las vendemos al precio P^* , toda la producción será comprada.

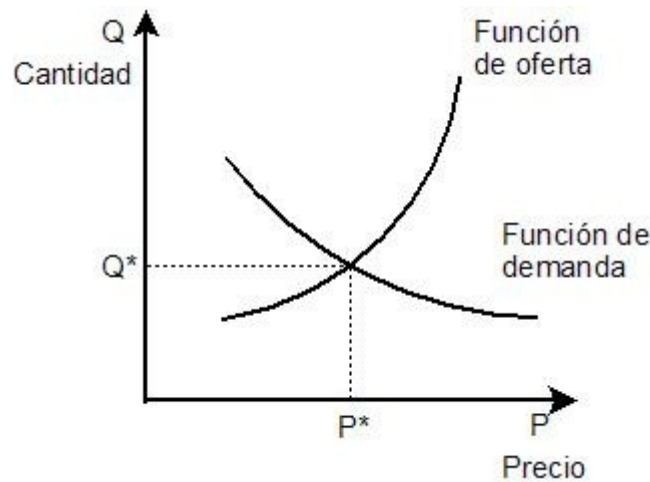


Figura 1.1: Descripción gráfica de las curvas de oferta y demanda y del punto de equilibrio en el sistema económico.

1.2 Representación de redes de transporte urbano

Para representar una red de transporte urbano lo mejor es usar un grafo dirigido ponderado (aquí un apéndice). Explicado de forma simple podemos decir que un grafo es un conjunto de puntos, llamados aristas, unidos por flechas llamadas aristas.

En el ámbito de las redes de transporte un nodo es la intersección de una o más vías de transporte y una arista el tramo de vía entre dos intersecciones. El grafo necesita que sea dirigido para representar las vías de único sentido de forma diferente a las de doble sentido. Es decir, si una vía es de doble sentido habrá dos aristas, con la misma dirección pero sentido contrario, pero si es de único sentido con una arista tendremos suficiente.

Centroides

Los usuarios de la red de transporte urbano desean viajar de su casa a su puesto de trabajo o de su casa a su bar favorito. Para representar estos lugares lo que hacemos es asociarlos a un nodo, de esta manera un nodo también representa varias localizaciones cercanas. Estos nodos reciben un nombre especial centroide. La principal diferencia respecto al resto de nodos es que estos nodos se usan para inyectar o retirar flujo en el grafo.

Por ejemplo, 10 personas que trabajan en el mismo edificio de oficinas les gusta ir al gimnasio después del trabajo, casualmente todos van al mismo gimnasio. En la representación lo que haremos será inyectar 10 unidades en el centroide asociado al edificio de oficinas y retirar 10 unidades en el centroide asociado al gimnasio. A esto lo llamaremos pareja O-D (centroide origen – centroide destino).

Funciones de rendimiento

Normalmente cuando se trabaja con grafos ponderados se asigna un coste fijo a cada arista. Por ejemplo, la arista A1 tiene asignado un coste de 30 segundos, quiere decir que si un usuario de la red quiere recorrer la arista A1 tardará 30 segundos siempre.

Como estamos trabajando con redes de transporte mediremos los costes en segundos.

Pero tenemos un problema, a la hora de modelizar redes de transporte la asignación de un coste fijo a las aristas no crea buenos modelos. El coste de recorrer un tramo de vía depende del número de usuarios de dicha vía. A mayor número de usuarios mayor tiempo pero no de una forma proporcional.

Lo correcto es el uso de funciones de rendimiento, como la mostrada en la figura X.X. En ella podemos observar que los tiempos de desplazamiento comienzan con un coste mínimo que va creciendo según aumenta el número de usuarios de la vía. Hasta que llega un punto en que el crecimiento es elevado, esto refleja que la vía está en estado de saturación y los usuarios están casi parados.



Figura 1.2 : Ejemplo de función de rendimiento usada en redes de transporte

La función de rendimiento más frecuente es la siguiente:

$$f(v) = t_0 * \left(1 + \alpha * \left(\frac{v}{c} \right)^\beta \right)$$

Donde:

- v : Flujo asignado a la arista.
- $f(v)$: Tiempo que tardará cada usuario en recorrer la arista.
- t_0 : Tiempo que tardaría cada usuario en recorrer la arista en condiciones ideales. Si es necesario, se puede calcular a partir de unos datos básicos del tramo de vía que representa la arista, dividiendo la longitud del tramo de vía entre la velocidad máxima permitida en ella.
- c : Capacidad máxima del tramo de vía.
- α y β : parámetros usados para representar el comportamiento de la vía en condiciones de saturación.

Otras funciones de rendimiento usadas son :

$$f(v) = a + b * \left(\frac{v}{c} \right) + c * \left(\frac{v}{c} \right)^2$$

y

$$f(v) = k_0 + k_1 * e^{K_2 \left(\frac{v}{c} \right)}$$

1.3. Equilibrio aplicado a redes de transporte urbano

La noción de equilibrio en el análisis de redes de transporte urbano surge de la relación entre los tiempos de desplazamiento por las aristas y los flujos de ellas. Asume que conocemos el número de usuarios que desear ir desde un determinado punto de origen hasta otro punto destino. Además, asume que hay varias rutas disponibles entre el origen y el destino. La pregunta que debemos hacernos es, **¿cómo va a ser la distribución de estos usuarios entre las posibles rutas?**

Si todos ellos escogen la misma ruta (inicialmente pensaremos que todos han escogido la ruta más corta), lo que se conseguirá es que esa ruta se congestione. Como resultado de esa congestión el tiempo de desplazamiento por esa ruta se incrementará hasta el punto que deje de ser la ruta más rápida. Así que algunos de los usuarios pasarán a usar otra ruta. La nueva ruta se verá congestionada por estos usuarios, el tiempo de desplazamiento se incrementará, dejará de ser la ruta más rápida y algunos usuarios pasarán a usar otra ruta, y así sucesivamente.

La determinación de los flujos en cada una de estas rutas implica encontrar la solución a un problema de equilibrio demanda / rendimiento. El flujo en cada arista es resultado de la suma de los flujos de muchas rutas entre muchos orígenes y destinos. Cada arista posee una función de rendimiento independiente al resto de aristas, esta función nos dice el coste del desplazamiento a partir del flujo de la arista.

En cambio, la demanda para viajar depende del comportamiento de los usuarios y no es independiente para cada arista. En lugar de ello, especifica cómo los usuarios escogen entre las posibles rutas que conectan cada pareja origen-destino (O-D).

Esta dicotomía entre la definición de las funciones de demanda y las funciones de rendimiento nos está diciendo que el estudio del problema del equilibrio demanda / rendimiento es del ámbito del sistema. En otras palabras, este es el motivo por el que aristas, rutas o parejas origen-destino no pueden ser estudiados de forma aislada.

En esta sección se discute la elección de rutas y las definiciones de equilibrio utilizadas en este texto.

1.4. Definición de equilibrio

El propósito de este capítulo es definir el término equilibrio a partir de un problema de equilibrios sobre redes de transportes.

El problema que estudiaremos aquí es la selección de rutas por parte de los usuarios del sistema para ir desde su punto de origen hasta su destino.

Este problema se puede representar así :

Dado :

1. Un grafo que represente la red de transporte urbano.
2. Las funciones de rendimiento asociadas a las aristas.
3. Una matriz origen-destino

Encuentra el flujo (y el tiempo de desplazamiento) en cada una de las aristas.

Este problema es conocido como “asignación de tráfico” porque la cuestión es cómo asignar la matriz origen-destino a la red. Los flujos de las aristas obtenidos serán usados para calcular una serie de medidas, éstas podrán ser usadas para evaluar la red. Diseños particulares de la infraestructura de transporte o las políticas de control de transportes suelen entrar en el análisis a través de la especificación de la propia red y las funciones de rendimiento.

Para resolver el problema de asignación de tráfico es necesario que se especifique la política de elección de ruta de los usuarios. Esta política puede ser vista como la función o el procedimiento que explica la demanda de desplazamiento por las rutas. La interacción entre las rutas escogidas por cada par origen-destino, y las funciones de rendimiento de todas las aristas de la red, determinan los flujos de equilibrio de la red y sus correspondientes costes de desplazamiento.

Es razonable asumir que cada usuario intentará minimizar el tiempo que tarda en desplazarse desde su origen hasta su destino. Esto no quiere decir que todos los usuarios que parten del mismo punto de origen y con el mismo destino vayan a usar la misma ruta. El coste de desplazamiento en cada arista varía con el flujo, es decir, los costes de las diferentes rutas de la red varían según cambien los flujos de las aristas. Encontraremos un estado estable cuando “**ningún usuario pueda mejorar su tiempo de desplazamiento cambiando unilateralmente su ruta**”.

Esta es la definición del estado en equilibrio de usuarios (UE).

Dado que podemos esperar que los usuarios se comportan de manera independiente, el estado UE asegura que en este punto no hay ninguna fuerza que intente desplazar los flujos de esta situación de equilibrio. En definitiva, este punto será estable, y de hecho, un verdadero equilibrio.

El estado UE no es la única definición posible de equilibrio. La hipótesis que cada usuario escoge la

ruta con menores costes de desplazamiento es razonable en algunos casos, pero esta hipótesis incluye una serie de presunciones que no siempre se cumplen.

Por ejemplo, la definición de UE implica que todos los usuarios poseen toda la información (es decir, ellos conocen el coste de todas las posible rutas) y que ellos siempre escogen la ruta correcta. En conclusión, asumimos que todos los usuarios se comportan igual.

Esta suposición puede ser rebajada haciendo la distinción entre el tiempo de desplazamiento que perciben los usuarios y el tiempo de desplazamiento actual. El tiempo de desplazamiento percibido puede ser visto como una variable aleatoria distribuida en la población de los usuarios del sistema. Es decir, cada usuario puede percibir un tiempo de desplazamiento diferente para la misma arista. El equilibrio será hallado cuando ningún usuario crea que pueda mejorar sus tiempos cambiando unilateralmente su ruta. A esto se le conoce como el equilibrio de usuarios estocástico (SUE).

El equilibrio de usuarios estocástico es una generalización de la definición de equilibrio de usuarios. Si aceptamos que el tiempo de desplazamiento percibido es totalmente exacto, todos los usuarios percibirán el mismo tiempo de desplazamiento y el equilibrio de usuarios estocástico será idéntico al equilibrio de usuarios (determinista). En otras palabras, el patrón de flujos resultante será el mismo en los dos casos.

Las definiciones dadas hasta el momento para el término equilibrio no son fáciles de usar de una forma operativa para resolver el patrón de flujos de equilibrio. Para sernos útil, la definición de equilibrio debe ser explicada y formulada en términos matemáticos.

Hay que mencionar un importante punto con respecto a los dos tipos de equilibrios. La demanda de transporte urbano deriva del patrón de actividades del área en estudio. Los horarios y localización de estas actividades nos dicen que la demanda de transporte difiere a lo largo del día. El estudio de equilibrios explicado en esta memoria sólo es aplicable si los flujos se mantienen estables durante todo el período de estudio. Por ello, los planificadores de transporte estudian los sistemas de transporte urbano para determinados períodos de tiempo como el amanecer, atardecer, mediodía, dependiendo del objetivo del estudio.

Los flujos origen-destino para cada uno de estos períodos pueden ser considerados constantes para nuestro propósito. Cuanto mayor sea el período de estudio, menos exacta será nuestra suposición. Los períodos de estudio tampoco pueden ser muy pequeños, deben ser sensiblemente superiores a la duración normal de los viajes.

1.4.1. Un ejemplo sencillo de equilibrio de usuarios

Observa la red con dos aristas de la imagen 1.3 . Esta red representa una pareja origen-destino con dos posibles rutas. T_1 y t_2 representan el tiempo de desplazamiento en las aristas 1 y 2 respectivamente, x_1 y x_2 representan el flujo de tráfico en estas aristas. El flujo total entre origen-destino es representado por q , donde $q = x_1 + x_2$

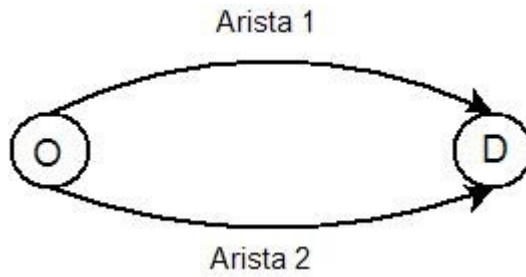


Imagen 1.3

Las funciones de rendimiento en estas aristas $t_1(x_1)$ y $t_2(x_2)$ se muestran en la imagen 1.4 . Para cada arista, la función de rendimiento asociada nos dice el tiempo de desplazamiento usando como parámetro el flujo de la arista.

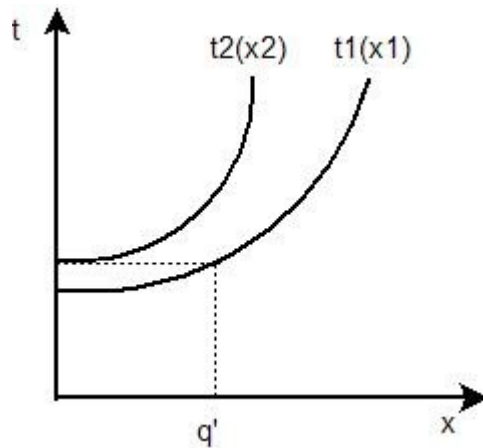


Imagen 1.4

Asumimos que el número de viajes entre O y D es muy pequeño. En otras palabras, q es muy pequeño. Si todos los usuarios quieren minimizar su tiempo de desplazamiento entonces escogerán viajar por la arista 1. Tal como podemos ver en la imagen 1.4 , esta arista posee un menor tiempo de desplazamiento inicial que la arista 2. Si q es pequeño, el incremento de tiempo debido al tráfico en

la arista 1 no es suficiente como para alcanzar el tiempo de desplazamiento inicial de la arista 2. Así que todos los usuarios (q) escogerán viajar por la arista 1 y ninguno la arista 2. Esta es una situación de equilibrio porque ningún usuario de la arista 1 posee alicientes para cambiar de ruta. Esta situación de equilibrio se mantendrá mientras $q < q'$, donde q' es el flujo que provoca que el tiempo de desplazamiento de la arista 1 sea igual al tiempo de desplazamiento inicial de la arista 2. Cuando q y q' se igualen, un nuevo usuario del sistema puede que escoja la arista 2, el tiempo de desplazamiento en la arista 2 se incrementará y el siguiente usuario puede que escoja la arista 1. Si el siguiente usuario escoge la arista 1, entonces el que lo siga escogerá la arista 2.

Si tratamos el flujo de tráfico como un flujo continuo, es evidente que más allá del punto $q = q'$, el equilibrio sólo podrá ser mantenido si el tiempo de desplazamiento en ambas aristas es igual. A partir de este punto los dos enlaces son usados, y si el tiempo de viaje no es igual, algunos usuarios puede que cambien de ruta y consigan bajar su tiempo. El proceso de cambio de ruta no se producirá si el tiempo de desplazamiento en todas las rutas es igual, esto no dará incentivos a los usuarios para cambiar de ruta.

Las dos situaciones de equilibrio que pueden suceder en nuestro ejemplo (la primera para $q < q'$ y la segunda para $q > q'$) dan lugar a una nueva definición operativa de equilibrio de usuarios sobre redes de transporte.

Definición de UE : Para cada pareja O-D, en un estado de equilibrio de usuarios, los tiempos de desplazamiento de todas las rutas usadas son iguales, y además menor o igual que el tiempo de desplazamiento de las rutas sin usar si las usara sólo un usuario.

Esta definición nos dice que en situación de equilibrio, las rutas que conectan todas las parejas O-D pueden ser repartidas en 2 grupos. El primer grupo incluye las rutas que tienen flujo de usuarios. Los tiempos para todas ellas serán el mismo. El segundo grupo incluye todas las rutas que no poseen flujo. El tiempo de estas rutas será, como mínimo, igual al de las rutas del primer grupo.

Usando esta definición, el ejemplo de dos enlaces mostrado en la imagen 1.2 puede resolverse para cualquier valor de q . Si la cantidad de usuarios (flujo) a asignar es menor a q' es obvio que todos deberán ser asignados a la arista 1. El único problema es asegurarse que la política de asignación de flujo a partir del punto en el que $q = q'$, garantice que el flujo asignado sea tal que mantenga el tiempo de desplazamiento igual en los dos enlaces. Si el tiempo de desplazamiento en estado de

equilibrio (t) es conocido (para el caso en que $q > q'$), la definición de equilibrio descrita anteriormente nos asegura que $t = t_1 = t_2$. En consecuencia, el flujo a asignar a cada enlace puede ser determinado por la inversa de la función de rendimiento de ese enlace. Gráficamente, podemos obtener este resultado usando la representación gráfica de las funciones de rendimiento, tal y como podemos observar en la imagen 1.5. Este método puede ser usado para cualquier valor de t , incluso los casos en los que $q < q'$. El problema entonces es determinar t .

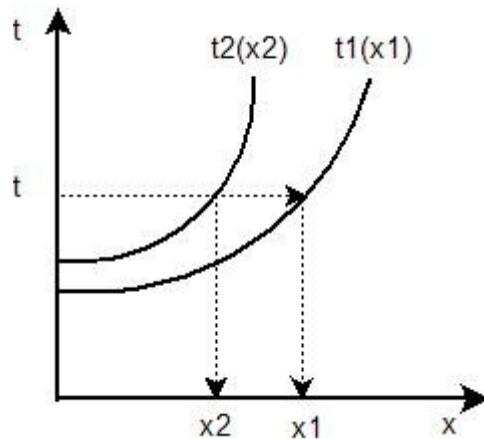


Imagen 1.5

El tiempo de desplazamiento en equilibrio puede ser calculado a partir de la creación de una nueva “curva de rendimiento”, la cual representa el tiempo de desplazamiento entre O-D como una función de flujo para O-D.

Esta curva se puede construir sumando las funciones de rendimiento de las aristas, como podemos observar en la imagen 1.5. La curva $t(q)$ de esta imagen es una función de rendimiento que nos da el tiempo de desplazamiento en equilibrio para O-D a partir del flujo de O-D. La construcción de la función de rendimiento con este método nos asegura que para cada valor posible del tiempo de desplazamiento, el flujo O-D asociado es la suma de los flujos de todos los enlaces. Una vez esté construida esta función, el tiempo de desplazamiento en equilibrio podrá ser calculado insertando en la función el flujo O-D. Si lo que tenemos son los tiempos de equilibrio, los flujos de los diferentes enlaces pueden ser calculados gráficamente, tal como se muestra en la imagen 1.6.

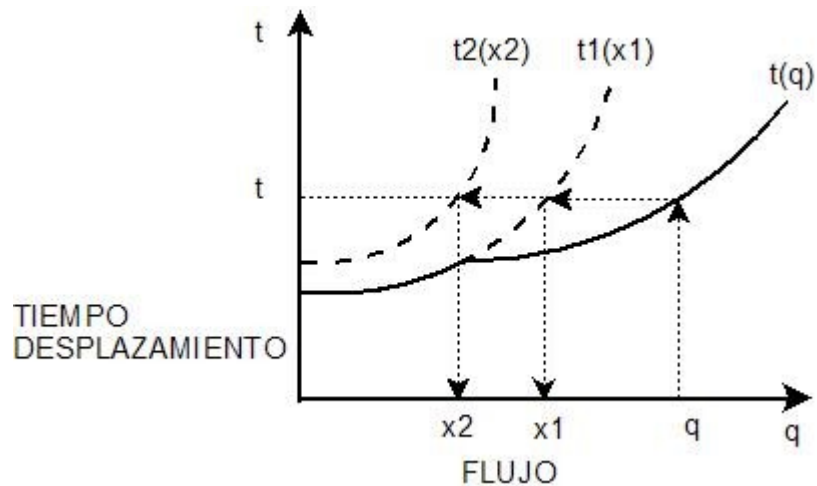


Imagen 1.6

Hay que fijarse en que la función de rendimiento de O-D coincide con la función de rendimiento de la arista 1 para $q < q'$. Esto quiere decir que para valores q menores a q' el tiempo de desplazamiento en equilibrio será dado por el tiempo del enlace 1. El tiempo de desplazamiento en el enlace 2 será mayor para este rango de flujos de O-D, tal y como lo exige la definición de equilibrio de usuarios.

El método gráfico que hemos usado para resolver este ejemplo no puede usarse para resolver grandes redes. En estas redes, el número de rutas que conectan cada pareja O-D será extremadamente elevado. Además, el flujo que pasa por cada arista proviene del desplazamiento de usuarios entre muchos pares O-D. En consecuencia, tal y como se ha mencionado anteriormente, toda la red tiene que ser resuelta de manera simultánea.

Capítulo 2 : Formulación del problema de asignación de flujos como un problema matemático

La asignación de tráfico o el problema de equilibrios en redes de transporte ya ha sido definido. Tal como hemos explicado, el problema es encontrar el flujo de los enlaces a partir del número de usuarios entre cada par origen-destino, la red, y las funciones de rendimiento asociadas a los enlaces.

Para encontrar el problema suponemos que cada usuario usa el camino que más minimiza el tiempo de desplazamiento entre su origen y su destino. Esta política de elección de rutas implica que en situación de equilibrio el tiempo de desplazamiento para todas las rutas usadas de cada pareja origen destino es el mismo. Además, este tiempo de desplazamiento será menor o igual al de las rutas no usadas. En este punto, podemos decir que la red está en un punto de equilibrio de usuario, ningún usuario podrá experimentar una reducción de tiempo si decide cambiar su ruta de forma unilateral.

En el capítulo anterior hemos enseñado cómo se puede encontrar el patrón de flujos para una red pequeña usando métodos gráficos. Desgraciadamente, estos métodos no pueden ser usados para resolver problemas con grandes redes, muchos enlaces y muchas parejas O-D.

El técnica descrita en este texto para resolver grandes problemas usa la técnica de la minimización equivalente. Esto quiere decir que hemos de formular un problema matemático, la solución del cual es el patrón de flujos en equilibrio de la red. Esta técnica suele ser usada en investigación operativa, en casos donde es más fácil minimizar el problema equivalente que resolver directamente un conjunto de restricciones.

Para que el problema de minimización sea útil es necesario que sólo tenga una única solución, que además cumpla las condiciones de equilibrio. Además, el problema tiene que ser relativamente fácil de resolver.

El propósito de este capítulo es estudiar la formulación del problema de minimización equivalente y sus propiedades.

Antes de examinar la formulación, presentaremos la notación para redes que usamos.

La red la representamos como un grafo dirigido que incluye un conjunto de nodos numerados de

forma consecutiva, N ; y un conjunto de aristas numeradas de forma consecutiva, A .

En muchos casos las aristas son identificadas usando los nodos que conecta (por ejemplo, la arista $m \rightarrow n$, tiene como origen el nodo m y destino el nodo n), ya que es muy útil para muchos algoritmos.

Nombraremos R al conjunto de nodos centroides origen (nodos en los que se introducen los flujos), y S al conjunto de nodos centroides destino (nodos en los que se finaliza los flujos).

El conjunto de nodos origen y el conjunto de nodos destino no son disjuntos, es decir, puede haber nodos que sean origen y destino de diferentes trayectos al mismo tiempo ($R \cap S \neq \emptyset$). Cada pareja O-D está conectada por una serie de caminos (rutas) a través de la red. A este conjunto le denominamos K_{rs} , donde r pertenece a R y s pertenece a S .

La matriz origen-destino se representa por q usando como entradas q_{rs} . En otras palabras,

q_{rs} , es el número de usuarios (flujo) que viajan desde r hasta s . También x_a y t_a representan el flujo y el tiempo de desplazamiento respectivamente sobre la arista a (donde a pertenece a A). Además, $t_a(\cdot)$ representa la relación entre el flujo y el tiempo de desplazamiento para la arista a . En otras palabras, $t_a(x_a)$ representa la función de rendimiento (también conocida como función de congestión de la arista o curva volumen-tiempo) asociada a la arista a .

De forma parecida, f_k^{rs} y c_k^{rs} representan el flujo y el tiempo de desplazamiento, respectivamente, de la ruta k que conecta el nodo origen r y el nodo destino s (k pertenece a K_{rs}).

El tiempo de desplazamiento de una ruta es la suma de los tiempos de desplazamiento de las aristas que forman esa ruta.

Esta relación puede formularse como :

$$c_k^{rs} = \sum_a t_a \delta_{a,k}^{rs} \quad \forall k \in K_{rs}, \quad \forall r \in R, \quad \forall s \in S$$

donde $\delta_{a,k}^{rs} = 1$ quiere decir que la arista a forma parte de la ruta k que conecta la pareja O-D $r-s$, y $\delta_{a,k}^{rs} = 0$ en caso contrario. Utilizando la misma variable como indicador, el

flujo en una arista puede ser expresado como una función dependiente del flujo de las rutas, es decir :

$$x_a = \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \quad \forall a \in A$$

Esta ecuación nos está diciendo que el flujo en cada arista es la suma de los flujos de todas las rutas que pasan a través de esa arista. Estas ecuaciones son conocidas como relaciones entre rutas y aristas.

Para facilitar la comprensión se eligió simplificar expresiones usando notación vectorial. Siguiendo

este criterio nos queda : $x = (\dots, x_a, \dots)$, $t = (\dots, t_a, \dots)$, $f^{rs} = (\dots, f_k^{rs}, \dots)$,
 $f = (\dots, f^{rs}, \dots)$, $c^{rs} = (\dots, c_k^{rs}, \dots)$, $c = (\dots, c^{rs}, \dots)$.

Además se suele usar Δ para representar la matriz de incidencias entre arista y rutas, los elementos de la matriz serán $\delta_{a,k}^{rs}$. Esta matriz se suele dividir en submatrices usando como criterio de separación las parejas O-D . Así obtenemos $\Delta = (\dots, \Delta^{rs}, \dots)$, donde Δ^{rs} es la matriz de incidencias entre rutas y aristas para la pareja O-D $r-s$. El número de filas de esta matriz es igual al número de aristas de la red y el número de columnas es igual a la cantidad de rutas existentes entre $r-s$. El elemento posicionado en la a-ésima fila y la k-ésima columna de la matriz Δ^{rs} es $\delta_{a,k}^{rs}$, en otras palabras $(\Delta^{rs})_{a,k} = \delta_{a,k}^{rs}$. Ahora podemos expresar algunos términos como multiplicación de matrices:

$$c = t * \Delta \quad , y, \quad x = f * \Delta^T$$

Veremos un pequeño ejemplo sobre esta notación.

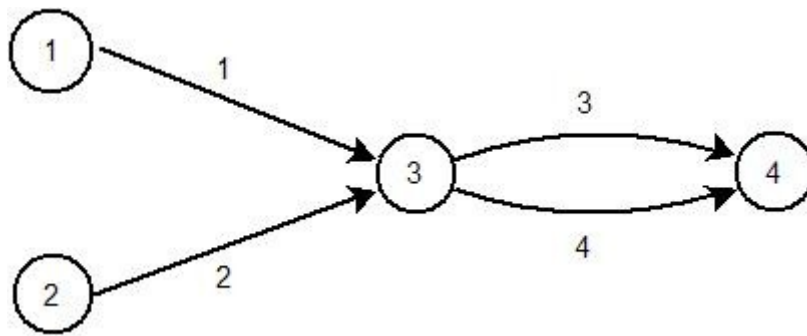


Imagen 2.1

Imaginemos que tenemos la red de la imagen 2.1 . Con dos parejas O-D 1-4 y 2-4, y 4 aristas.

Podemos asumir que para la pareja 1-4 la ruta 1 usa las aristas 1 y 3 y la ruta dos usa las aristas 1 y 4. Así mismo podemos asumir que para la pareja 2-4 la ruta 1 usa las aristas 2 y 3 y la ruta 2 usa las aristas 2 y 4.

Usando esta nomenclatura podemos escribir $\delta_{1,1}^{14} = 1$ y $\delta_{3,2}^{24} = 0$. Nos está diciendo que la arista 1 forma parte de la ruta 1 de la pareja 1-4 y la arista 3 no pertenece a la ruta 2 de la pareja 2-4. El uso de las ecuaciones que relacionan rutas y aristas nos dice cosas como,

$$c_1^{14} = t_1 \delta_{1,1}^{14} + t_2 \delta_{2,1}^{14} + t_3 \delta_{3,1}^{14} + t_4 \delta_{4,1}^{14} = t_1 + t_3$$

Nos enseña que el tiempo de desplazamiento de la ruta 1 de la pareja 1-4 es igual a la suma de los tiempos de las arista 1 y 3, algo fácilmente comprobable viendo el dibujo de la red. Generalizando se muestra que el tiempo de desplazamiento de una ruta es igual a la suma de los tiempos de las aristas que componen esa ruta.

También podemos escribir cosas como,

$$x_3 = f_1^{14} \delta_{3,1}^{14} + f_2^{14} \delta_{3,2}^{14} + f_1^{24} \delta_{3,1}^{24} + f_2^{24} \delta_{3,2}^{24} = f_1^{14} + f_1^{24}$$

Nos dice que el flujo de la arista 3 es igual al flujo de la ruta 1 de la pareja 1-4 más el flujo de la ruta 1 de la pareja 2-4 . Es decir, el flujo de una arista es igual a la suma del flujo de todas las rutas que incorporan esa arista.

A continuación mostramos una tabla que resume la notación usada para redes. Ampliaremos esta notación según lo necesitemos.

Notación básica de redes	
N	Conjunto de nodos de la red
A	Conjunto de aristas de la red
R	Conjunto de nodos centroides origen ; $R \subseteq N$
S	Conjunto de nodos centroides destino ; $S \subseteq N$
K_{rs}	Conjunto de rutas que unen a pareja O-D $r-s$; $r \in R, s \in S$
x_a	Flujo en la arista a ; $x = (\dots, x_a, \dots)$
t_a	Tiempo de desplazamiento para la arista a ; $t = (\dots, t_a, \dots)$
f_k^{rs}	Flujo en la ruta k que une la pareja O-D $r-s$; $f^{rs} = (\dots, f_k^{rs}, \dots)$; $f = (\dots, f^{rs}, \dots)$
c_k^{rs}	Tiempo de desplazamiento para la ruta k que une la pareja O-D $r-s$; $c^{rs} = (\dots, c_k^{rs}, \dots)$; $c = (\dots, c^{rs}, \dots)$
q_{rs}	Número de usuarios (flujo) que viajan desde r hasta s
$\delta_{a,k}^{rs}$	$\delta_{a,k}^{rs} = 1$, si la arista a pertenece a la ruta k de la pareja O-D $r-s$; $\delta_{a,k}^{rs} = 0$, en caso contrario $(\Delta^{rs})_{a,k} = \delta_{a,k}^{rs}$; $\Delta = (\dots, \Delta^{rs}, \dots)$

2.1. Formulación matemática del problema de asignación de tráfico en equilibrio de usuarios

El modelo matemático de optimización para la asignación de tráfico en equilibrio de usuario es :

$$\begin{aligned} \min \quad z(x) &= \sum_a \int_0^{x_a} t_a(w) dw \\ \text{s.t.} \quad x_a &= \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \quad \forall a \\ \sum_k f_k^{rs} &= q_{rs} \quad \forall r, s \\ f_k^{rs} &\geq 0 \quad \forall k, r, s \end{aligned}$$

La función objetivo de este modelo es la suma de las integrales de las funciones de rendimiento asociadas a las aristas de la red.

La primera restricción dice que el flujo que transcurre por una arista es igual a la suma del flujo de todas las rutas que usan esa arista.

La segunda restricción dice que para cada pareja O-D $r-s$, la suma de flujos de todas las rutas que unen r y s ha de ser igual al número de usuarios que viajan desde r hasta s .

La tercera restricción dice que el flujo de todas las rutas de todas las parejas O-D ha de ser mayor o igual a 0.

2.2. Técnicas de resolución del equilibrio de usuarios

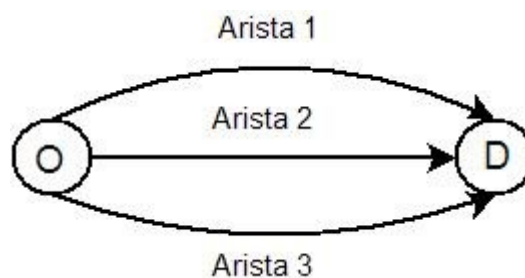
El problema de encontrar el equilibrio de usuarios sobre una red de transportes ya lo hemos planteado en los capítulos previos. Éste problema puede ser resumido como el problema de asignar todos los flujos entre parejas O-D sobre las aristas de la red, consiguiendo que para cada pareja O-D, el tiempo de desplazamiento entre todas las rutas usadas sea menor o igual al de todas las posibles rutas para O-D.

También hemos explicado la formulación matemática del problema de minimización equivalente.

En este capítulo pretendemos mostrar técnicas para resolverlo.

Lo dividiremos en dos partes, en la primera mostraremos un par de técnicas heurísticas para encontrar el patrón de flujos del equilibrio de usuarios. Este tipo de técnicas eran muy usadas antes del desarrollo de algoritmos para la resolución del equilibrio de usuarios, incluso hay sistemas que aún las usan. En la segunda parte explicaremos el algoritmo de las combinaciones convexas y cómo usarlo para resolver el problema de minimización para el equilibrio de usuarios (UE).

Para comprender mejor estas técnicas las aplicaremos sobre una pequeña red de ejemplo.



$$t_1 = 10 * \left(1 + 0.15 * \left(\frac{x_1}{2} \right)^4 \right)$$

$$t_2 = 20 * \left(1 + 0.15 * \left(\frac{x_2}{4} \right)^4 \right)$$

$$t_3 = 25 * \left(1 + 0.15 * \left(\frac{x_3}{3} \right)^4 \right)$$

$$x_1 + x_2 + x_3 = 10$$

2.3. Técnicas heurísticas para la búsqueda del equilibrio de usuarios

La cualidad principal de las técnicas heurísticas es la obtención de soluciones aproximadas requiriendo unos recursos computacionales muy inferiores comparados con los requeridos por otras técnicas que obtienen soluciones más precisas.

Nosotros nos centraremos en dos de las técnicas heurísticas que se usan para resolver problemas de equilibrio, “Capacity Restraint” y “Incremental Assignment”.

Un mecanismo fundamental de estas técnicas es el “Network Loading”. “Network Loading” es el proceso de asignación de todos los flujos entre parejas O-D sobre la red, teniendo en cuenta que el coste de desplazamiento para cada arista es constante. El criterio de selección de rutas será el usado en los modelos de asignación de tráfico, es decir, se escogerá la ruta más corta que una la pareja O-D. A esa ruta le asignaremos todo el flujo de O-D, este procedimiento es conocido como una asignación “all-or-nothing”.

En el procedimiento “all-or-nothing” recorreremos todas las parejas O-D, cada pareja O-D $r - s$ tiene un flujo asignado q_{rs} , q_{rs} es asignado a las aristas que forman parte del camino más corto que una el nodo centroide origen r y el nodo centroide final s . Todas las otras posibles rutas no reciben nada. Durante todo el procedimiento los tiempos de desplazamiento han de ser fijos, es decir, han de tener un valor.

Pero hay un problema, y es que los tiempos de desplazamiento de nuestras aristas dependen de funciones de rendimiento $t_a(x_a)$. Por ello cuando vayamos a aplicar el procedimiento “all-or-nothing” debemos especificar los tiempos usados.

Por ejemplo, aplicamos el procedimiento “all-or-nothing” a una red vacía, es decir, los tiempos de desplazamiento usados serán $t_a = t_a(0)$ para cada arista a .

El principal problema de esta técnica es la búsqueda de la ruta más corta que una cada pareja O-D. De hecho este problema es el que supone la principal dificultad computacional, hasta el punto que le destinaremos un capítulo aparte.

Cabe recordar que el procedimiento “all-or-nothing” no reconoce la dependencia entre los flujos de las aristas y los tiempos de desplazamiento en ellas, es decir ignora la existencia de equilibrios con lo que por si solo no sirve para la búsqueda de equilibrios.

2.3.1. Capacity restraint

En uno de los primeros intentos para encontrar equilibrios en problemas de asignación de tráfico, los planificadores de rutas decidieron crear un algoritmo iterativo conocido como “Capacity Restraint”. Esta técnica se basa en repetir el procedimiento de asignación “all-or-nothing” usando los tiempos de desplazamiento obtenidos en la anterior iteración.

Paso 0 : Inicialización.

Realizar una asignación “all-or-nothing” usando como tiempos $t_a^0 = t_a(0)$.

Calcular los flujos de las aristas $\{ x_a^0 \}$.

Poner el contador de iteraciones a 1 $\{ n := 1 \}$

Paso 1: Update.

Calcular: $t_a^n = t_a(x_a^{n-1})$

Paso 2: “Network Loading”.

Asignar todos los flujos a las red usando el procedimiento “all-or-nothing” junto a los tiempos de desplazamiento $\{ t_a^n \}$. Esto nos dará como resultado un conjunto de flujos asignados a las aristas de la red $\{ x_a^n \}$.

Paso 3: Test de convergencia.

Si $\max_a \{ |x_a^n - x_a^{n-1}| \} \leq \varepsilon$, STOP . El conjunto actual de flujos asignados a las aristas $\{ x_a^n \}$ será la solución.

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

La tabla siguiente nos muestra la aplicación de esta técnica en la red de ejemplo. Al observar los resultados vemos que el algoritmo no converge, sólo se dedica a cambiar el flujo de la arista 1 a la arista 2 y viceversa, olvidándose de la arista 3.

Número de iteración	Paso del algoritmo	Arista		
		1	2	3
0	Inicialización	$t_1^0 = 10$	$t_2^0 = 20$	$t_3^0 = 25$
		$x_1^0 = 10$	$x_2^0 = 0$	$x_3^0 = 0$
1	Actualización	$t_1^1 = 947$	$t_2^1 = 20$	$t_3^1 = 25$
	Loading	$x_1^1 = 0$	$x_2^1 = 10$	$x_3^1 = 0$
2	Actualización	$t_1^2 = 10$	$t_2^2 = 137$	$t_3^2 = 25$
	Loading	$x_1^2 = 10$	$x_2^2 = 0$	$x_3^2 = 0$
3	Actualización	$t_1^3 = 947$	$t_3^3 = 20$	$t_3^3 = 25$
	Loading	$x_1^3 = 0$	$x_3^3 = 10$	$x_3^3 = 0$
	⋮	⋮	⋮	⋮

Para corregir estos errores debemos modificar el algoritmo. Primero, en el paso 2 en vez de usar los tiempos de desplazamiento de la iteración anterior, usaremos una combinación de los tiempos obtenidos en las dos iteraciones previas. Esto introduce un efecto de suavizado “smoothing”. Segundo, aceptamos que el algoritmo no converge por lo que pararemos después de un determinado número de iteraciones N. El patrón de flujos final será la media de los flujos de las 4 últimas iteraciones (obviamente, N será mayor o igual a 4).

Así que el algoritmo queda así:

Paso 0 : Inicialización.

Realizar una asignación “all-or-nothing” usando como tiempos $t_a^0 = t_a(0)$.

Calcular los flujos de las aristas $\{ x_a^0 \}$.

Poner el contador de iteraciones a 1 $\{ n := 1 \}$.

Paso 1: Update.

Calcular: $\tau_a^n = t_a(x_a^{n-1})$

Paso 2: “Smoothing”.

Calcular:

$$t_a^n = 0.75 t_a^{n-1} + 0.25 \tau_a^n$$

Paso 3: “Network Loading”.

Asignar todos los flujos a las red usando el procedimiento “all-or-nothing” junto a los tiempos de desplazamiento $\{ t_a^n \}$. Esto nos dará como resultado un conjunto de

flujos asignados a las aristas de la red $\{ x_a^n \}$.

Paso 4: Test de parada.

Si $n = N$, ir al paso 5.

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

Paso 5: “Averaging”.

Calcular $x_a^* = 1/4 \sum_{l=0}^3 x_a^{n-l}$, STOP. El conjunto de flujos resultante $\{ x_a^* \}$ será la solución.

La siguiente tabla nos muestra la aplicación de esta técnica modificada en la red de ejemplo. Al observar los resultados vemos que la solución no es un equilibrio de usuarios. Hemos conseguido que se usen las tres rutas pero los tiempos de la ruta 1 son muy diferentes de los de las rutas 2 y 3.

Número de iteración	Paso del algoritmo	Arista		
		1	2	3
0	Inicialización	$t_1^0 = 10$	$t_2^0 = 20$	$t_3^0 = 25$
		$x_1^0 = 10$	$x_2^0 = 0$	$x_3^0 = 0$
1	Update	$\tau_1^1 = 947$	$\tau_2^1 = 20$	$\tau_3^1 = 25$
	Smoothing	$t_1^1 = 244$	$t_2^1 = 20$	$t_3^1 = 25$
	Loading	$x_1^1 = 0$	$x_2^1 = 10$	$x_3^1 = 0$
2	Update	$\tau_1^2 = 10$	$\tau_2^2 = 137$	$\tau_3^2 = 25$
	Smoothing	$t_1^2 = 186$	$t_2^2 = 49$	$t_3^2 = 25$
	Loading	$x_1^2 = 0$	$x_2^2 = 0$	$x_3^2 = 10$

3	Update	$\tau_1^3=10$	$\tau_2^3=20$	$\tau_3^3=488$
	Smoothing	$t_1^3=142$	$t_3^3=42$	$t_3^3=141$
	Loading	$x_1^3=0$	$x_3^3=10$	$x_3^3=0$
Average		$x_1^*=2.5$	$x_2^*=5.0$	$x_3^*=2.5$
		$t_1^*=13.7$	$t_2^*=27.3$	$t_3^*=26.8$

2.3.2.Incremental Assignment

Otra técnica usada para buscar el equilibrio de usuarios es la “Incremental Assignment” o asignación incremental, se basa en asignar sólo una parte del flujo de cada pareja O-D en cada iteración. Después de cada asignación volvemos a actualizar los tiempos de desplazamiento e iniciamos otra iteración.

El algoritmo para la técnica “Incremental Assignment” es:

Paso 0 : Inicialización.

Dividir el flujo de cada pareja O-D en N partes iguales, es decir:

$$\text{Calcular: } q_{rs}^n = q_{rs} / N$$

Poner el contador de iteraciones a 1 { $n := 1$ }.

Poner el flujo de las aristas a 0 { $x_a^0 = 0$ }.

Paso 1: Update.

$$\text{Calcular: } t_a^n = t_a(x_a^{n-1})$$

Paso 2: “Incremental loading”.

Asignar la fracción q_{rs}^n de cada pareja O-D usando la técnica “all-or-nothing” junto a los tiempos de desplazamiento { t_a^n }. Nos proporciona un patrón de flujos { w_a^n }.

Paso 3: Sumo de flujos.

$$\text{Calcular } x_a^n = x_a^{n-1} + w_a^n .$$

Paso 4: Test de parada.

Si $n = N$, STOP. El conjunto actual de flujos asignados a las aristas $\{ x_a^n \}$ será la solución.

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

La siguiente tabla nos muestra la aplicación de esta técnica en la red de ejemplo usando $N=4$.

El patrón de flujos resultante es $x = (5, 5, 0)$. Al observar los tiempos t_1^* , t_2^* , t_3^* vemos que la solución no es un equilibrio de usuarios. Hemos conseguido que se usen las tres rutas pero los tiempos de la ruta 1 son muy inferiores de los de las rutas 2 y 3.

Número de iteración	Paso del algoritmo	Arista		
		1	2	3
1	Update	$t_1^1 = 10$	$t_2^1 = 20$	$t_3^1 = 25$
	Incremental loading	$w_1^1 = 2.5$	$w_2^1 = 0$	$w_3^1 = 0$
	Summation	$x_1^1 = 2.5$	$x_2^1 = 0$	$x_3^1 = 0$
2	Update	$t_1^2 = 14$	$t_2^2 = 20$	$t_3^2 = 25$
	Incremental loading	$w_1^2 = 2.5$	$w_2^2 = 0$	$w_3^2 = 0$
	Summation	$x_1^2 = 5$	$x_2^2 = 0$	$x_3^2 = 0$
3	Update	$t_1^3 = 69$	$t_2^3 = 20$	$t_3^3 = 25$
	Incremental loading	$w_1^3 = 0$	$w_2^3 = 2.5$	$w_3^3 = 0$
	Summation	$x_1^3 = 5$	$x_2^3 = 2.5$	$x_3^3 = 0$
4	Update	$t_1^4 = 69$	$t_2^4 = 20.5$	$t_3^4 = 25$
	Incremental loading	$w_1^4 = 0$	$w_2^4 = 2.5$	$w_3^4 = 0$
	Summation	$x_1^4 = 5$	$x_2^4 = 5$	$x_3^4 = 0$

Tiempo de desplazamiento en la convergencia	$t_1^* = 69$	$t_2^* = 27.3$	$t_3^* = 25$
---	--------------	----------------	--------------

Es fácil ver que cuanto más incrementemos el valor N más nos aproximaremos al equilibrio de usuarios.

Una modificación común en el algoritmo es modificar el paso 2. Escoger las parejas O-D de forma aleatoria, usar la técnica “all-or-nothing” sólo con la pareja seleccionada. Actualizar los flujos y los tiempos de la red antes de seleccionar la siguiente pareja O-D.

2.4. Algoritmo de las combinaciones convexas para la búsqueda del equilibrio de usuarios

El algoritmo de las combinaciones convexas fue propuesto por primera vez por Frank y Wolfe en el año 1956. En principio estaba destinado para resolver problemas con funciones objetivo de orden cuadrático con restricciones lineales. Pero posteriormente fue generalizado y pasó a ser aplicable a cualquier función objetivo no lineal, continua, diferenciable y convexa. El algoritmo de Frank-Wolfe ha resultado ser muy útil para la búsqueda de equilibrios en redes de transporte.

Este algoritmo se engloba dentro del grupo de técnicas de búsqueda de direcciones de descenso. Estos algoritmos son algoritmos de tipo iterativo. Para cada iteración partimos de un punto obtenido de la iteración anterior, estudiamos la forma de la función próxima al punto y escogemos la dirección que nos proporcione un mayor descenso. Seguimos esa dirección hasta el punto en el que no podremos seguir descendiendo. Si ese nuevo punto no nos sirve como solución iniciamos una nueva iteración. Para entender mejor el funcionamiento de estos algoritmos mostramos la imagen 2.2 que muestra el comportamiento del algoritmo en un caso simple.

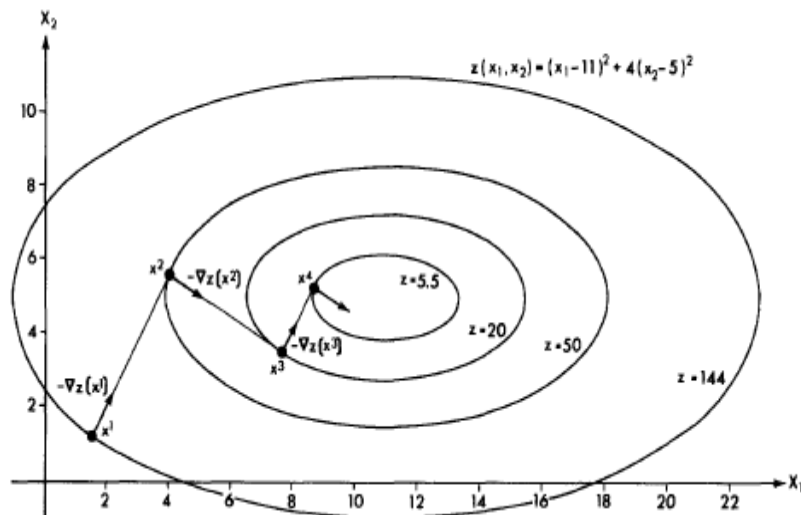


Imagen 2.2

El algoritmo de Frank-Wolfe selecciona la dirección de descenso respecto al punto x^n teniendo en cuenta dos criterios simultáneamente. El primero, la mayor dirección de descenso de la función objetivo en la vecindad del punto x^n . El segundo, cómo de lejos podemos llegar siguiendo la dirección escogida en el primer criterio.

Para encontrar la dirección de descenso, el algoritmo busca en el dominio factible de la función una solución auxiliar y^n , con la que la dirección de x^n a y^n nos de el mayor avance. La dirección de descenso nos la provee el vector unitario

$$\frac{(y^n - x^n)^T}{|y^n - x^n|}$$

y la pendiente de esta dirección nos la provee la proyección del gradiente negativo de la función objetivo en el punto x^n en la dirección de descenso, obteniendo :

$$-\nabla f(x^n) * \frac{(y^n - x^n)^T}{|y^n - x^n|}$$

El beneficio conseguido al movernos desde x^n a y^n es :

$$-\nabla f(x^n) * (y^n - x^n)^T$$

Ahora hemos de buscar el punto y factible que maximice el beneficio obtenido con el movimiento descrito en la expresión anterior. Para ello planteamos un problema matemático de minimización, convertimos la maximización en una minimización multiplicando la expresión anterior por -1 . Además como x^n es constante podemos eliminarlo de la expresión

$$(y - x^n)^T \text{ quedándonos } y^T.$$

Así que el problema a resolver queda así:

$$\begin{aligned} \min z(y) &= \nabla f(x^n) * y^T = \sum_i \left(\frac{\partial z(x^n)}{\partial x_i} \right) * y_i \\ \text{s.t.} & \\ & y \in P \end{aligned}$$

Siendo P la región factible de la función objetivo.

Una vez obtenido y ya tenemos la dirección de descenso. Ahora hemos de buscar un punto en el intervalo entre y^n y x^n que minimice el valor de la función objetivo $f(x)$. A este nuevo punto lo llamaremos x^{n+1} , porque será la solución de esta iteración. Este punto lo obtendremos resolviendo el problema matemático siguiente :

$$\begin{aligned} \min_{\lambda} f(x^n + \lambda(y^n - x^n)) \\ \text{s.t.} \\ 0 \leq \lambda \leq 1 \end{aligned}$$

que nos dará un factor λ , con el que podremos obtener x^{n+1} aplicando la siguiente fórmula:

$$x^{n+1} = x^n + \lambda(y^n - x^n)$$

Ahora sólo nos falta ver si este nuevo punto lo aceptamos como solución o iniciamos una nueva iteración, para ello aplicamos un test de convergencia.

Dada una solución factible x^n , la n -ésima iteración de la versión generalizada del algoritmo de Frank-Wolfe puede ser resumida así:

Paso 1: Búsqueda de la dirección de descenso.

Buscar y^n que resuelva el problema siguiente:

$$\begin{aligned} \min \quad & z(y) = \nabla f(x^n) * y^T = \sum_i \left(\frac{\partial z(x^n)}{\partial x_i} \right) * y_i \\ \text{s.t.} \quad & \\ & y \in P \end{aligned}$$

Paso 2: Determinar la longitud del paso.

Buscar λ_n que resuelva el siguiente problema:

$$\begin{aligned} \min_{\lambda} \quad & f(x^n + \lambda(y^n - x^n)) \\ \text{s.t.} \quad & \\ & 0 \leq \lambda \leq 1 \end{aligned}$$

Paso 3: Actualizar la solución.

$$\text{Calcular } x^{n+1} = x^n + \lambda(y^n - x^n)$$

Paso 4: Test de convergencia.

$$\text{Si } f(x^n) - f(x^{n+1}) \leq K, \text{ STOP.}$$

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

Algoritmo de Frank-Wolfe adaptado para la búsqueda de equilibrios en redes de transporte

Paso 0: Inicialización.

Realizar una asignación “all-or-nothing” usando como tiempos $t_a^0 = t_a(0)$.

Calcular los flujos de las aristas $\{x_a^1\}$.

Poner el contador de iteraciones a 1 $\{n := 1\}$.

Paso 1: Actualización de los costes.

$$\text{Calcular: } t_a^n = t_a(x_a^n)$$

Paso 2: Búsqueda de la dirección de descenso.

Realizar una asignación “all-or-nothing” usando como tiempos t_a^n para obtener los flujos auxiliares $\{ y_a^n \}$.

Paso 3: Determinar la longitud de paso usando búsqueda lineal.

Buscar la λ_n que resuelva el problema de búsqueda lineal:

$$\begin{aligned} \min \quad & \sum_a \int_0^{x_a^n + \lambda(y_a^n - x_a^n)} t_a(w) dw \\ \text{s.t.} \quad & 0 \leq \lambda \leq 1 \end{aligned}$$

Paso 4: Actualizar la solución.

$$\text{Calcular } x_a^{n+1} = x_a^n + \lambda_n (y_a^n - x_a^n), \quad \forall a \in A$$

Paso 5: Test de convergencia.

Aplicar un test de convergencia.

Si se satisface el test de convergencia, STOP. La solución actual $\{ x_a^{n+1} \}$ es el patrón de flujos del equilibrio de usuarios.

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

La siguiente tabla nos muestra la aplicación de esta técnica en la red de ejemplo. Al observar los resultados vemos que el algoritmo converge. El patrón de flujos final (3.54, 4.7, 1.71) nos genera una situación de equilibrio de usuarios.

Número de iteración	Paso del algoritmo	Arista			Función objetivo	Longitud del paso
		1	2	3		
0	Inicialización	$t_1^0=10$ $x_1^1=10$	$t_2^0=20$ $x_2^1=0$	$t_3^0=25$ $x_3^1=0$		
1	Update Dirección Movimiento	$t_1^1=947$ $y_1^1=0$ $x_1^2=4.04$	$t_2^1=20$ $y_2^1=10$ $x_2^2=5.96$	$t_3^1=25$ $y_3^1=0$ $x_3^2=0$	$z(x)=1975.00$	$\lambda=0.596$
2	Update Dirección Movimiento	$t_1^2=35$ $y_1^2=0$ $x_1^3=3.39$	$t_2^2=35$ $y_2^2=0$ $x_2^3=5$	$t_3^2=25$ $y_3^2=10$ $x_3^3=1.61$	$z(x)=197.00$	$\lambda=0.161$
3	Update Dirección Movimiento	$t_1^3=22.3$ $y_1^3=10$ $x_1^4=3.62$	$t_2^3=27.3$ $y_2^3=0$ $x_2^4=4.83$	$t_3^3=35.3$ $y_3^3=0$ $x_3^4=1.55$	$z(x)=189.98$	$\lambda=0.035$
4	Update Dirección Movimiento	$t_1^4=26.1$ $y_1^4=0$ $x_1^5=3.54$	$t_2^4=26.3$ $y_2^4=0$ $x_2^5=4.73$	$t_3^4=25.3$ $y_3^4=10$ $x_3^5=1.72$	$z(x)=189.44$	$\lambda=0.020$
5	Update Dirección Movimiento	$t_1^5=24.8$ $y_1^5=10$ $x_1^6=3.59$	$t_2^5=25.8$ $y_2^5=0$ $x_2^6=4.7$	$t_3^5=25.4$ $y_3^5=0$ $x_3^6=1.71$	$z(x)=189.33$	$\lambda=0.007$
	Actualización	$t_1^6=25.6$	$t_2^6=25.7$	$t_3^6=25.4$	$z(x)=189.33$	

Como podemos ver en la tabla, la técnica de Frank-Wolfe, en cada iteración coge flujo de las rutas congestionadas y lo deriva a las menos congestionadas. Este proceso va igualando los tiempos de todas las rutas y va acercando el sistema a la situación de equilibrio. Además, podemos apreciar que cada nueva iteración aporta menos a la mejora de la función objetivo.

En el proceso de resolución del problema de equilibrio de usuarios UE, cada iteración comporta un importante coste computacional, debido al esfuerzo que implica la búsqueda de las rutas mínimas

en el paso de búsqueda de la dirección de descenso. Por ello es importante obtener una buena solución con pocas iteraciones.

Con el algoritmo de Frank-Wolfe esto no es problema por dos motivos. El primero es, que las primeras iteraciones del algoritmo de Frank-Wolfe son las más eficientes en términos de la función objetivo. Es decir, que con unas pocas iteraciones podemos obtener una solución muy próxima al equilibrio. La segunda es que los criterios de convergencia usados no son muy estrictos, lo cual favorece su cumplimiento después de unas pocas iteraciones. El motivo de esto es que la precisión de la información inicial del sistema no justifica el esfuerzo necesario para obtener un patrón de flujos excesivamente preciso.

La congestión de la red es un factor muy importante a la hora de saber el número de iteraciones necesarias para resolver el problema. El redes muy descongestionadas la solución será muy parecida al resultado de la primera iteración. Según aumenta la congestión del sistema se irá necesitando más iteraciones. Este hecho lo podemos observar en la imagen 2.3, donde se muestra el patrón de convergencia del algoritmo de Frank-Wolfe para una red mediana y tres niveles de congestión. El patrón de convergencia mostrado en la imagen se basa en los valores de las función objetivo. Las tres curvas de la imagen corresponden con tres niveles de congestión de la red, bajo, medio y alto. Tal como se puede observar, la congestión del sistema es un factor importante a la hora de alcanzar la convergencia.

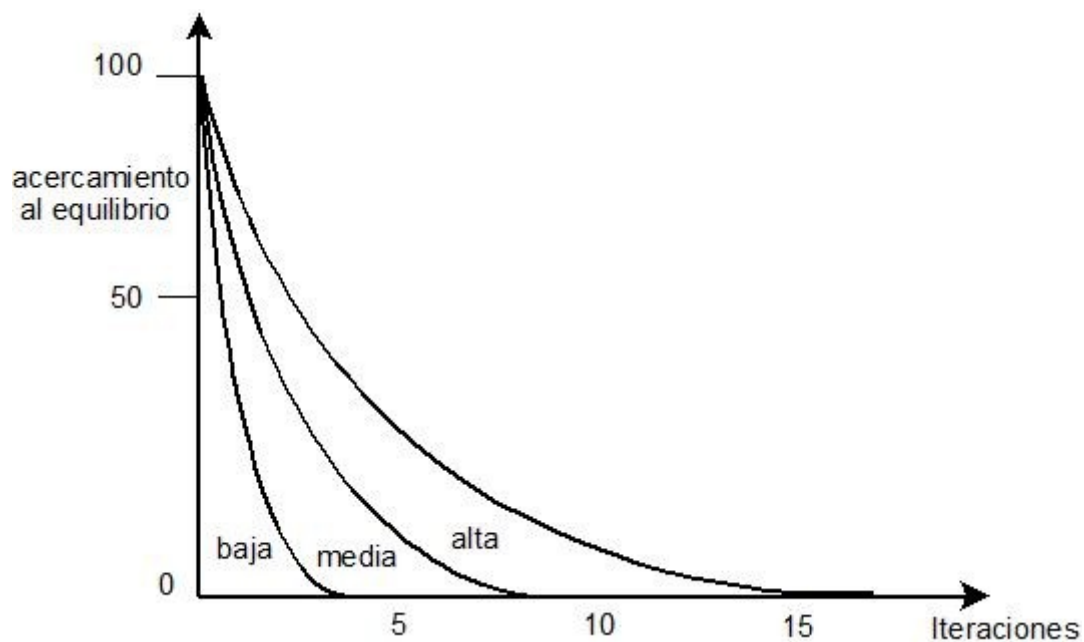


Imagen 2.3

Capítulo 3: Minimización unidimensional

Este capítulo trata sobre la minimización de una función no lineal con una sola variable, $z(x)$. Las condiciones explicadas en el capítulo X se han de cumplir también aquí; es decir, la variable x debe estar dentro de un intervalo finito $[a, b]$ y la función $z(x)$ debe ser continua y debe tener sólo un valor imagen para cualquier valor de x dentro del intervalo. Estas condiciones nos garantizan la existencia de un valor mínimo en $z(x)$ para alguna x en el intervalo fijado. Para el propósito de este capítulo además asumiremos que $z(x)$ es unimodal en el intervalo $[a, b]$, esto implica que sólo habrá un único mínimo en este intervalo.

El estudio de la minimización univariante es importante porque suele formar parte de los algoritmos o técnicas de resolución propuestas para la resolución de problemas de búsqueda mínimos en funciones multivariantes. Además, algunos de los principios usados en los algoritmos de resolución de funciones univariantes también son usados en los algoritmos de resolución de funciones multivariantes.

Este capítulo pretende explicar dos familias de algoritmos usadas para la minimización unidimensional. La primera es conocida como métodos de reducción de intervalo (*interval reduction*), e incluye las técnicas sección áurea (*golden section*) y bisección (*bisection method*). La segunda es conocida como ajuste de curvas cuadráticas (*quadratic curve fitting*), e incluye el método de Newton (*Newton's search*), el método de la falsa posición (*false position method*) y el de la aproximación cuadrática (*quadratic approximation method*).

3.1. Métodos de reducción de intervalo

Los métodos de reducción de intervalo son técnicas iterativas donde cada iteración trabaja con un determinado intervalo. El intervalo usado en la n -ésima iteración es una porción del intervalo inicial $[a, b]$, indicado como $[a^n, b^n]$, y que ha sido calculado para contener el punto mínimo x^* . En cada iteración este intervalo es estudiado y dividido en dos porciones: la porción en el que el punto mínimo no puede caer y la porción que contiene el punto mínimo. La porción que no contiene el punto mínimo es descartada y la otra porción será la que usemos en la siguiente iteración. Estas técnicas comienzan seleccionando para la primera iteración como intervalo inicial $[a, b]$ (es decir, $a^0 = a$ y $b^0 = b$). El intervalo será reducido en cada iteración hasta obtener una buena aproximación a x^* (es decir, un intervalo muy reducido).

Para comprender mejor el proceso de reducción de intervalo, definiremos la longitud del intervalo de la iteración n como I_n y el ratio de reducción del intervalo en la n -ésima iteración como $r_n = I_{n+1}/I_n$. En principio no tenemos motivos para decir que una de estas técnicas funcione mejor, en términos de reducción de intervalos, porque el ratio de reducción suele ser una constante ($r_n = r$ para cualquier iteración n).

Los algoritmos de reducción de intervalo suelen terminar cuando la longitud del intervalo actual I_n sea menor que una determinada constante. Entonces calcularemos x^* como el punto medio \bar{x} , del intervalo que nos queda después de las N iteraciones, I_n , es decir, $\bar{x} = (a^N + b^N)/2$. Si el punto óptimo debe de calcularse con un margen de error $\pm \epsilon$, entonces el número de iteraciones necesarias puede ser calculado con una función dependiente de la longitud del intervalo inicial I_0 .

La función sería :

$$N = ENT \left[\frac{\log(2\epsilon - \log(I_0))}{\log(r)} + 1 \right]$$

Función 3.1

donde $ENT[x]$ quiere decir la parte entera del argumento x .

Los diferentes algoritmos de reducción de intervalo difieren entre ellos en las reglas usadas para examinar el intervalo actual y decidir qué porción puede ser descartado.

Método de la sección áurea

La estrategia usada por el método de la sección áurea para la reducción del intervalo de búsqueda está basada en la comparación del valor de de la función $z(x)$ en dos puntos, x_L^n y x_R^n , donde $x_L^n < x_R^n$. Estos dos puntos deben estar dentro del intervalo de búsqueda $[a^n, b^n]$ de la n -ésima iteración. La política de selección de los puntos interiores es la característica más destacable de este método, pero primero explicaremos la técnica para descartar intervalos.

La técnica para descartar intervalos la explicaremos usando la imagen 3.2, la imagen representa una función unimodal, $z(x)$, en la cual se debe minimizar en el intervalo $[a^n, b^n]$. La gráfica superior (etiquetada como "Iteración N") muestra los dos puntos interiores de la n -ésima iteración

x_L^n y x_R^n . En este caso $z(x_L^n) > z(x_R^n)$, y como la función es ditónica, el mínimo ha de estar “a la derecha” de x_L^n ($x^* \geq x_L^n$), con esta información el intervalo $[a^n, x_L^n]$ queda descartado y la iteración enésima finaliza. El intervalo de la siguiente iteración, (la enésima +1), será $[a^{n+1}, b^{n+1}]$, donde $a^{n+1} = x_L^n$ y $b^{n+1} = b^n$; y los dos nuevos puntos interiores son x_L^{n+1} y x_R^{n+1} . La gráfica inferior nos muestra esta nueva iteración etiquetada como “Iteración N+1”. Si en la iteración enésima $z(x_L^n) < z(x_R^n)$, entonces hubiéramos descartado el intervalo $[x_R^n, b^n]$. Este es el caso de la iteración n+1, donde $z(x_L^{n+1}) < z(x_R^{n+1})$, así que descartamos el intervalo $[x_R^{n+1}, b^{n+1}]$ y el nuevo intervalo de búsqueda será $[a^{n+1}, x_R^{n+1}]$.

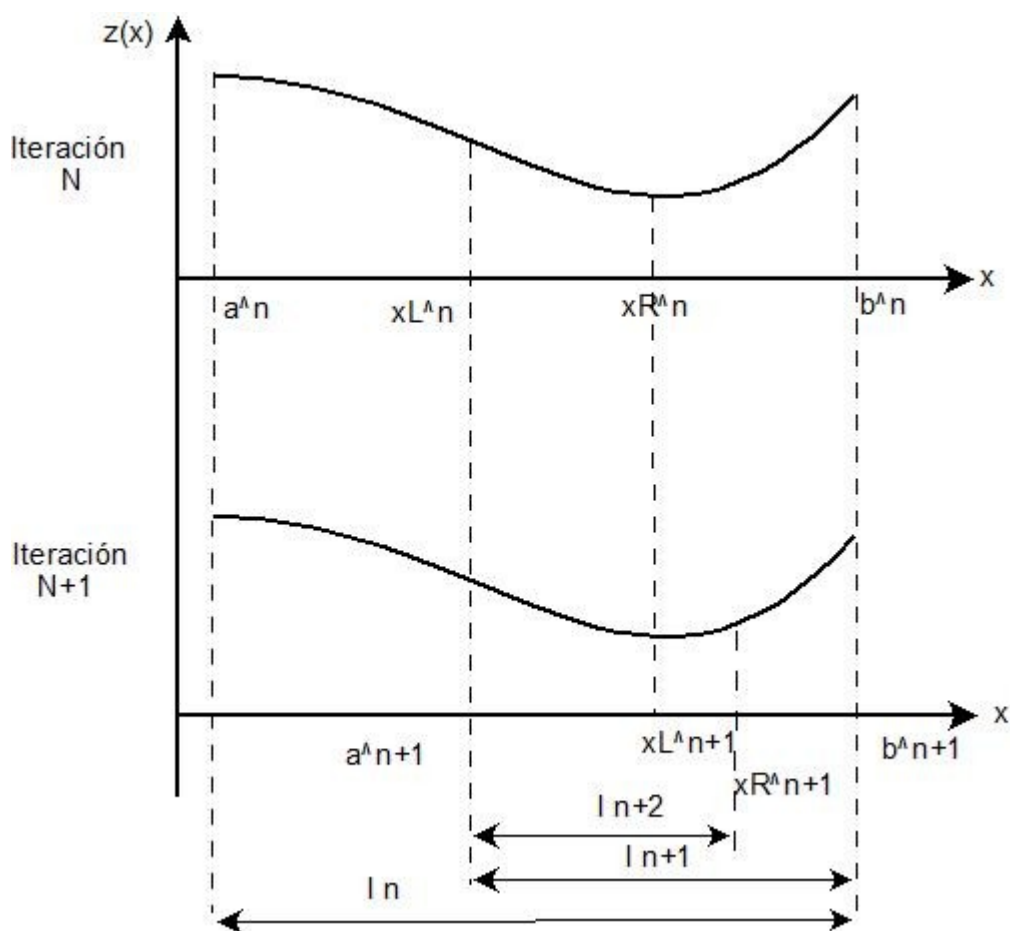


Imagen 3.2 : Secuencia de intervalos de reducción generados por el método de la sección áurea

La esencia del método de la sección áurea es la técnica usada para escoger x_L^n y x_R^n , dado un intervalo $[a^n, b^n]$. Esta técnica está diseñada para minimizar el número de evaluaciones de la

función. Cada vez que tenemos un nuevo intervalo el algoritmo necesita dos puntos interiores para trabajar. La técnica de la sección áurea usa uno de los puntos interiores de la última iteración (ya conocemos el valor de la función para este punto porque ya lo hemos calculado en la iteración anterior). Así que solo necesitamos evaluar un nuevo punto en cada iteración.

Esto lo podemos ver en la imagen X.X, donde en el paso de ir de la iteración N a la N+1 sólo hemos tenido que añadir y evaluar un nuevo punto, este punto es x_R^{n+1} de la gráfica inferior, el otro punto interior proviene de la iteración N, es decir, $x_L^{n+1} = x_R^n$. Se puede obtener una secuencia de puntos interiores manteniendo un ratio de reducción constante igual a la constante $r = 0.618$, más exacto

$r = \frac{1}{2} * (\sqrt{5} - 1)$. Manteniendo esta condición los puntos interiores son calculados de la siguiente

manera, x_R^n está a la derecha de a^n a la distancia de 0.618 de la longitud total del intervalo y

x_L^n está a la izquierda de b^n a la distancia de 0.618 de la longitud total del intervalo, también

podemos decir x_L^n está a la derecha de a^n a la distancia de 0.382 de la longitud total del intervalo.

El número $\frac{1}{2} * (\sqrt{5} - 1)$ es conocido como la sección áurea y de ahí deriva el nombre del método.

Debido al uso de esta constante y tal como podemos observar en la imagen, la secuencia de intervalos calculados durante una ejecución cumplen la siguiente condición:

$$I_n = I_{n+1} + I_{n+2}$$

Un método muy parecido al de la sección áurea es el de Fibonacci. Es ligeramente más eficiente gracias a la mejor posición de los primeros puntos interiores, pero requiere más estudio y añade más dificultad al sistema y no sale a cuenta respecto a la mejora que aporta.

A continuación mostramos en pseudo-código el algoritmo de la sección áurea. Este algoritmo necesita como parámetros de entrada la función a minimizar, los dos puntos que delimitan el intervalo de búsqueda y la precisión necesitada. Y genera como salida una estimación del punto óptimo x^* , el número de iteraciones N y la precisión del resultado, calculada como

$\frac{1}{2} * (b^N - a^N)$. Cabe recordar que el número de iteraciones necesarias puede ser obtenido a partir

de la precisión necesitada usando la función 3.1 de la introducción del capítulo.

Algoritmo del método de la sección áurea
goldenSection($z[x]$, a , b , ϵ)

```
{
     $r := 0.6180333$ 
     $n := 0$ 
     $continuar := True$ 
     $x_L := [b - a][1 - r] + a$ 
     $x_R := [b - a]r + a$ 
     $x := 0$ 
     $margenError := 0$ 
while (continuar)
{
     $n := n + 1$ 
    if (  $Z[x_L] \leq Z[x_R]$  )
    {
         $b := x_R$ 
         $x_R := x_L$ 
         $x_L := [b - a][1 - r] + a$ 
    }
    else
    {
         $a := x_L$ 
         $x_L := x_R$ 
         $x_R := [b - a]r + a$ 
    }
    if (  $b - a \leq 2\epsilon$  )
    {
         $continuar := False$ 
    }
}
 $x := 1/2[b + a]$ 
 $margenError := 1/2[b - a]$ 
```

```

    return x, n, margenError
}

```

Método de la bisección

Este método también es conocido como el método de Bolzano. En muchos casos, la derivada de la función a minimizar puede ser calculada fácilmente, este hecho se puede usar a la hora de buscar el mínimo. El método de la bisección aprovecha el hecho que la función sea unimodal y monótona a cada lado del punto mínimo. En otras palabras, la derivada de la función a minimizar, $dz(x)/dx$, es negativa para valores menores al mínimo y positiva para valores mayores al mínimo.

El algoritmo calcula la derivada de la función $z(x)$ en el punto medio del intervalo de la iteración actual N , $[a^n, b^n]$. Este punto lo llamaremos x^n . Si $dz(z^n)/dx < 0$, entonces $x^* > x^n$, esto implica que el intervalo $[a^n, x^n]$ no contiene el mínimo y puede ser descartado. El intervalo de búsqueda de la iteración siguiente será $[x^n, b^n]$. Si $dz(z^n)/dx > 0$, entonces $x^* < x^n$, y la búsqueda podrá centrarse en el intervalo $[a^n, x^n]$.

A continuación mostramos en pseudo-código el algoritmo del método de la bisección. Este algoritmo necesita como parámetros de entrada la función a minimizar, los dos puntos que delimitan el intervalo de búsqueda y la precisión necesitada. Y genera como salida una estimación del punto óptimo x^* , el número de iteraciones N y la precisión del resultado, calculada como

$$\frac{1}{2} * (b^N - a^N) .$$

Algoritmo del método de la bisección

```

biseccion( z[x] , a , b , ε )

```

```

{
    n := 0
    x := 0
    continuar := True
    margenError := 0
    while (continuar)
    {
        n := n + 1
        x := [b + a] / 2
    }
}

```

```

if (  $dZ[x]/dx \leq 0$  )
{
     $a := x$ 
}
else
{
     $b := x$ 
}
if (  $b - a \leq 2 * \epsilon$  )
{
     $continuar := False$ 
}
}
 $x := 1/2[b + a]$ 
 $margenError := 1/2[b - a]$ 
return x, n, margenError
}

```

Igual que en el método de la sección áurea el margen de error del problema puede ser especificado, a partir de este valor definimos el criterio de parada del algoritmo, El ratio de reducción de esta técnica es $r = 0.5$ y junto con el margen de error los podemos usar para calcular el número de iteraciones del algoritmo usando usando la función X.1 de la introducción del capítulo. Por ejemplo, con seis iteraciones el método de la bisección puede calcular el mínimo de $z(x)$ con una precisión del $\pm 1\%$ de la longitud del intervalo inicial. En cambio el método de la sección áurea necesita 9 iteraciones y 10 evaluaciones de la función $z(x)$ para obtener esta precisión. A continuación mostramos la tabla 3.1 que muestra el ratio entre el intervalo de la iteración actual y el intervalo inicial de búsqueda después de N evaluaciones de la función $z(x)$, para los dos métodos explicados hasta el momento.

Número de evaluaciones de la función, N	Longitud del intervalo actual respecto el intervalo inicial I_N/I_0	
	Sección áurea	Bisección
2	0.6180	0.2500
3	0.3819	0.1250
4	0.2361	0.0625
5	0.1459	0.0313
6	0.0902	0.0156
7	0.0557	0.0078
8	0.0344	0.0039
9	0.0213	0.0020
10	0.0132	0.0010

Tabla 3.1 Ratio de convergencia para los métodos de reducción del intervalo (I_N/I_0 respecto N en cada algoritmo)

La tabla nos muestra que el ratio de convergencia del método de la bisección decrece más rápidamente que el de la sección áurea. Pero, en el método de la bisección la derivada de la función $z[x]$ debe ser evaluada en cada iteración. Este hecho no es sencillo en algunos casos, así que es aconsejable usar el método de la bisección en los casos donde evaluar y calcular la derivada de la función $z(x)$ no sea mucho más complicado que evaluar la función $z(x)$.

3.2. Métodos de ajuste de curvas

Cuando la función a minimizar, $z(x)$, no sólo es unimodal sino que posee curvas suaves, esta propiedad puede ser aprovechada por algoritmos más eficientes que los de las técnicas de reducción de intervalo. Los métodos de ajuste de curvas trabajan de forma iterativa mejorando el punto óptimo obtenido en la iteración anterior. Las características de la función a estudiar alrededor del punto óptimo actual son usadas para generar una aproximación a la función $z(x)$. Los métodos de este capítulo usarán parábolas. El nuevo punto óptimo que calcularemos será el punto mínimo de la función aproximada generada. Los métodos de este capítulo difieren de muchos otros métodos en la técnica usada para generar la función aproximada de la función objetivo.

Los métodos de ajuste de curvas poseen muchas de las propiedades de los métodos de minimización generales en los que se calculan una serie de puntos x^1, \dots, x^N, \dots que convergen hacia un

punto mínimo x^* . Cada uno de estos puntos es obtenido a partir de un algoritmo que trabaja con los puntos previos y este algoritmo termina una vez que se cumple un cierto criterio de convergencia. Los criterios de convergencia usados en los métodos de ajuste de curvas se basan en varias reglas. Estas reglas se basan en la precisión requerida para la solución o en términos de coste-efectividad. Por ejemplo, el algoritmo puede terminar cuando la contribución marginal de una iteración a la mejora de la solución es muy pequeña.

Es decir, si $z(x^{n-1}) - z(x^n) \leq k$

donde k es una predeterminada constante o tolerancia, el algoritmo termina. También se pueden usar constantes adimensionales basadas en los cambios relativos entre sucesivas iteraciones como test de convergencia.

Por ejemplo :
$$\frac{z(x^{n-1}) - z(x^n)}{z(x^{n-1})}$$

En algunos casos el algoritmo puede terminar basándose en los cambios del valor de la variable, por ejemplo $|x^n - x^{n-1}| \leq \hat{k}$

donde \hat{k} otra constante predeterminada. La cercanía de una determinada solución al punto mínimo también puede ser probado calculando la derivada de $z(x)$ en el punto x^n . Si el cálculo de esta derivada es próximo a 0, el algoritmo termina.

La selección del criterio de convergencia debe basarse en la función a minimizar, el algoritmo usado, y el contexto del problema a resolver. A continuación explicamos algunos de los algoritmos de ajuste de curvas más comunes. Tal como hemos dicho antes estos algoritmos sólo se diferencian por la técnica usada para calcular la aproximación a $z(x)$.

Método de Newton

El método de Newton se aproxima a $z(x)$ en cada iteración con una curva (parábola) al punto de estudio actual, x^n . La parábola, $\hat{z}(x)$, se puede calcular a partir de la ecuación

$$\hat{z}(x) = z(x^n) + \frac{dz(x^n)}{dx}(x - x^n) + \frac{1}{2} \frac{d^2z(x^n)}{dx^2}(x - x^n)^2$$

Ecuación 3.2

Esta curva, $\hat{z}(x)$, es una parábola que coincide con $z(x)$ en x^n hasta la segunda derivada.

El siguiente posible punto solución , x^{n+1} , está en el punto que minimiza $\hat{z}(x)$, es decir, donde $\frac{d\hat{z}(x)}{dx}=0$. Este nuevo punto lo obtenemos a partir de la ecuación X.3

$$x^{n+1} = x^n - \frac{\frac{dz(x^n)}{dx}}{\frac{d^2z(x^n)}{dx^2}}$$

Ecuación 3.3

La ecuación 3.3 especifica, el cálculo que realiza el algoritmo para obtener x^{n+1} a partir de x^n . La ecuación 3.2 no la usamos para realizar cálculos, sólo para explicar la base matemática del método.

El método de Newton es muy eficiente, pero requiere que la función $z(x)$ sea derivable hasta el segundo grado. También necesita que la primera y segunda derivada sean calculadas en cada iteración, en muchos casos esto puede suponer un gran coste computacional.

Método de la falsa posición

El método de la falsa posición es muy similar al método de Newton. Su única diferencia es, que en vez de usar la segunda derivada de la función a minimizar en el punto x^n ; usa las primeras derivadas de la función en los puntos x^n y x^{n-1} para calcular una aproximación a la segunda derivada. Exactamente la aproximación se calcula :

$$\frac{d^2z(x^n)}{dx^2} \approx \frac{\frac{dz(x^{n-1})}{dx} - \frac{dz(x^n)}{dx}}{x^{n-1} - x^n}$$

Ecuación 3.4

Esta expresión se sustituirá en la ecuación 3.3 para obtener el cálculo que realiza el algoritmo de la falsa posición para obtener x^{n+1} . Este método es muy útil en aquellos casos donde la segunda derivada no exista o sea muy difícil de calcular.

Método del ajuste cuadrático

Este método no necesita las derivadas de la función a minimizar. El método del ajuste cuadrático usa tres puntos, x_1 , x_2 , x_3 . En la enésima iteración , estos puntos serán una selección del conjunto de posibles puntos solución x^1 , ... , x^n . Al igual que en los métodos anteriores, el

nuevo punto solución x^{n+1} es obtenido buscando el mínimo de una curva aproximada. Este nuevo punto lo obtendremos a partir de la ecuación siguiente :

$$x^{n+1} = 1/2 * \left(\frac{(x_2^2 - x_3^2)z(x_1) + (x_3^2 - x_1^2)z(x_2) + (x_1^2 - x_2^2)z(x_3)}{(x_2 - x_3)z(x_1) + (x_3 - x_1)z(x_2) + (x_1 - x_2)z(x_3)} \right)$$

Ecuación 3.5

El nuevo grupo de tres puntos que se usará en la siguiente iteración incluirá x^{n+1} y dos de los tres puntos anteriores x_1 , x_2 , x_3 , aquellos dos que tengan el menor valor en la función $z(x)$. Estos tres valores pasarán a ser los puntos x_1 , x_2 y x_3 de la siguiente iteración.

Dado un determinado problema, la elección entre los diferentes métodos de ajuste de curvas debe basarse en la dificultad de calcular las derivadas. Si no son muy difíciles el mejor es el método de Newton, si son difíciles de calcular lo mejor es usar el método del ajuste cuadrático. En general , los métodos de ajuste de curvas son más rápidos que los de reducción de intervalo cuando se aplica sobre funciones con curvas suaves. Los métodos de ajuste de curvas también son útiles en aquellos casos donde no esté muy claro donde recae el punto mínimo.

Capítulo 4 :Equilibrio de usuarios con demanda variable

La formulación matemática del problema de equilibrio de usuarios realizada en el capítulo 1 asume que el número de usuarios que viajan entre cada pareja O-D es fijo y conocido. En realidad este flujo depende de la calidad del servicio de la red de transportes. Por ejemplo, si los tiempos de desplazamiento crecen, puede que algunos usuarios decidan usar el transporte público, cambiar sus horarios de viaje o simplemente no viajar.

Para tener este fenómeno en cuenta, el flujo q_{rs} , entre cada pareja origen-destino $r-s$ de la red será una función dependiente del tiempo de desplazamiento entre r y s .

Es decir,

$$q_{rs} = D_{rs}(u_{rs}) \quad \forall r, s$$

donde u_{rs} es el tiempo de desplazamiento mínimo entre los nodos centroides r y s y

$D_{rs}()$ es la función de demanda para la pareja $r-s$. Normalmente, la forma de la función de demanda es la misma para todas las parejas O-D. Pero suelen incluir una serie de parámetros para representar las características de los nodos. Por ejemplo, la función de demanda puede ser

$q_{rs} = A_r B_s f(u_{rs})$, donde A_r y B_s son parámetros asociados al nodo origen r y al nodo destino s , y $f(u_{rs})$ una función dependiente de u_{rs} .

Un punto importante a destacar es que la única variable de las funciones de demanda debe u_{rs} .

La función de demanda debe ser decreciente respecto a los tiempos de desplazamiento. Es decir si

u_{rs} crece, q_{rs} disminuye y viceversa. Además debe poseer una cota superior, por ejemplo, el número máximo de usuarios que viajan de r a s no puede ser superior a la población del nodo origen r .

El problema que trataremos en este capítulo es buscar los flujos de las aristas, los tiempos de desplazamiento en las aristas y el flujo de cada pareja O-D que satisfagan las condiciones del equilibrio de usuarios.

Las técnicas heurísticas explicadas en el capítulo 2 pueden ser fácilmente modificadas para buscar el patrón de flujos del equilibrio de usuarios para problemas con demanda variable. Pero ya hemos visto que estas técnicas no generan los resultados precisos que queremos. En este capítulo formularemos el problema como un problema matemático de minimización y estudiaremos cómo solucionarlo.

4.1. Formulación matemática del problema de asignación de tráfico con demanda variable

El problema matemático de minimización para la asignación de tráfico con demanda variable es :

$$\min z(x, q) = \sum_a \int_0^{x_a} t_a(w) dw - \sum_{rs} \int_0^{q_{rs}} D_{rs}^{-1}(w) dw$$

s.t.

$$\begin{aligned} x_a &= \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \quad \forall a \\ \sum_k f_k^{rs} &= q_{rs} \quad \forall r, s \\ f_k^{rs} &\geq 0 \quad \forall k, r, s \\ q_{rs} &\geq 0 \quad \forall r, s \end{aligned}$$

La función objetivo de este modelo es la suma de las integrales de las funciones de rendimiento asociadas a las aristas de la red menos el sumatorio de integrales de las funciones de demanda asociadas a cada pareja de O-D $r-s$.

La primera restricción dice que el flujo que transcurre por una arista es igual a la suma del flujo de todas las rutas que usan esa arista.

La segunda restricción dice que para cada pareja O-D $r-s$, la suma de flujos de todas las rutas que unen r y s ha de ser igual al número de usuarios que viajan desde r hasta s .

La tercera restricción dice que el flujo de todas las rutas de todas las parejas O-D ha de ser mayor o igual a 0.

La cuarta restricción dice que el número de usuarios que viajan de r a s debe ser un número positivo.

Respecto al problema matemático de demanda inelástica en éste hemos modificado la función objetivo y hemos añadido una nueva restricción para establecer un equilibrio entre las funciones de rendimiento y las funciones de demanda de la red.

Nuestro objetivo es maximizar la diferencia entre el sumatorio de las áreas de las funciones de demanda y el sumatorio de las áreas de las funciones de rendimiento.

Para ver esto mejor usaremos una red de una pareja O-D y una arista.

La función de rendimiento de la única arista será $t = 1 + x$ y la función de demanda será

$$x = 5 - t .$$

Considerando que la inversa de la función de demanda es $t = 5 - x$ la función objetivo será:

$$\min z(x) = \int_0^x (1 + w) dw - \int_0^x (5 - w) dw$$

A continuación mostramos la resolución de forma gráfica.

Podemos observar que si $x = x^*$, es decir el punto de intersección de las dos funciones, obtenemos la mayor área.

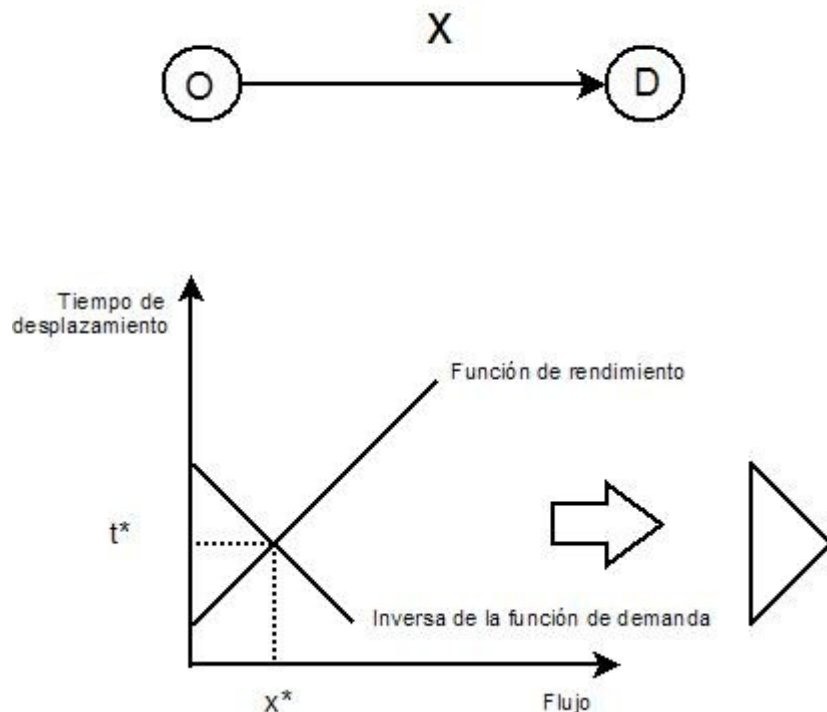


Imagen 4.1

El punto de equilibrio del sistema lo obtendremos con los valores x^* y t^* que son $x=2$ y $t=3$.

4.2. Resolución del problema de asignación de tráfico con demanda variable

En este capítulo queremos resolver el problema de asignación de tráfico con demanda variable.

La formulación del problema es :

$$\min z(x, q) = \sum_a \int_0^{x_a} t_a(w) dw - \sum_{rs} \int_0^{q_{rs}} D_{rs}^{-1}(w) dw$$

s.t.

$$\begin{aligned} f_k^{rs} &\geq 0 \quad \forall k, r, s \\ q_{rs} &= \sum_k f_k^{rs} \quad \forall r, s \\ q_{rs} &\leq \bar{q}_{rs} \quad \forall r, s \end{aligned}$$

Hemos añadido la última restricción a la formulación para establecer una cota superior para los flujos de cada pareja O-D. Si esta cota es elevada podemos ver que no modificará el problema, pero es necesaria añadirla porque es necesaria para las técnicas de resolución.

Las restricciones pueden ser formuladas usando sólo como variables el flujo de las rutas. Así que el flujo de las aristas pueden ser formulado como :

$$x_a = x_a(f) = \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \quad \forall a$$

y los flujos de las aristas como:

$$q_{rs} = q_{rs}(f^{rs}) = \sum_k f_k^{rs} \quad \forall r, s$$

Usando estas transformaciones, la formulación de nuestro problema queda así :

$$\min z(x, q) = \sum_a \int_0^{x_a} t_a(w) dw - \sum_{rs} \int_0^{q_{rs}} D_{rs}^{-1}(w) dw$$

s.t.

$$f_k^{rs} \geq 0 \quad \forall k, r, s$$

$$\sum_k f_k^{rs} \leq \bar{q}_{rs} \quad \forall r, s$$

La técnica usada para resolver este problema matemático será de nuevo el algoritmo de Frank-Wolfe.

Algoritmo de Frank-Wolfe adaptado para la búsqueda de equilibrios en redes de transporte con demanda variable

Paso 0: Inicialización.

Realizar una asignación “all-or-nothing” usando como tiempos $t_a^0 = t_a(0)$ y como

flujo para cada pareja O-D $r-s$ usar \bar{q}_{rs} .

Calcular los flujos de las aristas $\{x_a^1\}$.

Poner el contador de iteraciones a 1 $\{n := 1\}$.

Paso 1: Actualización de los costes.

Calcular: $t_a^n = t_a(x_a^n)$

Calcular: $D_{rs}^{-1}(q_{rs}^n) \quad \forall r, s$

Paso 2: Búsqueda de la dirección de descenso.

Para cada pareja O-D $r-s$ buscar $c_m^{rs^n}$. En nuestro caso m es la ruta más rápida en

la iteración n y $c_m^{rs^n}$ es el tiempo de desplazamiento de la ruta más rápida. Y a

continuación actualizar las variables auxiliares de flujo para las rutas g^{rs^n} así:

Si $c_m^{rs^n} < D_{rs}^{-1}(q_{rs}^n)$, entonces $g_m^{rs^n} = \overline{q_{rs}}$ y $g_m^{rs^n} = 0 \quad \forall k \neq m$.

Si $c_m^{rs^n} \geq D_{rs}^{-1}(q_{rs}^n)$, entonces $g_m^{rs^n} = 0 \quad \forall k$.

Una vez acabemos con todas las parejas O-D $r-s$, hemos de actualizar las variables auxiliares de flujo de las aristas y_a^n y las variables auxiliares de demanda v_{rs}^n del siguiente modo:

$$y_a^n = \sum_{rs} \sum_k g_k^{rs^n} \delta_{a,k}^{rs} \quad \forall a$$

$$v_{rs}^n = \sum_k g_k^{rs^n} \quad \forall r, s$$

Paso 3: Determinar la longitud de paso usando búsqueda lineal.

Buscar la λ_n que resuelva el problema de búsqueda lineal:

$$\min z(\lambda) = \sum_a \int_0^{x_a^n + \lambda(y_a^n - x_a^n)} t_a(w) dw - \sum_{rs} \int_0^{q_{rs}^n + \lambda(v_{rs}^n - q_{rs}^n)} D_{rs}^{-1}(w) dw$$

s.t.

$$0 \leq \lambda \leq 1$$

Paso 4: Actualizar la solución.

$$\text{Calcular } x_a^{n+1} = x_a^n + \lambda_n (y_a^n - x_a^n), \quad \forall a \in A$$

$$\text{Calcular } q_{rs}^{n+1} = q_{rs}^n + \lambda_n (v_{rs}^n - q_{rs}^n), \quad \forall r, s$$

Paso 5: Test de convergencia.

Aplicar un test de convergencia.

Si se satisface el test de convergencia, STOP. La solución actual x_a^{n+1} es el patrón de flujos del equilibrio de usuarios y q_{rs}^{n+1} la demanda de flujo.

En caso contrario, incrementar el contador de iteraciones $n := n + 1$ e ir al paso 1.

4.3. Resolución a través de la modificación de la representación de las redes de transporte

La adaptación del algoritmo de Frank-Wolfe para la resolución de problemas de asignación de tráfico con demanda variable es una tarea relativamente sencilla. Pero esta tarea puede resultar innecesaria; a través de la modificación de la representación de las redes de transporte podemos usar el algoritmo de Frank-Wolfe para sistemas con demanda inelástica para resolver nuestros casos con demanda elástica.

Estas técnicas se basan en la inserción de nodos y aristas virtuales. A continuación explicaremos una de estas técnicas.

La formulación con exceso de demanda

El problema de demanda variable puede ser resuelto reformulándolo como un problema de demanda inelástica que produce una nueva representación de la red. En esta representación, la variable e_{rs} representa el exceso de demanda, es decir, viajes no realizados entre el nodo origen r y el nodo destino s ($e_{rs} = \hat{q}_{rs} - q_{rs}$). Además definamos una nueva función complementaria a la inversa de la función de demanda, es decir $W_{rs}(e_{rs}) = D_{rs}^{-1}, \forall r, s$. Es más fácil ver la relación de estas dos imágenes en la imagen 4.2.

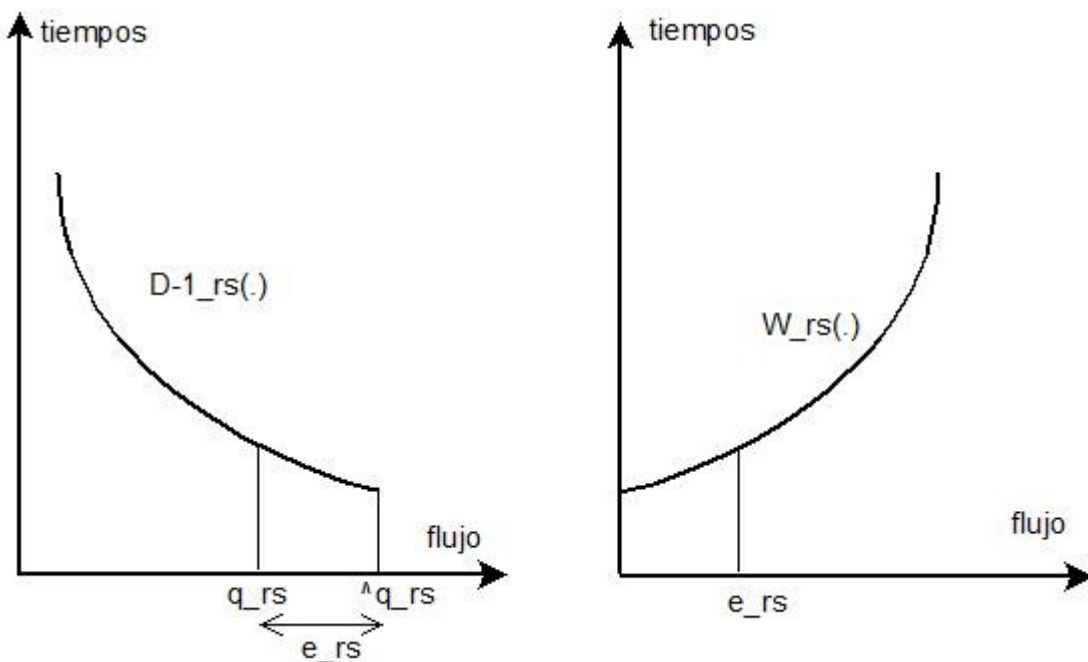


Imagen 4.2

Ahora consideremos la función objetivo de la formulación del problema de demanda variable :

$$\min z(x, q) = \sum_a \int_0^{x_a} t_a(w) dw - \sum_{rs} \int_0^{q_{rs}} D_{rs}^{-1}(w) dw$$

Cada término del segundo sumatorio puede ser descompuesto en dos integrales :

$$\int_0^{q_{rs}} D_{rs}^{-1}(w) dw = \int_0^{\hat{q}_{rs}} D_{rs}^{-1}(w) dw - \int_{q_{rs}}^{\hat{q}_{rs}} D_{rs}^{-1}(w) dw$$

La primera integral de la descomposición es una constante , esto quiere decir que la podemos retirar de la función objetivo porque no va a afectar el resultado de la minimización. La segunda integral de la descomposición puede ser expresada en término de exceso de demanda con un cambio de variables, $e_{rs} = \hat{q}_{rs} - q_{rs}$. La nueva variable de integración será $v = \hat{q}_{rs} - w$. los nuevos límites de la integral serán ($\hat{q}_{rs} - q_{rs}$) y cero . Con todo esto se obtiene:

$$\begin{aligned} - \int_{q_{rs}}^{\hat{q}_{rs}} D_{rs}^{-1}(w) dw &= - \int_{q_{rs} - \hat{q}_{rs}}^0 D_{rs}^{-1}(\hat{q}_{rs} - v)(-dv) \\ &= - \int_0^{e_{rs}} W_{rs}(v) dv \end{aligned}$$

Usando esto que acabamos de explicar, la función objetivo de la formulación del problema de demanda variable puede ser reescrita así :

$$\min z(x, e) = \sum_a \int_0^{x_a} t_a(w) dw - \sum_{rs} \int_0^{e_{rs}} W_{rs}(v) dv$$

donde $e = (\dots, e_{rs}, \dots)$ es el vector de flujos de exceso de demanda. Además debido a estos cambios las restricciones del problema de minimización quedan reescritas de la siguiente manera :

$$\begin{aligned} \sum_k f_k^{rs} + e_{rs} &= \hat{q}_{rs} \quad \forall r, s \\ f_k^{rs} &\geq 0 \quad \forall k, r, s \\ e^{rs} &\geq 0 \quad \forall r, s \end{aligned}$$

Podemos ver que los cambios han sido mínimos, sólo la introducción del vector de flujos de exceso de demanda. Con un vistazo a la función $W_{rs}(e_{rs})$ nos podemos dar cuenta que tiene el mismo comportamiento que las funciones de rendimiento asociadas a las aristas. Así que podemos añadir a la red una nueva arista que lleve el flujo e_{rs} . Y observando la primera restricción se deduce que esta nueva arista transportará el exceso de demanda entre cada pareja de nodos origen destino. La red resultante quedará tal como se ve en la imagen 5.3.

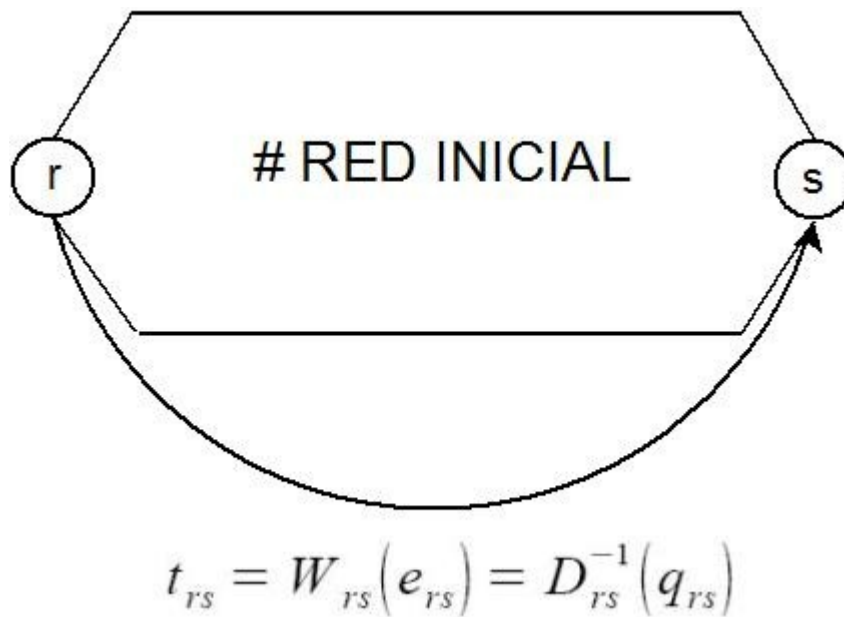


Imagen 5.3

Cabe recordar que cada una de estas nuevas aristas representa un flujo variable, es decir, hay que añadir una nueva arista por cada pareja O-D con demanda variable. Con todo esto podemos decir que nuestro nuevo problema puede ser resuelto con el algoritmo de Frank-Wolfe para problemas con demanda inelástica. En el punto de equilibrio, $W_{rs}(e_{rs}) = u_{rs}$, y como por definición

$W_{rs}(e_{rs}) = D_{rs}^{-1}(q_{rs})$, se cumplen las condiciones de equilibrio para los problemas de demanda elástica.

Capítulo 5: Especificación de la aplicación

5.1. Clase parametros

El propósito de esta clase es contener las constantes de la aplicación, tanto las internas de protección como las que leer del fichero de parámetros ParametrosSistema.txt

A continuación se detallan las operaciones disponibles:

Operación: `__init__()`

Clase retorno:

Semántica: Función creadora de un objeto de la clase parámetros

Precondiciones:

Postcondiciones: Creación de un objeto de la clase parámetros.

Inicialización de las variables y definición de las constantes.

Operación: `cargar()`

Clase retorno:

Semántica: Lectura de parámetros del sistema del fichero de texto ParametrosSistema.txt

Precondiciones: Debe existir el fichero ParametrosSistema.txt

Postcondiciones: Asignación de valores y rutas a las variables del objeto. En caso de errores el sistema usará los valores por defecto.

5.2. Clase dicParFunciones

El propósito de esta clase es ser un diccionario con los parámetros que se necesitan para el trabajo con juegos de prueba escritos en ficheros.

Operación: `__init__()`

Clase retorno:

Semántica: Función creadora de un objeto de la clase dicParFunciones

Precondiciones:

Postcondiciones: Creación de un objeto de la clase dicParFunciones

Inicialización del diccionario de parámetros de las funciones.

5.3. Clase Igiros

Esta clase está hecha para contener los giros prohibidos o penalizados de las redes de transporte que se usen.

Operación: `__init__()`

Clase retorno:

Semántica: Función creadora de un objeto de la clase Igiros.

Precondiciones:

Postcondiciones: Creación de un objeto de la clase Igiros.

Inicialización del diccionario de giros.

Operación: `setGiro(i:int, j:int, k:int, IDgiro:int)`

Clase retorno:

Semántica: Introducción del código de penalización o prohibición del giro en el diccionario bajo la entrada `i,j,k`.

Precondiciones:

Postcondiciones: Si existe en el diccionario la entrada `i,j,k` se actualiza el valor asociado a `IDgiro`.

Si no existe la entrada en el diccionario, se crea y se le asocia `Idgiro`.

Operación: `getGiro(i:int, j:int, k:int)`

Clase retorno: `int`

Semántica: Buscamos en el diccionario la entrada `i,j,k`.

Precondiciones:

Postcondiciones: Si no existe la entrada `i,j,k` en el diccionario, devolvemos `None`.

Si existe la entrada `i,j,k`; se devuelve el código de penalización o prohibición asociado a la entrada.

5.4. Clase Iasignacion

El propósito de esta clase es representar las parejas de nodos O-D y el flujo que viaja entre ellas, en la representación de un caso de estudio del software.

Operación: `__init__()`

Clase retorno:

Semántica: Función creadora de un objeto de la clase `lasignacion`.

Precondiciones:

Postcondiciones: Creación de un objeto de la clase `lasignacion`.

Inicialización del diccionario de asignaciones.

Operación: `setAsignacion(origen:int, destino:int, cantidad:int)`

Clase retorno:

Semántica: Introducción en el diccionario del número de usuarios que quieren ir del nodo origen al nodo destino, usando como entrada del diccionario la pareja de variables de entrada origen, destino.

Precondiciones:

Postcondiciones: Si existe en el diccionario la entrada origen, destino; se actualiza el valor asociado a cantidad.

Si no existe la entrada en el diccionario, se crea y se le asocia cantidad.

Operación: `getAsignacion(origen:int, destino:int)`

Clase retorno: `int`

Semántica: Buscamos en el diccionario la entrada origen, destino.

Precondiciones:

Postcondiciones: Si no existe la entrada origen, destino; en el diccionario devolvemos 0.

Si existe la entrada origen, destino; se devuelve el valor asociado a la entrada.

5.5. Clase *FrankWolfe*

Esta clase es la más importante de la aplicación. Su objetivo es la aplicación del algoritmo de Frank-Wolfe sobre un caso de estudio y la generación de la información necesaria para comprender la solución.

Operación: `FrankWolfe(G: XDiGraph, l: lasignacion, P: dicParFunciones, timeTotal: time)`

Clase retorno:

Semántica: Ejecución del algoritmo Frank-Wolfe usando como red G, listado de asignaciones l y parámetros P.

Precondiciones:

Postcondiciones: Generación de tres ficheros, dos ficheros de texto resultado.txt, resultadoIteracion.txt, y un fichero de imagen imagenResultado.png, con los resultados del algoritmo.

5.6.Fichero GoldenSection

Este fichero está pensado para la búsqueda de mínimos en funciones univariantes usando la técnica de la sección áurea.

Operación: golden(listado: arista[], P: dicParFunciones)

Clase retorno: float

Semántica: Resolución del problema usando la técnica de reducción de intervalos de la sección áurea . Usando como sumatorio el parámetro de entrada listado y usando como margen de error el valor toleranciaSolver del parámetro P .

Precondiciones: Los objetos arista deben haber pasado las fases previas del algoritmo de Frank-Wolfe

Postcondiciones: se devuelve un float que es el valor de la variable λ que resuelve el problema

$$\begin{aligned} \min \quad & \sum_a \int_0^{x_a^n + \lambda(y_a^n - x_a^n)} t_a(w) dw \\ \text{s.t.} \quad & 0 \leq \lambda \leq 1 \end{aligned}$$

Operación: calcular(alfa: float, listado: arista[])

Clase retorno: float

Semántica: Cálculo del sumatorio de integrales, usando el parámetro de entrada alfa como λ y como sumatorio de integrales el parámetro de entrada listado.

Precondiciones: Los objetos arista deben haber pasado las fases previas del algoritmo de Frank-Wolfe

Postcondiciones: se devuelve un float que es el resultado de

$$\sum_a \int_0^{x_a^n + \lambda(y_a^n - x_a^n)} t_a(w) dw$$

5.7. Clase resultadosIteracion

Clase auxiliar de FrankWolfe cuyo propósito es imprimir en un fichero la información útil de cada iteración del algoritmo de Frank-Wolfe,

Operación: `__init__()`

Clase retorno:

Semántica : Función creadora de un objeto de la clase resultadosIteracion

Precondiciones:

Postcondiciones: Creación de un objeto de la clase resultadosIteración.

Inicialización de las variables.

Operación: `imprimir()`

Clase retorno:

Semántica: Impresión de los datos contenidos en la clase en el fichero

Precondiciones:

Postcondiciones: Si el fichero existe, escritura de los datos en el fichero.

5.8. Clase caminos

Clase pensada para realizar asignaciones de flujo en una red basándonos en las rutas menos costosas.

Operación: `dijkstra (G:XDigraph, source:int, target:int=None)`

Clase retorno: (`diccionario, diccionario`)

Semántica: Resolución de un problema de caminos mínimos sobre la red G, tomando como nodo origen de los caminos el parámetro `source` y en caso de existir como nodo final del camino el parámetro `target`, para ello se usará el algoritmo Dijkstra.

Precondiciones: G ha de ser un grafo conexo y `source` identificar un nodo del grafo G

Postcondiciones: La tupla de salida contendrá en el primer parámetro los costes mínimo de ir del nodo `source` al resto, el segundo parámetro contendrá las rutas de coste mínimo entre el nodo `source` y el resto de nodos.

Operación: dEssopoPape(G:XDigraph, source:int, target:int=None)

Clase retorno: (diccionario, diccionario)

Semántica: Resolución de un problema de caminos mínimos sobre la red G, tomando como nodo origen de los caminos el parámetro source y en caso de existir como nodo final del camino el parámetro target, para ello se usará el algoritmo dEssopoPape.

Precondiciones: G ha de ser un grafo conexo y source identificar un nodo del grafo G

Postcondiciones: La tupla de salida contendrá en el primer parámetro los costes mínimo de ir del nodo source al resto, el segundo parámetro contendrá las rutas de coste mínimo entre el nodo source y el resto de nodos.

Operación: asignarX(G:XDigraph, L:lasignacion)

Clase retorno:

Semántica: Asignación inicial de los flujos indicados por L a la red G usando como criterio de asignación las rutas de coste mínimo. Esta asignación se hará a la variable auxiliar de las aristas x.

Precondiciones: G ha de ser un grafo conexo

Postcondiciones: A las aristas de la red G se les ha asignado los flujos indicados por L.

Operación: asignarY(G:XDigraph, L:lasignacion)

Clase retorno:

Semántica: Asignación de los flujos indicados por L a la red G usando como criterio de asignación las rutas de coste mínimo. Esta asignación se hará a la variable auxiliar de las aristas y.

Precondiciones: G ha de ser un grafo conexo

Postcondiciones: A las aristas de la red G se les ha asignado los flujos indicados por L.

5.9. Clase criterioBLB

Clase auxiliar de FrankWolfe que implementa el criterio de parada del algoritmo de Frank-Wolfe.

Operación: __init__(margen: float)

Clase retorno:

Semántica: Función creadora de un objeto de la clase criterioBLB.

Inicialización de la variable epsilon que es la tasa de error aceptable del criterio de parada.

Inicialización de la variable BLB con el valor menos infinito.

Precondiciones:

Postcondiciones: Creación de un objeto de la clase criterioBLB.

Asignación de la variable de entrada margen a la variable epsilon del objeto.

Operación: convergencia(costeFuncion: float, costeBLB: float)

Clase retorno: boolean

Semántica: Actualización del valor de la variable BLB del objeto criterio BLB.

Evaluación del criterio de parada usando la técnica BLB.

Precondiciones:

Postcondiciones: Si costeBLB es menor a la variable BLB del objeto, entonces BLB = coste BLB.

Si $\frac{\text{costeFuncion} - \text{BLB}}{\text{costeFuncion}} \leq \text{epsilon}$ devolvemos True.

En caso contrario devolvemos False.

5.10.Clase arista

Clase pensada para representar una arista de la red de un caso de estudio.

Operación: __init__(a: float=1.0, b: float=1.0, c: float=1.0, d: float=1.0, xn: float, yn: float)

Clase retorno:

Semántica: Función creadora de un objeto de la clase arista, este objeto contiene un objeto de la clase funcion.

Precondiciones:

Postcondiciones: Creación de un objeto de la clase arista.

Creación de un objeto de la clase funcion usando las variables a, b, c , d.

Inicialización de las variables xn, yn, tn, artificial.

Operación: actualizarx (alfa:float)

Clase retorno:

Semántica: Actualización del valor de la variable xn.

Precondiciones:

Postcondiciones: Actualización del valor de la variable x_n del objeto arista siguiendo la fórmula

$$x_n + \text{alfa}(y_n - x_n) \quad \text{siendo alfa el parámetro de entrada.}$$

Operación: `actualizart()`

Clase retorno:

Semántica: Actualización del valor de la variable t_n del objeto arista asignándole el resultado de la llamada a la función evaluar del objeto funcion asociado a la variable f .

Precondiciones:

Postcondiciones: Actualización del valor de la variable t_n del objeto arista asignándole el valor de la

llamada $f.\text{evaluar}(x_n)$, siendo la variable f un objeto funcion.

Operación: `resetY()`

Clase retorno:

Semántica: Se resetea el valor de la variable y_n .

Precondiciones:

Postcondiciones: Asignación del valor 0 a la variable y_n del objeto funcion.

Operación: `coste()`

Clase retorno: float

Semántica: Cálculo de la evaluación de la integral de la función asociada a la variable f respecto la variable x_n .

Precondiciones:

Postcondiciones: Se devuelve el resultado de la llamada $f.\text{evaluarIntegralX}(x_n)$, siendo la variable f un objeto de tipo funcion.

Operación: `calcBLB()`

Clase retorno: float

Semántica: Cálculo de la expresión $\int^{x_n} f(x) dx + f(x)(y_n - x_n)$

Precondiciones:

Postcondiciones: Se devuelve el resultado de la expresión:

$$f.\text{evaluarIntegralX}(x_n) + f.\text{evaluar}(x_n) * (y_n - x_n)$$

Operación: imprimir(file:Fichero)

Clase retorno:

Semántica: Impresión en el fichero file de las propiedades del objeto arista.

Precondiciones:

Postcondiciones: Impresión en el fichero file las variables del objeto arista xn, tn.

5.11. Clase aristaElastica

Clase pensada para representar una arista artificial de un caso de estudio de demanda variable. Estas aristas surgen cuando se aplica la técnica de exceso de demanda sobre estas redes para pasar de un problema de demanda elástica a uno de demanda inelástica.

Operación: `__init__`(BetaA: float, BetaB: float, Thetap: float, Thetaq: float , twB: float, gw: float, xn=0, yn=0)

Clase retorno:

Semántica: Función creadora de un objeto de la clase aristaElastica, este objeto contiene un objeto de la clase funcionElastica.

Precondiciones:

Postcondiciones: Creación de un objeto de la clase aristaElastica.

Creación de un objeto de la clase funcionElastica usando las variables BetaA, BetaB, Thetap, Thetaq, twB, gw.

Inicialización de las variables xn, yn, tn , artificial.

Operación: actualizarx (alfa:float)

Clase retorno:

Semántica: Actualización del valor de la variable xn.

Precondiciones:

Postcondiciones: Actualización del valor de la variable xn del objeto arista siguiendo la fórmula

$$xn + \text{alfa} (yn - xn) \quad \text{siendo alfa el parámetro de entrada.}$$

Operación: actualizart()

Clase retorno:

Semántica: Actualización del valor de la variable tn del objeto aristaElastica asignándole el resultado de la llamada a la función evaluar del objeto funcionElastica asociado a la variable f.

Precondiciones:

Postcondiciones: Actualización del valor de la variable tn del objeto arista asignándole el valor de la llamada $f.evaluar(xn)$, siendo la variable f un objeto funcionElastica.

Operación: resetY()

Clase retorno:

Semántica: Se resetea el valor de la variable yn.

Precondiciones:

Postcondiciones: Asignación del valor 0 a la variable yn del objeto funcionElastica.

Operación: coste()

Clase retorno: float

Semántica: Cálculo de la evaluación de la integral de la función asociada a la variable f respecto la variable xn.

Precondiciones:

Postcondiciones: Se devuelve el resultado de la llamada $f.evaluarIntegralX(xn)$, siendo la variable f un objeto de tipo funcionElastica.

Operación: calcBLB()

Clase retorno: float

Semántica: Cálculo de la expresión $\int^{xn} f(x) dx + f(x)(yn - xn)$

Precondiciones:

Postcondiciones: Se devuelve el resultado de la expresión:

$$f.evaluarIntegralX(xn) + f.evaluar(xn) * (yn - xn)$$

Operación: imprimir(file:Fichero)

Clase retorno:

Semántica: Impresión en el fichero file de las propiedades del objeto aristaElastica.

Precondiciones:

Postcondiciones: Impresión en el fichero file las variables del objeto aristaElastica xn, tn.

5.12. Clase Funcion

Clase auxiliar de arista cuyo propósito es la implementación de la función de rendimiento asociada a una arista.

Operación: `__init__(a: float=1.0, b: float=1.0, c: float=1.0, d: float=1.0)`

Clase retorno:

Semántica: Función creadora de un objeto de la clase Funcion

Precondiciones:

Postcondiciones: Creación de un objeto de la clase Funcion.

Asignación de los valores de entrada a las variables del objeto.

Cálculo de las constantes.

Operación: `evaluar(x: float)`

Clase retorno: float

Semántica: Calcular el valor de la función usando como variable el parámetro de entrada.

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular $a \left(1 + b \left(\frac{x}{c} \right)^d \right)$

Operación: `evaluarIntegralX(x:float)`

Clase retorno:float

Semántica:

Precondiciones: Calcular el valor de la integral de la función usando como variable el parámetro de entrada.

Postcondiciones: Se devuelve el resultado de calcular $a * x + \left(\frac{a * b}{(d + 1) * c^d} \right) x^{d+1}$

Operación: `evaluarIntegral(xn: float, yn:float, alfa:float)`

Clase retorno: float

Semántica: Calcular el valor de la integral de la función cambiando la variable x por la expresión

$x_n + \alpha(y_n - x_n)$ calculada a partir de las variables de entrada.

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular

$$a * (x_n + \alpha(y_n - x_n)) + \left(\frac{a * b}{(d + 1) * c^d} \right) (x_n + \alpha(y_n - x_n))^{d+1}$$

5.13. Clase funcionElastica

Clase auxiliar de aristaElastica cuyo propósito es la implementación de la función de rendimiento asociada a una aristaElastica a partir de la función de demanda elástica.

Operación: __init__(BetaA: float = 1.0, BetaB: float = 1.0, Thetap: float = 1.0, Thetaq: float = 1.0, twB: float = 1.0, gw: float = 1.0)

Clase retorno:

Semántica: Función creadora de un objeto de la clase FuncionElastica

Precondiciones:

Postcondiciones: Creación de un objeto de la clase FuncionElastica.

Asignación de los valores de entrada a las variables del objeto.

Cálculo de las constantes.

Operación: evaluar(x: float)

Clase retorno: float

Semántica: Calcular el valor de la función usando como variable el parámetro de entrada.

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular

$$\frac{1}{\beta_A} \left(\ln \left(\frac{g_w}{\hat{g}_w - x} \right) - 1 \right) + \beta_B t_w^B - \theta_p - \theta_q$$

Operación: evaluarIntegralX(x:float)

Clase retorno:float

Semántica: Calcular el valor de la integral de la función usando como variable el parámetro de entrada.

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular

$$\begin{aligned} & \frac{1}{\beta_A} (x + g_w - \hat{g}_w) \ln(x + g_w - \hat{g}_w) \\ & + \frac{1}{\beta_A} (\hat{g}_w - x) \ln(\hat{g}_w - x) - \frac{1}{\beta_A} (g_w - \hat{g}_w) \ln(g_w - \hat{g}_w) \\ & + \left(\frac{\beta_b * t_w^B - \theta_p - \theta_q}{\beta_a} \right) * x \end{aligned}$$

Operación: evaluarIntegral (xn: float, yn:float, alfa:float)

Clase retorno: float

Semántica: Calcular el valor de la integral de la función cambiando la variable x por la expresión

$$x_n + \alpha(y_n - x_n) \quad \text{calculada a partir de las variables de entrada.}$$

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular

$$\begin{aligned} & \frac{1}{\beta_A} (x_n + \alpha(y_n - x_n) + g_w - \hat{g}_w) \ln(x_n + \alpha(y_n - x_n) + g_w - \hat{g}_w) \\ & + \frac{1}{\beta_A} (\hat{g}_w - x_n + \alpha(y_n - x_n)) \ln(\hat{g}_w - x_n + \alpha(y_n - x_n)) - \frac{1}{\beta_A} (g_w - \hat{g}_w) \ln(g_w - \hat{g}_w) \\ & + \left(\frac{\beta_b * t_w^B - \theta_p - \theta_q}{\beta_a} \right) * x_n + \alpha(y_n - x_n) \end{aligned}$$

Operación: demanda(twA):

Clase retorno:

Semántica: Calcular el valor de la función de demanda usando como variable el parámetro de entrada.

Precondiciones:

Postcondiciones: Se devuelve el resultado de calcular:

$$\frac{g_w}{1 + e^{\beta_A t_w^A - \beta_B t_w^B + \theta_p + \theta_q}}$$

5.14.Fichero casoelastico

Fichero que contiene el algoritmo para transformar un problema de demanda elástica en uno de demanda inelástica.

Operación: caso2(G: XDiGraph)

Clase retorno:

Semántica: Función para generar la red de aristas y nodos correspondiente con el juego de pruebas caso2

Precondiciones:

Postcondiciones: G será un red que representa el juego de pruebas caso2.

Operación: asignacionesElasticas(G: XDiGraph, L: lasignacion):

Clase retorno:

Semántica: Función cuyo objetivo es la modificación de G y L para adaptar el sistema de demanda elástica que representan en uno de demanda inelástica.

Precondiciones:

Postcondiciones: G y L cumplen todas las propiedades de un problema de demanda inelástica equivalente al problema de demanda elástica inicial.

Capítulo 6: Características de Python

6.1. ¿Qué es Python?

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.

Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.

Lenguaje interpretado o de script

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilados).

La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo los lenguajes interpretados son más flexibles y más portables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semiinterpretado. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudocódigo máquina intermedio llamado bytecode la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones.

Tipado dinámico

La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable,

sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

Fuertemente tipado

No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario

convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o string) no podremos tratarla como un número (sumar la cadena “9” y el número 8). En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores.

Multiplataforma

El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.

Orientado a objetos

La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

Python también permite la programación imperativa, programación funcional, programación estructurada y programación orientada a aspectos.

6.2.¿Por qué Python?

Python es un lenguaje que todo el mundo debería conocer. Su sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de librerías disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar una aplicación en Python sea sencillo, muy rápido y, lo que es más importante, divertido.

La sintaxis de Python es tan sencilla y cercana al lenguaje natural que

los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar.

Python no es adecuado sin embargo para la programación de bajo nivel o para aplicaciones en las que el rendimiento sea crítico.

Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento

mayor de los programas disponibles.

6.3. Instalación de Python

Existen varias implementaciones distintas de Python: CPython, Jython, IronPython, PyPy, etc.

CPython es la más utilizada, la más rápida y la más madura. Cuando la gente habla de Python normalmente se refiere a esta implementación. En este caso tanto el intérprete como los módulos están escritos en C.

Jython es la implementación en Java de Python, mientras que IronPython es su contrapartida en C# (.NET). Su interés estriba en que utilizando estas implementaciones se pueden utilizar todas las librerías disponibles para los programadores de Java y .NET.

PyPy, por último, como habréis adivinado por el nombre, se trata de una implementación en Python de Python.

CPython está instalado por defecto en la mayor parte de las distribuciones Linux y en las últimas versiones de Mac OS. Para comprobar si está instalado abre una terminal y escribe python. Si está instalado se iniciará la consola interactiva de Python y obtendremos parecido a lo siguiente:

La primera línea nos indica la versión de Python que tenemos instalada. Al final podemos ver el prompt (>>>) que nos indica que el intérprete está esperando código del usuario. Podemos salir escribiendo exit(), o pulsando Control + D.

Si no te muestra algo parecido no te preocupes, instalar Python es muy sencillo. Puedes descargar la versión correspondiente a tu sistema operativo desde la web de Python, en <http://www.python.org/download/>. Existen instaladores para Windows y Mac OS. Si utilizas Linux es muy probable que puedas instalarlo usando la herramienta de gestión de paquetes de tu distribución, aunque también podemos descargar la aplicación compilada desde la web de Python.

Módulos y paquetes

Si los módulos sirven para organizar el código, los paquetes sirven para organizar los módulos. Los paquetes son tipos especiales de módulos (ambos son de tipo module) que permiten agrupar módulos relacionados. Mientras los módulos se corresponden a nivel físico con los archivos, los paquetes se representan mediante directorios.

En una aplicación cualquiera podríamos tener, por ejemplo, un paquete iu para la interfaz o un paquete bbdd para la persistencia a base de datos.

Para hacer que Python trate a un directorio como un paquete es necesario crear un archivo `__init__.py` en dicha carpeta. En este archivo se pueden definir elementos que pertenezcan a dicho paquete, como una constante `DRIVER` para el paquete `bbdd`, aunque habitualmente se tratará de un archivo vacío. Para hacer que un cierto módulo se encuentre dentro de un paquete, basta con copiar el archivo que define el módulo al directorio del paquete.

Como los módulos, para importar paquetes también se utiliza `import` y `from-import` y el caracter `.` para separar paquetes, subpaquetes y módulos.

```
import paq.subpaq.modulo
```

```
paq.subpaq.modulo.func()
```

Para encontrar algún módulo o paquete que cubra una cierta necesidad, puedes consultar la lista de PyPI (Python Package Index) en <http://pypi.python.org/>, que cuenta a la hora de escribir estas líneas, con más de 5200 paquetes distintos.

Capítulo 7: Formato de la información

En este capítulo vamos a tratar el formato de la información, cómo ha de ser la información de entrada y cómo se sacará la información.

7.1. Fichero de configuración del sistema

Junto al código hay un fichero de texto llamado ParametrosSistema.TXT, este fichero es el que se usa para declarar una serie de variables globales y las rutas de los ficheros de entrada y salida. Si usted pone el símbolo "#" se entenderá que esa línea es un comentario y el programa la ignorará. Para que el sistema lea una línea esta debe comenzar por alguna de las seis palabras claves, cada una de ellas corresponde con una configuración :

rutaJuegosPrueba : con este parámetro le estamos diciendo al software que vamos a introducir la ruta a los juegos de prueba que están fichero de texto.

Por ejemplo : rutaJuegosPrueba c:/jp2/

rutaFicherosSalida : con este parámetro le estamos diciendo al software que vamos a introducir el nombre o la ruta de los ficheros de salida.

Por ejemplo : rutaFicherosSalida PFC

Esto hará que los ficheros de salida se llamen PFCresultados.txt,

PFCresultadosIteracion.txt y PFCImagenResultado.PNG, además se guardarán en la carpeta por defecto que es donde está el código fuente.

toleranciaParada : con este parámetro le estamos diciendo al software que vamos a introducir la precisión que le vamos a exigir al criterio de parada del algoritmo FrankWolfe.

Por ejemplo : toleranciaParada 0.002

toleranciaSolver : con este parámetro le estamos diciendo al software que vamos a introducir la precisión requerida por el algoritmo de minimización univariante.

Por ejemplo : toleranciaSolver 0.001

iteracionesMax : con este parámetro le estamos diciendo al software que vamos a introducir el número máximo de iteraciones que ejecutará el algoritmo de FrankWolfe.

Por ejemplo : iteracionesMax 50

salidaPantalla : con este parámetro le estamos diciendo al software si queremos que vaya mostrando información por pantalla durante la ejecución.

Por ejemplo : salidaPantalla True

7.2. Parámetros de las funciones de rendimiento

La función de rendimiento más frecuente es la siguiente:

$$f(v) = t_0 * \left(1 + \alpha * \left(\frac{v}{c} \right)^\beta \right)$$

Donde:

- **v** : Flujo asignado a la arista.
- **f(v)** : Tiempo que tardará cada usuario en recorrer la arista.
- **t₀** : Tiempo que tardaría cada usuario en recorrer la arista en condiciones ideales. Si es necesario, se puede calcular a partir de unos datos básicos del tramo de vía que representa la arista, dividiendo la longitud del tramo de vía entre la velocidad máxima permitida en ella.
- **c** : Capacidad máxima del tramo de vía.
- **α y β** : parámetros usados para representar el comportamiento de la vía en condiciones de saturación.

Esto ya lo hemos explicado en el capítulo 1, pero cuando se trata de funciones de rendimiento de juegos de pruebas en ficheros de texto los parámetros t_0 , α , β y c es diferente.

Los parámetros α y β estarán en diccionarios de la aplicación cuya clave de entrada será el campo 7 de la definición de una arista del fichero de red base, que corresponde con número de función de coste.

El parámetro c proviene de la expresión $c = \gamma * n * h$. Donde γ será una variable global del sistema para evitar valores fuera de rango, por defecto vale 0. n será el campo 6 de la definición de una arista del fichero de red base, que corresponde con el número de carriles. h será el campo 9 de la definición de una arista del fichero de red base, que corresponde con campo libre 2.

El parámetro t_0 proviene de la expresión $t_0 = \frac{long}{\hat{vel} * 60}$. Donde long será el campo 3 de la definición de una arista del fichero de red base, que corresponde con la longitud en Km. \hat{vel}

estará en un diccionario de la aplicación cuya clave de entrada será el campo 7 de la definición de una arista del fichero de red base, que corresponde con número de función de coste.

7.3. Parámetros de las funciones de demanda variable

La función de demanda con la que trabajaremos será la siguiente :

$$g_w^A(t_w^A) = \left(\frac{g_w}{1 + e^{\beta_A t_w^A - \beta_B t_w^B + \theta_p + \theta_q}} \right)$$

Donde :

$$g_w = g_w^A + g_w^B \quad \forall w = (p, q) \in W$$

Es decir, para toda pareja de nodos O-D , existe un número total de usuarios que querrán ir entre esos puntos. Estos usuarios irán en transporte privado (A) o en transporte público (B).

número total de viajes = número viajes transporte privado + número viajes transporte público

θ_p, θ_q : corresponden con el coste de aparcamiento del vehículo privado en el punto de origen p y el punto de destino q.

t_w^A : es el tiempo de desplazamiento entre la pareja de nodos O-D en transporte privado.

t_w^B : es el tiempo de desplazamiento entre la pareja de nodos O-D en transporte público.

β_A : es el coste por unidad de tiempo al desplazarnos en transporte privado.

β_B : es el coste por unidad de tiempo al desplazarnos en transporte público.

Pero en el proceso de modificación de la red de demanda variable a un modelo de exceso de

demanda se requiere la inversa de la función, es decir, $(g_w^A(t_w^A))^{-1} = t_w^A(g_w^A)$.

Obteniendo esto:

$$t_w^A(g_w^A) = \frac{1}{\beta_A} \left(\ln \left(\frac{g_w}{g_w^A} - 1 \right) + \beta_B t_w^B - \theta_p - \theta_q \right)$$

Y tal como se definió en el problema no usamos g_w^A , si no que debemos usar

$$g_w^A + f_w^A = \hat{g}_w \implies g_w^A = \hat{g}_w - f_w^A$$

Donde:

f_w^A : Exceso de demanda privado para la la pareja w

\hat{g}_w : Cota máxima de demanda

Quedando la función de la siguiente manera :

$$t_w^A(f_w^A) = \frac{1}{\beta_A} \left(\ln \left(\frac{g_w}{\hat{g}_w - f_w^A} - 1 \right) + \beta_B t_w^B - \theta_p - \theta_q \right)$$

que ya puede ser usado como una función de demanda más.

Capítulo 8 : Juegos de prueba

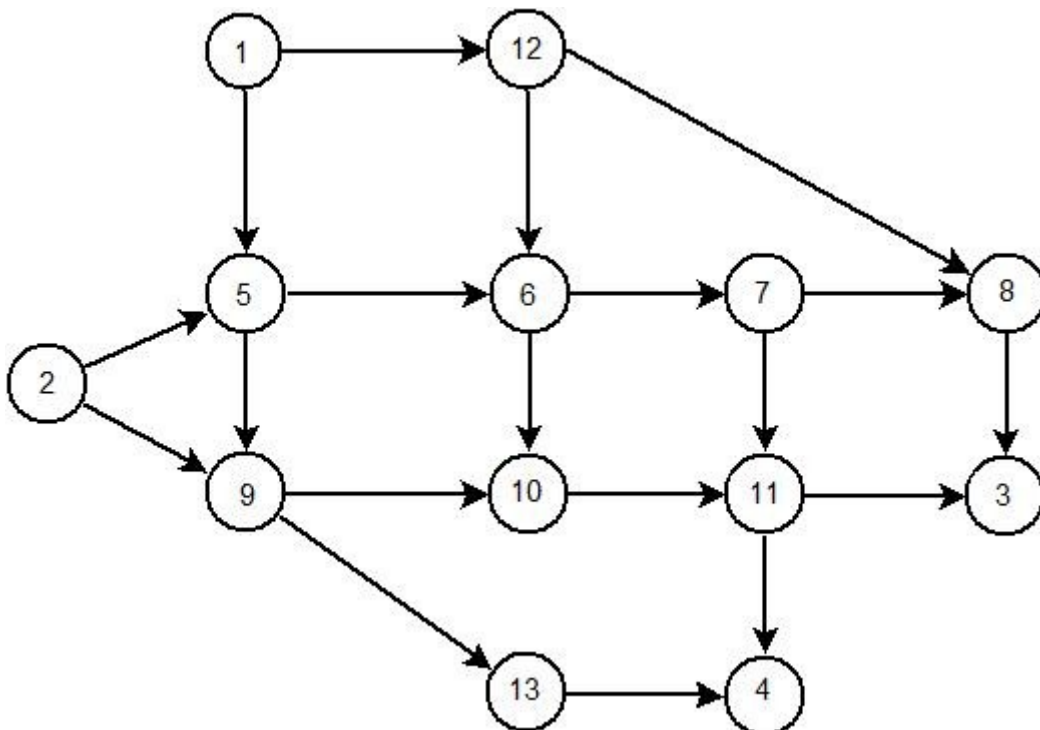
En este capítulo pretendemos mostrar el funcionamiento de la aplicación sobre una serie de ejemplos.

Trabajaremos sobre dos ejemplos, una red pequeña codificada dentro del código y otra más grande codificada en ficheros externos.

8.1. Juego de pruebas pequeño.

El primero será una red pequeña, posee 13 nodos, 19 aristas y sólo 4 parejas de nodos O-D. Debido a su reducido tamaño está codificado dentro del código la red y las funciones de rendimiento de cada arista, dentro del código será identificado como caso2 y caso2a. Como es una red pequeña la usaremos para hacer un estudio detallado, es decir, queremos mostrar cómo se cumplen las condiciones de equilibrio y estudiar el fenómeno de la saturación en una red.

A continuación mostramos una versión gráfica de la red:



Esta tabla nos muestra las parejas O-D y el flujo que ha de viajar para cada pareja.

<i>origen</i> \ <i>destino</i>	3	4
1	20	40
2	30	10

Las funciones de rendimiento de las aristas tendrán el siguiente formato:

$$f(v) = t_0 * \left(1 + \alpha * \left(\frac{v}{c} \right)^\beta \right)$$

Pero con 4 parámetros definidos para cada arista, t_o , que es el coste mínimo para ir por una arista; c , la capacidad de la arista; α y β para modelar el comportamiento de las aristas. Estos parámetros están explicados con más detalle en el capítulo 1.

Nodo origen	Nodo destino	Parámetro t_o	Parámetro c	Parámetro α	Parámetro β
1	12	10	20	0.15	4
1	5	20	40	0.15	4
2	5	20	40	0.15	4
2	9	25	30	0.15	4
5	6	10	25	0.25	4
5	9	20	50	0.10	3
12	6	25	30	0.15	4
12	8	10	20	0.15	4
6	7	20	40	0.15	4
6	10	15	25	0.20	4
7	8	20	40	0.15	4
7	11	15	25	0.20	4
8	3	10	20	0.15	4
9	10	15	25	0.20	4
9	13	15	25	0.20	4
10	11	15	25	0.20	4
11	3	15	25	0.20	4
11	4	25	30	0.15	4
13	4	15	25	0.20	4

Además usamos como parámetros de la aplicación :

toleranciaParada 0.0002
toleranciaSolver 0.001
iteracionesMax 50

Con toda esta información el sistema nos genera como resultado la siguiente información:

Primero un listado de todas las aristas, identificadas por su nodo origen y su nodo destino, con el flujo que "viajará" por esa arista x_n , y el coste de usarla t_n . La primera línea son los tiempos de ejecución de la aplicación, el primero es el del tiempo total de la aplicación, el segundo el tiempo usado para calcular las rutas de desplazamiento y el tercero el tiempo usado en la búsqueda de la longitud del paso.

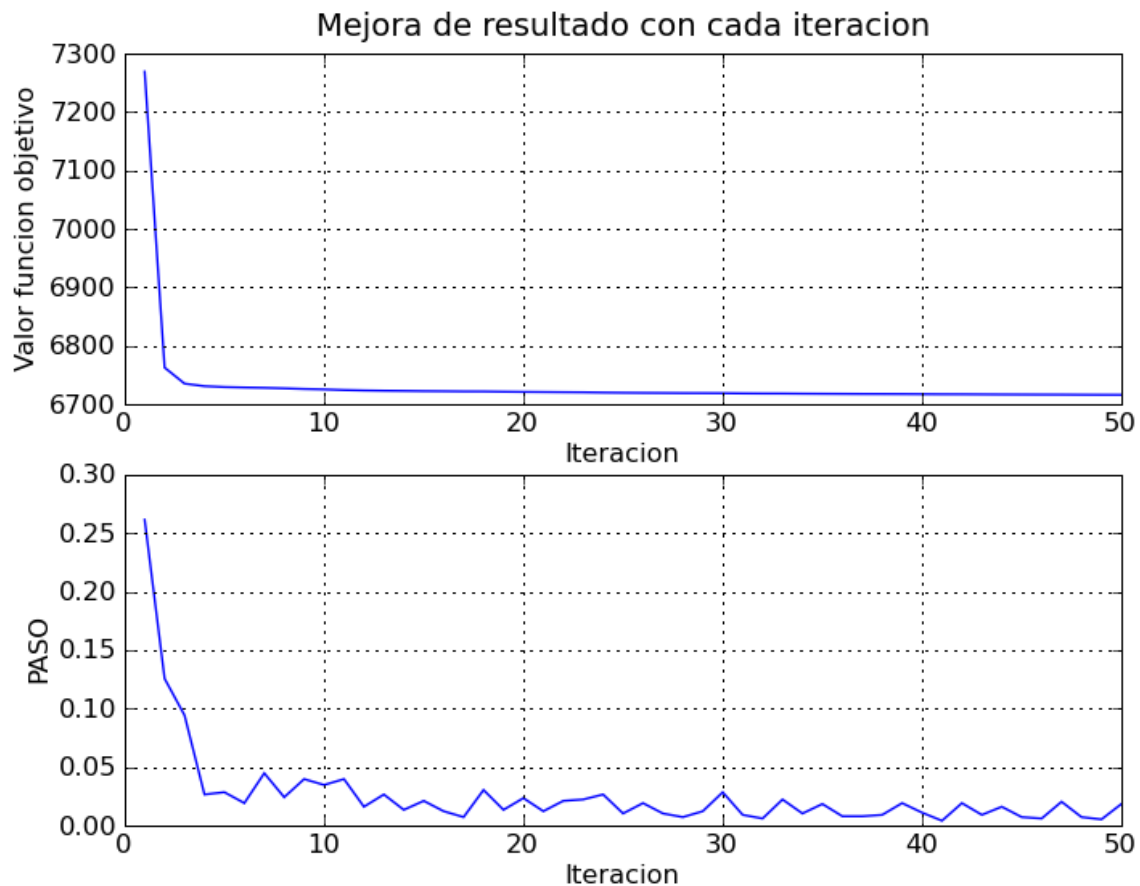
Ttotal: 1.1130001545 Tcaminos: 0.0230007171631 Tsolver: 0.0530002117157
1 2 xn: 20.0 tn: 11.5
1 5 xn: 40.0 tn: 23.0
2 9 xn: 31.7347156 tn: 29.6955308
2 5 xn: 8.2652844 tn: 20.0054691
5 9 xn: 25.6778635 tn: 20.2708923
5 6 xn: 22.5874209 tn: 11.6658909
6 10 xn: 6.2625726 tn: 15.0118133
6 7 xn: 16.3248483 tn: 20.0832296
7 8 xn: 4.5360433 tn: 20.0004961
7 11 xn: 11.788805 tn: 15.1483339
8 3 xn: 24.5360433 tn: 13.3977341
9 10 xn: 22.6343234 tn: 17.015725
9 13 xn: 34.7782558 tn: 26.2355001
10 11 xn: 28.896896 tn: 20.3550804
11 3 xn: 25.4639567 tn: 18.2289756
11 4 xn: 15.2217442 tn: 25.2485444
12 8 xn: 20.0 tn: 11.5
12 6 xn: 0.0 tn: 25.0
13 4 xn: 34.7782558 tn: 26.2355001

A continuación mostramos una serie de parámetros de cada una de las iteraciones de la aplicación. Podemos observar que a pesar de ser una red pequeña y no saturada ha llegado al máximo de iteraciones, esto es debido a la excesiva precisión que le hemos pedido a la aplicación 0.0002. El primer parámetro corresponde a la número de iteración que se está ejecutando. El segundo es el valor de la función objetivo que pretendemos minimizar. El tercero es un parámetro usado por el test de parada para estudiar si el valor de la función objetivo se puede seguir mejorando. El cuarto es el valor generado por el test de parada, podemos observar que siempre se mantiene por encima de nuestro valor de tolerancia de parada 0.0002. El quinto es la longitud del paso del algoritmo de Frank-Wolfe.

Iteracion: 1 costeFuncion: 7269.02921481 BLB: 2889.12314074 gap: 0.602543468273 paso: 0.26141952313
Iteracion: 2 costeFuncion: 6762.94962777 BLB: 6331.924732 gap: 0.0637332701691 paso: 0.125571486861
Iteracion: 3 costeFuncion: 6735.5632094 BLB: 6643.11513842 gap: 0.0137253661056 paso: 0.0942352531274
Iteracion: 4 costeFuncion: 6731.19740992 BLB: 6618.73017513 gap: 0.0130856764597 paso: 0.0265377869197
Iteracion: 5 costeFuncion: 6729.67557922 BLB: 6678.00873765 gap: 0.00767746393753 paso: 0.0284571656452
Iteracion: 6 costeFuncion: 6728.94161345 BLB: 6668.36216391 gap: 0.00756922540373 paso: 0.0191403055998
Iteracion: 7 costeFuncion: 6728.35680326 BLB: 6687.71275807 gap: 0.006040708954 paso: 0.0447184031568
Iteracion: 8 costeFuncion: 6727.45106457 BLB: 6638.65671457 gap: 0.00590688897081 paso: 0.0241653043405

Iteracion: 9 costeFuncion: 6726.38092979 BLB: 6690.55190143 gap: 0.00532664277208 paso: 0.0396934044162
Iteracion: 10 costeFuncion: 6725.67818205 BLB: 6659.31413063 gap: 0.00522271206997 paso: 0.0346684056755
Iteracion: 11 costeFuncion: 6724.5400085 BLB: 6689.04961494 gap: 0.00505433933431 paso: 0.0396934044162
Iteracion: 12 costeFuncion: 6723.84246721 BLB: 6666.46738175 gap: 0.00495112221014 paso: 0.0160346855847
Iteracion: 13 costeFuncion: 6723.39065285 BLB: 6699.48575829 gap: 0.00355548201626 paso: 0.0265377869197
Iteracion: 14 costeFuncion: 6723.06953206 BLB: 6666.84725398 gap: 0.0035078878266 paso: 0.0133821694233
Iteracion: 15 costeFuncion: 6722.68258934 BLB: 6703.855715 gap: 0.00280050026034 paso: 0.0210596843253
Iteracion: 16 costeFuncion: 6722.4818736 BLB: 6672.17588601 gap: 0.00277072648955 paso: 0.0121959281337
Iteracion: 17 costeFuncion: 6722.18008756 BLB: 6701.01924986 gap: 0.00272595680546 paso: 0.00717092939303
Iteracion: 18 costeFuncion: 6722.10637127 BLB: 6694.7396486 gap: 0.00271502045039 paso: 0.0303765443707
Iteracion: 19 costeFuncion: 6721.68985803 BLB: 6674.36107839 gap: 0.00265322313443 paso: 0.0133821694233
Iteracion: 20 costeFuncion: 6721.38355355 BLB: 6700.86142174 gap: 0.00260777240356 paso: 0.0234321669046
Iteracion: 21 costeFuncion: 6721.14768403 BLB: 6678.459798 gap: 0.00257277028258 paso: 0.0121959281337
Iteracion: 22 costeFuncion: 6720.89397267 BLB: 6701.88594605 gap: 0.00253511775882 paso: 0.0210596843253
Iteracion: 23 costeFuncion: 6720.69157616 BLB: 6679.60559165 gap: 0.00250507867696 paso: 0.022245925615
Iteracion: 24 costeFuncion: 6720.24549146 BLB: 6696.62748109 gap: 0.00243886573515 paso: 0.0265377869197
Iteracion: 25 costeFuncion: 6719.93334953 BLB: 6676.60679898 gap: 0.0023925288672 paso: 0.0102765494082
Iteracion: 26 costeFuncion: 6719.70865408 BLB: 6702.84049678 gap: 0.00235917059782 paso: 0.0191403055998
Iteracion: 27 costeFuncion: 6719.54804543 BLB: 6677.97903784 gap: 0.00233532528177 paso: 0.0102765494082
Iteracion: 28 costeFuncion: 6719.33663006 BLB: 6703.20237249 gap: 0.00230393503241 paso: 0.00717092939303
Iteracion: 29 costeFuncion: 6719.28321388 BLB: 6697.19104161 gap: 0.00229600366363 paso: 0.0121959281337
Iteracion: 30 costeFuncion: 6719.15059192 BLB: 6693.95872371 gap: 0.00227631107653 paso: 0.0284571656452
Iteracion: 31 costeFuncion: 6718.79913001 BLB: 6681.86824461 gap: 0.00222411992392 paso: 0.00909030811853
Iteracion: 32 costeFuncion: 6718.63597353 BLB: 6698.94068085 gap: 0.00219988976693 paso: 0.00598468810339
Iteracion: 33 costeFuncion: 6718.57241186 BLB: 6698.12231441 gap: 0.0021904499875 paso: 0.022245925615
Iteracion: 34 costeFuncion: 6718.33886399 BLB: 6682.96687114 gap: 0.00215576339389 paso: 0.0102765494082
Iteracion: 35 costeFuncion: 6718.16770497 BLB: 6701.8136284 gap: 0.00213034127657 paso: 0.018407168164
Iteracion: 36 costeFuncion: 6718.0176905 BLB: 6683.42525705 gap: 0.00210805867897 paso: 0.00790406682889
Iteracion: 37 costeFuncion: 6717.87187328 BLB: 6703.04294548 gap: 0.00208639857162 paso: 0.00790406682889
Iteracion: 38 costeFuncion: 6717.81176279 BLB: 6696.65918627 gap: 0.00207746931302 paso: 0.00909030811853
Iteracion: 39 costeFuncion: 6717.71957343 BLB: 6700.83518736 gap: 0.00206377451151 paso: 0.0191403055998
Iteracion: 40 costeFuncion: 6717.56067383 BLB: 6684.07981233 gap: 0.00204016896873 paso: 0.011009686844
Iteracion: 41 costeFuncion: 6717.38068483 BLB: 6698.22151956 gap: 0.00201342911268 paso: 0.00406530937789
Iteracion: 42 costeFuncion: 6717.33721067 BLB: 6699.97969681 gap: 0.00200697020885 paso: 0.0191403055998
Iteracion: 43 costeFuncion: 6717.16869765 BLB: 6682.75049802 gap: 0.00198193364648 paso: 0.00909030811853
Iteracion: 44 costeFuncion: 6717.0068989 BLB: 6702.77230494 gap: 0.00195789346373 paso: 0.0160346855847
Iteracion: 45 costeFuncion: 6716.89326862 BLB: 6686.6052254 gap: 0.00194100949633 paso: 0.00717092939303
Iteracion: 46 costeFuncion: 6716.78352123 BLB: 6697.46121515 gap: 0.00192470193356 paso: 0.00598468810339
Iteracion: 47 costeFuncion: 6716.72238538 BLB: 6698.9273981 gap: 0.00191561741644 paso: 0.0203265468895
Iteracion: 48 costeFuncion: 6716.54606988 BLB: 6685.86798742 gap: 0.00188941678415 paso: 0.00717092939303
Iteracion: 49 costeFuncion: 6716.43360168 BLB: 6700.13715727 gap: 0.00187270319767 paso: 0.00525155066753
Iteracion: 50 costeFuncion: 6716.38999937 BLB: 6699.70150581 gap: 0.00186622342753 paso: 0.018407168164

Y para que sean más fáciles de entender se generan también dos gráficas.



Al observar todos estos resultados lo primero que se puede decir es que la excesiva precisión que se pide exige un gran coste computacional y no aporta unos grandes cambios a los resultados obtenidos. Si hubiéramos usado una tolerancia de 0.01 sólo se habrían ejecutado 5 iteraciones, y observando las gráficas podemos observar como la aportación 45 siguientes iteraciones es mínima comparada con la de las 5 primeras.

En el capítulo 1 hemos de definido el equilibrio de usuarios de la siguiente manera : **Para cada pareja O-D, en un estado de equilibrio de usuarios, los tiempos de desplazamiento de todas las rutas usadas son iguales, y además menor o igual que el tiempo de desplazamiento de las rutas sin usar si las usara sólo un usuario.**

A provecharemos que este ejemplo es pequeño para comprobar este hecho.

A continuación miraremos todas las rutas para cada pareja de nodos centroides, junto con el coste de usar cada una de esas rutas y sacaremos conclusiones.

Pareja de Nodos 1-3, viajan 20 unidades.

RUTAS	COSTE
1 ⇒ 12 ⇒ 8 ⇒ 3	36.4
1 ⇒ 12 ⇒ 6 ⇒ 7 ⇒ 8 ⇒ 3	89.98
1 ⇒ 12 ⇒ 6 ⇒ 7 ⇒ 11 ⇒ 3	89.96
1 ⇒ 12 ⇒ 6 ⇒ 10 ⇒ 11 ⇒ 3	90.1
1 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 8 ⇒ 3	88.15
1 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 11 ⇒ 3	88.13
1 ⇒ 5 ⇒ 6 ⇒ 10 ⇒ 11 ⇒ 3	88.27
1 ⇒ 5 ⇒ 9 ⇒ 10 ⇒ 11 ⇒ 3	98.88

Sobre la pareja de Nodos 1-3, observamos que una ruta posee un coste inferior a todas las demás, y observando el estado final de las aristas dadas por el algoritmo, en especial el no uso de la arista 12-> 6 y el uso de las aristas 1-> 12 y 1->5 podemos afirmar que las 20 unidades viajan por una única ruta, la primera de la lista.

Pareja de Nodos 1-4, viajan 40 unidades.

RUTAS	COSTE
1 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 11 ⇒ 4	95.15
1 ⇒ 5 ⇒ 6 ⇒ 10 ⇒ 11 ⇒ 4	95.29
1 ⇒ 5 ⇒ 9 ⇒ 10 ⇒ 11 ⇒ 4	105.9
1 ⇒ 5 ⇒ 9 ⇒ 13 ⇒ 4	95.75

Sobre la pareja de Nodos 1-4, observamos que es el caso contrario, todas las rutas que comienzan en la arista 1->5 son usadas menos una. Y por los resultados del algoritmo podemos ver que es correcto y que se están usando esas tres rutas.

Pareja de Nodos 2-3, viajan 30 unidades.

RUTAS	COSTE
2 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 8 ⇒ 3	85.5
2 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 11 ⇒ 3	85.13
2 ⇒ 5 ⇒ 6 ⇒ 10 ⇒ 11 ⇒ 3	85.27
2 ⇒ 5 ⇒ 9 ⇒ 10 ⇒ 11 ⇒ 3	95.88
2 ⇒ 9 ⇒ 10 ⇒ 11 ⇒ 3	85.31

Sobre la pareja de Nodos 2-3 , podemos decir lo mismo que en el caso anterior, es decir usamos todas las rutas posibles excepto uno, la cuarta, claramente con unos costes superiores a las demás.

Pareja de Nodos 2-4, viajan 10 unidades.

RUTAS	COSTE
2 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 11 ⇒ 4	92.15
2 ⇒ 5 ⇒ 6 ⇒ 10 ⇒ 11 ⇒ 4	92.29
2 ⇒ 9 ⇒ 10 ⇒ 11 ⇒ 4	92.33
2 ⇒ 9 ⇒ 13 ⇒ 4	82.18

Sobre la pareja de Nodos 2-4 , podemos decir que es un caso idéntico al de la pareja 1-3, sólo se usa una ruta, la última.

Con estos datos se concluye que esta red cumple el criterio de equilibrio de usuarios.

Ahora queremos mostrar el efecto de la congestión sobre el funcionamiento del algoritmo.

Para ello hemos hecho una serie de ejecuciones del algoritmo de Frank-Wolfe con este ejemplo pero variando el flujo, siguiendo la siguiente tabla y un valor de $1 \leq K \leq 10$

<i>origen</i> \ <i>destino</i>	3	4
1	K *10	K *20
2	K *15	K *5

Además hemos modificado los parámetros del sistema, pidiendo una tolerancia de parada mucho menos precisa pero igualmente buena, también hemos incrementado el número máximo de iteraciones del sistema a 500.

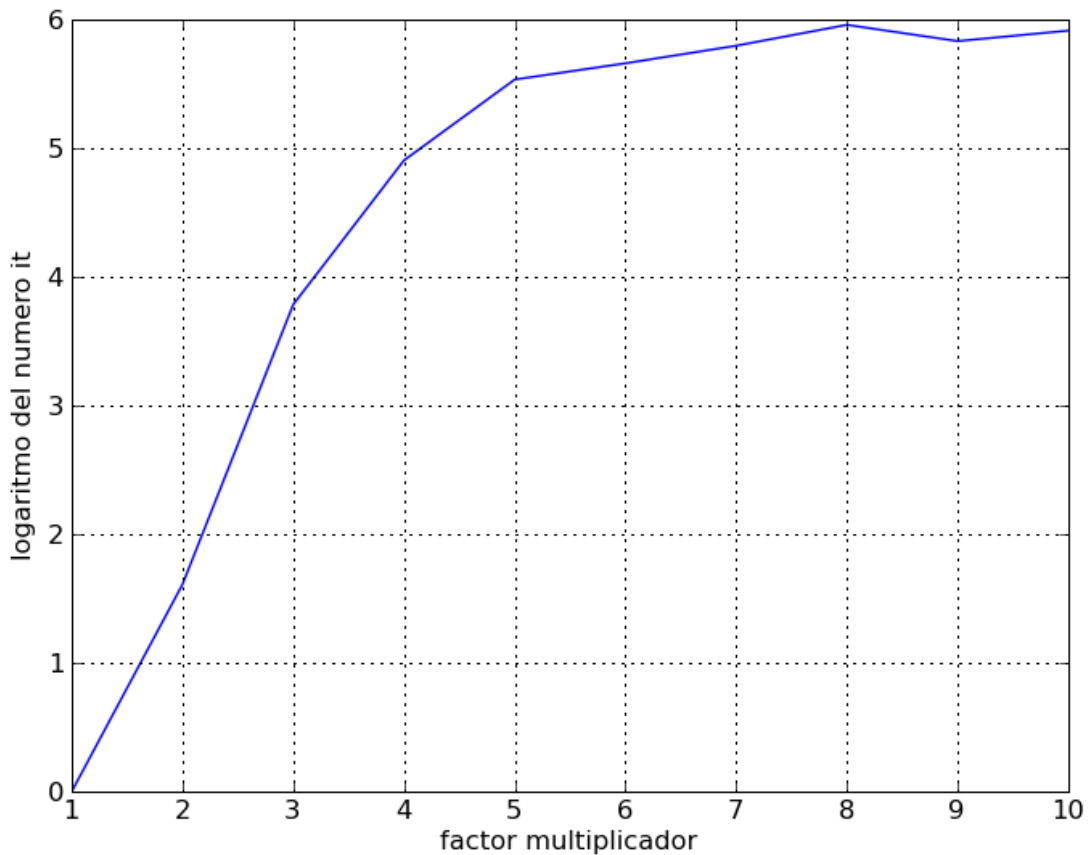
toleranciaParada 0.01

toleranciaSolver 0.001

iteracionesMax 500

Mostramos los resultados obtenidos en forma de tabla siguiendo la relación entre el valor del

parámetro k y el logaritmo del número de iteraciones que ha necesitado el algoritmo



Observando la gráfica podemos ver dos partes diferenciadas, la parte con valores de k menores a 5 y las superiores a 5. La primera parte nos muestra como se incrementa de forma elevada el número de iteraciones necesarias según el flujo del sistema aumenta. Y la segunda nos muestra un estancamiento en el número de iteraciones a pesar del incremento continuo de flujo. Estudiando la capacidad de las aristas de la red podemos observar que el límite de capacidad se alcanza con valores de K entre 4 y 5. Lo que ocurre a continuación de estos valores es un estado de completa saturación del sistema, es decir, el sistema está colapsado y empieza a carecer de sentido sus datos. Porque da igual que ruta se escoja, los costes son elevadísimos llegando a unas escalas donde una mínima variación del flujo produce grandes cambios de valor en las funciones de rendimiento del sistema, si comparamos esos valores con los que obtenemos con una red no saturada. Estos elevadísimos costes son los que provocan que el sistema siga intentando mejorar los resultados llegando a ejecutar entre 300 y 400 iteraciones, es muy importante vigilar si estamos introduciendo

una red que va a quedar saturada o no, esto se puede ver por las datos iniciales del sistema. Este efecto y otros ya mencionados anteriormente son los que indican por qué es tan importante poner un límite de iteraciones para el algoritmo.

8.2. Juego de pruebas mediano

Este segundo juego de pruebas será una red mediana, posee casi 350 nodos, y una 850 aristas y alrededor de 300 parejas de nodos O-D. Debido a su gran tamaño la red y las funciones de rendimiento de cada arista están codificadas dentro de ficheros de texto.

Además usamos como parámetros de la aplicación :

toleranciaParada 0.002
toleranciaSolver 0.001
iteracionesMax 50

Con toda esta información el sistema nos genera como resultado la siguiente información:

Primero deberíamos mostrar un listado de todas las aristas, identificadas por su nodo origen y su nodo destino, con el flujo que "viajará" por esa arista x_n , y el coste de usarla tn , pero debido a que hay más de 800 aristas nos lo saltaremos y sólo mostraremos, los tiempos de ejecución de la aplicación, el primero es el del tiempo total de la aplicación, el segundo el tiempo usado para calcular las rutas de desplazamiento y el tercero el tiempo usado en la búsqueda de la longitud del paso.

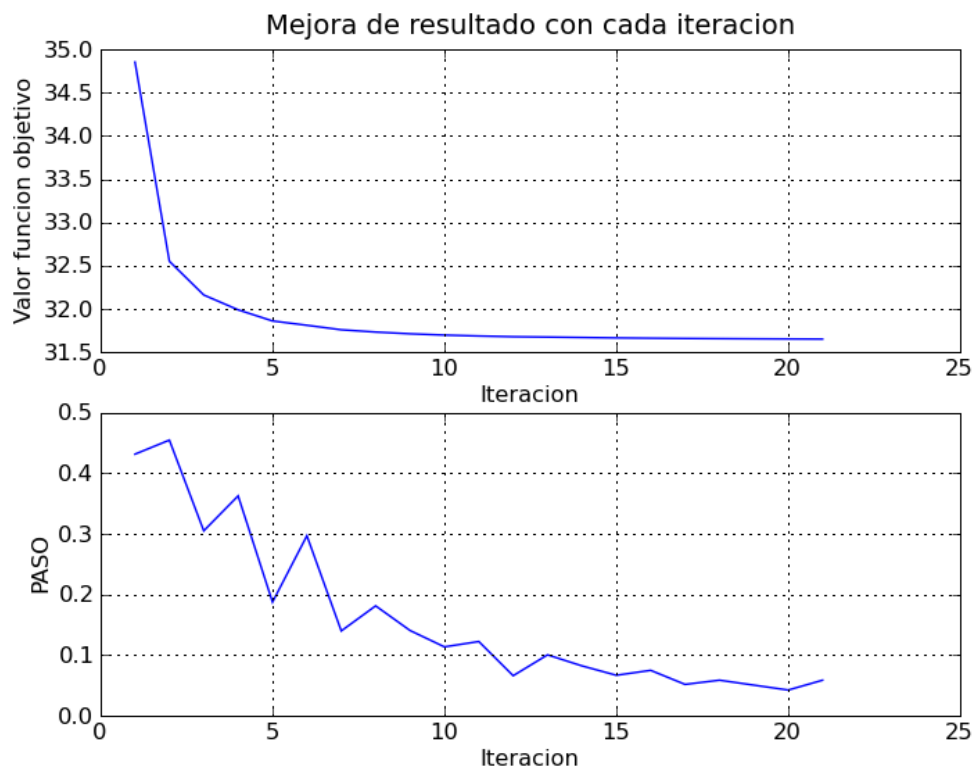
Ttotal: 7.30200004578 Tcaminos: 6.09799957275 Tsolver: 0.916999340057

Lo que sí mostraremos es la información sobre las iteraciones, su significado es idéntico al del ejemplo anterior, pero teniendo en cuenta que esta vez la tolerancia es de 0.002.

Iteracion: 1 costeFuncion: 34.8521443277 BLB: 23.5261047401 gap: 0.324973966624 paso: 0.431709413148
Iteracion: 2 costeFuncion: 32.5509604371 BLB: 30.8014836884 gap: 0.0537457797014 paso: 0.454915028125
Iteracion: 3 costeFuncion: 32.1612892551 BLB: 31.0434656822 gap: 0.034756802316 paso: 0.305178236924
Iteracion: 4 costeFuncion: 31.9898056532 BLB: 31.2852085311 gap: 0.0220256768583 paso: 0.36282570565
Iteracion: 5 costeFuncion: 31.8611222235 BLB: 31.3290738138 gap: 0.0166989852383 paso: 0.187510816892
Iteracion: 6 costeFuncion: 31.8112269077 BLB: 31.4612924937 gap: 0.0110003432138 paso: 0.297047618168
Iteracion: 7 costeFuncion: 31.7591528491 BLB: 31.3849731776 gap: 0.00937872482944 paso: 0.139913345647
Iteracion: 8 costeFuncion: 31.7329220977 BLB: 31.511497539 gap: 0.00697775509056 paso: 0.181299576862
Iteracion: 9 costeFuncion: 31.7126878382 BLB: 31.5007079294 gap: 0.00634415790632 paso: 0.140646483083
Iteracion: 10 costeFuncion: 31.6977342307 BLB: 31.5149916317 gap: 0.00576516282646 paso: 0.113602110654

Iteracion: 11 costeFuncion: 31.6873884473 BLB: 31.5500421929 gap: 0.00433441382003 paso: 0.122465866846
 Iteracion: 12 costeFuncion: 31.6789440081 BLB: 31.5534530066 gap: 0.00396133789885 paso: 0.066004639409
 Iteracion: 13 costeFuncion: 31.6748193354 BLB: 31.5823355595 gap: 0.00291978858467 paso: 0.100446493158
 Iteracion: 14 costeFuncion: 31.6701920385 BLB: 31.5512238126 gap: 0.0027741062919 paso: 0.0822658769206
 Iteracion: 15 costeFuncion: 31.6652715663 BLB: 31.5624935502 gap: 0.00261914718191 paso: 0.0667377768449
 Iteracion: 16 costeFuncion: 31.6618684994 BLB: 31.5808214115 gap: 0.00251194713521 paso: 0.0748683956007
 Iteracion: 17 costeFuncion: 31.6588272601 BLB: 31.579393566 gap: 0.00241612552193 paso: 0.0516627806229
 Iteracion: 18 costeFuncion: 31.6567874334 BLB: 31.5869641433 gap: 0.00220563410727 paso: 0.0586071580891
 Iteracion: 19 costeFuncion: 31.6547266963 BLB: 31.5879715221 gap: 0.00210885327883 paso: 0.0504765393333
 Iteracion: 20 costeFuncion: 31.6530277177 BLB: 31.5774963573 gap: 0.0020552913969 paso: 0.0423459205775
 Iteracion: 21 costeFuncion: 31.6514131115 BLB: 31.5981630018 gap: 0.00168239280568 paso: 0.0586071580891

También mostraremos las dos gráficas.



Con esta información podemos observar que la precisión que se pide al sistema y el número de iteraciones son proporcionales. Pero lo que se puede observar es que a pesar de que el número de iteraciones es la mitad el tiempo de ejecución es 7 veces superior. Esto nos indica que los tiempos de ejecución no dependen únicamente del número de iteraciones sino que también dependen de otras características. Destacamos tres posibilidades, la primera es la interacción con el usuario y el sistema operativo, es decir, la lectura y escritura de datos, la generación de gráficas y la compilación dinámica de python; la segunda es el algoritmo de minimización, estos algoritmos

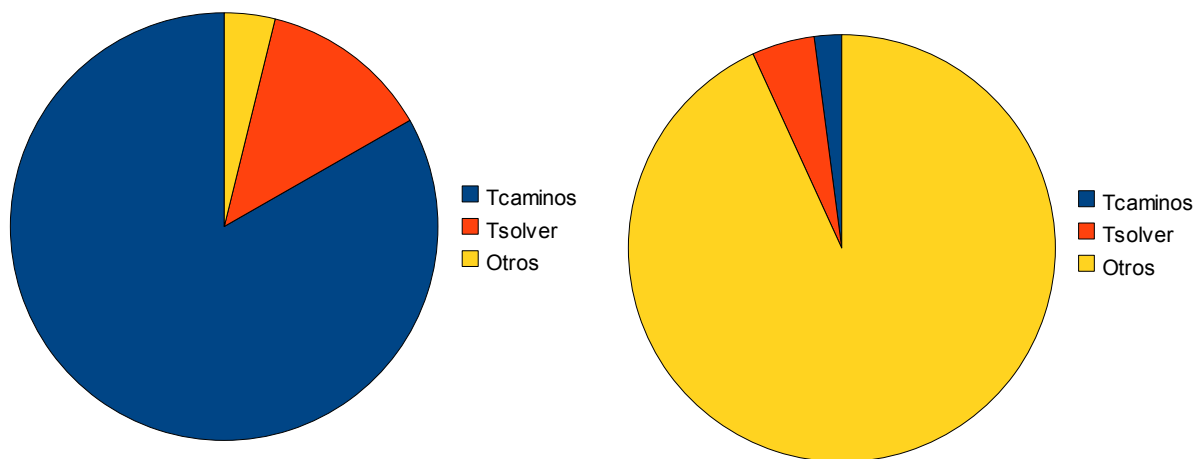
suelen conllevar un gran consumo de computación; el tercero es la búsqueda de caminos mínimos en el paso del algoritmo de FrankWolfe que busca la dirección de descenso.

Para estudiar estas posibilidades se hicieron dos ejecuciones de los juegos de prueba mediano y pequeño con los mismo parámetros de configuración, y dieron los siguientes resultados:

Red mediana. Ttotal: 16.5039999485 Tcaminos: 13.7420005798 Tsolver: 2.12899971008

Red pequeña. Ttotal: 1.1130001545 Tcaminos: 0.0230007171631 Tsolver: 0.0530002117157

Es más fácil de estudiar de forma gráfica, la de la izquierda corresponde con la red mediana y la de la derecha con la red pequeña. Observando todos los datos podemos sacra conclusiones sobre los tres factores posibles. El primero , etiquetado como otros, supone un coste casi fijo para cada iteración que ronda entre el medio segundo y el segundo. El segundo, etiquetado como Tsolver vemos que sí que depende del número de iteraciones y del número de aristas y que supone la mayor parte del coste en redes pequeñas. El tercero, etiquetado como Tcaminos, vemos que también depende del número de iteraciones pero que está intimamente ligado al tamaño del grafo y el número de parejas O-D, este es el mayor coste en los ejemplos más parecidos a casos reales, por lo que es primordial intentarlo reducir lo máximo posible.



8.3. Juego de pruebas con demanda elástica.

El punto más difícil ha sido la búsqueda de juegos de pruebas para la demanda elástica.

Por ello se trabajará con una red pequeña, exactamente, sobre la red del ejemplo pequeño.

Y como funciones de demanda se usará las anteriormente definidas:

$$t_w^A(f_w^A) = \frac{1}{\beta_A} \left(\ln \left(\frac{g_w}{\hat{g}_w - f_w^A} - 1 \right) + \beta_B t_w^B - \theta_p - \theta_q \right)$$

Pero existe un gran problema, que es la asignación de valores a los parámetros de las funciones más

concretamente $\beta_A, \beta_B, t_w^B, \theta_p, \theta_q$. En los intentos realizados para asignar valores no han salido

bien, este hecho se puede observar en dos hechos, todos los usuarios posibles de la red privada

\hat{g}_w deciden seguir usándola o todos los usuarios dejan de usarla. Por ello se decide usar una técnica que relacione el comportamiento de la red elástica con una versión de ella inelástica. En este caso nos basaremos en los resultados del primer ejemplo. El proceso para obtener los valores de

$\beta_A, \beta_B, t_w^B, \theta_p, \theta_q$ es el siguiente:

Paso 0)

$$\beta_A = \frac{1}{\sum_{w \in W} g_w^A t_w^A}$$

Esto nos da $\beta_A = 0.000126186$

Paso 1)

$$t_w^B \approx 1.15 t_w^A$$

$$g_w^B \approx g_w^A$$

$$\beta_B = \frac{1}{\sum_{w \in W} g_w^B t_w^B}$$

Esto nos da $\beta_B = 0.000109727$

Paso 2)

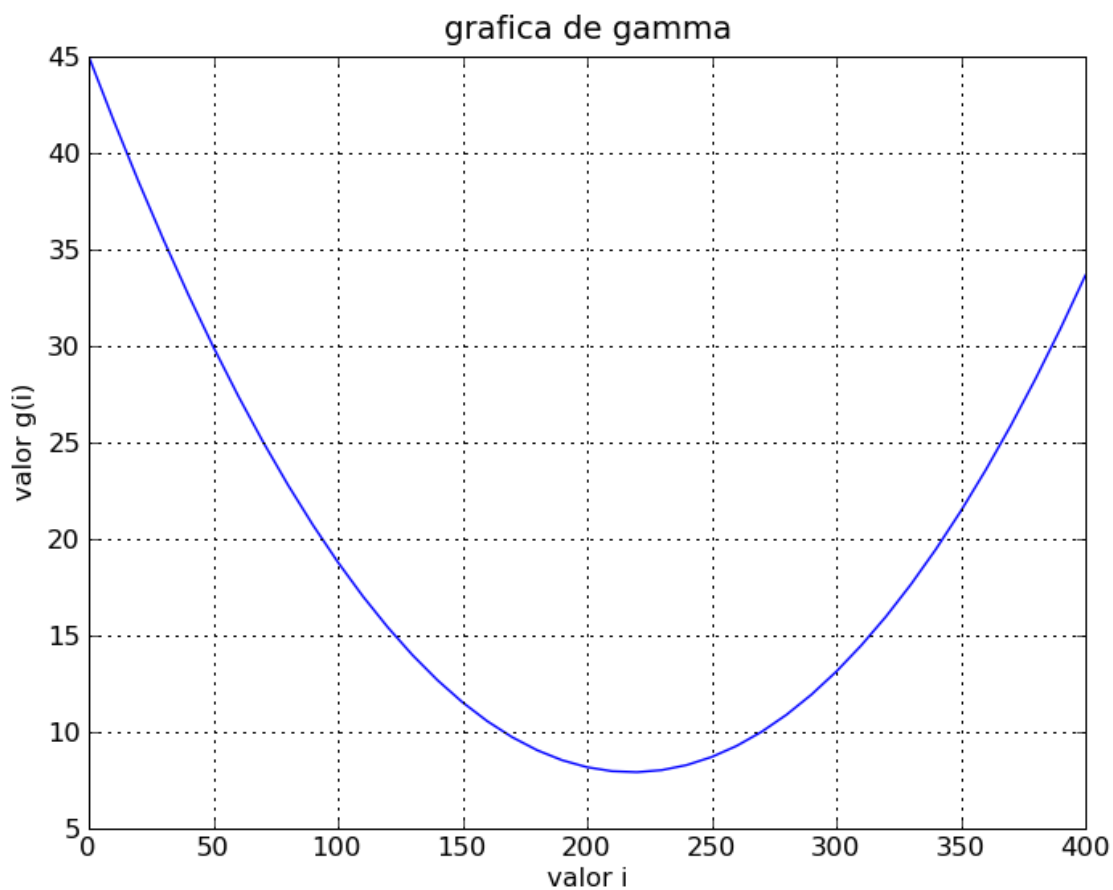
Fijamos tabla de precios :

coste viaje transporte privado : 3 €
 coste viaje transporte público : 1.5 €
 coste aparcamiento origen : 2 €
 coste aparcamiento destino : 6€

Hallar $\gamma > 0$ tal que minimice

$$\psi(\gamma) = \sum_{w \in W} (\gamma \beta_A t_w^A - 3)^2 + \sum_{w \in W} (\gamma \beta_B t_w^B - 1.5)^2$$

Esto se puede estudiar de una forma gráfica.



Lo que nos da un valor de 220.

Paso 3)

$$\theta_p = \frac{2}{\gamma^*} \quad \theta_q = \frac{6}{\gamma^*}$$

$$\theta_p = 0.0090909$$

$$\theta_q = 0.0272727$$

Con esta nueva información podemos realizar una nueva ejecución que sí nos asegura un comportamiento deseable del sistema. El margen de precisión es de 0.002.

El resultado que nos genera es :

Iteracion: 1 costeFuncion: 14523087.3223 BLB: -3915414.34627 gap: 1.26959931173 paso: 0.918920364369

Iteracion: 2 costeFuncion: 12413821.082 BLB: 12259308.3343 gap: 0.0124468321784 paso: 0.282705759382

Iteracion: 3 costeFuncion: 12391497.8284 BLB: 12351067.097 gap: 0.00326278000483 paso: 0.246344526908

Iteracion: 4 costeFuncion: 12386714.051 BLB: 12382566.9181 gap: 0.000334804924626 paso: 0.0691102594242

Ttotal: 0.0659999847412 Tcaminos: 0.00400018692017 Tsolver: 0.00600004196167

1 4 xn: 2.0649062 tn: 268.8328066

1 3 xn: 2.0724989 tn: 153.2089086

1 12 xn: 50.2527114 tn: 69.7873457

1 5 xn: 47.0121855 tn: 25.7243088

2 9 xn: 46.8394625 tn: 47.2840365

2 3 xn: 2.066206 tn: 249.0383601

2 4 xn: 2.0666076 tn: 242.9237403

2 5 xn: 50.3174932 tn: 27.5120293

5 9 xn: 45.5154186 tn: 21.5086747

5 6 xn: 51.8142601 tn: 56.1293332

6 10 xn: 11.6054305 tn: 15.1393177

6 7 xn: 51.8142601 tn: 28.4465332

7 8 xn: 11.612736 tn: 20.0213118

7 11 xn: 40.2015241 tn: 35.0600168

8 3 xn: 50.2600168 tn: 69.8221194

9 10 xn: 38.5321896 tn: 31.9299617

9 13 xn: 53.8226915 tn: 79.4499917

10 11 xn: 50.1376202 tn: 63.5306473

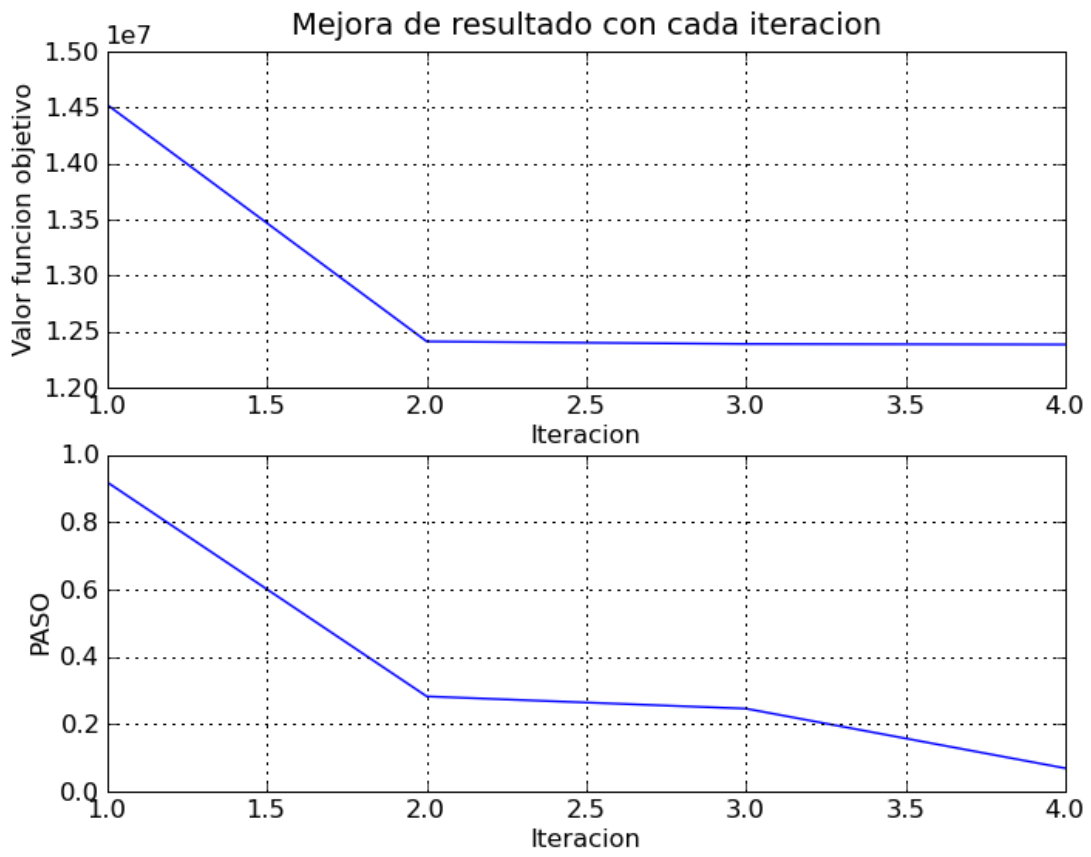
11 3 xn: 47.0354372 tn: 52.589103

11 4 xn: 43.3037071 tn: 41.2797058

12 8 xn: 38.6472809 tn: 30.914479

12 6 xn: 11.6054305 tn: 25.083983

13 4 xn: 53.8226915 tn: 79.4499917



Capítulo 9: Coste del proyecto

Costes del proyecto en hardware y software

Materiales	Coste
PC de gama media con licencia de Windows Vista	480,00 €
OpenOffice v. 3.0	gratuito
Python v. 2.5.4	gratuito
PythonWin	gratuito
Dia	gratuito
NumPy v. 1.2	gratuito
Matplotlib v. 0.98	gratuito
NetworkX v. 0.36	gratuito

Costes del proyecto en personal.

El proyecto se puede separar en cinco fases: el análisis de requerimientos, la especificación y diseño de la aplicación, la implementación de la aplicación, las pruebas y documentación. Esto no quiere decir que sean independientes ni que hasta una no esté completa no se pueda comenzar la siguiente, es decir, el proceso de desarrollo seguido se basa en la implementación de prototipos.

Análisis de requerimientos : Fase destinada al estudio del problema y análisis de los puntos clave que nos plantea este problema.

Diseño y especificación de la aplicación : En esta fase se pretende estudiar el análisis realizado del problema y proponer posibles soluciones en forma software.

Implementación: Fase destinada a la implementación en un lenguaje de programación de las posibles soluciones presentadas en la fase anterior.

Pruebas : En esta fase se plantean y se prueban una serie de juegos de prueba para verificar que el software desarrollado solucione el problema planteado inicialmente.

Documentación : Es importante la escritura de una buena documentación para dejar constancia del proceso realizado. Así se podrá usar para futuras modificaciones y que los usuarios de la aplicación puedan entenderla en caso de que quieran usarla o ampliarla.

Tarea	Número de horas
Análisis de requerimientos	180
Diseño y especificación de la aplicación	120
Implementación	200
Pruebas	80
Documentación	70

Todo este proceso debe ser llevado por un ingeniero o un matemático, y en caso de ser posible por un ingeniero informático.

Coste total del proyecto = coste del proyecto en hardware y software + coste del proyecto en personal = 480€ + 650 horas * 20 €/hora = 13480 €

Bibliografía

Redes de transporte y equilibrio de usuarios

Sheffy Y. (1985) "Urban Transportation Networks. Equilibrium Analysis with Mathematical Methods." Prentice Hall, Englewood Cliffs, New Jersey.

M. Florian "An introduction to network equilibrium models used in transportation problems"

Dimitri P. Bertsekas. "A simple and fast Label correcting algorithm for Shortest Paths", marzo 1992, Networks Vol. 23

Departamento EIO-UPC, Material de la asignatura Modelos de optimización para problemas de Transporte, del Máster en Logística, Transporte y Movilidad.

http://en.wikipedia.org/wiki/Category:Optimization_algorithms

<http://glossary.computing.society.informs.org/index.php?page=supplmnt.html>

<http://www.trb.org/default.asp>

http://www.gnu.org/software/gsl/manual/html_node/One-dimensional-Minimization.html

Python

Raúl González Duque, Python para todos . <http://mundogeek.net/tutorial-python/>

<http://mundogeek.net/traducciones/guia-estilo-python.htm>

<http://docs.python.org/index.html>

<http://numpy.scipy.org/>

<http://matplotlib.sourceforge.net/>

<http://networkx.lanl.gov>

<http://abel.ee.ucla.edu/cvxopt/>

<http://effbot.org/zone/readline-performance.htm>

[http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://en.wikipedia.org/wiki/Python_(programming_language))

Anexo A : MANUAL DE INSTALACIÓN

INSTALACIÓN PYTHON

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

Descargar la versión de **Python 2.5.x** o superior.

Página Web: <http://www.python.org/>

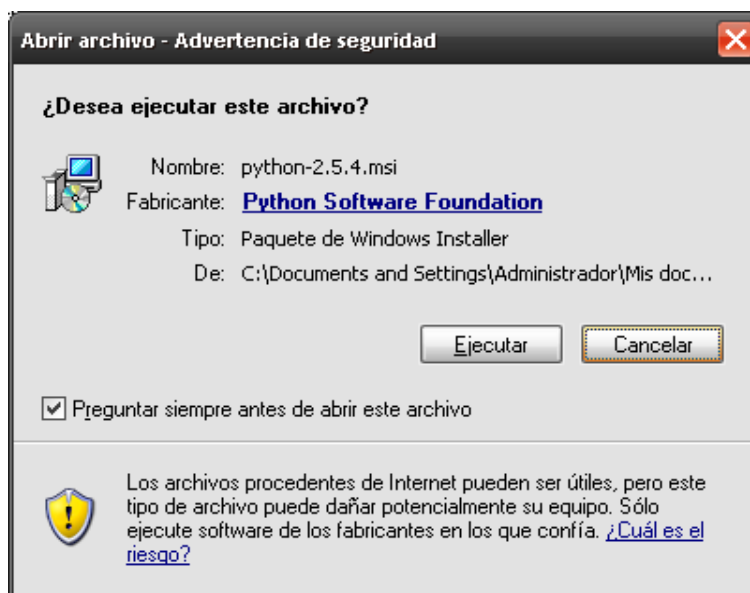
Enlace descarga : <http://www.python.org/download/>

Versión recomendada(2.5.4): <http://www.python.org/download/releases/2.5.4/>

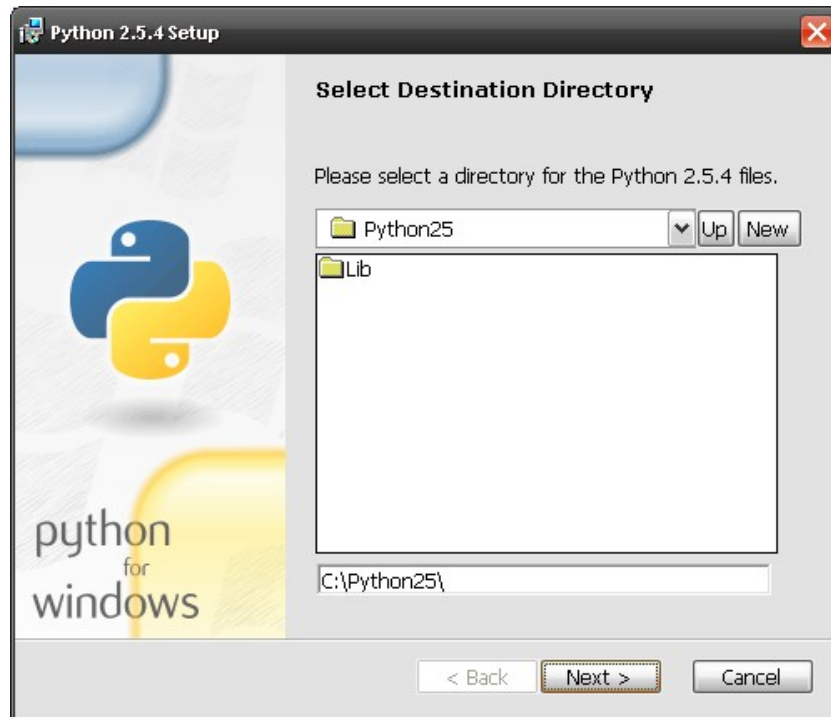
Una vez descargada, instalar en el disco duro.

Instalación Windows XP:

Paso 1. Ejecutar el instalador de Python.



Paso 2. Elegir directorio destino para la instalación. Presionamos Next para continuar con la instalación..



Paso 3. Seleccionar complementos a instalar. Mantendremos defecto los valores predeterminados, presionamos Next y comenzará la instalación.



Paso 4. Si la instalación acaba correctamente veremos la siguiente pantalla.



INSTALACIÓN NUMPY

NumPy es una extensión del lenguaje de programación **Python**, incorpora soporte para vectores multi-dimensionales y matrices, junto con una gran biblioteca de alto nivel funciones matemáticas para poder operar con estos vectores y matrices.

Descargar la versión 1.2.1 o superior.

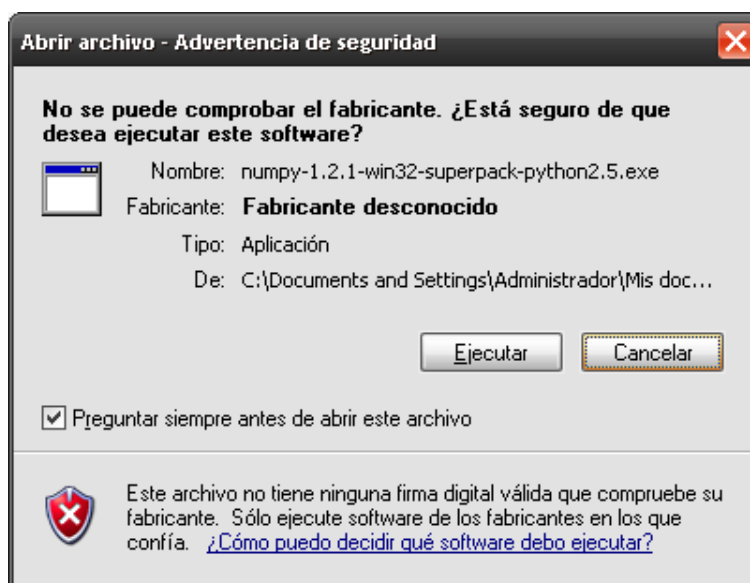
Página Web: <http://numpy.scipy.org/>

Enlace descarga: http://sourceforge.net/project/showfiles.php?group_id=1369&package_id=175103

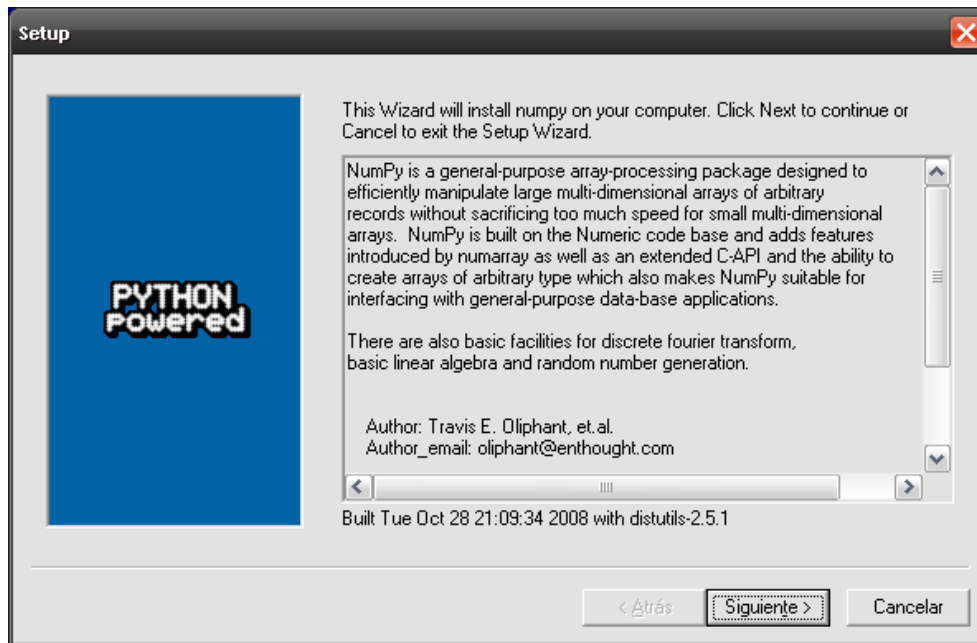
Una vez descargada, instalar en el disco duro.

Instalación Windows XP

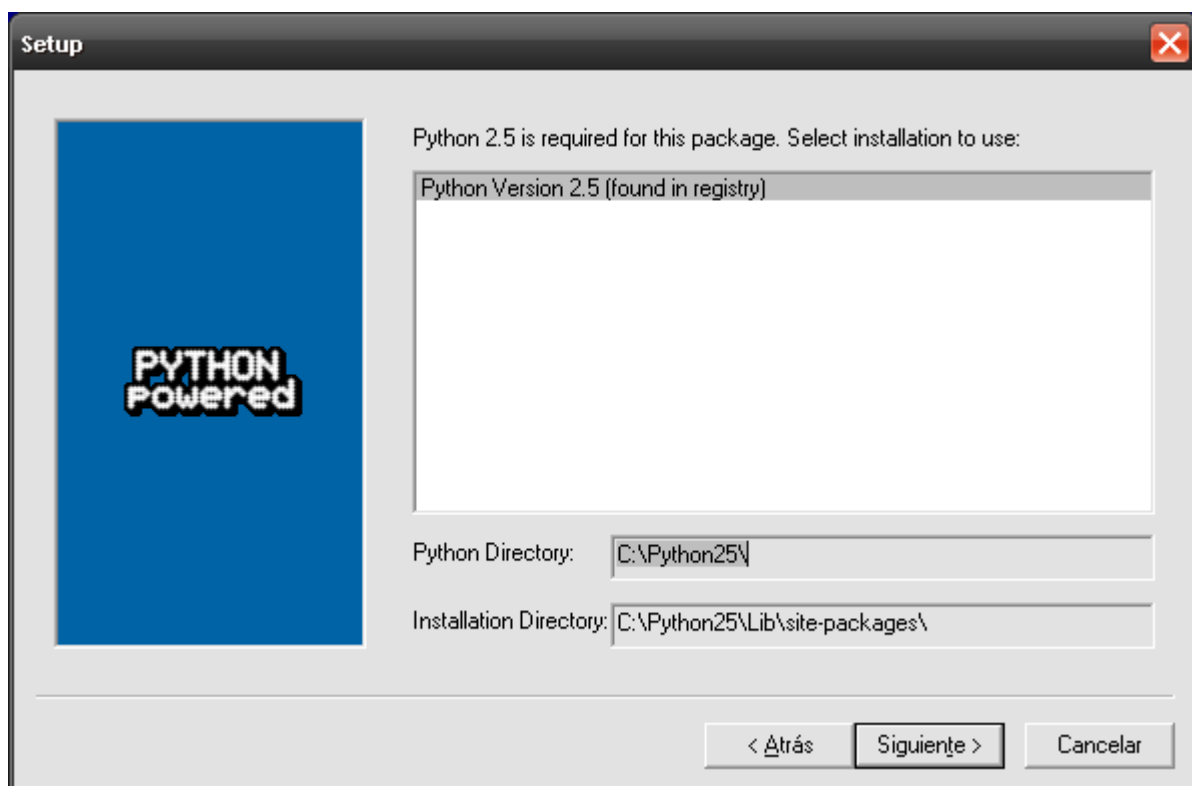
Paso 1. Ejecutar el instalador de NumPy.



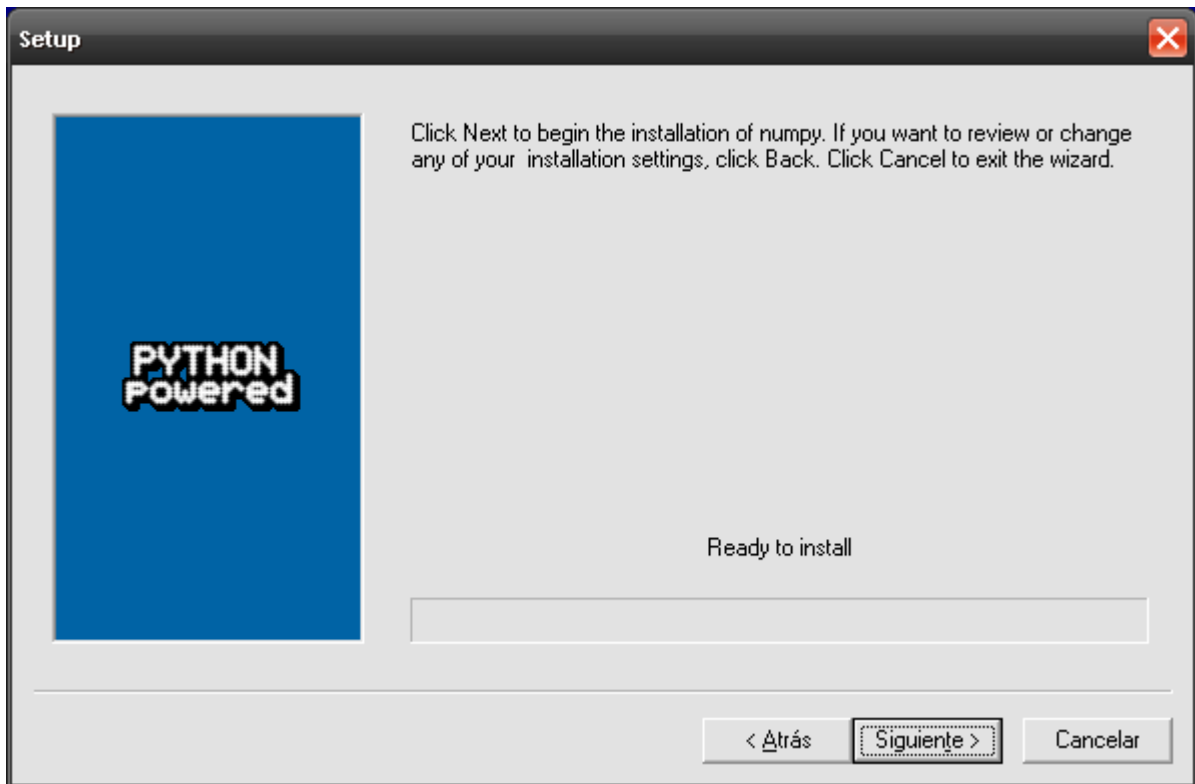
Paso 2. Iniciamos la instalación. Presionamos Siguiente.



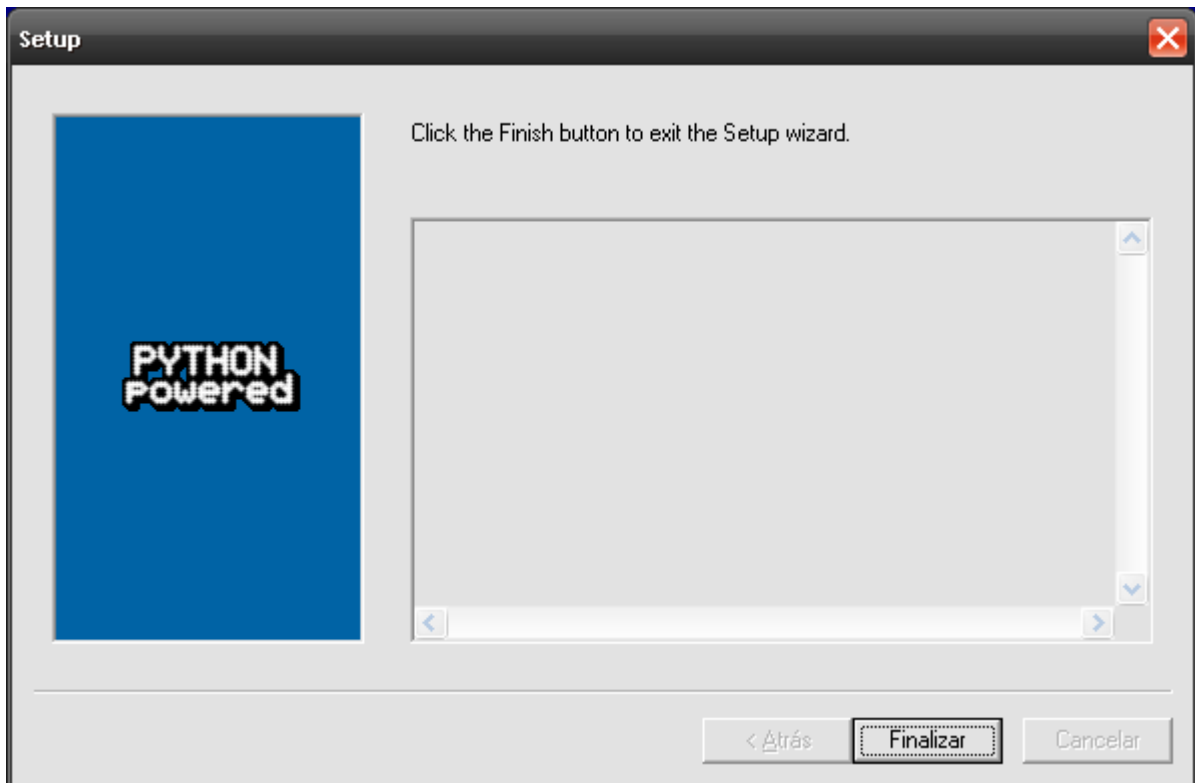
Paso 3. Elegimos directorio de instalación. Una vez seleccionado presionamos Siguiente.



Paso 4. Si todos los parámetros de la instalación son correctos, presionamos Siguiente y se iniciará instalación.



Paso 5. Si la instalación acaba correctamente veremos la siguiente pantalla.



INSTALACIÓN MATPLOTLIB

Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o vectores en el lenguaje de programación Python y su extensión matemática NumPy. Proporciona una API, **pylab**, diseñada para recordar a la de MATLAB.

Descargar la versión 0.98.5 (recomendada) o superior.

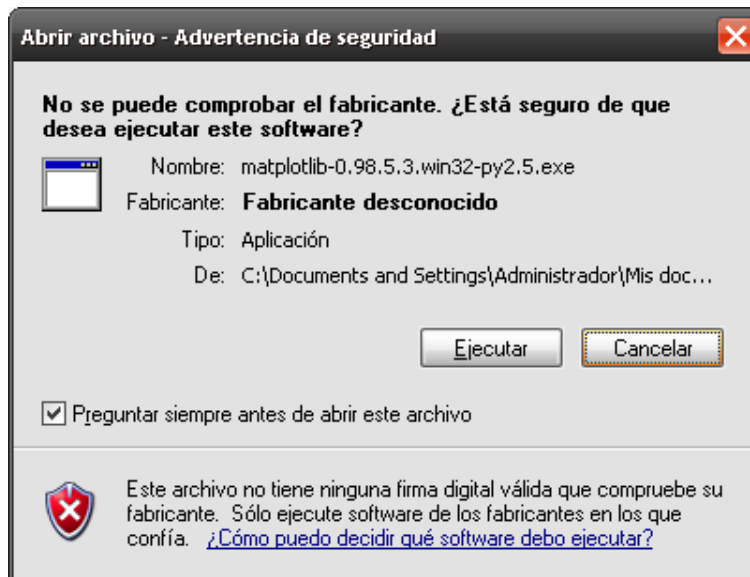
Página Web: <http://matplotlib.sourceforge.net/>

Enlace Descarga: (Versión 0.98.5) http://sourceforge.net/project/showfiles.php?group_id=80706&package_id=278194&release_id=646644

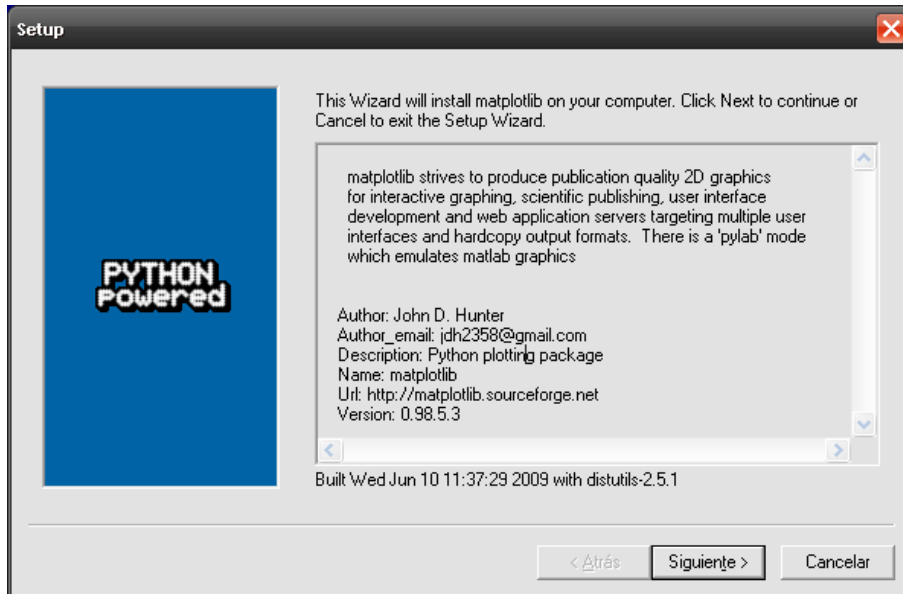
Una vez descargada, instalar en el disco duro.

Instalación Windows XP

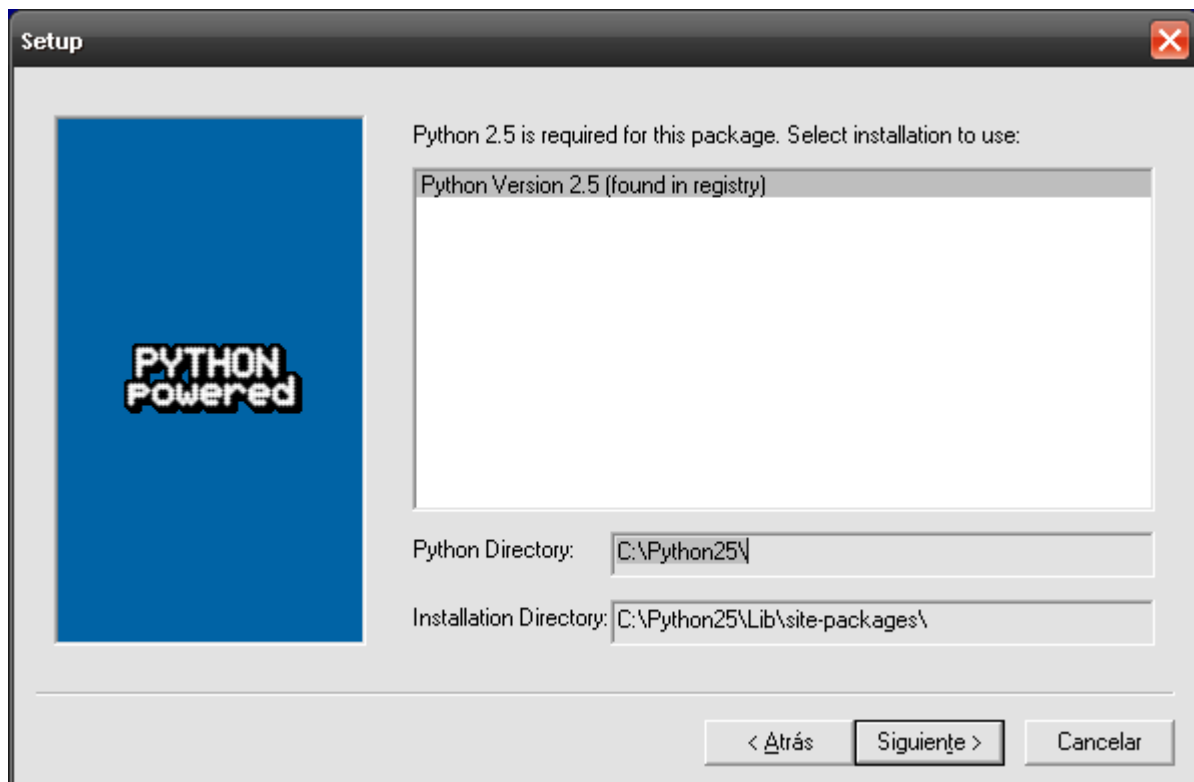
Paso 1. Ejecutar el instalador de Matplotlib



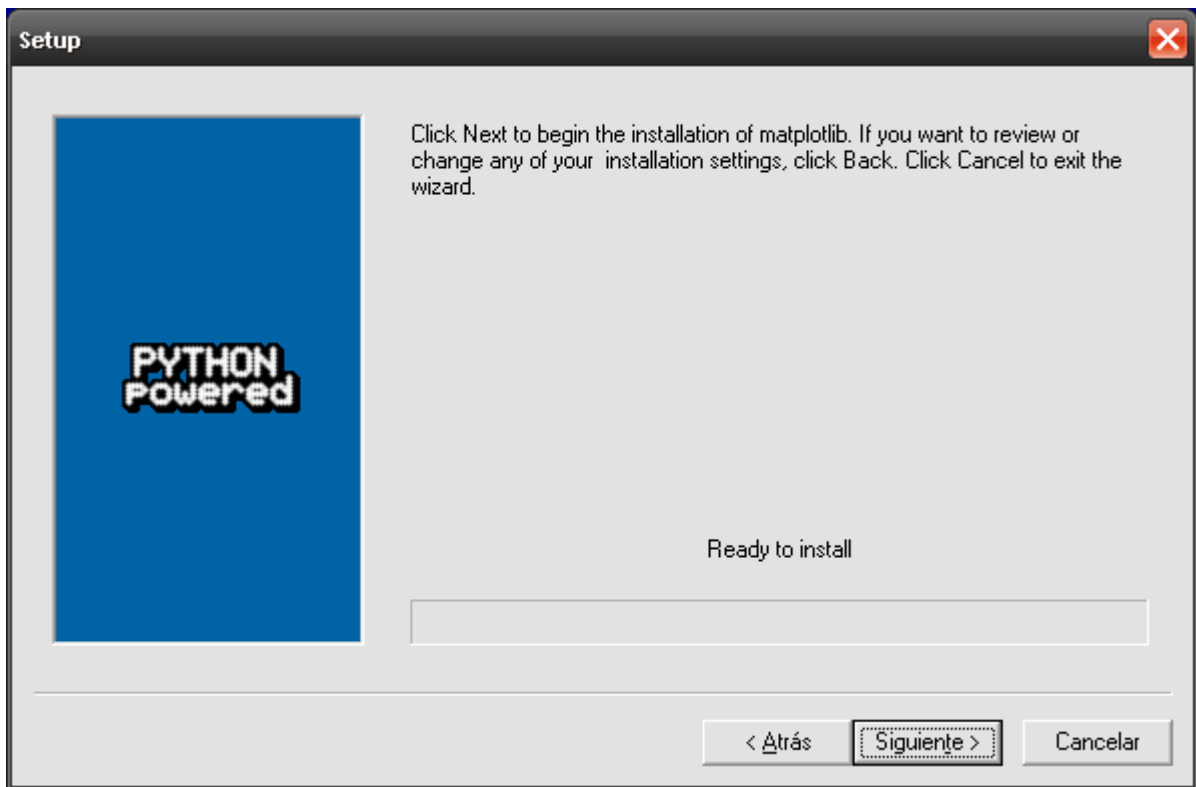
Paso 2. Iniciamos la instalación. Presionamos Siguiente.



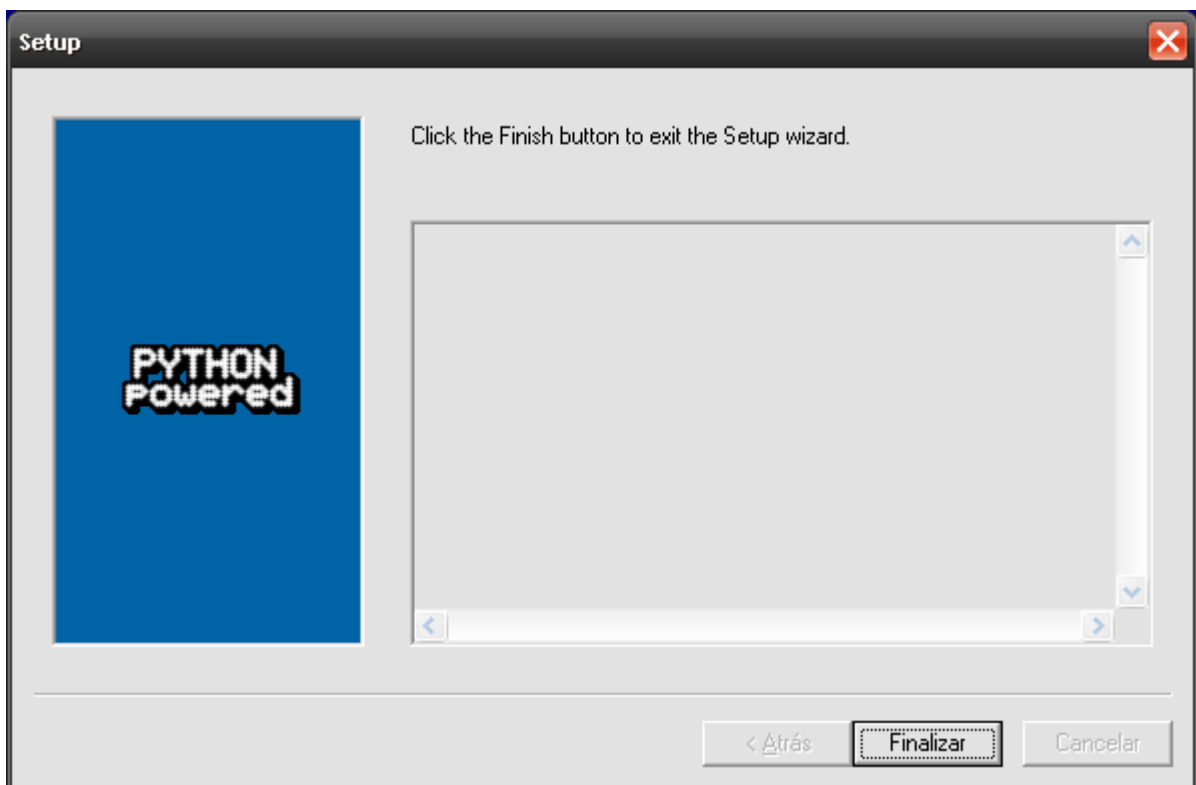
Paso 3. Elegimos directorio de instalación. Una vez seleccionado presionamos Siguiente.



Paso 4. Si todos los parámetros de la instalación son correctos, presionamos Siguiente y se iniciará instalación.



Paso 5. Si la instalación acaba correctamente veremos la siguiente pantalla.



INSTALAR NETWORKX

NetworkX es un paquete de **Python** para la creación, manipulación, estudio de la estructuras, dinámica y funciones de redes complejas.

Descargar la versión 0.36

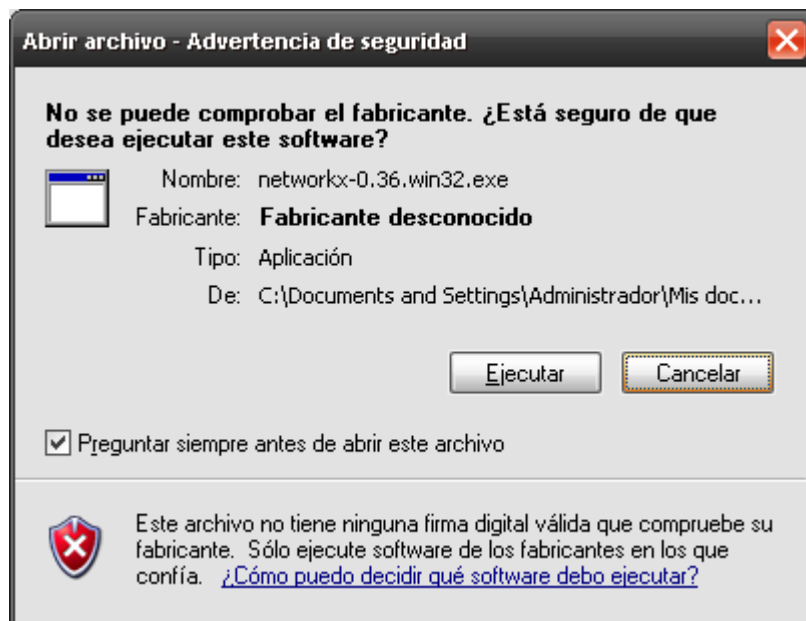
Página Web: <http://networkx.lanl.gov>

Enlace Descarga: <http://networkx.lanl.gov/download/networkx/>

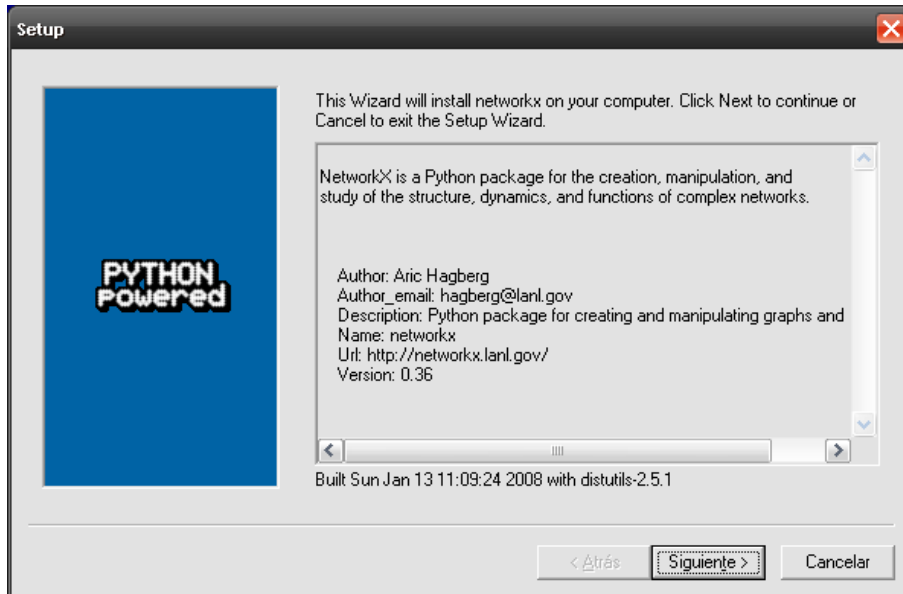
Una vez descargada, instalar en el disco duro.

Instalación Windows XP:

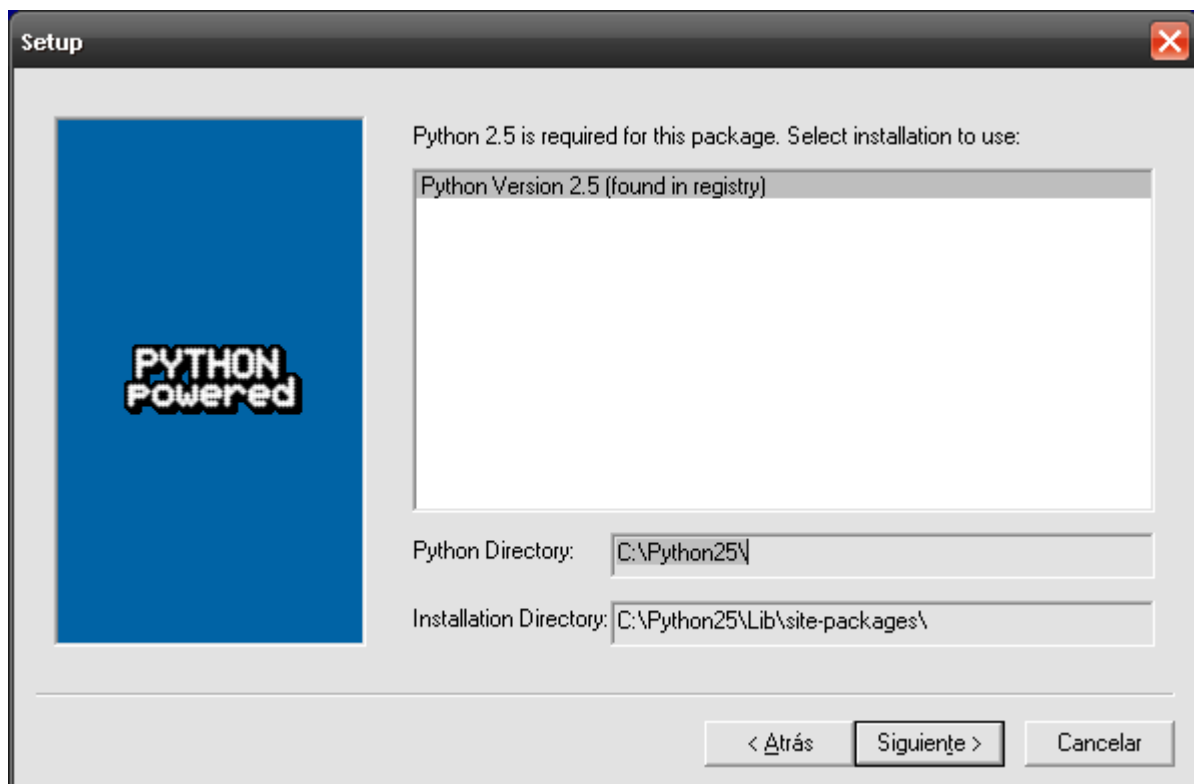
Paso 1. Ejecutar instalador de Networkx



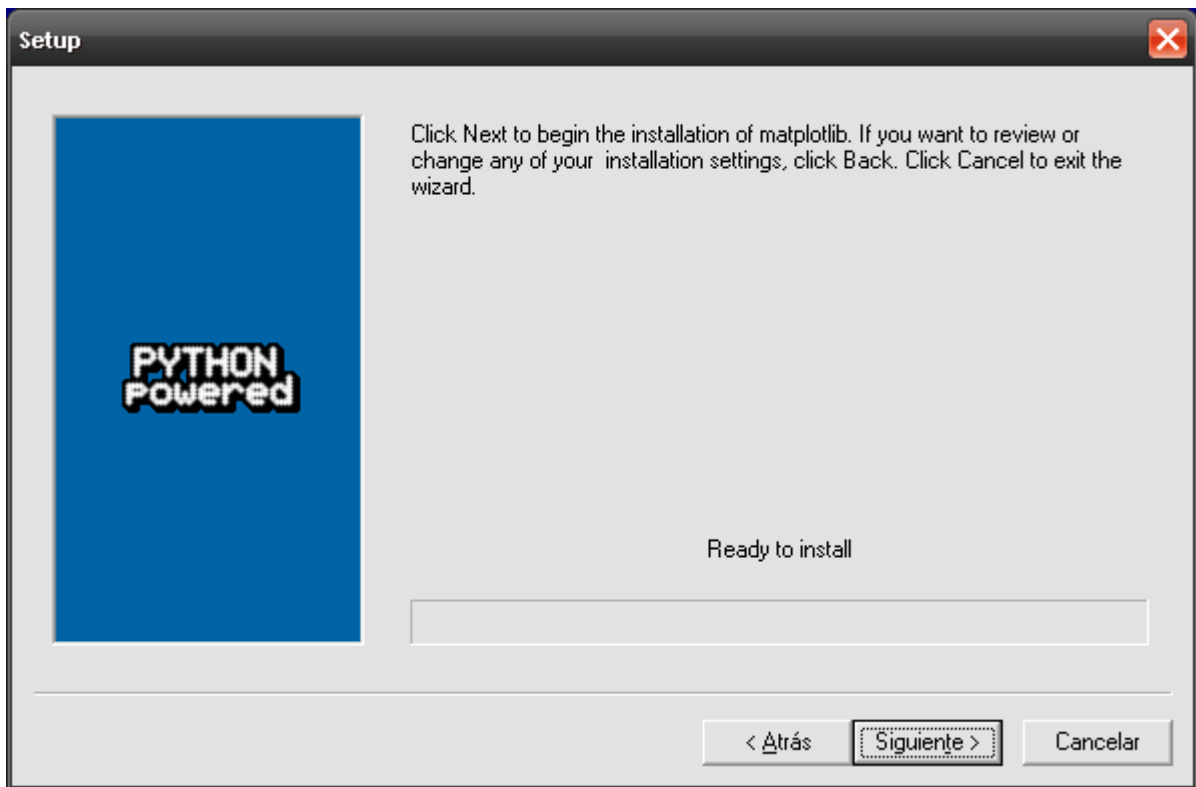
Paso 2. Iniciamos la instalación. Presionamos Siguiente.



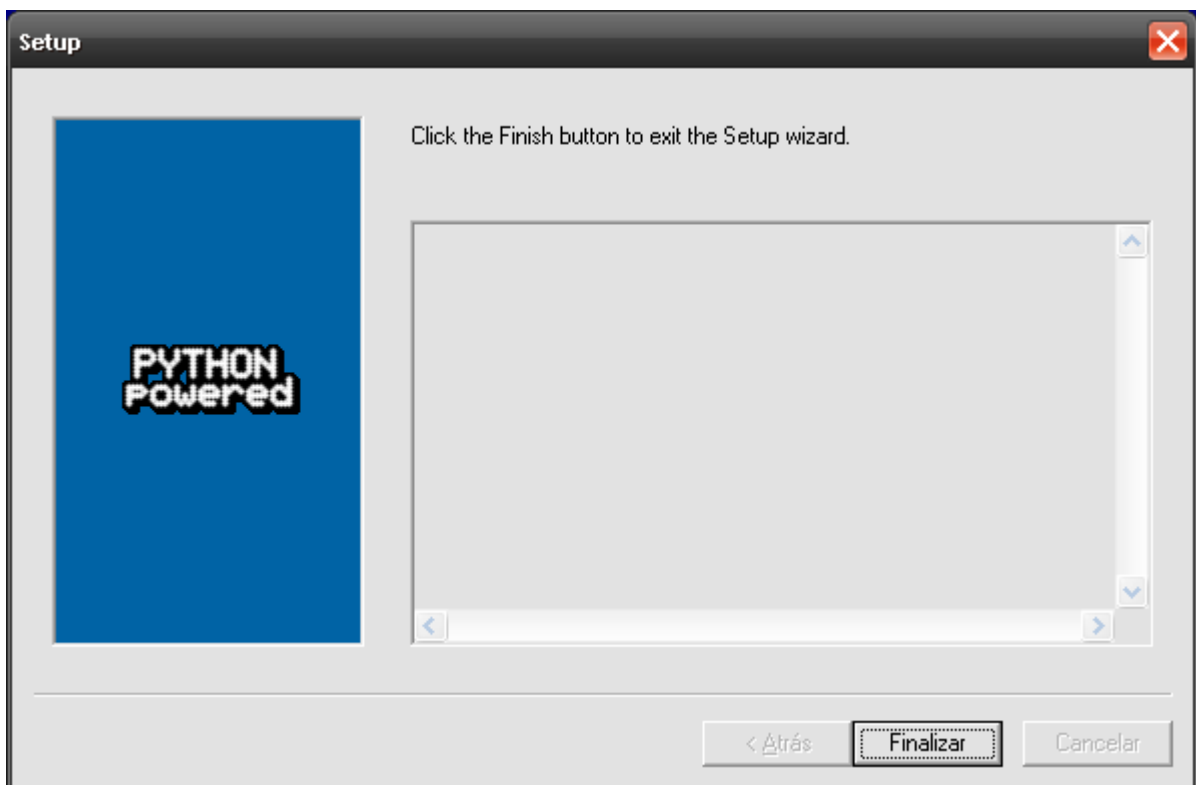
Paso 3. Elegimos directorio de instalación. Una vez seleccionado presionamos Siguiente.



Paso 4. Si todos los parámetros de la instalación son correctos, presionamos Siguiente y se iniciará instalación.



Paso 5. Si la instalación acaba correctamente veremos la siguiente pantalla.



INSTALAR CODIGO PFC

Paso 1. Descomprimir el fichero que contiene los códigos fuente de la aplicación (PFCEquilibrio.rar) en el directorio destino. Por ejemplo, C:\PFCEquilibrio

Paso 2. Descomprimir el fichero que contiene los juegos de prueba (jp2.rar) en el directorio C:\jp2.

CÓMO EJECUTAR

1. Para ejecutar usando el juego de pruebas implementado en los ficheros externos(jp2), lanzar el fichero `__init__.py`, que se encuentra dentro de la carpeta que contiene los códigos fuente. Por ejemplo: C:\PFCEquilibrio__init__.py
2. Para ejecutar usando el juego de pruebas de demanda elástica, lanzar el fichero **casoElastico.py**, que se encuentra dentro de la carpeta que contiene los códigos fuente. Por ejemplo: C:\PFCEquilibrio\casoElastico.py

En ambos casos, si la ejecución se ha realizado con éxito se generarán 3 ficheros de salida (dos de texto y una imagen) que contendrán los resultados. Estos ficheros se generarán dentro de la carpeta que contiene los códigos fuente (por ejemplo C:\PFCEquilibrio\).

Los ficheros generados tendrán la siguiente nomenclatura:

- PFCimagenResultado.png
- PFCresultados.txt
- PFCresultadosIteracion.txt

Para cualquier cambio en las rutas de ficheros o parámetros del sistema modificad el fichero ParametrosSistema.TXT. Si trabajáis sobre Unix, Linux , Ubuntu, etc... tened cuidado con el case sensitive y el formato de las rutas a los ficheros.

Para cualquier duda contactar con : jorge.olmeda@gmail.com

