

# 1.00 Clase adicional nº 7

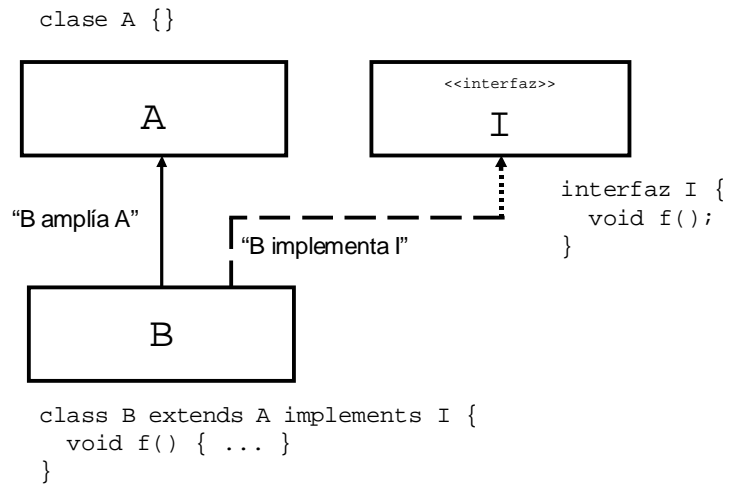
## Método de Newton, BP 6

1 de abril de 2002

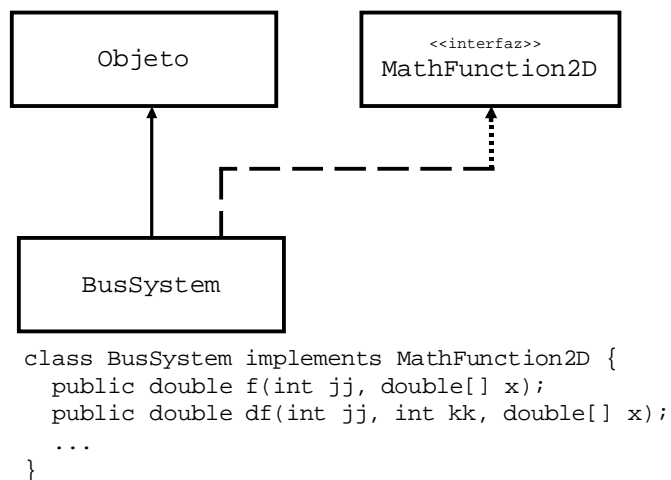
### Agenda

- Repasar dif. entre interfaces y clases
- BP nº 6: ¿qué estamos haciendo?
- Método de Newton en 1-D
- Método de Newton en 2-D
- Condiciones iniciales
- Clases e interfaces sugeridas
- Sugerencias

## Interfaces y clases

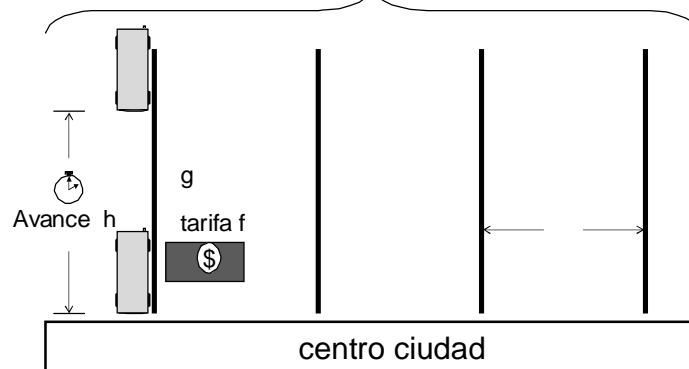


## Ejemplo del BP6



## BP6: ¿Cuál es el objetivo?

Rutas paralelas de autobuses con espacio equivalente (separados dist.  $g$ )

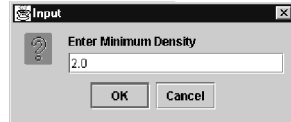


Dada una densidad de trayecto concreta  $p$ , ¿qué valores de  $h$ ,  $g$  y  $f$  maximizarán nuestro beneficio? (¿o minimizarán las pérdidas?)

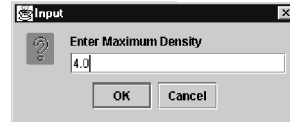
## Definiciones más importantes

- Densidad de trayecto  $p$ : número de trayectos por Km cuadrado y día
- Avance  $h$ : tiempo entre autobuses que viajan en la misma dirección y en la misma ruta (en minutos)
- $g$ : distancia entre rutas de autobuses (en millas)
- Tarifa  $f$ : precio del trayecto de autobús (en Centavos)

## E/S del programa



Input  
Enter Minimum Density  
2.0  
OK Cancel



Input  
Enter Maximum Density  
4.0  
OK Cancel

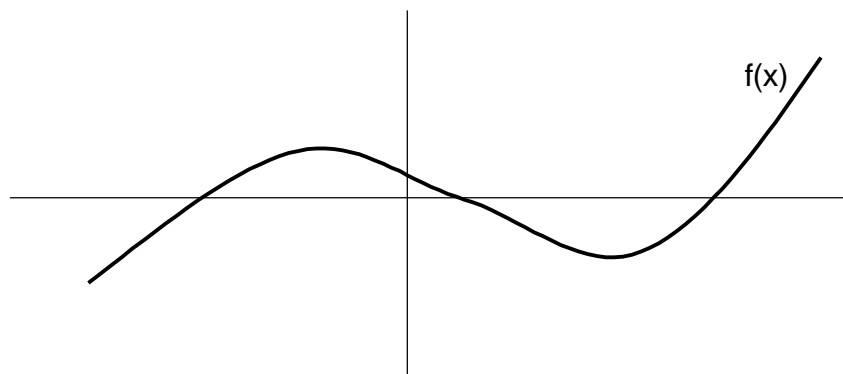
```
Density(p): 2.0
Approx. Route Spacing: 1.592529761260573
Approx. Fare: 100.35896047781914
Approx. Headway: 19.906622015757158
Route Spacing(g): 1.8540612181471507
Fare(f): 92.79322904645741
Headway(h): 23.17576522683938
Iterations: 6
Solution f0(g,f): -8.881784197001252E-16
Solution f1(g,f): -5.551115123125783E-17

Density(p): 2.5
...
```

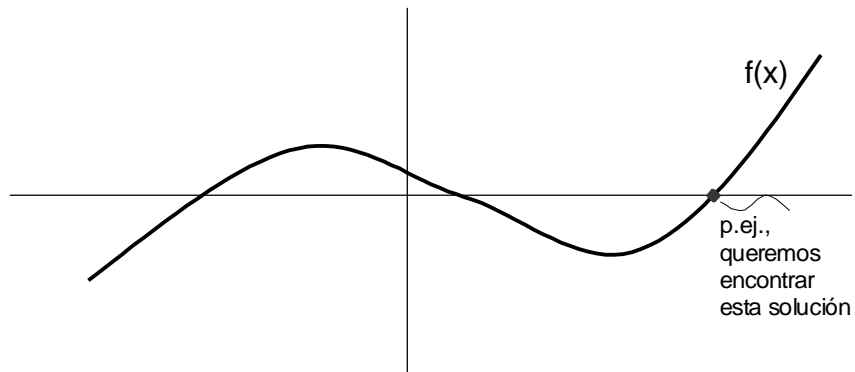
Solución aprox. de ecuaciones (7), (8) & (9)

Solución exacta (método Newton)

## Método de Newton (1D)

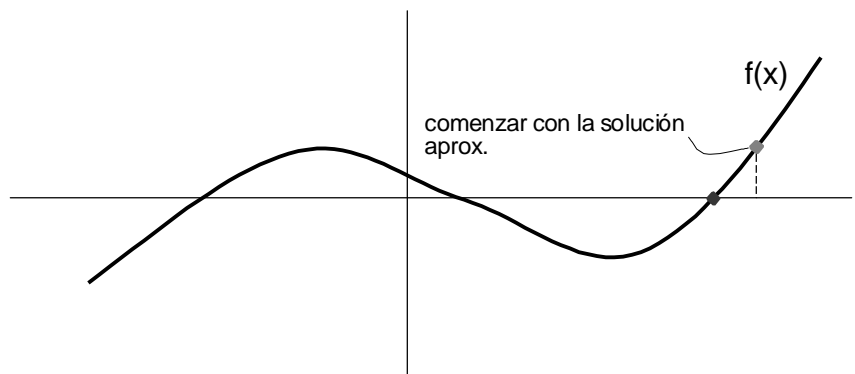


## Método de Newton (1D)



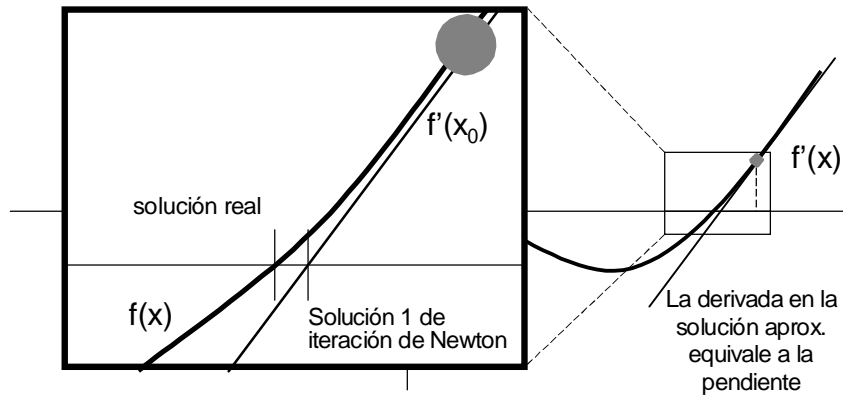
1. Comienza con una solución aproximada. (Esta función tiene 3 ceros. Evidentemente, necesitamos comenzar cerca de uno de ellos para converger en una solución concreta.)

## Método de Newton (1D)



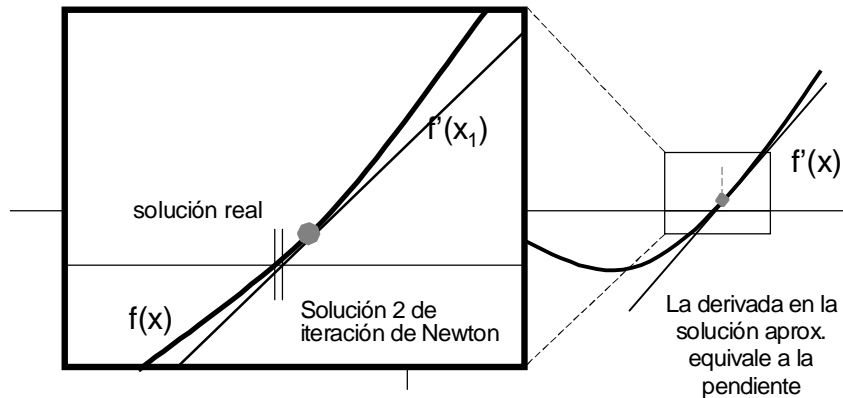
1. Comienza con una solución aproximada. (Esta función tiene 3 ceros. Evidentemente, necesitamos comenzar cerca de uno de ellos para converger en una solución concreta.)

## Método de Newton (1D)



2. Primero busca la derivada en la solución aproximada

## Método de Newton (1D)



3. Repite el proceso en la solución 1 de la iteración de Newton

## Método de Newton

4. Continúa hasta que se cumple lo siguiente:

- La solución está dentro de una tolerancia ( $10^{-15}$ ) de la solución real
- El número de iteraciones  $>$  algún máximo (50)
- La derivada se acerca a cero, es decir,  $\text{abs}(df(x)) \leq \text{TOLERANCIA}$

## Método de Newton: 2-D

- El método de Newton se puede ampliar a dimensiones mayores
- $n$  dimensiones se corresponden con un sistema de  $n$  ecuaciones de  $n$  variables
- En el boletín de problemas, tenemos 3 ecuaciones con 3 variables, pero dos son colineales y, por tanto, se pueden reducir a 2 ecuaciones con 2 incógnitas

## Condiciones iniciales (1)

- Valores iniciales para  $x_0$  y  $x_1$  de las ecuaciones (7) y (8)
- $x_0$  es  $g$  (distancia entre rutas)
- $x_1$  es  $f$  (tarifa de autobús)
- $h$  se puede calcular a partir de  $p$  (ec. (9))

## Condiciones iniciales (2)

- También probará con combinaciones de  $f = \{ 0.0, 0.00001, 10.0 \}$  y  $g = \{ 0.0, 0.00001, 10.0 \}$ . (Un total de 9 combinaciones.)
- En cada una, asegúrese de almacenar # iteraciones, con o sin solución convergente, así como la solución que se encontró (qué cero).
- Escriba los resultados y el análisis en sus comentarios.

## Método de Newton: 2-D

(g)

Las ecuaciones diferenciales parciales se proporcionan en el BP como (a)-(d).  $f_0$  es (0'') y  $f_1$  es (1'').

Para resolver las ecuaciones (j) a (l), utilice la regla de Cramer. Tenga en cuenta que está resolviendo para  $x_0$  y  $x_1$ , NO  $x_0$  y  $x_1$ . Asegúrese de que el código hace lo correcto.

## Interfaces

```
public interface MathFunction2D
{
    // jj=0 pretende calcular y devolver  $f_0$ 
    // jj=1 pretende calcular y devolver  $f_1$ 
    public double f(int jj, double[] x);

    // jj=0, kk=0 pretende devolver  $f_0/x_0$ 
    // jj=0, kk=1 pretende devolver  $f_0/x_1$ 
    // jj=1, kk=0 pretende devolver  $f_1/x_0$ 
    // jj=1, kk=1 pretende devolver  $f_1/x_1$ 
    public double df(int jj, int kk, double[] x);
};
```

## Clases sugeridas

```
// Incluye constantes y ecuaciones para
resolver. public class BusSystem amplía
MathFunction2D;

// implementa el método de Newton
public class Newton2D {
    public static double[] newt2D(MathFunction2D
                                  func, double[] x);
    ...
}

// para conservar el método main()
public class NewtonTest;
```

## NewtonTest (main)

- Entrada del usuario para density (p) mín y máx
- Para cada valor de p, calcula el resultado, p.ej., en un bucle
  - Crea un nuevo objeto bus con density p como argumento
  - Calcula la aproximación para un valor óptimo
  - Llama al método de cálculo de Newton `newt2D` (pasando el objeto bus y la aproximación inicial para la solución como argumento)
  - Muestra el resultado (puede agregar un método `getLastSolution` a `Newton2D` para recuperar el último resultado e imprimirlo)

## BusSystem

- Implementa `MathFunction2D` (métodos `f`, `df`)
- Método para obtener aproximaciones iniciales (`getRouteSpacing` y `getFare`), por lo que puede incluirlas en su método `Newton2D` para buscar las soluciones reales
- Métodos de establecimiento y obtención

## Newton2D

- Método estático `newt2D` (pasando las funciones y valores aprox. de  $x_0$  y  $x_1$  como argumentos, devuelve un `array double` como solución)
  - Repita el siguiente cálculo hasta que el valor absoluto de las dos funciones sea menor que la tolerancia
    - calcule el valor de  $y_0$ ,  $y_1$ ,  $a_{00}$ ,  $a_{01}$ ,  $a_{10}$ ,  $a_{11}$  (pasando  $x[]$  y sus índices correspondientes como argumentos)
    - Calcule  $\det (=a_{00} * a_{11} - a_{10} * a_{01})$
    - Vuelva a calcular  $x_0$ ,  $x_1$  con  $y_0$ ,  $y_1$ ,  $a_{00}$ ,  $a_{01}$ ,  $a_{10}$ ,  $a_{11}$  actualizados.  
**IMPORTANTE: puede utilizar  $x -= dx$  para encargarse del signo menos de la ecuación (g)**
    - Nota:  $y_0 = f_0(x_0, x_1)$ ,  $y_1 = f_1(x_0, x_1)$
- Otros métodos - de obtención (para almacenar el resultado del último `new2D`, esto es, `numberOfIterations` y la solución)

## Sugerencias

- Asegúrese de comprender el fundamento matemático antes de iniciar el código. Analícelo con su compañero. Intente escribirlo a mano.
- Utilice notación “punto cero” para los doubles. Es decir, escriba “1.0” en lugar de “1” ó “3.0” en lugar de “3”.