Imprimir
Note: You can choose to save the module's content as a PDF document, instead of printing.
To do so, click on the 'Print' link above. In the window that appears, select the PDF
printer as the printer of choice, then click 'Print' and enter any additional information needed.

Resolución de ecuaciones no lineales. Ceros y raíces

Introducción

Instructions: Seguir los enlaces al material. También hay parte del material colgado en "Recursos".

Introducción

En este capítulo pretendemos dar las nociones básicas de resolución numérica de ecuaciones no lineales. Empezamos con la descripción de como introducir funciones. Después recordamos tres de los métodos más básicos para resolución numérica de ecuaciones, que son el de bisección, Newton, y el de la secante. En ellos describimos sus ventajas y desventajas, así como su implementación en MATLAB para bisección y secante. La sección principal es la correspondiente al comando por defecto de MATLAB, comando fzero, para la resolución de ecuaciones. Este método combina la seguridad del método de bisección con la rapidez de otros.

Resolución numérica de ecuaciones no lineales con MATLAB

Instructions: Seguir la lección y enlaces.

Resolución numérica de ecuaciones no lineales con MATLAB

Introduciendo la función y dibujando su gráfica

El primer paso a considerar es la introducción de la función que define la ecuación no lineal $f(x) = 0$, y la cual resolveremos numéricamente. Para ello podemos utilizar el comando **inline**:

```
>> f=inline('exp(x)-x.^2');
```

De esta manera hemos introducido la función $f(x) = e^x - x^2$ y nos aparecerá en el workspace. Según aparece en la página de Mathworks, el comando inline desparecerá en futuras versiones de MATLAB. Lo que se conoce como funciones anónimas es lo que permanecerá. Así la función anterior se puede introducir alternativamente como:

```
>> f=@(x) exp(x)-x.^2;
```

Si es una función que vamos a utilizar en más sesiones, se recomienda guardarla en un fichero:

```
function y = f(x)
y = exp(x)-x.^2;
```

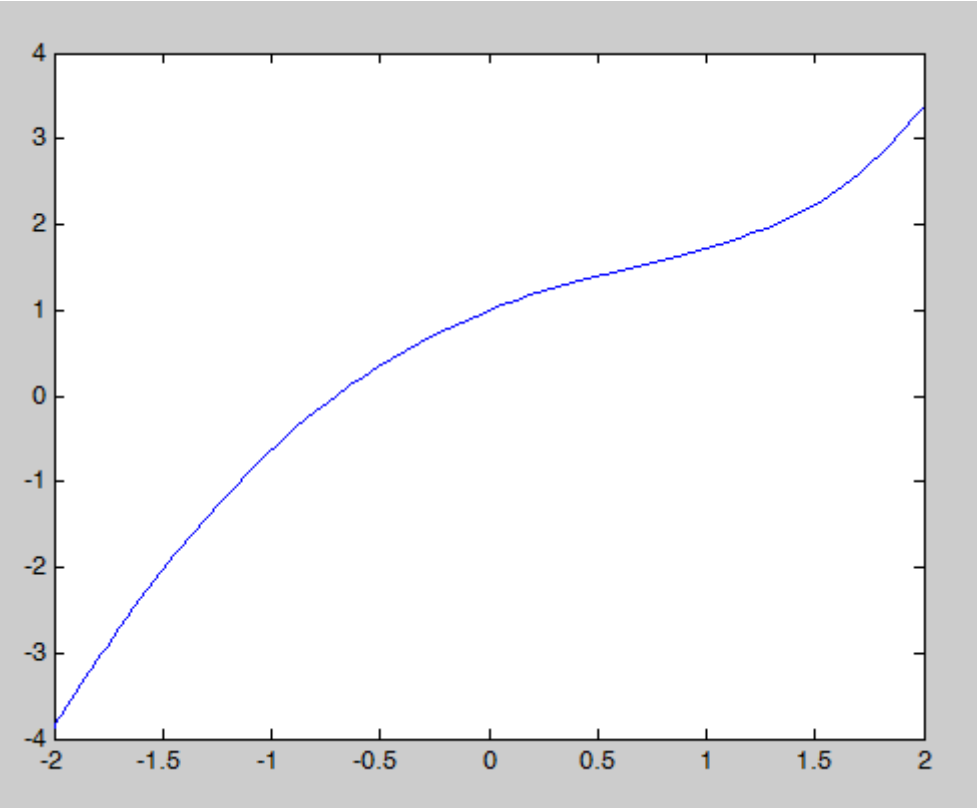
Esta función tiene un cero (raiz) entre -1 y 0. Podemos verificarlo mediante el estudio de su gráfica, utilizando el comando **fplot**:

```
>> fplot(f, [-2 2])
```

Lo anterior sería para una función en el workspace. Si pretendemos utilizar una función en un fichero del path, hay que preceder el nombre de la función con el símbolo @:

```
>> fplot(@f, [-2 2])
```

La gráfica obtenida es la siguiente:



Métodos básicos de resolución de ecuaciones

Importante: Para visualizar en este tema un número grande de cifras significativas al ejecutar las órdenes en Matlab, conviene utilizar el formato largo despues de iniciar sesión mediante

```
>> format long
```

Las variables almacenadas siguen teniendo la misma precisión, pero en pantalla veremos más cifras significativas cuando nos las tenga que mostrar.

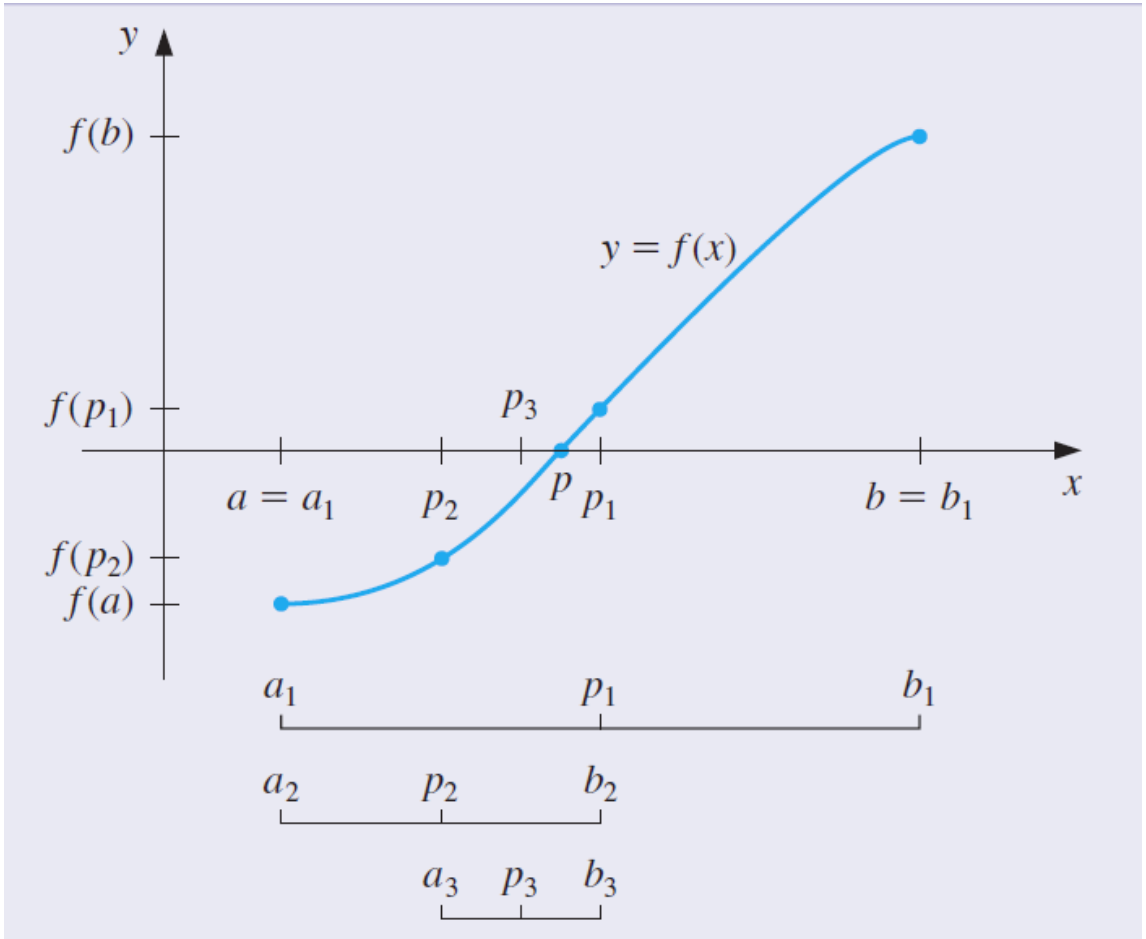
El método más básico para la resolución de ecuaciones es el método de bisección. Es un método lento (de convergencia lineal), pero seguro. Si queremos resolver la ecuación $f(x) = 0$ en el intervalo $[a, b]$, donde f es una función continua tal que $f(a)f(b) < 0$, realizamos la primera iteración tomando

$$p_1 = \frac{a + b}{2}$$

Sean $a_1 = a$ y $b_1 = b$. Puede ocurrir que $f(a_1)f(p_1) < 0$, en cuyo caso tomariamos $a_2 = a_1$ y $b_2 = p_1$, o bien $f(p_1)f(b_1) < 0$, donde llamaríamos $a_2 = p_1$ y $b_2 = b_1$. Ahora volveríamos a tomar el punto medio

$$p_2 = \frac{a_2 + b_2}{2}$$

y así sucesivamente. A continuación tenemos una ilustración gráfica del método:



Como hemos dicho este método es lento, ya que en cada paso reducimos a la mitad el intervalo donde se encuentra la raíz (equivalente a un bit por paso), pero nos aseguramos su convergencia a una raíz, independientemente de la longitud del intervalo inicial, siempre que la función tome signos distintos en sus extremos. Este método se puede programar fácilmente en MATLAB. Tenemos que prefijar un criterio de paro (por ejemplo, un valor positivo epsilon tal que el programa para cuando la distancia entre los extremos del intervalo es inferior a dicho valor), la función cuya raíz queremos encontrar (introducida como hemos comentado en la primera sección), y los extremos del intervalo inicial, a y b:

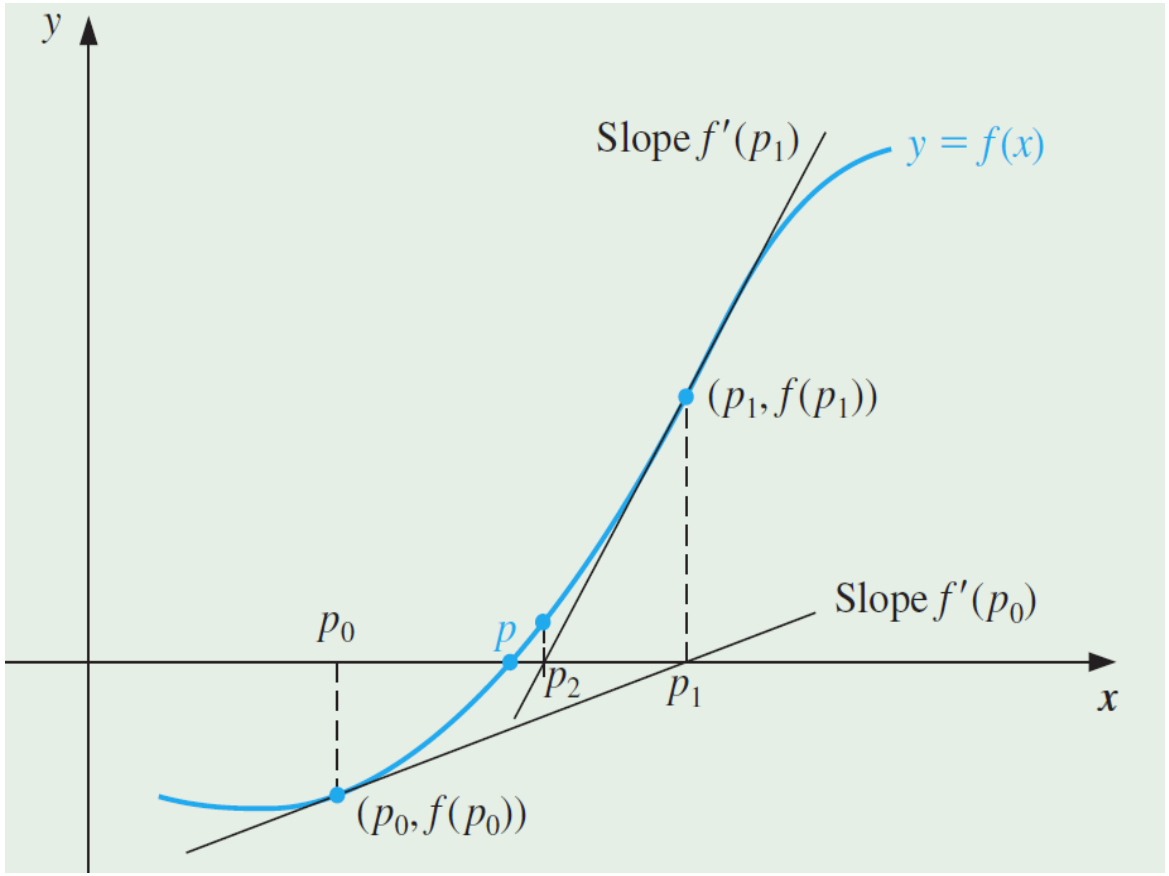
```
k = 0;
while abs(b-a) > epsilon
    x = (a + b)/2;
    if sign(f(x)) == sign(f(b))
      b = x;
```

```
    else
      a = x;
    end
    k = k + 1;
end
```

Con el programa anterior además contabilizamos mediante k el número de pasos realizados antes del paro.

Otro método muy conocido, y mucho más rápido (de orden cuadrático) que el de bisección, es el método de Newton (también conocido como Newton-Raphson). Como inconveniente está el tener que realizar el cálculo de la derivada, y que su convergencia no está asegurada de forma global. Aqui partimos de un punto inicial p_0 suficientemente cercano a la raíz, y suponemos que la derivada no se anula en un entorno de la raíz que contiene a dicho punto inicial. Trazamos la tangente a la gráfica de f en el punto $(p_0, f(p_0))$ y encontramos un punto de corte con el eje x, el cual pasa a ser el siguiente punto p_1 de la iteración. Analíticamente, cada punto se obtiene a partir del anterior mediante la siguiente fórmula:

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}$$



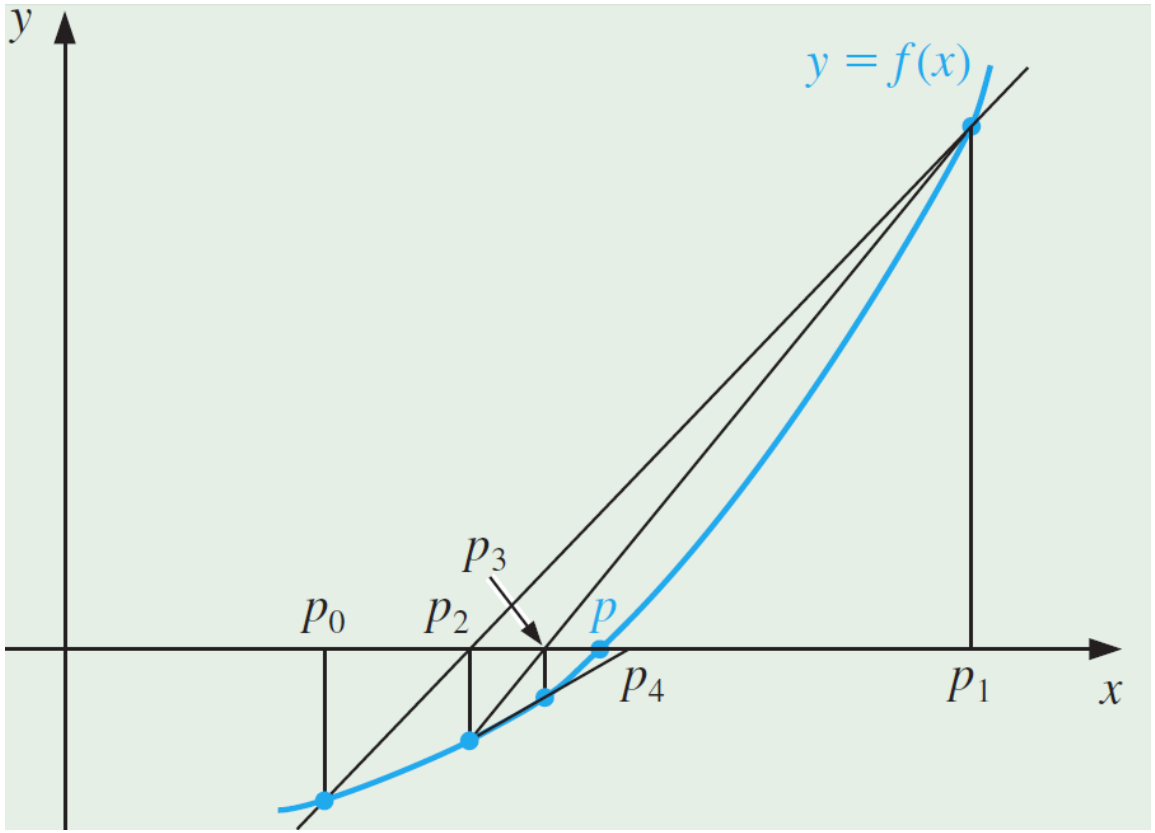
El tercer método bien conocido que recordamos es el de la secante. Necesitamos 2 valores iniciales, p_0 y p_1, contruimos la recta que pasa por $(p_0, f(p_0))$ y por $(p_1, f(p_1)$, la cual corta al eje x en un punto p_2. Repetimos la operación con p_1 y p_2, y así sucesivamente. La fórmula es la siguiente:

$$p_{n+1} = p_n - \frac{f(p_n)}{s_n},$$

siendo

$$s_n = \frac{f(p_n) - f(p_{n-1})}{p_n - p_{n-1}}$$

Por supuesto debemos estar bajo la hipótesis que $f(p_n) \neq f(p_{n-1})$ (o lo que es lo mismo, que la recta tangente no sea paralela al eje x). Aquí tenemos una ilustración gráfica:



La convergencia de este método es casi tan rápida como el de Newton. A continuación indicamos su programación en MATLAB. También aqui tenemos que prefijar un criterio de paro (ponemos, por ejemplo, un valor positivo epsilon tal que el programa para cuando la distancia entre dos iteraciones consecutivas es inferior a dicho valor), la función cuya raíz queremos encontrar, y los valores iniciales, a=p_0 y b=p_1:

```
k = 0;
while abs(b-a) > epsilon
   c = a;
   a = b;
   b = b + (b - c)/(f(c)/f(b)-1);
```

```
    k = k + 1;
end
```

El comando fzero de MATLAB

En MATLAB hay preprogramado un comando, **fzero**, para resolver numéricamente ecuaciones. Es un método muy efectivo ya que combina un método seguro, como el de bisección, con uno rápido cuando es posible aplicarlo, parecido al de la secante, aunque más rápido ya que interpola cuadráticamente 3 valores, en vez de interpolación lineal de 2 valores. Una descripción detallada del método se puede consultar en el capítulo correspondiente de Cleve Moler

Al utilizar el comando con la función anterior $f(x) = e^x - x^2$, que tiene un cero entre -1 y 0, podemos dar un valor inicial cercano, como -0.5:

```
>> fzero(f,-0.5)

ans =

  -0.703467422498392
```

Inicialmente el algoritmo trata de encontrar un intervalo que contenga a dicho valor inicial, y donde la función tenga signo distinto en los extremos. También podemos poner en vez de un valor inicial un intervalo con esas carácterísticas:

```
>> fzero(f,[-1 0])

ans =

  -0.703467422498392
```

Podemos obtener una información completa de todo lo realizado si especificamos:

```
>> [X,FVAL,EXITFLAG,OUTPUT] = fzero(f,-0.5)

X =

  -0.703467422498392

FVAL =
```

```
        0

    EXITFLAG =

        1

    OUTPUT =

          intervaliterations: 9

                  iterations: 5

                   funcCount: 24

                   algorithm: 'bisection, interpolation'

                    message: 'Zero found in the inter...'
```

Primero busca un intervalo alrededor del valor inicial donde la función tenga valores distintos en los extremos (9 iteraciones), despues itera el algoritmo (5 iteraciones), con funcCount indica el número de evaluaciones de funciones realizadas, el valor de X que da es donde encuentra la raiz, con FVAL determina el valor de la función en la raiz (que es 0), también en algorithm describe que ha iniciado con bisección y luego la interpolación cuadrática.

Actividades propuestas

Actividades

¿Qué tengo que hacer?

IMPORTANTE: No pases a las siguientes unidades hasta no haber sido capaz de completar con éxito estas actividades.

Las actividades propuestas para esta unidad consisten en la resolución con MATLAB de algunas ecuaciones no lineales. En alguna de ellas hay que comparar el algoritmo de MATLAB fzero con otros métodos.

Actividad 1:

Resolver la ecuación $x^3 - 2x - 5 = 0$, que tiene una única raiz real, mediante el comando fzero. Previamente ejectuta 'format long'. Muestra toda la información completa y observa que valor aparece en FVAL.

Podemos mejorar la precisión del algoritmo cambiando TolX. Para ello realiza:

```
>> miopcion=optimset('TolX',epsilon)
```

Donde epsilon es el valor de tolerancia que tú especifiques. A continuación:

```
>> [X,FVAL,EXITFLAG,OUTPUT] = fzero(f, [2 3], miopcion)
```

Siendo f el polinomio que ya habrás introducido. ¿Cambia la solución mostrada?

Sobre el mismo polinomio, prueba a utilizar e método de bisección y el de la secante, ambos con intervalo inicial $[2, 3]$, y compara las iteraciones utilizadas.

Actividad 2:

Averigua las 3 raices reales del polinomio $816x^3 - 3835x^2 + 6000x - 3125$. Para ello se recomienda utilizar previamente el comando fplot en el intervalo adecuado (posiblemente haya que realizar varios intentos para llegar a un intervalo suficientemente pequeño. A continuación utiliza el comando fzero con los valores iniciales adecuados.

Actividad 3:

En todos los casos se supone que la función es continua. Por comprobar que ocurre con un ejemplo que no lo sea, utiliza el comando fzero para la función $tan(x)$ con valor inicial 1, y mostrando la información completa. Otro problema distinto es con las raices múltiples. Toma el polinomio $(x - 1)^7$ y aplica el comando fzero con valor inicial, por ejemplo, 1.5 y comprueba el número de iteraciones necesario.

Bibliografía

Instructions: El capítulo correspondiente, de acceso libre, está en la carpeta de recursos.

Cleve B. Moler, Numerical Computing with MATLAB. Chapter 4.