

Prácticas de Matemáticas I – Sesión V
Ingeniería Técnica Industrial con Intensificación
en Diseño Industrial
UPNA - Campus de Tudela

VÍCTOR DOMÍNGUEZ BÁGUENA

Diciembre 2009

Prácticas Matlab 1

Sesión 5: Métodos de Bolzano, Newton y Secante

1.1. Introducción

El ánimo de esta práctica es profundizar someramente en el lenguaje propio de Matlab, manejando las estructuras esenciales de programación: bucles (`for`) y decisión (`if`). No entraremos en aspectos más profundos como la programación de funciones y subrutinas.

Como piedra de toque implementaremos tres métodos para la resolución de ecuaciones no lineales: método de bisección, Newton y secante.

1.2. Aspectos instrumentales: bucles y estructuras de decisión

1.2.1. Bucles: el comando `for`

En Matlab, la estructura

`for`

```
for j=inicio:paso:final
.....
end
```

implementa un bucle donde las líneas de código entre `for` y `end` son ejecutadas repetidamente con `j` tomando secuencialmente los valores del vector `inicio:paso:final` Por ejemplo,

```
>> for j=1:3; disp(j); end
1
2
3
>> for j=6:-2:2; disp(j-2); end
4
2
0
```

En general, si v es un vector,

```
for j=v
.....
end
```

procede a ejecutar las líneas internas, comprendidas entre `for` y `end`, con j tomando los valores del vector v . La variable j es una variable, por tanto, índice. Por ejemplo,

```
>> v=[2 5 3 1];for j=v; disp(j); end
2
5
3
1
```

Asociados a la instrucción `for`, y en general a cualquier bucle, encontramos los comandos `break` y `continue`. Con `break` se fuerza la salida inmediata del bucle. Por otro lado, `continue` provoca el fin de la ejecución con el valor actual de la variable índice y que se recomience el bucle con el siguiente valor

`break`
`continue`

1.2.2. Operadores lógicos y estructuras de decisión

Los operadores lógicos, operadores de comparación, más básicos en Matlab son

```
== igualdad,      ~= desigualdad,    > mayor,    < menor,
>= mayor o igual, <= menor o igual, & "y" lógico, | "o" lógico
```

El resultado de una comparación es 1 si es verdadero, 0 si es falso:

```
>> a=1; b=2; c=3;
>> a>0
ans =
    1
>> a>0 & b<3
ans =
    1
>> a<0 | b<1
ans =
    0
```

La estructura básica de decisión, en común con muchos otros lenguajes, es `if`. Su sintaxis es la siguiente:

`if`

- `if` simple: si la operación lógica efectuada es verdadera, se ejecutan las líneas de código comprendidas entre `if` y `end`

```
if (x<-1 | x>1)
    disp('valor absoluto de x mayor que 1')
end
```

- if compuesto: como el anterior, pero un nuevo conjunto de líneas, comprendidas entre `else` y `end` son ejecutadas en caso de que la operación lógica efectuada en el `if` sea falsa:

else

```

if (x<0 & x>-10)
    f=x^2;      % x entre -10 y 0
else
    f=sin(x^2); % x menor que -10 o mayor que 0
end

```

- if de decisión múltiple:

```

if (x<0)
    f=x.^2;
    disp('x menor que cero')
elseif (x<sqrt(pi))
    f=sin(x^2);
    disp('x en [0,sqrt(pi))')
elseif (x<2*sqrt(pi))
    f=(x-sqrt(pi))^2;
    disp('x en [sqrt(pi),2*sqrt(pi))')
else
    f=pi*cos(x^2);
    disp('x en [2*pi,infinito)')
end

```

Esta última instrucción es equivalente a anidar diferentes estructuras `if`, de la siguiente forma

```

if (x<0)
    f=x.^2;
    disp('x menor que cero')
else
    if (x<sqrt(pi))
        f=sin(x^2);
        disp('x en [0,sqrt(pi))')
    else
        if (x<2*sqrt(pi))
            f=(x-sqrt(pi))^2;
            disp('x en [sqrt(pi),2*sqrt(pi))')
        else
            f=pi*cos(x.^2);
            disp('x en [2*pi,infinito)')
        end
    end
end
end

```

Obviamente, la primera forma es más clara y concisa.

Está disponible también el bucle `while` que procede a ejecutar un bucle (que se cierra también con un `end`) mientras una condición sea verdadera. Por tanto, es algo más flexible que un `for`.

`while`
`rand`

Por ejemplo, el siguiente código

```
aux=0;
while aux<0.8
    aux=rand(1);
    disp(aux)
end
```

muestra en pantalla una sucesión de números aleatorios entre 0 y 1 mientras éstos sean menores que 0.8.

En general todos los bucles y estructuras que requieran *cerrarse*, lo hacen con `end`. El uso de `break` y `continue` es exactamente el mismo.

Nota Aunque estos comandos se pueden utilizar en el modo comando, su ámbito natural consiste en ficheros *script* y en funciones programadas en ficheros de texto (aspecto que no hemos visto en estas prácticas). En general Matlab se puede programar, y de hecho se hace, como un lenguaje tradicional como Pascal, C o Fortran aunque tienes sus peculiaridades dado que está enfocado hacia el cálculo numérico.

1.3. Métodos numéricos para la resolución de ecuaciones no lineales

En lo que sigue veremos tres métodos para la resolución de ecuaciones de la forma

$$f(x) = 0$$

donde f es una función en principio arbitraria. Se habla entonces de encontrar los **ceros** o las **raíces** de la función $f(x)$.

1.3.1. Método de bisección

El método de bisección está basado en el Teorema de Bolzano. Se fundamenta en el resultado que afirma que si una función continua toma valores opuestos en los extremos de un intervalo entonces necesariamente se anula en un punto interior. La idea es entonces construir subintervalos de longitud cada vez menor que contengan al cero¹.

La forma de proceder es la siguiente:

1. Empezamos con un intervalo $[a, b]$ y una función f que toma valores con signos opuestos en a, b

¹De hecho así es como se prueba el Teorema de Bolzano.

2. Se calcula $c = (a + b)/2$.
3. Se compara los signos de $f(a)$, $f(b)$, $f(c)$.
 - si $f(a)$ y $f(c)$ tiene signos opuestos se toma $b = c$.
 - si $f(a)$ y $f(c)$ tiene el mismo signo se toma $a = c$.
4. Se vuelve al punto 1.

Resta por fijar algunos detalles más como el criterio de parada (¿cuándo una solución aproximada se considera suficientemente buena?) o un control sobre el número máximo de iteraciones.

El algoritmo, escrito en un lenguaje más próximo al del ordenador es como sigue:

Método de Bisección

```

Datos de entrada: a,b, f,eps, nmax
fa=f(a), fb=f(b)
Para i=1,2,...,nmax
  c=(a+b)/2
  fc=f(c)
  si abs(fc)<eps entonces
    break
  fin si
  si fa*fc<0 entonces
    fb=fc
    b=c
  sino
    fa=fc
    a=c
  fin si
fin para

```

Observa que $nmax$ es el máximo número de iteraciones permitido mientras que si $|f(z)| < eps$, se entiende que z es la raíz de f .

En implementaciones prácticas el paso más caro es evaluar $f(x)$. En este sentido el algoritmo anterior recicla los valores anteriores posibles para reducir al máximo nuevas evaluaciones de la función. Observa que sólo hay una evaluación $f(c)$ por cada iteración dado que $f(a)$ y $f(b)$ son conocidas en la iteración anterior. Insisteremos en este hecho en los métodos siguientes.

1.1 Implementa en Matlab el método de bisección. Utiliza para ello bien un bucle for-end con break o bien una estructura de tipo while. Aplica el método para

calcular las soluciones de

$$x^2+4x = 7, \quad \cos(x) = x, \quad e^x - x = 0, \quad \arctan(x) = 1, \quad x^4 = \pi x - 1.$$

Elige cuidadosamente el intervalo de inicio del método

Nota La implementación se puede hacer en un fichero script. Así por ejemplo, esta sería la cabecera para el primer ejemplo

```
f=vectorize(inline('x^2+4*x-7','x'))
a=0; b=2;
eps=1e-6; nmax=100
.....
.....
```

Observa que puedes cambiar a mano los valores de **f** (la función), **a, b** (el intervalo donde se empieza la búsqueda), **eps** (el criterio de parada) y **nmax** (número máximo de iteraciones). Es posible, si así lo deseas, que se desplieguen por pantalla los valores de **a, b** en cada paso.

Otra forma distinta sería utilizar **input** como forma de introducir los valores iniciales.

1.3.2. Método de Newton

Es un método más potente para la resolución de ecuaciones no lineales. El precio que se paga es, que por un lado, requiere que la función sea más regular (requiere que la función sea derivable) y por otro lado una pérdida del carácter general de convergencia: el método converge habitualmente sólo si se arranca desde un punto situado suficientemente cerca de la raíz. Por ello suele ser habitual utilizar este método en combinación con un método menos potente pero más seguro. Así por ejemplo, el método de Bisección permite acercarse a la raíz, de forma que tras algunos pasos, *lanzar* el método de Newton con la aproximación dada por el método de Bisección. Aún así, no se asegura la convergencia del método. Por ejemplos es esperable que haya problemas si la función no es derivable en el cero de la función. Si por contra la derivada se anula en la raíz (cero doble), el método converge pero la velocidad de convergencia es mucho menor, comparable de hecho al del método de Bisección.

El método de Newton viene dado por el siguiente esquema: dado x_0 , aproximación inicial de la solución, definir

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \dots$$

De forma más detallada, éste es un algoritmo válido para el método.

Método de Newton

```

Datos x0,f,eps, nmax
Define fprima con la derivada de f
Hacer fx0=f(x0)
Para i=1,2,...,nmax
    x0=x0-fx0/fprima(x0)
    fx0=f(x0)
    si abs(fx0)<eps entonces
        break
    fin si
fin para

```

El algoritmo anterior no guarda los valores de la sucesión, simplemente el último valor calculado. Observa que cada paso del método de Newton requiere dos evaluaciones, una de la función y otra de su derivada.

1.2 Implementa el método de Newton. Utilízalo para calcular las soluciones de

$$\begin{aligned} \cos(x) = x, \quad 2^x = 7, \quad \sqrt{x} - \ln(x) = 2 \quad x^2 - 6x = 9, \\ \arctan(e^{\cos(x)}) = 3x \log(5 + \cos(x)), \quad \sqrt[3]{x} = 0 \end{aligned}$$

1.3.3. Método de la Secante

El Método de la Secante trata de solventar una de los principales inconvenientes del Método de Newton, concretamente que éste requiere conocer la derivada de f . En situaciones prácticas, es posible que la derivada de f no se pueda calcular, o bien que su evaluación resulte demasiado complicada (computacionalmente).

El Método de la Secante consiste en el siguiente esquema: dado x_0, x_1 dos valores suficientemente cerca de la raíz, definir

$$x_{n+2} = x_{n+1} - \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)} f(x_{n+1}), \quad n = 0, 1, \dots$$

Observa que a diferencia del método de Newton, se necesitan ahora dos aproximaciones iniciales, x_0 , y x_1 , para arrancar el método.

A continuación se expone un algoritmo sobre el que se podría trabajar para su implementación en el ordenador.

Método de la Secante

```

Datos x0,x1,f,eps, nmax
Hacer fx0=f(x0), fx1=f(x1)
Para i=1,2,...,nmax
    x2=x1-(x1-x0)/(fx1-fx0)*fx1

```

```

    fx2=f(x2)
    if abs(fx2)<eps entonces
        break
    fin si
    Hacer x0=x1, fx0=fx1
    Hacer x1=x2, fx1=fx2
fin para

```

Observa que con esta implementación cada iteración precisa únicamente de una evaluación de la función f ya que algunos valores de la iteración anterior son reciclados. Esto da lugar a un curioso fenómeno. El algoritmo pierde algo de su potencia (menor orden de convergencia) cuando se compara con el método de Newton, es decir, requiere de mayor número de iteraciones pero cada iteración es más rápida de ejecutar dado que sólo hay una evaluación nueva de la función por dos del Método de Newton (la función y su derivada).

De nuevo para arrancar el método es necesario que x_0 y x_1 estén cerca de la solución.

1.3 Implementa el método de la secante. Utilízalo para calcular las soluciones de las siguientes ecuaciones

$$\cos(x) = x, \quad 2^x = 7, \quad \sqrt{x} - \ln(x) = 2, \quad x^2 - 6x = 9,$$

$$\arctan(e^{\cos(x)}) = 3x \log(5 + \cos(x))$$