Cálculo con **MatLab**

Curso 2010/2011

Práctica 2: CÁLCULO DIFERENCIAL

1 LíMITES

El concepto de límite es la base del Cálculo Diferencial. La librería Symbolic Math Toolbox de **MatLab** permite calcular límites de funciones directamente mediante el comando **limit**, que sigue el formato

limit(f,x,a)

para calcular el límite de la función f cuando la variable x tiende al punto a.

Si f es una función de una única variable, no es necesario especificarla en el formato, pudiéndose usar

limit(f,a) o bien **limit(f)** que equivale a **limit(f,0)**

Para el cálculo de límites laterales se puede utilizar la opción correspondiente:

limit(f,x,a,'left')

limit(f,x,a,'right')

Se puede utilizar la capacidad del comando **limit** para actuar sobre vectores a la hora de calcular el límite de varias funciones en un mismo punto.

E j e m p l o s :

```
>> syms x
>> limit((1+1/x)^x,x,inf)
ans =
exp(1)
>> syms t, limit((1+t)^(1/t))
ans =
```

```
exp(1)
>> syms x,limit([1/x^2,sin(x)/x,log(x)],x,0,'right')
ans =
[ Inf, 1, -Inf]
```

Ejercicio 1.- Obtén la derivada de la función logaritmo neperiano recurriendo a la definición de derivada y utilizando el comando limit.

Ejercicio 2.- Utiliza el comando limit para calcular los límites laterales de la función tangente en el punto π/2. ¿Qué responde MatLab si intentamos calcular directamente el límite de la tangente en ese punto?

Ejercicio 3.- Estudia la existencia de asíntotas horizontales y verticales de $f(x) = \frac{3x^2+6x-1}{x^2+x-3}$ **y representa gráficamente la función f junto con sus asíntotas.**

2 DERIVACIÓN

2.1 Comando diff

El paquete Symbolic Math Toolbox de **MatLab** permite realizar las operaciones de derivación e integración simbólicas. En particular, el comando **diff** de **MatLab** permite calcular derivadas de una expresión algebraica o de una función de una o más variables. Ejecutando una orden que siga uno de los siguientes formatos,

diff(funcion,variable,k)

diff('expresion',variable,k)

obtendremos la k-ésima derivada de la función o expresión que tecleemos respecto de la variable introducida en segundo lugar.

Si no se indica en tercer lugar un número entero k, **MatLab** considera k=1.

Si no hay posibilidad de confusión en cuanto a la variable, podemos omitirla.

Nota.- También funcionan los formatos siguientes:

diff(funcion,k,variable)

diff('expresion',k,variable)

E j e m p l o

Supongamos que queremos hallar las derivadas no nulas del polinomio

$f(x)=ax^3+bx^2+cx+d$.

Podemos hacerlo como sigue:

```
>> syms a b c d x
>> f=a*x^3+b*x^2+c*x+d
f =
a*x^3+b*x^2+c*x+d
>> f1=diff(f)  %sin especificar la variable
f1 =
3*a*x^2+2*b*x+c
```

```
>> f1=diff(f,x)  %también calcula la derivada primera
f1 =
3*a*x^2+2*b*x+c
>> f2=diff(f,x,2)  %derivada segunda
f2 =
2*b+6*a*x
>> f3=diff(f,x,3)   %derivada tercera
f3 =
6*a
>> diff(f,3)  %lo mismo sin indicar la variable, el programa considera que es x
ans =
6*a
>> diff(f,x,4)  %a partir de ésta, las sucesivas derivadas son nulas
ans =
0
>> fb=diff(f,b) %derivamos respecto del segundo coeficiente
fb =
x^2
```

Observación.- Para derivar una constante, debemos primero definir la constante como una expresión simbólica. Por ejemplo,

```
>> k=sym('7');
```

```
>> diff(k)

ans =

0
```

Si intentamos derivar directamente el programa responde con un doble corchete:

```
>> diff(7)

ans =

[ ]
```

La operación de derivación, como otras, se puede extender a vectores (y también a matrices). Si pedimos la derivada de un vector respecto de una variable, Matlab calcula otro vector cuyos elementos son las derivadas, respecto de la variable indicada, de los elementos del vector dado. Por ejemplo:

```
>> syms x y

>> [2*x*y^2 7 x/y x-y]

ans =

[ 2*x*y^2, 7, x/y, x-y]

>> diff(ans,y)

ans =

[ 4*x*y, 0, -x/y^2, -1]
```

Recordemos que si la función que queremos derivar es polinómica, también podemos usar el comando **polyder**, que actúa sobre un polinomio escrito en forma de vector (teniendo en cuenta que **MatLab** identifica un polinomio con el vector de sus coeficientes escritos en orden decreciente, en cuanto a las potencias de x).

E j e m p l o s

En los siguientes ejemplos se introduce directamente la expresión que se desea derivar:

```
>>syms x
>> g=diff('cos(x)*sin(3*x)',x)
g =
3*cos(3*x)*cos(x) - sin(3*x)*sin(x)
>> subs(g,x,pi/3) %evaluamos la derivada en pi/3
ans =
-1.5000
>> diff('log(x)',x,3) %derivamos tres veces la función logaritmo
ans =
2/x^3
>> subs(ans,x,2) %evaluamos el resultado anterior para x=2
ans =
0.2500
>> diff('3*x^2-6*y^3=x*y','y')
ans =
(-18)*y^2 = x
```

Ejercicio 4.- Utiliza el criterio de la derivada segunda para hallar los extremos relativos de la función del ejercicio 3 y elabora una gráfica de la función en la que figuren destacados los correspondientes puntos del plano. Encuentra también un punto de inflexión de la función e incorpóralo a la representación gráfica.

2.2 Comando Taylor

Para obtener el polinomio de taylor de una función f de una única variable, podemos utilizar el comando **taylor** de **MatLab** y seguir alguno de los formatos siguientes, donde n es un entero positivo:

taylor(f,n,x0) proporciona el polinomio de taylor de grado n-1 de la función f relativo a un punto x0.

Si no se especifica el punto en cuestión, el programa considera que es 0, o lo que es lo mismo:

taylor(f,n) proporciona el polinomio de Mc-Laurin de grado n-1 de f.

Por defecto, el programa toma n=6. Es decir, que

taylor(f,x0) daría el mismo resultado que **taylor(f,6,x0)**

taylor(f) equivale en la práctica a **taylor(f,6,0)**

E j e m p l o s :

```
>> syms x,taylor(exp(x))

ans =

x^5/120 + x^4/24 + x^3/6 + x^2/2 + x + 1
```

```
>> syms a,taylor(exp(x),4,a)
ans =
exp(a) + (exp(a)*(a - x)^2)/2 - (exp(a)*(a - x)^3)/6 - exp(a)*(a - x)
```

Para apreciar mejor cuál es el polinomio que **MatLab** nos devuelve, podemos ejecutar la orden

```
>> pretty(ans)
```

Ejercicio 5.- Halla el polinomio de Mc-Laurin de grado 7 para la función del ejercicio anterior.

Ejercicio 6.- Halla el polinomio de Taylor de segundo grado de la función

$$f(x) = \sqrt[3]{1-x}$$

y utilízalo para obtener un valor aproximado de la raíz cúbica de 9/10. ¿Cuál es el error cometido en la aproximación?.

Ejercicio 7.- Calcula los polinomios de Taylor de grados 1, 2, 5 y 8 de la función seno relativos al punto π/6. Represéntalos junto con la función seno en cuatro ventanas gráficas que se puedan visualizar al mismo tiempo, en el rectángulo [0,π]×[0,3]. Hazlo de forma que la función se distinga del polinomio en el color y el trazo.

Comprueba además que el polinomio de Taylor de grado uno de la función seno relativo al punto π/6 coincide con la tangente a la función seno en ese mismo punto.

2.3 Extremos de funciones

Para calcular el punto de un intervalo (a,b) donde una función y=f(x) alcanza su valor mínimo, se puede usar el comando **fminbnd** con formato

>>fminbnd('imagen de la función',a,b)

Para calcular el punto del intervalo (a,b) en el que la función y=f(x) alcanza su valor máximo, hay que utilizar el mismo comando y calcular el mínimo de la función y=-f(x) en [a,b].

E j e m p l o

```
>>syms x,h=2*x^2+x-1;ezplot(h,[-2,2])
>> fminbnd('2*x^2+x-1',-2,2)
ans =
-0.2500
>> fminbnd('-(2*x^2+x-1)',-2,2)
ans =
1.9999
```

El mínimo relativo se corresponde con el único punto crítico de la función en el intervalo:

```
>> h1=diff(h)
h1 =
4*x + 1
```

```
>> solve(h1)

ans =

-1/4

>> h2=diff(h1) %La derivada segunda es siempre mayor que cero

h2 =

4
```

Ejercicio 8.- Define f1 y f2 de forma que sean las funciones derivadas de primer y segundo orden de la función

$$f(x) = \frac{1}{5+4\,cosx}$$

Representa f, f1, f2 y el eje de abscisas para valores de la variable independiente comprendidos entre π/2 y 3π/2. A continuación, estudia los extremos relativos de f en el intervalo I=[π/2,3π/2].

3 INTERPOLACIÓN

MatLab permite trabajar con una cantidad importante de técnicas de interpolación que permiten realizar ajustes rápidos que ocupan poca memoria.

La interpolación se define como la forma de estimar valores de una función entre aquellos dados por un conjunto de datos, y es una herramienta muy valiosa cuando no se puede evaluar rápidamente una función en puntos intermedios. Por ejemplo, esto ocurre cuando los puntos son el resultado de algunas medidas experimentales o procedimientos computacionales muy laboriosos.

Cuando queremos encontrar una función cuya gráfica pase a través de un conjunto finito de puntos obtenidos mediante experimentos, usamos la técnica de interpolación. Quizás el ejemplo más simple de interpolación sea las gráficas de **MatLab**. Por defecto, **MatLab** dibuja líneas rectas conectando los puntos que constituyen la gráfica. Esta interpolación lineal estima qué valores intermedios están en la línea recta que hay entre los puntos introducidos. Cuando el número de puntos aumenta y la distancia entre ellos disminuye, la interpolación lineal se hace más precisa.

Podemos observar un ejemplo al ejecutar las órdenes siguientes:

```
>> x1=linspace(0,2*pi,60);
>> x2=linspace(0,2*pi,6);
>> plot(x1,sin(x1),x2,sin(x2),'r-.')
>> xlabel('x'),ylabel('sin(x)'),title('Interpolación lineal')
>> axis('tight')
```

Existen múltiples métodos para hacer la interpolación dependiendo de las hipótesis que se hagan.

En su entorno de trabajo, **MatLab** proporciona entre otros el comando **interp1** para facilitar la interpolación. Por ejemplo, supongamos que tenemos dos vectores, X e Y, de coordenadas X(1),...,X(n) e Y(1),...,Y(n) respectivamente: podemos utilizar este comando para interpolar un vector de valores X0. Es decir, que dados los pares de puntos (X(i),Y(i)), **interp1** encuentra Y0(j) en el X0(j) deseado tal que Y0(j)=f(X0(j)), siendo f una función continua que se encuentra por interpolación. En este caso, se llama interpolación unidimensional porque la función f depende de una única variable.

La sintaxis de llamada es

Y0=interp1(X,Y,X0)

obteniéndose el vector Y0 tal que (X0(j),Y0(j)) son los puntos hallados por interpolación unidimensional sobre el conjunto de puntos dado.

Se puede utilizar la variante

Y0=interp1(Y,X0)

que supone que X=1:n siendo n la longitud de Y, así como

Y0=interp1(X,Y,X0,'*método*')

donde método es un argumento opcional, que permite especificar el método de interpolación deseado. Las posibilidades de elección para método son: *nearest* (vecino más cercano), *linear* (lineal), *cubic* (cúbica de Hermite), *spline*, *pchip* (interpolación cúbica polinomial de Hermite a trozos) y *v5cubic* (variante de la interpolación cúbica de **MatLab 5**). La elección del método determina la "suavidad" de la curva de interpolación. El método por defecto es *linear*.

Por ejemplo, para especificar interpolación cúbica en vez de lineal usaríamos la sintaxis

Y0=interp1(X, Y, X0, 'cubic')

E j e m p l o

Como parte de un proyecto de ciencia, Lisa registra la temperatura oficial en Spriengfield cada hora durante 12 horas de forma que se pueda usar esa información para facilitar datos sobre el clima local.

Lisa analiza sus datos, introduciendo un índice para las horas en que se tomaron los datos

```
>> horas=1:12;
```

y un vector para las lecturas de temperatura correspondientes, en grados Celsius

```
>> temps=[5 8 9 15 25 29 31 30 22 25 27 24];
```

Ejecutando las órdenes siguientes, **MatLab** muestra una gráfica en la que dibuja líneas que interpolan linealmente los puntos. Para estimar la temperatura en cualquier momento dado, Lisa podría intentar interpretar la gráfica visualmente. Alternativamente, podría usar el comando **interp1** como se indica a continuación:

```
>>plot(horas,temps,'+',horas,temps,'r')
>> t=interp1(horas,temps,9.25) %temp. estimada a las 9 y cuarto
t =
22.7500
>> t=interp1(horas,temps,4.75) %temp. estimada a las 5 menos cuarto
t =
22.5000
>> t=interp1(horas,temps,[3.25,6.5,7.25,11.75])
t =
10.5000 30.0000 30.7500 24.7500
```

En lugar de suponer que es una línea recta la que conecta los puntos, podemos pensar en alguna curva más suave. La hipótesis más común es la de tomar un polinomio de tercer orden, por ejemplo, un polinomio cúbico, utilizado para modelar cada segmento entre puntos consecutivos y tal que la pendiente de cada uno de estos polinomios coincide en los puntos. Este tipo de interpolación se llama *splines cúbicos* o simplemente *splines*.

Usando este método, Lisa encuentra la solución siguiente:

```
>> t=interp1(horas,temps,9.25,'spline') %temp.  estimada a las 9 y cuarto
t =
21.7763
>> t=interp1(horas,temps,4.75,'spline') %temp. estimada a las 5 menos cuarto
t =
22.8085
>> t=interp1(horas,temps,[3.25,6.5,7.25,11.75],'spline')
t =
9.8815    30.0427    31.3959    25.1792
```

Obsérvese que las respuestas obtenidas por Lisa con el método *spline* son diferentes de los resultados que le proporcionó la interpolación lineal. Como la interpolación es un proceso de estimar o "adivinar" valores, tiene sentido que usar diferentes reglas de estimación conduzca a resultados distintos.

Uno de los usos más comunes de la interpolación con splines es "suavizar" la curva que pasa por los puntos. Esto es, dado un conjunto de datos, se utiliza la interpolación con spline para evaluar los datos con puntos más próximos. Por ejemplo:

```
>> h=1:0.25:12; %estimar la temperatura cada cuarto de hora
>> t=interp1(horas,temps,h,'spline');
>> plot(horas,temps,'+',horas,temps,'r-.',h,t)
>> title('Temperatura en Springfield')
>> xlabel('Horas'),ylabel('Grados Celsius')
```

En la gráfica obtenida con las órdenes anteriores, la línea a trazos es la interpolación lineal, la línea sólida es la interpolación spline suavizada y los datos originales se marcan con una cruz. Al preguntar por una resolución "más fina" sobre el eje Horas y utilizando la interpolación por splines, tenemos una estimación de la temperatura más suave, pero no necesariamente más precisa. En particular, obsérvese que la pendiente de la curva solución no cambia de forma abrupta en los puntos.

A la hora de utilizar **interp1**, debemos tener en cuenta que no se le puede preguntar por resultados fuera del rango de la variable independiente; por ejemplo, la orden siguiente conduce a un error, ya que horas varía entre 1 y 12.

```
>> interp1(horas,temps,12.5)
ans =
NaN
```

Para interpolación unidimensional usando splines cúbicos, **MatLab** también dispone del comando **spline**. La sintaxis de llamada es en este caso

Y0=spline(X,Y,X0)

En **MatLab** también podemos interpolar datos usando splines y otros métodos en un pispás. Todo lo que tenemos que hacer es dibujar los puntos y en las opciones de nuestra ventana "*Figure*" escojer del menú *Tools* la opción *Basic Fitting* y seleccionar, por ejemplo, a continuación *spline interpolant.* Puedes probar a hacerlo desde la ventana gráfica obtenida con la orden

```
>>plot(horas,temps,'+')
```

Ejercicio 9.- Usando el comando interp1 con el método por defecto, hallar y representar 40 puntos de interpolación (x(i),y(i)) según la función y=sen *x* para valores de x igualmente espaciados entre 0 y 10. Resolver después el mismo problema usando el comando spline y comparar las gráficas en dos subventanas contiguas para observar que el tipo de interpolación así obtenido es más fino.

Ejercicio 10.- Consideramos el vector t que representa los años entre 1900 y 1990 (de 10 en 10) y el vector p con las poblaciones de Estados Unidos en esos años, en millones de habitantes.

p=[75'995 91'9720 105'7110 123'2030 131'6690 150'6970 179'3230 203'2120 226'5050 249'6330]

a) Inferir la población de Estados Unidos para 1975 mediante una función de interpolación.

b) Representar el polinomio interpolador spline entre los años 1900 y 2000 de año en año para los valores de p dados.

c) Representa los datos (t(i),p(i)) para i tomando valores entre 1 y 10. Utiliza el menú de la ventana gráfica correspondiente para dibujar después el polinomio interpolador spline.

4 MÉTODO DE BISECCIÓN

Se utiliza para aproximar una raíz de la ecuación f(x)=0 donde f es una función que verifica las hipótesis del teorema de Bolzano en un intervalo cerrado. Los pasos a efectuar son los siguientes:

1. Determinar un intervalo [a,b] tal que f(a) tenga distinto signo que f(b).
2. Hallar el punto medio c del intervalo.

3. Elegir, de entre [a,c] y [c,b], un intervalo donde la función cambie de signo.

4. Repetir los pasos 2 y 3 hasta conseguir la precisión deseada.

Partiendo del intervalo [a,b] , conseguimos una sucesión de intervalos de forma que en cada paso la longitud del intervalo escogido por contener a la raíz de f(x)=0 es la mitad de la longitud del intervalo considerado en el paso anterior.

Para ver un ejemplo de las posibilidades que presentan los ficheros .m a la hora de definir funciones, podemos elaborar un sencillo programa que facilite la puesta en práctica del método de bisección.

Editaremos un fichero de función eligiendo en el menú principal la opción *New* de *File* y seleccionando a continuación *M-File*. Podemos llamarlo **bisecci.m** y proporcionaremos entre los datos de entrada los extremos de un intervalo en el que f cambie de signo.

Comenzamos por declararlo como función y definir sus argumentos de entrada y salida. La primera línea del fichero **bisecci.m** puede quedar así, aunque cabe sofisticarla introduciendo otros parámetros de entrada y de salida:

function[c,fc] **=** bisecci(f,a,b,delta,Nmax)

A continuación incluímos nuestros comentarios; por ejemplo,

%BISECCI aproxima ceros de funciones en las hipótesis de Bolzano

% Entrada:

% f es una función continua,

% a y b son los extremos del intervalo de búsqueda,

% delta es la tolerancia para la solución

% Nmax es el número máximo de iteraciones permitidas

% Salida:

% c es la solución aproximada,

% fc=f(c)

En cada paso del método de bisección, hallamos el punto medio c del intervalo [a,b] y elegimos el subintervalo, [a,c] o [c,b] en el que se produce el cambio de signo. Hecho esto, renombramos las variables de modo que el semiintervalo seleccionado se llame [a,b] en el paso siguiente.

Una vez grabado el fichero, en la ventana de comandos definimos la función f que queremos utilizar. Por ejemplo,

>>syms x, f=cos(x)+1-x;

A continuación, solo tenemos que ejecutar la función **bisecci** introduciendo además de f los valores deseados para a, b, delta y Maxit.

En nuestro ejemplo, para aproximar una solución de la ecuación

$$\cos x + 1 - x = 0$$

en el intervalo [0'8,1'6] con un error menor que 0'0001, podemos ejecutar

>>[c,fc]=bisecci(f,0.8,1.6,0.0001,20)

Ejercicio 11.- Utilizar el algoritmo de dicotomía para aproximar una raíz de las siguientes ecuaciones con un error menor que 0'0001:

a) $x^3 - 5x + 2 = 0$ con [a,b]=[1,5]

b) $e^x - 2 - x = 0$ con [a,b]=[-2'4,1'6]

c) $\ln x - 5 + x = 0$ con [a,b]=[3'1,4'5]

5 REFERENCIAS

Matlab y Matemática computacional

Lantarón Sánchez, Sagrario / Llanas Juárez, Bernardo

Bellisco Madrid 2010

The Math Works Inc.

Matlab, edición de estudiante. Guía de usuario. Versión 4

Prentice Hall

Madrid 1995

Matemáticas en Ingeniería con Matlab

Peregrina Quintela Estévez

Universidade de Santiago de Compostela, 2000.

Matlab y sus Aplicaciones en las Ciencias y la Ingeniería

César Pérez López

Prentice Hall

Madrid 2002