|  |
| --- |
| UNIVERSIDAD NACIONAL DEL SUR - DCIC |
| Métodos de Computación Científica |
|  |
| Trabajo Práctico Nº 4: Aproximación de Funciones |
|  |
| **Gustavo Ferrer Dumrauf – L.U.: 82675** |
| **Fecha de Entrega : 24 de Octubre de 2012** |

|  |
| --- |
|  |

**Ejercicio 1:** Considere una función de dos dimensiones *f(x,y)* cuyos valores se conocen en todas la combinaciones de  y de . Entonces, la función *f(x,y)* puede aproximarse mediante el polinomio de Interpolación de Lagrange *P(x,y)* definido como:

Donde e  se conocen como los coeficientes de interpolación de Lagrange dados por:



a) Encuentre el polinomio de interpolación de Lagrange *P(x,y)* correspondiente a los siguientes datos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *yj* | *xi* | 0 | 1 | 2 |
| 0 |  | 1 | 2.7183 | 7.3891 |
| 2 |  | 7.3891 | 20.0855 | 54.5982 |
| 4 |  | 54.5982 | 148.4132 | 403.4288 |

**Solución:**

Se crea una función auxiliar X\_i para calcular el coeficiente de interpolación X\_i.

Se crea una función auxiliar Y\_i para calcular el coeficiente de interpolación Y\_j.

Se crea una función auxiliar “f” que calcula el valor de *f(x,y)*.

Finalmente, se cuenta con una función P(x,y) que aproxima el valor de *f(x,y)* en el punto dado. A continuación el código de dichas funciones:

|  |
| --- |
| function [ coef\_X\_i ] = X\_i( a, x, i )  % Calcula el coeficiente de interpolación X\_i para el punto de la abscisa  % "a"  % a: abscisa. P(a,b).  % x: vector de coeficientes x = [0 2 4]  % i: índice que indica el valor actual del vector x.    prod = 1;    for k = 1:1:3  if (k ~= i)  prod = prod \* ((a - x(k))/(x(i) - x(k)));  end  end    coef\_X\_i = prod;    end |

|  |
| --- |
| function [ coef\_Y\_j ] = Y\_j( b, y, j )  % Calcula el coeficiente de interpolación Y\_i para el punto de la ordenada  % al origen "b"  % b: ordenada al orígen de P(a,b).  % y: vector de coeficientes x = [0 2 4]  % j: índice que indica el valor actual del vector y.    prod = 1;    for k = 1:1:3  if (k ~= j)  prod = prod \* ((b - y(k))/(y(j) - y(k)));  end  end    coef\_Y\_j = prod;    end |

Notar que ambas funciones X\_i e Y\_j tienen exactamente el mismo código. Se las implementa como dos funciones por separado por una cuestión de claridad al calcular P(x,y).

|  |
| --- |
| function [ f\_val ] = f( x\_i, y\_j )  % x\_i: abscisa.  % y\_j: ordenada al origen.  % el primer valor posible para x\_i e y\_j es 0 y como el primer indice  % accesible de un vector en Matlab es 1 y no 0, se acomodan los valores de  % la función. Por ejemplo f(0,0) pasa a ser f(1,1).    f =[];    f(1,1) = 1;  f(1,3) = 7.3891;  f(1,5) = 54.5982;  f(2,1) = 2.7183;  f(2,3) = 20.0855;  f(2,5) = 148.4132;  f(3,1) = 7.3891;  f(3,3) = 54.5982;  f(3,5) = 403.4288;    f\_val = f(x\_i + 1, y\_j + 1); %se suma uno para acceder correctamente al elemento de la matriz.    end |

Obs: la función “f” está definida como una matriz, inicializada solo con los valores necesarios. El algoritmo que utiliza esta función asegura que no accede a posiciones no definidas.

|  |
| --- |
| function [ P\_xy ] = P( a , b)  % Evalúa al polinomio de interpolación de Lagrange en (a,b) que aproxima % a la  % función f(x,y) definida a continuación.  % P(x,y) = P(a,b) => a=x, b=y    x = [0 1 2];  y = [0 2 4];    suma = 0;    for i = 1:1:3  for j = 1:1:3  prod = X\_i(a,x,i) \* Y\_j(b,y,j) \* f(x(i),y(j));  suma = suma + prod;  end  end    P\_xy = suma;    end |

Función que calcula el polinomio de interpolación.

A modo de verificación se evalúa a p(x,y), aproximación de f(x,y), en los puntos de la tabla:

|  |
| --- |
| >> for i = 0:1:2  for j=0:2:4  f = P(i,j)  end  end  f =  1  f =  7.3891  f =  54.5982  f =  2.7183  f =  20.0855  f =  148.4132  f =  7.3891  f =  54.5982  f =  403.4288 |

Efectivamente, los valores obtenidos eran los esperados.

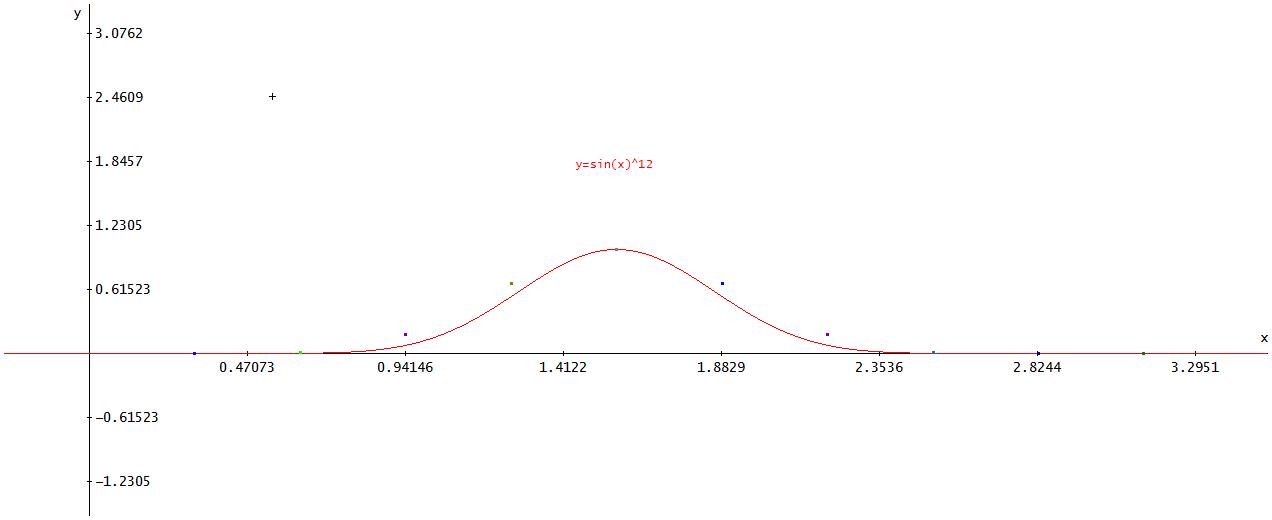
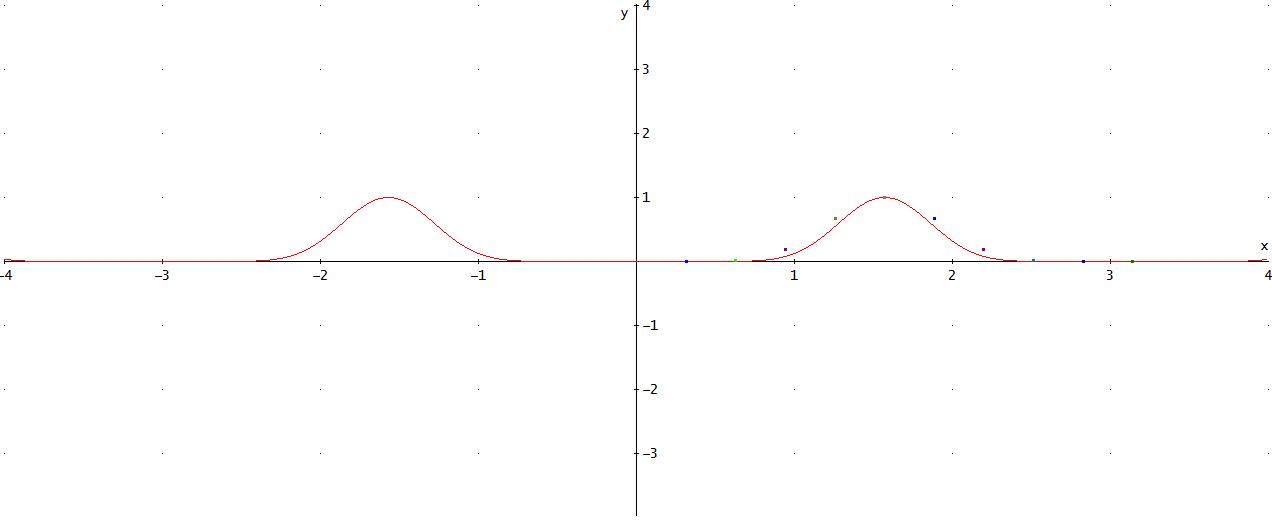
**b) Use el resultado para estimar el valor de *f* en (*x*=1.5, *y*=3.0)**

|  |
| --- |
| >> f = P(1.5,3)  f =  121.2896 |

**Ejercicio 2:** Considere los valores de la función  en los valores discretos de *x * como se indica en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| *i* | *xi* | *fi=f(xi)* 104 |
| 0 | 0 | 0 |
| 1 | 0.31416 | 0.8315 |
| 2 | 0.62832 | 142.4809 |
| 3 | 0.94248 | 1835.1291 |
| 4 | 1.25664 | 6693.5116 |
| 5 | 1.57080 | 10000.0000 |
| 6 | 1.88496 | 6693.3858 |
| 7 | 2.19912 | 1835.0555 |
| 8 | 2.51328 | 142.4702 |
| 9 | 2.82744 | 0.8314 |
| 10 | 3.14160 | 0 |

Curva  junto con los datos de la tabla:



Ingresamos en Matlab el vector “x” y el vector “y”:

|  |
| --- |
| >> x = 0: (pi/10) : pi;  >> y = sin(x).^12;  >> tabla = [x' y']  tabla =  0 0  0.3142 0.0000  0.6283 0.0017  0.9425 0.0786  1.2566 0.5476  1.5708 1.0000  1.8850 0.5476  2.1991 0.0786  2.5133 0.0017  2.8274 0.0000  3.1416 0.0000 |

Encontramos el valor interpolado de *f* para cada 

1. Use interpolación por splines lineales:

|  |
| --- |
| >> xi = (pi/20) : (pi/10) : (pi - pi/20);  >> yi = interp1(x,y,xi);  >> yy = (sin(xi)).^12;  >> inter1 = [xi' yy' yi']  inter1 =  0.1571 0.0000 0.0000  0.4712 0.0001 0.0009  0.7854 0.0156 0.0402  1.0996 0.2504 0.3131  1.4137 0.8619 0.7738  1.7279 0.8619 0.7738  2.0420 0.2504 0.3131  2.3562 0.0156 0.0402  2.6704 0.0001 0.0009  2.9845 0.0000 0.0000 |

Invocamos la función provista por Matlab “interp1” con los siguientes argumentos:

* x: valores ** entre 0 y 10.
* y = valores de *fi=f(xi)* 104
* xi = valores de.
* inter1: tabla con los xi en la primera columna, los valores del seno en cada xi en la segunda columna, y en la tercera los valores obtenidos de la interpolación.

Graficamos la función para verificar los resultados obtenidos:

|  |
| --- |
| >> plot(xi,yi,'--',xi,yi,'o',xi,yy) |

En rojo los valores reales de al función seno en el cada punto 

En azul y punteado la función en los valores interpolados por medio de una spline lineal. (Pasa por los puntos marcados con círculos).

Calculamos la medida del error cometido en la interpolación usando la relación:  


|  |
| --- |
| >> suma = 0;  >> for i =1:10  dif = inter1(i,2) - inter1(i,3);  suma = suma + dif^2;  end  >> E = suma  E =  0.0246 |

1. Use interpolación por splines cúbicas:

Usando los intervalos de xi hallados en el inciso anterior interpolamos usando splines cúbicas:

|  |
| --- |
| >> yc = interp1(x,y,xi,'cubic');  >> interc = [xi' yy' yc']  interc =  0.1571 0.0000 0.0000  0.4712 0.0001 0.0004  0.7854 0.0156 0.0241  1.0996 0.2504 0.2721  1.4137 0.8619 0.8314  1.7279 0.8619 0.8314  2.0420 0.2504 0.2721  2.3562 0.0156 0.0241  2.6704 0.0001 0.0004  2.9845 0.0000 0.0000 |

Graficamos para corroborar los resultados obtenidos:

|  |
| --- |
| >> plot(xi,yc,'--',xi,yc,'o',xi,yy)  >> xlabel('x'); ylabel('y'); title('Interpolación spline cúbica');  >> gtext('y=(sin^1^2(x))'); |

Se puede ver, comparando ambos resultados en el gráfico, que la spline cúbica interpola mucho mejor los datos.

Para corroborar esto efectuamos nuevamente el cálculo de la medida de error E, para los datos nuevos:

|  |
| --- |
| >> suma = 0;  >> for i =1:10  dif = interc(i,2) - interc(i,3);  suma = suma + dif^2;  end  >> E = suma  E =  0.0029 |

**Conclusión**

Evidentemente la medida del error cometido es mucho menor para splines cúbicas que para splines lineales, o lo que es equivalente, las splines cúbicas brindan una mejor aproximación que las splines lineales.