Sesión 7: Interpolación.

1 Interpolación polinómica de Lagrange

Introducimos los datos del problema.

```
--> xx : ([1.0, 2.0, 4.0, 6.0, 9.0]) $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $
```

Varias opciones para visualizar los datos:
Salida en otra ventana.

```
--> load(draw)$

    draw2d(point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy)
    )$
```

Salida en esta misma ventana.

```
--> xy : create_list([xx[i],yy[i]],i,1,n);
```

```
--> wxplot2d([discrete,xy])$
```

```
--> wxplot2d([discrete,xy],[style,points])$
```

Cambio del tamaño y color de los puntos.

```
--> wxplot2d([discrete,xy],[style,[points,2,3]])$
```

1.1 Fórmula de Lagrange

Introducimos los datos del problema.

```
--> xx : [1.0, 2.0, 4.0, 6.0, 9.0] $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $

    xy : create_list([xx[i],yy[i]],i,1,n);
```

Construimos la base de Lagrange.

```
--> l(i,x) := product((x-xx[j])/(xx[i]-xx[j]),j,1,i-1) *
                  product((x-xx[j])/(xx[i]-xx[j]),j,i+1,n)$
```

Polinomio interpolante.

```
--> pol1(x) := sum(yy[i]*l(i,x),i,1,n);
```

```
--> pol1(x);
```

```
--> expand(pol1(x));
```

Gráficas (con varias opciones para hacerlas).

```
--> load(draw)$

    draw2d(point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy),
        color = red,
        explicit(pol1(x), x, xx[1], xx[n])
    )$
```

```
--> wxplot2d( [[discrete,xy], pol1], [x, xx[1], xx[n]],
                [style, [points,3,2], [lines,1,1]]) $
```

```
--> wxplot2d( [[discrete,xy],pol1], [x, xx[1], xx[n]], [y,-3,16],
                [style, [points,3,1], [lines,1,2]]) $
```

1.2 Fórmula de Newton

Introducimos los datos del problema.

```
--> xx : ([1.0, 2.0, 4.0, 6.0, 9.0]) $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $

    xy : create_list([xx[i],yy[i]],i,1,n);
```

Construimos la tabla de diferencias divididas.
Para ello utilizamos una matriz triangular
inferior.

```
--> tdd : zeromatrix(n, n) $
    tdd[1] : yy $
    tdd : transpose(tdd) $

    for j : 2 thru n do(
        for i : j thru n do(
        tdd[i,j] : (tdd[i-1, j-1]-tdd[i,j-1])/(xx[i-j+1]-xx[i])
        )
    )$

    tdd;
```

```
--> pol2(x) := sum(tdd[i,i]*prod((x-xx[j]),j,1,i-1),i,1,n);
```

```
--> pol2(x);
```

```
--> expand(pol2(x));
```

Gráficas.

```
--> load(draw)$

    draw2d(point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy),
        color = green,
        explicit(pol2(x), x, xx[1], xx[n])
    )$
```

```
--> wxplot2d( [[discrete,xy],pol2],
                [ x, xx[1], xx[n]],
                [style, [points,3,1], [lines,1,3]]) $
```

1.3 Comparación de resultados

Salvo por los errores de redondeo, los polinomios pol1(x) y pol2(x) son iguales.

```
--> expand(pol1(x)-pol2(x));
```

Gráficamente no hay "diferencias".

```
--> wxplot2d( [[discrete,xy],pol1,pol2],
                [ x, xx[1], xx[n]],
                [style, [points,3,1], [lines,2,2],
                    [lines,1,3]]) $
```

1.4 Ejercicio 1

Repite la construcción de la tabla de diferencias divididas utilizando una matriz triangular superior.

1.5 Ejercicio 2

Siguiendo la idea de Lagrange, implementa el método correspondiente para hallar el polinomio que interpola los datos siguientes:
f(-1)=3, f'(-1)=4, f(0)=1, f'(0)=-2, f(2)=-1, f'(2)=4.

1.6 Ejercicio 3

Siguiendo la idea de Newton, implementa el método correspondiente para hallar el polinomio que interpola los datos del ejercicio anterior.

1.7 Ejercicio 4

Repite todos los cálculos propuestos sin utilizar las ideas de Lagrange y Newton. Es decir, a partir de la expresión general de un polinomio.

2 Interpolación con funciones Spline

2.1 Potencias truncadas

Para hacer los cálculos utilizaremos la técnica de las potencias truncadas. Definamos tales funciones.

```
--> potr(x, x0, m) := if x<x0 then 0 else (x-x0)^m $
```

2.2 Spline lineal

Introducimos los datos del problema.

```
--> xx : ([1.0, 2.0, 4.0, 6.0, 9.0]) $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $

    xy : create_list([xx[i],yy[i]],i,1,n);
```

Para evitar la racionalización de números decimales en coma flotante ejecutamos el siguiente comando.

```
--> keepfloat : true $
```

Planteamos el sitema a resolver. Con los cinco datos dados es suficiente para el planteo.

```
--> c1 : makelist(c1[i], i, 1, n)$

    s1(x) := c1[1] + c1[n]*x +
                sum(c1[i]*potr(x,xx[i],1),i,2,n-1)$

    eqn1 : makelist(s1(xx[i])= yy[i], i, 1, n)$

    coef1 : linsolve(eqn1,c1)$

    print(coef1)$
```

```
--> s1(x);
```

```
--> expand(s1(x));
```

Como acabamos de comprobar, no obtenemos una fórmula "explícita" del tipo que sí vimos para el polinomio interpolante.

Gráficas: obsérvese la orden
"ss1(x) := subst(coef1,s1(x))"
que permite utilizar s1(x).

```
--> ss1(x) := subst(coef1,s1(x)) $
```

```
--> load(draw)$
    draw2d(
        point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy),
        color = magenta,
        explicit(ss1(x), x, xx[1], xx[n])
    )$
```

```
--> wxplot2d( [[discrete,xy],ss1],
                [x, xx[1], xx[n]], [y,-3,15],
                [style, [points,3,1], [lines,1,4]]) $
```

2.3 Spline cuadrático (clase 1)

Introducimos los datos del problema.

```
--> xx : ([1.0, 2.0, 4.0, 6.0, 9.0]) $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $

    xy : create_list([xx[i],yy[i]],i,1,n);
```

Definimos las potencias truncadas y sus derivadas.

```
--> potr(x, x0, m) := if x<x0 then 0 else (x-x0)^m $
    dpotr(x, x0, m) := if x<x0 then 0 else m*(x-x0)^(m-1) $
```

Para evitar la racionalización de números decimales en coma flotante ejecutamos el siguiente comando.

```
--> keepfloat : true $
```

Planteamos el sitema a resolver. En este caso, además de los cinco datos dados, usamos una estimación de la derivada en el primer nodo.

```
--> c2 : makelist(c2[i],i,1,n+1)$

    s2(x) := c2[1] + c2[n]*x + c2[n+1]*x^2 +
                sum(c2[i]*potr(x,xx[i],2),i,2,n-1)$

    ds2(x) := c2[n] + 2*c2[n+1]*x +
                sum(c2[i]*dpotr(x,xx[i],2),i,2,n-1)$
```

```
--> eqn2 : append(makelist(s2(xx[i])= yy[i], i, 1, n),
           [ds2(xx[1])= (yy[1]-yy[2])/(xx[1]-xx[2])])$

    coef2 : linsolve(eqn2,c2)$

    print(coef2)$
```

```
--> s2(x);
```

```
--> expand(s2(x));
```

Gráficas: obsérvese la orden "ss2(x) := subst(coef2,s2(x))".

```
--> ss2(x) := subst(coef2,s2(x)) $
```

```
--> load(draw)$

    draw2d(
        point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy),
        color = cyan,
        explicit(ss2(x), x, xx[1], xx[n])
    )$
```

```
--> wxplot2d( [[discrete,xy],ss2], [x, xx[1], xx[n]], [y,-6,17],
              [style, [points,3,1], [lines,1,6]]) $
```

2.4 Spline cúbico sujeto horizontal (clase 2)

Introducimos los datos del problema.

```
--> xx : ([1.0, 2.0, 4.0, 6.0, 9.0]) $
    yy : ([2.0, 4.0, -1.0, 3.0, 8.0]) $
    n : length(xx) $

    xy : create_list([xx[i],yy[i]],i,1,n);
```

Definimos las potencias truncadas y sus derivadas.

```
potr(x, x0, m) := if x<x0 then 0 else (x-x0)^m $
dpotr(x, x0, m) := if x<x0 then 0 else m*(x-x0)^(m-1) $
```

Para evitar la racionalización de números decimales
en coma flotante ejecutamos el siguiente comando.

```
keepfloat : true $
```

Planteamos el sitema a resolver. En este caso, además
de los cinco datos dados, exigimos que las derivadas
en los nodos extremos sean iguales a cero.

```
c3 : makelist(c3[i],i,1,n+2)$

s3(x) := c3[1] + c3[n]*x + c3[n + 1]*x^2 + c3[n + 2]*x^3 +
            sum(c3[i]*potr(x, xx[i], 3), i, 2, n - 1)$

ds3(x) := c3[n] + 2*c3[n+1]*x + 3*c3[n+2]*x^2 +
            sum(c3[i]*dpotr(x,xx[i],3),i,2,n-1)$

eqn3 : append(makelist(s3(xx[i])= yy[i], i, 1, n),
        [ds3(xx[1])= 0, ds3(xx[n])=0])$

coef3 : linsolve(eqn3,c3)$

print(coef3)$
```

```
s3(x);
```

```
expand(s3(x));
```

Gráficas: obsérvese la orden
"ss3(x) := subst(coef3,s3(x))".

```
ss3(x) := subst(coef3,s3(x)) $
```

```
load(draw)$

draw2d(
    point_type = filled_circle,
    point_size = 1,
    points_joined = false,
    color = blue,
    points(xx,yy),
    color = green,
    explicit(ss3(x), x, xx[1], xx[n])
)$
```

```
wxplot2d( [[discrete,xy],ss3],
            [x, xx[1], xx[n]], [y,-3,15],
            [style,[points,3,1],[lines,1,3]]) $
```

2.5 Gráficas conjuntas de los splines calculados

```
--> load(draw)$

    draw2d(
        point_type = filled_circle,
        point_size = 1,
        points_joined = false,
        color = blue,
        points(xx,yy),
        color = magenta,
        explicit(ss1(x), x, xx[1], xx[n]),
        color = cyan,
        explicit(ss2(x), x, xx[1], xx[n]),
        color = green,
        explicit(ss3(x), x, xx[1], xx[n])
    )$
```

```
--> wxplot2d( [[discrete,xy],ss1,ss2,ss3],
              [x, xx[1], xx[n]],
              [style, [points,3,1],[lines,1,4],
              [lines,1,6],[lines,1,3]]) $
```

```
--> wxplot2d( [[discrete,xy],ss1,ss2,ss3],
              [x, xx[1], xx[n]], [y,-6,20],
              [style, [points,3,1],[lines,1,4],
              [lines,1,6],[lines,1,3]]) $
```

2.6 Ejercicio 5

Implementa los algoritmos del spline cúbico natural y del spline cúbico periódico mediante las potencias truncadas.

2.7 Ejercicio 6

Aplica los algoritmos desarrollados en el ejercicio anterior a los datos siguientes:
(1,5), (2,4), (4,-1), (6,3), (9,5).

2.8 Ejercicio 7

Repite todos los desarrollos y ejercicios de esta segunda sección con la técnica de construcción de splines por trozos.