MÈTODES NUMÈRICS Pràctiques d'Aproximació de Funcions. (ETSEIB)

1. **Polinomis**

   El polinomi $p(x) = 2x^2 - 8x - 10$, amb Matlab s'escriu com un vector:

   >> p=[2 -8 -10]

   Les seves arrels es calculen mitjançant la comana:

   >> rr=roots(p)

   >> ans= 5.0000; -1.0000

   Si volem fer el pas contrari, és a dir, obtindre el polinomi a partir de les arrels:

   >> pp=poly(rr)

   >> ans= 1.0000 -4.0000 -5.0000

   Com veiem el polinomi retornat sempre és mònic.

   Multiplicació: Per multiplicar dos polinomis farem servir `conv`

   >> conv([1 -5], [1 1])

   >> ans= 1 -4 -5

   Divisió: Per dividir dos polinomis farem servir `deconv` entenent que la divisió polinòmica té l'espressió $p(x) = s(x) * q(x) + r(x)$ de manera que es pot cridar `[s r]=deconv(p,q)`

   >> [s r]= deconv([1 -4 -5], [1 1])

   >> s= 1 -5

   >> r= 0 0 0

   Avaluació: Per avaluar el valor d'un polinomi en un punt farem servir `polyval`

   >> polyval(pp, -2)

   >> ans= 7

2. **Polinomis de Lagrange**

   Tenint en compte la forma del polinomi de Lagrange associat a un determinat punt x_i, una manera no optima però intuitiva de construir el polinomi de Lagrange associat a partir del conjunt de punts $x_0, \cdots, x_n$ seria construir el polinomi que els té a

tots com arrels i llavors dividir-lo pel factor associat a cada punt. Això ens donaria el numerador, i el denominador no és més que la substitució del polinomi obtingut en el punt x_i associat.

Exercici 1 : Calculeu els polinomis de Lagrange associats al conjunt de punts $x = \{-3, -2, 1, 2\}$ denotant-los per $L0, \cdots, L3$ respectivament.

Calculeu el polinomi interpolador de Lagrange en aquests punts per la funció $f(x) = e^x$. Avalue-lo en el punt 0.1 i compareu l'error amb el valor exacte.

3. **Avaluació Numèrica dels Polinomis de Lagrange**

Normalment no és necessari disposar d'una expressió explícita dels Polinomis de Lagrange, és suficient poder avaluar el valor que prenen en un punt determinat. Per això podem implementar una funció en Matlab que donat un conjunt de punts retorni el valor que pren el polinomi interpolador en aquests punts. Per això caldria definir una funció amb Matlab com la següent:

```
function pxx = lagrange_interp(x, f, xx)
% x és el vector d'abscissas.
% f és el vector de valors de f(x).
% xx representa el valor en el que volem fer la interpolació
% pxx és el valor que pren el polinomi interpolador de Lagrange en xx
pxx = 0;
n = length(x);
for j = 1 : n
    t = 1;
    for i = 1 : n
        if i ~= j
            t = t * (xx-x(i))/(x(j)-x(i));
        end
    end
    pxx = pxx + t*f(j);
end
```

Com exemple d'aplicació podríem fer la interpolació de la funció $f(x) = x^3 - 3x + 3$ en el conjunt de punts $x = \{-3, -2, -1, 0, 1, 2, 3\}$ segons fa el següent script de Matlab

```
%Considerem la corba y = x^3 - 3x + 3. Sabem que els punts
x = [-3 -2 -1 0 1 2 3];
y = [-15 1 5 3 1 5 21];
% estan sobre la corba.
% Quins son els valors quan x = -1.65 i 0.2?
x1 = -1.65;
y1 = lagrange_interp(x,y,x1)
x2 = 0.2;
y2 = lagrange_interp(x,y,x2)
%Pintem la nostra aproximació
plot(x, y, 'bo', x1, y1, 'ro', x2, y2, 'ro')
axis([-4 4 -17 23])
title('y = x^3 - 3x + 3')
xlabel('x')
ylabel('y')
hold on
xx=-3:0.1:3;
yy=xx.^3-3*xx+3;
plot(xx,yy)
```

4. **Interpolació i Aproximació amb Matlab**

   La comana més versàtil per resoldre problemes d'aproximació amb Matlab és la comana `polyfit`. La sintàxi d'aquesta comana és segons el propi help de Matlab:

   ```
   P = POLYFIT(X,Y,N) finds the coefficients of a polynomial
   P(X) of degree N that fits the data Y best in a least-squares
   sense. P is a row vector of length N+1 containing the
   polynomial coefficients in descending powers,
   P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X + P(N+1).
   ```

   En el cas en que `N` és exactament el número de punts més 1, el retorn és el polinomi interpolador.

Exercici 2 : Comproveu que en utilitzar els valors $x = \{-3, -2, -1, 0, 1, 2, 3\}$ i $y = \{-15, 1, 5, 3, 1, 5, 21\}$ el polinomi que retorna `p=polyfit(x,y,6)` és el polinomi interpolador de grau 6 en aquests punts, és a dir $p(x_i) = f(x_i)$.

La comana `polyfit` és molt més important, ja que en el cas que `N` sigui qulasevol altre valor més petit, aleshores Matlab retorna la solució mínim-quadràtica, per dir-ho fàcil, seria la millor aproximació possible en aquest cas. Per exemple:

>> x = [-3 -2 -1 0 1 2 3]

>> y = [-15 1 5 3 1 5 21]

>> p= polyfit(x,y,1)

>> p= 4.0000 3.0000

El polinomi retornat és la recta que millor aproxima aquests punts (la recta de *regressió*). El mateix passarà si anem variant el grau del polinomi d'aproximació. Per exemple:

```
x = [-3 -2 -1 0 1 2 3];
y = sin(x);
p=polyfit(x,y,3);
t=-3:0.1:3;
pt=polyval(p,t)
plot(x,y,'bo');
hold on;
plot(t,pt)
```

---

5. **Interpolació per trossos**

La interpolació per polinomis de grau elevat presenta mols problemes des d'el punt de vista numèric (*Fenòmen de Runge*) i per això en molts casos és preferible agrupar els punts de dos en dos, o de tres en tres, etc., per finalment obtenir una funció formada per diferents polinomis de grau baix. Aquesta és la tècnica dels *Splines* que es basa en interpolacions parcials (generalment de grau 3). Matlab incorpora les comanes `interp1`, `spline`, que contrueixen aquest tipus d'interpolació i, a més, retornen el valor que pren la funció d'interpolació en un nou punt.

En el següent exemple es pot veure la seva utilització per interpolar la funció $f(x) = \sin(x)$ a partir de 10 punts.

```
x = 0:10;
y = sin(x);
plot (x,y,'go')
```

```
hold on;
xx = 0:.25:10;
y1=interp1(x,y,xx);
plot(xx,y1,'-b')
hold on;
yy = spline(x,y,xx);
plot(xx,yy,'-.m')
```