INTERPOLAZIONE DI FUNZIONI TRAMITE SPLINES

- Come abbiamo visto nelle lezioni precedenti, è possibile approssimare delle funzioni tramite polinomi. In particolare, dati $n+1$ punti nel piano cartesiano è possibile costruire un polinomio di grado n che passi per i punti.

Spesso però, quando il numero di punti è elevato ($n > 6$), il polinomio interpolatore mostra un andamento insoddisfacente, esibendo delle forti oscillazioni tra i punti.

È necessario, a volte, trovare delle funzioni che interpolino i punti in modo morbido, senza oscillazioni.

Allo scopo si usano delle funzioni chiamate SPLINE, costituite da pezzi di polinomi dello stesso grado, collegati tra loro in maniera non spigolosa (smooth) nei punti di contatto. I punti nei quali la funzione cambia la sua forma analitica sono chiamati nodi (knots).

Si definisce **SPLINE di I grado** una funzione $S(x)$ tale che

1) $S(x)$ è definita nell'intervallo $[a,b]$
2) $S(x)$ è continua in $[a,b]$
3) L'intervallo $[a,b]$ è suddiviso in sotto-intervalli (partizione di $[a,b]$)

$$a \equiv x_0 < x_1 < \dots < x_n \equiv b$$

tali per cui $S(x)$ è un polinomio di primo grado in ogni intervallo $[x_j, x_{j+1}]$

$$S_j(x) = a_j x + b_j, \quad x \in [x_j, x_{j+1}]$$

È possibile estendere $S(x)$ al di fuori del suo dominio di definizione facendo sì che la funzione estrapoli quanto definito nei due intervalli vicini agli estremi

$$S(x) = S_0(x) \quad \text{per} \quad x < a$$

$$S(x) = S_{n-1}(x) \quad \text{per} \quad x > b$$

Vediamo ora come calcolare i parametri della Spline. Fissata la partizione

$$a \equiv x_0 < x_1 < x_2 < \dots < x_n \equiv b$$

abbiamo **n+1 nodi** (**n intervalli**) e quindi **2n coefficienti** da determinare $\{a_j\}$ e $\{b_j\}$

I vincoli che provengono dalla definizione della Spline sono

$2n$: ogni $S_j(x)$ ~~della~~ (e sono n polinomi) deve passare per i punti x_j e x_{j+1}

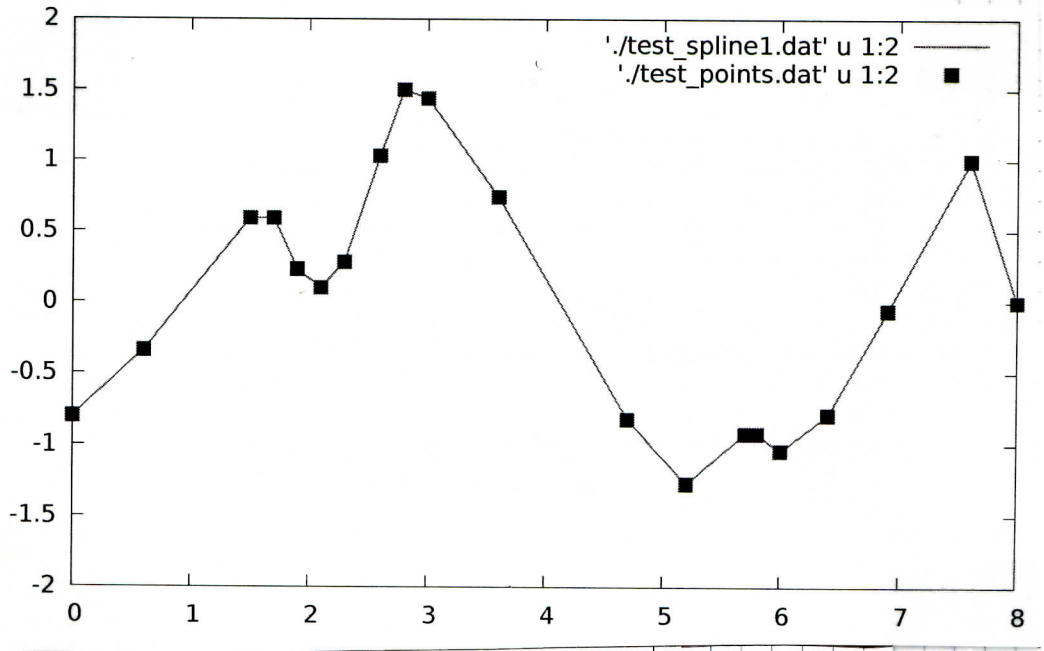
Possiamo pertanto scrivere ogni $S_j(x)$ come

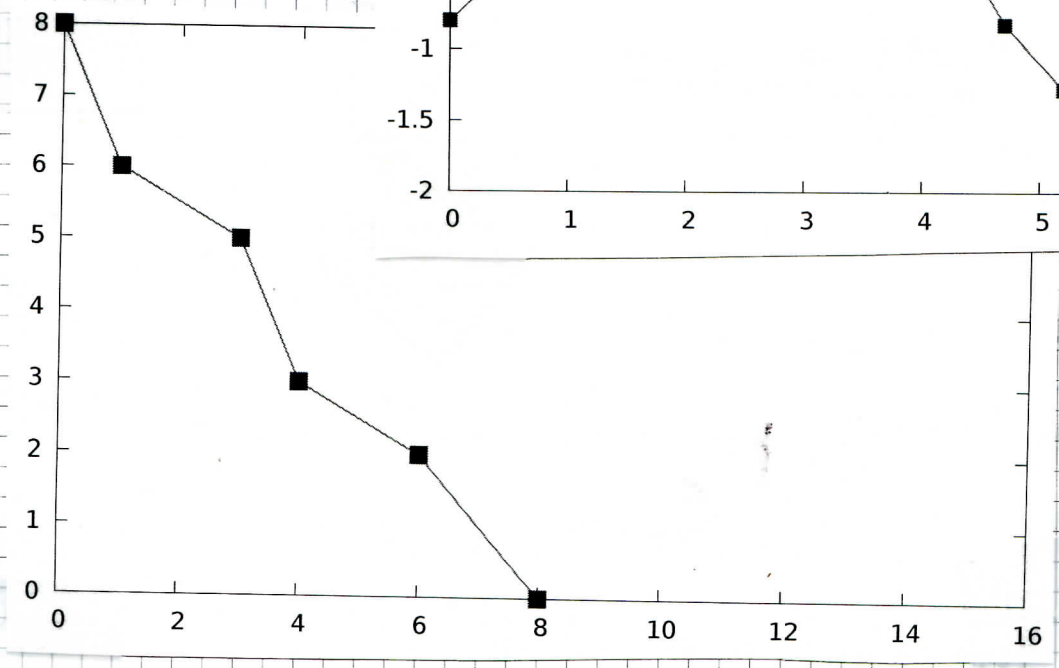
$$S_j(x) = y_j + m_j (x - x_j)$$

$$= y_j + (x - x_j) \frac{y_{j+1} - y_j}{x_{j+1} - x_j}$$

Vediamo ora come sia possibile scrivere un programma che interpoli dei punti con una Spline di I grado e applichiamolo ad alcuni set di punti.

```
double spline1(double x, int n,  double xp[], double yp[])
{
    int i=n-1;
    while (i>0) {
        if (x > xp[i]) {
            break;
        }
        i--;
    }

    return yp[i] + (x - xp[i])*(yp[i+1] - yp[i]) / (xp[i+1] - xp[i]);
}


in main()
{

    ...

    double xr = xp[0];
    double yr;
    while (xr < xp[nl-1]) {

        double yr = spline1(xr, nl, xp, yp);

        cout << xr << " " << yr << endl;

        xr += x_step;

    }

    ...
}
```





È possibile estendere la definizione di Spline ~~anche~~ ad un grado K.

S(x) è una Spline di grado K se soddisfa le seguenti caratteristiche:

1) S è definita nel dominio $[a,b]$
2) S e le sue derivate $S', S'', \dots, S^{(K-1)}$ sono funzioni continue in $[a,b]$
3) È possibile definire un insieme di punti x_j (chiamati nodi - (knots) di S) tali che

$$a \equiv x_0 < x_1 < \dots < x_m \equiv b$$

e inoltre

S è un polinomio al più di grado K in ogni sotto intervallo $[x_j, x_{j+1}]$

Nella definizione di Spline non viene menzionato il fatto che debba interpolare dei punti, anche se solitamente vengono usate come funzioni interpolatrici.

Il grado più utilizzato per le Spline è il III e le Spline sono chiamate Spline cubiche.

Tecnicamente le Spline di ordine dispari hanno un comportamento migliore di quelle di grado pari (interpolando i punti nei nodi)

Esiste un teorema matematico ~~m~~ che afferma che una funzione Spline cubica interpolante è la migliore funzione interpolante possibile.

Veniamo ora alla ricerca dei coefficienti dei polinomi delle singole cubiche.

Avremo a disposizione un insieme di $n+1$ punti nel piano $\{(x_i, y_i)\}$ $i = 0, 1, \dots n$ ordinati e assumeremo che gli $\{x_i\}$ siano i nodi (knots) della Spline.

La funzione che vogliamo costruire è costituita da n cubiche

$$S(x) = \begin{cases} S_0(x) & x \in [x_0, x_1] \\ S_1(x) & x \in [x_1, x_2] \\ \vdots & \\ S_{n-1}(x) & x \in [x_{n-1}, x_n] \end{cases}$$

con $S_j(x) = a_j + b_j x + c_j x^2 + d_j x^3$

Ogni cubica ha 4 coefficienti da determinare

Con $n+1$ nodi $\rightarrow$ n sottointervalli, avremo $4n$ coefficienti da determinare

Passiamo ora ai vincoli:

- ognuno dei polinomi deve passare per due punti (gli estremi)

$$S_i(x_i) = y_i \quad \text{e} \quad S_i(x_{i+1}) = y_{i+1}$$

↳ $2n$ vincoli

- le derivate I e II sono continue all'interno della partizione $k = 1, 2$

$$S_i^{(k)}(x_i) = S_{i+1}^{(k)}(x_{i+1}) \qquad i = 1, 2, \dots, n-1$$

↳ $2(n-1)$ vincoli

- si richiede infine che la derivata II si annulli agli estremi

$$S''(x_0) = S''(x_n) = 0$$

↳ 2 vincoli

VINCOLI TOTALI $2n + 2(n-1) + 2 = 4n$

la Spline così determinata prende il nome di Spline cubica naturale

Esistono altri modi per 'chiudere' il sistema di equazioni:

1) fissando la slope agli estremi

$$S'(x_0) = d_0$$
$$S'(x_n) = d_n$$

» CLAMPED SPLINE

2) imponendo condizioni periodiche agli estremi

$$S(x_0) = S(x_n)$$
$$S'(x_0) = S'(x_n)$$
$$S''(x_0) = S''(x_n)$$

» PERIODIC SPLINE

Consideriamo ora un esempio per calcolare i coefficienti della Spline

Esercizio

Scrivere le equazioni di una Spline cubica naturale che interpola i seguenti punti

$$(-1,1) \quad (0,2) \quad (1,-1)$$

Abbiamo 3 nodi e 2 sotto-intervalli.
Ricerchiamo una Spline così definita

$$S(x) = \begin{cases} S_0(x) = ax^3 + bx^2 + cx + d & x \in [-1,0] \\ S_1(x) = ex^3 + fx^2 + gx + h & x \in [0,1] \end{cases}$$

1) Condizione di interpolazione

$$S_0(-1) = 1 = -a + b - c + d$$
$$S_0(0) = 2 = d \rightarrow \boxed{d=2}$$
$$S_1(0) = 2 = h \rightarrow \boxed{h=2}$$
$$S_1(1) = -1 = e + f + g + h$$

2) Calcoliamo le derivate (I e II) e imponiamo le condizioni di continuità

$$S'(x) = \begin{cases} 3ax^2 + 2bx + c \\ 3ex^2 + 2fx + g \end{cases}$$

$$S''(x) = \begin{cases} 6ax + 2b \\ 6ex + 2f \end{cases}$$

$$S_0'(0) = S_1'(0) \rightarrow \boxed{c=g}$$
$$S_0''(0) = S_1''(0) \rightarrow \boxed{b=f}$$

3) richiediamo infine l'annullamento della derivata II agli estremi

$$S_0''(-1) = 0 \qquad -6a+2b=0 \Rightarrow \boxed{3a=b}$$

$$S_1''(+1) = 0 \qquad 6e+2f=0 \Rightarrow \boxed{3e=-f}$$

Ricapitolando i risultati ottenuti

$$d=2 \qquad h=2$$

$$-a+b-c=-1$$

$$e+f+g=-3$$

$$c=g$$

$$b=f$$

$$3a=b$$

$$3e=-f=-b=-3a \Rightarrow \boxed{e=-a}$$

Il sistema si riduce a

$$\begin{cases} 2a-c=-1 \\ 2a+c=-3 \end{cases}$$

$$\hookrightarrow 4a=-4 \Rightarrow \boxed{a=-1}$$

$$\hookrightarrow 2c=-2 \qquad \boxed{c=-1}$$

$$\boxed{b=-3} \qquad \boxed{d=2}$$

$$\boxed{e=+1} \qquad \boxed{f=-3} \qquad \boxed{g=-1} \qquad \boxed{h=2}$$

$$S(x)=\begin{cases} S_0(x)=-x^3-3x^2-x+2 \\ S_1(x)=x^3-3x^2-x+2 \end{cases}$$

Consideriamo infine un algoritmo che permetta di determinare i coefficienti di una generica Spline cubica

Dal momento che la derivata seconda è una funzione continua, possiamo porre

$$z_j \equiv S''(x_j) \qquad 0 \le j \le n$$

grazie all'annullamento della derivata II agli estremi,

$$\boxed{z_0 = z_n = 0}$$

All'interno di un generico intervallo $[x_i, x_{i+1}]$ $S''(x)$ è un polinomio lineare, e può essere scritto come

$$S_i''(x) = \frac{z_{i+1}}{h_i}(x - x_i) + \frac{z_i}{h_i}(x_{i+1} - x)$$

dove $h_i \equiv x_{i+1} - x_i$

Con la definizione adottata, avremo sempre

$$S_i''(x_i) = z_i$$

e

$$S_i''(x_{i+1}) = z_{i+1}$$

rispettando la definizione precedente di z_i

Integriamo due volte la derivata II in modo da ottenere $S(x)$ - definita a meno di costanti

$$S_i'(x) = \frac{1}{2}\frac{z_{i+1}}{h_i}(x-x_i)^2 + \frac{1}{2}\frac{z_i}{h_i}(x_{i+1}-x)^2 + C$$

e

$$S_i(x) = \frac{1}{6}\frac{z_{i+1}}{h_i}(x-x_i)^3 + \frac{z_i}{6h_i}(x_{i+1}-x)^3 + Cx + D$$

C e D sono costanti di integrazione.

Possiamo riscrivere il tutto come

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x-x_i)^3 + \frac{z_i}{6h_i}(x_{i+1}-x)^3 + C_i(x-x_i) + D_i(x_{i+1}-x)$$

Sfruttiamo ora le condizioni di interpolazione

$$S_i(x_i) = y_i \qquad S_i(x_{i+1}) = y_{i+1}$$

Si ottiene

$$S_i(x_i) = y_i = \frac{z_i}{6h_i}h_i^3 + D_i h_i$$

possiamo $h \equiv h_i$

$$\Rightarrow y_i = h\left(\frac{z_i}{6}h + D_i\right)$$

$$\Rightarrow D_i = \frac{y_i}{h} - \frac{h}{6}z_i$$

$$S_i(x_{i+1}) = y_{i+1} = \frac{z_{i+1}}{6h}h^3 + Ch$$

$$\Rightarrow C_i = \frac{y_{i+1}}{h} - \frac{z_{i+1}}{6} h$$

Riscrivendo l'espressione di $S(x)$ con le nuove costanti, appena determinate, avremo

$$\begin{aligned} S_i(x) = {} & \frac{1}{6} \frac{z_{i+1}}{h} (x - x_i)^3 + \frac{1}{6} \frac{z_i}{h} (x_{i+1} - x)^3 \\ & + \left(\frac{y_{i+1}}{h} - \frac{h}{6} z_{i+1} \right) (x - x_i) \\ & + \left(\frac{y_i}{h} - \frac{h}{6} z_i \right) (x_{i+1} - x) \end{aligned}$$

Rimangono infine da determinare i coefficienti $z_1, z_2, \dots, z_{n-1}$ tramite le condizioni di continuità di $S'(x)$ all'interno degli intervalli (nei nodi).

$$S'_{i-1}(x_i) = S'_i(x_i) \qquad i = 1, 2, \dots, n-1$$

Calcoliamo la derivata prima di $S(x)$

$$\begin{aligned} S'_i(x) = {} & \frac{1}{2} \frac{z_{i+1}}{h} (x - x_i)^2 - \frac{1}{2} \frac{z_i}{h} (x_{i+1} - x)^2 \\ & + \frac{y_{i+1}}{h} - \frac{h}{6} z_{i+1} \\ & - \frac{y_i}{h} + \frac{h}{6} z_i \end{aligned}$$

Imponiamo le continuità

$$S_e'(x_e) = -\frac{1}{2}\frac{z_e}{h_e}h_e^2 + \frac{y_{e+1}-y_e}{h_e} - \frac{h_e z_{e+1}}{6} + \frac{h_e z_e}{6}$$

$$= -\frac{h_e}{6}z_{e+1} - \frac{h_e}{3}z_e + \frac{y_{e+1}-y_e}{h_e}$$

$$S_{e-1}'(x_e) = \frac{1}{2}\frac{z_e}{h_{e-1}}h_{e-1}^2 + \frac{y_e}{h_{e-1}} - \frac{h_{e-1}}{6}z_e$$

$$- \frac{y_{e-1}}{h_{e-1}} + \frac{h_{e-1}}{6}z_{e-1}$$

$$= +\frac{h_{e-1}}{3}z_e + \frac{h_{e-1}}{6}z_{e-1} + \frac{y_e - y_{e-1}}{h_{e-1}}$$

Poniamo $S_e'(x_e) = S_{e-1}'(x_e)$

$$-\frac{h_e}{6}z_{e+1} - \frac{h_e}{3}z_e + \frac{y_{e+1}-y_e}{h_e} = \frac{h_{e-1}}{3}z_e + \frac{h_{e-1}}{6}z_{e-1} + \frac{y_e - y_{e-1}}{h_{e-1}}$$

$$h_{e-1}z_{e-1} + 2(h_{e-1}+h_e)z_e + h_e z_{e+1} = 6(b_e - b_{e-1})$$

con $b_e \equiv \frac{y_{e+1}-y_e}{h_e}$

Ponendo

$$u_i \equiv 2(h_{i-1} + h_i)$$

$$v_i \equiv 6(b_i - b_{i-1})$$

si ottiene un sistema di equazioni

$$\left.\begin{array}{l} z_0 = 0 \\ h_{i-1} z_{i-1} + u_i z_i + h_i z_{i+1} = v_i \\ z_n = 0 \end{array}\right\} \qquad 1 \le i \le n-1$$

Il sistema è tridiagonale ed è risolubile direttamente (vedi sistemi lineari tridiagonali)

Algoritmo

- dati $n+1$ punti (x_i, y_i)

1) calcolare $h_i = x_{i+1} - x_i$

$$b_i = \frac{y_{i+1} - y_i}{h_i}$$

2) porre $u_1 = 2(h_0 + h_1)$

$v_1 = 6(b_1 - b_0)$

e calcolare $u_i = 2(h_i + h_{i-1}) - \dfrac{h_{i-1}^2}{u_{i-1}}$

$$v_i = 6(b_i - b_{i-1}) - \frac{h_{i-1} v_{i-1}}{u_{i-1}}$$

3) imporre $z_0 = z_n = 0$

e determinare $z_i = \dfrac{v_i - h_i z_{i+1}}{u_i}$

```
// ----
// Calcola i coefficienti della spline di III grado
// Parametri:
//  - n : numero di punti da interpolare
//  - xp : coordinate x dei punti (array di dim n)
//  - yp : coordinate y dei punti (array di dim n)
//  - zs : coefficienti z della spline (array di dim n)
// ----
void spline3_coeff(int n, double xp[], double yp[], double zs[])
{
    double * h = new double[n];    // Distanza-x tra due punti consecutivi
    double * b = new double[n];
    double * u = new double[n];
    double * v = new double[n];

    for (int i=0; i<n-1; i++) {
        // cout << "xp[" << i << "] = " << xp[i] << " ";
        // cout << "yp[" << i << "] = " << yp[i] << endl;
        h[i] = xp[i+1] - xp[i];
        b[i] = (yp[i+1] - yp[i])/h[i];
        // cout << "h[" << i << "] = " << h[i] << endl;
        // cout << "b[" << i << "] = " << b[i] << endl;
    }

    u[1] = 2 * (h[0] + h[1]);
    v[1] = 6 * (b[1] - b[0]);

    for (int i=2; i<n-1; i++) {
        u[i] = 2 * (h[i] + h[i-1]) - h[i-1]*h[i-1]/u[i-1];
        v[i] = 6 * (b[i] - b[i-1]) - h[i-1]*v[i-1]/u[i-1];
        // cout << "u[" << i << "] = " << u[i] << " ";
        // cout << "v[" << i << "] = " << u[i] << endl;
    }

    zs[n-1] = 0.0;
    for (int i=n-2; i>0; i--) {
        zs[i] = (v[i] - h[i] * zs[i+1]) / u[i];
        // cout << "zs[" << i << "] = " << zs[i] << endl;
    }
    zs[0] = 0.0;

    delete h;
    delete b;
    delete u;
    delete v;
}

double spline3(double x, int n, double xp[], double yp[], double zs[])
{
    // - Select the interval
    int i=n-2;
    while (i>0) {
        if ((x - xp[i]) > 0) {
            break;
        }
        i--;
    }

    double h = xp[i+1] - xp[i];
    double temp = zs[i]/2.0 + (x - xp[i]) * (zs[i+1] - zs[i]) / (6*h);
    temp = -(h/6.0) * (zs[i+1] + 2*zs[i]) + (yp[i+1] - yp[i])/h
         + (x - xp[i]) * temp;

    return yp[i] + (x - xp[i]) * temp;

}
```

```
in main()
{

    ...

    spline3_coeff(n, xp, yp, zs);

    double xr = xp[0];
    double yr;
    while (xr < xp[n-1]) {

        double yr = spline3(xr, n, xp, yp, zs);

        cout << xr << " " << yr << endl;

        xr += x_step;

    }

    ...
}
```