Calcolo Numerico

(A.A. 2012-2013)

Esercitazioni

Lezione n. 14

Approssimazione

21-05-2013

Condizionamento

Esercizio

Studiare il condizionamento del seguente sistema lineare

$$\begin{cases} 2\,x + y = 3 \\ 2\,x + 1.001\,y = 0 \end{cases}$$

Soluzione La matrice dei coefficienti del sistema è

$$A = \begin{pmatrix} 2 & 1 \\ 2 & 1.001 \end{pmatrix}$$

Il numero di condizionamento di A è : $K(A) = \|A\|\|A^{-1}\|$.

$K(A)$ dipende dalla norma di matrici scelta.

Valutiamo allora $K_1(A), K_\infty(A)$ e $K_2(A)$, rispettivamente il numero di condizionamento di A rispetto alle norme $1, \infty$ e *spettrale*.

Si verifica facilmente che

$$A^{-1} = \frac{1}{det(A)} \begin{pmatrix} 1.001 & -1 \\ -2 & 2 \end{pmatrix}$$

con $det(A) = 0.002$.

Quindi

$$\|A\|_\infty = 3.001 \quad \|A^{-1}\|_\infty = \frac{4}{0.002} \quad \Rightarrow \quad K_\infty(A) = 3.001 \cdot 2 \cdot 1000 = 6002$$

$$\|A\|_1 = 4 \quad \|A^{-1}\|_1 = \frac{3.001}{0.002} \quad \Rightarrow \quad K_1(A) = 4 \cdot \frac{3.001}{0.002} = 6002$$

Per valutare la **norma spettrale** di A è necessario calcolare il **raggio spettrale** della matrice A^TA, infatti

$\|A\|_2 = \sqrt{\rho(A^TA)}$:

$$A^TA = \begin{pmatrix} 2 & 2 \\ 1 & 1.001 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 2 & 1.001 \end{pmatrix} = \begin{pmatrix} 8 & 4.002 \\ 4.002 & 2.002001 \end{pmatrix}$$

i cui autovalori λ_1 e λ_2 sono soluzione della seguente equazione di secondo grado

$$\lambda^2 - 10.002001\lambda + 0.000004$$

Poichè $\rho(A^TA) = max|\lambda_1, \lambda_2| = 10.00200060008001$, risulta

$\|A\|_2 = \sqrt{\rho(A^TA)} \approx 3.1626$.

Analogamente

$$A^{-1\,T}A^{-1} = \frac{1}{(det(A))^2}\begin{pmatrix} 1.001 & -2 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} 1.001 & -1 \\ -2 & 2 \end{pmatrix} = \begin{pmatrix} 5.002001 & -5.001 \\ -5.001 & 5 \end{pmatrix}$$

da cui risulta

$$\|A^{-1}\|_2 = \sqrt{\rho(A^{-1\,T}A^{-1})} \approx 1.5813 \cdot 10^3$$

e quindi

$$K_2(A) = \|A\|_2\|A^{-1}\|_2 = 3.1626 \cdot 1.5813 \cdot 10^3 = 5.0010 \cdot 10^3.$$

In tutti e tre i casi il **numero di condizionamento è molto alto**, ne segue che una piccola perturbazione sui dati, produce un errore non trascurabile sulla soluzione.

Per esempio, la soluzione del sistema procedente è $\mathbf{x} = (x, y)$, con $x = 1501.5$ e $y = -3000$.

Supponiamo ora di aver un errore pari a 0.001 sul coefficiente a_{22} della matrice A, cioè di dover risolvere il sistema

$$\begin{cases} 2\,x + y = 3 \\ 2\,x + 1.002\,y = 0 \end{cases}$$

In questo caso la soluzione diventa $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$, con $\tilde{x} = 751.5$ e $\tilde{y} = -1500$, cioè , scegliendo la norma infinito,

$$\frac{\|\delta\mathbf{x}\|\infty}{\|\mathbf{x}\|\infty} = \frac{1500}{3000} = \frac{1}{2}$$

mentre

$$\frac{\|\delta\mathbf{A}\|\infty}{\|\mathbf{A}\|\infty} = \frac{0.001}{3.001} = 3.33 \cdot 10^{-4}.$$

Esercizio

Scrivere la funzione Matlab **num_cond.m** che calcoli il numero di condizionamento di una matrice A rispetto alla norma indicata nella variabile $tipo\_norma$, che può assumere i valori 1, 2 o inf. Nel caso non venga specificato il valore della variabile $tipo\_norma$, la funzione deve considerare la norma infinito.

Considerare la matrice dell'esercizio precedente e usare la funzione **num_cond.m** per calcolarne il numero di condizionamento rispetto alle norme 1, 2 e infinito. Confrontare i risultati con la funzione predefinita di matlab **cond.m**.

Soluzione:

```
function [K] = num_cond(A,varargin)
% function [K] = num_cond(A,varargin)
% calcola il numero di condizionamento della matrice A rispetto alla
% norma indicata in varargin. Se non viene indicata alcuna norma,
% si considera la norma infinito

if nargin == 0
    error('E'' necessario definire una matrice!!!')
elseif nargin == 1
    dim = size(A);
    if dim(1)~=dim(2)
        error('La matrice deve essere quadrata')
    end
    tipo_norma = inf;
```

```
elseif nargin == 2
    tipo_norma = varargin{1};
    if (tipo_norma ~= 1) & (tipo_norma ~= 2) & (tipo_norma ~= inf)
       error('la norma deve essere 1 2 o inf')
    end
else
    error('troppe variabili di input!!!')
end


IA = inv(A); % matrice inversa di A

NA = norm(A,tipo_norma); % norma di A
NIA = norm(IA,tipo_norma); % norma della matrice inversa

K = NA*NIA; % numero di condizionamento
```

Dal **Command Window:**

```
>> A = [2 1 ; 2 1.001];
>> K = num_cond(A,1);
>> K
K =
    6.00200000000661e+003
>> K = cond(A,1)
K =
    6.00200000000661e+003
>> K = num_cond(A,inf);
>> K
K =
    6.00200000000661e+003
>> K = cond(A,inf)
K =
    6.00200000000661e+003
```

```
>> K = num_cond(A,2);
>> K
K =
    5.001000300040555e+003
>> K = cond(A,2)
K =
    5.001000300040541e+003
>> b = [3;0];
>> x =  A\b
x =
  1.0e+003 *
   1.501500000000017
  -3.000000000000033
>> Ap=[2 1 ; 2 1.002];
>> xp =  Ap\b
xp =
  1.0e+003 *
   0.751500000000000
  -1.500000000000000
```

Approssimazione di dati e funzioni

Problema

Data la **tabella** $\{x_i, y_i\}$, $i = 0, \ldots, n$, si vuole trovare una **funzione analitica** φ_M che **approssimi** i dati.

La **tabella** $\{x_i, y_i\}$ può essere il risultato di **misure sperimentali** oppure può rappresentare i valori di una funzione la cui **espressione analitica** è nota ma **complicata** da calcolare direttamente.

Per poter costruire una **funzione approssimante** bisogna stabilire

- in quale **classe** di funzioni si vuole operare
- il **metodo di approssimazione**

Interpolazione polinomiale

Tabella: $\{x_i, y_i\} \qquad i = 0, \ldots, n$

Intervallo di interpolazione: $[a, b] = [x_0, x_n]$

Funzione approssimante: $p_n(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$

Metodo di approssimazione: $p_n(x_i) = y_i \quad (i = 0, 1, \ldots, n)$ $\rightarrow$ **Interpolazione**

Risolvere il **problema dell'interpolazione** vuol dire individuare il polinomio p_n, cioè i **coefficienti reali** a_k, che soddisfano le **condizioni di interpolazione**. Questo equivale a risolvere il **sistema lineare**

$$\begin{array}{l} p(x_0) = \\ p(x_1) = \\ \\ \\ p(x_n) = \end{array} \left\{ \begin{array}{ccccccccc} a_0 & + & a_1 x_0 & + & a_2 x_0^2 & + & \cdots & + & a_n x_0^n & = & y_0 \\ a_0 & + & a_1 x_1 & + & a_2 x_1^2 & + & \cdots & + & a_n x_1^n & = & y_1 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_0 & + & a_1 x_n & + & a_2 x_n^2 & + & \cdots & + & a_n x_n^n & = & y_n \end{array} \right. \rightarrow \quad VA = Y$$

Unicità del polinomio interpolatore

$$\boxed{VA = Y} \text{ con } \underbrace{V = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{\text{Matrice di Vandermonde}} \quad A = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} \quad Y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

La **matrice di Vandermonde** di $n+1$ nodi **distinti** $\{x_i\}$, $i = 0, \ldots, n+1$, è **regolare** poiché

$$\det V = \prod_{j>i} (x_i - x_j) \neq 0$$

$\Rightarrow$ esiste un'**unica** soluzione A del sistema.

$\Downarrow$

Esiste **uno e uno solo** polinomio p_n di grado n che verifica le **condizioni di interpolazione**

$$p_n(x_i) = y_i \qquad i = 0, \ldots, n$$

Approssimazione ai minimi quadrati

Problema.
Data la **tabella** $\{x_i, y_i\}$, $i = 0, 1, \ldots, n$, si vuole trovare una **funzione analitica** φ_M che **approssimi** i dati.

In questo caso la **tabella** è il risultato di **misure sperimentali** ciascuna delle quali è affetta da un **errore di misura** ε_i.

Metodo di approssimazione: si sceglie la funzione approssimante φ_M in modo da **minimizzare**

$$\sum_{i=0}^{n} [\varphi_M(x_i) - y_i]^2 \qquad \textbf{Scarto quadratico}$$

oppure, introducendo i **pesi** $w_i > 0$, $\forall i$,

$$\sum_{i=0}^{n} w_i \, [\varphi_M(x_i) - y_i]^2 \qquad \textbf{Scarto quadratico pesato}$$

Polinomio algebrico ai minimi quadrati

Tabella: $\{x_i, y_i\} \qquad i = 0, 1, ..., n$

Funzione approssimante:

$$P_M(x) = a_0 + a_1 x + \cdots + a_{M-1} x^{M-1} + a_M x^M \qquad \boxed{M << n}$$

Metodo di approssimazione: si minimizza lo **scarto quadratico**

$$\sigma^2(a_0, a_1, ..., a_M) = \sum_{i=0}^{n} [\underbrace{a_0 + a_1 x_i + \cdots + a_{M-1} x_i^{M-1} + a_M x_i^M}_{P_M(x_i)} - y_i]^2$$

Funzioni di base:

$$\psi_0(x) = 1, \ \psi_1(x) = x, \ \ldots, \ \psi_k(x) = x^k, \ \cdots, \ \psi_M(x) = x^M$$

Esercizio

Scrivere la funzione matlab **vandermonde.m** che riceva in input un vettore $\mathbf{X}$ e un intero N e costruisca la matrice di Vandermonde $\mathbf{V}$ delle componenti del vettore $\mathbf{X}$ (nodi). Il numero di colonne di $\mathbf{V}$ deve essere $N + 1$.

Si generi un vettore $\mathbf{X}$ di 20 elementi equispaziati nell'intervallo $[0,\ 1]$ e se ne calcoli la matrice di Vandermonde $\mathbf{V}$ associata di dimensione 20×5 usando la funzione **vandermonde.m**.

Confrontare i risultati usando la funzione predefinita di Matlab **vander.m**

```
function [V] = vandermonde(X,N)
% function [V] = vandermonde(X)
% calcola la matrice di Vandermonde dei nodi X(i).
%
% INPUT
% X = vettore di nodi
% N = numero di colonne +1 della matrice di Vandermonde
%
% OUTPUT
% V = matrice la cui k-esima colonna e' data da X.^k

dimX = size(X);
if size(dimX)>1
    error('La variabile di input deve essere un vettore!')
end

for k = 0:N
    V(:,k+1) = X.^k;
end
```

Dal Command Window

```
>> x = linspace(0,1,20);
>> V = vandermonde(x,4);
>> V

V =

    1.0000         0         0         0         0
    1.0000    0.0526    0.0028    0.0001    0.0000
    1.0000    0.1053    0.0111    0.0012    0.0001
    1.0000    0.1579    0.0249    0.0039    0.0006
    1.0000    0.2105    0.0443    0.0093    0.0020
    1.0000    0.2632    0.0693    0.0182    0.0048
    1.0000    0.3158    0.0997    0.0315    0.0099
    1.0000    0.3684    0.1357    0.0500    0.0184
    1.0000    0.4211    0.1773    0.0746    0.0314
    1.0000    0.4737    0.2244    0.1063    0.0503
    1.0000    0.5263    0.2770    0.1458    0.0767
    1.0000    0.5789    0.3352    0.1941    0.1123
    1.0000    0.6316    0.3989    0.2519    0.1591
    1.0000    0.6842    0.4681    0.3203    0.2192
    1.0000    0.7368    0.5429    0.4001    0.2948
    1.0000    0.7895    0.6233    0.4921    0.3885
    1.0000    0.8421    0.7091    0.5972    0.5029
    1.0000    0.8947    0.8006    0.7163    0.6409
    1.0000    0.9474    0.8975    0.8503    0.8055
    1.0000    1.0000    1.0000    1.0000    1.0000
```

Alcune Funzioni

y = polyval (a,x)

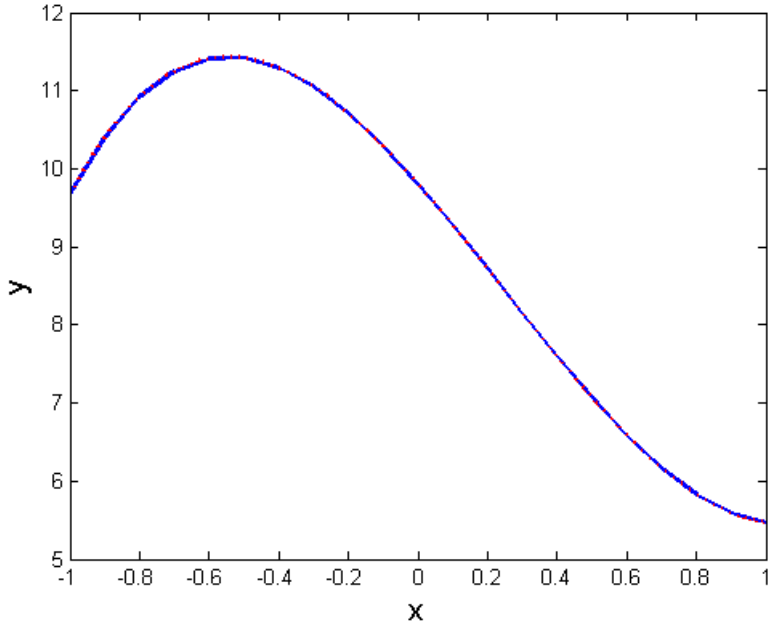
valuta il polinomio i cui coefficienti sono contenuti nel vettore $\mathbf{a}$ in corrispondenza degli elementi del vettore $\mathbf{x}$ Il grado massimo n del polinomio è pari dalla lunghezza del vettore $\mathbf{a}$ diminuita di 1. Il primo elemento di $\mathbf{a}$ è il coefficiente moltiplicativo del monomio gi grado massimo:

$$y_i = a_1\, x_i^n + a_2\, x_i^{n-1} + \cdot + a_n\, x_i + a_{n+1}$$

Esempio

```
>> a=[1 2 8]

a =

     1     2     8

>> polyval(a,5)

ans =

    43
```

```
>> a = [3 -2.23 -5.1 9.8];
>> x = -1:.1:1;
>> y = polyval(a,x);
>> plot(x,y)
>> hold on, fplot(@(x)[3*x^3-2.23*x^2-5.1*x+9.8],[-1 1],'r:')
>> xlabel('x')
>> ylabel('y')
```



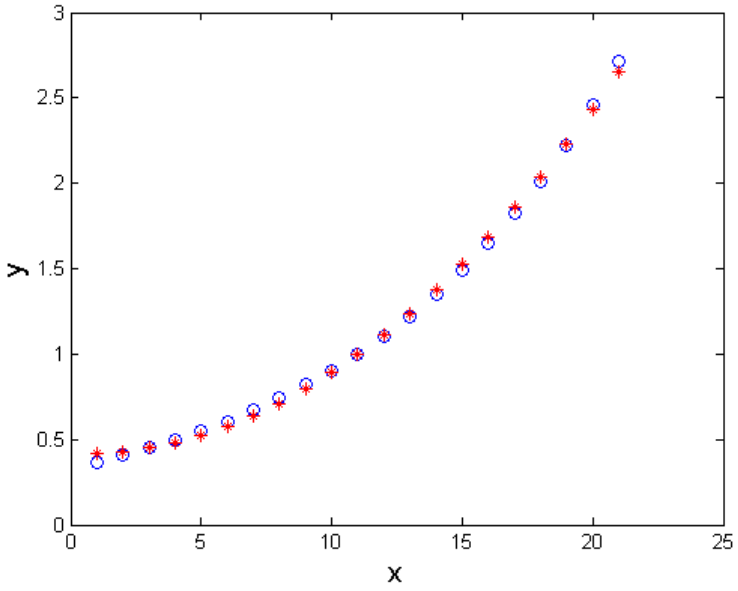
,

Alcune Funzioni

a = polyfit (x,y,n)

calcola i coefficienti **a** del polinomio di grado **n** che approssima i valori in **y** corrisponenti ai nodi in **x** usando la tecnica dei minimi quadrati.

```
>> a = polyfit(x,y,2);
>> yn = polyval(a,x);
>> figure, plot(y,'o')
>> hold on, plot(yn,'r*')
>> xlabel('x')
>> ylabel('y')
```



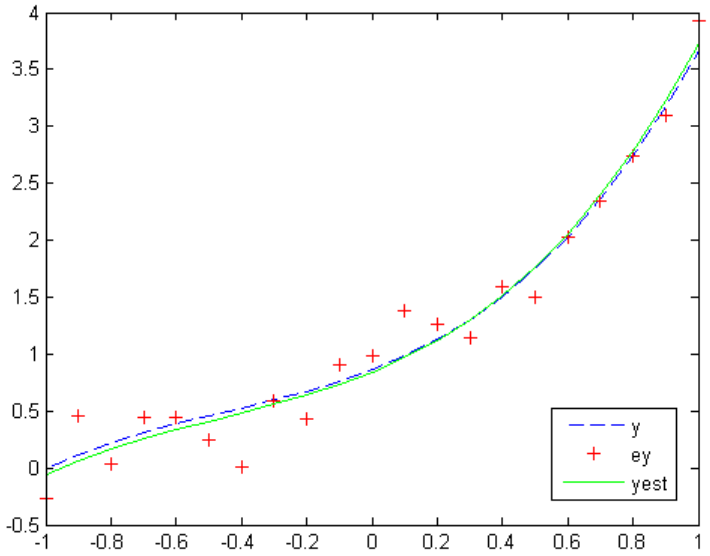
,

Esercizio

Scrivere uno script matlab che generi un vettore y contenente i valori di un polinomio di terzo grado in corrispondenza di nodi equidistanti di passo 0.1 nell'intervallo $[-1, 1]$. Si generi il vettore $noise$, tale che $\|noise\|_2 = 1$, della stessa dimensione di y usando la funzione **randn**. Sia $ey = y + noise$ (simulano i dati ottenuti in un esperimento numerico). Si stimi il polinomio di terzo grado che approssima i dati ey nel senso dei minimi quadrati. Sia $yest$ il vettore stimato e lo si confronti con il vettore originale y

```
>> a = [0.74 0.97 1.1 0.86];
>> x = -1:.1:1;
>> y = polyval(a,x);
>> noise = randn(1,length(y));
>> noise = noise/norm(noise);
>> ey = y + noise;
>> aest = polyfit(x,ey,3);
>> disp([a;aest])
    0.7400    0.9700    1.1000    0.8600
    0.7166    0.9985    1.1751    0.8379
>> yest = polyval(aest,x);
```

```
>> figure
>> plot(x,y,'b--')
>> hold on
>> plot(x,ey,'r+')
>> plot(x,yest,'g')
>> legend('y','ey','yest')
>> norm((y-yest))^2
ans =
    0.0324
```



,

Esercizio

La tabella seguente riporta le misure della densità relativa ρ dell'aria a diverse altezze h.

h (km)	0	1.525	3.050	4.575	6.100	7.625	9.150
ρ	1	0.8617	0.7385	0.6292	0.5328	0.4481	0.3741

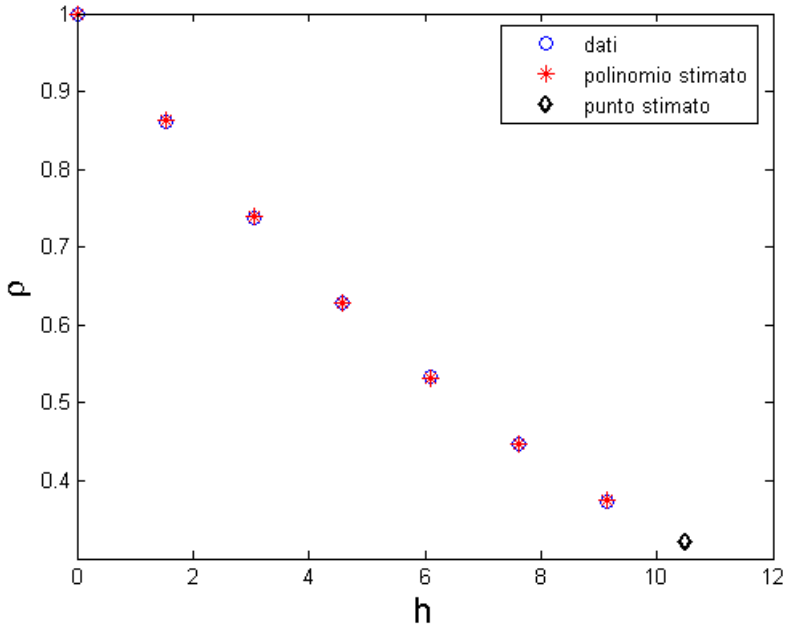
Si approssimi ρ con un polinomio di secondo grado e si stimi il valore di ρ in corrispondenza di $h = 10.5$ km.

Soluzione

```
h = [0 1.525  3.050  4.575 6.100 7.625 9.150];
rho = [1 0.8617  0.7385  0.6292 0.5328 0.4481 0.3741];

a = polyfit(h,rho,2);
disp(a)

rho_pol = polyval(a,h);

rho_punto = polyval(a,10.5);

figure, plot(h,rho,'o')
hold on, plot(h,rho_pol,'r*')
hold on, plot(10.5,rho_punto,'kd')
xlabel('h')
ylabel('\rho')
legend('dati','polinomio stimato','punto stimato')
```

dal command window

```
>> esercizio_rhoaria
0.0028    -0.0934     0.9989
```



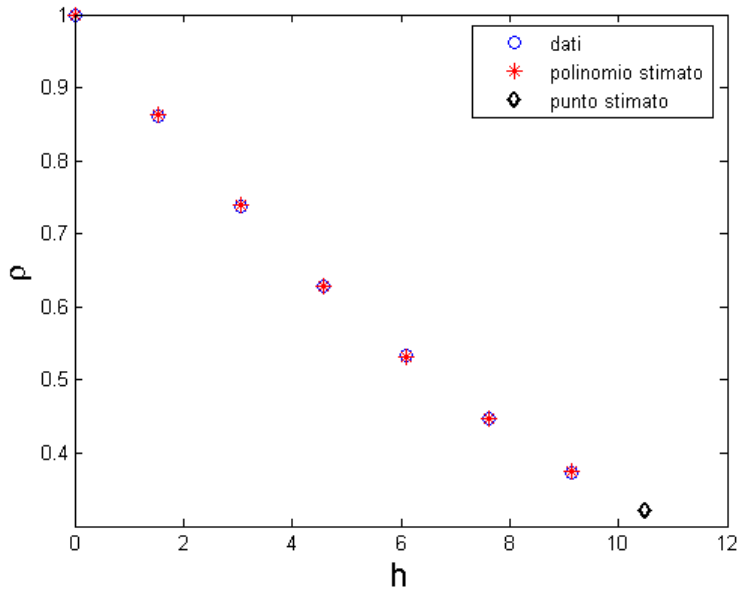
,

In modo analogo

```
V = vandermonde(h,2);
H = V'*V;
B = V'*rho';

a = H\B;
disp(a)

a = fliplr(a')
rho_pol = polyval(a,h);

rho_punto = polyval(a,10.5);

figure, plot(h,rho,'o')
hold on, plot(h,rho_pol,'r*')
hold on, plot(10.5,rho_punto,'kd')
xlabel('h')
ylabel('\rho')
legend('dati','polinomio stimato','punto stimato')
```

dal command window

```
>> esercizio_rhoaria
    0.9989
   -0.0934
    0.0028
a =
    0.0028   -0.0934    0.9989
```



,

Esercizio

Scrivere una funzione matlab che riceva in input il vettore dei nodi x e il vettore delle misure y e restituisca come output il vettore a dei coefficienti del polinomio che meglio approssima i dati nel senso dei minimi quadrati e l'errore dell'approssimazione err. La funzione deve prevedere come eventuale terzo input un vettore contenente i punti in cui valutare il polinomio stimato. Sia $ynew$ il vettore contenente i valori calcolati. $ynew$ è un'ulteriore variabile di output della funzione.

```
function [coeff_pol,varargout] = find_bestpol(xdata,ydata,varargin)
% [coeff_pol,varargout] = find_bestpol(xdata,ydata,varargin)
% calcola i coefficienti del polinomio che meglio approssima i dati ydata
% nei punti xdata. Come terzo vettore di input  possibile passare i punti
% in cui stimare la funzione approssimante (terza variabile di output).
%
% INPUT
% xdata = vettore dei nodi
% ydata = vettore dei valori della funzione nei nodi
%
%
% OUTPUT
% coeff_pol = vettore dei coefficienti del polinomio di migliore
%             approssimazione
%
```

```
% controllo degli input e degli putput
if nargin < 2
    error('dati di input non sufficienti!')
elseif nargin ==2
    xnew = [];
elseif nargin == 3
    xnew = varargin{1};
elseif nargin>3
    error('troppe variabili di input!!!')
end

if nargout > 2
    error('troppe variabili di output!!!')
elseif (nargin == 3) & (nargout~=2)
    error('il numero delle variabili di output deve essere 2!!!')
end
```

```
xdata = xdata(:);
ydata = ydata(:);

if (length(xdata) ~= length(ydata))
    error('il numero di dati deve essere  uguale al numero di nodi!!!')
end

% calcolo dei polinomi di approssimazione ai minimi quadrati di grado n
% con 1<=n<=nmax
if length(xdata)>10
    nmax = 10;
else
    nmax = max(length(xdata)-2,1);
end
for n = 1:nmax
    coeff_poli{n} = polyfit(xdata,ydata,n);
    ystime = polyval(coeff_poli{n},xdata);
    errore(n) = sum((ystime-ydata).^2);
end
```

```
% visualizza i coefficienti di tutti i polinomi calcolati e l'errore di
% approssimazione corrispondente
celldisp(coeff_poli)
disp(errore)
% determina il grado del polinomio in corrispondenza del quale si ha
% l'errore minimo
errore(find(errore<=.5*10^-14))=0;
[min_errore,pos_min_errore] = min(errore);
coeff_pol = coeff_poli{pos_min_errore};

% valuta il polinomio di migliore approssimazione nei punti xnew
if ~isempty(xnew)
    ynew = polyval(coeff_pol,xnew);
    varargout{1} = ynew;
end
```

Osservazione: Nella funzione si pongono a zero gli errori di approssimazione che risultano inferiori o uguali a $0.5 \cdot 10^{-14}$.

Esempio 1

Dal Command Window Supponiamo di conoscere i valori della funzione $f(x) = x^4$ (incognita) sulla griglia degli interi x tali che $1 \leq x \leq 100$. Trovare il polinomio di migliore approssimazione per i dati a disposizione. A tal fine usiamo la funzione find_bestpol.m.

```
>> x = [1:100];
>> y = x.^4;
>> [coeff_pol] = find_bestpol(x,y);
coeff_poli{1} =
  1.0e+007 *
   0.08180798000000 -2.08096966000000

coeff_poli{2} =
  1.0e+006 *
   0.01744342857143 -0.94370648571429  9.14067025714289
```

```
coeff_poli{3} =
  1.0e+006 *
   0.00020200000000 -0.01315957142857  0.29881571428571 -1.57650034285711

coeff_poli{4} =
  Columns 1 through 4
   1.00000000000000 -0.00000000000034  0.00000000002487 -0.00000000072642
  Column 5
   0.00000000618095

coeff_poli{5} =
  Columns 1 through 4
  -0.00000000000000  1.00000000000001 -0.00000000000078  0.00000000003372
  Columns 5 through 6
  -0.00000000060876   0.00000000322184

coeff_poli{6} =
  Columns 1 through 4
   0.00000000000000 -0.00000000000000  1.00000000000005 -0.00000000000290
  Columns 5 through 7
   0.00000000008301 -0.00000000106551  0.00000000484500
```

```
coeff_poli{7} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  1.00000000000029
  Columns 5 through 8
  -0.00000000001384  0.00000000034828 -0.00000000413917  0.00000001714164

coeff_poli{8} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  0.00000000000004
  Columns 5 through 8
   0.99999999999783  0.00000000006844 -0.00000000118606  0.00000000992706
  Column 9
  -0.00000002894008

coeff_poli{9} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  0.00000000000000
  Columns 5 through 8
  -0.00000000000013  1.00000000000529 -0.00000000012804  0.00000000172519
  Columns 9 through 10
  -0.00000001116357  0.00000002460266
```

```
coeff_poli{10} =
  Columns 1 through 4
   0.00000000000000 -0.00000000000000  0.00000000000000 -0.00000000000000
  Columns 5 through 8
   0.00000000000000 -0.00000000000005  1.00000000000123 -0.00000000001540
  Columns 9 through 11
   0.00000000005338  0.00000000051068 -0.00000000259616

errore =
 1.0e+016 *
  Columns 1 through 4
   1.83734693275675  0.14778539999400  0.00226077716367  0.00000000000000
  Columns 5 through 8
   0.00000000000000  0.00000000000000  0.00000000000000  0.00000000000000
  Columns 9 through 10
   0.00000000000000  0.00000000000000
```

L'errore di approssimazione decresce al crescere del grado del polinomio approssimante ma diventa molto piccolo già dal quarto grado. Infatti, considerando i coefficienti del polinomio approssimante di grado 5,

```
>> coeff_pol
coeff_pol =
  Columns 1 through 4
  -0.00000000000000   1.00000000000001 -0.00000000000078   0.00000000003372

  Columns 5 through 6
  -0.00000000060876   0.00000000322184
```

si osserva che il coefficiente del monomio di grado massimo **è nullo** in precisione di macchina, mentre il coefficiente relativo al monomio di grado 4 è proprio 1

Se i dati in input sono affetti da errore

```
>> noise = 10^6*randn(1,length(y));
>> ydata = y + noise;
>> [coeff_pol_noise] = find_bestpol(x,ydata);
```

L'errore di approssimazione non cambia in modo significativo dal quarto grado in poi.

```
errore =
1.0e+016 *
  Columns 1 through 4
   1.83865002747658   0.14958824155293   0.01023540754299   0.00942206611061
  Columns 5 through 8
   0.00941161142473   0.00937567370055   0.00935613603876   0.00934454980644
  Columns 9 through 10
   0.00933311094961   0.00933042093266
>> coeff_pol_noise
coeff_pol_noise =
  1.0e+006 *
  Columns 1 through 4
   0.00000000000000 -0.00000000000006   0.00000000001192 -0.00000000114151
  Columns 5 through 8
   0.00000005584007 -0.00000083139602 -0.00004604518277   0.00264017610832
  Columns 9 through 11
  -0.05305544205054   0.44998990122359 -1.12224028234669
```

Esempio 2

Valutare la complessità asintotica di un algoritmo di ordinamento di cui si conoscono i tempi di esecuzione al variare della lunghezza $\mathbf{n}$ del dato di input, come riportato in tabella:

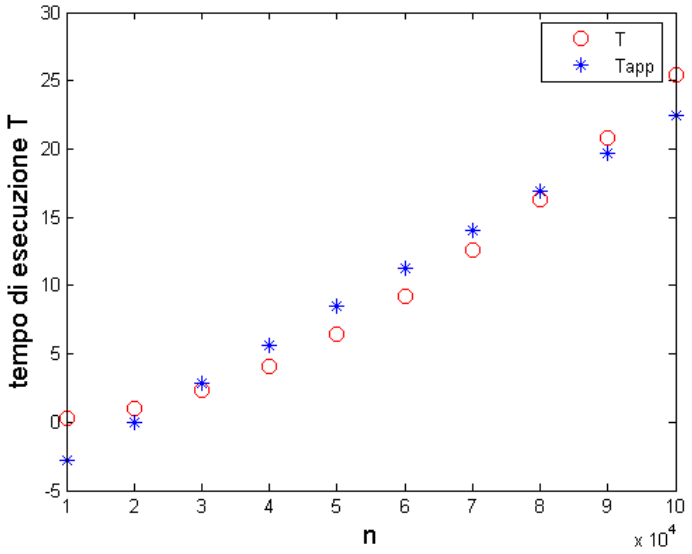
n	T (sec)
10000	0.312002
20000	0.998406
30000	2.293215
40000	4.102826
50000	6.411641
60000	9.204059
70000	12.604881
80000	16.317705
90000	20.826134
100000	25.474963

sapendo che il tempo di calcolo è una funzione polinomiale rispetto alla variabile $\mathbf{n}$ — precisamente $\mathbf{T = K\, n^P}$.

Soluzione

```
>> n = 10000:10000:100000;
>> T = [0.312002 0.998406 2.293215 4.102826 6.411641 9.204059...
        12.604881 16.317705 20.826134 25.474963];
>> [C,Tapp] = find_bestpol_mod(n,T,n,1);
% e' la funzione find_bestpol in cui il quarto
% input e' il grado massimo consentito al polinomio
% approssimante
% Valutiamo l'errore di approssimazione di T usando
% un polinomio di primo grado
coeff_poli{1} =
   0.0002810192593   -5.6014760666666
errore=
  34.21648743086648
```

Il grafico riporta i valori del tempo di calcolo T e quelli dati dalla approssimazione con un **polinomio di primo grado**

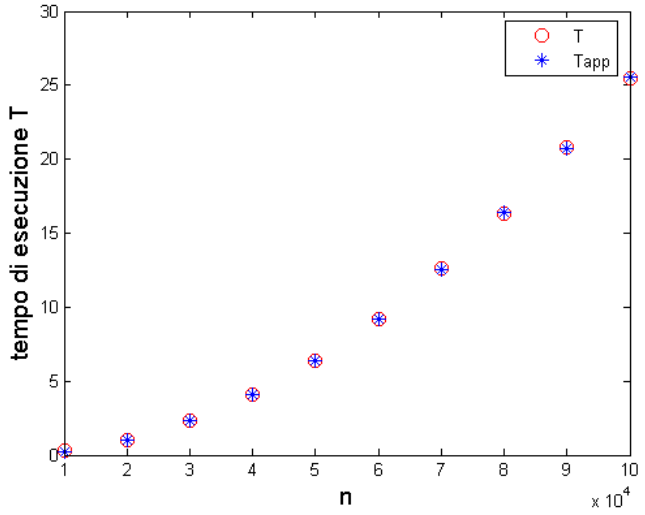



```
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```

Il tempo di calcolo T non dipende in modo perfettamente lineare da n.

Consideriamo ora un **polinomio di secondo grado**

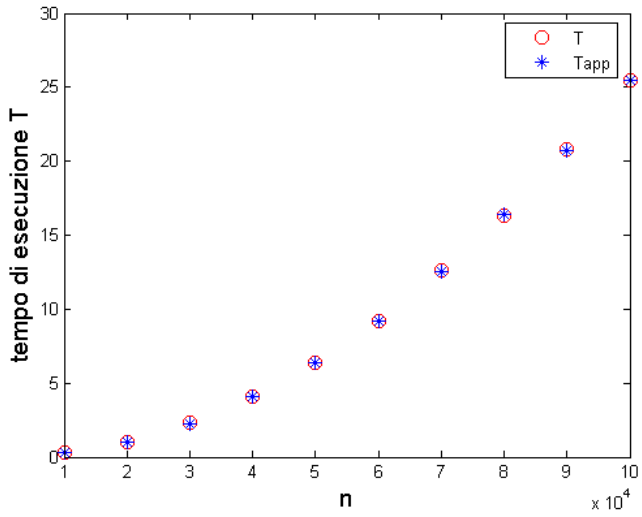
```
>> [C,Tapp] = find_bestpol_mod(n,T,n,2);
coeff_poli{2} =
   0.00000000254447   0.00000112747189  -0.00364031666667
errore=
  0.0320164301631 6
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```



In questo caso, l'errore di approssimazione è molto più piccolo e i valori di T sono ben approssimati da un ramo di parabola.

Considerando ora un **polinomio di terzo grado**

```
>> [C,Tapp] = find_bestpol_mod(n,T,n,3);
coeff_poli{3} =
  -0.00000000000000   0.00000000282247 -0.00001169437438   0.14092136666668
errore=
   0.02324800486742
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```



l'errore diminuisce anche se non in modo considerevole. Infatti, osservando il coefficienti del polinomio approssimante di terzo grado, si nota che il coefficiente del monomio di grado massimo è praticamente nullo (in precisione di macchina).

Quindi, si può concludere che la dipendenza del tempo di calcolo T dal numero di dati n è ben descritta da un polinomio di secondo grado.

Approssimazione trigonometrica

Se la funzione o i dati che si vogliono approssimare hanno un andamento **periodico** si sceglie come classe di **funzioni approssimanti** l'insieme $\mathcal{T}_N$ dei **polinomi trigonometrici**.

$$\mathcal{T}_N := \left\{ T_N(x) = \sum_{k=0}^{N} \Big(a_k \cos(k\,x) + b_k \sin(k\,x) \Big), a_k, b_k \in \mathbf{R} \;\; \forall k, x \in [0, 2\pi) \right\}$$

- **Interpolazione**: si usa quando si vogliono interpolare pochi **dati periodici** non affetti da errori
- **Approssimazione discreta ai minimi quadrati**: si usa quando si vogliono interpolare molti **dati periodici** affetti da **errori**

Approssimazione ai minimi quadrati di dati discreti periodici

Tabella: $\{x_i, y_i\}$ con nodi $x_i = i\,\dfrac{2\pi}{n+1}$, $i = 0, 1, ..., n$, **equispaziati**

Funzione approssimante:

$$T_M(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \Big(a_k \cos(k\,x) + b_k \sin(k\,x) \Big) \qquad \boxed{n+1 >> 2M+1}$$

Funzioni di base:

$\psi_0(x) = 1,\ \psi_1(x) = \cos(x),\ \psi_2(x) = \sin(x),\ \ldots\ \psi_{2M}(x) = \cos(Mx),\ \psi_{2M+1}(x) = \sin(Mx)$

Metodo di approssimazione: si **minimizza** lo **scarto quadratico**

$$\sigma^2(a_0, ..., a_M, b_1, ..., b_M) = \sum_{i=0}^{n} \Big[\underbrace{\frac{a_0}{2} + \sum_{k=1}^{M} \Big(a_k \cos(k\,x_i) + b_k \sin(k\,x_i) \Big)}_{T_M(x_i)} - y_i \Big]^2$$

Risolvere il problema dell'**approssimazione trigonometrica ai minimi quadrati** equivale a trovare i valori dei $2M+1$ coefficienti a_k, b_k che rendono minimo σ

Soluzione del sistema delle equazioni normali

Dalle **condizioni di ortogonalità** segue che la matrice H_t è una matrice **diagonale**.

$$H_t = diag\left(n+1, \frac{n+1}{2}, ..., \frac{n+1}{2}\right)$$

$\Rightarrow$ H_t **regolare** e **facilmente invertibile**

$\Rightarrow$

$$\begin{cases} a_k = \dfrac{2}{n+1} \displaystyle\sum_{i=0}^{n} y_i \cos(k\, x_i) & k = 0, 1, 2, ..., M \\ b_k = \dfrac{2}{n+1} \displaystyle\sum_{i=0}^{n} y_i \sin(k\, x_i) & k = 1, 2, ..., M \end{cases}$$

$\Rightarrow$

$$T_M(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \left(a_k \cos(k\, x) + b_k \sin(k\, x)\right)$$

Esercizio

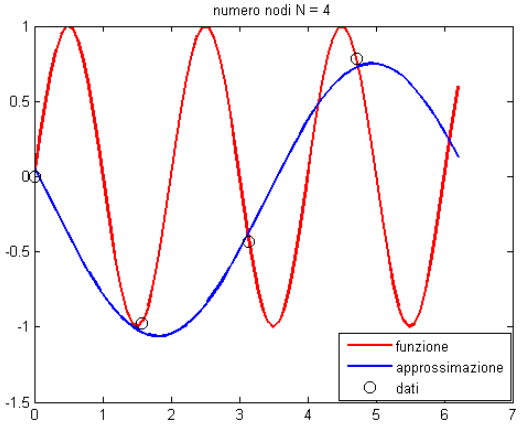
Si consideri i valori della funzione $f(x) = sin(\pi x)$ nei nodi $x_i, \quad i = 0, \ldots, N-1$, con $[x_0, x_{N-1}] = [-\pi, \pi]$. Scegliendo $N = 4, 8, ..., 40$, si determini il polinomio trigonometrico di migliore approssimazione per f usando la tecnica dei minimi quadrati.

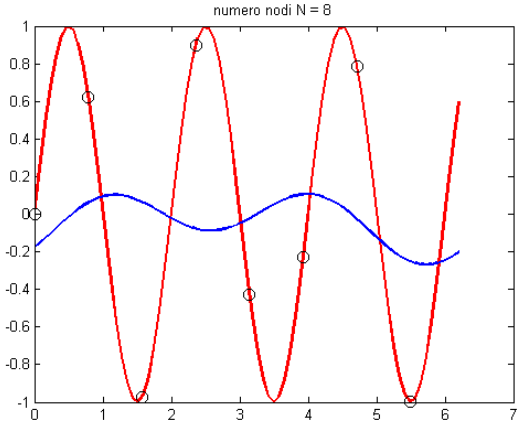
Soluzione

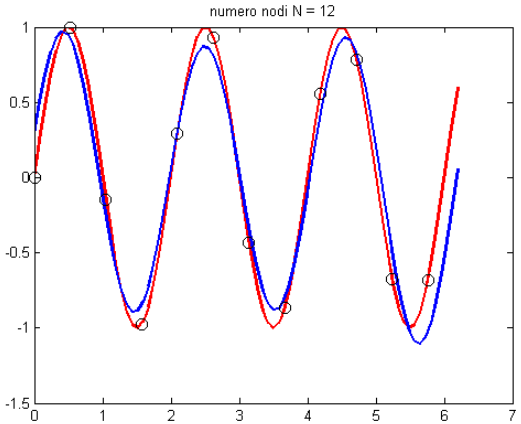
Scegliendo come grado del polinomio approssimante $\mathrm{M} = \frac{N}{4}$

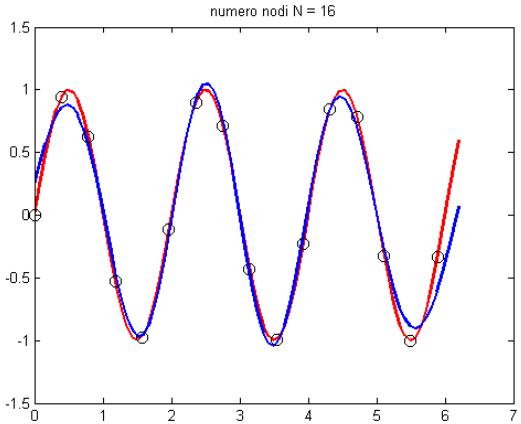
```
x = linspace(0,2*pi,80);
x = x(1:end-1);
y = (sin(pi*x));
for N=4:4:40,
    xdata = linspace(0,2*pi,N+1);
    xdata = xdata(1:end-1);
    ydata = (sin(pi*xdata));
    deg = 0.5*.5*N;

    a0 = 2*mean(ydata);
    for k = 1:deg
        a(k) = 2*mean(ydata.*cos(k*xdata));
        b(k) = 2*mean(ydata.*sin(k*xdata));
    end
    a0
    a
    b
```

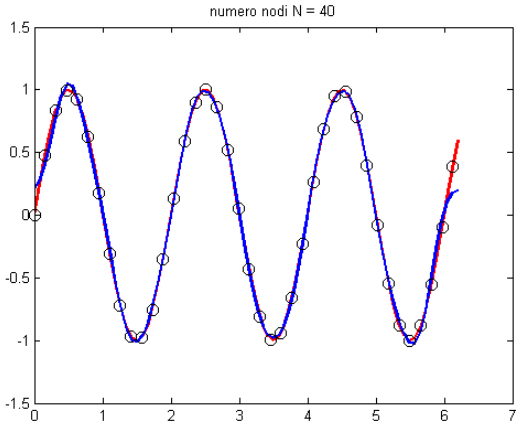
```
    for i = 1:length(x)
        ynew(i) = a0/2 + sum(a.*cos((1:deg)*x(i))+b.*sin((1:deg)*x(i)));
    end
    sol(N) = deg;
    err(N) = sum((ynew-y).^2);
    close all
    figure, plot(x,y,'r')
    hold on, plot(x,ynew), title(['numero nodi N = ', int2str(N)]),
    hold on, plot(xdata,ydata,'ko')
    legend('funzione','approssimazione','dati')
    pause
end

figure, plot((4:4:40)/4,err(4:4:40),'b')
xlabel('GRADO DEL POLINOMIO')
ylabel('ERRORE DI APPROSSIMAZIONE')
```

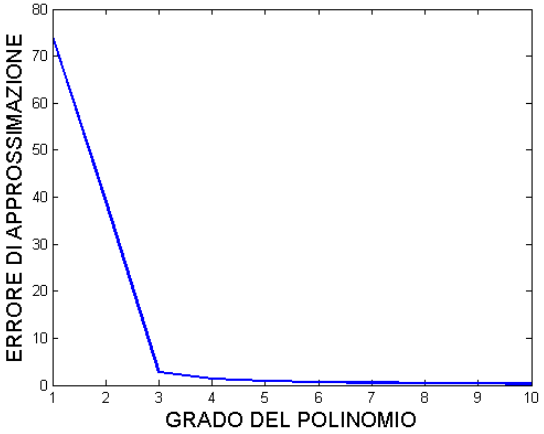

numero nodi N = 4
funzione
approssimazione
dati


numero nodi N = 8


numero nodi N = 12


numero nodi N = 16


numero nodi N = 40


ERRORE DI APPROSSIMAZIONE
GRADO DEL POLINOMIO

Un polinomio trigonometrico di grado 3 da una buona approssimazione dei dati. Infatti,

N = 4 — M = 1
a0 = -0.31006818968544
a = 0.21515060850005
b = -0.88045039089824

N = 8 — M = N/4 = 2
a0 = -0.08088335165033
a = -0.07769069212328 -0.06011651365733
b = 0.04613579986537 0.12430257838197

N = 12 — M = N/4 = 3
a0 = -0.03608498922724
a =
-0.03200163222932 -0.01112915902980 0.35074962121409
b = 0.03443149430075 0.09766155209760 0.87400978861193

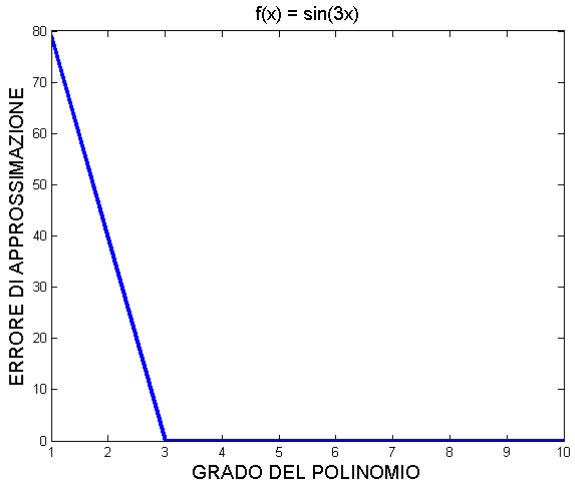
N = 16 — M = 4
a0 = -0.01591648661614
a =
-0.01172798848824 0.00948242091122 0.37201044630301
-0.11459241901756
b = 0.03132617931605 0.09122242485242 0.86371057309652
-0.14679867272115

N = 20 — M = 5
a0 = -0.00441794345553
a =
-0.00020396789852 0.02108565921460 0.38375564688870
-0.10262507597221 -0.06686884940153
b =
0.03002051144892 0.08856539913823 0.85960452458489
-0.15251781640554 -0.07046562882684

N = 40 — M = 10
a0 = 0.01733521051493
a =
0.02156455160120 0.04290111852377 0.40565225296549
-0.08060836225204 -0.04468555509506 -0.03439357564827
-0.02969891455888 -0.02709256736246 -0.02547642878236
-0.02440087841547
b =
0.02839441358101 0.08528900564644 0.85462788145124
-0.15927376529686 -0.07911469386633 -0.05364271498254
-0.04055441098598 -0.03231006968735
-0.02648544660747 -0.02205241174289

Il contributo maggiore è dato dai coefficienti a_3 e b_3.

Considerando la funzione $f(x) = sin(3x)$ si ha il seguente errore

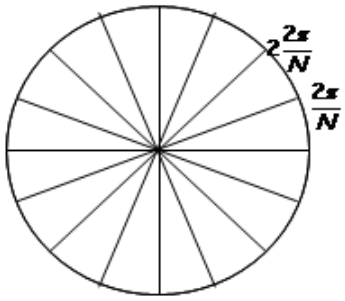


In questo caso, infatti,

a0 = 1.110223024625157e-016
a = 1.0e-015 *
0.09251858538543 0.20354088784795 0.35729181319074
b =
-0.00000000000000 0.00000000000000 1.00000000000000

i coefficienti $a_k \quad \forall k$ e $b_k, \quad \forall k \neq 3$ sono trascurabili (quasi nulli), mentre l'unico contributo è dato dal coefficiente $b_3 = 1$. La funzione $f(x) = sen(3x)$ è un elemento della base trigonometrica e quindi la sua approssimazione trigonometrica è esatta.

Trasformata di Fourier discreta



w = fft(v) calcola la trasformata di Fourier discreta del vettore v

$$\sum_{k=1}^{N} v(k) e^{-i\frac{2\pi}{N}(k-1)(n-1)}, \qquad 1 \le n \le N$$

v = ifft(w) trasformata di Fourier inversa 1D

$$\frac{1}{N}\sum_{n=1}^{N} w(n) e^{i\frac{2\pi}{N}(k-1)(n-1)}, \qquad 1 \le k \le N$$

Nota: i vettori v e w hanno indici strettamente positivi e quindi le sommatorie hanno indici compresi tra 1 e N, dove N è la lunghezza dei vettori v e w

Trasformata di Fourier discreta

Si può visualizzare usando modulo e fase, oppure parte reale e parte immaginaria

$w = Re(w) + i\ Im(w)$

$w = |w|\ e^{-i\phi(w)}$,

$|w| = (Re^2(w) + Im^2(w))^{1/2}$

$\phi(w) = arctg(Im(w)/Re(w))$

Real(w) restituisce la parte reale di w (w può essere un numero, un vettore o una matrice)

Imag(w) restituisce la parte immaginaria di w

Abs(w) restituisce il valore assoluto di w

Angle(w) restituisce la fase di w (2π-p)

Unwrap(angle(w)) restituisce la fase di w non periodica

Trasformata di Fourier discreta

Esempio:

```
>> v = [ones(1,5) zeros(1,125)];
>> w = fft(v);
>> figure, plot(real(w))
>> figure, plot(imag(w))
>> figure, plot(abs(w))
>> figure, plot(angle(w))
```

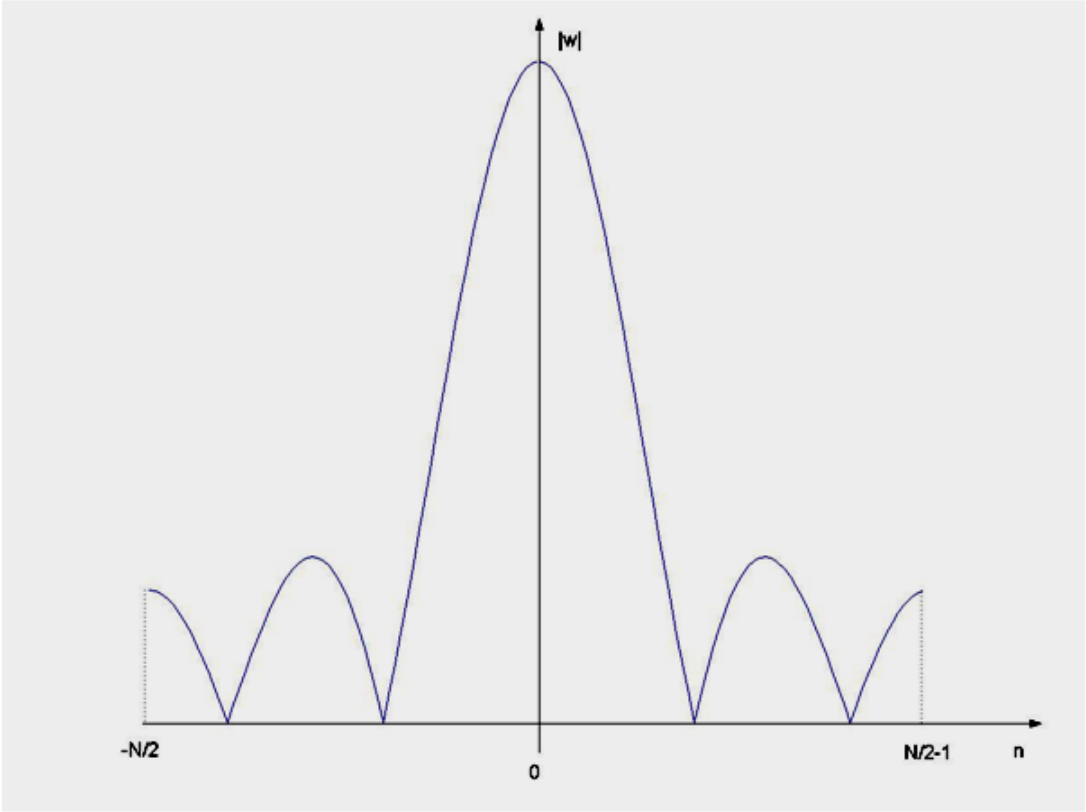
Osservazioni: Si può verificare che w(x+N) = w(x). Quindi w è periodica di periodo N.
Per visualizzare la fft simmetrica rispetto allo zero, si usa il comando fftshift

fftshift(w) trasla il vettore w di metà periodo verso destra
ifftshift(w) è la funzione inversa di fftshift: trasla il vettore w di metà periodo verso sinistra

Trasformata di Fourier discreta

Esempio:

```
>> figure, plot(fftshift(abs(w)))
>> figure, plot(fftshift(angle(w)))
```



Interpolazione trigonometrica e FFT

Se $N+1$ è il numero di nodi e N è pari, il polinomio trigonometrico di grado $M = \frac{N}{2}$

$$T_M(x) = \frac{a_0}{2} + \sum_{k=1}^{M} a_k cos(kx) + \sum_{k=1}^{M} b_k sin(kx)$$

con

$$a_k = \frac{2}{N+1} \sum_{i=0}^{N} v_i cos(kx_i) \qquad k = 0, \ldots, M$$

$$b_k = \frac{2}{N+1} \sum_{i=0}^{N} v_i sin(kx_i) \qquad k = 0, \ldots, M$$

$$x_i = i\frac{2\pi}{N+1}, \qquad i = 0, \ldots, N$$

interpola la funzione $f(x)$ tale che $y_i = f(x_i)$.

Indicando con $\mathbf{y} = [y_0 y_1 \dots y_N]$ e calcolando

$$\mathbf{c} = \mathbf{fft}(\mathbf{y}),$$

risulta

$$a_k = \frac{1}{2}\Re(c_k) \qquad \forall\, k = 0, \dots, M$$

$$b_k = -\frac{1}{2}\Im(c_k) \qquad \forall\, k = 1, \dots, M$$

con

$$\mathbf{c} = [\mathbf{c_0 c_1} \dots \mathbf{c_M}].$$

L'uguaglianza precedente deriva considerando le **formule di Eulero** nella definizione dei coefficienti a_k e b_k

$$cos(kx_i) = \frac{e^{jkx_i} + e^{-jkx_i}}{2}$$

$$sin(kx_i) = \frac{e^{jkx_i} - e^{-jkx_i}}{2j}$$

con j unità immaginaria.

Riconsideriamo la funzione $f(x) = sin(\pi x), \qquad x \in [0, 2\pi]$

```
x = linspace(0,2*pi,80);
x = x(1:end-1);
y = (sin(pi*x));
N=8

xdata = linspace(0,2*pi,N+1);
xdata = xdata(1:end-1);
ydata = (sin(pi*xdata));
%il grado del polinomio trigonometrico interpolante e'
%la meta' del numero di nodi
M = .5*N;
```

```
% calcolo dei coefficienti del polinomio trigonometrico
a0 = 2*mean(ydata);
for k = 1:M
    a(k) = 2*mean(ydata.*cos(k*xdata));
    b(k) = 2*mean(ydata.*sin(k*xdata));
end
a0
a
b

% calcolo dei coefficienti del pol. trigonometrico  dalla fft
fydata = fft(ydata);

a0f = fydata(1)/(N/2)
af = real(fydata(2:length(ydata)/2+1))/(N/2)
bf = -imag(fydata(2:length(ydata)/2+1))/(N/2)
```

Dal **Command window**

```
N =
     8
```

Coefficienti trigonometrici calcolati usando la definizione classica

```
a0 =
  -0.08088335165033

a =
  -0.07769069212328  -0.06011651365733   0.29284130062333  -0.22918483803511

b =
   0.04613579986537   0.12430257838197   0.92658619076361  -0.00000000000000
```

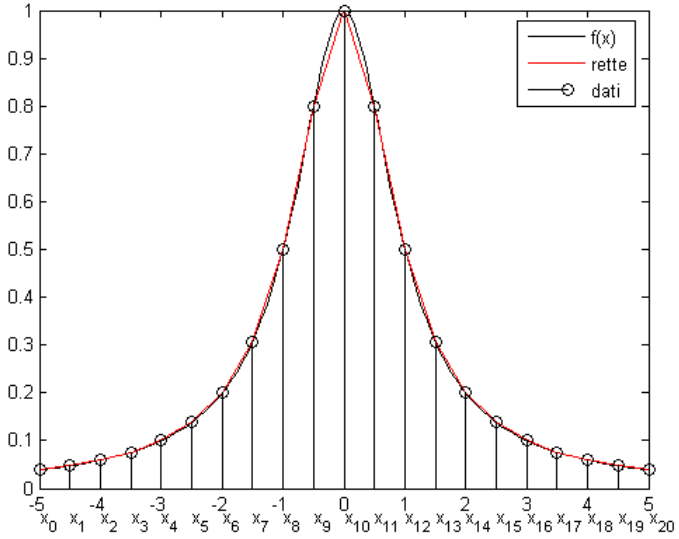
Coefficienti calcolati dallo output della funzione **fft**.

```
a0f =
  -0.08088335165033


af =
  -0.07769069212328  -0.06011651365733   0.29284130062333  -0.22918483803511

bf =
   0.04613579986537   0.12430257838197   0.92658619076361                  0
```

Formula dei trapezi:

$$\begin{cases} T_N(f) = \dfrac{h}{2}\left(f_0 + 2\sum_{j=1}^{N-1} f_j + f_N\right) \\ R_N^T(f) = -\left(\dfrac{b-a}{12}\right) h^2 f''(\tau) \qquad \tau \in [a,b] \end{cases}$$

Grado di precisione: $\nu = 1$



Esercizio

Scrivere la funzione Matlab **trapezi.m** che riceva come parametri di input i valori f_i di una funzione valutati su una griglia di punti equispaziati nell'intervallo $[a, b]$ e approssimi l'intergrale della funzione f nell'intervallo $[a, b]$ usando la formula dei trapezi. Usare la funzione per calcolare $\int_0^{-log(.5 \cdot 10^{-3})} e^{-x} cos^2(x) dx$ e confrontare i risultati con lo output della funzione **trapz.m** predefinita di Matlab.

```
function [int_value] = trapezi(fi,a,b)
%[int_value] = trapezi(fi,a,b,varargin)
% valuta l'integrale di una funzione f nell'intervallo [a,b]
% usando la formula dei trapezi, conoscendo il valore di f in punti
% equidistanti nell'intervallo [a,b]
%
% INPUT
%
% fi = valore della funzione in punti equidistanti dell'intervallo [a,b]
% a = estremo inferiore dell'intervallo di integrazione
% b = estremo superiore dell'intervallo di integrazione
%
% int_value = approssimazione dell'integrale di f in [a,b]

N = length(fi); % numero di punti
h = (b-a)/(N-1); % distanza tra i punti

% calcolo del valore dell'integrale
int_value = (fi(1) + 2 * sum (fi(2:N-1)) + fi(N))*h/2;
```

Dal Command Window

```
>> f = @(x)[exp(-x).*((cos(x)).^2 )];
>> xi = linspace(0,-log((.5*10^(-3))),292);
>> fi = f(xi);

>> int_value = trapezi(fi,0,-log(.5*10^(-3)));
>> disp(int_value)
   0.59989905470163

>> int_value = trapz(xi,fi);
>> disp(int_value)
   0.59989905470163
```

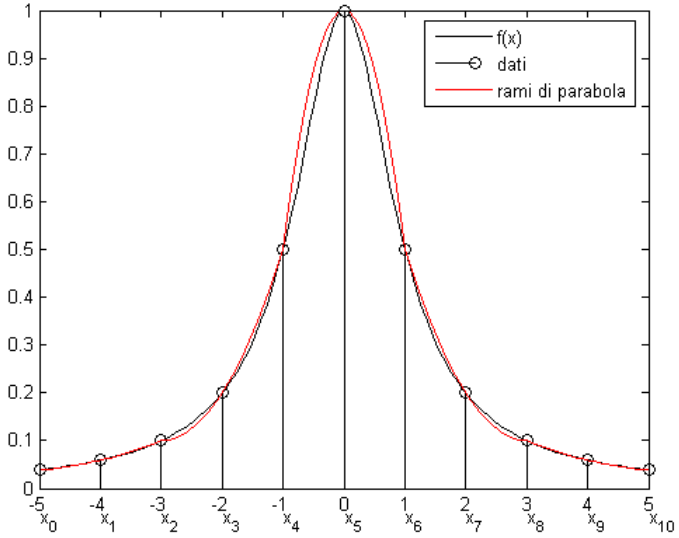
Esercizio

Si riscriva la funzione dell'esercizio precedente. I parametri di input sono: la funzione di cui si vuole calcolare l'integrale e il numero di punti N da usare nella formula dei trapezi.

Formula delle parabole:

$$\begin{cases} P_N(f) = \dfrac{h}{3}\left(f_0 + 4\sum_{j=0}^{N/2-1} f_{2j+1} + 2\sum_{j=1}^{N/2-1} f_{2j} + f_N \right) \\ R_N^P(f) = -\left(\dfrac{b-a}{180}\right) h^4 f^{(4)}(\tau) \qquad \tau \in [a,b] \end{cases}$$

Grado di precisione: $\nu = 3$



```
function [I] = parabole(xnodi,fnodi)
% I=parabole(xnodi,fnodi): Approssimazione di un integrale
% con la formula delle parabole
%
% INPUT
% xnodi = punti dell'intervallo di intergrazione
%         in cui si conosce il valore della funzione
% ynodi = valore della funzione da integrare valutata nei punti xnodi
%
% OUTPUT
% I = approssimazione dell'integrale

nnodi = length(xnodi);
a = xnodi(1);
b = xnodi(nnodi);
h = (b-a)/(nnodi-1);
I = h/3*(fnodi(1)+4*sum(fnodi(2:2:nnodi-1))+ ...
   2*sum(fnodi(3:2:nnodi-2))+fnodi(nnodi));
```

Usando la formula delle parabole per valutare l'integrale dell'esercizio precedente si ha

```
>> I = parabole(xi,fi)

I =

   0.59984121703556
```

Mentre usando la funzione predefinita di Matlab quad.m (usa la formula di Newton ricorsivamente in modo da ottenere un errore di almeno 10^{-6}).

```
>> quad(f,0,-log(.5*10^(-3)))

ans =

   0.59984217665775
```

Esercizio

Data la tabella

i	0	1	2	3	4
x_i	-0.5	-0.25	0	0.25	0.5
f_i	1.64872	1.28402	1.00000	0.77880	0.60653

relativa ai valori, affetti da errore, di una funzione $f(x) \in C^\infty(\mathbf{R})$,

1.1) approssimare $\int_{-0.5}^{0.5} f(x)dx$ usando la formula dei trapezi generalizzata su 5 nodi;

1.2) dare una stima dell'errore di troncamento;

1.3) stabilire se l'errore di propagazione può essere trascurato rispetto all'errore di troncamento motivando la risposta.

Soluzione

1.1 Utilizzando $N = 5$ nodi nell'intervallo $[a, b] = [-0.5, 0.5]$ si ha un passo

$$h = \frac{b-a}{N-1} = \frac{1}{4}.$$

Dunque, l' approssimazione dell'integrale è data da:

$$\begin{aligned} T_h(f) &= \frac{h}{2}(f_0 + 2\sum_{i=1}^{N-2} f_i + f_{N-1}) = \frac{h}{2}(f_0 + 2f_1 + 2f_2 + 2f_3 + f_4) = \\ &= 0.125(1.64872 + 2 \cdot 1.28402 + 2 \cdot 1 + 2 \cdot 0.7788 + 0.60653) = \\ &= 1.04761 \end{aligned}$$

1.2 Per valutare l'errore di troncamento, non avendo informazioni sulla derivata seconda di f, si applica la formula dei trapezi su tre nodi con passo $2\,h = 0.5$,

$$\begin{aligned} T_{2h}(f) &= \frac{2\,h}{2}(f_0 + 2f_2 + f_4) = \\ &= 0.25(1.64872 + 2 \cdot 1 + 0.60653) = 1.06381. \end{aligned}$$

Sapendo che $f(x) \in C^\infty$, si può utilizzare il criterio di Runge per la stima dell'errore di troncamento e quindi:

$$R_h(f) \approx \frac{T_h(f) - T_{2h}(f)}{3}$$

da cui

$$R_h(f) \approx \frac{1.04761 - 1.06381}{3} = -0.54 \cdot 10^{-2}$$

Usando l'estrapolazione di Richardson, la approssimazione dell'integrale migliora come segue

$$I(f) \approx T_h + (-0.54 \cdot 10^{-2}) = 1.04221.$$

1.3 I valori della funzione nei nodi sono dati con precisione $\epsilon = 0.5 \cdot 10^{-5}$. Poichè i coefficienti della formula di quadratura sono tutti positivi, risulta che l'errore di propagazione $R^*(f)$ è tale che

$$R^*(f) \leq \epsilon(b-a) = 0.5 \cdot 10^{-5}.$$

Quindi, **l'errore di propagazione è trascurabile rispetto all'errore di troncamento**.

Esercizio

Si determini

$$\int_0^{+\infty} cos^2(x)e^{-x}dx$$

a meno di un errore inferiore a 10^{-3}.

Soluzione

$f(x) = cos^2(x)e^{-x}$ è integrabile su $[0, +\infty)$, infatti

$$0 < \int_0^{+\infty} cos^2(x)e^{-x}dx < \int_0^{+\infty} e^{-x}dx = 1 < +\infty$$

Per poter applicare la formula dei trapezi, si scrive

$$I(f) = \int_0^{+\infty} cos^2(x)e^{-x}dx = \int_0^{\beta} cos^2(x)e^{-x}dx + \int_{\beta}^{+\infty} cos^2(x)e^{-x}dx$$

e si cerca $\beta > 0$ tale che

$$\int_{\beta}^{+\infty} cos^2(x)e^{-x}dx < 0.5 \cdot 10^{-3}.$$

In questo modo la formula dei trapezi si applica all'integrale

$$\int_0^{\beta} cos^2(x)e^{-x}dx$$

Si osserva che

$$\int_{\beta}^{+\infty} cos^2(x)e^{-x}dx < \int_{\beta}^{+\infty} e^{-x}dx = e^{-\beta} \leq 0.5 \cdot 10^{-3} \Leftrightarrow$$

$$\Leftrightarrow \beta \geq -ln(0.5 \cdot 10^{-3}) \approx 7.601.$$

Scegliendo il valore di β più piccolo, cioè $\beta = -ln(0.5 \cdot 10^{-3})$, si vuole determinare in quanti intervalli di uguale lunghezza è necessario suddividere l'intervallo $[0, \beta]$ affinchè la formula dei trapezi produca una stima dell'intergrale della funzione in questo intervallo con un errore inferiore a $0.5 \cdot 10^{-3}$.

A tal fine consideriamo l'espressione dell'errore di troncamento della formula dei trapezi, cioè

$$R_N^T(f) = -\left(\frac{b-a}{12}\right) h^2 f''(\tau) \qquad \tau \in [a, b]$$

e richiediamo

$$|R_N^T(f)| < 0.5 \cdot 10^{-3}.$$

Si osserva che $b - a = \beta$, mentre

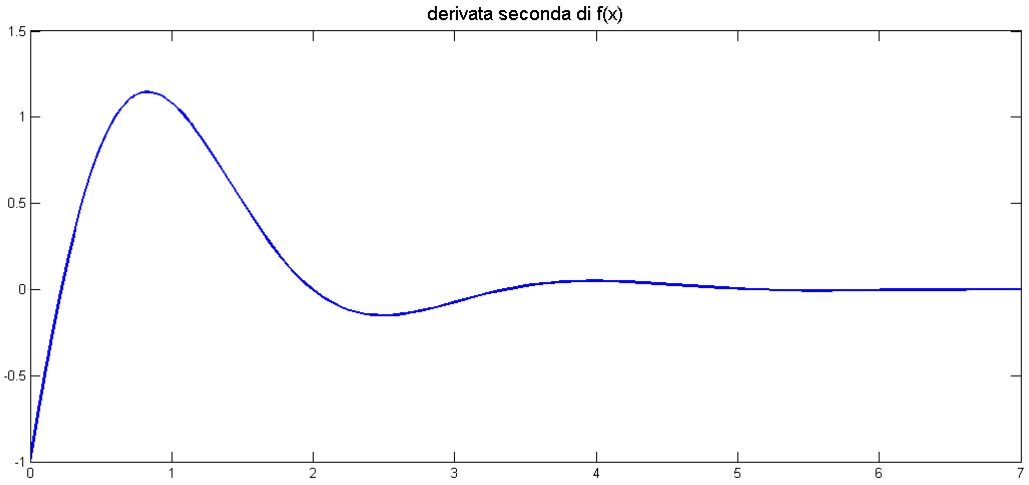
$$|R_N^T(f)| = \left(\frac{\beta}{12}\right) h^2 |f''(\tau)| \leq \left(\frac{\beta}{12}\right) h^2 \max_{x \in [0,\beta]} |f''(x)|.$$

Poichè

$$f''(x) = e^{-x}(2sin(2x) + cos^2(x) - 2cos(2x))$$

allora

$$max_{x \in [0\beta]} |f''(x)| \leq 1.15$$



Da cui

$$|R_N^T(f)| < 0.5 \cdot 10^{-3} \Leftrightarrow \left(\frac{\beta}{12}\right) h^2 \max_{x \in [0,\beta]} |f''(x)| < 0.5 \cdot 10^{-3}$$

cioè

$$h^2 \leq \frac{0.5 \cdot 10^{-3}}{\max_{x \in [0,\beta]} |f''(x)|} \frac{12}{\beta} = \frac{0.5 \cdot 10^{-3}}{1.15} \frac{12}{-ln(0.5 \cdot 10^{-3})} = 6.86 \cdot 10^{-4}$$

e quindi

$$h \leq 0.0262.$$

Se $N+1$ è il numero di punti da usare nella formula dei trapezi, allora

$$h = \frac{b-a}{N} = \frac{\beta}{N},$$

cioè

$$N \geq \frac{-ln(0.5 \cdot 10^{-3})}{0.0262} = 290.11$$

e quindi, essendo N un numero intero,

$$N \geq 291.$$

Quindi, sono necessari almeno **292** punti nella formula dei trapezi per raggiungere la precisione richiesta.