Polynomial interpolation

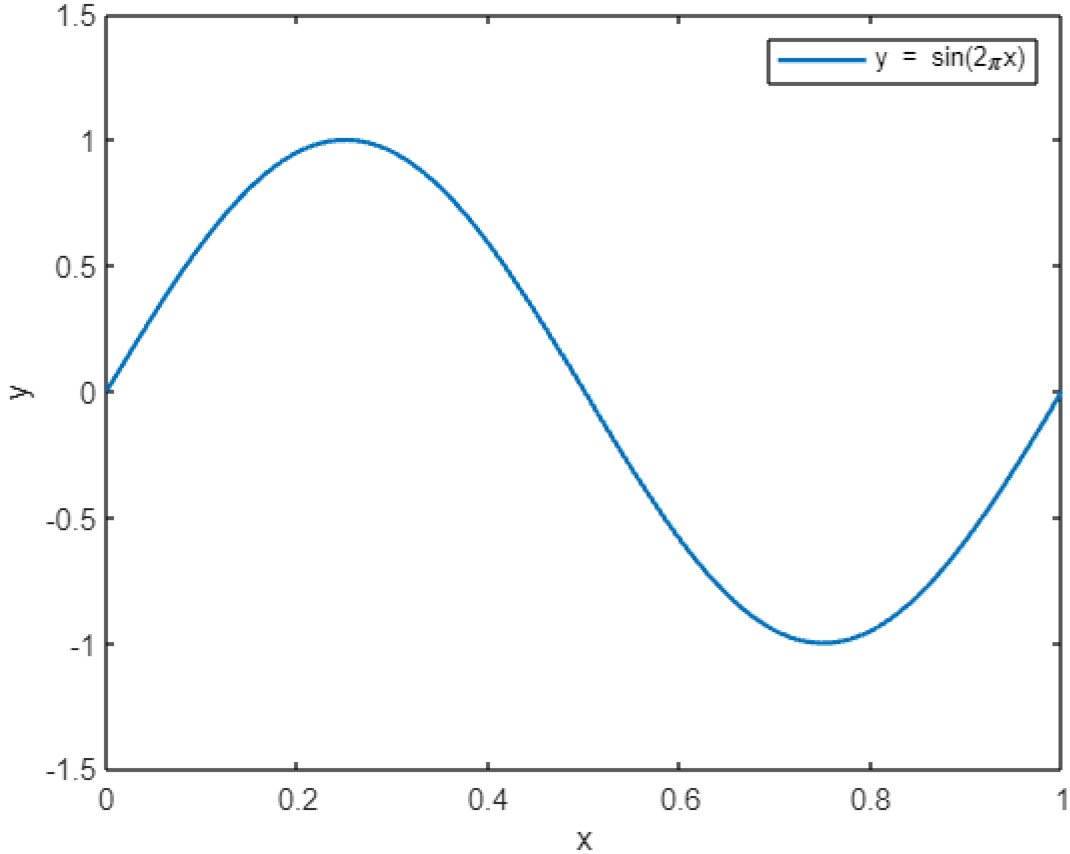
Nel seguito di questo notebook verrà presentato un approccio alla base del ML: fitting polinomiale. Nello specifico verranno generati dei punti casualmente distribuiti attorno alla funzione seno e si otterà il polinomio interpolante (overfitting) che attraversa tutti i punti (di learning). In seguito si rappresenteranno polinomi aventi gradi inferiori a $n-1$ (dove n rappresenta il numero di punti precedentemente generati) e si calcolerà lo scarto quadratico medio o root mean square error E_{RMS} per ogni grado.

```
% cleaning enviroment
clc
clear
```

Rappresento la funzione $y = sin(2\pi x)$ con $0 \le x \le 1$

```
% funzione seno
sen = @(x) sin(2*pi*x);

% genero vettori
x = linspace(0,1,100);
y = sen(x);
```

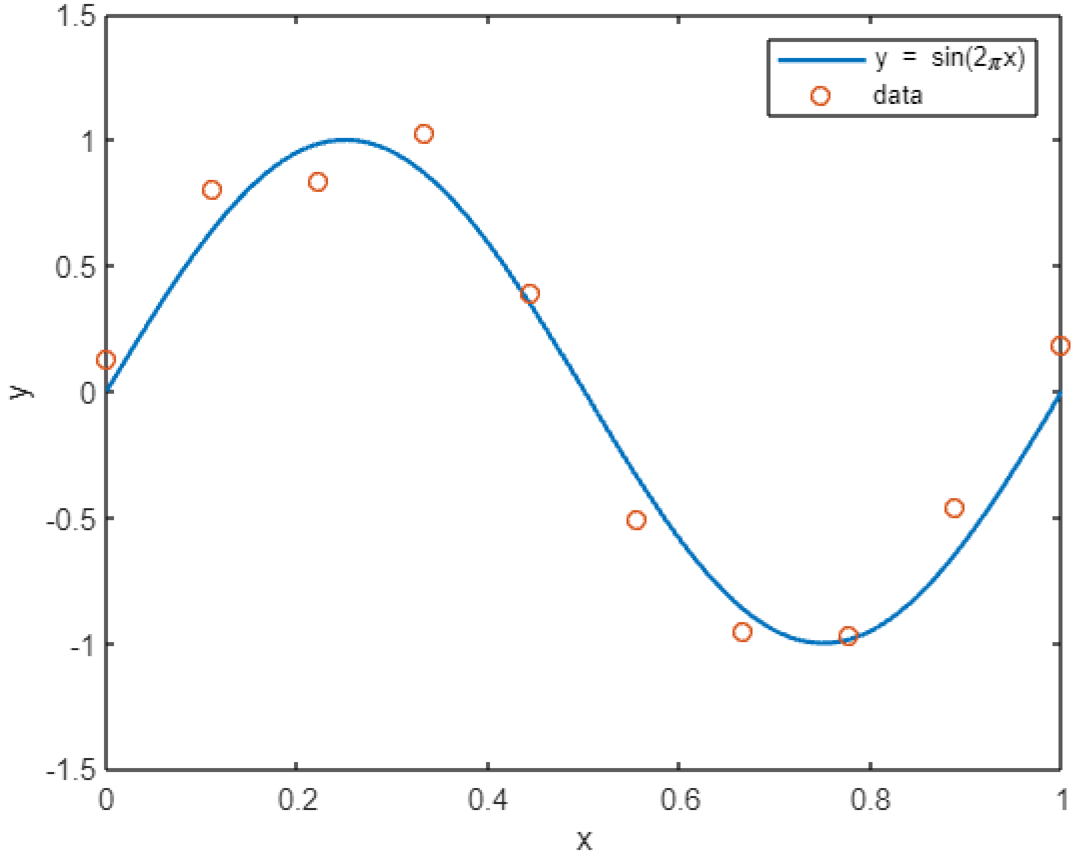
```
% plotto funzione seno
lw = 1.5; % plot line width
figure;
plot(x,y,"LineWidth",lw)
xlabel("x")
ylabel("y")
legend("y = sin(2\pix)")
xlim([0 1])
ylim([-1.5 1.5])
```



Genero set di learning avente `n_lrn` punti randomicamente distribuiti attorno alla funzione seno

```
% genero set di learning
n_lrn = 10;
x_lrn = linspace(0,1,n_lrn);
% rumore
eps = 0.2;
y_lrn = sin(2*pi*x_lrn) + rand_between(-eps,eps,n_lrn)';
```

```
% rappresento punti
figure;
plot(x,y,"LineWidth",lw)
hold on
plot(x_lrn,y_lrn,"o","LineWidth",1)
legend("y = sin(2\pix)","data")
xlabel("x")
ylabel("y")
hold off
xlim([0 1])
ylim([-1.5 1.5])
```



Il **polinomio interpolante** è quel polinomio la cui curva passa attraverso tutti i punti sperimentali. Se il polinomio ha forma generale $y = a_1 + a_2x + a_3x^3 + ... + a_nx^{n-1}$ dove

- n è il numero di punti da fittare;
- a_1, a_2, ..., a_n sono gli n coefficienti del polinomio;

Posto $\overline{x} = (x_1, x_2, ..., x_n)$, affinchè il polinomio attraversi tutti i punti del vettore $\overline{x}$ deve verificare le seguenti condizioni:

- $y_1 = a_1 + a_2x_1 + a_3x_1^3 + ... + a_nx_1^{n-1}$ (condizione passaggio per il punto x_1)
- $y_2 = a_1 + a_2x_2 + a_3x_2^3 + ... + a_nx_2^{n-1}$ (condizione passaggio per il punto x_2)
- ...
- $y_n = a_1 + a_2x_n + a_3x_n^3 + ... + a_nx_n^{n-1}$ (condizione passaggio per il punto x_n)

che rappresenta un sistema di n equazioni in n incognite. Il nostro obiettivo consiste nel risolvere il sistema per determinare gli n coefficienti a_1, a_2, ..., a_n e quindi il polinomio interpolante.

Sfruttando il formalismo matriciale è possibile rappresentare il sistema di n equazioni come di seguito

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{pmatrix}.$$

La matrice

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix}$$

prende il nome "matrice di Vandermonde" e si genera elevando gli elementi del vettore $\bar{x}$ da 0 a $n-1$.

In MATLAB è possibile generare la matrice di Vandermonde utilizzando la funzione `vander()`

```
% genero matrice di Vandermonde
V = fliplr(vander(x_lrn))
```

```
V = 10×10
    1.0000         0         0         0         0         0         0         0 ···
    1.0000    0.1111    0.0123    0.0014    0.0002    0.0000    0.0000    0.0000
    1.0000    0.2222    0.0494    0.0110    0.0024    0.0005    0.0001    0.0000
    1.0000    0.3333    0.1111    0.0370    0.0123    0.0041    0.0014    0.0005
    1.0000    0.4444    0.1975    0.0878    0.0390    0.0173    0.0077    0.0034
    1.0000    0.5556    0.3086    0.1715    0.0953    0.0529    0.0294    0.0163
    1.0000    0.6667    0.4444    0.2963    0.1975    0.1317    0.0878    0.0585
    1.0000    0.7778    0.6049    0.4705    0.3660    0.2846    0.2214    0.1722
    1.0000    0.8889    0.7901    0.7023    0.6243    0.5549    0.4933    0.4385
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
ans = 
'\left(\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & \frac{1}{9} & \frac{1}{81} & \frac{1}{7
```

```
% latex(sym(V))
```

Risolvo il sistema e determino i coefficienti a_1, a_2, ..., a_n eseguendo il prodotto matriciale $V^{-1} \cdot \bar{y}$ in cui $\bar{y} = (y_1, y_2, ..., y_n)$

Alla luce della forma matriciale, è possibile determinare i coefficienti α eseguendo il prodotto righe per colonna tra l'inversa della matrice di Vandermonde e il vettore colonna y.

In MATLAB è possibile eseguire questa operazione sia sfruttando la funzione `pinv()` che determina la matrice pseudoinversa

```
% a = pinv(V)*(y_lrn')
```

oppure utilizzando la sintassi `V\y_lrn'`

```
% determino i coefficienti
```

```
a = V\y_lrn'
```

```
a = 10×1
10^4 ×
    0.0000
    0.0041
   -0.0662
    0.4598
   -1.6307
    3.1999
   -3.5579
    2.1369
   -0.5742
    0.0282
```

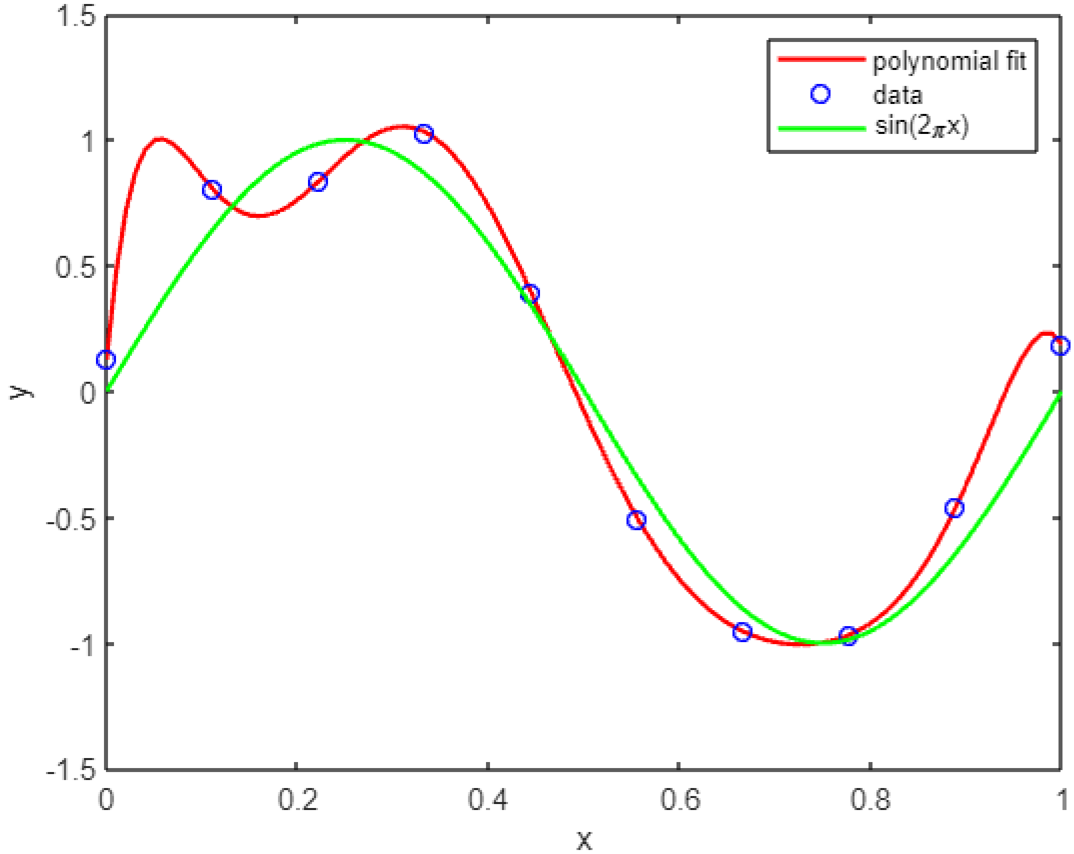
```
% ottengo il polinomio funzione degli scalari x e m (grado)
poly = @(x,m) (x.^(0:m))*(a(1:m+1));
```

Determiniamo i valori previsti

```
% over-fitting
% z = zeros(1,100);
% for i=1:100
%     z(i) = poly(x(i),n_lrn-1);
% end

% utilizzando la funzione poly_predict determino le ordinate previste dal
% modello
z = poly_predict(x,poly,n_lrn-1)
```

```
z = 1×100
    0.1259    0.4761    0.7175    0.8729    0.9617    1.0005    1.0031    0.9809 ⋯
```

```
% plotting predicted values
figure;
plot(x,z,"r","LineWidth",lw)
hold on
plot(x_lrn,y_lrn,'ob',"LineWidth",1)
plot(x,y,"g","LineWidth",lw)
hold off
legend("polynomial fit", "data", "sin(2\pix)")
xlabel("x")
ylabel("y")
ylim([-1.5 1.5])
xlim([0 1])
```

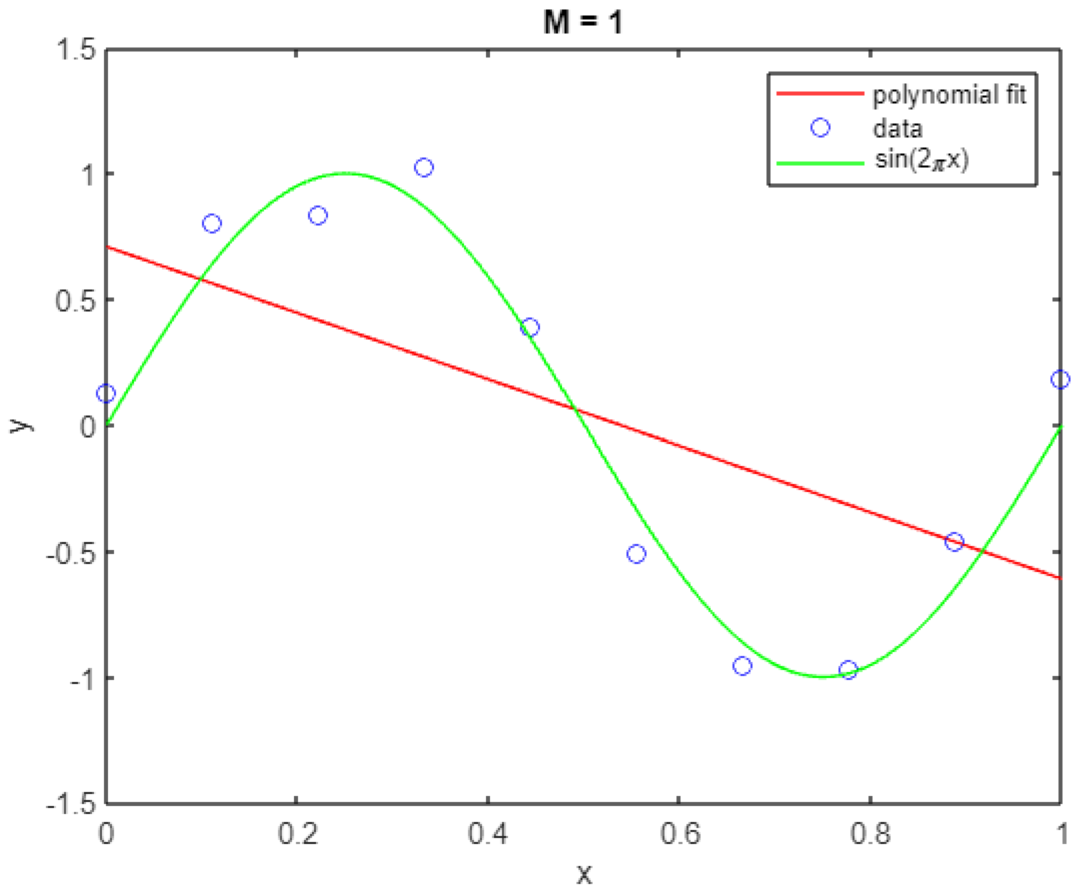


Cosa succede utilizzando polinomi di grado inferiore a $n-1$?

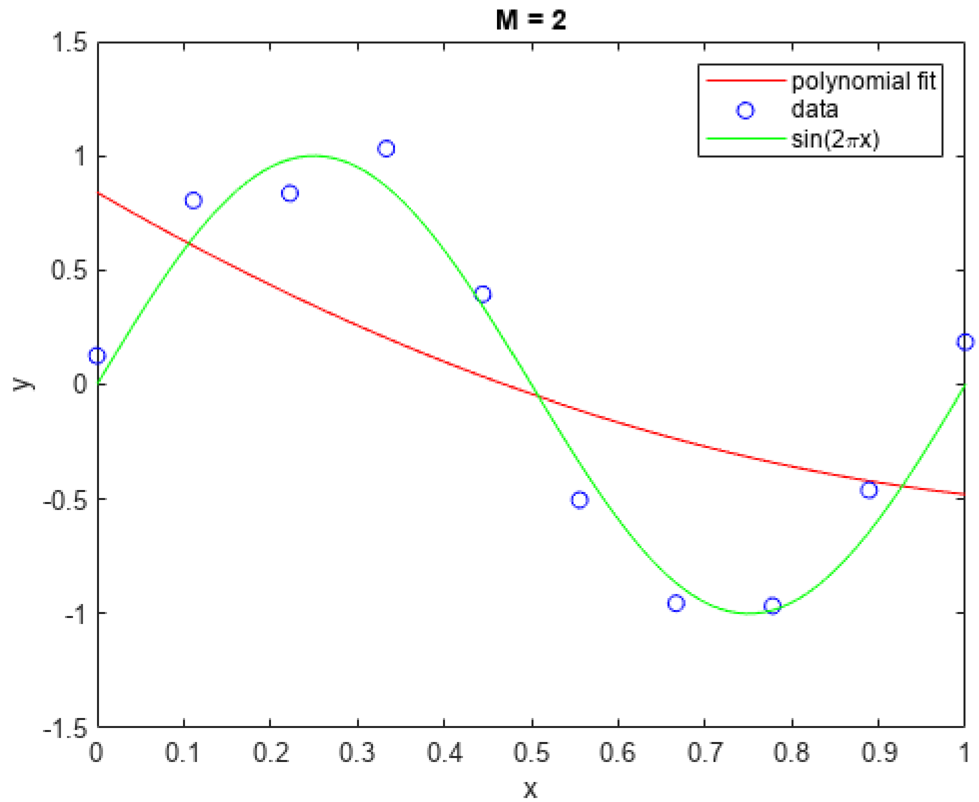
```
% plotting at different M (polynomial order)
for m = 1:3

    % funzione vander personalizzata che permette di costruire matrici di
    % Vandermonde incomplete in funzione del grado m fornito
    V = custom_vander(x_lrn,m)
    a = V\y_lrn'
    poly = @(x,m) (x.^(0:m))*(a(1:m+1));

    figure;
    plot(x,poly_predict(x,poly,m),"r")
    hold on
    plot(x_lrn,y_lrn,'ob')
    plot(x,y,"g")
    hold off
    legend("polynomial fit", "data", "sin(2\pix)")
    xlabel("x")
    ylabel("y")
    ylim([-1.5 1.5])
    xlim([0 1])
    title(sprintf("M = %d",m))
```

```
end
```

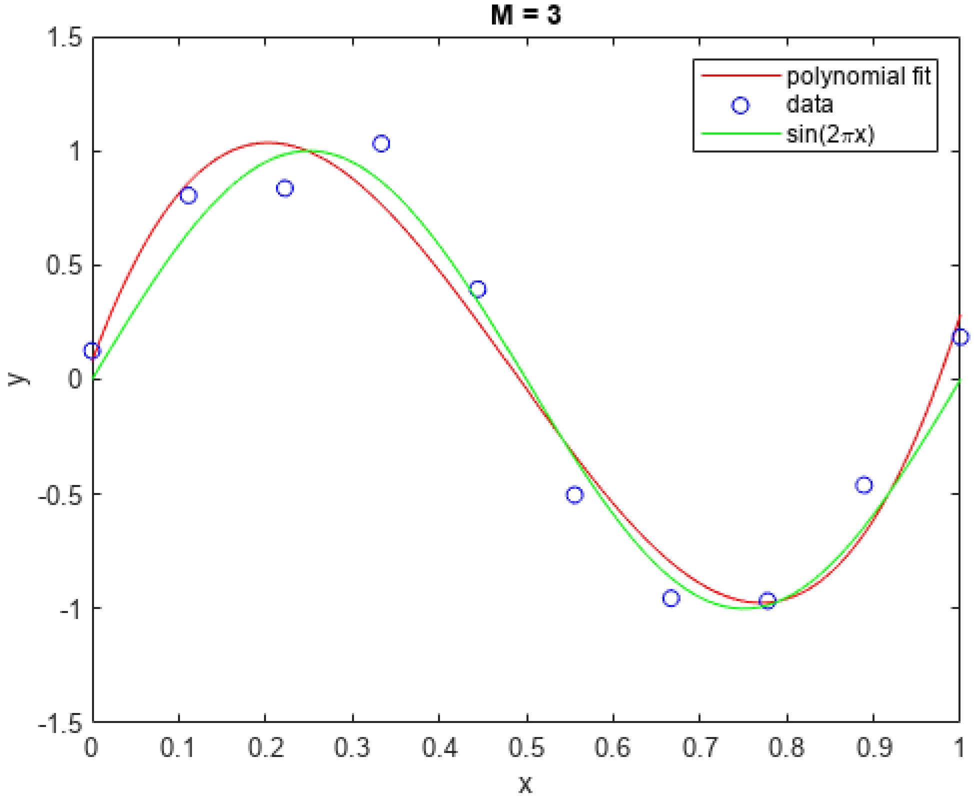
```
V = 10×2
    1.0000         0
    1.0000    0.1111
    1.0000    0.2222
    1.0000    0.3333
    1.0000    0.4444
    1.0000    0.5556
    1.0000    0.6667
    1.0000    0.7778
    1.0000    0.8889
    1.0000    1.0000
a = 2×1
    0.7089
   -1.3188
```




```
V = 10×3
    1.0000         0         0
    1.0000    0.1111    0.0123
    1.0000    0.2222    0.0494
    1.0000    0.3333    0.1111
    1.0000    0.4444    0.1975
    1.0000    0.5556    0.3086
    1.0000    0.6667    0.4444
    1.0000    0.7778    0.6049
    1.0000    0.8889    0.7901
    1.0000    1.0000    1.0000
a = 3×1
    0.8405
   -2.2067
    0.8880
```




```
V = 10×4
    1.0000         0         0         0
    1.0000    0.1111    0.0123    0.0014
    1.0000    0.2222    0.0494    0.0110
    1.0000    0.3333    0.1111    0.0370
    1.0000    0.4444    0.1975    0.0878
    1.0000    0.5556    0.3086    0.1715
    1.0000    0.6667    0.4444    0.2963
    1.0000    0.7778    0.6049    0.4705
    1.0000    0.8889    0.7901    0.7023
    1.0000    1.0000    1.0000    1.0000
a = 4×1
    0.0787
   10.3362
  -32.1700
   22.0386
```



Errore di learning

Per questa occasione utilizzeremo il root mean square error (o scarto quadratico medio)

$$E_{RMS} = \frac{1}{N}\sum_{i}^{N}(P_i - O_i)^2$$

dove

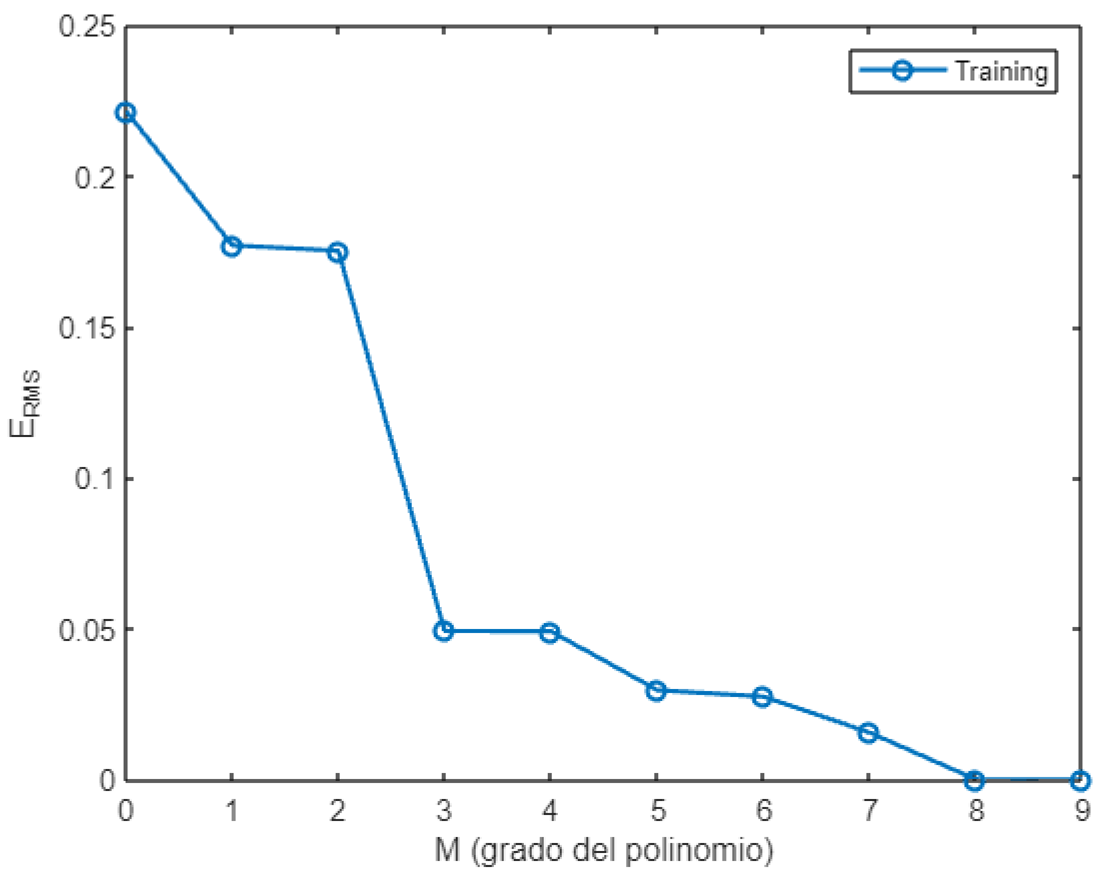
- N rappresenta il numero di punti;
- P_i il valore previsto;
- O_i il valore osservato

Noi siamo interessati all'andamento di E_{RMS} in funzione del grado m del polinomio quindi, se il polinomio completo ha grado M, calcoleremo l'errore $M + 1 = N$ volte

```
% initializing vectors
learning_error = zeros(1,n_lrn);
y_fit = learning_error;

for j = 1:n_lrn
    m = j-1;
    V = custom_vander(x_lrn,m);
    a = V\y_lrn';
    poly = @(x,m) (x.^(0:m))*(a(1:m+1));
```

```
    y_fit = poly_predict(x_lrn,poly,m);

    % calculating learning error
    learning_error(j) = sqrt(sum((y_fit-y_lrn).^2))/n_lrn;
end
```

```
% plotting learning error
plot(0:n_lrn-1,learning_error,"-o","LineWidth",lw)
xlabel("M (grado del polinomio)")
ylabel("E_{RMS}")
legend("Training")
```



Funzioni

```
function randbet = rand_between(a,b,n)
    randbet = a + (b-a).*rand(n,1);
end

% creo funzione custom_vander
% x: vettore a partire da cui calcolare la matrice di Vandermonde arrestata
% m: grado del polinomio personalizzato
function [output_matrix] = custom_vander(x,m)
    output_matrix = zeros(1, m+1);
```

```
    for i=1:length(x)
        output_matrix(i,:) = x(i).^(0:1:m);
    end
end

% funzione poly_predict permette di ottenere le ordinate dato un polinomio P
% x: vettore
% poly: polinomio (function handle)
% m: scalare. Grado del polinomio
function output_vector = poly_predict(x,poly,m)
    lx = length(x);
    output_vector = zeros(1,lx);
    for i=1:lx
        output_vector(i) = poly(x(i),m);
    end
end
```