Quale polinomio%3f

December 15, 2018

1 Problema

Sono date N osservazioni Gaussiane indipendenti di un polinomio. Trovare i coefficienti del polinomio assumendo di conoscere:

il grado del polinomio e la varianza dei dati

solo il grado del polinomio (trovare anche la varianza dei dati)

2 PUNTO 1

Cominciamo con l'analizzare il primo punto. Il polinomio da studiare è del tipo

$$P(x) = \beta_0 + \beta_1 x + \cdots + \beta_\ell x^\ell$$

Possiamo scrivere la relazione che lega le N misure y_i ai parametri β in forma matriciale, ossia

$$\vec{y} = W\vec{\beta} + \vec{\varepsilon}$$

che esplicitando abbiamo

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^\ell \\ 1 & x_2 & \cdots & x_2^\ell \\ \cdots & & & \\ 1 & x_N & \cdots & x_N^\ell \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \cdots \\ \beta_\ell \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \cdots \\ \varepsilon_N \end{bmatrix}$$

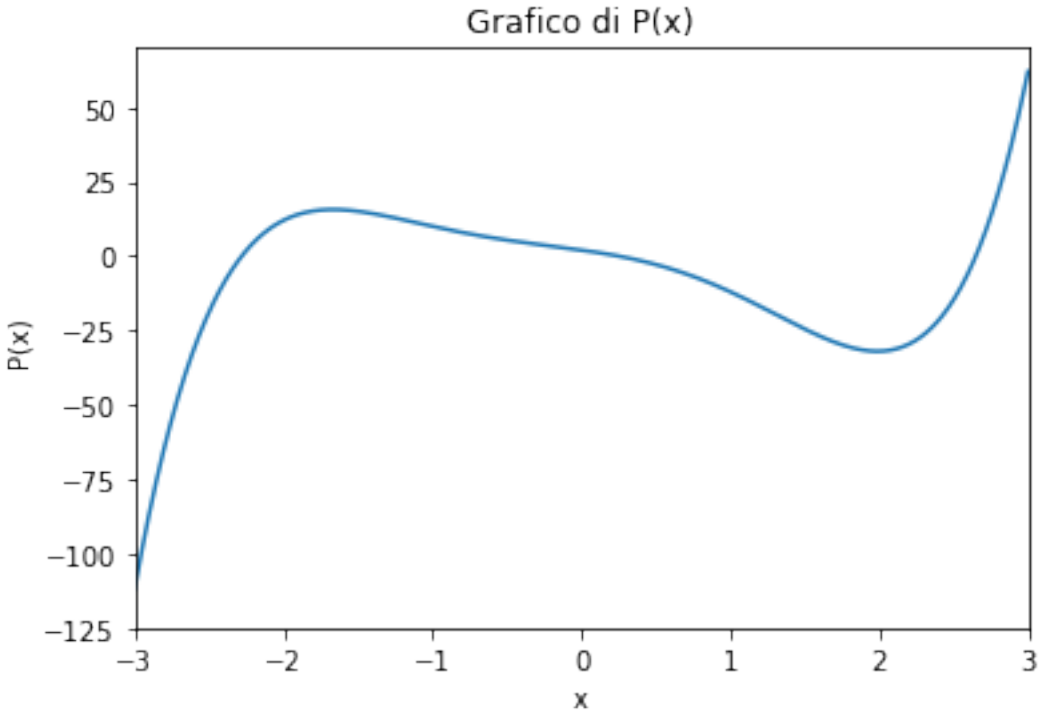
Dove le x_i sono le variabili indipedenti prive di errore, e le ε_i sono gli errori a cui è soggetta la misura. Vogliamo che la speranza matematica degli errori sia nulla, $E[\vec{\varepsilon}] = 0$, e cioè chiediamo che la speranza matematica delle misure coincida con il valore vero $E[\vec{y}] = W\vec{\beta}$. Per il nostro problema abbiamo scelto il seguente polinomio

$$P(x) = 2 - 7x - 3x^2 - 5x^3 + x^5$$

```
In [181]: import matplotlib.pyplot as plt
          import numpy as np

          x = np.linspace(-3, 3, 1000)
          y = 2-7*x-3*x**2-5*x**3+x**5

          plt.plot(x, y)
```

```
plt.xlim(-3,3)
plt.ylim(-125, 70)
plt.xlabel('x')
plt.ylabel('P(x)')
plt.title('Grafico di P(x)')
```

```
Out[181]: Text(0.5,1,'Grafico di P(x)')
```



Essendo un problema lineare, cioè le misure dipendono linearmente dai parametri, la miglior stima che possiamo avere è quella dataci dall'algoritmo di Gauss Markov. Esso afferma che la miglior stima è data da:

$$\hat{\beta} = (W^T C_y^{-1} W)^{-1} W^T C_y^{-1} \vec{y}$$

e la matrice di covarianza dei parametri è

$$C_\beta = (W^T C_y^{-1} W)^{-1}$$

Essendo, nel nostro caso, le misure indipendenti ed equivariate ($C_y = \sigma_y I$), il teorema di G-M coincide con la stima dei minimi quadrati

$$\hat{\beta} = (W^T W)^{-1} W^T \vec{y}$$

e la covarianza diventa

$$C_\beta = \sigma_y (W^T W)^{-1}$$

La stima di Gauss-Markov è efficiente, cioè la matrice delle covarianze raggiunge il limite inferiore fissato dalla disuguaglianza di Cramér-Rao:

$$C_\beta J > I$$

dove J è l'informazione di Fisher. Tuttavia, C_β dipende dalla matrice W e quindi dalla distribuzione delle variabili certe x_i nell'intervallo scelto.

Allora ci chiediamo:

Come distribuire le misure x nell'intervallo dato in modo da minimizzare la matrice di covarianza dei parametri?

Notiamo che un polinomio di grado ℓ è univocamente determinato se è noto il suo valore in $\ell + 1$ punti. Euristicamente, possiamo concentrare le misure x in $\ell + 1$ punti. Ma come distribuiamo questi $\ell + 1$ punti?

Possiamo procedere in due modi:

ANALITICAMENTE: cercare un soluzione analatica, diagonalizzando la matrice e minimizzando i termini.

NUMERICAMENTE: utilizzando diversi algoritmi, nel nostro caso esplorazione dello spazio dei parametri e la discesa del gradiente.

SOLUZIONE ANALITICA

Per una retta, la soluzione è nota; la matrice di covarianza può essere ridotta, prendendo baricentro nullo, a:

$$(W^T W)^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & \frac{N}{\sum_i x_i^2} \end{bmatrix}$$

Da questa matrice apprendiamo che è possibile trovare una set di misure che rende scorrelati i parametri e che inoltre conviene misurare agli estremi dell'intervallo per minimizzare l'ultimo termine.

Per una parabola, invece, la situazione è più complicata. Possiamo scrivere

$$W^T W = N \begin{bmatrix} 1 & \frac{\sum_i x_i}{N} & \frac{\sum_i x_i^2}{N} \\ \frac{\sum_i x_i}{N} & \frac{\sum_i x_i^2}{N} & \frac{\sum_i x_i^3}{N} \\ \frac{\sum_i x_i^2}{N} & \frac{\sum_i x_i^3}{N} & \frac{\sum_i x_i^4}{N} \end{bmatrix}$$

Riconosciamo i momenti di ordine 1,2,3 e 4. Possiamo porre a zero i momenti dispari (baricentro e skewness), semplificando

$$W^T W = N \begin{bmatrix} 1 & 0 & \frac{\sum_i x_i^2}{N} \\ 0 & \frac{\sum_i x_i^2}{N} & 0 \\ \frac{\sum_i x_i^2}{N} & 0 & \frac{\sum_i x_i^4}{N} \end{bmatrix}$$

Alleggerendo la notazione e invertendo la matrice otteniamo

$$(W^T W)^{-1} = \begin{bmatrix} \frac{m_4}{m_4 - m_2^2} & 0 & -\frac{m_2^2}{m_4 - m_2^2 -} \\ 0 & \frac{1}{m_2} & 0 \\ -\frac{m_2^2}{m_4 - m_2^2} & 0 & \frac{m_4}{m_4 - m_2^2} \end{bmatrix}$$

Ricordando la formula della curtosi

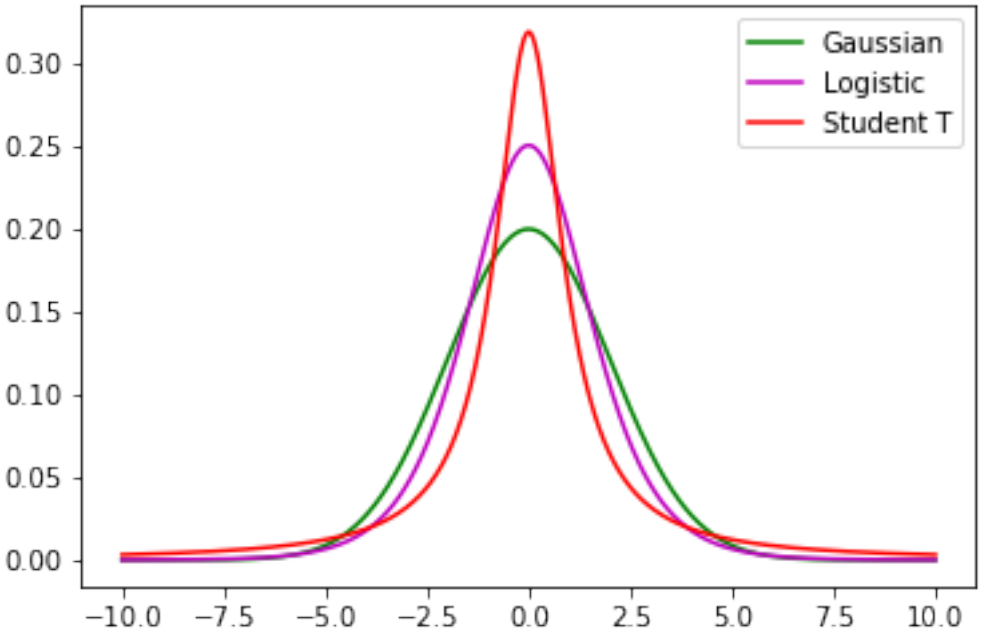
$$k = \frac{1}{N}\frac{\sum_i (x_i - \bar{x})^4}{\sigma^4} - 3 = \frac{m_4}{m_2^2} - 3$$

possiamo riscrivere la matrice

$$(W^T W)^{-1} = \begin{bmatrix} \frac{k+3}{k+2} & 0 & -\frac{1}{m_2(k+2)} \\ 0 & \frac{1}{m_2} & 0 \\ -\frac{1}{m_2(k+2)} & 0 & \frac{1}{m_2^2(k+2)} \end{bmatrix}$$

Si osserva che le varianze, eccetto la prima, dipendono inversamente dal momento di ordine 2. Inoltre tutti i termini delle varianze diminuiscono al crescere della curtosi, mentre i termini delle covarianze diventano minime sia al crescere che al decrescere della curtosi. Da qui si evince che non è possibile minimizzare tutti i termini massimizzando la varianza della distribuzione, dal momento che in generale distribuzioni con varianza grande hanno curtosi piccola.

```
In [188]: import pylab
          import numpy as np
          from scipy.stats import norm, logistic, t

          x = np.linspace(-10,10,1000)
          yg = norm.pdf(x, 0, 2)     # for example
          yl=logistic.pdf(x, 0, 1)
          yt=t.pdf(x, 1, 0, 1)
          pylab.plot(x,yg, 'g',label='Gaussian')
          pylab.plot(x,yl, 'm',label='Logistic')
          pylab.plot(x,yt, 'r',label='Student T')
          pylab.legend(loc='upper right')
          pylab.show()
```



SOLUZIONE NUMERICA Abbiamo utilizzato in prima analisi un algoritmo di esplorazione dello spazio dei parametri. Ossia, prova tutte le combinazioni possibili delle x_i (con $i = 1, \cdots N$) tali da minimizzare la matrice di covarianza.

Lo abbiamo testato per un polinomio di secondo grado con 3 misure.

```
In [183]: import time
          start=time.time()
          x0=np.arange(-3.,3.1,0.1)
          x1=np.arange(-3.,3.1,0.1)
          x2=np.arange(-3.,3.1,0.1)

          Cb_old=np.matrix([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
          for i0 in x0:
              for i1 in x1:
                  for i2 in x2:
                      W=np.matrix([[1, i0, i0**2], [1, i1, i1**2], [1, i2, i2**2]])
                  if np.linalg.det(np.matmul(W.transpose(),W))==0:
                      pass
                  else:
                      Cb=np.linalg.inv(np.matmul(W.transpose(),W))
                      if (np.absolute(Cb)<=np.absolute(Cb_old)).all():
                          Cb_old=Cb
                          #print Cb_old
                          X=np.array([i0,i1,i2])
          end=time.time()
          print 'il tempo necessario è ',end-start
          print 'la matrice di covarianza minima trovata è'
          print Cb_old
          print 'le x a cui corrisponde la Cb minima è'
          print X
```

```
il tempo necessario è  14.1932029724
la matrice di covarianza minima trovata è
[[ 1.          0.01149425 -0.11494253]
 [ 0.01149425  0.05771899 -0.00231216]
 [-0.11494253 -0.00231216  0.01981958]]
le x a cui corrisponde la Cb minima è
[ 2.66453526e-15 -2.90000000e+00  3.00000000e+00]
```

Il risultato è quello che ci aspettavamo, ossia se concentriamo le misure in tre punti, ossia agli estremi e nel vertice, la matrice di covarianza è minima. Questo algoritmo è molto efficace, in quanto ci restituisce la matrice di covarianza minima nell'intervallo desiderato, ma richiede una quantità di risorse che cresce esponenzialmente col numero di misure. Possiamo vedere che per sole 3 misure per un polinomio di secondo grado impiega ben 13 secondi, mentre per solo una misura in più il tempo è di oltre 4000s.

Possiamo "ottimizzare" l'algoritmo di esplorazione dello spazio delle fasi. Utilizziamo la classe degli algoritmi di ottimizzazione TNC (Truncated Newton Constrained). Cerchiamo le misure che rendano la norma della matrice di covarianza minima. Studiamo le distribuzione delle misure per una parabola.

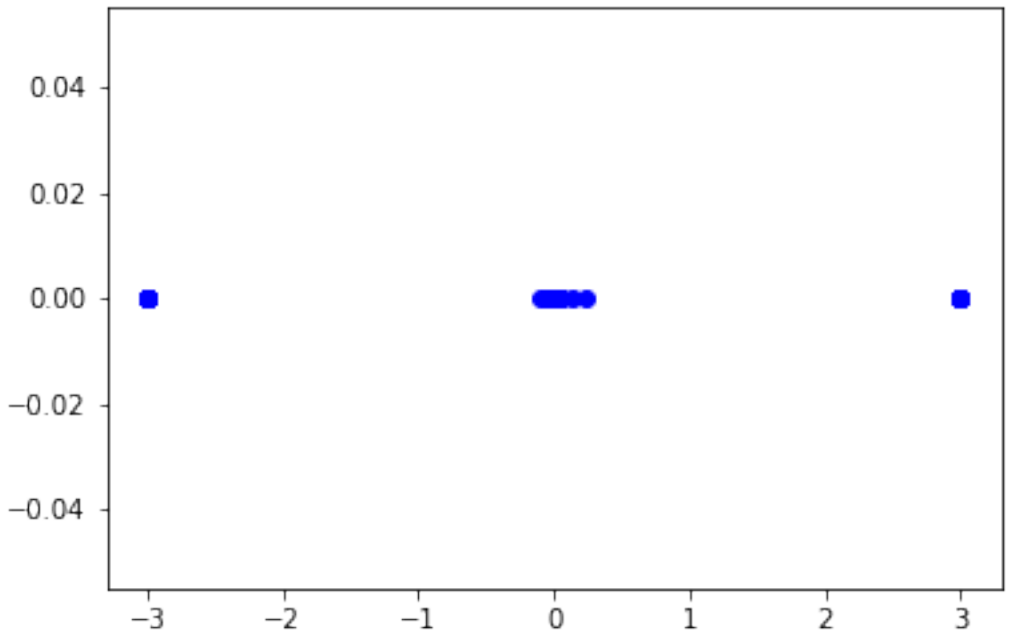
In [201]:
```
import scipy
from scipy import optimize, stats
import numpy as np

def cov_norm(X, degree=2):

    W = np.matrix([ [x**j for j in xrange(degree+1)] for x in X])
    Cb = np.linalg.inv(np.matmul(W.transpose(),W))
    norm = np.linalg.norm(Cb)

    return norm

def cov_tr(X, degree=2):

    W = np.matrix([ [x**j for j in xrange(degree+1)] for x in X])
    Cb = np.linalg.inv(np.matmul(W.transpose(),W))
    trace = np.trace(Cb)
    return trace

def optimize(measures_number,extreme1, interval_length, fun):

    limits = [(extreme1,extreme1+ interval_length) for i in xrange(measures_number)]
    Xin = np.random.uniform(extreme1,extreme1 + interval_length, measures_number)

    return scipy.optimize.minimize(fun, Xin, jac=None, hess=None, hessp=None, bounds
    tol=None, callback=None, options=None)
```

In [232]:
```
N = 100
x1 = -3
L = 6
fun = cov_norm

Res = optimize(N,x1,L,fun)

X1= Res['x']
xg = np.sum(X)/len(X)
sk = np.sum(X**3)/len(X)
five = np.sum
kur = stats.kurtosis(X,fisher = True )
W = np.matrix([ [x**j for j in xrange(2+1)] for x in X])
Cb = np.linalg.inv(np.matmul(W.transpose(),W))
```

```
        print 'il baricentro delle misure è {}'.format(xg)
        print 'la skewness delle misure è {}'.format(sk)
        print 'la kurtosis delle misure è {}'.format(kur)
        print 'la matrice di covarianza delle misure è \n{}'.format(Cb)
        cost=np.zeros(N)
        plt.plot(X1,cost,'bo')
```

```
il baricentro delle misure è 0.0333333333333
la skewness delle misure è 0.870333333333
la kurtosis delle misure è -1.5
la matrice di covarianza delle misure è
[[ 1.          0.01149425 -0.11494253]
 [ 0.01149425  0.05771899 -0.00231216]
 [-0.11494253 -0.00231216  0.01981958]]
```

```
Out[232]: [<matplotlib.lines.Line2D at 0x7fa6e00998d0>]
```



Per utilizzare l'algoritmo di Gauss-Markov per la stima di $\vec{\beta}$, il numero N delle misure deve essere almeno pari al numero dei parametri incogniti. Abbiamo utilizzato l'algoritmo precedente per vedere come posizione al meglio i seguenti numeri di misure:

$N = 6$ pari al numero di parametri incogniti

$N = 10$

$N = 20$

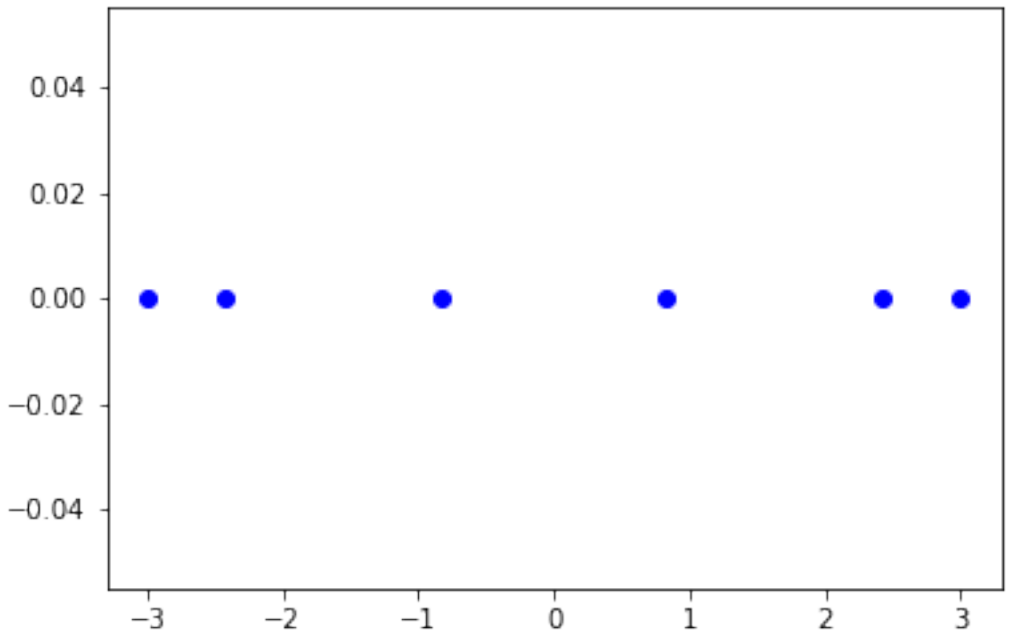
```
In [ ]: import scipy
        from scipy import optimize, stats
```

```
        import numpy as np

        def cov_norm(X, degree=5):

            W = np.matrix([ [x**j for j in xrange(degree+1)] for x in X])
            Cb = np.linalg.inv(np.matmul(W.transpose(),W))
            norm = np.linalg.norm(Cb)

            return norm

        def cov_tr(X, degree=5):

            W = np.matrix([ [x**j for j in xrange(degree+1)] for x in X])
            Cb = np.linalg.inv(np.matmul(W.transpose(),W))
            trace = np.trace(Cb)
            return trace

        def optimize(measures_number,extreme1, interval_length, fun):

            limits = [(extreme1,extreme1+ interval_length) for i in xrange(measures_number)]
            Xin = np.random.uniform(extreme1,extreme1 + interval_length, measures_number)

            return scipy.optimize.minimize(fun, Xin, jac=None, hess=None, hessp=None, bounds =
            tol=None, callback=None, options=None)
```

In [145]:
```
N = 6
x1 = -3
L = 6
deg=5
fun = cov_norm

Res = optimize(N,x1,L,fun)

X1= Res['x']
xg = np.sum(X)/len(X)
sk = np.sum(X**3)/len(X)
five = np.sum
kur = stats.kurtosis(X,fisher = True )
W = np.matrix([ [x**j for j in xrange(deg+1)] for x in X])
Cb = np.linalg.inv(np.matmul(W.transpose(),W))
print 'il baricentro delle misure è {}'.format(xg)
print 'la skewness delle misure è {}'.format(sk)
print 'la kurtosis delle misure è {}'.format(kur)
print 'la matrice di covarianza delle misure è \n{}'.format(Cb)
cost=np.zeros(N)
plt.plot(X1,cost,'bo')
print X1
```

```
il baricentro delle misure è 0.136636598114
```

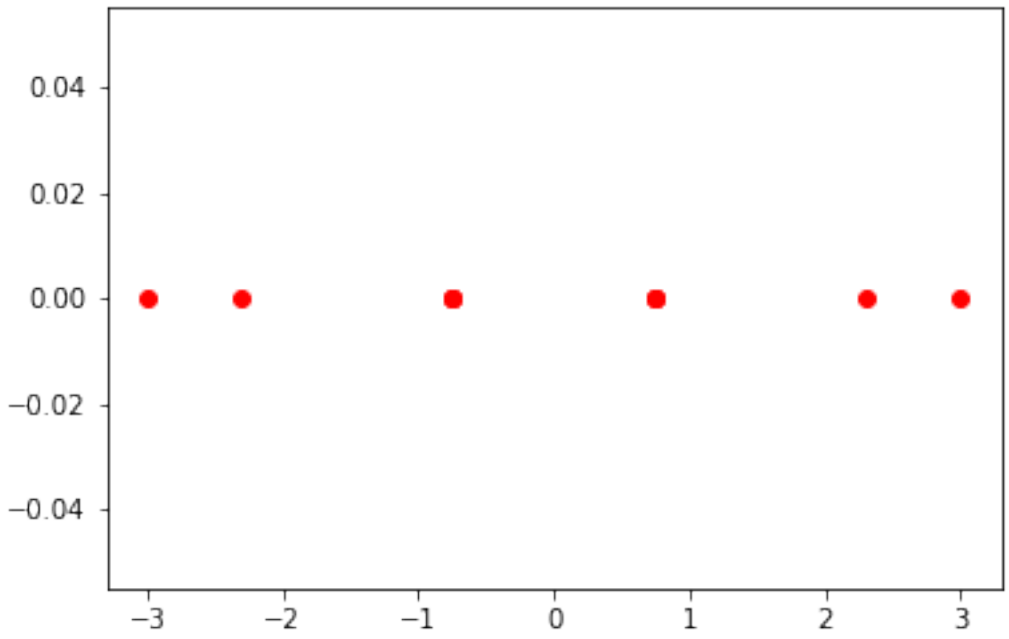
```
la skewness delle misure è 1.57595904977
la kurtosis delle misure è -1.08727621026
la matrice di covarianza delle misure è
[[ 0.10422894  0.00492412 -0.01659175 -0.00020478]
 [ 0.00492412  0.10838618  0.00106224 -0.0147389 ]
 [-0.01659175  0.00106224  0.00521564 -0.00045077]
 [-0.00020478 -0.0147389  -0.00045077  0.00234871]]
[ 2.4286894  -3.          0.82944469 -0.82944691  3.         -2.42869073]
```




```
In [146]: N = 10
          x1 = -3
          deg=5
          L = 6
          fun = cov_tr
          Res = optimize(N,x1,L,fun)
          X2 = Res['x']
          xg = np.sum(X)/len(X)
          sk = np.sum(X**3)/len(X)
          five = np.sum
          kur = stats.kurtosis(X,fisher = True )
          W = np.matrix([ [x**j for j in xrange(deg+1)] for x in X])
          Cb = np.linalg.inv(np.matmul(W.transpose(),W))
          print 'il baricentro delle misure è {}'.format(xg)
          print 'la skewness delle misure è {}'.format(sk)
          print 'la kurtosis delle misure è {}'.format(kur)
```

```
            print 'la matrice di covarianza delle misure è \n{}'.format(Cb)
            cost=np.zeros(N)
            print cost
            plt.plot(X2,cost,'ro')
            print X2
```

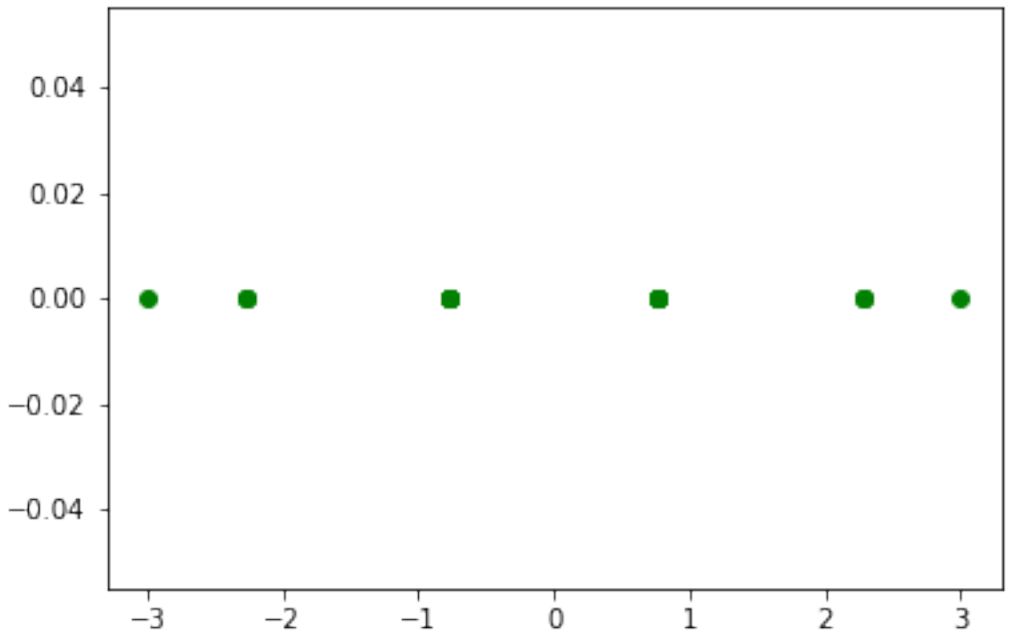
```
il baricentro delle misure è 0.136636598114
la skewness delle misure è 1.57595904977
la kurtosis delle misure è -1.08727621026
la matrice di covarianza delle misure è
[[ 0.10422894  0.00492412 -0.01659175 -0.00020478]
 [ 0.00492412  0.10838618  0.00106224 -0.0147389 ]
 [-0.01659175  0.00106224  0.00521564 -0.00045077]
 [-0.00020478 -0.0147389  -0.00045077  0.00234871]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[-2.31052343  0.75086556 -0.75095185 -0.75094378  2.31051242  3.
  0.75097548 -0.75085162  0.75090238 -3.        ]
```



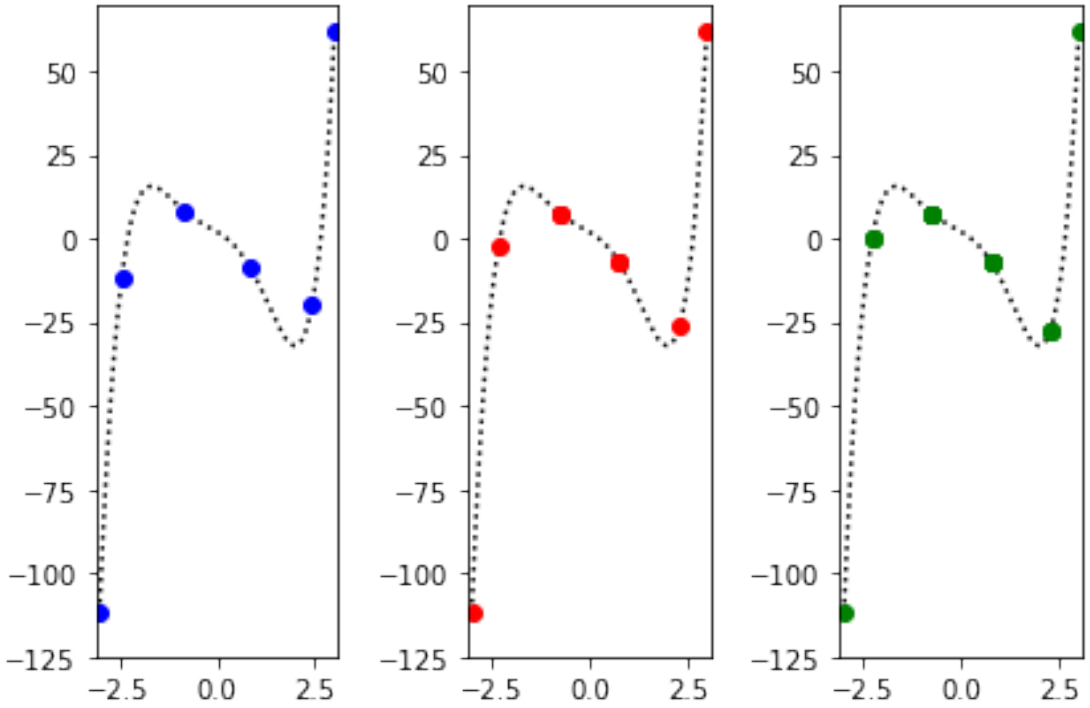
In [164]:
```
N = 20
x1 = -3
L = 6
deg=5
fun = cov_tr
Res = optimize(N,x1,L,fun)
X3 = Res['x']
```

```
            xg = np.sum(X)/len(X)
            sk = np.sum(X**3)/len(X)
            five = np.sum
            kur = stats.kurtosis(X,fisher = True )
            W = np.matrix([ [x**j for j in xrange(deg+1)] for x in X])
            Cb = np.linalg.inv(np.matmul(W.transpose(),W))
            print 'il baricentro delle misure è {}'.format(xg)
            print 'la skewness delle misure è {}'.format(sk)
            print 'la kurtosis delle misure è {}'.format(kur)
            print 'la matrice di covarianza delle misure è \n{}'.format(Cb)
            cost=np.zeros(N)
            print cost
            plt.plot(X3,cost,'go')
            print X3
```

```
il baricentro delle misure è 0.136636598114
la skewness delle misure è 1.57595904977
la kurtosis delle misure è -1.08727621026
la matrice di covarianza delle misure è
[[ 0.10422894  0.00492412 -0.01659175 -0.00020478]
 [ 0.00492412  0.10838618  0.00106224 -0.0147389 ]
 [-0.01659175  0.00106224  0.00521564 -0.00045077]
 [-0.00020478 -0.0147389  -0.00045077  0.00234871]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 2.27751371 -0.76997461  0.76988255 -3.          0.76980816  0.76992949
  2.27744434  3.         -0.76980413 -0.76988566 -2.27751974 -0.76994793
 -0.76984444 -2.27756261  2.277594    0.76981378 -2.27760017  0.76974424
 -0.76988374  0.76991293]
```



```
In [168]: x = np.linspace(-3, 3, 1000)
          y = 2-7*x-3*x**2-5*x**3+x**5

          plt.subplot(1,3, 1)
          plt.plot(x, y, ':k')

          plt.xlim(-3.1,3.1)
          plt.ylim(-125, 70)
          y_m1 = 2-7*X1-3*X1**2-5*X1**3+X1**5

          plt.plot(X1, y_m1, 'bo', label='N=6')


          plt.subplot(1, 3, 2)
          plt.plot(x, y, ':k')
          plt.xlim(-3.1,3.1)
          plt.ylim(-125, 70)

          y_m2 = 2-7*X2-3*X2**2-5*X2**3+X2**5
          plt.plot(X2, y_m2, 'ro', label='N=10')

          plt.subplot(1, 3, 3)
          plt.plot(x, y, ':k')
          plt.xlim(-3.1,3.1)
          plt.ylim(-125, 70)
          y_m3 = 2-7*X3-3*X3**2-5*X3**3+X3**5
          plt.plot(X3, y_m3, 'go', label='N=20')
          plt.tight_layout()
```



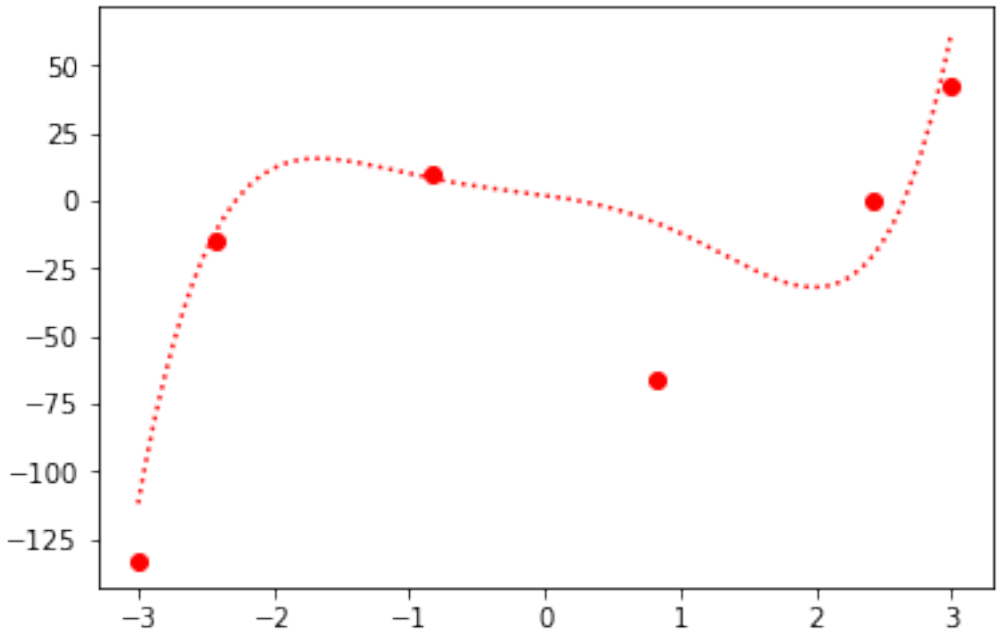
Notiamo da subito che il nostro criterio euristico di concentrare le misure in $\ell + 1$ punti per un polinomio di grado ℓ è confermato anche dall'algoritmo di ottimizzazione utilizzato, senza possedere alcuna informazione a riguardo.

Per quanto riguarda la varianza, invece, non possiamo fare altro che determinarla empiricamente, o meglio determinare empiricamente uno o più valori che sia ragionevole. Le misure sono equivariate e indipendenti per ipotesi, così da poter scrivere $C_y = \sigma_y I$. Come test di controllo usiamo una $\sigma_y = 0$. Allora possiamo generare con un generatore di numeri casuali con distribuzione gaussiana.

```
In [169]: beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          N=6 #numero di misure
          sigma=20
          Cy=sigma*np.identity(N)
          x_m1=np.linspace(-3, 3, N)
          ym=np.empty(N)
          W=np.empty((N,g+1))                    #generiamo la matrice W
          for j in range(0,g+1,1) :
              for i in range(0,N,1):
                  if j==0:
                      W[i][j]=1
                  else:
                      W[i][j]=X1[i]**j
          #print W
```

```
p=np.matmul(W,beta_real)
#print p

ym=np.random.normal(p,sigma)
#print ym

plt.plot(x, y, ':r')
plt.plot(X1, ym, 'ro')
```

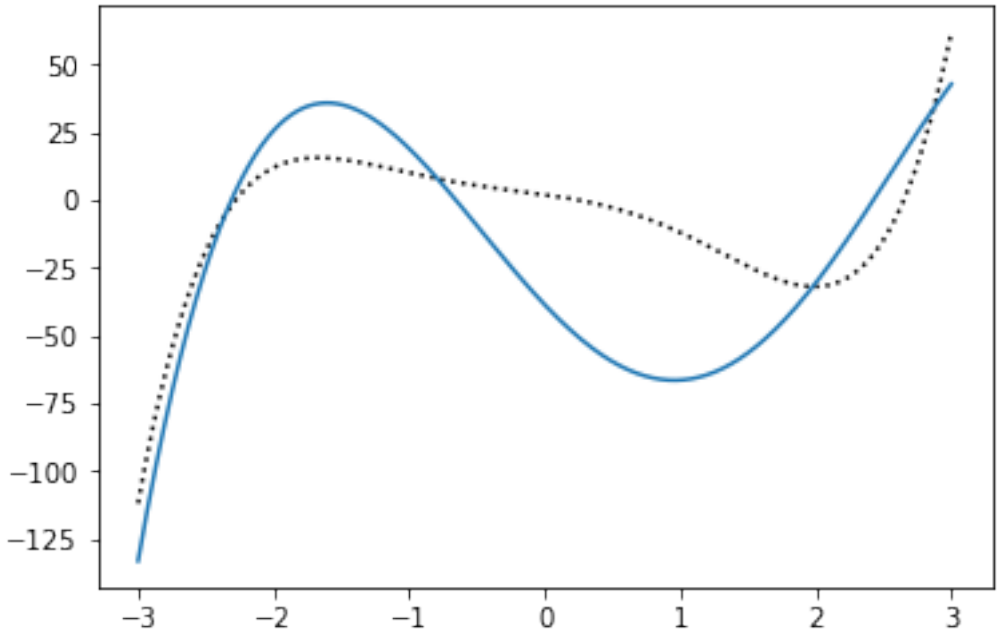
Out[169]: [<matplotlib.lines.Line2D at 0x7fa6e2166590>]



In [170]:
```
#beta_1=np.linalg.inv(np.matmul(np.matmul(W.transpose(),np.linalg.inv(Cy)),W))
#beta_2=np.matmul(np.matmul(W.transpose(),np.linalg.inv(Cy)),ym)
#beta_stima=np.matmul(beta_1,beta_2)
#print beta_stima

beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
beta_2=np.matmul(W.transpose(),ym)
beta_stima=np.matmul(beta_1,beta_2)
ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]
plt.plot(x, y,':k')
plt.plot(x, ys)

print 'I coefficienti sono:' ,
print('   '.join([str(round(elem,3)) for elem in beta_stima]))
```

```
        print 'La matrice di covarianza dei parametri stimati è'
        Cb=np.linalg.inv(np.matmul(np.matmul(W.transpose(),np.linalg.inv(Cy)),W))
        for row in Cb:
            print('    '.join([str(round(elem,3)) for elem in row if elem>0])),
            print('  '.join([str(round(elem,3)) for elem in row if elem<=0]))
```

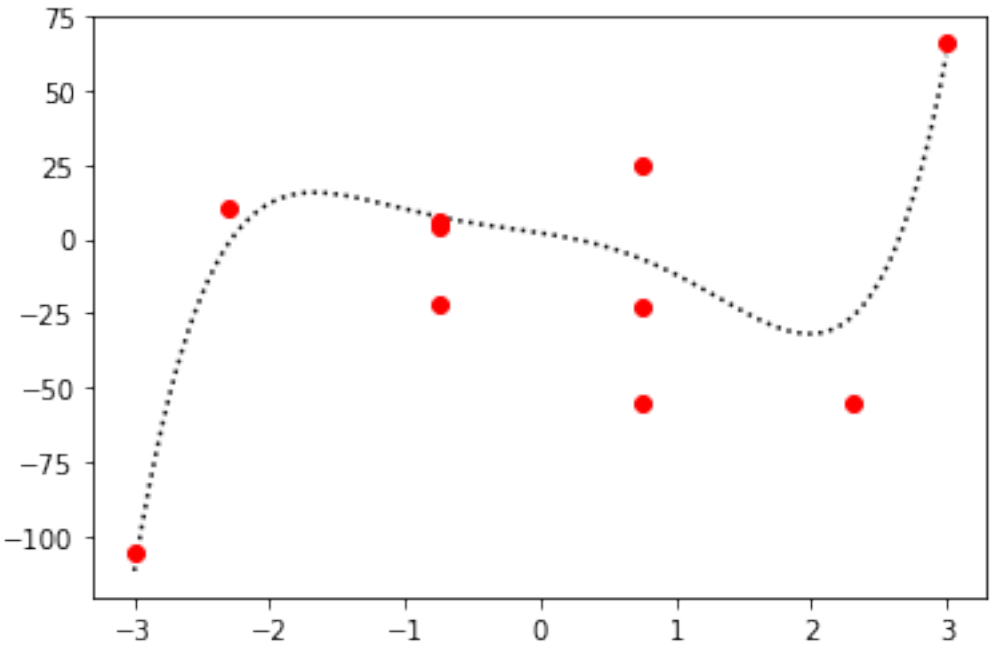
```
I coefficienti sono: -38.659   -52.775   16.87   10.117   -1.956   -0.11
La matrice di covarianza dei parametri stimati è
16.74    0.0    0.0    0.581 -6.916  -0.0
0.0    22.115    0.0    0.458 -6.563  -0.0
0.0    5.43    0.0 -6.916  -0.0  -0.549
0.0    2.402    0.0 -6.563  -0.0  -0.189
0.581    0.0    0.059 -0.0  -0.549  -0.0
0.458    0.0    0.016 -0.0  -0.189  -0.0
```




```
In [171]: beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          N=10 #numero di misure
          sigma=20
          Cy=sigma*np.identity(N)
          x_m2=np.linspace(-3, 3, N)
          ym=np.empty(N)
          W=np.empty((N,g+1))                    #generiamo la matrice W
          for j in range(0,g+1,1) :
              for i in range(0,N,1):
```

```
            if j==0:
                W[i][j]=1
            else:
                W[i][j]=X2[i]**j
        #print W

        p=np.matmul(W,beta_real)
        #print p

        ym=np.random.normal(p,sigma)
        #print ym

        plt.plot(x, y, ':k')
        plt.plot(X2, ym, 'ro')
```

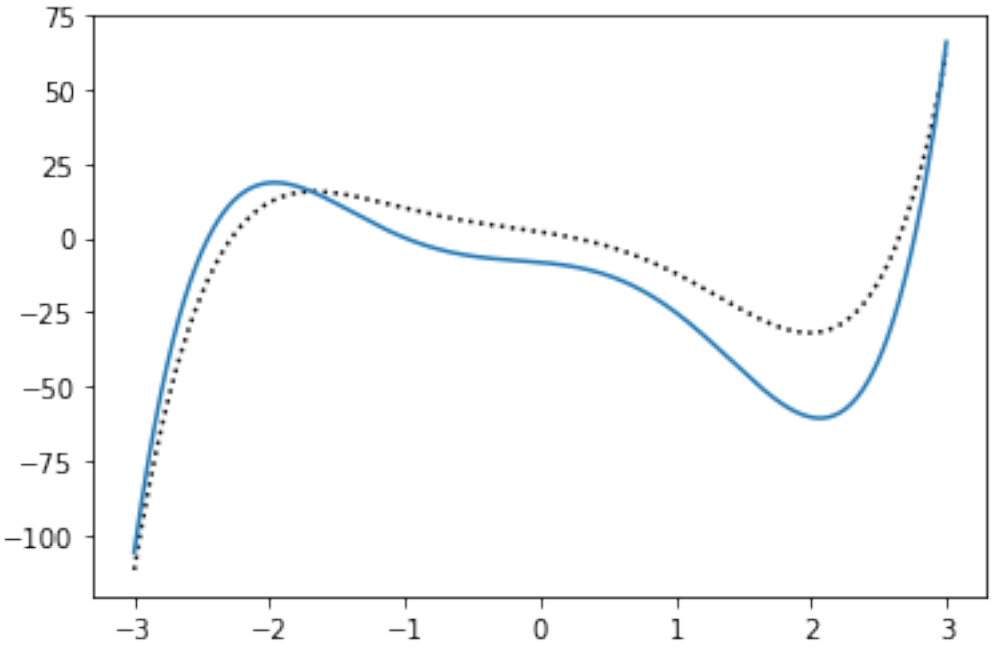
Out[171]: [<matplotlib.lines.Line2D at 0x7fa6e202ee90>]



In [172]:
```
beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
beta_2=np.matmul(W.transpose(),ym)
beta_stima=np.matmul(beta_1,beta_2)
ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]
plt.plot(x, y,':k')
plt.plot(x, ys)

print 'I coefficienti sono:' ,
print('   '.join([str(round(elem,3)) for elem in beta_stima]))
```

```
        print 'La matrice di covarianza dei parametri stimati è'
        Cb=np.linalg.inv(np.matmul(np.matmul(W.transpose(),np.linalg.inv(Cy)),W))
        for row in Cb:
            print('    '.join([str(round(elem,3)) for elem in row if elem>0])),
            print('  '.join([str(round(elem,3)) for elem in row if elem<=0]))
```

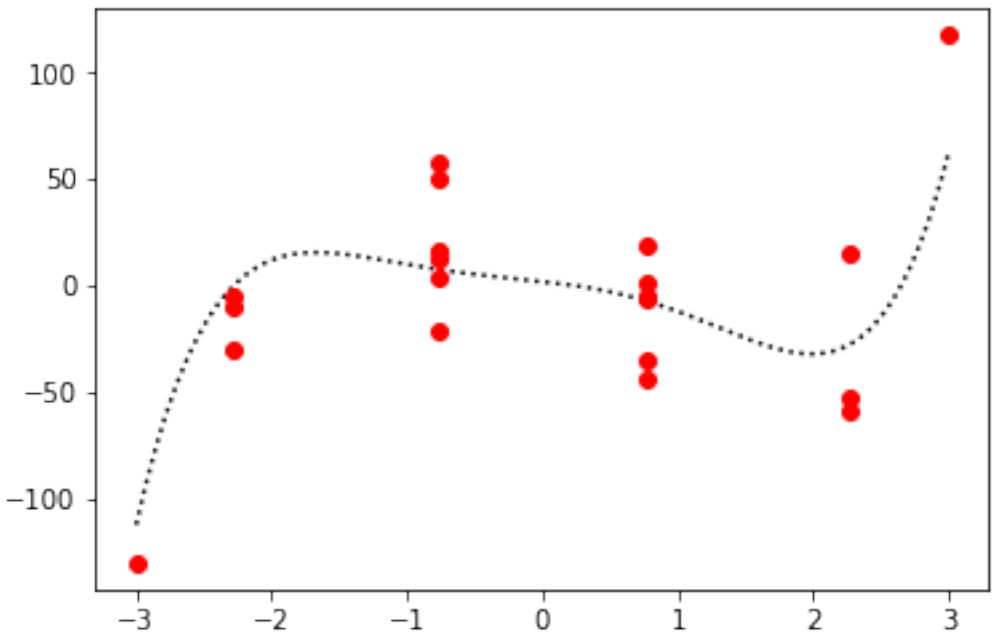
```
I coefficienti sono: -8.281   -4.101   -4.659   -9.962   0.372   1.512
La matrice di covarianza dei parametri stimati è
5.68    0.0    0.296    0.0 -3.19  -0.0
0.0    8.58    0.0    0.21 -0.0  -2.828
3.78    0.0    0.0 -3.19  -0.0  -0.404
0.0    1.35 -0.0  -2.828  -0.0  -0.118
0.296    0.0    0.045 -0.404  -0.0  -0.0
0.0    0.21    0.0    0.011 -0.118  -0.0
```




```
In [174]: beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          N=20 #numero di misure
          sigma=20
          Cy=sigma*np.identity(N)
          x_m3=np.linspace(-3, 3, N)
          ym=np.empty(N)
          W=np.empty((N,g+1))                    #generiamo la matrice W
          for j in range(0,g+1,1) :
```

```
        for i in range(0,N,1):
            if j==0:
                W[i][j]=1
            else:
                W[i][j]=X3[i]**j
    #print W

    p=np.matmul(W,beta_real)
    #print p

    ym=np.random.normal(p,sigma)
    #print ym
    x = np.linspace(-3, 3, 1000)
    y = 2-7*x-3*x**2-5*x**3+x**5

    plt.plot(x, y, ':k')
    plt.plot(X3, ym, 'ro')
```

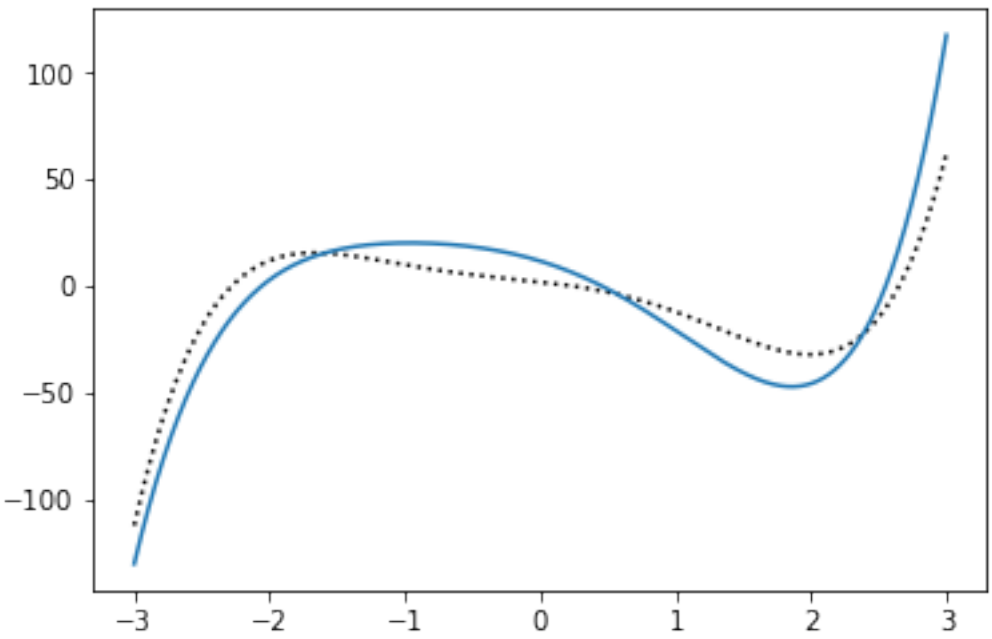
Out[174]: [<matplotlib.lines.Line2D at 0x7fa6e20dd750>]



In [175]:
```
beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
beta_2=np.matmul(W.transpose(),ym)
beta_stima=np.matmul(beta_1,beta_2)
ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]
plt.plot(x, y,':k')
plt.plot(x, ys)
```

```
        print 'I coefficienti sono:' ,
        print('   '.join([str(round(elem,3)) for elem in beta_stima]))

        print 'La matrice di covarianza dei parametri stimati è'
        Cb=np.linalg.inv(np.matmul(np.matmul(W.transpose(),np.linalg.inv(Cy)),W))
        for row in Cb:
            print('    '.join([str(round(elem,3)) for elem in row if elem>0])),
            print('  '.join([str(round(elem,3)) for elem in row if elem<=0]))
```

```
I coefficienti sono: 11.739   -19.37   -13.277   -2.138   1.253   0.986
La matrice di covarianza dei parametri stimati è
2.836    0.0    0.0    0.14 -1.469  -0.0
0.0    4.177    0.0    0.102 -1.375  -0.0
0.0    1.549    0.0 -1.469  -0.0  -0.176
0.0    0.608    0.0 -1.375  -0.0  -0.053
0.14    0.0    0.022 -0.0  -0.176  -0.0
0.102    0.0    0.005 -0.0  -0.053  -0.0
```



3 PUNTO 2

Possiamo ora analizzare il punto 2 del nostro problema. In questo caso ci è noto solamente il grado del polinomio e non conosciamo, come nel caso precedente, la varianza. Per questo, abbiamo usato il metodo della massima verosimiglianza, che ci permette di stimare sia i coefficienti sia

la varianza. Questo metodo "riversa" la nostra ignoranza nella distribuzione di probabilità delle misura, ossia per calcolare la verosimiglianza dobbiamo ipotizzare una pdf: noi abbiamo scelto una distribuzione gaussiana. Calcoliamo allora la verosimiglianza

$$L\left(\vec{y}\,|\,\vec{\beta},\sigma^2\right) = \frac{1}{\left(2\pi\sigma^2\right)^{\frac{N}{2}}} e^{-\sum_{i=1}^{N}\frac{\left(y_i - W\vec{\beta}\right)^2}{2\sigma^2}}$$

dove N è il numero di misure. Allora passiamo al logaritmo

$$\log L\left(\vec{y}|\vec{\beta},\sigma^2\right) = \log L\left(\vec{\beta},\sigma^2|\vec{y}\right) = -\log\left(2\pi\sigma^2\right)^{\frac{N}{2}} - \sum_{i=1}^{N}\frac{y_i - W\vec{\beta}}{2\sigma^2}$$

$$= -\frac{N}{2}\log\left(2\pi\sigma^2\right) - \sum_{i=1}^{N}\frac{y_i - \sum_{j=0}^{k} W_{ij}\beta_j}{2\sigma^2}$$

dove k è il grado del polinomio. A questo punto possiamo derivare e cercare i punti stazionari della verosimiglianza al variare dei parametri.

$$\frac{\partial}{\partial\beta_\ell}\log L\left(\vec{\beta},\sigma^2|\vec{y}\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N} 2\left(y_i - \sum_{j=0}^{k} W_{ij}\beta_j\right)\frac{\partial}{\partial\beta_\ell}\left(y_i - \sum_{j=0}^{k} W_{ij}\beta_j\right)$$

$$= -\frac{1}{\sigma^2}\sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{k} W_{ij}\beta_j\right) W_{i\ell} = -\frac{1}{\sigma^2}\sum_{i=1}^{N}\left(W_{i\ell}\,y_i - \sum_{j=0}^{k} W_{ij}\,W_{i\ell}\,\beta_j\right)$$

Così eguagliando a zero possiamo trovare i punti stazionari della funzione di verosimiglianza

$$\sum_{i=1}^{N}\left(-W_{i\ell}\,y_i + \sum_{j=0}^{k} W_{ij}\,W_{i\ell}\,\beta_j\right) = 0$$

da cui

$$\sum_{j=0}^{k}\sum_{i=1}^{N} W_{ij}\,W_{i\ell}\,\beta_j - \sum_{i=1}^{N} W_{i\ell}\,y_i = 0$$

$$\sum_{j=0}^{k}\sum_{i=1}^{N} W^T_{ji}\,W_{i\ell}\,\beta_j - \sum_{i=1}^{N} W^T_{\ell i}\,y_i = 0$$

$$\sum_{j=0}^{k}\sum_{i=1}^{N}\left(W^T\,W\right)_{j\ell}\,\beta_j = \sum_{i=1}^{N} W^T_{\ell i}\,y_i$$

$$\sum_{j=0}^{k}\sum_{i=1}^{N}\left(W^T\,W\right)^T_{\ell j}\,\beta_j = \sum_{i=1}^{N} W^T_{\ell i}\,y_i$$

in maniera più compatta si può scrivere

$$\left(W^T\,W\right)^T\beta = W^T y$$

$$\left(W^T\,W\right)\beta = W^T y$$

$$\beta = \left(W^T W\right)^{-1} W^T y$$

Coincide con la stima dei minimi quadrati. Per quanto riguarda la stima della varianza

$$\frac{\partial}{\partial \sigma^2} \log L\left(\vec{\beta}, \sigma^2 | \vec{y}\right) = -\frac{N}{2} \frac{\partial}{\partial \sigma^2} \log\left(2\pi\sigma^2\right) - \frac{\partial}{\partial \sigma^2} \sum_{i=1}^{N} \frac{\left(y_i - W\vec{\beta}\right)^2}{2\sigma^2}$$

$$-\frac{N}{2}\frac{1}{\sigma^2} - \sum_{i=1}^{N} \frac{\left(y_i - W\vec{\beta}\right)^2}{2} \frac{\partial \frac{1}{\sigma^2}}{\partial \sigma^2} = -\frac{N}{2\sigma^2} + \frac{1}{\sigma^4} \sum_{i=1}^{N} \frac{\left(y_i - W\vec{\beta}\right)^2}{2} = 0$$

E allora

$$-N\sigma^2 + \sum_{i=1}^{N} \left(y_i - W\vec{\beta}\right)^2 = 0$$

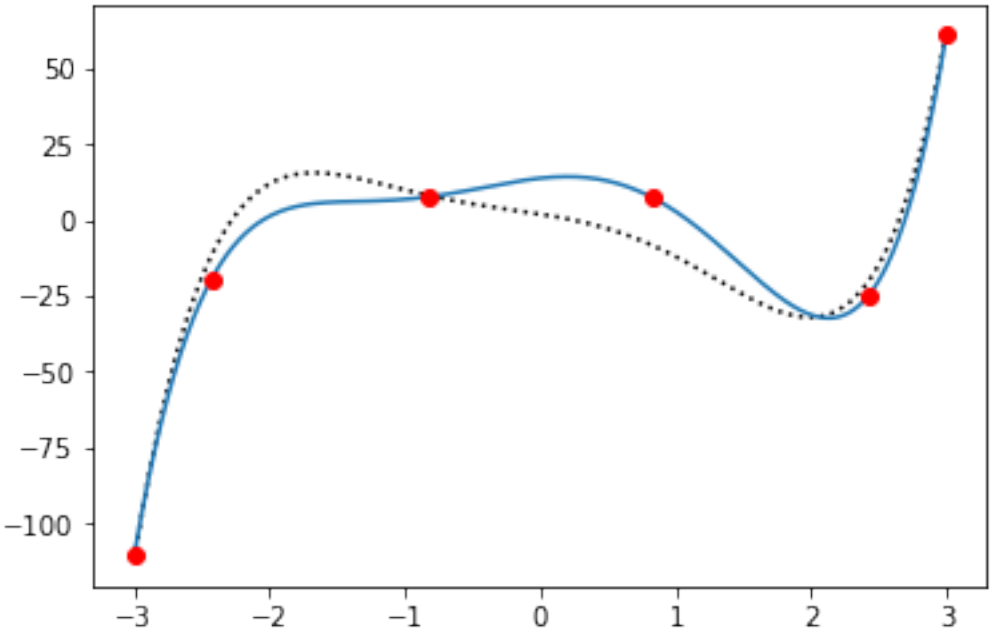
e così finalmente

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - W\vec{\beta}\right)^2$$

che è proprio lo scarto quadratico medio.

```
In [177]: N=6 #numero di misure
          beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          sigma=10
          W=np.empty((N,g+1))                  #generiamo la matrice W
          for j in range(0,g+1,1) :
              for i in range(0,N,1):
                  if j==0:
                      W[i][j]=1
                  else:
                      W[i][j]=X1[i]**j


          p=np.matmul(W,beta_real)

          ym=np.random.normal(p,sigma)

          beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
          beta_2=np.matmul(W.transpose(),ym)
          beta_stima=np.matmul(beta_1,beta_2)
          ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]

          plt.plot(x, y,':k')
          plt.plot(x, ys)
          plt.plot(X1,ym, 'ro')

          scarto=ym-np.matmul(W,beta_stima)
          sigma=np.sqrt(np.dot(scarto,scarto)/N)
          print 'La varianza stimata è', sigma**2
```

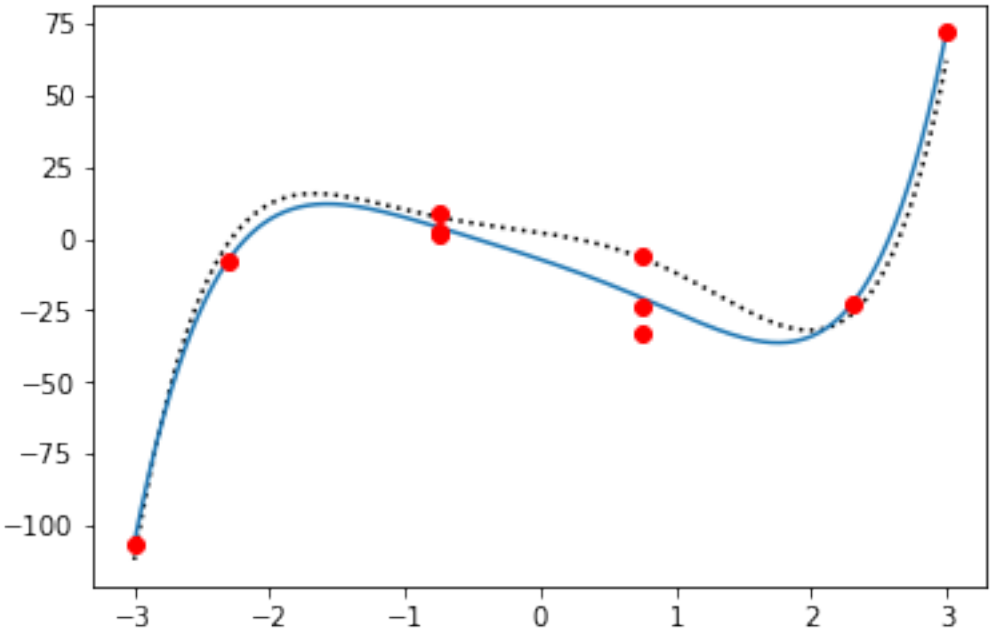
```
La varianza stimata è 7.803224062597172e-25
```




```
In [178]: N=10 #numero di misure
          beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          sigma=10
          W=np.empty((N,g+1))                    #generiamo la matrice W
          for j in range(0,g+1,1) :
              for i in range(0,N,1):
                  if j==0:
                      W[i][j]=1
                  else:
                      W[i][j]=X2[i]**j


          p=np.matmul(W,beta_real)

          ym=np.random.normal(p,sigma)

          beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
          beta_2=np.matmul(W.transpose(),ym)
          beta_stima=np.matmul(beta_1,beta_2)
          ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]

          plt.plot(x, y,':k')
```

```
plt.plot(x, ys)
plt.plot(X2,ym, 'ro')

scarto=ym-np.matmul(W,beta_stima)
sigma=np.sqrt(np.dot(scarto,scarto)/N)
print 'La varianza stimata è', sigma**2
```

```
La varianza stimata è 40.532170554123205
```




```
In [180]: N=20 #numero di misure
          x_m=np.linspace(-3, 3, N)#randomizzo l'intervallo
          beta_real=np.array([2,-7,-3,-5,0,1]) #questi sono i coefficienti del polinomio
          g=5 #grado del polinomio
          sigma=10
          W=np.empty((N,g+1))                    #generiamo la matrice W
          for j in range(0,g+1,1) :
              for i in range(0,N,1):
                  if j==0:
                      W[i][j]=1
                  else:
                      W[i][j]=X3[i]**j


          p=np.matmul(W,beta_real)
```

```
ym=np.random.normal(p,sigma)

beta_1=np.linalg.inv(np.matmul(W.transpose(),W))
beta_2=np.matmul(W.transpose(),ym)
beta_stima=np.matmul(beta_1,beta_2)
ys= beta_stima[0]+beta_stima[1]*x+beta_stima[2]*x**2+beta_stima[3]*x**3+beta_stima[4]

plt.plot(x, y,':k')
plt.plot(x, ys)
plt.plot(X3,ym, 'ro')

scarto=ym-np.matmul(W,beta_stima)
sigma=np.sqrt(np.dot(scarto,scarto)/N)
print 'La varianza stimata è', sigma**2
```

```
La varianza stimata è 66.51685308599397
```