INDICE

In [1]:
```
import numpy as np
```

Prodotto tra vettori/matrici

considero due vettori

In [2]:
```
v1=np.array([1,2,3])
v2=np.array([10,20,30])
```

elemento per elemento, tra array monodimensionali

Abbiamo visto che il prodotto tra vettori con l'operatore * consiste nel prodotto elemento per elemento.

In [3]:
```
v1*v2
```

Out[3]:
```
array([10, 40, 90], dtype=int32)
```

scalare, tra array monodimensionali

per fare il prodotto scalare tra `array` devo usare la funzione np.dot:

tra array monodimensionali

In [4]:
```
np.dot(v1,v2)
```

Out[4]:
```
140
```

tra un array riga ed uno colonna (vale il broadcasting sul primo vettore)

In [5]:
```
np.dot(v1,v2[:,None]) # equivalente a np.dot(v1[None,:],v2[:,None])
```

Out[5]:
```
array([140])
```

riga per colonna, tra matrici

le Numpy mettono anche a disposizione il tipo matrix

In [6]:
```
m1=np.matrix(v1)
m2=np.matrix(v2)
```

si tratta di array BIDIMENSIONALI:

In [7]:
```
m1.shape,m2.shape
```

Out[7]:
```
((1, 3), (1, 3))
```

per i quali sono ridefiniti gli operatori standard in modo da operare secondo le regole dell'algebra lineare:

In [8]:
```
try:
    m1*m2 #ERRORE
except Exception as err:
    print(err)
```

```
objects are not aligned
```

ridefinisco m2 come vettore colonna:

In [9]:
```
m2=np.matrix(v2[:,np.newaxis])
```

e quindi:

In [10]:
```
m1*m2
```

Out[10]:
```
matrix([[140]])
```

che è equivalente a:

In [11]:
```
np.dot(m1,m2)
```

Out[11]:
```
matrix([[140]])
```

Altri prodotti

vedi:

http://docs.scipy.org/doc/numpy/reference/routines.linalg.html#matrix-and-vector-products

numpy.linalg

il modulo numpy.linalg mette a disposizione molti strumenti

In [12]:
```
import numpy.linalg as la
```

Inversione di una matrice specificata come array

```
```

In [13]:
```
A=np.array([[1, 3, 5],
            [2, 5, 1],
            [2, 3, 8]])
```

In [14]:
```
AI = np.linalg.inv(A)
```

In [15]:
```
print('matrice:')
print(A)
print('matrice inversa:')
print(AI)
```

```
matrice:
[[1 3 5]
 [2 5 1]
 [2 3 8]]
matrice inversa:
[[-1.48  0.36  0.88]
 [ 0.56  0.08 -0.36]
 [ 0.16 -0.12  0.04]]
```

specificata come matrix

In [16]:
```
mA=np.matrix(A)
```

In [17]:
```
np.linalg.inv(mA)
```

Out[17]:
```
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
```

oppure

In [18]:
```
mA.I
```

Out[18]:
```
matrix([[-1.48,  0.36,  0.88],
        [ 0.56,  0.08, -0.36],
        [ 0.16, -0.12,  0.04]])
```

Risoluzione di un sistema lineare di equazioni

risolviamo il seguente sistema di equazioni usando numpy.linalg

$$\begin{aligned} x + 3y + 5z &= 10 \\ 2x + 5y + z &= 8 \\ 2x + 3y + 8z &= 3 \end{aligned}$$

con array

In [19]:
```
A=np.array([[1,3,5],[2,5,1],[2,3,8]])
b=np.array([10,8,3])
```

se passo b come riga:

In [20]:
```
la.solve(A,b)
```

Out[20]:
```
array([-9.28,  5.16,  0.76])
```

se passo b come colonna:

In [21]:
```
la.solve(A,b[:,np.newaxis])
```

Out[21]:
```
array([[-9.28],
       [ 5.16],
       [ 0.76]])
```

con matrici

in modo del tutto analogo al precedente ma se anche b è una matrice devo organizzarlo per colonna:

In [22]:
```
la.solve(np.matrix(A),np.matrix(b[:,np.newaxis]))
```

Out[22]:
```
matrix([[-9.28],
        [ 5.16],
        [ 0.76]])
```

Linear least-square problems

ho un modello che dipende in modo lineare da un certo numero di parametri (due, c1 e c2 nell'esempio). voglio determinare il set dei parametri che minimizzano la somma quadratica delle differenze tra un certo numero di osservazioni (10 nell'esempio)

In [23]:
```
#definisco il modello parametrico
def mymod(c1,c2):
    def wrapper(x):
        return c1*np.exp(-x)+c2*x
    return wrapper

#fisso i parametri
c1=5
c2=4
#definisco il modello con i parametri fissati
modello = mymod(c1,c2)


#variabile indipendente
xi = 0.1*np.arange(1,11)
#risposta attesa secondo il modello con i parametri fissati
yi = modello(xi)

#creo l'array delle 'osservazioni'
#aggiungo del rumore con distribuzione gaussiana a media nulla con varianza pari a
```

```
0.05*max(yi)
zi = yi + 0.05*max(yi)*np.random.randn(len(yi))
```

riscrivo il mio modello in modo che sia rappresentato da una matrice A moltiplicata per il vettore (riga) dei parametri

In [24]:
```
def matricemodello(x):
    return np.hstack( ( np.exp(-x)[:,newaxis] , x[:,newaxis]) )

A = matricemodello(xi)
```

verifico che la matrice A sia definita correttamente

In [25]:
```
print(yi)
print(np.dot(A,np.array([c1,c2])))
```

```
[ 4.92418709  4.89365377  4.9040911   4.95160023  5.0326533   5.14405818
  5.28292652  5.44664482  5.6328483   5.83939721]
[ 4.92418709  4.89365377  4.9040911   4.95160023  5.0326533   5.14405818
  5.28292652  5.44664482  5.6328483   5.83939721]
```

risolvo il problema lineare ai minimi quadrati usando il vettore z al quale ho aggiunto il rumore usando la funzione linalg.lstsq dove

lstsq(a, b) solves the equation A c = z by computing a vector c that minimizes the Euclidean 2-norm || z - A c ||^2

In [26]:
```
c , sommaresidui ,rango, sv = la.lstsq(A,zi)
```

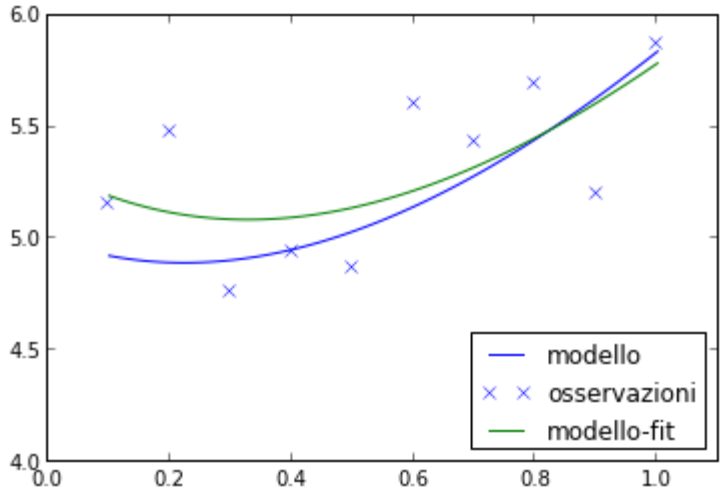
stampo la soluzione (i parametri)

In [27]:
```
print('c=',c)
```

```
c= [ 5.31571034  3.83108295]
```

plotto il modello di partenza, le osservazioni (con rumore gaussiano) e il modello ottenuto mediante la soluzione del problema.

In [39]:
```
xfitto=np.linspace(0.1,1.0,100)
ymod    = modello(xfitto)
ymodfit = mymod(c[0],c[1])(xfitto)
axis([0,1.1,4,6])
plot(xfitto,ymod,label='modello')
plot(xi , zi , 'bx',label='osservazioni')
plot(xfitto,ymodfit,label='modello-fit')
legend(loc='lower right')
savefig('./img/fit01.png')
```



Decomposizione in autovalori ed autovettori

Il modulo linalg mette a disposizione var metodi di scomposizione

In [29]:
```
A = np.matrix( [[1, 5, 2] , [2 , 4 , 1] , [3 ,  6  , 2]])
```

risolvo il problema con linalg.eig:

In [30]:
```
autoval , autovett = la.eig(A)
```

stampo gli autovalori

In [31]:
```
autoval
```

Out[31]:
```
array([ 7.9579162 , -1.25766471,  0.2997485 ])
```

stampo gli autovettori (organizzati per colonna)

In [33]:
```
autovett
```

Out[33]:
```
matrix([[-0.5297175 , -0.90730751,  0.28380519],
        [-0.44941741,  0.28662547, -0.39012063],
        [-0.71932146,  0.30763439,  0.87593408]])
```

estraggo gli autovettori (organizzati per colonna)

In [35]:
```
v1 = mat(autovett[:,0])
v2 = mat(autovett[:,1])
v3 = mat(autovett[:,2])
```

verifica

In [37]:
```
aval1,aval1,aval3=autoval
max(ravel(abs(A*v1-aval1*v1))) ,  max(ravel(abs(A*v2-aval2*v2))) ,
max(ravel(abs(A*v3-aval3*v3)))
```

Out[37]:
```
(4.4408920985006262e-15, 8.8817841970012523e-16, 3.8857805861880479e-16)
```