EE550 - Linear Least Squares Regression Simulation with Python

March 27, 2017

Mine Melodi Çalışkan - 2015705009

Consider a static system: $y(i) = b_0 + b_1 u(i) + b_2 u^2(i) + e(i)$

where $e(i)$ is zero-mean Gaussian noise with standard deviation 0.1

0.0.1

Generate 10 data points with i=0,1,...9 with the above model and plot them.

0.0.2

Take $b_0 = 1.1, \;\; b_1 = 0.45, \;\; b_2 = 0.1$

In [1]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt


np.random.seed(123)

mu, sigma = 0, 0.1
#zero-mean Gaussian noise with standard deviation 0.1
noise= np.random.normal(mu, sigma, 10)

#10 random data points
data=np.random.rand(10)*4

#original parameter values
b=np.array([1.1, 0.45, 0.1])

y=[]
for u in data:
    y.append(b[0]+b[1]*u+b[2]*(u**2))

for i in range(len(y)):
    y[i]+=noise[i]
```

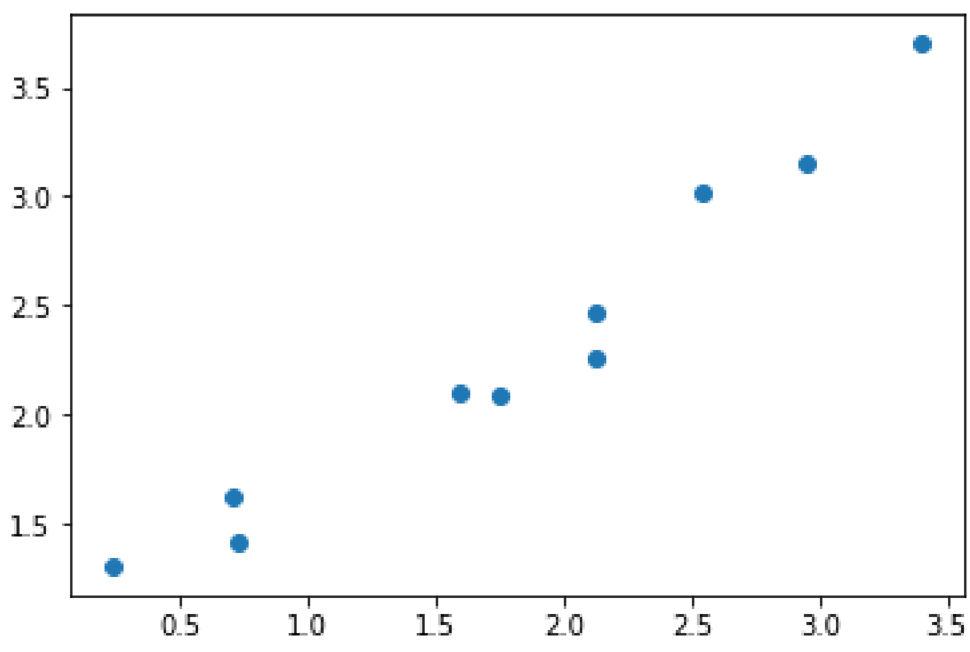
In [2]: data

Out[2]:
```
array([ 1.75428898,  0.23871159,  1.59217702,  2.95198162,  0.72996692,
        0.70180702,  2.1262055 ,  2.12731035,  2.53760383,  3.39772718])
```

In [3]:
```python
y #original outputs
```

Out[3]:
```
[2.088619962178587,
 1.3128530807055696,
 2.0982802761224684,
 3.1491818091364028,
 1.4239102603137068,
 1.6302101247515846,
 2.2661995294290351,
 2.4669433257243809,
 3.0124586732002672,
 3.6967581855778513]
```

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt


plt.scatter(data, y)
plt.show()
```



One estimation model can be written as: $y = \phi^T \theta$ with regression vector $\phi^T = [1 \; u(i) \; u^2(i)]$

$$\text{and parameters vector } \theta = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

0.0.3 Estimate the following model's parameters with least mean square solution.

$$\begin{aligned} \text{Model a: } & y(i) = b_0 \\ \text{Model b: } & y(i) = b_0 + b_1 u \\ \text{Model c: } & y(i) = b_0 + b_1 u + b_2 u^2 \\ \text{Model d: } & y(i) = b_0 + b_1 u + b_2 u^2 + b_3 u^3 \end{aligned}$$

Write the polynomial for each case with estimated parameters.

0.0.4 With each model's parameter estimates plot the estimated polynomial (i.e plot "y vs. u" for each model with the estimated parameters.)

Parameter estimation formula with least mean square solution: $\theta = (\phi^T \phi)^{-1} \phi^T Y$

$$Cost = \frac{1}{2} E^T . E$$

```
In [5]: modelA=[0]
        modelB=[0,1]
        modelC=[0,1,2]
        modelD=[0,1,2,3]
```

Model a

```
In [6]: resultA = np.array([x**p for x in data for p in modelA])
        resultA=resultA.reshape(10,1)

In [7]: resultA

Out[7]: array([[ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.],
               [ 1.]])

In [8]: X_A=resultA
        X_A1=np.linalg.inv(np.dot(X_A.T,X_A))
        X_A2=np.dot(X_A.T,y)
        X_A3=np.dot(X_A1,X_A2)
        a0=X_A3
```

```
In [9]: a0
```

```
Out[9]: array([ 2.31454152])
```

```
In [10]: print "a0 :" , a0[0]
```

```
a0 : 2.31454152271
```

$$\Rightarrow y_a(i) = 2.31454152$$

```
In [11]: def model_a(coefficient):
             y_a=[]
             for i in range(10):
                 a_=coefficient*(data[i]**0)
                 y_a.append(a_)
             return y_a
```

```
In [12]: y_a=model_a(a0[0])
```

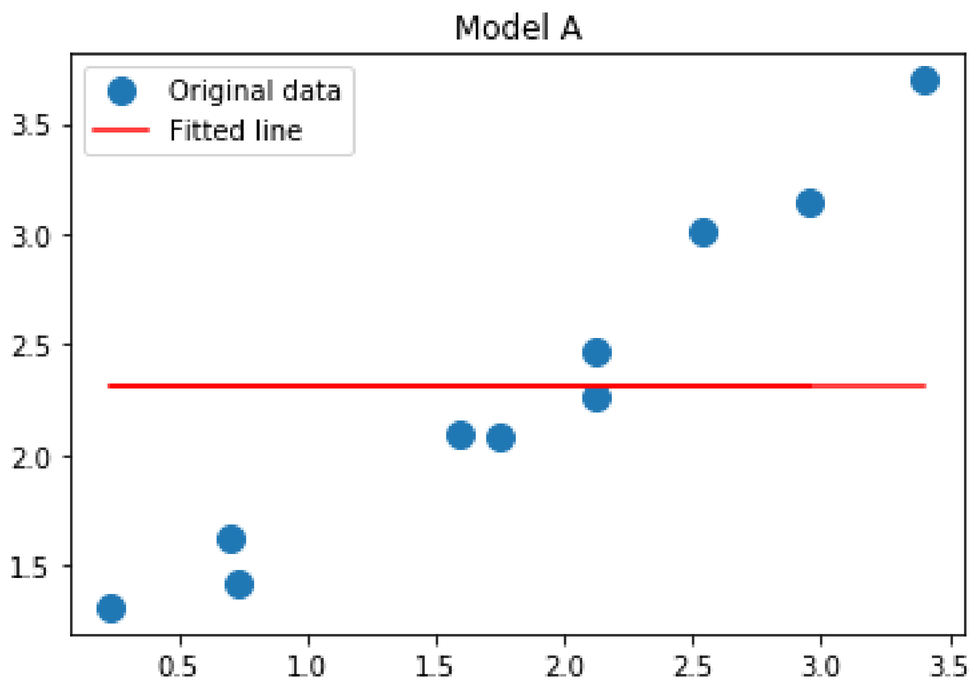
```
In [13]: Error_a=(np.array(y)-np.array(y_a)).T
```

```
In [14]: Cost_a=0.5*Error_a.T.dot(Error_a)
```

```
In [15]: Cost_a
```

```
Out[15]: 2.7412608192917456
```

```
In [16]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         plt.plot(data, y_a, 'r', label='Fitted line')
         plt.legend()
         plt.title('Model A')
         plt.show()
```



Model b

```
In [17]: resultB = np.array([x**p for x in data for p in modelB])
         resultB=resultB.reshape(10,2)
```

```
In [18]: resultB
```

```
Out[18]: array([[ 1.        ,  1.75428898],
                [ 1.        ,  0.23871159],
                [ 1.        ,  1.59217702],
                [ 1.        ,  2.95198162],
                [ 1.        ,  0.72996692],
                [ 1.        ,  0.70180702],
                [ 1.        ,  2.1262055 ],
                [ 1.        ,  2.12731035],
                [ 1.        ,  2.53760383],
                [ 1.        ,  3.39772718]])
```

```
In [19]: X_B=resultB
         X_B1=np.linalg.inv(np.dot(X_B.T,X_B))
         X_B2=np.dot(X_B.T,y)
         X_B3=np.dot(X_B1,X_B2)
         [b0,b1]=X_B3
```

```
In [20]: print "b0 :" , b0
         print "b1 :" , b1
```

```
b0 : 0.964037602019
b1 : 0.743760481702
```

$$\Rightarrow y_b(i) = 0.964037602019 + 0.743760481702 * u(i)$$

```
In [21]: def model_b(coefficient):
             y_b=[]
             for i in range(10):
                 b_=coefficient[0]+coefficient[1]*data[i]
                 y_b.append(b_)
             return y_b
```

```
In [22]: y_b=model_b([b0,b1])
```

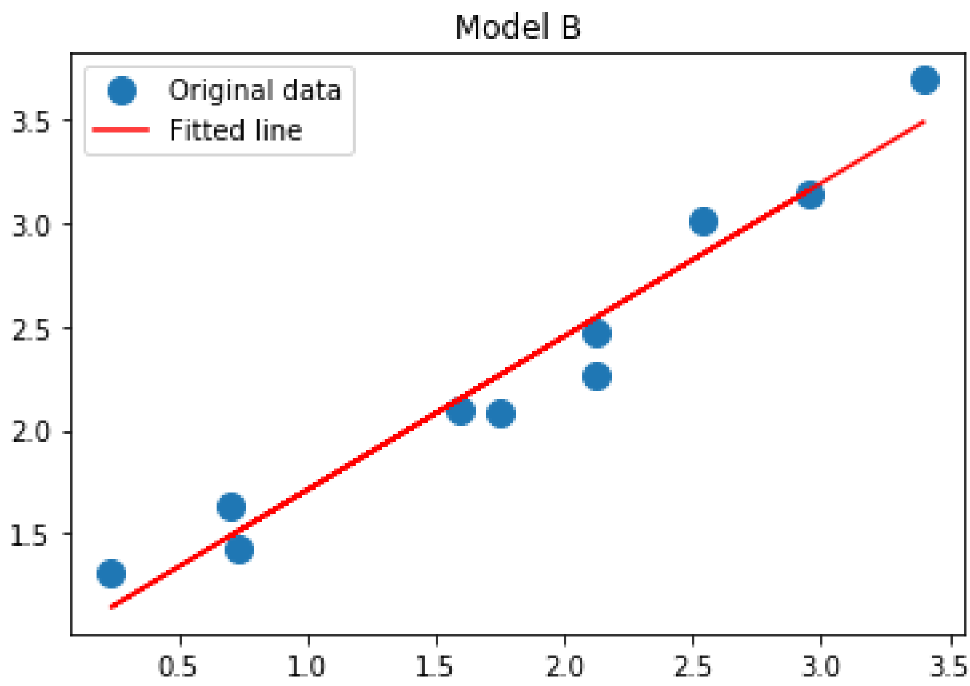
```
In [23]: Error_b=(np.array(y)-np.array(y_b)).T
```

```
In [24]: Cost_b=0.5*Error_b.T.dot(Error_b)
```

```
In [25]: Cost_b
```

```
Out[25]: 0.12228544948010207
```

```
In [26]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         plt.plot(data, y_b , 'r', label='Fitted line')
         plt.legend()
         plt.title('Model B')
         plt.show()
```



Model c

```
In [27]: resultC = np.array([x**p for x in data for p in modelC])
         resultC=resultC.reshape(10,3)
```

```
In [28]: resultC
```

```
Out[28]: array([[  1.        ,   1.75428898,   3.07752982],
                [  1.        ,   0.23871159,   0.05698322],
                [  1.        ,   1.59217702,   2.53502767],
                [  1.        ,   2.95198162,   8.7141955 ],
                [  1.        ,   0.72996692,   0.53285171],
                [  1.        ,   0.70180702,   0.4925331 ],
                [  1.        ,   2.1262055 ,   4.52074981],
                [  1.        ,   2.12731035,   4.52544932],
                [  1.        ,   2.53760383,   6.43943322],
                [  1.        ,   3.39772718,  11.54454996]])
```

```
In [29]: X_C=resultC
         X_C1=np.linalg.inv(np.dot(X_C.T,X_C))
         X_C2=np.dot(X_C.T,y)
         X_C3=np.dot(X_C1,X_C2)
         [c0,c1,c2]=X_C3
```

```
In [30]: print "c0 :" ,c0
         print "c1 :" ,c1
         print "c2 :" ,c2
```

```
c0 : 1.24415957039
c1 : 0.293150560134
c2 : 0.126789455065
```

$$\Rightarrow y_c(i) = 1.24415957039 + 0.293150560134 * u(i) + 0.126789455065 * u^2(i)$$

```
In [31]: def model_c(coefficient):
             y_c=[]
             for i in range(10):
                 c_=coefficient[0]+coefficient[1]*data[i]
                 +coefficient[2]*(data[i]**2)
                 y_c.append(c_)
             return y_c
```

```
In [32]: y_c=model_c([c0,c1,c2])
```

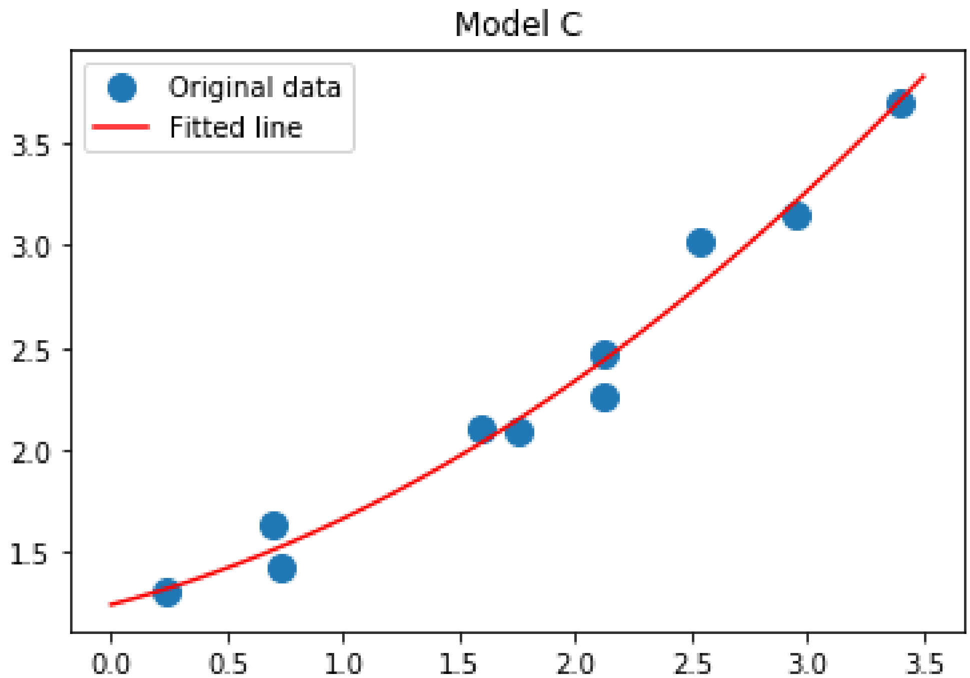
```
In [33]: Error_c=(np.array(y)-np.array(y_c)).T
```

```
In [34]: Cost_c=0.5*Error_c.T.dot(Error_c)
```

```
In [35]: Cost_c
```

```
Out[35]: 0.055450588884792172
```

```
In [36]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         x = np.linspace(0, 3.5, 1000)
         plt.plot(x, c0+x*c1+(x**2)*c2 , 'r', label='Fitted line')
         plt.legend()
         plt.title('Model C')
         plt.show()
```



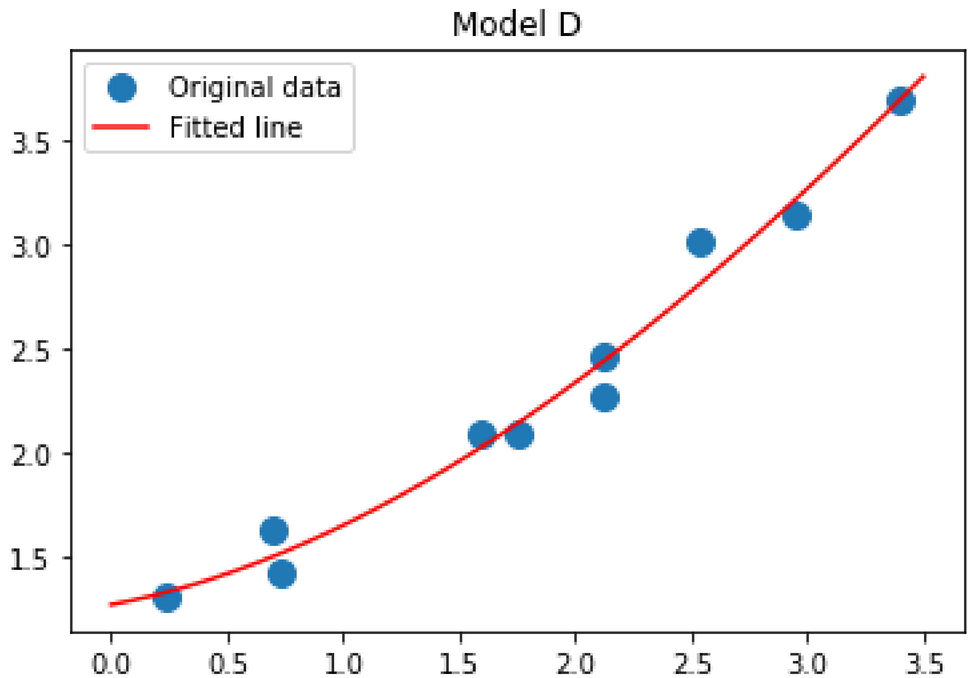
Model d

```
In [37]: resultD = np.array([x**p for x in data for p in modelD])
         resultD=resultD.reshape(10,4)

In [38]: resultD

Out[38]: array([[  1.00000000e+00,   1.75428898e+00,   3.07752982e+00,
              5.39887665e+00],
            [  1.00000000e+00,   2.38711586e-01,   5.69832215e-02,
              1.36025552e-02],
            [  1.00000000e+00,   1.59217702e+00,   2.53502767e+00,
              4.03621280e+00],
            [  1.00000000e+00,   2.95198162e+00,   8.71419550e+00,
              2.57241450e+01],
            [  1.00000000e+00,   7.29966922e-01,   5.32851707e-01,
              3.88964120e-01],
            [  1.00000000e+00,   7.01807025e-01,   4.92533100e-01,
              3.45663189e-01],
            [  1.00000000e+00,   2.12620550e+00,   4.52074981e+00,
              9.61204309e+00],
            [  1.00000000e+00,   2.12731035e+00,   4.52544932e+00,
              9.62703517e+00],
            [  1.00000000e+00,   2.53760383e+00,   6.43943322e+00,
              1.63407304e+01],
            [  1.00000000e+00,   3.39772718e+00,   1.15445500e+01,
              3.92252312e+01]])
```

```
In [39]: X_D=resultD
         X_D1=np.linalg.inv(np.dot(X_D.T,X_D))
         X_D2=np.dot(X_D.T,y)
         X_D3=np.dot(X_D1,X_D2)
         [d0,d1,d2,d3]=X_D3

In [40]: print "d0 :" ,d0
         print "d1 :" ,d1
         print "d2 :" ,d2
         print "d3 :" ,d3

d0 : 1.2732159194
d1 : 0.20697345394
d2 : 0.183218992761
d3 : -0.0101217909375
```

$\Rightarrow y_d(i) = 1.2732159194 + 0.20697345394 * u(i) + 0.183218992761 * u^2(i) - 0.0101217909375 * u^3(i)$

```
In [41]: def model_d(coefficient):
             y_d=[]
             for i in range(10):
                 d_=coefficient[0]+coefficient[1]*data[i]
                 +coefficient[2]*(data[i]**2)+coefficient[3]*(data[i]**3)
                 y_d.append(d_)
             return y_d

In [42]: y_d=model_d([d0,d1,d2,d3])

In [43]: Error_d=(np.array(y)-np.array(y_d)).T

In [44]: Cost_d=0.5*Error_d.T.dot(Error_d)

In [45]: Cost_d

Out[45]: 0.055196168590172449

In [46]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         x = np.linspace(0, 3.5, 1000)
         plt.plot(x, d0+x*d1+(x**2)*d2+(x**3)*d3  , 'r', label='Fitted line')
         plt.legend()
         plt.title('Model D')
         plt.show()
```



0.0.5 Estimation with Recursive Least Squares

(1) Recursive equation of coefficients θ : $\theta(t) = \theta(t-1) + K(t)(Y(t) - \phi^T(t)\theta(t-1))$ where $K(t) = P(t)\phi(t)$

(2) Recursive equation of gain matrix P : $P(t) = P(t-1) - P(t-1)\phi(t)[1 + \phi^T(t)P(t+1)\phi(t)]^{-1}\phi^T(t)P(t-1)$

Note: Here the term $[1 + \phi^T(t)P(t+1)\phi(t)]$ is scalar.

(3) Recursive equation of K: Using Equation (2);

$$
\begin{aligned}
K(t) &= P(t)\phi(t) = [P(t-1) - P(t-1)\phi(t)[1 + \phi^T(t)P(t+1)\phi(t)]^{-1}\phi^T(t)P(t-1)]\phi(t) \\
&= P(t-1)\phi(t)[I - [\phi^T(t)P(t-1)\phi(t) + 1]^{-1}\phi^T(t)P(t-1)\phi(t)] \\
&= P(t-1)\phi(t)[\phi^T(t)P(t-1)\phi(t) + 1]^{-1}\{[\phi^T(t)P(t-1)\phi(t) + 1] - \phi^T(t)P(t-1)\phi(t)\} \\
&= P(t-1)\phi(t)[\phi^T(t)P(t-1)\phi(t) + 1]^{-1}
\end{aligned}
$$

(4) Recursive equation of P in the code: $P(t) = P(t-1) - K(t)\phi^T(t)P(t-1)$

(5) Error term: $E(t) = y(t) - \phi^T\theta(t)$

```
While LMS algorithm is widely used for batch processing,
RLS algorithm can be used for real-time regression problems,
since it has a sequential nature and it doesn't requires
computation of a n*n matrix inverse.
```

In [47]:
```python
def recursive_leastsquares(N,inputs,outputs):

    #Initial values of coefficients are set to zero
    Theta=np.zeros((N,1))

    #Gain matrix initialization
    #It requires large values
    P=np.eye(N)*1000

    #Regressor initialization
    X = np.ones((10,N))

    #Estimated output generations
    #Here only our 4 models are considered.
    if N==1:
        X=X

    elif N==2:
         for i in range(10):
            X[i,1] = inputs[i]

    elif N==3:
        for i in range(10):
            X[i,1] = inputs[i]
            X[i,2] = inputs[i]**2

    else:
        for i in range(10):
            X[i,1] = inputs[i]
            X[i,2] = inputs[i]**2
            X[i,3] = inputs[i]**3

    #Recursion part
    for n in range(10):

        R=np.array([X[n,:]])
        K=P.dot(R.T)/(1+np.dot(R,P).dot(R.T)) #Equation (3)
        P=P-K*R.dot(P) #Equation (4)
        E=outputs[n]-R.dot(Theta) #Equation (5)
        Theta=Theta+K*E #Equation (1)

    #Returns estimated coefficients
    return  Theta
```

In [48]:
```python
inputs=data
```

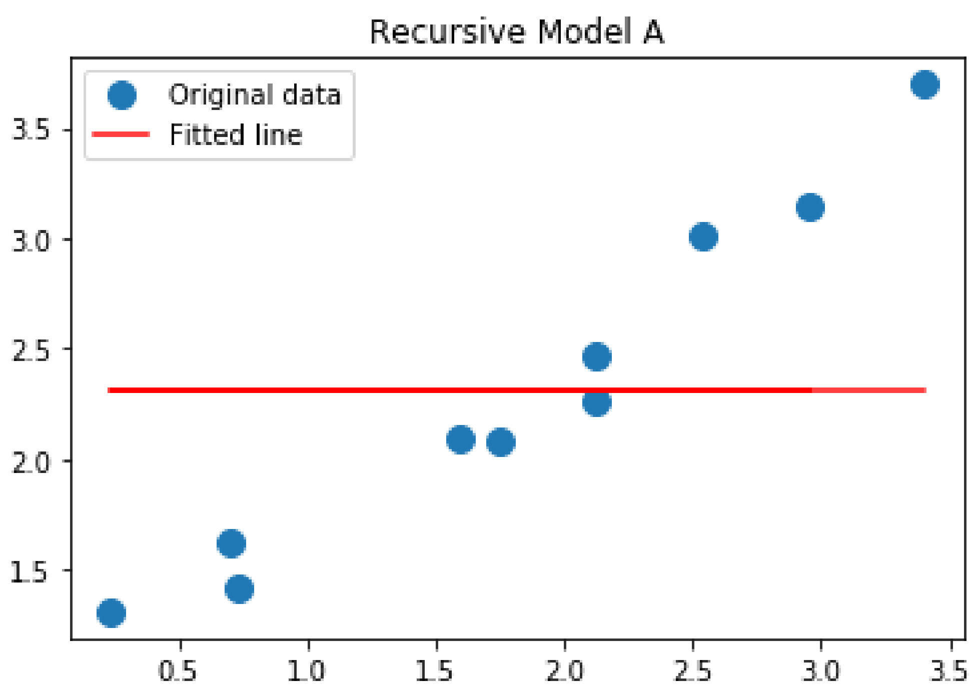
In [49]:
```python
outputs=np.array([y]).transpose()
```

Recursive Model a

```
In [50]: Ra_b0=recursive_leastsquares(1,data,outputs)

In [51]: print "Ra_b0", Ra_b0.flatten()

Ra_b0 [ 2.31431009]
```

$$\Rightarrow y_{recursive_a}(i) = 2.31431009$$

```
In [52]: Ra_y=model_a(Ra_b0[0][0])

In [53]: Error_Ra=(np.array(y)-np.array(Ra_y)).T

In [54]: Rcost_a=0.5*Error_Ra.T.dot(Error_Ra)

In [55]: Rcost_a

Out[55]: 2.7412610870933056

In [56]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         plt.plot(data, Ra_y, 'r', label='Fitted line')
         plt.legend()
         plt.title('Recursive Model A')
         plt.show()
```

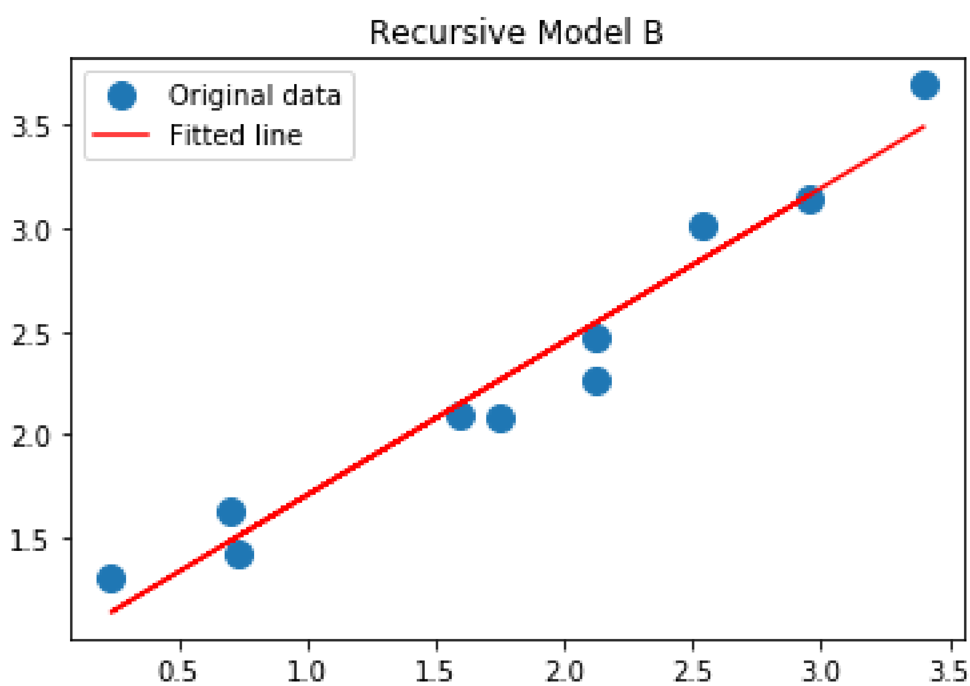


Recursive Model b

```
In [57]: Rb_=recursive_leastsquares(2,data,outputs).flatten()

In [58]: Rb_

Out[58]: array([ 0.9637483 ,  0.74386673])
```

$$\Rightarrow y_{recursive_b}(i) = 0.9637483 + 0.74386673 * u(i)$$

```
In [59]: Rb_y=model_b(Rb_)

In [60]: Error_Rb=(np.array(y)-np.array(Rb_y)).T

In [61]: Rcost_b=0.5*Error_Rb.T.dot(Error_Rb)

In [62]: Rcost_b

Out[62]: 0.12228554937033143

In [63]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         plt.plot(data, Rb_y, 'r', label='Fitted line')
         plt.legend()
         plt.title('Recursive Model B')
         plt.show()
```

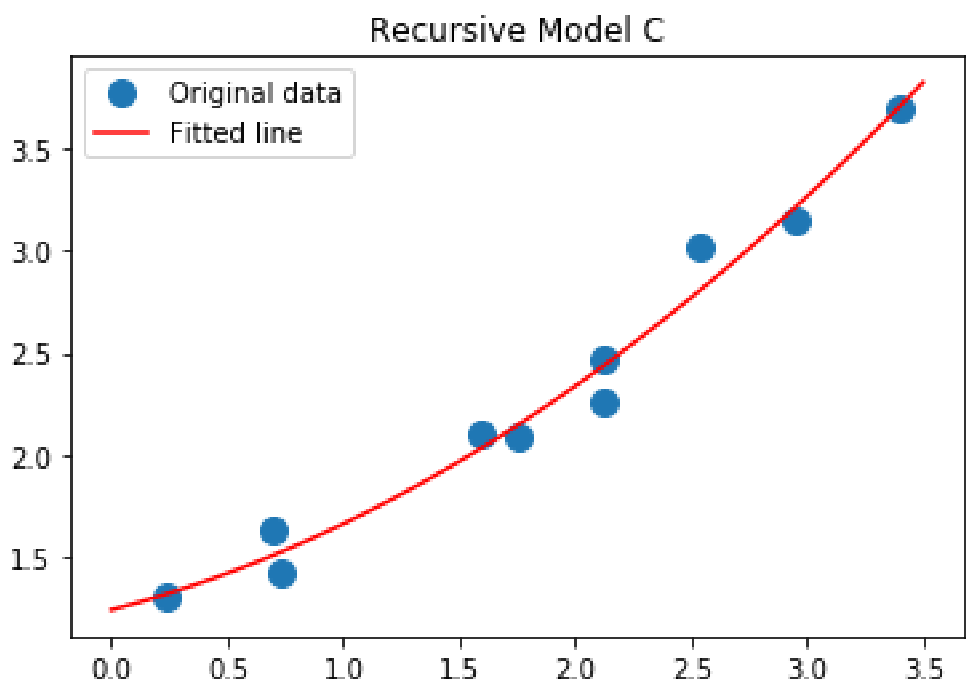


Recursive Model c

```
In [64]: Rc_=recursive_leastsquares(3,data,outputs).flatten()

In [65]: Rc_

Out[65]: array([ 1.24317313,  0.29413912,  0.12656964])
```

$$\Rightarrow y_{recursive_c}(i) = 1.24317313 + 0.29413912 * u(i) + 0.12656964 * u^2(i)$$

```
In [66]: Rc_y=model_c(Rc_)

In [67]: Error_Rc=(np.array(y)-np.array(Rc_y)).T

In [68]: Rcost_c=0.5*Error_Rc.T.dot(Error_Rc)

In [69]: Rcost_c

Out[69]: 0.055451070565717707

In [70]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         x = np.linspace(0, 3.5, 1000)
         plt.plot(x,
                  Rc_[0]+x*Rc_[1]+(x**2)*Rc_[2] ,
                  'r', label='Fitted line')
         plt.legend()
         plt.title('Recursive Model C')
         plt.show()
```



** Recursive Model d**

```
In [71]: Rd_=recursive_leastsquares(4,data,outputs).flatten()
```

```
In [72]: Rd_
```

```
Out[72]: array([ 1.27045056,  0.21302459,  0.17975731, -0.00954896])
```

$$\Rightarrow y_{recursive_d}(i) = 1.27045056 + 0.21302459 * u(i) + 0.17975731 * u^2(i) - 0.00954896 * u^3(i)$$

```
In [73]: Rd_y=model_d(Rd_)
```

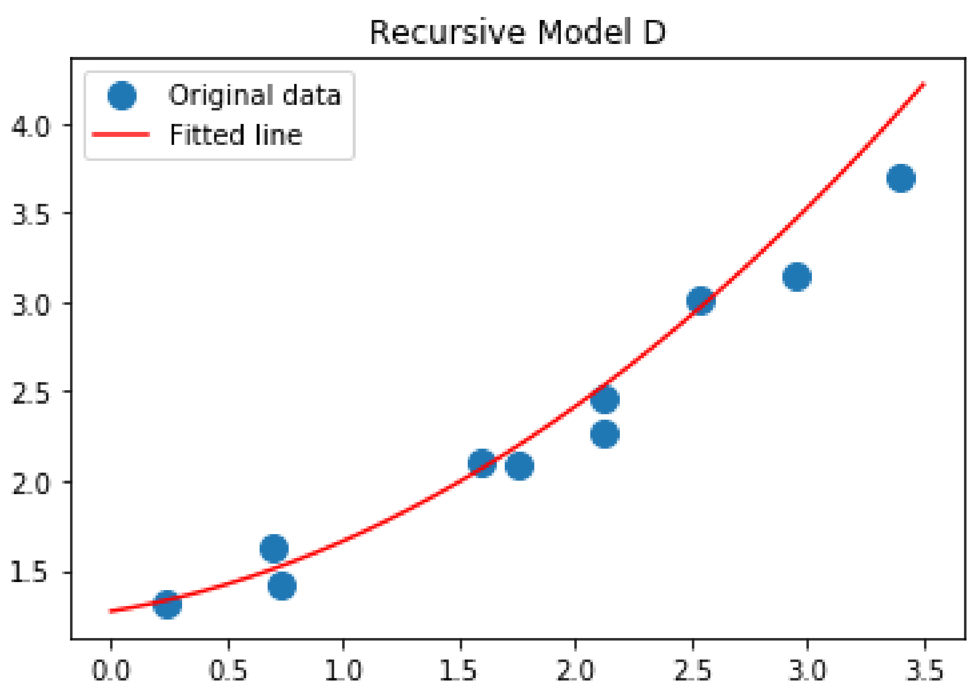
```
In [74]: Error_Rd=(np.array(y)-np.array(Rd_y)).T
```

```
In [75]: Rcost_d=0.5*Error_Rd.T.dot(Error_Rd)
```

```
In [76]: Rcost_d
```

```
Out[76]: 0.05519759456463692З
```

```
In [77]: plt.plot(data, y, 'o', label='Original data', markersize=10)
         x = np.linspace(0, 3.5, 1000)
         plt.plot(x,
                  Rd_[0]+x*Rd_[1]+(x**2)*Rd_[2] ,
                  'r', label='Fitted line')
         plt.legend()
         plt.title('Recursive Model D')
         plt.show()
```



0.0.6 Generate a table showing each model's parameters along with the value of the cost function.

```
In [78]: from astropy.table import Table

         rows1=[('a',a0[0], 0,0,0),('b', b0, b1,0,0), ('c', c0, c1,c2,0),
                ('d', d0,d1,d2,d3)]

         t1 = Table(rows=rows1, names=('Model', 'b0*', 'b1*', 'b2*','b3*'))
         print(t1)
```

```
Model      b0*            b1*            b2*             b3*
----- -------------- -------------- -------------- ----------------
    a  2.31454152271            0.0            0.0              0.0
    b 0.964037602019 0.743760481702            0.0              0.0
    c  1.24415957039 0.293150560134 0.126789455065              0.0
    d   1.2732159194  0.20697345394 0.183218992761 -0.0101217909375
```

```
In [79]: cost_rows1=[('a',Cost_a),('b',Cost_b), ('c',Cost_c),
                ('d', Cost_d)]

         cost_t1 = Table(rows=cost_rows1, names=('model', 'cost'))
         print(cost_t1)
```

```
model      cost
----- ---------------
    a   2.74126081929
    b   0.12228544948
    c 0.0554505888848
    d 0.0551961685902
```

```
In [80]: rows2=[('Recursive a',Ra_b0[0][0], 0,0,0),('Recursive b', Rb_[0], Rb_[1],(
                ('Recursive c', Rc_[0], Rc_[1],Rc_[2],0),
                ('Recursive d', Rd_[0],Rd_[1],Rd_[2],Rd_[3])]

         t2 = Table(rows=rows2, names=('Model', 'b0*', 'b1*', 'b2*','b3*'))
         print(t2)
```

```
   Model          b0*            b1*            b2*              b3*
----------- -------------- -------------- -------------- -----------------
Recursive a   2.3143100917            0.0            0.0               0.0
Recursive b 0.963748295868 0.743866734406            0.0               0.0
Recursive c  1.24317313113 0.294139123399 0.126569637724               0.0
Recursive d  1.27045055588 0.213024589116 0.179757314603 -0.00954896302397
```

```
In [81]: cost_rows2=[('Recursive a',Rcost_a),('Recursive b',Rcost_b),
                     ('Recursive c',Rcost_c),('Recursive d', Rcost_d)]

         cost_t2 = Table(rows=cost_rows2, names=('model', 'cost'))
         print(cost_t2)
```

```
   model           cost
----------- ---------------
Recursive a   2.74126108709
Recursive b   0.12228554937
Recursive c 0.0554510705657
Recursive d 0.0551975945646
```

In either case Model c and Model b gives better results than the others since our model is a noisy version of Model c.