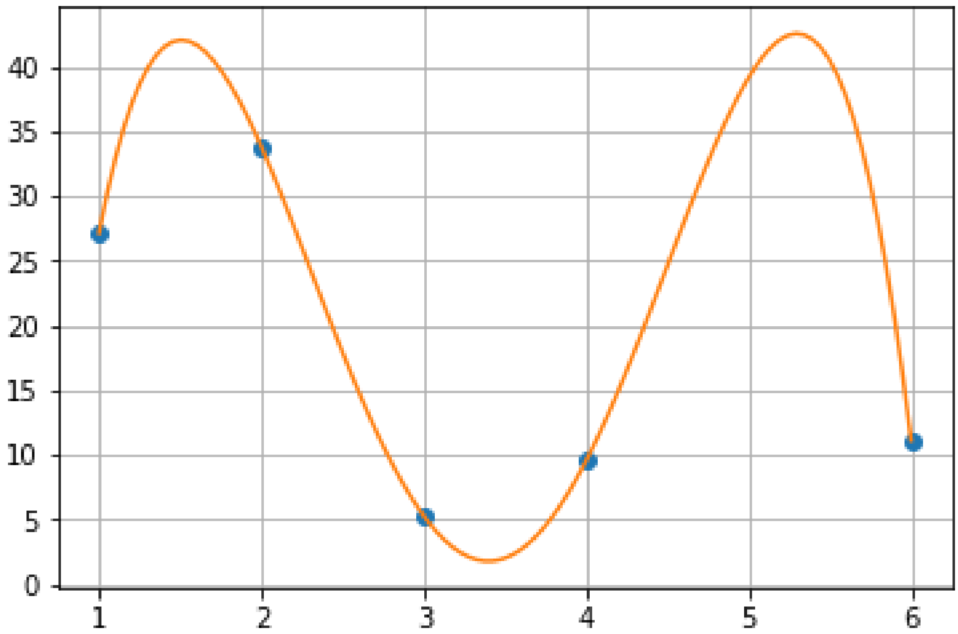
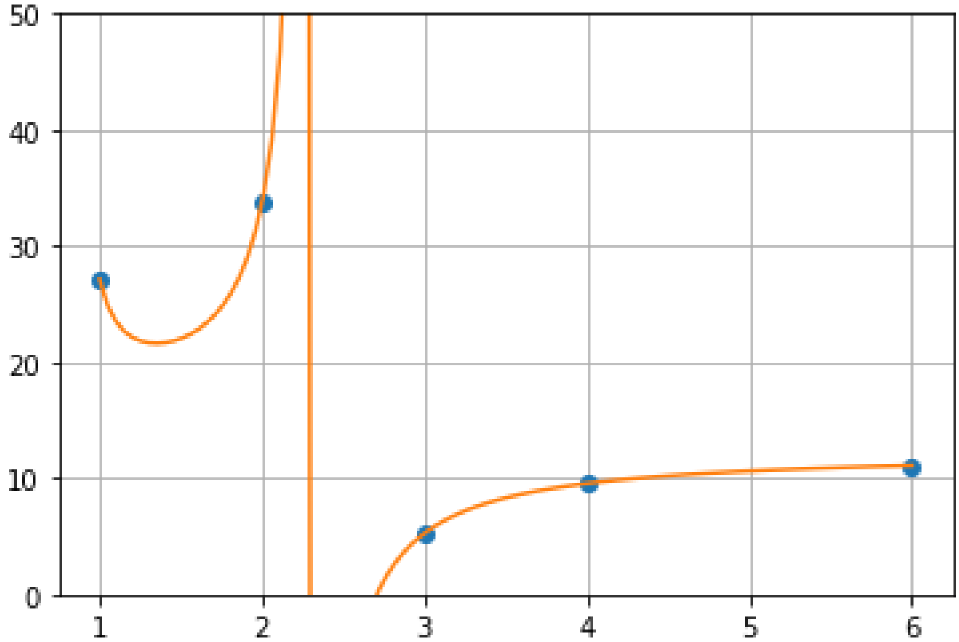
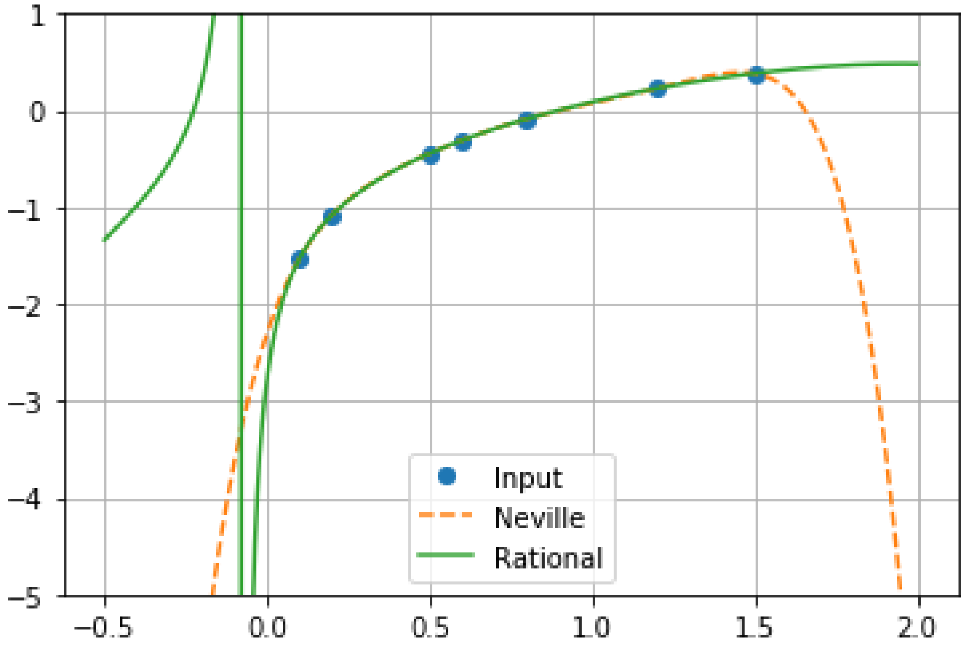
```
## neville fitting

#xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
#yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])

xar = np.array([1,2,3,4,6])
yar = np.array([27.130,33.772,5.257,9.549,11.098])

#xx   = np.linspace(1,4.5,100)
xlen = (len(xx))
xx = np.linspace(1,6,100)
yy   = np.zeros(xlen)

ncount = 0
#for x in xx:
#    yy[ncount] = neville(xar,yar,x)
#    ncount += 1

yy = mapar(lambda x:neville(xar,yar,x),xx)

plt.plot(xar,yar,'o',xx,yy,'-')
plt.grid()
plt.show()
```



```python
## rational fitting

#xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
#yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])

xar = np.array([1,2,3,4,6])
yar = np.array([27.130,33.772,5.257,9.549,11.098])

#xx   = np.linspace(1,4.5,100)
xx = np.linspace(1,6,100)
xlen = (len(xx))
yy   = np.zeros(xlen)

ncount = 0
#for x in xx:
#    yy[ncount] = rational(xar,yar,x)
#    ncount += 1

yy = mapar(lambda x:rational(xar,yar,x),xx)

plt.plot(xar,yar,'o',xx,yy,'-')
plt.ylim(0,50)
plt.grid()
plt.show()
```

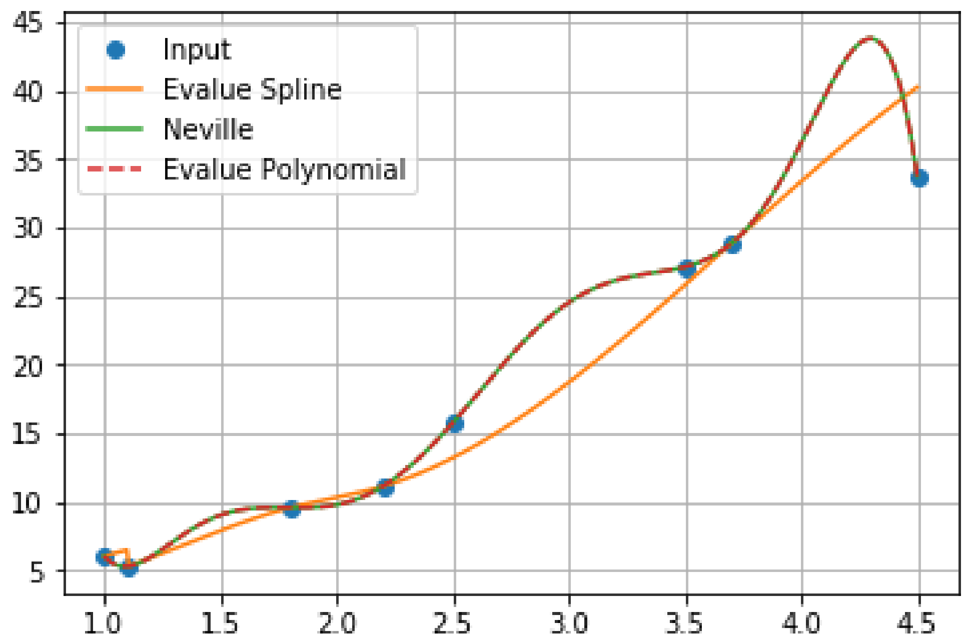


```python
xData = np.array([0.1,0.2,0.5,0.6,0.8,1.2,1.5])
yData = np.array([-1.5342,-1.0811,-0.4445,-0.3085, -0.0868,0.2281,0.3824])

x = np.linspace(-0.5,2,1000)

nev = mapar(lambda x:neville(xData,yData,x),x)
rat = mapar(lambda x:rational(xData,yData,x),x)

plt.ylim(-5,1)
plt.plot(xData,yData,'o',x,nev,'--',x,rat,'-')
plt.legend(['Input','Neville','Rational'])
plt.grid()
plt.show()
```



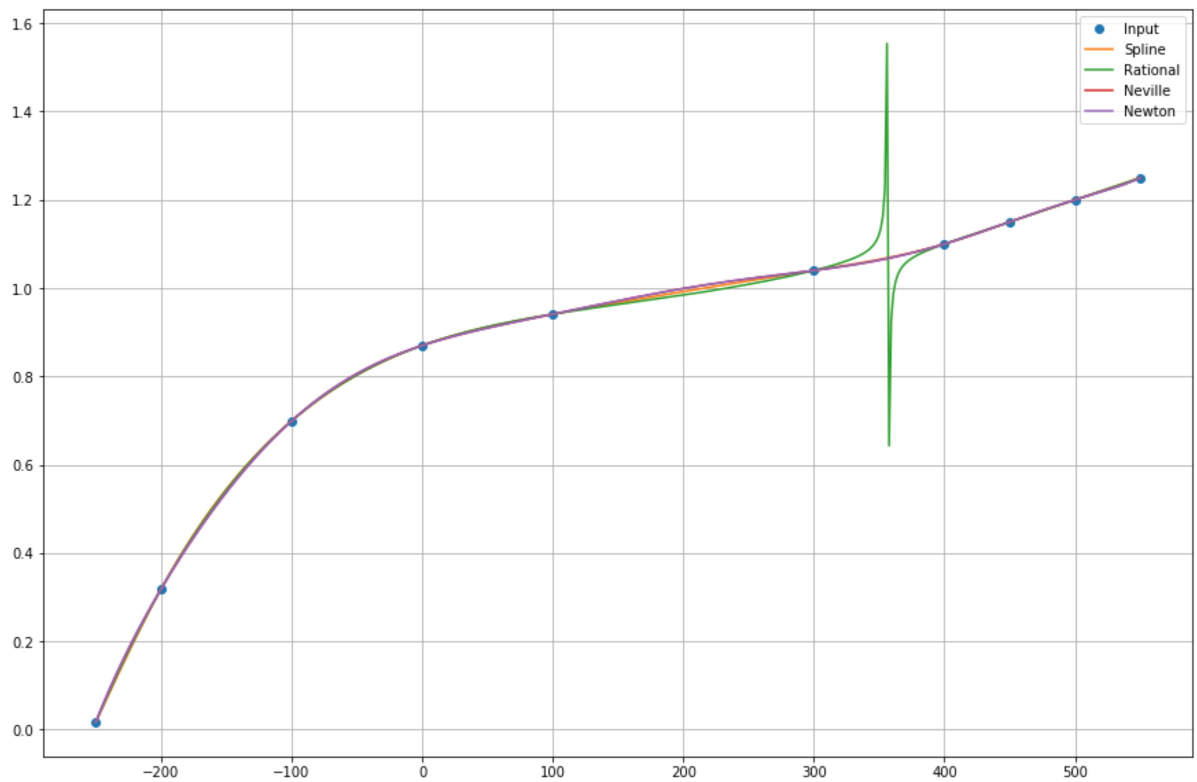

Rational fitting is more naturaler than Polynomial fitting but It can divergence.

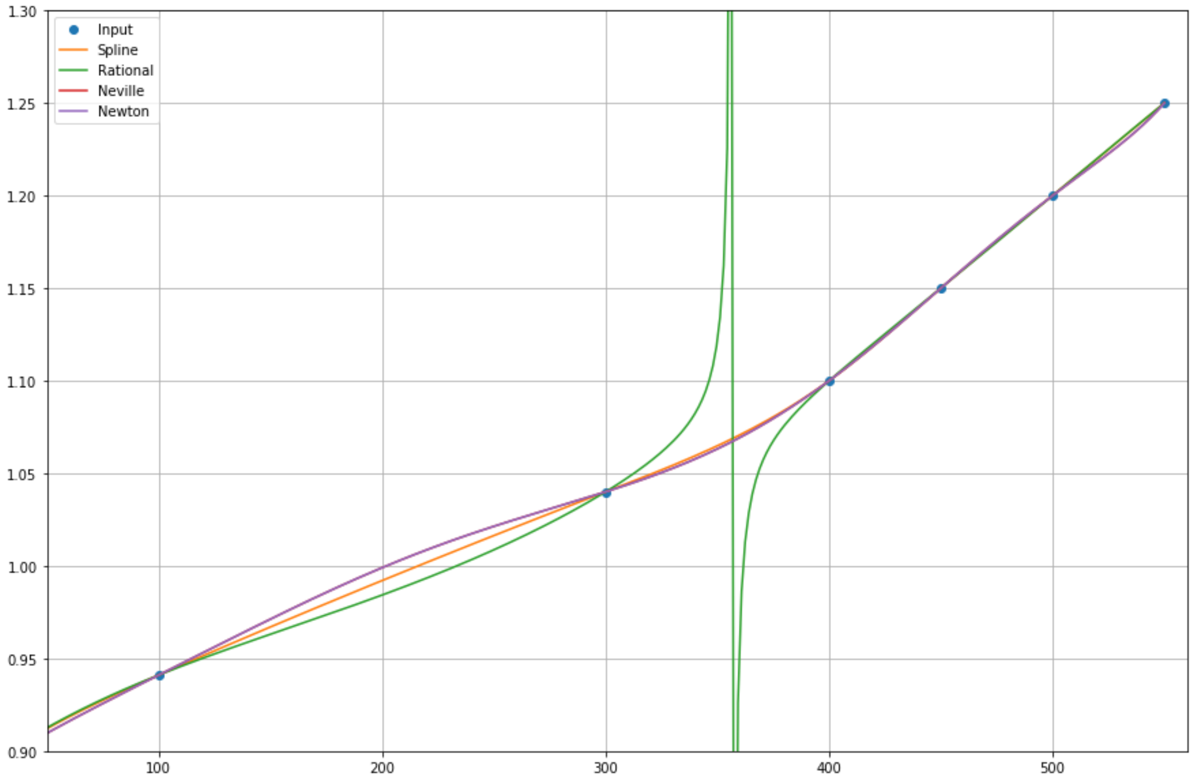
```python
xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
#xar = np.array([1,2,3,4,5,6,7,8])
yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])

x = np.linspace(1,4.5,500)

a = coeffts(xar,yar)
k = curvatures(xar,yar)

pol = mapar(lambda x:evalPoly(a,xar,x),x)
nev = mapar(lambda x:neville(xar,yar,x),x)
spl = mapar(lambda x:evalSpline(xar,yar,k,x),x)

#plt.ylim(0,50)
plt.plot(xar,yar,'o',x,spl,'-',x,nev,'-',x,pol,'--')
plt.legend(['Input','Evalue Spline','Neville','Evalue Polynomial'])
plt.grid()
plt.show()
```




Spline fitting is more stable than Polynomial fitting.

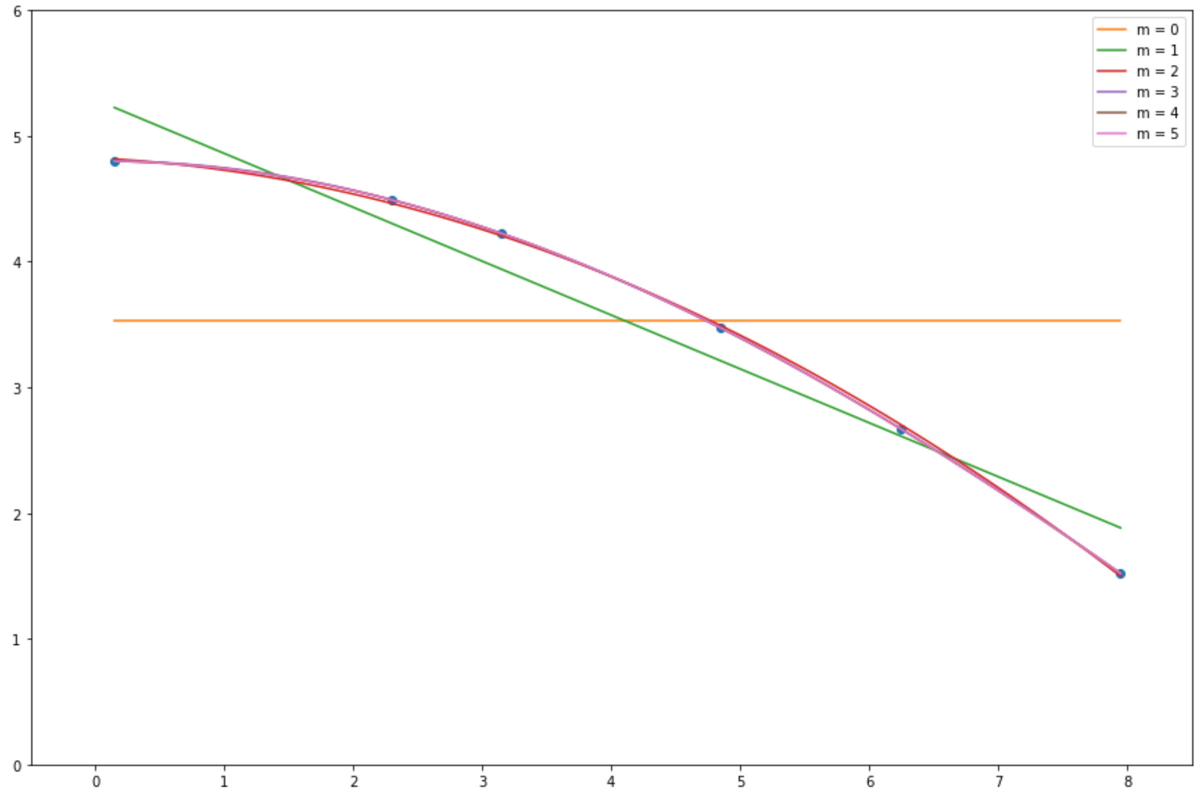
```
## Compare fitting done with Newton, Neville, rational and splines
## What is the optimal number of points for fitting the above data with a polynomial?

xData = np.array([-250,-200,-100,0,100,300,400,450,500,550])
yData = np.array([0.0163,0.318,0.699,0.870,0.941,1.04,1.1,1.15,1.2,1.25])

x = np.linspace(min(list(xData)),max(list(xData)),500)

a = coeffts(xData,yData)
k = curvatures(xData,yData)

pol = mapar(lambda x:evalPoly(a,xData,x),x)
nev = mapar(lambda x:neville(xData,yData,x),x)
rat = mapar(lambda x:rational(xData,yData,x),x)
spl = mapar(lambda x:evalSpline(xData,yData,k,x),x)

plt.figure(figsize=[15,10])

plt.plot(xData,yData,'o',x,spl,'-',x,rat,'-',x,nev,'-',x,pol,'-')
plt.legend(['Input','Spline','Rational','Neville','Newton'])
plt.grid()
plt.show()
```






Spline fitting no divergence and has stablelity. so if i have to data fitting, i use Spline fitting code.

```python
## Fitting Linear Forms
## Use polyFit(xData,yData,m)
## x   0.15   2.30   3.15   4.85   6.25   7.95
## y   4.79867  4.49013        4.2243   3.47313   2.66674  1.51909

xar = np.array([0.15,2.30,3.15,4.85,6.25,7.95])
yar = np.array([4.79867,4.49013,4.2243,3.47313,2.66674,1.51909])

x = np.linspace(max(xar),min(xar),100)
y = []

def f(x,m):
    c = polyFit(xar,yar,m)
    p = 0
    for i in range(len(c)):
        p += c[i]*(x**i)
    return p

plt.figure(figsize=[15,10])
plt.xlim(-0.5,8.5)
plt.ylim(0,6)

plt.plot(xar,yar,'o')

y = []
for m in range(6):
    y.append(mapar(lambda x:f(x,m),x))
    plt.plot(x,y[m],'-',label = 'm = '+str(m))

plt.legend()
plt.show()
```



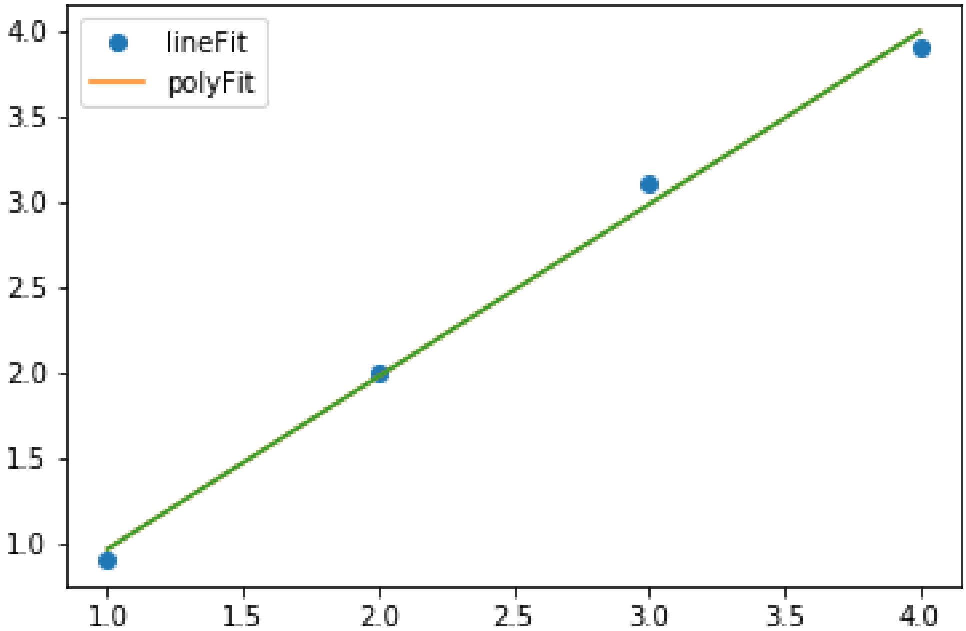
```python
## Fitting a Straight Line, m = 1
## xData, yData

# S(a,b)
def lineFit(xData,yData):
    n = len(xData)
    xbar, ybar = sum(xData)/n, sum(yData)/n

    # b=b0/b1
    b0, b1 = 0, 0
    for i in range(n):
        b0 += yData[i]*(xData[i]-xbar)
        b1 += xData[i]*(xData[i]-xbar)
    b = sum0/sum1
    a = ybar - xbar*b

    return a,b
```

```python
print(' lineFit:',lineFit(xar,yar),'\npolyFit',polyFit(xar,yar,1))
```

```
lineFit: (5.2897069095959335, -0.42864833499292493)

polyFit [ 5.28970691 -0.42864833]
```

almost same

```
xar = np.array([1,2,3,4])
yar = np.array([0.9,2,3.1,3.9])
x = np.linspace(max(xar),min(xar),100)
c = lineFit(xar,yar)
d = polyFit(xar,yar,1)
y = mapar(lambda x:c[0] + c[1]*x,xar)
y1 = mapar(lambda x:d[0] + d[1]*x,xar)

plt.plot(xar,yar,'o',x,y,'-',x,y1,'-')
plt.legend(['lineFit','polyFit'])
plt.show()
```

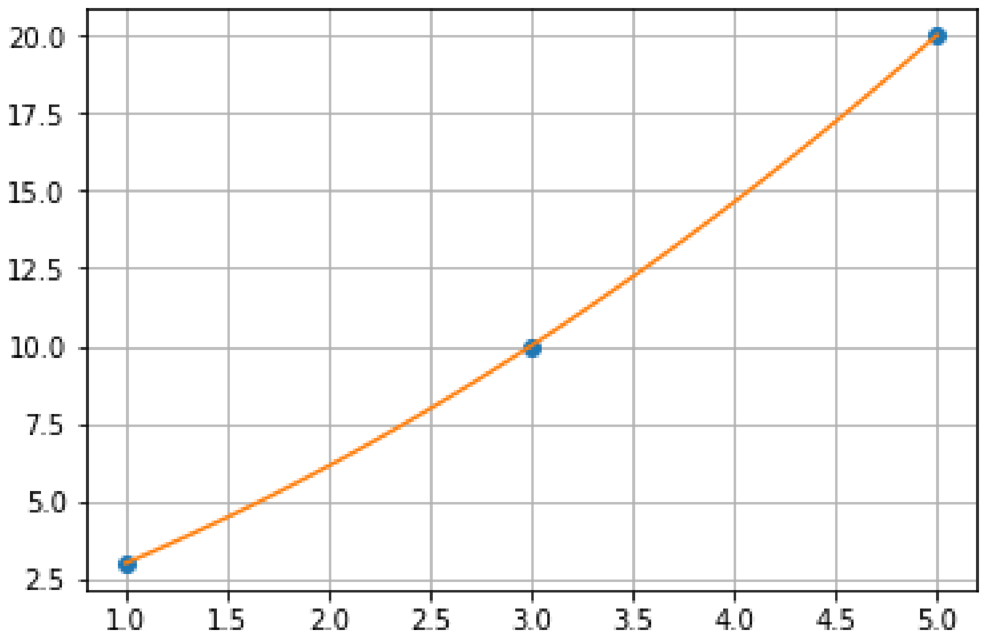


1. Use Lagrange’s method to determine y at x=2 in the following dataset

x = [1, 3, 5]
y = [3, 10, 20]

```
def lagrangePoly(x,xData,yDaya):
    n = len(xData)
    l = []
    for i in range(0,n):
        a = 1.
        for j in range(0,n):
            if (i != j):
                a = a * (x - xData[j])/(xData[i] - xData[j])
        l.append(a)

     p = 0
     for i in range(0,n):
         p += yData[i]*l[i]

     return p
```

```
xData = [1, 3, 5]
yData = [3, 10, 20]

x = np.linspace(min(xData),max(xData))

plt.plot(xData,yData,'o',x,lagrangePoly(x,xData,yData),'-')
plt.grid()
plt.show()
```



2. Consider the dataset

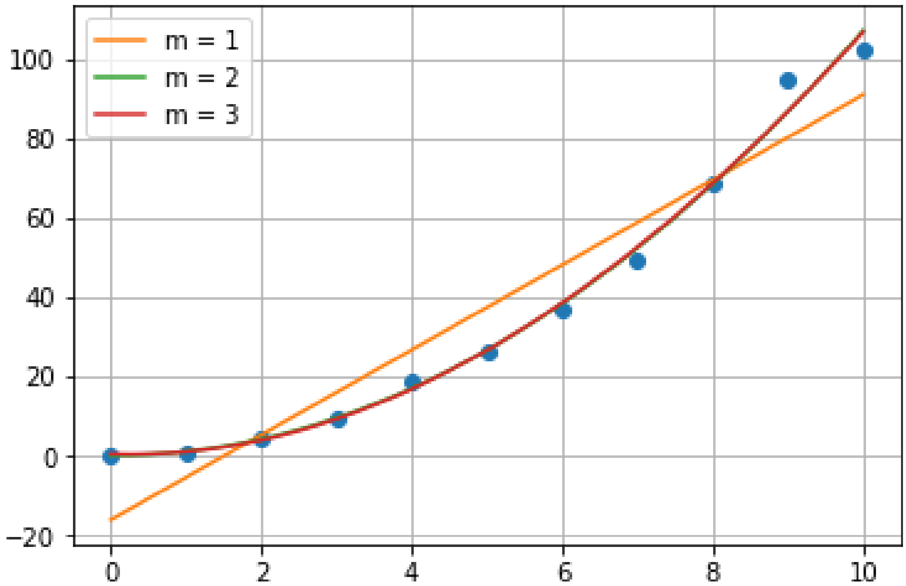
x = [0, 1, 2, …, 10]
y = ( x*(1+0.1*random) )$^2$

functions to make coefficients from piazza

```
def swapRows(v,i,j):
    if len(v.shape) == 1:
        v[i],v[j] = v[j],v[i]
    else:
        v[[i,j],:] = v[[j,i],:]

def gaussPivot(a,b,tol=1.0e-12):
    n = len(b)
  # Set up scale factors
    s = np.zeros(n)
    for i in range(n):
        s[i] = max(np.abs(a[i,:]))
    for k in range(0,n-1):
      # Row interchange, if needed
        p = np.argmax(np.abs(a[k:n,k])/s[k:n]) + k
        if abs(a[p,k]) < tol: error.err('Matrix is singular')
        if p != k:
            swapRows(b,k,p)
            swapRows(s,k,p)
            swapRows(a,k,p)
      # Elimination
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a[i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
    if abs(a[n-1,n-1]) < tol: error.err('Matrix is singular')
  # Back substitution
    b[n-1] = b[n-1]/a[n-1,n-1]
    for k in range(n-2,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b

def polyFit(xData,yData,m):
    a = np.zeros((m+1,m+1))
    b = np.zeros(m+1)
    s = np.zeros(2*m+1)
    for i in range(len(xData)):
        temp = yData[i]
        for j in range(m+1):
            b[j] = b[j] + temp
            temp = temp*xData[i]
        temp = 1.0
        for j in range(2*m+1):
            s[j] = s[j] + temp
            temp = temp*xData[i]


    for i in range(m+1):
        for j in range(m+1):
            a[i,j] = s[i+j]
    return gaussPivot(a,b)
```

main code

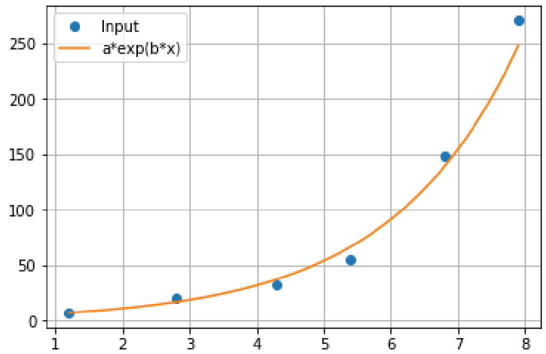
```
r = np.random.random

xData = np.linspace(0,10,11)
yData = np.zeros(11)
for i in range(11):
    yData[i] = (i*(1+0.1*r()))**2

x = np.linspace(0,10,201)

def f(x,m):
    c = polyFit(xData,yData,m)
    p = 0
    for i in range(len(c)):
        p += c[i]*(x**i)
    return p

def mapar(f,ar):
    return np.array(list(map(f,ar)))

plt.plot(xData,yData,'o')

for m in range(1,4):
    y.append(mapar(lambda x:f(x,m),x))
    plt.plot(x,mapar(lambda x:f(x,m),x),'-',label = 'm = '+str(m))

plt.legend()
plt.grid()
plt.show()
```



3. Find the fit to the following dataset with the log of an exponential form
$\ln(ae^{bx}) = \ln a + bx$
x = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]
ln(y) = [ 2, 3, 3.5, 4, 5, 5.6]
Compare the resulting fitting coefficients and standard deviation when
a) The weights $W_i = 1$

```
def logFit(xData,yData):
    n = len(xData)
    xbar, ybar = sum(xData)/n, sum(yData)/n

    # b=b0/b1
    b0, b1 = 0, 0
    for i in range(n):
        b0 += yData[i]*(xData[i]-xbar)
        b1 += xData[i]*(xData[i]-xbar)
    b = b0/b1
    lna = ybar - xbar*b
    a = exp(lna)

    # standard deviation
    sigma = 0
    for i in range(n):
        sigma += (yData[i] - lna - b*xData[i])**2
    sigma = sqrt(sigma/(n-2))

    return sigma,a,b
```

```
xData = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]
lnyData = [ 2, 3, 3.5, 4, 5, 5.6]

x = np.linspace(min(xData),max(xData),100)

sigma,a, b = logFit(xData,lnyData)
y = mapar(lambda x:a*exp(b*x),x)

yData = mapar(lambda y:exp(y),lnyData)
plt.plot(xData,yData,'o',x,y,'-')
plt.legend(['Input','a*exp(b*x)'])
plt.show()
plt.grid()
print('coefficients: ln(a) =',np.log(a),' a =',a,' b =',b)
print('standard deviation =',sigma)
```

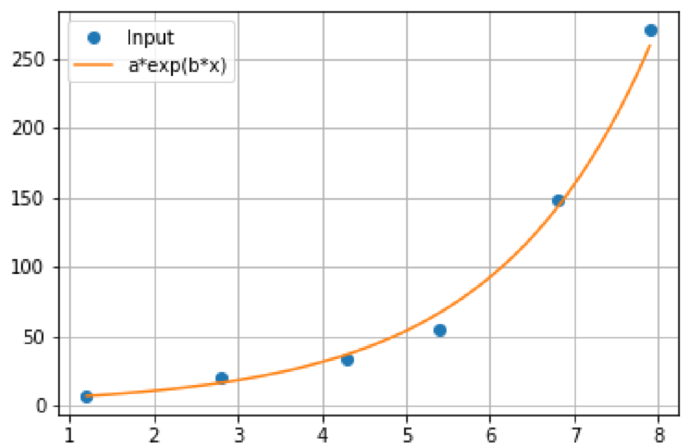



coefficients:
ln(a) = 1.3658356516156647
a = 3.9189965998699043
b = 0.5248234538840145
standard deviation = 0.1534083838943964

b) The weights $W_i = y_i$

```python
def logFit(xData,yData,weight=np.ones(len(xData))):
    n = len(xData)
    xhat, yhat = 0, 0
    for i in range(n):
        xhat += xData[i]*weight[i]**2
        yhat += yData[i]*weight[i]**2
    xhat /= np.dot(weight,weight); yhat /= np.dot(weight,weight)

    # b=b0/b1
    b0, b1 = 0, 0
    for i in range(n):
        b0 += yData[i]*(xData[i]-xhat)*weight[i]**2
        b1 += xData[i]*(xData[i]-xhat)*weight[i]**2
    b = b0/b1
    lna = yhat - xhat*b
    a = exp(lna)

    # standard deviation
    sigma = 0
    for i in range(n):
        sigma += (yData[i] - lna - b*xData[i])**2
    sigma = sqrt(sigma/(n-2))

    return sigma,a,b
```

```python
xData = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]
lnyData = [ 2, 3, 3.5, 4, 5, 5.6]

x = np.linspace(min(xData),max(xData),100)
yData = mapar(lambda y:exp(y),lnyData)

sigma,a, b = logFit(xData,lnyData,lnyData)
y = mapar(lambda x:a*exp(b*x),x)

plt.plot(xData,yData,'o',x,y,'-')
plt.legend(['Input','a*exp(b*x)'])
plt.grid()
plt.show()
print('coefficients: ln(a) =',np.log(a),' a =',a,' b =',b)
print('sigma =',sigma)
```




coefficients:

ln(a) = 1.27323946688785

a = 3.572406530568428

b = 0.5422627686511121

sigma = 0.16141345153011627