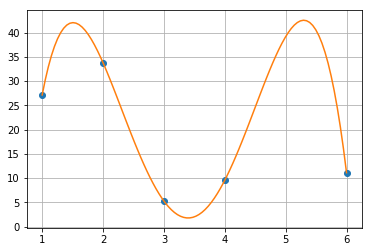
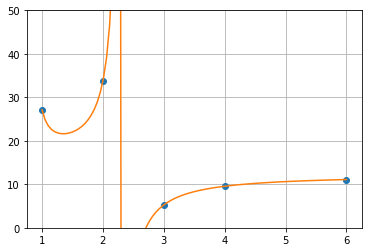
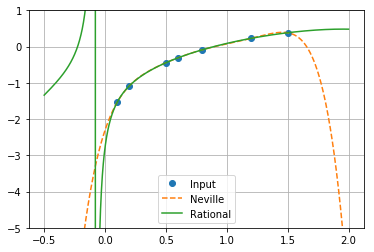
1. ## neville fitting
3. #xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
4. #yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])
6. xar = np.array([1,2,3,4,6])
7. yar = np.array([27.130,33.772,5.257,9.549,11.098])
9. #xx = np.linspace(1,4.5,100)
10. xlen = (len(xx))
11. xx = np.linspace(1,6,100)
12. yy = np.zeros(xlen)
14. ncount = 0
15. #for x in xx:
16. # yy[ncount] = neville(xar,yar,x)
17. # ncount += 1
19. yy = mapar(lambda x:neville(xar,yar,x),xx)
21. plt.plot(xar,yar,'o',xx,yy,'-')
22. plt.grid()
23. plt.show()



1. ## rational fitting
3. #xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
4. #yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])
6. xar = np.array([1,2,3,4,6])
7. yar = np.array([27.130,33.772,5.257,9.549,11.098])
9. #xx = np.linspace(1,4.5,100)
10. xx = np.linspace(1,6,100)
11. xlen = (len(xx))
12. yy = np.zeros(xlen)
14. ncount = 0
15. #for x in xx:
16. # yy[ncount] = rational(xar,yar,x)
17. # ncount += 1
19. yy = mapar(lambda x:rational(xar,yar,x),xx)
21. plt.plot(xar,yar,'o',xx,yy,'-')
22. plt.ylim(0,50)
23. plt.grid()
24. plt.show()

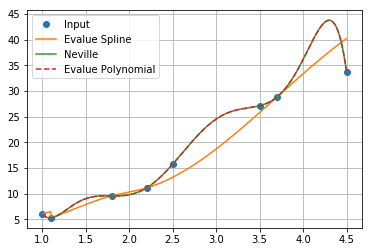


1. xData = np.array([0.1,0.2,0.5,0.6,0.8,1.2,1.5])
2. yData = np.array([-1.5342,-1.0811,-0.4445,-0.3085, -0.0868,0.2281,0.3824])
4. x = np.linspace(-0.5,2,1000)
6. nev = mapar(lambda x:neville(xData,yData,x),x)
7. rat = mapar(lambda x:rational(xData,yData,x),x)
9. plt.ylim(-5,1)
10. plt.plot(xData,yData,'o',x,nev,'--',x,rat,'-')
11. plt.legend(['Input','Neville','Rational'])
12. plt.grid()
13. plt.show()



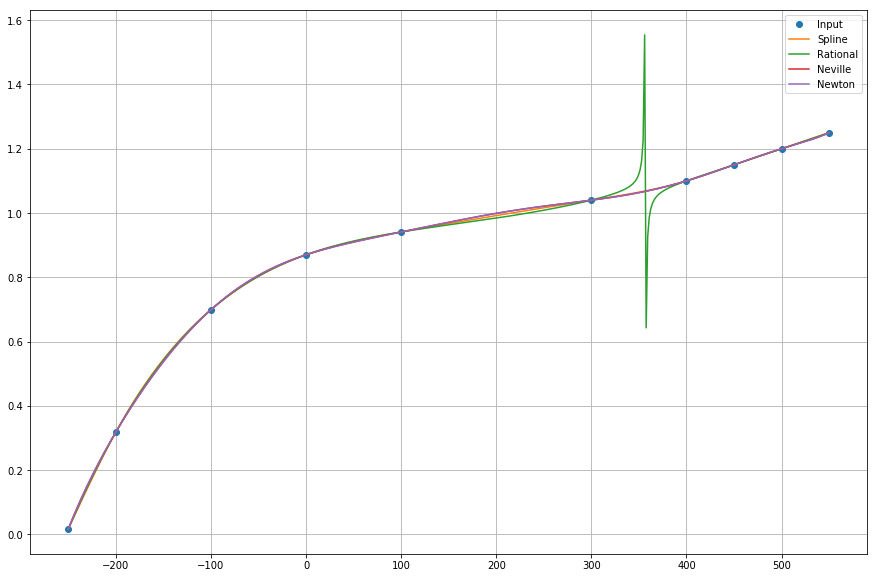
Rational fitting is more naturaler than Polynomial fitting but It can divergence.

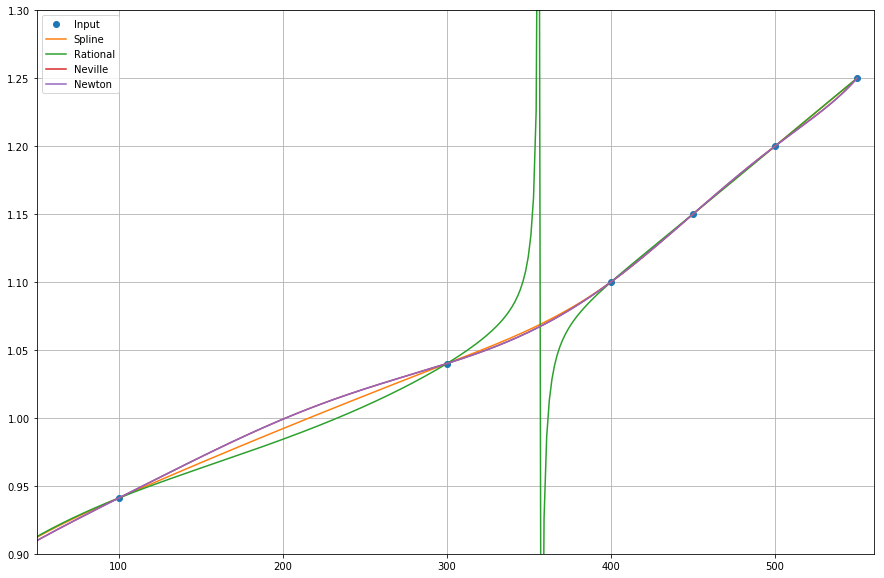
1. xar = np.array([1, 2.5, 3.5, 4.5, 1.1, 1.8, 2.2, 3.7])
2. #xar = np.array([1,2,3,4,5,6,7,8])
3. yar = np.array([6.008,15.722,27.130,33.772,5.257,9.549,11.098,28.828])
5. x = np.linspace(1,4.5,500)
7. a = coeffts(xar,yar)
8. k = curvatures(xar,yar)
10. pol = mapar(lambda x:evalPoly(a,xar,x),x)
11. nev = mapar(lambda x:neville(xar,yar,x),x)
12. spl = mapar(lambda x:evalSpline(xar,yar,k,x),x)
14. #plt.ylim(0,50)
15. plt.plot(xar,yar,'o',x,spl,'-',x,nev,'-',x,pol,'--')
16. plt.legend(['Input','Evalue Spline','Neville','Evalue Polynomial'])
17. plt.grid()
18. plt.show()



Spline fitting is more stable than Polynomial fitting.

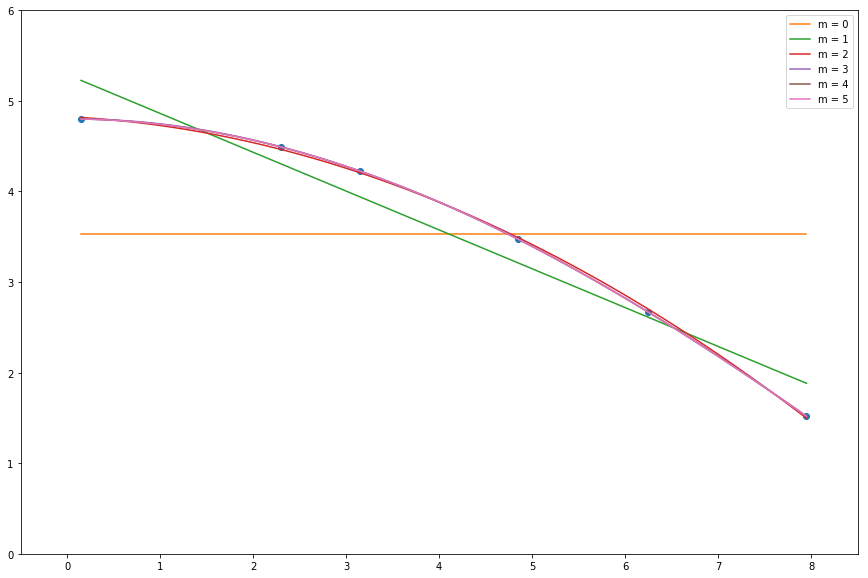
1. ## Compare fitting done with Newton, Neville, rational and splines
2. ## What is the optimal number of points for fitting the above data with a polynomial?
4. xData = np.array([-250,-200,-100,0,100,300,400,450,500,550])
5. yData = np.array([0.0163,0.318,0.699,0.870,0.941,1.04,1.1,1.15,1.2,1.25])
7. x = np.linspace(min(list(xData)),max(list(xData)),500)
9. a = coeffts(xData,yData)
10. k = curvatures(xData,yData)
12. pol = mapar(lambda x:evalPoly(a,xData,x),x)
13. nev = mapar(lambda x:neville(xData,yData,x),x)
14. rat = mapar(lambda x:rational(xData,yData,x),x)
15. spl = mapar(lambda x:evalSpline(xData,yData,k,x),x)
17. plt.figure(figsize=[15,10])
19. plt.plot(xData,yData,'o',x,spl,'-',x,rat,'-',x,nev,'-',x,pol,'-')
20. plt.legend(['Input','Spline','Rational','Neville','Newton'])
21. plt.grid()
22. plt.show()





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

1. ## Fitting Linear Forms
2. ## Use polyFit(xData,yData,m)
3. ## x 0.15 2.30 3.15 4.85 6.25 7.95
4. ## y 4.79867 4.49013 4.2243 3.47313 2.66674 1.51909
6. xar = np.array([0.15,2.30,3.15,4.85,6.25,7.95])
7. yar = np.array([4.79867,4.49013,4.2243,3.47313,2.66674,1.51909])
9. x = np.linspace(max(xar),min(xar),100)
10. y = []
12. def f(x,m):
13. c = polyFit(xar,yar,m)
14. p = 0
15. for i in range(len(c)):
16. p += c[i]\*(x\*\*i)
17. return p
19. plt.figure(figsize=[15,10])
20. plt.xlim(-0.5,8.5)
21. plt.ylim(0,6)
23. plt.plot(xar,yar,'o')
25. y = []
26. for m in range(6):
27. y.append(mapar(lambda x:f(x,m),x))
28. plt.plot(x,y[m],'-',label = 'm = '+str(m))
30. plt.legend()
31. plt.show()



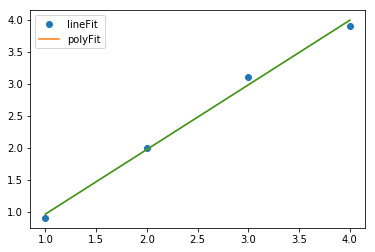
1. ## Fitting a Straight Line, m = 1
2. ## xData, yData
4. # S(a,b)
5. def lineFit(xData,yData):
6. n = len(xData)
7. xbar, ybar = sum(xData)/n, sum(yData)/n
9. # b=b0/b1
10. b0, b1 = 0, 0
11. for i in range(n):
12. b0 += yData[i]\*(xData[i]-xbar)
13. b1 += xData[i]\*(xData[i]-xbar)
14. b = sum0/sum1
15. a = ybar - xbar\*b
17. return a,b
18. print(' lineFit:',lineFit(xar,yar),'\npolyFit',polyFit(xar,yar,1))

 lineFit: (5.2897069095959335, -0.42864833499292493)

polyFit [ 5.28970691 -0.42864833]

almost same

1. xar = np.array([1,2,3,4])
2. yar = np.array([0.9,2,3.1,3.9])
3. x = np.linspace(max(xar),min(xar),100)
4. c = lineFit(xar,yar)
5. d = polyFit(xar,yar,1)
6. y = mapar(lambda x:c[0] + c[1]\*x,xar)
7. y1 = mapar(lambda x:d[0] + d[1]\*x,xar)
9. plt.plot(xar,yar,'o',x,y,'-',x,y1,'-')
10. plt.legend(['lineFit','polyFit'])
11. plt.show()

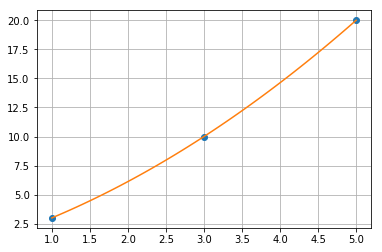


1. Use Lagrange’s method to determine y at x=2 in the following dataset

x = [1, 3, 5]

y = [3, 10, 20]

1. def lagrangePoly(x,xData,yDaya):
2. n = len(xData)
3. l = []
4. for i in range(0,n):
5. a = 1.
6. for j in range(0,n):
7. if (i != j):
8. a = a \* (x - xData[j])/(xData[i] - xData[j])
9. l.append(a)
11. p = 0
12. for i in range(0,n):
13. p += yData[i]\*l[i]
15. return p
16. xData = [1, 3, 5]
17. yData = [3, 10, 20]
18. x = np.linspace(min(xData),max(xData))
19. plt.plot(xData,yData,'o',x,lagrangePoly(x,xData,yData),'-')
20. plt.grid()
21. plt.show()



2. Consider the dataset

x = [0, 1, 2, …, 10]

y = ( x\*(1+0.1\*random) )2

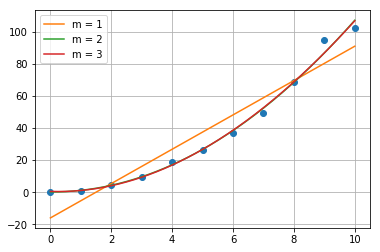
functions to make coefficients from piazza

1. def swapRows(v,i,j):
2. if len(v.shape) == 1:
3. v[i],v[j] = v[j],v[i]
4. else:
5. v[[i,j],:] = v[[j,i],:]
7. def gaussPivot(a,b,tol=1.0e-12):
8. n = len(b)
9. # Set up scale factors
10. s = np.zeros(n)
11. for i in range(n):
12. s[i] = max(np.abs(a[i,:]))
13. for k in range(0,n-1):
14. # Row interchange, if needed
15. p = np.argmax(np.abs(a[k:n,k])/s[k:n]) + k
16. if abs(a[p,k]) < tol: error.err('Matrix is singular')
17. if p != k:
18. swapRows(b,k,p)
19. swapRows(s,k,p)
20. swapRows(a,k,p)
21. # Elimination
22. for i in range(k+1,n):
23. if a[i,k] != 0.0:
24. lam = a[i,k]/a[k,k]
25. a[i,k+1:n] = a[i,k+1:n] - lam\*a[k,k+1:n]
26. b[i] = b[i] - lam\*b[k]
27. if abs(a[n-1,n-1]) < tol: error.err('Matrix is singular')
28. # Back substitution
29. b[n-1] = b[n-1]/a[n-1,n-1]
30. for k in range(n-2,-1,-1):
31. b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
32. return b
34. def polyFit(xData,yData,m):
35. a = np.zeros((m+1,m+1))
36. b = np.zeros(m+1)
37. s = np.zeros(2\*m+1)
38. for i in range(len(xData)):
39. temp = yData[i]
40. for j in range(m+1):
41. b[j] = b[j] + temp
42. temp = temp\*xData[i]
43. temp = 1.0
44. for j in range(2\*m+1):
45. s[j] = s[j] + temp
46. temp = temp\*xData[i]

49. for i in range(m+1):
50. for j in range(m+1):
51. a[i,j] = s[i+j]
52. return gaussPivot(a,b)

 main code

1. r = np.random.random
3. xData = np.linspace(0,10,11)
4. yData = np.zeros(11)
5. for i in range(11):
6. yData[i] = (i\*(1+0.1\*r()))\*\*2
8. x = np.linspace(0,10,201)
10. def f(x,m):
11. c = polyFit(xData,yData,m)
12. p = 0
13. for i in range(len(c)):
14. p += c[i]\*(x\*\*i)
15. return p
17. def mapar(f,ar):
18. return np.array(list(map(f,ar)))
20. plt.plot(xData,yData,'o')
22. for m in range(1,4):
23. y.append(mapar(lambda x:f(x,m),x))
24. plt.plot(x,mapar(lambda x:f(x,m),x),'-',label = 'm = '+str(m))
26. plt.legend()
27. plt.grid()
28. plt.show()



3. Find the fit to the following dataset with the log of an exponential form



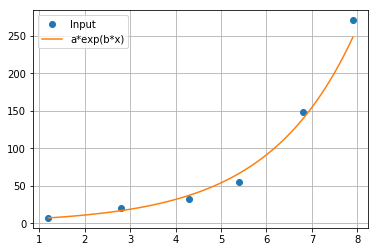
x = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]

ln(y) = [ 2, 3, 3.5, 4, 5, 5.6]

Compare the resulting fitting coefficients and standard deviation when

a) The weights Wi = 1

1. def logFit(xData,yData):
2. n = len(xData)
3. xbar, ybar = sum(xData)/n, sum(yData)/n
5. # b=b0/b1
6. b0, b1 = 0, 0
7. for i in range(n):
8. b0 += yData[i]\*(xData[i]-xbar)
9. b1 += xData[i]\*(xData[i]-xbar)
10. b = b0/b1
11. lna = ybar - xbar\*b
12. a = exp(lna)
14. # standard deviation
15. sigma = 0
16. for i in range(n):
17. sigma += (yData[i] - lna - b\*xData[i])\*\*2
18. sigma = sqrt(sigma/(n-2))
20. return sigma,a,b
21. xData = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]
22. lnyData = [ 2, 3, 3.5, 4, 5, 5.6]
24. x = np.linspace(min(xData),max(xData),100)
26. sigma,a, b = logFit(xData,lnyData)
27. y = mapar(lambda x:a\*exp(b\*x),x)
29. yData = mapar(lambda y:exp(y),lnyData)
30. plt.plot(xData,yData,'o',x,y,'-')
31. plt.legend(['Input','a\*exp(b\*x)'])
32. plt.show()
33. plt.grid()
34. print('coefficients: ln(a) =',np.log(a),' a =',a,' b =',b)
35. print('standard deviation =',sigma)



coefficients:

ln(a) = 1.3658356516156647

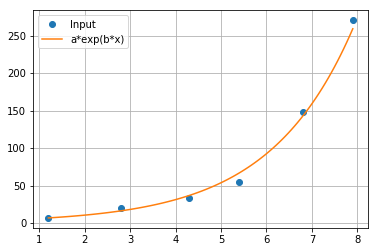
a = 3.9189965998699043

b = 0.5248234538840145

standard deviation = 0.1534083838943964

b) The weights Wi = yi

1. def logFit(xData,yData,weight=np.ones(len(xData))):
2. n = len(xData)
3. xhat, yhat = 0, 0
4. for i in range(n):
5. xhat += xData[i]\*weight[i]\*\*2
6. yhat += yData[i]\*weight[i]\*\*2
7. xhat /= np.dot(weight,weight); yhat /= np.dot(weight,weight)
9. # b=b0/b1
10. b0, b1 = 0, 0
11. for i in range(n):
12. b0 += yData[i]\*(xData[i]-xhat)\*weight[i]\*\*2
13. b1 += xData[i]\*(xData[i]-xhat)\*weight[i]\*\*2
14. b = b0/b1
15. lna = yhat - xhat\*b
16. a = exp(lna)
18. # standard deviation
19. sigma = 0
20. for i in range(n):
21. sigma += (yData[i] - lna - b\*xData[i])\*\*2
22. sigma = sqrt(sigma/(n-2))
24. return sigma,a,b
25. xData = [1.2, 2.8, 4.3, 5.4, 6.8, 7.9]
26. lnyData = [ 2, 3, 3.5, 4, 5, 5.6]
28. x = np.linspace(min(xData),max(xData),100)
29. yData = mapar(lambda y:exp(y),lnyData)
31. sigma,a, b = logFit(xData,lnyData,lnyData)
32. y = mapar(lambda x:a\*exp(b\*x),x)
34. plt.plot(xData,yData,'o',x,y,'-')
35. plt.legend(['Input','a\*exp(b\*x)'])
36. plt.grid()
37. plt.show()
38. print('coefficients: ln(a) =',np.log(a),' a =',a,' b =',b)
39. print('sigma =',sigma)



coefficients:

ln(a) = 1.27323946688785

a = 3.572406530568428

b = 0.5422627686511121

sigma = 0.16141345153011627