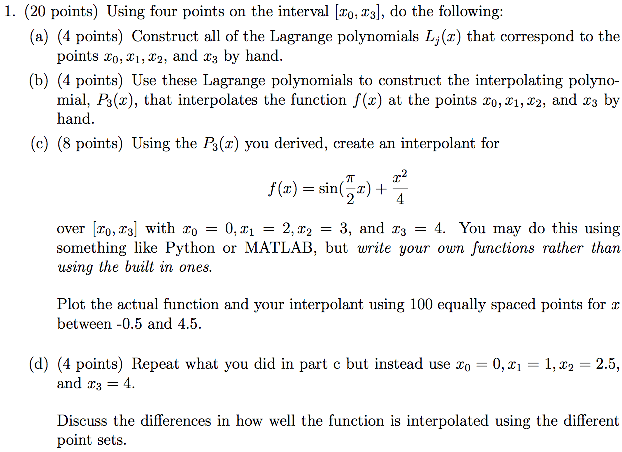
**NE 155: Assignment 2**



for for

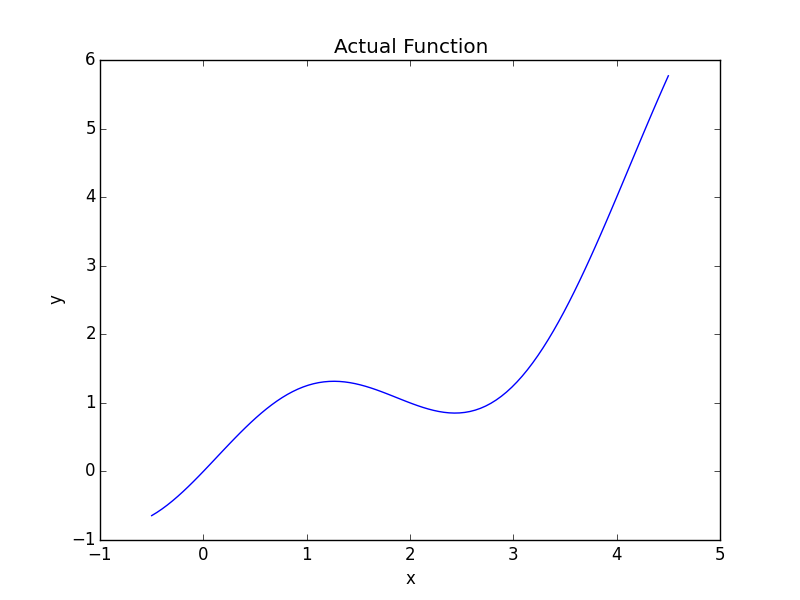
for for

1. The polynomial is given by:

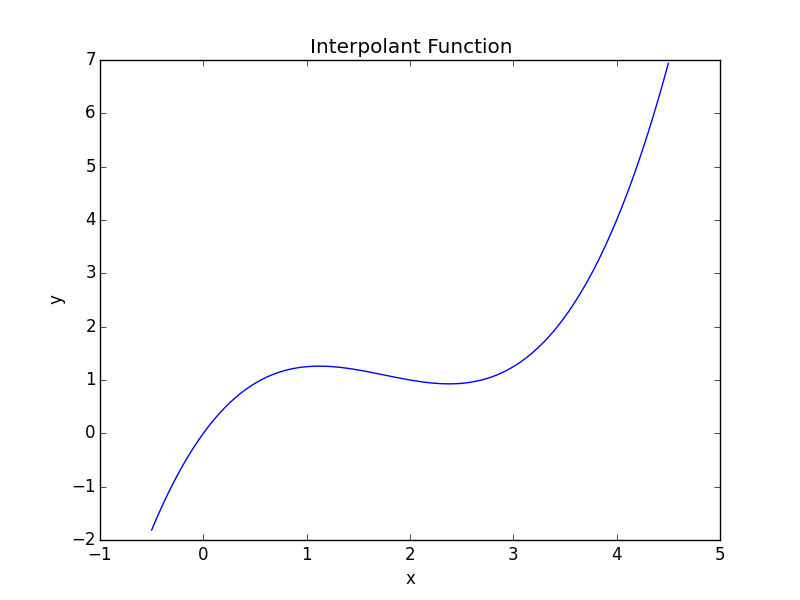
for

for

|  |
| --- |
| NE\_155\_q1  *Code to generate plot of functions* |
| import numpy as np  import matplotlib.pyplot as plt  x = np.linspace(-0.5, 4.5, 100)  def y\_actual(x):  return np.sin((np.pi / 2) \* x) + (x \*\* 2) / 4  y\_a = []  for i in x:  y\_a.append(y\_actual(i))  plt.xlabel('x')  plt.ylabel('y')  plt.title('Actual Function')  plt.plot(x, y\_a)  plt.show() |



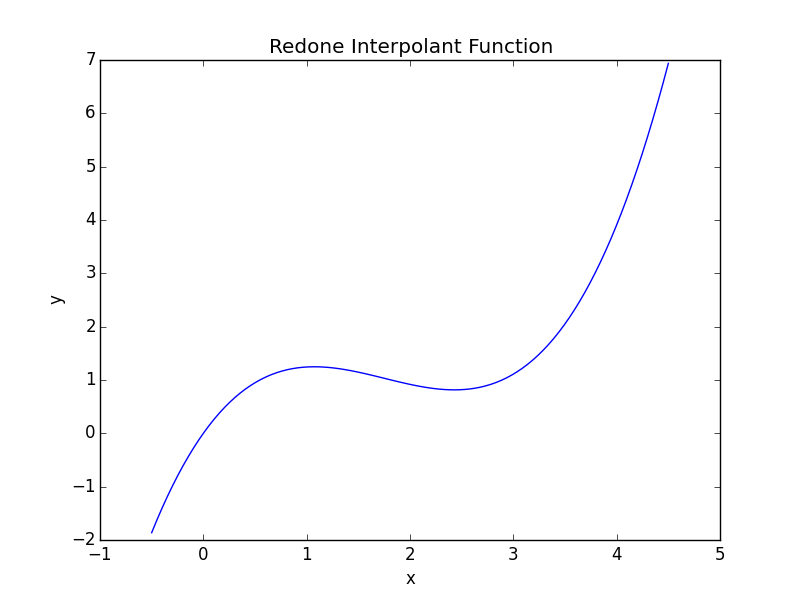
|  |
| --- |
| NE\_155\_q1  *Code to generate plot of functions* |
| import numpy as np  import matplotlib.pyplot as plt  ## Interpolant Function  def y\_inter(x):  '''  interpolant function  '''  return (x \*\* 3) / 3 - (7 / 4) \* x \*\* 2 + (8 / 3) \* x  y\_in = []  for i in x:  y\_in.append(y\_inter(i))  plt.xlabel('x')  plt.ylabel('y')  plt.title('Interpolant Function')  plt.plot(x, y\_in)  plt.show() |



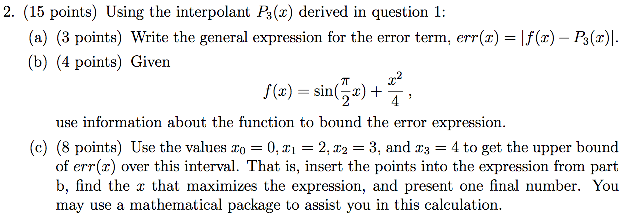
for

for

|  |
| --- |
| NE\_155\_q1  *Code to generate plot of functions* |
| ## Redone Interpolant  def y\_reinter(x):  '''  redone interpolant function  '''  return (0.348 \* x \*\* 3) - (1.829 \* x \*\* 2) + 2.725 \* x  y\_rein = []  for i in x:  y\_rein.append(y\_reinter(i))  plt.xlabel('x')  plt.ylabel('y')  plt.title('Redone Interpolant Function')  plt.plot(x, y\_rein)  plt.show() |



The function is interpolated slightly better, however, the difference between the previous interpolation and the new one is not notable.



total error function:

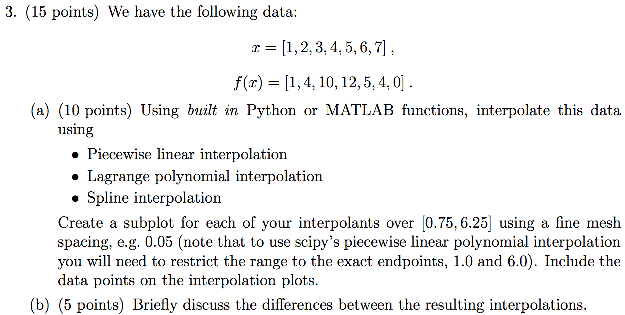
The fiv(x) function is at a maximum at x = 1 and is at a minimum at x = 3. At x = 1, the function fiv(x) is equal to . In terms of magnitude, the function is at 0 for the values 0, 2, and 4.

1. Error(x) =

The error becomes:

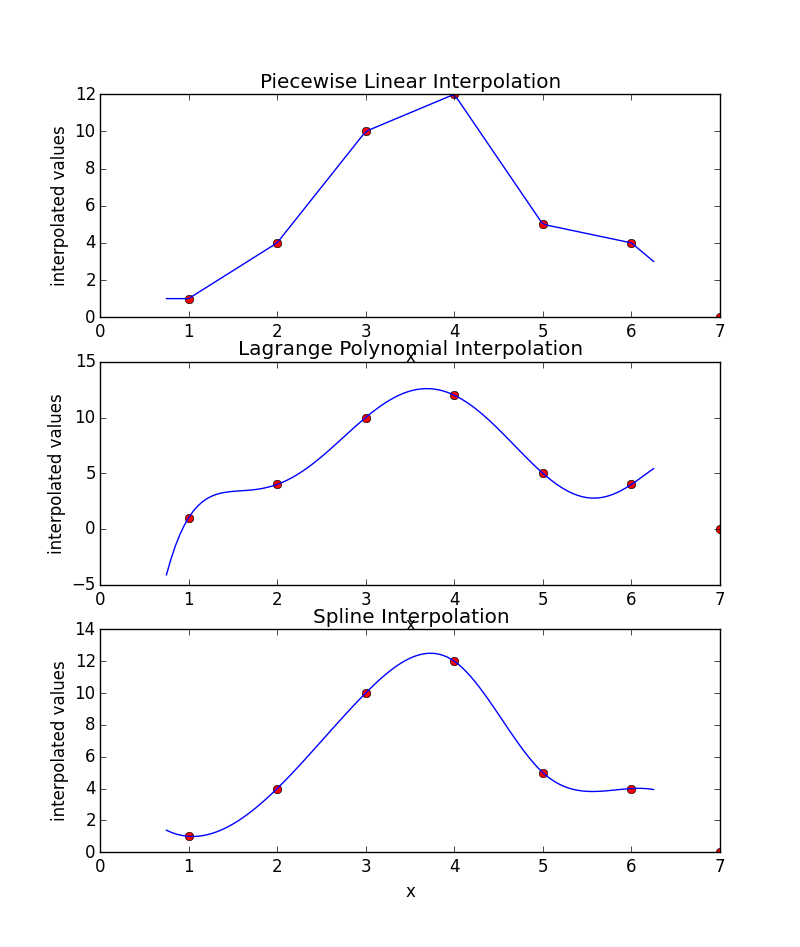
After optimizing this function utilizing Mathematica, the largest magnitude error is obtained at

x = 0.830691 where the error is |-1.63448| = 1.63448.

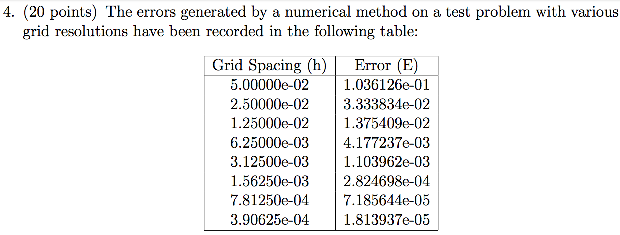


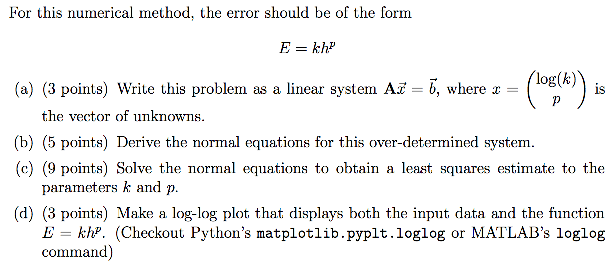
1. Using Python:

|  |
| --- |
| NE\_155\_q3  *Code used to generate three subplots: one for each interpolating method* |
| import numpy as np  import matplotlib.pyplot as plt  import scipy.interpolate  ## Interpolating using Python's built-in functions  x = np.linspace(0.75, 6.25, 110)  xp = [1, 2, 3, 4, 5, 6, 7]  fp = [1, 4, 10, 12, 5, 4, 0]  ## piecewise-linear-interpolation component  pli = np.interp(x, xp, fp)  plt.subplot(311)  plt.xlabel('x')  plt.ylabel('interpolated values')  plt.title('Piecewise Linear Interpolation')  plt.plot(xp, fp, 'ro')  plt.plot(x, pli)  ## Lagrange-Polynomial-interpolation component  lpi = scipy.interpolate.lagrange(xp, fp)  y\_lpi = []  for i in x:  y\_lpi.append(lpi(i))  plt.subplot(312)  plt.xlabel('x')  plt.ylabel('interpolated values')  plt.title('Lagrange Polynomial Interpolation')  plt.plot(xp, fp, 'ro')  plt.plot(x, y\_lpi)  ## Spline-interpolation component (cubic)  data\_points = scipy.interpolate.splrep(xp, fp)  si = scipy.interpolate.splev(x, data\_points)  plt.subplot(313)  plt.xlabel('x')  plt.ylabel('interpolated values')  plt.title('Spline Interpolation')  plt.plot(xp, fp, 'ro')  plt.plot(x, si)  plt.show() |



1. The piecewise linear interpolation simply connects the data points directly and linearly. The Lagrange interpolation creates a polynomial function that passes through all the data points. The spline interpolation behaves like a cubic function between each pair of data points so that the end result is a combination of connected cubic functions. The difference between the three different interpolation methods is the end behavior. This is due to the difference in equations between all of them. For the linear interpolation, the ends follow the linear trend and therefore do not curve. The Lagrange interpolation graph shows the first end curving downward and the last end curving upward because the end points follow the trend of the polynomial function. For the spline interpolation, the endpoints behave like cubic functions. Therefore, for the first end point, the graph curves upward because it is following the trend of the function from the first data point. The last endpoint curves downward because of the cubic function the last data point goes by.





In matrix format:

1. =
2. inv(ATA) \* (ATA) \* x = inv(ATA) \* AT \* b

|  |
| --- |
| NE\_155\_q4  *Code that solves for the values “k” and “P”* |
| import numpy as np  h = [5 \* 10 \*\* (-2), 2.5 \* 10 \*\* (-2), 1.25 \* 10 \*\* (-2), 6.25 \* 10 \*\* (-3),  3.125 \* 10 \*\* (-3), 1.5625 \* 10 \*\* (-3), 7.8125 \* 10 \*\* (-4),  3.90625 \* 10 \*\* (-4)]  E = [1.036126 \* 10 \*\* (-1), 3.333834 \* 10 \*\* (-2), 1.375409 \* 10 \*\* (-2),  4.177237 \* 10 \*\* (-3), 1.103962 \* 10 \*\* (-3), 2.824698 \* 10 \*\* (-4),  7.185644 \* 10 \*\* (-5), 1.813937 \* 10 \*\* (-5)]  log\_h = np.log(h)  log\_E = np.log(E)  A = np.zeros((8, 2))  for i in range(8):  A[i][0] = 1  A[i][1] = log\_h[i]  b = np.zeros((8,1))  for i in range(8):  b[i][0] = log\_E[i]  x\_A = np.dot((np.linalg.inv(np.dot(np.transpose(A), A))), np.transpose(A))  x = np.dot(x\_A, b)  p = x[1][0]  k = np.exp(x[0][0])  print("The value of p is {}".format(p))  print("The value of k is {}".format(k)) |
| Returns:  In [**35**]: run NE\_155\_q4  The value of p is 1.790291393986538  The value of k is 28.432770995208546 |

1. .

|  |
| --- |
| NE\_155\_q4  *Code generated the plot containing input values and actual function* |
| import numpy as np  import matplotlib.pyplot as plt  h = [5 \* 10 \*\* (-2), 2.5 \* 10 \*\* (-2), 1.25 \* 10 \*\* (-2), 6.25 \* 10 \*\* (-3),  3.125 \* 10 \*\* (-3), 1.5625 \* 10 \*\* (-3), 7.8125 \* 10 \*\* (-4),  3.90625 \* 10 \*\* (-4)]  E = [1.036126 \* 10 \*\* (-1), 3.333834 \* 10 \*\* (-2), 1.375409 \* 10 \*\* (-2),  4.177237 \* 10 \*\* (-3), 1.103962 \* 10 \*\* (-3), 2.824698 \* 10 \*\* (-4),  7.185644 \* 10 \*\* (-5), 1.813937 \* 10 \*\* (-5)]  log\_h = np.log(h)  log\_E = np.log(E)  A = np.zeros((8, 2))  for i in range(8):  A[i][0] = 1  A[i][1] = log\_h[i]  b = np.zeros((8,1))  for i in range(8):  b[i][0] = log\_E[i]  x\_A = np.dot((np.linalg.inv(np.dot(np.transpose(A), A))), np.transpose(A))  x = np.dot(x\_A, b)  p = x[1][0]  k = np.exp(x[0][0])  #print("The value of p is {}".format(p))  #print("The value of k is {}".format(k))  plt.loglog(h, E)  def function(k, p, h):  return k \* h \*\* p  axis = np.linspace(0, .05, 100)  values = []  for i in axis:  values.append(function(k, p, i))  plt.loglog(axis, values, "r")  plt.xlabel('log of h')  plt.ylabel('log of E')  plt.title('Graph of input data vs. function')  plt.legend(("input data", "function"))  plt.show() |

