NUR Assignment 1

Qing Zhou

October 10, 2020

Abstract

In this document an example solution template is given for the exercises in the course Numerical recipes for astrophysics.

1 Redshift distribution of galaxies

In this section we look at question 1 subquestion a, we want to add this extra explanation bla bla, and also note bla bla.

Below shows the script for the basic LU decomposition algorithm:

```python
#!/usr/bin/env python
# coding: utf-8
import numpy as np


##########################################################################
##########################################################################
#######################        Main        ###############################
##########################################################################
##########################################################################
def LU_decomposition(A,b):
    """
    Function that implements the LU decomposition
    without improviements.
    ----------------------------------------------
    Inputs:
    A: a square matrix of shape (m,n), where m == n
    b: array where A*x = b
    Output:
    x: array, solution to A*x = b
    """
    #For this algotirhm to work, it is preassumed that m = n
    m,n = A.shape
    #create L,U matrices of shape (n,n)
    L = np.identity(n)
    U = np.identity(n)
    for k in range(n):
        U[0][k] = A[0][k]
        for i in range(n):
            if  (i <= k)&(i>0):
                U[i][k] = A[i][k]-sum(L[i][:i]*U[:,k][:i])
            if i>k:
                L[i][k] = (A[i][k]-sum(L[i][:k]*U[:,k][:k]))/U[k][k]

    #now solve for Ly = b
    y = np.zeros(n)
    y[0] = b[0]/L[0][0]
    for i in range(1,n):
        y[i]  = (b[i]-np.sum(L[i][:i]*y[:i]))/L[i][i]
    x = np.zeros(n)
    x[-1]  = y[n-1]/U[n-1][n-1]
    for i in range(n-1)[::-1]:
        x[i] = (y[i]-np.sum(U[i][i+1:n]*x[i+1:n]))/U[i][i]
```

```
44
45      return np.float32(x),np.float32(L),np.float32(U)
```

problem2.py

(a) The script for implementing the basic LU decomposition is:

```
#!/usr/bin/env python
# coding: utf-8
import numpy as np
from problem2 import LU_decomposition

##############################################################################
##############################################################################
#######################  subquestion 1  ######################################
##############################################################################
##############################################################################

wss = np.loadtxt('./wss.dat',dtype=np.float32)
wgs = np.loadtxt('./wgs.dat',dtype=np.float32)
x1,L1,U1 = LU_decomposition(wss,wgs)
print("Implementing basic LU decomposition...")
print("L matrix is: \n",L1)
print("U matrix is: \n",U1)
print("f is found to be: \n",x1)
print("The sum of f is:  ",x1.sum())
```

problem2a.py

Giving output:

```
Implementing basic LU decomposition...
L matrix is:
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 4.84681368e-01  1.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 3.46305460e-01  4.79143113e-01  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.96869457e-01  3.68807539e-02  8.63272488e-01  1.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.73147732e-01  3.54658850e-02 -2.05040108e-02  1.04066283e-02
    1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 6.26378879e-02  2.16549486e-02  2.87634999e-01  5.98211065e-02
    2.57926099e-02  1.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.31988809e-01 -6.13486990e-02  1.49788707e-01  4.94392291e-02
    1.54886991e-02  4.30113710e-02  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.31053910e-01 -6.95498884e-02 -5.71495369e-02  1.56852361e-02
    1.81494956e-03  2.09974945e-01  4.38529823e-05  1.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 7.50082880e-02  6.08870238e-02  1.20215043e-01  1.31490333e-02
   -7.62948301e-03  1.62050739e-01  1.49567509e-02  2.52222782e-03
    1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

```
 [ 3.08648031e-02  9.35540721e-03  7.20559135e-02  7.04647182e-03
   5.25621977e-03  5.00929216e-03  2.50519756e-02  3.60618494e-02
   9.10045728e-02  1.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.84808392e-02  3.52620371e-02  2.69351271e-03  1.67475753e-02
   1.85003150e-02  1.90348756e-02  8.52948800e-03  2.21738815e-02
   1.64001975e-02  5.16007245e-01  1.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.15461074e-01  3.73354577e-03  9.89140011e-03  4.37374134e-03
   1.11311264e-02  3.53991538e-02 -2.66360817e-03  6.07269257e-03
  -4.34605521e-04  2.09625401e-02  2.78662890e-02  1.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.51263643e-02  4.48409282e-03  1.95106398e-02  1.21246753e-02
   3.01830843e-03  3.32811624e-02  9.81919467e-03  1.61087401e-02
   1.92829520e-02  1.43403560e-01  1.97946057e-02  1.87682863e-02
   1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 5.96246272e-02  1.92432404e-02 -1.97117813e-02  7.22517306e-03
   4.66073351e-03  2.17840937e-03  1.03474883e-02  8.90145358e-03
   7.75486929e-03  1.00532994e-01  1.07453400e-02  1.67277865e-02
   1.85721308e-01  1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.75414512e-02  1.07614798e-02  7.10494351e-03  1.15235848e-02
   5.87824499e-03  4.46968526e-02 -3.59236123e-03  4.46045166e-03
  -1.08140102e-03  6.44806400e-02  6.82418933e-03  1.36786569e-02
   8.62050429e-02  3.90839912e-02  1.00000000e+00  0.00000000e+00]
 [ 2.84050666e-02  1.36084948e-02  1.12585388e-02  1.23763736e-02
   3.15039116e-03  4.75169457e-02 -3.20238993e-03 -1.81960943e-03
   4.47277399e-03  5.13768196e-02  1.60502945e-03  5.65985404e-03
   7.07940161e-02 -6.52331044e-04  4.11365293e-02  1.00000000e+00]]
U matrix is:
 [[ 3.8729522e-01  3.5511550e-01  2.4105136e-01  1.2011241e-01
   9.5767759e-02  5.5753987e-02  4.3184314e-02  4.8650619e-02
   4.6280783e-02  4.1748010e-02  1.5442266e-02  3.4991931e-02
   3.8863142e-05  3.5273787e-02  1.8746665e-02  1.4195884e-02]
 [ 0.0000000e+00  6.4229566e-01  5.2636076e-02  1.1845486e-01
   5.7947241e-02 -1.1274246e-02  2.0965276e-02  1.1997357e-02
   2.4274621e-02  1.1364750e-02  3.6911447e-02  1.1864158e-02
   8.9300219e-03 -7.1676387e-03  2.5727447e-02  1.4197157e-02]
 [ 0.0000000e+00  0.0000000e+00  4.5728818e-01 -6.9314577e-02
   1.4412680e-01  4.2997789e-02  5.2235853e-02  9.0307826e-03
   4.4341579e-02 -1.2927374e-02  1.3020856e-02  1.6807407e-02
   2.5719987e-02  1.8923694e-02 -7.8611113e-03  7.8932615e-03]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  3.0082359e+00
   1.4908837e-01  8.5588507e-02 -4.2440076e-03  9.9818502e-03
  -2.3786290e-02  4.3120045e-02 -1.2415510e-02 -9.0503171e-03
   1.9589730e-02  2.0542830e-02  2.2282749e-02  9.3729850e-03]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   3.6154723e+00  2.1857719e-01  1.3998836e-01 -7.9651410e-03
   8.6318970e-02 -1.0253605e-02  6.5092884e-02  2.8211998e-02
   3.4316048e-02  3.5014611e-02  3.4908112e-02 -2.2996257e-03]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  9.2191118e-01  4.6946958e-01  2.3898284e-01
   7.2151326e-02  1.1446174e-01  7.0232332e-02  1.6109798e-02
   9.9658519e-03  1.2704053e-02  2.9288661e-02  4.2145029e-02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  0.0000000e+00  5.4357710e+00  9.3825758e-02
   2.1317337e-01  1.4064805e-01  1.1458433e-01  5.1458035e-02
   5.4152068e-02  4.6974987e-02  4.1891176e-02  2.3090668e-02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  0.0000000e+00  0.0000000e+00  5.4507527e+00
   3.5153979e-01  3.2936286e-02  8.7390639e-02  9.4043255e-02
   6.9624089e-02  1.9796446e-02  4.6559718e-02  1.7583711e-02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   4.5448461e+00  3.0814981e-01  1.7478195e-01  5.7888508e-02
   1.8797452e-02  3.2354794e-02  1.4824679e-02  4.2381033e-02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
   0.0000000e+00  8.5001129e-01  4.8166107e-02  1.8473802e-02
   7.8590631e-02  1.1493106e-01  4.7482174e-02  4.2533785e-02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
```

```
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   1.0541049e+01   1.8095914e-01
   -3.5862379e-02   7.1174391e-02   3.2057595e-02   9.0812584e-03]
  [ 0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   1.1159947e+01
    3.9806420e-01   1.2652490e-01   9.4277970e-02   6.1478283e-02]
  [ 0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    1.6329607e+00   2.1451204e-01   1.8160620e-01   1.4701302e-03]
  [ 0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   7.5738993e+00   4.3910307e-01   1.9497925e-01]
  [ 0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   3.6933832e+00   2.7487665e-01]
  [ 0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   0.0000000e+00
    0.0000000e+00   0.0000000e+00   0.0000000e+00   3.9906437e+00]]
f is found to be:
 [0.00345958 0.03231731 0.08087177 0.10022925 0.13331331 0.12322342
 0.13155991 0.10453997 0.08667092 0.05877324 0.04246465 0.03845601
 0.01786364 0.02487733 0.01350275 0.00787694]
The sum of f is:    1.0
```

problem2a.txt

(b) The script for implementing the basic LU decomposition is:

```python
#!/usr/bin/env python
# coding: utf-8
import numpy as np
from problem2 import LU_decomposition

##########################################################################
##########################################################################
#######################  subquestion 2  ##################################
##########################################################################
##########################################################################
def iter_LU(A,b):
    x,L,U = LU_decomposition(A,b)
    deltab = np.array([sum(i) for i in A*x])-b
    dx = np.float32(LU_decomposition(A,deltab)[0])
    return x-dx

wss = np.loadtxt('./wss.dat',dtype=np.float32)
wgs = np.loadtxt('./wgs.dat',dtype=np.float32)
x = iter_LU(wss,wgs)
print("f is found to be: \n",x)
print("The sum of f is:  ",x.sum())
```

problem2b.py

Giving output:

```
f is found to be:
 [0.00345958 0.03231731 0.08087177 0.10022925 0.13331331 0.12322342
 0.13155992 0.10453996 0.08667092 0.05877324 0.04246465 0.03845601
 0.01786364 0.02487733 0.01350275 0.00787694]
The sum of f is:    1.0
```

problem2b.txt

2 Redshift distribution of galaxies

In this section we look at question 1 subquestion a, we want to add this extra explanation bla bla, and also note bla bla.

Below shows the script for problem 3:

(a) The script for implementing the basic LU decomposition is:

```
#!/usr/bin/env python
# coding: utf-8
import numpy as np
from math import *
import matplotlib.pyplot as plt


##########################################################################
##########################################################################
#######################  subquestion 1  ##################################
##########################################################################
##########################################################################
def trapezoid(f, a, b, n):
    """
    Composite trapezoidal rule

    """

    # Initialization
    h = (b - a) / n
    x = a

    # Composite rule
    T0 = f(a)
    for i in range(1, n):
        x  = x + h
        T0 += 2*f(x)

    return (T0 + f(b))*h/2

def romberg(f, a, b, row):
    """
    Romberg integration

    Ri,j = CTR(hi) for i = 0, where CTR is Composite Tranpezoidal Rule
    Ri,j = 4^{j-1}*Ri,j-1-Ri-1,j-1/(4^{j-1}-1) for i > 1

    """

    R = np.zeros((row, row))
    for i in range(row):
        R[i][0] = trapezoid(f, a, b, 2**i)

        for j in range(0, i):
            R[i][j+1] = (4**(j+1) * R[i][j] - R[i-1][j]) / (4**(j+1) - 1)


    return R[-1][-1]

a = 2.2
b = .5
c = 3.1
intn = lambda x: (x/b)**(a-3)*e**(-(x/b)**c)*x**2
integral = romberg(intn,1e-8,5,16)
A = 1/integral
print("A is solved to be: ",A)
```

problem3a.py

Giving output:

```
A is solved to be:  19.32986560821743
```

problem3a.txt

(b) The script for implementing the basic LU decomposition is:

```
#!/usr/bin/env python
# coding: utf-8
import numpy as np
from math import *
from problem3a import romberg
import matplotlib.pyplot as plt

##############################################################################
##############################################################################
#####################  subquestion 2  ########################################
##############################################################################
##############################################################################


def digitize(x,bins):
    index = []
    for i in range(len(x)):
        for k in range(len(bins)):
            if bins[k]>=x[i]:
                ind =  k
                break
        index.append(ind)
    return np.asarray(index)


def AkimaSpline(x, y):
    n = len(x)
    x_new = np.linspace(min(x),max(x),100)
    dx = np.array([x[i+1]-x[i] for i in range(len(x)-1)])
    dy = np.array([y[i+1]-y[i] for i in range(len(y)-1)])

    m = dy/dx
    m1 = 2.0 * m[0] - m[1]
    m2 = 2.0 * m1 - m[0]
    m3 = 2.0 * m[n - 2] - m[n - 3]
    m4 = 2.0 * m3 - m[n - 2]
    marr = np.concatenate(([m1], [m2], m, [m3], [m4]),axis=None)
    dm = np.abs(np.array([marr[i+1]-marr[i] for i in range(len(marr)-1)]))

    f1 = dm[2:n + 2]
    f2 = dm[0:n]
    f12 = f1 + f2
    a = y.copy()
    ids = np.arange(len(x))

    b = marr[1:n + 1]

    b[ids] = (f1[ids] * marr[ids + 1] + f2[ids] * marr[ids + 2]) / f12[ids]
    c = (3.0 * m - 2.0 * b[0:n - 1] - b[1:n]) / dx
    d = (b[0:n - 1] + b[1:n] - 2.0 * m) / dx ** 2


    bins = digitize(x_new, x[1:])

    index = bins

    xj = x_new - x[index]

    #plt.plot(x_new, index*100, 'r.')
    y_new = ((xj * d[index] + c[index]) * xj + b[index]) * xj + a[index]


    return x_new,y_new
```

```
a = 2.2
b = .5
c = 3.1
intn = lambda x: (x/b)**(a-3)*e**(-(x/b)**c)*x**2
integral = romberg(intn,1e-8,5,16)
A = 1/integral
Nsat = 100
log10n = lambda x: np.log10(A*Nsat)+(a-3)*np.log10(x/b)-(x/b)**c*np.log10(e)
x = np.array([1e-4,1e-2,1e-1,1,5])
log10x = np.log10(x)
log10y  = log10n(x)
data = np.c_[log10x,log10y]
xval,yval=AkimaSpline(log10x,log10y)
plt.figure(figsize=(5,5))
plt.plot([i[0] for i in data],[i[1] for i in data],'ob',label='data points')
plt.plot(xval,yval,label='Akima interpolation')
plt.xlabel(r'$\log_{10}x$')
plt.ylabel(r'$\log_{10}n(x)$')
plt.legend()
plt.savefig('./problem3.png')
plt.show()
```

problem3b.py

Giving output:

```
A is solved to be:  19.32986560821743
```

problem3b.txt

```
#!/usr/bin/env python
# coding: utf-8
import numpy as np
from math import *

##############################################################################
##############################################################################
#######################  subquestion 3  ######################################
##############################################################################
##############################################################################
def Stirlings(k):
    """
    Stirlings approximation can be written as:
    k!  ~ sqrt(2*pi*k)*(e/k)^(k)*(1+1/(12*k)+1/(288*k^2)+O(1/k^4))
    """
    if k == 0:
        return 1
    else:
        return np.sqrt(2*np.pi*k)*(k/e)**k*(1+1/12/k+1/288/k**2)

def Poisson(lamb,k):
    return lamb**k*np.exp(-lamb)/Stirlings(k)
arr1 = np.array([1,5,3,2.6])
arr2 = np.array([0,10,21,40])

for i,k in zip(arr1,arr2):
    print("lambda = ",i,",k = ",k,",P = ",'%.6e' % Poisson(i,k))
```

problem3c.py

Giving output:

```
lambda =  1.0 ,k =  0 ,P =  3.678794e-01
lambda =  5.0 ,k =  10 ,P =  1.813274e-02
lambda =  3.0 ,k =  21 ,P =  1.019340e-11
lambda =  2.6 ,k =  40 ,P =  3.615124e-33
```

problem3c.txt

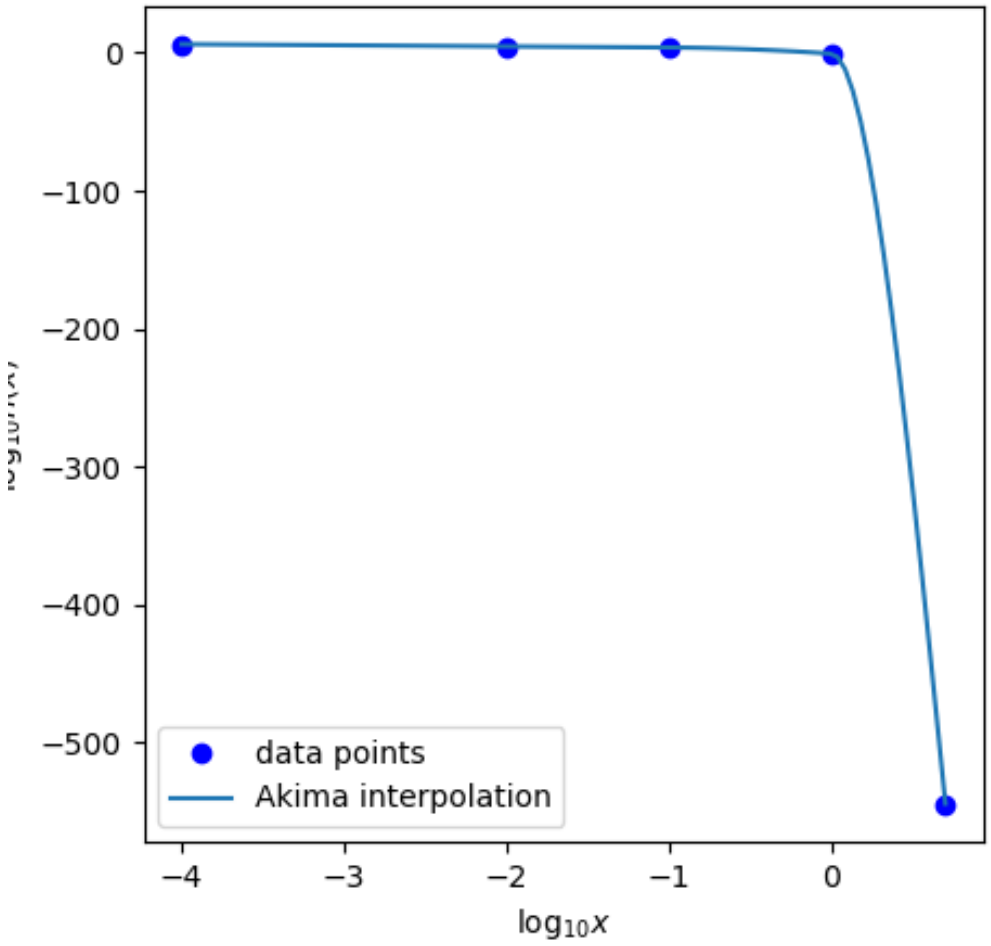

Figure 1: Log-log plot for n as a function of x. Here Akima sub-spline algorithm is used for interpolation.