NUR Assignment I: solutions

Tina Neumann

March 9, 2023

Abstract

This document displays my solution for the first assignment in the course numerical recipes for astrophysics, summer term 2023.

1 Exercise 1: Poisson distribution

In this exercise the poisson probability distribution for integer k should be calculated with the given values.

The poisson probability distribution is given as:

$$P_\lambda(k) = \frac{\lambda^k exp(-\lambda)}{k!} \tag{1}$$

While calculations a straight-forward derivation with 32-bit values would run into overflow-problems for the k-factorial, the given equation was converted into log-space which gives the following equation:

The result is calculated the following way:

$$log(P_\lambda(k)) = log(\lambda) * k - \lambda - \sum(k) \tag{2}$$

The $P_\lambda(k)$ for the given values are calculated for 32-bit and 64-bit values with into log-space converted equation and as comparison the same is derived using *scipy* via the given equation.

```
#!/usr/bin/env python
# coding: utf-8

# # Numerical recipies
# Assignment (09.03.23)
#
# Tina Neumann

# In[28]:


import numpy as np
import timeit


# In[40]:


### Exercise 1: Poisson distribution
# define given distribution
def poiss_32(lam, k):
    '''Function determines the poisson distribution P(k) of
    input: a positive mean (lam) and an integer (k)
    output: P(k)'''
    lam = np.float32(lam) #redefine as 32-bit integers
    k = np.int32(k)
    # to decrease the calculation time:
    # multiply by inverse instead of dividing
    # the values are considered to be in log-scale: log(products)—> sums
    # define k!
```

```python
    f_frac = 1.
    for f in range(1,k+1): #range leaves out last element
        f_frac += np.log(f)

    log_p = np.int32(k)*np.log(np.float32(lam)) - np.float32(lam) - np.int32(f_frac) # logarithmic P(k)
    p = np.exp(np.float32(log_p)) #revert log-scale
    print('k! = ', f_frac)
    print('log(P(k)) = ', log_p)
    return p


# In[41]:


def poiss_64(lam, k):
    '''Function determines the poisson distribution P(k) of
    input: a positive mean (lam) and an integer (k)
    output: P(k)'''
    lam = np.float64(lam) #redefine as 64-bit integers
    k = np.int64(k)
    # define k!
    f_frac = 1.
    for f in range(1,k+1): #range leaves out last element
        f_frac += np.log(f)

    log_p = k*np.log(lam) - lam - f_frac #logarithmic P(k)
    p = np.exp(log_p) #revert log-scale
    print('k! = ', f_frac)
    print('log(P(k)) = ', log_p)
    return p


# In[42]:


from scipy.special import factorial
def poiss(lam, k):
    '''Function determines the poisson distribution P(k) of
    input: a positive mean (lam) and an integer (k)
    output: P(k)'''
    lam = float(lam)
    k = int(k)
    #with numpy functions
    # define k!
    f_frac = factorial(k)
    p = lam**k*np.exp(-lam)*f_frac**(-1)

    return p


# In[43]:
# Save a text file
with open("1_PoissonDistribution.txt", "w") as file:
    file.write('# The given lambda, k & results for the Poisson-distribution P(k) evaluated with 32-bit data types \n')
# read-in given values
with open('input_1a.txt') as f:
    lines = f.readlines()[2:]
    for line in lines:
            #define lambda, k
            mean_lam, k_int = line.split('\t')
            print(mean_lam,k_int)
            # problem with rounding, calculate significant digit
            dig = 1
            res = poiss_32(mean_lam,k_int)
            while res*(dig*1e1)**(-1) >= 1.:
                dig += 1

            k_int = k_int[:-1] #str has additional '\n'
```

```
            print('The given values are lam & k: '+ mean_lam, ' & '+ k_int)

            # add values to output file
            with open("1_PoissonDistribution.txt", "a+") as file:
                file.write(str(mean_lam+'\t'+k_int+'\t'+str(round(poiss_32(mean_lam,
    k_int),dig+5))+'\n'))
            print('For 32 bits this results in a poisson distribution of P(k) = ', round
    (poiss_32(mean_lam,k_int),dig+5)) #print 6 relevant digits
            print('For 64 bits this results in a poisson distribution of P(k) = ', round
    (poiss_64(mean_lam,k_int),dig+5)) #print 6 relevant digits
            print('For numpy-functions this results in a poisson distribution of P(k) =
    ', round(poiss(mean_lam,k_int),dig+5)) #print 6 relevant digits
```

1_PoissonDistribution.py

The result of $P_\lambda(k)$ are given in:

```
# The given lambda, k & results for the Poisson-distribution P(k) evaluated with 32-bit
    data types
1    0    0.135335
5    10   0.007405
3    21   0.0
2.6  40   0.0
101  200  0.0
```

1_PoissonDistribution.txt

2 Exercise 2: Vandemonde matrix

The Vandermonde-matrix can be obtained to find an unique solution for a given Langrangian polynomial . The coefficients of the Langrangian polynomial are calculated as:

$$y_i = \sum j = 0N - 1 g_j x_i^j \tag{3}$$

2.1 2.a) Approximation with LU-decomposition

The entries of the Vandermonde-matrix ($V_{ij} = x_i^j$) are given and can read in by the following script:

```
#This script is to get you started with reading the data and plotting it
#You are free to change whatever you like/do it completely differently

import numpy as np
import sys
import os
import matplotlib.pyplot as plt

data=np.genfromtxt(os.path.join(sys.path[0],"Vandermonde.txt"),comments='#',dtype=np.
    float64)
x=data[:,0]
y=data[:,1]
xx=np.linspace(x[0],x[-1],1001) #x values to interpolate at

fig,ax=plt.subplots()
ax.plot(x,y,marker='o',linewidth=0)
plt.xlim(-1,101)
plt.ylim(-400,400)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.show()
```

vandermonde.py

With the code the following result for **c** is obtained.
The distribution of the data points is vizualized in the plot above.

2.2 2.b) Approximation with Nevilles algorithm

The result of $P_\lambda(k)$ are given in:

2.3 2.d) Time previous sub-exercises

Within *timeit* the number of repititions considered to determine the runtime can be adjustested via the *repeat*-option. The default is set to 1e7. In the script below the number of repetitions are choosen as xxx so that code runs completely within less than 1min.

```
## Numerical recipies
# Assignment I, exercise 2 (09.03.23)
#
# Tina Neumann

# In[1]:
import timeit
import os
import matplotlib.pyplot as plt
import numpy as np
import sys

## Exercise 2.d) time routines with appropiate 'parameters`
repeat = int(5e3) #default 1e7: choose so that code runs below <1min

## Exercise 2.a) solve matrix via LU-decomposition
# create vandemonde-matrix as polynomial
print('The following time is needed to apply LU-decomposition and solve with the vandemonde-matrix the equation for c takes [sec]:')
print(timeit.timeit('lmat, umat, a_matrix = LU_decomp(vdM);xsol, ysol = solve_LU(y,lmat,umat,len(y))', setup = 'from readin_data import x,y,vdM,LU_decomp,solve_LU',number = repeat))


## Exercise 2.b) interpolation with nevilles algorithm
print('The time required to interpolate the matrix with Nevilles algorithm is [sec]:')
print(timeit.timeit('y_nev=[];err_nev=[];[nevilles_algo(a,x,y) for a in xx]', setup = 'from readin_data import nevilles_algo,xx,x,y',number = repeat))

## Exercise 2.c) solve matrix via 10xLU-decomposition
```

2d_timeit.py