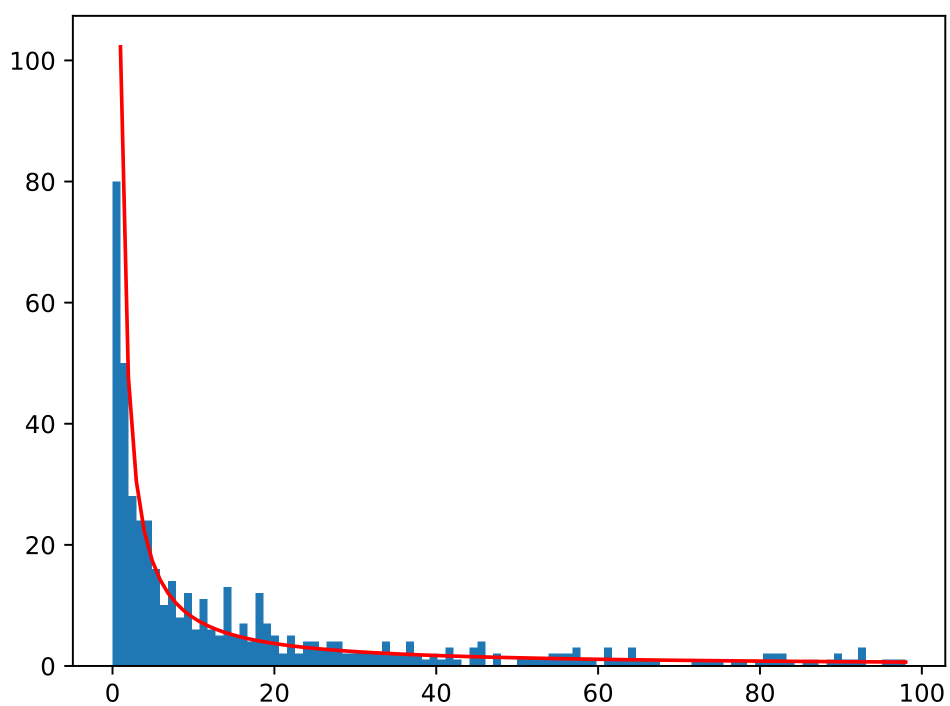
Lukas Fu Homework 1

Exercises

Exercise 3.1

The distribution of fire sizes are shown in the histogram in figure **??**. A large amount of smaller fire sizes and then sharply declining, and for larger fires the distribution is close to random but still declining. The distribution of larger fires are likely random due to the small number of large fires, which makes it hard to get a clean distribution.



Figur 1: Histogram of the sizes of forest fires using a lattice size of 256, growth rate of 0.01 and lightning strike probability of 0.2.

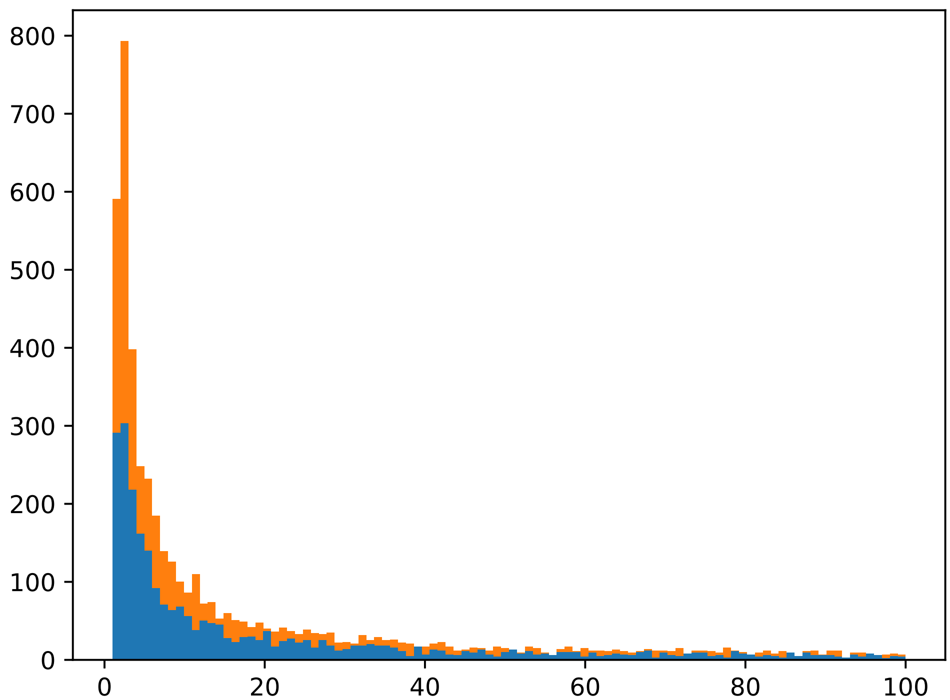
Exercise 3.3

The figure shows the power law distribution using the algorithm described and the formula of inverse of C, represented by colours orange and blue respectively. The number of trees used for the C-inverse method is 7000.

Furthermore we can analytically see that the two methods should be the same.

$$p(n) \propto x^{-n} \rightarrow C(n) = const. \cdot \int p(n)dn = C \cdot x^{-n+1}, \tag{1}$$

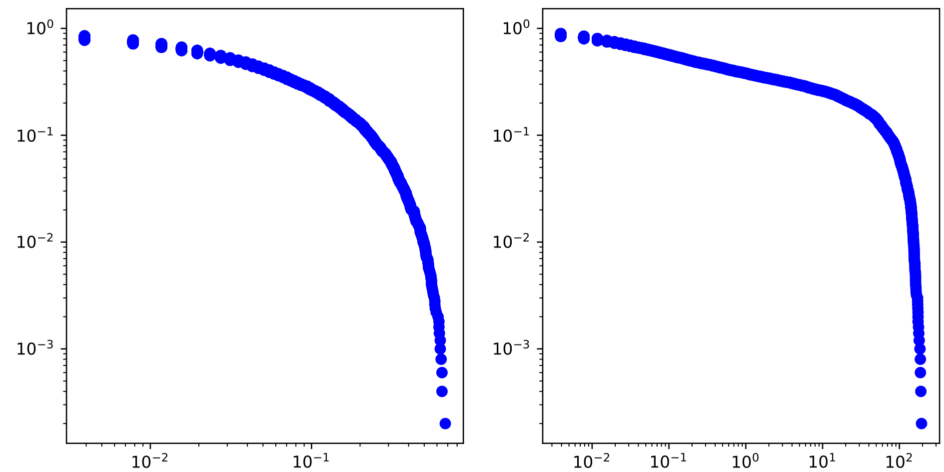
and the inverse of x^a is $x^{1/a}$.



Figur 2: Histogram of the Power law distribution. The orange bars show the distribution using C with the algorithm, and the blue shows the distribution using the formula of the inverse of C with rounding to the closest integer.

Exercise 3.4

For 5000 counts of forest fires, the following graphs show the cCDF for lattice sizes of $N = 16$ and $N = 256$. The deviation from linearity shown in the left graph for $N = 16$ could be caused by the fire sizes being limited by the lattice size and therefore reduced spread of data.



Figur 3: The left graph shows the $C(n)$ vs n/N^2 for $N = 16$, and the right for $N = 256$.

Code

Main - Forest Fire

```
import numpy as np
from tkinter import *
from PIL import Image
from PIL import ImageTk as itk
import time
import matplotlib.pyplot as plt
import math

# graphical code
# parameters (lattice size, growth parameter, lightning strike frequency)
N, p, f = 256, 0.01, 0.2

res = 500   # Animation resolution
tk = Tk()
tk.geometry( str(int(res*1.1)) + 'x' + str(int(res*1.3)) )
tk.configure(background='white')

canvas = Canvas(tk, bd=2)            # Generate animation window
tk.attributes('-topmost', 0)
canvas.place(x=res/20, y=res/20, height=res, width=res)
color = ['#0008FF', '#DB0000', '#12F200']

growth = Scale(tk, from_=0, to=0.03, orient=HORIZONTAL, label='Growth probability', font=("Hel
growth.place(relx=.12, rely=.85, relheight=0.12, relwidth=0.33)
growth.set(p)            # Parameter slider for growth rate

p_lightning = Scale(tk, from_=0, to=1, orient=HORIZONTAL, label='Lightning rate', font=("Helve
p_lightning.place(relx=.57, rely=.85, relheight=0.12, relwidth=0.33)
p_lightning.set(f)          # Parameter slider for lightning rate

S = np.zeros((N, N))   # status array, 0 = no tree, 1 = tree, 2 = burned, 3 = on fire
forest = np.zeros((N, N, 3))
fire_count = 0
fire_size_array = []

timestep = 10000
# for T in range(timestep):
while fire_count < 5000:
    growth_rate = growth.get()
    LP = p_lightning.get()
    # tree growth happens with probability of growth rate --- check lecture note
    S[(np.random.rand(N, N) < growth_rate) & (S == 0)] = 1
    # lightning strike location selected
    lightning_location = (np.random.rand(2)*N).astype(int)
    # if lightning strikes a tree
    if (S[lightning_location[0], lightning_location[1]] == 1) and (np.random.rand() < LP):
        # start a fire event, increment fire_count
        fire_count += 1
        # expand fire by checking adjacent tiles
        S[lightning_location[0], lightning_location[1]] = 3   # location struck set on fire
        # check directions right, left, up, down
        fire_size = 0
        while sum(sum(S == 3)) > 0:
            x = zip(np.where(S == 3)[0], np.where(S == 3)[1])
            for i, j in x:
```

```
                if S[min(i+1, N-1), j] == 1:
                    S[min(i+1, N-1), j] = 3
                    fire_size += 1
                if S[max(i-1, 0), j] == 1:
                    S[max(i-1, 0), j] = 3
                    fire_size += 1
                if S[i, min(j+1, N-1)] == 1:
                    S[i, min(j+1, N-1)] = 3
                    fire_size += 1
                if S[i, max(j-1, 0)] == 1:
                    S[i, max(j-1, 0)] = 3
                    fire_size += 1
                S[i, j] = 2
        fire_size_array.append(fire_size)

    # create image of forest, black background
    forest[:, :, :] = 0
    # burned trees are red
    forest[:, :, 0] = (S == 2) * 255
    # grown tree are green
    forest[:, :, 1] = (S == 1) * 255
    # image update
    img = itk.PhotoImage(Image.fromarray(np.uint8(forest), 'RGB').resize((res, res)))
    # recreate canvas
    canvas.create_image(0, 0, anchor=NW, image=img)
    tk.title('Fires:' + str(fire_count))
    tk.update()
    # if there are trees burning, add time delay
    if sum(sum(S == 2)) > 0:
        time.sleep(0.03)
    # set burning trees, 2 -> 0
    S[S == 2] = 0
Tk.mainloop(canvas)  # closes window and finish

plt.hist(fire_size_array, bins=100)
plt.show()
# np.savetxt('fire1c.csv', fire_size_array, delimiter=',')  #N=256
# np.savetxt('fire4a.csv', fire_size_array, delimiter=',')  #N=16
# np.savetxt('fire4b.csv', fire_size_array, delimiter=',')  #N=256
```

3.1

```
import numpy as np
import matplotlib.pyplot as plt

N=256
data = np.loadtxt('fire1c.csv')
data = data[data<100]
counts, bins, bars = plt.hist(data,100)
n = np.array([x for x in bins])

alpha = 1.1
p = 100*n**(-alpha)
plt.plot(n,p, 'r')
plt.show()
```

3.3

```
import numpy as np
```

```
import matplotlib.pyplot as plt

#Trees
nMin = 1
nMax = 5000
nNumber = 7000  # to simulate
alpha = 1.1

nList = np.linspace(nMin,nMax,nMax-nMin+1)
pList = nList**(-alpha)  # Probability for n
pList = 1/sum(pList)*pList  # probability after making sure 0<P<1
cList = [sum(pList[i:]) for i in range(len(nList))]
#random number
randomNumberList = np.random.rand(nNumber)

cArray = np.array(cList)
for i in range(len(randomNumberList)):  # finds index of the random number closest to cArray
    idx = (np.abs(cArray - randomNumberList[i])).argmin()  # x = abs(CDF-r)--> lowest value in
    randomNumberList[i] = nList[idx]  # to closest index. found with argmin()
# alters randomNumberList to match nList by reordering randomNumberList

graphMin=1
graphMax=100
plt.hist(randomNumberList,graphMax,range=[graphMin, graphMax],color='C1',alpha=0.8)
# plot random generated distr.

ntrees = np.round(nMin*np.random.rand(nNumber)**(1/(1-alpha)))  # rounding c^-1
plt.hist(ntrees,graphMax,range=[graphMin, graphMax],alpha=0.4)  # plot c^-1 distribution
plt.show()
counts1,bin1,staple1 = plt.hist(ntrees,graphMax,alpha=0.4)
counts2,bin2,staple2 = plt.hist(ntrees,graphMax,range=[graphMin, graphMax],alpha=0.4)
# random distr. is orange, ntrees is brown
```

3.4

```
import numpy as np
import matplotlib.pyplot as plt

N=16
data_load = np.loadtxt('fire4a.csv')
fire_array = np.array(data_load)
fire_array=np.sort(fire_array)
k = len(fire_array)
C = [(k-x)/k for x in range(k)]
plt.subplot(1,2,1)
plt.loglog(fire_array/pow(N,2),C,'bo')

data_load = np.loadtxt('fire4b.csv')
fire_array = np.array(data_load)
fire_array=np.sort(fire_array)
k = len(fire_array)
C = [(k-x)/k for x in range(k)]
plt.subplot(1,2,2)
plt.loglog(fire_array/pow(N,2),C,'bo')
plt.show()
```