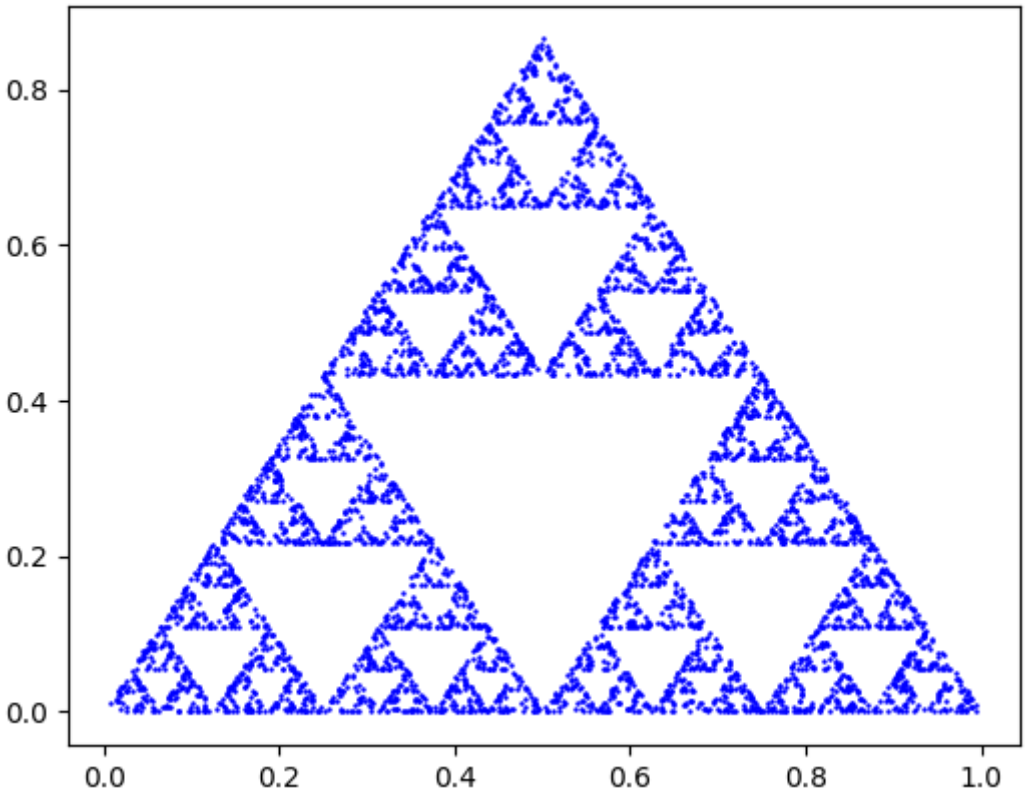
گزارش کار تمرین دوم شبیه‌سازی رایانه‌ای در فیزیک

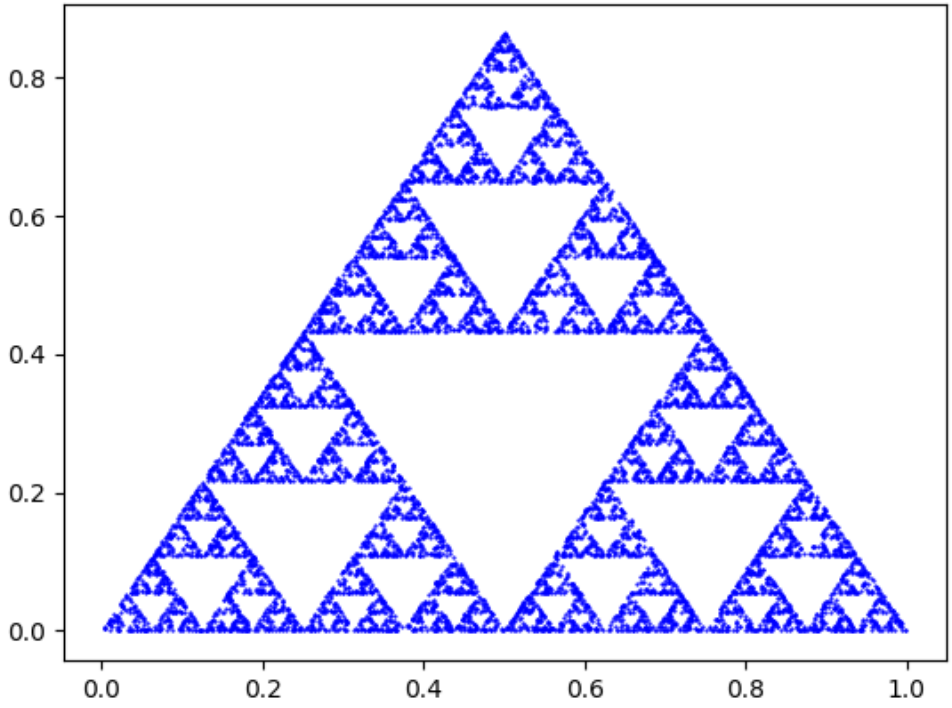
علی اکرامیان – 99100563

تمرین 2.5 (مثلث سرپینسکی):

من برای تولید این مجموعه، سه تابع تعریف کرده‌ام. تابع اول f1 صرفا ایکس و وای را نصف می‌کند و ریترن می‌کند. تابع دوم f2 ابتدا ایکس و وای را نصف می‌کند سپس ایکس را به اندازه‌ی نصف انتقال به جلو می‌دهد. تابع f3 نیز نصف کرده و ایکس را یک چهارم و وای را رادیکال سه چهارم انتقال می‌دهد که به نوک مثلث می‌رسد. حال باید رندوم نقطه در صفحه درست کنیم و به طور رندوم نیز این توابع سه‌گانه را اثر دهیم تا به مثلث سرپینسکی برسیم. این‌گونه عمل می‌کنم که یک حلقه‌می‌زنم و N سری نقطه تولید می‌کنم. (N طبق نوتیشن خود کد است) حال یک ایکس و وای رندوم بین اعداد 0 تا 10 تولید می‌شود (با کتابخانه‌ی رندوم) و یک حلقه‌ی دیگر درون این می‌زنم که 20 تابع رندوم از بین f1 و f2 و f3 انتخاب کرده و اثر دهد و 20 بار از بین این 1 تا 3 انتخاب کرده (با کتابخانه‌ی رندوم) و روی این نقطه‌ی رندوم از صفحه اثر می‌دهد. و سپس این نقطه را پلات می‌کنیم (با کتابخانه‌ي matplotlib) حال اگر برای N نقطه این کار را انجام بدهیم و N را زیاد کنیم، به مثلث سرپینسکی میرسیم.
حال نتایج را می‌بینیم:



برای 5000 نقطه‌ي تصادفی در صفحه



برای 10000 نقطه‌ي تصادفی در صفحه

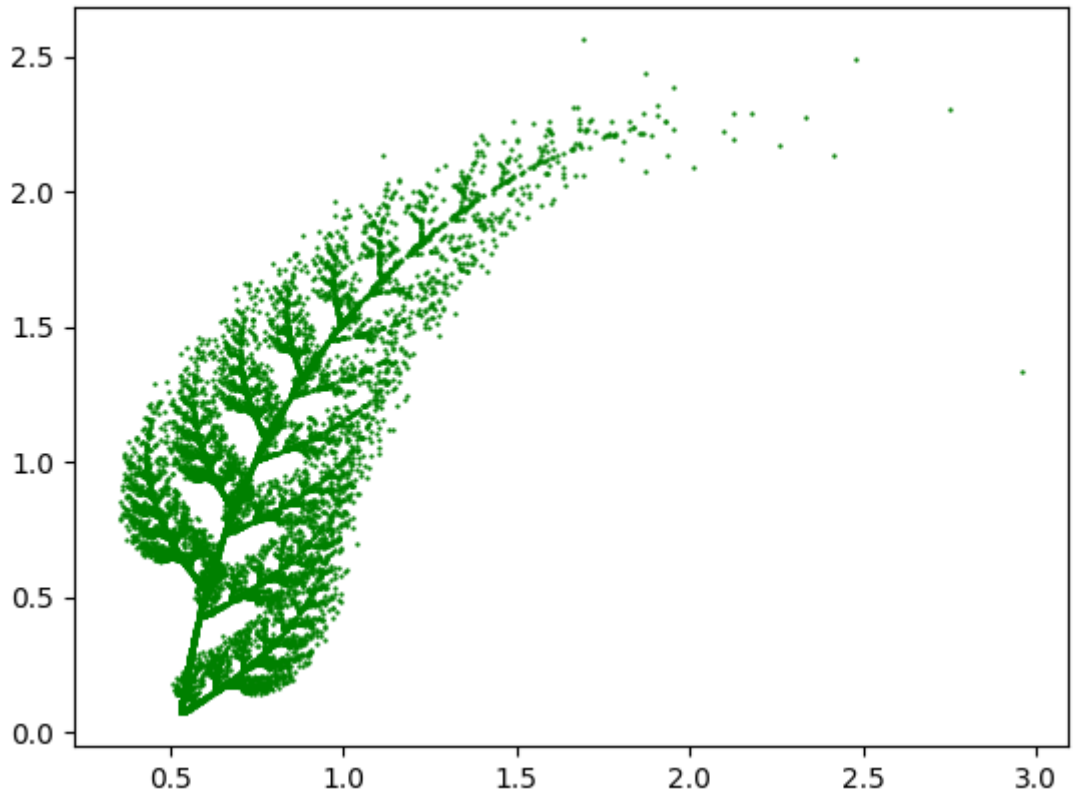
کد را نیز در صفحه‌ي بعد می‌بینیم.

simulation > HW2 > Sierpinski-random.py > ...

```python
import matplotlib.pyplot as plt
import numpy as np
import random as rnd

t=np.sqrt(3)/2
def f1(x0,y0):
    x0=x0/2
    y0=y0/2
    return x0,y0
def f2(x0,y0):
    x0=x0/2
    y0=y0/2
    x0=x0+0.5
    return x0,y0
def f3(x0,y0):
    x0=x0/2
    y0=y0/2
    x0=x0+0.25
    y0=y0+0.5*t
    return x0,y0

for i in range(1,10001):
    a=rnd.randint(0,10)
    b=rnd.randint(0,10)
    for j in range(1,20):
        u=rnd.randint(1,3)
        if u==1:
            a,b=f1(a,b)
        if u==2:
            a,b=f2(a,b)
        if u==3:
            a,b=f3(a,b)
    plt.plot(a,b,'bo',markersize=0.55)

plt.show()
```

تمرین 2.6 (برگ سرخس):

برای این تمرین از 4 تابع استفاده می‌کنیم که تابع اول با ضریب‌های 0.9 طول و عرض را اسکیل می‌کند و سپس 5 درجه آن را به سمت راست می‌چرخاند و 0.1 آن را به سمت راست و 0.5 به سمت بالا می‌برد. تابع دوم f2 نیز طول و عرض را با نسبت 0.3 اسکیل کرده و آن را 40 درجه می‌چرخاند و سپس با نسبت 0.5 و 0.4 عرض و طول را اسکیل می‌کند. تابع f3 نیز عرض را با نسبت 0.35- و طول را با نسبت 0.35 که چون عرض منفی است یعنی یک بار قرینه شده است. و سپس آن را 60 درجه چرخانده و 0.6 و 0.1- انتقال می‌دهیم. حال می‌ماند ساقه‌ها که آن را هم با تابع f4 می‌کشیم: ابتدا عرض را با نسبت 0.01 اسکیل کرده و طول را با 0.2 اسکیل می‌کنیم سپس 3 درجه چرخانده و 0.5 و 0.3- انتقال می‌دهم. حال اگر مانند تمرین قبل برای 20000 نقطه حلقه بزنیم که نقطه‌ی رندوم جنریت کند و سپس حلقه بزنیم تا تابع را به طور رندوم بردارد نتیجه‌ی زیر را می‌گیریم. البته احتمال برداشتن تابع f1 را بیشتر کردم چون سر آن شکل بهتری می‌گیرد:



کد را نیز در صفحه‌ی بعد می‌بینیم.

در ضمن ماتریس R5 ماتریس دوران است که با عملگر ضرب ماتریسی " @ " آن را در بردار مختصات نقطه‌ی خودمان ضرب کردم.

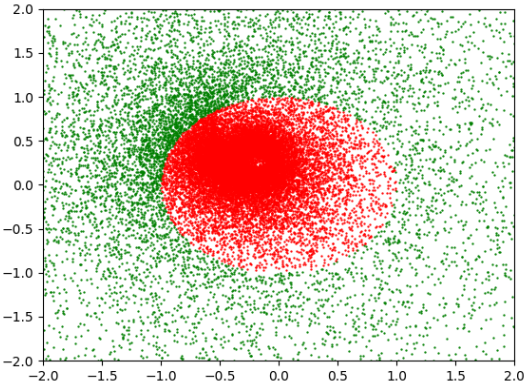
simulation > HW2 > fern.py > ...

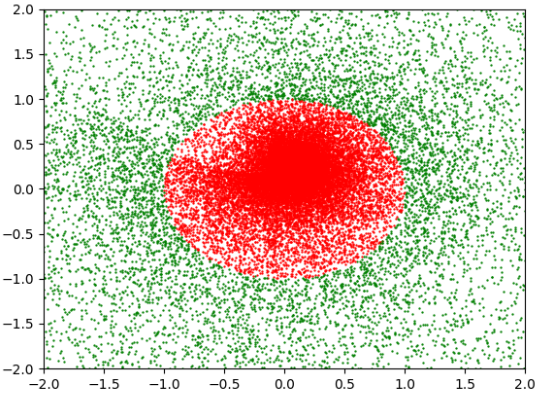
```python
import matplotlib.pyplot as plt
import numpy as np
import random as rnd

t=np.sqrt(3)/2
rad=np.pi/180
def f1(x0,y0):
    x0*=0.9
    y0*=0.9
    R5=np.array([[np.cos(-5*rad), -np.sin(-5*rad)], [np.sin(-5*rad), np.cos(-5*rad)]])
    res=R5 @ (np.array([x0,y0]))
    x0,y0=res[0],res[1]
    x0+=0.1
    y0+=0.4
    return x0,y0
def f2(x0,y0):
    x0*=0.3
    y0*=0.3
    R5=np.array([[np.cos(40*rad), -np.sin(40*rad)], [np.sin(40*rad), np.cos(40*rad)]])
    res=R5 @ (np.array([x0,y0]))
    x0,y0=res[0],res[1]
    x0+=0.5
    y0+=0.4
    return x0,y0
def f3(x0,y0):
    x0*=(-0.35)
    y0*=0.35
    R5=np.array([[np.cos(-60*rad), -np.sin(-60*rad)], [np.sin(-60*rad), np.cos(-60*rad)]])
    res=R5 @ (np.array([x0,y0]))
    x0,y0=res[0],res[1]
    x0+=0.6
    y0+=-0.1
    return x0,y0
```

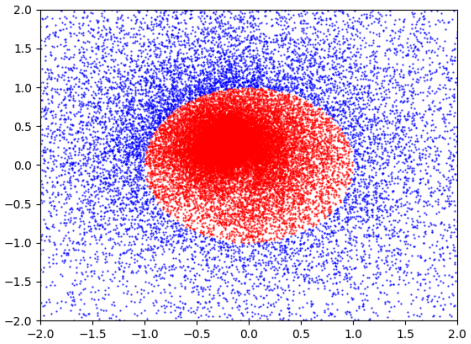
```python
def f4(x0,y0):
    x0*=0.01
    y0*=0.2
    R5=np.array([[np.cos(-3*rad), -np.sin(-3*rad)], [np.sin(-3*rad), np.cos(-3*rad)]])
    res=R5 @ (np.array([x0,y0]))
    x0,y0=res[0],res[1]
    x0+=0.5
    y0+=-0.3
    return x0,y0
for i in range(1,20001):
    a=rnd.randint(0,10)
    b=rnd.randint(0,10)
    for j in range(1,20):
        u=rnd.randint(1,7)
        if u==3:
            a,b=f4(a,b)
        if u==1:
            a,b=f2(a,b)
        elif u==2:
            a,b=f3(a,b)
        else:
            a,b=f1(a,b)
    plt.plot(a,b,'go',markersize=0.65)

plt.show()
```

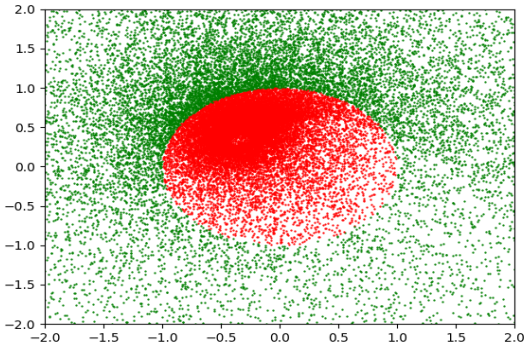
تمرین 2.7 (مجموعه‌های ژولیا):

برای تولید این مجموعه‌ها ابتدا یک تابع تعریف می‌کنیم که یک عدد مختلط می‌گیرد z و یک ثابت c نیز که برای انتقال است و سپس fz=z^2+c را ریترن می‌کند. حال یک حلقه می‌زنیم و 100000 تا نقطه از صفحه را به طور رندوم تولید کرده و یک حلقه‌ی دیگر میزنیم که 5 بار تابع f را روی این z اثر میدهد و اگر نرم آن بیشتر از 1 شود به رنگ آبی/سبز و اگر کمتر از 1 شود به رنگ قرمز آن را در صفحه پلات می‌کند.حال برای چند c مختلف نشان می‌دهیم:









کد را نیز در صفحه‌ي بعد می‌بینیم.

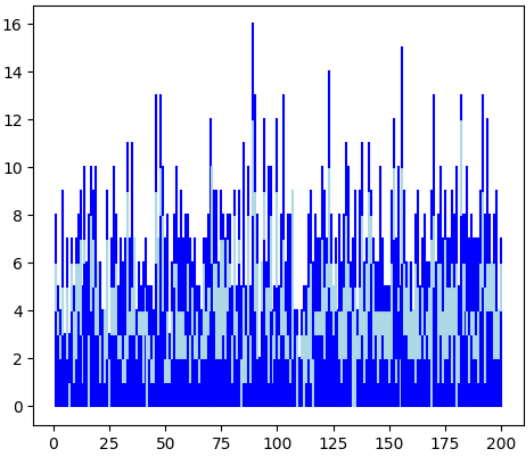
که آن چیزی که می‌خواستیم نشد متاسفانه :(

simulation > HW2 > julia.py > ...

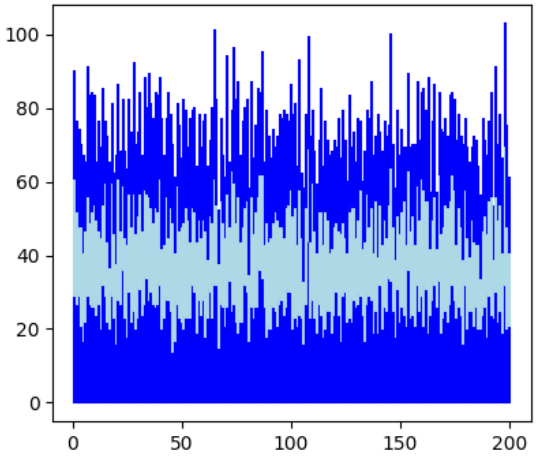
```python
import matplotlib.pyplot as plt
import numpy as np
import random as rnd
def f(z,c):
    fz=z**2+c
    return fz
c=complex(-0.2,0.3)
print(c)
for j in range(1,100001):
    x,y=rnd.random(),rnd.random()
    z=complex(x,y)
    for i in range(1,5):
        z=f(z,c)
        #print(z)
    norm=(z.real**2+z.imag**2)**0.5
    if norm>1:
        plt.plot(z.real,z.imag,'bo',markersize=0.5)
    if norm<=1:
        plt.plot(z.real,z.imag,'ro',markersize=0.5)

plt.xlim([-2,2])
plt.ylim([-2,2])
plt.show()
```

تمرین 2.7 (ول نشست):

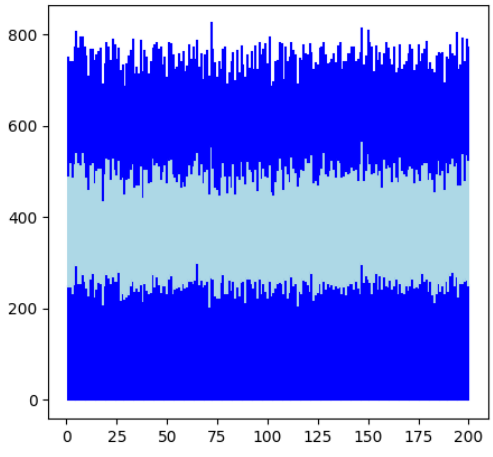
ابتدا یک آرایه‌ی base تعریف می‌کنیم که 1 تا 200 است و لایه‌ی زیرین را تشکیل می‌دهد. سپس یک آرایه‌ی متشکل از 0 می‌سازیم که 200 تا است و ارتفاع را قرار است نشان دهد. حال یک حلقه می‌زنیم و N عدد رندوم تولید می‌کنیم و به طور رندوم به یکی ازین درایه‌های height اساین می‌کنیم. حال این می‌شود لایه‌ی اول برف ما. حال اگر این آرایه را کپی کنیم و این کار را 3 بار دیگر انجام دهیم تا 3 لایه برف داشته باشیم، حال یکی درمیان با رنگ آبی پررنگ و کم رنگ پلات می‌کنیم. و خواهیم دید که هرچه N زیاد شود، سطح برف نرم‌تر (یکسان‌تر) می‌شود. برای کشیدن نیز یک خط از نقطه‌ی ارتفاع هر درایه height به هر درایه‌ی آرایه‌ي لایه‌ی قبلی تا base می‌کشیم. در انتها میانگین هر لایه را نیز محاسبه می‌کنیم و پرینت می‌کنیم ارتفاع میانگین تا هر لایه را. نتیجه‌ی نهایی را برای چند N می‌بینیم:



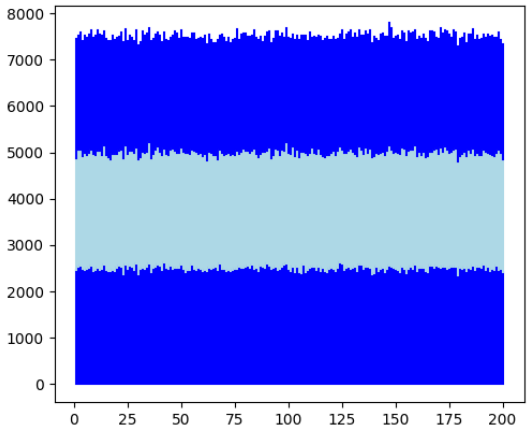
N = 500



N = 5000

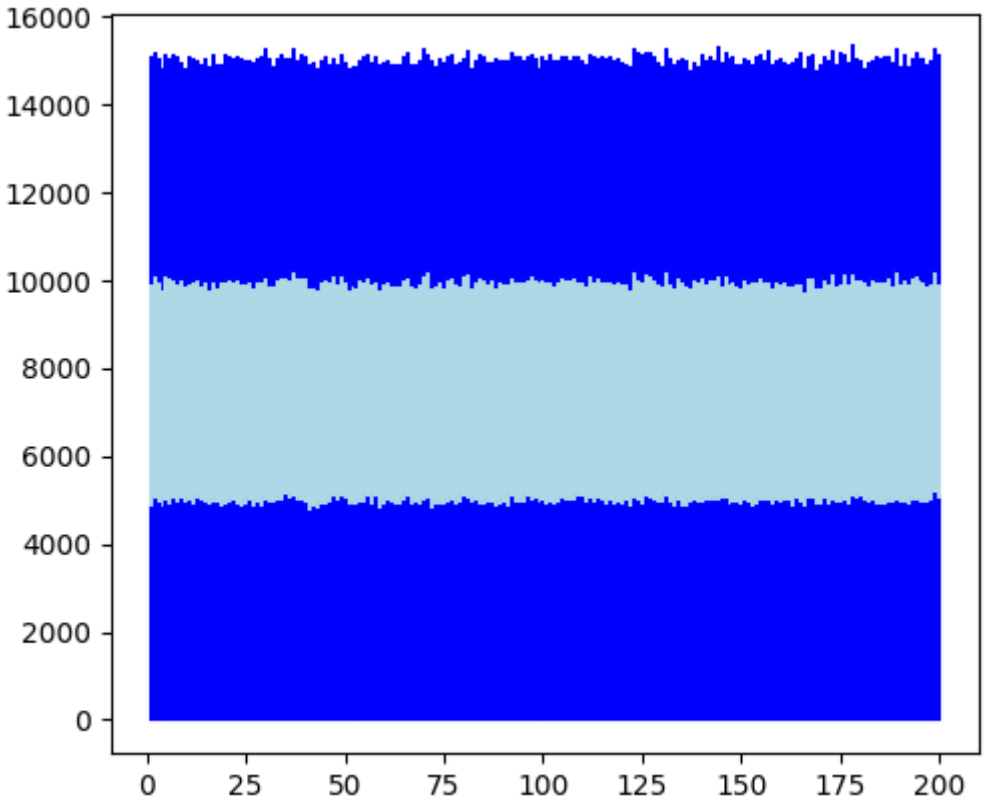


N = 50000



N = 5000

حال برای 1 میلیون داده‌ی رندوم این کار را انجام می‌دهیم!



که برای عدد یک میلیون می‌بینیم سطح برف‌ها تقریبا صاف است.

میانگین و واریانس را نیز می‌توان به راحتی حساب و پرینت کرد. در ادامه کد را می‌بینیم.

simulation > HW2 > RBD.py > ...

```python
import matplotlib.pyplot as plt
import numpy as np
N=1000000
base=np.arange(1,201)
height=np.zeros(200)
for i in range(1,N+1):
    rnd=np.random.randint(1, 201)
    height[(rnd-1)]+=1
height1=height.copy()
height=np.zeros(200)
for i in range(1,N+1):
    rnd=np.random.randint(1, 201)
    height[(rnd-1)]+=1
height2=height.copy()
height2+=height1
height=np.zeros(200)
for i in range(1,N+1):
    rnd=np.random.randint(1, 201)
    height[(rnd-1)]+=1
height3=height2+height
for i in range(0,200):
    plt.plot([base[i],base[i]],[0,height1[i]],'b')
    plt.plot([base[i],base[i]],[height1[i],height2[i]],'lightblue')
    plt.plot([base[i],base[i]],[height2[i],height3[i]],'b')
av_height1=sum(height1)/200
av_height2=sum(height2)/200
av_height3=sum(height3)/200
print(av_height1,av_height2,av_height3)
plt.show()
```