گزارش تمرین سری پنجم

شبیه سازی رایانه ای در فیزیک

علیرضا رضایی

97100762

1 فهرست

2 تمرین 4.1

تمرین ۴٫۱ -

حل

$$x(t) = x(t-T) + al \Rightarrow \langle x^2(t)\rangle = \langle (x(t-T) + al)^2 \rangle$$

$$\Rightarrow \langle x^2(t)\rangle = \langle x^2(t-T)\rangle + 2\langle al\, x(t-T)\rangle + l^2\langle a^2\rangle \quad \boxed{1}$$

$$\langle a^2\rangle = \frac{p(1)^2 + q(-1)^2}{p+q} = 1 \quad \boxed{2}$$

چون a و x مستقل از هم اند:

$$\langle al\, x(t-T)\rangle = \langle a\rangle l \langle x(t-T)\rangle \quad \boxed{3}$$

$$\langle a\rangle = \frac{p\times 1 + q\times(-1)}{p+q} = p-q \quad \boxed{4}$$

با جاگذاری $\boxed{2}$، $\boxed{3}$، $\boxed{4}$ در $\boxed{1}$:

$$\langle x^2(t)\rangle = \langle x^2(t-T)\rangle + 2(p-q)l\langle x(t-T)\rangle + l^2 \quad \boxed{5}$$

ص ۲

از طرفی :

$$\langle x(t)\rangle = \langle x(t-\tau)\rangle + \langle a\rangle L$$

$$\Rightarrow \langle x(t-\tau)\rangle = \langle x(t-2\tau)\rangle + \langle a\rangle L \quad \boxed{1*}$$

$$\Rightarrow \langle x(t-2\tau)\rangle = \langle x(t-3\tau)\rangle + \langle a\rangle L \quad \boxed{2*}$$

با جاگذاری $\boxed{2*}$ در $\boxed{1*}$:

$$\langle x(t-\tau)\rangle = \langle x(t-3\tau)\rangle + 2\langle a\rangle L$$

و با ادامه ی همین روند و با توجه به اینکه $\langle x(0)\rangle = 0$ (چون x در ۰ کاملا در مکان مشخص صفر است) :

$$\boxed{\langle x(t-\tau)\rangle = \frac{L}{\tau}(p-q)(t-\tau)} \quad \boxed{4}$$

ص ۳

با جاگذاری $\boxed{6}$ در $\boxed{5}$:

$$\langle x^2(t)\rangle = \langle x^2(t-\tau)\rangle + \frac{2l^2}{\tau}(p-q)^2(t-\tau) + l^2 \quad \boxed{7}$$

و با یک تغییر متغیر :

$$\langle x^2(t-\tau)\rangle = \langle x^2(t-2\tau)\rangle + \frac{2l^2}{\tau}(p-q)^2(t-2\tau) + l^2 \quad \boxed{8}$$

و با قراردادن $\boxed{8}$ در $\boxed{7}$:

$$\langle x^2(t)\rangle = \langle x^2(t-2\tau)\rangle + \frac{2l^2}{\tau}(p-q)^2(2t-3\tau) + 2l^2$$

و با ادامه ی همین روند خواهیم داشت :

$$\langle x^2(t)\rangle = \cancel{\langle x^2(0)\rangle}^{\,0} + \frac{2l^2}{\tau}(p-q)^2\left(\frac{t^2}{\tau} - \sum_{n=1}^{\frac{t}{\tau}} n\tau\right) + \frac{t}{\tau}l^2 \quad \boxed{9}$$

با کمی ساده سازی و توجه به اینکه $x(0)$ حتماً صفر است خواهیم داشت :

$$\langle x^2(t)\rangle = \frac{2l^2}{\tau}(p-q)^2\left(\frac{t^2}{\tau} - \tau\sum_{n=1}^{\frac{t}{\tau}} n\right) + \frac{t}{\tau}l^2$$

و با استفاده از رابطه ی $1+2+3+\dots+n = \frac{n(n+1)}{2}$ خواهیم داشت :

$$\langle x^2(t)\rangle = \frac{2l^2}{\tau}(p-q)^2\left(\frac{t^2}{\tau} - \tau\frac{\frac{t}{\tau}\left(\frac{t+\tau}{\tau}\right)}{2}\right) + \frac{t}{\tau}l^2$$

$$= \frac{2l^2}{\tau}(p-q)^2\left(\frac{t^2}{\tau} - \frac{t(t+\tau)}{2\tau}\right) + \frac{t}{\tau}l^2 \quad \boxed{10}$$

از طرفی طبق رابطه‌ی ۴ کتاب :

$$\langle x(t)\rangle^2 = \frac{l^2}{T^2}t^2(p-q)^2 \qquad \boxed{11}$$

با ترکیب $\boxed{11}$ و $\boxed{10}$:

$$\sigma^2 = \frac{2l^2}{T}(p-q)^2\left(\frac{t^2}{T} - \frac{t(t+T)}{2T}\right) + \frac{t}{T}l^2 - \frac{l^2}{T^2}t^2(p-q)^2$$

با فاکتور گرفتن از $\frac{t}{T}l^2$:

$$\sigma^2 = \frac{t}{T}l^2\left[2(p-q)^2\left(\frac{t}{T} - \frac{t+T}{2T}\right) + 1 - \frac{t}{T}(p-q)^2\right]$$

$$= \frac{t}{T}l^2\left[(p-q)^2\left(\frac{2t}{T} - \frac{t}{T} - \frac{T}{T} - \frac{t}{T}\right) + 1\right] = \frac{t}{T}l^2\left[1-(p-q)^2\right]$$

$$= \frac{t}{T}l^2\left[1-(p^2+q^2)+2qp\right] \qquad \boxed{12}$$

از طرفی : $p+q=1 \xrightarrow{\text{توان ۲}} p^2+q^2+2pq = 1 \Rightarrow p^2+q^2 = 1-2pq \qquad \boxed{13}$

با جاگذاری $\boxed{13}$ در $\boxed{12}$:

$$\sigma^2 = \frac{t}{T}l^2\left[1-1+2pq+2pq\right] = \frac{t}{T}l^2\left[4pq\right]$$

$$\Rightarrow \boxed{\sigma^2 = \frac{4l^2}{T}pqt}$$

3 تمرین 4.2

3.1 مقدمه

در این تمرین میخواهیم صحت روابط 4 و 5 از فصل پنج کتاب را بررسی کنیم.

$$\begin{aligned} < x(t) > &= < x(t-\tau) > + < a > l \\ &= < x(t-\tau) > + (p-q)l \\ &= < x(t-2\tau) > + (p-q)l + (p-q)l = \frac{l}{\tau}(p-q)\, t \end{aligned} \tag{4}$$

$$\sigma^2 = < x^2 > - < x >^2 = \frac{4l^2}{\tau} pq\, t. \tag{5}$$

برای این کار همانطور که از فرمول ها پیداست نیاز داریم به ازای یک تعداد قدم زمانی مشخص (که معادل می شود با t در کدمان) و یک p مشخص ، رندوم واک را انجام داده و فاصله اش را از مبدا (x) بدست بیاوریم. این کار را برای آن t مشخص ، به تعداد زیادی انجام داده و از x های بدست آمده میانگین می گیریم تا مقدار چشمداشتی x را بدست آوریم. (و یا از توان دوم x ها میانگین میگیریم تا در رابطه ی 5 استفاده کنیم) و با این کار ما مقادیر چشم داشتی و انحراف معبار استاندارد ناشی از شبیه سازی را داریم و می توانیم با مقادیری که از تئوری برای این پارامتر ها آمده اند مقایسه کنیم.

برای توضیح کد ها هم همه ی جزییات فنی بصورت کامنت گذاری در کد توضیح داده شده است و فقط به توضیح تابع رندوم واک که قسمت اصلی ماجراست در اینجا بسنده می کنیم.

برای شبیه سازی رندوم واک از تابع توزیع رندوم باینومیال از کتابخانه ی نامپای استفاده کرده ایم که تعداد کل قدم ها (n_walk_step) و احتمال به سمت راست رفتن (p) را از ما می گیرد و به ما میگوید که با این احتمال ، این متحرک چند قدم از کل قدم هایش را به سمت راست برداشته است.

با کمک این مقدار می توانیم تعداد قدم های به سمت چپ را هم با یک تفریق ساده ی قدم های سمت راست از کل قدم ها بدست بیاوریم و سپس با محاسبه ی اختلاف تعداد قدم های رفه به سمت راست و چپ ، تعداد موثر قدم ها را بدست آورده و با ضرب در طول هر قدم ، مکان نهایی (x) متحرک را بدست آوریم.

این تابع باینومیال یک متغیر دیگر به اسم number_of_test هم می گیرد که به آن تعداد این تست را انجام می دهد و مکان های نهایی مختلف را در قالب یک آرایه از x ها خروجی می دهد که میتوانیم روی آن ها اعمال میانگین گیری و به توان 2 رساندن و غیره را انجام دهیم.

این کار را همانطور که در متن سوال خواسته برای p های مختلف تکرار کرده و نتایج را گزارش می دهیم.

3.2 نتایج

همانطور که مشاهده می کنیم اختلاف مقادیر شبیه سازی شده و محاسبه شده بسیار کم است و در حالت p=0.5 هم چون در قسمت مقدار چشم داشتی ، مقدار چشمداشتی محاسبقه شده دقیقا صفر است این خطا 100 درصد شده وگرنه اختلاف خیلی ناچیزی نسبت به طول شبکه دارند.

علاوه بر این مورد برای بقیه ی موارد هم علاوه بر درصد خطای نسبی ، اختلاف های مربوط به مقادیر چشمداشتی نسبت به طول شبکه (200 واحد) به وضوح خیلی کوچک (در همه ی موارد کمتر از 1 واحد) است.

```
*** by error we mean percentage of relative error ***

for p=0.1:
calculated expectation value is: -8000.0
simulated expectation value is: -8000.21624
Error is: -0.0 %
calculated std is: 3600.0000000000005
simulated std is: 3587.9073202610016
Error is: 0.34 %

for p=0.2:
calculated expectation value is: -6000.0
simulated expectation value is: -6000.34318
Error is: -0.01 %
calculated std is: 6400.000000000001
simulated std is: 6453.725627489388
Error is: -0.83 %

for p=0.30000000000000004:
calculated expectation value is: -3999.999999999999
simulated expectation value is: -3999.6775
Error is: 0.01 %
calculated std is: 8400.0
simulated std is: 8394.844593752176
Error is: 0.06 %

for p=0.4:
calculated expectation value is: -1999.9999999999995
simulated expectation value is: -2000.19786
Error is: -0.01 %
calculated std is: 9600.0
simulated std is: 9561.72361142002
Error is: 0.4 %

for p=0.5:
calculated expectation value is: 0.0
simulated expectation value is: 0.32518
Error is: -100.0 %
calculated std is: 10000.0
simulated std is: 10004.5215779676
Error is: -0.05 %
```

```
for p=0.6:
```

```
calculated expectation value is: 1999.9999999999995
simulated expectation value is: 1999.78374
Error is: 0.01 %
calculated std is: 9600.0
simulated std is: 9628.499271612149
Error is: -0.3 %
```

```
for p=0.7000000000000001:
```

```
calculated expectation value is: 4000.0000000000014
simulated expectation value is: 4000.40158
Error is: -0.01 %
calculated std is: 8399.999999999998
simulated std is: 8403.508933501318
Error is: -0.04 %
```

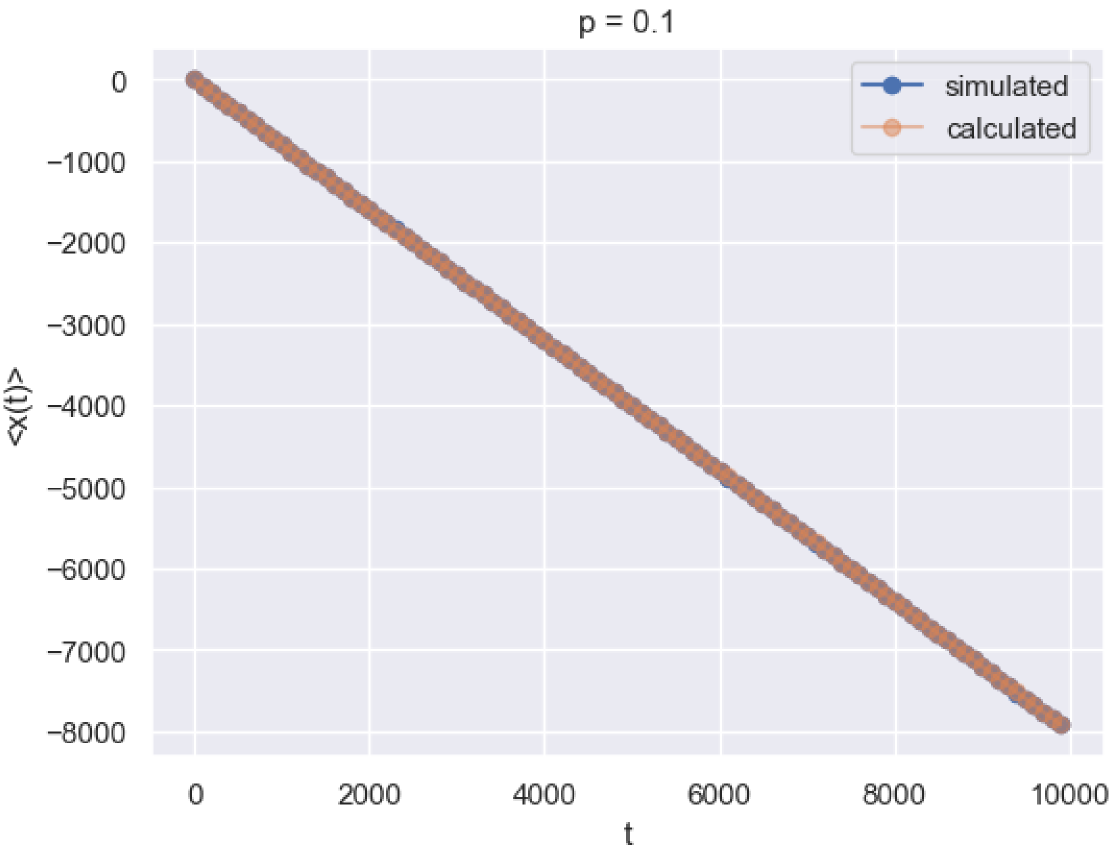
```
for p=0.8:
```

```
calculated expectation value is: 6000.000000000001
simulated expectation value is: 6000.24374
Error is: -0.0 %
calculated std is: 6399.999999999999
simulated std is: 6389.720470808446
Error is: 0.16 %
```

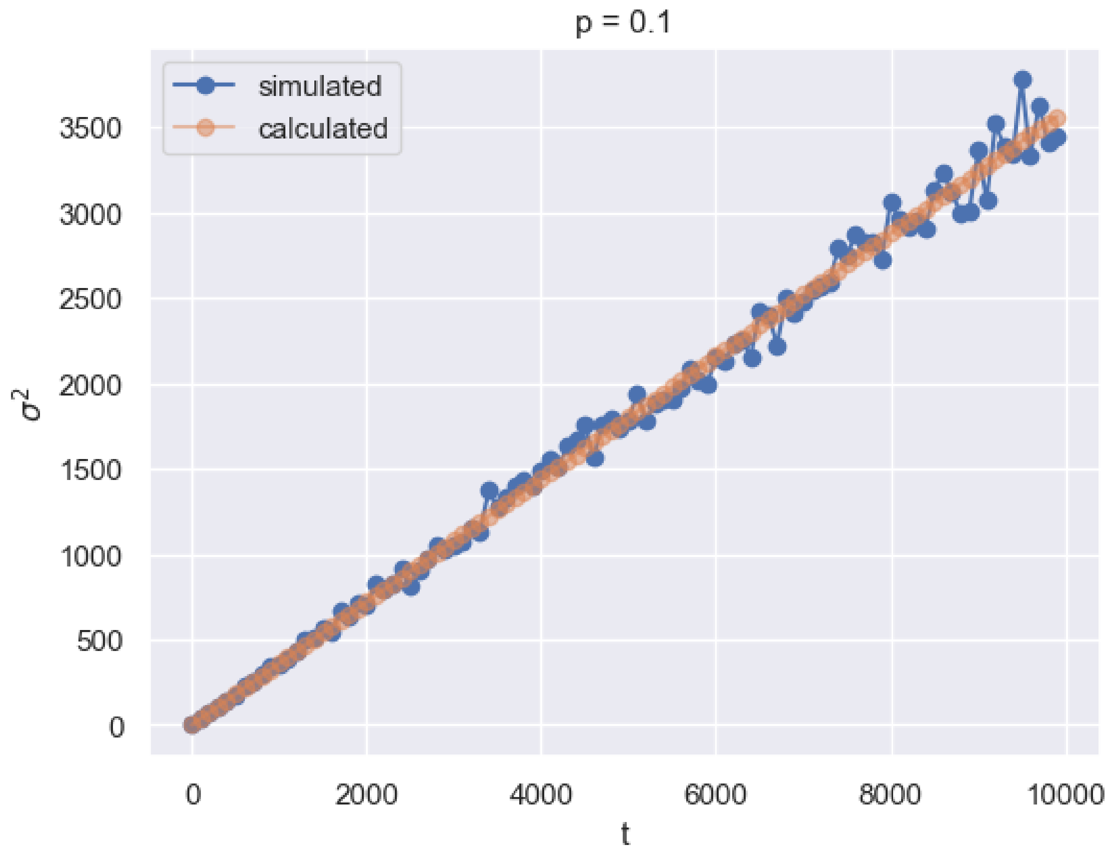
```
for p=0.9:
```

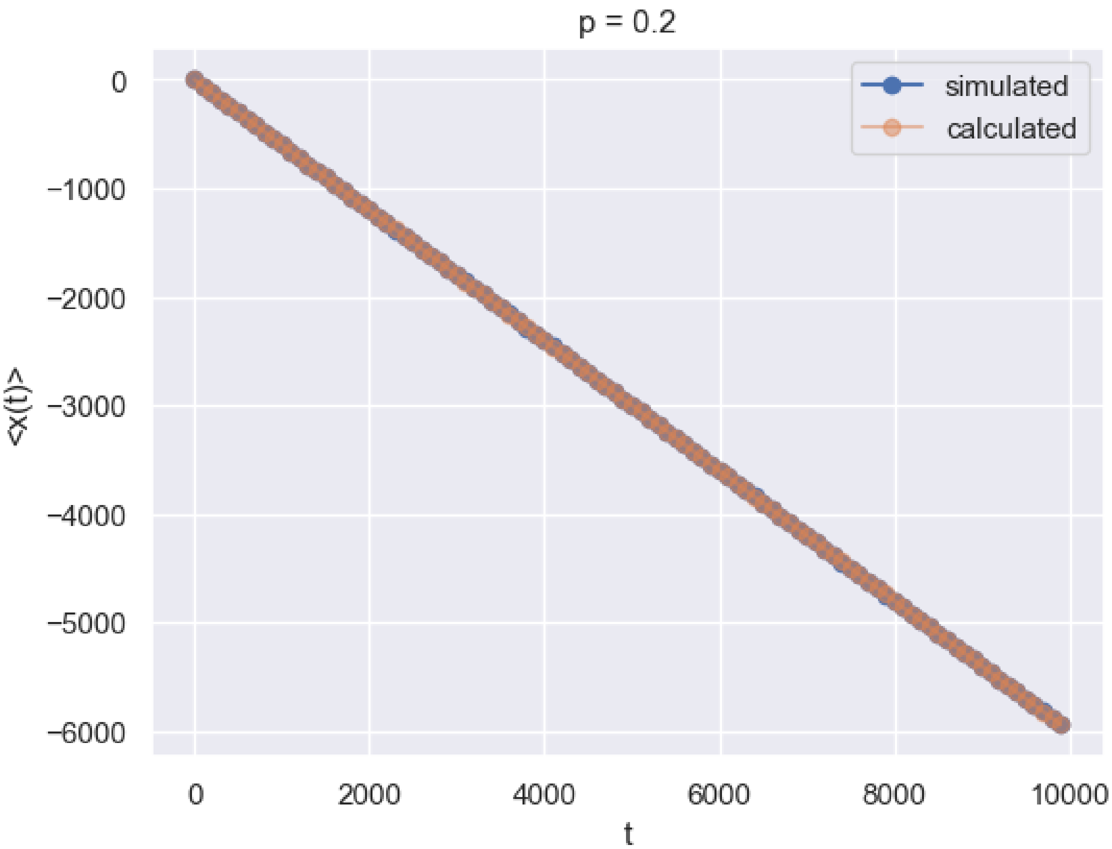
```
calculated expectation value is: 8000.0
simulated expectation value is: 8000.00046
Error is: -0.0 %
calculated std is: 3599.999999999999
simulated std is: 3615.751719787717
Error is: -0.44 %
```

در اینجا کار را به اتمام نمی رسانیم و نمودار پارامتر هایمان را برای زمان های مختلف هم میکشیم که از صحت نتایج مطمئن تر شویم:

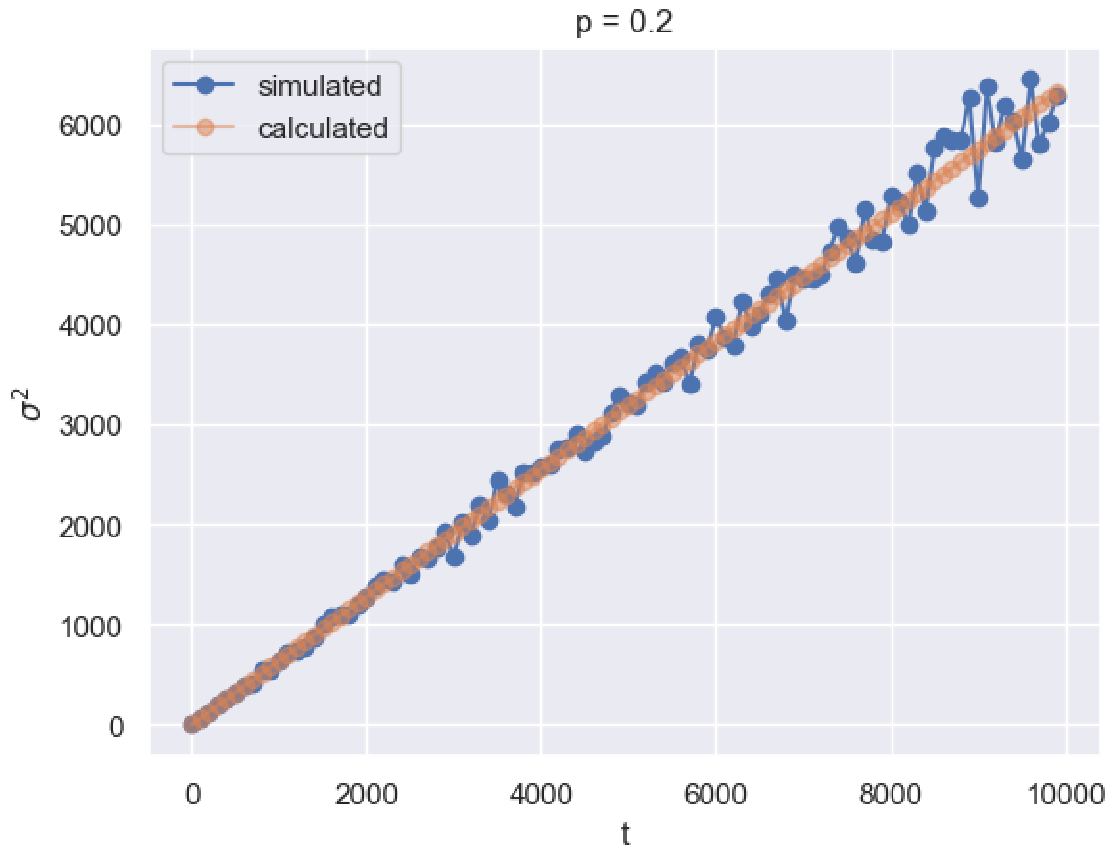


p = 0.1
simulated
calculated
0
−1000
−2000
−3000
−4000
−5000
−6000
−7000
−8000
<x(t)>
0
2000
4000
6000
8000
10000
t

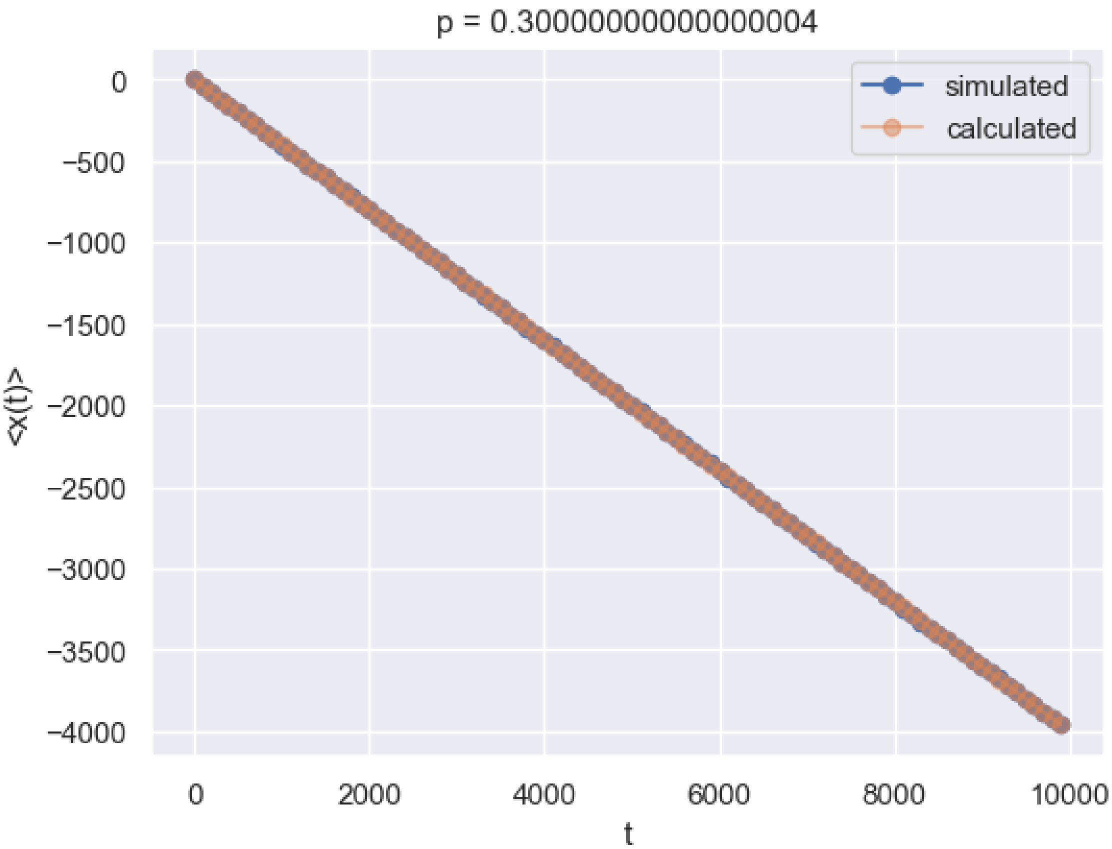

p = 0.1
simulated
calculated
σ^2
t
0
500
1000
1500
2000
2500
3000
3500
0
2000
4000
6000
8000
10000



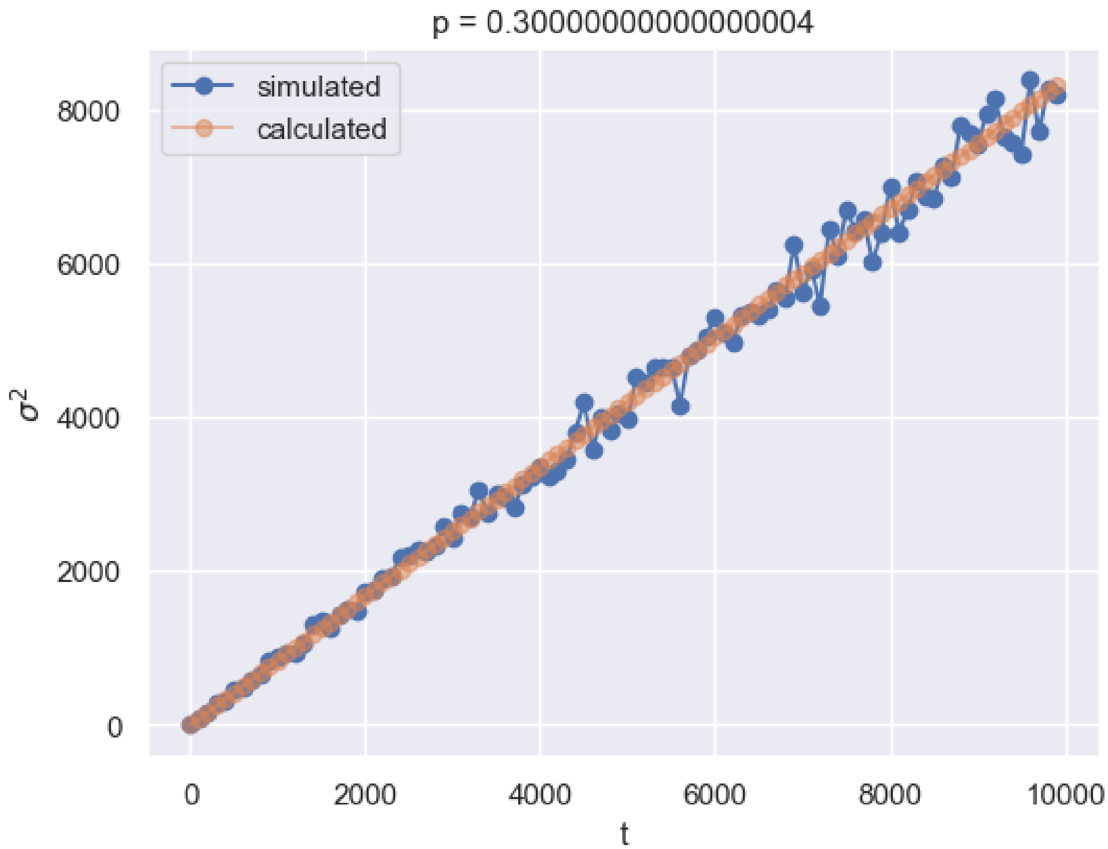
p = 0.2
simulated
calculated
0
−1000
−2000
−3000
−4000
−5000
−6000
<x(t)>
0
2000
4000
6000
8000
10000
t



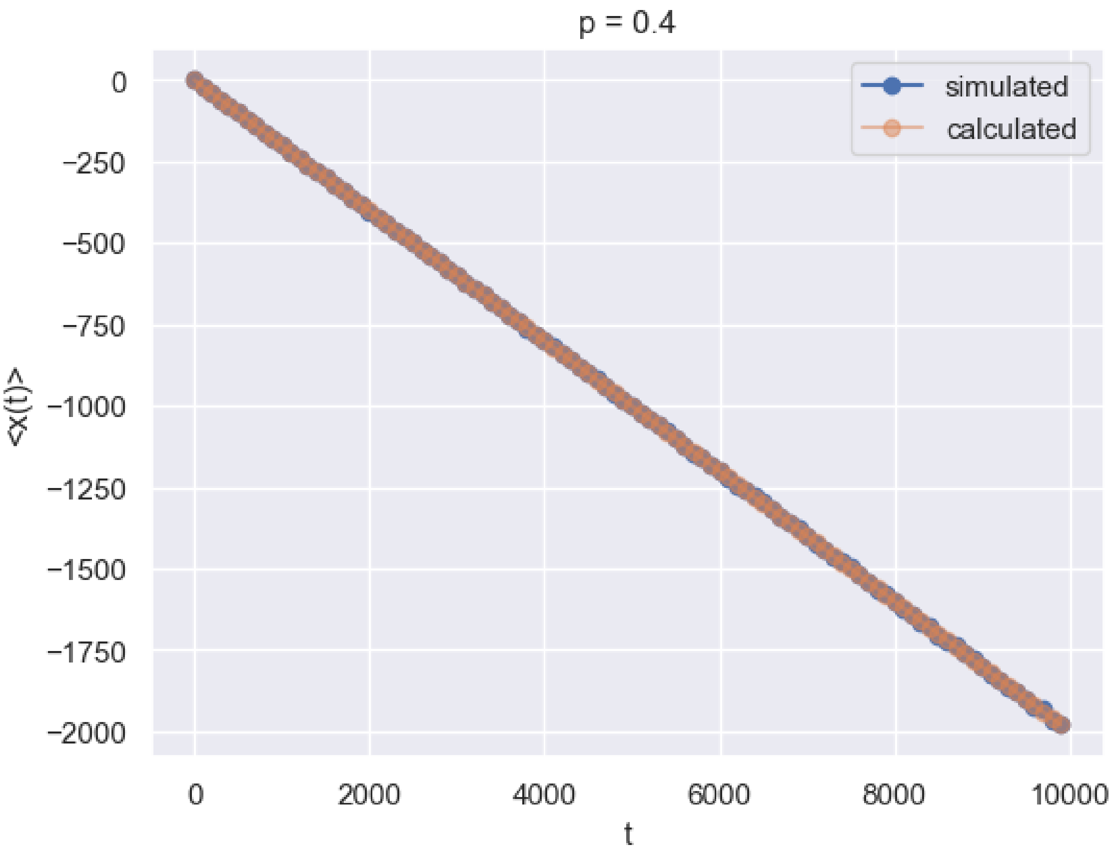
p = 0.2
simulated
calculated
6000
5000
4000
3000
2000
1000
0
σ^2
0
2000
4000
6000
8000
10000
t

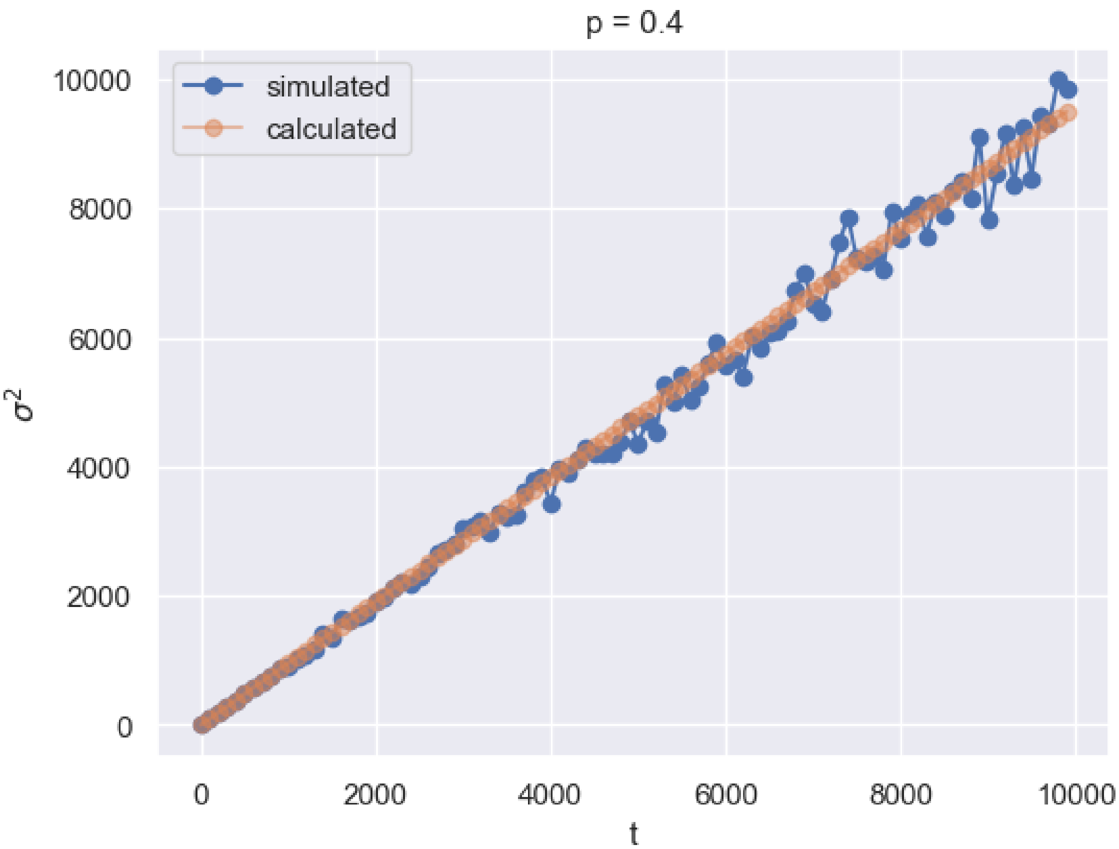


p = 0.30000000000000004
simulated
calculated
0
−500
−1000
−1500
−2000
−2500
−3000
−3500
−4000
<x(t)>
0
2000
4000
6000
8000
10000
t

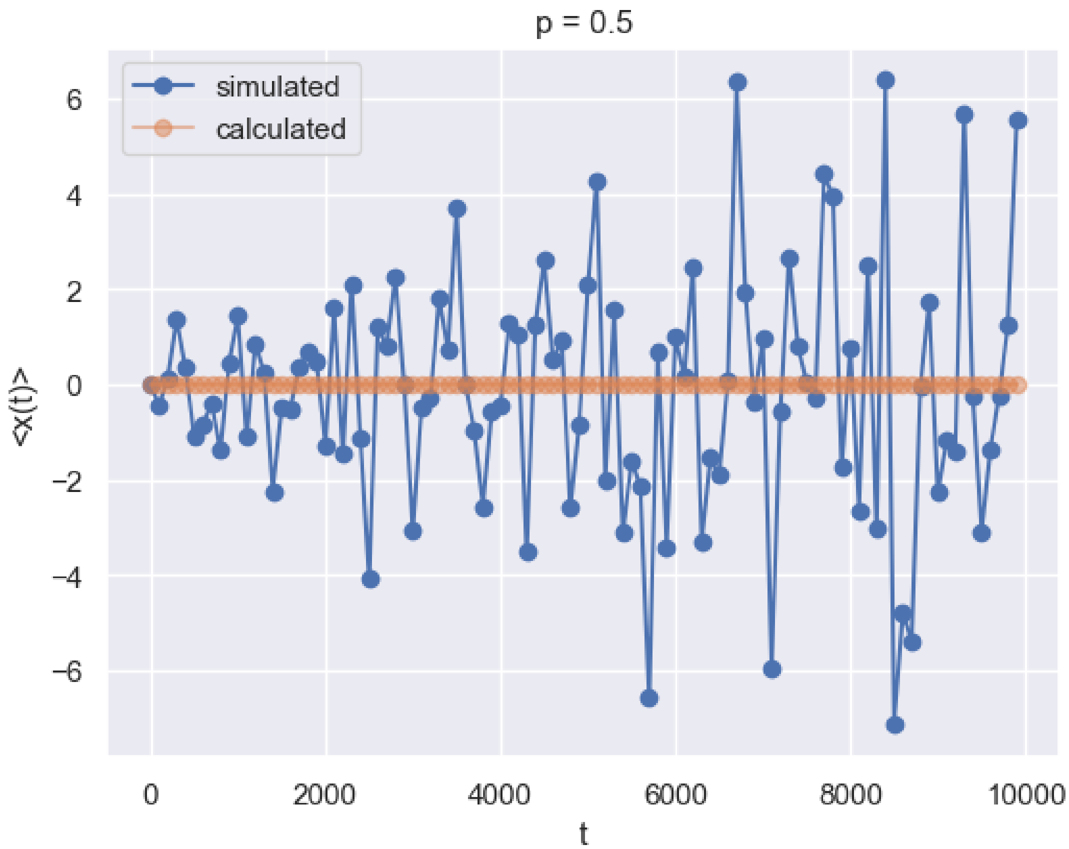


p = 0.30000000000000004
simulated
calculated
σ^2
t
0
2000
4000
6000
8000
10000

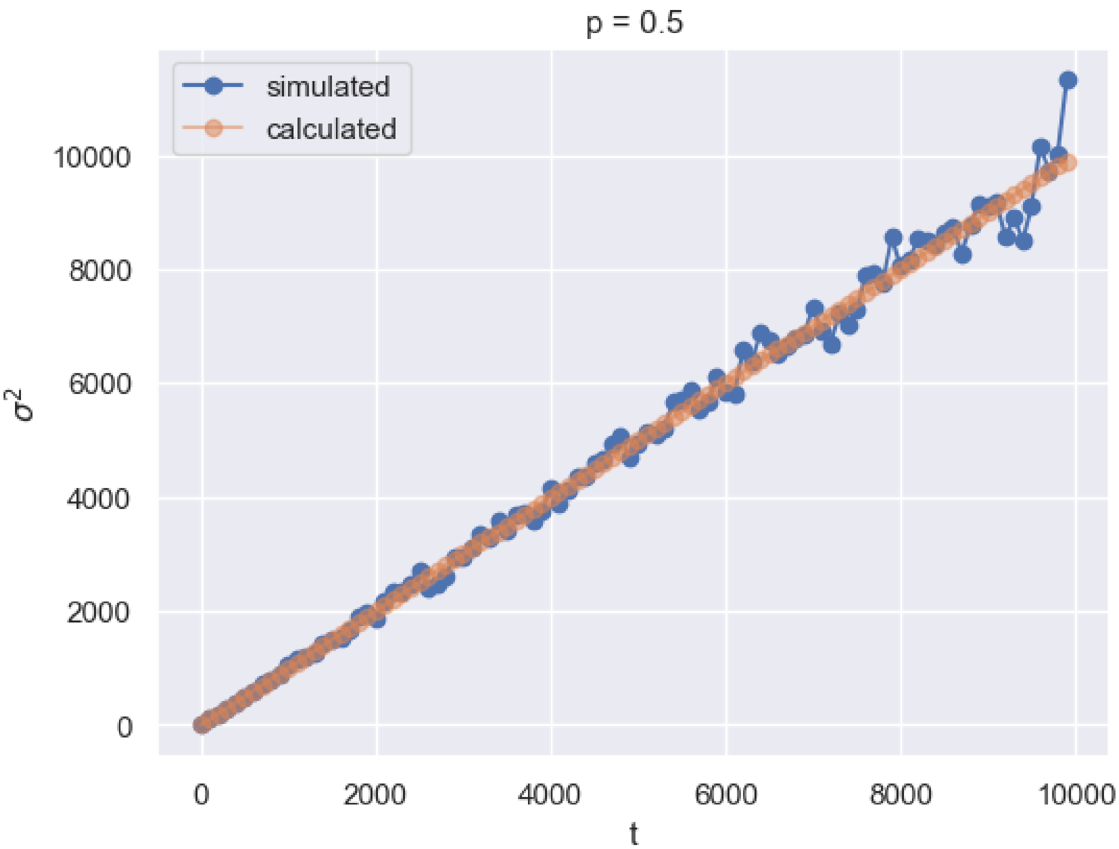

p = 0.4
simulated
calculated
<x(t)>
t
0
−250
−500
−750
−1000
−1250
−1500
−1750
−2000
0
2000
4000
6000
8000
10000

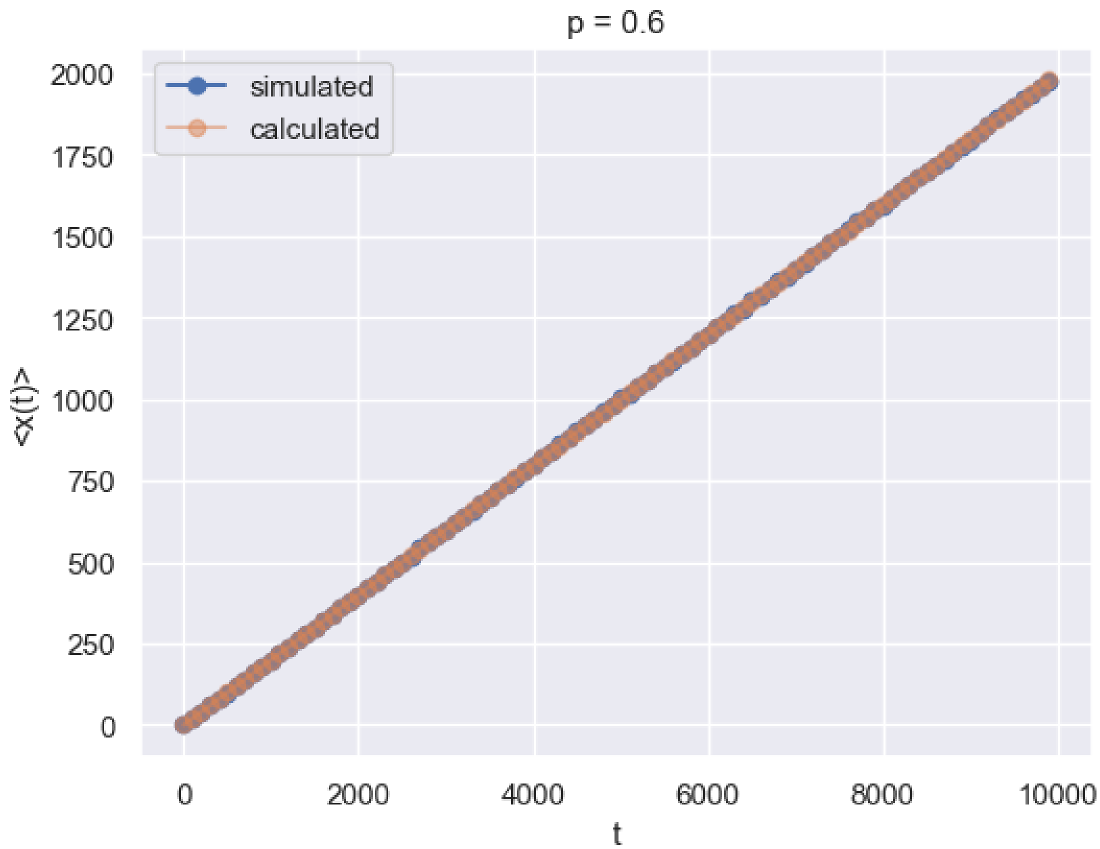


p = 0.4
simulated
calculated
10000
8000
6000
4000
2000
0
σ^2
0
2000
4000
6000
8000
10000
t

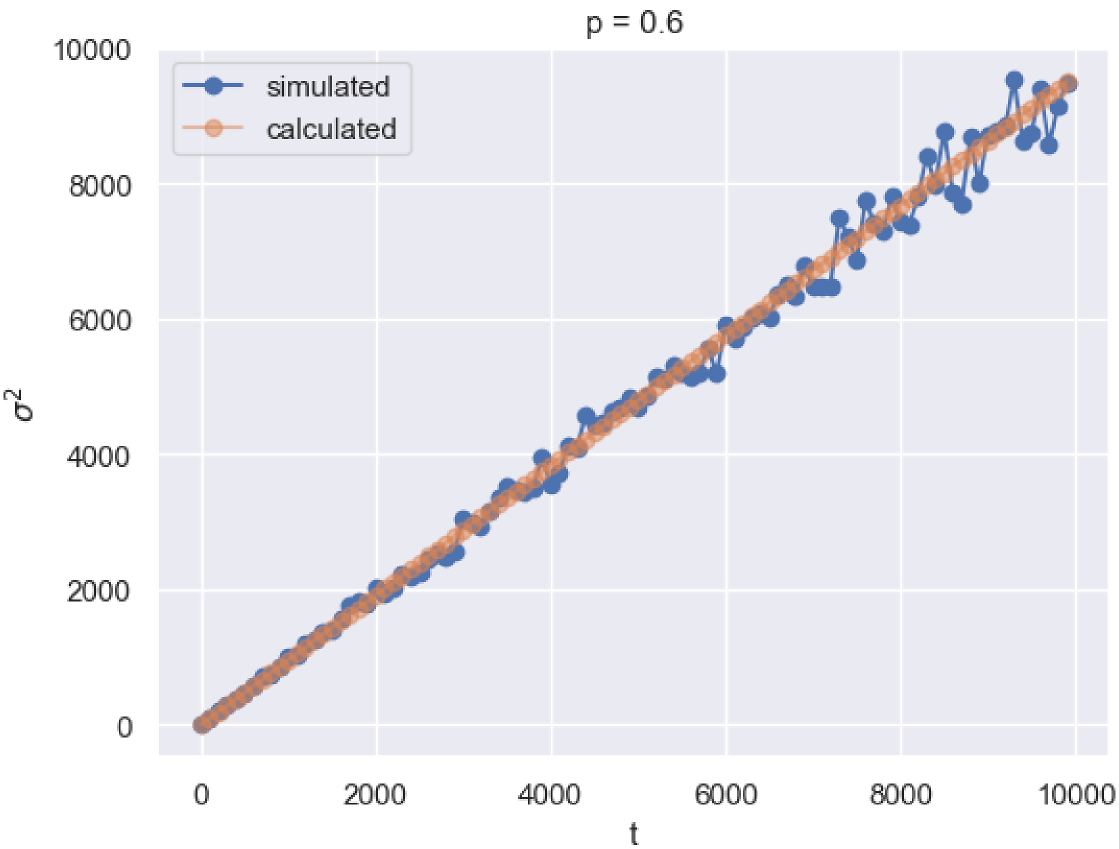


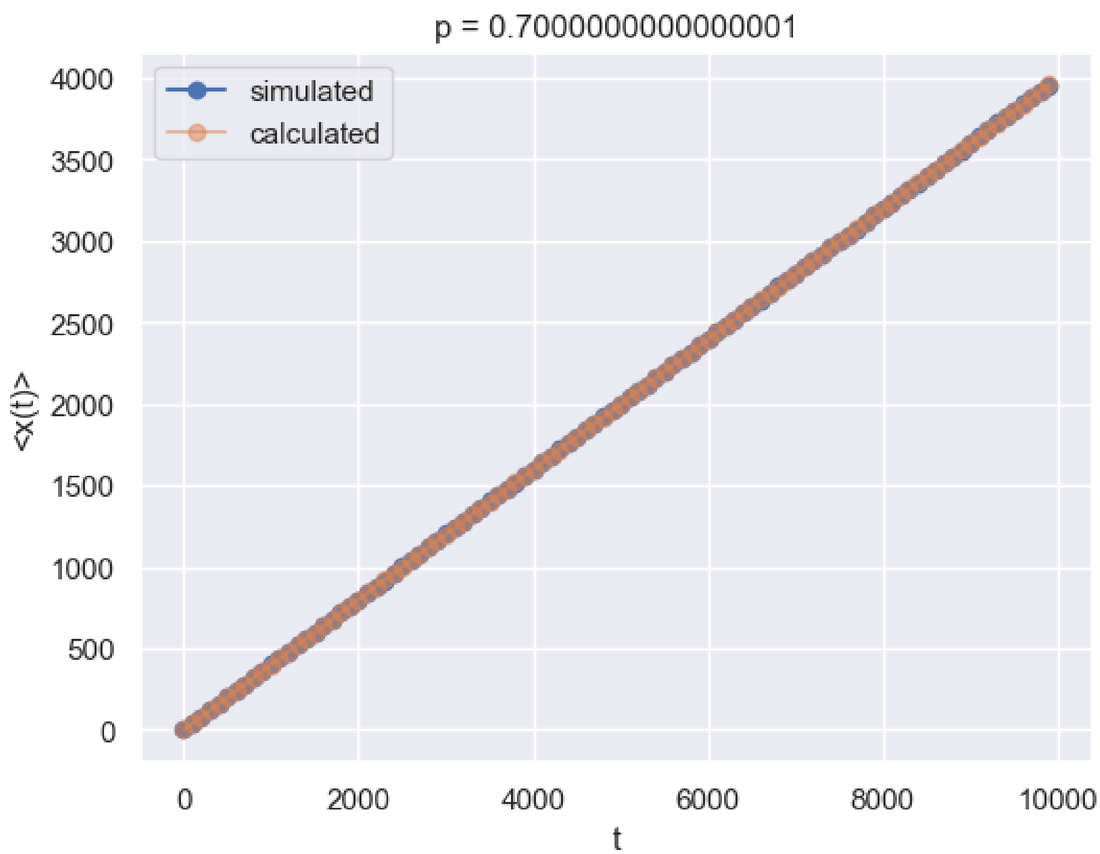
p = 0.5
simulated
calculated
<x(t)>
t
6
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000

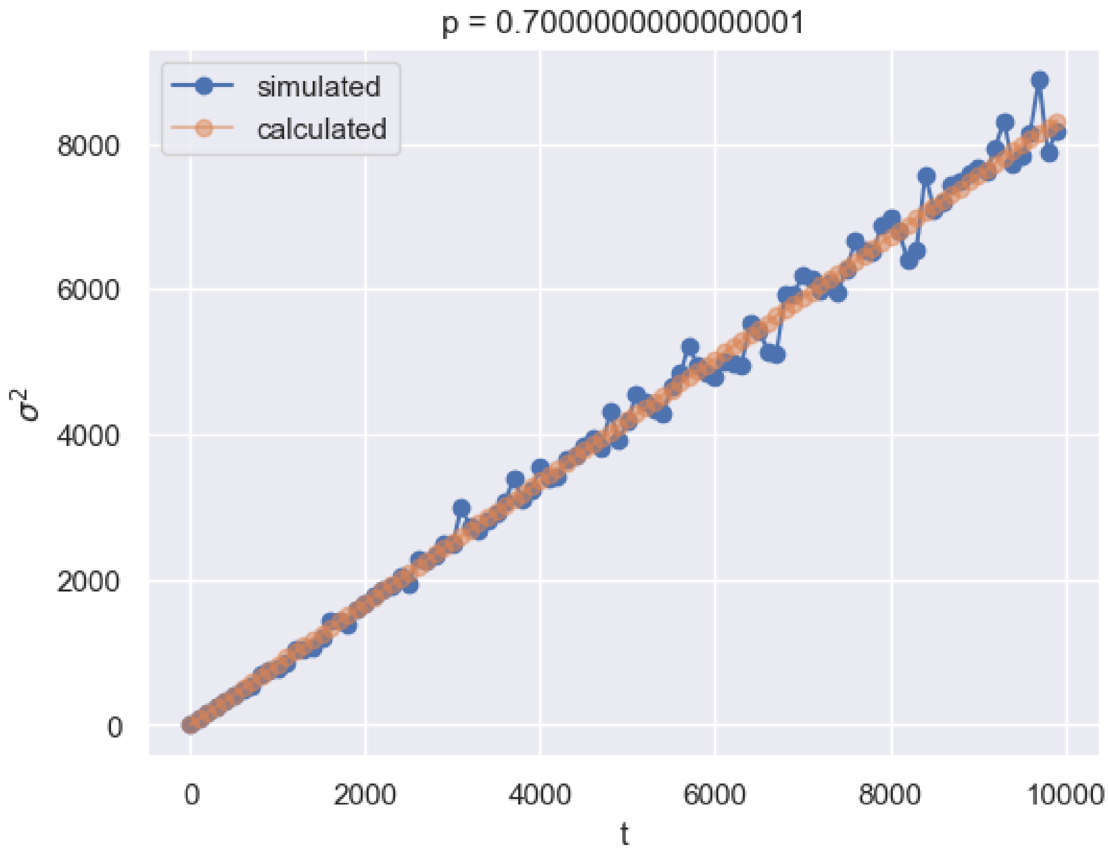

p = 0.5
simulated
calculated
σ^2
t
0
2000
4000
6000
8000
10000

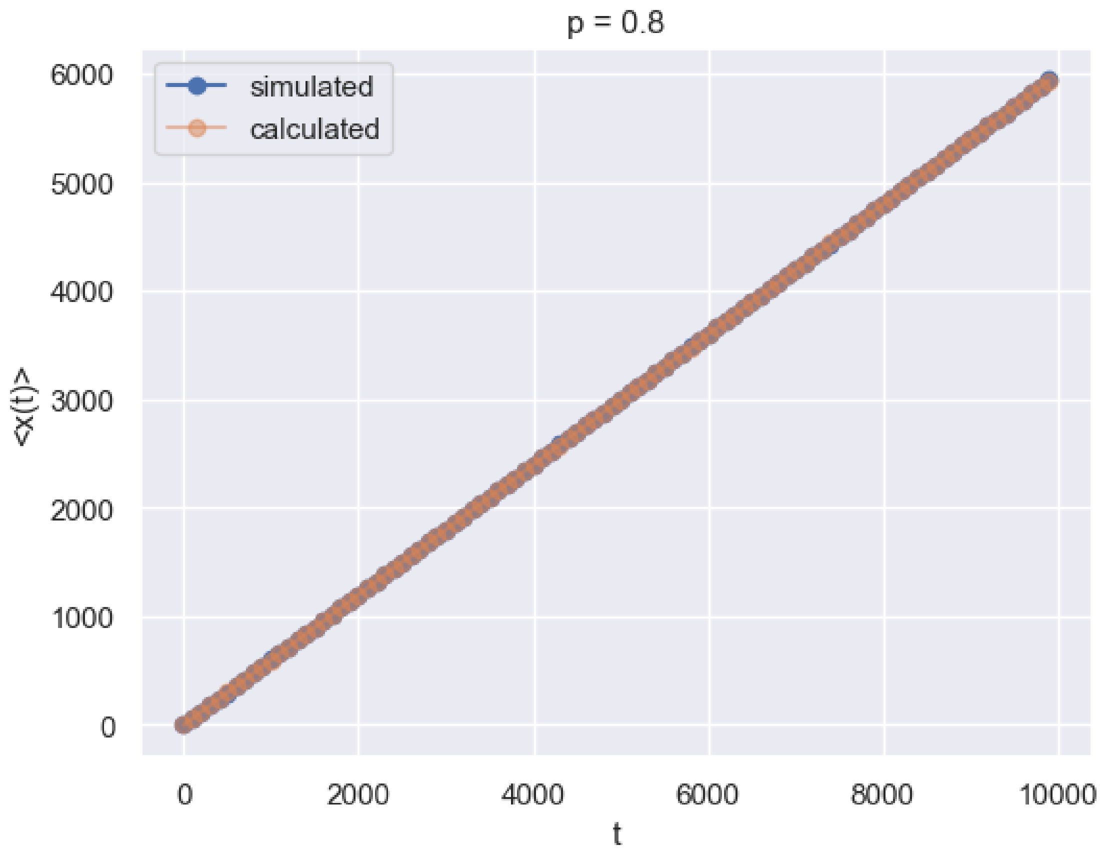


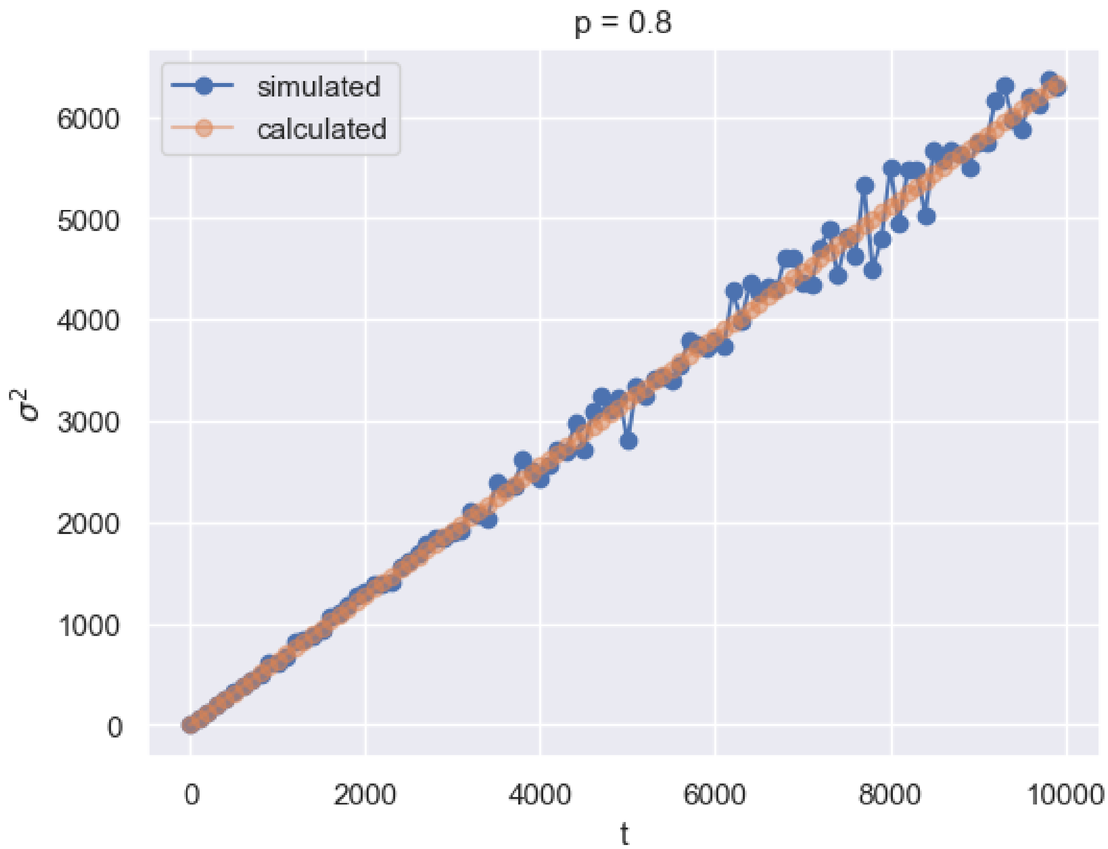
p = 0.6
simulated
calculated
<x(t)>
t
0
250
500
750
1000
1250
1500
1750
2000
0
2000
4000
6000
8000
10000


p = 0.6
10000
8000
6000
4000
2000
0
σ^2
simulated
calculated
0
2000
4000
6000
8000
10000
t

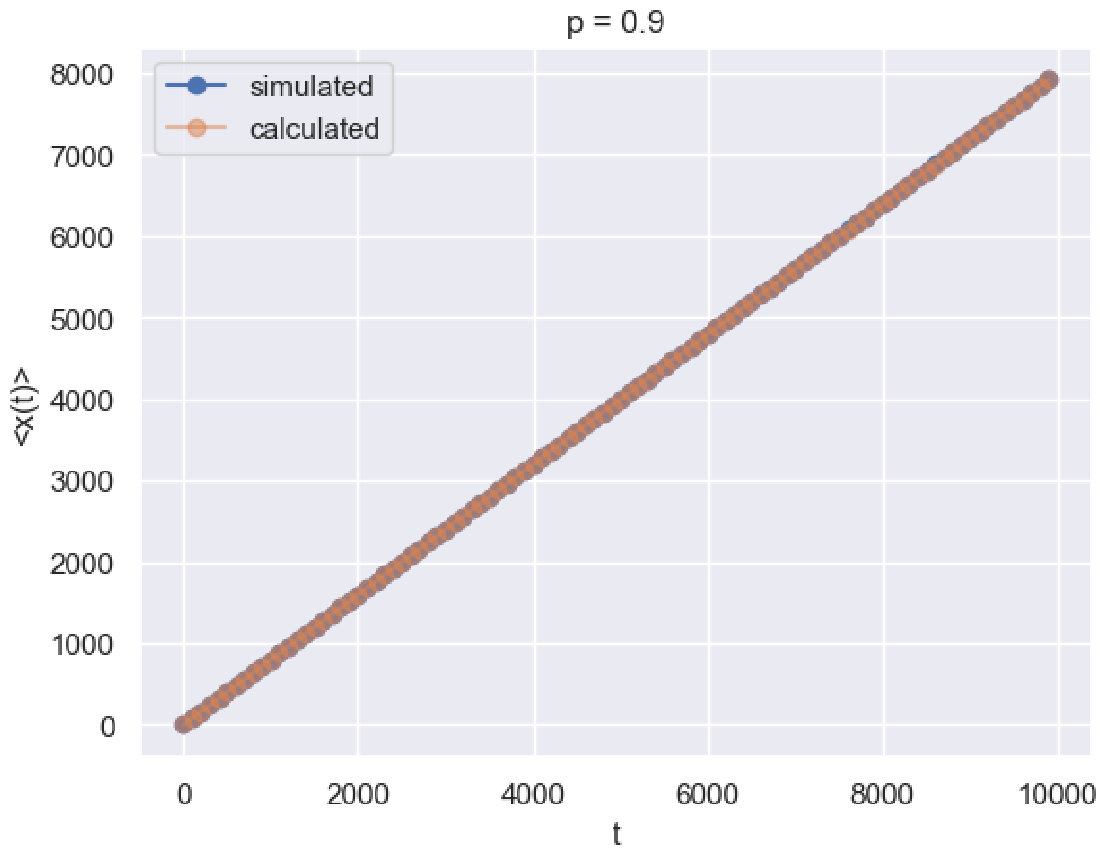

p = 0.7000000000000001
simulated
calculated
<x(t)>
t
0
500
1000
1500
2000
2500
3000
3500
4000
0
2000
4000
6000
8000
10000


p = 0.7000000000000001
simulated
calculated
σ^2
t
0
2000
4000
6000
8000
10000

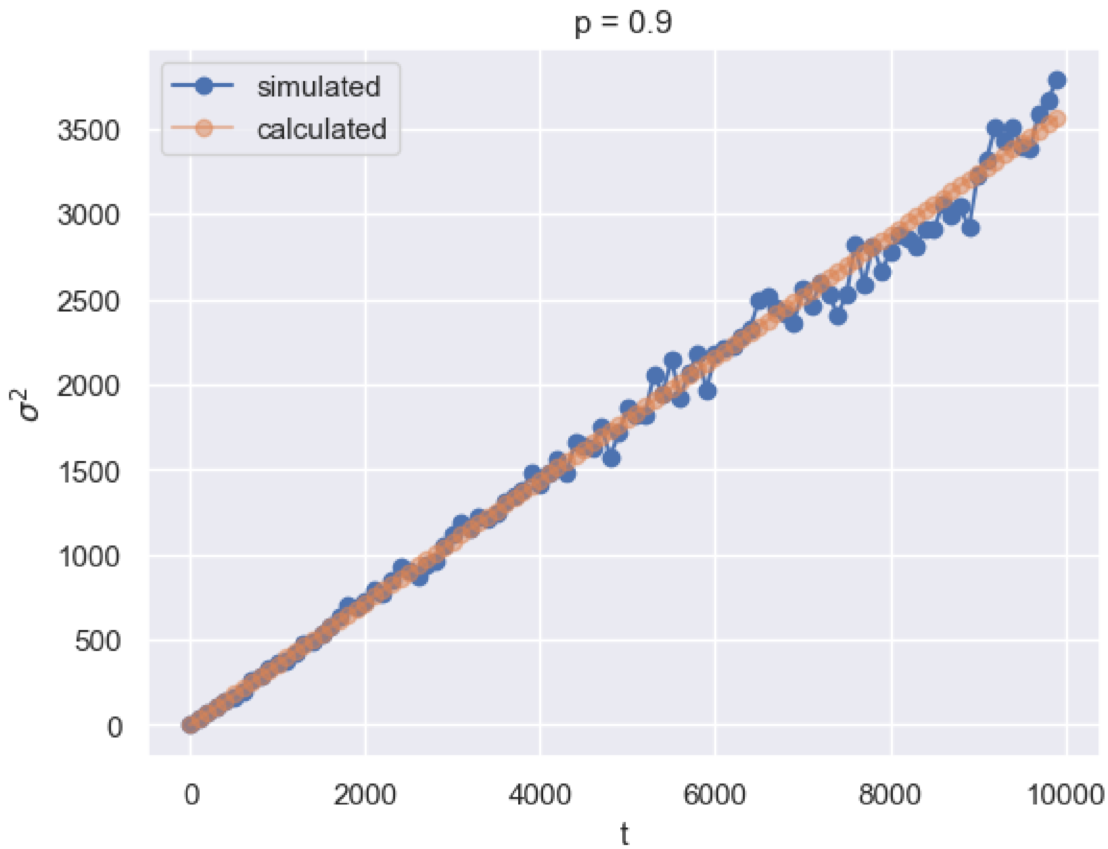

p = 0.8
simulated
calculated
6000
5000
4000
3000
2000
1000
0
<x(t)>
0
2000
4000
6000
8000
10000
t



p = 0.8
simulated
calculated
6000
5000
4000
3000
2000
1000
0
σ^2
0
2000
4000
6000
8000
10000
t



p = 0.9
simulated
calculated
<x(t)>
t
0
1000
2000
3000
4000
5000
6000
7000
8000
2000
4000
6000
8000
10000



می توانیم پا را از این هم فراتر گذاشته و شیب بهترین خط را هم برای هر نمودار بدست آوریم و با مقدار تئوری اش مقایسه کنیم که به علت دقت خیلی زیادی که حد اقل به طور چشمی می بینیم و اینکه در قسمت قبل هم درصد خطا های نسبی را بدست آورده ایم از این کار اجتناب کرده ایم.

مقادیر شبیه سازی شده در همه ی نمودار ها تقریبا به طور دقیق بر مقادیر تئوری منطبق شده اند بجز در p=0.5 که توضیحاتی مثل قبل دارد و در اینجا هم با اینکه زمان های خیلی بزرگی را در نظر گرفته ایم ولی مقدار شبیه سازی شده برای مثال اختلافی حد اکثر شش واحدی با مقدار تئوری اش پیدا کرده.

4 تمرین 4.3

4.1 مقدمه

در این تمرین می خواهیم دوباره رندوم واک یک بعدی را این با شرط مرز جاذب تکرار کنیم. یعنی اگر متحرک از مرز ها گذشت متوقف (کشته) شود.

زمان مرگ هر متحرک را را برای تعداد زیادی اجرا بدست آورده و از آنها میانگین می گیریم و آن را به عنوان متوسط طول زندگی متحرک در این شبکه گزارش می دهیم.

این کار را برای نقاط شروع اولیه ی متفاوتی تکرار کرده و بستگی متوسط طول زندگی متحرک را به مکان شروع زندگی اش نشان می دهیم.

لازم به ذکر است که مرز ها را هم در نقاط 0 و 20 گرفته ایم. (طول شبکه 20 واحد است.) و اگر متحرک در این خانه ها باشد هنوز زنده است ولی اگر بخواهد به خانه های 1- و 21 برود می میرد.

در مورد کد ها هم همه ی جزییات فنی بطور کامنت گذاری در کد توضیح داده شده است و باز هم مطابق سوال قبل فقط به توضیح تابع رندوم واک بسنده می کنیم.

این تابع سه متغیر برای نقطه ی شروع و احتمال به سمت راست رفتن و تعداد تکرار می گیرد و به به ازای این تعداد تگرار ، حلقه ای را تکرار می کند که در آن متغیر از نقطه ی شروع مشخص شده شده کرده و با احتمالی به سمت راست یا چپ می رود و یک واحد به میزان زمان زندگی اش اضافه می شود و سپس چک می شود که آیا متحرک هنوز زنده است یا نه (در مرز ها افتاده است یا نه) و اگر هنوز زنده بود دوباره این حرکت را تکرار می کند و اگر هم نبود که از این حلقه خارج می شویم و تست بعدی را شروع می کنیم.

در نهایت آرایه ای به طول تعداد تکرار مشخص شده ، از طول عمر ها برای این نقطه ی شروع خواهیم داشت.

تابع میانگین این طول عمر ها را به عنوان میانگین طول عمر برای این نقطه ی شروع ، خروجی می دهد.

با تکرار این تابع برای نقطه شروع های مختلف و کشیدن نمودار میانگین طول عمر بر حسب نقطه ی شروع زندگی می توانیم بستگی این دو به هم را نشان دهیم.

4.2 نتایج

4.2.1 ورودی های کاربر:

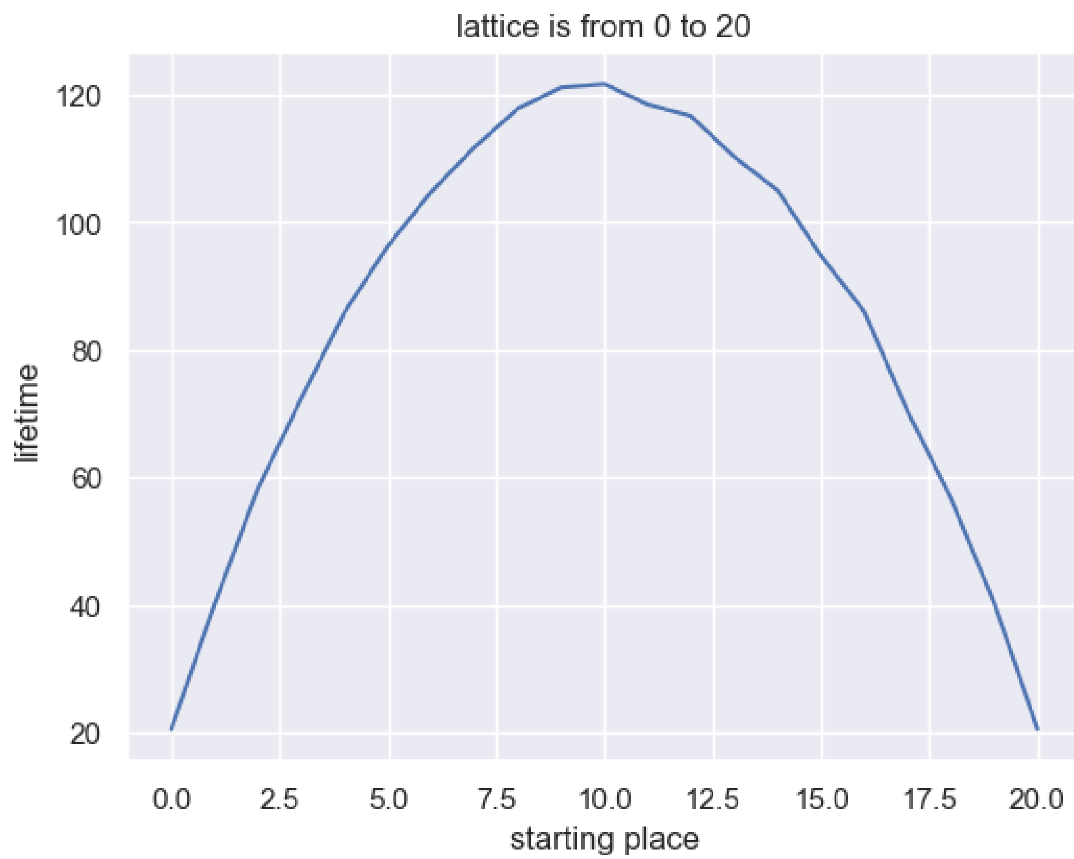
please enter p: 0.5
please enter number of tests: 10000
please enter an array of starting places:
(note that the lattice is from 0 to 20)

np.linspace(0, 20, 21)

4.2.2 خروجی:

در این جا خروجی تصویر شده روی نمودار را آورده ایم و خروجی های عددی مربوط به این نمودار نیز برای استفاده در تمرین بعدی ، به صورت فایل هایی در کنار کد ذخیره شده اند.

همان طور که می شد پیشبینی کرد متحرک اگر با احتمال به راست رفتنی برابر با 0.5 بخواهد شروع کند ، اگر از وسط دو مرز زندگی اش را شروع کند بیشترین عمر میانگین را خواهد داشت. (چون از هر دو مرز دور است.)


lattice is from 0 to 20
120
100
80
60
40
20
lifetime
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
starting place

5 تمرین 4.4

5.1 مقدمه

در این سوال میخواهیم به روش سر شماری (یک روش احتمالاتی) همان مساله ی قبل را حل کنیم و نتایج را با هم مقایسه کنیم.

در این سوال کل این کار را با تابع calculate انجام میدهیم که توضیح آن منجبر به توضیح کل الگوریتم می شود پس فقط به توضیح آن می پردازیم و جزییات فنی بقیه ی بخش های کد را به صورت کامنت گذاری در کد توضیح می دهیم.

برای حل به روش سرشماری نیز مانند حالت شبیه سازی ، ما به انتخاب یک نقطه ی شروع و یک p که همان احتمال به سمت راست رفتن رندوم واکر است احتیاج داریم.

یک آرایه به نام lattice تعریف میکنیم که در آن احتمال های وجود متحرک در لحظه ی مورد بررسی برای همه ی مکان های ممکن در شبکه را ذخیره می کنیم. طبیعی است که در لحظه ی شروع ، احتمال در مکان اولیه برابر با 1 و در بقیه ی مکان ها برابر با صفر باشد.

در قدم های زمانی بعدی آرایه ای به نام new_lattice می سازیم و همه ی مقادیرش را در ابتدا مساوی صفر قرار می دهیم. سپس به ازای هر نقطه در شبکه ی اصلی (lattice) ، به فرض اینکه مقدار عددی اش (احتمال حضور در آن نقطه در مرحله ی قبل) A باشد ، نقطه ی متناظرش در آرایه ی new_lattice را انتخاب کرده و احتمال حضور در خانه ی سمت راستی اش را به اضافه ی pA و احتمال حضور در خانه ی سمت چپی اش را به اضافه ی qA ، که q=1-p است می کنیم.

با تکرار این روش احتمال حضور متحرک را در هر مکان و در هر قدم زمانی بدست می آوریم.

دقت میکنیم که احتمال بودن متحرک در مرز ها در همه ی قدم های زمانی برابر با صفر است چون متحرک بلافاصله بعد از رسیدن به آن نقاط می میرد. پس اینگونه احتمال کل زنده بودن متحرک (جمع احتمال همه ی خانه ها در هر قدم زمانی) در هر قدم زمانی کمتر می شود.

متغیری به نام min_existing_probability_to_lose تعریف کرده و این شرط را قرار می دهیم که اگر احتمال کل زنده بودن متحرک در هر قدم زمانی از این مقدار کمتر شد ، متحرک مرده فرض شود. (این مقدار را در این شبیه سازی 0.37 گرفته ایم.)

نکته ی قابل ذکر دیگر این است که خانه های 1- و lattice_size+1 چاه هستند یعنی اگر متحرک در خانه های 0 و lattice_size باشد هنوز زنده است ولی اگر یک قدم به سمت چاه مجاورش بردارد می میرد.

5.2 نتایج

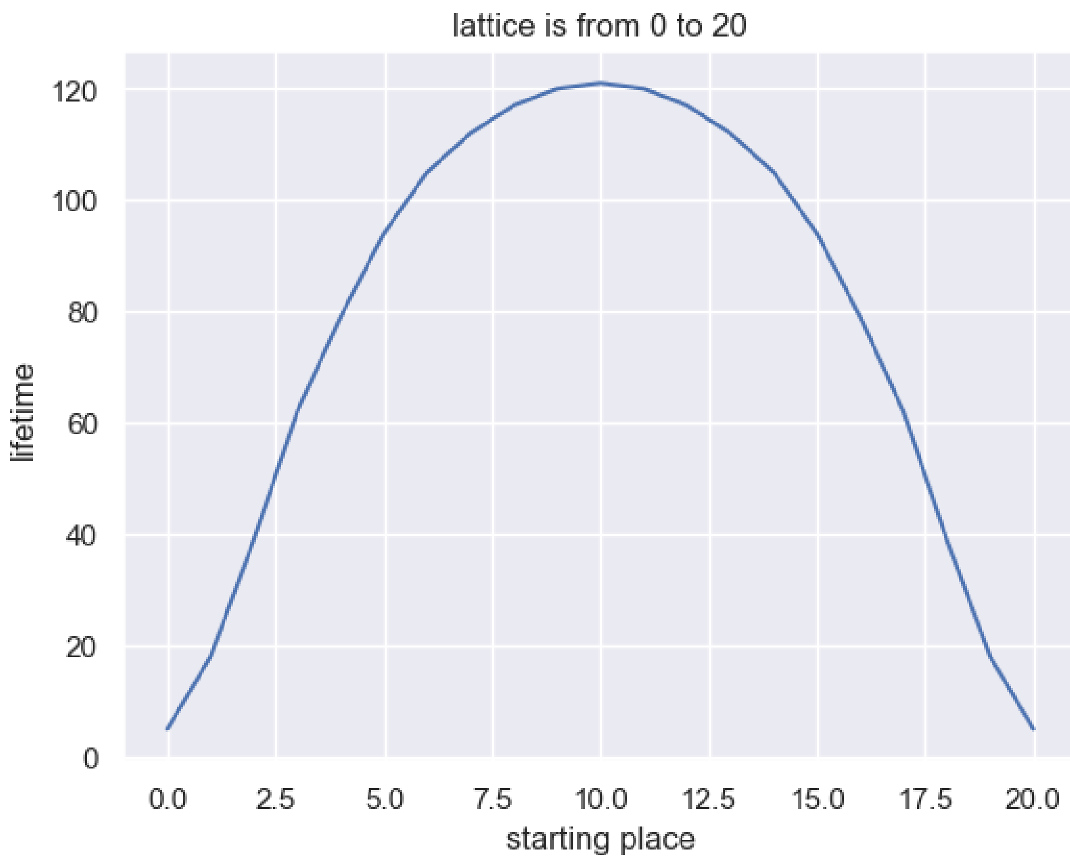
5.2.1 ورودی های کاربر:

please enter p: 0.5
please enter an array of starting places:
(note that the lattice is from 0 to 20)

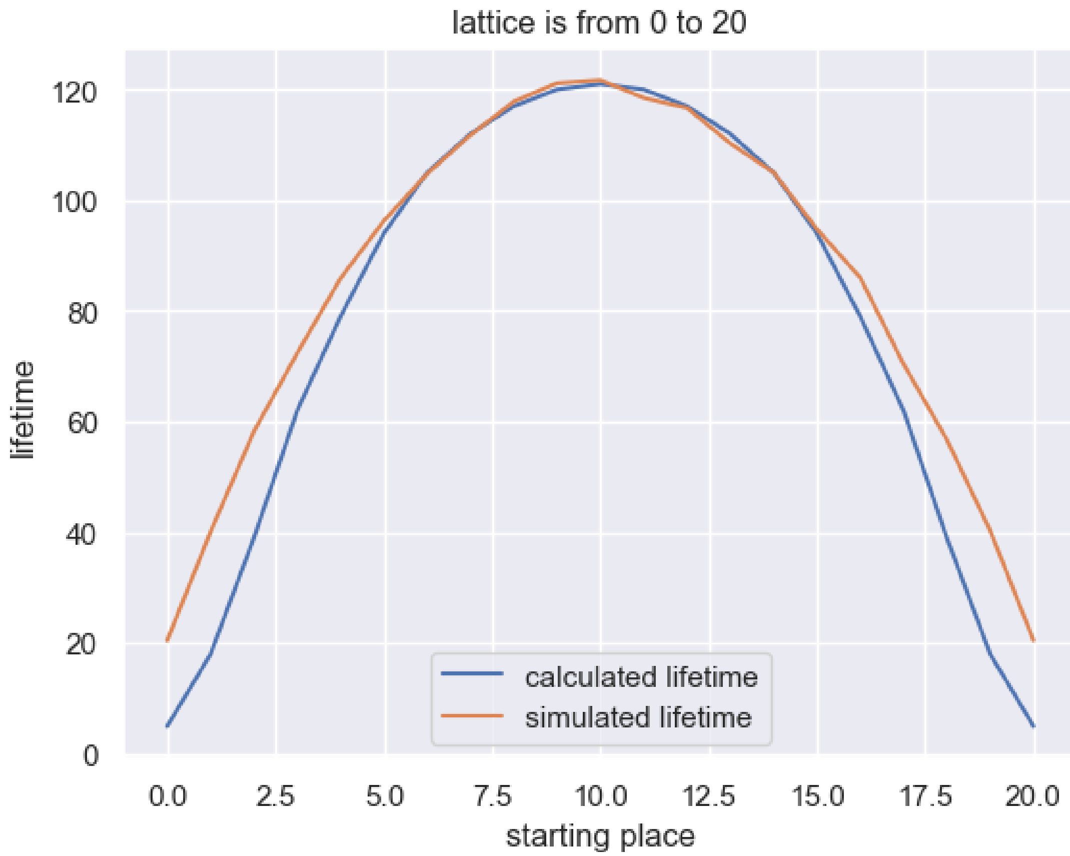
np.linspace(0, 20, 21, dtype=int)

5.2.2 خروجی:

صحت خروجی را می توانیم با یکی از چیز هایی که مطمعنیم تا حدودی بسنجیم ، که آن هم این است که در شبکه کاملا تقارن داریم و طول عمر ها باید در اطراف مرکز دو نقطه ی مرزی به طور مساوی تقسیم شده باشند. (برای مثال طول عمر در خانه ی 0 با 20 و 1 با 19 باید برابر باشد.) که این طور هم شد.



5.2.3 مقایسه ی نتایج این تمرین و تمرین قبلی


lattice is from 0 to 20
lifetime
starting place
calculated lifetime
simulated lifetime
0
20
40
60
80
100
120
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

6 تمرین 4.5

6.1 مقدمه

در این تمرین میخواهیم ولگشت دو بعدی را شبیه سازی کنیم و صحت رابطه ی زیر را تحقیق کنیم.

$$< r^2 > = 2d\, Dt \tag{12}$$

در این تمرین هم هسته ی مرکزی تابع random_walk است ، پس فقط به توضیح این تابع بسنده کرده و بقیه ی جزییات فنی را به طور کامنت گذاری در کد توضیح می دهیم.

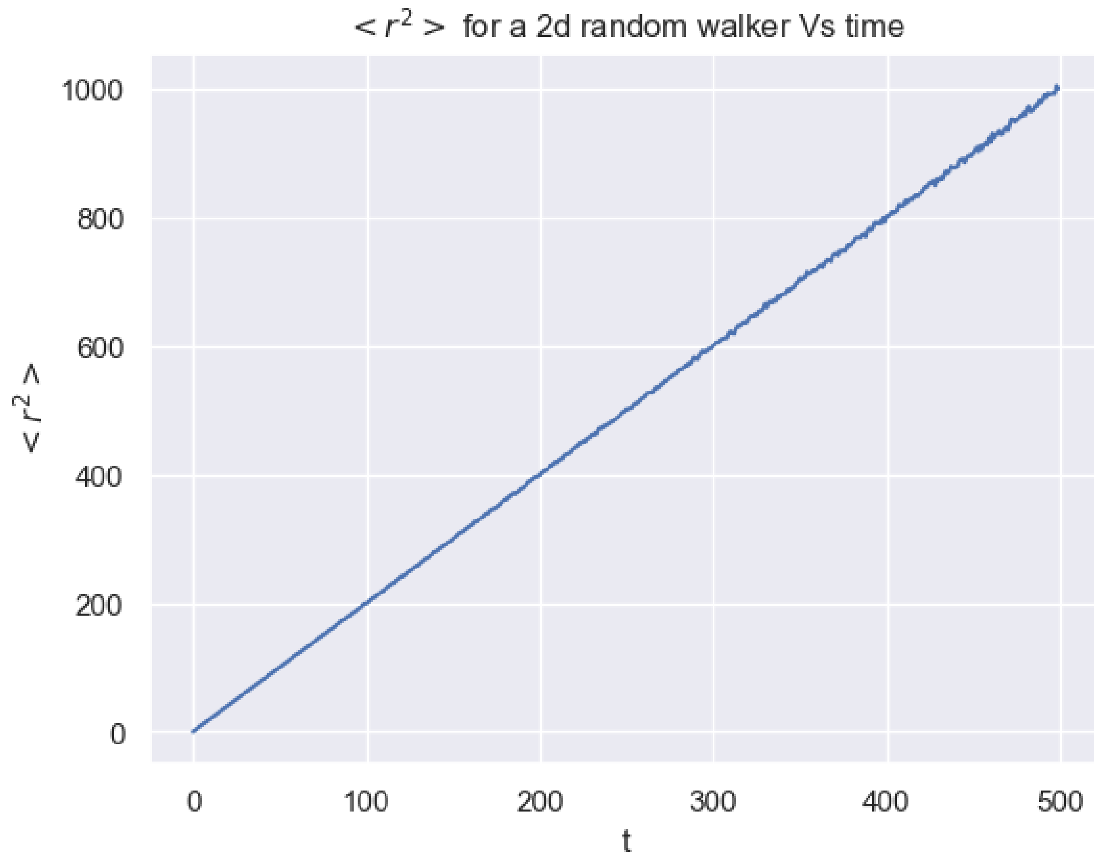
این تابع سه ورودی می گیرد:

1- t_max که همان ماکسیمم زمان شبیه سازی است. یعنی از زمان صفر شروع میکنیم و به ازای همه ی زمان های بعد از آن تا رسیدن به t_max شبیه سازی را انجام می دهیم.
2- P که احتمال به راست یا بالا رفتن است. (این جا طبق خواسته ی سوال هر دو را برابر می گیریم ولی اگر این طور نبود هم تغییر چندانی در الگوریتم و کد حاصل نمی شد.)
3- Number_of_tests که تعداد دفعاتی است که ما میخواهیم هر مرحله از این شبیه سازی را تکرار کنیم.

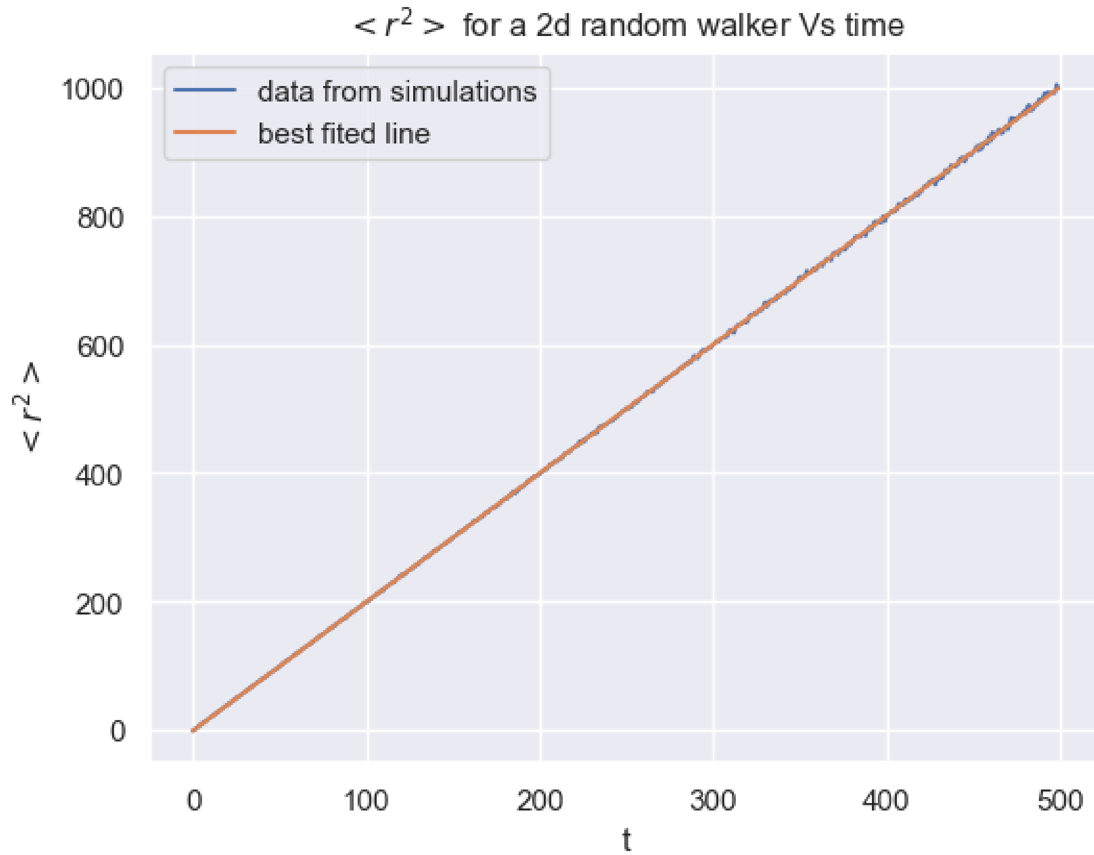
تابع آرایه ای به طول t_max را که هر خانه اش مقدار <r^2> ی مربوط به یکی از تست ها است را بر می گرداند.

عمل شبیه سازی هم بسار آسان است ، در هر قدم مشخص میکنیم که اگر زمان شبیه سازی t ثانیه باشد r^2 چقدر می شود و این کار را به اندازه ی number_of_tests بار انجام داده و سپس از آن ها میانگین میگیریم و می شود <r^2> ی مربوط به این زمان و در نهایت هم همه ی <r^2> ها را در قالب یک آرایه خروج می دهیم.

6.2 نتایج



حال بهترین خط را بر داده ها فیت کرده و شیبش را بدست می آوریم:



the best line is: <r^2> = 2.001t + -0.279
with score: 1.0

برای بررسی رابطه ی 12 خواهیم داشت:

$D = \frac{l^2}{2\tau}$

که قدم های زمانی و مکانی ، هر دو را 1 گرفتیم پس:

D = ½

از طرفی هم چون حرکت دو بعدی است پس:

d = 2

و در نهایت طبق رابطه ی 12 باید داشته باشیم:

<r^2> = 2t

که این همان رابطه ای است که ما در شبیه سازی هم بدست آورده ایم. (بهترین خطی که به نمودار فیت شده)

درصد خطای نسبی شیب تئوری و شبیه سازی = 0.05%

7 تمرین 4.6

7.1 مقدمه

در این تمرین میخواهیم این مورد را شبیه سازی کنیم که اگر یه خط افقی داشته باشیم و دانه را از فاصله ای در بالای آن رها کنیم و آن دانه رندوم واک انجام دهد چه اتفاقی می افتد.

دقت میکنیم که دانه را با فاصله ای مشخص از بلند ترین بوته رها میکنیم و هر بار که بلند ترین بوته بلند تر می شود این فاصله را افزایش می دهیم. همچنین مکان افقی رها کردن دانه هم کاملا رندوم است.

درباره ی شرایط مرزی هم از این شرایط استفاده کردیم که خانه های سر و ته مسیر به همدیگر متصل شده اند.

دانه هر جایی که به هر بوته ای برخورد کند به آن می چسبد.

اگر دانه از یک ارتفاعی بالا تر رفت آن را از دست رفته در نظر می گیریم و سراغ دانه ی بعدی می رویم.

این شبیه سازی دو هسته ی مرکزی دارد که در توابع check_the_neighbours و random_walk پیاده سازی شده اند و در این جا به توضیح همین دو تابع اکتفا می کنیم و بقیه ی جزییات فنی را در قالب کامنت گذاری های موجود در کد توضیح می دهیم.

تابع random_walk دو ورودی می گیرد:

1- P که همان احتمال به سمت راست و یا بالا رفتن است.
2- max_number_of_layers که همان حداکثر تعداد دانه هایی است که میخواهیم بنشانیم. (آنهایی که از دست می روند را در نظر نمی گیریم و به جایشان دانه های جدید می فرستیم.)

این تابع حلقه ای را تکرار می کند که در آن یا باید به تعداد max_number_of_layers لایه نشانده شود و یا طول بلند ترین بوته ی ایجاد شده به طول شبکه برسد.

این تابع در هر اجرای حلقه یک دانه با مختصات رندوم و ویژگی های گفته شده در ابتدای این مقدمه تولید میکند و به ازای هر دانه با کمک تابع check_the_neighbours چک می کند که آیا این نقطه همسایه ای دارد یا نه (که اگر داشته باشد یعنی به نقطه ای برخورد کرده و باید به آن بچسبد.) و یا اینکه از محدوده ی بررسی خارج شده یا نه.

اگر همسایه ای داشت که به آن می چسبد و سراغ ذره ی بعدی می رویم.

اگر از محدوده ی مورد بررسی خارج شده بود که آن را از دست رفت هفرض میکنیم و ذره ی جدیدی را دوباره با مختصات رندوم و ویژگی های گفته شده انتخاب می کنیم و همه ی کار ها را دوباره رویش انجام می دهیم.

اگر هم هیچ کدام از دو مورد بالا نبود به ذره اجازه ی رندوم واک به اندازه ی یک قدم را می دهیم و دو باره همه ی این شرایط را بررسی می کنیم.

تابع check_the_neighbours مقادیر زیر را در ورودی می گیرد:

1- x, y همان مختصات ذره اند.

2- max_height که طول بلند ترین بوته است و از آن برای چک کردن اینکه ذره در منطقه ی مورد بررسی است یا نه استفاده می شود به این صورت که اگر ذره 20 پیکسل بالا تر از این ارتفاع برود ، آن را از دست رفته در نظر می گیریم.)

3- Lattice که شبکه است.

4- Lattice_size که طول شبکه است.

5- number_of_deposited_layers که تعداد دانه هایی است که تا به حال نشانده شده اند و از آن برای رنگ بندی لایه ها استفاده می شود. (به ازای هر 1000 دانه که نشانده شوند رنگ دانه های جدید عوض می شود.)

این تابع فقط چند شرط ساده را بررسی میکند و اگر هر کدام از آنها رخ داده بودند در خروجی اش اعلام می کند.

اگر y ذره بیشتر از 20 پیکسل بالای بلند ترین بوته باشد آن را از دست رفته در نظر گرفته و مقدار "went out" را خروجی می دهد.

در غیر این صورت اگر ذره بالای حد اکثر ارتفاع شبکه باشد که به این معنی است که قطعا هیچ همسایه ای در حال حاضر ندارد مقدار "at the top" را خروجی می دهد که باعث می شود در تابع random_walk ، رندوم واک جدیدی انجام شود و ذره به نقطه ی جدیدی منتقل شود.

و در غیر این دو صورت هم چک می شود که آیا ذره ای در خانه های کناری (بالا ، پایین، چپ و یا راست) این ذره هستند یا نه، که اگر بودند مقدار "sticked" را خروج می دهد که یعنی ذره باید در همین نقطه بچسبد.

اگر هم هیچ کدام از این شرایط حاصل نشدند یعنی ذره در حال حاضر در مکانی از شبکه قرار دارد که هیچ همسایه ای ندارد و هیچ شرط مرزی ای را هم رد نکرده پس تابع مقدار "no neighbours" را خروجی می دهد که باعث انجام رندوم واک جدیدی در تابع random_walk می شود.

7.2 نتایج

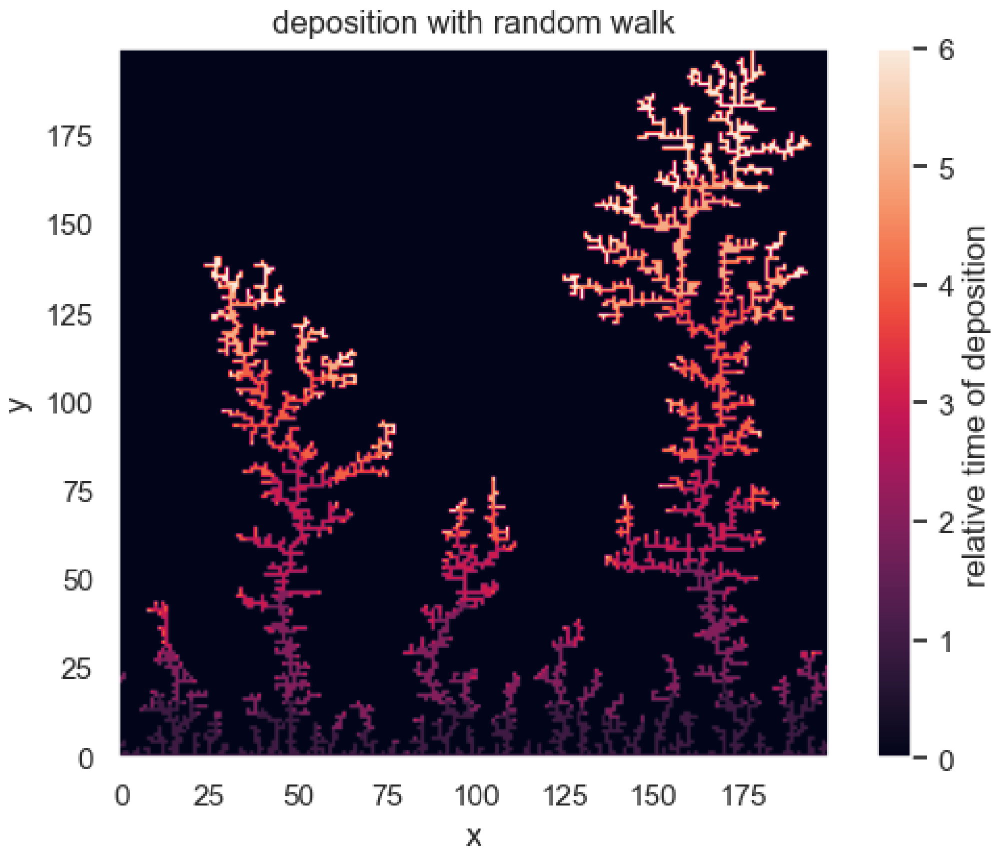
در این تمرین دو بار شبیه سازی را تکرار کردیم.

آزمایش اول:

```
max number of layers:  10000
number of deposited layers:
0                                   10000
```

```
height of the branch with the maximum height:
0                                     198
```

```
*** a branch riched the maximum height ***
number of deposited layers: 5882
```

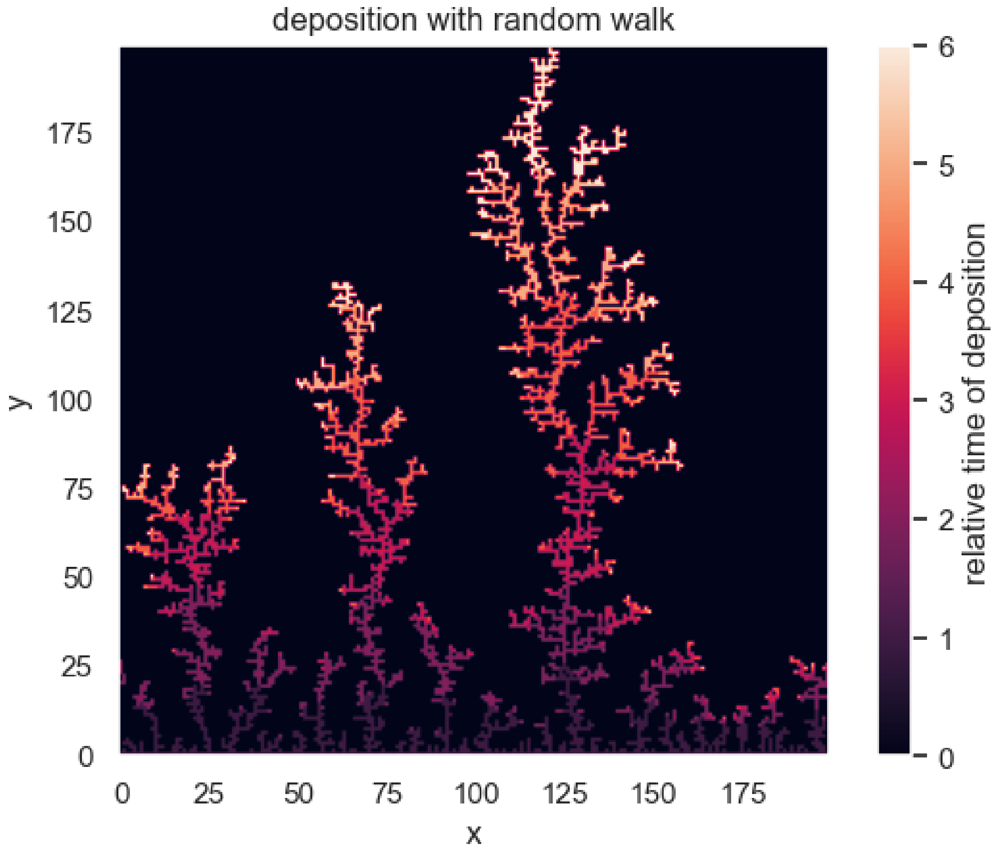


آزمایش دوم:

```
max number of layers:  10000
number of deposited layers:
0                                    10000

height of the branch with the maximum height:
0                                      198

*** a branch riched the maximum height ***
number of deposited layers: 5561
```


deposition with random walk
y
x
0
25
50
75
100
125
150
175
relative time of deposition
0
1
2
3
4
5
6

8 تمرین 4.7

8.1 مقدمه

در این تمرین میخواهیم در ابتدا تمام گشت ها را برای طول N محاسبه کرده و سپس از آنها گشت هایی که خود پرهیز نیستند (از نقطه ای تکراری می گذرند) را حذف کنیم و نمودار های خواسته شده در سوال را رسم کنیم.

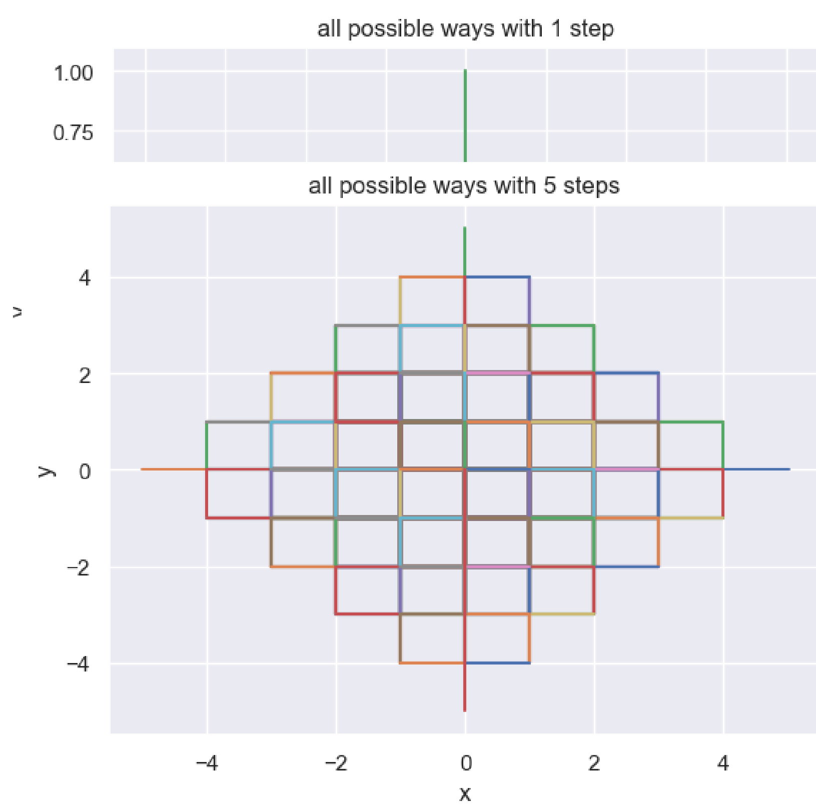
ابتدا در تابع find_all_ways که یک ورودی برای تعداد مسیر ها می گیرد (N) تمان گشت ها را محاسبه می کنیم.

به این صورت که از نقطه ی [0, 0] شروع کرده و در مرحله ی بعد این نقطه را پاک کرده و چهار آرایه به صورت های زیر را جایگزینش می کنیم:

1- آرایه ای شامل این نقطه و نقطه ی سمت راستش یعنی [1, 0]
2- آرایه ای شامل این نقطه و نقطه ی سمت چپش
3- آرایه ای شامل این نقطه و نقطه ی بالایی اش
4- آرایه ای شامل این نقطه و نقطه ی پایینی اش

حال در قدم های بعد هر کدام از این آرایه ها را با چهار آرایه ی دیگر به همین صورت جایگزین می کنیم. این گونه تمام گشت های ممکن به طول N را در یک شبکه ی دو بعدی داریم.

برای تست تابعی که ساخته ایم هم نمودار مسیر ها را برای دو مقدار مختلف N رسم می کنیم که تا حدی از درست کار کردن تابعمان مطمعن شویم.



در مرحله ی بعد با کمک تابع delete_ways_with_repetitive_points گشت هایی که از نقاط تکراری میگذرن را حذف می کنیم.

به این صورت که هر گشت را تبدیل به یک set می کنیم و اگر آن گشت عضو تکراری داشته باشد ، پایتون خود به خود آن را در مجموعه حذف می کند و این گونه طول مسیری که به set تبدیل شه از طول مسیر اصلی کوتاه تر می شود و میتوانیم با مشاهده ی چنین حالتی مسیر های با نقاط تکراری را شناسایی کرده و از مجموعه ی اصلی حذفشان کنیم.

برای تست این تابع هم داریم:

for N=2:

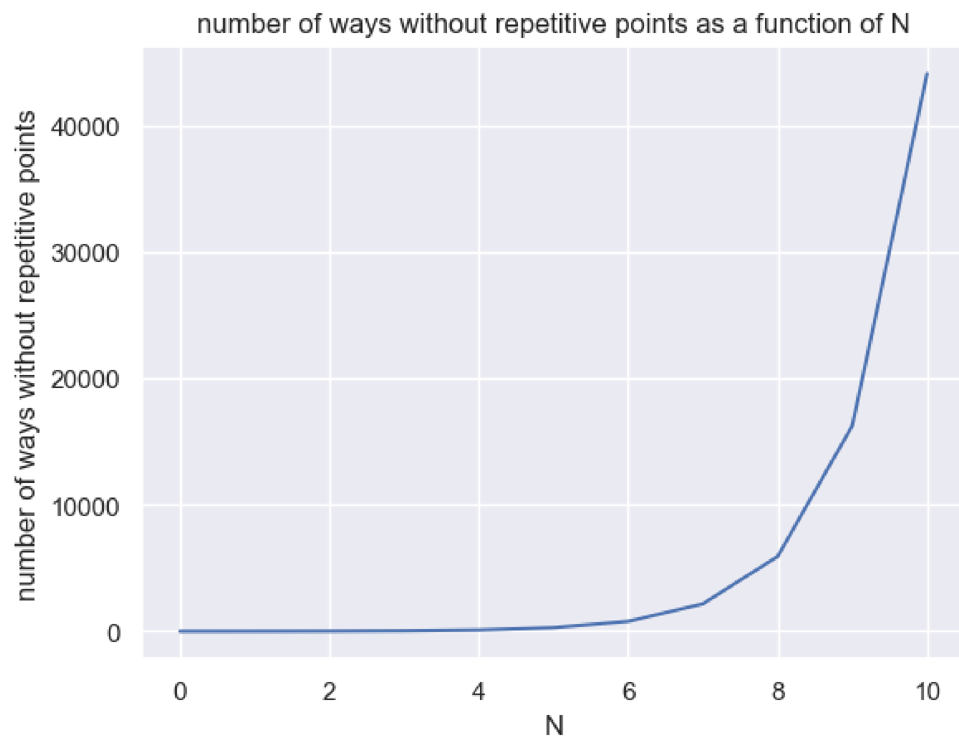
all ways:
[[(0, 0), (1, 0), (2, 0)], [(0, 0), (1, 0), (0, 0)], [(0, 0), (1, 0), (1, 1)], [(0, 0), (1, 0), (1, -1)], [(0, 0), (-1, 0), (0, 0)], [(0, 0), (-1, 0), (-2, 0)], [(0, 0), (-1, 0), (-1, 1)], [(0, 0), (-1, 0), (-1, -1)], [(0, 0), (0, 1), (1, 1)], [(0, 0), (0, 1), (-1, 1)], [(0, 0), (0, 1), (0, 2)], [(0, 0), (0, 1), (0, 0)], [(0, 0), (0, -1), (1, -1)], [(0, 0), (0, -1), (-1, -1)], [(0, 0), (0, -1), (0, 0)], [(0, 0), (0, -1), (0, -2)]]

ways with nonrepetitive points:
[[(0, 0), (1, 0), (2, 0)], [(0, 0), (1, 0), (1, 1)], [(0, 0), (1, 0), (1, -1)], [(0, 0), (-1, 0), (-2, 0)], [(0, 0), (-1, 0), (-1, 1)], [(0, 0), (-1, 0), (-1, -1)], [(0, 0), (0, 1), (1, 1)], [(0, 0), (0, 1), (-1, 1)], [(0, 0), (0, 1), (0, 2)], [(0, 0), (0, -1), (1, -1)], [(0, 0), (0, -1), (-1, -1)], [(0, 0), (0, -1), (0, -2)]]

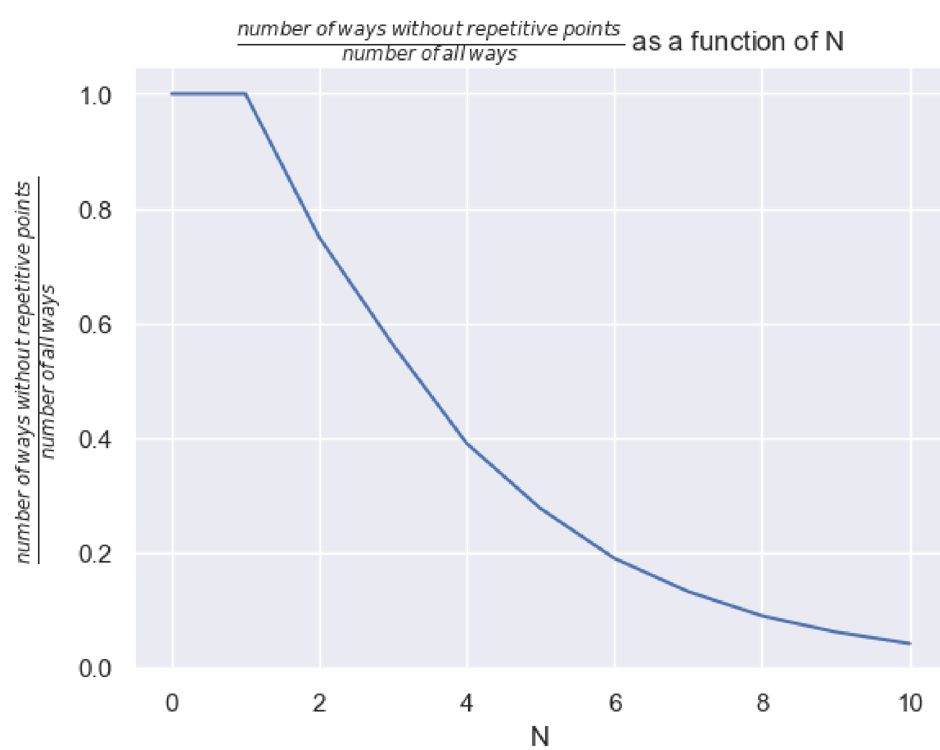
در قسمت بعدی هم تعداد مسیر های با نقاط غیر تکراری را برای N های مختلف می شماریم که به این روش انجام می شود که ابتدا کل مسیر ها را پیدا کرده ، مسیر های با نقاط تکراری را حذف می کنیم و تعداد مسیر های باقی مانده را می شماریم.

در قسمت آخر هم نتایج را در قالب نمودار هایی رسم می کنیم.

8.2 نتایج



number of ways without repetitive points as a function of N
number of ways without repetitive points
0
10000
20000
30000
40000
0
2
4
6
8
10
N



number of ways without repetitive points / number of all ways as a function of N
number of ways without repetitive points / number of all ways
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
N