فیزیک محاسباتی

مجتبی اعلائی

۱۹ آذر ۱۴۰۱

فهرست مطالب

فصل ۱

آموزش پایتون

پایتون [۱] یک زبان سطح بالا و مفسری است که در سال ۱۹۹۱ توسط "خودو فان راسام" [۲] هلندی ایجاد شده است. فلسفه‌ی اصلی این زبان خواناگی و کوتاهی آن است که در هر دو موفق بوده و باعث رواج این زبان در تمام مقاطع و همچین در تمام گرایش‌ها شده است. با گسترش هر چه بیشتر کتابخانه‌های پایتون مانند matplotlib تقریبا با چند خط برنامه کارهایی را می‌توان انجام داد که شاید در زبان‌های دیگر مانند فورترن یا C نیاز به ۱۰ها خط می‌بود. به دلیل مفسری بودن پایتون، سرعت آن نسبت به زبان‌هایی مانند فورترن پایین است با این‌حال به دلیل اینکه نوشت یک برنامه در این زبان زمان انسانی کمتری می‌برد، در کل یک برنامه نویس زمان کمتری برای نوشتن صرف خواهد کرد که به اصطلاح زمان انسانی [۳]و یک برنامه است در حالی که زمان اجرای برنامه بر روی کامپیوتر [۴] یعنی زمانی که کامپیوتر در حال پردازش است زمانی که دز خیلی مواقع از اهمیت کمتری برخورد دارد است.

۱.۱ نصب پایتون

پایتون را می‌توان به راحتی برروی سیستم عامل‌های مختلفی نصب کرد. پایتون دارای دو نسخه اصلی است: پاینون 2.x که در حال حاضر نسخه 2/7 آن موجود است و پایتون 3.x که در حال حاضر نسخه 3/6

[۱] Python
[۲] Guido van Rossum
[۳] Human time
[۴] Computational time

آن در دسترس است. هر دوی این نسخه‌ها به صورت جداگانه در حال توسعه پیدا کرده‌اند ولی در واقع نسخه 2.x تقریبا از توسعه یافتن بازایستاده است و بهتر است از نسخه‌های 3.x استفاده شود. دستورات در دو نسخه تفاوت چندانی ندارد و شاید تنها تفاوت قابل ملموس دستور چاپ (print) است که در پایتون 2.x بدون پرانتز استفاده می‌شود و در پایتون 3.x باید با پرانتز استفاده شود.

به دلیل اینکه کتابخانه‌های پایتون در کوتاه کردن برنامه‌نویسی بسیار موثر هستند بنابراین نصب کتابخانه‌ها پایتون به همراه پایتون غیر قابل اجتناب است. هر کدام از کتابخانه‌های پایتون را می‌توان به طور جداگانه نصب کرد ولی به دو دلیل ممکن است در نصب این کتابخانه‌ها به مشکل برخورد کنیم. اول اینکه ممکن است برای نصب یک کتابخانه، کتابخانه‌های وابسته دیگری نیز باید نصب شوند و دوم اینکه در نصب یک کتابخانه باید سازگاری نسخه‌های کتابخانه با همدیگر و همچین با نسخه‌ی پایتون نصب شده رعایت شود. برای حل این مشکلات پُروژه‌ی Anaconda طراحی شده است. کافی که با ورد به سایت این پروژه:

https://www.anaconda.com/

بسته‌ی Anaconda را نصب کنید. این بسته برای سه سیستم عامل رایج یعنی لینوکس،ویندوز و مک بر روی وب سایت پُروژه موجود است. با نصب Anaconda علاوه بر نصب پایتون، تمام کتابخانه‌های مورد نیاز برای این درس نصب می‌شود. همچین محیط برنامه نویسی Spyder ،که مناسب برای پایتون است، نیز نصب می‌شود.

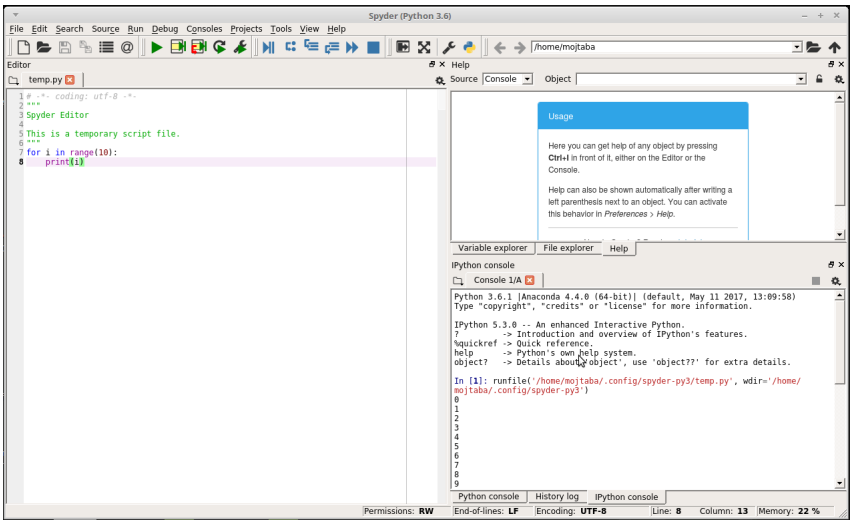
۲.۱ محیط‌های مناسب برای برنامه‌نویسی با پایتون

از چند روش می‌توان یک برنامه پایتون را نوشت و اجرا کرد. یکی از این روش‌ها استفاده از IDE [۵] یا محیط توسعه مجتمع است. در واقع یک IDE یک واسط گرافیکی است که در آن می‌توان علاوه بر برنامه‌نویسی، به عیب‌یابی [۶] برنامه و اجرای آن و همچنین مدیریت یک پُروژه، که شامل چند برنامه مرتبط به هم است، پرداخت. در IDE معمولا اکثر دستورات در حفظ آن قرار دارد و با نوشت حرف اول کلمات کلیدی برای کامل شدن دستور دیگر نیاز به تایپ کامل آن نیست و با کلید Tab یا با کمک موس می‌توان دستور را کامل کرد. معروف‌ترین IDE های برای پایتون به شرح زیر است:

[۵] Integrated Development Environment
[۶] debugging

- IDLE
- WingIDE
- Spyder
- PyCharm

IDLE محیط برنامه‌نویسی خود پایتون است به همراه نصب پایتون نصب می‌شود که البته خیلی کارآمد نیست. از میان IDE ها کامل‌تر Spyder رایگان است و نسبتا IDE کامل و مناسب است. شکل ۱.۱ تصویر از محیط Spyder را نشان می‌دهد. PyCharm یک نسخه رایگان در اختیار علاقمندان به

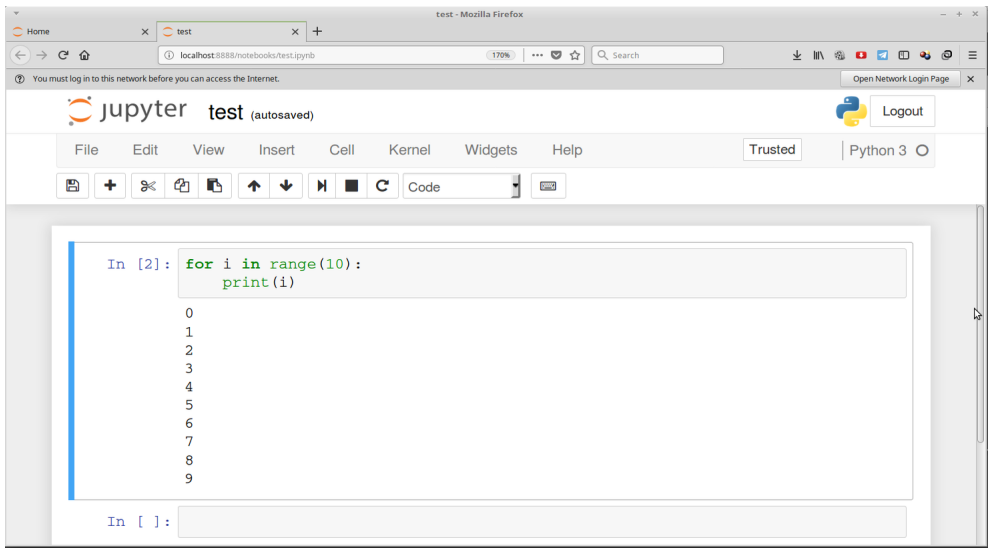


شکل ۱.۱

پایتون قرار داده است منتها نسخه اصلی آن که نیاز به پرداخت هزینه است کامل‌تر و کارآمدتر است.

روش دوم استفاده از محیط ipython به همراه محییر نوت بوک [۷] است که در حال حاضر به نام Jupyter معرف شده است. محیط Jupyter مانند محیط‌های IDE های کامل نیست منتها ابزارهای مناسبی را در اختیار نویسنده قرار می‌دهد. این محیط از مروگرهای ایترنتی مانند Firefox به عنوان رابط گرافیکی استفاده می‌کند. شکل ۲.۱ تصویری از این محیط را نشان می‌دهد. روش سوم استفاده از ویرایش‌گرهایی مانند Vim یا Emace در سیستم عامل‌های لینوکس و مک و یا ++Notepad در سیستم

[۷] notenook



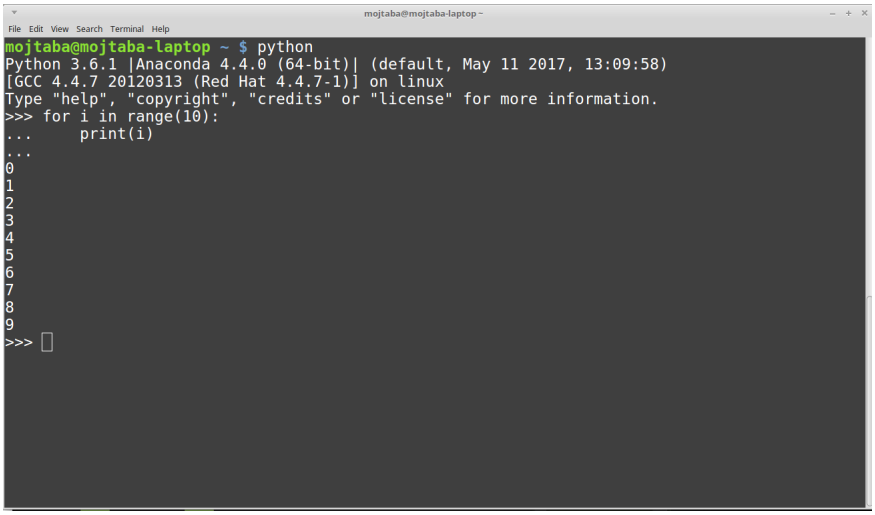
شکل ۲.۱

عامل ویندوز است. در واقع این ویرایش‌گرها نیز محیط‌های مناسبی برای برنامه‌نویسی محیا کرده‌اند. برای مثال دستورات پایتون در این ویرایشگرها شناخته می‌شوند و به صورت رنگی از بقیه متن متمایز می‌شوند. بعد از نوشتن برنامه می‌توان برنامه را به صورت مستقیم در یک شل [۸] اجرا کرد.

روش دیگر که پایتون در اختیار کاربران قرار داده است استفاده از شل پایتون است که دستورات را به راحتی می‌توان در آن، به صورت مستقیم، اجرا کرد. منتها عملا کارایی این شل بیشتر برای چک کردن و امتحان کردن دستورات و نوشتن برنامه‌های چند خطی است. تصویر ۳.۱ شل پایتون را در لینوکس نشان می‌دهد.

نویسنده استفاده از Jupyter یا Spyder در تمام سیستم‌عامل ها و در لینوکس استفاده از ویرایشگرهای قوی مانند Vim را توصیه می‌کند.

[۸] shell


mojtaba@mojtaba-laptop ~
File Edit View Search Terminal Help
mojtaba@mojtaba-laptop ~ $ python
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:09:58)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(10):
...   print(i)
...
0
1
2
3
4
5
6
7
8
9
>>>

شکل ۳.۱

کتاب‌نامه

فصل ۲

حل معادلات دیفرانسیل معمولی

۱.۲ روش اویلر

در فیزیک معمولا قوانین با معالات دیفرانسیل مطرح می‌شوند. تعداد کمی از این معادلات حل تحلیلی دارند. برای مثال می‌توان به معادله واپاشی هسته‌ها اشاره کرد:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = -\frac{N}{\tau} \qquad (۱.۲)$$

که N تعداد هسته‌هایی را نشان می‌دهد که در زمان t هنوز واپاشی نکرده‌اند. τ نیز معیاری از مدت زمان واپاشی را نشان می‌دهد. در حقیقت τ زمان عمر میانگین یک هسته‌ی رایواکتیو قبل از واپاشی است. این معادله بسیار ساده است و در واقع حل این معادله را به صورت تحلیلی می‌دانیم:

$$N(t) = N(۰)e^{-t/\tau} \qquad (۲.۲)$$

حال اگر به فرض به خواهیم چنین معادله‌ای را به صورت عددی حل کنیم باید چه کنیم؟

در ابتدا باید درک درستی از مفهوم حل‌های عددی در ذهن داشته باشیم. اصولا اکثر مدل‌ها و معادلات در علم حل دقیقا ندارند و ناچار باید به سراغ حل‌های تقریبی و عددی رفت. در تمام حل‌های عددی معادلات، معمولا چندین متغیر به عنوان وردی و در خروجی جواب معادله را خواهیم داشت. برای

مثال تابع $x(t)$ را در نظر بگیرید که تابعی از متغیر t است. معمولا مسئله‌هایی که در فیزیک با آن مواجه هستیم به صورت یک معادله دیفرانسیلی خواهد بود (همانند آنچه در واپاشی دیدید):

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x,t) \qquad (۳.۲)$$

برای حل چنین معادلاتی، به راحتی به تعریف مشتق می‌توان مراجعه کرد. مشتق تابع $x(t)$ نسبت به t به صورت زیر بیان می‌شود:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} \equiv \lim_{\Delta t \to ۰} \frac{x(t+\Delta t) - x(t)}{\Delta t} \Longrightarrow \frac{x(t+\Delta t) - x(t)}{\Delta t}\ , \qquad (۴.۲)$$

روش اویلر در حقیقت استفاده از این تعریف است. بنابراین طبق روش اویلر جواب در $t+\Delta t$ به صورت تقریبی به فرم زیر در خواهد آمد:

$$x(t+\Delta t) \approx x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}\Delta t\ . \qquad (۵.۲)$$

پس طبق روش اویلر اگر مشتق تابع $x(t)$ را نسبت به t داشته باشیم می‌توانیم مقدار این تابع را در زمان $t+\Delta t$ نیز داشته باشیم. به همین ترتیب می‌توانیم جواب را در $t+۲\Delta t$ و همین طور در $t+۳\Delta t$ و ...بدست آوریم:

$$\begin{aligned} x(t+\Delta t) &\approx x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}\Big|_t \Delta t \qquad (۶.۲)\\ x(t+۲\Delta t) &\approx x(t+\Delta t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}\Big|_{t+\Delta t} \Delta t \\ x(t+۳\Delta t) &\approx x(t+۲\Delta t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}\Big|_{t+۲\Delta t} \Delta t \\ \vdots &\quad \vdots \end{aligned}$$

بنابراین حل معادله واپاشی هسته‌ها به صورت زیر خواهد بود:

$$\begin{aligned} N(t+\Delta t) &\approx N(t) - \frac{N(t)}{\tau}\Delta t \\ N(t+۲\Delta t) &\approx N(t+\Delta t) - \frac{N(t+\Delta t)}{\tau}\Delta t \\ N(t+۳\Delta t) &\approx N(t+۲\Delta t) - \frac{N(t+۲\Delta t)}{\tau}\Delta t \\ \vdots &\quad \vdots \end{aligned} \tag{۷.۲}$$

در حل عددی باید شهود خوبی در مورد مفهوم مش‌بندی داشته باشیم. در یک حل عددی سعی در این است مقدار مثلا تابع $x(t)$ در نقاط

$$t_۰, t_۰ + \Delta t, t_۰ + ۲\Delta t, t_۰ + ۳\Delta t, \ldots \tag{۸.۲}$$

محاسبه شود. بنابراین ما نیاز به استفاده از متغیرهای آرایه‌ای (و یا لیست) داریم. پس برای متغیر t و هم برای متغیر x باید یک آرایه (یا لیست) در نظر گرفته شود. برای مثال اگر مقدار Δt را برابر با ۰/۱ در نظر گرفته شود و بخواهیم تابع x را از $t = ۰$ تا $t = ۲$ محاسبه کنید، در پایتون لیست متغیر t به چندین روش زیر قابل تولید است:

```
ti=0
tf=2.0
dt=0.1
#method 1
t=ti
tlist=[ti]
while t<tf:
    tlist.append(t)
    t=t+dt
print(tlist)

#method 2
n=int((tf-ti)/dt)
tlist=[]
for i in range(n):
    t=ti+i*dt
    tlist.append(t)
print(tlist)
```

```
19
#method 3
21 n=int((tf-ti)/dt)
tlist=[0]*n
23 for i in range(n):
    tlist[i]=ti+i*dt
25 print(tlist)
```

: tlist.py

خروجی لیستی به صورت زیر خواهد بود:

```
1 [0.0, 0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6000000000000001,
   0.7000000000000001, 0.8, 0.9, 1.0, 1.1, 1.2000000000000002, 1.3,
   1.4000000000000001, 1.5, 1.6, 1.7000000000000002, 1.8,
   1.9000000000000001]
```

که اگر از خطای گردکردن صرف نظر کنیم به صورت ساده خروجی به شکل زیر خواهد شد:

```
1 [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3,
   1.4, 1.5, 1.6, 1.7, 1.8, 1.9]
```

معمولا به چنین لیستی مش مربوط به متغیر t گفته می‌شود و برای دسترسی به هر یک از این مقادیر باید از اندیس استفاده کنیم. پس وقتی از $t_0 + i \times \Delta t$ سخن گفته می‌شود، معمولا در زبان کامپیوتر به صورت‌های زیر نمایش داده می‌شود:

$$t_0 + i \times \Delta t \equiv t_i \equiv \texttt{t[i]} \qquad (۹.۲)$$

که اندیس `i` مکان مش را برای متغیر t نشان می‌دهد.

۱.۱.۲ خطای روش اویلر

همانطور که مشاهده کردید یک روش عددی به دلیل استفاده از تقریب دارای خطا می‌باشد. برای برآورد خطا در این روش (اویلر) به سادگی می‌توان از مفهوم بسط تیلور استفاده کرد:

$$x(t+\Delta t) = x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}x}\Delta t + \frac{\mathrm{d}^2x(t)}{\mathrm{d}x^2}\frac{(\Delta t)^2}{2} + \cdots + \frac{\mathrm{d}^n x(t)}{\mathrm{d}x^n}\frac{(\Delta t)^n}{n!} + \ldots \qquad (۱۰.۲)$$

بنابراین خطا در هر گام در روش اویلر از مرتبه $(\Delta t)^۲$ است که به صورت $\mathrm{O}(\Delta t^۲)$ نمایش داده می‌شود که منظور از O مرتبه بزرگی است. اما در N گام یعنی

$$N = \frac{t_f - t_i}{\Delta t} \qquad (۱۱.۲)$$

، که t_i زمان اولیه و t_f زمان نهایی است (که تصمیم داریم محاسبه بین این دو زمان انجام شود)، خطا از مرتبه‌ی Δt خواهد بود چرا که

$$Error \propto N \times (\Delta t)^۲ \propto \Delta t \qquad (۱۲.۲)$$

بنابراین خطا دارای مرتبه بزرگتری خواهد شد ($\mathrm{O}(\Delta t)$) .

تمرین ۱ : استفاده از روش اویلر برای حل معادله واپاشی هسته‌ها برنامه‌ای بنویسید که

- (مرحله اول)مقدار دهی اولیه:
  مقادیر N ، t_i ، t_f و Δt را دریافت کند (از ورودی). برای سادگی می‌توان t_i را برابر با صفر قرار دهیم.
- (مرحله اول) حل معادله واپاشی به صورت عددی:
  با فرض اینکه τ برابر ۱ روز باشد، معادله واپاشی را به صورت عددی حل کنید. برای حل عددی باید به روش زیر عمل کرد:
  - برای t و N یک لیست در نظر بگیرید:

```
t=[ti]
N=[N0]
```

در زمان های بعدی به صورت زیر مقدار N و t را در لیست ذخیره می‌کنیم:

$$\begin{aligned} N(t_{j+۱}) &= N_{j+۱} = N_j - N_j/\tau \Delta t \\ t_{j+۱} &= t_j + \Delta t \end{aligned}$$

که در پایتون به صورت زیر عمل می‌کنیم:

```
N.append(N[j]-N[j]/tau * dt)
t.append(t[j]+dt)
```

- (مرحله دوم): مقدار عددی N را برحسب زمان برای مقادیر مختلف Δt همراه با مقادیر دقیق N ، که از حل تحلیلی می‌توان بدست آورد، رسم کنید.
- (مرحله سوم): مقدار خطا (یعنی اختلاف بین حل دقیق و حل عددی در یک زمان مشخص مثلا ۱۰ $= t_f$) را برحسب Δt بدست آورید. در مقادیر بسیار کوچک Δt چه اتفاقی خواهد افتاد. چه رفتار در مقادیر معقول Δt مشاهد خواهیم کرد.

۲.۱.۲ سقوط آزاد

همانطور که می‌دانیم اگر گلوله‌ای را به صورت عمود پرتاب کنیم رابطه‌ی زیر برای مکان ذره از سطح پرتاب بدست می‌آید:

$$y = -\frac{۱}{۲}gt^۲ + v_۰t \qquad (۱۳.۲)$$

و سرعت آن بر حسب زمان به صورت زیر خواهد شد:

$$\frac{dy}{dt} = -gt + v_۰ \qquad (۱۴.۲)$$

در نقطه اوج سرعت صفر است:

$$\frac{dy}{dt} = ۰ \longrightarrow -gt + v_۰ = ۰ \longrightarrow t = \frac{v_۰}{g} \qquad (۱۵.۲)$$

و در نتیجه ارتفاع برابر با

$$h = \frac{v_۰^۲}{۲g}$$

اگر بخواهیم مکان ذره را به صورت عددی از روش اویلر بدست آوریم، اینچنین عمل می‌کنیم:

$$\begin{aligned} y(t=۰) &\approx y(t=۰)=۰ \qquad (۱۶.۲)\\ y(t=\Delta t) &\approx y(t=۰)+\frac{\mathrm{d}y}{\mathrm{d}t}\Big|_{t=۰}\Delta t\\ y(t=۲\Delta t) &\approx y(t=\Delta t)+\frac{\mathrm{d}y}{\mathrm{d}t}\Big|_{t=\Delta t}\Delta t\\ y(t=۳\Delta t) &\approx y(t=۲\Delta t)+\frac{\mathrm{d}y}{\mathrm{d}t}\Big|_{t=۲\Delta t}\Delta t\\ &\vdots \qquad \vdots \end{aligned}$$

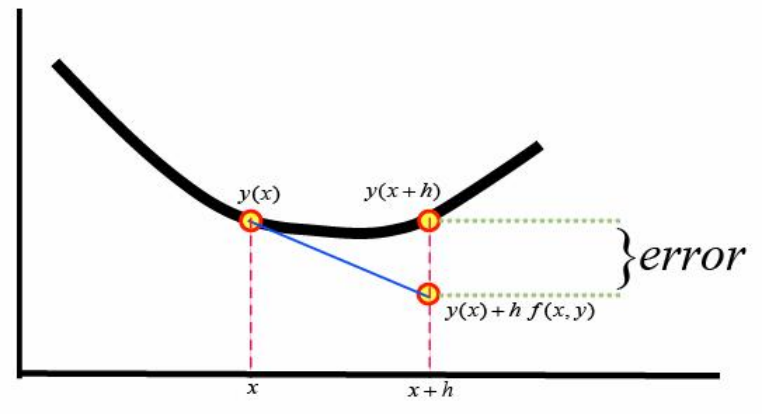
یعنی در هر نقطه زمانی می‌توان از نقطه‌ی قبلی استفاده کرد و نقطه‌ی بعدی را تخمین زد:

۲.۲ روش رانگ-کوتا

حال پرسشی که ممکن است در این مرحله مطرح شود این است که چگونه می‌توان خطا را کاهش داد. برای کاهش خطا می‌توان چندین کار انجام داد. یک راه حل ساده ولی ناکارآمد استفاده از جملات مراتب بالاتر در بسط تیلور است. ولی به جای استفاده از چنین شیوه‌ای می‌توان به دنبال الگوریتم‌های بهتر گشت که هزینه محاسباتی کمتری داشته باشند و از طرفی خطای محاسبات را کاهش دهند. یکی از این الگوریتم‌های هوشمند، الگوریتم رانگ-کوتای مرتبه دوم [۱] است. ایده‌ی اصلی این الگوریتم استفاده از مشتق تابع در وسط گام است، یعنی استفاده از مشتق در $t+\Delta t/۲$ است. همانطور که از شکل نیز مشخص است به نظر می‌رسد که استفاده مشتق در بین گام‌های به جای ابتدای هر گام ایده‌ی مناسبی است. حال باید نشان دهیم که واقعا استفاده از مشتق در بین گام‌ها باعث کاهش خطا می‌شود. برای این کار باز به بسط تیلور رجوع می‌کنیم. $x(t+\Delta t)$ و $x(t)$ را حول $x(t+\frac{\Delta t}{۲})$ بسط می‌دهیم:

$$\begin{aligned} x(t+\Delta t) &= x(t+\frac{\Delta t}{۲})+\frac{dx}{dt}\Big|_{t+\frac{\Delta t}{۲}}\frac{\Delta t}{۲}+\frac{۱}{۲}\frac{d^۲x}{dt^۲}\Big|_{t+\frac{\Delta t}{۲}}(\frac{\Delta t}{۲})^۲+O(\Delta t^۳) \qquad (۱۷.۲)\\ x(t) &= x(t+\frac{\Delta t}{۲})-\frac{dx}{dt}\Big|_{t+\frac{\Delta t}{۲}}\frac{\Delta t}{۲}+\frac{۱}{۲}\frac{d^۲x}{dt^۲}\Big|_{t+\frac{\Delta t}{۲}}(\frac{\Delta t}{۲})^۲+O(\Delta t^۳) \qquad (۱۸.۲) \end{aligned}$$

[۱] second-order Runge-Kutta



شکل ۱.۲ : این شکل استفاده از مشتق در ابتدای هر گام و خطای آن در روش اویلر را نشان می‌دهد.

با تفریق کردن معادله اول از معادله دوم و مرتبه کردن جملات به معادله زیر می‌رسیم:

$$\begin{aligned} x(t+\Delta t) &= x(t) + \frac{dx}{dt}\Big|_{t+\frac{\Delta t}{۲}} \Delta t + O(\Delta t^{۳}) \\ &= x(t) + f\Big(x(t+\frac{\Delta t}{۲}), t+\frac{۱}{۲}\Big)\Delta t + O(\Delta t^{۳}) \end{aligned} \qquad (۱۹.۲)$$

که در معادله دوم منظور از f همان مشتق تابع x است. نکته قابل توجه این است که مشتق تابع x ممکن است وابسته به خود x نیز باشد بنابراین این مشتق یعنی $\frac{dx}{dt}\Big|_{t+\frac{\Delta t}{۲}}$ به صورتی که در معادله نمایش داده شده یعنی $f\Big(x(t+\frac{\Delta t}{۲}), t+\frac{۱}{۲}\Big)$ نشان داده می‌شود. در اینجا سوالی که ممکن است به ذهن خطور کند این است که اگر مشتق x یا همان f به نقطه $x(t+\frac{\Delta t}{۲})$ وابسته باشد، باید چه کنیم. برای حل این مشکل می‌توان از الگوریتم اویلر برای حدس زدن در این نقطه استفاده کرد:

$$x(t+\frac{\Delta t}{۲}) = x(t) + f(x,t)\frac{\Delta t}{۲} \qquad (۲۰.۲)$$

بنابراین به صورت عملی الگوریتم مرتبه دوم رانگ_کوتا به شکل زیر در خواهد آمد:

$$\begin{aligned} k_۱ &= f(x,t)\Delta t \qquad (۲۱.۲)\\ k_۲ &= f\Big(x+\frac{۱}{۲}k_۱, t+\frac{۱}{۲}\Delta t\Big)\Delta t \\ x(t+\Delta t) &= x(t)+k_۲ \end{aligned}$$

خطا در این روش از $O(\Delta t^۳)$ می‌باشد (چرا؟). مثال زیر یک نمونه برنامه برای رانگ_کوتای مرتبه دوم است.

```
from math import sin
from numpy import arange
from matplotlib.pyplot import plot, xlabel, ylabel, show
# dx/dt= -x^3+sin(t)
def f(x,t):
    return -x**3 + sin(t)

a= 0.0
b= 10.0
N=200
dt= (b-a)/N

tpoints = arange(a,b,dt)
xpoints = []

x = 0.0
for t in tpoints:
    xpoints.append(x)
    k1 = dt* f(x,t)
    k2 = dt* f(x+0.5*k1, t+0.5*dt)
    x += k2

plot(tpoints, xpoints)
xlabel("t")
ylabel("x(t)")
show()
```

: rk2.py

اگر در بسط تیلور از جملات بالاتر یعنی $\Delta t^۳$ و $\Delta t^۴$ استفاده کنیم، ما می‌توانیم الگوریتم رانگ‌ـ کوتای مرتبه چهار که به صورت زیر است را بدست آوریم:

$$\begin{aligned} k_۱ &= f(x,t) \\ k_۲ &= f(x+\frac{۱}{۲}k_۱, t+\frac{۱}{۲}\Delta t)\Delta t \\ k_۳ &= f(x+\frac{۱}{۲}k_۲, t+\frac{۱}{۲}\Delta t)\Delta t \\ k_۴ &= f(x+k_۳, t+\Delta t)\Delta t \\ x(t+\Delta t) &= x(t)+\frac{۱}{۶}(k_۱+۲k_۲+۲k_۳+k_۴) \end{aligned} \tag{۲۲.۲}$$

مثال زیر همان مثال قبلی منتها با الگوریتم رانگ‌ـ کوتای مرتبه چهارم:

```
from math import sin
from numpy import arange
from matplotlib.pyplot import plot,xlabel,ylabel,show

def f(x,t):
    return -x**3 + sin(t)

a = 0.0
b = 10.0
N = 100
dt = (b-a)/N

tpoints = arange(a,b,dt)
xpoints = []
x = 0.0

for t in tpoints:
    xpoints.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x+0.5*k1, t+0.5*dt)
    k3 = dt*f(x+0.5*k2, t+0.5*dt)
    k4 = dt*f(x+k3, t+dt)
    x += (k1+2*k2+2*k3+k4)/6

plot(tpoints,xpoints)
xlabel("t")
ylabel("x(t)")
```

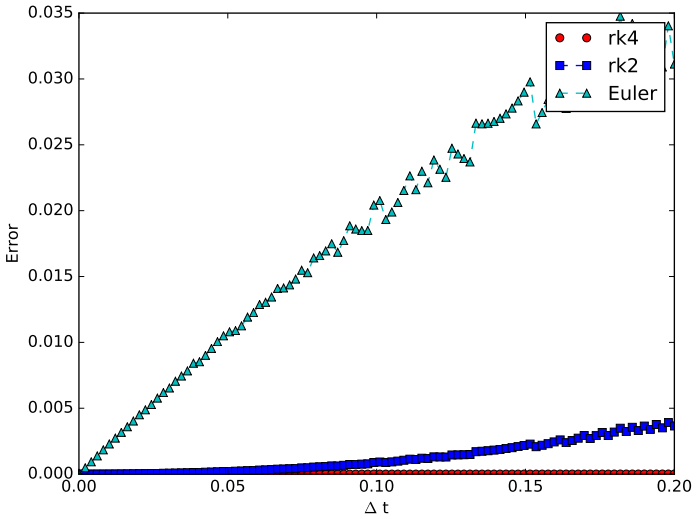
```
28 show ()
```

: rk4.py

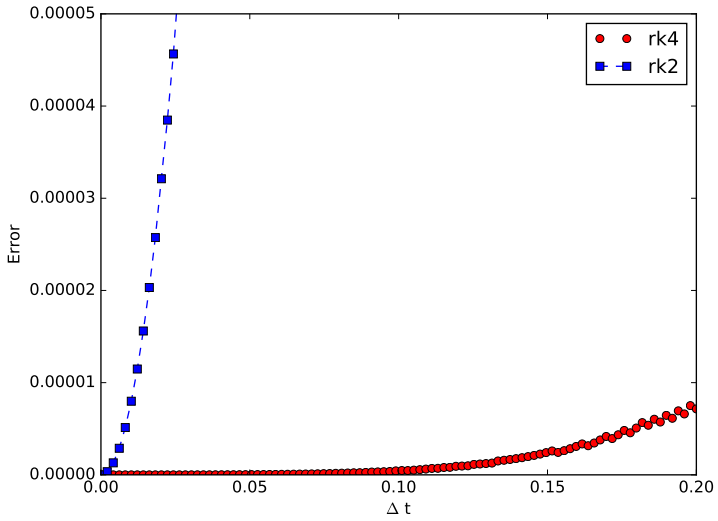
۱.۲.۲ مقایسه روش‌های اویلر، رانگ-کوتای مرتبه دوم و رانگ-کوتای مرتبه چهارم

همانظور که اشاره شد، خطا در هر گام برای الگوریتم اویلر از مرتبه $O(\Delta t^۲)$ است. و مجموع خطا های در گام آخر از مرتبه $O(\Delta t)$ است. این موضوع به وضوح در شکل ۲.۲ مشخص شده است. برای رسم این شکل معادله مربوط به واپاشی با سه روش اویلر، رانگ-کوتای مرتبه دوم و چهارم حل شده است و در زمان پایانی (گام آخر) قدر مطلق اختلاف مقدار عددی از مقدار دقیق برای گام های زمانی مختلف، Δt ، رسم شده است.

به وضوح مشاهده می‌شود که خطا در روش اویلر با Δt به صورت خطی رفتار می‌کند. همانطور که در بخش قبل اشاره کردیم، در روش رانگ-کوتای مرتبه دوم خطا از مرتبه $O(\Delta t^۳)$ و بنابراین در گام آخر از مرتبه $O(\Delta t^۲)$ است. در روش رانگ-کوتای مرتبه چهارم، خطا در هر گام از مرتبه $O(\Delta t^۵)$ و خطا در گام آخر از مرتبه $O(\Delta t^۴)$ می‌باشد. براینکه این موضوع را در شکل ببینیم، شکل قبل را در ناحیه کوچکتر (در راستای y) رسم کرده‌ایم (شکل ۳.۲)



شکل ۲.۲: مقایسه روش های اویلر، رانگ_کوتای مرتبه دوم و چهارم



شکل ۳.۲: مقایسه روش های رانگ_کوتای مرتبه دوم و چهارم

برای رسم خطاهای از برنامه زیر استفاده شده است:

```
from math import sin, exp
import numpy as np
import matplotlib.pyplot as plt
# dN/dt= -N(t)/tau

tau=1
def f(x):
    return -x/tau

N0=1000


def euler_alg(f,tf,dt):
    N_step=int(tf/dt)
    t_list=[0]*(N_step+1)
    x_list=[0]*(N_step+1)
    x_list[0]=N0
    for i in range(N_step):
        x_list[i+1]=x_list[i]+f(x_list[i])*dt
        t_list[i+1]=t_list[i]+dt
    return t_list[-1], x_list[-1]

def rk2_alg(f,tf,dt):
    N_step=int(tf/dt)
    t_list=[0]*(N_step+1)
    x_list=[0]*(N_step+1)
    x_list[0]=N0
    for i in range(N_step):
        x=x_list[i]
        k1 = dt* f(x)
        k2 = dt* f(x+0.5*k1)
        x += k2
        t_list[i+1]=t_list[i]+dt
        x_list[i+1]=x
    return t_list[-1], x_list[-1]


def rk4_alg(f,tf,dt):
    N_step=int(tf/dt)
    t_list=[0]*(N_step+1)
```

```
    x_list=[0]*(N_step+1)
    x_list[0]=N0
    for i in range(N_step):
      x=x_list[i]
      k1 = dt*f(x)
      k2 = dt*f(x+0.5*k1)
      k3 = dt*f(x+0.5*k2)
      k4 = dt*f(x+k3)
      x += (k1+2*k2+2*k3+k4)/6
      t_list[i+1]=t_list[i]+dt
      x_list[i+1]=x
    return t_list[-1], x_list[-1]

tf=10
dt_list=np.linspace(0.0001, 0.2, 100)

rk2_error=[]
rk4_error=[]
euler_error=[]
for dt in dt_list:

    t,rk4=rk4_alg(f,tf,dt)
    t,rk2=rk2_alg(f,tf,dt)
    t,eul=euler_alg(f,tf,dt)
    N_exac=N0*exp(-t/tau)
    rk4_error.append(abs(rk4-N_exac))
    rk2_error.append(abs(rk2-N_exac))
    euler_error.append(abs(eul-N_exac))


plt.plot(dt_list, rk4_error ,'ro')
plt.plot(dt_list, rk2_error, 'bs--')
plt.plot(dt_list, euler_error, 'c^--')
plt.legend(["rk4", "rk2", "Euler"])
plt.xlabel(r"$\Delta$ t")
plt.ylabel("Error")
plt.xlim(0.0, 0.2 )
plt.savefig('rk4-rk2-euler.pdf')
plt.show()
```

: rk4_rk2_euler_radioactive.py

۲.۲.۲ مسیر یک گلوله توپ (پرتابه) با حضور مقاومت هوا

شاید یکی از ساده‌ترین مسایل که در حین سادگی حل تحلیلی نمی‌توان برای آن پیدا کرد، حرکت یک پرتابه تحت نیروی جاذبه و در حضور مقاومت هوا است. اگر یک گلوله توپ کروی را در نظر بگیرید نیروی مقاومت هوا به صورت زیر خواهد شد:

$$F = \frac{۱}{۲}\pi R^۲ \rho C v^۲ \qquad (۲۳.۲)$$

که در این معادله R شعاع گلوله، ρ چگالی هوا، v سرعت گلوله و C ضریب پس کشی [۲] هوا است. بنابراین معادلات حرکت گلوله توپ در مختصات (x, y) به صورت زیر خواهند شد:

$$a_x = \dot{v_x} = -\frac{\pi R^۲ \rho C}{۲m} v_x \sqrt{v_x^۲ + v_y^۲} \qquad (۲۴.۲)$$

$$a_y = \dot{v_y} = -g - \frac{\pi R^۲ \rho C}{۲m} v_y \sqrt{v_x^۲ + v_y^۲} \qquad (۲۵.۲)$$

حال اگر فرض کنیم $B_{drag} = \frac{۱}{۲}\pi R^۲ \rho C$ ، از دو معادله بالا می‌توان معادلات زیر را نوشت:

$$\begin{aligned}
\dot{x} &= \frac{\mathrm{d}x}{\mathrm{d}t} = v_x \\
\dot{y} &= \frac{\mathrm{d}y}{\mathrm{d}t} = v_y \\
\dot{v_x} &= \frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{B_{\mathrm{drag}} v v_x}{m} \\
\dot{v_y} &= \frac{\mathrm{d}v_y}{\mathrm{d}t} = -g - \frac{B_{\mathrm{drag}} v v_y}{m}
\end{aligned}$$

بنابراین حل معادلات بالا به روش اویلر به صورت زیر خواهد بود:

$$\begin{aligned}
x_{i+۱} &= x_i + v_{x,i}\Delta t \\
y_{i+۱} &= y_i + v_{y,i}\Delta t \\
v_{x,i+۱} &= v_{x,i} - \frac{B_{\mathrm{drag}} v_i v_{x,i}}{m}\Delta t \\
v_{y,i+۱} &= v_{y,i} - g\Delta t - \frac{B_{\mathrm{drag}} v_i v_{y,i}}{m}\Delta t\,,
\end{aligned}$$

[۲] drag cofficient

که

$$v_i = \sqrt{v_{x,i}^۲ + v_{y,i}^۲}\,.$$

تمرین ۲: استفاده از روش اویلر و رانگ-کوتا برای حل معادلات حرکت گلوله توپ در حضور مقاومت هوا: با فرض پارامترهای زیر:

$$C = ۰/۴۷ \text{ ،} \rho = ۱/۲۲ kg/m^{-۳} \text{ ،} m = ۱ Kg \text{ ،} R = ۸ cm$$

- برای یک مقدار دلخواه زاویه پرتاب (از وردی برنامه دریافت شود) θ ، و یک مقدار دلخواه از سرعت اولیه گلوله توپ (از وردی برنامه دریافت شود) $v.$ ، به روش‌های اویلر، رانگ-کوتای مرتبه دوم و چهارم مسیر پرتابه را در یک نمودار رسم کنید (رسم y بر حسب x) . مختصات اولیه را به صورت $(x. = ۰, y. = ۰)$ در نظر بگیرید. برنامه را طور بنویسید که وقتی گلوله توپ به زمین خورد محاسبات متوقف شود.
- فرض کنید که مقاومت هوا وجود ندارد، یعنی $C = ۰/۰$. حال برد گلوله توپ را برای یک مقدار دلخواه زاویه پرتابه و یک مقدار دلخواه سرعت اولیه با استفاده از سه روشی، که در قسمت اول بیان شد، محاسبه کنید و با مقدار دقیق آن یعنی $۲v_.^۲ sin(\theta)cos(\theta)/g$ مقایسه کنید. کدام روش دقیق تر است؟ (نکته: برای اینکه خطا مشخص تر شود بهتر است از مقادیر بزرگ برای سرعت استفاده کنیم)

راهنمایی حل تمرین ۲: می‌توان برای رانگ-کوتای مرتبه چهارم از تابع زیر (در یک بعد) استفاده کنیم.

```
def rk4(x, v, a, dt):
    """Returns final (position, velocity) tuple after
    time dt has passed.

    x: initial position (number-like object)
    v: initial velocity (number-like object)
    a: acceleration function a(x,v,dt) (must be callable)
    dt: timestep (number)"""
    x1 = x
    v1 = v
    a1 = a(x1, v1, 0)
```

```
12
        x2 = x + 0.5*v1*dt
14      v2 = v + 0.5*a1*dt
        a2 = a(x2, v2, dt/2.0)
16
        x3 = x + 0.5*v2*dt
18      v3 = v + 0.5*a2*dt
        a3 = a(x3, v3, dt/2.0)
20
        x4 = x + v3*dt
22      v4 = v + a3*dt
        a4 = a(x4, v4, dt)
24
        xf = x + (dt/6.0)*(v1 + 2*v2 + 2*v3 + v4)
26      vf = v + (dt/6.0)*(a1 + 2*a2 + 2*a3 + a4)

28      return xf, vf
```

: rk4_dim1.py

۳.۲ استفاده از scipy.integrate.odeint برای حل معادلات دیفرانسیل معمولی

odeint یک تابع در کتابخانه‌ی scipy است که می‌توان با استفاده از آن معادلات دیفرانسیلی معمولی را حل کرد. بدین منظور ابتدا باید معادله دیفرانسیل خود را برای odeint تعریف کرد. برای این کار چنین عمل می‌کنیم: یک تابع تعریف می‌کنیم که این تابع مشتق اول، دوم و ... را برمی گرداند. این تابع دو آرگومان خواهد داشت. یکی از آرگومان ها یک آرایه است که عنصر اول آن خود تابع و عناصر بعدی آن مشتق‌های مراتب بالاتر می‌باشند. یکی دیگر از آرگومان‌های این تابع متغییری است که تابع با آن تعریف می‌شود. دلیل وارد کردن این متغییر به خاطر این است که در تعریف مشتق‌های مرتبه‌های مختلف ممکن است نیاز به استفاده از متغییر تابع داشته باشیم. برای مثال برای معادله واپاشی تعریف این تابع به صورت زیر خواهد شد:

```
def deriv(N,t): # return derivatives of the array N
2       tau=10.0
        return array([ -N[0]/tau])
```

: deriv.py

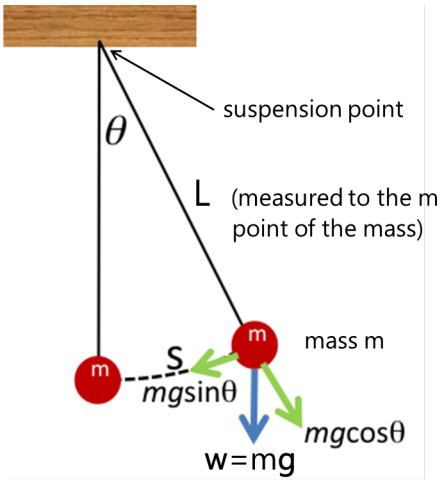
همانطور که گفته شد در اینجا N[۰] خود تابع است و N[۰]/tau- مشتق اول این تابع. اگر معادلاتی برای مشتقات مراتب بالاتر وجود داشت باید آنها را زیر وارد کنید که در اینجا به دلیل اینکه معادله دیفرانسیل ما، یک معادله مرتبه اول است احتیاج به آنها نداریم. بعد از آن باید مقادیر اولیه تابع و مشتقات آن را نیز برای حل معادله دیفرانسیل تعیین کنیم. مثال زیر حل معادله واپاشی هسته‌ها را به روش odeint به شما نشان می‌دهد:

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

tau =10
def deriv(N,t): # return derivatives of the array N
     return np.array([ -N[0]/tau])

time= np.linspace(0.0,100.0,1000)
Ninit= np.array([10000])# initial values
N =odeint(deriv,Ninit,time)


Nexact=Ninit[0]*np.exp(-time/tau)

plt.figure()
plt.plot(time,N) # y[:,0] is the first column of y
plt.plot(time,Nexact)
plt.xlabel("t")
plt.ylabel("N")
plt.show()
```

: ode.py

۴.۲ حرکت هماهنگ ساده

در فیزیک به مثال‌های زیادی از حرکت‌های تناوبی برخورد می‌کنیم. شاید ساده‌ترین مثال حرکت یک پاندول باشد. به نوسانات یک پاندول حرکت هماهنگ ساده گفته می‌شود. شکل ۴.۲ جزییات این



شکل ۴.۲: حرکت هماهنگ ساده

حرکت را نشان می‌دهد. برای مقادیر کوچک θ می‌توان نیروی مماسی وارد به جرم آویزان را به صورت زیر نوشت:

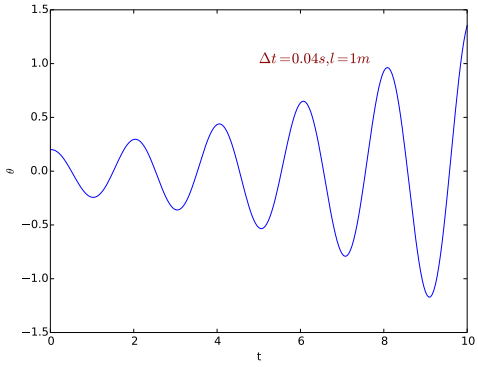
$$F_\theta = -mg\sin\theta \approx -mg\theta \qquad (۲۶.۲)$$

بنابراین معادله حرکت به صورت زیر در می‌آید:

$$\ddot{\theta} = \frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{l}\theta \qquad (۲۷.۲)$$

که بر حسب فرکانس این معادله به صورت زیر در می‌آید:

$$\begin{aligned} \frac{\mathrm{d}\omega}{\mathrm{d}t} &= -\frac{g}{l}\theta \\ \frac{\mathrm{d}\theta}{\mathrm{d}t} &= \omega \end{aligned}$$



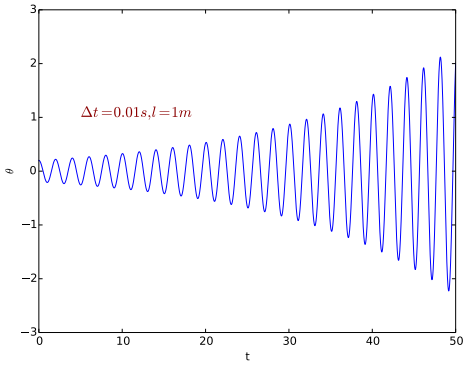
شکل ۵.۲: محاسبه θ بر حسب زمان به روش اویلر

حال می‌توانیم این معادلات را به روش اویلر حل کنیم:

$$\begin{aligned} \theta_{i+1} &= \theta_i + \omega_i \Delta t \\ \omega_{i+1} &= \omega_i - \frac{g}{l}\theta_i \Delta t \ , \end{aligned}$$

که در این معادلات $\theta_i = \theta(t_i)\ ,\ \omega_i = \omega(t_i)$. حال فرض کنید طول نخی که جرم از آن آویزان است برابر با $l = ۱m$ باشد. با انتخاب گام زمانی برابر با $\Delta t = ۰/۰۴s$ نتیجه عجیبی بدست می‌آید که در شکل ۵.۲ رسم شده است.

در این شکل θ بر حسب زمان رسم شده است. همانطور که مشاهده می‌کنید با افزایش زمان، دامنه نوسانات پاندول نیز افزایش پیدا می‌کند که کاملا مخالف مشاهدات واقعی از حرکت یک پاندول است. در واقعا باید به پاندول انرژی وارد شود تا چنین حرکتی از خود نشان دهد، در حالی که در معادله حرکتی که در بالا معرفی شد چنین جمله‌ی وجود ندارد! شاید اندازه‌ی گام زمانی، Δt ، بزرگ در نظر گرفته شده است و این باعث چنین فاجعه‌ای در محاسبات شده است. همانطور که گفته شد، گام زمانی باید براساس زمان‌های نوعی که در سیستم وجود دارد انتخاب شود. در اینجا زمان نوعی سیستم همان زمان تناوب است که برابر خواهد بود با: $T = ۲\pi\sqrt{\frac{l}{g}}$. برای $l = ۱m$ این مقدار برابر با $۲s$ خواهد بود. بنابراین انتخاب Δt برابر با $۰/۰۴s$ نباید باعث چنین خطایی شود چرا که طبق معیاری تقریبی که از آن صحبت شد می‌توانیم گام زمانی را به صورت زیر انتخاب کنیم: $T/۱۰۰ - T/۵۰$. با این حال



شکل ۶.۲: محاسبه θ برحسب زمان به روش اویلر.

اگر فرض کنیم این خطا ناشی از انتخاب نادرست گام زمانی است باید با کاهش گام زمانی بتوانیم این خطا را حذف کنیم. به همین دلیل یک بار دیگر محاسبات را با گام زمانی کمتر تکرار می‌کنیم. این بار $\Delta t = ۰/۰۱s$ در نظر می‌گیرم که برابر با $T/۲۰۰$ می‌باشد. در شکل ۶.۲ باز θ بر حسب زمان به روش اویلر رسم شده است. همانطور که مشاهده می‌کنیم خطایی که در دامنه‌ی نوسان پاندول قبلا مشاهده کردیم این بار خود را در زمان‌های بالاتر نشان می‌دهم. پس واقعا مشکل مقدار انتخاب گام زمانی نیست و باید به دنبال منبع خطا گشت. در حقیقت اتقاقی که در نمودار های بالا اتفاق افتاده است حاکی از افزایش انرژی در سیستم است. بدین منظور بهتر است مقدار انرژی که از روش اویلر در هر گام محاسبه می‌شود را بدست آوریم:

$$\begin{aligned} E = E_{\text{kin}} + E_{\text{pot}} &= \frac{m}{۲} l^۲ \omega^۲(t) + mgl[۱ - \cos\theta(t)] \qquad (۲۸.۲) \\ &\approx \frac{m}{۲} l^۲ \omega^۲(t) + \frac{m}{۲} gl\theta^۲(t) \end{aligned}$$

حال انرژی کل را می‌توان در گام بعدی به صورت زیر بدست آورد:

$$\begin{aligned} E_{i+1} &= \frac{ml^2}{2}\left[\omega_{i+1}^2 + \frac{g}{l}\theta_{i+1}^2\right] = \frac{ml^2}{2}\left[\left(\omega_i - \frac{g}{l}\theta_i\Delta t\right)^2 + \frac{g}{l}(\theta_i + \omega_i\Delta t)^2\right] \\ &= \frac{ml^2}{2}\left[\omega_i^2 + \frac{g}{l}\theta_i^2\right] + \frac{mgl}{2}\left(\frac{g}{l}\theta_i^2 + \omega_i^2\right)(\Delta t)^2 \\ &= E_i + \frac{mgl}{2}\left(\frac{g}{l}\theta_i^2 + \omega_i^2\right)(\Delta t)^2 \,. \end{aligned}$$

بنابراین طبق معادله بالا انرژی کل پایسته نیست و در هر گام جمله‌ی $\frac{mgl}{2}\left(\frac{g}{l}\theta_i^2 + \omega_i^2\right)(\Delta t)^2$ به انرژی گام قبلی اضافه می شود! برای حل این مشکل به جایی استفاده از الگوریتم اویلر یعنی:

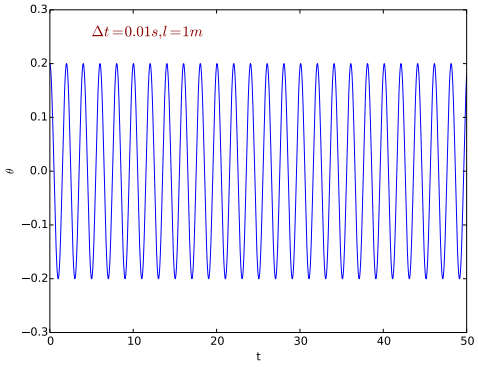
$$\begin{aligned} \theta_{i+1} &= \theta_i + \omega_i\Delta t \\ \omega_{i+1} &= \omega_i - \frac{g}{l}\theta_i\Delta t \,, \end{aligned}$$

با کمی تغییر در الگوریتم از الگوریتم زیر استفاده می‌کنیم:

$$\begin{aligned} \omega_{i+1} &= \omega_i - \frac{g}{l}\theta_i\Delta t \\ \theta_{i+1} &= \theta_i + \omega_{i+1}\Delta t \,. \end{aligned}$$

این الگوریتم به الگوریتم اویلر-کرامر [۳] معرف است. در شکل ۷.۲ تغییرات θ برحسب زمان با روش اویلر-کرامر رسم شده است. همانطور که مشاهده می‌کنید مقدار دامنه‌ی نوسان پاندول تغییر نمی‌کند. دلیل این امر این است که الگوریتم اویلر-کرامر انرژی کل در هر گام پایسته نگه می‌دارد. به عنوان یک تمرین این موضوع را اثبات کنید.

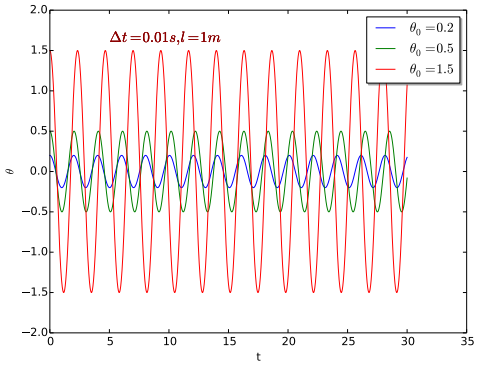
[۳] Euler-Cromer algorithm



شکل ۷.۲: محاسبه θ بر حسب زمان به روش اویلر_کرامر

۱.۴.۲ حرکت هماهنگ ساده‌ی واداشته

در مثال‌های قبلی مقدار θ. ، یعنی مقدار اولیه زاویه رها شدن پاندول، را به طور اختیار انتخاب کردیم. قاعدتا این مقدار اولیه نباید باعث تغییر رفتار در حرکت هماهنگ ساده شود. این موضوع را می‌توان در شکل ۸.۲ به راحتی مشاهده کرد. در این شکل می‌توان دید که مقدار اولیه θ تاثیری بر روی حرکت نوسانی پاندول نمی‌گذارد.



شکل ۸.۲: محاسبه θ بر حسب زمان به روش اویلر_کرامر با مقادیر اولیه مختلف برای θ

ولی اگر به واسطه یک عامل خارجی حرکت پاندول را مختل کنیم، آنگاه مقادیر اولیه زاویه رها شدن

نیز اهمیت پیدا می‌کنند. معادله حرکت یک نوسانگر که توسط یک نیروی خارجی (به طور متناوب) مختلط می‌شود را می‌توان به صورت زیر در نظر گرفت:

$$\frac{\mathrm{d}^۲\theta}{\mathrm{d}t^۲} = -\frac{g}{l}\sin\theta - q\frac{\mathrm{d}\theta}{\mathrm{d}t} + F_D\sin(\Omega_D t)\ . \qquad (۲۹.۲)$$

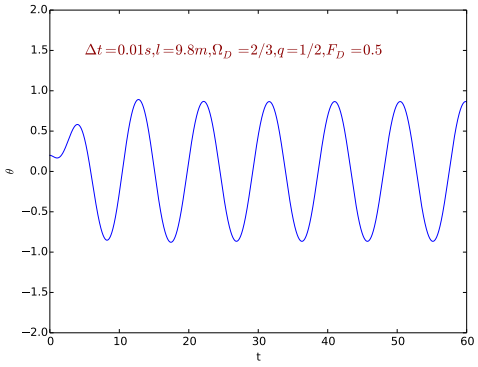
که در این معادله، جمله $-q\frac{\mathrm{d}\theta}{\mathrm{d}t}$ مقاومت یا در واقع کاهش در حرکت نوسانی [۴] را نشان می‌دهد که قبلا در نظر نمی‌گرفتیم. مقدار q قدرت این کاستن را نشان می‌دهد. جمله‌ی $F_D\sin(\Omega_D t)$ یک نیروی واداشته‌ی سینوسی [۵] را نشان می‌دهد که F_D دامنه‌ی این نیرو و Ω_D مقدار فرکانس زاویه نوسانات این نیرو را نشان می‌دهد. باز مانند حرکت یک پرتابه در مقاومت هوا برای حل چنین معادله‌ی جواب تحلیلی وجود ندارد و باید حتما از روش‌های عددی استفاده کنیم. برای اینکه با اثرات نیروی واداشته بیشتر آشنا شویم و ببینیم چگونه تغییر در یک پارامتر می‌تواند باعث تغییر اساسی در حرکت پاندول شود، دو حالت ساده‌ی زیر را به عنوان شرایط اولیه مسله درنظر می‌گیرم:

$$F_D = ۰/۵,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_۰ = ۰/۲$$
$$F_D = ۱/۲,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_۰ = ۰/۲$$

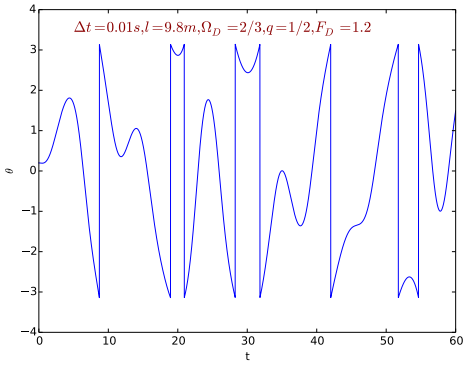
تفاوت این دو حالت فقط در مقدار F_D است. شکل‌های ۹.۲ و ۱۰.۲ نشان می‌دهد که چگونه این تفات کوچک باعث تغییر اساسی در رفتار حرکت نوسانی می‌شود. در شکل ۹.۲ هنوز حرکت نوسانی حفظ شده است در حالی که در شکل ۱۰.۲ حرکت نوسانی کاملا از بین رفته است. دلیل پرش‌های عمودی در شکل ۱۰.۲ به خاطر این است اگر زاویه θ از بازه‌ی $-\pi$ تا π خارج شود برای اینکه بتوان آن را رسم کرد باز به این باز برگردانده می‌شود. بنابراین این پرش‌ها به معنی ناپیوستگی در θ نمی‌باشند. قسمتی از برنامه که به زبان پایتون است در زیر آورده شده است

```
for i in range(n):

    omega[i+1]=omega[i] - g/l *sin(theta[i])*dt \
                -q * omega[i]*dt + F_D*sin(omega_D*t[i])*dt
```

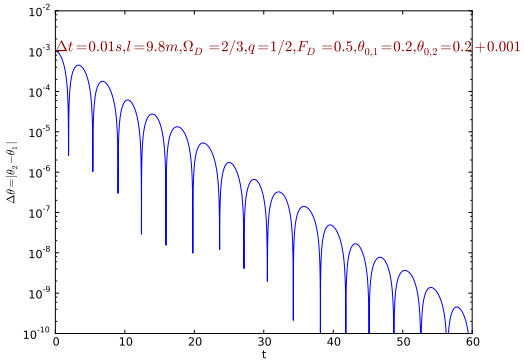
[۴] damping
[۵] sinusoidal driving force



شکل ۹.۲: محاسبه θ بر حسب زمان به روش اویلر_کرامر با شرایط:
$F_D = ۰/۵,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_۰ = ۰/۲$



شکل ۱۰.۲: محاسبه θ بر حسب زمان به روش اویلر_کرامر با شرایط:
$F_D = ۱/۲,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_۰ = ۰/۲$



شکل ۱۱.۲: $F_D = ۰/۵,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_{۰,۱} = ۰/۲, \theta_{۰,۲} = ۰/۲ + ۰/۰۰۱$

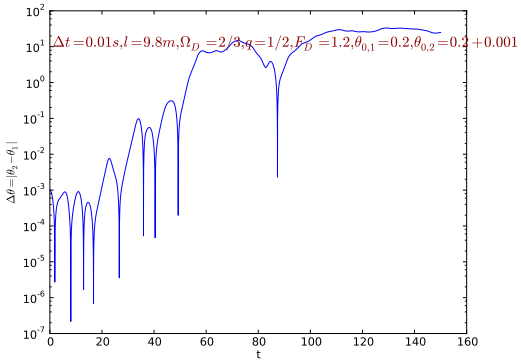
```
    theta[i+1]=theta[i] + omega[i+1]*dt
    t[i+1] = t[i] +dt
    if theta[i+1] >  pi:
        theta[i+1] = theta[i+1] - 2.0 * pi
    if theta[i+1] < -pi:
        theta[i+1] = theta[i+1] + 2.0 * pi
```

: hm1.py

۲.۴.۲ آشوب در حرکت نوسانی

دو پاندول کاملا یکسان را در نظر بگیرید، با طول یکسان، با ضریب کاهش یکسان و با نیروی واداشته یکسان.اگر تنها تفاوت این دو پاندول زاویه اولیه رها شدن آنها باشد، به طوری که این دو زاویه تفاوتی از مرتبه‌ی یک هزارم داشته باشند، انتظار داریم که تفاوت چندانی در حرکت آنها مشاهده نکنیم. برای اینکه چنین نتیجه گیری را انجام دهیم باید ابتدا زاویه $\theta_۱(t)$ و $\theta_۲(t)$ را بر حسب زمان محاسبه کنیم و سپس کمیت $\Delta\theta(t) = |\theta_۱(t) - \theta_۲(t)|$ در طول زمان بدست آوریم. انتظار ما این است که این کمیت کمتر و کمتر شود تا اینکه این دو نوسانگر دقیقا شبیه هم نوسان کنند. ولی اتفاق جالبی رخ می‌دهد: بستگی به مقدار F_D رفتار دو نوسانگر می‌تواند هم زمان شود و یا می‌تواند کاملا رفتاری آشوبناک نسبت به هم از خود نشان دهند. در شکل‌های ۱۱.۲ و ۱۲.۲ می‌توانید چنین رفتارهایی را مشاهده کنید.

در شکل ۱۱.۲ $F_D = ۰/۵$ است و مقدار زاویه اولیه برای نوسانگر اول $\theta_{۰,۱} = ۰/۲$ و مقدار



شکل ۱۲.۲: $F_D = ۱/۲,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_{۰,۱} = ۰/۲, \theta_{۰,۲} = ۰/۲ + ۰/۰۰۱$

زاویه اولیه برای نوسانگر دوم $\theta_{۰,۲} = \theta_{۰,۱} + ۰/۰۰۱$ می‌باشد. همانطور که مشاهده می‌کنید $\Delta\theta(t) = |\theta_۱(t) - \theta_۲(t)|$ به سمت صفر میل می‌کند (نمودار لگاریتمی است). همین شرایط اولیه را ما برای F_D در نظر می‌گیرم با این تفاوت که $F_D = ۱/۲$. در شکل ۱۲.۲ مشاهده می‌کنید که چگونه دو نوسانگر از هم فاصله می‌گیرند. در این حالت ما اصطلاحا می‌گوییم رفتار دو نوسانگر نسبت به هم آشوبناک است. یا به طور ساده چنین سیستم‌های، سیستم‌های آشوبناک هستند چرا که بشدت به شرایط اولیه حساس هستند. در هر دو شکل برای $\Delta\theta$ می‌توان رفتاری به صورت نمایی مشاهده کرد:

$$\log(\Delta\theta) \sim \lambda t \qquad (۳۰.۲)$$

و یا

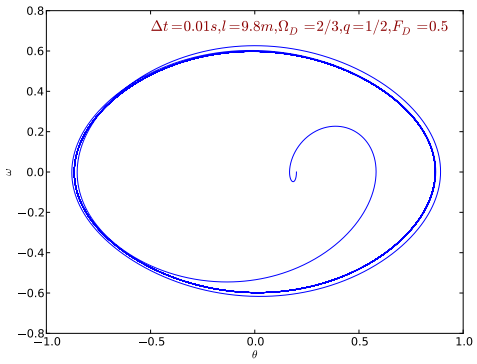
$$\Delta\theta \approx e^{\lambda t} \qquad (۳۱.۲)$$

پارامتر λ به نمای لیاپانوف [۶] معرف است. بنابراین رفتار $\Delta\theta$ را می‌توان در هر دو رژیم یعنی غیر آشوبناک و آشوبناک با نمای لیاپانوف توصیف کرد. در حالت آشوبناک $\lambda > ۰$ و در حالت غیر آشوبناک $\lambda < ۰$. بنابراین گذار به رژیم آشوبناک وقتی اتفاق می‌افتد که $\lambda = ۰/۰$ باشد.

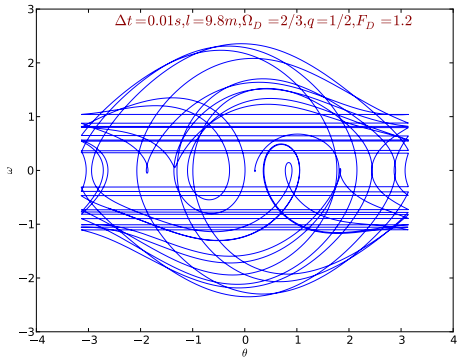
می‌توان آشوب را در نمودارهای فضایی فاز نوسانگرهایی هارمونیک نیز مشاهده کرد (شکل‌های:

[۶] Lyapunov exponent

۱۳.۲ و ۱۴.۲).



شکل ۱۳.۲: $F_D = ۰/۵,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_{.,۱} = ۰/۲, \theta_{.,۲} = ۰/۲ + ۰/۰۰۱$



شکل ۱۴.۲: $F_D = ۱/۲,\ q = ۱/۲,\ \Omega_D = ۲/۳,\ \theta_{.,۱} = ۰/۲, \theta_{.,۲} = ۰/۲ + ۰/۰۰۱$

تمرین ۳: استفاده از روش اویلر_کرامر برای حرکت پاندول:

- با محاسبه θ بر حسب زمان با روش اویلر و اویلر_کرامر نشان دهید که روش اویلر برای سیستم‌های نوسانی ناکارآمد است (لطفا نمودار θ را بر حسب زمان برای هر دو روش رسم کنید.)

- انرژی کل را با استفاده از دو روش محاسبه کنید و آن را بر حسب زمان رسم کنید.
- دو نوسانگر با نیروهای واداشته‌ی $F_D = ۰/۵$ و $F_D = ۱/۲$ در نظر بگیرید. با فرض پارامترهای زیر برای دو نوسانگر

$$q = ۱/۲,\ l = ۹/۸\ g = ۹/۸\ \Omega_D = ۲/۳,\ \theta_{۰,۱} = ۰/۲,\ \theta_{۰,۲} = \theta_{۰,۱} + ۰/۰۰۱$$

مقدار اختلاف زاویه‌ای لگاریتمی این دو نوسانگر، $\log(\Delta\theta(t))$ ، را بر حسب زمان رسم کنید.

۳.۴.۲ الگوریتم رانگ‌ـ کوتای مرتبه ۲ و مرتبه ۴ برای حرکت یک پاندول

الگوریتم مرتبه ۲:

$$\begin{aligned}
&k_۱^\theta = \omega_i, && k_۱^\omega = -\tfrac{g}{l}\theta_i \\
&k_۲^\theta = \omega_i + \Delta t k_۱^\omega, && k_۲^\omega = -\tfrac{g}{l}\left(\theta_i + \Delta t k_۱^\theta\right) \\
&\theta_{i+۱} = \theta_i + \tfrac{\Delta t}{۲}\left(k_۱^\theta + k_۲^\theta\right), && \omega_{i+۱} = \omega_i + \tfrac{\Delta t}{۲}\left(k_۱^\omega + k_۲^\omega\right)
\end{aligned} \qquad (۳۲.۲)$$

الگوریتم مرتبه ۴:

(۳۳.۲)

$$\begin{aligned}
&k_۱^\theta = \omega_i, && k_۱^\omega = -\tfrac{g}{l}\theta_i \\
&k_۲^\theta = \omega_i + \tfrac{\Delta t}{۲} k_۱^\omega, && k_۲^\omega = -\tfrac{g}{l}\left(\theta_i + \tfrac{\Delta t}{۲} k_۱^\theta\right) \\
&k_۳^\theta = \omega_i + \tfrac{\Delta t}{۲} k_۲^\omega, && k_۳^\omega = -\tfrac{g}{l}\left(\theta_i + \tfrac{\Delta t}{۲} k_۲^\theta\right) \\
&k_۴^\theta = \omega_i + \Delta t\ k_۳^\omega, && k_۴^\omega = -\tfrac{g}{l}\left(\theta_i + \Delta t\ k_۳^\theta\right) \\
&\theta_{i+۱} = \theta_i + \tfrac{\Delta t}{۶}\left(k_۱^\theta + ۲k_۲^\theta + ۲k_۳^\theta + k_۴^\theta\right), && \omega_{i+۱} = \omega_i + \tfrac{\Delta t}{۶}\left(k_۱^\omega + ۲k_۲^\omega + ۲k_۳^\omega + k_۴^\omega\right)
\end{aligned}$$

کتاب‌نامه

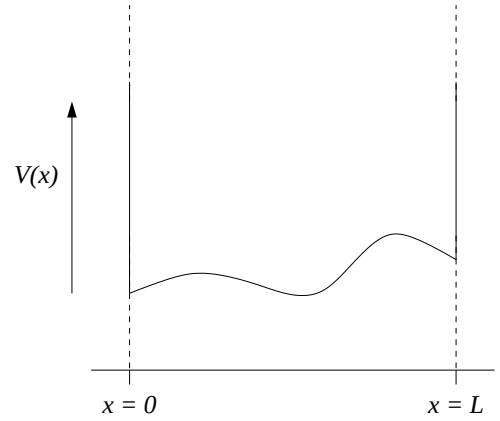
فصل ۳

کار با آرایه ها در پایتون: numpy

۱.۳ تعریف آرایه در numpy

۲.۳ مکانیک کوانتومی ماتریسی

یک چاه کوانتومی یک بعدی با دیواره‌های بی‌نهایت با طول L در نظر بگیرید. اگر پتانسیل داخل چاه دیگر صفر نباشد و با x تغییر کند (شکل ۱.۳)، دیگر نمی‌توانیم مسئله ویژه مقداری $\hat{H}\psi(x) = E\psi(x)$ را به صورت تحلیلی حل کنیم.



شکل ۱.۳: یک چاه کوانتومی یک بعدی با دیواره‌های بی نهایت. پتانسیل در داخل چاه کوانتومی با x تغییر می‌کند. در چاه کوانتومی معمولی در داخل چاه پتانسیل ثابت است، $V = ۰$ ، در حالی که در اینجا ما فرض می‌کنیم پتانسیل با x تغییر می‌کند، $V(x)$.

همان‌طور که می‌دانیم الکترون در یک چاه پتانسیل معمولی با دیواره‌های بی‌نهایت:

$$V = \begin{cases} ۰, & ۰ < x < L, \\ \infty, & x \leq ۰, \text{or} x \geq L. \end{cases} \quad (۱.۳)$$

دارای ویژه حالت‌های سینوسی به شکل زیر است:

$$\phi_n = \sqrt{\frac{۲}{L}} \sin(\frac{n\pi x}{L}) \quad (۲.۳)$$

این ویژه حالت‌ها دارای این خاصیت هستند که بر روی دیواره‌ها یعنی $x = ۰$ و $x = L$ صفر می‌شوند.
برای وقتی که پتانسیل داخل چاه با مکان تغییر کند:

$$V = \begin{cases} V(x), & ۰ < x < L, \\ \infty, & x \leq ۰, \text{or} x \geq L. \end{cases} \quad (۳.۳)$$

در این حالت به هامیلتونی یک جمله پتانسیل نیز اضافه می‌شود:

$$\hat{H} = -\frac{\hbar^۲}{۲M}\frac{\mathrm{d}^۲}{\mathrm{d}x^۲} + V(x).$$

باز به دلیل دیواره‌های بی‌نهایت چاه، ویژه حالت‌ها حتما باید در $x = ۰$ و $x = L$ صفر شوند. بنابراین می‌توان ویژه حالت‌ها را برای چنین چاه‌ی به صورت بسط سینوسی در نظر گرفت چرا که $\sin(\frac{n\pi x}{L})$ برای روی دیواره‌های صفر است. پس ویژه حالت k ام به صورت زیر خواهد بود

$$\psi_k(x) = \sqrt{\frac{۲}{L}}\sum_{n=۱}^{\infty} C_{n,k}\sin(\frac{n\pi x}{L}) \qquad (۴.۳)$$

که در اینجا C_n ضرایب بسط هستند و اصطلاحا $\sqrt{\frac{۲}{L}}\sin(\frac{n\pi x}{L})$ پایه های ما را تشکلیل می‌دهد. پایه‌هایی که انتخاب کرده‌ایم راست بهنجار هستند:

$$\frac{۲}{L}\int_{۰}^{L}\sin\frac{\pi m x}{L}\,\sin\frac{\pi n x}{L}\,\mathrm{d}x = \begin{cases} ۱ & \text{if } m = n، \\ ۰ & \text{otherwise}، \end{cases}$$

حال اگر H یعنی هامیلتونی را بر روی $\psi_k(x)$ اثر دهیم، معادله ویژه مقداری (شرودینگر) به صورت زیر خواهد شد:

$$\hat{H}\Big\{\sum_n C_{n,k}\sqrt{\frac{۲}{L}}\sin(\frac{n\pi x}{L})\Big\} = E_k\sum_n C_{n,k}\sqrt{\frac{۲}{L}}\sin(\frac{n\pi x}{L}) \qquad (۵.۳)$$

حال دو طرف معادله را در $\sqrt{\frac{۲}{L}}\sin(\frac{m\pi x}{L})$ ضرب می‌کنیم و از ۰ تا L انتگرال می‌گیریم. به دلیل خاصیت راست بهنجاری پایه‌هایی که انتخاب کردیم به رابطه‌ی زیر می‌رسیم:

$$\frac{۲}{L}\sum_n\int_{۰}^{L} C_{n,k}\sin(\frac{m\pi x}{L})\hat{H}\sin(\frac{n\pi x}{L})dx = E_k C_{m,k} \qquad (۶.۳)$$

با تعریف عناصر هامیلتونی در این پایه‌ها به صورت زیر:

$$\begin{aligned} H_{m,n} &= \frac{۲}{L}\int_{۰}^{L} \sin(\frac{m\pi x}{L})\hat{H}\sin(\frac{n\pi x}{L})dx \\ &= \frac{۲}{L}\int_{۰}^{L} \sin(\frac{m\pi x}{L})\Big[-\frac{\hbar^{۲}}{۲M}\frac{\mathrm{d}^{۲}}{\mathrm{d}x^{۲}}+V(x)\Big]\sin(\frac{n\pi x}{L})dx \end{aligned} \qquad (۷.۳)$$

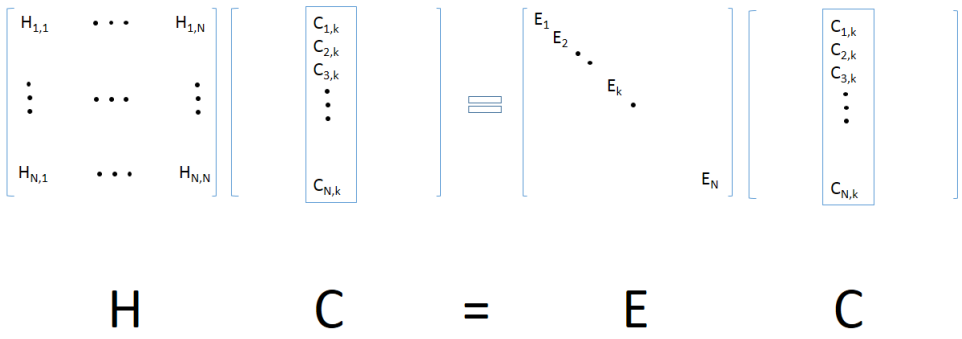
رابطه ویژه مقدار به صورت زیر می‌شود:

$$\sum_{n} H_{m,n}C_{n,k} = E_k C_{m,k} \qquad (۸.۳)$$

که به صورت کلی‌تر می‌توان با نمایش ضرب ماتریسی رابطه بالا را نشان داد:

$$\mathbf{HC} = \mathbf{EC} \qquad (۹.۳)$$

که ستون kام ماتریس $\mathbf{C}$ ویژه حالت k را نشان می‌دهد. و ماتریس $\mathbf{E}$ یک ماتریس قطری است که اعضای روی قطر آن نمایش‌گر ویژه مقادیر هستند. در شکل ۲.۳ این موضوع به وضوح توضیح داده شده است:



شکل ۲.۳

۱.۲.۳ نمایش معادله ویژه مقدار با استفاده از نمادهای کت و برا

تمرین ۴ (قسمت اول): محاسبه ویژه‌حالت‌ها و ویژه مقادیر یک چاه پتانسیل با پتانسیل متغیر $V(x) = \frac{ax}{L}$ با استفاده از پایه‌های سینوسی.

در این تمرین سه ویژه مقدار اول انرژی را بر حسب الکترون ولت چاپ کنید. همچنین چگالی سه ویژه حالت ($|\psi(x)|^۲$) را بر حسب x رسم کنید.

- نکته۱:داخل چاه پتانسیل به طول L با دیوارهای بی‌نهایت پتانسیل متغیر $V(x) = \frac{ax}{L}$ را در نظر بگیرید. تابعی بنویسید که عناصر ماتریس هامیلتونی $H_{m,n}$ را بر حسب n, m و L محاسبه کند. برای این منظور از انتگرال زیر می‌توان استفاده کرد:

$$\int_۰^L x \ \sin\frac{\pi m x}{L} \ \sin\frac{\pi n x}{L} \mathrm{d}x = \begin{cases} ۰ & \text{if } m \neq n \text{ and both of } m \text{ and } n \text{ are even or odd} \\ -(\frac{۲L}{\pi})^۲ \frac{mn}{(m^۲-n^۲)^۲} & \text{if } m \neq n \text{ and one of } m \text{ or } n \text{ is even and the other is odd} \\ \frac{L^۲}{۴} & \text{if } m = n \end{cases}$$

در اینجا فرض کنید که $L = ۵\text{Å}$ و $a = ۱۰eV$ و محاسبات را با استفاده از واحد اتمی که در آن $\hbar = m_e = e = ۱$ انجام دهید. در واحد اتمی واحد طول شعاع بوهر است که برابر با $a_۰ = ۰٫۵۲۹۱۷۷۲۱\text{Å}$ است و واحد انرژی هارتری است که برابر با $۱Hartree = ۲۷٫۲۱۱۳۸۳eV$ است.

- نکته۲:در تئوری برای حل این مسئله باید ابعاد ماتریس H بی‌نهایت باشد یا به عبارتی بسط تابع موج باید تا بی‌نهایت ادامه پیدا کند:

$$\psi_k(x) = \sqrt{\frac{۲}{L}} \sum_{n=۱}^{\infty} C_{n,k} \sin(\frac{n\pi x}{L}) \qquad (۱۰.۳)$$

ولی در عمل لازم نیست که تا بی‌نهایت بسط را ادامه دهیم. بدین منظور فرض کنید بسط را تا ۱۰ جمله اول ادامه دهیم. بنابراین ماتریس H یک ماتریس ۱۰×۱۰ خواهد بود و ویژه مقادیر

تا $M = ۱۰$ بسط داده می‌شوند:

$$\psi_k(x) = \sqrt{\frac{۲}{L}} \sum_{n=۱}^{M=۱۰} C_{n,k} \sin(\frac{n\pi x}{L}) \qquad (۱۱.۳)$$

۳.۳ حل معادله شرودینگر در فضای حقیقی

اگر به معادله شرودینگر دقت کنیم، به سادگی متوجه این موضوع خواهیم شد که معادله شرودینگر، یک معادله دیفرانسیلی است که مانند تمام معادلات دیفرانسیلی آن را می‌توان به صورت عددی حل کرد. بنابراین به جای این که از پایه‌ها برای بسط توابع موج استفاده کنیم (روشی که در بخش ابتدایی توضیح داده شد)، می‌توانیم از روش‌های عددی استفاده کرد. روش عددی که برای حل معادله شرودینگر استفاده می‌شود به روش اختلاف محدود فضای حقیقی [۱] معرف است. در این روش فضای حقیقی (واقعی) را مش بندی می‌کنیم و با توجه به اینکه مقدار پتانسیل را در هر نقطه داریم، می‌توانیم مقدار تابع موج را نیز در هر نقطه پیدا کنیم. برای توضیح بیشتر این روش یک مثال ساده، یعنی نوسانگر هماهنگ ساده، را با این روش حل می‌کنیم.

۱.۳.۳ حل نوسانگر ساده یک بعدی به روش فضای حقیقی

معادله شرودینگر (مستقل از زمان) یک نوسانگر یک بعدی به صورت زیر است:

$$-\frac{\hbar^۲}{۲m}\frac{d^۲}{dx^۲}\psi(x) + \frac{۱}{۲}kx^۲\psi(x) = E\psi(x) \qquad (۱۲.۳)$$

با انتخاب $\hbar = m = k = ۱$ معادله به صورت ساده تر زیر می‌شود:

$$-\frac{۱}{۲}\frac{d^۲}{dx^۲}\psi(x) + \frac{۱}{۲}x^۲\psi(x) = E\psi(x) \qquad (۱۳.۳)$$

سپس برای سادگی بیشتر معادله را در ۲ ضرب می‌کنیم:

$$-\frac{d^۲}{dx^۲}\psi(x) + x^۲\psi(x) = ۲E\psi(x) \qquad (۱۴.۳)$$

[۱] Real-Space Finite-Difference method

همان طور که از معادله مشخص است، می‌بایست برای محاسبه مشتق دوم $\psi(x)$ چاره‌ای پیدا کنیم. مشتق دوم یک تابع، $f(x)$ ، را می‌توان به صورت زیر تقریب زد:

$$\frac{d^۲f}{dx^۲} = \frac{f(x+h) - ۲f(x) + f(x-h)}{h^۲} + O(h^۲) \qquad (۱۵.۳)$$

که h گام (همان Δx) یا همان اندازه مش بندی فضایی است. برای اینکه مش بندی مشخصی را در نظر بگیریم، ابتدا برای متغیر x یک کمینه ($X_{\min}$) و بیشینه ($X_{\max}$) در نظر می‌گیریم که در حقیقت با آنها مرز تابع (موج) را مشخص می‌کنیم. اگر فضای حقیقی را به تعداد N مش بندی کنیم. فاصله‌ی بین نقاط فضا (نقاط مش) یعنی همان h به صورت زیر خواهد شد:

$$h = \frac{X_{\max} - X_{\min}}{N} \qquad (۱۶.۳)$$

و بنابراین مکان نقطه i ام مش به صورت زیر خواهد بود:

$$x_i = X_{\min} + ih \quad i = ۱, ۲, ..., N-۱ \qquad (۱۷.۳)$$

پاورقی: مشتق دوم یک تابع به صورت عددی

به راحتی می‌توان با استفاده از بسط تیلور معادله ۱۵.۳ را بدست آورد. $f(x-h)$ و $f(x+h)$ را بسط تیلور می‌دهیم:

$$f(x-h) \approx f(x) - f'(x)h + \frac{f''(x)}{۲!}h^۲ + O(h^۳) \qquad (۱۸.۳)$$

$$f(x+h) \approx f(x) + f'(x)h + \frac{f''(x)}{۲!}h^۲ + O(h^۳) \qquad (۱۹.۳)$$

با جمع دو معادله به معادله زیر می‌رسیم:

$$f(x-h) + f(x+h) \approx ۲f(x) + f''(x)h^۲ + O(h^۴) \qquad (۲۰.۳)$$

که دقیقا همان معادله ۱۵.۳ است.

پس معادله شرودینگر ۱۴.۳ برای نقطه‌ی k ام مش به صورت زیر خواهد شد:

$$-\frac{\psi(x_k+h)-۲\psi(x_k)+\psi(x_k-h)}{h^۲}+x_k^۲\psi(x_k)=۲E\psi(x_k) \qquad (۲۱.۳)$$

یا به طور خلاصه‌تر

$$-\frac{\psi_{k+۱}-۲\psi_k+\psi_{k-۱}}{h^۲}+x_k^۲\psi_k=۲E\psi_k \qquad (۲۲.۳)$$

که در اینجا

$$\begin{aligned}\psi_k &\equiv \psi(x_k)\\ \psi_{k+۱} &\equiv \psi(x_k+h)\\ \psi_{k-۱} &\equiv \psi(x_k-h)\end{aligned}$$

بنابراین قسمت انرژی جنبشی را می‌توان برای هر نقطه مش به صورت زیر نوشت:

$$c_۱\psi_{k+۱}+c_۰\psi_k+c_۱\psi_{k-۱} \qquad (۲۳.۳)$$

که $c_۰=۲/h^۲$ و $c_۱=-۱/h^۲$. با فرض شرط مرزی $\psi(x_۰)=\psi_۰=۰$ و $\psi(x_N)=\psi_N=۰$ برای نقاط مختلف مش روابط زیر را (برای انرژی جنبشی) می‌توان نوشت:

$$\begin{aligned}k=۱ \quad & c_۰\psi_۱+c_۱\psi_۲\\ k=۲ \quad & c_۱\psi_۱+c_۰\psi_۲+c_۱\psi_۳\\ k=۳ \quad & c_۱\psi_۲+c_۰\psi_۳+c_۱\psi_۴\\ \ldots \quad & \ldots\\ k=N-۱ \quad & c_۱\psi_{N-۲}+c_۰\psi_{N-۱}\end{aligned} \qquad (۲۴.۳)$$

با توجه به رابطه‌های بالا، می‌توان این روابط را به صورت ماتریسی نوشت:

$$\begin{bmatrix} c_0 & c_1 & 0 & 0 & \dots \\ c_1 & c_0 & c_1 & 0 & \dots \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ 0 & & \ddots & \ddots & c_1 \\ 0 & 0 & \dots & c_1 & c_0 \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-2} \\ \psi_{N-1} \end{bmatrix} \tag{۲۵.۳}$$

پس معادله ۲۲.۳ را می‌توان به صورت ماتریس زیر نوشت:

$$\begin{bmatrix} c_0+V_1 & c_1 & 0 & 0 & \dots \\ c_1 & c_0+V_2 & c_1 & 0 & \dots \\ \vdots & c_1 & c_0+V_k & \ddots & \vdots \\ 0 & & \ddots & \ddots & c_1 \\ 0 & 0 & \dots & c_1 & c_0+V_{N-1} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-2} \\ \psi_{N-1} \end{bmatrix} = 2E \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-2} \\ \psi_{N-1} \end{bmatrix} \tag{۲۶.۳}$$

که در اینجا

$$V_k = x_k^2$$

بنابراین به راحتی در پایتون و با استفاده از کتابخانه numpy می‌توانیم ماتریس که در معادله بالا ظاهر می‌شود را قطر کرد. برای مثال برنامه زیر ۵ ویژه مقدار اول ماتریس بالا را به ما می‌دهد:

```
import numpy as np
import numpy.linalg as la
N=50
Xmin=-10.0
Xmax=10.0
h=(Xmax-Xmin)/N
x=np.arange(Xmin,Xmax,h)
c0=2/(h**2)
c1=-1/(h**2)
T=c0*np.diag(np.ones(N-1))    \
  +c1*np.diag(np.ones(N-2),1)+c1*np.diag(np.ones(N-2),-1)
V=np.diag(x[1:,]**2)
```

```
H=T+V
val=la.eigvalsh(H)
print(val[0:5])
```

: real_space.py

با توجه به اینکه ویژه‌مقادیر یک نوسانگر هارمونیک به صورت زیر می‌باشد:

$$E_n = (n + \frac{۱}{۲})\hbar\omega \qquad (۲۷.۳)$$

و با توجه به اینکه در اینجا ما فرض کردیم $\hbar = k = m = ۱$ است، و از طرفی هامیلتونی را در ۲ ضرب کردیم، ویژه مقادیری که برنامه به ما می‌دهد باید با ویژه مقادیر زیر قابل مقایسه باشد:

$$E_n = ۲n + ۱ \qquad (۲۸.۳)$$

به عبارتی اعداد فرد ویژه مقادیر هستند. جواب‌هایی که از اجرای برنامه بالا بدست می‌آید را در جدول ... می‌توان مشاهد کرد. وقتی تعداد نقاط مش‌بندی یعنی N را بالا می‌بریم به جواب‌های دقیق‌تر می‌رسیم.

N	$E_۰$	$E_۱$	$E_۲$	$E_۳$	$E_۴$
۵۰	۰,۹۸۹۸۹۶۸۶	۲,۹۴۹۰۵۱۵۲	۴,۸۶۶۲۲۳۲۲	۶,۷۳۹۹۱۵۹۲	۸,۵۶۸۴۴۱۹۴
۱۰۰	۰,۹۹۷۴۹۳۷۰	۲,۹۸۷۴۴۳۰۳	۴,۹۶۷۲۷۷۲۶	۶,۹۳۶۹۱۷۴۵	۸,۸۹۶۲۸۲۵۷
۵۰۰	۰,۹۹۹۸۹۹۹۹	۲,۹۹۹۴۹۹۹۱	۴,۹۹۸۶۹۹۶۵	۶,۹۹۷۴۹۹۰۹	۸,۹۹۵۸۹۸۱۱
۱۰۰۰	۰,۹۹۹۹۷۵۰۰	۲,۹۹۹۸۷۴۹۹	۴,۹۹۹۶۷۴۹۸	۶,۹۹۹۳۷۴۹۴	۸,۹۹۸۹۷۴۸۸
۲۰۰۰	۰,۹۹۹۹۹۳۷۵	۲,۹۹۹۹۶۸۷۵	۴,۹۹۹۹۱۸۷۵	۶,۹۹۹۸۴۳۷۵	۸,۹۹۹۷۴۳۷۴

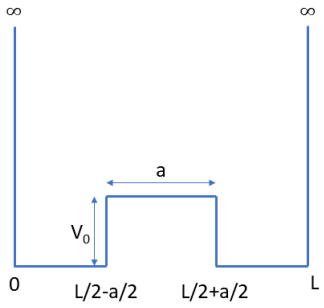
تمرین ۴(قسمت دوم): محاسبه ویژه‌حالت‌ها و ویژه مقادیر یک چاه پتانسیل با پتانسیل متغیر $V(x) = \frac{ax}{L}$ با استفاده از روش فضای حقیقی.

در این تمرین سه ویژه مقدار اول انرژی را بر حسب الکترون ولت چاپ کنید. همچنین چگالی سه ویژه حالت ($|\psi(x)|^۲$) را بر حسب x رسم کنید. همچنین نتایج خود را با محاسبه قسمت اول مقایسه کنید. برای مقاسه هم ویژه مقادیر و هم ویژه حالت را مقایسه کنید. برای مقایسه ویژه حالت‌ها، چگالی سه ویژه حالت قسمت دوم را با چگالی ویژه حالت‌های قسمت اول با هم رسم کنید.

تمرین : محاسبه ویژه حالت‌ها و ویژه مقادیر برای چاه پتانسیل نامتناهی با یک سد در میان یک چاه پتانسیل نامتناهی با یک سد در میان این چاه در نظر بگیرید. مقدار پتانسیل در نقاط مختلف به صورت زیر است:

$$V(x)=\begin{cases}\infty & x<۰\\ ۰ & ۰<x<L/۲-a/۲\\ V. & L/۲-a/۲\le x\le L/۲+a/۲\\ ۰ & x>L/۲+a/۲\\ \infty & x>L\end{cases}$$

و شکل این چاه به صورت زیر خواهد بود:



شکل ۳.۳

مقدار طول L و a را به ترتیب برابر با ۵ و ۱ آنگستروم و مقدار $V.$ را ۵۰ اکترون ولت در نظر بگیرید.

الف-

با توجه به مطالب فصل ۳ جزوه، به روش بسط تابع موج بر حسب توابع ویژه چاه نامتناهی مقدار ۳ ویژه حالت اول را برحسب اکترون ولت محاسبه کنید. همچنین ۳ ویژه حالت (۳ ویژه تابع) اول را رسم کنید. تعداد تابع‌های پایه را ۵۰۰ در نظر بگیرید.

ب-

با توجه به مطالب فصل ۳ جزوه، به روش فضای حقیقی مقدار ۳ ویژه حالت اول را بر حسب اکترون ولت محاسبه کنید. تعداد مش رو ۴۰۰ در نظر بگیرید.

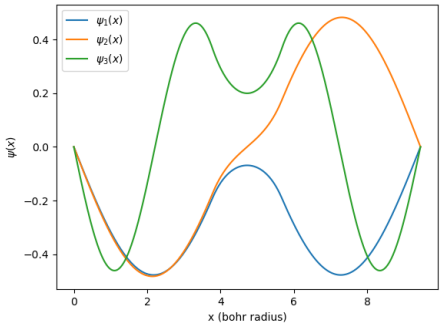
راهنمایی:

سه ویژه مقدار اول باید برابر با:

۲۷/۴۰۰۵, ۷/۳۲۴۸, ۷/۱۰۰۱

(بر حسب الکترون ولت)

و سه ویژه تابع اول به صورت:



شکل ۴.۳

هامیلتونی را می‌توان به صورت دو جمله جنبشی و پتانسیل نوشت یعنی : $H_{mn} = T_{mn} + V_{mn}$

از معادله زیر برای بدست آوردن قسمت V_{mn} هامیلتونی استفاده کنید:

$$\frac{۲}{L}\int_{۰}^{L} \sin(\frac{m\pi x}{L})V(x)\sin(\frac{n\pi x}{L})dx$$
$$= \begin{cases} \frac{۲V_{۰}}{\pi}\{\frac{۱}{m-n}\cos((m-n)\frac{\pi}{۲})\sin(\frac{\pi}{L}(m-n)\frac{a}{۲}) - \frac{۱}{m+n}\cos((m+n)\frac{\pi}{۲})\sin(\frac{\pi}{L}(m+n)\frac{a}{۲})\} & n \neq m \\ \frac{V_{۰}}{L}\{a - \frac{L}{n\pi}\cos(n\pi)\sin(n\pi\frac{a}{L})\} & n = m \end{cases}$$

برای قسمت جنبشی نیز به راحتی می‌توان نشان داد:

$$T_{mn} = \frac{۱}{۲}(\frac{n\pi}{L})^{۲}\delta_{mn}$$

۴.۳ حل معادله شرودینگر وابسته به زمان

معادله شرودینگر تک ذره در یک بعد به صورت زیر است:

$$-\frac{\hbar^{۲}}{۲M}\frac{\partial^{۲}\psi}{\partial x^{۲}} = i\hbar\frac{\partial\psi}{\partial t} \qquad (۲۹.۳)$$

معادله شرودینگر در واقع یک معادله پخش است به همین دلیل ابتدا به چگونگی حل معادله پخش، که به صورت زیر است، می‌پردازیم.

$$\frac{\partial \psi}{\partial t} = D\frac{\partial^۲ \psi}{\partial x^۲} \qquad (۳۰.۳)$$

که در این معادله D ضریب پخش است. برای حل این معادله می‌توان از روش FTCS [۲] استفاده کنیم. در این روش مشتق دوم را به صورت عددی به شکل زیر تقریب می‌زنیم:

$$\frac{\partial^۲ \phi}{\partial x^۲} = \frac{\phi(x+a,t) + \phi(x-a,t) - ۲\phi(x,t)}{a^۲} \qquad (۳۱.۳)$$

بنابراین

$$\frac{d\phi}{dt} = \frac{D}{a^۲}[\phi(x+a,t) + \phi(x-a,t) - ۲\phi(x,t)] \qquad (۳۲.۳)$$

معادله بالا را می‌توان به صورت زیر دید:

$$\frac{d\phi}{dt} = f(\phi, t) \qquad (۳۳.۳)$$

پس طبق روش اویلر:

$$\phi(t+\Delta t) = \phi(t) + \Delta t f(\phi, t) \qquad (۳۴.۳)$$

و یا به صورت کامل‌تر:

$$\phi(x, t+\Delta t) = \phi(x,t) + \Delta t \frac{D}{a^۲}[\phi(x+a,t) + \phi(x-a,t) - ۲\phi(x,t)] \qquad (۳۵.۳)$$

مثال: حل معادله حرارت

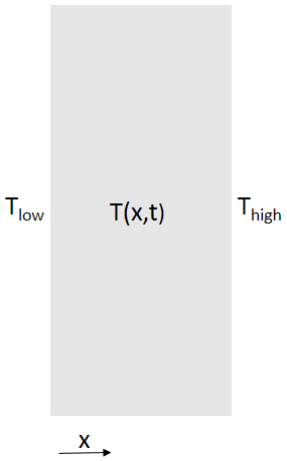
معادله حرارت در واقع شبیه معادله پخش است و نشان می‌دهد که چگونه دما در طی گذر زمان در

[۲] Forward-time centered-space method

یک جسم تغییر می‌کند. این معادله به صورت زیر می‌باشد:

$$\frac{\partial T}{\partial t} = D\frac{\partial^۲ T}{\partial x^۲} \qquad (۳۶.۳)$$

برای مثال شکل زیر یک تیغه فولادی را نشان می‌دهد که دریک طرف آن دما پایین T_{low} و در طرف دیگر آن دما بالاتر T_{high} است. با فرض اینکه دما در دو طرف تیغه ثابت نگه داشته می‌شود، می‌توان از معادله بالا برای محاسبه دما، $T(x,t)$ ، در هر مکان (در عرض تیغه، x) و در هر زمان استفاده کرد.



شکل ۵.۳

برنامه زیر مقدار $T(x,t)$ را در زمان‌های مختلف $t_۱ = ۰/۰۱s,\ t_۲ = ۰/۱s,\ t_۳ = ۰/۴s\ t_۴ = ۱/۰s$ در یک تیغه فولادی محاسبه می‌کند. در این برنامه فرض شده است که $T_{low} = ۰$ و $T_{high} =$ ۵۰ است. عرض تیغه به ۱۰۰ قسمت تقسیم شده است (یعنی اندازه مش بندی ۱۰۰). عرض تیغه $L = ۰/۰۱m$ و ضریب پخش گرمایی آن $D = ۴/۲۵ \times ۱۰^{-۶} m^۲ s^{-۱}$ است که مربوط به فولاد ضد زنگ می‌باشد.

```
from numpy import empty
from matplotlib.pyplot import plot,xlabel,ylabel,show, legend

# Constants
L = 0.01        # Thickness of steel in meters
D = 4.25e-6     # Thermal diffusivity
```

```
N = 100          # Number of divisions in grid
a = L/N          # Grid spacing
h = 1e-4         # Time-step
epsilon = h/1000

Tlo = 0.0        # Low temperature in Celcius
Tmid = 20.0      # Intermediate temperature in Celcius
Thi = 50.0       # Hi temperature in Celcius

t1 = 0.01
t2 = 0.1
t3 = 0.4
t4 = 1.0
t5 = 10.0
tend = t5 + epsilon

# Create arrays
T = empty(N+1,float)
T[0] = Thi
T[N] = Tlo
T[1:N] = Tmid
Tp = empty(N+1,float)
Tp[0] = Thi
Tp[N] = Tlo

# Main loop
t = 0.0
c = h*D/(a*a)
while t<tend:

    # Calculate the new values of T
    for i in range(1,N):
        Tp[i] = T[i] + c*(T[i+1]+T[i-1]-2*T[i])
    T,Tp = Tp,T
    t += h

    # Make plots at the given times
    if abs(t-t1)<epsilon:
        plot(T, label="t="+str(t1))
    if abs(t-t2)<epsilon:
        plot(T,label="t="+str(t2))
    if abs(t-t3)<epsilon:
```

```
49            plot(T,label="t="+str(t3))
      if abs(t-t4)<epsilon:
51            plot(T,label="t="+str(t4))
      if abs(t-t5)<epsilon:
53            plot(T,label="t="+str(t5))

55 xlabel("x")
   ylabel("T")
57 legend()
   show()
```

: prog_ch2/heat.py

در هر زمان طبق معادلات بالا باید برای تمام نقاط مکانی به صورت زیر عمل کنیم:

```
for i in range(1,N):
2     Tp[i]= T[i]+c*(T[i+1]+T[i-1]-2*T[i])
```

ولی محاسبات به این شکل سنگین می‌شود. به همین دلیل می‌توان از مفهوم برش [۳] استفاده کرد:

```
Tp[1:N]= T[1:N]+c*(T[2:N+1]+T[0:N-1]-2*T[1:N])
```

۱.۴.۳ حل معادله‌ی شرودینگر به روش Crank-Nielson

می‌توان نشان داد که روش FTCS ، روش پایداری نیست. روش بهتر برای حل معادله‌ی شرودینگر روش Crank-Nielson است که می‌توان به صورت زیر آن را توضیح داد. همان طور که بیان شد، برای معادله شرودینگر طبق روش FTCS عمل کرد:

$$\psi(x,t+\Delta t)=\psi(x,t)+\Delta t\frac{i\hbar}{2ma^2}[\psi(x+a,t)+\psi(x-a,t)-2\psi(x,t)] \quad (۳۷.۳)$$

این معادله را می‌توان برای زمان $t-\Delta t$ نیز تکرار کرد:

$$\psi(x,t-\Delta t)=\psi(x,t)-\Delta t\frac{i\hbar}{2ma^2}[\psi(x+a,t)+\psi(x-a,t)-2\psi(x,t)] \quad (۳۸.۳)$$

[۳] slicing

حال اگر در معادله بالا $t + \Delta t$ را با t جایگزین کنیم ($t \longrightarrow t + \Delta t$) معادله زیر بدست می‌آید (طرف راست معادله را ابتدا باز نویسی می‌کنیم):

(۳۹.۳)

$$\psi(x,t+\Delta t)-\Delta t\frac{i\hbar}{۲ma^{۲}}[\psi(x+a,t+\Delta t)+\psi(x-a,t+\Delta t)-۲\psi(x,t+\Delta t)]=\psi(x,t)$$

حال دو طرف معادله ۳۷.۳ و ۳۹.۳ را باهم جمع و بر ۲ تقسیم می‌کنیم:

$$\begin{aligned}\psi(x,t+\Delta t) \quad - \quad & \Delta t\frac{i\hbar}{۴ma^{۲}}[\psi(x+a,t+\Delta t)+\psi(x-a,t+\Delta t)-۲\psi(x,t+\Delta t)] \\ = \quad & \psi(x,t)+\Delta t\frac{i\hbar}{۴ma^{۲}}[\psi(x+a,t)+\psi(x-a,t)-۲\psi(x,t)] \quad (۴۰.۳)\end{aligned}$$

مسلما این معادله را نمی‌توان به صورت جایگزینی قبلی انجام داد و بطور ساده حل کرد. برای توضیح روش Crank-Nicolson به مسله ساده ذره در چاه پتانسیل با دیواره‌های بی‌نهایت بر می‌گردیم. در این صورت ψ بر روی مرز دیواره‌ها یعنی $x = ۰$ و $x = L$ برابر با صفر است. اگر فاصله بین $x = ۰$ تا $x = L$ را با گام‌های a تقسیم‌بندی کنیم، $\psi(x,t)$ در هر زمان را می‌توان به صورت یک بردار نمایش داد:

$$\psi(t)=\begin{bmatrix}\psi(a,t)\\ \psi(۲a,t)\\ \psi(۳a,t)\\ \vdots\\ \psi((N-۱)a,t)\end{bmatrix} \qquad (۴۱.۳)$$

حال با تعریف دو ماتریس سه قطری [۴] A و B به صورت زیر:

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & & & \\ a_2 & a_1 & a_2 & & \\ & a_2 & \ddots & \ddots & \\ & & \ddots & \ddots & a_2 \\ & & & a_2 & a_1 \end{bmatrix} \tag{۴۲.۳}$$

$$\mathbf{B} = \begin{bmatrix} b_1 & b_2 & & & \\ b_2 & b_1 & b_2 & & \\ & b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_2 \\ & & & b_2 & b_1 \end{bmatrix} \tag{۴۳.۳}$$

که در اینجا a_1 ، a_2 ، b_1 و b_2 به صورت زیر تعریف می‌شوند:

$$\begin{aligned} a_1 &= 1 + \Delta t \frac{i\hbar}{2ma^2} \\ a_2 &= -\Delta t \frac{i\hbar}{4ma^2} \\ b_1 &= 1 - \Delta t \frac{i\hbar}{2ma^2} \\ b_2 &= \Delta t \frac{i\hbar}{4ma^2} \end{aligned}$$

با این تعاریف می‌توان معادله ۴۰.۳ را به صورت ساده زیر نوشت:

$$\mathbf{A}\psi(t + \Delta t) = \mathbf{B}\psi(t) \tag{۴۴.۳}$$

[۴] Tridiagonal matrix

در واقع اگر معادله بالا را بازتر کنیم به صورت زیر خواهد شد:

(۴۵.۳)

$$\begin{bmatrix} a_1 & a_2 & & & \\ a_2 & a_1 & a_2 & & \\ & a_2 & \ddots & \ddots & \\ & & \ddots & \ddots & a_2 \\ & & & a_2 & a_1 \end{bmatrix} \begin{bmatrix} \psi(a,t+\Delta t) \\ \psi(2a,t+\Delta t) \\ \psi(3a,t+\Delta t) \\ \vdots \\ \psi(L-a,t+\Delta t) \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & & & \\ b_2 & b_1 & b_2 & & \\ & b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_2 \\ & & & b_2 & b_1 \end{bmatrix} \begin{bmatrix} \psi(a,t) \\ \psi(2a,t) \\ \psi(3a,t) \\ \vdots \\ \psi(L-a,t) \end{bmatrix}$$

به عبارتی معادله بالا را می‌توان به صورت یک چند معادله چند مچهول درنظر گرفت، یعنی:

$$\mathbf{A}\mathbf{x} = \mathbf{v} \tag{۴۶.۳}$$

که

$$\mathbf{x} = \begin{bmatrix} \psi(a,t+\Delta t) \\ \psi(2a,t+\Delta t) \\ \psi(3a,t+\Delta t) \\ \vdots \\ \psi(L-a,t+\Delta t) \end{bmatrix}$$

و

$$\mathbf{v} = \mathbf{B} \begin{bmatrix} \psi(a,t) \\ \psi(2a,t) \\ \psi(3a,t) \\ \vdots \\ \psi(L-a,t) \end{bmatrix} = \mathbf{B}\psi(t)$$

روش‌های زیادی برای حل چند معادله چند مجهول وچود دارد که معرف‌ترین آن روش حذف گوسی است. در کتابخانه numpy و scipy برای حل چند معادله چند مجهول از روشی استفاد می‌شوذ که در آن ماتریس $\mathbf{A}$ به ماتریس بالا و پایین مثلثی تجریه می‌شود. این روش به اختصار به LU decomposition بیان می‌شود که LU اشاره به lower upper یا همان پایین بالا (مثلثی) دارد. دستور solve در زیر کتابحانه

linalg با استفاده از این روش چند معادله چند مجهول را حل می‌کند. برای مثال معادله‌های زیر را درنظر بگیرید:

$$۲w + x + ۴y + z = -۴$$
$$۳w + ۴x - y - z = ۳$$
$$w - ۴x + y + ۵z = ۹$$
$$۲w - ۲x + y + ۳z = ۷$$

این معادلات را می‌توان به صورت ماتریسی نوشت:

$$\begin{bmatrix} ۲ & ۱ & ۴ & ۱ \\ ۳ & ۴ & -۱ & -۱ \\ ۱ & -۴ & ۱ & ۵ \\ ۲ & -۲ & ۱ & ۳ \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -۴ \\ ۳ \\ ۹ \\ ۷ \end{bmatrix}$$

برای حل این معادلات در scipy کافی است چنین عمل کنیم:

```
import numpy as np
import scipy.linalg as la
a=np.array([[2,1,4,1], [3,4,-1,-1], [1,-4,1,5], [2,-2,1,3]])
b=np.array([-4,3,9,7])
x=la.solve(a,b)
print(np.dot(a,x)-b) # to check if the answer is satisfied the equation
```

کتابخانه scipy برای ماتریس‌های خاص مانند ماتریس‌های سه قطری روش‌های سریع‌تری را به کار می‌برد. در scipy ماتریس‌های سه قطر در زیر مجموعه‌ی ماتریس‌های به نام ماتریس‌های banded طبق بندی می‌شود. یک ماتریس banded در واقع یک ماتریس است که اعضای قطر مرکزی و اعضای چندین قطربالاتر و پایین تر آن غیر صفر هستند و مابقی صفر هستند. برای واضح شدن موضوع به مثال زیر که یک ماتریس banded است توجه کنید. یک ماتریس ۶×۶ را در نظر بگیرید. اعضای این

ماتریس را به صورت عمومی می‌توان با $a_{i,j}$ نشان داد:

$$\mathbf{A} = \begin{bmatrix} a_{۰,۰} & a_{۰,۱} & a_{۰,۲} & a_{۰,۳} & a_{۰,۴} & a_{۰,۵} \\ a_{۱,۰} & a_{۱,۱} & a_{۱,۲} & a_{۱,۳} & a_{۱,۴} & a_{۱,۵} \\ a_{۲,۰} & a_{۲,۱} & a_{۲,۲} & a_{۲,۳} & a_{۲,۴} & a_{۲,۵} \\ a_{۳,۰} & a_{۳,۱} & a_{۳,۲} & a_{۳,۳} & a_{۳,۴} & a_{۳,۵} \\ a_{۴,۰} & a_{۴,۱} & a_{۴,۲} & a_{۴,۳} & a_{۴,۴} & a_{۴,۵} \\ a_{۵,۰} & a_{۵,۱} & a_{۵,۲} & a_{۵,۳} & a_{۵,۴} & a_{۵,۵} \end{bmatrix} \tag{۴۷.۳}$$

حال اگر اعضای قطر، بالای قطر و دو قطر پایین غیر صفر و مابقی صفر باشند این ماتریس به صورت زیر می‌شود:

$$\mathbf{A} = \begin{bmatrix} a_{۰,۰} & a_{۰,۱} & ۰ & ۰ & ۰ & ۰ \\ a_{۱,۰} & a_{۱,۱} & a_{۱,۲} & ۰ & ۰ & ۰ \\ a_{۲,۰} & a_{۲,۱} & a_{۲,۲} & a_{۲,۳} & ۰ & ۰ \\ ۰ & a_{۳,۱} & a_{۳,۲} & a_{۳,۳} & a_{۳,۴} & ۰ \\ ۰ & ۰ & a_{۴,۲} & a_{۴,۳} & a_{۴,۴} & a_{۴,۵} \\ ۰ & ۰ & ۰ & a_{۵,۳} & a_{۵,۴} & a_{۵,۵} \end{bmatrix} \tag{۴۸.۳}$$

برای معرفی ساده‌تر این ماتریس و حل سریعتر آن در scipy چنین عمل می‌کنیم: ابتدا دو پارامتر u و l را مشخص می‌کنیم که نشان دهنده‌ی تعداد قطرهای بالا و پایین [۵] است. در این مثال $u = ۱$ است که به معنی یک قطر بالا است و $l = ۲$ که به معنی دو قطر غیر صفر پایین است. سپس ماتریس ab که اعضای سطر آن به ترتیب اعضای قطرهای بالا، قطر مرکزی و قطرهای پایین ماتریس اصلی را نشان می‌دهد را به

[۵] u=upper, l=lower

صورت زیر معرفی می‌کنیم:

$$\mathbf{ab} = \begin{bmatrix} ۰ & a_{۰,۱} & a_{۱,۲} & a_{۲,۳} & a_{۳,۴} & a_{۴,۵} \\ a_{۰,۰} & a_{۱,۱} & a_{۲,۲} & a_{۳,۳} & a_{۴,۴} & a_{۵,۵} \\ a_{۱,۰} & a_{۲,۱} & a_{۳,۲} & a_{۴,۳} & a_{۵,۴} & ۰ \\ a_{۲,۰} & a_{۳,۱} & a_{۴,۲} & a_{۵,۳} & ۰ & ۰ \end{bmatrix} \tag{۴۹.۳}$$

می‌توان نشان داد که روش حذف گوسی برای حل معادله $\mathbf{Ax} = \mathbf{v}$ برای زمانی که $\mathbf{A}$ همانند مثال بالا یک ماتریس banded است، سریع‌تر است که در scipy.linalg ازهمین روش برای حل استفاده می‌شود. کافی‌است از دستور زیر برای حل استفاده کنیم که در آن به جای ماتریس $\mathbf{A}$ باید از ماتریس $\mathbf{ab}$ و از پارامترهای l و u برای معرفی ماتریس استفاده کنیم:

```
import scipy.linalg as la
x=la.solve_banded((l,u),ab,v)
```

مثال:ماتریس سه قطری ۱۰۰۰ × ۱۰۰۰ زیر را در نظر بگیرید:

$$\mathbf{A} = \begin{bmatrix} ۱ & ۲ & ۰ & ۰ & ۰ & \dots & ۰ \\ ۷ & ۸ & ۹ & ۰ & ۰ & \dots & ۰ \\ ۰ & ۷ & ۸ & ۹ & ۰ & \dots & ۰ \\ ۰ & ۰ & ۷ & ۸ & ۹ & \dots & ۰ \\ ۰ & ۰ & ۰ & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \dots & ۷ & ۸ & ۹ \\ ۰ & ۰ & ۰ & \dots & ۰ & ۳ & ۴ \end{bmatrix} \tag{۵۰.۳}$$

و

$$\mathbf{v} = \begin{bmatrix} ۱ \\ ۲ \\ ۲ \\ ۲ \\ \vdots \\ ۲ \\ ۲ \\ ۳ \end{bmatrix} \tag{۵۱.۳}$$

برای حل معادله خطی (چند معادله چند مجهول) زیر:

$$\mathbf{Ax} = \mathbf{v} \tag{۵۲.۳}$$

برنامه زیر این معادله را به دو روش معمولی و روشی که برای ماتریس‌های banded استفاده می‌شود، حل می‌کند و زمان محاسبات که در هر دو روش مصرف می‌شود را چاپ می‌کند:

```
import numpy as np
import scipy.linalg as la
import time


#------ Build [A] array and {v} column vector

m = 200    # size of array, make this 8000 to see time benefits

A=np.diag(7 * np.ones(m-1), -1)  + np.diag( 8 * np.ones(m) )  + np.diag
    ( 9 * np.ones(m-1), 1)

A[0, 0] = 1
A[0, 1] = 2
A[m-1, m-2] = 3
A[m-1, m-1] = 4


```

```
v = 2*np.ones(m)
v[0] = 1
v[-1] = 3

ti = time.time()
#------ Solve using scipy.linalg.solve

x = la.solve(A, v)      # solve A*x = v for x

print('scipy.linalg.solve time', time.time()-ti, 'seconds')


#------ Build ab array
ab = np.zeros((3,m))
ab[0,:] = 9
ab[1,:] = 8
ab[2,:] = 7
# Fix end points
ab[0,1] = 2
ab[1,0] = 1
ab[1,-1] = 4
ab[2,-2] = 3

ti = time.time()
#------ Solve using scipy.linalg.solve_banded

x = la.solve_banded ((1,1),ab,v)      # solve A*x = v for x

print('scipy.linalg.solve_banded', time.time()-ti, 'seconds')
```

: prog_ch2/matrix_banded.py

تمرین ۵: حل معادله شرودینگر وابسته به زمان برای یک ذره در چاه پتانسیل با دیواره‌های بی‌نهایت به روش Crank-Nicolson

فرض کنید تابع موج در چاه پتانسیل با دیواره‌های بی‌نهایت به طول L در زمان صفر به صورت زیر باشد:

$$\psi(x,\cdot) = \exp[-\frac{(x-x_\cdot)^2}{2\sigma^2}]e^{ikx}$$

عرض چاه $L = 100\text{Å}$ ، جرم ذره را برابر با جرم الکترون، $k = 5\text{Å}^{-1}$ ، $\sigma = 1\text{Å}$ و $x_\cdot = L/2$ درنظربگیرید. سپس در زمان‌های مختلف (به روش Crank-Nicolson) $\psi^*(x)\psi(x)$ را رسم کنید.

زمان پایانی را نیز $t_{end} = ۱۰۰$ a.u. در نظر بگیرید و حداقل برای ۵ زمان مختلف بین $t = ۰$ و t_{end} ، $\psi^{\star}(x)\psi(x)$ را رسم کنید. در صورت ممکن تغییرات را به صورت انیمیشن نشان دهید.

- نکته۱:باید توجه داشت که فاکتور e^{ikx} به معنی این است که تابع موج دارای یک تکانه غیر صفر است و به سمت راست حرکت خواهد کرد.
- نکته۲:باید به این نکته توجه کرد که تمام معادلات بر حسب اعداد مختلط هستند.
- نکته۳:برای رهایی از خطای گردکردن ناشی از اعداد کوچکی در مسئله، مانند $\hbar$ ، تمام واحدها باید به واحد اتمی تبدیل شوند. همانطور که قبلا هم اشاره شده است در واحدهای اتمی $e = ۱,\ \hbar = ۱,\ m_e = ۱$. بنابراین واحد زمان در این واحد اتمی برابر است با

$$\hbar/E_h = ۱a.u. = ۲/۴۱۸۸۸ \times ۱۰^{-۱۷}s$$

که E_h یک هارتری است (واحد انرژی در واحدهایی اتمی). بنابراین اگر $\Delta t = ۰/۱$ یا $\Delta t = ۰/۰۵$ در واحد اتمی در نظر بگیریم، در واحد واحد SI این مقادیر به صورت زیر خواهند بود:

$$\Delta t = ۰/۱a.u. \rightarrow \Delta t = ۲/۴۱۸۸۸ \times ۱۰^{-۱۸}s$$
$$\Delta t = ۰/۰۵a.u. \rightarrow \Delta t = ۱/۲۰۹۴۴ \times ۱۰^{-۱۸}s$$

در این مسئله $\Delta t = ۰/۰۴a.u.$ در نظر بگیرید که حدود $۱۰^{-۱۸}s$ خواهد بود.

- نکته۴: تعداد تقسیم بندی فضایی را برابر با ۱۰۰۰ در نظر بگیرید، یعنی $N = ۱۰۰۰$ و بنایراین فاصله نقاط مش‌بندی برابر است با $a = L/N$. با توجه به اینکه همه‌ی کمیت‌ها باید به واحد اتمی تبدیل شوند بنابراین

$$L = ۱۰۰\text{Å} = ۱۰۰ \times ۱/۸۸۹۷۲۶۱ \text{ a.}$$
$$a = L/N = ۰/۱۸۸۹۷۲۶۱\text{a.}$$

که $a.$ نماد شعاع بوهر است که واحد طول در واحدهای اتمی است.

- نکته۵: باتوجه به نکته قبلی، پارامترهایی مانند $a_۱$ در واحدهایی اتمی به صورت زیر خواهد شد:

$$a_۱ = ۱ + \Delta t \frac{i\hbar}{۲ma^۲} = ۱ + ۰/۰۴ \times \frac{i}{۲ \times ۱ \times (۰/۱۸۸۹۷۲۶۱)^۲}$$

- نکته۶:برای محاسبه $\mathbf{B}\psi(t)$ نیز چنن عمل می‌کنیم.

$$\mathbf{B}\psi(t) = \mathbf{v}$$

$$v_i = b_۱\psi_i + b_۲(\psi_{i+۱} + \psi_{i-۱})$$

که در پایتون می‌توانیم به صورت زیر بیان شود:

```
for i in range(1:N-1):
    v[i]= b1 * psi[i] + b2*(psi[i+1]+psi[i-1])
# for first and end point (psi is zero at x=0 and x=L)
v[0]  = b1 * psi[0] + b2 * psi[1]
v[N-1]= b1 * psi[N-1] + b2 * psi[N-2]
```

دو جمله آخر در این برنامه باتوجه به این نوشته شده است که ψ در $x = ۰$ و $x = L$ صفر است. در اینجا نیز می‌توان به جای استفاده از حلقه‌ی از تکنیک برش (slicing) استفاده کرد تا سرعت محاسبات افزایش یابد.

```
v[1:N-1]= b1 * psi[1:N-1] + b2*(psi[2:N]+psi[0:N-2])
# for first and end point (psi is zero at x=0 and x=L)
v[0]  = b1 * psi[0] + b2 * psi[1]
v[N-1]= b1 * psi[N-1] + b2 * psi[N-2]
```

- نکته۷:برای رسم نمودارهای $\psi^\star\psi$ باید به این نکته توجه کرد که ضرب یک عدد مختلط در مزدوج آن در پایتون، یک عدد از نوع مختلط خواهد شد با این تفاوت که قسمت موهومی آن صفر است! پس برای رسم $\psi^\star\psi$ باید قسمت حقیقی آن رسم شود.

```
>>> (1+2j)*(1-2j)
(5+0j)
```

- نکته۸: برای مزدوج کردن بردارها در numpy می‌توانید از دستور conj و همچنین برای بدست آوردن قسمت حقیقی یک بردار مختلط می‌توانید از دستور real در numpy استفاده کنید:

```
>>> import numpy as np
>>> vec=np.array([1+1j,2+3j])
>>> vec2=np.conj(vec)*vec
>>> vec2
array([  2.+0.j,  13.+0.j])
>>> np.real(vec2)
array([  2.,  13.])
```

فصل ۴

معادلات لاپلاس و حل عددی آن

۱.۴ حل ساده‌ی معادله لاپلاس

معادله لاپلاس به صورت زیر است:

$$\nabla^۲\phi(x,y,z) = \frac{\partial^۲\phi}{\partial x^۲} + \frac{\partial^۲\phi}{\partial y^۲} + \frac{\partial^۲\phi}{\partial z^۲} = ۰ \qquad (۱.۴)$$

ابتدا فرض کنید این معادله را برای یک فضایی دو بعدی بخواهیم حل کنیم، یعنی:

$$\nabla^۲\phi(x,y) = \frac{\partial^۲\phi}{\partial x^۲} + \frac{\partial^۲\phi}{\partial y^۲} = ۰ \qquad (۲.۴)$$

برای حل عددی این معادله چنین عمل می‌کنیم؛ ابتدا با استفاده از بسط تیلور داریم:

$$\begin{aligned}
\phi(x+\Delta x,y) &= \phi(x,y) + \Delta x\frac{\partial\phi(x,y)}{\partial x} + \frac{۱}{۲}(\Delta x)^۲\frac{\partial^۲\phi(x,y)}{\partial x^۲} \qquad (۳.۴)\\
\phi(x,y+\Delta y) &= \phi(x,y) + \Delta y\frac{\partial\phi(x,y)}{\partial y} + \frac{۱}{۲}(\Delta y)^۲\frac{\partial^۲\phi(x,y)}{\partial y^۲}\\
\phi(x-\Delta x,y) &= \phi(x,y) - \Delta x\frac{\partial\phi(x,y)}{\partial x} + \frac{۱}{۲}(\Delta x)^۲\frac{\partial^۲\phi(x,y)}{\partial x^۲}\\
\phi(x,y-\Delta y) &= \phi(x,y) - \Delta y\frac{\partial\phi(x,y)}{\partial y} + \frac{۱}{۲}(\Delta y)^۲\frac{\partial^۲\phi(x,y)}{\partial y^۲}
\end{aligned}$$

با فرض اینکه $\Delta x = \Delta y$ و همچنین با توجه به اینکه $\frac{\partial^۲\phi}{\partial x^۲} + \frac{\partial^۲\phi}{\partial y^۲} = ۰$ ، با جمع معادلات بالا به معادله زیر می‌رسیم:

$$\phi(x+\Delta x, y) + \phi(x, y+\Delta y) + \phi(x-\Delta x, y) + \phi(x, y-\Delta y) = ۴\phi(x,y) \quad (۴.۴)$$

یا به صورت واضح‌تر:

$$\phi(x,y) = \frac{۱}{۴}[\phi(x+\Delta x, y) + \phi(x-\Delta x, y) + \phi(x, y+\Delta y) + \phi(x, y-\Delta y)] \quad (۵.۴)$$

و برای سه بعد:

$$\begin{aligned} \phi(x,y,z) = \ & \frac{۱}{۶} \quad [\phi(x+\Delta x, y, z) + \phi(x-\Delta x, y, z) \\ & + \quad \phi(x, y+\Delta y, z) + \phi(x, y-\Delta y, z) \\ & + \quad \phi(x, y, z+\Delta z) + \phi(x, y, z-\Delta z)] \end{aligned} \quad (۶.۴)$$

حالا از معادلات بالا می‌توان استفاده کرد و معادلات بالا را حل کرد. شیوه حل ساده به نام روش واهلش [۱] یا روش ژاکوبی معرف است و در واقع یک روش تکرار است. مثلا برای دو بعد ϕ' را از روی ϕ های قبلی بدست می‌آوریم:

$$\phi'(x,y) = \frac{۱}{۴}[\phi(x+\Delta x, y) + \phi(x-\Delta x, y) + \phi(x, y+\Delta y) + \phi(x, y-\Delta y)] \quad (۷.۴)$$

این روند را با جاگزینی ϕ' به جای ϕ مرتب ادامه می‌دهیم:

$$\phi'(x,y) \xrightarrow{\text{به روز رسانی با معادله بالا}} \phi(x,y) \quad (۸.۴)$$

تا اینکه اختلاف ϕ و ϕ' از یک حدی کمتر شود ($|\phi - \phi'| < \epsilon$) . بنابراین الگوریتم به صورت زیر خواهد بود:

۱. فضا را مش‌بندی می‌کنیم، هر نقطه از آن را با دو متغییر x و y نشان می‌دهیم.

[۱] relaxtioan

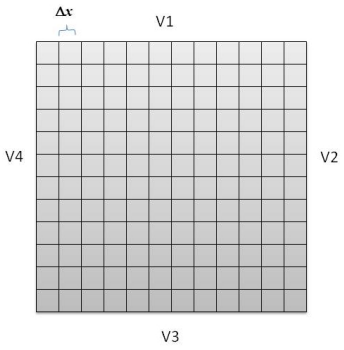
۲. به $\phi(x,y)$ ها در تمام نقاط مقادیر اولیه نسبت می‌دهیم. همچنین طبق شرایط مرزی مسئله $\phi(x,y)$ را در نقاط مرزی مقدار دهی می‌کنیم.

۳. معادله زیر را برای ϕ های جدید (به غیر از روی مرزها) استفاده می‌کنیم:

$$\phi'(x,y) = \frac{۱}{۴}[\phi(x+\Delta x,y) + \phi(x-\Delta x,y) + \phi(x,y+\Delta y) + \phi(x,y-\Delta y)]$$

۴. در صورتی‌که $|\phi - \phi'| < \epsilon$ بود محاسبه به پایان می‌رسد و در غیر اینصورت به مرحله ۳ باز می‌گردیم و محاسبات را تکرار می‌کنیم.

مثال: حل معادله لاپلاس در یک مربع با شرایط مرزی $V_۲ = V_۳ = V_۴ = ۰, V_۱ = ۱Volt$ در ابتدا مربع را مش‌بندی می‌کنیم و $\phi(x,y)$ را مقدار دهی اولیه می‌کنیم و از روش واهلش مقدار ϕ ها را در تمام نقاط پیدا می‌کنیم. برنامه زیر این کار را برای ما انجام می‌دهد. برنامه زیر این کار را انجام می‌دهد:



شکل ۱.۴: یک مربع با شرایط مرزی $V_۱ = ۱volt, V_۲ = V_۳ = V_۴ = ۰$. مربع را به $M \times M$ مش بندی کرده‌ایم

```
from numpy import empty, zeros, max
#from pylab import imshow, gray, show, colorbar
from matplotlib.pyplot import imshow, gray, show, colorbar


# Constants
M = 100         # Grid squares on a side
V = 1.0         # Voltage at top wall
epsilon = 1e-5 # Target accuracy
```

```
# Create arrays to hold potential values
phi = zeros([M+1, M+1], float)
phi[0,:] = V  #V_1
#phi[:,M] = V_2
#phi[M,:] = V_3
#phi[:,0] = V_4
phiprime = empty( [M+1,M+1], float)

# Main loop
n=1
delta = 1.0
while delta > epsilon:
    # Calculate new values of potential
    for i in range(M+1):
        for j in range(M+1):
            if i==0 or i==M or j==0 or j==M:
                phiprime[i,j] = phi[i,j]
            else:
                phiprime[i,j] = ( phi[i+1,j] + phi[i-1,j] \
                                + phi[i,j+1] + phi[i,j-1] ) / 4.0
    # Claculate maximum difference from old values
    delta = max(abs(phi-phiprime))
    # Swap the two arrays around
    phi, phiprime = phiprime, phi
    n = n +1
    print (n, delta)

# Make a plot
imshow(phi)
colorbar()
#gray()
show()
```

: laplace.py

حواس‌مان باید به این باشد که دستور phi = phiprim دستور خطرناکی است چرا که درست است که مقادیر phiprime

در phi ریخته می‌شود ولی از آن به بعد هر تغییری در phiprime در phi نیز اعمال می‌شود. بنابراین بهتر است از دستور تعویض [۲]

[۲] swap

استفاده شود که به صورت زیر است:

$$\text{phi}, \text{phiprime} = \text{phiprime}, \text{phi} \tag{۹.۴}$$

یا به صورت تک تک مقدار دهی شود:

```
for i in range(M+1):
    for j in range(M+1):
        phi[i,j]=phiprime[i,j]
```

: laplace_for.py

۲.۴ حل معادله پواسون

همانطور که می‌دانیم معادله پواسون به صورت زیر است:

$$\nabla^۲\phi = -\frac{\rho}{\epsilon_۰} \tag{۱۰.۴}$$

حل عددی این معادله نیز شبیه حل معادله لاپلاس است. همانند قبل

$$\begin{aligned}
\phi(x+\Delta x, y) &= \phi(x,y) + \Delta x\frac{\partial\phi(x,y)}{\partial x} + \frac{۱}{۲}(\Delta x)^۲\frac{\partial^۲\phi(x,y)}{\partial x^۲} \\
\phi(x, y+\Delta y) &= \phi(x,y) + \Delta y\frac{\partial\phi(x,y)}{\partial y} + \frac{۱}{۲}(\Delta y)^۲\frac{\partial^۲\phi(x,y)}{\partial y^۲} \\
\phi(x-\Delta x, y) &= \phi(x,y) - \Delta x\frac{\partial\phi(x,y)}{\partial x} + \frac{۱}{۲}(\Delta x)^۲\frac{\partial^۲\phi(x,y)}{\partial x^۲} \\
\phi(x, y-\Delta y) &= \phi(x,y) - \Delta y\frac{\partial\phi(x,y)}{\partial y} + \frac{۱}{۲}(\Delta y)^۲\frac{\partial^۲\phi(x,y)}{\partial y^۲}
\end{aligned} \tag{۱۱.۴}$$

با جمع معادلات بالا و با فرض اینکه $\Delta x = \Delta y = a$ و با توجه به معادله پواسون $\frac{\partial^۲\phi(x,y)}{\partial x^۲} + \frac{\partial^۲\phi(x,y)}{\partial y^۲} = -\frac{\rho}{\epsilon_۰}$ داریم:

$$\phi(x,y) = \frac{۱}{۴}[\phi(x+a,y)+\phi(x-a)+\phi(x,y+a)+\phi(x,y-a)] + \frac{۱}{۴}\frac{\rho(x,y)}{\epsilon_۰}a^۲ \tag{۱۲.۴}$$

۳.۴ روش های سریع‌تر در حل معادله لاپلاس

۱.۳.۴ روش فراواهلش

هر چه بتوانیم در روش ژاکوبی یا همان واهلش زوتر به جواب درست در هر نقطه برسیم، در نهایت سریع‌تر به جواب نهایی رسیده‌ایم. فرض کنید که در یک نقطه خاص مقدار پتانسیل نهایی ۰/۵ است و فرض کنید مقدار اولیه که به پتانسیل در این نقطه نسبت داده‌ایم برابر به ۰/۱ است. سپس فرض کنید با استفاده از الگوریتم واهلش مقدار این پتانسیل در این نقطه به خصوص به صورت زیر تغییر می‌کند:

$$\phi(x,y) = ۰/۱ \xrightarrow{\text{الگوریتم واهلش}} \phi(x,y) = ۰/۲ \cdots \xrightarrow{\text{الگوریتم واهلش}} \phi(x,y) = ۰/۵ \qquad (۱۳.۴)$$

حال اگر به جای اینکه منتظر بمانیم تا مقدار پتانسیل به مقدار نهایی خود یعنی ۰/۵ برسد، مثلا آن را به جای اینکه در گام بعدی الگوریتم به ۰/۲ برود آن را به ۰/۴ شوت کنیم! در نتیجه زودتر می‌توانیم به جواب نهایی برسیم. این روش اصطلاحا روش فرا واهلش نامیده می‌شود. برای اینکه این شیوه را اعمال کنیم به صورت زیر عمل می‌کنیم: در الگوریتم واهلش در هر گام در حقیقت به $\phi(x,y)$ مقداری به صورت زیر اعمال می‌شود:

$$\begin{aligned} \phi'(x,y) &= \phi(x,y) + \Delta\phi(x,y) \qquad (۱۴.۴) \\ \Delta\phi(x,y) &= \phi'(x,y) - \phi(x,y) \end{aligned}$$

حال در روش فرا واهلش چنین عمل می‌کنیم:

$$\phi_\omega(x,y) = \phi(x,y) + (۱+\omega)\Delta\phi(x,y) \qquad (۱۵.۴)$$

یعنی به جای اینکه $\Delta\phi(x,y)$ به $\phi(x,y)$ در هر گام اضافه شود، مقدار $(۱+\omega)\phi(x,y)$ اضافه می‌شود. در معادله بالا $\omega > ۰$ است. به عبارتی ما مقدار $\phi(x,y)$ را کمی بیشتر از روش واهلش تغییر می‌دهیم (فرا واهلش انجام می‌دهیم). حال با فرض اینکه

$$\Delta\phi(x,y) = \phi'(x,y) - \phi(x,y)$$

و با توجه به روش واهلش که گفته شد، $\phi_\omega(x,y)$ به صورت زیر خواهد شد:

$$\begin{aligned}\phi_\omega(x,y) &= \phi(x,y)+(1+\omega)\{\phi'(x,y)-\phi(x,y)\} \qquad (16.4)\\ &= (1+\omega)\phi'(x,y)-\omega\phi(x,y)\\ &= \frac{(1+\omega)}{4}\Big\{\phi(x+a,y)+\phi(x-a)+\phi(x,y+a)+\phi(x,y-a)\Big\}\\ &\quad -\omega\phi(x,y)\end{aligned}$$

بهتر است مقدار ω بین ۰ و ۱ انتخاب شود چرا که در غیر اینصورت الگوریتم ناپایدار خواهد شد و نمی‌توان به جواب رسید.

۲.۳.۴ روش گوس-زایدل

در روش گوس-زایدل [۳] به جای اینکه از دو پتانسیل ϕ و ϕ' استفاده کنیم، فقط از یک پتانسیل در جایگزینی‌ها استفاده می‌کنیم. در واقع همان موقع به جای اینکه از ϕّ‌های قدیم استفاده کنیم از ϕ های جدید نیز که بدست آمده است نیز استفاده می‌کنیم به عبارتی

$$\phi(x,y)=\frac{1}{4}[\phi(x+\Delta x,y)+\phi(x-\Delta x,y)+\phi(x,y+\Delta y)+\phi(x,y-\Delta y)] \quad (17.4)$$

این روش نیز مانند روش قبل نسبت به روش واهلش سریع‌تر است.

ترکیب دو روش یعنی فراواهلش و روش گوس-زایدل باعث می‌شود که الگوریتم حل معادله لاپلاس در حدود یک مرتبه سریع‌تر انجام شود. ترکیب دو روش به معادله حل زیر می‌رسد:

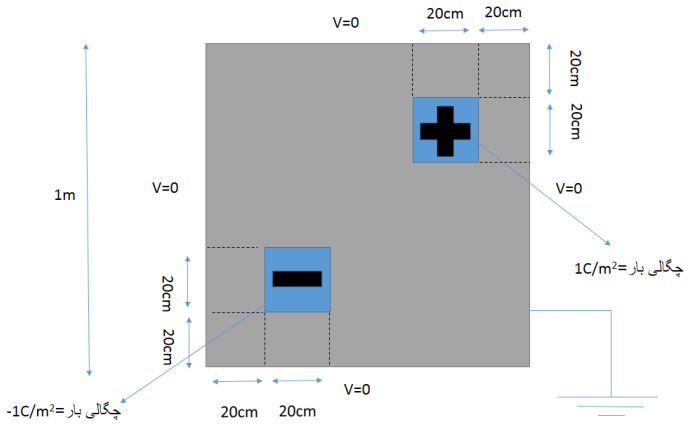
$$\begin{aligned}\phi(x,y) &= \frac{(1+\omega)}{4}\Big\{\phi(x+a,y)+\phi(x-a)+\phi(x,y+a)+\phi(x,y-a)\Big\} \qquad (18.4)\\ &\quad -\omega\phi(x,y)\end{aligned}$$

تمرین ۶: حل معادله پواسون و حل معادله لاپلاس به روش‌های مختلف

- پتانسیل را در شکل زیر به روش واهلش بدست آورید و رسم کنید. برای سادگی مقدار $\epsilon_0 = 1$ قرار دهید. در این مسئله دو مربع کوچک هر کدام دارای بارهای مخالف هستند. باید توجه کرد

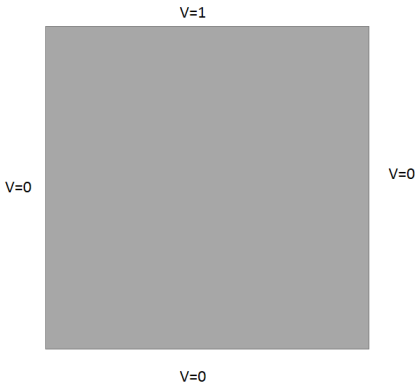
[۳] Gauss-Seidel

برای حل مسئله از معادله پواسون باید استفاده کرد.



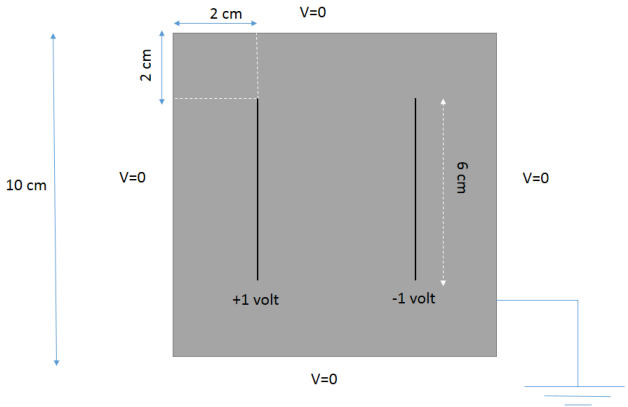
شکل ۲.۴

- پتانسیل را برای شکل مربعی زیر به روش گوس-زایدل+فراواهلش (یعنی روشی ترکیبی گوس-زایدل و فراواهلش)بدست آورید. مسئله را برای ω های مختلف بدست آورید و زمان گیری کنید و مقدار بهینه ω را پیدا کنید. بدین منظور نمودار زمان محاسبات را برحسب ω رسم کنید. ω را از ۰ تا ۱ تغییر دهید. برای زمان گیری از دستور time( ) یا clock( ) در مدول (کتابخانه) time استفاده کنید.



شکل ۳.۴

- با توجه به مقدار بهینه ω پتانسیل را برای خازن زیر بدست آورید و نمودار پتانسیل ϕ را رسم کنید. داخل مربع دو میله با پتانسیل‌های $+۱\text{volt}$ و -۱volt قرار دارد.



شکل ۴.۴

نکته: برای حل تمام قسمت‌های این تمرین از مش ۱۰۰×۱۰۰ استفاده کنید (یعنی $M = 100$ (

فصل ۵

فرایندهای تصادفی

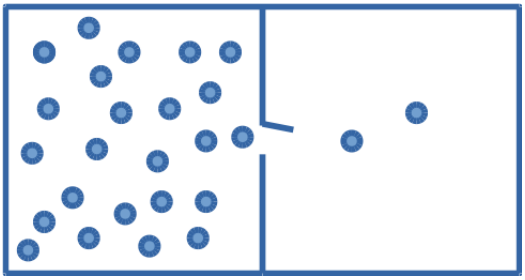
در این بخش قصد داریم نشان دهیم چگونه از تصادف می‌توان آماری قابل پیش‌بینی بدست آورد. ساده‌ترین فرایند تصادفی که در طبیعت قابل مشاهده است فرایند پخش ذرات است. شکل ۱.۵ پخش دانه‌های کلوییدی رنگ را داخل یک محلول نشان می‌دهد که شاید روزانه به گونه‌های مختلفی به آن بر می‌خوریم (مثلا حل دانه‌های شکر داخل یک فنجان چای).

شکل ۱.۵: پدید پخش قطر رنگ در یک محلول (نکته: در این شکل علاوه بر پدیده‌ی پخش، نیروی گرانش نیز نقش بازی می‌کند بنابراین پخش قطر رنگ به اینگونه یک پدیده‌ی کاملا تصادفی به حساب نمی‌آید).

برای اینکه پدیده‌هایی مانند پخش را به شکل ساده‌تری توضیح دهیم می‌توانیم از مثال پخش ذراتی که در ابتدا در یک طرف ظرف هستند شروع کنیم (شکل ۲.۵). این پدیده به انبساط آزاد [۱] معروف

[۱] Free expansion

است. مطابق شکل تعدادی ذره در یک طرف جعبه در نظر بگیرید که با یک تیغه از طرف دیگر جعبه جدا شده‌است. بر روی تیغه یک دریچه وجود دارد که با باز شدن آن ذرات می‌توانند از آن عبور کنند و به طرف دیگر بروند. می‌دانیم که بعد از مدتی ذرات در دو طرف تیغه به طور مساوی تقسیم می‌شوند.



شکل ۲.۵: پخش ذرات

حال برای شبیه سازی چنین فرایندی باید چگونه عمل کنیم؟ یک راه این است که از روش‌های معین مانند دینامیک مولکولی استفاده کنیم. یعنی در هر گام زمانی کوچک مسیر هر ذره را محاسبه کنیم و به این شکل می‌توان حرکت ذرات را در طی زمان مشاهده‌کنیم. یک راه دیگر نیز وجود دارد که می‌توان برای شبیه سازی فرایندهای تصادفی از آن سود برد و آن بهربردن از طبیعت تصادفی این پدیده‌ها است. به عبارتی برای شبیه سازی این پدیده‌ها از کامپیوتر به عنوان ابزاری برای تولید اعداد تصادفی و در پی آن استفاده از این اعداد در الگوریتم‌های مناسب برای شبیه‌سازی این پدیده‌ها است. اساس اینگونه از شبیه‌سازی‌ها که برپایه‌ی مدل‌های احتمالاتی کار می‌کنند این است که فرض می‌شود حرکت ذرات (یا به طور کلی تر حالت‌های ذرات) آنقدر پیچیده هستند که می‌توان رفتاری تصادفی برای آنها در نظر گرفت. مثلا پیش بینی نتیجه پرتاب یک سکه (خط یا شیر) را می‌توان از اینگونه پدیده‌ها به حساب آورد.

حال با این دید می‌توان به سراغ حل مسئله پخش ذرات در جعبه رفت. برای حل این مسئله به سادگی می‌توان فرض کرد که احتمال عبور از دریچه برای یک ذره، در هر گام شبیه‌سازی، چه ذره در سمت چپ باشد و چه در سمت راست باشد مساوی است. البته در اینجا فرض می‌کنیم ذرات باهم برهمکنش ندارند. با این فرض که فرض کاملا صحیح می‌باشد، تابعی مانند LOC برای هر ذره تعریف می‌کنیم. اگر ذره‌ی امi در طرف چپ قرار گرفته باشد $\mathrm{LOC(i)} = -1$ و اگر ذره در طرف راست قرار گرفت باشد $\mathrm{LOC(i)} = +1$. حال می‌توانیم در هر قدم یک عدد صحیح تصادفی بین ۱ تا N تولید کنید (این عدد را با i نشان می‌دهیم)، که این عدد به معنی انتخاب ذره امi است. سپس علامت $\mathrm{LOC(i)}$ را عوض می‌کنیم. در مورد الگوریتم‌های تولید اعداد تصادفی در بخش بعدی صحبت خواهیم کرد. اگر

در این مسئله فقط تعداد ذرات در هر یک از دو قسمت جعبه مهم باشد، می‌توان مسئله را خیلی ساده‌تر بررسی کرد. با فرض اینکه احتمال رفتن یک ذره از طرف راست به چپ و از طرف چپ به راست برابر باشد، می‌توان اینگونه عمل کرد: اگر n تعداد ذرات در طرف چپ باشد، تعداد ذرات در طرف راست برابر با $n' = N - n$ خواهد بود. بنابراین در مسئله فقط n وارد می‌شود. از طرفی احتمال اینکه یک ذره به صورت تصادفی از طرف چپ به راست برود با n/N و برعکس از سمت چپ به راست با n'/N مشخص می‌شود. بنابراین با الگوریتم زیر می‌توان تحول تعداد ذرات را در گام‌های پیاپی حدس زد:

۱. یک عدد تصادفی r بین ۰ تا ۱ تولید می‌کنیم ($۰ \leq r < ۱$)

۲. r با مقدار n/N مقایسه می‌کنیم.

۳. اگر $r \leq n/N$ ، یک ذره را از طرف چپ به راست می‌بریم یعنی:

$$n \longrightarrow n - ۱$$

در غیر این‌صورت یک ذره را از طرف راست به چپ می‌بریم یعنی:

$$n \longrightarrow n + ۱$$

۴. مراحل ۱ تا ۳ را تکرار می‌کنیم.

این الگوریتم ساده در واقع یک الگوریتم مونت کارلو است که در فصل بعدی در مورد آن توضیح خواهیم داد.

۱.۵ تولید اعداد تصادفی در کامپیوتر

برای شبیه سازی فرایندهای تصادفی، ما احتیاج به اعداد تصادفی داریم. ولی مشکل از اینجا شروع می‌شود که در حقیقت با استفاده از کامپیوتر نمی‌توانیم اعداد تصادفی واقعی تولید کنیم. اعدادی که از طریق کامپیوتر به اصطلاح تصادفی تولید می‌شوند در حقیقت شبه تصادفی [۲] هستند. به الگوریتم‌هایی که این اعداد شبه تصادفی را تولید می‌کنند، تولید کننده‌ی اعداد شبه تصادفی [۳] گفته می‌شود. این اعداد به دلیل

[۲] pseudorandom

[۳] pseudorandom number generator

اینکه طبق یک الگوریتم خاص بدست می‌آیند، قابل پیش بینی هستند و بعد از مدتی از دوباره این اعداد تکرار می‌شوند. به همین دلیل به این اعداد، اعداد شبه تصادفی گفته می‌شود.

یک الگوریتم ساده برای تولید اعداد شبه تصادفی استفاده از معادله‌ی زیر است:

$$x' = (ax + c) \mod m \qquad (۱.۵)$$

در اینجا a ، c ، x و m همگی اعداد صحیح هستند. علامت mod عملگر باقیمانده تقسیم [۴] را نشان می‌دهد، که باقیمانده تقسیم $ax + c$ بر m را محاسبه می‌کند (که مقدار آن را با x' نشان داده‌ایم). در پایتون عملگر باقیمانده تقسیم با علامت ٪ نشان داده می‌شود و باقیمانده‌ی تقسیم $ax + c$ بر m در صورتی که $m > (ax + c)$ باشد برابر با $ax + c$ خواهد بود. برای تولید اعداد شبه تصادفی با استفاده از معادله بالا، ابتدا یک مقدار اولیه برای x در نظر می‌گیرم. به این مقدار اولیه، بذر [۵] گفته می‌شود که کاملا به صورت اختیاری انتخاب می‌شود. بذر در تمام الگوریتم‌های تولید اعداد تصادفی وجود دارد و نقش آن تعیین مکان شروع رشته‌ی اعداد تصادفی است. وقتی x اولیه یا همان بذر را داخل معادله قرار می‌دهیم، معادله مقدار x' را به ما می‌دهد. حال اگر به جای x در معادله x' قرار دهیم (یعنی $x = x'$) دوباره یک x' جدید تولید می‌شود. بنابراین به این طریق می‌توانیم یک رشته از x' ها تولید کنیم. می‌توان نشان داد اگر

$$\begin{aligned} a &= ۱۶۶۴۵۲۵ \\ c &= ۱۰۱۳۹۰۴۲۲۳ \\ m &= ۴۲۹۴۹۶۷۲۹۶ \end{aligned}$$

، طولانی‌ترین رشته (بدون تکرار) تولید می‌شود. بنابراین با برنامه زیر

```
import matplotlib.pyplot as plt
N = 1000
#Paramters for psudorandom genrator:
a = 1664525
c = 1013904223
m = 4294967296
x = 1
```

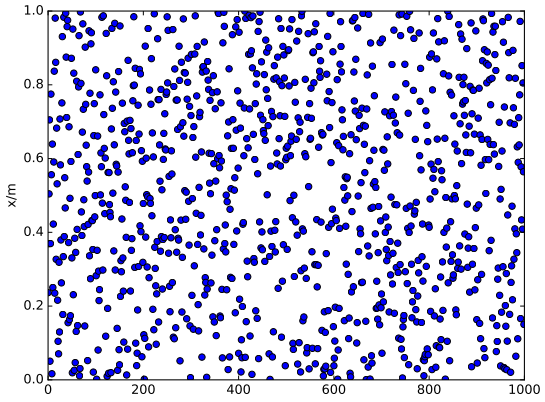
[۴] Modulus operator

[۵] seed

```
results = []

for i in range(N):
    x = (a*x+c)%m
    results.append(x/m)
plt.plot(results,"o")
plt.ylabel("x/m")
plt.savefig("pseudorandom.pdf")
plt.show()
```

: pseudorandom.py

می‌توان اعداد تصادفی بین ۰ و ۱ تولید کنیم (به جای x از x/m استفاده می‌کنیم که در نتیجه اعداد به جای ۰ تا m بین ۰ تا ۱ می‌شوند). نتایج برنامه بالا به صورت شکل ۳.۵ خواهد بود.



شکل ۳.۵: ۱۰۰۰ عدد شبه تصادفی تولید شده با استفاده از معادله ۱.۵ .

در صورتی که مثلا پارامترهای a ، c و m را کوچک بگیریم، به راحتی می‌تواند دید که رشته‌ی اعدادی که تولید می‌شوند به صورت متناوبی تکرار می‌شوند. برای مثال خروجی برنامه زیر:

```
import matplotlib.pyplot as plt
N = 12
#Paramters for psudorandom genrator:
a = 2
c = 3
m = 10
x = 1
```

```
results = []

for i in range(N):
    x = (a*x+c)%m
    results.append(x/m)
print(results)
```

: pseudorandom1.py

اعدادی به صورت زیر خواهند بود:

$$[۰/۵, ۰/۳, ۰/۹, ۰/۱, ۰/۵, ۰/۳, ۰/۹, ۰/۱, ۰/۵, ۰/۳, ۰/۹, ۰/۱]$$

اگر x اولیه را که در اینجا نقش بذر را بازی می‌کند به جای ۱ مثلا عدد ۳ قرار دهیم، فقط مکان رشته تغییر خواهد کرد. برای مثال، نتیجه زیر برای وقتی است که $x = ۳$ قرار دهیم:

$$[۰/۹, ۰/۱, ۰/۵, ۰/۳, ۰/۹, ۰/۱, ۰/۵, ۰/۳, ۰/۹, ۰/۱, ۰/۵, ۰/۳] \qquad (۲.۵)$$

برای تولید اعداد تصادفی بهتر در پایتون می‌توان از الگوریتم شبه اعداد تصادفی بهتری به نام MersenneTwister استفاده کرد. این الگوریتم را می‌توان با فراخوانی کتابخانه random پایتون استفاده کرد [۶] . همچنین زیر کتابخانه‌ی random در کتابخانه‌ی numpy نیز از این الگوریتم استفاده می‌کند [۷] . تابع‌های مهمی که در کتابخانه‌ی random پایتون وجود دارد در جدول ۱.۵ آورده شده است:

[۶] https://docs.python.org/3.5/library/random.html
[۷] https://docs.scipy.org/doc/numpy/reference/routines.random.html

تابع	خروجی تابع
random()	یک عدد حقیقی بین ۰ و ۱ که به صورت تصادفی از یک توزیع یکنواخت اعداد بین ۰ و۱ انتخاب می‌شود
randrange(n)	یک عدد تصادفی صحیح بین ۰ و $n-1$
randrange(m, n)	یک عدد تصادفی صحیح بین m و $n-1$
randrange(m, n, k)	یک عدد تصادفی صحیح بین m و n با گام‌های k
randint(a, b)	معادل تابع randrange(a, b + 1)
choice(seq)	انتخاب تصادفی یک عضو از seq که در واقع seq می‌تواند یک لیست باشد

همان‌طور که گفته شد، با بذر می‌توانیم مکان شروع رشته‌ای که توسط یک تولید کننده‌ی اعداد شبه تصادفی ایجاد می‌شود را مشخص کرد. برای مثال اگر برنامه‌ی زیر را ده‌ها بار تکرار کنیم همیشه یک خروجی کاملا یکسان مشاهده خواهیم کرد:

```
from random import randrange, seed
seed(42)
for i in range(4):
    print(randrange(10))
```

: seeds.py

با تابع seed در کتابخانه‌ی random پایتون مکان شروع رشته‌ی تصادفی را مشخص می‌کنیم. اگر از تابع seed استفاده نکنیم، در هر بار اجرای برنامه بالا رشته‌ی اعداد تصادفی متفاوتی چاپ می‌شود. در واقع در هر بار اجرا پایتون، یک بذر متفاوتی را انتخاب می‌کند. سوالی که در اینجا مطرح می‌شود این است که اصولا چرا در (برخی) شبیه سازی ها از تابعی مانند seed استفاده می‌شود و یا به عبارتی، مفید بودن این تابع در کجا است؟ دلیل اصلی استفاده از تابعی مانند seed تکرار پذیری شبیه‌سازی است. خصوصا زمانی که نیاز به تست برنامه و اشکال یابی وجود داشته باشد.

۱.۱.۵ توزیع یکنواخت و غیریکنواخت

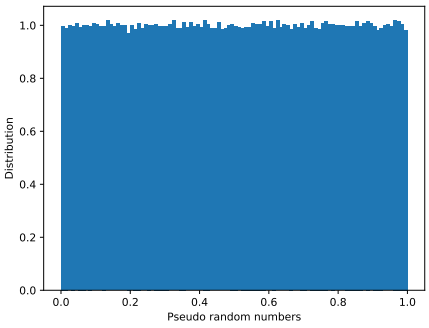
اعداد تصادفی را می‌توان با توزیع خاصی تولید کرد. روش ساده‌ای که در بالا به آن اشاره شد (استفاده از معادله‌ی ۱.۵) یا دستور random از کتابخانه random پایتون، اعداد تصادفی با توزیع یکنواخت تولید می‌کنند. برای اینکه یکنواختی توزیع اعداد تصادفی را در این روش مشاهد کنیم، کافی است از

دستور hist در matplotlib برای رسم نمودار ستونی (هیستوگرام [۸]) توزیع اعداد تصادفی استفاده کنید. با برنامه زیر می‌توان نمودار ستونی توزیع که با استفاده از معادله‌ی ۱.۵ تولید می‌شود را رسم کرد:

```
import matplotlib.pyplot as plt
N = 1000000
#Paramters for psudorandom genrator:
a = 1664525
c = 1013904223
m = 4294967296
x = 1
results = []

for i in range(N):
    x = (a*x+c)%m
    results.append(x/m)
plt.hist(results,100, normed=True)
plt.xlabel("Pseudo random numbers")
plt.ylabel("Distribution")
plt.savefig("pseudorandom_hist.pdf")
plt.show()
```

: pseudorandom_hist.py

شکل ۴.۵ نشان می‌دهد که توزیع اعداد تصادفی که با استفاده از معادله‌ی ۱.۵ تولید شده‌است تقریبا یکنواخت است. اعداد تصادفی را می‌توان طبق توزیع خاصی نیز تولید کرد. برخی از توزیع‌های رایج در



شکل ۴.۵: توزیع ۱۰۰۰۰۰۰ عدد شبه تصادفی تولید شده با استفاده از معادله ۱.۵ .

[۸] Histogram

کتابخانه‌های random در numpy و کتابخانه random پایتون موجود است. مثلا به راحتی می‌توان توزیع اعداد تصادفی را طبق توزیع گوسی [۹] انتخاب کرد. توزیع گوسی یا همان توزیع نرمال [۱۰] با معادله‌ی

$$p(x) = \frac{۱}{\sqrt{۲\pi\sigma^۲}} e^{-\frac{(x-\mu)^۲}{۲\sigma^۲}} \qquad (۳.۵)$$

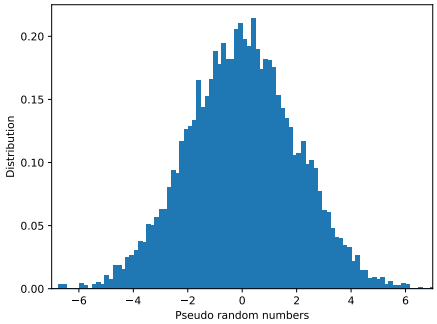
بیان می‌شود. پارامتر μ مقدار متوسط اعداد تصافی در این توزیع را نشان می‌دهد (مکان قله تابع گوسی) و پارامتر دیگر σ انحراف استاندار اعداد تصادفی را تعیین می‌کند. در واقع اگر مقدار واریانس اعداد تصادفی در این توزیع را محاسبه کنیم یعنی $\langle(x - \langle x\rangle)^۲\rangle$ به مقدار $\sigma^۲$ می‌رسیم. در کتابخانه random پایتون تابع gauss(mu،sigma) اعداد تصادفی با توزیع گوسی تولید می‌کند که در آن پارامتر mu همان μ و پارامتر sigma همان σ است. برنامه زیر توزیع گوسی تولید شده با این تابع را برای ۱۰۰۰۰ عدد تصادفی نشان می‌دهد:

```
import matplotlib.pyplot as plt
from random import gauss
N = 10000
results = []
sigma=2.0
mu=0.0
for i in range(N):
    results.append(gauss(mu,sigma))

plt.hist(results,100, normed=True)
plt.xlim(-7,7)
plt.xlabel("Pseudo random numbers")
plt.ylabel("Distribution")
plt.savefig("gauss_hist.pdf")
plt.show()
```

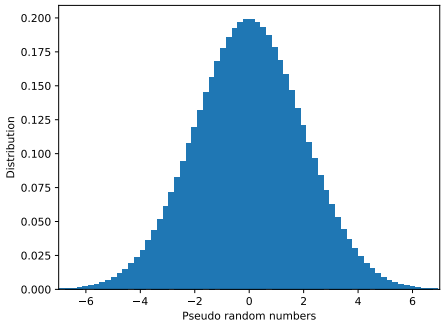
: gauss_hist.py

ظاهرا نمودار ۵.۵ تقریبا گوسی شده است. برای اینکه مطمئن شویم که توزیع کاملا گوسی می‌شود باید تعداد اعداد تصادفی تولید شده را بسیار بالا ببریم. مثلا در نمودار ۶.۵ تعداد اعداد تصادفی که طبق توزیع گوسی تولید شده‌است را به ۱۰۰۰۰۰۰۰ افزایش داده‌ایم. در فصل بعد (مونت کارلو) در مورد روش تولید اعداد تصادفی طبق یک توزیع دلخواه توضیح خواهیم داد.

[۹] Gaussian distribution
[۱۰] Normal distribution



شکل ۵.۵: توزیع ۱۰۰۰۰ عدد شبه تصادفی تولید شده طبق توزیع گوسی با $\sigma = ۲$ و $\mu = ۰$.

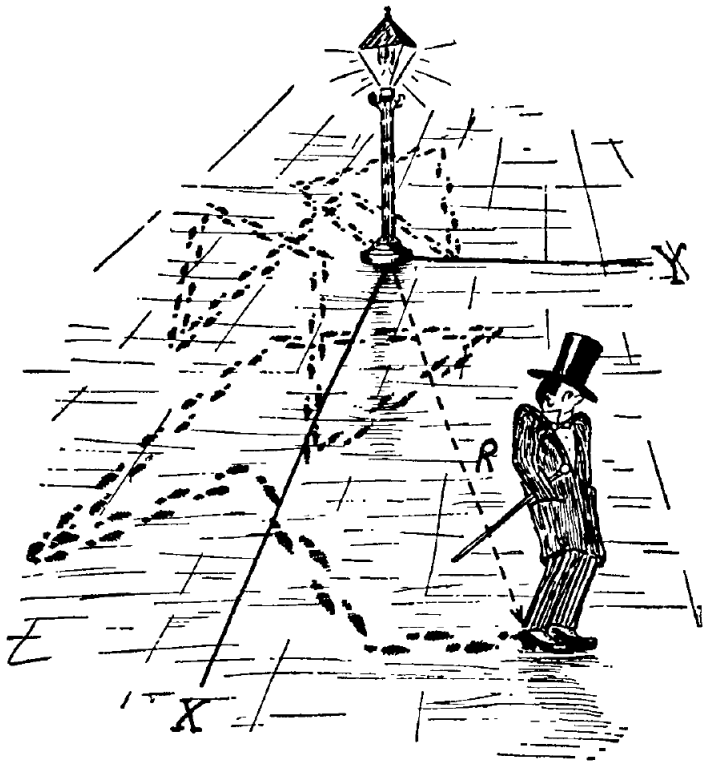


شکل ۶.۵: توزیع ۱۰۰۰۰۰۰۰ عدد شبه تصادفی تولید شده طبق توزیع گوسی با $\sigma = ۲$ و $\mu = ۰$.

۲.۵ ولگشت

بسیاری از پدیده‌های مانند حرکت ذره‌ی غبار در هوا یا یک ذره کلوئیدی در یک مایع را می‌توان یک فرایند تصادفی در نظر گرفت. چنین فرایندهایی تصادفی و بسیار دیگر را می‌توان با استفاده از مفهوم ولگشت یا گام تصادفی و یا قدم‌زدن تصادفی [۱۱] شبیه سازی کرد. یک گام‌زن تصادفی را می‌توان به یک آدم مست سرگردان تشبیه کرد (شکل ۷.۵) که در هر جهت به صورت تصادفی گام می‌زند و گام‌های قبلی آن هیچ ربطی به گام‌های بعدی ندارند. به اصطلاح می‌گوییم که گام‌هابدون تاریخچه هستند و هیچ گونه همبستگی بین گام‌ها وجود ندارد.

[۱۱] Random Walk



شکل ۷.۵: گام زدن یک آدم مست که در واقع یک ولگشت دو بعدی را نشان می‌دهد

۱.۲.۵ ولگشت در یک بعد

برای سادگی از ولگشت در یک بعد شروع می‌کنیم. یک گام‌زن تصادفی را در نظر بگیرید که با احتمال p به سمت جلو و با احتمال q به سمت عقب برمی‌گردد. بنابراین $p + q = ۱$ است. برای سادگی نیز، طول گام‌ها را برابر و به اندازه‌ی a در نظر می‌گیریم. مکان یک گام‌زن بعد از N گام برابر خواهد بود با:

$$x_N = \sum_{i=۱}^{N} s_i \qquad (۴.۵)$$

که در اینجا $s_i = \pm a$. اگر تعدادی گام‌زن داشته باشیم که هر کدام از آن‌ها N گام بردارند، مقدار متوسط جابه‌جایی آنها یعنی $\langle x_N \rangle$ برابر خواهد بود با:

$$\begin{aligned} \langle x_N \rangle &= \langle \sum_{i=1}^{N} s_i \rangle = \sum_{i=1}^{N} \langle s_i \rangle = N \langle s \rangle \\ &= N(pa - qa) = N(p-q)a \end{aligned} \qquad (۵.۵)$$

نکته اصلی که در ساده‌سازی بالا انجام شده است این است که در یک ولگشت گام‌ها از هم مستقل هستند بنابراین متوسط گیری بر روی جمع گام‌ها را می‌توان بر روی تک تک گام‌ها برد یعنی:

$$\langle \sum_{i=1}^{N} s_i \rangle = \sum_{i=1}^{N} \langle s_i \rangle$$

در صورتی‌که احتمال به جلو و عقب رفتن یکسان باشد یعنی $p = q = 1/2$ ، طبق معادله ۵.۵ $\langle x_N \rangle = 0$ است. بنابراین کمیت $\langle x_N \rangle$ توصیفی از گستردگی و پخش شدگی یک گام‌زن به ما نمی‌دهد. پارامتر مفید که به نوعی پخش شدگی گام‌زن یا گستردگی آن را در فضا نشان می‌دهد متوسط مجذور جابه‌جایی خالص است که با Δx^2 نشان می‌دهیم و به صورت زیر تعریف می‌شود (این کمیت به نام واریانس در آماری شناخته می‌شود) :

$$\begin{aligned} \Delta x^2 &\equiv \langle (x - \langle x \rangle)^2 \rangle \\ &= \langle x^2 \rangle - \langle x \rangle^2 \end{aligned} \qquad (۶.۵)$$

برای وقتی $p = q = 1/2$ مقدار Δx^2 با $\langle x^2 \rangle$ برابر خواهد بود. در این صورت کافی است $\langle x^2 \rangle$ را محاسبه کنیم.

$$\begin{aligned} \langle x_N^2 \rangle &= \langle \sum_{i=1}^{N} (\sum_{j=1}^{N} s_i s_j) \rangle \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \langle s_i s_j \rangle \end{aligned} \qquad (۷.۵)$$

جملات $s_i s_j$ وقتی $i \neq j$ است، با احتمال مساوی $+a$ و $-a$ خواهد بود (برای وقتی که $p = q = ۱/۲$) . بنابراین $\langle s_i s_j \rangle$ برای وقتی $i \neq j$ است برابر با صفر است. بنابراین

$$\langle x_N^۲ \rangle = \sum_{i=۱}^{N} \langle s_i^۲ \rangle = \sum_{i=۱}^{N} a^۲ = N\, a^۲ \qquad (۸.۵)$$

با برنامه‌ی زیر به راحتی می‌توانیم یک گام‌زن تصادفی را شبیه‌سازی کنیم:

```
from random import random

x=0

for i in range(N):
    p = random()
    if p < 0.5:
        x=x+1
    else:
        x=x-1
```

: rw0.py

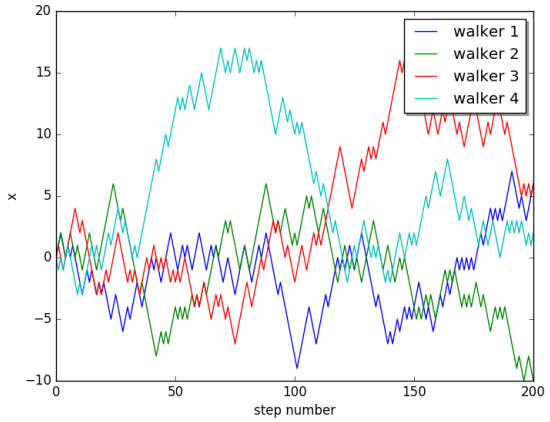
در این برنامه فرض کرده‌ایم $a = ۱$ است. باید توجه داشت که x_N و $\langle x_N \rangle$ خیلی متفاوت هستند. در واقع مقدار x_N می‌تواند از $-Na$ تا $+Na$ متغییر باشد. ولی متوسط آن یعنی $\langle x_N \rangle$ در صورتی‌که متوسط گیری بر روی بی‌نهایت گام‌زن انجام پذیرد، برابر با صفر خواهد بود. با برنامه‌ی زیر می‌توان چهار گام‌زن را شبیه‌سازی کرد و به راحتی دید که x_N صفر نمی‌شود.

```
from random import random
import matplotlib.pyplot as plt
N=200; M=4; x1=0
x=[x1]
for j in range(M):
    x1=0
    x=[x1]
    for i in range(N):
        p= random()
        if p < 0.5:
            x1=x1+1
        else:
            x1=x1-1
        x.append(x1)
    plt.plot(x)
```

```
16
plt.show()
```

: rw1_show.py

خروجی این برنامه در شکل ۸.۵ نشان داده شده است. شکل ۱۰.۵ مقدار $\langle x_N \rangle$ که برروی تعداد



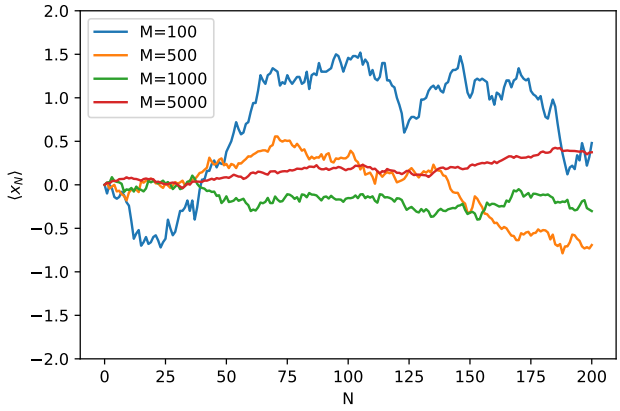
شکل ۸.۵: مکان ۴ گام‌زن تصادفی با ۲۰۰ گام

گام زن‌های مختلف متوسط گیری شده است نشان می‌دهد. همانطور که می‌توان دید با افزایش تعداد گام‌زن‌ها مقدار $\langle x_N \rangle$ به صفر نزدیک‌تر می‌شود. شکل ؟؟ نیز مقدار $\langle x_N^2 \rangle$ را برای تعداد مختلف گام‌زن‌ها نشان می‌دهد. باز می‌توان مشاهده کرد که افزایش تعداد گام‌زن‌ها باعث شده نتیجه به یک خط نزدیک‌تر شود، به عبارتی $\langle x_N^2 \rangle \propto N$.

تمرین ۷ (قسمت اول): ولگشت در یک بعد و دو بعد

۱. برای یک ولگشت در یک بعد (با فرض $p = q = 1/2$ و $a = 1$):

- $\langle x_N \rangle$ را بر حسب N محاسبه و رسم کنید.
- $\langle x_N^2 \rangle$ را بر حسب N محاسبه و رسم کنید.



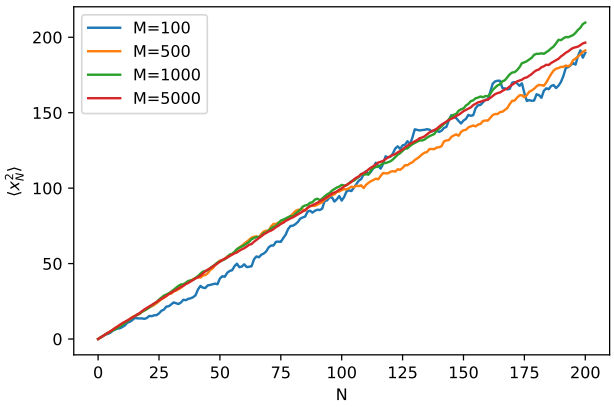
شکل ۹.۵: مقدار $\langle x_N \rangle$ برای متوسط گیری بر روی تعداد مختلف گام‌زن‌ها

(راهنمایی: برای انجام این تمرین ۵۰۰۰ گام‌زن در نظر بگیرید و متوسط گیری را بر روی این تعداد گام‌زن انجام دهید و مقدار بیشنه‌ای که هر گام‌زن قدم بر می‌دارد را ۱۰۰ در نظر بگیرد)

۲. نمودار توزیع (فراوانی) x_N را در گام N=۱۰۰ رسم کنید. در این قسمت تعداد گام‌زن‌ها را ۱۰۰۰۰ در نظر بگیرید. برای رسم این توزیع از دستور hist در matplotlib کمک بگیرید. ،

برای شبیه‌سازی ولگشت در دو بعد به شرط اینکه حرکت در چهار جهت $+x$ ، $-x$ ، $+y$ و $-y$ هم احتمال و اندازه‌ی گام‌ها یکسان باشد می‌توان از برنامه زیر استفاده کرد:

```
for step in range(N):

    p = random.randrange(4)
    if   p == 0:
            x += 1
    elif p == 1:
            y += 1
    elif p == 2:
            x -= 1
```



شکل ۱۰.۵: مقدار $\langle x_N^2 \rangle$ برای متوسط گیری بر روی تعداد مختلف گام‌زن‌ها

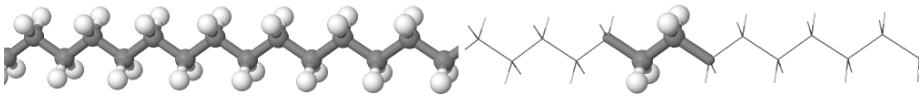
```
    elif p == 3:
11          y -= 1
```

: rw2_show.py

۳.۵ پلیمرها و ولگشت خود اجتناب

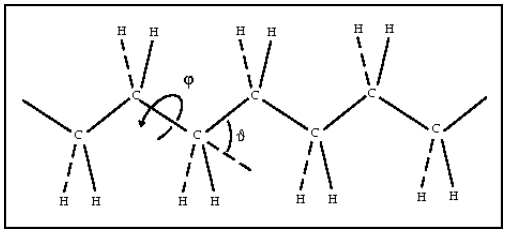
همان‌طور که می‌دانید یک پلیمر [۱۲] ، یک مولکول بسیار بزرگ است، که از تکرار زیر واحدهایی که به هم متصل شده‌اند و به آن‌ها مونومر [۱۳] گفته می‌شود تشکیل شده است. یکی از ساده‌ترین پلیمرها، پلی‌اتیلن [۱۴] است که از به‌هم متصل شدن مونومر C_2H_4 درست می‌شود. شکل ۱۱.۵ این پلیمر را به همراه مونومر آن نشان می‌دهد. مسلما پیکربندی یک پلیمر به صورت خطی، که در شکل ۱۱.۵ نشان داده شده است، نیست و معمولا پلیمرها پیکربندی‌های آمورف را به خود می‌گیرند. یعنی مونومرهای نسبت به هم یک زاویه‌های دلخواه دارند و این باعث می‌شود که یک پلیمر به جای اینکه یک پیکربندی خطی داشته باشد، دارای یک پیکربندی کج و موجی شود. این درجه آزادی در زاویه‌های در شکل ۱۲.۵ نشان داده شده است. بنابراین یک پلیمر معمولا یک پیکربندی آمورف شبیه چیزی که در شکل ۱۳.۵ به نمایش دار

[۱۲] Polymer
[۱۳] Monomer
[۱۴] Polyethylene



(آ) واحد تکرار شونده یا همان مونومر (C_2H_4) پلیمر پلی‌اتیلن (ب) پولی‌اتیلن، یک پلیمر که واحدهای سازنده‌ی آن یعنی مونومرهای آن مولکول‌های C_2H_4 است

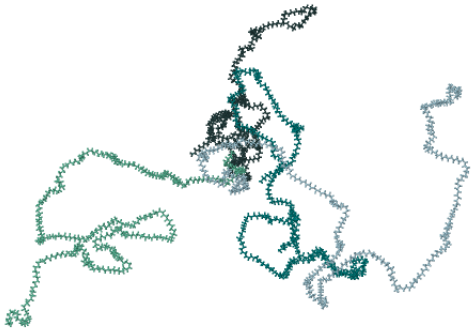
شکل ۱۱.۵: پلیمر پلی‌اتیلن با مونومر آن



شکل ۱۲.۵: درجه آزادی زاویه‌ای در پولی‌اتیلن

آمده است را به خود می‌گیرد [۱۵] . همان‌طور که مشاهده می‌کنید، این زنجیره‌ی پلیمر خطی نیست و بیشتر شبیه مسیر یک گام‌زن تصادفی است. در واقع ما می‌توانیم پلیمرهای انعطاف‌پذیر [۱۶] را با گام‌زن‌های تصادفی شبیه سازی کنیم. منتها نکته‌ی اصلی برای شبیه سازی پیکربندی یک پلیمر این است که پلیمرها معمولا خود را قطع نمی‌کنند بنابراین برای شبیه‌سازی پلیمرها باید از مفهوم ولگشت خود اجتناب [۱۷] استفاده کرد که در قسمت بعدی به آن می‌پردازیم.

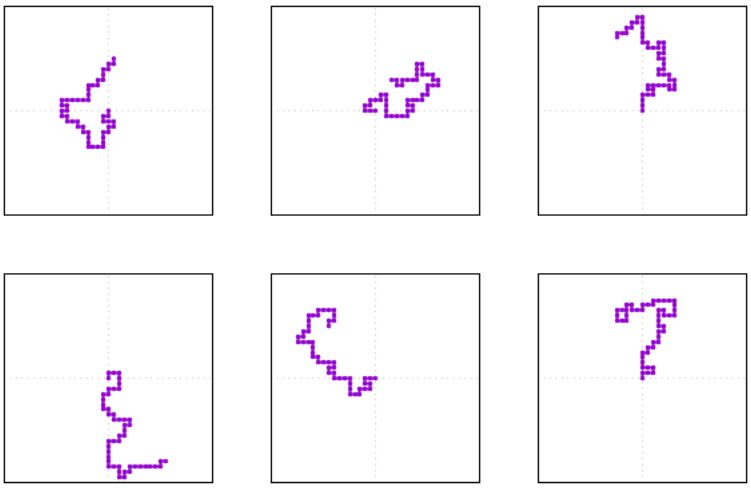
[۱۵] http://www.chemtube3d.com/polymer/_PolyethyleneF.html
[۱۶] Flexible polymers
[۱۷] Self-avoiding walk



شکل ۱۳.۵: یک زنجیره‌ی پلی‌اتیلن که از ۸۰۱ مولکول C_2H_4 تشکیل شده است.

۱.۳.۵ ولگشت خود اجتناب

در ولگشت خود اجتناب، گام‌زن‌های تصادفی نباید به مکان‌هایی بروند که قبلا در آن مکان‌ها در گام‌های قبلی بوده‌اند. نمونه‌ای از مسیر این نوع ولگشت در شکل ۱۷.۵ آورده شده است. برای اینکه آمار صحیح از این نوع ولگشت داشته باشیم، یک راه، شمارش دقیق تمام مسیرها برای N گام است. روش‌هایی برای شمارش دقیق تمام حالت‌های ممکن وجود دارد ولی این روش‌ها بسیار پر هزینه هستند مثلا برای ولگشت بر روی یک شبکه‌ی دو بعدی مربعی تا گام $N = ۷۱$ توانسته‌اند شمارش دقیق انجام‌دهند و یا بر روی شبکه‌ی مکعبی تا $N = ۳۶$ گام شمارش دقیق انجام شده است. در جدول ۱.۵ نتایج حاصل از شمارش دقیق برای ولگشت خود اجتناب در شبکه مربعی تا $N = ۲۰$ آورده شده است. در این جدول تعداد کل حالت‌های ممکن برای یک گام‌زن تصادفی خوداجتناب بر حسب N و همچنین متوسط مجذور فاصله یعنی $\langle \mathbf{r}_N^۲ \rangle$ آورده شده است. برای شبیه‌سازی ولگشت خود اجتناب چندین روش وجود دارد که در ادامه به دو تای آن‌ها اشاره می‌کنیم.



شکل ۱۴.۵: چند گام‌زن تصادفی خوداجتناب با ۵۰ گام

ساخت گام‌زن‌ها تصادفی خود اجتناب به صورت تصادفی

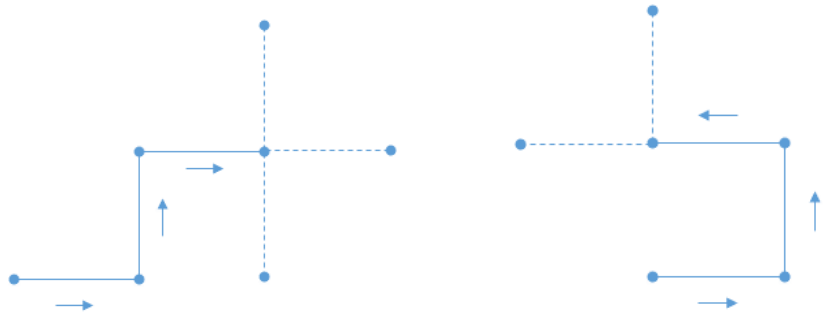
شاید در ابتدا، شبیه سازی ولگشت خود اجتناب ساده به نظر آید. در واقع آنچه که برای شبیه‌سازی ولگشت خود اجتناب در ابتدا به ذهن می‌آید این است که هر گام‌زن به حرکت تصادفی خود ادامه می‌دهد و به محض اینکه به موقعیت که قبلا در آن گام نهاده است می‌رسد، از دوباره به مکانی که در گام قبل بوده است برمی‌گردد و جهت تصادفی دیگری را برای ادامه حرکت انتخاب می‌کند. ولی این الگوریتم ساده یک مشکل اساسی دارد و آن مشکل این است که در این صورت یک گام‌زن تصادفی خود اجتناب دیگر گام‌های کاملا تصادفی برنمی‌دارد و گام‌های بعدی آن دارای شرط می‌شوند که این خلاف فرض تصادفی بودن گام‌ها با احتمال‌های مشخصی است که از ابتدای حرکت تعیین شده‌اند. مشکل شبیه‌سازی ولگشت خود اجتناب تقابل تصادفی بودن و از طرفی خود اجتناب بودن است. در واقع تمام حالت‌های ممکن (یعنی پیکربندی‌های ممکن یا مسیرهای ممکن) برای یک گام‌زن تصادفی خود اجتناب باید هم احتمال باشند (البته در صورتی‌که تمام جهت‌ها هم‌احتمال باشند). برای اینکه این موضوع را روشن‌تر کنیم، به دو مسیر آ و ب که در شکل ۱۵.۵ آورده‌شده است دقت کنید. در هر دو مسیر، گام‌زن ۳ گام برداشته است. منتها حالت‌هایی که برای گام چهار یک گام‌زن خود اجتناب می‌توان داشت با خط چین نشان داده شده‌است. در مسیر آ دو حالت برای گام چهار وجود دارد در حالی که در مسیر ب برای گام چهارم

جدول ۱.۵

N	$\langle \mathbf{r}_N^2 \rangle$	تعداد کل گام‌زن‌ها
۱	۱٫۰	۴
۲	۲٫۶۷	۱۲
۳	۴٫۵۶	۳۶
۴	۷٫۰۴	۱۰۰
۵	۹٫۵۶	۲۸۴
۶	۱۲٫۵۷	۷۸۰
۷	۱۵٫۵۶	۲۱۷۲
۸	۱۹٫۰۱	۵۹۱۶
۹	۲۲٫۴۱	۱۶۲۶۸
۱۰	۲۶٫۲۴	۴۴۱۰۰
۱۱	۳۰٫۰۲	۱۲۰۲۹۲
۱۲	۳۴٫۱۹	۳۲۴۹۳۲
۱۳	۳۸٫۳۰	۸۸۱۵۰۰
۱۴	۴۲٫۷۹	۲۳۷۴۴۴۴
۱۵	۴۷٫۲۲	$۶٫۴۱۷ \times ۱۰^۶$
۱۶	۵۱٫۹۹	$۱٫۷۲۶ \times ۱۰^۷$
۱۷	۵۶٫۷۲	$۴٫۶۴۷ \times ۱۰^۷$
۱۸	۶۱٫۷۷	$۱٫۲۴۷ \times ۱۰^۸$
۱۹	۶۶٫۷۷	$۳٫۳۵۱ \times ۱۰^۸$
۲۰	۷۲٫۰۸	$۸٫۹۷۷ \times ۱۰^۸$

۳ حالت می‌توان متصور شد. در نتیجه تعداد حالت‌های ممکن برای چهارگام برای مسیر ب $\frac{۳}{۴}$ بیشتر از مسیر آ است. حال اگر گام‌زن با شرط خود اجتناب گام بردارد، این به این معنی خواهد بود که به صورت مصنوعی احتمال مسیر ب برای چهار گام بیشتر از حالت آ در نظر گرفته‌شده است که این موضوع خلاف فرضی است که انجام داده‌ایم، که به ما می‌گوید تمام مسیرها یا پیکربندی‌ها باید با احتمال مساوی وجود داشته باشند. پس باید به دنبال الگوریتم دیگری باشیم.

بنابراین ساده‌ترین الگوریتم که برای شبیه‌سازی ولگشت خود اجتناب به ذهن می‌آید تولید گام‌زن‌های است که به طور تصادفی مسیر خود را قطع نکرده باشند. برای تولید گام‌زن‌های خود اجتناب با N گام چنین عمل می‌کنیم: شروع به تولید گام‌زن‌هایی با N گام می‌کنیم و در هر لحظه‌ای که گام‌زن خود را قطع کرد این گام‌زن را ازبین می‌بریم و از ابتدا یک گام زن جدید ایجاد می‌کنیم. بدین ترتیب با تکرار این فرایند می‌توانید گام‌زن‌های خود اجتنابی با N گام داشته باشیم. برنامه زیر $\mathrm{n_{walks}}$ گام‌زن با $\mathrm{n_{steps}}$ گام تولید می‌کند که از این تعداد فقط تعداد کمی می‌توانند گام‌های خود را تکمیل کنند و موفق شوند. با گسترش برنامه زیر می‌توان مقدار $\langle r_N^2 \rangle$ برای هر N حساب کرد.



(ب) (آ)

شکل ۱۵.۵: دو گام‌زن تصادفی با ۳ گام برداشته و حالت‌های ممکن برای گام چهارم آنها

```
while walks < n_walks:

    sites = []              # list of lattice sites
    x = 0; y = 0            # start at origin
    sites.append([x, y])    # append to visited sites list
    walk_succeeded = True   # ok so far!

    # loop over desired number of steps
    for step in range(n_steps):

        # take a random step ENWS
        direction = random.randrange(4)
        if   direction == 0:
            x += 1      # step East
        elif direction == 1:
            y += 1      # step North
        elif direction == 2:
            x -= 1      # step West
        else:
            y -= 1      # step South

        # check whether the site has been visited
        if sites.count([x,y]) == 1:
              visited = True
        else:
              visited = False
        if visited:
```

```
            walk_succeeded = False
            break       # kill the walker
        else:
            sites.append([x, y])

    if walk_succeeded:
        walks += 1
```

: swalk.py

مشکل اصلی این الگوریتم ساده زمان بر بودن این الگوریتم است در واقع هر چه تعداد گام‌ها بیشتر می‌شود احتمال این که گام زن خود را قطع کند بیشتر و بیشتر خواهد شد. در واقع می‌توان نشان داد که نسبت گام‌زن های موفق به کل گام زن های تولید شده به صورت نمایی با تعداد گام کاهش پیدا می‌کند:

$$\frac{\text{تعداد گام‌زن‌های موفق}}{\text{تعداد کل گام‌زن‌های تولید شده}} \sim e^{-\lambda N} \qquad (۹.۵)$$

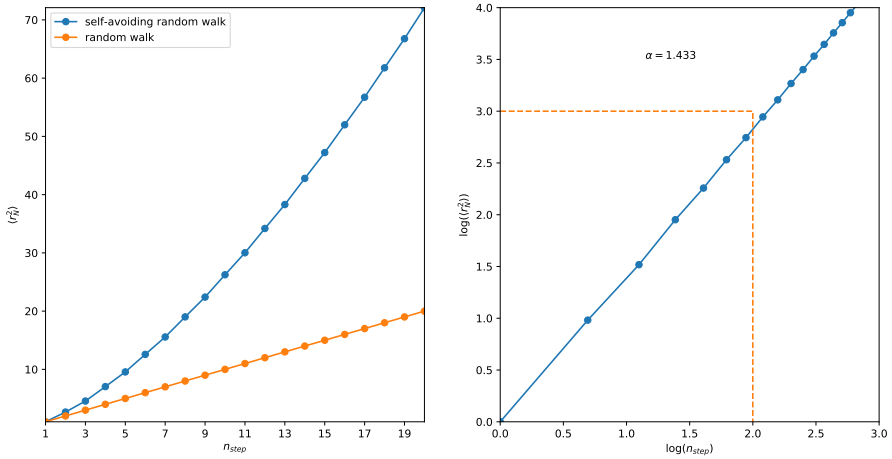
در معادله‌ی بالا λ ضریب فرسایش نامیده می‌شود. بنابراین همانطور که از معادله مشخص است برای تولید گام‌زن‌های با تعداد گام‌های زیاد باید گام‌زن‌های زیادی تولید شود تا از بین آنها بتوان گام‌زن های موفق را جدا کرد. تفاوت اصلی ولگشت تصادفی و ولگشت خود اجتناب را می‌توان در رفتار $\langle r_N^2 \rangle$ بر حسب N مشاهد کرد. همانطور که گفت شد در ولگشت تصادفی

$$\langle r_N^2 \rangle \sim N \qquad (۱۰.۵)$$

در حالی که برای ولگشت خود اجتناب دیگر رابطی خطی بین $\langle r_N^2 \rangle$ و N وجود ندارد و در عوض یک رابطه توانی بین این دو کمیمت وجود دارد:

$$\langle r_N^2 \rangle \sim N^\alpha \qquad (۱۱.۵)$$

که $1 < \alpha < 2$ است. این عدد برای شبکه مربعی (دو بعد) برابر با ۳/۲ و برای شبکه مکعبی (سه بعد) ۶/۵ است. می‌توان داده جدول ۱.۵ را، که داده های دقیق هستند رسم کرد و مقدار α را از آن محاسبه کرد.



شکل ۱۶.۵ : شکل سمت چپ، مقایسه رفتار ولگشت با ولگشت خود اجتناب در دو بعد را نشان می‌دهد. شکل سمت راست نمودار $\log(r_N^2)$ نسبت $\log(N)$ برای مقادیر دقیقی است که در جدول ۱.۵ آورده شده است. خط‌های خط چین نسبت ۳/۲ را نشان می‌دهد که قاعدتا نمودار باید از تقاطع این خط‌های خط چین بگذرد. در واقع شیب این نمودار که مقدار α را نشان می‌دهد دقیقا مقدار ۳/۲ را نشان نمی‌دهد. در حقیقت این مقدار برای وقتی بدست می‌آید که تعداد گام ها را زیاد کنیم.

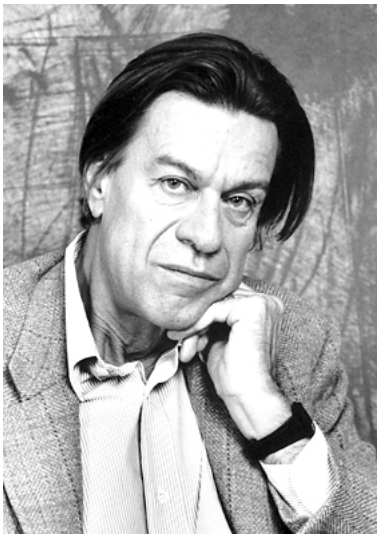
تمرین ۷ (قسمت دوم):محاسبه $\langle r_N^2 \rangle$ بر حسب N و بدست آوردن α

- مقدار $\langle r_N^2 \rangle$ را با استفاده از الگوریتمی که در بالا توضیح داده شد بدست آورید یعنی تولید ولگشت ها و کنار گذاشتن ولگشت‌های که خود را قطع کرده اند و آن را بر حسب N رسم کنید.
- برای بدست آوردن α باید نمودار $\log(\langle r_N^2 \rangle)$ را بر حسب $\log(N)$ رسم کنید و سپس شیب این خط را بدست آورید.
- برای بدست آوردن شیب از تابع numpy.polyfit یا scipy.optimize.curve_fit استفاده کنید.
- به دلیل اینکه استفاده از این الگوریتم بسیار زمان بر است بنابراین مقدار بیشینه ای که برای تعداد گام انتخاب می‌کنید به ۲۵ محدود کنید و تعداد متوسط گیری را (ولگشت‌های موفق)

را به ۲۰۰ محدود کنید. مسلما برای بدست آوردن جواب های بهتر باید تعداد ولگشت‌های موفق بیشتری تولید شود

روش خزیدن برای مسئله ولگشت خود اجتناب

روش‌های زیاد برای تولید گام‌زن‌ها با خصوصیت خود اجتنابی وجود دارد. شاید یکی از کاراترین این روش‌های روش خزیدن (Reptation Method) است. این روش توسط پیر–ژیل دوژن [۱۸] مطرح شد. پیر دوژن کلمه‌ی Reptation را از کلمه Reptile که به معنی خزنده است، ساخته است. ایده‌ی خزیدن

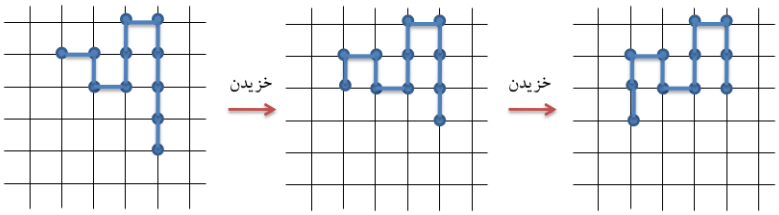


شکل ۱۷.۵: پیر–ژیل دوژن فیزیکدان فرانسوی (۲۰۰۷-۱۹۳۲). او به خاطر تحقیقاتش بر روی کریستال‌های مایع و پلیمرها مفتخر به دریافت جایزه نوبل در سال ۱۹۹۱ شد.

را می‌توان برای تولید حالت‌های مختلف ولگشت خود اجتناب استفاده کرد. شکل ۱۸.۵ یک گام‌زن را نشان می‌دهد که در یک شبکه مربعی اصطلاحا می‌خزد. الگوریتم خزیدن برای تولید پیکربندی‌های مختلف یک گام‌زن خود اجتناب می‌توان به صورت زیر باشد:

۱. یک گام‌زن خود اجتناب با N گام تولید می‌کنیم (از الگوریتم که در بخش قبلی گفته شده استفاده می‌کنیم). اطاعات این پیکربندی را برای متوسطگیری ذخیره می‌کنیم.

[۱۸] Pierre-Gilles de Gennes



شکل ۱۸.۵

۲. به صورت تصادفی گام ابتدایی یا انتهای گام‌زن را انتخاب و آن را حذف می‌کنیم.

۳. به صورت تصادفی ابتدا یا انتهای گام‌زن جدید را انتخاب‌می‌کنیم و یک گام در جهت تصادفی به آن اضافه می‌کنیم.

۴. در صورتی که مرحله‌ی قبلی موفقیت آمیز بود (یعنی گام‌زن مسیر خود را قطع نکرده است)، این پیکربندی جدید را به عنوان گام‌زن موفق به پیکربندی‌های موفق در گام N ام اضافه می‌کنیم تا در متوسط گیری‌ها از آن استفاده کنیم، سپس به مرحله ۲ می‌رویم. در صورت عدم موفقیت گام‌زن به مرحله‌ی اول می‌رویم.

تمرین ۶ (قسمت سوم):محاسبه $\langle r_N^2 \rangle$ بر حسب N و بدست آوردن α به روش خزیدن

- مقدار $\langle r_N^2 \rangle$ را با استفاده از الگوریتمی خزیدن محاسبه و آن را بر حسب N رسم کنید.
- برای بدست آوردن α باید نمودار $\log(\langle r_N^2 \rangle)$ را بر حسب $\log(N)$ رسم کنید و سپس شیب این خط را بدست آورید.
- برای بدست آوردن شیب از تابع numpy.polyfit یا scipy.optimize.curve_fit استفاده کنید.
- در این روش تعداد کل گام‌زن‌های تولید شده برای هر N را مجموعا به ۱۰۰۰ عدد برسانید. بیشینه N را نیز ۴۰ درنظر بگیرید.

۴.۵ پخش، آنتروپی و پیکان زمان

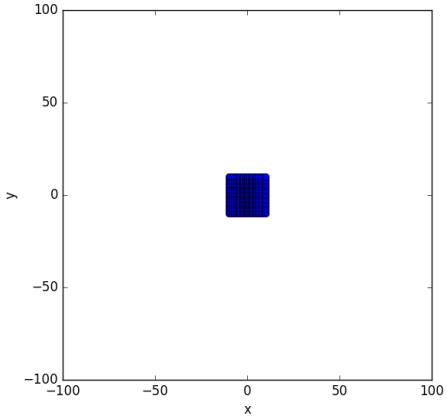
پدیده‌ی پخش دقیقا از مفهوم ولگشت تصادفی تبعید می‌کند. در حقیقت ذرات در محلول (مانند ذرات خامه در قهوه)، به دلیل حرکت‌های گرمایی مولکول‌های محلول، به صورت تصادفی حرکت می‌کنند که در حقیقت ما آن را به عنوان حرکت براونی [۱۹] می‌شناسیم. ما می‌توانیم این پدیده را با استفاده از ولگشت تصادفی شبیه‌سازی کنیم و یکی از پدیده‌های جالب یعنی رسیدن به تعادل و همچنین قانون دوم ترمودینامیک را با آن نشان دهیم. بدین منظور پدیده‌ی پخش در دو بعد را شبیه‌سازی می‌کنیم. در ابتدا حدود ۴۰۰ ذره را در وسط یک مربع قرار می‌دهید. در برنامه‌ی زیر این کار به راحتی انجام شده‌است. همچنین در این برنامه، ذرات نیز نمایش داده شده‌اند.

```
import matplotlib.pyplot as plt
import numpy as np
r=[]
m=10  # inital mesh size for particles: (2m+1)*(2m+1)
for i in range(-m,m+1):
    for j in range(-m,m+1):
        r.append([i,j])     # initial postions
ra=np.array(r)
plt.xlim(-100,100)
plt.ylim(-100,100)
plt.xlabel("x")
plt.ylabel("y")
plt.gca().set_aspect('equal', adjustable='box') #to equalize the scales
    of x-axis and y-axis
plt.plot(ra[:,0],ra[:,1], 'o')
plt.savefig("initial_pos.png")
plt.show()
```

: diff_2d_initial.py

در این برنامه r مکان اولیه ذرات است. ما این ذرات را در یک مش‌بندی $(۲m+۱)\times(۲m+۱)$ چیده‌ایم. که در اینجا $m=۱۰$ قرار داده‌ایم. برای نتایج بهتر می‌توانیم m را بزرگ‌تر کنیم. سپس این ذرات را در مربعی به ابعاد ۲۰۰×۲۰۰ قرار می‌دهیم. بنابراین مربع ما در جهت x از $x=-۱۰۰$ تا $x=۱۰۰$ و در جهت y از $y=-۱۰۰$ تا $y=۱۰۰$ گسترده شده‌است. با اجرای برنامه‌ی بالا موقعیت اولیه ذرات که در وسط جعبه (مربع) قرار گرفته‌اند، به نمایش در می‌آید(شکل ۱۹.۵). حال

[۱۹] Brownian motion



شکل ۱۹.۵: مکان اولیه گام‌زن‌ها در شبکه مربعی

فرض می‌کنیم هر یک از این نقاط یک گام‌زن تصادفی است. و این گام‌زن‌های تصادفی بر روی شبکه مربعی که تعریف کرده‌ایم می‌تواند قدم‌بردارند. یعنی در هر گام مانند گام‌زن تصادفی در دو بعد، چهار گزینه برای حرکت بر روی شبکه در گام بعدی (با احتمال مساوی) وجود دارد:

$$x = x + ۱$$

$$x = x - ۱$$

$$y = y + ۱$$

$$y = y - ۱$$

در صورتی‌که گام‌زن به مرز رسید یعنی موقعیت آن، (x, y) ، در حالت‌های زیر قرار گرفت:

$$
\begin{aligned}
x &= ۱۰۰ \\
&\text{یا} \\
x &= -۱۰۰ \\
&\text{یا} \\
y &= ۱۰۰ \\
&\text{یا} \\
y &= -۱۰۰
\end{aligned}
$$

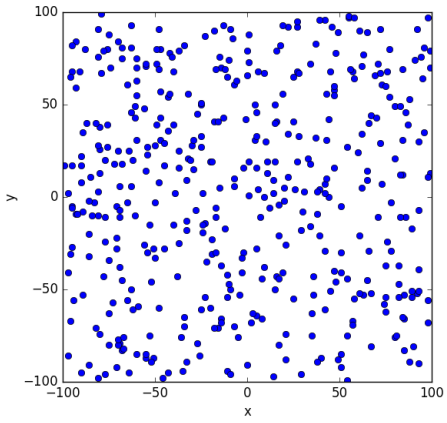
نباید ذرات از مرز مربع عبور کنند و خارج شوند. در واقعیت نیز ذرات کلوییدی وقتی به دیوار ظرف برخورد می‌کنند، برمی‌گردند. بنابراین یک شرط مرزی ساده بر روی حرکت ذرات اعمال می‌کنم. این شرط مرزی این است که در صورتی‌که ذره‌ای به مرز رسید، آن ذره را به موقعیت (نقطه) قبلی‌ش بر می‌گردانیم. حال برای اینکه پدیده‌ی پخش را شبیه سازی کنیم، به صورت تصادفی یک گام‌زن در مربع را انتخاب می‌کنیم و اجازه می‌دهیم که گام‌زن یک گام تصادفی بردارد. این کار را مرتب تکرار می‌کنید. برنامه‌ی زیر این کار را برای ما انجام می‌دهد:

```
m=10
N=(2*m+1)*(2*m+1) # number of molecules or walkers
L=100 # a number for confining walker inside –LXL mesh
Nstep=5000000
for k in range(Nstep):
  # choosing a walker
  n=randrange(N)
  x0=r[n][0]
  y0=r[n][1]
 #moving walker
  p=randrange(4)
  if p== 0:
     r[n][0]=x0+1
  elif p==1:
     r[n][1]=y0+1
```

```
    elif p==2:
        r[n][0]=x0-1
    else:
        r[n][1]=y0-1
    # walker confined in in region 200x200 form x=-100 to x=100 and y
      =-100 to y=100
    if r[n][0]== L or r[n][1]==L or r[n][0]==-L or r[n][1]==-L:
        r[n][0]=x0
        r[n][1]=y0
```

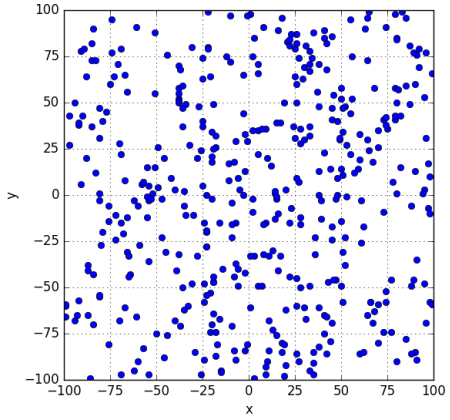
: diff_2d.py

در این برنامه Nstep تعداد دفعات تکرار است. در واقع Nstep تعداد کل گام‌هایی است که کل گام‌زن‌ها برمی‌دارند. بنابراین Nstep تعداد گام‌های یک گام‌زن به تنهایی نیست. بعد اجاری برنامه بالا مکان گام‌زن‌ها تغییر می‌کند و در جعبه (مربع) کاملا پخش می‌شوند. شکل مکان نهایی ذرات را بعد از $Nstep = ۵۰۰۰۰۰۰$ گام نشان می‌دهد. تا اینجا، کار خاصی انجام نداده‌ایم. برای اینکه از چنین



شکل ۲۰.۵: مکان نهایی گام‌زن‌ها در شبکه مربعی بعد ۵۰۰۰۰۰۰ گام

شبیه‌سازی ساده‌ای فیزیک جالبی بدست آوریم، می‌توانیم در حین پخش شدن گام‌زن‌ها، مقدار آنتروپی این گام‌زن‌ها (ذرات) را محاسبه کنیم. برای این منظور فضا را به مش‌بندی بزرگ‌تری تقسیم‌بندی می‌کنیم، مثلا ۸×۸ . شکل زیر ۲۱.۵ این مش‌بندی را نشان می‌دهد. حال در هر گام (یا در هر ۱۰۰۰ گام)، احتمال حضور ذرات یا همان گام‌زن‌ها را در هر یک از جعبه‌های فرضی که با مش‌بندی جدید بوجود



شکل ۲۱.۵: مکان نهایی گام‌زن‌ها در شبکه مربعی. در این شکل جعبه (مربع) به مش‌بندی ۸ × ۸ (جعبه‌های فرضی)تقسم بندی شده است. که در نهایت ۶۴ جعبه‌ی فرضی بوجود می‌آید.

می‌آیند را محاسبه‌می‌کنیم. این احتمال با معادله‌ی زیر داده می‌شود:

$$p_i = \frac{\text{تعداد گام‌زن‌ها یا همان ذرات در جعبه‌ی i ام}}{\text{تعداد کل گام‌زن‌ها یا همان کل ذرات}} \qquad (۱۲.۵)$$

سپس با استفاده از فرمول شانون که آنتروپی را به احتمال ربط می‌دهد می‌توان آنتروپی را محاسبه کرد. فرمول شانون [۲۰] به صورت زیر است:

$$S = -\sum_i p_i \log_۲(p_i) \qquad (۱۳.۵)$$

در این فرمول لگاریتم در مبنای ۲ است. می‌توان این فرمول با لگاریتم در مبنای e نیز نوشت که فقط یک ثابت به آن اضافه می‌شود. اگر این ثابت را در نظر نگیریم، فرمول شانون را می‌توان به صورت زیر در نظر گرفت:

$$S = -\sum_i p_i \ln(p_i) \qquad (۱۴.۵)$$

[۲۰] Shannon's formula

با استفاده از این فرمول، می‌توان آنتروپی را در گام‌های مختلف محاسبه کرد و رفتار آنتروپی وقتی تعداد گام‌ها زیاد می‌شود را مشاهد کرد. باید توجه کرد که در محاسبه‌ی $p_i \ln(p_i)$ ،اگر $p_i = ۰$ باشد $p_i \ln(p_i) = ۰$ می‌شود. منتها در کامپیوتر ممکن است به مشکل برخورد کنیم چرا که $\ln(۰)$ تعریف نشده‌است. برای همین برای محاسبه‌ی $p_i \ln(p_i)$ اول شرط اینکه $p_i > \epsilon$ است را اعمال می‌کنیم که $\epsilon = ۱۰^{-۶}$ یا یک عدد کوچک‌تر است. در واقع محاسبه جملاتی که در آن $p_i \simeq ۰$ است در جمع $-\sum p_i \ln p_i$ لزومی ندارد و اثری در جمع نمی‌گذارند. می‌توان به راحتی نشان داد که در فرمول شانون، وقتی آنتروپی بیشینه می‌شود که تمام p_i با هم برابر شوند. در این مسئله نیز این موضوع صادق است. تساوی احتمال‌های p_i وقتی اتفاق می‌افتد که احتمال حضور ذرات در تمام جعبه‌های فرضی برابر باشد. به سادگی مشخص است که این احتمال برابر با

$$p_i = \frac{۱}{N_b} \qquad (۱۵.۵)$$

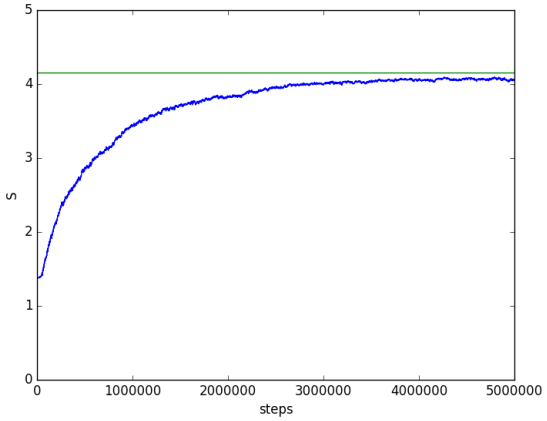
که N_b تعداد جعبه‌های فرضی است که در اینجا این تعداد برابر با ۶۴ است. بنابراین مقدار بیشینه‌ی آنتروپی برابر است با:

$$\begin{aligned} S_{\max} &= -\sum_{i=۱}^{N_b} p_i \ln(p_i) \\ &= -N_b \times (\frac{۱}{N_b}) \times \ln(\frac{۱}{N_b}) \\ &= \ln(N_b) \end{aligned}$$

اگر $N_b = ۶۴$ باشد، $S_{\max} = ۴/۱۵۸۸۸۳$ خواهد شد. نمودار S بر حسب گام‌ها در شکل ۲۲.۵ نمایش داده شده‌است. برای اینکه به عدد بیشینه نزدیک‌تر شویم باید تعداد ذرات و تعداد گام‌ها را افزایش دهیم.

تمرین ۷: محاسبه آنتروپی پدیده‌ی پخش در دو بعد

همانند آنچه در متن گفته شد، ابتدا تعدادی ذره را در وسط یک مربع به ابعاد ۱۰۰ × ۱۰۰ قرار دهید. سپس هر ذره را به عنوان یک گام‌زن تصادفی درنظر بگیرد و بگذارید همانند آنچه در



شکل ۲۲.۵: آنتروپی بر حسب گام‌ها. خط مستقیم مقدار بیشینه آنتروپی را نشان می‌دهد که در این مسئله برابر با $S_{\max} = ۴/۱۵۸۸۸۳$ است.

متن به آن اشاره شد، گام‌زن‌ها حرکت کنند. سپس در هر ۱۰۰۰ گام آنتروپی را از معادله‌ی ۱۴.۵ حساب کنید و آن را بر حسب گام رسم کنید. آخرین موقعیت ذرات را نیز رسم کنید.

فصل ۶

انتگرال گیری به روش مونت کارلو

در اکثر موارد در فیزیک و علوم دیگر به انتگرال‌گیری‌های پیچیده و چند بعدی برخورد می‌کنیم که حل آن‌ها به صورت صریح قابل انجام نیست. معمولا این انتگرال‌گیری‌ها را می‌توان به روش‌های عددی مختلفی حل کرد. روش‌های عددی برای حل انتگرال‌گیری‌ها را می‌توان به دو بخش اصلی تقسیم بندی کرد: ۱ـ انتگرال‌گیری‌هایی که براساس مش‌بندی فضا به صورت منظم بنا شده‌اند ۲ـ انتگرال‌گیری هایی که براساس انتخاب اعداد تصادفی به جای مش‌بندی انجام می‌شود. در این فصل قصد داریم تا روش‌های دوم را به صورت مختصر توضیح دهید. روش‌های دوم در واقع به روش‌های مونت کارلو معروف هستند و یادگیری این روش‌های کمک زیاد برای درک روش‌های شبیه سازی‌های کامپیوتری می‌کند که به نحوی به انتگرال‌گیری‌های مونت کارلو مرتبط هستند.

۱.۶ روش‌های انتگرال‌گیری عددی با استفاده از مش‌بندی