MATH327: Statistical Physics, Spring 2022

Computer Project — Part 1 Solutions

Exercise 1: Pseudo-random numbers

The exact mean for the uniform distribution is

$$\mu = \langle u \rangle = \int u\, p(u)\, du = \int_0^1 u\, du = \frac{1}{2}\left[u^2\right]_0^1 = \frac{1}{2},$$

at the middle of the interval $0 \le u < 1$ where $p(u) = 1$, as we might have guessed by inspection. The expectation value $\langle u^2 \rangle$ is just as easy to compute,

$$\langle u^2 \rangle = \int u^2\, p(u)\, dx = \int_0^1 u^2\, du = \frac{1}{3}\left[u^3\right]_0^1 = \frac{1}{3},$$

and produces the exact standard deviation

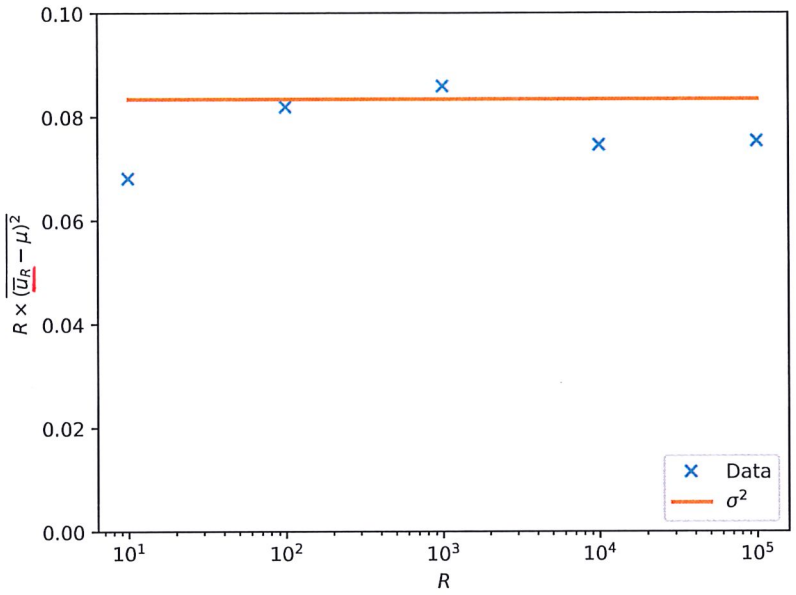
$$\sigma = \sqrt{\langle u^2 \rangle - \langle u \rangle^2} = \sqrt{\frac{1}{3} - \frac{1}{4}} = \frac{1}{\sqrt{12}} \approx 0.2887.$$

The Python code on the next page estimates the mean and standard deviation for the given R, producing the numerical results in the table below. This portion of the code runs in roughly 0.2 seconds on replit.com.

R	10	100	1000	10,000	100,000
$\overline{u}_R$	0.5580	0.4547	0.5026	0.4961	0.5003
$\overline{\sigma}_R$	0.2899	0.2923	0.2826	0.2886	0.2888

As R increases, these estimates for both the mean and standard deviation clearly approach the exact values above, matching them to several significant figures by the time $R = 100{,}000$. Although some small-R results show the larger discrepancies we would expect, others highlight the possibility of ending up close to the exact values by chance. In particular, $\overline{\sigma}_R$ for $R = 10$ is much closer to the exact result compared to $R = 100$.

In order to get a grip on these fluctuations, we need to repeat these estimates multiple times, which leads us to the next part of this exercise. The 99 additional estimates of the mean for each R run in roughly 20 seconds on replit.com, and produce the plot on the next page. In this plot, all five numerical results fall in the narrow range $0.068 \lesssim R \times \overline{(\overline{u}_R - \mu)^2} \lesssim 0.086$ even as R increases by four orders of magnitude. (Note the logarithmic scale on the x-axis.) In particular, the four results for $R \ge 100$ are within $\sim$10% of the constant $\sigma^2 = \frac{1}{12} \approx 0.083$ we would expect from page 14 of the lecture notes, with the $R = 10$ outlier still only $\sim$18% off. So we can conclude that we do indeed see constant results consistent with the expectation shown by the orange line.




```
# ---------------------------------------------------------------------
import random
import time
import numpy as np
import matplotlib.pyplot as plt
secs = -time.time()
random.seed(327)

# Set up known results and arrays to store fluctuations around them
mu = 0.5
R_list = [10, 100, 1000, 10000, 100000]
numR = len(R_list)
fluctSq = []        # Fluctuations (u_R - mu)^2

# Estimate mean and standard deviation for each R
print("   R   u_R    sigma_R")
for i in range(numR):
  R = R_list[i]
  sum = 0.0;
  sumSq = 0.0

  for r in range(0, R):
    u = random.random()
    sum += u;
    sumSq += u * u
  sum /= R;
  sumSq /= R
  fluctSq.append((sum - mu) * (sum - mu))
  print("%6d %.4f %.4f" % (R, sum, np.sqrt(sumSq - sum * sum)))
print("First estimates took %.4g seconds" % (secs + time.time()))
```

```
# Repeat N-1 more times, adding to fluctSq and ignoring st. dev.
N = 100
for n in range(N - 1):
  for i in range(numR):
    R = R_list[i]
    sum = 0.0
    for r in range(0, R):
      u = random.random()
      sum += u
    sum /= R
    fluctSq[i] += (sum - mu) * (sum - mu)

# Average over five fluctSq and multiply by R
for i in range(numR):
  fluctSq[i] *= R_list[i] / N

# Plot points, which should be constant
# For reference, also plot the expected constant, sigma^2=1/12
plt.plot(R_list, fluctSq, linestyle='None', marker="x", label='Data')
x = np.arange(10, 100000, 10)
p = 1.0 / 12.0 + 0.0 * x        # Trick to give p and x the same size
plt.plot(x, p, label='$\sigma^2$')
plt.xlabel('$R$')
plt.xscale('log')
plt.ylabel('$R\\times \overline{(\overline{u}_R - \mu)^2}$')
plt.ylim([0, 0.1])
plt.legend(loc='lower right')
plt.savefig('sol_ex1_const.pdf', bbox_inches='tight')
plt.clf()   # Clear figure

secs += time.time()
print("Exercise 1 took %.4g seconds\n" % secs)
# ----------------------------------------------------------------
```

Exercise 2: Inverse transform sampling

Starting from the relation $p(u)du = p(x)dx$ with $p(u) = 1$, we have

$$p(x) = \frac{du}{dx} = \frac{d}{dx}u(x) = \frac{d}{dx}\left(\sin(x) + \frac{1}{2}\right) = \cos(x).$$

When $u = 0$ we have $F(u) = \arcsin(-1/2) = -\pi/6$, while $u = 1$ produces $F(u) = \arcsin(1/2) = \pi/6$. Since $F(u)$ is monotonic throughout this domain, we can conclude $-\frac{\pi}{6} \le x < \frac{\pi}{6}$.

$\int_{-\pi/6}^{\pi/6} p(x)\,dx = 1$

This is all the information we need to compute the exact mean

$$\mu = \langle x \rangle = \int_{-\pi/6}^{\pi/6} x\, p(x)\, dx = \int_{-\pi/6}^{\pi/6} x\, \cos(x)\, dx = 0,$$

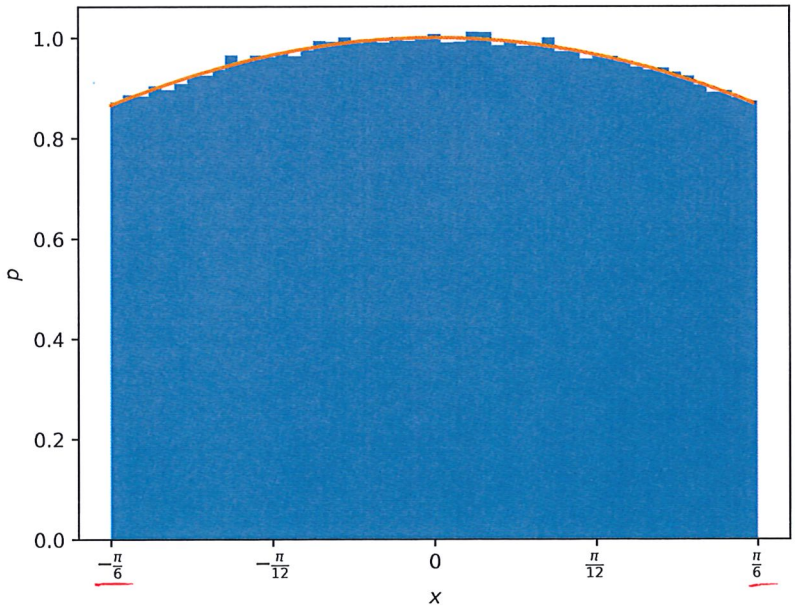
because $x\, \cos(x)$ is an odd function. The standard deviation is therefore just the square root of the variance

$$\sigma^2 = \langle x^2 \rangle = \int_{-\pi/6}^{\pi/6} x^2\, \cos(x)\, dx = \left[(x^2 - 2)\sin(x) + 2x\cos(x)\right]_{-\pi/6}^{\pi/6}$$
$$= 2\left[\left(\frac{\pi^2}{36} - 2\right)\left(\frac{1}{2}\right) + \frac{\pi}{3}\left(\frac{\sqrt{3}}{2}\right)\right],$$

which reduces to

$$\sigma = \sqrt{\frac{\pi^2 + 12\pi\sqrt{3} - 72}{36}} = \frac{\sqrt{\pi^2 + 12\pi\sqrt{3} - 72}}{6} \approx 0.2966.$$

The Python code on the next page produces $\overline{x}_R = 0.00006$ and $\sigma_R = 0.2965$, again matching the exact values to several significant figures. The code runs in roughly 5 seconds on replit.com, most of which is spent producing the plot below. This plot confirms that the (blue) histogram of $\{x_r\}$ matches the distribution $p(x)$ shown by the orange line.



```
# ---------------------------------------------------------------------
import random
import time
import numpy as np
import matplotlib.pyplot as plt
secs = -time.time()
random.seed(327)

# Initialize array and then fill it
dat = []
R = 1000000
for r in range(0, R):
  dat.append(np.arcsin(random.random() - 0.5))
mu = np.mean(dat)
sigma = np.std(dat)
print("u_R = %.4g and sigma_R = %.4g" % (mu, sigma))

# Plot the normalized histogram
bins = 51
plt.hist(dat, bins, density=True)
x = np.arange(-np.pi / 6.0, np.pi / 6.0, 0.01)
y = np.cos(x)
plt.xlabel('$x$')
plt.xticks(np.arange(-np.pi / 6.0, 0.53, np.pi / 12.0), \
           ['$-\\frac{\pi}{6}$', '$-\\frac{\pi}{12}$', '0', \
            '$\\frac{\pi}{12}$', '$\\frac{\pi}{6}$'])
plt.ylabel('$p$')
plt.plot(x, y)
plt.savefig('sol_ex2_hist.pdf', bbox_inches='tight')
plt.clf()   # Clear figure

secs += time.time()
print("Exercise 2 took %.4g seconds\n" % secs)
# ---------------------------------------------------------------------
```

Exercise 3: Random walks

(a) Central limit theorem

In class we saw (pages 23–25 of the lecture notes) that the central limit theorem relates

$$\langle X(N)\rangle = N\mu = 0 \qquad \ell_2(N) = \sqrt{N\sigma^2} = \frac{\sqrt{N\left(\pi^2 + 12\pi\sqrt{3} - 72\right)}}{6}.$$

(b) Fixed walk length

Setting $N = 100$, we have $\ell_2(100) = 10\sigma \approx 2.966$. The numerical results in the table below are produced by the following Python code, which runs in roughly 5 seconds on replit.com. Although small $R \lesssim 100$ lead to significant discrepancies, for $R = 10{,}000$ the results match a couple significant figures of the large-N predictions from the central limit theorem.

R	10	100	1000	10,000
$\overline{X}_R$	0.2068	-0.4556	0.0362	0.0122
$\overline{(\ell_2)}_R$	4.0341	2.9126	2.8573	2.9582

```
# ---------------------------------------------------------------
import random
import time
import numpy as np
secs = -time.time()
random.seed(327)

# Fix Nstep=100 and consider several R
Nstep = 100
print("  R   X_R    l2_R")
for R in [10, 100, 1000, 10000]:
  X = 0.0;    XSq = 0.0

  for r in range(0, R):
    d = 0.0
    for i in range(0, Nstep):
      d += np.arcsin(random.random() - 0.5)
    X += d
    XSq += d * d
  X /= R
  XSq /= R
  l = np.sqrt(XSq - X * X)
  print("%5d %2.4f %.4f" % (R, X, l))

secs += time.time()
print("Exercise 3b took %.4g seconds\n" % secs)
# ---------------------------------------------------------------
```

(c) Diffusion constant

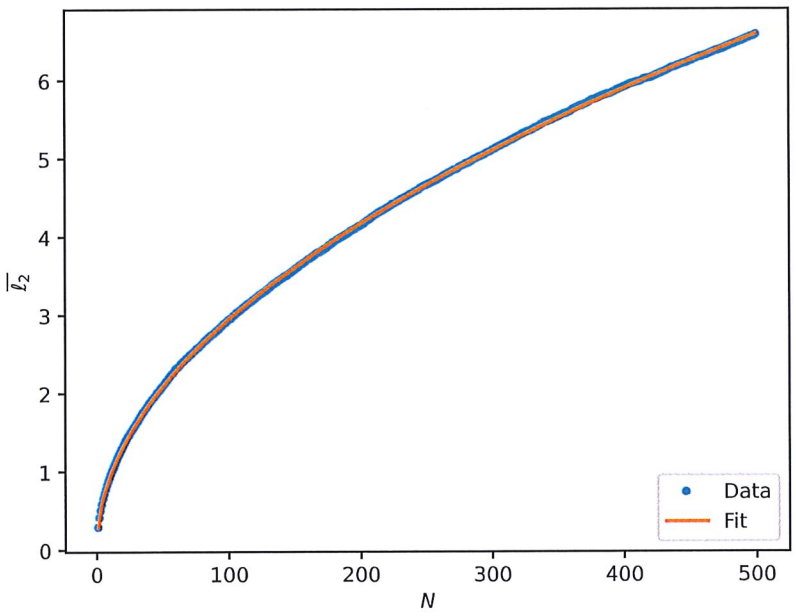
Following the hint to ignore potential correlations allows us to use the trick mentioned in the demo — treating the first N steps of each 500-step walk as an N-step walk. The plot below is produced by the following Python code, which runs in roughly 40 seconds on replit.com. In the plot there is no visible tension between the numerical data (blue points) and the fit (orange line) that produces

$$C = -0.0054 \qquad\qquad D = 0.2950.$$

As usual, these results match a couple significant figures of the analytical $C_{\text{exact}} = 0$ and

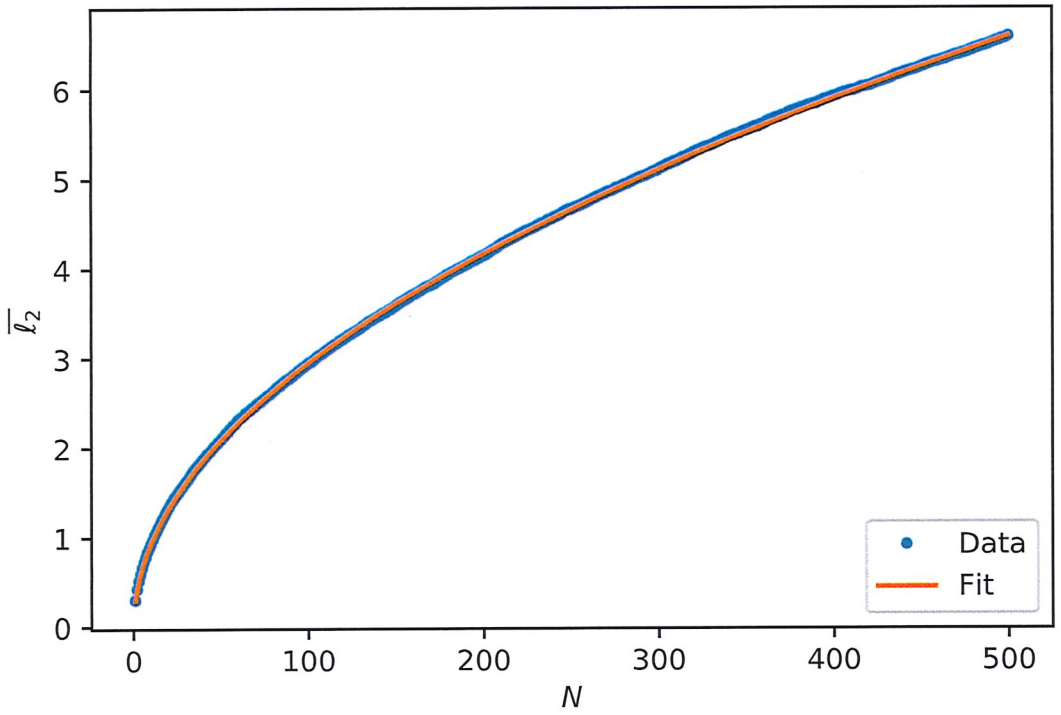
$$D_{\text{exact}} = \frac{\ell_2(N)}{\sqrt{N}} = \sigma = \frac{\sqrt{\pi^2 + 12\pi\sqrt{3} - 72}}{6} \approx 0.2966.$$

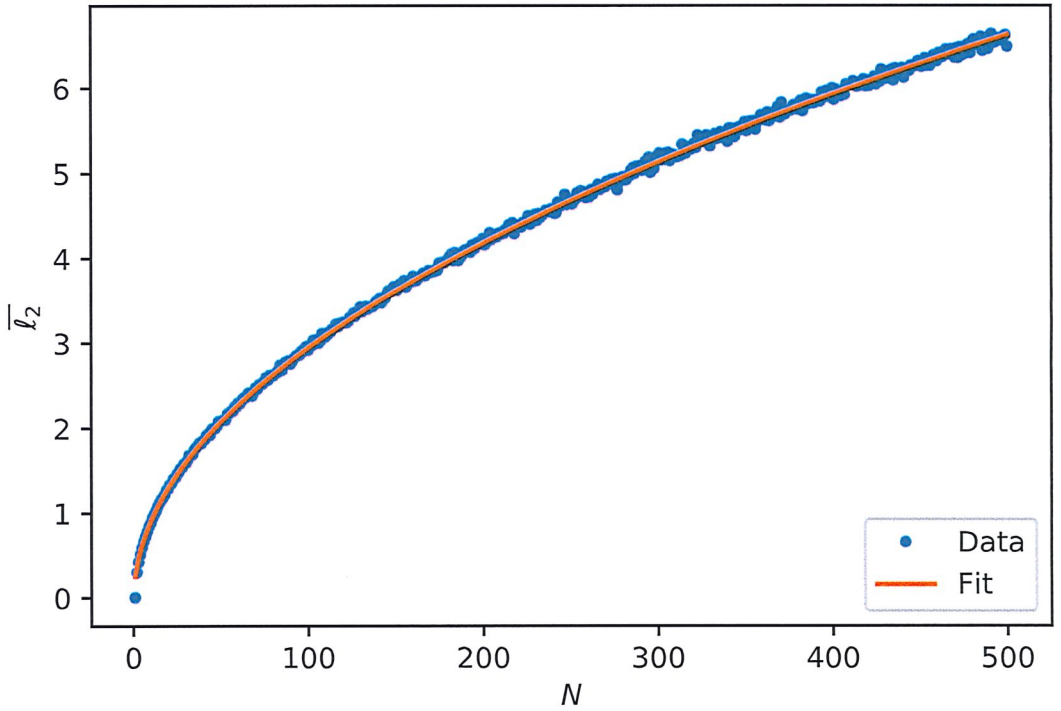
Everything works as expected when the central limit theorem is applicable.




```
# ---------------------------------------------------------------
import random
import time
import numpy as np
import matplotlib.pyplot as plt
secs = -time.time()
random.seed(327)
```

```
# Fix R=10000 and consider every Nstep = 1, ..., 500
# Use a single loop to accumulate l(N) after each step
Nstep = 500
R = 10000
X = np.zeros(Nstep)
XSq = np.zeros_like(X)
l = np.zeros_like(X)
for r in range(0, R):
  d = 0.0
  for i in range(0, Nstep):
    d += np.arcsin(random.random() - 0.5)
    X[i] += d
    XSq[i] += d * d

for i in range(0, Nstep):
  X[i] /= R
  XSq[i] /= R
  l[i] = np.sqrt(XSq[i] - X[i] * X[i])

# Now fit l(N) = C + D * sqrt(N)
# This is a linear fit vs. sqrt(N)
steps = np.arange(1, 501, 1)    # Count 1--500 rather than 0--499
sqrtN = np.sqrt(steps)
output = np.polyfit(sqrtN, l, 1)
C=output[1]
D=output[0]
print("C = %.4g" % C)
print("D = %.4g" % D)

# Plot both data and fit
plt.plot(steps, l, linestyle='None', marker=".", label='Data')
x = np.arange(1, 500, 0.1)
p = C + D * np.sqrt(x)
plt.plot(x, p, label='Fit')
plt.legend(loc='lower right')
plt.xlabel('$N$')
plt.ylabel('$\overline{\ell_2}$')
plt.savefig('sol_ex3_fit.pdf', bbox_inches='tight')
plt.clf()   # Clear figure

secs += time.time()
print("Exercise 3c took %.4g seconds\n" % secs)
# ----------------------------------------------------------------
```


$\overline{\ell_2}$
N
Data
Fit


$\overline{\ell_2}$
N
Data
Fit

MATH327: Statistical Physics, Spring 2022

Computer Project — Part 2

Instructions

In this second part of the computer project you will numerically analyze *anomalous diffusion* in a one-dimensional random walk, building on the numerical methods you developed for ordinary diffusion and checked against exact analytic predictions in the first part of the project.

There are two exercises below, which include some background information on the Cauchy–Lorentz distribution and anomalous diffusion. While the exercises mention some syntax specific to Python, you may use a different programming option if you prefer. This demo illustrates all the Python programming tools needed for the project. Even running slowly in the cloud via replit.com, the computing for each exercise should complete in a few minutes or less.

This part of the project is **due by 23:59 on Thursday, 24 March**, and anonymous marking is turned on. Submit it by file upload on Canvas.[1] **Both** your answers to the questions below and the code that produces your results must be submitted. These can be uploaded as separate files or in a tar/zip archive, as you prefer. With the exception of Mathematica `.nb` files, it will be quicker for me to check code submitted in its native format (for example, a `.py` file for Python code or a `.m` file for MATLAB code).

Exercise 4: Cauchy–Lorentz distribution

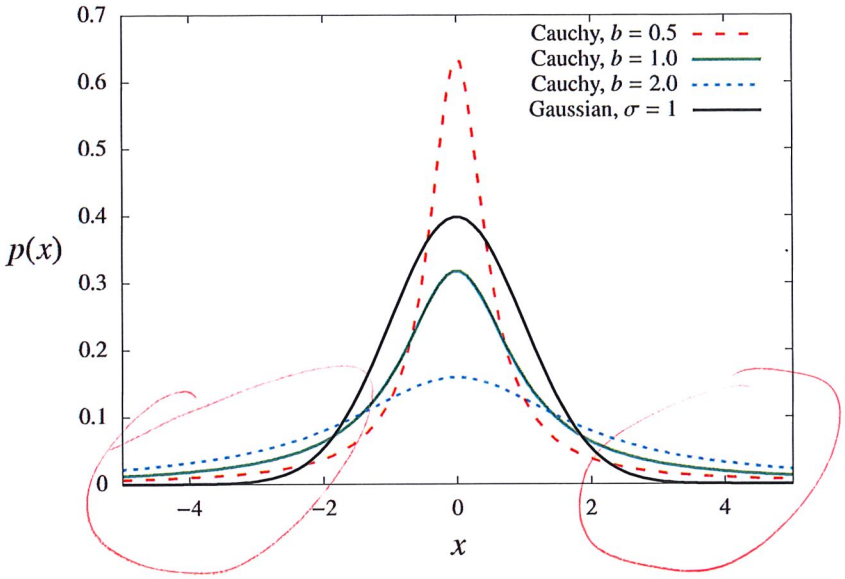
Background

The first part of this project involved the distribution $\cos(z)$ for $-\frac{\pi}{6} \le z < \frac{\pi}{6}$, for which we were able to apply the central limit theorem based on its mean 0 and standard deviation $\frac{\sqrt{\pi^2+12\pi\sqrt{3}-72}}{6}$. We now consider a more interesting distribution,

$$p_C(x) = \left(\frac{1}{b\pi}\right) \frac{1}{1+(x/b)^2} \qquad x \in \mathbb{R} \tag{1}$$

which is known as the Cauchy (or Cauchy–Lorentz) distribution. Here b is a constant parameter that controls the width of the peak in $p_C(x)$ around $x = 0$. The figure below illustrates this by plotting the Cauchy–Lorentz distribution for each of $b = 1/2$, $b = 1$ and $b = 2$, comparing them to the normal (or gaussian) distribution $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

[1] By submitting solutions to this assessment you affirm that you have read and understood the Academic Integrity Policy detailed in Appendix L of the Code of Practice on Assessment and have successfully passed the Academic Integrity Tutorial and Quiz. The marks achieved on this assessment remain provisional until they are ratified by the Board of Examiners in June 2022.



The figure shows how the peak of the Cauchy–Lorentz distribution around $x = 0$ becomes higher and narrower as b decreases. Even when its peak is very narrow, as $|x|$ increases $p_C(x)$ again becomes larger than the gaussian distribution, simply because the latter decreases exponentially quickly while $p_C(x)$ decreases only $\sim 1/x^2$. These "fat tails" at large $|x|$ make the Cauchy–Lorentz distribution both interesting and challenging to analyze.

Task

Fix $b = 2$ in the Cauchy–Lorentz distribution, so that Eq. 1 becomes

$$p_C(x) = \left(\frac{1}{\pi}\right)\frac{2}{4+x^2} \qquad\qquad x \in \mathbb{R}. \tag{2}$$

What is the integral of this distribution over its full range,

$$I \equiv \int_{-\infty}^{\infty} p_C(x)\,dx = \int_{-\infty}^{\infty}\left(\frac{1}{\pi}\right)\frac{2}{4+x^2}dx?$$

The usual starting point to analyze a probability distribution is finding its mean and standard deviation, by evaluating

$$\langle x\rangle = \int x\,p(x)\,dx \qquad\qquad \langle x^2\rangle = \int x^2\,p(x)\,dx.$$

For the Cauchy–Lorentz distribution in Eq. 2, consider instead the functions

$$f(a) = \int_{-a}^{a} x\,p_C(x)\,dx = \int_{-a}^{a}\left(\frac{1}{\pi}\right)\frac{2x}{4+x^2}dx$$
$$g(a) = \int_{-a}^{a} x^2\,p_C(x)\,dx = \int_{-a}^{a}\left(\frac{1}{\pi}\right)\frac{2x^2}{4+x^2}dx.$$

How do $f(a)$ and $g(a)$ behave in the limit $a \to \infty$?

[6 marks]

Turning to a numerical analysis of the Cauchy–Lorentz distribution, the first step is to determine the transform $F(u)$ that will map the uniformly distributed pseudo-random numbers u to $x = F(u) \in \mathbb{R}$. Recall that the uniform distribution implemented by the Python function `random.random()` is

$$p(u) = \begin{cases} 1 & \text{for } 0 \leq u < 1 \\ 0 & \text{otherwise} \end{cases} .$$

What is the transform F that provides $x = F(u)$ distributed according to $p_C(x)$ in Eq. 2?

Hints: Guided by the relation

$$p_C(x) = p(u)\frac{d}{dx}F^{-1}(x),$$

it will suffice to propose an ansatz for $F(u)$ based on integrating $p_C(x)$, and then follow the steps in Exercise 2 to confirm that this ansatz produces the desired distribution. Integrating will introduce a constant of integration, which can be chosen so that $x \to -\infty$ as $u \to 0$ and $x \to \infty$ as $u \to 1$.

[6 marks]

Now initialize the random number generator with seed $s = 327$. Generate $R = 1{,}000{,}000$ pseudo-random numbers $x_r = F(u_r)$ using the transform you found. Plot the histogram of these million $\{x_r\}$ and check whether it agrees with the Cauchy–Lorentz distribution shown above.

Hints: You will need to set an appropriate range for the x-axis of this histogram. A range $-10 \leq x \leq 10$ with roughly 200 bins should suffice to show all the interesting features. In Python this can be done by providing

```
bins = np.arange(-10.0, 10.0, 20.0/201.0)
```

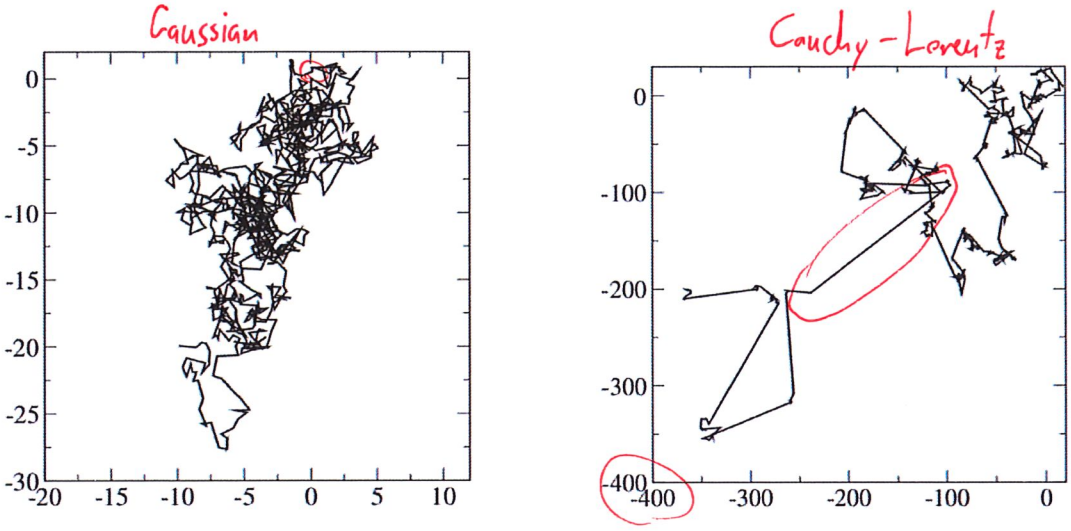
to the Matplotlib `hist` function used previously. In this exercise it is optional to plot $p_C(x)$ itself on top of this histogram—if you choose to do so, you may need to adjust its normalization (and you should think about why this is needed).

[8 marks]

Exercise 5: Anomalous diffusion

Background

The "fat tails" of the Cauchy–Lorentz distribution mean that $p_C(x)$ provides larger probabilities for *rare events* (with large $|x|$) to occur, compared to the gaussian distribution. This feature of the distribution is illustrated in the figures below, each of which shows a thousand-step random walk in two dimensions—randomly selecting both the size of each step and the direction $0 \leq \phi < 2\pi$ in which to step. The walk shown on the left uses step sizes drawn from a gaussian distribution. Even in two dimensions, random walks of this sort obey the law of diffusion, with a diffusion length growing proportionally to the square root of the number of steps, $\ell_2(N) = \sqrt{\langle [X(N)]^2 \rangle - \langle X(N) \rangle^2} \propto \sqrt{N}$.



The walk shown on the right instead uses step sizes drawn from a Cauchy–Lorentz distribution. Note that the axes for this figure cover a much larger range! The fat tails of the Cauchy–Lorentz distribution result in occasional very large jumps, leading to random walks that do not obey the law of diffusion.

Returning to one-dimensional random walks, some of the results from Exercise 4 motivate defining the **generalized diffusion length**

$$\ell_\theta(N) = \left\langle |X(N)|^\theta \right\rangle^{1/\theta}, \tag{3}$$

which depends on a positive real parameter $\theta > 0$. Since θ is not necessarily an integer, the absolute value is needed to ensure $\ell_\theta \in \mathbb{R}$, rather than becoming complex valued. If $\langle X(N) \rangle = 0$ and ℓ_θ is well-defined with $\theta = 2$, then this generalized diffusion length could reproduce the standard deviation ℓ_2 and exhibit the ordinary law of diffusion, $\ell_2 \propto N^{1/2}$.

For the Cauchy–Lorentz distribution, ℓ_θ is ill-defined for any $\theta \geq 1$. This parameter θ can take only values $0 < \theta < 1$. The resulting ℓ_θ exhibits **anomalous diffusion**,

$$\ell_\theta(N) \propto N^\alpha,$$

where the exponent is either $\alpha > \frac{1}{2}$ (called *super-diffusion*) or $0 < \alpha < \frac{1}{2}$ (called *sub-diffusion*). This exercise investigates the exponent α for the distribution $p_C(x)$ in Eq. 2, and checks whether or not α depends on θ.

$\alpha = \frac{1}{2}$ for ordinary diffusion

Task a: Fixed number of steps

Reset by initializing the random number generator with seed 327. With fixed $N = 100$, generate R 100-step random walks,

$$X_r(N) = \sum_{i=1}^{N} x_i \qquad\qquad r = 1, 2, \cdots, R,$$

for each of the five $R = 10$, 100, 1000, $10{,}000$ and $100{,}000$. Use the resulting X_r to numerically estimate

$$\overline{\ell_\theta(N)}_R \approx \left[\frac{1}{R} \sum_{r=1}^{R} |X_r(N)|^\theta \right]^{1/\theta}$$

for three values of $\theta = 0.25$, 0.5 and 0.75. (**Hint:** NumPy provides both an `abs` function to take the absolute value, and a `power` function to compute non-integer powers.)

[12 marks]

Task b: Anomalous diffusive exponent

Reset by initializing the random number generator with seed 327. Then fix $R = 10{,}000$ and estimate $\overline{\ell_\theta(N)}_R$ for *every* $N = 1, 2, \cdots, 250$, again considering $\theta = 0.25$, 0.5 and 0.75. Instead of reporting your numerical results, plot all three $\ell_\theta(N)$ vs. N in a single figure. (**Hint:** You can ignore potential correlations between $\ell_\theta(N)$ for different values of N.)

[8 marks]

Now fit your numerical results for each $\theta = 0.25$, 0.5 and 0.75 to the function

$$\ell_\theta(N) = DN^\alpha.$$

Report your results for D and α, and comment on their sensitivity to the value of θ. (**Hint:** Optionally testing different values of R, N or θ may help to distinguish between real sensitivity vs. statistical fluctuations, if you are unsure whether or not an observed effect is significant.)

[10 marks]

MATH327: Statistical Physics, Spring 2022

Tutorial comments — Mixing entropy

We are given the initial entropy

$$S_0 = 2S_I(N,V) = 5N + 2N\log\left(\frac{V}{N\lambda_{\text{th}}^3}\right)$$

and the partition function for the combined system

$$Z_C = \frac{1}{N!}\frac{1}{N!}\left(\frac{2V}{\lambda_{\text{th}}^3}\right)^{2N}.$$

We could simply repeat the process we went through in class, of using this partition function to determine the Helmholtz free energy $F = -T\log Z$, the internal energy $\langle E\rangle = \frac{\partial}{\partial\beta}(\beta F)$ and then the entropy $S = \beta(\langle E\rangle - F)$, but there is a trick we can use to reuse that work we've already done. Specifically, if we rewrite the combined partition function as

$$Z_C = \left[\frac{1}{N!}\left(\frac{2V}{\lambda_{\text{th}}^3}\right)^{N}\right] \times \left[\frac{1}{N!}\left(\frac{2V}{\lambda_{\text{th}}^3}\right)^{N}\right],$$

we can recognize that it describes two completely independent gases of N indistinguishable particles each in a container of volume $2N$. The entropy is therefore

$$S_C = 2S_I(N,2V) = 5N + 2N\log\left(\frac{2V}{N\lambda_{\text{th}}^3}\right)$$

and the mixing entropy is just

$$S_{\text{mix}} = S_C - S_0 = 2N\log 2.$$

This is exactly the same mixing entropy we computed in class for the case in which all $2N$ particles were completely distinguishable! Of course, both S_C and S_0 are smaller than what we would have for fully distinguishable particles, even though the difference between them comes out the same. In a sense, this single distinguishing feature (colour) between the two sets of indistinguishable particles suffices to introduce the same relative amount of additional information upon mixing them.

Before moving on to the final system, let me highlight a more generally trick that we can use to consider *differences* of entropies. Recalling

$$S = -\frac{\partial}{\partial T}F = \frac{\partial}{\partial T}(T\log Z),$$

we can write the difference as

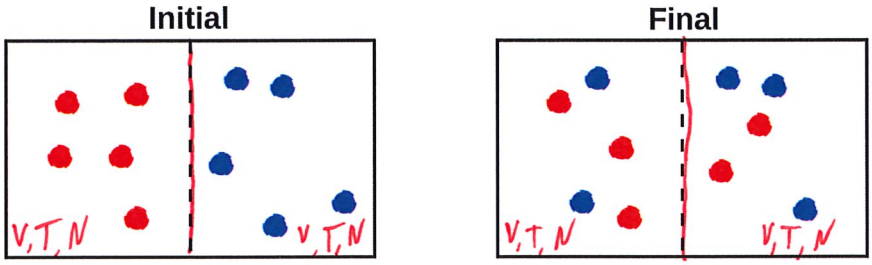
$$\begin{aligned} S_{\text{mix}} = S_C - S_0 &= \frac{\partial}{\partial T}(T\log Z_C - T\log Z_0) = \frac{\partial}{\partial T}\left(T\log\frac{Z_C}{Z_0}\right) \\ &= \log\frac{Z_C}{Z_0} + T\frac{\partial}{\partial T}\log\frac{Z_C}{Z_0}. \end{aligned}$$

MATH327: Statistical Physics, Spring 2022

Tutorial problem — Mixing entropy

Let's consider a slight variation to the particle exchange thought experiment we worked through in class. Initially, we still have two canonical ideal gases, initially separated by a wall, each with N particles in volume V at temperature T. **All** $2N$ particles have identical physical properties, *except* that those initially in the left compartment (the "reds") are distinguishable from those in right compartment (the "blues") by their colour. Call this initial system Ω_0. We have already computed its entropy $S_0 = 2S_I(N,V) = 5N + 2N\log\left(\frac{V}{N\lambda_{\rm th}^3}\right)$, where $\lambda_{\rm th} = \sqrt{2\pi\hbar^2/(mT)}$.

We then carry out the procedure of removing the wall, allowing the combined system to reach thermodynamic equilibrium, and then re-inserting the wall to re-separate the two systems. Call the combined system Ω_C with entropy S_C. As discussed in class, it's safe to assume that N particles end up in each of the two re-separated systems. However, red and blue particles can now appear in either of the two re-separated systems. Call this final system Ω_F with entropy S_F. The initial and final systems are illustrated by the figure below.



2N
2V

The first task is to compute the mixing entropy $S_{\rm mix} = S_C - S_0$, where in the combined system Ω_C we now have two sets of N indistinguishable particles, but can distinguish between the two sets. The starting point is the partition function

$$Z_C = \frac{1}{N!}\frac{1}{N!}Z_1^{2N} = \frac{1}{N!}\frac{1}{N!}\left(\frac{2V}{\lambda_{\rm th}^3}\right)^{2N},$$

where $Z_1 = 2V/\lambda_{\rm th}^3$ is the single-particle partition function.

The second task is to compute the final entropy S_F, to see whether $S_F \geq S_C$ as demanded by the second law of thermodynamics. We can break this up into two steps. The first of these is to compute the partition function Z_F of the two re-separated systems (each with N particles), summing over all ways of dividing the red and blue particles between them. The following special case of the Zhu–Vandermonde identity may be useful for this step:

$$\sum_{k=0}^{N}\binom{N}{k}^2 = \binom{2N}{N}.$$

Finally, use your result for Z_F to determine the final entropy S_F.

This is useful because there are often many cancellations in the ratio of the partition functions. In particular, in our case the ratio is independent of T (which appears in λ_{th}), so the second term vanishes and

$$S_{\text{mix}} = \log \frac{Z_C}{Z_0} = \log \left[\frac{\frac{1}{(N!)^2} \left(\frac{2V}{\lambda_{\text{th}}^3} \right)^{2N}}{\left(\frac{1}{N!} \left[\frac{V}{\lambda_{\text{th}}^3} \right]^N \right)^2} \right] = 2N \log 2.$$

reproducing our result above without having to rely on expressions from class.

Now the fun begins. Based on our previous work, we can assume that there will be N particles in each of the re-separated systems. If ν of the N particles in the left system are red, then the remaining $N - \nu$ must be blue, leaving $N - \nu$ red particles and ν blue particles in the right system. The corresponding contribution to the partition function is therefore

$$Z_\nu = \left[\frac{1}{\nu!} \left(\frac{V}{\lambda_{\text{th}}^3} \right)^\nu \times \frac{1}{(N-\nu)!} \left(\frac{V}{\lambda_{\text{th}}^3} \right)^{N-\nu} \right]^2 = \frac{1}{(\nu!)^2 \left([N-\nu]! \right)^2} \left(\frac{V}{\lambda_{\text{th}}^3} \right)^{2N}.$$

We can relate the factorials to a squared binomial coefficient,

$$Z_\nu = \frac{1}{(N!)^2} \binom{N}{\nu}^2 \left(\frac{V}{\lambda_{\text{th}}^3} \right)^{2N}.$$

This will allow us to apply the Zhu–Vandermonde identity when we sum over all possible values of $0 \leq \nu \leq N$ that describe how the red and blue particles can be divided between the two systems:

$$Z_F = \sum_{\nu=0}^{N} Z_\nu = \frac{1}{(N!)^2} \left(\frac{V}{\lambda_{\text{th}}^3} \right)^{2N} \sum_{\nu=0}^{N} \binom{N}{\nu}^2 = \frac{1}{(N!)^2} \left(\frac{V}{\lambda_{\text{th}}^3} \right)^{2N} \binom{2N}{N}$$

We can now use our trick above to find $S_F = (S_F - S_C) + S_C$. The partition function ratio is still T-independent, so

$$S_F - S_C = \log \frac{Z_C}{Z_0} = \log \left[\frac{\left(\frac{1}{N!} \left[\frac{V}{\lambda_{\text{th}}^3} \right]^N \right)^2 \binom{2N}{N}}{\frac{1}{(N!)^2} \left(\frac{2V}{\lambda_{\text{th}}^3} \right)^{2N}} \right] = \log \binom{2N}{N} - 2N \log 2.$$

Repeating some work from a previous tutorial, we can use Stirling's formula to approximate

$$\log \binom{2N}{N} = \log[(2N)!] - 2\log(N!) \approx 2N \log(2N) - 2N - 2N \log N + 2N = 2N \log 2,$$

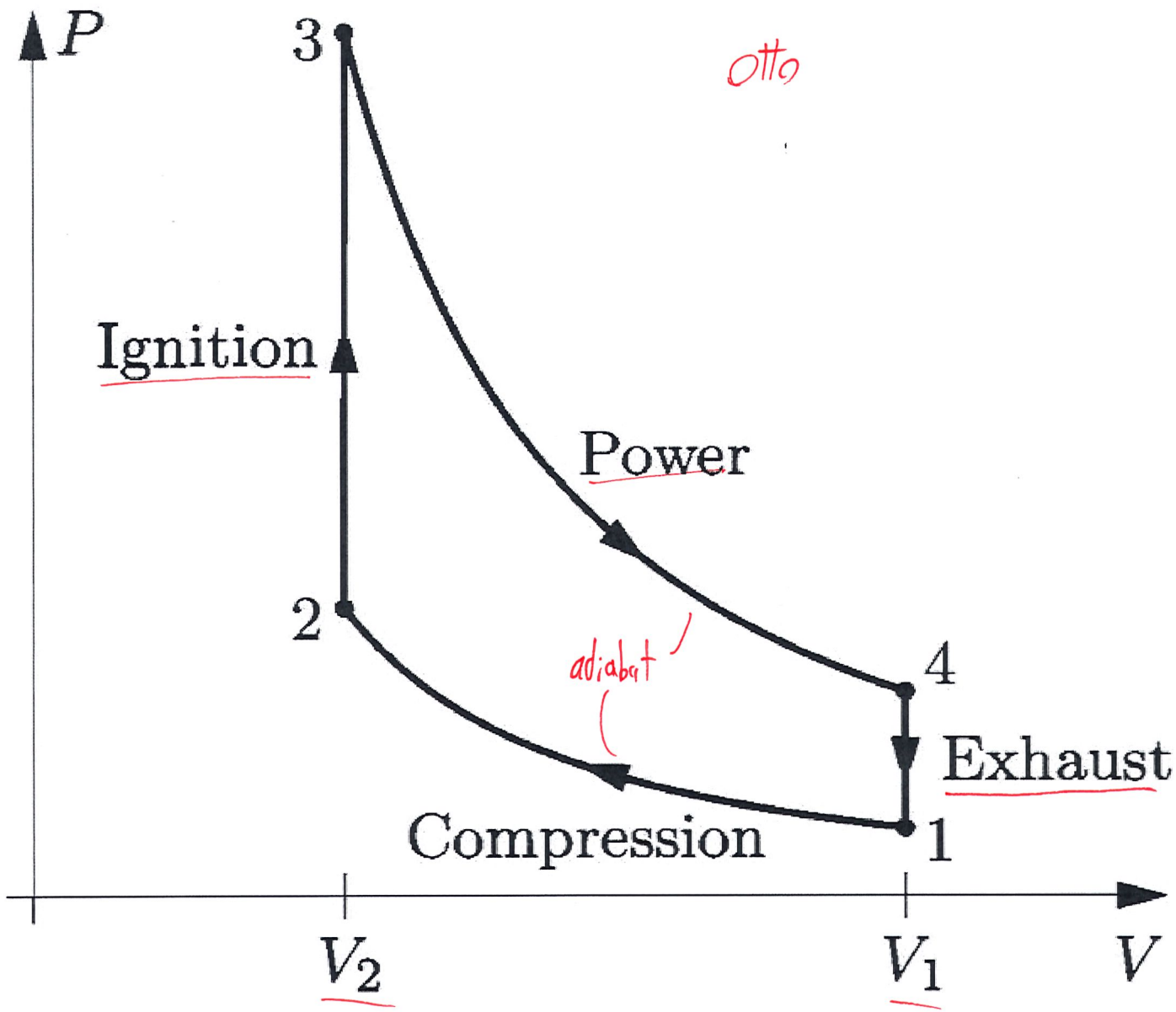
so that

$$S_F = (S_F - S_C) + S_C \approx (2N \log 2 - 2N \log 2) + S_C = S_C.$$

So at this level of approximation, everything remains consistent with the second law of thermodynamics, as it should:

$$S_F \approx S_C > S_0.$$

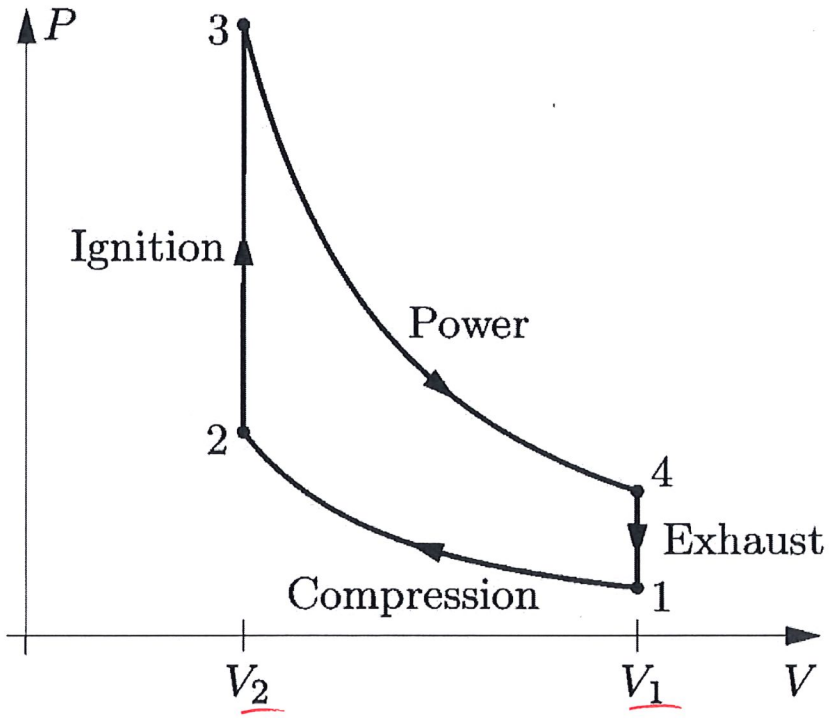
However, if you extend this exercise by retaining the $\log\left(\sqrt{2\pi N}\right)$ terms in Stirling's formula, you will find a negative $S_F - S_C$, which should leave you concerned. (Try it and see!) The resolution to this extra conundrum is to recall that the full computation of the entropy still requires summing over re-separated systems with different numbers of particles, $N \pm k$. Those additional contributions will enter at $\mathcal{O}(\log N)$, which is usually negligible, but matters when all the larger factors cancel out in the difference $S_F - S_C$.


P
3
Ignition
Power
2
4
Exhaust
Compression
1
V_2
V_1
V

MATH327: Statistical Physics, Spring 2022

Tutorial problem — Otto cycle

The figure below shows the 'Otto cycle' that describes an idealized petrol engine. The compression and expansion ('power') stages are adiabatic, while the volume is fixed at V_2 for the 'ignition' stage that burns the fuel to produce heat, and at $V_1 > V_2$ for the 'exhaust' stage that replaces the burnt fuel with cooler, fresh gas. The **compression ratio** is defined as $r \equiv V_1/V_2 > 1$.



The efficiency η of the Otto cycle depends *only* on the compression ratio r. What is this efficiency? How does it compare to the efficiency of the Carnot cycle? How should V_1 and V_2 be chosen to maximize the efficiency?

Hint: Given the labels in the diagram above, T_1 would be the low temperature of the cold reservoir while T_3 would be the high temperature of the hot reservoir. The corresponding Carnot cycle efficiency is therefore $\eta_{\text{Carnot}} = 1 - \frac{T_1}{T_3}$, and the comparison is easiest if the Otto cycle efficiency is expressed in terms of temperatures rather than volumes.