Mathematical Programming MATH20014: Group projects

September 29, 2025

Outline of the group projects

The group projects for MATH20014 will be organised as follows. From the set of potential projects detailed below, each of you will choose a favourite project and backup projects in decreasing order of preference, according to their interest to you. The projects are all of a similar level of difficulty, but focus on different areas of mathematics, including mechanics, statistical mechanics, combinatorics, cryptography, number theory and knot theory.

Each group will then consist of four or five students who have shown interest in the same project; the groups will be assembled by the team of lecturers.

Format of the group projects

We expect the group project to take the form of a single jupyter notebook, similar to the lectures and tutorials. Your jupyter notebook will contain the different pieces of code, and the same jupyter notebook will also need to contain explanations, formulas, graphs, context and references for your project.

As you have seen in the lectures and tutorials, jupyter notebooks understand latex, so typesetting equations is straightforward. On the other hand, you may also want to embed or run other forms of content in your notebook. To do this you may also create and submit—alongside your notebook file—other types of file, including the following.

- Python code (.py) files with, for example, modules or classes that you have created, or scripts (for example scripts that carry out the time-consuming calculations or animations required by some projects).
- High-resolution images and movies obtained as output from your programs/functions.

Assessment criteria

The projects presented here contain a core part (marked **[core]** or **(core)** below) which you need to satisfactorily complete to obtain a pass mark. The other parts are extensions that you can pick and choose. The grading criteria fall under headings 1-5 below. A detailed outline can be found in the file `assessment_criteria.pdf` on Blackboard (in the same folder as the present file).

1. **Correctness.**

   (Does your code do what it is supposed to do?)

2. **Quality: structure and design.**

   (Is the structure of your code clear and easy to maintain and is it user friendly?)

3. **Documentation and Clarity.**

   (Is your code written in such a way that the interested programmer can easily understand what is going on?)

4. **Scope.**

   (How well have you developed the various parts of the project including at least one of the extensions/non core parts.)

5. **Project Writing.**

   (How good is the surrounding text of the project itself?)

Peer assessment and project submission

There will also be a peer assessment of your group where you rate the commitment and performance or your group members. This will take a similar form to the group projects in Mathematical Investigations in year 1. **The projects and peer assessments are due at 12 Noon Wednesday 4th December.**

How to collaborate on a computing project

The first step is to make sure that you have a shared drive that is accessible to all members of your team, for example by creating a shared folder on Onedrive. Then create your Jupyter notebook for the project, and edit it. A key issue is **version control**, i.e. that there is only one current version of the project (and not, say, five different branches that five different people are working on). At the same time, you will want to keep the previous versions of the notebook, so that potentially important pieces are not overwritten or lost and you can always undo changes that were a mistake.

The simplest (though not the most efficient) form of version control is to save a copy of the Jupyter notebook with the current date and your name each time you make an edit, and to always, always start from the most recently modified file.

A second very important point: the projects generally start from a basic idea and then develop this through several stages. It is crucial that you do the beginning pieces first. It is also crucial that you work on **one** version of it **together**. For example, to figure out energy conservation in planetary motion, you need to have a working integrator of the four ODEs first, etc. Please do not try to develop pieces of the project individually and then merge them, as this is almost impossible to do due people's different choices for the structure of the code. You should begin by having an online group meeting where you work through the project and set up the code structure, followed by regular meetings.

Setting up python on your own computer.

For collaboration using a shared folder on oneDrive you will need to use Python and Jupyter either on a workstation in the Computer Lab or on your own computer. In order to use your own computer for this we recommend that you install the appropriate version of the Anaconda package manager from: `https://docs.anaconda.com/anaconda/install/` . Once set up, run Anaconda Navigator. From there launch Jupyter Notebook of Jupyter Lab.

Alternative ways of collaborating on a computing project

- **GitHub** If you are keen to explore more formal options for working on a coding project as a team, please consider using the **git software**, and optionally the github online code sharing platform (`https://github.com/`).

- **Noteable** provides an option for collaborative editing. Instructions for this can be found here: https://noteable.edina.ac.uk/documentation/collab-editing/. However (as noted in the instructions) "When inviting someone to collaboratively edit a notebook, you are bypassing all other login/authentication processes, and inviting them to access your personal workspace." Moreover access to the shared file(s) is revoked when the person sharing shuts down their server. So, although this is an option we do not recommend using Noteable for collaboration at present.

1 Planetary motion

We will investigate how planets move around stars and how moons move around planets. Everything starts with Newton's equations of motion for the objects (eg, for our solar system, the Sun, the eight planets and their moons). Let $\mathbf{r_i}$ denote the position of the ith object and m_i its mass. Then the equations of motion are

$$m_i \ddot{\mathbf{r}}_\mathbf{i} = G \sum_{j \neq i} m_i m_j \frac{\mathbf{r_j} - \mathbf{r_i}}{|\mathbf{r_j} - \mathbf{r_i}|^3}, \tag{1}$$

where G is the universal gravitational constant. In units where distances are measured in meters and masses in kilograms, $G = 6.6734810^{-11} \frac{\mathrm{m}^3}{\mathrm{kg\,s}^2}$.

Gravity is a conservative force. This means that the total energy of the system, $E = T + V$, ie the sum of the kinetic and potential energies, does not vary in time. The total energy is given by

$$E = \tfrac{1}{2} \sum_i m_i |\dot{\mathbf{r}}_i|^2 - G \sum_{i>j} \frac{m_i m_j}{|\mathbf{r}_j - \mathbf{r}_i|}. \tag{2}$$

(Note the restriction $i > j$ on the second sum. This is to avoid double counting; there is a single contribution to the potential energy from each pair of masses.)

Newton's equations of motion for conservative forces like gravity have a special property. They are examples of what are called *Hamiltonian systems.* If you are taking Mechanics 2, you will already have learned about Lagrangian mechanics, which is closely related to Hamiltonian systems; indeed, you will be studying Hamiltonian systems in the last part of Mechanics 2.

Hamiltonian systems have a special mathematical property, called *symplectic structure.* We would like to use a numerical integration scheme that preserves this structure. Unfortunately, the standard algorithms that we have discussed previously, namely the Euler method and the Runge-Kutta algorithm, do not preserve symplectic structure. Therefore, for this project, we will use a different integration scheme, which does preserve the symplectic structure of Newtonian dynamics with conservative forces. This is the *velocity Verlet algorithm.* Fortunately, it is quite easy to code.

[A few comments about numerical accuracy: It turns out that the velocity Verlet algorithm is a second-order method; that is, the errors in the approximation are proportional to $(dt)^3$, where dt is the time step. In this respect, it is inferior to the 4th-order Runge-Kutta algorithm. There are higher-order versions of the velocity Verlet algorithm that preserve symplectic structure, but we will not consider them here.]

Consider Newton's equations of motion for a particle moving in three dimensions subject to a conservative force, expressed as a system of first-order differential equations for the position $\mathbf{r}(t)$ and velocity $\mathbf{v}(t)$:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{3}$$

$$\dot{\mathbf{v}} = \mathbf{F}(\mathbf{r})/m = \mathbf{a}(\mathbf{r}). \tag{4}$$

Note that $\mathbf{a}(\mathbf{r})$ is the acceleration. The fact that the force is conservative means that it depends only on position, and moreover that it is minus the gradient of a potential energy, ie $\mathbf{F}(\mathbf{r}) = -\nabla U(\mathbf{r})$.

The velocity Verlet algorithm is given by

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{v}_k \, dt + \frac{1}{2}\mathbf{a}(\mathbf{r}_k) \, dt^2 \tag{5}$$

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \frac{1}{2} \left(\mathbf{a}(\mathbf{r}_k) + \mathbf{a}(\mathbf{r}_{k+1})\right) \, dt \tag{6}$$

Here, dt is the time step, which is regarded as small, and $\mathbf{r}_k$ and $\mathbf{v}_k$ denote the position and velocity of the particle after k time steps (ie, at time $t = k\,dt$). Take careful note of when and where the terms in the velocity Verlet algorithm are evaluated. For example, the expression for $\mathbf{v}_{k+1}$ involves the acceleration at both $\mathbf{r}_k$ and $\mathbf{r}_{k+1}$.

1.1 Project activities

The non-core activities 5–9 below are pick and choose according to your interests. You should choose one of these activities to develop. If you would like to develop more than one, you will only be marked on one of these extensions, so please make it clear which you would like to be included in the project marking.

1. **[core]** Use the velocity Verlet algorithm to solve Newton's equations of motion for the one-dimensional harmonic oscillator, $m\ddot{x} = -kx$, or
$$\dot{x} = v, \qquad \dot{v} = -\frac{k}{m}x. \tag{7}$$
Compare the numerical solution and the exact solution by plotting $x(t)$ and the phase-space trajectories $(x(t), v(t))$ for both.

2. **[core]** Simulate the motion of an earth-like planet around the sun. The mass of the sun is $1.989\,10^{30}$ kg, and the radius of the earth's orbit is on average 149.6 million kilometers (i.e. $R = 149.6\;10^9$ m). You can assume that the sun is fixed and that only the earth moves, which is a pretty good approximation in practice. As a consequence of the conservation of angular momentum, you can assume that the sun and the earth lie in a plane, say the xy–plane. With this assumption, the position and velocity of the Earth, $\mathbf{r}$ and $\mathbf{v}$, can be taken to be two-dimensional vectors.

3. **[core]** Verify that you find a closed orbit for a planet with the earth's initial position $\mathbf{r}_0 = R\,\hat{\mathbf{x}}$ and tangential initial velocity $\mathbf{v_0} = 29.8\frac{\text{km}}{\text{s}}\,\hat{\mathbf{y}}$. Check that the total energy remains constant. Analytically derive the speed v_0 by assuming that the Earth's orbit is circular. (The equation that describes circular motion is $F = mv^2/r$, where r denotes the radius of the circle, F denotes the magnitude of the force, which is directed towards the centre of the circle, and v denotes the magnitude of velocity, which is assumed to be constant.)

4. **[core]** Now vary the total energy of the orbit by changing the value, but not the direction, of $\mathbf{v_0}$ over a range of values that produce both bounded and unbounded trajectories. Plot the trajectories, which are now either ellipses or hyperbola (or, for a specific value of the velocity, a parabola). For the elliptical trajectories, calculate the perihelion P (closest distance to the Sun) and the aphelion A (furthest distance from the Sun). From these, calculate the semimajor axis a, the semiminor axis b, and the eccentricity ϵ of the elliptical orbit using the following formulas:
$$\begin{aligned} a &= (A+P)/2, \\ b &= \sqrt{AP}, \\ \epsilon &= \frac{A-P}{A+P}. \end{aligned}$$
Construct a criterion to distinguish elliptical and hyperbolic trajectories. Determine the escape speed of an Earth-like planet from the Sun's gravitational field. Determine the corresponding energy, and compare it to the theoretical result, which you can derive from $(T+V)|_{\text{initial}} = (T+V)|_{r=\infty}$.

5. For the elliptical trajectories, test **Kepler's second law**, which states that “A line joining a planet and the Sun sweeps out equal areas during equal intervals of time”. In other words, if A is the area enclosed by i) the line from the Sun along the positive x-axis, ii) the line from the Sun to the planet, and iii) the orbital path the planet, then A increases at a constant rate. This result is a consequence of the conservation of angular momentum. In your text, given an analytic derivation of Kepler's second law.

6. Investigate the **three-body problem** where three objects interact with each other gravitationally. For example, consider three Sun-like stars with different but not too dissimilar masses. To simplify the simulation, assume that all motion takes place in the xy-plane. Even with this simplification, the three-body problem does not have an analytical solution and almost always leads to chaotic trajectories (unless one of the stars is ejected and escapes to infinity).

7. Simulate the **solar system** including multiple or all planets (note that planets exert gravitational forces on each other). To simplify the simulation, assume that all motion takes place in the xy-plane (in fact, the orbits of the planets are nearly but not quite coplanar). It is hard to produce accurate results o, since the periods of the outer planets are much longer than the inner planets. We therefore obtain *numerically stiff* differential equations, where we need a small time step because of the inner planets, but also a long simulation time for the outer planets. Several compromises are possible, e.g. only simulating inner or outer planets, and not simulating long enough to find closed trajectories for outer planets.

8. Simulate the trajectory of a large (at least Jupiter-sized) **exoplanet** in a star system, with the planet close to the star. Drop the assumption that the star itself is not moving, and compute the “wobble” of the star's trajectory itself around the common centre of gravity. Make an estimate at what distance this wobble would be observable from earth: this is one of the methods for detecting exoplanets. Also, determine the scale of the sun's motion due to its gravitational interaction with the earth.

9. **Solar eclipses.** Predict the occurrence of solar eclipses by simulating the motion of the Sun, Earth and the Moon.

   A solar eclipse occurs when the Moon lies between the Sun and the Earth in such a way that the Moon's shadow falls on the surface of the Earth. In order to determine when this occurs, you will need to know the Earth's position to an accuracy of less than its radius. Given that the span of the Earth's orbit is on the order of 10^{11} km and the radius of the Earth is on the order of 10^4 km, this means the position of the Earth needs to be accurate to more than seven significant digits.

   This is an inherently three-dimensional problem; you should not assume that the Sun, Earth and Moon move in a plane. If they did, there would be a solar eclipse roughly every month, ie once during every lunar period. In fact, the plane of the Moon's orbit about the Earth is tilted with respect to the Earth's orbit about the Sun by about 5°.

   You may assume that the Sun remains fixed at the origin.

   (a) **Physical constants.**

• Mass of Sun: $1.988420392 \times 10^{30}$kg	Radius of Sun: 696340×10^3m
• Mass of Earth: $5.972000000 \times 10^{24}$kg	Radius of Earth: 6371×10^3m
• Mass of Moon: $7.345828157 \times 10^{22}$kg	Radius of Moon: 1737×10^3m

- Gravitational constant: $G = 6.67348 \times 10^{-11}\mathrm{m}^3/\mathrm{sec\,kg}$

Note: the ratios of the masses of the Sun, Moon and Earth are known much more accurately than the masses themselves. The masses are known to about five significant digits, whereas the ratios are known to about ten significant digits. If the only forces are gravitational, it is only the mass ratios that matter. Indeed, we could choose units of mass so that the Earth has mass one. The preceding values of the masses are in the correct ratio to about ten significant digits.

(b) **Initial conditions at 00:00:00, 21 June 2010,**

- Earth position: $(-0.012083728, -1.394770664, -0.604680716) \times 10^{11}\mathrm{m}$
- Moon position: $(-0.015537064, -1.395982236, -0.605576290) \times 10^{11}\mathrm{m}$
- Earth velocity: $(\ \ 2.930141099, -0.032094528, -0.013869403) \times 10^{4}\mathrm{m/s}$
- Moon velocity: $(\ \ 2.967343467, -0.121872473, -0.051163801) \times 10^{4}\mathrm{m/s}$.

The coordinate system is chosen so that the Earth's equator lies in the xy-plane. Note that the plane of the Earth's orbit is tilted with respect to the equator; this is why we have seasons.

Note: In principle, at a given day and time, the positions of the Earth and Moon relative to the sun may be found by observation. There are now online calculators which provide this information, for example `https://clearskytonight.com`. The velocities of the Earth and Moon can be determined from their positions at two different times.

(c) **Criteria for an eclipse.** Let $\mathbf{s}$ be a point on the surface of the sun facing the Earth, and let $\mathbf{e}$ be a point on the surface of the Earth. Think of $\mathbf{s}$ as a source of light. If the line from $\mathbf{s}$ to $\mathbf{e}$ intersects the Moon, then $\mathbf{s}$ cannot be seen from $\mathbf{e}$. The set of lines from $\mathbf{s}$ that are blocked by the Moon form a cone $C_{\mathbf{s}}$. The intersection of the cones $C_{\mathbf{s}}$ form the black cone shown in the figure on the next page, called the umbral shadow, or umbra, of the Moon. At a point inside the umbra, every point on the Sun, ie the entire Sun, is blocked by the Moon. The union of the cones $C_{\mathbf{s}}$ form the gray cone shown in the figure on the next page, called the penumbral shadow, or penumbra, of the Moon. At a point inside the penumbra, some points of the Sun, ie part of the Sun, is blocked by the Moon.

During each orbit of the Moon about the Earth, there is an instant when the angle between the Sun-to-Earth and Sun-to-Moon directions is at a minimum. This is a candidate moment for the central time of a solar eclipse (the central time is midway through the eclipse). The test is whether the Earth intersects the umbra (total eclipse) or just the penumbra (annular eclipse or partial eclipse).

Predict the occurrence of solar eclipses since 2010, say for up to 15 years (so you'll capture the solar eclipse this year in North America on 8 April). You might concentrate on total eclipses, or total eclipses and annular eclipses, or total, annular and partial eclipses. See if you can predict the central time of the eclipse to within hours.

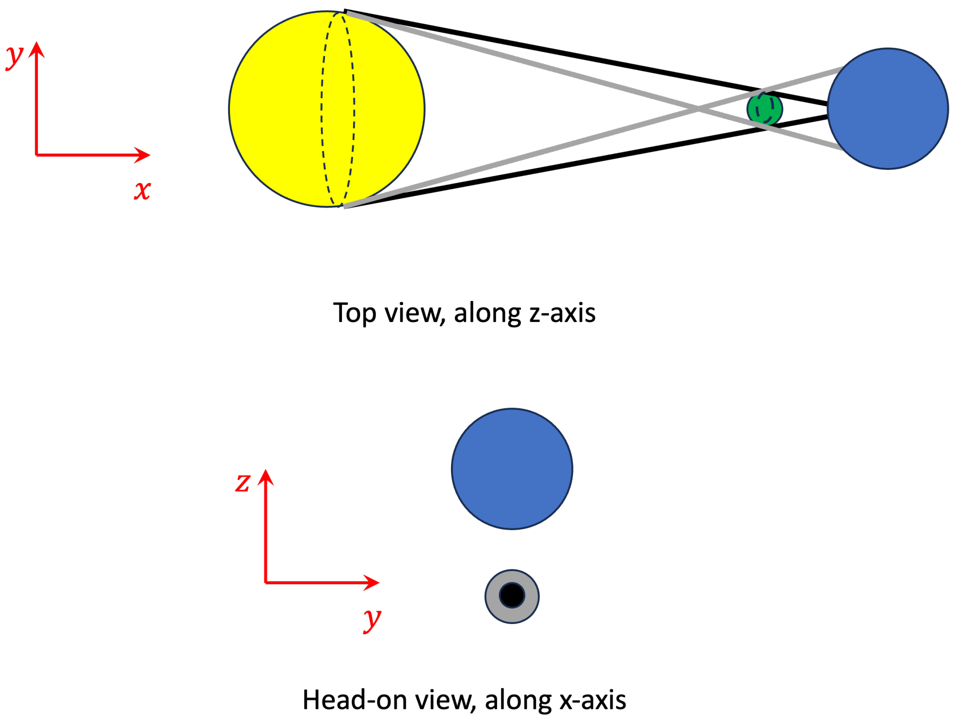

Figure 1: Geometry of a solar eclipse. For simplicity, assume that the Earth's orbit lies in the xy-plane (this is a different coordinate frame than the one used for the initial conditions). The figures depict an instant when the Sun, Earth and Moon are in closest alignment, with the direction of alignment along the x-axis; this is the central time of an eclipse, should one occur. The first panel shows the view from above, along the z-axis. The umbra is the black cone, whose edges are formed by joining points from the equator of the Sun in the yz-plane to corresponding points on the equator of the Moon in the yz-plane. The penumbra is the grey cone, whose edges are formed by joining points from the equator of the Sun in the yz-plane to opposite points on the equator of the Moon in the yz-plane. The lower panel shows the head-on view along the x-axis. The umbra is the black circle, and the penumbra is the grey circle. In this case, there is no solar eclipse.

2 Hybrid Cryptography

Hybrid Cryptography is a fundamental tool for the security of the internet. For example every time a browser shows a padlock icon, a hybrid encryption system is at work. A Hybrid system uses public-key cryptography to securely share a secret key and then uses that secret key with a faster symmetric key cryptography algorithm for bulk data transfer. Hybrid systems typically use either the Rivest Shamir Adleman (RSA) or Diffie Hellman (DH) protocols for secure key sharing and the Advanced Encryption Standard (AES) for bulk data transfer. Examples of Hybrid systems based on RSA+AES are TLS (HTTPS) and PGP/GPG. Examples of systems based on DH+AES are TLS (again), SSH and VPN[1].

In the core sections 1-3 of this project you will design a hybrid cryptography system using the RSA protocol for key exchange and the Vigenère cipher[2] for bulk data transfer. In section 4—the last core section—you will improve the security of your system by adding an extra layer of random encoding. You are also expected to complete one (but not more) of the extensions outlined in section 5 (breaking the Vigenère cipher), section 6 (cracking the RSA protocol), and section 7 (implementing RSA+AES and DH+AES).

Note. The text, Jupyter Notebook, and pdf files mentioned below are available in the GitHub repository for this project at `https://github.com/cmh42/hc`.

Alphabet: crucial simplification. In (core) parts 1-4 the idea is to work with the 26 letter alphabet, for example you could use one, or both, of the following string constants[3]:

$$\texttt{string.ascii\_lowercase} := \text{`abcdefghijklmnopqrstuvwxyz'}$$
$$\texttt{string.ascii\_uppercase} := \text{`ABCDEFGHIJKLMNOPQRSTUVWXYZ'}$$

In these parts of the project you should prepare all input messages so that they only contain alphabetic characters (i.e. no space characters, no punctuation etc.). For the preparation of the input messages and the enciphering/deciphering process you can decide to work entirely in lower case or entirely in upper case or (and this is nicer) you can choose to preserve the case of the alphabetic text from the original input message throughout. This allows you to model the whole process in a streamlined and simplified way. (The point to note is that once we introduce bigram encoding this simplified approach can be used as the core enciphering/deciphering process of a system that handles messages that may contain many other charaters, including space characters and punctuation[4].)

Remark. The Vigenère keyword is not assumed to be an actual word. In fact it can be of any length up to the length of the message to be encrypted. Note that we will refer to it as the *Vigenère key* or simply as the *key* if the context is unambiguous.

1. **(core)** Implement functions to encrypt and decrypt messages using the Caesar cipher and the Vigenère cipher. As noted above, your functions only need to handle messages containing alphabetic characters. Your Caesar cipher functions should be able to perform 26 possible shifts (including the trivial shift) and your Vigenère cipher should have 26 possible choices for any character in the key.

   Note. You should test your functions using randomly generated Caesar shifts and randomly generated Vigenère keys. You should organise this so that the reader can perform these tests. For your tests you might, for example, extract the alphabetic content

[1]For the acronyms here: TLS is *Transport Layer Security*, HTTPS is *Hypertext Transfer Protocol Secure*, PGP is *Pretty Good Privacy*, GPG is *Gnu Privacy Guard*, SSH is *Secure Shell* and VPN is *Virtual Private Network*.

[2]The Vigenère cipher is a method of encrypting text by the use of a sequence of shift/Caesar ciphers based on the letters of a key word.

[3]To use these constants you will need to have included the statement: `import string`.

[4]We do not however develop this assertion in this project - but once you have completed section 4 you will see why it is true.

of two or more of the message_*.txt files provided for this project. (See the example in extract_alphabetic_content.ipynb.) You should implement encryption and decryption to and from files. Similar comments apply to the tests that you should apply throughout this project.

2. **(core)** Implement a function to systematically break the Caesar cipher using letter frequency analysis. (Regarding the latter see the example in get_online_texts.ipynb.)

3. **(core)** Write functions that implement the Hybrid System described below with the encryption and decryption of the message carried out using your Vigenère functions from above and that of the key being carried out by the RSA functions from lectures. Note that, since the Vigenère key may be long—for example 200 characters—your system will need to slice it in to one or more parts (so no slicing if only one part) in preparation for integer conversion/encoding and RSA encryption with the resulting ciphertext integers being transmitted as a tuple[5].

   Hybrid System. Alice generates her private and public key. Bob generates a Vigenère key and Vigenère encrypts/enciphers his message with this key. Then, after slicing it into parts (if necessary) he encodes and RSA encrypts his Vigenère key using Alice's public key and finally sends *both* the resulting tuple of ciphertext integers *and* his Vigenère encrypted message to Alice. Alice uses her private key to RSA decrypt the tuple of ciphertext integers. She then converts/decodes the resulting integers to strings and so reconstructs the Vigenère key. She uses this to Vigenère decrypt/decipher Bob's message.

4. **(core)** There are $26 \cdot 25 = 650$ many 2-grams[6] made up of distinct letters. Redesign your system by performing a random encoding of each letter of the alphabet in to one or more of such 2-grams. Do this in such a way that the frequency of occurrence of each letter is disguised. For example—assuming that the frequency of the letters 'b' and 'e' are 1.5% and 12.7% respectively and that (in this part) you have decided to make use of 400 (of the 650) 2-grams—you can disguise the frequency of 'b' using a randomly choosen set of 2-grams of size 6 ($= 0.015 \times 400$) to represent different instances of this letter. On the other hand to disguise the frequency of 'e' you can use a randomly chosen set of 2-grams of size 51 ($\approx .127 \times 400$) to represent different instances of this letter[7]. Your message is now randomly encoded before Vigenère encryption and decoded after Vigenère decryption. Your encoding information should be recorded in a key which is then appended to the Vigenère key. As before, the resulting string should be sliced into parts for integer conversion and RSA encryption before transmission as a tuple of ciphertext integers. After transmission and RSA decryption of both keys the receiver will be able to obtain the original message. Discuss and compare the security of this hybrid system and that of part 3, taking the implications of part 5 below into account.

5. **(extension)** Implement a function to systematically break the Vigenère cipher. To do this you will need to first perform a *Kasiski* style analysis of the positions of repeated n-grams in the encrypted message to work out the length of the key. You will then reapply the letter frequency analysis that you developed in part 2 to establish the letters of the key.

6. **(extension)** In our presentation of the *RSA* protocol (in the 2023 lectures on the GitHub page) we use 512-bit (i.e. 154 digits in decimal) primes p and q. The security of the protocol

[5]For example, the RSA protocol with 512 bit primes can safely handle strings of length $\lfloor(512-1)/8\rfloor = 63$ (where we note that each character is encoded with 8 bits and a 1 is added to the front of the ciphertext integer). Thus if the Vigenère key is of length 200 we will want to slice it into 4 parts, convert/encode these 4 strings into integers for RSA encryption, and transmit the resulting 4 ciphertext integers as a 4-tuple.

[6]An n-gram is a string of n letters. The term n-gram is often used to denote a contiguous sequence (i.e. a substring) of n letters within a longer string. For example 'ing' is a 3-gram in 'Flying high'.

[7]Once the set of 2-gram encodings of e.g. 'e' has been chosen, the choice of which 2 gram to use for which instance of 'e' is made randomly during the encoding process and does not need to be recorded.

relies on the fact that it is VERY HARD to recover p and q—i.e. to factorise N—from $N = p \cdot q$ if you only know N.

- To see that this is indeed the case, and also to see what happens when we allow p and q to be smaller, you should begin by testing the performance of the `smallest_factor` function[8] from lectures. To do this generate primes p, q and input $N = p \cdot q$ to the `smallest_factor` function. Starting with $l = 16$ bit primes write an algorithm that shows the average computation time on input $N = p \cdot q$ for k-bit primes p, q for $k = l, l + 1, l + 2, \dots$. Continue this analysis for as long as the outcome is a matter of minutes—e.g. up to 15 minutes.
- Using the function `smallest_factor` is clearly not an efficient way of factorising large integers. A better way of doing this is via the *Pollard rho* method. Write a function `pollard_rho` that implements the Pollard rho method using the outline given in the file `pollard_rho.pdf`. Carry out the analysis that you carried out on `smallest_factor` on your function.

Plot your results for both `smallest_factor` and `pollard_rho` showing the expected outcomes (extrapolated from your results) on longer bit lengths. Hence conjecture at what bit length the use of each function becomes unfeasible.

7. **(extension)** Implement the following hybrid systems.

   - RSA for secure key exchange with AES for bulk data transfer.
   - DH for secure key exchange with AES for bulk data transfer.

   Expected approach. You should use the RSA functions/tools from section 3 and design the necessary tools for DH in a similar way[9]. For the AES algorithm you may use a specialised python library such as `pycryptodome` and code sourced from the internet. You should provide concise examples of the implementation of each hybrid system. For message data you can use text (e.g. from a `message_*.txt` file in the GitHub repository) with no restriction on the type of character present in the text. You should also provide a brief overview of the AES algorithm.

Note on the extensions. In this project you are expected to develop one extension. Doing more will not achieve more marks: the point is to concentrate on the quality of the ideas and the design of your code in both the core sections and the extension that you choose.

[8]Note the `decompose` function (which uses `smallest_factor` as a subfunction) is not required here since you are working under the assumption that $N = p \cdot q$ with p and q prime. Thus once you have found one factor, which must be either either p or q, and assuming for the sake of argument that it is p, then you can extract the other factor q directly by dividing N by p.

[9]Check out the GitHub repository `https://github.com/cmh42/hc` for resources on Diffie Hellman: a Jupyter Notebook on Diffie Hellman should be available there by 20th October 2025.

3 Aliquot sequences

For a positive integer n the sum of proper divisors function is

$$s(n) = \sum_{\{d|n, 0<d<n\}} d.$$

It gives the sum of all positive divisors of n, excluding n. Interest in this function goes back to the Pythagoreans (6th century B.C.E.). Aliquot sequences are the seqeunces formed by repeatedly applying this function.

Definition. *For each $n \in \mathbb{N}$ the sequence*

$$A_n = \{n, s(n), s^2(n), s^3(n), \dots\}$$

generated by applying the function s repeatedly is the aliquot sequence starting at n. If $s^j(n) = 0$ for some j, the sequence terminates after j.

There are some interesting seqeunces. For example, $s(220) = 284$ and $s(284) = 220$. This means $\{220, 284, 220, 284, 220, \dots\}$ is an infinitely looping aliquot sequence. These are called 'amicable' numbers. Other examples arise from 'perfect' numbers such as 6 which has the property that $s(6) = 6$. There are also sequences which terminate at 0, such as $\{7, 1, 0\}$. The same should happen starting at any prime.

Surprisingly little is known about aliquot sequences. It is easy to see that there are three possible types of aliquot sequences:

1. Those which terminate at zero.
2. Those which enter a loop.
3. Those which continue infinitely but do not contain repeats.

It is not currently known whether there are any of type 3 but equally it is possible that most aliquot sequences are of this type. The goal of this project is explore these sequences computationally.

One of the difficulties is that calculating $s(n)$ becomes computationally difficult once n is large because it involves factoring n into primes. Your project should consist of two main parts:

- Write code to compute some aliquot sequences.
- Use this code to explore questions about these sequences.

The efficiency of your code will determine how effectively you can investigate the sequences.

1. **(core)** Write a function to calculate $s(n)$. (Note: In week 8.2 (Appendix) there is a method that exploits the fact that the sum of **all** divisors is a multiplicative function.)
2. **(core)** For a given n, compute the aliquot sequence starting at n (up to a sensible point).

   Practical suggestion: Write your code so that it computes at most the first k terms of the sequence. Also, write it so that for some i, your code stops computing new terms once $s^j(n) > i$. At first, you can use lower values but you should aim to get code that runs in a reasonable time for $k = 30$ and $i = 10^9$.
3. **(core)** Find a way to detect loops. (There's a little bit to this – look at the sequence starting at 562 to see why.)

4. **(core)** For each $n < 20000$ try to classify it according to the end state of the aliquot sequence starting at n. Although there are three theoretical types, classify sequences into the *four* cases where it either terminates at zero or enters a loop or the calculation is cut short because you reached term k or the sequence exceeded i (where k and i are as in part 2. Present this classification by counting the number of sequences of this type, using the parameters $i = 10^9$ and $k = 30$.

5. **(core)** The classification in part 4 will depend on the choices of the parameters k, i and the largest starting number of $20,000$. There will be a better classification if these parameters are higher at the cost of the code taking longer to run. Investigate how much you can improve the classification while your code still runs in a reasonable amount of time. (The meaning of 'reasonable' depends on how patient you are, but we recommend about 30 seconds.)

6. **(extension)** Use your classification to investigate the following questions. How long can the loop be if an aliquot sequence ends in a loop? If a sequence terminates at zero, how long can it be (relative to the starting n)?

7. **(extension)** Find a way of plotting to visualise aliquot sequences which enter loops. Try to find an appealing way of demonstrating the different types of behavior sequences can have.

8. **(extension)** A related concept to perfect numbers are 'abundant' numbers with $s(n) > n$ and 'deficient' numbers, where $s(n) < n$. A looping aliquot sequence should contain some of each. Compute the number of each up for all integers to a fixed value, and for the sets of numbers within aliquot sequences. Do these patterns suggest anything about aliquot sequences?

9. **(extension)** Investigate whether you can make your code faster using ideas from dynamic programming. For example, consider storing values of the sum of divisors function, or precomputing factorisations for small integers (e.g. for all integers up to 10^6) or investigate using the sieve of Eratosthenes to factor many integers at once. Evaluate the effectiveness of your changes by checking if they improve the results of earlier parts of the project, such as part 4. (Since this may be more experimental, it is valid to discuss attempts even if they don't end up speeding the code up.)

Note on the extensions. In this project, you should complete the core sections and choose to develop one or two extensions. (Doing more will not achieve more marks: the point is to concentrate on the quality of the ideas and design of the extension(s) that you choose.)

4 Knot Theory and Recursion

Continued fractions

Let $\frac{p}{q}$ be a rational number, where p and q are coprime. A continued fraction for $\frac{p}{q}$ is a sequence $[a_1, a_2, \dots a_n]$ where each $a_i \in \mathbb{Z}$ so that

$$\frac{p}{q} = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \frac{1}{a_n}}}}.$$

You might also see continued fractions for any real number. Rational numbers have finite continued fraction expansions. Continued fractions can be calculated recursively using the quotients you get in Euclid's algorithm. They are not unique, for example

$$[1,3,2] = 1 + \cfrac{1}{3 + \cfrac{1}{2}} = \frac{9}{7} = 2 + \cfrac{1}{-2 + \cfrac{1}{2 + \cfrac{1}{-3}}} = [2,-2,2,-3].$$

To do:

1. (**core**) Write at least two functions which calculate a continued fraction for a given $\frac{p}{q}$ (to generate different expansions).
2. (**core**) Write a function that recalculates the rational number from a list $[a_1, \dots a_n]$.

2-bridge links

A *knot or link* is a embedded loop (for a knot) or several loops (for a link) in 3-dimensional space. Two knots are equivalent if one can be deformed into the other without passing through itself. It is usually best to visualise them using knot diagrams. The diagrams in Figure 2 are called 2-bridge because the pictures each have two local maxima.

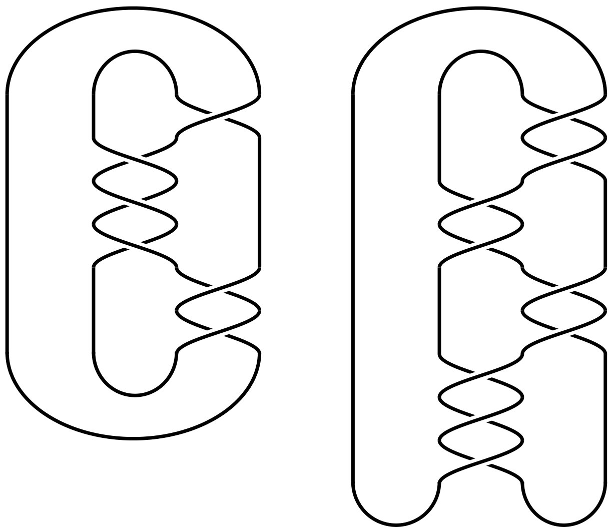

Figure 2: The 2-bridge diagrams corresponding to $[1,3,2]$ and $[2,-2,2,-3]$. They are different pictures of the same knot.

For every rational number $\frac{p}{q}$ (with p and q coprime) there is an associated 2-bridge link $S(p,q)$. Here's the algorithm for drawing it: Start with four parallel strands. Then take a continued fraction expansion for $\frac{p}{q}$. For each a_i add twists between two strands with a_i crossings – use the rightmost two strands if i is odd and the middle two if i is even. The crossings should be right-handed (i.e. right strand over left as you go down) if i is odd and a_i is positive and should change orientation if you switch either of these. Then connect all these twist regions together and connect the top and bottom of the diagram. The pattern for this depends on if you had an odd or even number of twist regions (but the key is that it shouldn't immediately trivialise the last one). (See Figure 2, [1, Figure 3.9 on p59] or [2, Section 1]. The sign choice varies depending on where you look but the important thing is to pick one and stick with it.)

To do:

3. (**core**) Write code that prints a 2-bridge link diagram for a given $\frac{p}{q}$. It is fine for the diagram to be quite ugly, e.g. built from pieces similar to

```
if crossing_type == 1:
    print("| |  \ / ")
    print("| |   /  ")
    print("| |  / \ ").
```

4. (**core**) Use this function help you check these facts about 2-bridge links for three or four cases:

   (a) $S(p,q)$ is a knot if p is odd and has 2 components if p is even;

   (b) the link $S(p,q)$ does not depend on the choice of continued fraction, even though the diagram does;

   (c) if $n \in \mathbb{Z}$ then $S(p,q) = S(p,q+np)$ (i.e. we only need $q \mod p$);

   (d) changing the sign of all the crossings in $S(p,q)$ gives you $S(p,-q) = S(p,p-q)$.

d-invariant

The d invariant of $S(p,q)$ is a collection of p rational numbers[10]. These can be denoted by $d(S(p,q),i)$ and when $p > 0$ they are defined recursively by the rules[3, Proposition 4.8][11]:

- For any q and i, $d(S(1,q),i) = 0$;
- If $a\%p$ denotes the integer between 0 and p which is congruent to $a \mod p$ then $d(S(p,q),i) = d(S(p,q\%p),i\%p)$;
- When $p > q > 0$ and $0 \le i < p+q$,

$$d(S(p,q),i) = \frac{(2i+1-p-q)^2 - pq}{4pq} - d(S(q,p),i).$$

Since these are rational, you'll need to need to be careful about rounding errors. For example

$$d(S(9,7)) = [0, \frac{-2}{9}, \frac{4}{9}, 0, \frac{4}{9}, \frac{-2}{9}, 0, \frac{-8}{9}, \frac{-8}{9}].$$

To do:

[10] Technically there is a missing step. First we should use the link to determine a 3-dimensional manifold $L(p,q)$ and then d is an invariant of that manifold.

[11] The orientation convention in that paper is different, hence the different sign.

5. (**core**) Write a function that calculates $d(S(p,q),i)$.
6. (**core**) Write a function that computes the list of d invariants for $S(p,q)$.

   Extensions examine these questions about the d invariant:

7. (**extension**) For examples where you know $S(p,q_1) = S(p,q_2)$ (from part 4) determine how this is reflected in the d-invariant. To find more relations, use part 4 to investigate the relation between the links $S(p,q)$ and $S(p,q')$ when $qq' \equiv 1 \mod p$. (Note: It may be useful to compare the continued fractions for p/q and p/q'.)
8. (**extension**) Does it completely distinguish between 2-bridge links. In other words, is it true that whenever $S(p,q_1)$ and $S(p,q_2)$ are inequivalent links they have differing d invariants? Check this for a collection of example, such as when $p < 100$. If possible, try to refine your code to work for all $p < 200$ or $p < 1000$.
9. (**extension**) In the case where $p = m^2$ for an odd number m, it is interesting to look at which knots $S(m^2,q)$ have the property that m of the d-invariants are zero. (This is related whether the knot is what is called a slice knot – this is something you can detect from a diagram but it's hard to computationally.) For $m < 105$ determine which of the knots $S(m^2,q)$ have this property and compare the numbers $\frac{m^2}{q}$ to the set $\mathcal{R}$ in [2, Definition 1.1].

Note on the extensions. In this project, you should complete the core sections and choose to develop one or two extensions. (Doing more will not achieve more marks: the point is to concentrate on the quality of the ideas and design of the extension(s) that you choose.)

References

[1] P. Cromwell, *Knots and Links*, Cambridge University Press (2004)

[2] P. Lisca, *Lens spaces, rational balls and the ribbon conjecture*, Geom. Topol. **11** (2007) 429–472. (Also at https://arxiv.org/abs/math/0701610.)

[3] P. Oszváth and Z. Szabó, *Absolutely graded Floer homologies and intersection forms for four-manifolds with boundary*, Adv. Math. **173** (2003), 179–261. (Also at http://arxiv.org/abs/math/0110170v2.)

5 Percolation

Do an image web search for "critical percolation" for some inspiration.

Consider the square grid $\{0,\dots,n-1\}^2 = \big\{(i,j) : i,j \in \{0,\dots,n-1\}\big\}$. We call its elements **sites**. We declare each site to be **yellow** with probability p, or else **blue** with probability $1-p$, where p is a parameter between 0 and 1, and different sites receive independent colours. Two sites are considered **adjacent** if they are at Euclidean distance exactly 1, so each site is adjacent to at most 4 others (but fewer if it is at a side or corner of the grid). A **path** means a finite sequence of sites in which each consecutive pair is adjacent. We are interested in the probability

$$F_n(p) = \mathbb{P}\Big(\text{there is a yellow path connecting the left and right sides of the grid}\Big).$$

1. (**core**) Produce a visualization of the randomly coloured grid for small values of n and several choices of p, and sample it several times. This could be as simple as a text representation of a matrix, but we recommend using a 'pcolor' plot as for the Mandelbrot set tutorial, or something similar. Get a rough idea of how the probability $F_n(p)$ behaves by inspecting the visualization manually.

2. (**core**) Write a function to determine whether there is a yellow path connecting the left and right sides.

   Here is a suggested approach. Iteratively determine the set of sites that are reachable by yellow paths starting from the left side, as follows. Initially all yellow sites on the left side are known to be reachable. For every known reachable site x and every adjacent yellow site y not currently known to be reachable, we can declare y reachable. Keep doing this until no new reachable sites can be added. Then check whether any site on the right edge is reachable. Test your code by comparing it with the visualization. It might be helpful to also label the reachable sites in the visualization. Do not worry about efficiency for now.

   Hence estimate $F_n(p)$ for some small n, perhaps 5 or 10, and some choices of p.

3. (**core**) We now want to investigate larger n. Make your function in (b) more efficient, so that it runs as quickly as possible. Here are some suggestions. Ensure that each site is not examined more times than necessary. One approach is to maintain a list of reachable sites and iterate over it in order, examining the neighbours, and adding any new reachable sites discovered to the end of the list. To check whether a site is already known to be reachable, using "`if site in list`" is not efficient because it iterates over the whole list. Instead, maintain a separate n-by-n array (or a python `set`). The search can be stopped as soon as we reach any site on the right side of the grid.

4. (**core**) The function has a limit: $F(p) = \lim_{n\to\infty} F_n(p)$ which satisfies

   $$F(p) = \begin{cases} 0, & p < p_c \\ 1, & p > p_c, \end{cases}$$

   where p_c is called the **critical point**. Plot graphs of $F_n(p)$, and use them to estimate p_c as precisely as you can. Even with the optimizations discussed above, expect to run your code for a few hours at least.

Possible directions for extensions:

A good project might involve work on extensions roughly equal to that needed to thoroughly address the core. Quality is more important than quantity. An in-depth exploration of a few directions is typically preferable to a superficial treatment of large number.

5. Do the same for rectangular grids of different shapes, e.g. $2n$-by-n or $3n$-by-$2n$. The critical point should be the same.

6. Investigate similarly the probability

$$G_n(p) = \mathbb{P}\Big(\text{there is a yellow path connecting the centre of the grid to the boundary}\Big)$$

(where, for simplicity, n can be assumed to be odd) and its limit $G(p) = \lim_{n\to\infty} G_n(p)$. Try to understand what the graph of G looks like.

7. Do the same for the **triangular lattice**, which consists of sites at the corners of equilateral triangles, with 6 triangles meeting at each site. This can be conveniently implemented as a square grid, but with diagonal pairs of sites of the form x and $x + (1,1)$ now in addition considered adjacent. The critical point of site percolation on the triangular lattice has a very simple exact form. Use your estimates to try and guess it. (You can look it up to check).

8. For the triangular lattice, G has an asymptotic power law behaviour near p_c:

$$G(p_c + \epsilon) \approx \text{const } \epsilon^{\beta} \quad \text{as } \epsilon \downarrow 0.$$

Estimate the power β. You will need to use the exact value of p_c, and even then it is tricky to get a good estimate. You may want to look up the expected answer.

9. Implement a different method for detecting crossings, in which we explore the boundary between reachable and non-reachable sites by "wall following". How many sites typically need to be explored? How does the answer depend on n and p?

10. Implement the following more sophisticated method, which allows us to effectively sample all values of p simultaneously. Each site is assigned an independent Uniform$[0,1]$ random variable. The model with parameter p is then defined by declaring all those sites with label less than p yellow. We can compute the minimum p for which a yellow path connecting the regions of interest exists, by a modification of the iterative scheme used before.

11. (**advanced**) For the triangular lattice, take the parameter to be exactly $p = p_c$, take a region in the shape of an equilateral triangle of side length n (now with respect to the true geometry of the lattice, not the square grid implementation above). Investigate the probability $T_n(r)$ that the base of the triangle is connected by a yellow path to the segment of the right side that extends down distance rn from the top vertex. In the limit $n \to \infty$ there is a very simple formula for this. Moreover, it should be the same if the equilateral triangle is rotated by an arbitrary angle.

12. (**advanced**) Investigate the following percolation model. We have infinitely many equally spaced parallel wires. Each wire has breaks according to a Poisson process. Each adjacent pair of wires has bridges connecting them according to a Poisson process. The critical point is when the rates of the Poisson processes are equal. At the critical point it should have the same asymptotic behaviour for crossing probabilities of equilateral triangles as above, although this has not been proved rigorously. It is also necessary to choose the correct spacing of wires as a function of the rate of the Poisson process for this to work out correctly.

6 Code Breaking with Statistical Physics

In the appendix there are several secret messages for you to decrypt, in roughly increasing order of difficulty. Each message is written using the 26 letters of the alphabet together with spaces. (There are no other symbols such as punctuation). Each message is encrypted using a simple substitution cipher; that is, a certain (secret) permutation of the 27 characters has been applied to each character of the message in turn. Your job is to crack the code and discover the message. Ideally your final answers you should include your best guess at the correctly punctuated and cased message, as well as the raw decrypted version. Of course, the last part will require human involvement.

The provided messages are there to motivate and focus you, but are not necessarily the final destination. The broader goal is to investigate decryption methods.

For your convenience, a notebook containing the messages as strings will be provided on the Blackboard page.

1. (**core**) In the first message, the cipher is simply a cyclic shift of the 27 characters. E.g. a shift of 2 would map $A \mapsto C$, $B \mapsto D$, $\ldots$, $Y \mapsto$ space, $Z \mapsto A$, space $\mapsto B$. Find the shift and decode the message. One simple approach is to try all possibilities and check by hand which one produces readable text.

2. (**core**) The other messages each use an arbitrary permutation of the characters, so trying all the possibilities is no longer practical. Instead we will make use of statistical properties of text. In preparation for this, find a large body of English text (the larger the better) to download and analyse. Replace all lower case letters with upper case, and find a way to get rid of extra characters. One simple approach is to replace every character that is not a capital letter or a space with a space, and then replace multiple consecutive spaces with a single space. More nuanced schemes are possible - for instance, apostrophes and hyphens could be treated differently.

   For example, the following code will download the text of the novel Moby Dick and store it in a single very long string.

   ```
   import requests
   url='http://www.gutenberg.org/files/2701/2701-0.txt'
   moby=requests.get(url).text
   ```

   To avoid repeatedly spamming someone's server, it is best to save the text to your own machine and access it from there:

   ```
   with open('C:/data/moby.txt','w',errors='ignore') as f:
       f.write(moby)   # write to file


   with open('C:/data/moby.txt','r',errors='ignore') as f:
       moby=f.read()   # read from file
   ```

3. (**core**) One simple approach is to use character frequencies. In the sample English text, count the frequency with which each of the 27 characters appears, and sort them by frequency. Do the same for the coded message, and try the cipher in which the kth most common character in the message maps to the kth most common letter in English, for each k.

4. (**core**) The above may not be sufficient to get the full correct answer, but for the longer messages it should be close. Write a program to make it easy for you to try swapping the roles of any two characters (of your choice) and immediately see the effect of the decoded message. Then, starting from the result above, adjust the cipher by hand until is readable.

5. (**core**) The above method is less likely to work for the later messages, because they are not long enough to get accurate frequency counts (and some might involve unusual language). Instead use the following approach based on ideas from statistical physics. In the sample text, record the frequency (i.e. number of occurrences) of each of the 27^2 bigrams, i.e. pairs of consecutive characters such as SH, as well as the 27 characters. For each pair of characters i, j compute
$$p(i,j) := \frac{\text{frequency}(ij)+1}{\text{frequency}(i)}.$$
This represents the probability that i is followed by j; the +1 is a 'fudge factor' to avoid zeros. To any potential decoded message $m = i_1 i_2 \dots i_n$ we assign a score
$$S(m) = \sum_{j=1}^{n-1} \log p(i_j, i_{j+1}),$$
which measures how plausible m is as a piece of English text (it represents the logarithm of the probability of seeing m under a certain "Markov model"). Write a function that computes $S(m)$ for any message m.

6. (**core**) Implement this method, called the Metropolis algorithm (an example of a Monte Carlo method). Start with any guess m at the decrypted message, which might simply be the encrypted message itself or the result of applying a randomly chosen cipher. Now repeatedly do the following.

   Choose two characters at random and swap their roles in the cipher, to get a new candidate message m'. If $S(m') > S(m)$ then replace m with m'. If $S(m') \leq S(m)$ then replace m with m' with probability
$$\exp \frac{S(m') - S(m)}{T},$$
   otherwise just keep m.

   Repeat this for many steps (e.g. 100000 or 1000000).

   Here $T > 0$ is a parameter, which represents temperature in a physics analogy, and indicates how likely the algorithm is to accept steps that make the score worse (in hopes of finding a route making it better later). You may need to experiment to find the value of T that works best. Start by trying $T = 1$. One approach (called simulated annealing) is to start with T large (say 10), and gradually reduce it as the algorithm proceeds, ending at something small (say 0.1), perhaps by reducing it by a constant ratio at each step.

   It is useful to print the current guess m every so often (e.g. every 10000 steps), and monitor its progress.

   It may be useful to try running the algorithm multiple times, from different starting points, and to adjust the parameters, and to make manual adjustments to the cipher at the end (as before).

Possible directions for extensions:

You should focus on one or possibly two extension directions. A good project will typically involve work on extensions roughly equal to that needed to thoroughly address the core.

Quality is more important than quantity. An in-depth exploration of a few directions is typically preferable to a superficial treatment of large number.

Other directions besides those suggested are possible. If you have ideas, it is advisable to discuss them with your supervisor.

For the extensions below, besides the messages provided, you will likely find it helpful to produce your own coded messages.

7. How can the method be improved? The final few messages will require thinking outside the box. Does it help to consider trigrams, or just common ones, or whole words? What if the message is in a unknown language? Can the language be detected automatically? Can the method cope with small errors (typos) in the message, or even correct them?

8. For the simple frequency analysis approach in steps 3-4, how much adjustment is typically needed, and how can that be quantified systematically? Can you represent the required permutation graphically? How does the answer depend on the length of the message or other factors? Are some letters more problematic than others? Can you derive a probabilistic model of what is going on?

9. Can you analyse the progress of the Metropolis algorithm over time, perhaps by plotting graphs of some quantities? How do the results depend on the length, or other factors? Can this help with adjusting T, and choosing an appropriate number of steps to run for?

10. Can the Metropolis method be adapted to other ciphers such as transposition ciphers, the Vigenere cipher, multiple-substitution ciphers (perhaps using a key that is itself a piece of English text), or a one-time pad that itself consists of English text? You would need to look up what these are (e.g. on Wikipedia), and write your own programs for encrypting as well as decrypting, so that you have ciphertexts to work with.

Coded messages

1. GPKZFER JRSER EKWIGIWKWVRZ YZRCWMWCRSEVRYWEWISCRGLIGFJWRGIFYISDD
   EYRCSEYLSYWRGPKZFEJRVWJ YERGZ CFJFGZPRWDGZSJ QWJRUFVWRIWSVST
   C KPRN KZR KJREFKSTCWRLJWRFXRJ YE X USEKR EVWEKSK FER
   KJRCSEYLSYWRUFEJKILUKJRSEVRFTAWUKRFI WEKWVRSGGIFSUZRS
   DRKFRZWCGRGIFYISDDWIJRNI KWRUCWSIRCFY
   USCRUFVWRXFIRJDSCCRSEVRCSIYWRJUSCWRGIFAWUKJ

2. RECFV KUWE VJCRWQFCRCFICYWEZRWKLVJWQF SCRRZ ALVWJLAACFWNATZVW
   NFWIZRZT FWELYWVCSTWNRWLVTE NBEWZTWHLRWCLRMWS FWJCWHE WUACHWEZJWR
   WHCVVWT WRCCWTELTWECWHLRWQF S NAYVMWCPKZTCYWTECWJ JCATWTELTWEZVT
   AWKNGZTTWRWGF LYWGLKUWELYWYZRLQQCLFCYWTEF NBEWTECWY FWJMWK
   JFLYCWFNRECYWT WTECWTLGVCWVLZYW NTWLVVWTECWRVZQRW SWQLQCFWK
   ATLZAZABWYLAKZABWJCAWZAWSF ATW SWEZJWLAYWTEFCHWEZJRCVSWZAT
   WLAWZATFZKLTCWLAYWCVLG FLTCWKLVKNVLTZ AWS FWTH WE NFRWZWHLTKECYWEZJWLRWECWK
   ICFCYWRECCTWLSTCFWRECCTW SWQLQCFWHZTEWSZBNFCRWLAYWVCTTCFRWR WK JQVCTCVMWLGR
   FGCYWZAWEZRWTLRUWTELTWECWELYWCIZYCATVMWS FB TTCAWJMWQFCRCAKCWR
   JCTZJCRWECWHLRWJLUZABWQF BFCRRWLAYWHEZRTVCYWLAYWRLABWLTWEZRWH FUWR
   JCTZJCRWECWHLRWQNDDVCYWLAYWH NVYWRZTWS FWV ABWRQCVVRWHZTEWLWSNFF
   HCYWGF HWLAYWLWILKLATWCMCWSZALVVMWECWRQFLABWSF JWEZRWKELZFWHZTEWLWKFMW
   SWRLTZRSLKTZ AWLAYWHLVUCYWNQWLAYWY HAWTECWF JWFNGGZABWEZRWELAYRWT
   BCTECFWTECAWECWHF TCWLWV ABWTCVCBFLJWNQ AWLWKLGVCWS FJWZSWJMWLARHCFWT
   WTEZRWZRWLRWZWE QCWM NWHZVVWELICWLWICFMWQFCTTMWKLRCWT WLYYWT WM
   NFWK VVCKTZ AWHLTR AWRLZYWECWZWCPQCKTWTELTWHCWRELVVWGCWLGVCWT
   WB WY HAWT WA FS VUWT J FF HWLAYWT WTLUCW NFWSFZCAYWR
   JCWICFMWYCSZAZTCWACHRWLRWT WTECWRCKFCTW SWEZRWLAA MLAKCWZWK
   ASCRRWTELTWZWHLRWSZVVCYWHZTEWKNFZ RZTMWGNTWZWHLRWLHLFCWTELTWE
   VJCRWVZUCYWT WJLUCWEZRWYZRKV RNFCRWLTWEZRW HAWTZJCWLAYWZAWEZRW
   HAWHLMWR WZWHLZTCYWNATZVWZTWRE NVYWRNZTWEZJWT WTLUCWJCWZAT WEZRWK
   ASZYCAKCWGNTWTECFCWHLRWLWYCVLMWZAWTELTWLARHCFZABWTCVCBFLJWLAYWTH WYLMRW
   SWZJQLTZCAKCWS VV HCYWYNFZABWHEZKEWE VJCRWQFZKUCYWNQWEZRWCLFRWLTWCICFMWFZABW
   SWTECWGCVVW AWTECWCICAZABW SWTECWRCK AYWTECFCWKLJCWLWVCTTCFWSF
   JWEZVT AWKNGZTTWLVVWHLRWXNZCTWHZTEWEZJWRLICWTELTWLWV ABWZARKFZQTZ
   AWELYWLQQCLFCYWTELTWJ FAZABWNQ AWTECWQCYCRTLVW SWTECWRNAYZLVWECWZAKV
   RCYWLWK QMW SWZTWHEZKEWZRWECFCWFCQF YNKCYWCVRZCWQFCQLFCWT
   WJCCTWTEMWB YWE VJCRWGCATW ICFWTEZRWBF TCRXNCWSFZCDCWS FWR
   JCWJZANTCRWLAYWTECAWRNYYCAVMWRQFLABWT WEZRWSCCTWHZTEWLAWCPKVLJLTZ AW
   SWRNFQFZRCWLAYWYZRJLMWEZRWSLKCWHLRWELBBLFYWHZTEWLAPZCTM

3. TULOMREMAEKBLRSWCTMWBIHB BTN KKUT NBTULOMRB DWBKSYMBYSWMRDBTULOMRKBJUNNBRMQM
   NBKE EUKEUT NBUDPSRY EUSDB ISCEBEOMBLN UDEMAEB DWBEO
   EBUDPSRY EUSDBT DBSPEMDBIMBCKMWBESBIRM XBEOMBTULOMRB
   PEMRBEOMBWUKTSQMRHBSPBPRMVCMDTHB D NHKUKBIHBEOMB R IBY EOMY EUTU DB
   DWBLSNHY EOB NBXUDWUB NKSBXDSJDB KB NXUDWCKBUDBEOMBDUDEOBTMDECRHBDM
   RNHB NNBKCTOBTULOMRKBTSCNWBIMBIRSXMDBIHB DBUDPSRYMWB EE TXMRBKCTOBTN
   KKUT NBTULOMRKBKEUNNBMDGSHBLSLCN RUEHBESW HBEOSCFOBYSKENHB KBLCZZNMKB
   NBXUDWUBJRSEMB BISSXBSDBTRHLESFR LOHBMDEUENMWBRUK N OBPUBUKEUXOR
   GB NBYC YY BY DCKTRULEBPSRBEOMBWMTULOMRUDFBTRHLESFR LOUTBYMKK
   FMKBJOUTOBWMKTRUIMWBEOMBPURKEBXDSJDBCKMBSPBPRMVCMDTHB D NHKUKB DWBTRHLE
   D NHKUKBEMTODUVCMKB DBUYLSRE DEBTSDERUICEUSDBSPBUIDB WN DBJ KBSDBK
   YLNMBKUZMBPSRBCKMBSPBPRMVCMDTHB D NHKUK

4. RJLHXTNERXWZEWZSXLESNIFWX TEKJAAS NYSVEWZSECYWKZEKJTVYAW
   TWVERSLSELYTTXTNETYPDSLVEDJVH AXVVELJASER VED VXK AAIEWJECJEWZSEK
   AKYA WXJTVEWZSEVYPVEDSLCJRVHXEV XCEVJERZSTEWZSEVWSLTEK PSEULSSEJTEPJTC
   IEPJLTXTNERSEK AKYA WSCEWZ WERSEVZJYACEASWEWJTTSVEJUER WSLED AA VWEXTE
   WEWZSELS LEJUEWZSEOSVVSAEWJEFYVZEWZSEVWSLTECJRTE TCEAXUWEWZSEDJR

5. IZWLXSZF QEFLZTQUVLSZMIHZCIVLUZEDDZIUPZXLYAHLPZCQZLICZECZMIHZC QAO CZ
   QMLWLXZC ICZECHZIBBLCECLZTEO CZKLZHCETADICLPZKSZIZBEULIBBDL

6. RVRRKZHKTVYPJCGVCHGVGXECUKRKZGRSKZKWGVGRSPKAGVCHGVGHZVLJCAYOGRZET
   KRKZGSKWGTOGSKZJGAVPZOGHVCHOGRPQUYPCLGRSKGAVCQOGJAGSPWGYVHOGAZPKCHGRSKGCOT
   SGPCGOKYYJMGQVCGMKGWKKGRSKGTVWRKZGWRZJUKGMSVRGVGNEVKZKGAKYYJM

7. LOJ WUZCWKCTFHHFHTWUZFHQWMZEUW EAWKCWUZCWCHV

8. EYENXRPSURPGUUIRKSUGURSHYY NXRCUUIREPRCHBIRENBRMAOKUGIRKUGURS
   NX NXROREHABRFEAUBONRBGUKROHPRSURBGONURENBRPORSUGRJ JURKEIRI
   NX NXRORPKEIRJ CGOFSRIENXRIPGEPSIJUWIRENBRGUUAIRISURB
   GABRPSUYREMMRMHRFAUEGAWRORKSUNRPSUGURFEYRERWUAARORMOGU
   XNRIVHUUAIRPSEPRBENXRSUGRPEJIEAPUUG URO

9. WDYRDYLDQCSLR KTYDPYZ LXSKTYWDYQ U KTYZOITYGIDYCDYJSILPTYLDUDKOTYCIQYKQTP
   QPYODYPDRZTYCDQPDQCLDYODTYJL GIDRDQPTYSLH
   QKGIDTYCDTYMSKTDLKDTYCSIULKLYODTYADINYZSILYXKNDLYODYB
   ODKCSTJSZDYCDYOSMTJILKPDYCDYHSYPDLYHLYJDYIQDYOIDILYRSRDQP
   QYDYCDYJSQTJKDQJDYODYTSRRDKOYSIYDP KDQPYZOSQHYTYODTYRDIMODTYO
   YJF RMLDYODYPSIPYCSQPYWDYQDP KTYGIIQDYZDPKPDYZ LPKDYDPY
   YOKQTDQTKMKOKPDYCIGIDOYWDYLDPSILQ KTYUKPDYRIQKL

10. GYYUXPUXGTBQTDFU

11. ZVVFEKTFZKKO

12. RIZOGS BXTPK YWH JQNE LUC MADFV

7 Recursive Animations and The Towers of Hanoi

Animations can be a useful tool to visualise and understand solutions to mathematical problems that can be represented graphically. In the core sections 1-3 of this project you will create animations of various recursive problems using the python graphical library `pygame`, in part by integrating its use with other tools that you have learnt during the course. Then in core section 4 you will write code to solve the Towers of Hanoi problem and print out configurations of solutions to the problem. In the final core section 5 you will create `pygame` animations of the Tower of Hanoi problem. You are expected to complete one (but not more) of the extensions outlined in section 6 (Towers of Hanoi variants), section 7 (smooth animation of the Towers of Hanoi) section 8 (game version) and section 9 (Mandelbrot set animation).

Note. The Python, Jupyter Notebook, and pdf files mentioned below are available in the GitHub repository for this project at `https://github.com/cmh42/toh`.

1. **(core)** Animating fractal constructions gives us direct insight into how the function generating the fractal operates. You are given a pygame animation function `draw_sierpinski` (in the `animation_examples.ipynb` file) that draws the Sierpinski triangle. Using this function the user is able to choose the *depth* of the triangle and to stop and start the animation. You should develop this function so that the user is also able to change the speed of the animation. You should also add colours to the triangle drawing.

2. **(core)** Fractals can simulate shapes found in the natural word. One simple example of this is the construction of a recursively defined tree. The tree of depth 1 is just a trunk with three straight branches. Then given the tree T of depth n the tree of depth $n+1$ is the tree T where every branch has been replaced by a tree of depth 1. Figure 3 shows trees of depth 1, 2, 3 and 4. Figure 4 shows snapshots of the animated drawing of a tree of depth 9.

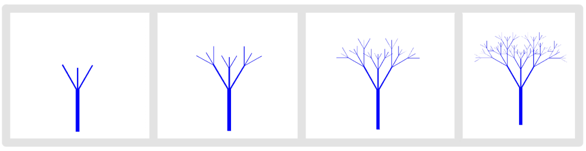

Figure 3: Recursive Trees of depth 1, 2, 3 and 4.

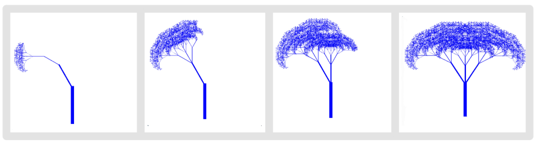

Figure 4: Drawing the Tree of depth 9.

   You should develop a pygame animation function that constructs a similar recursive tree. The user should be able to choose the depth of the tree and should be able to start and stop the animation and control its speed. The tree should be coloured with the trunk and branches being brown and the leaves (i.e. the last level of branches) being green.

3. **(core)** As can be seen in `week4_bonus_material_julia_sets.ipynb` (in the GitHub repository) there are interesting Julia sets with parameter $j_p = 0.7885e^{ai}$ where a is a small non-negative real number. In fact if we allow a to vary in repeated cycles over $[0, 2\pi]$ we are able to create a film like animation of Julia sets with this form of parameter. The images in Figure 5 are snapshots of such Julia sets.

   Your task is to develop a function (using matplotlib.pyplot tools) to generate a number (for example 200) of image files of the Julia sets with parameter $j_p = 0.7885e^{ai}$ where the

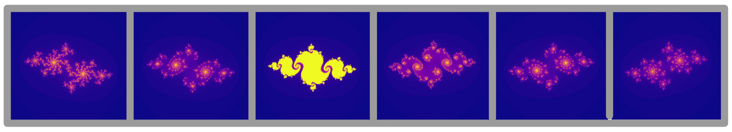

Figure 5: Julia sets

numbers a are chosen at equally spaced intervals in $[0, 2\pi]$. You should then develop a pygame function to display these files as a film like animation. The user should have speed and stop/start control over the animation.

4. **(core)** In the *Towers of Hanoi* problem we have three poles and n discs that fit onto the poles. The discs differ in size and are initially arranged in a stack on one of the poles, in order from the largest disc (disc n) at the bottom to the smallest disc (disc 1) at the top. The problem is to move the stack of discs from one pole to another pole while obeying the following rules.

   - Move only one disc at a time.
   - Never place a disc on one smaller than it.

   This problem can be solved by issuing a sequence/list of instructions for moving the discs in the appropriate way. We assume that the poles are arranged in a row and that each instruction to move a disc specifies its number and whether to move it left or right. We allow *wrap around*: if a disc is on the left pole an instruction to move left means to wrap around to the right pole, whereas if a disc is on the right pole an instruction to move right means to wrap around to the left pole. A solution to the Towers of Hanoi problem for 3 discs is displayed in Figure 6.

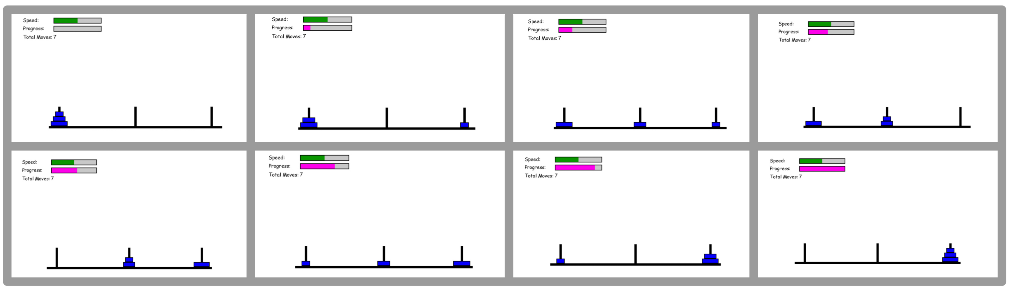

Figure 6: Towers of Hanoi with 3 discs

We can solve the problem by recursion. First we move the stack comprising the top $n - 1$ discs to an empty pole, then we move the largest disc (i.e. the bottom disc of the starting stack) to the other empty pole. Finally we move the stack of $n - 1$ discs onto (the pole with) the largest disc. Figure 7 illustrates this recursive approach.

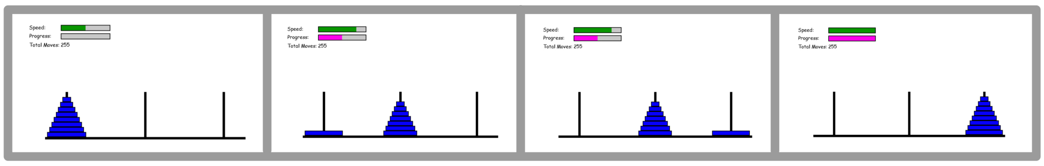

Figure 7: Solving the Towers of Hanoi recursively

Your task here is to write a function that implements this idea to create, given input n, the list of moves required to solve the Towers of Hanoi problem for n discs[12]. You should then develop a function that displays a printout of the sequence of configurations of a solution to the problem for n discs. (To be used for a small number of discs in practice - this is a warm up for the graphical animation.)

5. **(core)** Using the function you developed in part 4 to compute the list of moves for the Towers of Hanoi problem you should now develop a pygame function that gives an animation of the problem. The user should be able to control the initial number of discs, stopping/starting, and the speed of the animation. The initial number of discs may be any integer in the interval $[1, 16]$. (Discs should be moved directly from one pole to another, i.e. without showing any intermediate motion.)
6. **(extension)** Work out the definition for functions required to solve the two following variants of the Towers of Hanoi problem. (1) The problem where wrap around is not permitted: you can only move discs between adjacent poles. (2) The problem where initially there are $2n$ discs numbered according to size (as before). The discs with odd number are in a stack on the left pole, the discs with even number are in a stack on the right pole. The problem is to move the discs with odd number to the right pole and the discs with even number to the left pole (still satisfying the two rules given above, with wrap around allowed). Develop your pygame animation function to show solutions to these two problems.
7. **(extension)** Develop a version of your Towers of Hanoi pygame animation function from part 5 that moves the discs smoothly between the poles (i.e. now showing the intermediate motion). Once you have done this, add a menu (in pygame) so that the user can choose to either run the original animation or the smooth animation.
8. **(extension)** Develop a version of the Towers of Hanoi pygame animation where a player can play the problem as a game, by moving the discs from pole to pole.
9. **(extension)** Create an animation in which the user is able to repeatedly focus in and out on any part of the Mandelbrot set. (See the tutorial 4 bonus Notebooks in the GitHub repository for examples of Mandelbrot set images generated using matplotlib.pyplot.) To do this you may want to integrate the use of a function that generates image files (using matplotlib.pyplot) with a pygame function to control the animation.

Note on the extensions. In this project you are expected to develop one extension. Doing more will not achieve more marks: the point is to concentrate on the quality of the ideas and design of your code in the core sections and the extension that you choose.

Technical Note. To run the video animations in this project you will need to work locally, i.e. using your own distribution of python or that of the University. (You cannot do this via the Noteable server.)

References

[1] R. Sedgewick, K. Wayne, and R. Dondero. *Introduction to Programming in Python: An Interdisciplinary Approach.* Addison-Wesley Professional, 2015.

[12] To get a better idea of how to do this see the description of the Towers of Hanoi problem in the file `towers_of_hanoi.pdf`. Note that this is an excerpt of Section 2.3 of [1]. (Do not be confused however by the fact that [1] uses its own specialist libraries and functions, such as `stdio` and `stdio.writeln`.)

$$\begin{tikzcd}
X \arrow[r] \arrow[rr, bend right, "\text{Direct pathway}"'] & Y \arrow[r] \arrow[phantom, u, "\text{Indirect pathway}"] & Z
\end{tikzcd}$$

Figure 8: Schematic of the FFL network for gene Z that is regulated by gene X and gene Y.

8 The feed-forward loop and gene expression

One of the goals of systems biology is to understand gene networks, which involves understanding how the expression of one gene affects the expression of other genes. Ordinary differential equations can be used to describe the dynamics of gene regulation networks, which are composed of network motifs that recur throughout the network. One motif of significant interest is the feed-forward loop (FFL) and we will study this network pattern in this project. The FFL is a 3 gene pattern, with genes X, Y and Z. We are interested in how gene X regulates gene Z. There are two ways it can do this:

- Directly - expression of X can activate (or repress) gene Z directly
- Indirectly - expression of X can activate (or repress) gene Y. Gene Y then activates (or represses) gene Z

We call the network coherent when the direct path and indirect path have the same effect (i.e. both paths repress Z or both paths activate Z). The network is called incoherent if the direct and indirect path play different roles (e.g. if the direct path activates Z but the indirect path activates Y, which then suppresses Z). For a short video explanation of this network, see https://www.youtube.com/watch?v=YoM7VmAzylE.

The ODEs describing the concentrations $y(t)$ and $z(t)$ of genes Y and Z that are regulated by gene X with input concentration $x(t)$ are

$$\frac{dy}{dt} = B_y - \alpha_y y(t) + \beta_y f(x(t), K_{xy}), \tag{8}$$
$$\frac{dz}{dt} = B_z - \alpha_z z(t) + \beta_z g(x(t), y(t), K_{xz}, K_{yz}), \tag{9}$$

where B_i is the base rate of production of gene i, α_i is the degradation rate of gene i, β_i is related to maximal expression of gene i, and K_{ij} represents how much of gene i is required to regulate gene j. $f(x(t), K_{xy})$ and $g(x(t), y(t), K_{xz}, K_{yz})$ are regulation functions and their form depends on whether genes are activated or repressed. If X activates Y, then

$$f(x(t), K_{xy}) = \frac{(x/K_{xy})^H}{1 + (x/K_{xy})^H} \tag{10}$$

If X represses Y then

$$f(x(t), K_{xy}) = \frac{1}{1 + (x/K_{xy})^H} \tag{11}$$

Unless otherwise stated, use $H = 2$ throughout this project

The regulation function g combines the regulation function f for X regulating Z and Y regulating Z:

$$g(x(t), y(t), K_{xz}, K_{yz}) = f(x(t), K_{xz}) f(y(t), K_{yz}) \tag{12}$$

where the appropriate form of f must be chosen for activation or repression.

8.1 Project activities

All core activities should be attempted. You should pick two of the non-core activities 5–8 to develop according to your interests.

1. **[core]** First, consider expression of a single gene G with concentration $g(t)$. Write a fourth order Runge-Kutta algorithm with an appropriate time step to solve for the concentration of gene product, $g(t)$ which has dynamics governed by

$$\frac{dg}{dt} = B - \alpha g + \beta \frac{g^2}{g^2 + k^2}, \tag{13}$$

using constants $B = 0.1, \alpha = 0.1, \beta = 2, k = 1$ and initial concentration $g(0) = 0.5$

2. **[core]** Consider the 3 gene pathway described by equations 8, 9 and a constant value for X, where X activates Y, X activates Z and Y activates Z. Choose the appropriate form of functions f, g for this situation. Use a fourth order Runge-Kutta algorithm to solve for the expression levels of Y and Z. You can write your own algorithm or use the built in solver solve_ivp from the SciPy library, using the RK45 option. Use $B_y = B_z = 0, \beta_y = \beta_z = 1, x(t) = 1$ and choose the remaining parameters and initial conditions from one of the gene networks listed in Table 1. These values were obtained from data on measured gene expression levels, which is presented in the publication "Modeling Gene Regulation Networks Using Ordinary Differential Equations" which can be accessed at https://doi.org/10.1007/978-1-61779-400-1_12.

3. **[core]** Using the same coherent 3 gene pathway, investigate the effects of using different parameters in ODEs 8, 9 and discuss how they affect the dynamics of the system.

4. **[core]** There are 8 different combinations of activation and repression that are possible in this network. Investigate how different combinations of activation and repression affect the dynamics by choosing the appropriate forms of f and g. Discuss the features that you see in your solutions with coherent and incoherent networks.

5. In complex, interlocked FFL networks, in addition to regulating gene Z_1, genes X_1 and Y_1 also regulate gene X_2, which controls Y_2 and Z_2 in a FFL. Furthermore, X_2 and Y_2 regulate Z_3. This network is shown in the schematic in Fig. 9. Investigate the dynamics of this interlocked system built up of incoherent FFLs.

6. In other parts of this project, it is assumed that $H = 2$ in equations 10, 11. Investigate the effects of using other values of H and discuss whether this affects the accuracy of parameters inferred from real data.

7. Consider variations in this network that include feedback into X. Describe the possible regulatory effects between genes, write down the corresponding ODEs, show solutions and discuss.

8. When gene expression data is available, it is a challenging problem to infer the ODE parameters for the network for various reasons, including the lack of analytic solutions and noise in the data. A numpy data file "FFL_project_Q8_data.npy" is on the Blackboard project page (a csv version is also available). You can load the npy file into your notebook using np.load('file_name'). This is data produced from a 3 gene coherent FFL with some additional noise. Investigate which model parameters can reproduce this network without the noise. (Here are some of the parameters to get you started: $X = 1, B_y = B_z = 0, \alpha_y = \alpha_z = 0.2, \beta_y = \beta_z$.)

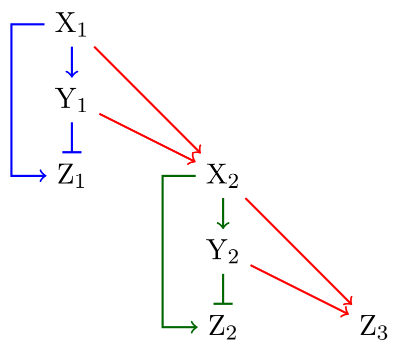

Figure 9: Complex interlocked feed forward loops

FFL 1: X: Gene GCN4; Y: Gene LEU3; Z: Gene ILV5							
Parameters and initial conditions	α_y	α_z	K_{xy}	K_{xz}	K_{yz}	$y(0)$	$z(0)$
Estimates	0.44	0.69	0.90	0.60	0.56	0.55	0.47
FFL 2: X: Gene GCN4; Y: Gene LEU3; Z: Gene ILV1							
Parameters and initial conditions	α_y	α_z	K_{xy}	K_{xz}	K_{yz}	$y(0)$	$z(0)$
Estimates	0.44	0.90	0.90	0.75	1.21	0.55	0.70
FFL 3: X: Gene PDR1; Y: Gene PDR3; Z: Gene PDR5							
Parameters and initial conditions	α_y	α_z	K_{xy}	K_{xz}	K_{yz}	$y(0)$	$z(0)$
Estimates	0.32	0.56	2.11	1.06	0.76	0.92	2.02

Table 1: Parameters for gene expression data for 3 gene networks