Introduction to Programming Project 1

- Instructor : Andrew Yarmola andrew.yarmola@uni.lu
- Course Schedule : Wednesday 14h00 - 15h30 Campus Kirchberg B21
- Course Website : sites.google.com/site/andrewyarmola/itp-uni-lux
- Office Hours : Thursday 16h00 - 17h00 Campus Kirchberg G103 and by appointment.

Project 1

Over the next three weeks. Project 1 will be due Wednesday Nov 23. You will work in teams of 2 on one of the following projects. You can also suggest your own projects pending my approval. Each team will have to do a different project. The goal of each project is for you to write a piece of software that solves a specific programming challenge. **Once you have chosen a project, I can provide extensive help with both programming and mathematics.**

Here are the general topics for each project.

- Topic 1 : (Homology) Implement the notions of a simplex, simplicial complex, and write a method to compute homology with $\mathbb{Z}/p\mathbb{Z}$ coefficients

- Topic 2 : (Chebyshev Polynomials) Implement the notions of a fraction (i.e. rational numbers), polynomials (with rational coefficients), Chebyshev polynomials, the Chebyshev inner product for polynomials, and a method for decomposing a polynomials in the Chebyshev basis.

- Topic 3 : (Finite Fields) Implement the notions of $\mathbb{Z}/p\mathbb{Z}$ for p prime, polynomials over $\mathbb{Z}/p\mathbb{Z}$, irreducibility over $\mathbb{Z}/p\mathbb{Z}$, and field extension over $\mathbb{Z}/p\mathbb{Z}$.

- Topic 4 : (Elliptic Curve Cryptography) Implement the notions of an Elliptic curve over $\mathbb{Z}/p\mathbb{Z}$ of p prime, points and arithmetic on an Elliptic curve, the Diffie-Hellman key exchange, a simple encryption and decryption algorithm.

- Topic 5 : (Graphs) Implement the notion of an undirected graph. Create methods to check graph connectedness, find a minimal spanning tree, find a shortest path between two vertices, compute the closure of a graph, and find a maximal clique of a graph.

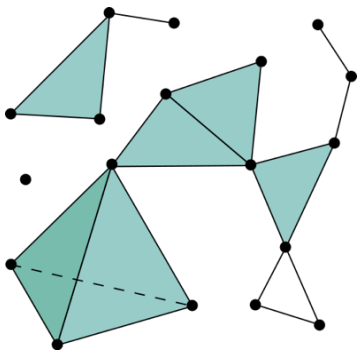
- Topic 6 : (Itinerary) Given an itinerary file of flights find the number of days spent in each country. You will have to look up countries from airport codes. You will generate a CSV spread sheet of the resulting data.

- Topic 7 : (Markov Chain Monte Carlo) Given a probability distribution, implement the Metropolis-Hastings random walk on a square lattice and apply it to computing expectations of a given function. This is called Markov Chain Monte Carlo.

Please let me know your team and choice of project by Wednesday Nov 9. You will submit your work on GitHub, so please make a GitHub team for this project. Additionally, **each team with have to write a series of tests to check that their code does what it should**.

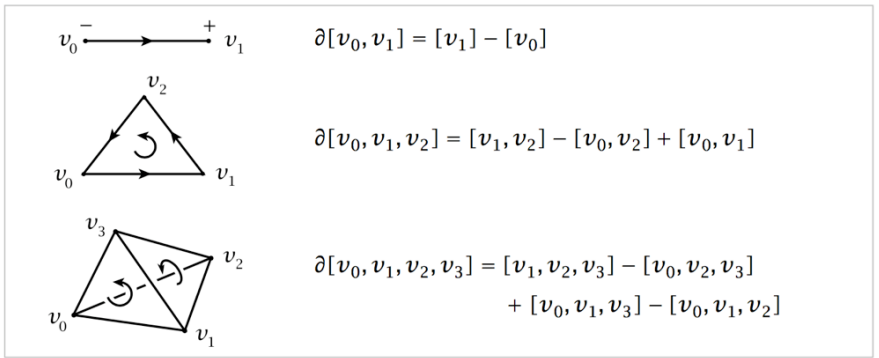
Topic 1 : Homology

A **simplicial complex** is a topological space that is a union of **simplices** (i.e. n-dimentional triangles) along their bounrady. In particular, any **face** of a simplex in the complex, must also be in the complex. They look like



We will make one simplification : a simplex will always be determined by its vertex set. Thus, let S denote a set of vertices, then a family X of non-empty finite subsets of S is **an abstract simplicial complex** if, for every set $\Delta \in X$, and every non-empty subset $\Gamma \subset \Delta$, Γ is also an element of X. See more at https://en.wikipedia.org/wiki/Abstract_simplicial_complex.

Given an (abstract) simplical complex X, one can compute the **simplical homology** of this complex. This amounts to defining a linear map associated to the complex and doing some row-reduction. To do this, we **order all the vertices** of X so that each simplex has a **canonical presentation** as a list of vertices in increasing order. Using the ordering, we can define a **boundary operator** ∂_n that takes a n-simplex to a **linear sum** of $(n-1)$-simplcies. Here is a picture,



Let $\Delta_n(X)$ be the $\mathbb{Z}/p\mathbb{Z}$-module (i.e. vector space) generated by the n-simplices X and p prime. Then, ∂ extends to a linear map $\partial_n : \Delta_n(X) \to \Delta_{n-1}(X)$. The n^{th} **homology group** of X is defined to be $H_n(X) = \ker \partial_n \,/\, \text{im } \partial_{n+1}$. See book pages 105-107 of https://www.math.cornell.edu/~hatcher/AT/ATch2.pdf (these are PDF pages 8-10). Since $H_n(X)$ is $\mathbb{Z}/p\mathbb{Z}$-module, its completely determiend by its dimension,

The goals of this project are as follows :

- learn about `array` in the `numpy` module (see the course text). We will do this in a week or two in class.
- write a simplicial complex class that allows for adding of vertices and new simplecies.
- construct a way to save and read simplicial complexes to and from a file.
- construct a method that computes the simplical homology $H_n(X)$ over $\mathbb{Z}/p\mathbb{Z}$ for a given dimension n and prime p.
- write a series of tests to check your code

Topic 2 : Chebyshev polynomials

Chebyshev polynomials $T_n(x)$ can be defined using the recurrence relation

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2\,x\,T_n(x) - T_{n-1}(x). \end{aligned}$$

These polynomials are **ortho-normal** as functions from the interval $[-1, 1]$ to $\mathbb{R}$ using the inner product

$$\langle u, v \rangle = \int_{-1}^{1} \frac{u\,v}{\sqrt{1-x^2}}\,dx$$

for integrable $u, v : [-1, 1] \to \mathbb{R}$. In particular, given any polynomial $p(x)$ with rational coefficients, you can find an expansion of of this polynomial in terms of Chebyshev polynomials. Further, if $\deg p = n$ then

$$p(x) = \sum_{i=0}^{n} a_i T_i(x)$$

where $a_i \in \mathbb{Q}[\pi]$

Your goals for this project will be to :

- learn about operator overloading in python (see http://blog.teamtreehouse.com/operator-overloading-python). We will also do this in class in the next few weeks.
- define a `class` that represents a fraction and allows for adding, subtracting, and multiplying (you can also do division if you want).
- define a `class` that represents a polynomial with rational coefficients (using your class form above)
  - should write functions that add, subtract, and multiply polynomaials
- define a `class` that represents an element of $\mathbb{Q}[\pi]$
- a function that computes the (Chebyshev) inner product of polynomials
- a function that returns T_n for a given n
- a function that given a polynomial $p(x)$ returns a_i such that $p(x) = \sum_{i=0}^{n} a_i T_i(x)$
- write a series of tests to check your code
- if you have time, write code that lets you read polynomials from file in some format.

Topic 3 : Finite Fields

I assume most of you are familiar with the notion of finite fields. We will start with the field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ and build field extensions using irreducible polynomials. The key algorithms here will be to determine if a polynomial is irreducible over $\mathbb{F}_p$ and to generate (randomly) irreducible polynomials of a given degree. The key will be

Theorem The product of all irreducible monic polynomials over $\mathbb{F}_p$ of degree dividing m is equal to $x^{p^m} - x$.

Using this Theorem and some `gcd` arguments, you should be able to test if a polynomial is irreducible or not. Once you have an irreducible polynomial $p(x)$ over $\mathbb{F}_p$ of degree k, then $GF(p^k) = \mathbb{F}_p[x]/p(x)$. You can represent $\alpha \in GF(p^k)$ as $\alpha = \sum_{i=0}^{k-1} a_i x^i$ and perform arithmetic module $p(x)$.

Your goals for this project will be :

- learn about operator overloading in python (see http://blog.teamtreehouse.com/operator-overloading-python). We will also do this in class in the next few weeks.
- define a `class` representing an element of $\mathbb{F}_p$ and allows for adding, subtracting, multiplying, inverses, and division.
- define a `class` representing a polynomial over $\mathbb{F}_p$ and allows for adding, subtracting, multiplying, and computing the `gcd`.
- in the polynomial class, write a function to test irreducibility of a polynomial
- in the polynomial class, write a function to generate a random irreducible polynomial of a given degree
- define a `class` representing an element of $GF(p^k)$ and allows for adding, subtracting, multiplying, inverses, and division.
- write a series of tests to check your code
- if you have time, write code that lets you read polynomials from file in some format.

Topic 4 : Elliptic Curve Cryptography

For this project your job will be to write software that uses Elliptic Curve Cryptography (ECC) to encode and decode messages. Recall that an **elliptic curve** $E_{a,b}(\mathbb{F}_p)$ (in standard form) over $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ is the set of solutions to an equation of the form

$$y^2 = x^3 + a\,x + b$$

where $a, b \in \mathbb{F}_p$ and $-16\,(4\,a^3 + 27\,b^2) \neq 0$. A **point** on an elliptic curve is a solution $(x, y) \in E_{a,b}(\mathbb{F}_p) \subset \mathbb{F}_p^2$. Your task in this exercise will be to build a tool for doing arithmetic on an elliptic curve and using this to encrypt text.

As you may recall, if we add a special point O to $E_{a,b}(\mathbb{F}_p)$, called the **point at infinity**, then $E_{a,b}(\mathbb{F}_p)$ becomes a group under elliptic curve group law (see https://en.wikipedia.org/wiki/Elliptic_curve for details). Elliptic Curve Cryptography is based no the fact that given a point $Q \in E_{a,b}(\mathbb{F}_p)$ and $n\,Q = Q + \cdots + Q$, it is very hard to compute the integer n.

The Diffie-Hellman key exchange works as follows. Say Alice wants Bob to send her a secret message. She picks a (large) prime p, an elliptic curve $E_{a,b}$, a point $Q \in E_{a,b}(\mathbb{F}_p)$, and a **secret** integer d_A. She shares (publicly) with Bob the data p, a, b, Q and $d_A\,Q$. Bob pick as **secret** integer d_B and sends Alice $d_B\,Q$.

Alice can compute $K = d_A\,(d_B Q)$ and Bob can compute $K = d_B\,(d_A Q)$ using their secret integers respectively. Now, let $x(K)$ be the x coordinate of K. Alice and Bob can use $x(K)$ as a **shared cipher key** to encrypt and decrypt their messages using a pre-agreed upon **single key cipher algorithm**.

Your goals for this project will be :

- learn about operator overloading in python (see http://blog.teamtreehouse.com/operator-overloading-python). We will also do this in class in the next few weeks.
- define a `class` representing a point on an elliptic curve $E_{a,b}(\mathbb{F}_p)$
- your `class` above should allow for adding points on the elliptic curve, finding inverses, and multiplying them by integers.
- define a `class` called `ECC` that
  - can be initialized with the elliptic curve information (i.e. p, a, b) or generates it randomly if not supplied the data.
  - can set or generate a personal secret (i.e. d_A)
  - can on request set or pick $Q \in E_{a,b}(\mathbb{F}_p)$ for sharing
  - can encrypt a message given the partners public point (i.e. $d_B\,Q$)
  - can decrypt a message using the same information
    - the workflow should be something like this : Alice creates an instance of the `ECC` class and randomly pick p, a, b and Q. She can either randomly generate d_A or pick her favorite number. She sends the parameters $(p, a, b, Q, d_A\,Q)$ to Bob and he creates his own instance of `ECC` with those parameters. He then generates $d_B\,G$ and send it to Alice. She can give it to her encrypt function to generate an encrypted message. After receiving the message, Bob can decrypt it using the `ECC` class.
- write your code so that lets you read and write encrypted and decrypted messaged from file.
- for the encryption and decryption algorithm, just use the `cryptography.fernet` module along with `int.to_bytes()` and `base64.b64encode()` from the `base64` module
- write a series of tests to check your code

Topic 5 : Graphs

For this project you will implement the notion of an undirected graph and code some classical graph algorithms. Your graph `class` should be able to do the following :

- create methods to add vertices and edges to your graph
- create methods to write and load the graph from a text file (pick how you want to encode things)
- learn how to use the `heapq` module : https://docs.python.org/3.5/library/heapq.html
- create a method to check graph connectedness
- create a method to compute a minimal spanning tree of the graph (Prim's algorithm)
  - a **minimal spanning tree** for a graph G is an acyclic subgraph which contains every vertex of G and uses the minimal number of edges
- create a method to compute a shortest path between two vertices (Dijkstra's algorithm)
- create a method to compute the closure of a graph
  - two vertices are said to be **adjacent** in a graph G if they are connected by an edge.
  - the degree $\deg(v)$ of a vertex v in a graph G is the number of edges incident at v
  - suppose G is a graph of n vertices, the **closure** $\mathrm{cl}(G)$ is obtained from G by (recursively) adding an edge between all vertices u, v with $\deg(v) + \deg(u) \geq n$.
- create a method to find a maximal clique of a graph (Bron–Kerbosch algorithm)
  - a **clique** is a subgraph H of G where every vertex in H has an edge to every other vertex in H
- write a series of tests to check your code

Topic 6 : Itinerary

The goal of this project is write a piece of software that determines the number of days you have spent in a country given your travel itinerary. You will be given a text file where each line contains flight information of the form

```
ATL Jan 13 2016 14:00 > PAR Jan 14 2016 00:10
PAR Feb 7 2016 15:37 > IST Feb 7 23:45
...
```

The airport codes will always be `IATA` codes (International Air Transport Association airport codes). Your job is to parse this data and determine the countries where these airports are located, the number of days spent in each country, and output this information in a human readable CSV file. Here is a more detailed description :

- write code to parse each line of the file to get the necessary information for you computations
- write code to use an **online** service to loop up the country associated to each `IATA` code in the file
  - for the online service you can use http://www.airport-data.com/api/doc.php
- create a function that given some date range and the itinerary file will output the number of days spent in each country during the supplied date range.
  - for visa purposes, the days you spend in a country are counted as the day you enter, the day you leave, and all days in between.
- write a series of tests to check your code

Topic 7 : Markov Chain Monte Carlo

The purpose of this project is to use statistical methods to estimate the integral

$$\mathbb{E}_\mu(f) = \int_{\mathbb{R}^n} f\,\mu\; d\text{vol}$$

where μ is probability density on $\mathbb{R}^n$ with compact support and f is an integrable function. Monte Carlo integration is a general tool where you pick a random collection of points in your domain and, by averaging the integrand over those points, you can approximate the integral. The purpose of Markov Chain Monte Carlo methods is to use your knowledge of μ to improve your sampling sequence.

Using μ we can construct a discrete-time Markov chain (of random variables) X^t that converges to μ. More precisely, we can write a function that generates a sequence $\{x^t\}_{t=1}^\infty \in \mathbb{R}^n$ such that

$$\lim_{T\to\infty} \frac{1}{T}\#\{x^t \in A \mid t \le T\} = \int_A \mu\; d\text{vol}.$$

Using such Markov chain one can show that

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t\le T} f(x^t) = \mathbb{E}_\mu(f).$$

You will use the **Metropolis–Hastings algorithm** to generate your sequence $\{x^t\}_{t=1}^\infty$. For this algorithm, you choose (or are given) some **symmetric transition distribution** g with mean 0 (e.g. a uniform distribution on a ball or a normal distribution). To build $\{x^t\}_{t=1}^\infty$ you randomly pick x^1 in your domain and then

- given x^t, pick a proposed y from the distribution $x^t + g$.
- flip a biased coin with probability $\min\{1, \mu(y)/\mu(x^t)\}$ of getting heads
- if the coin yields heads, let $x^{t+1} = y$, else $x^{t+1} = x^t$. You can now use $\{x^t\}_{t=1}^\infty$ and a running average to approximate $\mathbb{E}_\mu(f)$. Additionally, you can approximate the error using the Central Limit Theorem!

The goals of your project are as follows.

- learn about random sampling the in `numpy` module (see https://docs.scipy.org/doc/numpy/reference/routines.random.html)
- given a python function μ that represents a probability density, write a generator to yield a Markov sequence $\{x^t\}_{t=1}^\infty$.
  - you should allow for an input of a python generator g but default to something reasonable from the `numpy` module.
- write a method to approximate $\mathbb{E}_\mu(f)$ up to some desired (estimated) error or gives up after a good enough time if the error estimate is very hard to reach.
- write a series of tests to check your code