vkcfm Documentation

Vince Knight

Sep 18, 2018

Contents

The class notes are available here: https://vknight.org/cfm/

CHAPTER 1

Class meetings

1.1 Week 01: Introduction

1.1.1 Pre: Discussing programming + teaching approach

1.1.2 Lab: Installation of software assistance

1.1.3 Post: Installation issues and jupyter demo

1.2 Week 02: Variables, Conditional Statements and Loops

1.2.1 Pre: Demo of Bisection Method

Corresponding lab sheet:

- Variables, Conditional Statements and Loops

Objectives

- Motivate the use of code through an algorithm for solving equations numerically;
- Describe variable assignment;
- Conditional statements;
- While loops.

Notes

Explain to students that we are going to use programming to solve the following equation:

$$x^3 - 2 - 3cos(x) = 0$$

Give students time to attempt to solve this equation.

Some might use calculators and attempt to approximate: ask how and why.

We're going to step back and consider a different equation first.

Ask students to consider the equation:

$$x^3 = 2$$

How would we find solutions to this equation?

Agree on $x = \sqrt[3]{2} = (2)^{1/3}$ and show how we can compute this in Python:

```
>>> (2) ** (1 / 3)
1.259921...
```

Note that we can also assign variables:

```
>>> x = 2 ** (1 / 3)
>>> x
1.259921...
```

We can then use this to check:

```
>>> x ** 3
2.0
```

Let us see what this value of `x` gives in the equation we're considering.

Python has a number of libraries (we will see this more later) that give various functionality, include the `math` library, let us import it for `math.sqrt`. **Demo tab completion at this stage.**:

```
>>> import math
>>> x ** 3 - 2 - 3 * math.cos(x)
-0.917676...
```

Ask students if they have an idea of how to find an actual solution?

- What is an actual solution?
- What is an acceptable solution?

What does the following imply:

```
>>> x += 1
>>> x ** 3 - 2 - 3 * math.cos(x)
11.44955...
```

Have a discussion and discuss the fact that this implies that a solution is somewhere between $2^(1/3)$ and $2^(1/3) + 1$.

Then discuss the *bisection* method:

- Draw a plot;
- Explain the logic.

Do some steps by hand:

Let us reset our value of `x`:

```
>>> x -= 1
>>> x
1.259921...
```

Let us set up a and b:

```
>>> a, b = x, x + 1
>>> f_of_a = a ** 3 - 2 - 3 * math.cos(a)
>>> f_of_b = b ** 3 - 2 - 3 * math.cos(b)
```

Now consider the mid point of a and b:

$$\frac{a+b}{2}$$

$$>>> mid_point = (a + b)/2$$

Compute f of this mid point:

```
>>> f_of_mid_point = mid_point ** 3 - 2 - 3 * math.cos(mid_point)
>>> f_of_mid_point
4.01504...
```

Have a discussion about what our next :math:‘a‘ and :math:‘b‘ should be (draw this possibly).

How can we test if two numbers of the same sign?:

```
>>> f_of_a * f_of_mid_point
-3.6845...
>>> f_of_b * f_of_mid_point
45.9704...
>>> f_of_a * f_of_mid_point > 0
False
>>> f_of_b * f_of_mid_point > 0
True
```

Now discuss what the next step would be:

```
>>> b = mid_point
>>> f_of_b = f_of_mid_point
>>> mid_point = (a + b) / 2
>>> f_of_mid_point = mid_point ** 3 - 2 - 3 * math.cos(mid_point)
>>> f_of_mid_point
1.25989793...
>>> f_of_a * f_of_mid_point > 0
False
>>> f_of_b * f_of_mid_point > 0
True
```

How long should we repeat this for?:

```
>>> f_of_mid_point < 10 ** (-6)
False
```

We will use `if` statements and `for` loops to get the computer to automate this for as long as we require.

Write the following:

```
>>> a, b = x, x + 1
>>> f_of_a = a ** 3 - 2 - 3 * math.cos(a)
>>> f_of_b = b ** 3 - 2 - 3 * math.cos(b)
>>> tol = 10 ** (-6)
>>> while abs(f_of_a) > tol:
...     mid_point = (a + b) / 2
...     f_of_mid_point = mid_point ** 3 - 2 - 3 * math.cos(mid_point)
...     if f_of_mid_point * f_of_a > 0:  # If a and mid point have same sign
...         a = mid_point
...         f_of_a = f_of_mid_point
...     else:
...         b = mid_point
...         f_of_b = f_of_mid_point
```

Now that this has run we can confirm:

```
>>> mid_point
1.3731447...
>>> f_of_mid_point < tol
True
```

Lab sheet

Show how these three components will be gone over in the lab sheet. Encourage students to investigate them fully to make sure they understand the approach.

Highlight that there is room for improving the code which we will do in the **next** lab sheet.

1.2.2 Lab: Variables, Conditional Statements and Loops Lab Sheet

1.2.3 Post: Reactive discussion + peer exercise

1.3 Week 03: Data Structures and Functions

1.3.1 Pre: Demo of creating a random number generator

Corresponding lab sheet:

- Data Structures and Functions

Objectives

- Motivate the use of lists and functions by creating a random number generator.
- Describe list manipulation;
- Describe functions and start discussing writing good code;
- Give insight about random numbers.

Notes

Explain to students that we are going to use programming to generate random numbers.

Ask students to discuss in groups how they would generate a random number?

Bring this discussion together, perhaps some students will talk about using dice, flipping a coin, etc…

Ask how would we be able to check that a number is being generated randomly?

Lead the conversation to the notion of being able to predict a number no better than by "chance".

Discuss how this could be done by a computer, there are actually only very few **true** random number generators:

- Atmospheric noise.
- Thermal noise.
- Cosmic background radiation measured over a short amount of time.

We are going to look at something called *pseudo*-random number generators.

There are a number we could choose:

- Middle Square Method
- Blum Blum Shub
- Linear Congruence Generator

We will consider the last one (LCG) which was considered for a little while to be state of the art (before being proved to be cyrptographically unsafe: ie predictable).

This generator takes the form:

$$X_{n+1} = aX_n + c \text{mod} m X_0 = \text{s}$$

Where a, c, m, s are some parameters.

In groups choose some parameters and ask students to generate some random numbers.

Here is an example using $a = 2$, $c = 1$, $s = 0$ and $m = 4$:

n	X_n
0	0
1	1
2	3
3	3
4	3
5	3

Is this random? Did any other group come up with more randomness?

Now let us code this, to do so we will make use of two new programming concepts:

- Lists
- Functions

First let us write a function that represents the definition of the random number generator:

```
>>> def random_number_generator(previous_term,
...                             modulus=4,
...                             multiplier_a=2,
...                             multiplier_c=1):
...     """
...     Generate a random number using
...     the linear congruential generator
...     """
...     return (previous_term * multiplier_a + multiplier_c) % modulus
```

Let us confirm the table above:

```
>>> random_number_generator(previous_term=0)
1
>>> random_number_generator(previous_term=1)
3
>>> random_number_generator(previous_term=3)
3
>>> random_number_generator(previous_term=3)
3
```

This becomes quickly tedious: it would be much easier to be able to "hold" the calculated numbers in a container of some sort. In python these are called lists:

```
>>> seed = 0
>>> random_numbers = [seed]
>>> number_of_numbers = 10
>>> for _ in range(number_of_numbers):
...     random_numbers.append(random_number_generator(random_numbers[-1]))
>>> random_numbers
[0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

If the numbers we were generating were truly random what would the mean of our list be?:

```
>>> sum(random_numbers) / len(random_numbers)
2.545...
>>> sum(range(4)) / 4
1.5
```

Our choice of parameters is clearly poor, here is a more common choice:

```
>>> modulus = 2 ** 32
>>> multiplier_a = 1664525
>>> multiplier_c = 1013904223
```

I am going to use the code I wrote previously so I will wrap it in a function:

```
>>> def generate_random_numbers(number_of_numbers, seed, modulus, multiplier_a,␣
↪multiplier_c):
...     """Generate N random numbers"""
...     random_numbers = [seed]
...     for repetition in range(number_of_numbers):
...         random_numbers.append(random_number_generator(random_numbers[-1],
...                                                       modulus=modulus,
...                                                       multiplier_a=multiplier_a,
...                                                       multiplier_c=multiplier_c,))
...     return random_numbers
```

Let us generate a thousand random numbers:

```
>>> random_numbers = generate_random_numbers(number_of_numbers=10 ** 3,
...                                          seed=0,
...                                          modulus=modulus,
...                                          multiplier_a=multiplier_a,
...                                          multiplier_c=multiplier_c)
>>> sum(random_numbers) / len(random_numbers)
2114463563.02497...
```

We will see in a few weeks time how to plot with python but here's a quick example:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(random_numbers)
[<matplotlib.lines...
```

Lab sheet

Show how these two things will be gone over in the lab sheet. Potentially discuss how the previous demo could be improved.

Highlight that this is just a demo of using lists and functions: not a course on random number generation: in fact this algorithm is known to not be sufficient and **also** python and most programming languages all have random number generators that can be used directly.

1.3.2 Lab: Data Structures and Functions

1.3.3 Post: Reactive discussion + peer exercise

1.4 Week 04: Lab Sheet 03: Reading and writing to file and recursion

1.4.1 Pre: Demo of Collatz conjecture

Corresponding lab sheet:

- Recursion and Reading and writing to file

Objectives

- Motivate the use of recursion and writing/reading to file through the study of the Collatz conjecture;
- Describe recursion;
- Describe reading and writing to file.

Notes

Tell students we are going to investigate a particular mathematical process defined by the following recursive relationship:

$$f(n) = \begin{cases} n/2 & \text{if } n = 0 \mod 2 \\ 3n+1 & \text{if } n = 0 \mod 2 \end{cases}$$

Ask students what :math:'f(2)' is?

Realise that we have what we will call a base case: computing $n = 2$ essentially ends the process.

What happens when we recursively call :math:'f(n)'?

Demonstrate what is meant by this with $n = 3$:

$$f(3) = 3 \times 3 + 1 = 10 \tag{1.1}$$
$$f(10) = 10/2 = 5 \tag{1.2}$$
$$f(5) = 3 \times 5 + 1 = 16 \tag{1.3}$$
$$f(16) = 16/2 = 8 \tag{1.4}$$
$$f(8) = 8/2 = 4 \tag{1.5}$$
$$f(4) = 4/2 = 2 \tag{1.6}$$
$$f(2) = 2/1 = 1 \tag{1.7}$$

We see that with $n = 3$ the process eventually finishes.

Ask students if they think this will always be the case? Why?

In groups, using pen and paper repeatedlycompute :math:'f(n)' for :math:'nin{2, 3, 4, 5, 6, 7, 8, 9, 10}

We know it terminates for $n \in \{2, 3, 4, 5, 8, 10\}$.

For $n = 6$:

$$f(6) = 6/2 = 3 \tag{1.8}$$

which we know terminates.

For $n = 7$:

$$f(7) = 3 \times 7 + 1 = 22 \tag{1.9}$$
$$f(22) = 22/2 = 11 \tag{1.10}$$
$$f(11) = 3 \times 11 + 1 = 34 \tag{1.11}$$
$$f(34) = 34/2 = 17 \tag{1.12}$$
$$f(17) = 3 \times 17 + 1 = 52 \tag{1.13}$$
$$f(52) = 52/2 = 26 \tag{1.14}$$
$$f(26) = 26/2 = 13 \tag{1.15}$$
$$f(13) = 3 \times 13 + 1 = 40 \tag{1.16}$$
$$f(20) = 20/2 = 10 \tag{1.17}$$

which we know terminates.

Ask students if they think this will always be the case? Why?

Explain that there is overwhelming evidence that this does indeed always terminate but that we do not know for sure if it is true.

Ask students if they know what this is called?

A conjecture.

Mention the following text from the corresponding wiki page:

> "Paul Erdős said about the Collatz conjecture: 'Mathematics may not be ready for such problems.' He also offered $500 for its solution.[9] Jeffrey Lagarias in 2010 claimed that based only on known information about this problem, 'this is an extraordinarily difficult problem, completely out of reach of present day mathematics.'"

In groups: write some code to check if the process terminates for a given :code:'n'

First let us write the function itself:

```
>>> def func(N):
...     """Function to apply the Collatz transformation to an integer N"""
...     if N % 2 == 0:
...         return int(N / 2)
...     return 3 * N + 1
>>> func(10)
5
>>> func(5)
16
```

Describe recursion as having two steps:

- A base case: in our case this is $N = 1$.
- Recursive relationship: in our case $a(N) = f(a(N-1))$

Let us write the process:

```
>>> def collatz_process(N, count=0):
...     """
...     Recursively call the collatz process and return the number of
...     times it was called. This is called the "stopping time".
...     """
...     if N == 1:
...         return count
...     count += 1
...     return collatz_process(func(N), count=count)
>>> collatz_process(4)
2
>>> collatz_process(7)
16
```

Ask students, to in group discuss the code and check if there are any questions.

Finally, explain that so far essentially all mathematicians have been able to do is test this process numerically and have found that it **always** terminates.

One way to do this is to keep track of the results so far on disk:

```
>>> with open("collatz_data.txt", "w") as f:
...     for N in range(2, 50):  # Could swap this for an infinite loop
...         stopping_time = collatz_process(N)
...         string = str(N) + "," + str(stopping_time) + "\n"
...         f.write(string)
```

We could then read this in and check from a given point or perhaps give the file to other to use.

Lab sheet

Show how these recursion will be gone over in the lab sheet. Also discuss reading and writing: highlight where the files needs to be.

1.4.2 Lab: Reading and writing to file, recursion and algorithms

1.4.3 Post: Reactive discussion + peer exercise

1.5 Week 05: Lab Sheet 04: Object Oriented Programming

1.5.1 Pre: Demo of group theory exercise

Corresponding lab sheet:

- Object oriented programming

Objectives

- Motivate the use of classes by exploring the permutation group.

Notes

Tell students we are going to investigate some group theory.

Ask them to speak in groups and write down the four properties of a group.

- Closure: if $a, b \in G$ then $\in G$.
- Identity: $\exists e in G: \ e \cdot a = a \cdot e = a \forall; a \in G$
- Inverse: $\forall\ a \in G \exists\ a^{-1} \in G: \ a \cdot a^{-1} = e$
- Associativity: $\forall; a, b, c \in G: \ (a \cdot b) \cdot c = a \cdot (b \cdot c)$

Consider the permutation group: https://en.wikipedia.org/wiki/Permutation_group. On two elements: $\{0, 1\}$ there are two permutations:

- $\sigma_{01}(0) = 0$ and $\sigma_{01}(1) = 1$ (identity)
- $\sigma_{10}(0) = 1$ and $\sigma_{10}(1) = 0$ (flip)

Ask student to write down the multiplication table for the permutation group on 2 elements:

$$\begin{pmatrix} 01 & 10 \\ 10 & 01 \end{pmatrix}$$

Explain that we can create an **abstract** object in Python that allows us to manipulate these elements in the same way that we can manipulate integers and/or floats. Or in the same way that `<insert popular video game>` manipulates characters.

Here is how we create a very basic class that allows us to create an element corresponding to a given σ:

```
>>> class Permutation():
...     """A class that corresponds to an element of a permutation group"""
...     def __init__(self, sigma):
...         """When creating an instance set attributes."""
...         self.sigma = sigma
...         self.N = len(sigma)
...     def permute(self, vector):
...         """Given a vector of integers permute them."""
...         return [self.sigma[i] for i in vector]
```

We can use this to create specific elements of the Permutation group:

```
>>> pi_01 = Permutation([0, 1])
>>> pi_10 = Permutation([1, 0])
>>> pi_01.sigma, pi_01.N
([0, 1], 2)
>>> pi_10.sigma, pi_10.N
([1, 0], 2)
```

The `permute` method allows us to permute a given vector: for example what does σ_{10} do to $(0, 1)$:

```
>>> pi_10.permute([0, 1])
[1, 0]
```

We now have all we need to define the group operation on the permutation class:

```
>>> class Permutation():
...     """A class that corresponds to an element of a permutation group"""
...     def __init__(self, sigma):
...         """When creating an instance set attributes."""
...         self.sigma = sigma
...         self.N = len(sigma)
...     def permute(self, vector):
...         """Given a vector of integers permute them."""
...         return [self.sigma[i] for i in vector]
...     def operate(self, other):
...         """Define the group operation on self and other"""
...         return Permutation(self.permute(other.permute(range(self.N))))
```

Redefining our new instances:

```
>>> pi_01 = Permutation([0, 1])
>>> pi_10 = Permutation([1, 0])
>>> pi_10.operate(pi_01)
<__main__.Permutation ...>
```

We see that a new instance of the *Permutation* class has been produced (which is expected) but we cannot really tell what it is. Let us implement another special method to do so:

```
>>> class Permutation():
...     """A class that corresponds to an element of a permutation group"""
...     def __init__(self, sigma):
...         """When creating an instance set attributes."""
...         self.sigma = sigma
...         self.N = len(sigma)
...     def permute(self, vector):
...         """Given a vector of integers permute them."""
...         return [self.sigma[i] for i in vector]
...     def __repr__(self):
...         return str(self.sigma)
...     def operate(self, other):
...         """Define the group operation on self and other"""
...         return Permutation(self.permute(other.permute(range(self.N))))
```

Redefining our new instances:

```
>>> pi_01 = Permutation([0, 1])
>>> pi_10 = Permutation([1, 0])
```

(continues on next page)

(continued from previous page)

```
>>> pi_10.operate(pi_01)
[1, 0]
```

We see that when σ_{01} operates on *sigma_{10}* we get *sigma_{10}* back. A nice way to be able to check this using Python's == operator is to include a new special method:

```
>>> class Permutation():
...     """A class that corresponds to an element of a permutation group"""
...     def __init__(self, sigma):
...         """When creating an instance set attributes."""
...         self.sigma = sigma
...         self.N = len(sigma)
...     def permute(self, vector):
...         """Given a vector of integers permute them."""
...         return [self.sigma[i] for i in vector]
...     def __repr__(self):
...         return str(self.sigma)
...     def __eq__(self, other):
...         return self.sigma == other.sigma
...     def operate(self, other):
...         """Define the group operation on self and other"""
...         return Permutation(self.permute(other.permute(range(self.N))))
```

Let us confirm this now:

```
>>> pi_01 = Permutation([0, 1])
>>> pi_10 = Permutation([1, 0])
>>> pi_10.operate(pi_01) == pi_10
True
```

One final change we're going to make is replace the `operate` method to use a special python method:

```
>>> class Permutation():
...     """A class that corresponds to an element of a permutation group"""
...     def __init__(self, sigma):
...         """When creating an instance set attributes."""
...         self.sigma = sigma
...         self.N = len(sigma)
...     def permute(self, vector):
...         """Given a vector of integers permute them."""
...         return [self.sigma[i] for i in vector]
...     def __repr__(self):
...         return str(self.sigma)
...     def __eq__(self, other):
...         return self.sigma == other.sigma
...     def __mul__(self, other):
...         """Define the group operation on self and other"""
...         return Permutation(self.permute(other.permute(range(self.N))))
```

We can now use the $\star$ operator:

```
>>> pi_01 = Permutation([0, 1])
>>> pi_10 = Permutation([1, 0])
>>> pi_10 * pi_01 == pi_10
True
```

Ask student to write code that uses this class to obtain the multiplication table for our group:

```
>>> def display_multiplication_table(elements):
...     for first in elements:
...         products = []
...         for second in elements:
...             products.append(first * second)
...         print(products)
```

We can now use this:

```
>>> permutations = [pi_01, pi_10]
>>> display_multiplication_table(elements=permutations)
[[0, 1], [1, 0]]
[[1, 0], [0, 1]]
```

Let us modify this to look at permutations of size $N = 3$. Explain that we will make use of a very handy Python library for creating permutations of things:

```
>>> import itertools
>>> N = 3
>>> permutations = [Permutation(list(perm)) for perm in itertools.
↪permutations(range(N))]
>>> permutations
[[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]]
```

Let us take a look at the multiplication table:

```
>>> display_multiplication_table(elements=permutations)
[[0, 1, 2], [0, 2, 1], [1, 0, 2], [1, 2, 0], [2, 0, 1], [2, 1, 0]]
[[0, 2, 1], [0, 1, 2], [2, 0, 1], [2, 1, 0], [1, 0, 2], [1, 2, 0]]
[[1, 0, 2], [1, 2, 0], [0, 1, 2], [0, 2, 1], [2, 1, 0], [2, 0, 1]]
[[1, 2, 0], [1, 0, 2], [2, 1, 0], [2, 0, 1], [0, 1, 2], [0, 2, 1]]
[[2, 0, 1], [2, 1, 0], [0, 2, 1], [0, 1, 2], [1, 2, 0], [1, 0, 2]]
[[2, 1, 0], [2, 0, 1], [1, 2, 0], [1, 0, 2], [0, 2, 1], [0, 1, 2]]
```

Can students see the various properties closure, associativity, inverse and identity?

In any remaining time, invite students to write code that checks these conditions.

Walk and discuss with them.

Closure:

```
>>> def test_closure(elements):
...     return all(first * second in elements
...                for first in elements
...                for second in elements)
>>> test_closure(elements=permutations)
True
```

Identity:

```
>>> def test_specific_identity_element(elements, identity):
...     return all(first * identity == first for first in elements)
>>> def test_identity_element(elements):
...     return any(test_specific_identity_element(elements=elements,␣
↪identity=identity)
...                for identity in permutations)
>>> test_identity_element(elements=permutations)
True
```

Inverse:

```
>>> def test_inverse_element_for_given_identity(elements, identity):
...     has_inverse = []
...     for first in elements:
...         products = []
...         for second in elements:
...             products.append(first * second)
...         has_inverse.append(identity in products)
...     return all(has_inverse)
>>> def test_inverse_element(elements):
...     return any(test_inverse_element_for_given_identity(elements=permutations,␣
↪identity=identity)
...                for identity in elements)
>>> test_inverse_element(elements=permutations)
True
```

Associativity:

```
>>> def test_associativity(elements):
...     return all((first * second) * third == first * (second * third)
...                for first, second, third in itertools.product(elements, repeat=3))
>>> test_associativity(elements=permutations)
True
```

These can all be brought together:

```
>>> def test_group(elements):
...     return (test_closure(elements=elements) and
...             test_identity_element(elements=elements) and
...             test_inverse_element(elements=elements) and
...             test_associativity(elements=elements))
>>> test_group(elements=permutations)
True
```

Not all subsets of a group are a group:

```
>>> test_group(elements=permutations[:-1])
False
```

We can also use this to easily check for larger group sizes:

```
>>> N = 4
>>> permutations = [Permutation(list(perm)) for perm in itertools.
↪permutations(range(N))]
>>> test_group(elements=permutations)
True
>>> N = 5  # This takes a little while
>>> permutations = [Permutation(list(perm)) for perm in itertools.
↪permutations(range(N))]
>>> test_group(elements=permutations)
True
```

Lab sheet

Highlight the concepts we've seen in the lab sheet. Highlight that most of the code we have written in this session is "normal python code", there is just a little bit of novelty surrounding the ability to create abstract things.

1.5.2 Lab: Object Oriented Programming

1.5.3 Post: Reactive discussion + peer exercise

1.6 Week 06: Symbolic mathematics

1.6.1 Pre: Demo of Sympy

Corresponding lab sheet:

- Symbolic mathematics with Sympy

Objectives

- Understand the difference between symbolic and numeric variables;
- Manipulate symbolic expressions;
- Use basic Sympy.

Notes

Ask students if the following expression is true and if/how they would check this using Python?

$$(\alpha + \beta)(\alpha - \beta)^2 = \alpha^3 - \alpha^2\beta - \alpha\beta^2 + \beta^3$$

Let students discuss in groups and write some code to do this.

Expect them to say that they would "try this for a large number of examples". Explain that this won't prove anything but just tries a number of examples: in the same way that you could do this by hand.

Some might say they would use wolfram alpha: explain that we're essentially going to see how to use an equivalent version of that with Python.

Explain that we are going to use a Python library called Sympy which allows us to do symbolic mathematics:

```
>>> import sympy as sym
>>> alpha, beta = sym.symbols("alpha, beta")
>>> sym.expand((alpha + beta) * (alpha - beta) ** 2)
alpha**3 - alpha**2*beta - alpha*beta**2 + beta**3
```

Now say we're going to use Sympy to answer a "textbook" secondary school question:

Identify all points of inflection of the following quartic:

$$f(x) = -x^4 + 9x^2 + 4x - 12$$

Ask students in groups to identify the steps we need to solve:

1. Identify derivative;
2. Identify roots of derivative;
3. Evaluate the second derivate at these roots.

Some other suggestions might be to look at roots of the function itself as well as asymptotic limits (say we can do this too!).

Let us first write a Python function:

```
>>> def f(x):
...     return - x ** 4 + 9 * x ** 2 + 4 * x - 12
>>> f(5)
-392
```

Now let us pass a symbolic variable to this function:

```
>>> x = sym.Symbol("x")
>>> f(x)
-x**4 + 9*x**2 + 4*x - 12
```

We can compute the derivative:

```
>>> der = sym.diff(f(x), x)
>>> der
-4*x**3 + 18*x + 4
```

We can find the roots of this derivative using Sympy's function called `solveset`:

```
>>> poi = sym.solveset(der, x)  # Find the points of inflection
>>> poi
{-2, 1 + sqrt(6)/2, -sqrt(6)/2 + 1}
```

Finally, we can take the second derivative and evaluate it at each point:

```
>>> second_der = sym.diff(der, x)
>>> for point in list(poi):  # We convert the poi to a list
...     print(point, (second_der.subs({x: point})) > 0)
-2 False
1 + sqrt(6)/2 False
-sqrt(6)/2 + 1 True
```

We see that 2 points of inflection give negative second derivative (so they are local maxima), whereas the other is a local minimum.

Lab sheet

Discuss the Sympy labsheet showing that there are other things that are covered as well.

Highlight that this and the labsheet is a brief overview and that there is a large amount of capability in Sympy.

1.6.2 Lab: Symbolic Mathematics

1.6.3 Post: Reactive discussion + peer exercise

1.7 Week 07: Library Toolkit

1.7.1 Pre: Demo of libraries

Monty Hall

Page Rank

SIR Model

1.7.2 Lab: Overview of Python libraries

Outcome: agree on code base to use.

The goal of this lab session is to agree on a choice of programming approach.

Students should still work through the final chapter of the course (an overview of a number of libraries) but the main outcome of the session should be a conversation with the tutor and an agreement on a course of action.

Email to students

Send this to students in preparation (BCc tutors):

```
Subject: Lab session goal: deciding on a programming strategy.

---

Dear students,

The goal of the next lab session is to identify the research project you
would like to tackle for your individual assessment <url>.

You should aim to work through the chapter of the course demonstrating a
number of Python libraries available to you: <>.

Your lab tutor will aim to have a conversation with you about the specific
project you plan to tackle. To make this conversation productive try and
have an idea of this before the lab session. The outcome of your
conversation should be an agreed set of programming concepts to consider for
your topic.

Thanks,
Vince
```

1.7.3 Post: Library research skill session?

1.8 Week 08: Produce code

1.8.1 Pre: Review of all programming concepts

Objectives

- Students to create their own summary of concepts from all chapters.

Ask each individual to spend 5 minutes writing a brief bullet point list of the concepts.

Once that's done, each student to spend 5 minutes comparing their list with a neighbor.

Now in groups of 4: come up with small examples for each thing on their list (10 minutes). 10 minutes.

Following this spend time as a class writing a list on the board and possibly giving more clarification.

1.8.2 Lab: Feedback on their own code

The goal is to take a look at students code and give immediate feedback. Time permitting students can obtain help.

Email to students

Send this to students in preparation (BCc tutors):

```
Subject: Lab session goal: feedback on your code

---

Dear students,

The goal of the next lab session is to feedback on your code.

Your lab tutor will aim to have a quick look at the code you have written
based on your conversation from the previous week. They will give you
potential areas for improvement and possibly direct you towards other
interesting aspects to consider.

For this to be as beneficial as possible, you should aim to have written
your code beforehand.

Note that this is not a session to debug your code for you but if there is
time after tutors have spoken to all all students they will possibly be able
to assist you.

Thanks,
Vince
```

1.8.3 Post: Reactive discussion + peer exercise

1.9 Week 09: LaTeX refresher

1.9.1 Pre: LaTeX Refresher + Writing Mathematics

Notes

Briefly discuss LaTeX and point students towards https://vknight.org/tex/

Describe plagiarism

Ask students to spend 3 minutes discussing what plagiarism is. This should be a reminder/revision of what was discussed during Library session.

Writing mathematics

Ask students to spend 5 minutes reading ../../../../assets/pdf/writing-mathematics-guidelines/main.pdf.

Spend 5 minutes in groups discussing principles of mathematical writing.

Spend 10 minutes capturing this on the board.

Some principles might include:

- Know your audience;
- Nudge your reader "with gentle commands";
- Writing well does not necessarily mean writing more;
- Give a roadmap for long arguments (help your reader);
- Use sentences;
- Use words;
- Use paragraphs;
- Use examples;
- Simplify and refine: until you submit it, it is a draft;
- Don't follow the guidelines when it's appropriate not to.

Highlight that next week we will be peer reviewing other work:

- First class meeting: review the three example pieces of work
- Last class meeting: peer review draft of each others work

The three examples we will consider can be found here, note that they are not perfect and we will critically discuss them:

- `sir.pdf`
- `page-rank.pdf`
- `monty-hall.pdf`

1.9.2 Lab: Start writing

The goal of this session is to ensure writing of the report is going well. This includes a refresher of LaTeX as well as specific consideration given to writing of Mathematics.

Email to students

Send this to students in preparation (BCc tutors):

```
Subject: Lab session goal: writing of your report

---

Dear students,

The goal of the next lab session is to feedback on the writing of your
report.

Your lab tutor will aim to have a quick look at your writing to date and
feedback on specific use of LaTeX as well as mathematical writing.

For this to be as beneficial as possible, you should aim to have refreshed
your LaTeX (see <>) and at the very least started your report.

Thanks,
Vince
```

1.9.3 Post: Reactive discussion + peer exercise

1.10 Week 10: Finishing

1.10.1 Pre: Critically discuss some of the writing of the previous pieces

Objectives

- Use peer review to discuss and better understand marking criteria for individual research project.

Notes

5 mins: Discuss the marking criteria:

- Code (50%): This is all about demonstrating a good understanding of the code taught and also demonstrating the **learning/understanding** of a new coding concept. **This does not mean copying and pasting a completely different tutorial.**
- Content (25%): Demonstrating an actual good understanding of a specific topic. For example if you write a piece of work on computational numerical integration and makes errors this will lose marks. However, if you choose to write about Pythagoras's theorem and do a great job demonstrating understanding than that will gain you marks.
- Presentation (20%): Spelling and general writing but also clarity of mathematical discussion.
- Lab work (5%): How engaged you have been through the 11 weeks, preparation for labs etc...

5 mins: Ask students to form groups of 3 or 4. Decide on a piece of work to discuss.

5 mins: As individuals rate according to marking criteria.

10 mins: As a group discuss.

10 mins: As a class discuss together.

Repeat the above with another piece of work.

Remind students that next time they should bring as complete a draft as possible.

Some notes about each piece:

- `sir.pdf`: This is well written and does a good job investigating new pieces of mathematics. The conclusion is a bit weak: so what is the question we can be left asking? The final figure could have also investigated other areas parameters.
- `page-rank.pdf`: the writing of this piece is weak. It could be clearer, with more definitions used, small details like the lack of the Figure caption etc... It does however do a nice job of exploring a completely new area of mathematics and delving in to networkx and numpy.
- `monty-hall.pdf`: this is a well written piece of work, it brings together exact mathematics through sympy as well as the writing out numeric simulations using the basic building blocks of programming.

1.10.2 Lab: Finishing/identification of difficulties

This final lab session is to help identify any final difficulties.

Email to students

Send this to students in preparation (BCc tutors):

```
Subject: Lab session goal: final queries

---

Dear students,

The next lab session will be your final one. Your tutor will aim to have a
conversation with you which should result in the identification of any
remaining difficulties.

Thanks,
Vince
```

1.10.3 Post: Peer review of first draft

Objectives

- Use peer review to discuss and give feedback on individual research project.

Notes

5 mins: Discuss the marking criteria:

- Code (50%): This is all about demonstrating a good understanding of the code taught and also demonstrating the **learning/understanding** of a new coding concept. **This does not mean copying and pasting a completely different tutorial.**
- Content (25%): Demonstrating an actual good understanding of a specific topic. For example if you write a piece of work on computational numerical integration and makes errors this will lose marks. However, if you

choose to write about Pythagoras's theorem and do a great job demonstrating understanding than that will gain you marks.

- Presentation (20%): Spelling and general writing but also clarity of mathematical discussion.
- Lab work (5%): How engaged you have been through the 11 weeks, preparation for labs etc…

5 mins: Exchange coursework with another individual

15 mins: Read and write feedback

10 mins: Pair and discuss feedback

10 mins: Class discussion

1.11 Week 11: Slack

CHAPTER 2

For Tutors

The role of lab tutors is vital to the success of this course.

Records for each student are to be kept on copies of `record.pdf` which will be used in the final marking of their coursework.

During each lab session, the `record.pdf` document should assist with:

- Speak to each student; making a note of their attendance.
- During the first 6 weeks: assist where necessary.
- From week 7 onwards: engage in discussions to help scaffold progress of students' coursework.

Throughout: feedback specific and/or general difficulties to Vince Knight.

After week 10: return the filled in copies of `record.pdf` to Vince Knight.

2.1 Guidance

- With code queries: point students at section of notes where the answer is and then tell them what is needed.
- With writing queries: ask them relevant questions (refer them to marking criteria etc...).
- Make an effort to ensure you are fair and not advantaging a given student.
- Aim to be encouraging and enthusiastic.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search