MATH 157: Mathematics in the world
Homework 5 (Due March 12th, 2019 at 1:00PM)

Please solve Problems 1-6 completely.

Problem 0

Please start thinking about whether you want to take a final exam, or to work on a final project. Write a paragraph about your thoughts about it (you will indicate your decision in HW6). Remember if you plan to do both, then the average of the two will be scored. The details are the following:

- The **final exam** will only contain variations of problems from either the homework, or problems from in-class worksheets that have been discussed.
- The **final project** is a paper to write (generally 5-10 pages, but there can be exceptions) about a topic related to the coursework, to be decided together with me and Patrick; the choice of topic will be due together with HW7.

Problem 1

1. Try to write the following Fibonacci sequence as a fraction.

$$\sum_{k=0}^{\infty} F_k x^k$$

   where $F_0 = 0$, $F_1 = 1$ and F_n is the n-th Fibonacci number.

2. Use similar arguments as in the class to find a closed formula for the Fibonacci numbers. [1]

[1] The following identity may be useful:

$$\frac{1}{(x-a)(x-b)} = \frac{1}{ab(a-b)}\left(\frac{a}{1-\frac{x}{b}} - \frac{b}{1-\frac{x}{a}}\right)$$

The problem may involve finding roots of a polynomial Ax^2+Bx+C, I highly recommend to use symbols a, b for the two roots during your computation, and substitute the actual values of the two roots only in the last step. You may also want to use equalities $ab = C/A$ and $a+b = -B/A$ to simplify your computations.

Problem 2

Given integers $k \geq 2$ and $n \geq 1$, let $T_{k,n}$ denote the number of ways of tiling a $k \times n$ strip with $k \times 1$ blocks. For example, we studies the case $k = 2$ in class and showed that $T_{2,n} = F_{n+1}$. In this problem, we will explore the sequences $T_{k,n}$ for arbitrary values of k. [2]

1. Find a recursive relation satisfied by $T_{k,n}$.
2. Compute $T_{k,n}$ for $1 \leq n \leq k$.
3. Use the previous two parts to find a closed formula for the generating function

$$f_k(x) = \sum_{n \geq 1} T_{k,n} x^n.$$

Problem 3

In this problem, we will introduce a new method of using generating functions to evaluate the sums. Consider

$$a_n = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k},$$

and the generating function

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

1. By substituting the expressions of a_n, express $f(x)$ as a double sum.
2. Switching the order of the inner sum and the outer sum, get a new expression of $f(x)$ as a double sum. Make sure to state clearly the range of the summations.
3. Evaluate the inner sum. [3]
4. Get a closed formula for $f(x)$ using the previous part.
5. Compare your formula with Problem 1, find a closed formula for the sum

$$a_n = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k}.$$

[2] If the two indices confuse you, you can imagine a fixed value of k and a new sequence $T_n = T_{k,n}$.

[3] You may use the following identity which is a variation of what we did in class:

$$x^2 \cdot \sum_{n=0}^{\infty} \binom{r+n}{n} x^{n+2r} = (\frac{x^2}{1-x})^{r+1}$$

Problem 4

Imagine we independently flip $n \geq 1$ coins (not necessarily fair).

1. Show that if at least one coin is fair, then the probability of getting an even number of heads is $1/2$. [4]

2. Prove the converse statement, that is, if the probability of obtaining an even number of heads is $1/2$, then at least one of the coins is fair.

Problem 5

1. Find a combinatoric proof of the equality

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

holds for all $n \geq 0$. [5]

2. Explain how your proof technique generalizes to show that

$$\sum_{k=0}^{n} \binom{n}{k}\binom{m}{k} = \binom{m+n}{n}$$

for all integers $m \geq n \geq 0$.

Problem 6

In this problem, we will learn how to draw the Julia set using Python. First, we demonstrate drawing a simple gradient image.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

## This function controls the image value at position x, y
f = lambda x, y: x + y

## Form a matrix of the values of f
```

[4] Hint: In class we used the expressions $(p \pm q)^n$ where $p = q = 1/2$. Try to generalize the argument using $\prod_{i=1}^{n}(p_i \pm q_i)$ where $p_i + q_i = 1$.

[5] We have already seen that $\binom{2n}{n}$ is the number of paths from $(0,0)$ to (n,n) which only move in the positive x and y directions. Given an integer $k \geq 0$, how many of these paths pass through $(k, n-k)$?

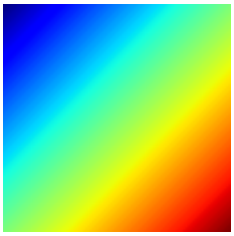
Once you find the answer to this question, compare it to previous equalities we proved. This is a very slick proof technique called *double counting* or *counting two ways*.

```
a = np.fromfunction(np.vectorize(f), (100, 100))

## Save the result to an image
mpimg.imsave("gradient.png", a)

## Show the image
plt.imshow(a)
plt.show()
```

The result is the following image saved in `gradient.png`. [6]



Second, we should explain how to use complex numbers in Python. Fortunately, this is very easy and requires no additional setup. For example, the number $1+2i$ is written as `1 + 2j`. All basic arithmetic functions carry over to complex numbers automatically. If you are interested in using some of the more advanced mathematical functions which typically reside in the `math` module (e.g., `log`, `exp`), you should instead use the `cmath` module.

```
import cmath

z = 1 + 1j

## 2j
print(z**2)

## (1.46869393992+2.28735528718j)
print(cmath.exp(z))
```

It is important to understand that functions are first rate objects in Python. They can be passed to other functions as arguments or returned as output. Consider the following example.

```
# This function returns the function "addition by x"
```

[6] The gradient matrix we constructed has values between 0 and 198. To choose actual RGB colors, matplotlib uses what is called a colormap. In this case, the lowest values are blue, and the highest values are red. You can experiment with different colormaps by adding an argument `cmap` to `mpimg.imsave` or `plt.imshow`. For example, try `plt.imshow(a, cmap='hot')`. See `http://matplotlib.org/examples/color/colormaps_reference.html` for a complete list of the built in colormaps.

```
def addition_by(x):
    return lambda y: x + y

## Given a function f and an input value x, this function prints f(x)
def print_value_of_function(f, x):
    print(f(x))

## Prints 11
print_value_of_function(addition_by(10), 1)
```

We are now ready to describe the fractal images we plan to construct. [7] Given a complex number $c \in \mathbb{C}$, consider the function $f_c \colon \mathbb{C} \to \mathbb{C}$ given by $f_c(z) = z^2 + c$. [8] The image we would like to generate shades the point corresponding to a complex number z according the behavior of the recursive sequence

$$z_0 = z, \qquad z_{n+1} = f_c(z).$$

Depending on the starting value z, the sequence may converge to 0, diverge, or even stabilize to a fixed value.

Since infinite sequences are hard to study computationally, we need to simplify our problem. First, we choose a threshold $T > 0$ (a real constant). Given an starting point z, we will construct the sequence z_n looking for elements satisfying $|z_n| > T$ (imagine this corresponds to divergence). We will color z according to the smallest value n such that $|z_n| > T$. To avoid a potentially infinite loop when z_n converges to 0, we will use a value $n_{\max}$ for starting points z such that $|z_n| \leq T$ for all $n < n_{\max}$.

1. Our fist task is to translate between the image coordinates and complex numbers. Imagine we would like to use a canvas of size $x_{\max}$ pixels by $y_{\max}$ pixels which should be mapped to the complex rectangle

   $$\{a + bi \mid a_{\min} \leq a \leq a_{\max}, b_{\min} \leq b \leq b_{\max}\}.$$

   Write a function `complexify(x, y, xmax, ymax, amin, amax, bmin, bmax)` returning the complex number $a + ib$. [9]

2. Write a function `compute_value(z, f, T, nmax)` which given a function f (imagine $f = f_c$ for some $c \in \mathbb{C}$), computes the value n associated with z, the threshold T,

[7] If you are interested in learning more about this topic, see `http://en.wikipedia.org/wiki/Julia_set` and `http://en.wikipedia.org/wiki/Mandelbrot_set`.

[8] In Python, that is `f_c = lambda c: (lambda z: z**2 + c)`.

[9] This may sound confusing due to the large number of variables involved, but it is quite simple actually. First, you can break the problem in two parts: determine a from x, $x_{\max}$, $a_{\min}$, $a_{\max}$, and likewise for b. If we focus on computing a, the task at hand is mapping the possible values $x = 0, \ldots, x_{\max} - 1$ equidistantly within the interval $[a_{\min}, a_{\max}]$.

and $n_{\max}$. See above for a detailed description of this process. [10]

3. Write a function

   `draw_fractal(f, filename, xmax, ymax, amin, amax, bmin, bmax, T, nmax)`

   which draws the fractal corresponding to the function f (and the rest of the auxiliary arguments), and saves it in the file given by `filename`.

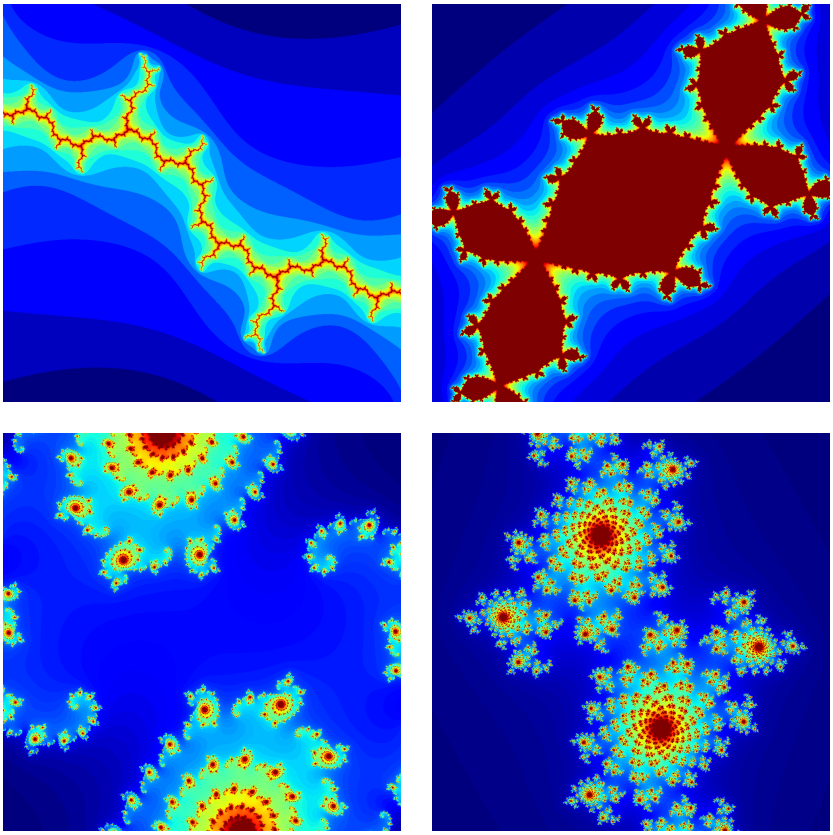
4. Use `draw_fractal` to construct four images given by the following parameters. [11]

f	$x_{\max}$	$y_{\max}$	$a_{\min}$	$a_{\max}$	$b_{\min}$	$b_{\max}$	T	$n_{\max}$
$f_c,\ c = 1i$	500	500	-0.1	0.1	-0.1	0.1	2	25
$f_c,\ c = -0.123 + 0.745i$	500	500	-1	1	-1	1	100	30
$f_c,\ c = -0.75 - 0.2i$	500	500	-0.5	0.5	-0.5	0.5	10	100
$f_c,\ c = -0.75 - 0.3i$	500	500	-1	1	-1	1	10	130

[10] While the functions f_c produce many fascinating images, we could also use higher degree polynomials as well as other trigonometric or transcendental functions. We are implementing `compute_value` in a generic manner, so we can potentially use it to plot other types of fractals.

[11] On my computer, it takes about 5s to construct a 500×500 image. If you are trying to debug your code, try using a lower resolution first (e.g., 100×100) until you get everything working.

It is worth noting that our implementation is easy to understand but rather inefficient. The simplest way to make it faster is by modifying the functions `compute_value` and `complexify` to operate on matrices rather than single values. This operation is called vectorization and is one of the main tricks of modern high-performance computing. We were able to cheat numpy that our functions are vectorized using `np.vectorize`.



5. (Extra credit) Draw up to 5 other fractals. Try using higher degree polynomials, trigonometric functions, or other color maps. **Be creative!**