

MM2090

Final Assignment

Author

Archish S

me20b032@smail.iitm.ac.in

Batch 7

This is the submission file for
End Semester Assignment

Indian Institute Of Technology Madras
June 29, 2021

Contents

1	Question 1	2
1.1	Task	2
1.2	Solution	2
1.2.1	Approach	2
1.2.2	Requirements	2
1.2.3	Output	2
1.2.4	Images	2
1.3	Code	4
2	Question 2	6
2.1	Task	6
2.2	Solution	6
2.2.1	Approach	6
2.2.2	Requirements	6
2.2.3	Output	6
2.3	Code	8
3	Question 3	9
3.1	Task	9
3.2	Solution	9
4	Question 4	10
4.1	Task	10
4.2	Solution	10
4.2.1	Rules	10
4.2.2	Approach	10
4.2.3	Requirements	10
4.2.4	Observation	10
4.2.5	Output	10
4.3	Code	12
5	Question 5	14
5.1	Task	14
5.2	Solution	14
5.2.1	Approach	14
5.2.2	Requirements	14
5.2.3	Observation	15
5.2.4	Output	15
5.3	Code	16
6	Question 6	19
6.1	Task	19
6.2	Solution	19
6.2.1	Approach	19
6.2.2	Requirements	19
6.2.3	Images	20
6.2.4	Output	20
6.3	Code	22

1 Question 1

1.1 Task

Generate 10 random numbers y_i between -1 and +1. Use these as points (x_i, y_i) where $x_i = i$ and show them as a scatter plot. Fit a higher order polynomial over these points to generate a pattern of how a random noise would look like. Sample the polynomial to generate about 100 points in the interval x_1 to x_{10} . Superpose a plot of this data along with original points (x_i, y_i) . Identify the peaks (location and height) programmatically. Print them out and confirm those with the plot.

1.2 Solution

Link to the GitHub repository for this question: [GitHub](#) ¹

1.2.1 Approach

I used the scikit-learn² library to generate a polynomial of degree n and used linear regression to fit the polynomial to the given data. Although this polynomial over fits the given data, it is fine as it retains the randomness of generated coordinates.

To find the extrema, I am traversing through the sampled polynomial and identifying the locations where the polynomial attains local maxima or minima.

1.2.2 Requirements

Language of choice: python

```
pip3 install numpy
pip3 install matplotlib
pip3 install sklearn
pip3 install scipy
```

1.2.3 Output

Locations of Extremas

Maxima at $x = 1.4084308430843084$ and $y = 2.195104229252479$

Minima at $x = 3.164766476647665$ and $y = -0.7656559740882756$

Maxima at $x = 5.218071807180718$ and $y = -0.19116876616108414$

Minima at $x = 7.025372537253725$ and $y = -0.5251533421000687$

Maxima at $x = 8.292639263926393$ and $y = -0.2132152812141186$

Minima at $x = 9.61087108710871$ and $y = -1.7288452082816121$

1.2.4 Images

¹Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_1

²Source: <https://scikit-learn.org/stable/>

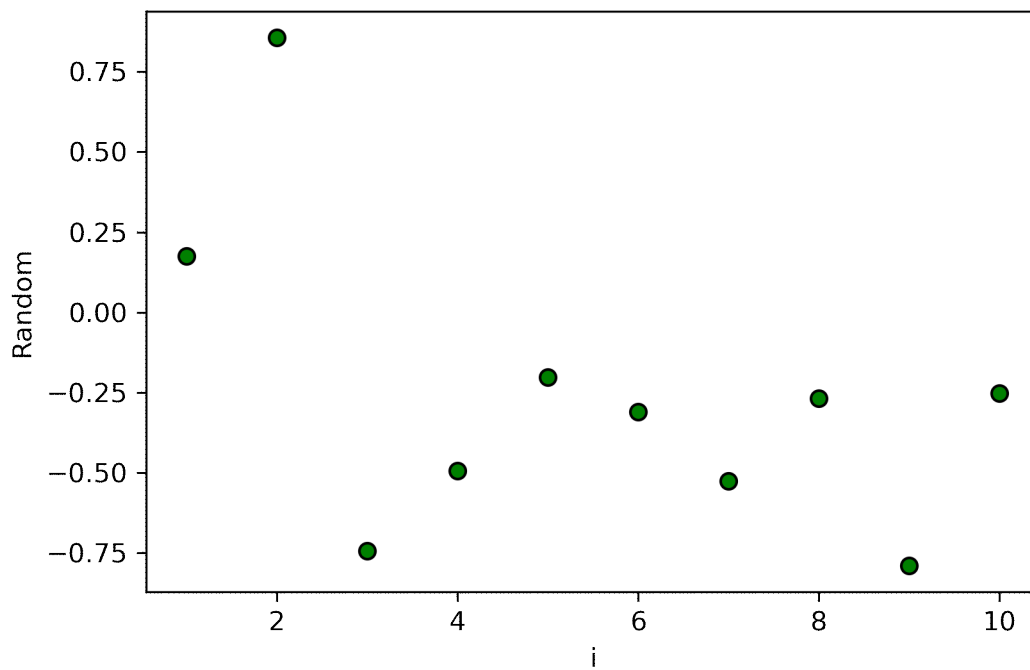


Figure 1: Generated Points

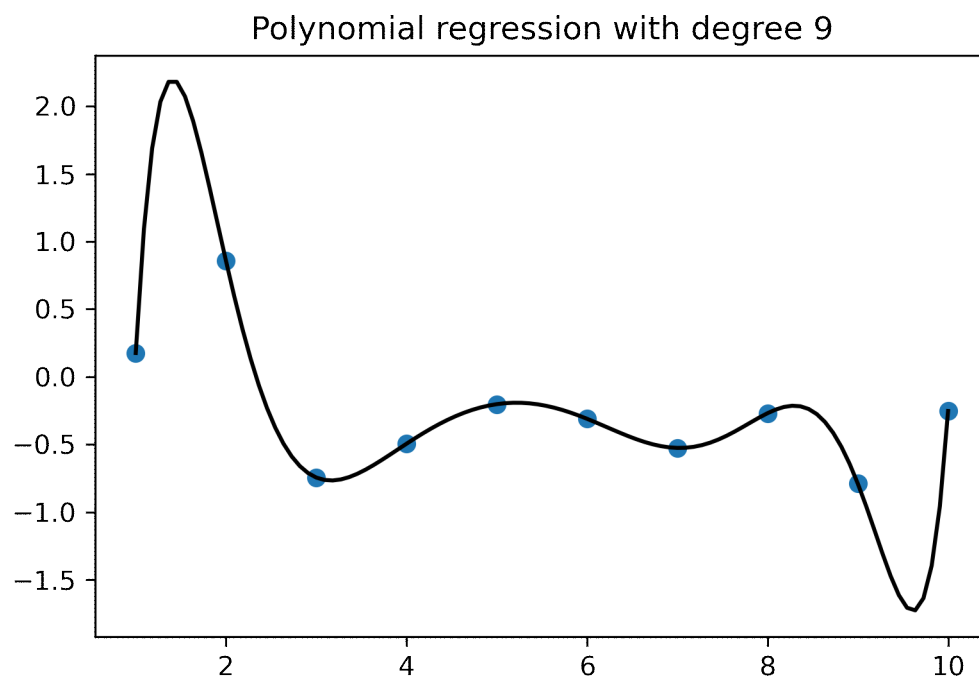


Figure 2: Fitted with Polynomial

1.3 Code

```
import numpy as np

from scipy import optimize

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

x_data = np.array(range(1,11))
y_data = -1+2*np.random.rand(10)

plt.figure()
plt.scatter(x_data, y_data, c='green', edgecolors='black')
plt.xlabel("i")
plt.ylabel("Random")
plt.savefig("Points.png", dpi=300)
plt.show()

# Using SciKit Learn to fit a nth degree polynomial
degree=9
polyreg=make_pipeline(PolynomialFeatures(degree), LinearRegression())
polyreg.fit(x_data.reshape(-1,1), y_data)

# Sampling the fitted polynomial
x_fit = np.linspace(1,10, num=100)
y_fit = polyreg.predict(x_fit.reshape(-1,1))

plt.figure()
plt.scatter(x_data, y_data)
plt.plot(x_fit, y_fit, color="black")
plt.title("Polynomial_regression_with_degree_"+str(degree))
plt.savefig("Fitted.png", dpi=300)
plt.show()

# Computing the gradient
grad = np.gradient(y_fit, x_fit)

plt.figure()
plt.scatter(x_data, y_data, c='green', edgecolors='black')
plt.plot(x_fit, y_fit, color="blue")
plt.plot(x_fit, grad, color="orange")
plt.ylim([-5,5])
```

```

plt.title("Polynomial_regression_with_degree_"+str(degree))
plt.show()

x_find = np.linspace(0.1,9.9, num=10000)
extrema_x = []
extrema_y = []

for t in range(1, len(x_find)-1):
    data = x_find[t-1:t+2].reshape(-1,1)
    fit = polyreg.predict(data)

    if fit[1]>fit[0] and fit[1]>fit[2]:
        print("Maxima_at_x=", x_find[t], "and_y=", fit[1])
        extrema_x.append(x_find[t])
        extrema_y.append(fit[1])
    elif fit[1]<fit[0] and fit[1]<fit[2]:
        print("Minima_at_x=", x_find[t], "and_y=", fit[1])
        extrema_x.append(x_find[t])
        extrema_y.append(fit[1])

plt.figure()
plt.scatter(x_data, y_data, c='green', edgecolors='black')
plt.scatter(extrema_x, extrema_y, c='red')
plt.plot(x_fit, y_fit, color="blue")
plt.plot(x_fit, grad, color="orange")
plt.ylim([-5,5])
plt.title("Polynomial_regression_with_degree_"+str(degree))
plt.show()

```

2 Question 2

2.1 Task

Consider the following sample images. Visually you can count that the number of objects in the images are 3, 2 and 5, respectively. Write a program that can take these images as inputs and report the number of objects as output. Assume that the objects are colored uniformly but different from the background which is also uniform. Do not assume the objects to be of same size or shape.

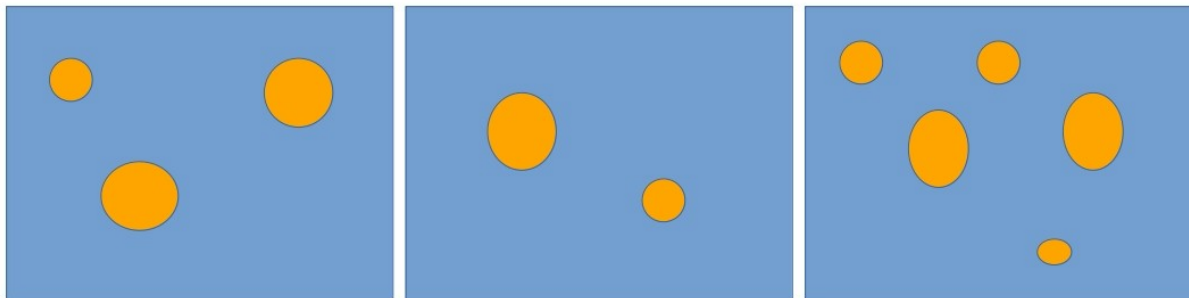


Image-1

Image-2

Image-3

2.2 Solution

Link to the GitHub repository for this question: [GitHub](#) ³

2.2.1 Approach

I am using the OpenCV⁴ library to pre-process the image – converting it to grayscale, and resizing for uniformity. Followed by applying a binary filter to sort the different colours into just two values – 0 and 255. Basically, every value greater than the minimum pixel value +10 is mapped to 255 and the rest to 0. Given that the image contains only two colours, this process will distinguish the two colours in every case.

To count the number of objects, I am using the fact that the gradients across the normal to the contours of the said region is non zero and uniform. `findContours` identifies continuous boundaries of regions having same colour, where the length of countours gives the number of the objects.

2.2.2 Requirements

Language of choice: python

```
pip3 install numpy
pip3 install matplotlib
pip3 install opencv-python
```

2.2.3 Output

³Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_2

⁴Source: <https://opencv.org/>

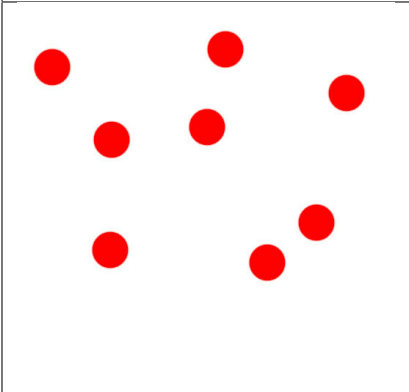
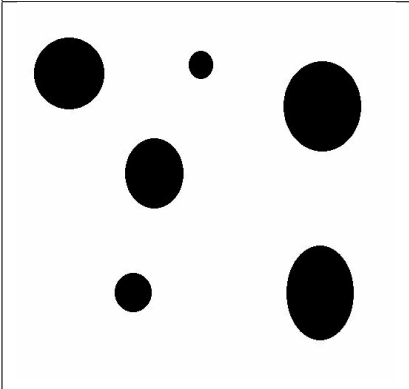
IMAGE	PREDICTION
	<p>Number of objects = 8</p>
	<p>Number of objects = 6</p>

Table 1: Predicted Number of Objects

2.3 Code

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

def getObjects(path):

    # Importing the image and preprocessing the image
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (512,512))

    # Updating the pixels which have a value greater than the
        minimum with 255 and the others with 0.\
    # Essentially, making a binary representation of the image.
    -, thresh = cv2.threshold(img, np.min(img)+10, 255, cv2.
        THRESH_BINARY_INV)

    # Locating the continuous closed regions where there is positive
        gradient across the normal.
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL
        , cv2.CHAIN_APPROX_SIMPLE)

    objects = str(len(contours))

    return thresh, int(objects)

# Processing Image 1
thresh, objects = getObjects('Image-1.jpg')
plt.imshow(thresh)

print("Number_of_objects_=", objects)

# Processing Image 2
thresh, objects = getObjects('Image-2.jpg')
plt.imshow(thresh)

print("Number_of_objects_=", objects)
```

3 Question 3

3.1 Task

Consider a temperature sensor placed near a valuable asset in a highly secure space. It records the local temperature every 5 seconds and writes it to a stream. The temperature data is flushed such that it keeps the data of only one hour. That is, you can view a window of only 720 data points at a time. Each new data point entering the window flushes the oldest data point out. Write two programs that do the following tasks.

- Program-A when executed will continuously send temperature data that looks almost flat (say, room temperature with a small random noise within 0.5 Kelvin). This is to simulate the data coming from the temperature sensor. At predetermined instance, introduce a spike (sudden increase of temperature by 10 K) for 10 seconds and revert back to room temperature. This spike is caused due to the presence of an intruder.
- Program-B keeps reading the incoming data, detects the spike and reports the instance when it took place. Verify if the detection is accurate.

Think of the output from Program-A being piped to Program-B to perform this check. The stderr from both the programs about the spike should match. The stdout of Program-A contains the temperature data. You should submit a report on how this is made along with the two codes.

3.2 Solution

Link to the GitHub repository for this question: [GitHub](https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_3) ⁵

⁵Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_3

4 Question 4

4.1 Task

Simulate the Langton's ant. Look up Wikipedia about what this problem is. Make sure you provide the formulation, program implementation and sample images to verify the behavior of the ant motion in the box. Consider a box of size $64 * 64$ for

4.2 Solution

Link to the GitHub repository for this question: [GitHub](#) ⁶

4.2.1 Rules

⁷ Initialize the state space of dimensions $64*64$ with all 0s, i.e., black pixels. Arbitrarily identify the starting location of the ant and update according to:

- At a white square, turn 90° clockwise, flip the colour of the square, move forward one unit,
- At a black square, turn 90° counter-clockwise, flip the colour of the square, move forward one unit.

4.2.2 Approach

At every time step, to keep track of the orientation of the ant, I exploited the use of residues of 4. So basically, computed the modulo of **orient** and 4 after every update. Following which the location of the ant at the start of the $(t + 1)^{th}$ time-step is calculated. Accordingly, the pixels are updated. The states are sampled every so often to observe the actions of the ant.

4.2.3 Requirements

Language of choice: python

```
pip3 install numpy
pip3 install matplotlib
pip3 install opencv-python
```

4.2.4 Observation

After running the simulation several times, one can observe that the state evolves exactly in a similar manner every time. During the first few moves, the ant moves predictably in a symmetric manner. As the time progresses, the state appears chaotic resembling an unsymmetric disc. At around 10000 time-steps, the ant reaches an order of repetition, a sequence of 104 steps which the ant repeats to infinity.

Every simulation resembled the exact same start state followed by the repeating highway. Although there is no conclusive proof that this occurs for every start state, the trajectory is unbounded in every case.

4.2.5 Output

Please find attached the video of the states.

⁶Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_4

⁷Source: https://en.wikipedia.org/wiki/Langton%27s_ant

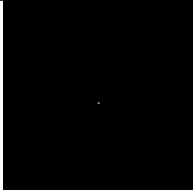
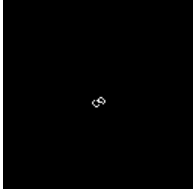
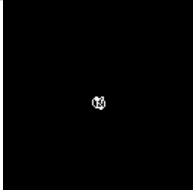

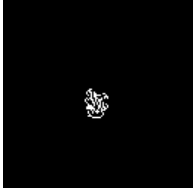
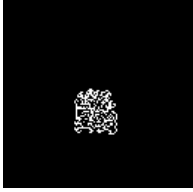
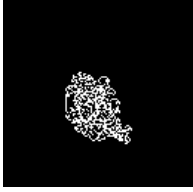
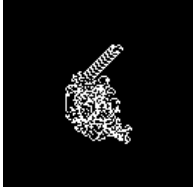
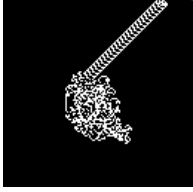
STATE	TIMESTEPS
	0
	100
	200
	500
	1000
	5000
	10000
	11000
	19000

Table 2: Simulation of States

4.3 Code

```
import numpy as np

import cv2

import matplotlib.pyplot as plt

# Initializing the State
n = 128
state = np.zeros((n,n))

plt.imshow(state)

# Randomly identifying a start location
start = ( int(n/3+np.random.rand()*n/3) , int(n/3+np.random.rand()*n/3) )

# Evolving the State
N = 2*10**4 # Number of timesteps

pos = start
orient = 0
for time in range(N):
    # Updating the current state
    if state[pos]==0.0:
        state[pos] = 255.0
        orient = (orient+1)%4 # Turning clockwise
    elif state[pos]==255.0:
        state[pos] = 0.0
        orient = (orient+3)%4 # Turning anti-clockwise

    # Finding the next state
    if orient == 0:
        if pos[1]-1>0:
            pos = (pos[0] , pos[1]-1)
    elif orient == 1:
        if pos[0]+1<n:
            pos = (pos[0]+1 , pos[1])
    elif orient == 2:
        if pos[1]+1<n:
            pos = (pos[0] , pos[1]+1)
    elif orient == 3:
        if pos[0]-1>0:
            pos = (pos[0]-1 , pos[1])

    if time<1500:
        if time%100==0:
            fname = "Iteration-"+str(time)+".png"
            cv2.imwrite(fname, state)
```

```
else:
    if time%1000==0:
        fname = "Iteration-"+str(time)+".png"
        cv2.imwrite(fname, state)
```

5 Question 5

5.1 Task

Seam carving is a special image manipulation. Look it up on Wikipedia to understand what it means. You are free to assumptions to simplify the problem definition as you require. Choose an image that contains two objects of interest separated by large distance and illustrate how your implementation of seam carving works on it.

5.2 Solution

Link to the GitHub repository for this question: [GitHub](#) ⁸

5.2.1 Approach

We compute the energy of the image using gradient magnitude ⁹ and find the seam with the minimum energy, a line which is connected to the adjacent pixels either via an edge or a corner. We use dynamic programming to keep track of the minimum seam.

The idea is, if we remove the so calculated minimum seam, the image doesn't lose important features, but we have successfully decreased the dimensions without resizing image.

This process can be repeated any number of times, to produce an image of desired dimensions.

- **Computing Energy**

The energy is computed by $e_i(I) = \left| \frac{\partial I}{\partial x} \right| + \left| \frac{\partial I}{\partial y} \right|$.

Using Sobel Filters, $p'_u = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \otimes I$ and $p'_v = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \otimes I$, we compute the derivative as $e_i(I) = |p'_u| + |p'_v|$.

- **Finding the Seam with least Energy**

To compute the seam with the least energy, we dynamically compute $M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$ where $M(i, j)$ stores the minimum energy value stored upto the pixel (i, j) .

The minimum energy thus will be stored in the last row of ****M**** and ****backtrack**** stores the list of pixels present in this s.

- **Deleting the Minimum Seam**

We recursively remove the seam with minimum energy till we reach the image of required size.

5.2.2 Requirements

Language of choice: python

```
pip3 install numpy
pip3 install matplotlib
pip3 install scipy
```

⁸Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_5

⁹Source: https://en.wikipedia.org/wiki/Image_derivatives

5.2.3 Observation

The algorithm¹⁰ is a computationally expensive version to achieve Seam Carving¹¹. Since the operations have to be sequential, there is no possibility of using parallel computing realms to speed up the process.

NOTE: It took around 25 mins for the first image and over 1 hour for the second image.

5.2.4 Output

ORIGINAL	SEAM CARVED	SCALE
		0.5
		0.5

Table 3: Processed Images

¹⁰Inspired By: <https://karthikkaranth.me/blog/implementing-seam-carving-with-python/>

¹¹Source: https://en.wikipedia.org/wiki/Seam_carving

5.3 Code

```
import cv2
import numpy as np

from scipy.ndimage.filters import convolve

from tqdm import trange

class Seam():
    def __init__(self, path):
        self.image = cv2.imread(path)
        self.name = path.split('.')[0]

    def findEnergy(self, image):
        filter_du = np.array([ [1.0, 2.0, 1.0], [0.0, 0.0, 0.0],
                               [-1.0, -2.0, -1.0],])
        filter_dv = np.array([ [1.0, 0.0, -1.0], [2.0, 0.0, -2.0],
                               [1.0, 0.0, -1.0],])

        filter_du = np.stack([filter_du] * 3, axis=2)
        filter_dv = np.stack([filter_dv] * 3, axis=2)

        convolved = np.absolute(convolve(image, filter_du)) + np.
            absolute(convolve(image, filter_dv))

        energy = convolved.sum(axis=2)

        return energy

    def minimum_seam(self, image):
        row, column, _ = image.shape
        energy = self.findEnergy(image)

        M = energy.copy()
        backtrack = np.zeros_like(M, dtype=np.int)

        for i in range(1, row):
            for j in range(0, column):

                if j == 0:
                    idx = np.argmin(M[i-1, j:j+2])
                    backtrack[i, j] = idx + j
                    min_energy = M[i-1, idx+j]
                else:
                    idx = np.argmin(M[i-1, j-1:j+2])
                    backtrack[i, j] = idx + j - 1
                    min_energy = M[i-1, idx+j-1]

            M[i, j] += min_energy
```

```

    return M, backtrack

def carve(self, image):
    row, column, _ = image.shape

    M, backtrack = self.minimum_seam(image)

    mask = np.ones((row, column), dtype=np.bool)

    j = np.argmin(M[-1])

    for i in reversed(range(row)):

        mask[i, j] = False
        j = backtrack[i, j]

    mask = np.stack([mask] * 3, axis=2)

    image = image[mask].reshape((row, column-1, 3))

    return image

def crop(self, image, scale):
    row, column, _ = image.shape
    new = int(scale * column)

    for i in range(column - new):
        image = self.carve(image)

    return image

def crop_column(self, scale_column):
    image = self.image
    image = self.crop(image, scale_column)

    fname = self.name + "_scaled_column_by_" + str(scale_column)
        + ".png"
    cv2.imwrite(fname, image)
    return image

def crop_row(self, scale_row):
    image = self.image
    image = np.rot90(image, 1, (0, 1))
    image = self.crop(image, scale_row)
    image = np.rot90(image, 3, (0, 1))

    fname = self.name + "_scaled_row_by_" + str(scale_row) + ".
        png"
    cv2.imwrite(fname, image)

```

```
return image
```

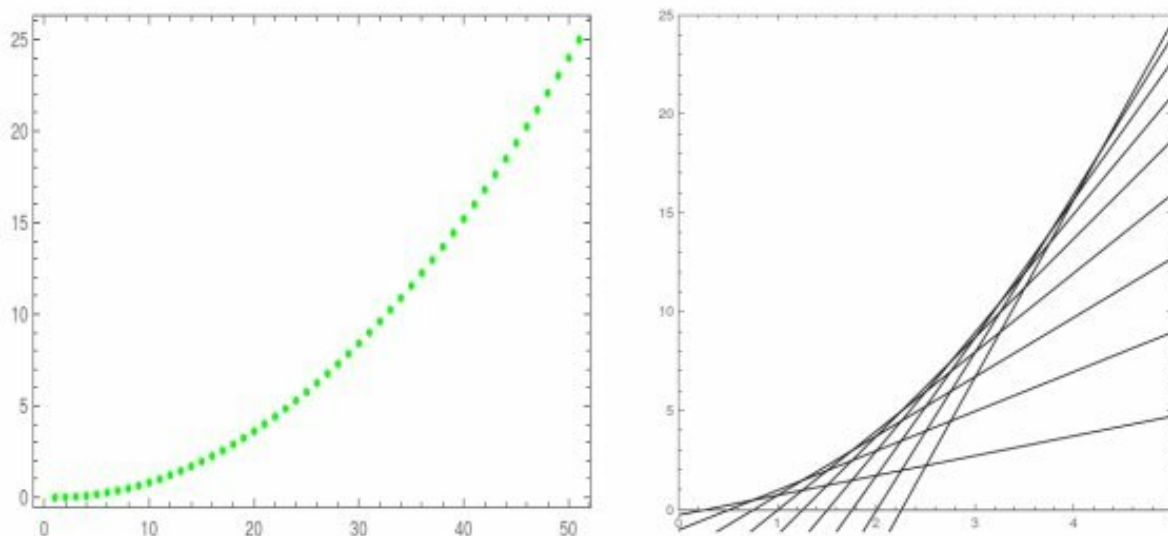
```
# Processing Image 1  
img1 = Seam("Image-1.jpg")  
out1 = img1.crop_column(0.5)
```

```
# Processing Image 2  
img2 = Seam("Image-2.jpg")  
out2 = img2.crop_column(0.5)
```

6 Question 6

6.1 Task

Consider a transformation by which a function $y_i(x_i)$ is transformed to another function $m_i(c_i)$ where m and c are slope and intercept of the original curve at different points on the curve represented by y as function of x . The envelope of lines of slope m drawn at each value of c has the same shape as that of the curve drawn using the points (x, y) . This is illustrated a parabola as shown in the figures below. Write a program that converts a function given as $y(x)$ to the new form either graphically or analytically. Make sure your process works even when the function $y(x)$ is changed to any other smooth and well-behaved function.



6.2 Solution

Link to the GitHub repository for this question: [GitHub](https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_6) ¹²

6.2.1 Approach

Sampling the given function for 50 points in the range $0 \leq x \leq 100$ and representing the said function using a scatter plot. The transformed function is nothing but the set of lines whose slope correspond to the slope of the tangent at a point (x_i, y_i) on the polynomial. Using the derivative operation in sagemath, I am computing the slope at a point x_i for $x_i \in f(x)$. At a point x_i , the modified function is given by $\frac{y-f(x_i)}{x-x_i} = \frac{\partial f(x)}{\partial x} \bigg|_{x=x_i}$.

6.2.2 Requirements

Language of choice: sagemath

```
pip3 install numpy
pip3 install matplotlib
pip3 install pandas
```

¹²Repo: https://github.com/Xerefic/MM2090-Solutions/tree/master/Final_Assignment/question_6

6.2.3 Images

The generated plots are for $f(x) = 0.01x^2$ NOTE: Please ensure that the function that is to be plotted and operated on is scaled. If the function is not scaled, the output will look congested, and one cannot distinguish the lines in the output.

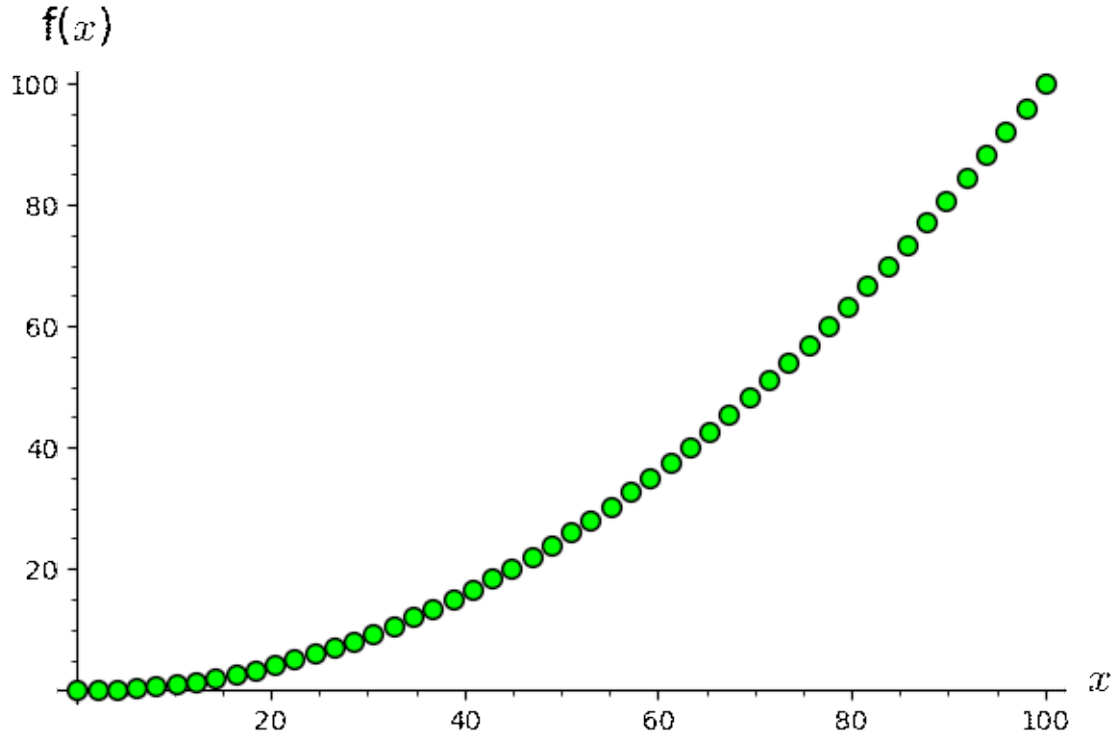


Figure 3: Function Plot

6.2.4 Output

Coordinates	Slope	X-Intercept	Y-Intercept
[0.0, 0.0]	0.000000	∞	0.000000
[10.20, 1.04]	0.204082	5.102041	-1.041233
[20.41, 4.16]	0.408163	10.204082	-4.164931
[30.61, 9.37]	0.612245	15.306122	-9.371095
[40.82, 16.66]	0.816327	20.408163	-16.659725
[51.02, 26.03]	1.020408	25.510204	-26.030820
[61.22, 37.48]	1.224490	30.612245	-37.484382
[71.43, 51.021]	1.428571	35.714286	-51.020408
[81.63, 66.64]	1.632653	40.816327	-66.638900
[91.84, 84.34]	1.836735	45.918367	-84.339858

Table 4: Modified Function

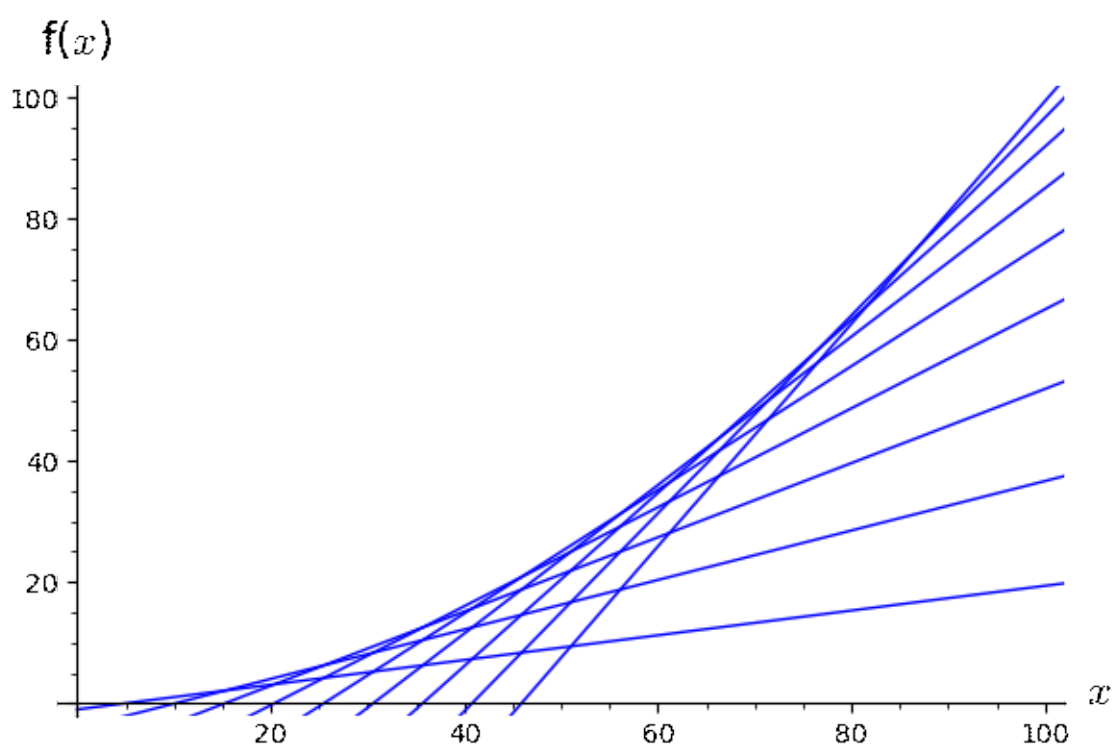


Figure 4: Modified Function Plot

6.3 Code

```
import numpy as np
import pandas as pd

import sage.plot.scatter_plot as scatter

import matplotlib.pyplot as plt

def plotTangent(equation):
    deq = equation.derivative(x)

    x_data = np.linspace(0, 100, num=50)

    # Makes the equation callable
    func = fast_callable(equation, vars=[x])
    slope = fast_callable(deq, vars=[x])

    # Plotting the function
    plots = [ (x_data[i], func(x_data[i])) for i in range(len(x_data)) ]

    g = Graphics()
    g += scatter.scatter_plot(plots, facecolor='lime')

    # Plotting the tangent lines
    p = Graphics()
    data = []
    for i in range(0, len(x_data), 5):
        x0 = x_data[i]
        y0 = func(x_data[i])
        s = slope(x_data[i])

        points = [ (x, y0+s*(x-x0)) for x in [0, x0-y0/s, x0, x0*10] ]
        p += line(points)
        data.append({ "Coordinates": [x0,y0] , "Slope": s, "X-Intercept": x0-y0/s, "Y-Intercept": y0-s*x0})

    mini = min(func(x_data[0]), func(x_data[-1]))
    maxi = max(func(x_data[0]), func(x_data[-1]))

    g.save("Plotted.png", axes_labels=['$x$', 'f($x$)'])
    p.save("Tangents.png", axes_labels=['$x$', 'f($x$)'], xmin=0,
        xmax=100, ymin=mini, ymax=maxi)

    return p, g, data

p, g, data = plotTangent(0.01*x^2)
```

```
# Plotting the Data
g.show()
p.show(xmin=0, xmax=100, ymin=-1, ymax=100)

data = pd.DataFrame(data)
print(data)
```