PS0_G1

January 20, 2024

1 Problem Set 0

1.1 Exercise 1

[61]:
```
import os
os.chdir("/Users/arthurjohnson/Library/CloudStorage/
  ↪OneDrive-UniversityofEdinburgh/Year 4/PNM for Economics/
  ↪PNM-for-Econ-Group-1")
import this
```

This tells us that in order to become 'good' coders, we should adhere to writing simple code (I presume so as to minimise operation time). Further, it can benefit us in the future with simpler code, as it allows us to have a higher recall.

Don't leave errors in your code - they will come back to bite you.

Ensure you name your code so that it is easier to refer to both later in the code file, and when copying across to new code.

1.2 Exercise 2

[40]:
```
str_x = "Facts about the distribution of wealth have been highlighted in a␣
  ↪large number of studies, including Wolff (1992, 1998), Cagetti and De Nardi␣
  ↪(2008), and Moritz and Rios-Rull (2015). A striking aspect of the wealth␣
  ↪distribution in the US is its degree of concentration. Over the past 30␣
  ↪years or so, for instance, households in the top % of the wealth␣
  ↪distribution have held about one-third of the total wealth in the econ- omy,␣
  ↪and those in the top 5% have held more than half. At the other extreme, more␣
  ↪than 10% of households have little or no assets. While there is agreement␣
  ↪that the share held by the richest few is very high, the extent to which the␣
  ↪shares of the richest have changed over time (and why) is still the subject␣
  ↪of some debate (Piketty 2014, Kopczuk 2014, Saez and Zucman 2014, and␣
  ↪Bricker et al. 2015)."

print('No. of occurences of "wealth":', str_x.count('wealth'))
print('No. of occurences of "distribution":', str_x.count('distribution'))
print('No. of occurences of "households":', str_x.count('households'))
print('No. of occurences of "assets":', str_x.count('assets'))
```

```
No. of occurences of "wealth": 4
No. of occurences of "distribution": 3
No. of occurences of "households": 2
No. of occurences of "assets": 1
```

Note: the above are all case sensitive! Shortcut to needing to count case-insensitive is to make the text (in this case 'str_x') all lowercase, and then rerun. You can achieve this by running the following: 'str_x_lower = str_x.lower()'

1.3 Exercise 3

1.3.1 Exercise 3a

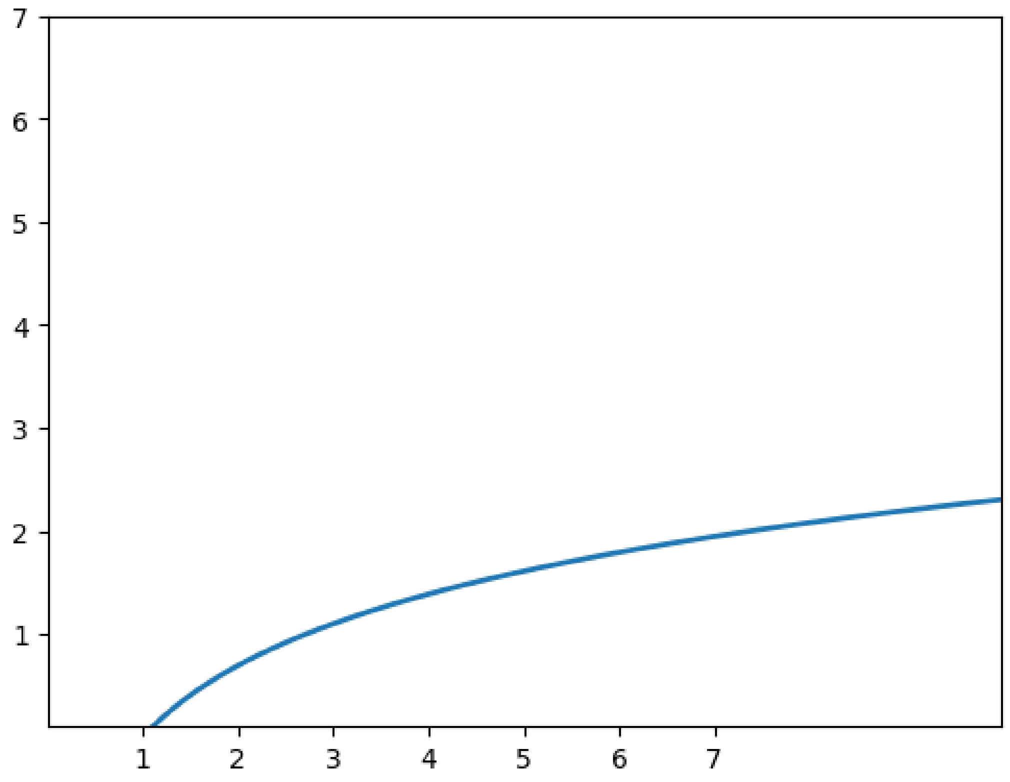
[68]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0.0000001,10,num = 100)
y = np.log(x)
z = np.sin(x)

print("y type:", type(y))
print("y size:", y.size)
print("z type:", type(z))
print("z size:", z.size)
```

```
y type: <class 'numpy.ndarray'>
y size: 100
z type: <class 'numpy.ndarray'>
z size: 100
```

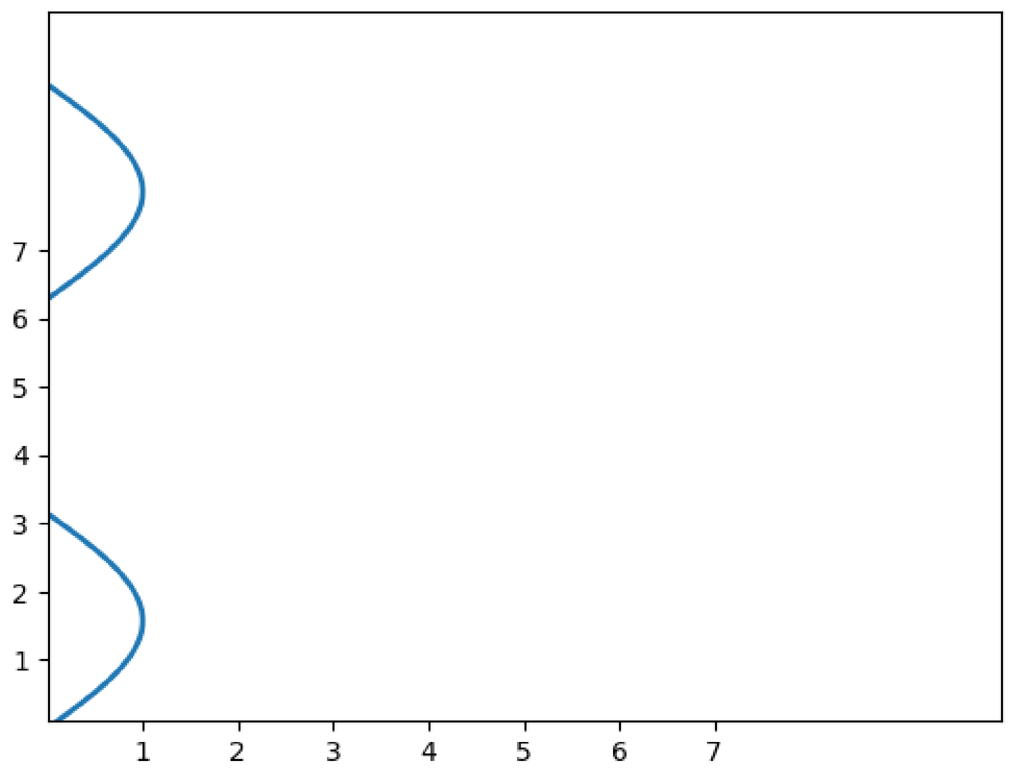
They are both NumPy arrays of size 100.

1.3.2 Exercise 3b

[64]:
```
fig, ax = plt.subplots()

ax.plot(x, y, linewidth = 2)

ax.set(xlim = (0, 10),
       xticks = np.arange(1, 8),
       ylim = (0.10),
       yticks = np.arange(1, 8))
plt.show()
```



[65]:
```
fig, ax = plt.subplots()

ax.plot(z, x, linewidth = 2)

ax.set(xlim = (0, 10),
       xticks = np.arange(1, 8),
       ylim = (0.10),
       yticks = np.arange(1, 8))
plt.show()
```

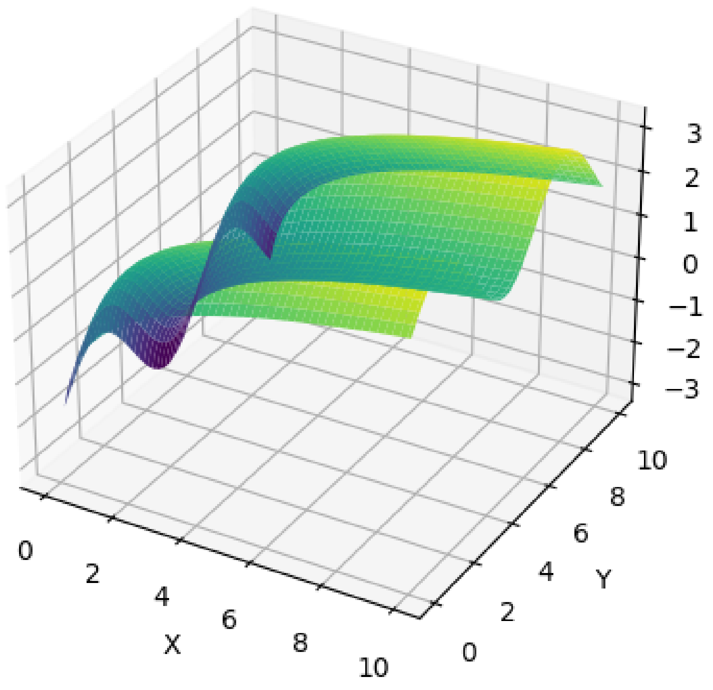


1.4 Exercise 4

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# General setup
x = np.linspace(0,10,100)
y = np.linspace(0,10,100)

X, Y = np.meshgrid(x, y)

Z = np.log(X) + np.sin(Y)

# Creating the plot:
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
```

```
plt.show()
```

```
/var/folders/1z/rmhh3bk123qg9411_qfj88580000gn/T/ipykernel_65310/737231513.py:11
: RuntimeWarning: divide by zero encountered in log
  Z = np.log(X) + np.sin(Y)
```

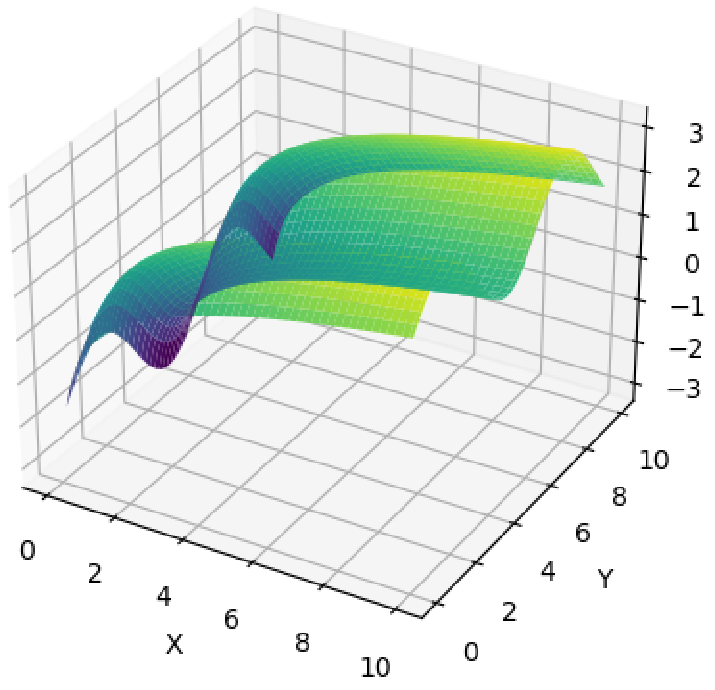


1.4.1 The above obviously comes up with an error, given than log(0) can't be calculated, when x=0. We found that we could skirt the warning (which we presume goes against the good practices of Python), or we could add a very small value to the equation. We have decided to put both, as we felt that the warning at least adheres strictly to the equation posed, but the latter is only a marginal difference.

1.5 Skirt warning:

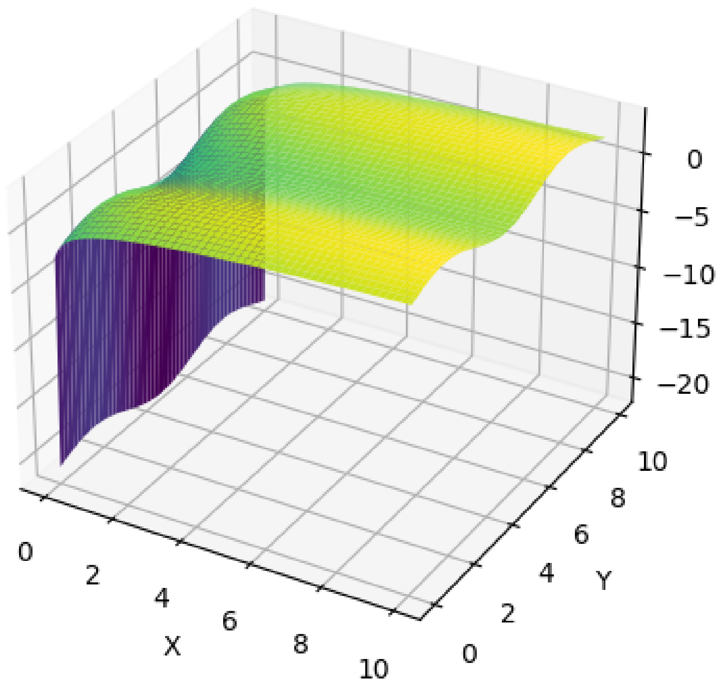
[7]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# General setup
x = np.linspace(0,10,100)
y = np.linspace(0,10,100)
```

```
X, Y = np.meshgrid(x, y)

np.seterr(divide = 'ignore')
Z = np.log(X) + np.sin(Y)
np.seterr(divide = 'warn')

# Creating the plot:
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```



1.6 Adjust equation

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# General setup
x = np.linspace(0,10,100)
y = np.linspace(0,10,100)

X, Y = np.meshgrid(x, y)

Z = np.log(X + 0.000000001) + np.sin(Y)

# Creating the plot:
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```



1.7 Exercise 5

[111]:
```python
import numpy as np
# Parameters values
alpha = 0.5
y = 10
p1 = 1
p2 = 2

def utility_function(x1,alpha,y,p1,p2):
    x2 = (y-p1*x1)/p2                      # Income not spent first good is␣
 ↪spent on the second
    utility = x1**alpha * x2**(1-alpha)
    return utility

obj_func = lambda x1: -utility_function(x1,alpha,y,p1,p2)


from scipy.optimize import minimize_scalar

min_output = minimize_scalar(obj_func)
x1_optimal = min_output.x
x2_optimal = (y - p1 * x1_optimal) / p2
print("Optimal x1:", round(x1_optimal,2))
print("Optimal x2:", round(x2_optimal,2))
```

```
Optimal x1: 5.0
Optimal x2: 2.5
```