ENGN 2020:

Homework #5

Brown University School of Engineering

Assigned: March 11, 2019, Due: March 18, 2019

Problem 1

Consider the three parabolas p_1, p_2, and p_3 shown in Figure 1. Your task is to use constrained numerical optimization to find the shortest path

$$p_1 \rightarrow p_2 \rightarrow p_3$$

Part a. Write this as a formal optimization; as in:

$$\begin{array}{ll} \text{minimize} & f(\underline{\mathbf{x}}) \\ \text{subject to} & g_1(\underline{\mathbf{x}}) = 0 \\ & \vdots \end{array}$$

Be explicit about f, g_1, etc.

Paper Submission

(Turn in your solution as a PDF to Canvas.)
Points: 1

Part b. Construct a suitable loss function that contains both the function to be minimized and the constraints. When submitting your code, set any factors that weight components of the loss function to 1.

Submission

Label: hw5_2b
Points: 1
Input variables: `xs` (a list of floats)
Output: a float

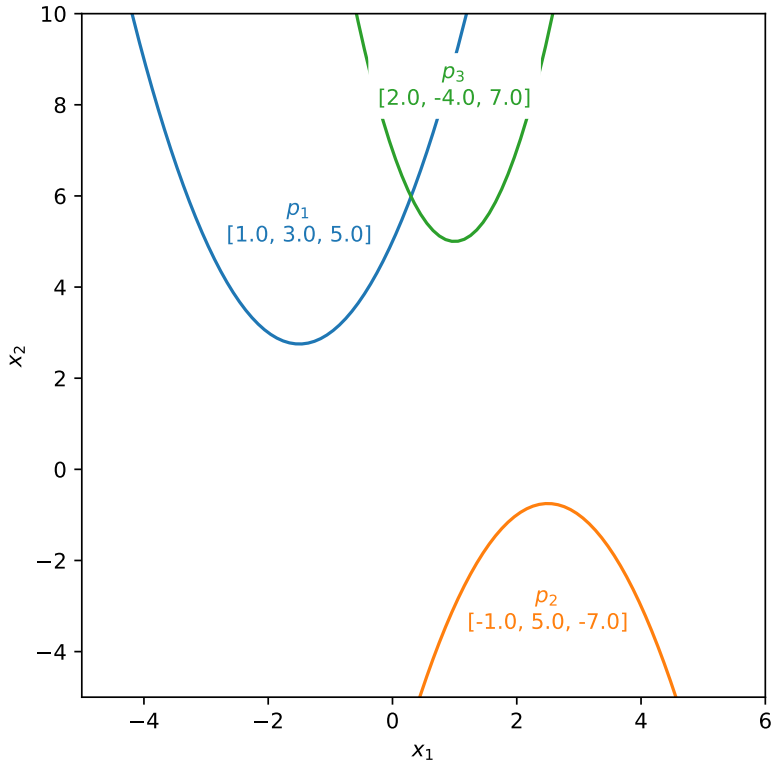

Figure 1: Three parabolas. The numbers in brackets indicate the coefficients of the respective polynomials.

Part c. Use your loss function with a suitable scipy optimizer to find a solution. (You should no longer feel the need to set your weighting factors to 1.) Plot your solution on a figure that looks like Figure 1.

Paper Submission

(Turn in your solution, including code, as a PDF to Canvas.)
Points: 1

Problem 2

K23-1-10 and K23-1-12.

Paper Submission

(Turn in a PDF to Canvas.)
Points: 1

Problem 3

You will need to use the code you write in this problem for problem(s) in subsequent sets. The textbook suggests making an adjacency matrix to represent a graph on a computer. This turns out to be a horrible way to represent graphs of large systems, partly because this will be a huge $n \times n$ matrix almost completely full of zeros. A second disadvantage of using a matrix representation is that you have to pre-compute the edges of all vertices, even the ones you don't care about. For large systems, this is typically a bad idea.

Instead, we typically use adjacency as a list, defining the adjacency function Adj of a vertex A as the set of its neigbhors:

$$\text{Adj}(A) = \{E, K, L\}$$

which in this example, means that vertex A has connections to E, K, and L. In an object-oriented code like python, it is simplest to make a `Vertex` class that has a `get_children` command to define the edges; this command can either return a stored list or calculate it "on the fly".

```
class Vertex:
    def __init__(self, name):
        self.name = name
    def get_children(self):
        # Problem-specific code goes here.
```

In its simplest implementation, the `get_children` method could read from a dictionary like that below. For example, this means that vertex A has directional links to verteces B, D, and G; since, for example, we see no link back from B to A we can infer this describes a digraph.

```
links = {
        'A': ['B', 'D', 'G'],
        'B': ['E', 'G', 'H'],
        'C': ['A', 'H', 'I'],
        'D': ['F'],
        'E': ['H', 'A', 'C'],
        'F': ['G', 'I'],
        'G': ['C'],
        'H': ['A', 'E'],
        'I': ['C', 'J']
        }
```

Part a. Fill in the missing code above to create a Vertex class, that when successful should behave like:

```
>>> v = Vertex('F')
>>> print(v.get_children())
['G', 'I']
```

Submission

Label: hw5_3a
Points: 1
Class initialization variables: `name` (a string)

Class method(s): `get_children` (which takes no arguments and returns a list of strings)

Part b. In a problem next week, we are going to try to find the shortest forward distance between any two pages on the Brown Engineering web domain[1]; that is, if you start on one page, what is the minimum number of times you need to click to reach another specified page. Here, we will set up the graph tools to enable you to do that.

We want to find all the links in a web page. From your web browser, you should in general be able to click an option to see the web page's raw html[2]; you should take a look at the html of any Brown Engineering page to know what we are dealing with. In python, a simple way to request the content of a website works as follows (try it!):

```
import requests
page = requests.get('https://www.brown.edu/academics/engineering/about')
print(page.content)
```

To find all the links in this block of text, you *could* write a script to read through each line and find things that look like URLs. However, you can bet that a common task like this has a pre-packaged solution in python, and you should use those when possible to make life easier. In this case, you can find the links with

```
from lxml import html
tree = html.fromstring(page.content)
links = tree.xpath('//a/@href')
```

If you examine the list, you will notice there are two types of links: (a) those that start with http:// and (b) those that don't. We want to catch both types, so we want to extract from this list anything that starts with:

```
http://www.brown.edu/academics/engineering/
https://www.brown.edu/academics/engineering/
/academics/engineering/
```

Write a function that takes in a list of links and returns a new list which contains only the Brown Engineering links (in the same order).

Submission

Label: hw5_3b
Points: 0.5
Input variables: `links` (a list of strings)
Output: a list of strings

Part c. Write a function that takes in a web address and returns links found on that page which point to a Brown Engineering site.

[1] `https://www.brown.edu/academics/engineering/`
[2] In Firefox, it is Tools > Web Developer > Page Source, for example.

Submission

Label: hw5_3c
Points: 0.5
Input variables: `website` (a string)
Output: a list of strings

Part d. Modify your `Vertex` class, so that the `name` is the complete web address and whose `get_children` method returns a list of links found on that page which point to a Brown engineering site.

Submission

Label: hw5_3d
Points: 1
Class initialization variables: `name` (a string)
Class method(s): `get_children` (which takes no arguments and returns a list of strings)

Problem 4

The graph described by the earlier `links` dictionary is simple enough to be visualized, which can help you understand if you network is fully connected and which verteces are dead ends or hard to reach. Not surprisingly, you can find a python package to help you draw these: Daft[3], which you should be able to easily install with

```
pip3 install --user daft
```

Create a suitably laid-out figure showing all the vertices and directional connections in the `links` dictionary. Systematic solutions—such as using your `Vertex` class and laying an arbitrary number of vertices out along a circle—are encouraged!

Paper Submission

(Turn in both your code and the figure as a PDF to Canvas.)
Points: 1

[3] `http://daft-pgm.org/`