CS111 F21 Homework 9

Michael Glushchenko, 9403890

December 7, 2021

1 Newton's Method

Here's a function I wrote for this question:

```
def newtons_method_sqrt(value, x0):
    # print the value we are estimating, the correct result,
    print("\n-------------------------------------------------------\n",\
          "Estimating the square root of",\
          value, " starting at ", x0,\
          ":\n-------------------------------------------------------\n")
    # we do this for every tolerance
    for i in range(len(tolerances)):
        correct_xk = np.sqrt(value)
        precision = num_of_decimals[i] # only show digits that matter given the tolerance
        # precision = 15 # show 15 decimal places
        data = []       # a data list that'll hold our findings
        xk, k = x0, 0   # set the initial xk and counter variables
        # every iteration, we will add a row of
        # information to the data array
        while(tolerances[i] < abs(round(xk, num_of_decimals[i]) - correct_xk)):
            # record the [k, kx, and kx**2] of this step
            data.append([k, xk, xk**2.0])
            # take the iterative step
            k += 1
            xk = (0.5)*(xk + value/xk)
        # record the last iteration's information
        data.append([k, xk, xk**2])
        # set the floatfmt string using tolerance and
        # print the table for every tolerance for the given value
        my_floatfmt = "." + str(precision) + "f"
        print("tolerance =", tolerances[i], "\n",\
                tabulate(data, headers=["k", "xk", "xk**2", "correct xk"],\
                tablefmt="grid", floatfmt=my_floatfmt),\
                "\nsteps taken:", k, "\n")
    # to let us know we are done estimating
    # the value for each necessary tolerance
    print("Done estimating the value", value)
```

I also set the tolerances array according to the question asking for 4, 8, and 16 digits of significance:

```
tolerances = [1e-3, 1e-7, 1e-15]    # precision to 4, 8, and 16 digits
num_of_decimals = [3, 7, 15]
```

Now, I hope to simply give my function

```
newtons_method_sqrt(value, x0)
```

the values I need to test (4 and 2), and the starting value of x_0, and I should get my tabulated data printed to the console.

1.1 Computing $\sqrt{4}$ starting at $x_0 = 1$

Running the following line of code

```
newtons_method_sqrt(4.0, 1.0)
```

I get the following output:

```
------------------------------------------------------------
 Estimating the square root of 4.0  starting at  1.0 :
------------------------------------------------------------

tolerance = 0.001
 +-----+--------+----------+
|   k |     xk |    xk**2 |
+=====+========+==========+
|   0 | 1.000  |    1.000 |
+-----+--------+----------+
|   1 | 2.500  |    6.250 |
+-----+--------+----------+
|   2 | 2.050  |    4.202 |
+-----+--------+----------+
|   3 | 2.001  |    4.002 |
+-----+--------+----------+
steps taken: 3

tolerance = 1e-07
 +-----+-----------+-----------+
|   k |        xk |     xk**2 |
+=====+===========+===========+
|   0 | 1.0000000 | 1.0000000 |
+-----+-----------+-----------+
|   1 | 2.5000000 | 6.2500000 |
+-----+-----------+-----------+
|   2 | 2.0500000 | 4.2025000 |
+-----+-----------+-----------+
|   3 | 2.0006098 | 4.0024394 |
+-----+-----------+-----------+
|   4 | 2.0000001 | 4.0000004 |
+-----+-----------+-----------+
steps taken: 4

tolerance = 1e-15
 +-----+-------------------+-------------------+
|   k |                xk |             xk**2 |
+=====+===================+===================+
|   0 | 1.000000000000000 | 1.000000000000000 |
+-----+-------------------+-------------------+
|   1 | 2.500000000000000 | 6.250000000000000 |
+-----+-------------------+-------------------+
|   2 | 2.050000000000000 | 4.202500000000000 |
+-----+-------------------+-------------------+
|   3 | 2.000609756097561 | 4.002439396192742 |
+-----+-------------------+-------------------+
|   4 | 2.000000092922295 | 4.000000371689188 |
+-----+-------------------+-------------------+
|   5 | 2.000000000000002 | 4.000000000000009 |
+-----+-------------------+-------------------+
|   6 | 2.000000000000000 | 4.000000000000000 |
+-----+-------------------+-------------------+
steps taken: 6

Done estimating the value 4.0
```

I would like to comment on the last row of tolerance $1e-3$ and tolerance $1e-7$ tables: it displays 2.001 and 2.0000001 because I used such decimal formatting that rounds it up to the last displayed digit; if I were to display all 15 decimals here, the 4th and 8th digits above would show 0. I just thought displaying the precision amount of digits was logical.

1.2 Computing $\sqrt{2}$ starting at $x_0 = 1$

Running the following line of code

```
newtons_method_sqrt(2.0, 1.0)
```

I get the following output:

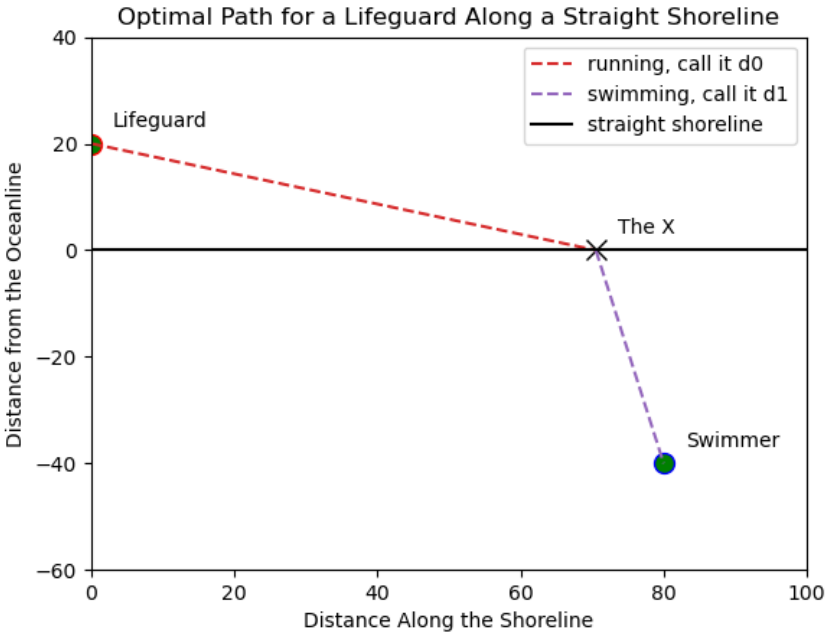
```
------------------------------------------------------------
 Estimating the square root of 2.0  starting at  1.0 :
------------------------------------------------------------

tolerance = 0.001
 +-----+-------+---------+
|   k |    xk |   xk**2 |
+=====+=======+=========+
|   0 | 1.000 |   1.000 |
+-----+-------+---------+
|   1 | 1.500 |   2.250 |
+-----+-------+---------+
|   2 | 1.417 |   2.007 |
+-----+-------+---------+
|   3 | 1.414 |   2.000 |
+-----+-------+---------+
steps taken: 3

tolerance = 1e-07
 +-----+-----------+-----------+
|   k |        xk |     xk**2 |
+=====+===========+===========+
|   0 | 1.0000000 | 1.0000000 |
+-----+-----------+-----------+
|   1 | 1.5000000 | 2.2500000 |
+-----+-----------+-----------+
|   2 | 1.4166667 | 2.0069444 |
+-----+-----------+-----------+
|   3 | 1.4142157 | 2.0000060 |
+-----+-----------+-----------+
|   4 | 1.4142136 | 2.0000000 |
+-----+-----------+-----------+
steps taken: 4

tolerance = 1e-15
 +-----+-------------------+-------------------+
|   k |                xk |             xk**2 |
+=====+===================+===================+
|   0 | 1.000000000000000 | 1.000000000000000 |
+-----+-------------------+-------------------+
|   1 | 1.500000000000000 | 2.250000000000000 |
+-----+-------------------+-------------------+
|   2 | 1.416666666666667 | 2.006944444444444 |
+-----+-------------------+-------------------+
|   3 | 1.414215686274510 | 2.000006007304882 |
+-----+-------------------+-------------------+
|   4 | 1.414213562374690 | 2.000000000004511 |
+-----+-------------------+-------------------+
|   5 | 1.414213562373095 | 2.000000000000000 |
+-----+-------------------+-------------------+
steps taken: 5

Done estimating the value 2.0
```

The number of iterations question is answered by the "steps taken:" in the output above.

2 Lifeguard & Swimmer

2.1 Straight Shoreline

Here's the problem described in **2.1**:



In the chart above, the lifeguard covers d_0 at 5 m/s, and the same lifeguard covers d_1 at 1 m/s. We want to formulate the problem to minimize the **time** the lifeguard takes to get to the swimmer. Thus, we want to minimize

$$\frac{d_0}{5} + d_1,$$

where d_0 and d_1 formulas follow from the Pythagorean:

$$d_0 = \sqrt{20^2 + x^2}, \quad d_1 = \sqrt{(80 - x)^2 + 40^2}.$$

So, in this problem, we are looking to minimize $f(x)$, where

$$f(x) = \frac{\sqrt{20^2 + x^2}}{5} + \sqrt{(80 - x)^2 + 40^2}.$$

The optimization will give us an x, where $(x, 0)$ to which the lifeguard should run to minimize the time he takes, and $f(x)$ at that point will give us the actual time the lifeguard will take to reach the swimmer by going through the point we specified. Without further ado, we go to Python to finish the problem.

Here's the code we use the define $f(x)$, and to minimize it:

```
def f(x):
    first = np.sqrt(x**2.0 + 20.0**2.0) / 5.0
    second = np.sqrt((80.0 - x)**2.0 + 40.0**2.0)
    return first + second
```

```
result = scipy.optimize.minimize(f, 80.0)
```

```
      fun: 55.73714459499687
 hess_inv: array([[41.9592]])
      jac: array([-4.7684e-07])
  message: 'Optimization terminated successfully.'
     nfev: 14
      nit: 5
     njev: 7
   status: 0
  success: True
        x: array([72.1435])
```

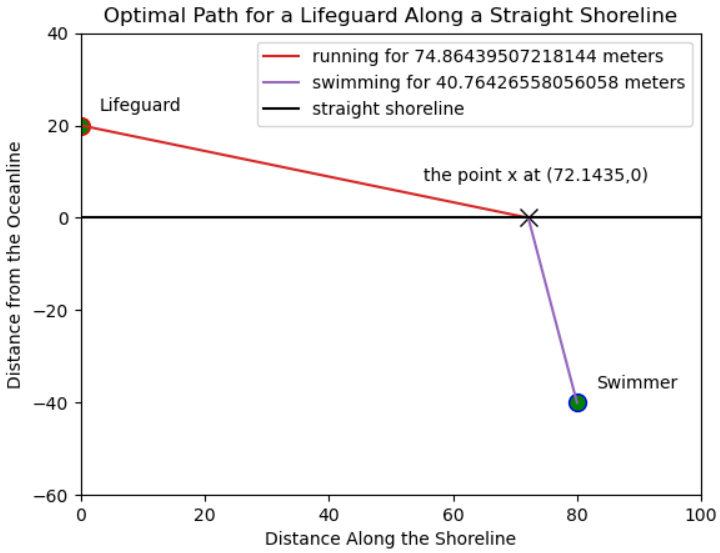
The code above tells us that the lifeguard should run to the point

$$(72.1435, 0),$$

and then swim to the swimmer from there. The result above also tells us that

$$f(72.1435) \approx 55.737 \text{ seconds},$$

the amount of time it'll take the lifeguard to get to the swimmer in this case. Now, we go ahead and plot our findings, displaying the path the lifeguard should take in order to optimally reach the swimmer "in time." Here's the plot, and the code used to make the plot follows on the next page (for fluidity of the solution).

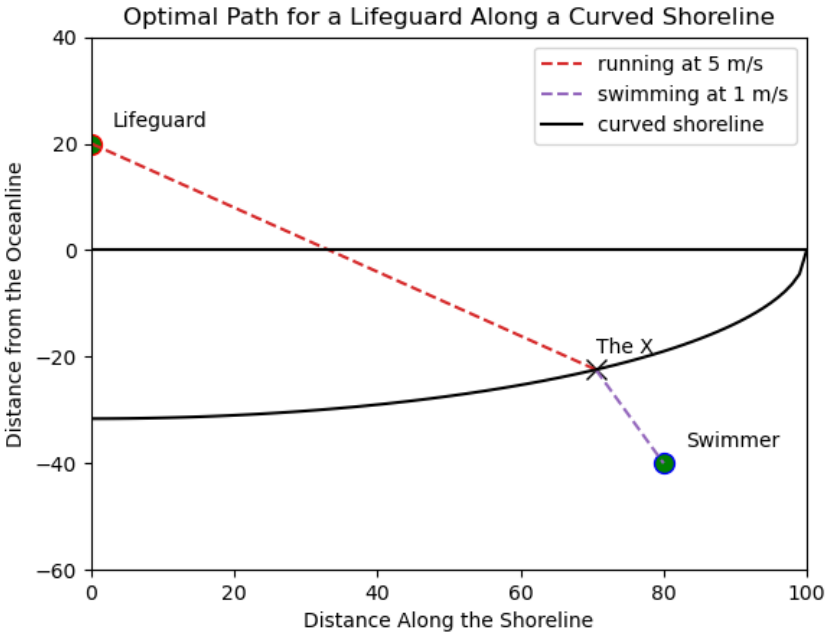


The code used to produce the above pathes and charts:

```
# modify our f to give the distances so we can plot them on the chart
def modified_f(x):
    first = np.sqrt(x**2.0 + 20.0**2.0) / 5.0
    second = np.sqrt((80.0 - x)**2.0 + 40.0**2.0)
    return (first * 5.0, second)
modified_f(72.1435) # first term is the running distance,
                    # second term is the swimming distance
plt.xlim([0, 100]) # x limits
plt.ylim([-60, 40]) # y limits
plt.xlabel( "Distance Along the Shoreline" )
plt.ylabel( "Distance from the Oceanline" )
plt.title( "Optimal Path for a Lifeguard Along a Straight Shoreline" )

# this is the lifeguard
plt.plot([0], [20], marker="o", markersize=10,\
         markeredgecolor="red", markerfacecolor="green")
plt.text(0 + 3, 20 + 3, 'Lifeguard') # label
# this is the swimmer
plt.plot([80], [-40], marker="o", markersize=10,\
         markeredgecolor="blue", markerfacecolor="green")
plt.text(80 + 3, -40 + 3, 'Swimmer') # label
# the spot the lifeguard should run to
plt.plot([result['x'][0]], [0], marker="x", markersize=10,\
         markeredgecolor="black", markerfacecolor="black")
plt.text(result['x'][0] - 17, 0 + 8, str('the point x at ('\
                                         + str(round(result['x'][0], 4))\
                                         + ',0)'))# label
# now we need a line from the lifeguard to the x
lifeguard, x = [0, result['x'][0]], [20, 0]
plt.plot(lifeguard, x, label = str('running for '\
                                   + str(modified_f(result['x'][0])[0])\
                                   + ' meters')) # plot
# and we need a line from the x to the swimmer
x, swimmer = [result['x'][0], 80], [0, -40]
plt.plot(x, swimmer, label = str('swimming for '\
                                 + str(modified_f(result['x'][0])[1])\
                                 + ' meters')) # plot
plt.axhline(0, color='black', label = 'straight shoreline') # draw the shoreline
plt.legend()
plt.show() # show the plot
```

2.2 Curved Shoreline

The problem described in **2.2** is very similar to **2.1**, except we have an equation for our shoreline. Here's the drawing which the problem describes:



Now, instead of $y = 0$ as our shoreline, we have

$$y(x) = -\sqrt{1000 - \frac{x^2}{10}}.$$

Nevertheless, the idea remains, the same, we still want to minimize

$$\frac{d_0}{5} + d_1,$$

where d_0 is the red dashed line on the chart, and d_1 is the purple dashed line on the chart. However, now our d_0 and d_1 are defined as follows:

$$d_0 = \sqrt{(y+20)^2 + x^2}, \quad d_1 = \sqrt{(80-x)^2 + (-40-y)^2}.$$

So, in **2.2**, we want to minimize $f(x)$, where

$$f(x) = \frac{\sqrt{(y(x)+20)^2 + x^2}}{5} + \sqrt{(80-x)^2 + (-40-y(x))^2}.$$

The optimization will give us an x coordinate (from which we can also calculate the y-coordinate) along the shoreline, to which the lifeguard should run. The $f(x)$ will, once again, give us the actual amount of time it takes the lifeguard to get there. The next page contains the Python used for this question.

Here, we define $y(x)$, as well as $f(x)$, and use scipy to minimize $f(x)$:

```
def y(x):
    return -np.sqrt(1000.0 - (x**2.0) / 10.0)
def f(x):
    first = np.sqrt(x**2.0 + (y(x) + 20.0)**2.0) / 5.0
    second = np.sqrt((80.0 - x)**2.0 + (-40.0 - y(x))**2.0)
    return first + second
```

```
result = scipy.optimize.minimize(f, 80.0)
result
```

```
      fun: 34.08474202400345
 hess_inv: array([[16.3784]])
      jac: array([4.7684e-07])
  message: 'Optimization terminated successfully.'
     nfev: 12
      nit: 4
     njev: 6
   status: 0
  success: True
        x: array([70.5215])
```

```
y(result['x'][0])
```

```
-22.420351359105403
```

We can now see that the lifeguard should run to the point

$$(70.5215, -22.420343524910585),$$

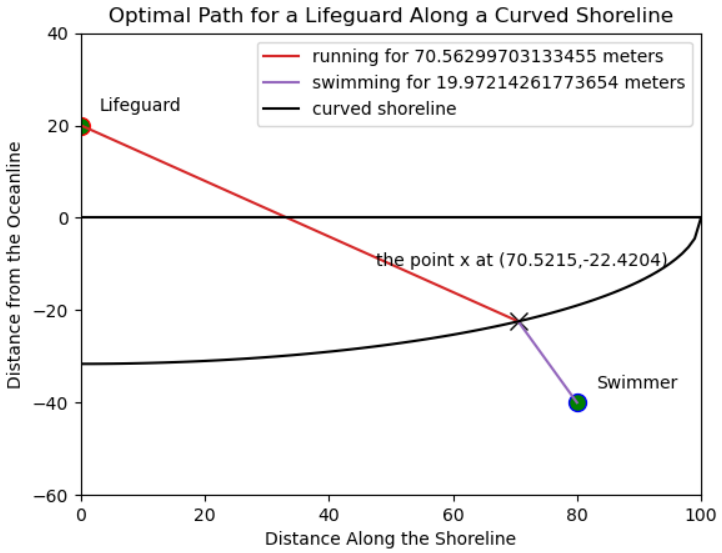
and then swim to the swimmer. The result above also tells us that

$$f(70.5215) = 34.08474202400345 \text{ seconds},$$

which is the amount of time it'll take the lifeguard to get to the swimmer in this case. We can now plot the path that the lifeguard should take to the swimmer; the next page contains the code used to make the plot (very similar to **2.1**), as well as the plot itself.

```
# modify our f to give the distances so we can plot them on the chart
def modified_f(x):
    first = np.sqrt(x**2.0 + (y(x) + 20.0)**2.0) / 5.0
    second = np.sqrt((80.0 - x)**2.0 + (-40.0 - y(x))**2.0)
    return (first * 5.0, second)

modified_f(70.5215) # first term is the running distance,
                    # second term is the swimming distance
plt.xlim([0, 100]) # x limits
plt.ylim([-60, 40]) # y limits
plt.xlabel( "Distance Along the Shoreline" )
plt.ylabel( "Distance from the Oceanline" )
plt.title( "Optimal Path for a Lifeguard Along a Curved Shoreline" )
# this is the lifeguard
plt.plot([0], [20], marker="o", markersize=10,\
         markeredgecolor="red", markerfacecolor="green")
plt.text(0 + 3, 20 + 3, 'Lifeguard') # label
# this is the swimmer
plt.plot([80], [-40], marker="o", markersize=10,\
         markeredgecolor="blue", markerfacecolor="green")
plt.text(80 + 3, -40 + 3, 'Swimmer') # label
# the spot the lifeguard should run to
plt.plot([result['x'][0]], [y(result['x'][0])], marker="x", markersize=10,\
         markeredgecolor="black", markerfacecolor="black")
plt.text(result['x'][0], y(result['x'][0]) + 3, 'The X') # label
# now we need a line from the lifeguard to the x
lifeguard, x = [0, result['x'][0]], [20, y(result['x'][0])]
plt.plot(lifeguard, x, label = str('running for '\
                                   + str(modified_f(result['x'][0])[0])\
                                   + ' meters')) # plot
# and we need a line from the x to the swimmer
x, swimmer = [result['x'][0], 80], [y(result['x'][0]), -40]
plt.plot(x, swimmer, label = str('swimming for '\
                                 + str(modified_f(result['x'][0])[1])\
                                 + ' meters')) # plot
# display the x-axis
plt.axhline(0, color= 'black')
# draw the curved shoreline
x_vals = np.linspace(0,100,100)
y_vals = np.array([y(x_vals[i]) for i in range(len(x_vals))])
plt.plot(x_vals, y_vals, color = 'black', label = 'curved shoreline')

plt.legend()
plt.show() # show the plot
```

The code produces the following plot:



3 Convex Functions

3.1 Max of Convex Functions is Convex

Given: Function f and g, both convex.
Claim: $h(x) = \max\{f(x), g(x)\}$ is convex.
Proof: We want to show the following:

$$\forall \alpha \in [0,1],\ x, y \in \mathcal{R}^n,$$
$$h(\alpha x + (1-\alpha)y) \leq \alpha h(x) + (1-\alpha)h(y).$$

We start on the left side, and make our way to the right: $\forall \alpha \in [0,1],\ x, y \in \mathcal{R}^n$,

$$h(\alpha x + (1-\alpha)y) = \max\{f(\alpha x + (1-\alpha)y), g(\alpha x + (1-\alpha)y)\}$$

Then, by convexity of f and g, we know that the above expression is

$$\leq \max\{\alpha f(x) + (1-\alpha)f(y), \alpha g(x) + (1-\alpha)g(y)\}$$
$$\leq \max\{\alpha f(x), \alpha g(x)\} + \max\{(1-\alpha)f(y), (1-\alpha)g(y)\}$$
$$= \alpha \max\{f(x), g(x)\} + (1-\alpha)\max\{f(y), g(y)\}$$
$$= \alpha h(x) + (1-\alpha)h(y).$$
$$\Rightarrow h(\alpha x + (1-\alpha)y) \leq \alpha h(x) + (1-\alpha)h(y), \quad \forall \alpha \in [0,1],\ x, y \in \mathcal{R}^n$$
$$\Rightarrow h(x) = \max\{f(x), g(x)\} \text{ is convex.}$$

3.2 Min of Convex Functions isn't Necessarily Convex

Goal: Give an example of two convex functions whose min is not convex.
Answer: Take, for example,

$$f(x) = (x-1)^2 \quad \text{and} \quad g(x) = (x+1)^2, \text{ with } h(x) = \min\{f(x), g(x)\}.$$

Now we just have to show that $\exists \alpha \in [0,1],\ x, y \in \mathcal{R}^n$, such that

$$h(\alpha x + (1-\alpha)y) > \alpha h(x) + (1-\alpha)h(y).$$

We can look at the drawing of the two parabolas to see what the graph of $h(x)$ looks like; from the graph, we choose $x = -1$ and $y = 1$ as our x and y inputs, since between them is the area of "non-convexity." Now, let's just pick $\alpha = \frac{1}{2}$. The calculation then goes like this:

$$h(\alpha x + (1-\alpha)y) = h(\frac{1}{2}(-1) + (1-\frac{1}{2})(1))$$
$$= h(-\frac{1}{2} + \frac{1}{2}) = h(0) = 1.$$

But then

$$\alpha h(x) + (1-\alpha)h(y) = \frac{1}{2}h(-1) + (1-\frac{1}{2})h(1) = 0 + 0 = 0.$$

Clearly,

$$h(\alpha x + (1-\alpha)y) > \alpha h(x) + (1-\alpha)h(y).$$