C200 Programming Assignment № 8

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

April 7, 2023

The HW is due on **Friday, April, 14 at 10:59 PM EST**. Please commit, push and submit your work to the Autograder before the deadline.

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Root Finding with Newton-Raphson

We discussed in lecture the general problem of finding roots and how ubiquitous it is. Given a function f and interval $[a, b]$ find the $x' \in [a, b]$ (presuming it exists) such that $f(x') = 0$. We call x' a root. For example, if $f(x) = x^6 - x - 1$, $f(2) = 61$ and $f(1) = -1$. Then there must be some value $x' \in [1, 2]$ such that $f(x') = 0$. Using this approach we find

$$f(1.1347305283441975) = 6.573837356116385e - 05$$

Note that the number $6.573837356116385e - 05 = 0.00006573837356116385$ is so small that for the purpose of this problem, we can think of it as approximately equal to 0.

The Newton-Raphson is an algorithm to find roots that uses a function and its derivative to find a root. It is described recursively:

$$x_0 = estimate \tag{1}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

To remind you, a derivative is a function that characterizes the way a function changes as inputs change. Equation 4 is the typical definition. Equation 5 is a common approximation of the derivative we saw in the last homework. Fig. 1 shows an iteration and the approximation of the root.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3}$$

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad h \text{ is tiny, positive} \tag{4}$$

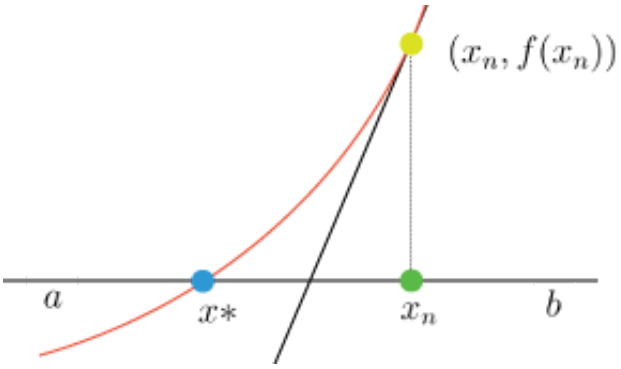

Figure 1: The root is $x*$. Our approximation x_n moves toward the root as long as we're larger than our threshold. Observe in the graphic that $f(b)$ is positive and $f(a)$ is negative insuring that there exists a root $x*$, $f(x*)$.

Another use of this approximation is to find maximal profit. Consider the following example:

$$\text{cost}(x) = 2000 - 500x \tag{5}$$

$$\text{revenue}(x) = 2000x - 10x^2 \tag{6}$$

$$\text{profit} = \text{revenue}(x) - \text{cost}(x) \tag{7}$$

You **must** observe that it is easy to write equation-7 in terms of x as: $(2000x - 10x^2) - (2000 + 500x)$. Since this is an equation, we can find the maximal profit by taking the derivative and solving for zero (finding the root). In this case, we'll find that the maximum profit occurs when $x = 75$ (however, we want to approximate the root rather than explicitly calculating the derivative and setting it to 0).

When the program is run we have the following output:

```
f(2) = 61
f(1) = -1
f(1.1347305283441975) = 6.573837356116385e-05 ~ 0.0
x = 74.99999941508389
The maximum profit is about $54250.0
```

with the plot:

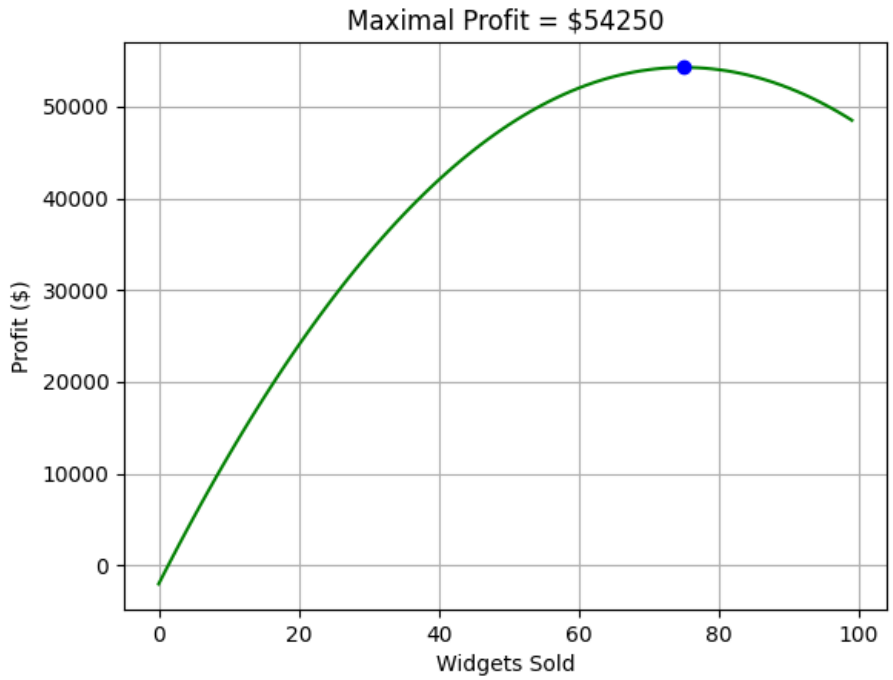

Figure 2: Using Newton-Raphson to find maximal profit $x = 75$.

Deliverables Programming Problem 1

- Complete the functions.
- You should use equation-5 for approximating the derivative.

Problem 2: Bisection

In this problem you'll implement the bisection method. The general idea is to take an interval $[a, b]$ where the root $x* \in [a, b]$ exists and continually move halfway to either the left or right. I'm using the algorithm as it's presented in *Elementary Analysis* $2nd$ ed., Atkinson. You should be excited you can interpret the pseudo-code! Here τ is our threshold and c is the approximation to $x*$.

B1 Define $c = (a + b)/2$

B2 If $b - c \leq \tau$, then accept c as the root and stop.

B3 If sign$[f(b)] \cdot$ sign$[f(c)] \leq 0$, then set $a = c$.
Otherwise, set $b = c$. Return to step **B1**.

You are free to implement this using for, while, or recursion, though my implementation is using a while loop. The sign() function should return -1 if the argument is non-positive (negative or zero) and return 1 if it's positive. We'll use the first function for problem 1.

The following code:

```
1 x = bisect(lambda x: x**6 - x - 1,1.0,2.0)
2 print(f"f({x}) = {f(x)} ~ {round(f(x),4)}")
```

has the output:

```
1 f(1.1347274780273438) = 3.4357108073646e-05 ~ 0.0
```

Programming Problem 2

- Complete the functions.
- You can implement the function using either a bounded loop, unbounded loop, or recursively

Problem 3: Secant Method

The secant method uses two numbers to approximate the root, the two numbers being endpoints of a line whose intercept approximates $x*$. The graphic shows one of the circumstances (there are two, but it's not necessary for the implementation here).

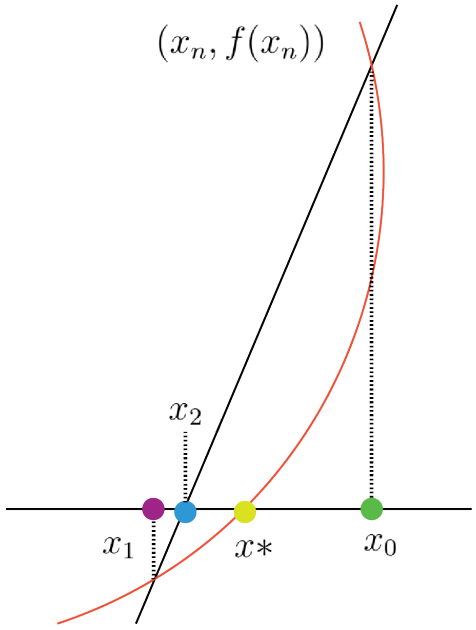

Figure 3: The root is $x*$. We use two points, x_0, x_1 to determine x_2 which is the approximation to $x*$.

The recurrence is:

$$x_0 = \text{lower bound starting value} \tag{8}$$

$$x_1 = \text{upper bound starting value} \tag{9}$$

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{10}$$

Although the recurrence looks a little intimidating, the code is actually minimal! With secant implemented, the following code

```
1 x0,x1 = 1,2
2 f = lambda x: x**6 - x - 1
3 print(f(x0),f(x1))
4 x = secant(x0,x1,f)
```

has output:

```
1 -1 61
2 f(1.1347241383964999) = -5.164002558899483e-11 ~ -0.0
```

The following pseudocode presents the algorithm which you should implement in your secant() function.

Algorithm 1 root for $f(x)$ in $[x_0, x_1]$

1: **secant**$(x_0, x_2, f, \tau = .00001)$ ▷ default for τ
2: **Output** x' where $f(x') \approx 0$
3: **while** something $> \tau$ **do** ▷ Use τ and look at f(1)–it's negative
4: update x_0 ▷ Eq. 10
5: update x_1 ▷ Eq. 10
6: **return** x_1

Deliverables Programming Problem 3

- Complete the function–pay attention to the starting values and how it affects the threshold.

Problem 4: Simpson's Rule

In this problem, we will implement Simpson's Rule–a loop that approximates integration over an interval. Suppose we want to find the value of the integral below:

$$\int_a^b f(x)\,dx \tag{11}$$

We *could* use those pesky rules of integration–who's got time for all that, right? Or, as computer scientists, we could implement virtually all integration problems. Simpson's Rule is way of approximating an integration using parabolas (See Fig. 4). For the integration, we have to pick an even number of subintervals n and sum them up.

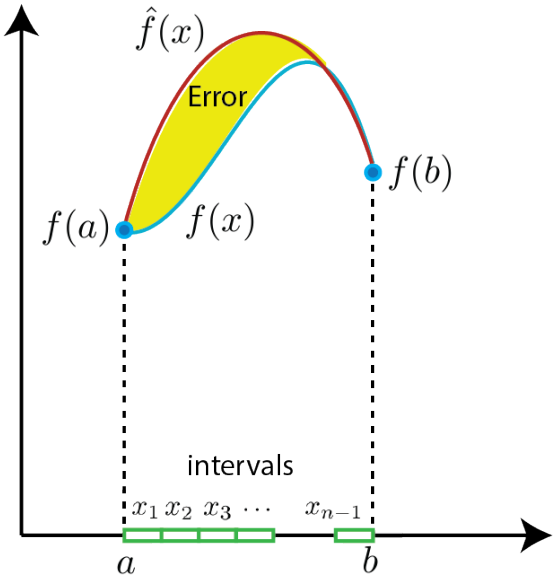

Figure 4: The function $f(x)$ integrated over a, b is approximated by $\hat{f}(x)$ using n equally sized invervals. The yellow illustrates the error of the approximation.

The *rule* is found on lines (14)-(15). Observe that when the index is odd that there is a coefficient of 4; when the index is even (excluding start and end, meaning $f(x0)$ and $f(xn)$ have no coefficients), the coefficient is 2.

$$\Delta x = \frac{b-a}{n} \tag{12}$$

$$x_i = a + i\Delta x, \qquad i = 0, 1, 2, \ldots, n-1, n \tag{13}$$

$$x_0 = a + 0\Delta x = a \tag{14}$$

$$x_n = a + n\frac{b-a}{n} = b \tag{15}$$

$$\int_a^b (x)\,dx \approx \frac{b-a}{3n}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots \tag{16}$$

$$+ 2f(x_{n-2} + 4f(x_{n-1}) + f(x_n)] \tag{17}$$

For example, the third row is the approximation to the integral

$$\int_0^\pi sin(x)\,dx = 2$$

using $n = 4$ intervals.

For those who want to verify,

$$\int_0^6 3t^2 + 1 = (\tfrac{1}{3})3t^3 + t\Big|_0^6 \tag{18}$$

$$= 216 + 6 = 222 \tag{19}$$

The following code:

```
data = [[lambda x:3*(x**2)+1, 0,6,2],[lambda x:x**2,0,5,6],
        [lambda x:math.sin(x), 0,math.pi, 4],[lambda x:1/x, 1, 11, 6]]

for d in data:
    f,a,b,n = d
    print(simpson(f,a,b,n))

area = simpson(lambda t: 3*(t**2) + 1,0,6,10)
t = np.arange(0.0, 10.0,.1)
fig,ax = plt.subplots()
s = np.arange(0,6.1,.1)
ax.plot(t, (lambda t: 3*(t**2) + 1)(t),'g')
plt.fill_between(s,(lambda t: 3*(t**2) + 1)(s))
ax.grid()
ax.set(xlabel ="x", ylabel=r"$f(x)=3x^2 + 1$",
title = r"Area under the curve $\int_0^6\,f(x)$ ~" + f"{round(area,2)}")
plt.show()
```

has output:

```
222.0
41.66666666666667
2.0045597549844207
2.4491973405016885
```

and plot:

Programming Problem 4

- Complete the functions.

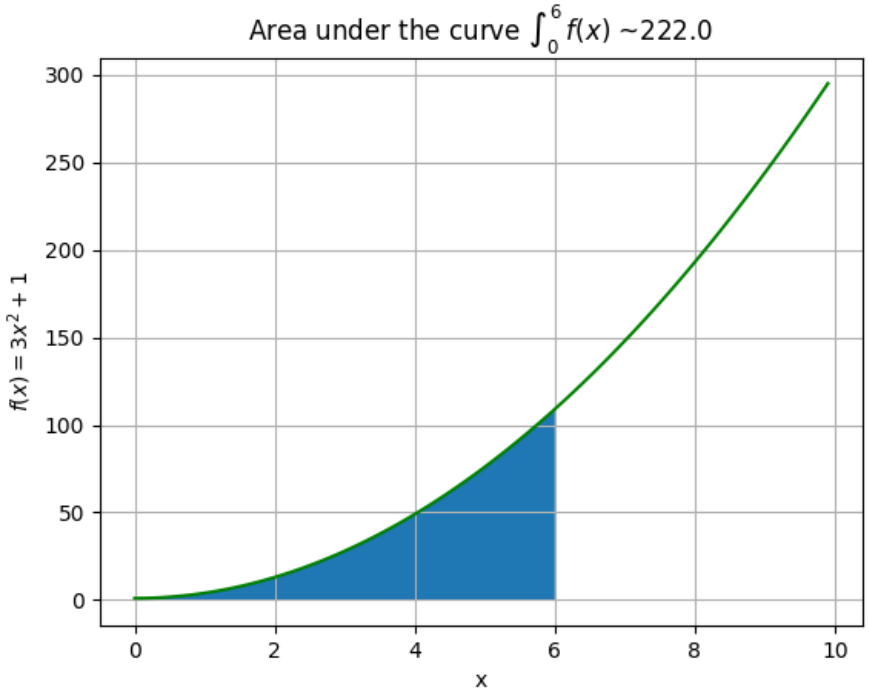

Figure 5: Using Simpson's rule to approximate an integral.

Problem 5: Central Dogma, DNA to RNA to Protein

The central dogma in biology is that DNA $\rightarrow$ RNA $\rightarrow$ protein. If you want, you can read more about it at `https://en.wikipedia.org/wiki/Central_dogma_of_molecular_biology`. In this problem you will read in a two files: The first file (amino_acids.txt) contains mapping of codons (a triplet of three DNA bases is called as **codon**) , by mapping we mean a rule that tells which specific codons will convert into a specific amino acid, and the second file (DNA.txt) contains a DNA sequence. The first file will help you to understand the mappings, and then by using the mappings, we will convert the sequence from the second file (DNA.txt) into amino acids. In essence, we will write a program to convert the DNA into protein as living organisms do! (a protein is a sequence of amino acids)

To start with, the contents of amino_acids.txt are shown below in the table.

Name	Abr.	Codons
Isoleucine	I	ATT, ATC, ATA
Leucine	L	CTT, CTC, CTA, CTG, TTA, TTG
Valine	V	GTT, GTC, GTA, GTG
Phenylalanine	F	TTT, TTC
Methionine	M	ATG
Cysteine	C	TGT, TGC
Alanine	A	GCT, GCC, GCA, GCG
Glycine	G	GGT, GGC, GGA, GGG
Proline	P	CCT, CCC, CCA, CCG
Threonine	T	ACT, ACC, ACA, ACG
Serine	S	TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine	Y	TAT, TAC
Tryptophan	W	TGG
Glutamine	Q	CAA, CAG
Asparagine	N	AAT, AAC
Histidine	H	CAT, CAC
Glutamic acid	E	GAA, GAG
Aspartic acid	D	GAT, GAC
Lysine	K	AAA, AAG
Arginine	R	CGT, CGC, CGA, CGG, AGA, AGG
Stop codons	-	TAA, TAG, TGA

The first column is the full name of the amino acid. The second column is the abbreviation as one letter initial (for the same amino acid). For the Stop_codons, we use a dash. The rightmost column are what three letters of DNA are used to make the amino acid. The amino acid Arginine has an abbreviation R. There are six codon (three bases of DNA) that code for Arginine: CGT, CGC, CGA, CGG, AGA, AGG, as can be seen in the table.

About DNA.txt: It's basically a FASTA file. Please don't be confused by the name-it's just a text file that folllows certain rules, hence the special name. It has two parts: a header (information about the DNA sequence that it contains) and the DNA sequence itself. Here's the one you'll be using (don't copy them from here, we have already pushed both files to your repositories).

```
>HSGLTH1 Human theta 1-globin gene
CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTT
TTTAGTAGAGACTAAATACCATATAGTGAACACCTA
AGACGGGGGGCCTTGGATCCAGGGCGATTCAGAGG
GCCCCGGTCGGAGCTGTCGGAGATTGAGCGCGCGC
GGTCCCGGGATCTCCGACGAGGCCCTGGACCCCCG
GGCGGCGAAGCTGCGGCGCGGCGCCCCCTGGAGGC
CGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGAT
GGCGCTGTCCGCGGAGGACCGGGCGCTGGTGCGCG
```

```
CCCTGTGGAAGAAGCTGGGCAGCAACGTCGGCGTCT
ACACGACAGAGGCCCTGGAAAGGTGCGGCAGGCTG
GGCGCCCCCGCCCCCAGGGGCCCTCCCTCCCCAAG
CCCCCCGGACGCGCCTCACCCACGTTCCTCTCGCAG
GACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGT
CAGAGCCCACGGCCAGAAGGTGGCGGACGCGCTGA
GCCTCGCCGTGGAGCGCCTGGACGACCTACCCCAC
GCGCTGTCCGCGCTGAGCCACCTGCACGCGTGCCA
GCTGCGAGTGGACCCGGCCAGCTTCCAGGTGAGCG
GCTGCCGTGCTGGGCCCCTGTCCCCGGGAGGGCCC
CGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGG
TGCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCT
CCTGGGCCACTGCCTGCTGGTAACCCTCGCCCGGCA
CTACCCCGGAGACTTCAGCCCCGCGCTGCAGGCGTC
GCTGGACAAGTTCCTGAGCCACGTTATCTCGGCGCT
GGTTTCCGAGTACCGCTGAACTGTGGGTGGGTGGCC
GCGGGATCCCCAGGCGACCTTCCCCGTGTTTGAGTA
AAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCT
CTCGAGGTCAGGACGCGAGAGGAAGGCGC
```

Though not necessary but if you want, you can read about this gene here: `https://pubmed.ncbi.nlm.nih.gov/3422341/`. In the file, the first line describes the sequence providing the name and other attributes. The remaining lines is the DNA sequence (ignore all whitespace), all lines after the header are part of the same sequence so there are no line breaks (ignore whitespaces) in the sequence.

Example: how to translate DNA into a protein

To convert from DNA to protein, we use a sequence of codons. We'll bold the protein when its translated.

Let's look at the first twelve bases: CCACTGCACTCA. Every three bases uniquely determine an amino acid.

1. Start with the first codon CCA, CCACTGCACTCA.
2. Looking at the first file we see: Proline, P, CCT, CCC, CCA, CCG. This means we can rewrite CCA as P.
3. Looking at the next codon CTG, **P**CTGCACTCA
4. We find it matches Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG. So our protein is PL.
5. The next three are CAC **PL**CACTCA.
6. The table has Histidine, H, CAT, CAC.

7. The final three are TCA. **PLH**TCA

8. This matches Serine, S, TCT, TCC, TCA, TCG, AGT, AGC.

9. The protein is **PLHS**.

If you are at the end and only have two bases, you cannot match a protein, so you ignore them. Suppose we had CCAC. We know CCA is P. Then we only have C left. We ignore it.

How to approach this problem

Our main task is to take the DNA (by reading DNA.txt) and produce a string of single letters (as shown in above example, points 1-9) that reflect the encoding. Here is one way to solve this.

1. First you'll read in the table from amino_acids.txt, and create a dictionary whose entries are:

$$aa\_d \quad = \quad \{(c_0, c_1, \ldots, c_n) : [name, letter], \ldots\}$$

where c_i is a three letter codon, $name$ is the full name of the amino acid, and $letter$ is the single letter for the amino acid. Make sure you follow the format of keys and values exactly as shown here. The function get_amino_acid() takes the file name and returns a dictionary. The dictionary is also shown below in the code listing.

2. Read in the DNA sequence, the function get_DNA() takes a file name and returns a faste data structure [header, DNA] (FASTA data structure) where header is the first line of the file DNA.txt and DNA is the DNA sequence (the sequence of A,T,G,C after the first line) (ignoring any whitespace). The read-in FASTA sequence is also shown below in the code listing.

3. The function translate() takes a FASTA data structure (that you created in step-2) and returns a string that is the translation using the dictionary (that you created in step-1). We can do a simple print to see whether our translation is the same as actual. An example of translate() function is also shown below in the code listing (you are encouraged to cross-check via pen and paper).

This code creates the dictionary and FASTA file (as a list), translates, and validates:

```
print("Dictionary")
print(aa_d)
print("FASTA file")
print(DNA_d)
print("Translations match:", str(protein == actual))

#should return "PLHS"
print(translate(["nothing", "CCACTGCACTCA"]))

```

```
#should return "D-"
print(translate(["nothing", "GACTAA"]))
```

has output:

```
Dictionary
{('ATT', 'ATC', 'ATA'): ['Isoleucine', 'I'], ('CTT', 'CTC', 'CTA', 'CTG', 'TTA', 'TTG'): ['Leucine', 'L'], ('GTT', 'GTC', 'GTA', 'GTG'): ['Valine', 'V'], ('TTT', 'TTC'): ['Phenylalanine', 'F'], ('ATG',): ['Methionine', 'M'], ('TGT', 'TGC'): ['CYSteine', 'C'], ('GCT', 'GCC', 'GCA', 'GCG'): ['Alanine', 'A'], ('GGT', 'GGC', 'GGA', 'GGG'): ['Glycine', 'G'], ('CCT', 'CCC', 'CCA', 'CCG'): ['Proline', 'P'], ('ACT', 'ACC', 'ACA', 'ACG'): ['Threonine', 'T'], ('TCT', 'TCC', 'TCA', 'TCG', 'AGT', 'AGC'): ['Serine', 'S'], ('TAT', 'TAC'): ['Tyrosine', 'Y'], ('TGG',): ['Tryptophan', 'W'], ('CAA', 'CAG'): ['Glutamine', 'Q'], ('AAT', 'AAC'): ['Asparagine', 'N'], ('CAT', 'CAC'): ['Histidine', 'H'], ('GAA', 'GAG'): ['Glutamic_acid', 'E'], ('GAT', 'GAC'): ['AsparTic acid', 'D'], ('AAA', 'AAG'): ['Lysine', 'K'], ('CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'): ['Arginine', 'R'], ('TAA', 'TAG', 'TGA'): ['Stop_codons', '-']}
FASTA file
['>HSGLTH1 Human theta 1-globin gene', 'CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTTTTTAGT
AGAGACTAAATACCATATAGTGAACACCTAAGACGGGGGGC
CTTGGATCCAGGGCGATTCAGAGGGCCCCGGTCGGAGCTGT
CGGAGATTGAGCGCGCGCGGTCCCGGGATCTCCGACGAGGC
CCTGGACCCCCGGGCGGCGAAGCTGCGGCGCGGCGCCCCCT
GGAGGCCGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGATGGCGCT
GTCCGCGGAGGACCGGGCGCTGGTGCGCGCCCTGTGGAAGA
AGCTGGGCAGCAACGTCGGCGTCTACACGACAGAGGCCCTG
GAAAGGTGCGGCAGGCTGGGCGCCCCCGCCCCCAGGGGCCC
TCCCTCCCCAAGCCCCCCGGACGCGCCTCACCCACGTTCCTC
TCGCAGGACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGTCAGAGC
CCACGGCCAGAAGGTGGCGGACGCGCTGAGCCTCGCCGTGG
AGCGCCTGGACGACCTACCCCACGCGCTGTCCGCGCTGAGC
CACCTGCACGCGTGCCAGCTGCGAGTGGACCCGGCCAGCTT
CCAGGTGAGCGGCTGCCGTGCTGGGCCCCTGTCCCCGGGAG
GGCCCCGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGGT
GCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCTCCTGGGC
CACTGCCTGCTGGTAACCCTCGCCCGGCACTACCCCGGAGAC
TTCAGCCCCGCGCTGCAGGCGTCGCTGGACAAGTTCCTGAGC
CACGTTATCTCGGCGCTGGTTTCCGAGTACCGCTGAACTGTG
GGTGGGTGGCCGCGGGATCCCCAGGCGACCTTCCCCGTGTTTG
AGTAAAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCTCTC
```

```
GAGGTCAGGACGCGAGAGGAAGGCGC']
Translations match: True
PLHS
D-
```

Deliverables Problem 5

- Carefully read the instructions in the starter file a8.py. You may not be able to run it directly in VSC, so follow the instructions in the starter code.
- Complete the functions get_DNA(), get_amino_acids() and translate() as per the specifications.
- You are allowed to use replace for space and "{'. Other uses of it will actually be more difficult.

Problem 6: Marginal Cost

When producing something, the cost typically varies. The **marginal cost** is the cost incurred by producing an additonal unit of product or service. This is the derivative in disguise. Given a function $cost(x)$, we can determine the derivative as:

$$\frac{d\,cost(s)}{dx} \approx \lambda x : \frac{cost(x+h) - cost(x-h)}{2h} \tag{20}$$

for a very small value of h. For this problem assume the cost function is:

$$cost(x) = 0.0001x^3 - 0.08x^2 + 40x + 5000 \tag{21}$$

The following code:

```
U,C = [],[]
for unit in range(200,650,50):
    U.append(unit)
    mc = round(marginal_cost(cost)(unit),0)
    C.append(mc)
    print(f"For {unit} marginal cost is {mc}")
plt.plot(U,C,'b-')
plt.plot(300,round(marginal_cost(cost)(300)),'ro')
plt.xlabel("Units of Production")
plt.ylabel("Cost $")
plt.title(r"Marginal cost Cost(x) =  $0.0001x^3 - 0.08x^2 + 40x + 5000$")
plt.show()
```

has the output:

```
For 200 marginal cost is 20.0
For 250 marginal cost is 19.0
For 300 marginal cost is 19.0
For 350 marginal cost is 21.0
For 400 marginal cost is 24.0
For 450 marginal cost is 29.0
For 500 marginal cost is 35.0
For 550 marginal cost is 43.0
For 600 marginal cost is 52.0
```

and plot:

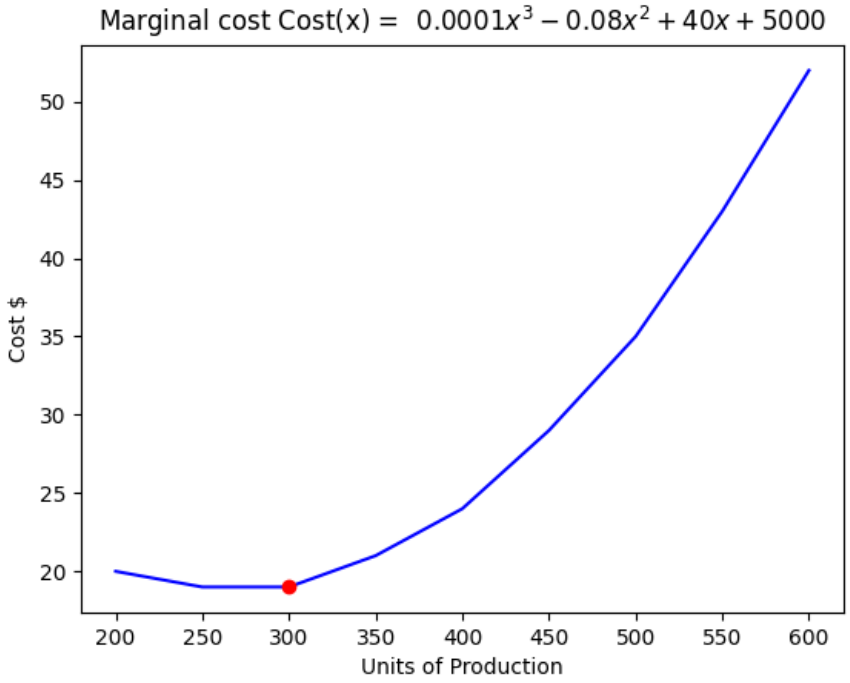

Figure 6: Marginal cost for sales 200-600 units. Observe the cost increases at 300.

Deliverables Problem 6

- Complete the lambda and marginal_cost functions. The lambda function is the example in this problem.
- Interpret your findings.

Programming Pairs

aalesh@iu.edu, hnichin@iu.edu
jacbooth@iu.edu, ss126@iu.edu
rpmahesh@iu.edu, snuthala@iu.edu
qhodgman@iu.edu, trewoo@iu.edu
dk80@iu.edu, yasmpate@iu.edu
agoldswo@iu.edu, grschenc@iu.edu
ljianghe@iu.edu, patel88@iu.edu
eddykim@iu.edu, parisbel@iu.edu
sl92@iu.edu, pvinod@iu.edu
jl263@iu.edu, ajneel@iu.edu
sjkallub@iu.edu, sousingh@iu.edu
arjbhar@iu.edu, psaggar@iu.edu
vilokale@iu.edu, lenonti@iu.edu
jasoluca@iu.edu, kvanever@iu.edu
bgabbert@iu.edu, mszczas@iu.edu
hc51@iu.edu, mw154@iu.edu
tifhuang@iu.edu, jarymeln@iu.edu
joshbrin@iu.edu, chrinayl@iu.edu
ekkumar@iu.edu, ornash@iu.edu
abdufall@iu.edu, peschulz@iu.edu
mgorals@iu.edu, yz145@iu.edu
cheng47@iu.edu, btpfeil@iu.edu
aladkhan@iu.edu, avpeda@iu.edu
stfashir@iu.edu, sowmo@iu.edu
joracobb@iu.edu, cartmull@iu.edu
actonm@iu.edu, greymonr@iu.edu
pkasarla@iu.edu, edshipp@iu.edu
joecool@iu.edu, dpepping@iu.edu
mgambett@iu.edu, leplata@iu.edu
nwbarret@iu.edu, sjvaleo@iu.edu
dilkang@iu.edu, roelreye@iu.edu
aketcha@iu.edu, bpoddut@iu.edu
moesan@iu.edu, mostrodt@iu.edu
ecastano@iu.edu, criekki@iu.edu
lsherbst@iu.edu, lmamidip@iu.edu
ogift@iu.edu, isarmoss@iu.edu
anderblm@iu.edu, arnpate@iu.edu
jacklian@iu.edu, davthorn@iu.edu
sbehman@iu.edu, macsvobo@iu.edu

sakinolu@iu.edu, caitreye@iu.edu
davgourl@iu.edu, webejack@iu.edu
sihamza@iu.edu, jyamarti@iu.edu
katzjor@iu.edu, cmulgrew@iu.edu
jonhick@iu.edu, jorzhang@iu.edu
jegillar@iu.edu, zisun@iu.edu
swcolson@iu.edu, jmissey@iu.edu
akundur@iu.edu, hvelidi@iu.edu
klongfie@iu.edu, nmonberg@iu.edu
lgflynn@iu.edu, wuyul@iu.edu
nbulgare@iu.edu, elyperry@iu.edu
siqidong@iu.edu, dannwint@iu.edu
georliu@iu.edu, dwinger@iu.edu
nokebark@iu.edu, yijwei@iu.edu
rythudso@iu.edu, nireilly@iu.edu
jackssar@iu.edu, chnico@iu.edu
wdoub@iu.edu, alescarb@iu.edu
chnbalta@iu.edu, alexschu@iu.edu
arykota@iu.edu, esmmcder@iu.edu
mdgamble@iu.edu, svaidhy@iu.edu
bradhutc@iu.edu, danwils@iu.edu
brakin@iu.edu, rair@iu.edu
coldjone@iu.edu, jvalleci@iu.edu
sgcolett@iu.edu, shethsu@iu.edu
ghyatt@iu.edu, crfmarti@iu.edu
marbelli@iu.edu, kalomart@iu.edu
dombish@iu.edu, jonvance@iu.edu
bcarl@iu.edu, nkmeade@iu.edu
jl335@iu.edu, imalhan@iu.edu
dud@iu.edu, schwani@iu.edu
aaberma@iu.edu, almoelle@iu.edu
jaybaity@iu.edu, shahneh@iu.edu
achimeba@iu.edu, zhaozhe@iu.edu
nbakken@iu.edu, bensokol@iu.edu
aadidogr@iu.edu, omilden@iu.edu
neybrito@iu.edu, sjxavier@iu.edu
tyfeldm@iu.edu, jmom@iu.edu
kbchiu@iu.edu, tshore@iu.edu
ivycai@iu.edu, saseiber@iu.edu
sfawaz@iu.edu, vthakka@iu.edu
jolawre@iu.edu, bashih@iu.edu

hollidaa@iu.edu, cjromine@iu.edu
sijatto@iu.edu, cartwolf@iu.edu
moalnass@iu.edu, sabola@iu.edu
megofolz@iu.edu, aw149@iu.edu
egillig@iu.edu, johwarre@iu.edu
abolad@iu.edu, jcpilche@iu.edu
anthoang@iu.edu, jwescott@iu.edu
aibitner@iu.edu, dy11@iu.edu