C200 Programming Assignment № 9

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

Indiana University, Bloomington, IN, USA

November 10, 2022

Introduction

In this homework, you'll work on convergence, approximation, and classes. Remember, to be successful in the class, review the slides–repeating the code yourself. Each homework problem is drawn from the lecture slides. Please do not post-pone starting the homework.

- We have resumed generating random programming partners from HW8. Pairs of programming partners for HW9 is present at the end of this document.
- **Due Date** Friday, November 18 11:00PM EST. Please submit to the Autograder (`c200.luddy.indiana.edu`) and push your code by the deadline.
- Review the lecture slides by reproducing the code.

Instructions for submitting to the Autograder (`c200.luddy.indiana.edu`)

- 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).
- Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily caught by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.
- Check 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.
- **Remove** ir-relavant library imports that are not explicitly allowed by the HW. For exmaple, if we did not use the library 'tkinter' then please don't import it in the code.

- Given that you already tried above points (bullet 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.
- 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

Root finding is a ubiquitous need. A root is a value for which a given function equals zero. In the simplest case, for a function $f : \mathbb{R} \to \mathbb{R}$,

$$f(x*) = 0 \quad x* \in [a, b] \tag{1}$$

$x*$ is called a root. The Newton-Raphson is an algorithm to find roots that uses a function $f(n)$ and its derivative. The following algorithm finds successively better approximations to a root:

$$x_0 = estimate \tag{2}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{(Df)(x_n)} \tag{3}$$

See Fig. 1 for a visualization. One weakness of this technique is that you cannot begin with an estimate that is less than the root. For the function $f(x) = x^2 - 2$ (which is simply $\sqrt{2}$, if you start with 1, you'll get an erroneous answer. On the other hand, if you begin with 100, you'll find the root. In this homework, all the estimates are greater than the root. We can employ the technique of convergence using a threshold τ to stop at an acceptable precision:

$$x_0 = estimate \tag{4}$$

$$x_{n+1} = \begin{cases} x_n - \frac{f(x_n)}{(Df)(x_n)} & f(x_n) > \tau \\ x_n & otherwise \end{cases} \tag{5}$$

The derivative makes a new function from f (in the starter code, the derivative calculation has to be implemented in the **D(f)** function).

$$(Df) = \lambda x : \frac{f(x+h) - f(x-h)}{2h} \quad h \text{ is tiny, positive} \tag{6}$$

$$\tag{7}$$

The following listing shows Newton-Raphson for f using $h = .00001$ and $\tau = 0.0001$. that will be used in the homework.

$$f(x) = x^6 - x - 1 \tag{8}$$

$$(Df) = \lambda x : \frac{f(x + .00001) - f(x - .00001)}{2(.00001)} \tag{9}$$

In Table 1 we start the the algorithm with an initial guess of 1.5 The next value to the right shows this is $f(1.5) \approx 8.890625$ which is significantly greater than our threshold. The algorithm stops on the last line (output colored in blue) which is very close to zero.

The code:

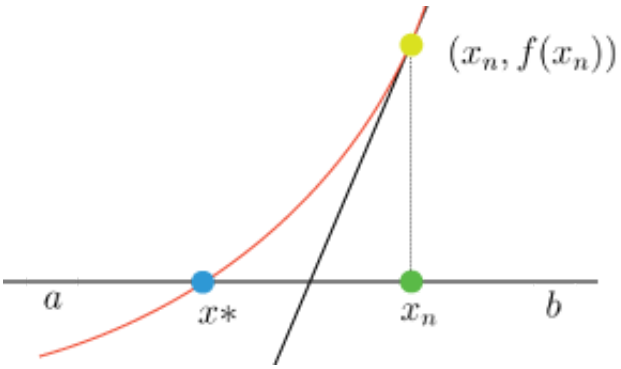

Figure 1: The root is $x*$. Our approximation x_n moves toward the root as long as we're larger than our threshold. Observe in the graphic that $f(b)$ is positive and $f(a)$ is negative insuring that there exists a root $x*$, $f(x*)$.

x	$f(x)$	$(Df)(x)$
1.5	8.890625	44.56250000703931
1.3004908836219782	2.537264144112494	21.319672162123382
1.1814804164432096	0.5384585848412315	12.812868825062827
1.1394555902943637	0.0492352512051355	10.524929241917391
1.1347776252388793	0.0005503238766089158	10.290289315828538
1.134724145316234	7.113601707686712e-08	

Table 1: Using Newton-Raphson to determine a root. The last row shows where x is sufficiently close to the root to stop.

```
1  def D(f):
2      pass
3
4  def newton(f,x,tau):
5      pass
6
7  p1 = [[lambda x:x**2 - 2, 100],[lambda x:x**6-x-1,1.5],
8        [lambda x:x**3-(100*(x**2))-x + 100,0]]
9  tau = 0.0001
10
11 for f,g in p1:
12     root = newton(f,g,tau)
```

yields

```
1  1.4142156862745259 6.007304928168367e-06
2  1.134724145316234 7.113601707686712e-08
3  100.0 0.0
```

Programming Problem 1

- Note that **D(f)** returns a lambda function, not a numeric value. While implementing **D(f)** use equation-6, and h=0.00001.
- Implement newton(f,x,tau) using the equation on line (5). You should be using D(f) in your implementation.
- You are free to implement the function recursively or with a while-loop.

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$[f(a)] =$ sign$[f(c)]$, 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.

```
1 def sign(x):
2     pass
3
4 def bisection(f,a,b,tau):
5     pass
6
7 print(bisection(lambda x:x**3-x-2,1.0,2.0,0.0001))
8 print(bisection(lambda x:x**6-x-1,1.0,2.0,.0001))
```

will produce

```
1 1.52142333984375
2 1.13470458984375
```

Programming Problem 2

- Complete the sign() and bisect() functions.

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).

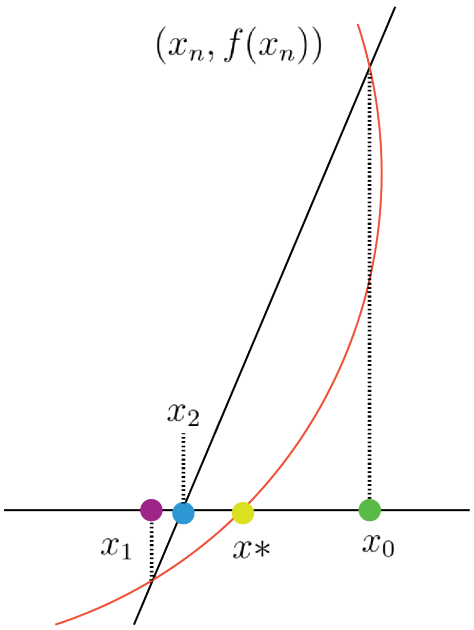

Figure 2: The root is $x*$. We use two points, x_0, x_1 to determine x_2 which is the approximation to $x*$.

The recurrence is:

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

We need to add τ, but also observe that we can potentially get negative numbers. So:

$$x_{n+1} = \begin{cases} x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} & |f(x_n)| > \tau \\ x_n & otherwise \end{cases} \tag{11}$$

The code:

```
1  def secant(f,x0,x1,tau):
2      pass
3
4  print(secant(lambda x:x**6-x-1,2.0,1.0,.0001))
5  print(secant(lambda x:x**3-x-2,1.0,2.0,0.0001))
```

produces:

```
1 1.134723645948705
2 1.5213763166697438
```

Programming Problem 3

- Complete the secant() function based on the equation on line (11).
- **Hint:** Using a while loop to implement this function will be relatively easier.

Problem 4: Simpson's Rule

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

$$\int_a^b f(x)\,dx \tag{12}$$

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. 3). For the integration, we have to pick an even number of subintervals n and sum them up.

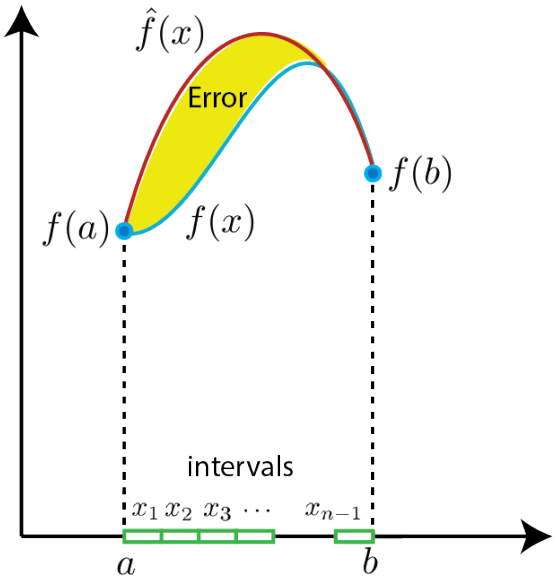

Figure 3: The function $f(x)$ integrated over a, b is approximated by $\hat{f}(x)$ using n equally sized invervals. The yellow illustrates the error of the approximation.

The *rule* is found on equations (17)-(18). Observe that when the index is odd that there is a

coefficient of 4; when the index is even (excluding start and end), the coefficient is 2.

$$\Delta x = \frac{b-a}{n} \tag{13}$$
$$x_i = a + i\Delta x, \qquad i = 0, 1, 2, \ldots, n-1, n \tag{14}$$
$$x_0 = a + 0\Delta x = a \tag{15}$$
$$x_n = a + n\frac{b-a}{n} = b \tag{16}$$
$$\int_a^b (x)\, dx \approx \frac{b-a}{3n}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots \tag{17}$$
$$+ 2f(x_{n-2} + 4f(x_{n-1}) + f(x_n)] \tag{18}$$

The code:

```
1 def simpson(f,a,b,n):
2    pass
3
4 print(simpson(lambda x: 3*(x**2)+1,0,6,2))
5 print(simpson(lambda x: (x**2),0,5,6))
6 print(simpson(lambda x: 1/x,1,11,6))
7 print(simpson(lambda x: math.sin(x),0,math.pi,10))
```

has output:

```
1 222
2 0
3 0
4 2.0001095173150043
```

Programming Problem 4

- Complete the function simpson(f,a,b,n) based on the equation provided on lines (17) and (18).

Problem 5: Permutations

Given a list of symbols [1,2,3], the set of all permutations is every distinct list that can built based on order using all the objects. The number of permutations is $n!$. The list above has $3! = 6$ permutations.

```
1 [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

The order of each object is unimportant. The simplest approach is think of building the list recursively. If we break this up into two lists,say, [1,2],[3] then we put 1 in front of 3 and behind 3: [2],[[1,3],[3,1]] (we're not doing all the possibilities–just the process). You will have to think

about this! Important thing is to note that, recursion has to be implemented locally within the function, it can't done by calling **permutation()** recursively, but by creating a local function within **permutation()** and calling that recursively. The following code:

```
def permutation(lst):
    pass

print(permutation([1,2,3]))
print(permutation([1,2,3,4]))
```

produces:

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

[[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 3, 4, 2],
 [1, 4, 2, 3], [1, 4, 3, 2], [2, 1, 3, 4], [2, 1, 4, 3],
 [2, 3, 1, 4], [2, 3, 4, 1], [2, 4, 1, 3], [2, 4, 3, 1],
 [3, 1, 2, 4], [3, 1, 4, 2], [3, 2, 1, 4], [3, 2, 4, 1],
 [3, 4, 1, 2], [3, 4, 2, 1], [4, 1, 2, 3], [4, 1, 3, 2],
 [4, 2, 1, 3], [4, 2, 3, 1], [4, 3, 1, 2], [4, 3, 2, 1]]
```

Programming Problem 5

- Complete the function permutation().

Problem 6: Fraction Class

In this problem, you'll start to build a fraction class. In this class we'll initially just want (x/y) where x, y are reduced to lowest terms. You'll have to complete the reduce(self) method that reduces terms. The most straight forward way is using the gcd algorithm we saw in this week's lecture. Note, you should reduce both self.numerator and self.denominator until they can't be reduced any further.

Note that when we print the instance (object of a class), the __str__ function will be called automatically, and instance of the class is returned, not a value. So, x, y, z, and a are all instances of the fraction class and so when we print them, the __str__ function is called, that returns the representation in the format: numerator/denominator.

The code:

```
class fraction:
    def __init__(self,numerator,denominator):
        self.numerator = numerator
        self.denominator = denominator
        self.reduce()
```

```
    def get_numerator(self):
        return self.numerator
    def get_denominator(self):
        return self.denominator
    def reduce(self):
        pass
    def __str__(self):
        return str(self.numerator) + "/" + str(self.denominator)

x = fraction(2*3*4,4*3*5)
y = fraction(2*7,7*2)
z = fraction(13,14)
a = fraction(13*2*7,14)
print(x)
print(y)
print(z)
print(a)
```

has output:

```
2/5
1/1
13/14
13/1
```

Programming Problem 6

- Complete the method reduce(self).

Pairs

C200 Programming Pairs

adamshm@iu.edu, aalesh@iu.edu
dadeyeye@iu.edu, jlcarrie@iu.edu
aaher@iu.edu, chataway@iu.edu
omakinfi@iu.edu, aibitner@iu.edu
shakolia@iu.edu, sarmayo@iu.edu
abalbert@iu.edu, sberck@iu.edu
megalbin@iu.edu, nbalacha@iu.edu
waasali@iu.edu, skalivas@iu.edu
ahmalman@iu.edu, dl61@iu.edu
anders14@iu.edu, emcgough@iu.edu
nsantoin@iu.edu, sharpky@iu.edu

jaybaity@iu.edu, mihough@iu.edu
ianbaker@iu.edu, dboecler@iu.edu
aiballou@iu.edu, skrasher@iu.edu
jabarbu@iu.edu, adhichin@iu.edu
cmbeaven@iu.edu, egshim@iu.edu
olibelch@iu.edu, csmalarz@iu.edu
evberg@iu.edu, mattcrum@iu.edu
sbi@iu.edu, comojica@iu.edu
jbilbre@iu.edu, ecaggian@iu.edu
jetblack@iu.edu, lewiserj@iu.edu
ablashe@iu.edu, nkyryk@iu.edu
pblasio@iu.edu, jgruys@iu.edu
obowcott@iu.edu, patelkus@iu.edu
sabowe@iu.edu, sunreza@iu.edu
abrandtb@iu.edu, kyeosen@iu.edu
owebrook@iu.edu, brkapla@iu.edu
browpr@iu.edu, laynicho@iu.edu
ttbrowne@iu.edu, ptstorm@iu.edu
stebutz@iu.edu, phiprice@iu.edu
petcarmi@iu.edu, elyryba@iu.edu
carcast@iu.edu, dixonjh@iu.edu
chenjunx@iu.edu, kt10@iu.edu
tchigudu@iu.edu, sampopek@iu.edu
scclotea@iu.edu, owasmith@iu.edu
jconcial@iu.edu, cgoeglei@iu.edu
lizcoro@iu.edu, halejd@iu.edu
lcosens@iu.edu, jonhick@iu.edu
giancost@iu.edu, ashmvaug@iu.edu
bcrick@iu.edu, seckardt@iu.edu
cwcrotty@iu.edu, kninnema@iu.edu
pcullum@iu.edu, ryou@iu.edu
tdearbor@iu.edu, ttsegai@iu.edu
edeporte@iu.edu, rtrujill@iu.edu
jacdick@iu.edu, mvanworm@iu.edu
adolata@iu.edu, wjduncan@iu.edu
tdonoho@iu.edu, joelna@iu.edu
maldowde@iu.edu, tanaud@iu.edu
aareads@iu.edu, jeffsung@iu.edu
ebya@iu.edu, alenmurp@iu.edu
ceifling@iu.edu, jhaile@iu.edu
augeike@iu.edu, samstuar@iu.edu

jpenrigh@iu.edu, dazamora@iu.edu
jjepps@iu.edu, etprince@iu.edu
jfahrnow@iu.edu, rvu@iu.edu
nfarhat@iu.edu, snsung@iu.edu
chafiel@iu.edu, bdzhou@iu.edu
riflemin@iu.edu, istorine@iu.edu
foxjust@iu.edu, jthach@iu.edu
magacek@iu.edu, amurli@iu.edu
landgarr@iu.edu, jurzheng@iu.edu
rghafoor@iu.edu, jtohland@iu.edu
dgodby@iu.edu, jolindse@iu.edu
goel@iu.edu, fdkussow@iu.edu
gonzavim@iu.edu, cjvanpop@iu.edu
wgranju@iu.edu, davingo@iu.edu
eg8@iu.edu, kegupta@iu.edu
mahgree@iu.edu, albperez@iu.edu
kusgupta@iu.edu, mwroark@iu.edu
dgusich@iu.edu, anuttle@iu.edu
chaleas@iu.edu, matzhang@iu.edu
hallzj@iu.edu, petersgm@iu.edu
thamed@iu.edu, nakoon@iu.edu
alehami@iu.edu, mew17@iu.edu
hardenja@iu.edu, jwa14@iu.edu
harpebr@iu.edu, howelcar@iu.edu
rilmhart@iu.edu, wodmaxim@iu.edu
brohelms@iu.edu, spletz@iu.edu
hernaga@iu.edu, mijherr@iu.edu
lohernan@iu.edu, jttrinkl@iu.edu
aubhighb@iu.edu, patevig@iu.edu
bhmung@iu.edu, jschlaef@iu.edu
jchobbs@iu.edu, jensprin@iu.edu
phoen@iu.edu, jrosebr@iu.edu
nichhoff@iu.edu, burshell@iu.edu
tahoss@iu.edu, ertrice@iu.edu
gmhowell@iu.edu, bj13@iu.edu
thuhtoo@iu.edu, oakagzi@iu.edu
leghuang@iu.edu, isclubia@iu.edu
ellhuds@iu.edu, rdkempf@iu.edu
milahusk@iu.edu, cadwinin@iu.edu
bizzo@iu.edu, zekerobe@iu.edu
mahajenk@iu.edu, antreye@iu.edu

rjjorge@iu.edu, stefschr@iu.edu
rkabra@iu.edu, abvekoes@iu.edu
nekern@iu.edu, pyahne@iu.edu
aketcha@iu.edu, aokhiria@iu.edu
vkethine@iu.edu, shnaka@iu.edu
keysa@iu.edu, askrilof@iu.edu
drewkimb@iu.edu, mr86@iu.edu
mkirolos@iu.edu, tcpatel@iu.edu
arkirt@iu.edu, chrinayl@iu.edu
daknecht@iu.edu, esisay@iu.edu
nolknies@iu.edu, sousingh@iu.edu
limingy@iu.edu, rsstarli@iu.edu
jonllam@iu.edu, kpmorse@iu.edu
joluca@iu.edu, conthom@iu.edu
lyncsara@iu.edu, bwinckle@iu.edu
vimadhav@iu.edu, ereno@iu.edu
martiro@iu.edu, ayuraiti@iu.edu
namcbrid@iu.edu, vsivabad@iu.edu
nmccarry@iu.edu, coolds@iu.edu
mccoyry@iu.edu, skmcmaho@iu.edu
kmcinto@iu.edu, zhaofan@iu.edu
gmeinerd@iu.edu, ianwhit@iu.edu
dmetodie@iu.edu, as145@iu.edu
luilmill@iu.edu, wemurray@iu.edu
benrmitc@iu.edu, kviele@iu.edu
jnmroch@iu.edu, lufayshi@iu.edu
jamundy@iu.edu, eawidema@iu.edu
sndashi@iu.edu, patel88@iu.edu
jnjeri@iu.edu, mmpettig@iu.edu
gokeefe@iu.edu, derthach@iu.edu
lanounch@iu.edu, ssetti@iu.edu
sehpark@iu.edu, rraguram@iu.edu
parksdr@iu.edu, svuppunu@iu.edu
jpascov@iu.edu, lvanjelg@iu.edu
perkcaan@iu.edu, zaschaff@iu.edu
rpogany@iu.edu, eweidne@iu.edu
jarabino@iu.edu, agvore@iu.edu
csradtke@iu.edu, kereidy@iu.edu
abramjee@iu.edu, uzrivera@iu.edu
marebey@iu.edu, tavalla@iu.edu
kreddiva@iu.edu, sjvaleo@iu.edu

kyrhod@iu.edu, rogerju@iu.edu
darisch@iu.edu, zhangjoe@iu.edu
jbromers@iu.edu, blakruss@iu.edu
nysach@iu.edu, btao@iu.edu
oschwar@iu.edu, nasodols@iu.edu
jscrogha@iu.edu, azaporo@iu.edu
rorshiel@iu.edu, branwade@iu.edu
samsieg@iu.edu, wilsdane@iu.edu
msisodiy@iu.edu, xujack@iu.edu
msmelley@iu.edu, shawwan@iu.edu
nrs5@iu.edu, notsolo@iu.edu
mysoladi@iu.edu, evmtaylo@iu.edu
johsong@iu.edu, cmw26@iu.edu
samsteim@iu.edu, jomayode@iu.edu
ntatro@iu.edu, joeywill@iu.edu
davthorn@iu.edu, zwoolley@iu.edu
avincelj@iu.edu, envu@iu.edu
nmwaltz@iu.edu, bzurbuch@iu.edu

H200 Programming Pairs

ethcarmo@iu.edu, joshprat@iu.edu
ligonza@iu.edu, zfhassan@iu.edu
nagopi@iu.edu, arangwan@iu.edu
tkefalov@iu.edu, jarenner@iu.edu
alindval@iu.edu, sturaga@iu.edu
venguyen@iu.edu, nzaerhei@iu.edu
marafoth@iu.edu, huntang@iu.edu
avreddy@iu.edu, snresch@iu.edu
lorivera@iu.edu, bbcolon@iu.edu
aktumm@iu.edu, swa5@iu.edu