BNFO 591
INTRODUCTION TO HIGH PERFORMANCE COMPUTING IN BIOINFORMATICS AND THE LIFE SCIENCES

FORTRAN HOMEWORK 2: RECURSION, SUMMATIONS, LOOPS AND ARRAYS

TARYNN M. WITTEN

ABSTRACT. The problems in this assignment involve developing programs that will use loops, recursion, summations and arrays to calculate the Fibonacci numbers F_n, the factorial function $n!$ and the gamma function $\Gamma(x)$. You will be building a `FORTRAN` program library for later use in understanding timing and optimization.

1. Overview

Sometimes you will run into situations where you have to write a program to actually calculate a particular scientific function that is not in a program library like `Gnu Scientific Program Library` or some other library. Most of the functions you might encounter will require you to use loops, recursion and arrays in order to complete the needed calculations. For example, the gamma function $\Gamma(x)$, which is related to $n!$, is an important function because it is how we know that 0!=1 (among other things). The Gamma function $\Gamma(x)$ is defined by the following scary-looking equation:

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt \tag{1}$$

where x is any real-valued number on the interval $(-\infty, \infty)$. In the case where x is an integer $n = 0, 1, 2, 3, 4 \ldots ...$, it turns out that $\Gamma(n+1) = n!$. But the question remains, how do we solve the general integral in equation(1) for any value of x. For example, it is possible to show (mathematically) that $\Gamma(\frac{1}{2}) = \sqrt{\pi}$. But that's a unique situation. What if your value of x isn't nice. For example, suppose you wanted to know the value of $\Gamma(e)$? How do you actually write a program to calculate this function?

As part of this assignment, we are going to look at two different ways to compute the function $\Gamma(x)$. In fact, there are two major ways. One way is to approximate the act of integrating using an integration approximation rule (we will talk about this ... *do not be afraid*) and the other is to approximate the function $\Gamma(x)$ itself using an infinite series or infinite product rule. For example, it can be shown (see `https://en.wikipedia.org/wiki/Gamma_function`) that

Date: September 16, 2015.

$$\Gamma(x) = \frac{1}{x} \prod_{n=1}^{\infty} \frac{\left(1 + \frac{1}{n})\right)^x}{1 + \frac{x}{n}} \tag{2}$$

Take a breath. You do not have to know where this comes from. Looks pretty scary but it really isn't. We will talk about it before you have to do it. In the meantime, if we equate equations (1) and (2) we obtain the following final equation which relates the integral to an infinite product.

$$\Gamma(x) = \int_0^{\infty} e^{-t} t^{x-1} dt = \frac{1}{x} \prod_{n=1}^{\infty} \frac{\left(1 + \frac{1}{n})\right)^x}{1 + \frac{x}{n}} \tag{3}$$

So, in summary, we can approximate the integral using an integration approximation rule (we will discuss this idea in a bit) or we can use the relationship we have just demonstrated in equation (3).

The programs that you will be writing in this assignment involve various important components of `FORTRAN` needed to write a scientific program like we have just discussed. And you will be needing them later when we talk about such things as timing and optimization. Consequently, over the next few assignments, you will be building a library of `FORTRAN` programs that you will return to use at a later point in this course. For now, just write code that works. That is, code that gives the correct answer. Don't try to write beautiful code. But do remember to document your code.

2. Your Assignment

You will be given until the beginning of class on Tuesday, 29 September 2015 to complete the following homework assignment. *No late assignments will be accepted.* You will be writing a number of programs. So here are the overarching rules.

(1) Each homework problem must be separately **STAPLED** together. No staple, no grade.
(2) Each homework must have your name, the program name (like prime number), the page number and the date on every page so that if pages get separated I know whose page belongs to whom and in what order.
(3) The complete homework problem set must be bound together with a binder clip (no staples, no paperclips).
(4) Each assignment must contain a printout of your solution `FORTRAN` program, the requested results and any supplementary work (in that order).
(5) All code must be documented.
(6) You must hand in a bibliography of any resources you used to help you do the problem. This can include books, websites, etc. There is no shame in using resources as long as you didn't just copy a program that was already on the net or in a book. **SHAME ON YOU** if you did! **You won't learn if you don't attempt to do the problems by yourself**.
(7) Don't try to be efficient in your programming. Just write code that works. We will talk about being efficient and about timing and optimization in a bit. Just get it done. However, do remember item (5) above.
(8) Make sure you have copies of your code on both Stampede and Compile and make sure that your code runs on both of them.

3. Iteration and Recursion: Loops and Arrays

3.1. **Loops & Arrays: PROGRAM 1 - Calculating the First m Prime Numbers.** In this programming section, you will be using, along with any other `FORTRAN` statements, loops and arrays to write a program to compute a list of the first m prime numbers as follows. REMEMBER: A prime number is a number that is divisible only by itself and the number one.

- The program should use the `PARAMETER` statement to define the value of m. For now you can assume that the value is $m = 100$.
- As each prime number is found, you must store the number in an array called `PRIMES`
- Once you have completed finding the first m prime numbers, add a set of statements that will write the prime numbers to an output file called `my_prime.out` .
- Save your program on both machines.
- Next, add a set of `FORTRAN` statements that will print out the first 10 prime numbers to demonstrate that you actually solved the problem. You may supply a screen capture image to demonstrate the compile, link, execute and output of your program on both Stampede and Compile. Make sure each is labeled appropriately.
- Make sure you follow the homework rules above.

3.2. **Loops & Arrays: PROGRAM 2 - Calculating $n!$.** You are going to write a program that computes $n!$ in two ways. Remember, $n! = 1 \times 2 \times 3 \ldots \times (n-1) \times n$. By the way, using our product notation that we used in equation (2) above, we could express $n!$ as

$$n! = \prod_{j=1}^{n} j = 1 \times 2 \times 3 \ldots \times (n-1) \times n \tag{4}$$

So see, it isn't as scary as you thought. It's just a nice way to compress things. Now, back to the assignment. Your factorial program should progress as follows:

- The program should ask the user to supply an integer value n for which the user wants to compute the factorial.
- It is given that $0! = 1$ and $1!=1$. You may use these as givens in your program.
- The program should read the value of n from the screen and compute the value of $n!$ as follows:
  (1) The first program section simply multiplies the numbers from 1 to n as given in the definition of $n!$ above in equation (4).
  (2) The second program section utilizes the relationship that $(n+1)! = (n+1) \times n!$. This called recursive computation.
  (3) These two program sections must be independent of each other in that you cannot use the results from one to compute the results from the other.
- Once you have verified that your program computes the factorials correctly, remove the screen interaction portion and modify your program to compute the factorial values of the numbers from 1 to m where m is a predefined parameter.

- The program should use the `PARAMETER` statement to define the value of m. For now you can assume that the value is $m = 100$.
- As each factorial number is found, you must store the number in an array called `FACTORIALS`
- Once you have completed finding the first m factorials, add a set of statements that will write the factorials to an output file called `my_factorials.out` . You should output two columns, n and $n!$.
- Next, add a set of `FORTRAN` statements that will print out the first 10 factorial numbers to demonstrate that you actually solved the problem. You may supply a screen capture image to demonstrate the compile, link, execute and output of your program on both Stampede and Compile. Make sure each is labeled appropriately.
- Make sure you follow the homework rules above.

3.3. **Loops & Arrays: PROGRAM 3 - Calculating the Fibonacci Numbers F_n.** The Fibonacci numbers are defined by at two-step recursion relationship $F_{n+1} = F_n + F_{n-1}$ where $F_0 = 1$ and $F_1 = 1$ are givens. Your job is to write a program that computes F_n as follows:

- The program should ask the user to supply an integer value n for which the user wants to compute the F_n.
- It is given that $F_0 = 1$ and $F_1 = 1$. You may use these as givens in your program.
- The program should read the value of n from the screen and compute the value of F_n .
- Once you have verified that your program computes the Fibonacci numbers correctly, remove the screen interaction portion and modify your program to compute the Fibonacci values F_n from 1 to m where m is defined in the next step.
- The program should use the `PARAMETER` statement to define the value of m. For now you can assume that the value is $m = 100$.
- As each F_n number is found, you must store the number in an array called `FIBONACCI`
- Once you have completed finding the first m Fibonacci numbers, add a set of statements that will write the Fibonacci numbers to an output file called `my_fibonacci.out` .
- Save your program on both machines.
- Next, add a set of `FORTRAN` statements that will print out the first 10 Fibonacci numbers to demonstrate that you actually solved the problem. You may supply a screen capture image to demonstrate the compile, link, execute and output of your program on both Stampede and Compile. Make sure each is labeled appropriately.
- Make sure you follow the homework rules above.

3.4. **Loops & Arrays: PROGRAM 4 - Computing the Gamma Function $\Gamma(x)$.** As we discussed earlier, the Gamma function $\Gamma(x)$ is an important mathematical function that is related to the factorial function $n!$ when the variable x is an integer n. In this section, you are going to do a number of calculations and computations based upon $\Gamma(x)$.

We begin this portion of the homework as follows. You are to write a `FORTRAN` program to compute $\Gamma(x)$ in two different ways:

- The program should ask the user to supply an integer value x for which the user wants to compute $\Gamma(x)$.
- The program should read the value of x from the screen and compute the value of $\Gamma(x)$ as follows:
  (1) The first program section simply uses the equation(2)above where we substitute the value of x for the value of t in the equation.
  (2) To verify that your program is working, you will have to test it against a known result. We know one already. For that, you will need the machine value of π. You can obtain this as follows. You can include a statement in your program as follows: `PI=4.0*ATAN(1.0)`. Of course, you could also get fancy and write the following in your `DECLARATION` segment `REAL, PARAMETER :: PI = 4 * atan(1.0)` as well.
  (3) You can verify that your program is working by letting $x = \frac{1}{2}$. Your result should be $\Gamma(\frac{1}{2}) = \sqrt{\pi}$. Does it make a difference if you let $x = \frac{1}{2}$ where you don't use a decimal point vs. $x = 0.5$? If you want to further test the accuracy of your program, you can use the following formula:

$$\Gamma(n + \frac{1}{2}) = \frac{1 \times 3 \times 5 \times 7 \times \ldots \times (2n-1)}{2^n}\sqrt{\pi} \tag{5}$$

  to supply other values of n to your program and see if your program computes correctly.
  (4) But Dr. Witten, how do we know if it is correct? Well, that's a great question. How about we say that if the difference between the value of machine $\sqrt{\pi}$ and your calculated value of $\Gamma(\frac{1}{2})$ is less that 0.00001, the we will stop multiplying terms. Your job is to figure out how many terms you will need to obtain this difference.
  (5) Once you have verified that your program is producing the correct values, alter it so that it computes the values of $\Gamma(x)$in increments of 0.01 from $x = -4.5$ to $x = 4.5$. Write the values of x and $\Gamma(x)$ as two columns to a file called `my_gamma.out`.
  (6) Using Gnuplot you are to plot the function $\Gamma(x)$ *vs.* x in increments of 0.01 from $x = -4.5$ to $x = 4.5$. Compare your result to that illustrated in Figure [1]. Of course, please make sure that your plot is appropriately labeled. How you do this is up to you. You can do it incrementally or you can be really geeky and write a Bash script to do it all. Remember the homework rules here.
  (7) You are now going to add a second section to your program. The second program section utilizes the a numerical approximation to an integral called Simpson's rule. Appendix [1] of this paper discusses the basics behind how you perform the computation.

(8) These two program sections must be independent of each other in that you cannot use the results from one to compute the results from the other.

- Write a program to implement Equation (11) in `FORTRAN`.
- Your program should use either the `FUNCTION` or the `SUBROUTINE` statement to implement the actual function being integrated.
- Obviously, you cannot have an upper limit of ∞ so you will need to figure out how big a number you have to put there in order to obtain an accurate answer for $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ using Simpson's rule. Use the same accuracy criteria we used above.
- Once you have verified that your program computes the $\Gamma(\frac{1}{2})$ correctly, use equation (5) and compute $\Gamma(n + \frac{1}{2})$ for $n = 0, 1, 2 \ldots 10$ and output your answers to a new file called `my_gammaplusahalf.out`. Note that you may have to change your upper limit b in order to get an accurate answer for each of the the calculations.
- Save your program on both machines.
- You may supply a screen capture image to demonstrate the compile, link, execute and output of your program on both Stampede and Compile. Make sure each is labeled appropriately.
- Make sure you follow the homework rules above.

4. Closing Thoughts

You may earn extra credit point, up to 10 points per programming assignment as follows.

4.1. **Bonus Points For:**

(1) If you repeat the above assignment and hand in your results in another programming language.

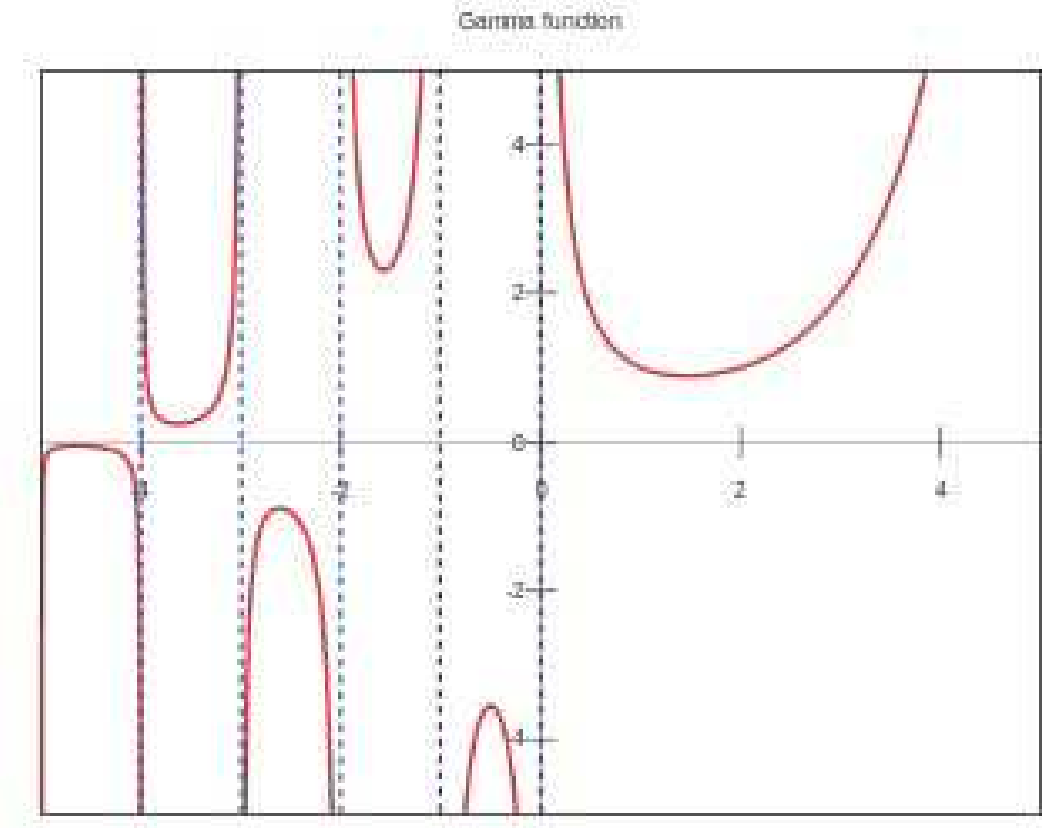

FIGURE 1. Plot of the Gamma Function $\Gamma(x)$ vs. x

5. Appendix 1

5.1. **Numerical Integration: Rectangular and Trapezoidal Approximations.** If you remember, the integral of a function $f(x)$ represents the area under the curve. If we express it as follows

$$\int_a^b f(x)dx \tag{6}$$

then we are saying that we want the area under the curve $f(x)$ between the points $x = a$ and $x = b$. This is illustrated in Figure [2]

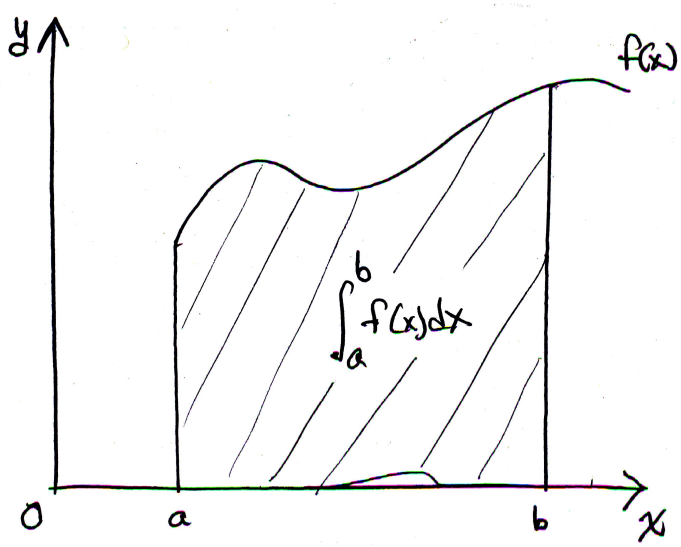

Figure 2. Illustration of the area under the curve of a function $f(x)$ between the points $x = a$ and $x = b$

Oftentimes, there is no actual mathematical formula to calculate the integral. That is, we cannot look it up in a table of integrals. And so, we have to calculate it numerically. To do this we use integral approximation formulae.

There are many ways to approximate an integral. Each way has it's pros and cons. And each way has different degrees of accuracy. We are going to talk about one particular method called Simpson's rule. Now we have already said that the integral is the area under the curve. So, one way to calculate the integral is to somehow approximate the area under the curve. And the easiest way, so that we may illustrate our point, is to draw vertical boxes of a fixed width. Let's look at one very simple way to do this. It's called the left hand, rectangular approximation scheme. We illustrate this in Figure [3].

First we divide the interval $[a, b]$ into n sub-divisions of width $\Delta x = \frac{b-a}{n}$. It follows that the division points along the x-axis would be denote by $a + 0\Delta x, a + 1\Delta x, a + 2\Delta x, a + 3\Delta x \ldots a + (n-1)\Delta x$ as is illustrated in Figure [3]. Now let's compute the area of the vertical segment A_0. Clearly the width is Δx but what is the height. Since we measure height from the left, the height would be $f(a+0\Delta x)$.

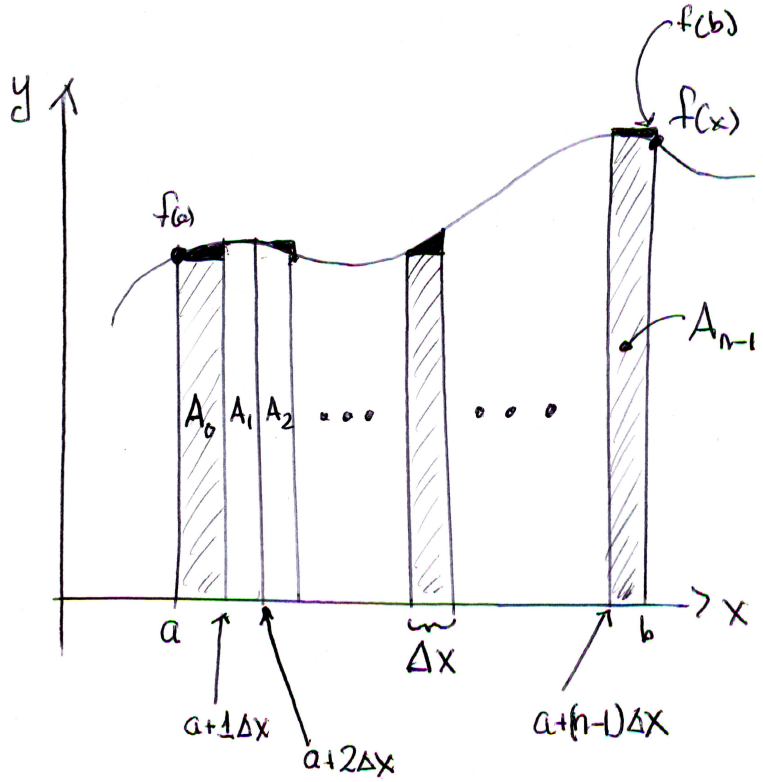

FIGURE 3. Left hand rectangular approximation scheme to find the area of the curve under $f(x)$ between the points $x = a$ and $x = b$

And it follows that the area A_0 would be $A_0 = f(a+0\Delta x)\Delta x$. Now what is the area of the segment A_1? Width is still Δx. Height, from the left, is $f(a+1\Delta x)$ so the area is $A_1 = f(a+1\Delta x)\Delta x$. Similarly, $A_2 = f(a+2\Delta x)\Delta x$. With a bit of thought you can see that the A_j area is just $A_j = f(a+j\Delta x)\Delta x$. Convince yourself that the last area $A_{n-1} = f(a+(n-1)\Delta x)\Delta x$. Now, adding all of these together, we obtain the following integral approximation formula (left hand rectangular approximation formula)

(7)
$$\int_a^b f(x)dx \approx f(a+0\Delta x)\Delta x+f(a+1\Delta x)\Delta x+f(a+2\Delta x)\Delta x+\ldots+f(a+(n-1)\Delta x)\Delta x$$

or more simply

$$\int_a^b f(x)dx \approx \sum_{j=1}^{n-1} f(a+j\Delta x)\Delta x \tag{8}$$

As you can see from the dark areas above and below the curve at the top of each rectangle, this is not always a great approximation, especially when the curve $f(x)$ varies a great deal. So what's another way to do the approximation. Well, you could do the rectangles from the right (which is called the right hand rectangular integral approximation) but it has the same problem as the left hand version. So, how could we improve the approximation. We, we could do the approximation using a trapezoid as is illustrated in Figure [4].

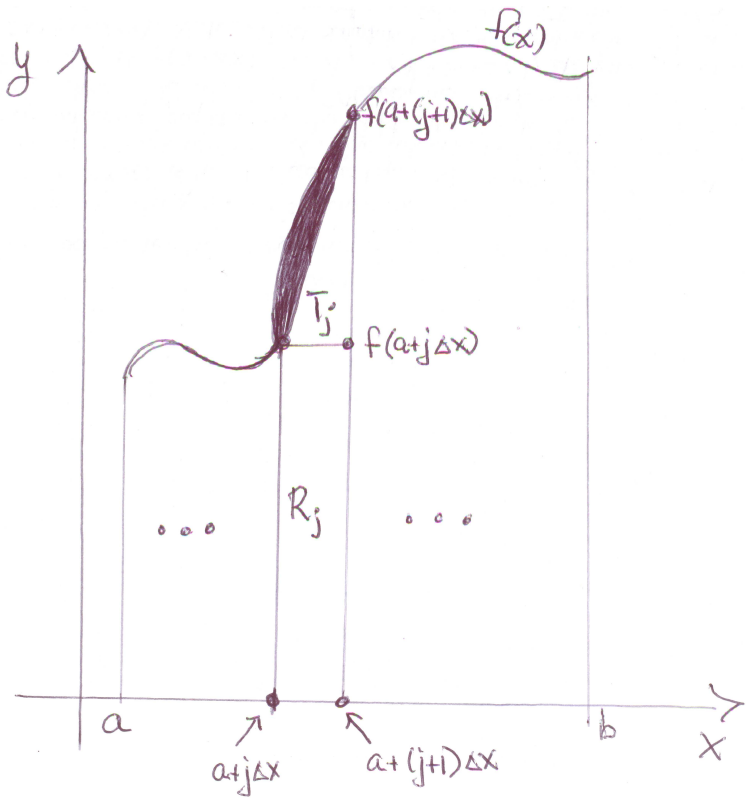

FIGURE 4. Trapezoidal approximation scheme to find the area of the curve under $f(x)$ between the points $x = a$ and $x = b$

Clearly, the trapezoid works well if the segments of the curve $f(x)$ are sort of linear along the triangular approximation. In this illustration, I have tried to give you an example of what a problem would be if the curve were nonlinear along an approximation segment. So, what is the area A_j of the approximating trapezoid? Obviously, we can see that it is the sum of the area of the left hand approximating rectangle R_j and the triangular portion T_j. How to we calculate those? You have already calculated R_j. It is just what we called A_j previously when we did the left hand approximation. All we need is the triangular area. Remembering that the area of a triangle is $\frac{1}{2}bh$ we may observe that $b = \Delta x$. Examining Figure[4] we can see that the height h of the triangle T_j is just given by $h = f(a+(j+1)\Delta x) - f(a+j\Delta x$.

Consequently, the area of the trapezoidal approximation element is just $A_j = T_j + R_j$ which is given by $A_j = f(a + j\Delta x)\Delta x + \frac{1}{2}\left[f(a + (j + 1)\Delta x) - f(a + j\Delta x\right]\Delta x$. After a bunch of canceling of terms, it is possible to show that this formula can be considerably simplified to the following

(9)
$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}\left[f(a + 0\Delta x) + 2f(a + 1\Delta x) + \ldots + 2f(a + (n - 1)\Delta x) + f(a + n\Delta x)\right]$$

or more simply expressed by the following formula

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}\left[f(a + 0\Delta x) + f(a + n\Delta x) + 2\sum_{j=1}^{n-1} f(a + j\Delta x)\right] \tag{10}$$

At this point you probably are saying enough is enough already! How do you write a program to do this? So, here is a sample of my code to help you think about writing a program. As always, I don't guarantee it is error free. However, I think that it is. So always test my examples before just implementing them.

```
PROGRAM Trapezoidal_Rule
!
! This program is a sample program to compute the value of an integral of a function f(x)
! on the interval [a,b]. It uses the trapezoidal approximation to compute the integral's
! final value
!
     IMPLICIT NONE ! Remove implicit FORTRAN variable definitions
     REAL, EXTERNAL :: f ! Define a function f(x)
     INTEGER :: N ! Number of panel subdivisions
     REAL :: A,B ! Endpoints of the integral interval [a,b]
     REAL :: INTEGRAL ! Value of the integration result
     INTEGER :: I ! Counter
     REAL :: DX ! Width of the panel subdivision
     REAL :: X ! x-value being used to calculation f(x)
!
! Tell the user to input the values of the number of panel subdivisions N, and the
! endpoints of the interval "a" and "b"
!
     PRINT *, 'Please input the number of panel subdivisions'
     READ *,N
     PRINT *, 'Please input the left endpoint of the integration interval'
     READ *, A
     PRINT *, 'Please input the right endpoint of the integration interval'
     READ *,B
!
! Compute the partition width DX = (B-A)/N
!
     DX=(B-A)/FLOAT(N) ! What problem might I have had if I didn't use the FLOAT command?
!
```

```
! Begin the initialization process to compute the integral value INTEGRAL
!
     INTEGRAL = f(A)+f(B) ! Refer to the equation for the trapezoidal rule in the text
!
! Now compute the values of the remaining points
! NOTE: There are many ways to code this part
!
     DO I=1,N-1 ! Use a loop to do the summing in the same formula above
     X=A+I*DX ! Increment the value on the x-axis of the panel element
!
! I am now going to compute the remaining part of the integral formula in the above
! equation. So I am going to recursively add all of the values of f(X) together as
! given above. Note that I am doing something that I just mentioned in class. So
! that I can make the program more general, that is, be able to alter my function
! f(x) without hard coding it into the main part of the program, I am defining a
! function f (see REAL, EXTERNAL :: f statement above). This function sits outside
! as a sort of subroutine.
!

     INTEGRAL = INTEGRAL + 2*f(X)
     END DO
!
! Complete the integral calculations in the above text formula
!
     INTEGRAL = INTEGRAL * DX/2.0
!
! Output the results
!
     PRINT *, 'The value of the integral is = ',INTEGRAL
!
END PROGRAM Trapezoidal_Rule
!
! Now, we have ended the MAIN program. But we have not yet defined the function f.
! Remember that DECLARATION statement REAL, EXTERNAL :: f ? That is telling the MAIN
! routine that there is piece of code sitting outside the main routine that is an
! external function. This way, I don't have to alter my main routine every time I
! want to change the function that I want to integrate. So we have to tell the
! MAIN routine that there is a function
!
     REAL FUNCTION f(X) ! Tell the program that this is a function statement
     IMPLICIT NONE ! Remove the implicit FORTRAN definitions
     REAL X ! Define X to be a variable of TYPE REAL
     f(X)= X*X+3*X ! Put whatever function you want here
     END FUNCTION f ! End the function
!
! This is called a function subroutine
```

5.2. **Numerical Integration: Simpson's Rule.** Okay, so we have noticed that when the function $f(x)$ behaves smoothly and in a reasonably linear fashion over

each little segment Δx of our approximating panel subdivision, then the trapezoidal approximation works reasonably well. But, as you see in Figure[4] this is not true if the function is curving about. So what do we use as the next approximating panel shape? What's the next function that is more complex than a line? Well, how about a parabola? And that is where Simpson's Rule is derived from. We won't go into it now. I am going to give you the formula to use for your homework.

As before, divide your interval up into n intervals of width Δx. To simplify the notation so that we don't have the complicated formulae we have above, let's define $x_j = a + j\Delta x$. Then the given integral approximation is of the form

(11)
$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 4f(x_{n-1}) + f(x_n)\right]$$

This is the formula you will need to use for your homework assignment on Program - 4, the Gamma function $\Gamma(x)$.

As a quick aside, we have not discussed the errors of approximation for each of these methods. There are ways to actually compute the error bounds on each of these different approximation schemes. However, that is for a course in Numerical Analysis. So, if you think you are going to go on to do real scientific programming and calculations, you should definitely take a course in Numerical Analysis. That's where you learn how really messed up you can get if you don't pay attention to how computers perform their calculations.

6. Appendix 2

6.1. **Adding Date and Time Stamps to Code - BONUS POINTS.** You have been asked to put your name and date on your homework. And you can even put the time you ran the program if you want. Wouldn't it be nice if you could have a little `SUBROUTINE` that you could call at the beginning of each program that would output this information for you. Then all you have to do is figure out how you want to place it in your homework. Life `FUNCTION` statements, `SUBROUTINE` statements go after the `END PROGRAM NAME` statement as they are `EXTERNAL` to the main program. I am going to write a program that is a `MAIN` program and you can figure out how to implement it for extra credit as a `SUBROUTINE` to your programs. You might, as you develop the subroutine also want to have it print your name and the course number BNFO 591 as well. I suggest that you go online and look up how the builtin `FORTRAN` subroutine `DATE_AND_TIME` actually works. Look at the type of arguments that it expects. Also, note the way this program outputs things. What is it actually doing and how would you modify it (1) to be a subroutine that you can use regularly with all of your programs and (2) so that it prints the header the way that you want it to? This time I am not documenting the code. You figure it out. That's why it is bonus points.

```
program print_time_and_date
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    call date_and_time(date,time,zone,values)
    call date_and_time(DATE=date,ZONE=zone)
    call date_and_time(TIME=time)
    call date_and_time(VALUES=values)
    print '(a,2x,a,2x,a)', date, time, zone
    print '(8i5)', values
end program print_time_and_date
```

Center for the Study of Biological Complexity, Virginia Commonwealth University, Richmond, VA 23284-2030 US
E-mail address: tmwitten@vcu.edu