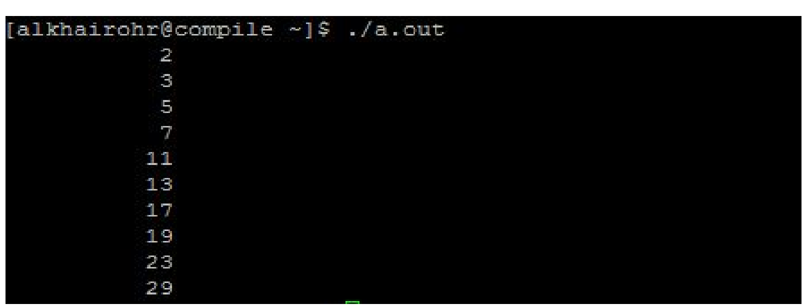
BNFO 591: High Performance Computing
Assignment II

Programs in Fortran and Python:

1. *m* Prime Numbers
2. Calculating *n*!
3.Fibonacci series
4.Gamma Function

Authors:
Hasan Alkhairo
Skyler Kuhn
Alexandrea Stylianou

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 1: Calculating the First m Prime Numbers.
! Skyler Kuhn, Hasan Alkhairo, Alex Stylianou
! BNFO591: High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program primeNum
implicit none

integer :: i,j,darklord,x, count = 0
integer, parameter :: m=100
logical :: tf
integer :: primes(1:50), h

open(unit=11,file='my_prime.out',status='old')
do 666 j=2,m,1 !This loop starts 2 and will go to m (100) and increments by 1
	tf = .true. !This flag is used to indicate whether or a j is a prime number
	do 66 x=2,j-1,1 !starting from 2 increment x by 1 to j-1. For each value x, divide J by it to
check if there is a remainder of 0.
		if (modulo(j,x) == 0) then !if J is divisble by x with no remainder then J is not a prime
number. Set flag to false and exit loop.
			!print *, j
			tf = .false. !so when this flag is set to false then this value J will not written to a
file or appended to the array primes
			EXIT !Exit 66. J will then be incremented by 1
		end if
	66 continue
	if (tf) then !if the flag is true then j is a prime number.
		!print *, j
		write(11,*) j !write J to file my_primt.out
		primes(count) = j !append j to primes
		count=count+1 !count is the index for primes.
	end if
666 continue
do 13 darklord=0,9,1 !this loop is used to print the first 10 integers in the primes array
	print *, primes(darklord)
13 continue

end program primeNum
```



Python code for Prime Problem

```
prime_num = 100
for x in range(2, prime_num+1):
  for i in range(2, x):
      if x % i == 0:
         break
  else:
      print x,
print 'Done'
```

Output:
/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 Done

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Program 2, Part I: Calculating n! without recursion (See RecursiveFactorial.f90 for recursive n!)
!Hasan Alkhairo, Skyler Kuhn, Alex Stylaniou
!BNF0 591 High Performance Computing
!Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program factorial
implicit none

integer :: i,counter=2,x,j
integer (kind=8) :: f=1
integer (kind = 8) :: factorials(0:100)
integer, parameter :: m = 100
!print *, 'Provide a number for its factorial'
!read *, n

open(unit=13,file='my_factorials.out',status='old')

factorials(0) = 1
factorials(1) = 1

do 10 j=2,m,1
	do 11 i=1,j,1
		f = f * i
	11 continue
	factorials(counter) = f
	counter = counter + 1
```

```
      f = 1
10 continue

do 12 x=0,counter,1
      !print *, factorials(x)
      write(13,*) factorials(x)
12 continue
end program factorial
```

```
[alkhairohr@compile ~]$ ./a.out
                     1
                     1
                     2
                     6
                    24
                   120
                   720
                  5040
                 40320
                362880
```

Python Code for n!:

```
n = 100
factorial = 1
list_of_factorials = []
for x in range(1,100,1): #This loop is set to 100 to get the first 100 factorials
  for y in range(1,x,1): #Lets say x is 5. y will be 1,2,3,4,5. each time it is multiplied to factorial
    factorial = factorial * y
  list_of_factorials.append(factorial) #array with all factorials
  factorial = 1 #set to 1 to restart next factorial calculation
for x in list_of_factorials: #print all factorials in array
  print x
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 2, Part II: Recursive n! Computation
! Hasan Alkhairo, Skyler Kuhn, Alex Stylaniou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Program RecursiveFactorial
implicit none
integer (kind=8) :: Factorial, n, x
integer :: counter = 1, k
integer (kind=8) :: FactorialArray(0:100)
integer, parameter :: m = 100

open(unit=666,file='RecursiveFactorial.out',status='old') ! This is the ouput file (contains 100!)
FactorialArray(0) = 1 ! Adding 0! as the first value of the Array
do 66 n=1,m,1 ! Iterates from 1 to 100
```

```
        write (666,*) Factorial(n) ! Writing to the Output file
        FactorialArray(counter) = Factorial(n) ! Appending to the Array
        counter = counter + 1 ! Needed for the Array Indices
66 continue

do 6 k=0,9,1
        x = FactorialArray(k)
        x = huge(x)
        print *, x  ! Prints the first Ten Values
6 continue

end program RecursiveFactorial

!!!!!!!!!!!!!!!!!!!!!!!!!!! FUNCTION !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Recursive Function Factorial(n) Result(illuminati)
        implicit none
        integer (kind = 8),intent(IN)::n
        integer (kind = 8) :: illuminati
        if (n==0) then   ! Base Case Breaks out of Recursion
                illuminati = 1
        else
                illuminati = n * Factorial(n-1)  ! Recursively Called the Function
                !illuminati = huge(illuminati)
        end if

end Function Factorial
```

```
[alkhairohr@compile ~]$ ./a.out
                       1
                       1
                       2
                       6
                      24
                     120
                     720
                    5040
                   40320
                  362880
```

Python program for n! *with* Recursion

```
def factorial( num ):
  if num <1:   # base case
      return 1
  else:
      return num * factorial( num - 1 )  # Recursive Call
for i in range(1,101,1 ):
   print("{}! = {}".format(i, factorial(i)))
```

Output for Recursive Factorial Function:

```
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5 /Users/nbskuhn/Desktop/Python/factorial_recursion.py
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 10333147966386144929666651337523200000000
36! = 371993326789901217467999448150835200000000
37! = 13763753091226345046315979581580902400000000
38! = 523022617466601111760007224100074291200000000
39! = 20397882081197443358640281739902897356800000000
40! = 815915283247897734345611269596115894272000000000
41! = 33452526613163807108170062053440751665152000000000
42! = 1405006117752879898543142606244511569936384000000000
```

….

……

……….

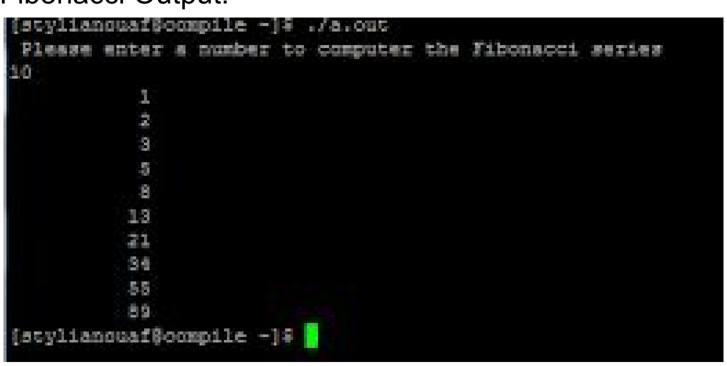
```
90! = 1485715964481761497309522733620825737885569961284688766942216863704985393094065876545992131370884059645617234469978112000000000000000000000
91! = 135200152767840296255166568759495142147586866476906677791741734597153670771559994765685283954750449427751168336768008192000000000000000000000
92! = 12438414054641307255475324325873553077577991715875414356840239582938137710983519518443046123837041347353107486982656753664000000000000000000000
93! = 1156772507081641574759205162306240436214753229576413535186142281213246807121467315215203289516844845303838996289387078090752000000000000000000000
94! = 108736615665674308027365285256786601004186803580182872307497374434045199869417927630229109214583415458560865651202385340530688000000000000000000000
95! = 10329978488239059262599702099394727095397746340117372869212250571234293987594703124871765375385424468563282236864226607350415360000000000000000000000
96! = 991677934870949689209571401541893801158183648651267795444376054838492222809091499987689476037000748982075094738965754305639874560000000000000000000000
97! = 96192759682482119853328425949563698712343813919172976158104477319333745612481875498805879175589072651261284189679678167647067832320000000000000000000000
98! = 9426890448883247745626185743057242473809693764078951663494238777294707070023223798882976159207729119823605850588608460429412647567360000000000000000000000
99! = 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
100! = 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 3, Part I: Fibonacci Computation
! Hasan Alkhairo, Skyler Kuhn, Alex Stylaniou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Screenshot of Program:

```
PROGRAM    Fibonacci

  IMPLICIT   NONE

  !declare variables
  INTEGER :: first_num, second_num, sum_of, i, fib
  ! print statement for user
  print*,'Please enter a number to computer the Fibonacci series'
  read(*,*) fib

  !initialize variables
  first_num = 0
  second_num = 1


  DO i = 1, fib, 1
    sum_of = first_num + second_num
    first_num = second_num !overwrite value for first num, this number will now be the second number
    second_num = sum_of    !overwrite the value for the second num, this number will now be the sum
                    ! when the do loop iterates to the next value the second number and the sum will now
added together

    WRITE (*,*) sum_of
  END DO
END PROGRAM Fibonacci
```

Fibonacci Output:

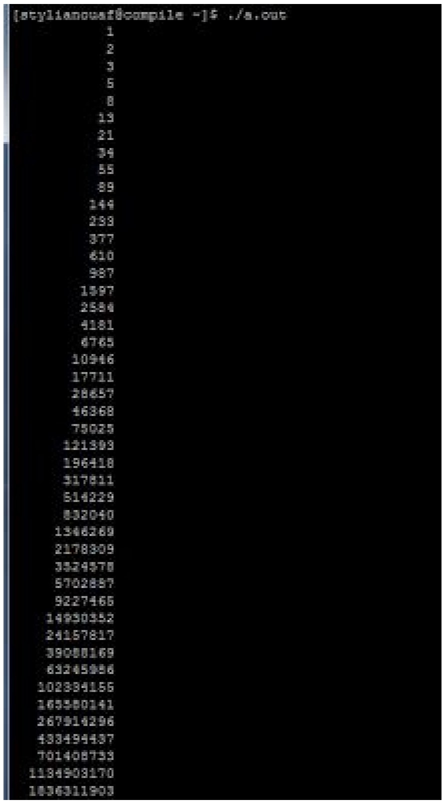


Python:

```
response = input("Please enter a number to calculate fibonacci: ")
def fib(n):
    fib_list=[]
    x, y = 0, 1
    counter = 0
    while counter < n:
        sum_of = x + y
        x = y
        y = sum_of
        fib_list.append(x)
        counter = counter + 1
    print (fib_list)
fib(int(response))
Please enter a number to calculate fibonacci: 10
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 3, Part II: Fibonacci Computation
! Hasan Alkhairo, Skyler Kuhn, Alex Stylaniou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM    Fibonacci

  IMPLICIT   NONE

  !declare variables
  INTEGER :: first_num, second_num, sum_of, i, fib, counter = 0,x
  !integer, allocatable, dimension(:) :: fib_list ! list will hold numbers
  !allocate(fib_list(fib))
  integer :: fib_list(0:100)
  !initialize variables
  first_num = 0
  second_num = 1
  fib = 100

  DO i = 1, fib, 1
    sum_of = first_num + second_num
    first_num = second_num !overwrite value for first num, this number will now be the second number
    second_num = sum_of    !overwrite the value for the second num, this number will now be the sum
                           ! when the do loop iterates to the next value the second number and the sum will now
added together
    fib_list(counter) = sum_of
    counter = counter + 1
  END DO

  do 12 x=0,counter,1
    write (*,*) fib_list(x)
  12 continue
END PROGRAM Fibonacci
```

Output (see next page):



```
[stylianouaf@compile ~]$ ./a.out
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
102334155
165580141
267914296
433494437
701408733
1134903170
1836311903
```

Python Code:

```
def fib(n):
    fib_list=[]
    x, y = 0, 1
    counter = 0
    while counter < n:
        sum_of = x + y
        x = y
        y = sum_of
        fib_list.append(x)
        counter = counter + 1
    print (fib_list)
fib(100)
```

Output:

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733, 1134903170, 1836311903, 2971215073, 4807526976, 7778742049, 12586269025, 20365011074, 32951280099, 53316291173, 86267571272, 139583862445, 225851433717, 365435296162, 591286729879, 956722026041, 1548008755920, 2504730781961, 4052739537881, ...]

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 3, Part III: Fibonacci Computation
! Hasan Alkhairo, Skyler Kuhn, Alex Stylianou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

PROGRAM     Fibonacci

  IMPLICIT   NONE
  INTEGER :: first_num, second_num,sum_of, i,counter,x
  !I am initalizing all my variables
  INTEGER,DIMENSION(100) :: fib_list            ! initializing an array
  INTEGER,PARAMETER :: m=100                    !making m a parameter, parameter will not
                                                !allow the variable to be changed
  first_num = 0
  second_num = 1
  counter = 0
  open(unit=1,file="my_fib.out",status='new')   !creating an output file

  DO i = 1, m, 1                                !do fibonacci sequence until it reaches 100
        sum_of = first_num + second_num         !add first number and second number
        first_num = second_num                  !first number is now the second number
        second_num = sum_of                     !second number is now the sum
        fib_list(counter) = sum_of              !insert the sum of the fibonacci sequence
                                                        !into the array
        counter = counter + 1                           !counter
  END DO

  DO  x=0,counter,1
        IF (x<10) THEN                                  ! homework indicates to only print
                                                        !out first 10 sequences.
        write(1,*)fib_list(x)                           !write out first 10 sequences to a list
        END IF
  END DO
  close(1)

END PROGRAM Fibonacci
```

```
[stylianouaf@compile ~]$ cat my_fib
         1
         2
         3
         5
         8
        13
        21
        34
        55
        89
```

###
Python code:

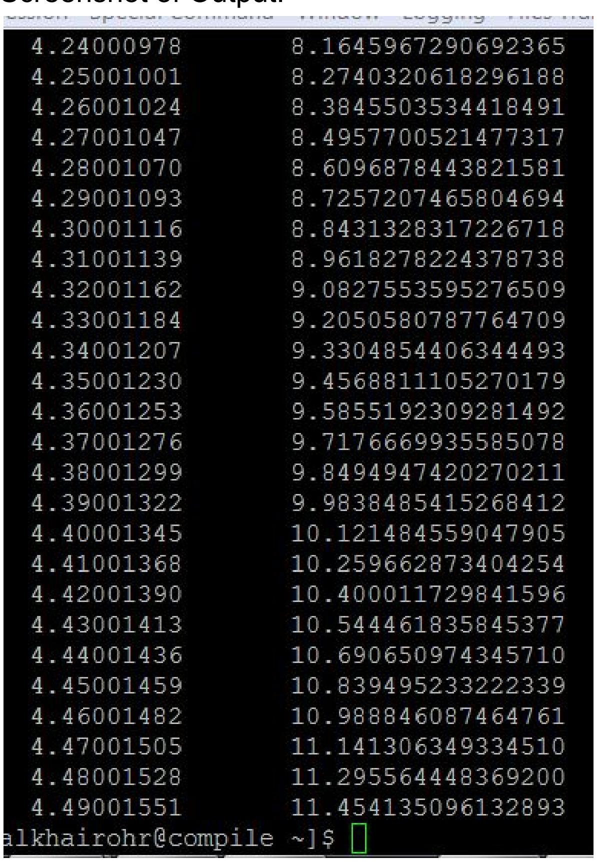
```
def fib(n):
    fib_list=[]
    x, y = 0, 1
    counter = 0
    while counter < n:
        sum_of = x + y
        x = y
        y = sum_of
        fib_list.append(x)
        counter = counter + 1
    print (fib_list)
fib(10)
```

Output; [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 4, Part I (Non-Simpson Approximation): Computing the Gamma Function
! Hasan Alkhairo, Skyler Kuhn, Alex Stylaniou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program GammaFunction

implicit none

real :: j,x = 0.5, constant,difference,percision,previous_value,i,low = -4.5
real (kind = 8) :: f
integer, parameter :: m = 100
logical :: tf

!print *, 'Provide a number for its factorial'
!read *, n

open(unit=13,file='sim_gamma.out',status='old')

tf = .true.
percision = .0000001   !Used later in the loop in place of infinity, compared to the difference of
the prev & cur val
i = 1
f = 1
previous_value = 5
do while (low.LE.4.5)! replaces x value. From -4.5 to 4.5
    do while (tf) ! will break out of loop if the difference <= 0.0000001 or the loop reaches
10,000 iterations
        f = f * ((1+(1/i))**low)/(1+(low/i))  !1:0.94,....,727:0.9999997638,728:0.9999997645
        difference = ABS(f - previous_value) !1:0.94-1 = 0.06; 728:7e-10
        previous_value = f
        i = i + 1
        if (difference.LE.percision) then !0.06 >= .0000001
            tf = .false.
            f = (1/low)*f
        end if
        if (i.GE.10000) then     ! loops 10,000 times (to aviod an infinite loop scenario)
            if (f.GE.0) then
                tf =.false.   ! break out of loop
                f = 100000    ! set f to a really high number, it is approaching infinity
            else
```

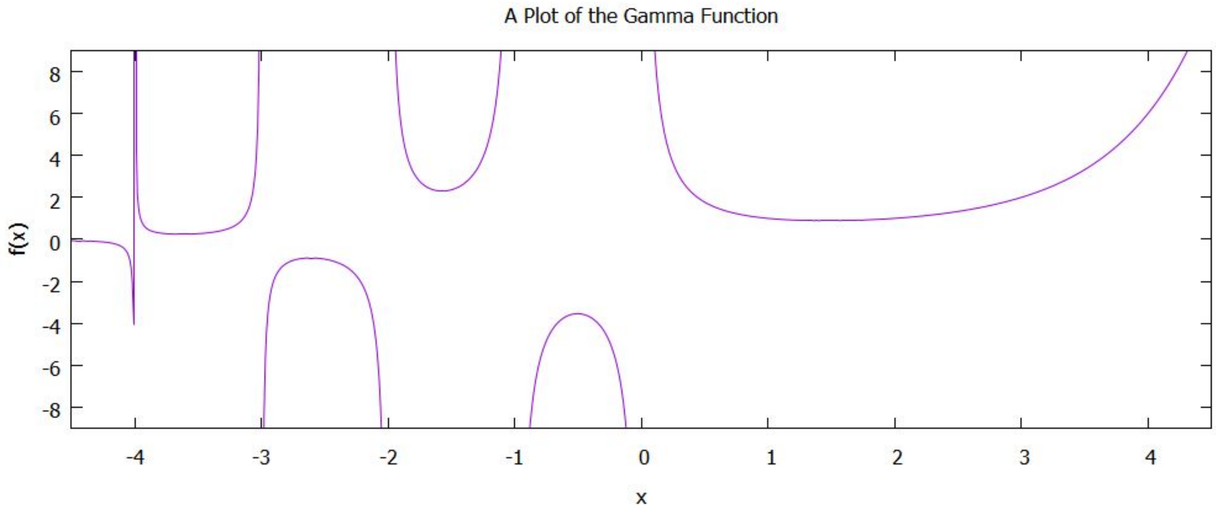
```
                    tf = .false.  ! break out of loop
                    f = -100000   ! set f to a really small_val numer, it is approaching negative
infinity
                end if
            end if
        end do
        tf = .true.
        print *, low, f
        write (13,*) low, f
        f = 1
        i = 1
        previous_value = 5
        !print *, f
        low = low + 0.01   ! increment by 0.01
end do

end program GammaFunction
```

Screenshot of Output:



```
   4.24000978        8.1645967290692365
   4.25001001        8.2740320618296188
   4.26001024        8.3845503534418491
   4.27001047        8.4957700521477317
   4.28001070        8.6096878443821581
   4.29001093        8.7257207465804694
   4.30001116        8.8431328317226718
   4.31001139        8.9618278224378738
   4.32001162        9.0827553595276509
   4.33001184        9.2050580787764709
   4.34001207        9.3304854406344493
   4.35001230        9.4568811105270179
   4.36001253        9.5855192309281492
   4.37001276        9.7176669935585078
   4.38001299        9.8494947420270211
   4.39001322        9.9838485415268412
   4.40001345        10.121484559047905
   4.41001368        10.259662873404254
   4.42001390        10.400011729841596
   4.43001413        10.544461835845377
   4.44001436        10.690650974345710
   4.45001459        10.839495233222339
   4.46001482        10.988846087464761
   4.47001505        11.141306349334510
   4.48001528        11.295564448369200
   4.49001551        11.454135096132893
alkhairohr@compile ~]$
```

GNUPlot of Gamma Program (See output above):



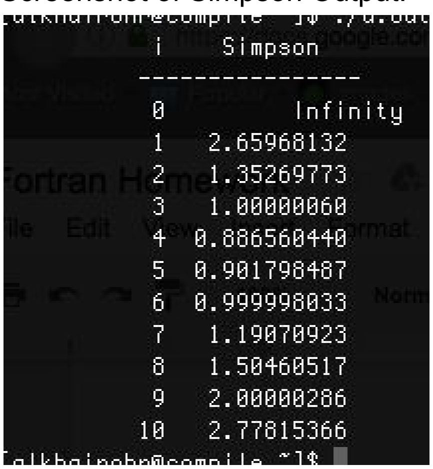
GNUPLOT Commands:

```
################################################################
# Skyler Kuhn, Alexandrea Stylaniou, Hasan Alkhairo
# BNFO 591: High Performance Computing
# Assignment 2: GNUPLOT COMMANDS
################################################################

Terminal type set to 'wxt'
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out.txt' using 1:2
         warning: Cannot find or open file "C:\Users\BCCL-USER\Desktop\my_gamma.out.txt"
         No data in plot

gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2 with lines
gnuplot> set xrange[-4.5:4.5]
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2 with lines
gnuplot> set xrange[-4.5:4.5]
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2 with lines
gnuplot> set xlabel "x"
gnuplot> set ylabel "f(x)"
gnuplot> set title "A Plot of the Gamma Function"
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2 with lines
gnuplot> plot 'C:\Users\BCCL-USER\Desktop\my_gamma.out' using 1:2 with lines title ""

gnuplot>
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Program 4, Part II (Simpson Approximation) of Gamma Function
! Hasan Alkhairo, Skyler Kuhn, Alex Stylianou
! BNFO 591 High Performance Computing
! Dr. Witten
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
program ComputeGammaSimpson
implicit none

  integer :: i
 ! Pretty Printing of Output
  print *, "         i           Simpson"
  print *, "         ----------------"
  do i=0, 10
          print *, i, sim_gamma(i/3.0)
  end do

contains

pure function intgamma(x, y) result(res)  ! Used for the infinite replacement
!pure function needed for forall statement
          real :: res
          real, intent(in) :: x, y !intent(in) is initializing input variables from the function

          res = x**(y-1.0) * exp(-x)
  end function intgamma ! end this shit

function sim_gamma(a) result(g)
          !initializing variables
          real :: g
          real, intent(in) :: a

          real, parameter :: small_val = 1.0e-4 !constant
          integer, parameter :: points = 100000

          real :: infinity, dx, p, sp(2, points), x  !sp() casts variables to single precision.
          integer :: i
          logical :: flag
          x = a !initializing input value from intgamma(x) is being set to the input value from
                    !sim_gamma(a) this is how values can be passed from one function to another

          flag = .false.
          if ( x < 1.0 ) then
```

```
                flag = .true.
            x = x + 1
            end if

            ! Calculating a replacement for Infinity
            ! Computing the Integral
            infinity = 1.0e4
            do while ( intgamma(infinity, x) > small_val )
            infinity = infinity * 10.0
            end do

            ! Simpson Approximation
            dx = infinity/real(points)
            sp = 0.0
            forall(i=1:points/2-1) sp(1, 2*i) = intgamma(2.0*(i)*dx, x)
            forall(i=1:points/2) sp(2, 2*i - 1) = intgamma((2.0*(i)-1.0)*dx, x)
            g = (intgamma(0.0, x) + 2.0*sum(sp(1,:)) + 4.0*sum(sp(2,:)) + &
            intgamma(infinity, x))*dx/3.0

            if ( flag ) g = g/a

    end function sim_gamma

end program ComputeGammaSimpson
```

Screenshot of Simpson Output:



```
  i    Simpson
-----------------
  0          Infinity
  1   2.65968132
  2   1.35269773
  3   1.00000060
  4  0.886560440
  5  0.901798487
  6  0.999998033
  7   1.19070923
  8   1.50460517
  9   2.00000286
 10   2.77815366
```