**APPM 4600 — HOMEWORK # 6**

For all homeworks, you should use Python. **Do not use** symbolic software such as Maple or Mathematica.

1. Find the constants $x_0$, $x_1$ and $c_1$ so that the quadrature formula

$$\int_0^1 f(x)dx = \frac{1}{2}f(x_0) + c_1 f(x_1)$$

has the highest possible degree of precision.

2. Consider the definite integral $\int_{-5}^{5} \frac{1}{1+s^2} ds$.

   (a) Write a code to approximate it using a composite Trapezoidal rule. To do this, partition the interval $[-5, 5]$ into equally spaced points $t_0, t_1, \ldots, t_n$.
   Write another code to approximate $\int_{-5}^{5} \frac{1}{1+s^2} ds$ using a composite Simpson's rule. To do this, partition the interval $[-5, 5]$ into equally spaced points $t_0, t_1, \ldots, t_n$ where $n = 2k$ is even. The even indexed points should be the endpoints of your subintervals.
   You may combine the two into one code that selects the desired method if you wish.

   b) Use the error estimates derived in class to choose $n$ so that

$$\left| \int_{-5}^{5} \frac{1}{1+s^2} ds - T_n \right| < 10^{-4} \quad \text{and} \quad \left| \int_{-5}^{5} \frac{1}{1+s^2} ds - S_n \right| < 10^{-4},$$

   where $T_n$ is the result of the composite Trapezoidal rule and where $S_n$ is the result of the composite Simpson's rule. Compute the actual error for both and compare with the target $10^{-4}$.

   c) Run your code with the predicted values of $n$ and compare your computed values $S_n$ and $T_n$ with that of `SCIPY`'s `quad` routine on the same problem. Run the built in quadrature twice, once with the default tolerance of $10^{-6}$ and another time with the set tolerance of $10^{-4}$. Report the number of function evaluations required in both cases and compare these to the number of function values your codes (both $S_n$ and $T_n$) required to meet the tolerance

3. Assume the error in an integration formula has the asymptotic expansion

$$I - I_n = \frac{C_1}{n\sqrt{n}} + \frac{C_2}{n^2} + \frac{C_3}{n^2\sqrt{n}} + \frac{C_4}{n^3} + \cdots$$

   Generalize the Richardson extrapolation process to obtain an estimate of $I$ with an error of order $\frac{1}{n^2\sqrt{n}}$. Assume that three values $I_n$, $I_{n/2}$ and $I_{n/4}$ have been computed.

4. The gamma function is defined by the formula

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx, \quad t > 0.$$

   Write a program to compute the value of this function from the definition using each of the following approaches:

(a) Truncate the infinite interval of integration and write a composite trapezoidal rule code to perform the numerical integration. You will need to do some experimentation or analysis to determine where to truncate the interval based upon the usual trade-offs between accuracy and efficiency. Please describe your reasoning for your choice of interval and step size for the Trapezoidal Rule. Compare the relative accuracy of this solution with the value given by the Python gamma function (`scipy.special.gamma`) at $t = 2, 4, 6, 8, 10$ (Recall $\Gamma(k) = (k-1)!$ for positive integer $k$) .

(b) Use the Matlab adaptive quadrature routine `quad` to solve the above integral on the same interval you used for part (a). Compare the accuracy of this solution with the one you obtained in part (a) at the same values of $t$. Also, compare the number of function evaluations required by the two methods.

(c) Gauss-Laguerre quadrature is designed for the interval $[0, \infty)$ and the weight function $w(x) = e^{-x}$. It is therefore ideal to use for this problem. Call the Numpy subroutine `numpy.polynomial.laguerre.laggauss` to to obtain the $n$ weights **w** and $n$ abscissae **x** for Gauss-Laguerre quadrature and use this to approximate $\Gamma(t)$. Keep in mind Gauss-Laguerre is a generalized gaussian rule of the form:

$$\int_0^\infty f(x)e^{-x}dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

1. Find the constants $x_0$, $x_1$ and $c_1$ so that the quadrature formula

$$\int_0^1 f(x)dx = \frac{1}{2}f(x_0) + c_1 f(x_1)$$

has the highest possible degree of precision.

The degree of precision of a quadratic formula is $n$ if and only if the error is zero for all polynomials of degree $k = 0, 1, \ldots, n$, but non-zero for some polynomial of degree $n+1$

To get the highest degree of precision in

$$\int_0^1 f(x)\,dx = \frac{1}{2}f(x_0) + c_1 f(x_1)$$

We have 3 unknowns; $x_0$, $c_1$, $x_1$

let $f(x) = 1$

$$\int_0^1 1\,dx = \frac{1}{2}(1) + c_1 \rightarrow c_1 = \frac{1}{2}$$

let $f(x) = x$

$$\int_0^1 x\,dx = \frac{1}{2}x_0 + \frac{1}{2}x_1 \rightarrow \frac{1}{2} = \frac{1}{2}x_0 + \frac{1}{2}x_1$$

$$1 = x_0 + x_1$$

$$x_1 = 1 - x_0$$

let $f(x) = x^2$

$$\int_0^1 x^2\,dx = \frac{1}{3} = \frac{1}{2}x_0^2 + \frac{1}{2}x_1^2 = \frac{1}{2}x_0^2 + \frac{1}{2}(1-x_0)^2$$

$$\frac{1}{3} = \frac{1}{2}x_0^2 + \frac{1}{2}(1 - 2x_0 + x_0^2) = x_0^2 - x_0 + \frac{1}{2}$$

$$0 = x_0^2 - x_0 + \frac{1}{6}$$

$$x_0 = \frac{3 \pm \sqrt{3}}{6} \qquad x_1 = 1 - \frac{3 \pm \sqrt{3}}{6}$$

Highest degree - 2

If we look at cubic $f(x) = x^3$

$$\int_0^1 x^3\,dx = \frac{1}{4} \quad \text{and} \quad \frac{1}{2}\left(\frac{3+\sqrt{3}}{6}\right)^3 + \frac{1}{2}\left(\frac{3-\sqrt{3}}{6}\right)^3 = \frac{1}{4}$$

For $n = 4$

$$\int_0^1 x^5\,dx = \frac{1}{5} \quad \text{but} \quad \frac{1}{2}\left(\frac{3+\sqrt{3}}{6}\right)^4 + \frac{1}{2}\left(\frac{3-\sqrt{3}}{6}\right)^4 \approx 0.1949$$

there is error

so the highest degree is 3 but this could just be a coincidence for the form

$$\int_0^1 f(x)\,dx = c_0 f(x_0) + c_1 f(x_1)$$

and here w/ 4 unknowns highest degree should be 4 and maybe $c_0$ just turns out to be $\frac{1}{2}$

Question 2)

The error in the composite Trapezoid method is given by

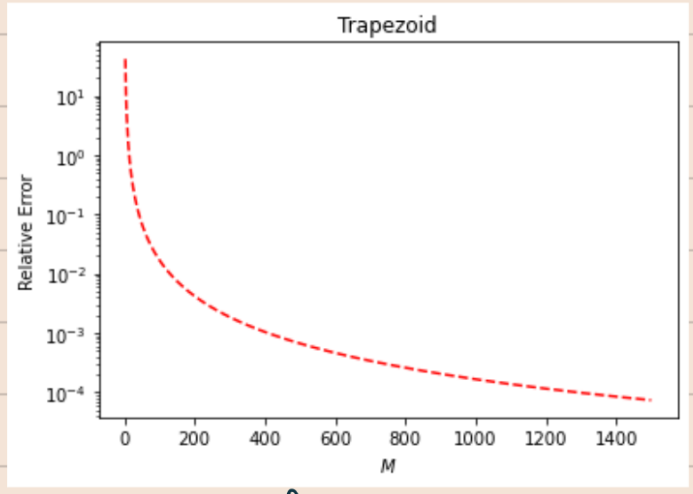
$$\left| \int_{-5}^{5} \frac{1}{1+s^2}\, ds - T_n \right| \leq \frac{b-a}{12} h^2 f''(\mu) < 10^{-4}$$

Max error when $f''(\mu)$ is max. $\rightarrow \max(f(x)) = 2 \;@\; x = 0$

For equally spaced pts. $h = \frac{b-a}{n} = \frac{10}{n}$

$$\frac{10}{12} \frac{100}{n^2} 2 < 10^{-4}$$

Solving for $n \geq \sqrt{\frac{2000}{12 \times 10^{-4}}} = 1291$




The same can be seen from the plot of the error

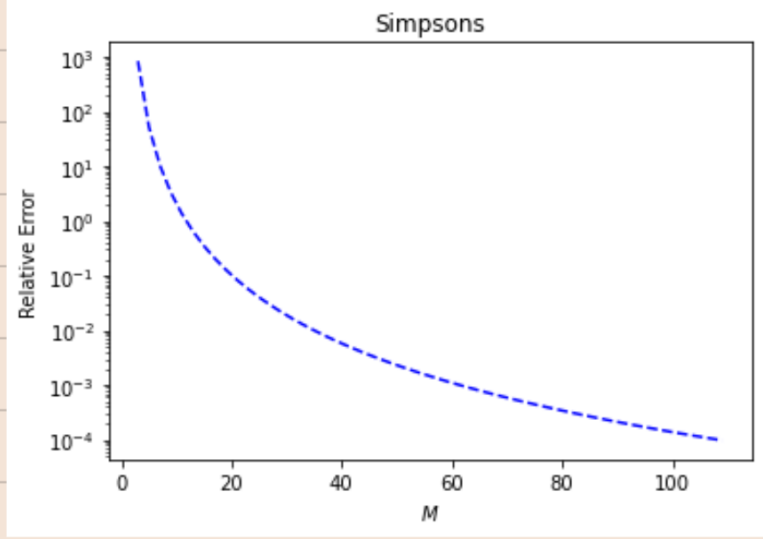
For the simpsons implementation

$$\left| \int_{-5}^{5} \frac{1}{1+s^2}\, ds - S_n \right| \leq \frac{b-a}{180} h^4 f^{(4)}(\mu) < 10^{-4}$$

$$\max(f^{(4)}(\mu)) = 24 \;@\; x = 0$$

$$\frac{10}{180} \left(\frac{10^4}{n^4}\right) (24) < 10^{-4}$$

$$n \geq \left(\frac{10^4 \cdot 24}{18 \cdot 10^{-4}}\right)^{1/4} \approx 108$$




c) Using SCIPY quad function

| Tol | quad | Trapezoid | Simpsons |
|---|---|---|---|
| $10^{-4}$ | 63 | 1291 | 108 |
| $10^{-6}$ | 147 | 12910 | 340 |

* All code files and plots on my Github

# Code for Q2

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 22 13:39:44 2024

@author: 2021n
"""

# get lgwts routine and numpy
import numpy as np;
import numpy.linalg as la;
import matplotlib.pyplot as plt

from scipy import integrate


def eval_composite_trap(M,a,b,f):
  x = np.linspace(a,b,M);
  h = (b-a)/(M-1);
  w = h*np.ones(M);
  w[0]=0.5*w[0]; w[M-1]=0.5*w[M-1];

  I_hat = np.sum(f(x)*w);
  return I_hat,x,w;

def eval_composite_simpsons(M,a,b,f):
  x = np.linspace(a,b,M);
  h = (b-a)/(M-1);
  # Explain why this defines the weights for Simpsons
  w = (h/3)*np.ones(M);
  w[1:M:2]=4*w[1:M:2];
  w[2:M-1:2]=2*w[2:M-1:2];

  I_hat = np.sum(f(x)*w);
  return I_hat,x,w;

f = lambda x: 1/(1+x**2)
fpp = lambda x: 2*(3*(x**2) - 1)/((x**2)+1)**3
fpppp = lambda x: 24*(5*(x**4) - 10*(x**2) + 1)/((x**2)+1)**5
print(fpp(0))


tol = 1e-4
a = -5
b = 5

Ms = np.arange(3,200,2); nM = len(Ms)
I_trap = np.zeros((nM,))
I_simp = np.zeros((nM,))
# storage for error
err_trap = np.zeros((nM,))
err_simp = np.zeros((nM,))
for iM in range(nM):
    M = Ms[iM]
    h = (b-a)/(M-1)
    I_trap[iM],_,_ = eval_composite_trap(M,a,b,f)
    I_simp[iM],_,_ = eval_composite_simpsons(M, a, b, f)
    err_trap[iM] = ((b-a)/12)*(h**2)*np.abs(fpp(0))
```

```
    err_simp[iM] = ((b-a)/180)*(h**4)*np.abs(fpppp(0))

    if err_simp[iM] < tol:
        print(err_trap[iM])

fig,ax = plt.subplots(1)
ax.semilogy(Ms,err_trap,'r--')
ax.set_xlabel('$M$')
ax.set_title('Trapezoid')
ax.set_ylabel('Relative Error');
plt.show()

fig,ax = plt.subplots(1)
ax.semilogy(Ms,err_simp,'b--')
ax.set_xlabel('$M$')
ax.set_title('Simpsons')
ax.set_ylabel('Relative Error');
plt.show()



I_quad,abserr,infodict = integrate.quad(f,-5,5, full_output=1, epsabs=1e-6)
print(infodict['neval'])
```

3. Assume the error in an integration formula has the asymptotic expansion

$$I - I_n = \frac{C_1}{n\sqrt{n}} + \frac{C_2}{n^2} + \frac{C_3}{n^2\sqrt{n}} + \frac{C_4}{n^3} + \cdots$$

Generalize the Richardson extrapolation process to obtain an estimate of $I$ with an error of order $\frac{1}{n^2\sqrt{n}}$. Assume that three values $I_n$, $I_{n/2}$ and $I_{n/4}$ have been computed.

$$I - I_n = \frac{C_1}{n\sqrt{n}} + \frac{C_2}{n^2} + \frac{C_3}{n^2\sqrt{n}} + \frac{C_4}{n^3} + \cdots$$

$$I - I_{n/2} = \frac{2\sqrt{2}\,C_1}{n\sqrt{n}} + \frac{4C_2}{n^2} + \frac{4\sqrt{2}\,C_3}{n^2\sqrt{n}} + \frac{8C_4}{n^3} + \cdots$$

$$I - I_{n/4} = \frac{8C_1}{n\sqrt{n}} + \frac{16C_2}{n^2} + \frac{32C_3}{n^2\sqrt{n}} + \frac{64C_4}{n^3} + \cdots$$

$$\phi_1(n) = 2\sqrt{2}(I - I_n) - (I - I_{n/2}) = (2\sqrt{2} - 4)\frac{C_2}{n^2} + (2\sqrt{2} - 4\sqrt{2})\frac{C_3}{n^2\sqrt{n}} + \cdots$$

$$\phi_2 = 8(I - I_n) - (I - I_{n/4}) = -8\frac{C_2}{n^2} + (8 - 4\sqrt{2})\frac{C_3}{n^2\sqrt{n}} + \cdots$$

$$\frac{\phi_1}{2\sqrt{2}-4} + \frac{\phi_2}{8} = \left[\frac{(2\sqrt{2} - 4\sqrt{2})}{2\sqrt{2}-4} + \frac{(8 - 4\sqrt{2})}{8}\right]\frac{C_3}{n^2\sqrt{n}} + \cdots$$

$$= \frac{2\sqrt{2}}{2\sqrt{2}-4}(I - I_n) - \frac{(I - I_{n/2})}{2\sqrt{2}-4} + I - I_n - \frac{1}{8}(I - I_{n/4})$$

$$= \left[\frac{7}{8} - \frac{2 + 3\sqrt{2}}{4}\right]I + \sqrt{2}I_n + \frac{I_{n/2}}{2\sqrt{2}-4} + \frac{I_{n/4}}{8}$$

$$I - \frac{8(7\sqrt{2}+16)I_n}{79} + \frac{I_{n/2}}{(2\sqrt{2}-4)\left(\frac{7-8\sqrt{2}}{8}\right)} + \frac{I_{n/4}}{7 - 8\sqrt{2}}$$

$$= \frac{4(-44 - 39\sqrt{2})}{79}\frac{C_3}{n^2\sqrt{n}} + \cdots$$

Now the highest order is $\frac{1}{n^2\sqrt{n}}$

We can also obtain this by solving

$$\begin{bmatrix} 1 & 1 & 1 \\ 2\sqrt{2} & 4 & 4\sqrt{2} \\ 8 & 16 & 32 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

and $a(I - I_n) + b(I - I_{n/2}) + c(I - I_{n/4}) = \frac{C_3}{n\sqrt{n}} + \cdots$

Current coeffs of $C_1$ $C_2$ $C_3$ — multiplying factor — New coeff. of $C_1$ $C_2$ $C_3$

Q4)

a) The infinite interval was truncated from $\infty$ to 100 as the function

$$f(x) = x^{t-1} e^{-x}$$

decreases really quickly and the major contributions to the integral is near 0

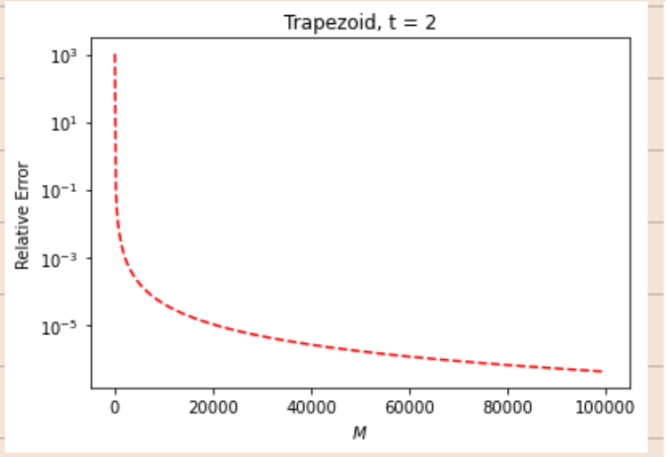
Subsequently to achieve an accuracy $< 10^{-3}$, n is set to 100000 with a step size of 200

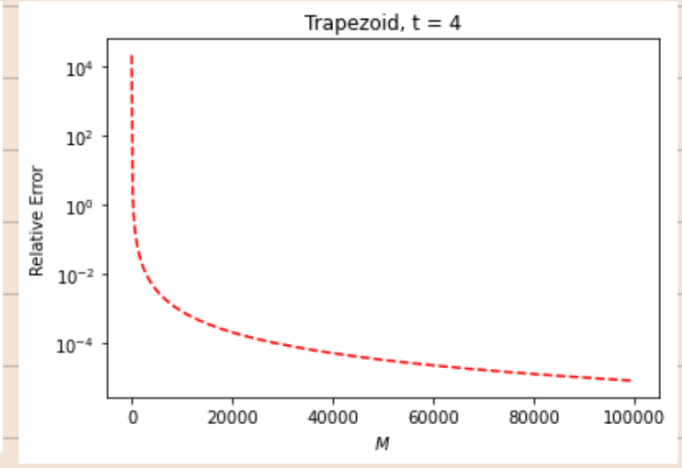
| t | max(f''(x)) | Γ(t) | error of trapezoid |
|---|---|---|---|
| 2 | 0.05 | 1 | $4.1654 \times 10^{-7}$ |
| 4 | 1.009 | 6 | $8.442427 \times 10^{-6}$ |
| 6 | 5.461 | 120 | $4.5692022 \times 10^{-5}$ |
| 8 | 111.355 | 5040 | $9.316446 \times 10^{-4}$ |
| 10 | 4854.711 | 362880 | 0.040616 |

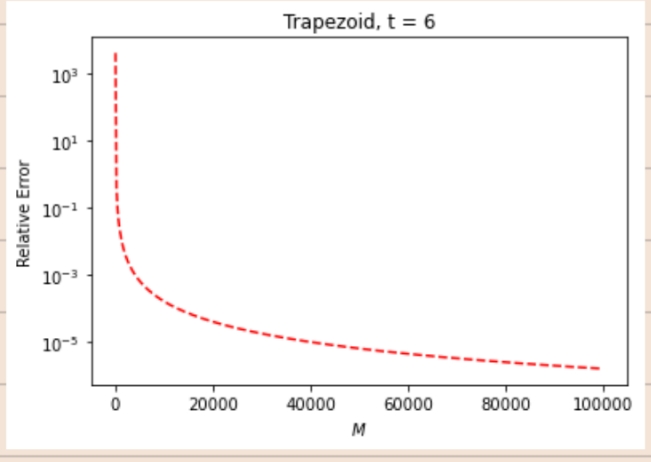
Error is increasing as t increases

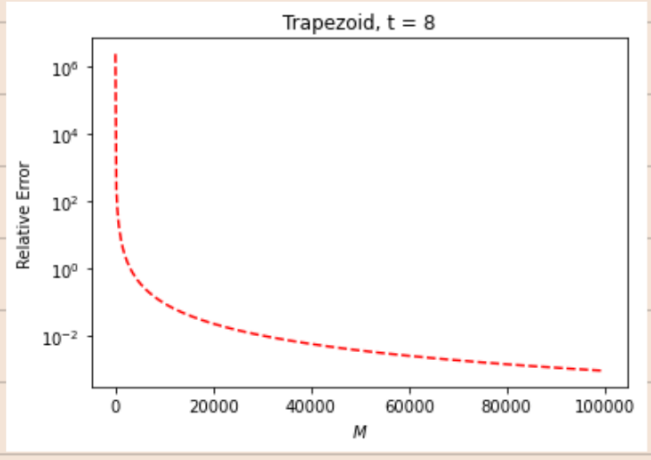
Plots on the next page

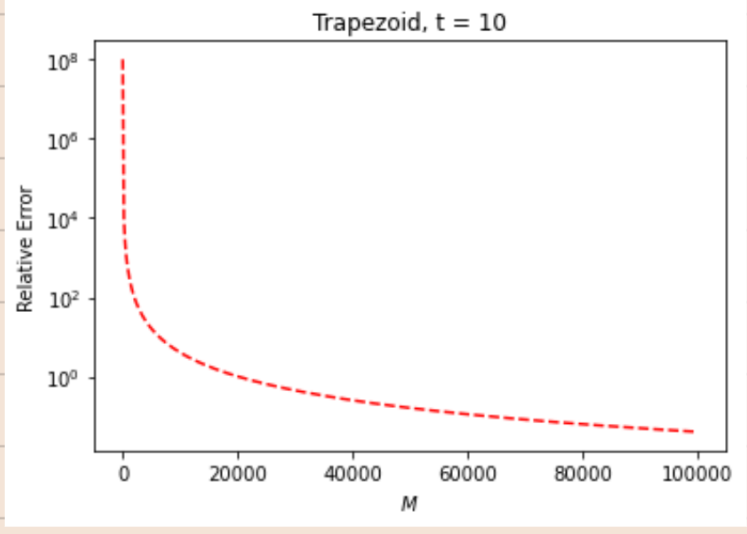
Plots for part a, with different t



# Code for Q4a and b

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 23 12:13:59 2024

@author: 2021n
"""

# get lgwts routine and numpy
import numpy as np;
import numpy.linalg as la;
import matplotlib.pyplot as plt

from scipy import integrate
from scipy import special


def eval_composite_trap(M,a,b,f):
  x = np.linspace(a,b,M);
  h = (b-a)/(M-1);
  w = h*np.ones(M);
  w[0]=0.5*w[0]; w[M-1]=0.5*w[M-1];

  I_hat = np.sum(f(x)*w);
  return I_hat,x,w;

def eval_composite_simpsons(M,a,b,f):
  x = np.linspace(a,b,M);
  h = (b-a)/(M-1);
  # Explain why this defines the weights for Simpsons
  w = (h/3)*np.ones(M);
  w[1:M:2]=4*w[1:M:2];
  w[2:M-1:2]=2*w[2:M-1:2];

  I_hat = np.sum(f(x)*w);
  return I_hat,x,w;


t = 10

f = lambda x: (x**(t-1))*np.exp(-x)
fpp = lambda x: (x**(t-3))*np.exp(-x)*(x**2 + (2 - 2*t)*x + t**2 - 3*t + 2)
#fpppp = lambda x: x**(t-5)*(x**4+(4-4*t)*x**3+(6*t**2-18*t+12)*x**2+(-4*t**3+24*t**2-44*t+24)*x+t*


tol = 1e-4
a = 0
b = 100

Ms = np.arange(3,100000,200); nM = len(Ms)
I_trap = np.zeros((nM,))
I_simp = np.zeros((nM,))
# storage for error`
err_trap = np.zeros((nM,))
err_simp = np.zeros((nM,))
for iM in range(nM):
    M = Ms[iM]
    h = (b-a)/(M-1)
```

```
    I_trap[iM],_,_ = eval_composite_trap(M,a,b,f)
    #I_simp[iM],_,_ = eval_composite_simpsons(M, a, b, f)
    err_trap[iM] = ((b-a)/12)*(h**2)*np.abs(fpp(4.176))
    #err_simp[iM] = ((b-a)/180)*(h**4)*np.abs(fpppp(0))

print(err_trap[iM])

fig,ax = plt.subplots(1)
ax.semilogy(Ms,err_trap,'r--')
ax.set_xlabel('$M$')
ax.set_title('Trapezoid, t = ' + str(t))
ax.set_ylabel('Relative Error');
plt.show()

"""
fig,ax = plt.subplots(1)
ax.semilogy(Ms,err_simp,'b--')
ax.set_xlabel('$M$')
ax.set_title('Simpsons')
ax.set_ylabel('Relative Error');
plt.show()
"""

I_quad,abserr,infodict = integrate.quad(f,0,100,args=(), full_output=1, epsabs=tol)

print(I_quad)
print(infodict['neval'])
```

b) Using the quad function with tol $= 10^{-4}$

For smaller t the trapezoid will be more accurate b/c n is so high but when t increases, the quad becomes more accurate with much less operations

| t | function evaluations |
|---|---|
| 2 | 147 |
| 4 | 147 |
| 6 | 147 |
| 8 | 189 |
| 10 | 189 |

c) When $n = 100$, using the Gauss-Legendre quadrature

$\Gamma(2)$ is approximated correctly upto $10^{-10}$

$\Gamma(10)$ is approximated correctly upto $10^{-7}$

All code uploaded to my Github

Code for Q 4 c

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 23 17:06:59 2024

@author: 2021n
"""


# get lgwts routine and numpy
import numpy as np;
import numpy.linalg as la;
import matplotlib.pyplot as plt
from numpy import polynomial

from scipy import integrate
from scipy import special

x1,w = polynomial.laguerre.laggauss(100)

t = 10
f = lambda x: x**(t-1)

I = np.sum(f(x1)*w)
print(I)
```