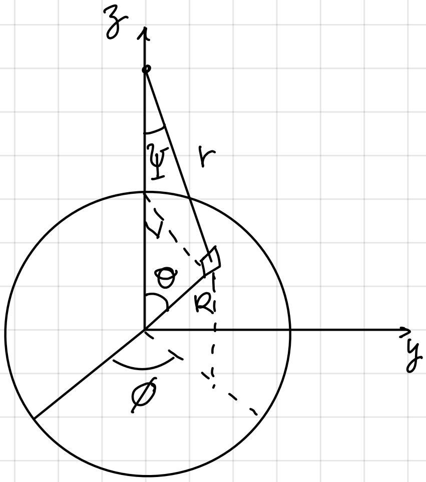
Assignment 2

Problem 1:

We have a infinitessimally thin spherical shell of charge with Radius R.
We want the electric field E:



$$dq = \sigma da = \sigma R^2 \sin\theta \, d\theta \, d\phi$$

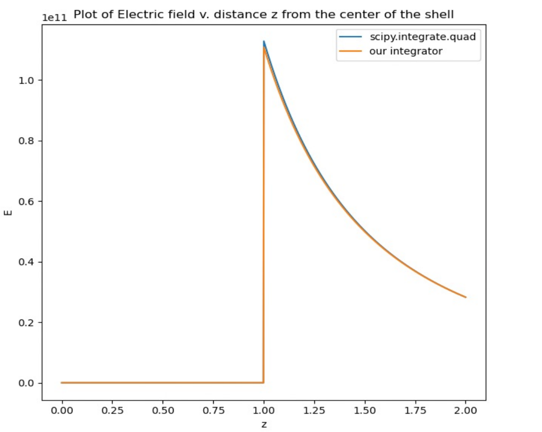
$$r^2 = R^2 + z^2 - 2Rz\cos\theta$$

$$\cos\Psi = \frac{z - R\cos\theta}{r}$$

Hence $$E_z = \frac{1}{4\pi\epsilon_0}\int \frac{\sigma R^2 \sin\theta \, d\theta \, d\phi \,(z - R\cos\theta)}{(R^2 + z^2 - 2Rz\cos\theta)^{3/2}}$$

$$E_z = \frac{1}{4\pi\epsilon_0}(2\pi R^2 \sigma)\int_0^{\pi} \frac{(z - R\cos(\theta))\sin\theta}{(R^2+z^2-2Rz\cos\theta)^{3/2}} d\theta$$

for $R = 1$ m, $\sigma = 1 \frac{C}{m^2}$, $\epsilon_0 = 8.85 \times 10^{-12}$



The singularity is for $z = R$: for $z < R$ $E = 0$ then $z > R$ it shoots then decay $\sim \frac{1}{R^2}$. Quad doesnot care as it shows the expected behavior.
However, our integrator has a trouble for $z = R$ (Run time error).
We fixed it skipping $z = R$.

Problem 2:

For the integral in Pb1 we get a ratio of 0,43 function calls with the new integrator (1st 3 lins on the picture)

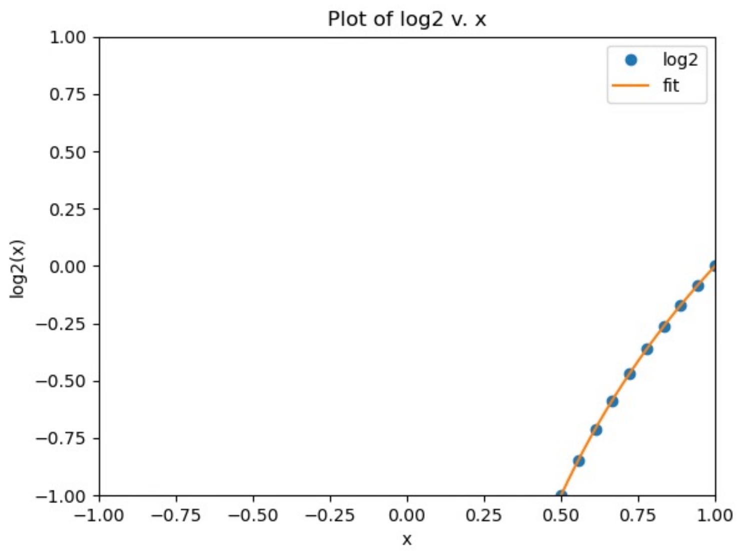
For the integral $\int_0^{\pi} \frac{1}{x^2-x+1}\,dx$ we get a ratio of 0,44 function calls with the new integrator (2nd 3 lins on the picture)

For the integral $\int_0^{\pi} \frac{1}{\sqrt{x}\,(x+1)}\,dx$ we get a ratio of 0,42 function calls with the new integrator (3rd 3 lins on the picture)

Picture of the output:

```
The answer of the integrate was -1.1089518192619607e-11
The answer, to the integrate_adaptive was -1.1089518192619607e-11
The time for E has reduced by a ratio of 0.4317596566523605



The answer of the integrate was 2.0525895930942055
The answer, to the integrate_adaptive was 2.0525895930942055
The time for f1 has reduced by a ratio of 0.44018264840182647



The answer of the integrate was 1.5016938844568517
The answer, to the integrate_adaptive was 1.5016938844568517
The time for f1 has reduced by a ratio of 0.4154340836012862
```

Problem 3 :



We use 6 terms for an accuracy of 1.12×10^{-6}.

Problem 1

```
[ ]: #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

#constants

e_0 = 8.85*10**(-12)
sigma = 1
R = 1
z = np.linspace(0, 2, 1000)

#Quad
def integrand(theta, i):
    A = (i - R*np.cos(theta))*np.sin(theta)
    B = ( R**2 + i**2 - 2*R*i*np.cos(theta))**(3/2)
    return A / B
Integral_list = []
for i in z:
    Integral_list.append(quad(integrand, 0, np.pi, args=(i))[0])
Integral_list = np.array(Integral_list)
E = (1/(4 * np.pi * e_0))*(2*np.pi* R**2 * sigma)*Integral_list


#Our integrator

def integrate(fun,a,b,tol):
    print('calling function from ',a,b)
    x=np.linspace(a,b,5)
    dx=x[1]-x[0]
```

```
    y=fun(x)
    #do the 3-point integral
    i1=(y[0]+4*y[2]+y[4])/3*(2*dx)
    i2=(y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])/3*dx
    myerr=np.abs(i1-i2)
    if myerr<tol:
        return i2
    else:
        mid=(a+b)/2
        int1=integrate(fun,a,mid,tol/2)
        int2=integrate(fun,mid,b,tol/2)
        return int1+int2

#integration of our integration without z = R
Integral_list_our = []
z_0 = np.linspace(0, 0.99,500)
z_1 = np.linspace(1.01, 2,500)
for i in z_0:
    def integrand(theta):
        A = (i - R*np.cos(theta))*np.sin(theta)
        B = ( R**2 + i**2 - 2*R*i*np.cos(theta))**(3/2)
        return A / B
    Integral_list_our.append(integrate(integrand,0,np.pi, 1e-8))
for i in z_1:
    def integrand(theta):
        A = (i - R*np.cos(theta))*np.sin(theta)
        B = ( R**2 + i**2 - 2*R*i*np.cos(theta))**(3/2)
        return A / B
    Integral_list_our.append(integrate(integrand,0,np.pi, 1e-8))
Integral_list_our = np.array(Integral_list_our)
E_our=(1/(4 * np.pi * e_0))*(2*np.pi* R**2 * sigma)*Integral_list_our


#plot settings
plt.figure(figsize=(18,9))
plt.plot(z, E, label="scipy.integrate.quad")
plt.plot(z, E_our, label="our integrator")
plt.xlabel("z")
plt.ylabel("E")
plt.title("Plot of Electric field v. distance z from the center of the shell")
plt.legend()
plt.show()
```

Problem 2

[ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""
import numpy as np

#constants
e_0 = 8.85*10**(-12)
sigma = 1
R = 1
i = 0.1

#Series of function to compare to
def integrand(theta):
        A = (i - R*np.cos(theta))*np.sin(theta)
        B = ( R**2 + i**2 - 2*R*i*np.cos(theta))**(3/2)
        return A / B
def function1(x):
    return 1 / (x**2 -x +1)
def function2(x):
    return 1 / (np.sqrt(x)*(x +1))

#integrate function
def integrate(fun,a,b,tol, extra=None):
    if not isinstance(extra, list):
        extra = [{}, 0]
    x=np.linspace(a,b,5)
    dx=x[1]-x[0]

    #we store the data of what x were checked in a dictionnary in extra as well␣
 ↪as the count of calls

    y = []
    for value in x:
        if value in extra[0].keys():
            y.append(fun(value))
            extra[1] = extra[1] + 1
        else:
            y_value = fun(value)
            extra[0][value] = y_value
            extra[1] = extra[1] + 1
            y.append(y_value)

    y = np.array(y)

    #do the 3-point integral
```

```
    i1=(y[0]+4*y[2]+y[4])/3*(2*dx)
    i2=(y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])/3*dx
    myerr=np.abs(i1-i2)
    if myerr<tol:
        return i2, extra[1]
    else:
        mid=(a+b)/2
        int1, p=integrate(fun,a,mid,tol/2, extra)
        int2, p=integrate(fun,mid,b,tol/2, extra)
        return int1+int2, extra[1]


def integrate_adaptive(fun,a,b,tol,extra=None):
    if not isinstance(extra, list):
        extra = [{}, 0]
    #we store the data of what x were checked in a dictionnary in extra as well␣
 ↪as the count of calls

    x=np.linspace(a,b,5)
    dx=x[1]-x[0]
    y = []
    for value in x:
        if value in extra[0].keys():
            y.append(extra[0].get(value))
        else:
            y_value = fun(value)
            extra[0][value] = y_value
            extra[1] = extra[1] + 1
            y.append(y_value)



    y = np.array(y)
    #do the 3-point integral
    i1=(y[0]+4*y[2]+y[4])/3*(2*dx)
    i2=(y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])/3*dx
    myerr=np.abs(i1-i2)
    if myerr<tol:
        return i2, extra[1]
    else:
        mid=(a+b)/2
        int1, p=integrate_adaptive(fun,a,mid,tol/2, extra)
        int2, p=integrate_adaptive(fun,mid,b,tol/2, extra)
        return int1+int2, extra[1]

##print the result of the integration for different functions and compare the␣
 ↪number of calls
```

```python
ans, count =integrate(integrand,0,np.pi, 1e-8)
print('The answer of the integrate was ' + str(ans))
ans, count_a=integrate_adaptive(integrand,0,np.pi, 1e-8)
print('The answer, to the integrate_adaptive was ' + str(ans))

ratio = (count_a)/(count)
print("The time for E has reduced by a ratio of " + str(ratio))

print("\n\n\n")
ans, count =integrate(function1,0,np.pi, 1e-8)
print('The answer of the integrate was ' + str(ans))
ans, count_a=integrate_adaptive(function1,0,np.pi, 1e-8)
print('The answer, to the integrate_adaptive was ' + str(ans))

ratio = (count_a)/(count)
print("The time for f1 has reduced by a ratio of " + str(ratio))

print("\n\n\n")

ans, count =integrate(function2,0.1,np.pi, 1e-8)
print('The answer of the integrate was ' + str(ans))
ans, count_a=integrate_adaptive(function2,0.1,np.pi, 1e-8)
print('The answer, to the integrate_adaptive was ' + str(ans))

ratio = (count_a)/(count)
print("The time for f1 has reduced by a ratio of " + str(ratio))
```

Problem 3

[ ]:
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 13 18:24:36 2022

@author: louis
"""

import numpy as np
from matplotlib import pyplot as plt

x_true = np.linspace(0.5, 1, 10)
x = np.linspace(0.5, 1, 1001)
y = np.log2(x)

coef = np.polynomial.chebyshev.chebfit(x,y,6) #computes the chebyshev␣
↪coefficients
```

```
fit = np.polynomial.chebyshev.chebval(x,coef) #computes the chebyshev fit with␣
↪respect to x

print(coef)
print(np.std(np.log2(x) - fit)) #computes the error on the fit
plt.xlim(-1, 1)
plt.ylim(-1, 1)

#Plot settings

plt.plot(x_true,np.log2(x_true),"o",label="log2")
plt.plot(x,fit,label="fit")
plt.legend()
plt.xlabel("x")
plt.ylabel("log2(x)")
plt.title("Plot of log2 v. x")
plt.show()

#mylog2 function

def mylog2(x):
    mantissa, exponent = np.frexp(x) #getting the mantissa and the exponent
    return np.polynomial.chebyshev.chebval(mantissa,coef) + exponent #computing␣
↪the log2
```