Computational physics -- problem set 2

The code and derivations for PS2 of PHYS512.

In [3]:
```
# import all the fun stuff
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
import seaborn as sns
sns.set_style("white")
```

Q1

First, let's do some E\&M to setup the integral into a one-dimensional integral. The electric field is given by:

$$\mathbf{E}(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\int_V d^3\mathbf{r}' \frac{\rho(\mathbf{r})dV}{|\mathbf{r}-\mathbf{r}'|^2} = \frac{1}{4\pi\epsilon_0}\int_V d^3\mathbf{r}' \frac{dq\mathbf{s}}{s^{3/2}} \tag{1}$$

with $\mathbf{s} = \mathbf{r} - \mathbf{r}'$. Choosing our point source to lie on the z-axis, the electric field in the x- and y-directions will cancel by spherical symmetry of the thin shell. Thus, only E_z is non-zero. Moreover, let the shell carry a charge q such that $dq = \sigma dA = \sigma R^2 \sin\theta d\theta d\phi$. The seperation vector is $|\mathbf{r}-\mathbf{r}'|^2 = R^2 + z^2 - 2Rz\cos\theta$. All told, the integral we will attempt to solve is:

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

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

We could just go and solve this integral, but it is neater to change the variable of integration via $u = \cos\theta$, $du = -\sin\theta d\theta$:

$$E_z = \frac{\sigma R^2}{2\epsilon_0}\int_{-1}^{1} du \frac{z - Ru}{(R^2+z^2-2Rzu)^{3/2}}. \tag{4}$$

Setting all lengths to be written in units of R, we have

$$E_z = \frac{\sigma}{2\epsilon_0}\int_{-1}^{1} du \frac{z - u}{(1+z^2-2zu)^{3/2}}. \tag{5}$$

In [3]:
```
# define the integrand
def integrand_shell(u,z):
    """
    Integrand for the electric field integration
```

```
    Lengths are in units of R, i.e. R = 1.
    """
    # numerator
    num = z - u

    # denominator
    den = (1 + z ** 2 - 2 * z * u)**(3/2)

    return num / den
```

In [62]:

```
# integration with quad
def electric_field_quad(z,a,b,return_error=False):
    """
    Electric field integration with quad
    [a,b] is integration range
    """
    # define a function that takes a single argument
    # this allows us to input "z" as an array!!!
    func = lambda u: integrand_shell(u,z)

    # integrate it from -1 to 1
    E_at_z, err = quad_vec(func, a, b)

    if return_error:
        return E_at_z, err
    else:
        return E_at_z

def integrate(func,a,b,tol=1e-6):
    """
    Simple adaptive integrator for an arbitrary
    single-variable function
    """
    x=np.linspace(a,b,5)
    dx=x[1]-x[0]
    y=func(x)
    # do the 3-point integral with 2dx
    i1=(y[0]+4*y[2]+y[4])/3*(2*dx)
    # do the 3-point integral with 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(func,a,mid,tol/2)
        int2=integrate(func,mid,b,tol/2)
        return int1+int2

def electric_field_integrator(z,a,b,tol=1e-6):
    """
    Homemade electric field integrator
    """
    func = lambda u: integrand_shell(u,z)

    E_at_z = integrate(func,a,b,tol)

    return E_at_z
```

In [98]:
```
# compute the electric field as a function of z
# scipy quad case
res = 300+1 # no. of z points
zs = np.linspace(0,3,num=res)
E_z = electric_field_quad(zs,-1,1)
```

```
<ipython-input-3-a1bdb74e3ac3>:13: RuntimeWarning: invalid value encountered i
n true_divide
  return num / den
```

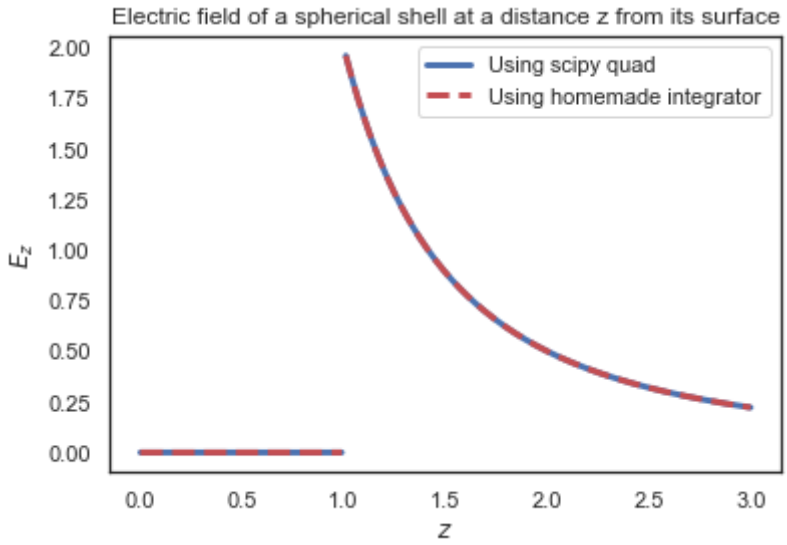
One of the values is indeed $z = R = 1$:

In [99]:
```
zs[100]
```

Out[99]:
```
1.0
```

In [101…
```
# compute the electric field as a function of z
# homemade case
E_z_homemade = np.zeros(res, dtype=float)
for i in range(res):
    z = zs[i]
    if z == 1: # hardcode case it cannot deal with
        E_z_homemade[i] = np.nan
    else:
        E_z_homemade[i] = electric_field_integrator(z,-1,1,tol=1e-6)
```

In [102…
```
%matplotlib inline

plt.figure()

plt.title("Electric field of a spherical shell at a distance z from its surface

plt.plot(zs,E_z,c='b',ls='-',lw=3,label="Using scipy quad")
plt.plot(zs,E_z_homemade,c='r',ls='--',lw=3,label="Using homemade integrator")


plt.xlabel(r"$z$")
plt.ylabel(r"$E_z$")

plt.legend()

plt.show()
```



They are clearly in good agreement, and what we expect the field to look like (i.e. 0 within the shell and a point source $1/z^2$ outside of it).

There \textbf{is} a singularity in the integral, at $z = 1$. Looking closely at our integrand, this is due to the denominator going to zero at $z = u = 1$, causing a divergence. `scipy.integrate.quad` does not care -- at most, it will issue a warning saying that we have divided by zero. Our homemade integrator, however, \textbf{does} care, since it works in a recursive fashion. This means that as long as the error between the 3-point integral at dx and $2dx$ is greater than a given tolerance, it will repeat. But recall that the integrand is divergent: near the singularity, the 3-point integrals at different widths will be vastly different from each other, since there is a huge jump from one point to the next which is not captured by the 3-point integral with $2dx$. So the algorithm will continue its recursion, without ever achieving an error less than a given tolerance due to the singular nature of the integrand always allowing for large jumps in arbitrarily small widths.

To remedy this for plotting, we hard-coded the singularity in our homemade integrator. If we let it run at $z = R = 1$, we indeed get a recursion error:

In [81]:
```
electric_field_integrator(z=1,a=-1,b=1,tol=1e-6)
```

```
<ipython-input-3-a1bdb74e3ac3>:13: RuntimeWarning: invalid value encountered i
n true_divide
  return num / den
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-81-44204bdda5d7> in <module>
----> 1 electric_field_integrator(z=1,a=-1,b=1,tol=1e-6)

<ipython-input-62-2f23b2feb10d> in electric_field_integrator(z, a, b, tol)
     44     func = lambda u: integrand_shell(u,z)
     45 
---> 46     E_at_z = integrate(func,a,b,tol)
     47 
     48     return E_at_z

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 
```

```
<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
```

```
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 
```

```
<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
```

```
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
```

```
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39 

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
```

```
     39

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     35         mid=(a+b)/2
     36         int1=integrate(func,a,mid,tol/2)
---> 37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2
     39

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     34     else:
     35         mid=(a+b)/2
---> 36         int1=integrate(func,a,mid,tol/2)
     37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2

... last 1 frames repeated, from the frame below ...

<ipython-input-62-2f23b2feb10d> in integrate(func, a, b, tol)
     34     else:
     35         mid=(a+b)/2
---> 36         int1=integrate(func,a,mid,tol/2)
     37         int2=integrate(func,mid,b,tol/2)
     38         return int1+int2

RecursionError: maximum recursion depth exceeded while calling a Python object
```

Apologies for making you scroll through all of that! Moral of the story: our integrator \textbf{does} care, `scipy.quad` does \textbf{not}.

Q2

We want to write an adaptive integrator without repeat function calls for the same point. Indeed, the adaptive integrator we saw in class sometimes calls the function twice at a given point. Indeed, it starts off with 5 points. If the error between the 3-point integrals (double

and single dx) is below the tolerance, great! It returns the the more precise 3-point integral (single dx). If not, it splits the range into two regions, each with 5 points. Of these 9 total points, 5 have already been evaluated! So there is no need to call the function at these points again. We should find a way to feed them to our adaptive integration as entries.

In [ ]:
```
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
```

In [56]:
```
def integrate_adaptive(fun,a,b,tol,extra=None,calls=0):
    """
    Adaptive integrate function
    Does not call `fun` at the same point twice
    """
    # initialize count of function calls
    global count # global variable because it is referenced outside of definiti

    # if it is the first call:
    if extra == None:

        # set up the x-values
        x=np.linspace(a,b,5)
        dx=x[1]-x[0]
        y0,y1,y2,y3,y4=fun(x)
        count += 5 # there are five function calls here

    else:
        # unpack extra
        dx, known_vals = extra

        # unpack known values
        y0, y2, y4 = known_vals # beginning, middle, end

        # compute new x-points
        x1 = a + dx
        x3 = a + 3*dx

        # evaluate function at these new points
        y1 = fun(x1)
        y3 = fun(x3)
        count += 2 # there are 2 additional function calls here

    #do the 3-point integrals
    i1=(y0+4*y2+y4)/3*(2*dx)
    i2=(y0+4*y1+2*y2+4*y3+y4)/3*dx
```

```
    # compute the error
    myerr=np.abs(i1-i2)

    # if the error is below some tolerance, return integral
    if myerr<tol:
        return i2

    # if not, rinse and repeat
    else:
        mid=(a+b)/2
        dx_new = dx/2 # the new dx value
        tol_new = tol/2
        known_yvals_left = [y0,y1,y2] # start, middle, end for left half
        known_yvals_right = [y2,y3,y4] # start, middle, end for right half
        int1=integrate_adaptive(fun,a,mid,tol_new,extra=[dx_new,known_yvals_lef
        int2=integrate_adaptive(fun,mid,b,tol_new,extra=[dx_new,known_yvals_rig
        return int1+int2
```

Copy paste Jon's technique for comparison:

In [57]:
```
def integrate(fun,a,b,tol):
    global count
    x=np.linspace(a,b,5)
    dx=x[1]-x[0]
    y=fun(x)
    count+=5 # add 5 to the count
    #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
```

In [63]:
```
sinfun = np.sin
heavysidefun = lambda x: 1.0*(x>0)
expfun = np.exp

def offset_gauss(x):
    return 1+10*np.exp(-0.5*x**2/(0.1)**2)

# sin(x)
count = 0
sin_int = integrate_adaptive(fun=sinfun,a=0,b=np.pi,tol=1e-6,extra=None) # our
print("Function calls for sin(x) with our method: {}".format(count))
count = 0
sin_int_jon = integrate(fun=sinfun,a=0,b=np.pi,tol=1e-6) # jon's way in class
print("Function calls for sin(x) with Jon's method: {}".format(count))

# exp(x)
count = 0
exp_int = integrate_adaptive(fun=expfun,a=0,b=np.pi,tol=1e-6,extra=None) # our
print("Function calls for exp(x) with our method: {}".format(count))
count = 0
```

```
exp_int_jon = integrate(fun=expfun,a=0,b=np.pi,tol=1e-6) # jon's way in class
print("Function calls for exp(x) with Jon's method: {}".format(count))


# gauss
count = 0
gauss_int = integrate_adaptive(fun=offset_gauss,a=-1,b=1,tol=1e-6,extra=None) #
print("Function calls for gauss with our method: {}".format(count))
count = 0
gauss_int_jon = integrate(fun=offset_gauss,a=-1,b=1,tol=1e-6) # jon's way in cl
print("Function calls for gauss with Jon's method: {}".format(count))
```

```
Function calls for sin(x) with our method: 97
Function calls for sin(x) with Jon's method: 235
Function calls for exp(x) with our method: 161
Function calls for exp(x) with Jon's method: 395
Function calls for gauss with our method: 409
Function calls for gauss with Jon's method: 1015
```

Clearly, our method saves a significant amount of function calls. Are the integrals correct, at least?

In [64]:
```
print("Difference between methods for sin(x): {}".format(sin_int-sin_int_jon))
print("Difference between methods for exp(x): {}".format(exp_int-exp_int_jon))
print("Difference between methods for gauss: {}".format(gauss_int-gauss_int_jor
```

```
Difference between methods for sin(x): 5.329070518200751e-15
Difference between methods for exp(x): 6.750155989720952e-14
Difference between methods for gauss: 0.0
```

Yes, they do. In fact, there should be no error between the two methods, since we use the exact same x and y values. The small errors come from the fact that we decided not to re-compute x as a linspace and just take the points of interest by using `xi = a + i*dx`. In doing so, there was a bit of error on the `dx` stemming from the division by 2 in the previous step.

All in all, we wrote a recursive variable step size integrator that does \textbf{not} call $f(x)$ multiple times.

Q3

Now, we'll tackle the Chebyshev fit of the function $f(x) = \log_2(x)$ from $x = 0.5$ to $x = 1$. For the first part of the problem, we should use the function `numpy.polynomial.chebyshev.chebfit`.

In [316…
```
# this is the fit function
from numpy.polynomial.chebyshev import chebfit
# this is the polynomial generator
from numpy.polynomial.chebyshev import chebval
# this is the Chebyshev class
from numpy.polynomial.chebyshev import Chebyshev
# this is the polynomial convertor from cheb to normal polys
from numpy.polynomial.chebyshev import cheb2poly
```

Next, set up the x and y data. We also need to re-scale the x-data to have it on -1 to $+1$ for the Chebyshev fit. For starters, we consider 101 points and a truncated fit at the 10th order.

In [317…
```
# number of points we'll be initially using
res_init = 101
x = np.linspace(0.5,1,num=res_init)
x_fix = 4 * x - 3
y = np.log2(x)
```

In [318…
```
# compute the cheb coefficients and residuals (we don't care about the other st

# let's take a tenth degree polynomial for now
deg = 10

# run the fit and return the residuals
coeffs = chebfit(x_fix,y,deg=deg,full=False)

# put the coefficients in front of the polynomials
poly_cheb = chebval(x_fix,coeffs)
```

Of interest to us is the maximal error between our fit and the true values. This will be our measure of "accuracy in the region", which for our purposes we would like to have at 10^{-6}.

In [319…
```
max(poly_cheb-y)
```

Out[319]:
```
1.941186988929644e-09
```

The fit is obviously good, but let's see what we can do with fewer points and lesser polynomial degree. Let's try 11 points and `deg` $= 7$.

In [320…
```
# number of points we'll be using
res = 11
x = np.linspace(0.5,1,num=res)
x_fix = 4 * x - 3
y = np.log2(x)

# compute the cheb coefficients and residuals (we don't care about the other st

# let's take a seventh degree polynomial now
deg = 7

# run the fit and return the residuals
coeffs = chebfit(x_fix,y,deg=deg,full=False)

# put the coefficients in front of the polynomials
poly_cheb = chebval(x_fix,coeffs)
```

In [321…
```
max(poly_cheb-y)
```

Out[321]:
```
1.8725337769254224e-07
```

It works to within our accuracy ($< 10^{-6}$)! Now, we input our coefficients into the class `Chebyshev` to make the polynomial.

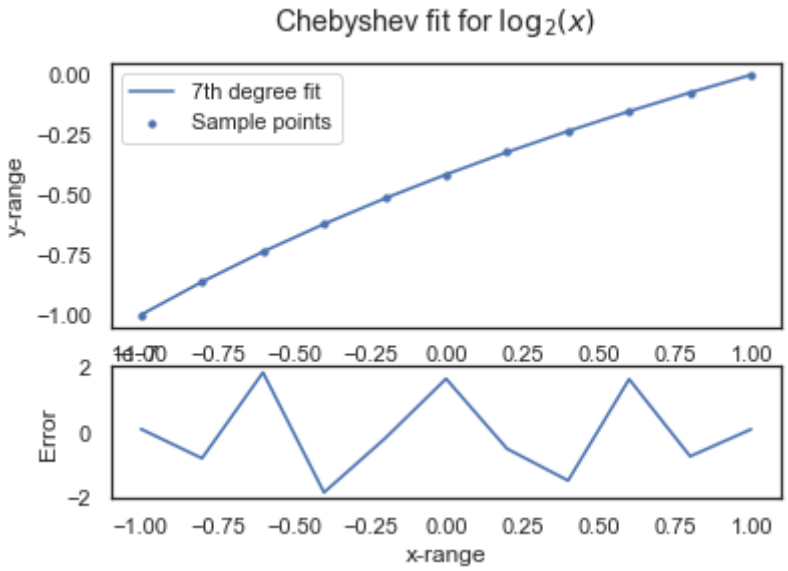
In [322…
```
Chebyshev(coeffs) # just to visualize the polynomial symbolically
```

Out[322]:
$$x \mapsto \text{-0.45689351086986496}\,T_0(x) + 0.49505472238334425\,T_1(x) - 0.04246916635277\text{3}\,T_5(x) - 1.21618790425248\text{e-}05\,T_6(x) + 1.770323338526702\text{e-}06\,T_7(x)$$

where T_i is the ith Chebyshev polynomial. Let's plot it:

In [324…
```
plt.figure()

fig, ax = plt.subplots(nrows=2, ncols=1, gridspec_kw={'height_ratios': [2,1]})

plt.suptitle("Chebyshev fit for $\log_2(x)$")
ax[0].scatter(x_fix,y,s=10,label="Sample points")
ax[0].plot(x_fix,poly_cheb,label="{}th degree fit".format(deg))
ax[0].legend()
ax[0].set_ylabel("y-range")

ax[1].set_xlabel("x-range")
ax[1].set_ylabel("Error")

ax[1].plot(x_fix,poly_cheb-y)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



The orange line (Chebyshev fit) blocks the blue one (x and y data). Note that the x-data has been re-scaled to the correct range (-1 to 1). Since we have the coefficients handy, let's package all of this into a single neat function:

In [325…
```
def log2_cheb(x):
    """
    Returns log base 2 of a number between 0.5 and 1 to within an accuracy < 1(
    """
    # set up the truncated fit
    res = 11
    xx = np.linspace(0.5,1,num=res)
    yy = np.log2(xx)
```

```
    # re-scale the x-data
    xx_fix = 4 * xx - 3

    # set the polynomial degree
    deg = 7

    # run the fit and return the residuals
    coeffs = chebfit(xx_fix,yy,deg=deg,full=False)

    # compute answer
    ans = chebval(4*x - 3, coeffs) # <- don't forget to re-scale

    return ans
```

In [326…
```
# check to see that it works on a random point
print("Error on x=0.7 is:",log2_cheb(0.7)-np.log2(0.7))
# check to see that it works on first point
print("Error on x=0.5 is:",log2_cheb(0.5)-np.log2(0.5))
# check to see that it works on last point
print("Error on x=1.0 is:",log2_cheb(1.)-np.log2(1.))
```

```
Error on x=0.7 is: -1.6509636213690726e-08
Error on x=0.5 is: 1.2487550327122676e-08
Error on x=1.0 is: 1.1721328962988053e-08
```

We can also convert it into a simple readable polynomial using `cheb2poly`:

In [327…
```
poly_coeffs = cheb2poly(coeffs)

def log2_function_poly(x):
    """
    Computes log2 in range [0.5,1] using chebyshev polynomials to an accuracy <
    """
    # we already know this
    deg = 7

    # re-scale x
    x_new = 4*x - 3

    ans = 0
    for i in range(deg+1):
        ans += poly_coeffs[i] * x_new**i

    return ans
```

In [328…
```
# quick check
print("Error at 0.7 is (poly):",log2_function_poly(0.7)-np.log2(0.7))
```

```
Error at 0.7 is (poly): -1.650963632471303e-08
```

Thus, we have our function which takes in x between 0.5 and 1 and returns $\log_2(x)$ to an accuracy $< 10^{-6}$.

Now, we will write our own function for \textit{any} positive number. A nice way to map this onto the range (-1,1) is by splitting it into its mantissa (which is by default between -1 and 1) and its two-exponent. For positive values, it is between (0,1), by definition. So, any positive number $x = m \times 2^n$ will have $\log(x) = \log(m) + n \log 2$ where n is an integer, always.

Thus, assuming we know $\log 2$, our task is reduced to computing $\log(m)$ in a range (0,1). Sounds a whole lot like what we were doing just before!

But wait, it gets better! Since `np.frexp` breaks a number down into its mantissa and exponent, a positive mantissa will actually already have a range (0.5,1). This is because if the mantissa is less than 0.5, it will actually prefer to rewrite itself with as $(2^k \times m) \times 2^{n-k}$, thus making the mantissa fall into the range (0.5,1) always! For example, if $x = 0.2 * 2^3$, the correct re-writing will be $x = 0.8 * 2^1$ ($k = 1$ in this case). Check it:

In [329…
```
np.frexp(0.2*2**3.)
```

Out[329]:
```
(0.8, 1)
```

As expected! So, we just need to repeat our above procedure but for `np.log` instead of `np.log2`:

In [330…
```
def mylog2(x):
    """
    Returns natural log of a number between 0.5 and 1 to within an accuracy < 1
    """
    # get the mantissa and exponent
    m, n = np.frexp(x)

    # compute log(m)

    # set up the truncated fit
    res = 11
    mm = np.linspace(0.5,1,num=res)
    yy = np.log(mm)

    # re-scale the x-data
    mm_fix = 4 * mm - 3

    # set the polynomial degree
    deg = 7

    # run the fit and return the residuals
    coeffs = chebfit(mm_fix,yy,deg=deg,full=False)

    # compute answer
    logm = chebval(4*m - 3, coeffs) # <- don't forget to re-scale

    # next, compute log2
    # we will assume that log2 is known, since it is a universal constant
    # we could compute it by calling this function, but then we would get an ir
    # since np.frexp(2) = 1, 1 , we cannot use the method prescribed above to c
    log2 = np.log(2)

    return logm + n * log2
```

To finish up, let's run some quick tests

In [331…
```
test_array = np.linspace(0.1,100,num=10000)
print("Maximum error for our mylog2 function is:",max(mylog2(test_array)-np.log
```

```
Maximum error for our mylog2 function is: 1.8969676496283228e-07
```

Well within our desired accuracy range! So, we have first constructed a function which computes $\log_2$ of a number between 0.5 and 1 in the form `log2_cheb(x)`. Then, we used a slight modification of this function to build `mylog2`, which can take any positive number and return its natural logarithm $\log(x)$. Both of these functions have an accuracy better than 10^{-6}.

In [ ]: