Computational Physics: HW1

9.5|10

Billy Strickland

September 27, 2019

**I used double precision for all the following problems* ✓

1 Derivation

1.1

The code in Fig. 1 was used to calculate the derivatives.

f(x)	forward	central	extrapolated	true value
$cos(x)$	-0.09983346704	-0.09983341711	-0.09983341671	-0.09983341664
e^{-x}	-0.90483737280	-0.90483741832	-0.90483741860	-0.90483741803

✓

Returned values of the derivatives at x=0.1 of the two functions using the three different methods. All methods return reasonable values for the derivatives based on convergence of the derivative.

1.2

I calculated relative error for each method with respect to the step size Δx. I then plotted the errors as a function of step sizes. I used the code in Fig. 2 to do this, and I have plots of the relative error vs. step size at the values x=0.1 in Fig. 3, and x=10 in Fig. 4. The scaling does match what we expect, since for midpoint method, we expect the relative error to scale as $\sqrt{C}$, with $C = 10^-14$ double precision. Therefore we should see a minimum in the relative error around 10^{-7} for midpoint method, which is what we see in the graphs. ✓

1.3

We see effects of truncation error on the side of smaller values of h, and round off error on the side of larger values of h. ✓

2 Integration

2.1

Code that implements each method of integration are provided in Fig. 5.

f(x)	midpoint	trapezoid	simpsons	true value
e^{-x}	0.63212055882	0.63212055882	0.63212055882	0.63212055882

Returned values of the integrals for bounds x=0 to x=1 of the function using the three different methods. All methods return reasonable values for the integrals, based on convergence.

2.2

To calculate the errors in the integrals, I used this code. It includes a method similar to that of 1b, where I looped over the number of bins, and calculated the integrals at each value for the number of bins. What is plotted is the relative error for each value of the integral at a given number of bins. I sampled from $j = 1$ to $j = 2^2 5\ 10^7$.

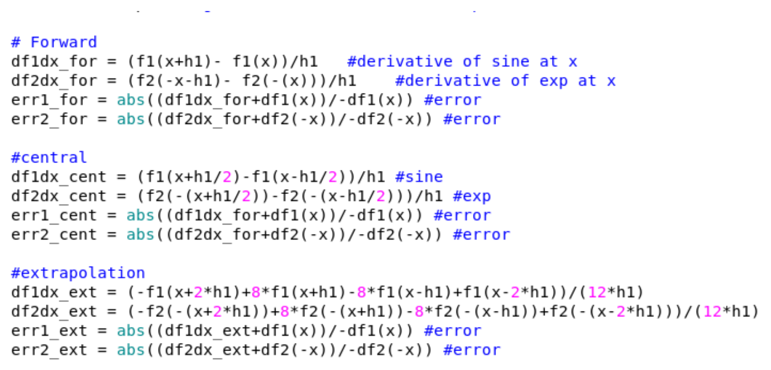
describe scalings and error regimes here

2.3

The plots of the relative errors of the derivatives of the two functions using the three different methods are shown in Fig. 7.

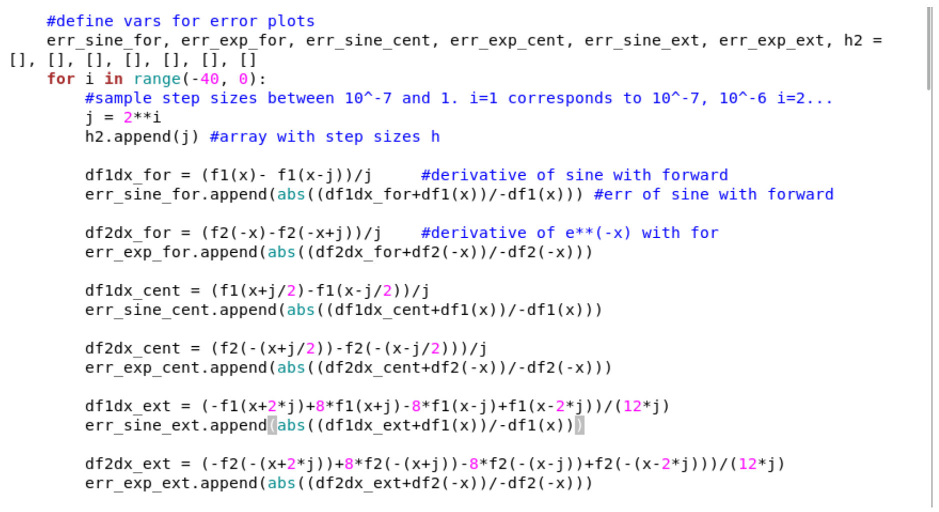
3 BAO Peak

In Fig. 8 is the code I used to load the data file, interpolate the power spectrum, integrate for different values of k and r, and plot the result as a function of r. In Fig. 9 I have the plot of the interpolated power spectrum as a function of k in Fig. 9. I plotted the value of $r^2\xi(r)$ as a function of r, and noted the position of the BAO peak in Fig. 10.



```
# Forward
df1dx_for = (f1(x+h1)- f1(x))/h1   #derivative of sine at x
df2dx_for = (f2(-x-h1)- f2(-(x)))/h1    #derivative of exp at x
err1_for = abs((df1dx_for+df1(x))/-df1(x)) #error
err2_for = abs((df2dx_for+df2(-x))/-df2(-x)) #error

#central
df1dx_cent = (f1(x+h1/2)-f1(x-h1/2))/h1 #sine
df2dx_cent = (f2(-(x+h1/2))-f2(-(x-h1/2)))/h1 #exp
err1_cent = abs((df1dx_for+df1(x))/-df1(x)) #error
err2_cent = abs((df2dx_for+df2(-x))/-df2(-x)) #error

#extrapolation
df1dx_ext = (-f1(x+2*h1)+8*f1(x+h1)-8*f1(x-h1)+f1(x-2*h1))/(12*h1)
df2dx_ext = (-f2(-(x+2*h1))+8*f2(-(x+h1))-8*f2(-(x-h1))+f2(-(x-2*h1)))/(12*h1)
err1_ext = abs((df1dx_ext+df1(x))/-df1(x)) #error
err2_ext = abs((df2dx_ext+df2(-x))/-df2(-x)) #error
```

Figure 1: Algorithms for forward, central, and extrapolated methods for solving derivatives. $f1(x) = cos(x)$ and $f2(x) = e^x$. $h1$ is the width of my steps, called Δx above. $df1$ and $df2$ are the known values of the derivatives, as given by 3 and 4. The errors are then absolute value of the true value of the derivative minus the calculated value, over the true value.



```
    #define vars for error plots
    err_sine_for, err_exp_for, err_sine_cent, err_exp_cent, err_sine_ext, err_exp_ext, h2 = 
[], [], [], [], [], [], []
    for i in range(-40, 0):
        #sample step sizes between 10^-7 and 1. i=1 corresponds to 10^-7, 10^-6 i=2...
        j = 2**i
        h2.append(j) #array with step sizes h

        df1dx_for = (f1(x)- f1(x-j))/j     #derivative of sine with forward
        err_sine_for.append(abs((df1dx_for+df1(x))/-df1(x))) #err of sine with forward

        df2dx_for = (f2(-x)-f2(-x+j))/j    #derivative of e**(-x) with for
        err_exp_for.append(abs((df2dx_for+df2(-x))/-df2(-x)))

        df1dx_cent = (f1(x+j/2)-f1(x-j/2))/j
        err_sine_cent.append(abs((df1dx_cent+df1(x))/-df1(x)))

        df2dx_cent = (f2(-(x+j/2))-f2(-(x-j/2)))/j
        err_exp_cent.append(abs((df2dx_cent+df2(-x))/-df2(-x)))

        df1dx_ext = (-f1(x+2*j)+8*f1(x+j)-8*f1(x-j)+f1(x-2*j))/(12*j)
        err_sine_ext.append(abs((df1dx_ext+df1(x))/-df1(x)))

        df2dx_ext = (-f2(-(x+2*j))+8*f2(-(x+j))-8*f2(-(x-j))+f2(-(x-2*j)))/(12*j)
        err_exp_ext.append(abs((df2dx_ext+df2(-x))/-df2(-x)))
```

Figure 2: Calculations of errors as a function of step sizes. Looping over step sizes from 10^{-11} to 1.To do this, I looped over step sizes, j, ranging from 10^{-12} to 1 (i=-40 corresponded to 2^{-40}, or 10^{-12}, and i=0 corresponded to 2^0. I computed this error as a function of step size for each function, $f1$ and $f2$, and for each of the three methods.

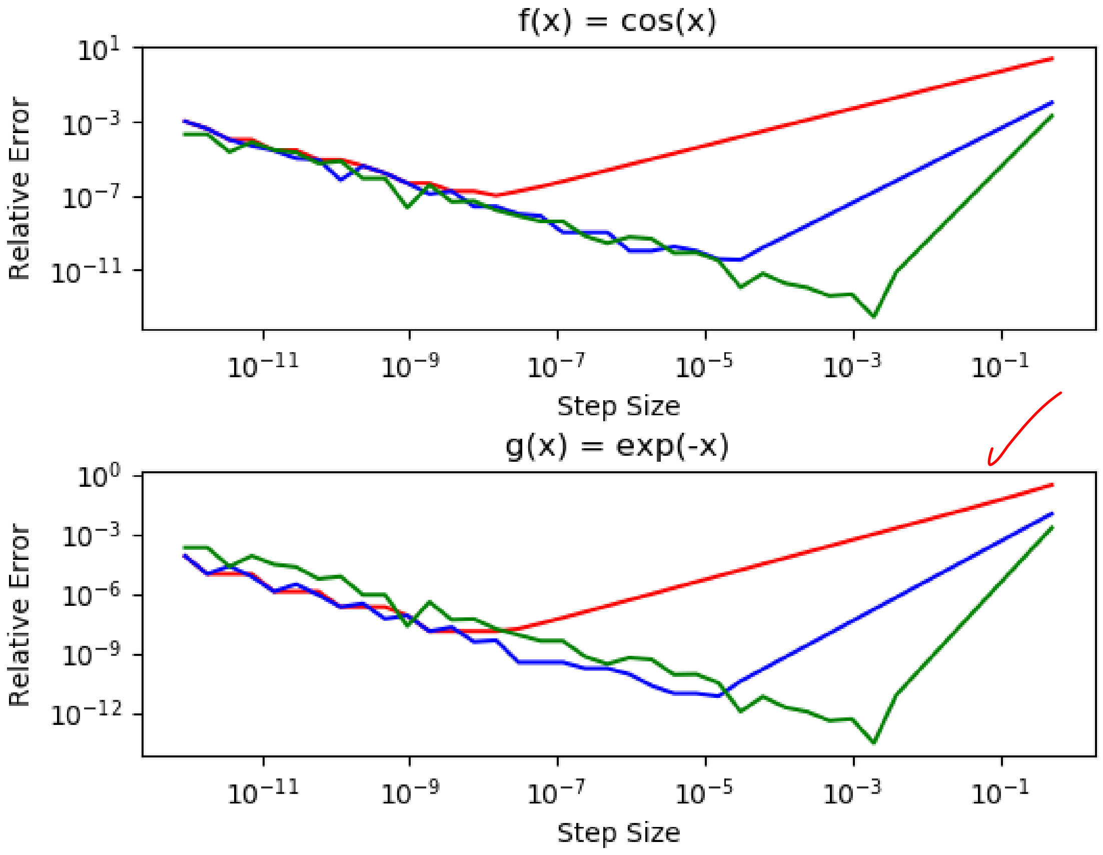


Figure 3: Relative Error as a function of step sizes for the functions f and g at x=10. Red is forward, blue is central, and green is extrapolated.

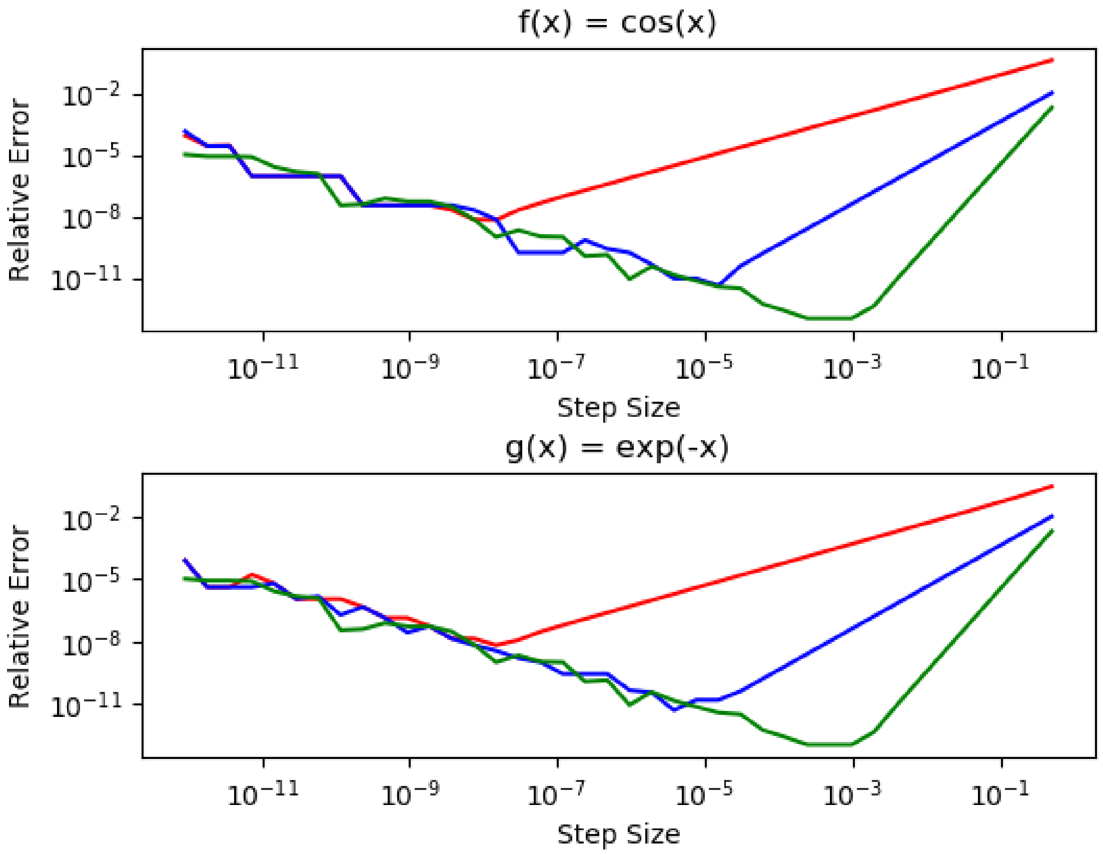
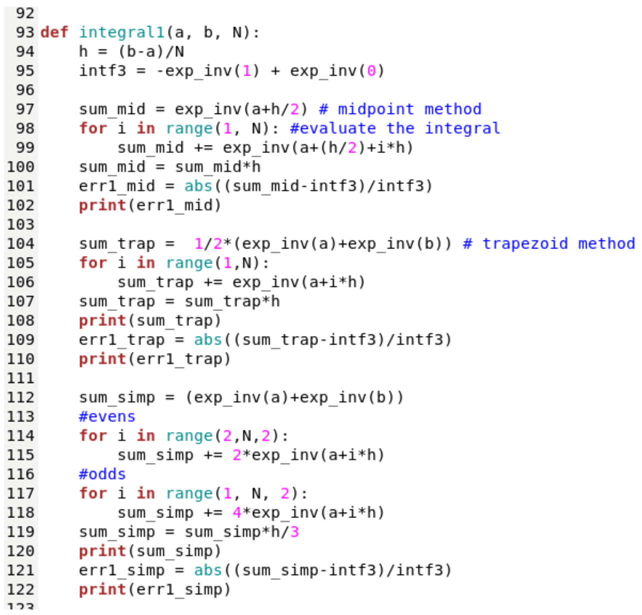
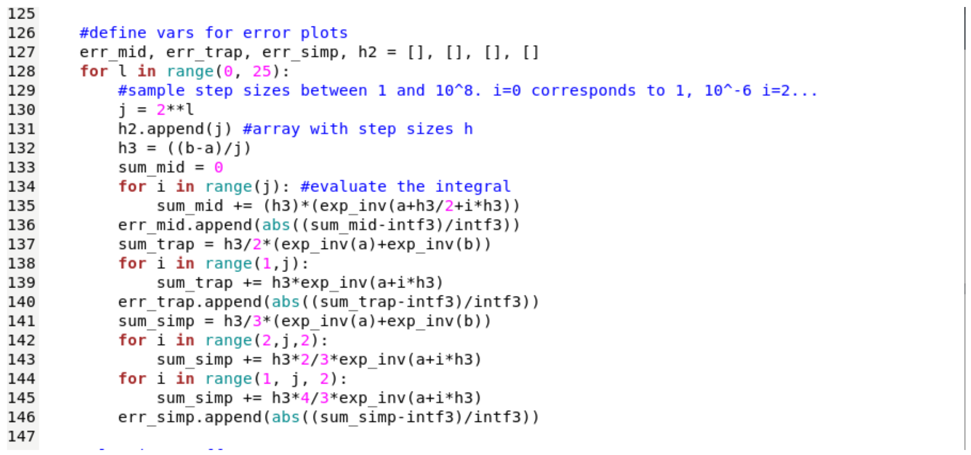


Figure 4: The same calculation at x=0.1



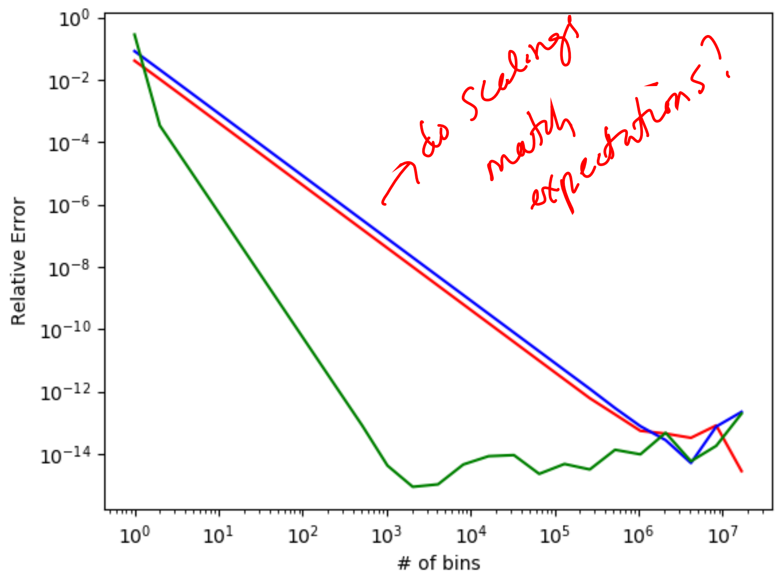
```python

def integral1(a, b, N):
    h = (b-a)/N
    intf3 = -exp_inv(1) + exp_inv(0)

    sum_mid = exp_inv(a+h/2) # midpoint method
    for i in range(1, N): #evaluate the integral
        sum_mid += exp_inv(a+(h/2)+i*h)
    sum_mid = sum_mid*h
    err1_mid = abs((sum_mid-intf3)/intf3)
    print(err1_mid)

    sum_trap =  1/2*(exp_inv(a)+exp_inv(b)) # trapezoid method
    for i in range(1,N):
        sum_trap += exp_inv(a+i*h)
    sum_trap = sum_trap*h
    print(sum_trap)
    err1_trap = abs((sum_trap-intf3)/intf3)
    print(err1_trap)

    sum_simp = (exp_inv(a)+exp_inv(b))
    #evens
    for i in range(2,N,2):
        sum_simp += 2*exp_inv(a+i*h)
    #odds
    for i in range(1, N, 2):
        sum_simp += 4*exp_inv(a+i*h)
    sum_simp = sum_simp*h/3
    print(sum_simp)
    err1_simp = abs((sum_simp-intf3)/intf3)
    print(err1_simp)
```

Figure 5: Source code for calculating the integrals of the a given function, exp inv = f(x) = e^{-t}, using midpoint, trapezoid, and Simpson's methods. The step size is given by h, which is given by the bounds of your integral, which are 0 and 1, and N, the total number of bins. For each integral, I iterate over the value of the function at various points, and making a sum of all the points multiplied by the step size in some way corresponding to the method used.



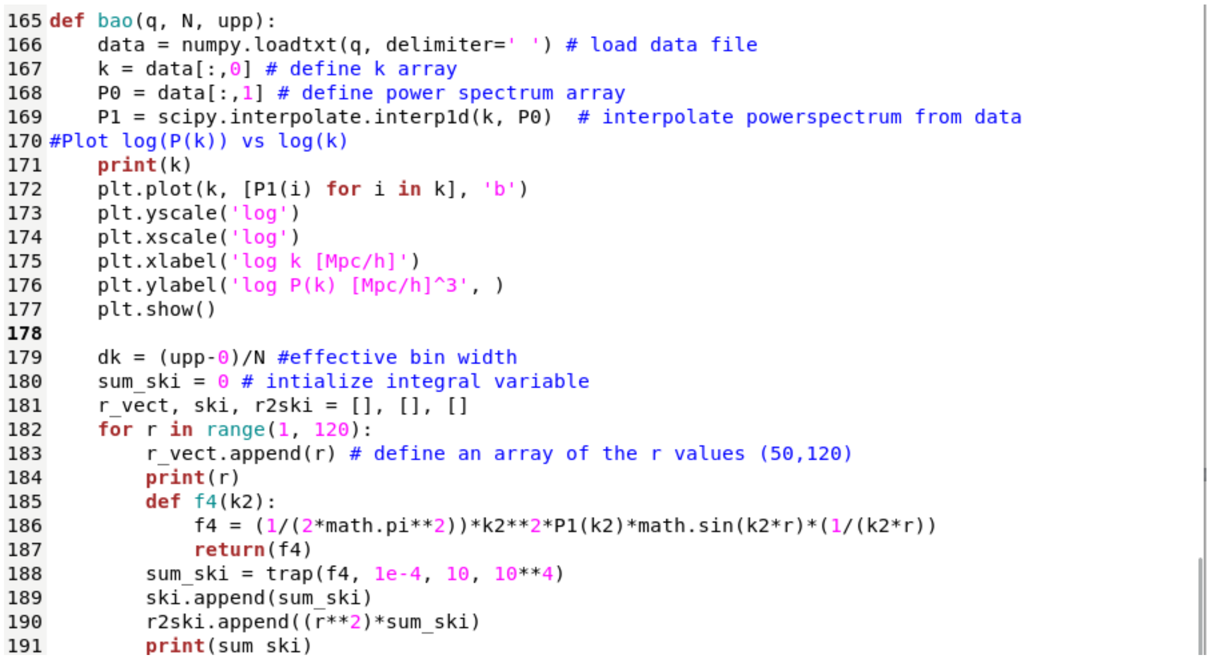
```python

    #define vars for error plots
    err_mid, err_trap, err_simp, h2 = [], [], [], []
    for l in range(0, 25):
        #sample step sizes between 1 and 10^8. i=0 corresponds to 1, 10^-6 i=2...
        j = 2**l
        h2.append(j) #array with step sizes h
        h3 = ((b-a)/j)
        sum_mid = 0
        for i in range(j): #evaluate the integral
            sum_mid += (h3)*(exp_inv(a+h3/2+i*h3))
        err_mid.append(abs((sum_mid-intf3)/intf3))
        sum_trap = h3/2*(exp_inv(a)+exp_inv(b))
        for i in range(1,j):
            sum_trap += h3*exp_inv(a+i*h3)
        err_trap.append(abs((sum_trap-intf3)/intf3))
        sum_simp = h3/3*(exp_inv(a)+exp_inv(b))
        for i in range(2,j,2):
            sum_simp += h3*2/3*exp_inv(a+i*h3)
        for i in range(1, j, 2):
            sum_simp += h3*4/3*exp_inv(a+i*h3)
        err_simp.append(abs((sum_simp-intf3)/intf3))

```

Figure 6: Code I used to calculate the relative errors for each f the integral methods. I did this in a similar way to the derivatives, where I looped over a variable l that tells me the value of the number of bins j. For each value of the number of bins, I calculated the integral of the function provided using each integration method. I then took those values and subtracted them from the expected value of the integral, dividing by the expected value, and plotted the relative error for each integration method.



→ do scalings match expectations?

Figure 7: The relative error as a function of the of bins used to calculate the integral. Midpoint method is in red, trapezoid is in blue, and Simpson's is in green. We can see that trapezoid method and midpoint method both scale similarly with the number of bins. However, for a given value of the number of bins, trapezoid will actually do worse than midpoint, since errors in midpoint will in general cancel each other out, where, depending on the function, trapezoid will totally underestimate/overestimate the value of the integral. Round off error affects all the methods around a bin size of 10^7, however, we can see that Simpson's method actually reaches a minimum relative error much quicker, at around 10^3 bins. This tells us that Simpson's method can give the best approximation of the value of the integral in the least amount of computations



```python
def bao(q, N, upp):
    data = numpy.loadtxt(q, delimiter=' ') # load data file
    k = data[:,0] # define k array
    P0 = data[:,1] # define power spectrum array
    P1 = scipy.interpolate.interp1d(k, P0)  # interpolate powerspectrum from data
#Plot log(P(k)) vs log(k)
    print(k)
    plt.plot(k, [P1(i) for i in k], 'b')
    plt.yscale('log')
    plt.xscale('log')
    plt.xlabel('log k [Mpc/h]')
    plt.ylabel('log P(k) [Mpc/h]^3', )
    plt.show()

    dk = (upp-0)/N #effective bin width
    sum_ski = 0 # intialize integral variable
    r_vect, ski, r2ski = [], [], []
    for r in range(1, 120):
        r_vect.append(r) # define an array of the r values (50,120)
        print(r)
        def f4(k2):
            f4 = (1/(2*math.pi**2))*k2**2*P1(k2)*math.sin(k2*r)*(1/(k2*r))
            return(f4)
        sum_ski = trap(f4, 1e-4, 10, 10**4)
        ski.append(sum_ski)
        r2ski.append((r**2)*sum_ski)
        print(sum_ski)
```

Figure 8: Source code I used to plot the correlation function as a function of r, from data taken on the power spectrum as a function of wavenumber. In the first lines, I load the data and define arrays for the wavenumber and powerspectrum. I then use these to interpolate P(k), and call this in my integral below. I plug in P(k) for different values of k, and integrate some function with P(k) over the range of k values provided. I used 10^{-4} to 10^{3} as my bounds for the integral, since bounds above 10^{3} did not end up changing the value of the integral, and used a number of bins, N = 10,000 in order to produce the plots below. N = 10,000 gave convergence in the values of the integrals.

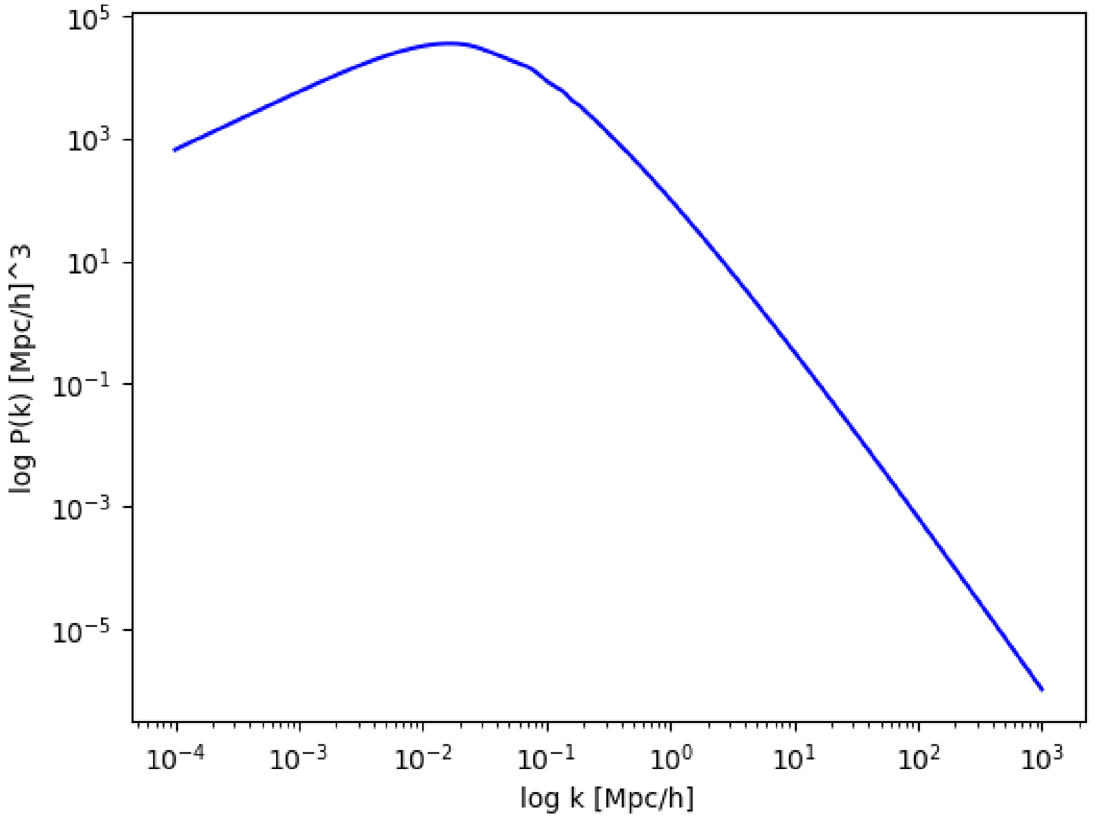


Figure 9: Power spectrum as a function of k, the wavenumber of measured light. This plot is taking an interpolated function P(k), using data files provided to us. I did this to check that the interpolated P(k) returns the same plot for the power spectrum.

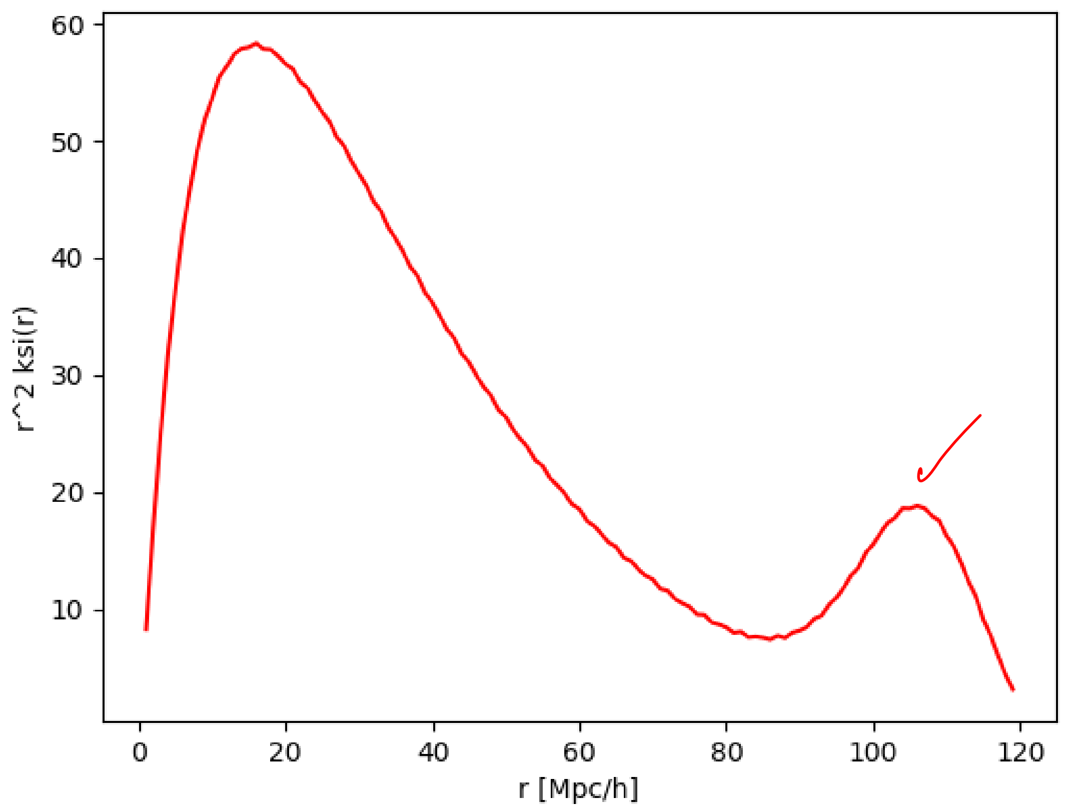


Figure 10: Correlation function scaled by r^2. We observe a peak in this data around r = 105 Mpc/h, known as the baryon acoustic oscillation peak.

✓