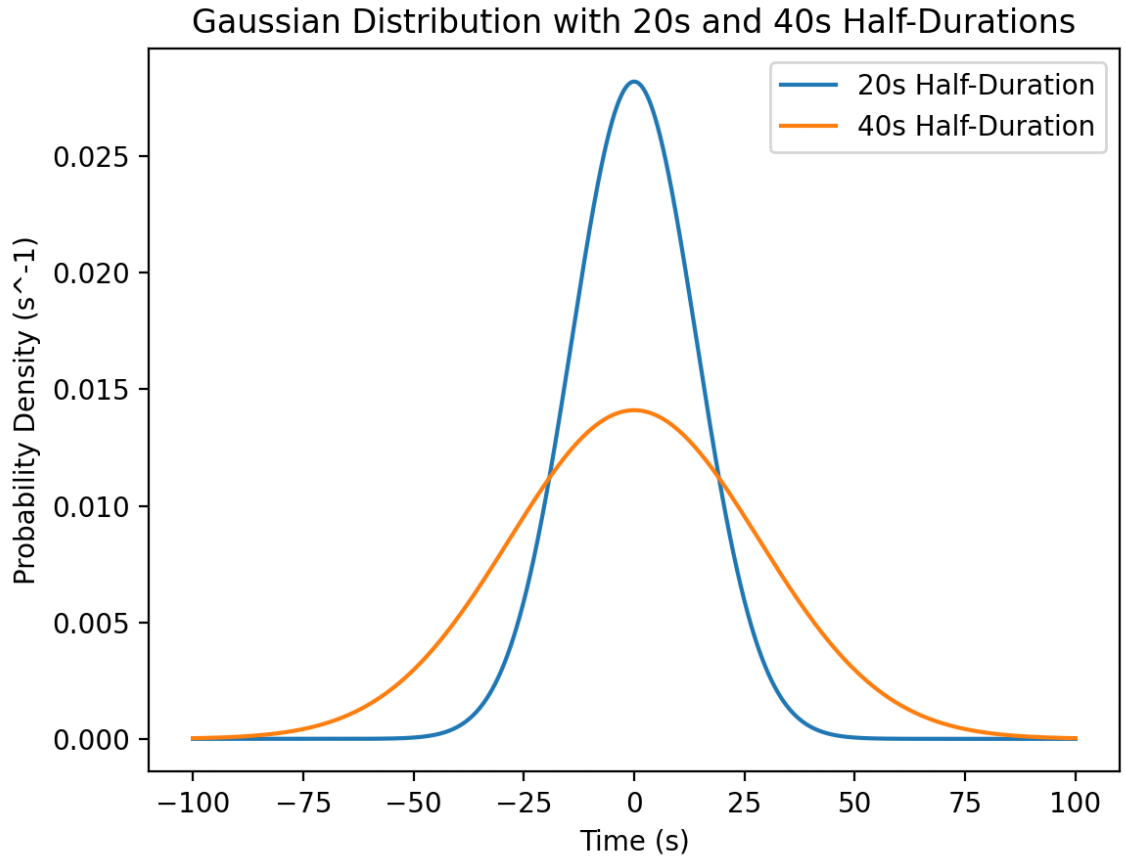
No Collaboration on any questions.

Exercise 1.

1.

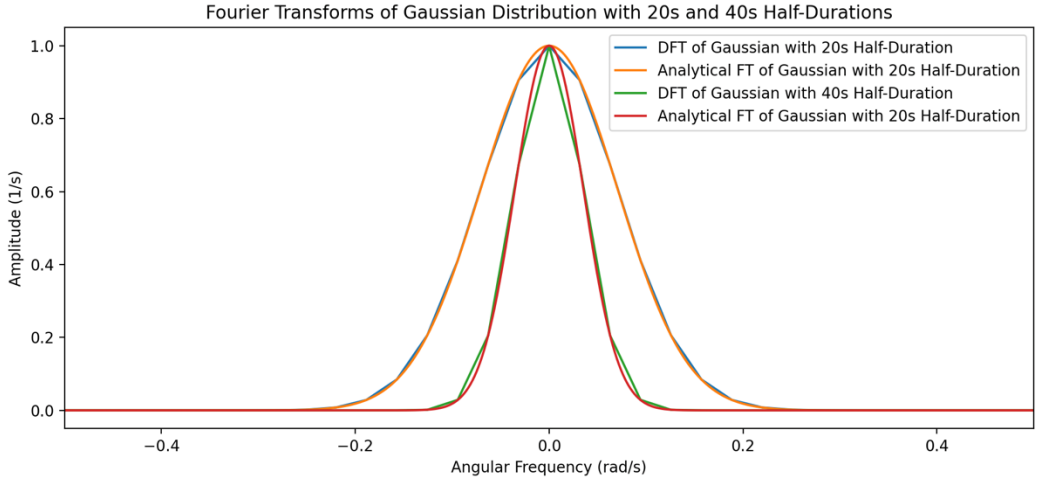



```python
def Gaussian(t,h):
    return (1/(np.sqrt(np.pi)*h))*np.exp(-(t/h)**2)

def Fourier(w,h):
    return np.exp(-((w**2)*(h**2))/4)

x = np.arange(-100,100,0.001)
plt.plot(x,Gaussian(x,20), label = "20s Half-Duration")
plt.plot(x,Gaussian(x,40), label = "40s Half-Duration")
plt.xlabel("Time (s)")
plt.ylabel("Probability Density (1/s)")
plt.title("Gaussian Distribution with 20s and 40s Half-Durations")
plt.legend()
plt.show()
```

2.




```
g1 = Gaussian(x,20)
A = np.fft.fft(g1)*0.001
A = np.fft.fftshift(A)
f_axis1 = np.fft.fftshift(np.fft.fftfreq(len(g1), 0.001) )
w_axis1 = 2*np.pi*f_axis1

g2 = Gaussian(x,40)
B = np.fft.fft(g2)*0.001
B = np.fft.fftshift(B)
f_axis2 = np.fft.fftshift(np.fft.fftfreq(len(g2),0.001))
w_axis2= 2*np.pi*f_axis2

C = Fourier(x,20)
D = Fourier(x,40)

plt.plot(w_axis1,np.abs(A), label = "DFT of Gaussian with 20s Half-Duration")
plt.plot(x,C, label = "Analytical FT of Gaussian with 20s Half-Duration")
plt.plot(w_axis2,np.abs(B), label = "DFT of Gaussian with 40s Half-Duration")
plt.plot(x,D, label = "Analytical FT of Gaussian with 20s Half-Duration")
plt.xlim(-0.5,0.5)
plt.xlabel("Angular Frequency (rad/s)")
plt.ylabel("Amplitude (1/s)")
plt.title("Fourier Transforms of Gaussian Distribution with 20s and 40s Half-Durations")
plt.legend()
plt.show()
```

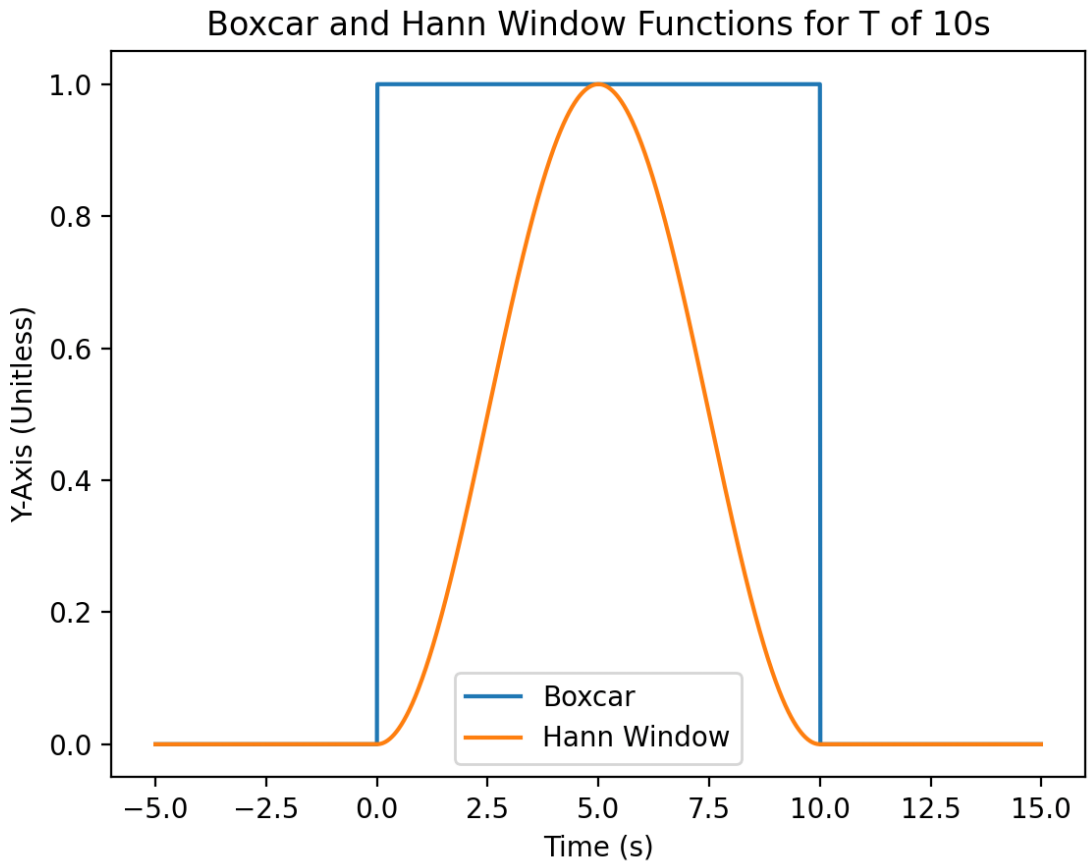
3.

By filtering a signal using a Gaussian via convolution, you can supress values above and below a certain time depending on how short the half-duration of the Gaussian is. The shorter the half-duration, the greater the amount of suppression on values above and below the desired time. For example, filtering by a 20s half-duration will return more strongly the signal at t = 0s and less of the signal above and below that time than filtering with a 40s half-duration.

4.

However, as the Fourier transform of a Gaussian with a shorter half-duration is wider on the frequency axis, the more you filter down a time signal to a precise time the less resolution you have in frequency. This explains the time-frequency uncertainty principle as the more you narrow down the time using a Gaussian filter with shorter half-duration the less you know of the frequency of the signal and the more you filter the frequency with a Fourier transform the less you know of the time.

Exercise 2.
1.



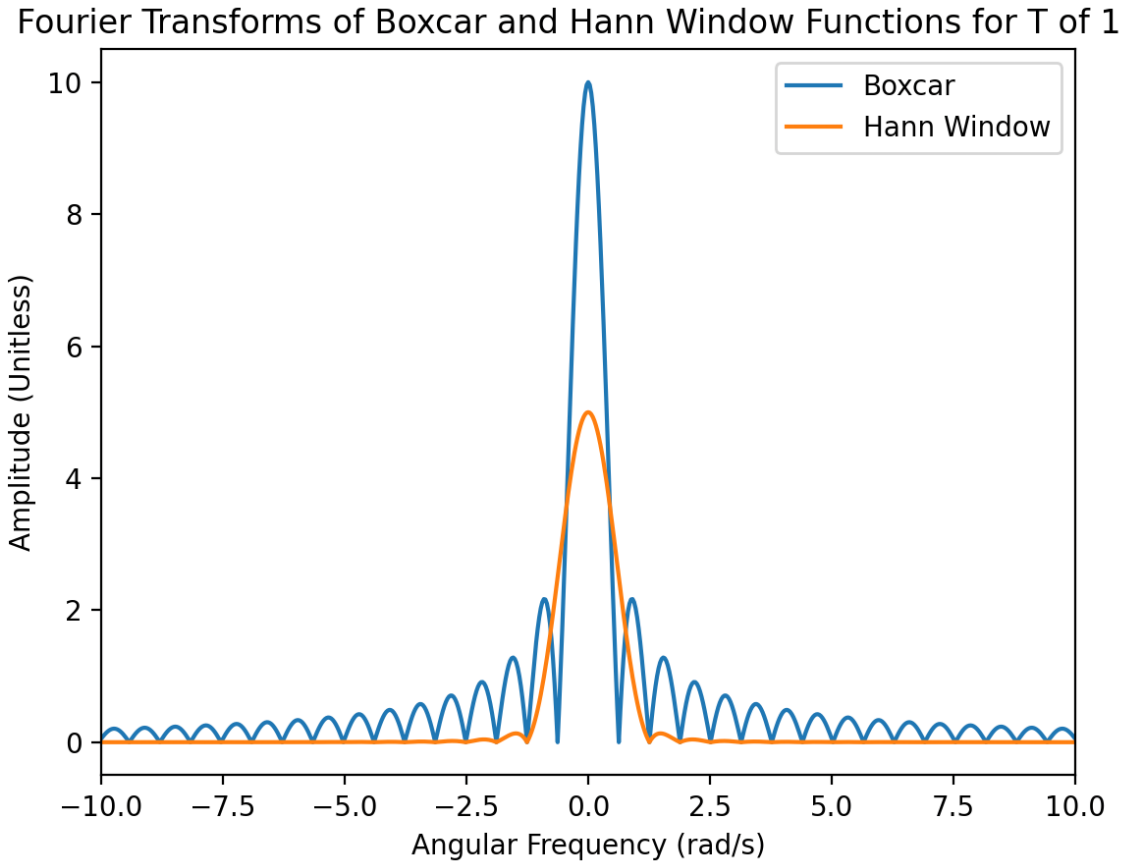
```
def Boxcar(t,T):
    result = []
    for i in t:
        if i >= 0 and i <= T:
            result.append(1)
        else:
            result.append(0)
    return result

def Hann(t,T):
    result = []
    for i in t:
        if i >= 0 and i <= T:
            result.append(0.5*(1-np.cos(2*np.pi*i/T)))
        else:
            result.append(0)
    return result

t = np.arange(-100,100,0.01)

plt.plot(t,Boxcar(t,10), label = "Boxcar")
plt.plot(t,Hann(t,10), label = "Hann Window")
plt.xlim(-5,15)
plt.xlabel("Time (s)")
plt.ylabel("Y-Axis (Unitless)")
plt.title("Boxcar and Hann Window Functions for T of 10s")
plt.legend()
plt.show()
```

3.




```
A = np.fft.fft(Boxcar(t,10))*0.01
A = np.fft.fftshift(A)
f_axis1 = np.fft.fftshift(np.fft.fftfreq(len(A), 0.01))
w_axis1 = 2*np.pi*f_axis1

B = np.fft.fft(Hann(t,10))*0.01
B = np.fft.fftshift(B)
f_axis2 = np.fft.fftshift(np.fft.fftfreq(len(B), 0.01))
w_axis2 = 2*np.pi*f_axis2

plt.plot(w_axis1,np.abs(A), label = "Boxcar")
plt.plot(w_axis2,np.abs(B), label = "Hann Window")

plt.xlim(-10,10)
plt.xlabel("Angular Frequency (rad/s)")
plt.ylabel("Amplitude (Unitless)")
plt.title("Fourier Transforms of Boxcar and Hann Window Functions for T of 10s")
plt.legend()
plt.show()
```

4.
Truncating using the frequency spectrum will weigh frequencies of different values depending on the value of the function used. In all cases, the result will be the preservation of frequencies closest to the centre of the truncation and the dampening of frequencies outside. The specific way frequencies are weighted and dampened will depend on the type of truncation used.

5.
Continuing from the previous part, for example, looking at the truncation effects of the Boxcar FT reveals that while the central frequency is the most heavily weighted with a steeper peak than the Hann FT, it also retains some information about other multiples of the frequencies in decreasing weight. Meanwhile the Hann FT truncates most other frequencies, but the central frequency curve is wider than the Boxcar FT.

Exercise 3.
1.

$$\int_0^\infty k\,(S(k)-1)\sin(kr)\,dk$$

$$= \int_0^\infty k(S(k)-1)\left(\frac{e^{ikr}-e^{-ikr}}{2i}\right)dk$$

$$= \int_0^\infty \frac{1}{2i}k(S(k)-1)e^{ikr}\,dk - \int_0^\infty \frac{1}{2i}k(S(k)-1)e^{-ikr}\,dk$$

$$= \int_0^\infty \frac{1}{2i}k(S(k)-1)e^{ikr}\,dk + \int_{-\infty}^0 \frac{1}{2i}k(S(-k)-1)e^{ikr}\,dk$$

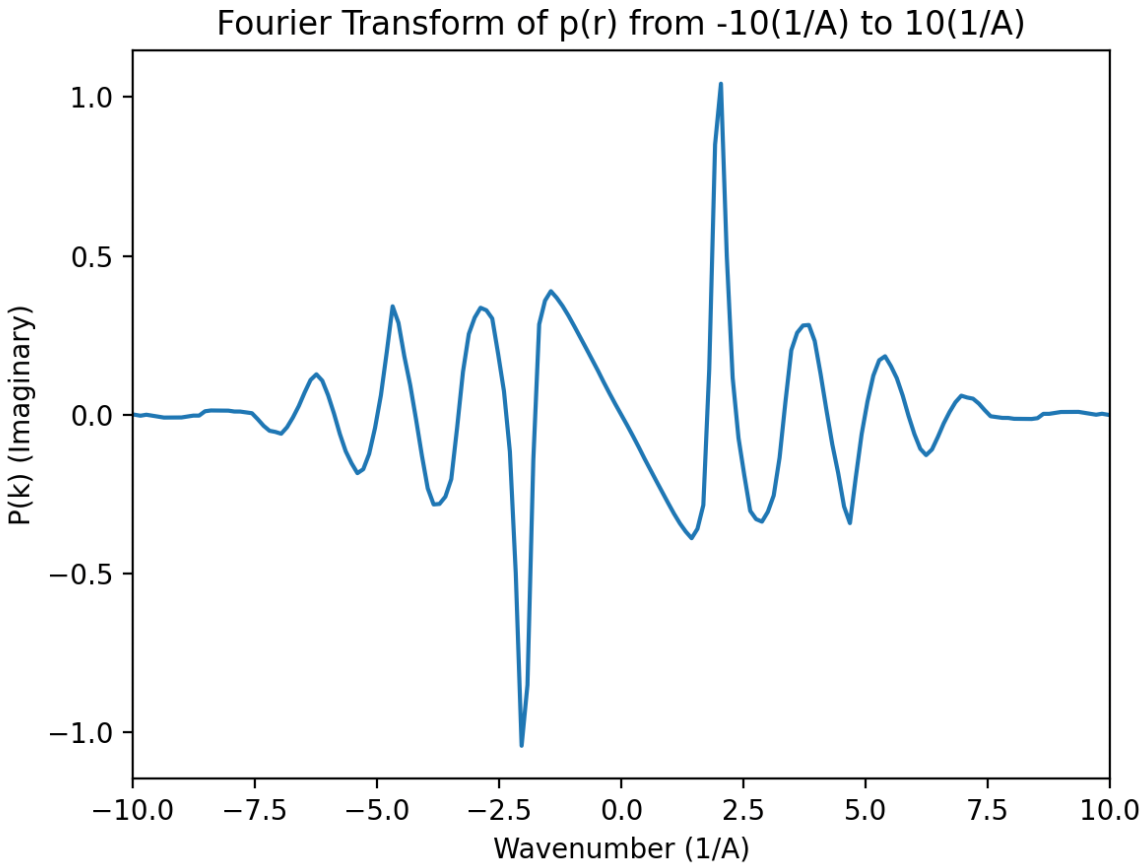
$S(k)$ even so $S(k) = S(-k)$

$$= \int_{-\infty}^\infty \frac{1}{2i}k(S(k)-1)e^{ikr}\,dk$$

2.
Making the substitutions, the Fourier transform is given by:

$$P(k) = \frac{i}{\pi}k(S(k)-1)$$

Which is a purely imaginary function as both k and $(S(k)-1)$ are real and they are multiplied by i. Additionally, the function is odd as k, being linear, is odd, $S(k)$ is given even, so a vertical shift of minus one is also still even, and the product of an odd and even function is odd. The result of this is that $p(r)$ should be a purely real odd function.




```
from argon import YanData, dk, massRho, molWeight, Navogadro

Data = list(YanData[:0:-1]) + list(YanData)
x = np.arange(-len(YanData)*dk,len(YanData)*dk,dk)[1::]

Result = []

for i in range(len(Data)):
    Result += [(1j/np.pi)*x[i]*(Data[i]-1)]

plt.plot(x,np.imag(Result))
plt.xlabel("Wavenumber (1/A)")
plt.ylabel("P(k) (Imaginary)")
plt.title("Fourier Transform of p(r) from -10(1/A) to 10(1/A)")
plt.xlim(-10,10)
plt.show()
```

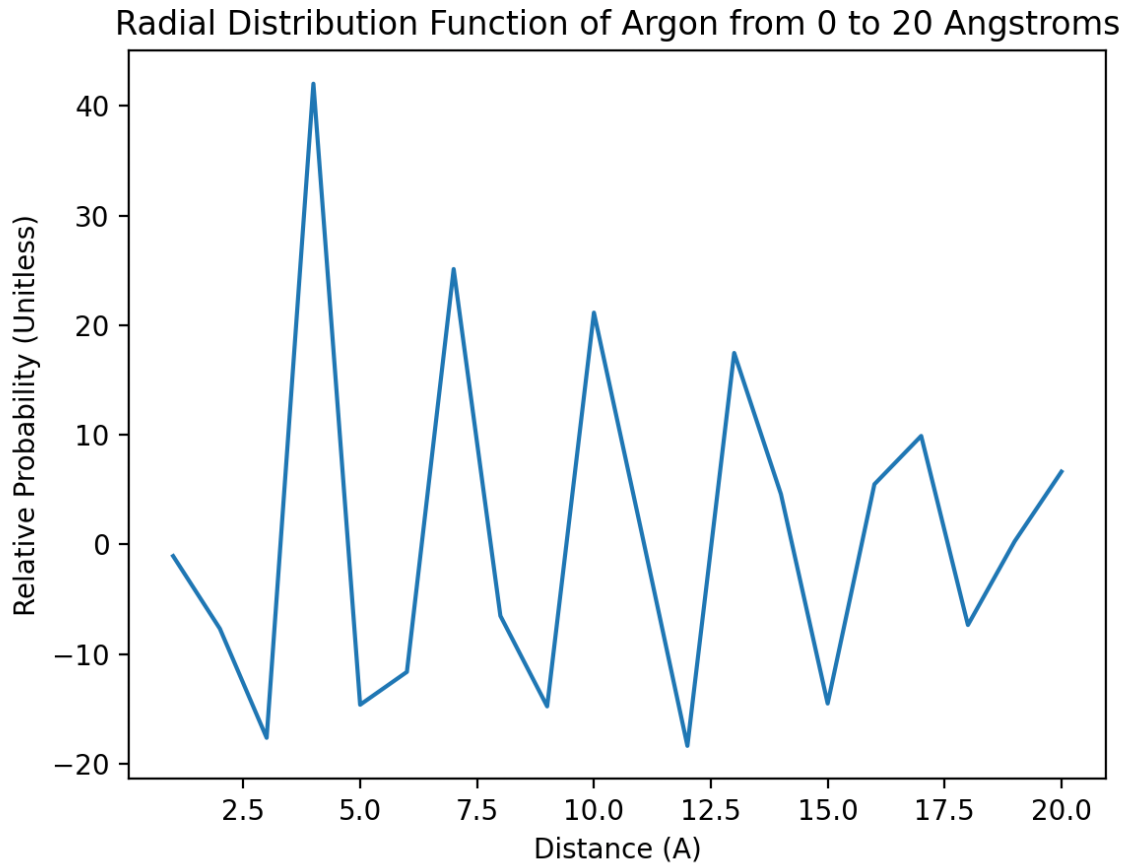
3.

```
rho = (massRho/molWeight)*Navogadro*10**(-24)

def RDF(S,dk,rho,r):
    k = np.arange(0,len(S)*dk,dk)
    return 1 + (1/2*np.pi**2*r)*dk*sum(k*(S-1)*np.sin(k*r))

def RDFcalc(S,dk,rho):
    Result = []
    for i in range(1,21):
        Result.append(RDF(S,dk,rho,i))
    return [Result,list(np.arange(1,21))]
```

4.



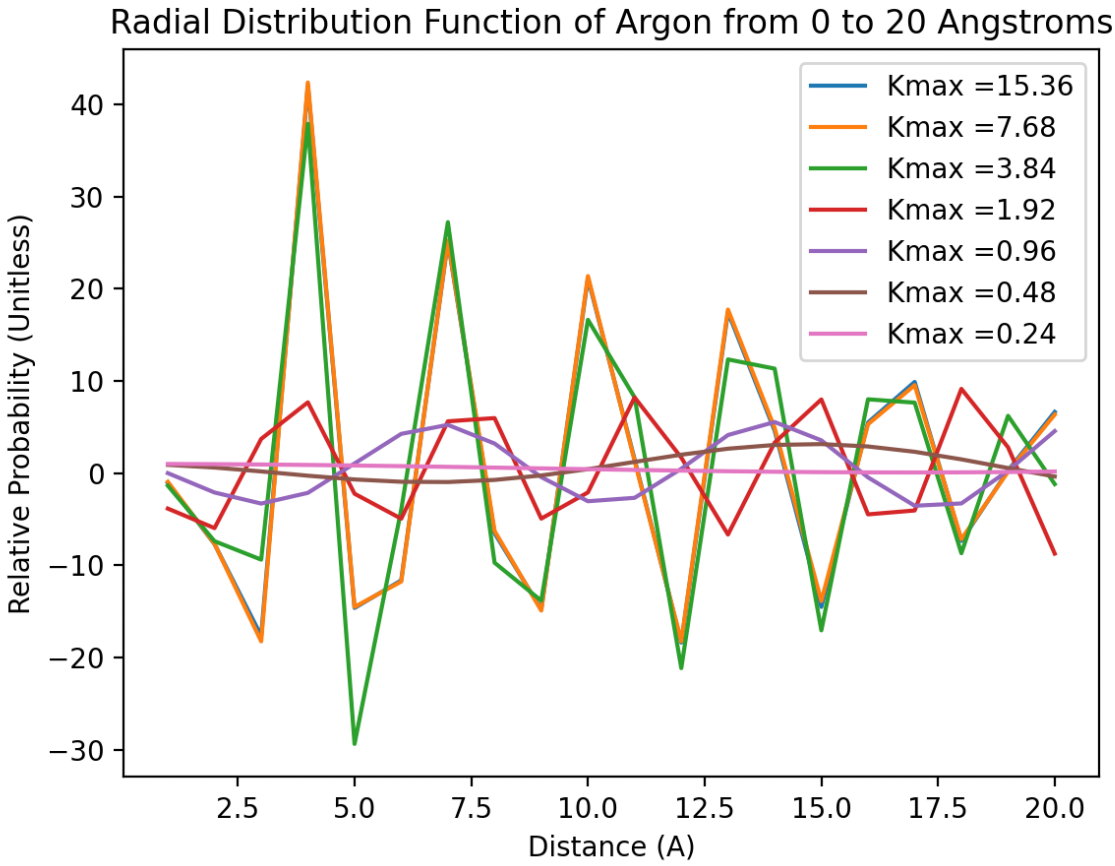
Structure Factor was gathered using a Δk value of 0.12 inverse Angstroms, therefore the results should only be accurate up to slightly above 8 Angstroms.

```
plt.plot(RDFcalc(YanData,dk,rho)[1],RDFcalc(YanData,dk,rho)[0])
plt.xlabel("Distance (A)")
plt.ylabel("Relative Probability (Unitless)")
plt.title("Radial Distribution Function of Argon from 0 to 20 Angstroms")
plt.show()
```

5.

Roughly 5.15 Angstroms factoring in the first and second peaks as the third peak is above 8 Angstroms.

6.

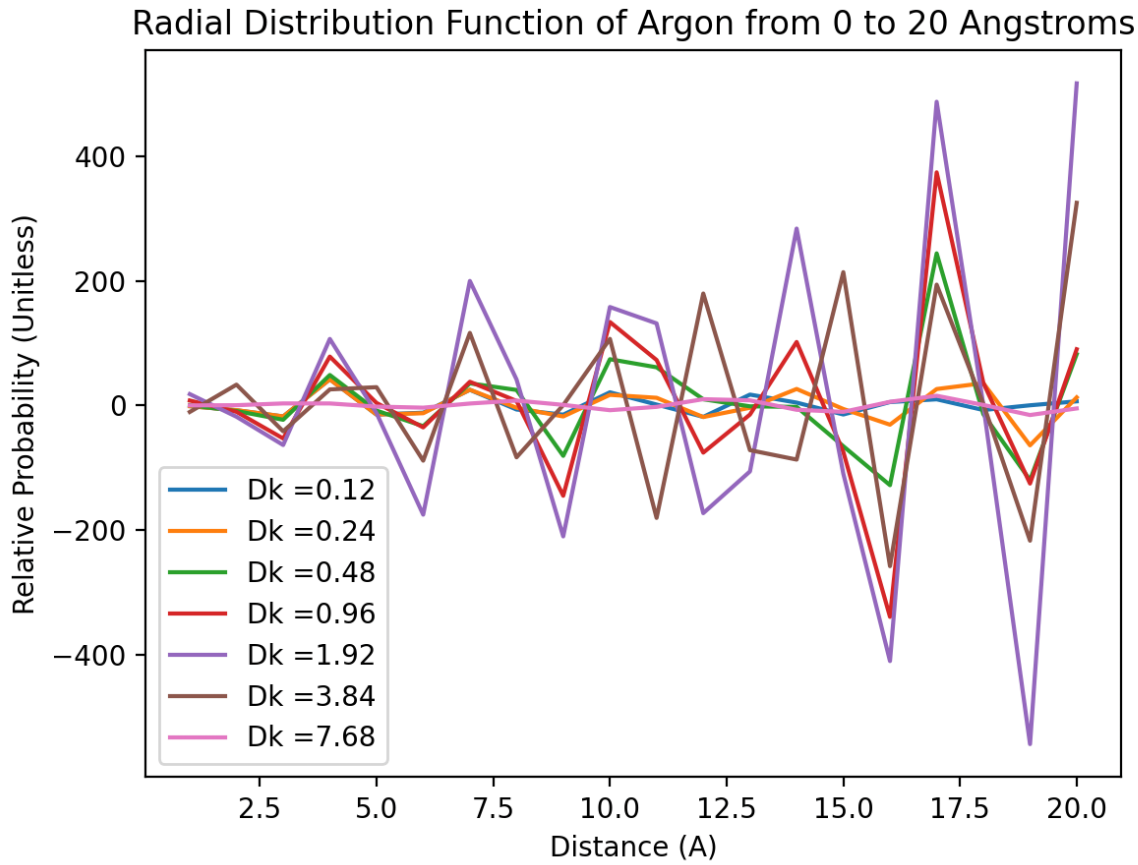


K_{max} = 3.84 Angstroms seems to be the last value that still has distinct peaks. Although below the predicted expected distance of Argon molecules from each other, 3.84 Angstroms is still relatively close and so returns decent peaks; however, the next jump down to 1.92 Angstroms is significantly below and so is no longer able to properly determine the average distance as below that distance it may seem that many different distances are equally probable.

```
YanDataOld = YanData

while len(YanData) > 1:
    plt.plot(RDFcalc(YanData,dk,rho)[1],RDFcalc(YanData,dk,rho)[0], label = "Kmax ="+str(len(YanData)*dk))
    YanData = YanData[0:len(YanData)//2]
plt.xlabel("Distance (A)")
plt.ylabel("Relative Probability (Unitless)")
plt.title("Radial Distribution Function of Argon from 0 to 20 Angstroms")
plt.legend()
plt.show()
```

7.



Only Dk = 0.12 and 0.24 were accurate for the first two peaks. Afterwards up to Dk = 0.96 was still ok but beyond that the first two peaks were inaccurate. This might have to do again with the locations of the peaks at nearly round numbers causing the greater Dk values to over/undershoot and lose the potential to resolve these peaks

```
YanData = YanDataOld

while dk < 8:
    plt.plot(RDFcalc(YanData,dk,rho)[1],RDFcalc(YanData,dk,rho)[0], label = "Dk ="+str(dk))
    dk = dk*2
    YanData = YanData[0:-1:2]
plt.xlabel("Distance (A)")
plt.ylabel("Relative Probability (Unitless)")
plt.title("Radial Distribution Function of Argon from 0 to 20 Angstroms")
plt.legend()
plt.show()
```