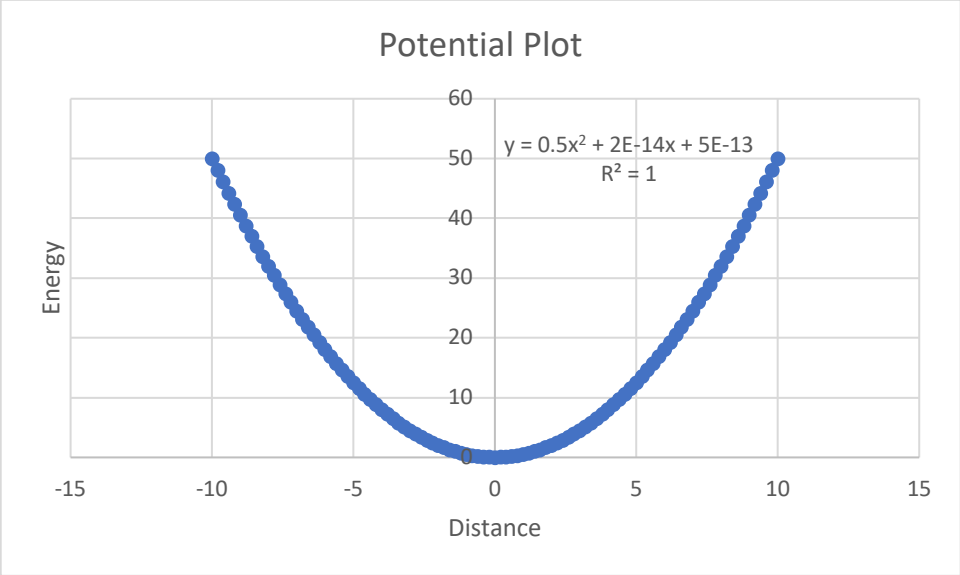
Sahad Vasanji | Student ID# 20603358 | April 1, 2022

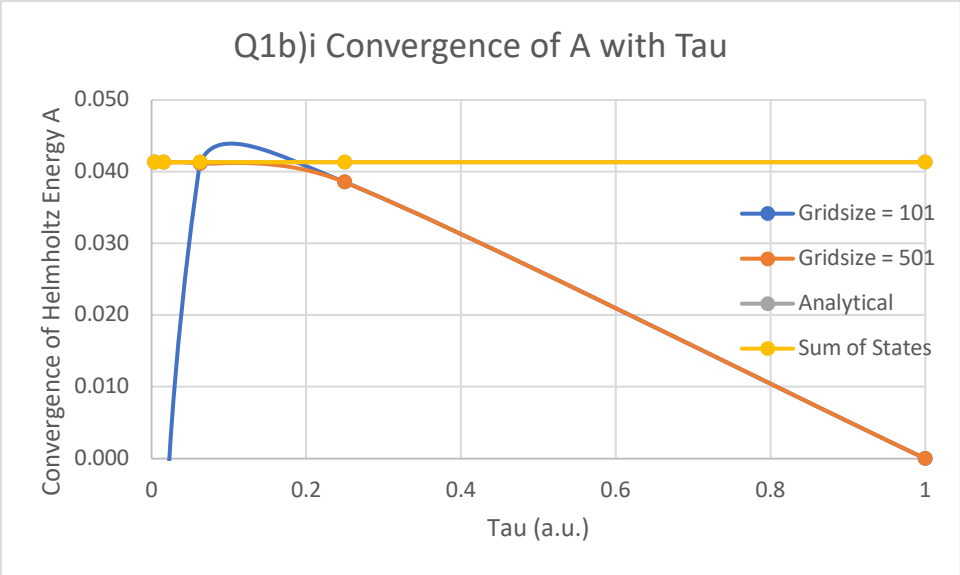
Q1a

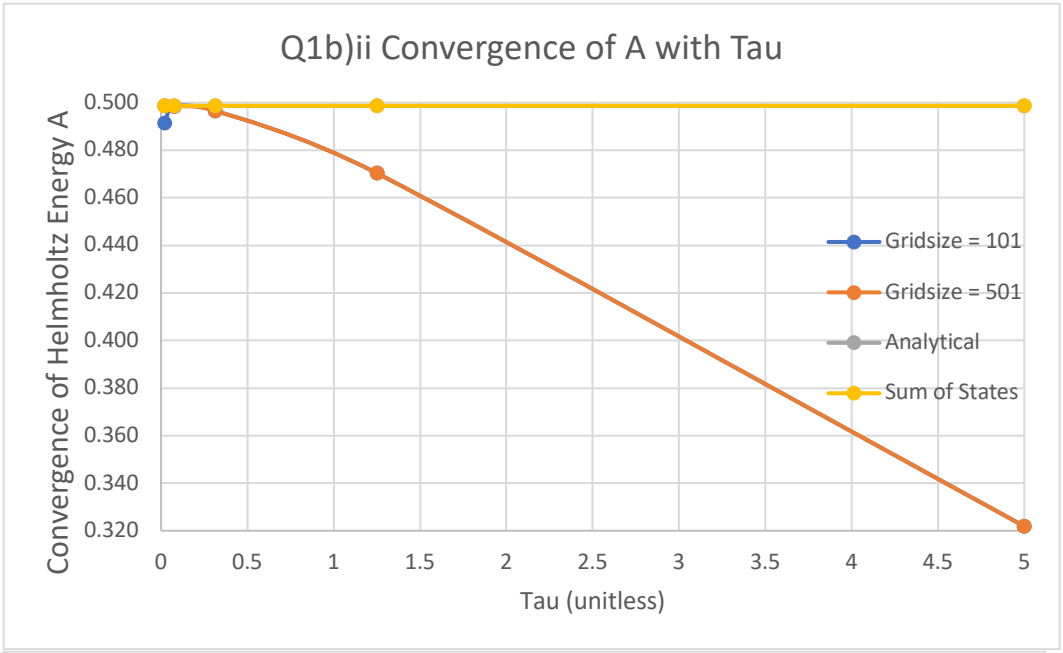


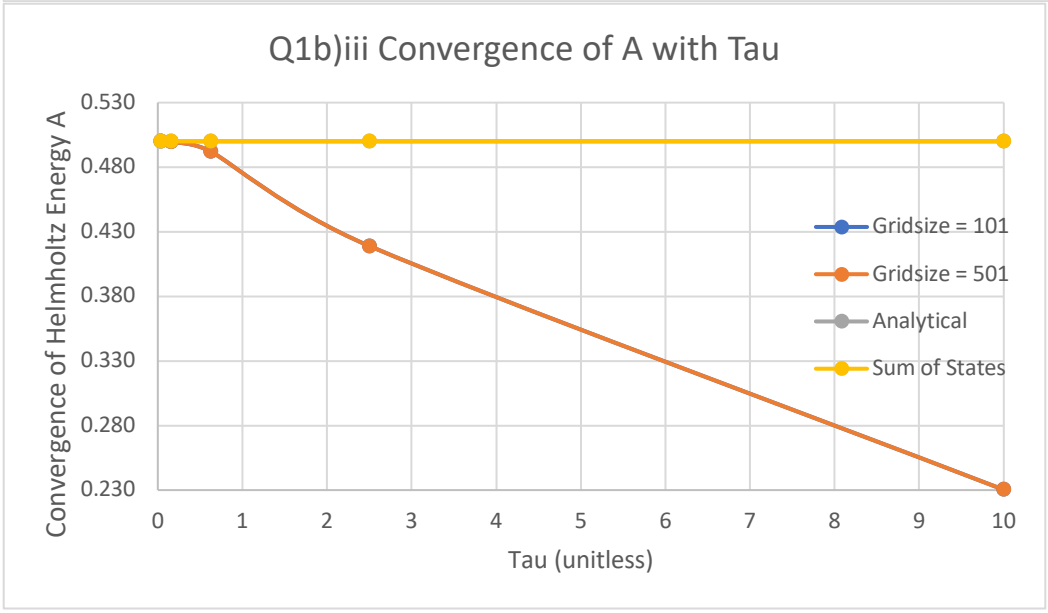
In this code, Beta and Tau are not used in the calculation of the potential: the only inputs used are xmax and gridsize, explaining why the potential is the same for all possible combinations of Beta and Tau. This is also because Beta (and thus Tau) are related to the kinetic energy, not the potential energy.

Q1b

For Parts B and C of Questions 1&2, a short For loop script was used in addition to a modified nmm.py to be used as a user defined function. The gridsize value was manually changed, and the script run again.





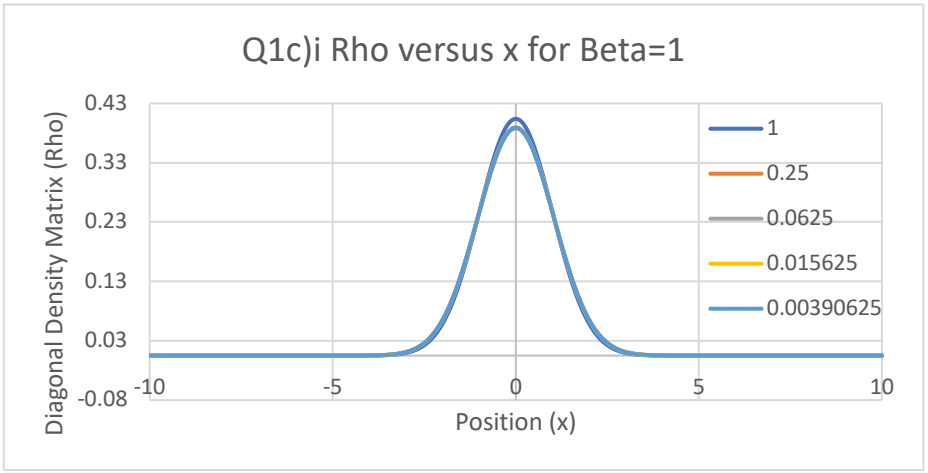


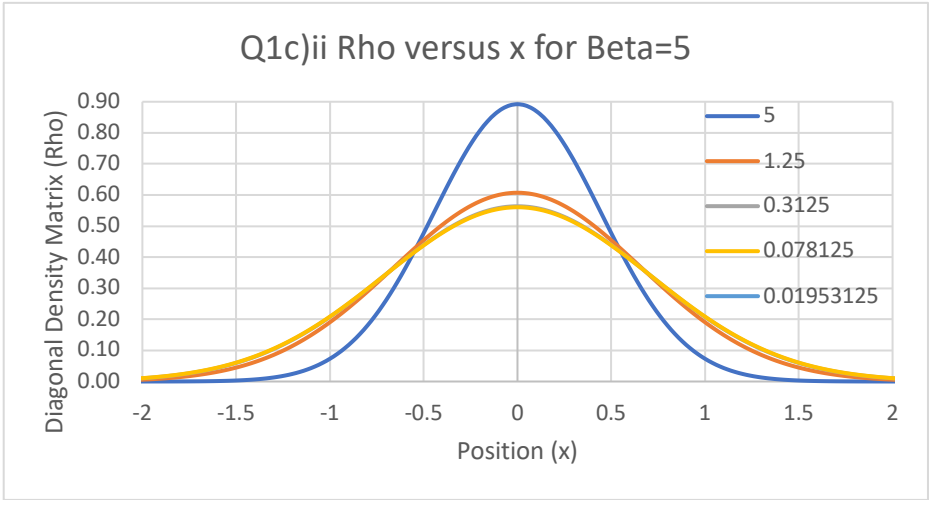
For all 3 Beta values as the number of beads increase initially (i.e., as Tau decreases initially), a linear relation is seen for the Path Integral Matrix Multiplication (PIMM)'s value of A, converging towards the analytical value of A. In other words, the number of beads generally leads to a better approximation of the free energy with linear scaling. This is analogous to the scaling of computational cost, which also scales linearly with the increase in bead number. However, for Beta = 1 and 5, 256 beads does not provide the most accurate free energy value. Increasing the grid size from 101 to 501 substantially increases the convergence's accuracy to the analytical value, something particularly exemplified in the Beta=1,5 examples. This makes sense, considering that more grid points constituting the coordinates will provide an increase in accuracy, up to a certain threshold. The Analytical and Sum of States values for the free energy

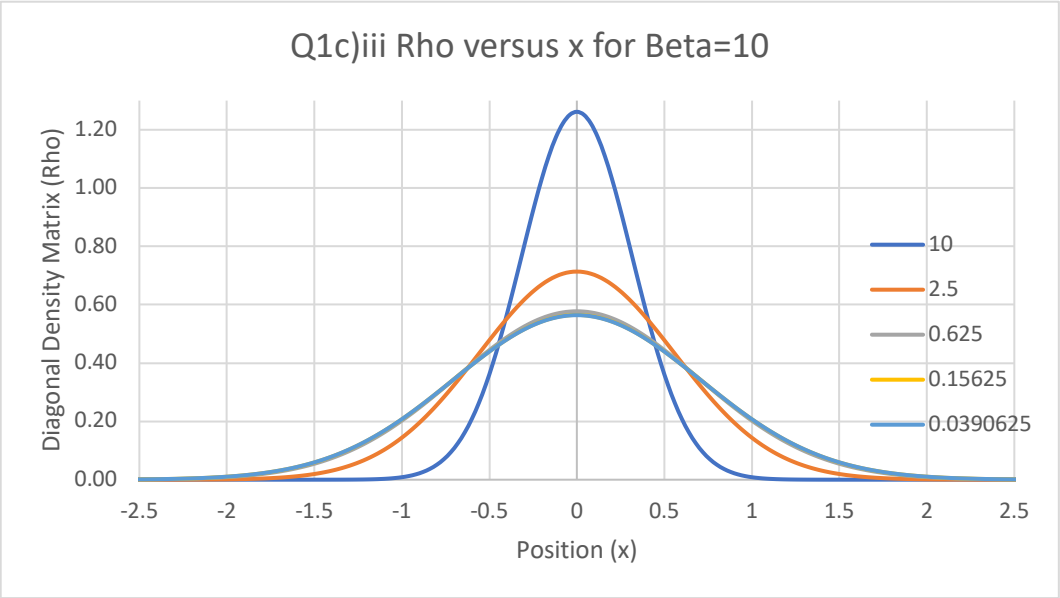
closely match (within 13 decimals for Beta = 1,5; within 12 decimal places for Beta=10), suggesting both can be used as veridical benchmarks for the PIMM's convergence.

For higher bead values (smaller Tau) at Beta=1, a logarithmic convergence is seen, which then becomes effectively linear for Beta=5, and then polynomial followed by logarithmic for Beta=10. Considering additional beads increase computational time by a factor equal to the number of beads, Q1b suggests that adding more beads is much more advantageous at lower temperatures (higher Beta values), but will only ensure faster convergence up until a certain point (i.e., when the polynomial convergence changes to logarithmic). This illustrates what is taught in the coursenotes, where nuclear quantum effects are particularly important to account for at lower temperatures (and, intuitively, for lighter atomic systems).

Q1c (Note the legend indicates the Tau values of the curves)**

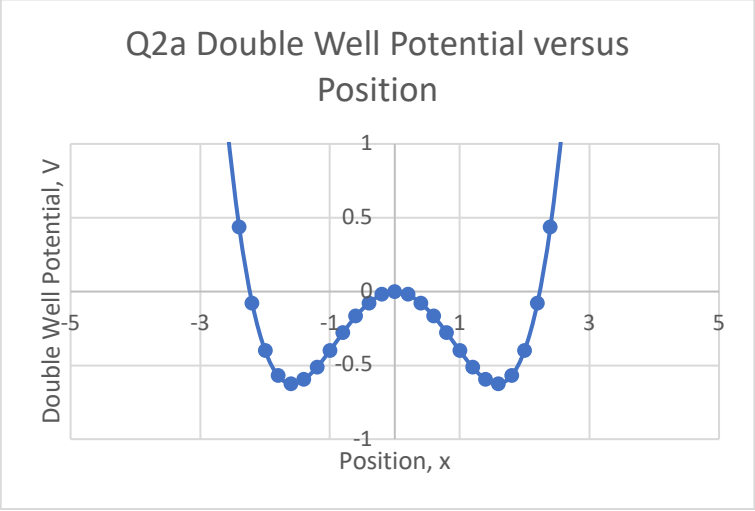






Apart from the expected convergence to a central minimum for increasing bead numbers, for all graphs we see that as Beta increases (i.e., Temperature decreases), the Density matrix's converged-upon minima increases. Furthermore, for the lower temperature (i.e., higher Beta) graphs, Rho's central maxima decreases much more for increasing bead numbers. This can be attributed to larger decreases in Tau for the same number of beads, since Tau = Beta/P. Conversely, for the higher temperature (i.e., lower Beta) graphs, Rho varies much less with Tau, indicating a much smaller dependency on the number of beads. A phenomena that's much more noticeable in the higher Beta graphs is the density matrix's broadening as the number of beads is increased. This could be because with more beads present in the model, there's a higher probability of more spatial regions being simultaneously occupied.

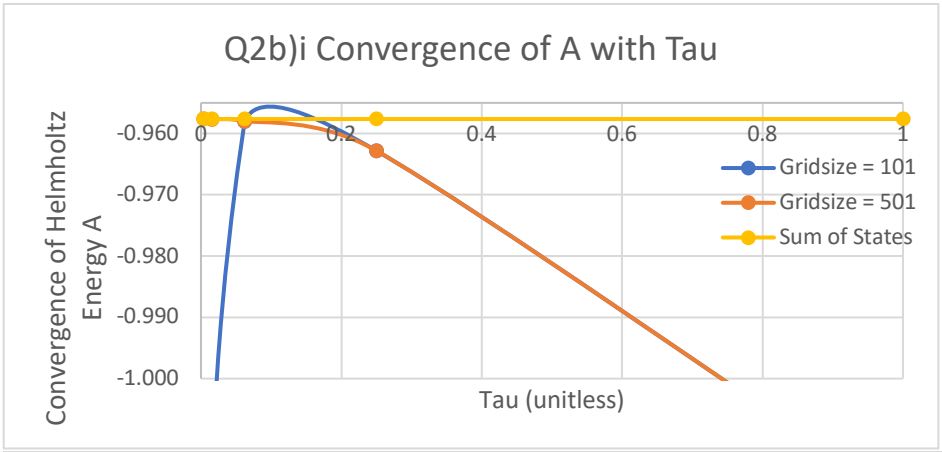
Q2a

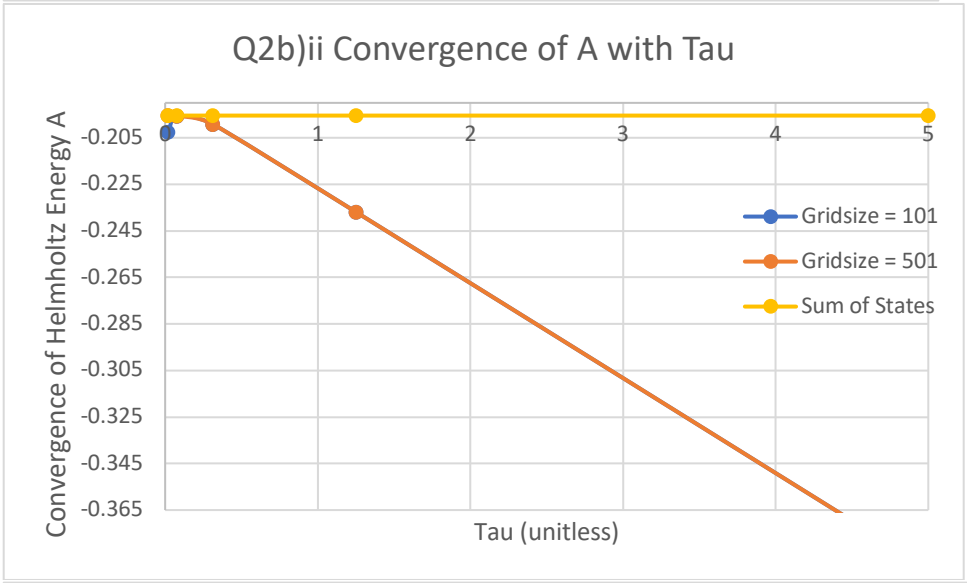


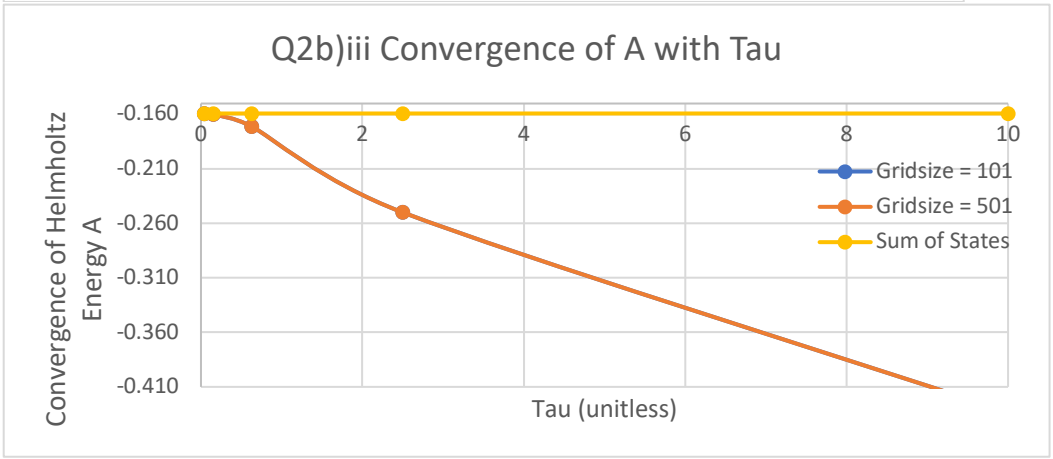
Similar to Q1a, the calculation of the potential seen in the code does not involve Beta or Tau, explaining why this potential remains unchanged for various combinations of these two variables. Beta and Tau, both depending on temperature, are related to the kinetic energy if

anything, and not the potential energy. Xmax, and gridsize are the only two inputs related to calculation of this double well potential.

Q2b





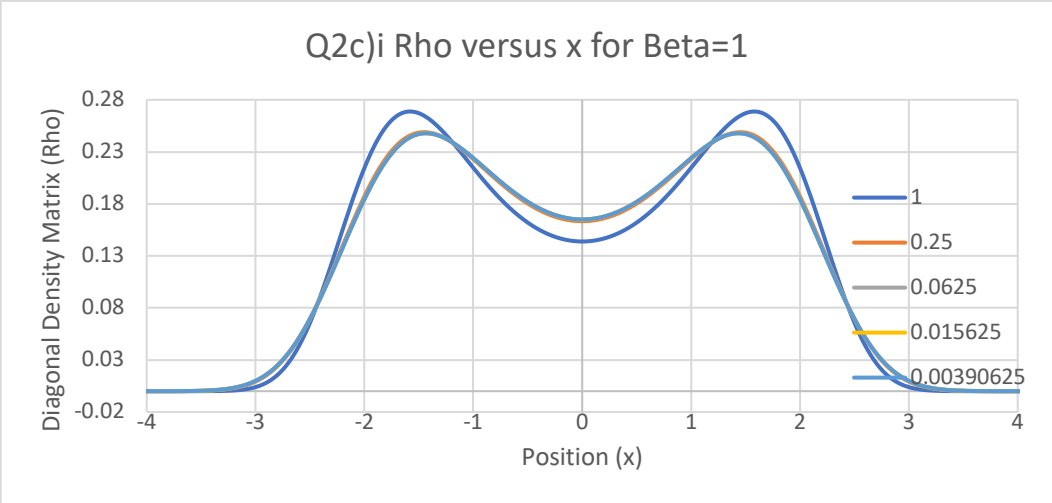


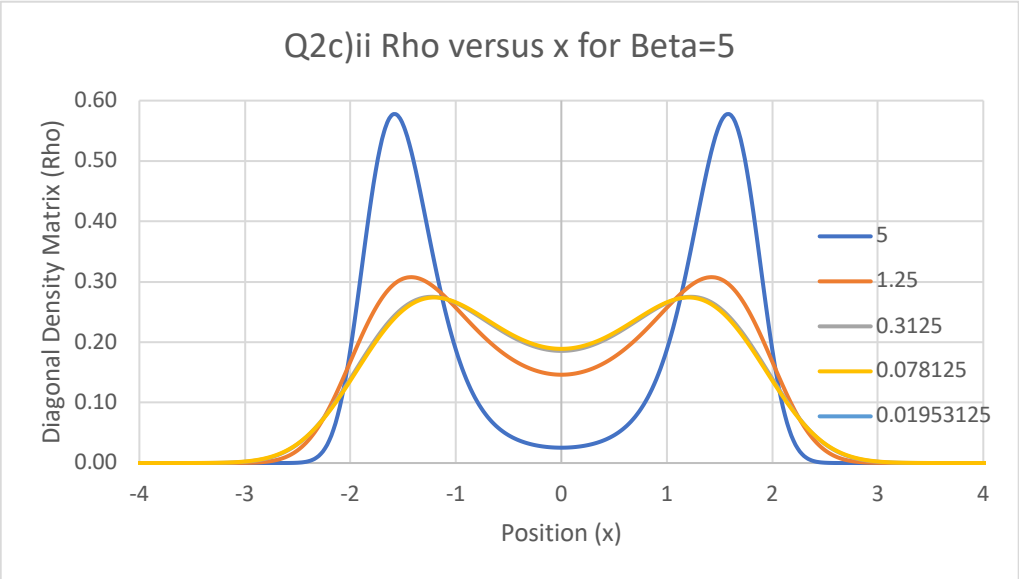
Question 2b's graphs are nigh-congruent to Q1b's, albeit with different Free energy values. This suggests that the Double Well potential demonstrates a similar convergence behavior to the

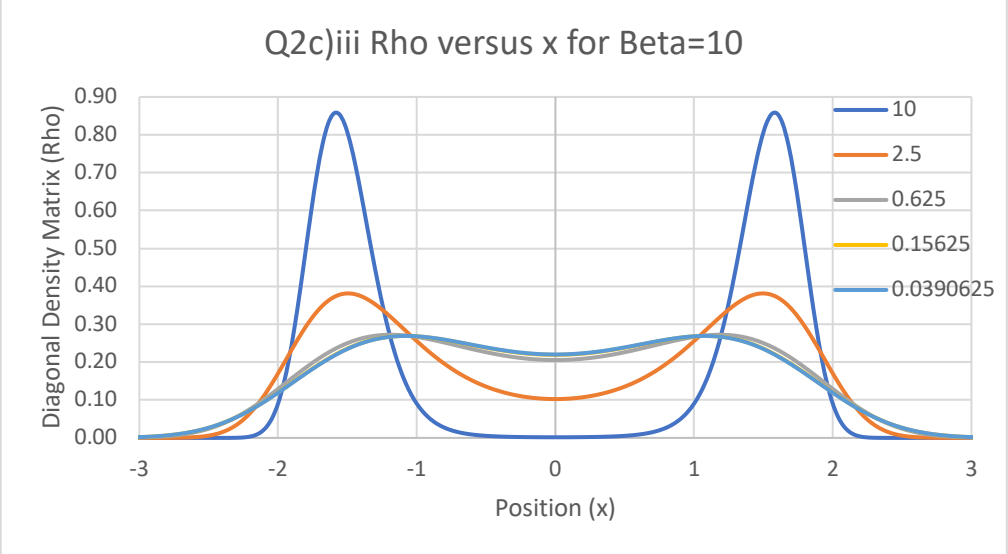
Harmonic Oscillator potential. For the smaller grid size and lower beta we see that increasing the Bead number converges only up until 16 beads, beyond which the free energy diverges. We also see the linear relation between free energy convergence and Tau for Beta = 1 and 5, and slowly changing into a polynomial relation for higher P values with Beta=10. At lower Beta values (higher temperatures) it's noticed that the Gridsize is particular important for ensuring convergence: the 501 option provided a better accuracy than the 101 option, especially for smaller Tau (i.e., higher bead values).

For higher bead values (smaller Tau) at Beta=1, a logarithmic convergence is seen, which then becomes effectively linear for Beta=5, and then polynomial followed by logarithmic for Beta=10. Considering additional beads increase computational time by a factor equal to the number of beads, Q1b suggests that adding more beads is much more advantageous at lower temperatures (higher Beta values), but will only ensure faster convergence up until a certain point (i.e., when the polynomial convergence changes to logarithmic). This illustrates what is taught in the coursenotes, where nuclear quantum effects are particularly important to account for at lower temperatures (and, intuitively, for lighter atomic systems).

Q2c (Note the legend indicates the Tau values of the curves)**







Similar to Q1b, the Double Well maxima values have a proportional relationship with Beta (as Beta increases, the maxima's values increase), and display narrowing for increased Beta. This is expected, since at higher Beta (lower temperature), the density is more likely to be concentrated near the centre of the well. What's different from Q1b is the local minimum's value is inversely proportional to the Beta value, and its width proportional to the Beta value. The broadening of Rho's minima for increased Beta (lower temperature) is expected, since at lower temperatures it's increasingly unlikely to find particles localized at the Double Well's local maxima. Rho's dependence on Tau involves several behaviors. Local minima narrowing is seen as the bead number increases: this is because having more beads increases the probability of the density not being confined to just the centre of the wells.

Script Used for Parts b and c of Questions 1&2

```
1. import numpy as np
2. from sys import argv, exit
```

```
3.  import matplotlib.pyplot as plt
4.
5.  bvals = [1, 5, 10]
6.  pvals = [1, 4, 16, 64, 256]
7.  gsvals = [101, 501]
8.  xmax = 10
9.
10.
11. for b in bvals:
12.     A_array = open('A values for Beta = ' + str(b) + ' ' + \
13.                    'Gridsize=' + str(gsvals[1]) ,  'w')
14.     for p in pvals:
15.         A = nmm(b, p, xmax, gsvals[1], 'DW')
16.         A_array.write(str(A[0]) + ' ' + str(A[1]) + ' ' + str(A[2]) + '\n')
17.
18.     A_array.close()
```

Modified nmm.py (now a user-defined function that returns the 3 free energy values)

```
1.  def nmm(beta, P, xmax, gridSize, potType):
2.
3.      import numpy as np
4.      from sys import argv, exit
5.      import matplotlib.pyplot as plt
6.
7.      def kinetic(size,mass,dx):
8.          T=np.zeros((size,size),float)
9.          h_bar=1.
10.         for i in range(size):
11.             for ip in range(size):
12.                 T1 = h_bar*h_bar / (2.0*mass*dx*dx) * np.power(-1.,i-ip);
13.                 if i==ip:
14.                     T[i,ip] = T1 * (np.pi*np.pi/3.0);
15.                 else:
16.                     T[i,ip] = T1 * 2.0/((i-ip)*(i-ip))
17.         return T
18.
19.     mass=1
20.     hbar=1.
21.     omega=1.
22.
23.     # beta=1/kBT value
24.     beta = float(beta)
25.     P = int(P)
26.     tau=beta/P
27.     xmax = float(xmax)
28.
29.     size = int(gridSize)
30.     dx=2.*xmax/(size-1.)
31.     grid=np.zeros(size,float)
32.     for i in range(size):
33.          grid[i]=-xmax+i*dx
34.
35.
36.
37.     pot_type = potType
38.
39.     yliml=-1.
40.     ylimh=20.
```

```
    if pot_type == 'HO':
        def V(x):
            return (1./2.)*(mass*omega**2)*x**2
        '''
        print('-------------------------------------------------------------------------
-----------------')
        print('For the harmonic oscillator, we know the partition function and free
energy exactly')
        print('Z(analytical;harmonic,beta='+str(beta)+')=
',1./(2.*np.sinh(beta*hbar*omega/2.)))
        print('A(analytical;harmonic,beta='+str(beta)+')= ',-
np.log(1./(2.*np.sinh(beta*hbar*omega/2.)))/beta)
        print('-------------------------------------------------------------------------
-----------------')
        '''
    elif pot_type == 'DW':
        a = -.5
        b = .1
        def V(x):
            return a*x**2+b*x**4
    else:
        print('Incorrect potential specified')
        exit()

    #Build the free particle density matrix
    rho_free=np.zeros((size,size),float)
    for i1,x1 in enumerate(grid):
        for i2,x2 in enumerate(grid):
            rho_free[i1,i2]=np.sqrt(mass/(2.*np.pi*tau*hbar*hbar))*(np.exp(-
(mass/(2.*hbar*hbar*tau))*(x1-x2)*(x1-x2)))

    #Build the potential density matrix
    rho_potential=np.zeros((size),float)
    potential=np.zeros((size),float)
    for i1,x1 in enumerate(grid):
        potential[i1]=V(x1)
        rho_potential[i1]=np.exp(-(tau/2.)*potential[i1])

    list_i1 = []
    list_x1 = []
    #Output the potential to a file
    potential_out=open('V_'+pot_type+'_'+str(beta)+'_'+str(tau),'w')
    for i1,x1 in enumerate(grid):
        #i1 is the enumeration of all items in the grid, 1-100
        #x1 is -10 to 10
        potential_out.write(str(x1)+' '+str(potential[i1])+'\n')
    potential_out.close()

    #for i1,x1 in enumerate(grid):
    #    list_i1.append(int(i1))
    #    list_x1.append(int(x1))
    #print(list_i1)
    #print(list_x1)

    ##
    #print(grid)
    ##

    # construct the high temperature density matrix
    rho_tau=np.zeros((size,size),float)
    for i1 in range(size):
```

```
        for i2 in range(size):
            rho_tau[i1,i2]=rho_potential[i1]*rho_free[i1,i2]*rho_potential[i2]

    # form the density matrix via matrix multiplication
            #set initial value of rho
            rho_beta=rho_tau.copy()

            for k in range(P-1):
                rho_beta=dx*np.dot(rho_beta,rho_tau)

            # calculate partition function from the trace of rho
            Z=0.
            Z_tau=0.
            V_estim=0.
            for i in range(size):
                Z_tau+=rho_tau[i,i]*dx
                Z+=rho_beta[i,i]*dx
                V_estim+=rho_beta[i,i]*dx*potential[i]
            '''
            print('------------------------------------------------------------------------
    ---------------------')
            print('Path integral matrix multiplication')
            print('Z(beta=',beta,',tau=',tau,')= ',Z)
            print('A(beta=',beta,',tau=',tau,')= ',-np.log(Z)/beta)
            print('------------------------------------------------------------------------
    ---------------------')
            '''
            rho_x = []
            rho_y = []
            rho_beta_out=open('rho_'+pot_type+'_'+str(beta)+'_'+str(tau),'w')
            for i1,x1 in enumerate(grid):
                rho_x.append(grid[i1])
                rho_y.append(rho_beta[i1,i1]/Z)
                #rho_beta_out.write(str(x1)+' '+str(rho_beta[i1,i1]/Z/x1/x1)+'\n')
                rho_beta_out.write(str(x1)+' '+str(rho_beta[i1,i1]/Z)+'\n')
            rho_beta_out.close()

            #plt.plot(rho_x,rho_y)
            #plt.savefig('test.pdf')


            # compare Z Trotter, Analytical, truncated sum over states
            # Hamiltonian matrix
            H=kinetic(size,mass,dx)
            for i in range(size):
                    H[i,i]+=potential[i]
            eig_vals,eig_vecs=np.linalg.eig(H)

            ix = np.argsort(eig_vals)
            eig_vals_sorted = eig_vals[ix]
            eig_vecs_sorted = eig_vecs[:,ix]

            # sum over states
            Z_sos=0.
            for e in eig_vals_sorted:
                Z_sos+=np.exp(-beta*e)
            '''
            print('------------------------------------------------------------------------
    ---------------------')
            print('Sum over states solution')
            print('Z_sos= ',Z_sos)
```

```
154.            print('A_sos(beta=',beta,')= ',-np.log(Z_sos)/beta)
155.            print('------------------------------------------------------------------------
    ----------------------')
156.            '''
157.            #make the attributes that must be accessed
158.            nmm.beta = beta
159.            nmm.P = P
160.            nmm.xmax = xmax
161.            nmm.size = size
162.            nmm.pot_type = pot_type
163.
164.            A_anal= -np.log(1./(2.*np.sinh(beta*hbar*omega/2.)))/beta
165.            A_pimm = -np.log(Z)/beta
166.            A_sos = -np.log(Z_sos)/beta
167.
168.
169.            return A_anal, A_pimm, A_sos
```