Quantum Mechanics II Computational Project

Alexander Lange, Sara Ratliff, Roman Kosarzycki

December 5, 2020

Part 1

To perform Gauss-Jordan elimination with pivoting, we will consider an inhomogeneous system of N linear equations, which can be represented in the following matrix form:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} \qquad [1]$$

The Gauss-Jordan algorithm transforms the $N \times N$ matrix $\mathbf{A}$ into a triangular matrix:

$$\begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1N} \\ 0 & a'_{22} & \cdots & a'_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a'_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_N \end{pmatrix} \qquad [2]$$

First, we created the augment matrix $\tilde{\mathbf{A}}$ by appending the vector $\mathbf{b}$ to form a $N \times (N+1)$ matrix:

$$\tilde{\mathbf{A}} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2N} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} & b_N \end{pmatrix} \qquad [3]$$

Given this augment matrix, we then go row by row, first pivoting and then transforming each variable a_{ji}. Pivoting is the process of finding the largest value in the first column a_{j1} (including the last column) and moving that whole row to the j^{th} row. Then each of individual components are transformed according to the following:

$$a_{ij} \longrightarrow a'_{ij} = a_{ij} - \frac{a_{ik}}{a'_{kk}} a'_{kj} \; ; \; k = 1, 2, \cdots, N \; ; \; i = k+1, k+2, \cdots, N \; ; \; j = 1, 2, \cdots, N+1 \qquad [4]$$

This process was accomplished in our code in the following manner for an $N \times N$ array A:

```
#Define functions first
```

```
def pivot(A,k):
        Define Amax to be first diagonal term A[k,k]
        if  any other element in same col > Amax,
           for all elements in col
                Amax = larger element

def GaussianEliminationWPivot(A):
        #Elimination
        for i in range(N):
            Recall function pivot(A,i)
            Elimination for each element in each column
                        Eq. 43 from writeup
        x=Nx1 empty matrix
        #Backward Substitution method
        for i in range(N):
                sum=0
                for j in range(N):
                        EQ. 37 from write-up.
                        Sum value is changed in this procedure.
        #Returns solutions to matrix equation
        return x

def random(i,N,valmin,valmax):
    Define N number of 2 random matrices for testing.
    A = random generated number of a NxN between min value and
       max value
    B = random generated number of a 1xN between min value and
       max value
    return A,B


def ToList(A,B):
    Convert A to list (dtype required)
    Convert B to list
    Append B to A as new col

    return A

#Check by matrix multiplication
A,B = random(1,10,0,10)
A_List=ToList(A,B)
sols = GaussianEliminationWPivot(A_List)
check = abs(np.matmul(A,sols) - B)< Some Margin of
   Error (1e-14 ~50%)
print(np.matmul(A,sols))
print(check)
```

Part 2

In order to perform the Gaussian integration, we used the following relation:

$$\int_0^\infty dq\, f(q) = \sum_{i=1}^{N} \tilde{\omega}_i f_i + R_N[f,q]\ ,\ f_i = f(q_i)\ ,\ \tilde{\omega}_i = \omega_i \left[\frac{dq}{dx}\right]_{x=x_i} \qquad [5]$$

The term R_N represents the remainder, but we will choose a sufficiently large N such that the remainder is negligible. The weight functions are defined using Gauss-Legendre quadrature. For our purposes, we will use the following definition of q:

$$q(x) = q_0 \frac{1+x}{1-x} \qquad [6]$$

Eq. (6) allows q to vary from 0 to ∞ when x varies from -1 to 1. Our code completes this process in the following manner:

```
#Define functions

def func(q):
        Define the function being integrated
        return function(q)

def q(x):
        return q(x) from eq. (6)

def weight_func(Mesh_size):
        return mesh asbsesca and weight for a given mesh size

def GaussInteg(X,N,K):
        Takes in asbseca, mesh size, and mesh weights
        for j in range (N):
                Sum in eq. (22) from write-up
        return sum

#Set mesh size

N=mesh size

#Body of code

asbsesca = weight_func(N)[0]
weights = weight_func(N)[1]
Integral=GaussInteg(asbsesca,N,weights)
```

Part 3

Next, the potential matrix elements $U(p, q)$ were generated for a given mesh.

Given the potential:

$$V(r) = V_R \frac{e^{-\mu_R r}}{r} - V_A \frac{e^{-\mu_A r}}{r} \quad [7]$$

With the following chart and appropriate unit conversions:

MT model	V_R (MeVfm)	μ_R (fm^{-1})	V_A (MeVfm)	μ_A (fm^{-1})
I	1438.720	3.110	513.968	1.550
III	1438.720	3.110	626.885	1.550

[8]

To construct the U matrix, using equations 3 and 4 from the write-up:

$$U(p, q) = \frac{2}{\pi p q} \int_0^\infty dr \sin(pr)[mV(r)] \sin(qr) \quad [9]$$

and

$$U(p, 0) = U(0, p) = \frac{2}{\pi p} \int_0^\infty drr[mV(r)] \sin(pr) \quad \text{and} \quad U(0, 0) = \frac{2}{\pi} \int_0^\infty drr^2[mV(r)] \quad [10]$$

For specific cases used in the code, the U matrix elements can be calculated in the following way:

$$U_{ij} = U(q_i, q_j) = \sum_{k=1}^{N} w_{rk} \frac{\sin(q_i r_k)}{q_i} V(r_k) \frac{\sin(q_j r_k)}{q_j} \; ; \; q_i \text{ and } q_j \neq 0 \quad [11]$$

$$U_i^{(0)} = U(0, q_i) = \sum_{k=1}^{N} w_{rk} r_k \frac{\sin(q_i r_k)}{q_i} V(r_k) \quad [12]$$

$$U^{(00)} = U(0, 0) = \sum_{k=1}^{N} w_{rk} r_l^2 V(r_k) \quad [13]$$

A U matrix can be populated using the following pseudocode:

```
#Define functions

def Mesh(mesh_size):
        return mesh asbsesca and weight for a given mesh size

def UpdatedMesh(mesh_size):
        q,w=Nx1 empty matrix
```

```
    for i in range(N):
        q=(1+asbsesca)/(1-asbsesca)
        w=2*weight/(1-weight)**2
    return q and w

def Vi(r,i):
    #this is defined according to eq. (15) in the write up
    Assign values for Vr, mu_r, mu_a, conversion
    if (i=1):
        Assign Va for MTI model
    if (i=3):
        Assign Va for MTIII model
    Use variables to define V(r)
    return V

def U(N,n):
    U=NxN empty matrix
    q,r=q from UpdatedMesh
    w=w from UpdatedMesh
    for i in range (N) and j in range (N):
        Use eq.(11) from write-up here
        Specific form is dependent on q[i] and q[j]
        if q[i] and q[j] aren't 0
            for k in range(N):
                Take sum in eq.(11)
        if q[i] or q[j] are 0
            for k in range(N):
                Take sum in eq.(12)
        if q[i] and q[j] are 0
            for k in range(N):
                Take sum in eq.(13)
        Set the sum equal to matrix element U[i,j]
    return U

#U(N,n) is for set values of qi and qj, the next function allows for
   user input of qi and qj

def Uij(N,n,qi,qj):
    #Unlike U(N,n), this function allows for user input and also
       just gives the matrix element U_ij
    r=q from UpdatedMesh
    w=w from UpdatedMesh
    Use eq.(11) from write-up here
    Specific form is dependent on qi and qj
    if qi and qj aren't 0
        for k in range(N):
            Take sum in eq.(11)
    if qi or qj are 0
```

```
        for k in range(N):
            Take sum in eq.(12)
if qi and qj are 0
        for k in range(N):
            Take sum in eq.(13)
Set sum equal to Uij
return Uij
```

The following is a 3D plot for $U(p, q)$:

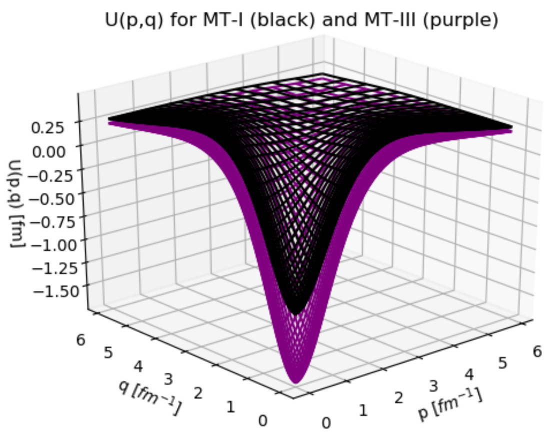

Figure 1: 3D wireframe plot of U(p,q) where MT-I is given in black and MT-III is given in purple

Part 4

In order to implement the Gauss-Jordan elimination in part 1, an augmented matrix needs to be generated. The following is a 3D plot for $K(p, q)$:

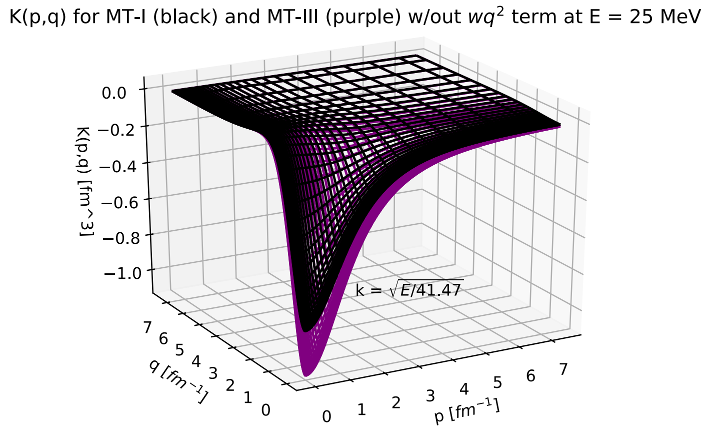

Figure 2: 3D wireframe plot of K(p,q) where MT-I is given in black and MT-III is given in purple and E = 25MeV

For the scattering problem, $K_{ij}^{(k)}$ is given by the following:

$$K_{ij}^{(k)} = w_{q_j} q_j^2 \frac{U_{ij} - U_i^{(k)}}{k^2 - q_j^2} \tag{14}$$

The integral equation $W(p, k)$ is given as a function of K as individual components:

$$W_i^{(k)} = U^{(k)})_i + \sum_{j=1}^{N} K_{ij}^{(k)} W_j^{(k)} \ , \ i = 1, \cdots, N \tag{15}$$

$$W^{(kk)} = U^{(kk)} + \sum_{j=1}^{N} w_{q_j} q_j^2 \frac{U_j^{(k)} - U^{(kk)}}{k^2 - q_j^2} W_j^{(k)} \tag{16}$$

The results for the bound-state solution is slightly different:

$$K_{ij}^{(\alpha)} = w_{q_j} q_j^2 \frac{U_{ij} - U_i^{(0)}}{-\alpha^2 - q_j^2} \ ; \ i, j = 1, \cdots, N \tag{17}$$

$$W_i^{(\alpha)} = U_i^{(0)} + \sum_{j=1}^{N} K_{ij}^{(\alpha)} W_j^{(\alpha)} \ , \ i = 1, \cdots, N \tag{18}$$

This result can be used to determine the Jost function. The code used for this part calls on some previously defined functions:

```
def Uk_func(N,n,k):
        #Specific case of Uij
        Uk=Nx1 empty matrix
        w=w from updatedmesh
        q,r=q from updatedmesh
        #use "if" statement to prevent infinities
        if k=0:
                for i and l in range(N):
                        Uk= sum from eq.12 in write-up
        else:
                for i and l in range(N):
                        Uk= sum from eq.14 in write-up
        return Uk

def Ukk_func(N,n,k):
        #Another specific case of Uij
        w=w from updatedmesh
        r=q from updatedmesh
        if k=0:
                for i and l in range(N):
                        Uk= sum from eq.13 in write-up
        else:
                for i and l in range(N):
                        Uk= sum from eq.15 in write-up
        return Uk

def K_func(N,n,k):
        K=NxN empty matrix
        w=w from updatedmesh
        q=q from updatedmesh
        for i and j in range(N):
                Calc. individual components of K using eq.[14]
        return K

def Wk_func(N,n,k):
        #this is solved in a similar manner to eq.[15]
        Define identity matrix
        A=identity-K
        A_aug=A appended with column vector Uk
        Wk=GaussianEliminationWPivot(A_aug)
        #calls on previous function for Gauss elimination
        return Wk

def Wkk_func)N,n,k):
        w=w from updatedmesh
        q=q from updatedmesh
        for j in range(N):
```

```
        sum=solution to the sum in eq.[16]
Wkk=Ukk+sum
return Wkk
```

Part 5

Finally, the system of linear equations from part 4 was solved and the Jost function was calculated. This function was of the following form for the scattering problem:

$$\mathrm{Re}F(k) = 1 - \sum_{i=1}^{N} w_{q_i} \frac{q_i^2 W_i^{(k)} - k^2 W^{(kk)}}{k^2 - q_i^2} \quad [19]$$

$$\mathrm{Im}F(k) = \frac{\pi}{2} k W^{(kk)} \quad [20]$$

The phase shift and scattering length are given by the following:

$$\tan\delta(k) = -\frac{\mathrm{Im}F(k)}{\mathrm{Re}F(k)}\ ,\ a = \frac{\pi}{2}\frac{W^{(00)}}{1 + \sum_{i=1}^{N} w_{q_i} W_i^{(0)}} \quad [21]$$

For the bound-state problem, the functions are slightly different:

$$K_{ij}^{(\alpha)} = w_{q_j} q_j^2 \frac{U_{ij} - U_i^{(0)}}{-\alpha^2 - q_j^2}\ ;\ i,j = 1,\cdots,N \quad [22]$$

$$W_i^{(\alpha)} = U_i^{(0)} + \sum_{j=1}^{N} K_{ij}^{(0)} W_j^{(\alpha)} \quad [23]$$

The Jost function is given by the following for the bound-state problem.

$$F(-\alpha^2) = 1 + \sum_{i=1}^{N} w_{q_i} q_i^2 \frac{W_i^{(\alpha)}}{\alpha^2 + q_i^2} \quad [24]$$

The momentum space wave function is given by the following:

$$\tilde{\psi}(q_i) = -C\frac{W^{(\alpha_B}}{\alpha_B^2 + q_i^2}\ ,\ \frac{1}{C^2} = \sum_{i=1}^{N} w_{q_i} q_i^2 \left(\frac{W_i^{(\alpha)}}{\alpha_B^2 + q_i^2}\right)^2 \quad [25]$$

We consider the Jost function for energies between -0 MeV and -5 MeV with $-\alpha^2 = E/41.47$. The code for this section is as follows:

```
def Re_F_func(N,n,k):
        w=w from updatedmesh
        q=q from updatedmesh
        for in in range(N):
                sum=solution to sum in eq.[19]
        ReF=1-sum
        return ReF
```

```
def Im_F_func(N,n,k):
     ImF=solution to eq.[20]
     return ImF

def phase_shift_func(N,n,k):
     phase_shift=arctan(-ImF/ReF)
     return phase_shift

def scat_length_func(N,n):
     for i in range(N):
          sum=solution for sum in eq.[21]
     a=solution to eq.[21] with sum
     return a

def K_func_alpha(N,n,k):
     K=NxN empty matrix
     q=q from updatedmesh
     for i,j in range(N):
          K[i,j] = solution eq.[22] for matrix elements
     return K

def Wk_func_alpha(N,n,k):
     identity=NxN identity matrix
     A=identity-K
     A_aug=A appended with column vector Uk_func(N,3,0)
     Wk=GaussianEliminationWPivot(A_aug)
     return Wk

def F_Jost_func(N,n,k):
     w=w from updatedmesh
     q=q from updatedmesh
     for i in range(N):
          sum=solution to sum in eq.[24]
     F_Jost=1+sum
     return F_Jost

def C_norm(N,n,k):
     w=w from updatedmesh
     q=q from updatedmesh
     for i in range(N):
          sum=solution to sum in eq.[25]
     C=sqrt(1/sum)
     return C

def phi_i_func(N,n,k,i):
     q=q from updatedmesh
     phi= solution to eq.[25]
     return phi
```

For the scattering case, we plot the phase shift against energy, yielding the plot in Figure 3.

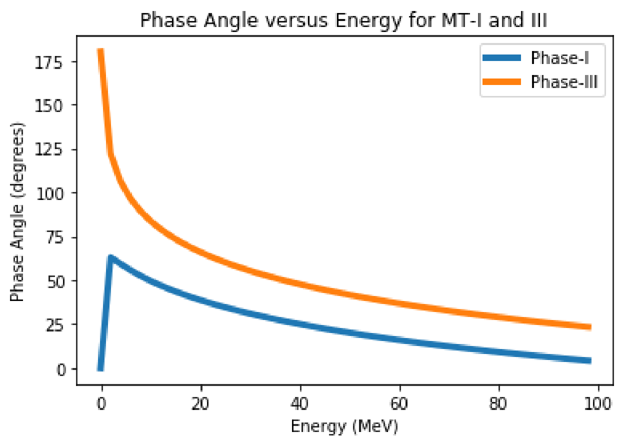

Figure 3: Plot of the phase shift (in degrees) for MT-I and MT-III against the energy in MeV

Looking now to solve the bound-state for MT-III, we plot our Jost-like function against energy in Figure 4. With this, we are able to find our bound-state energy from the root of the function. Our result is -1.800 MeV.

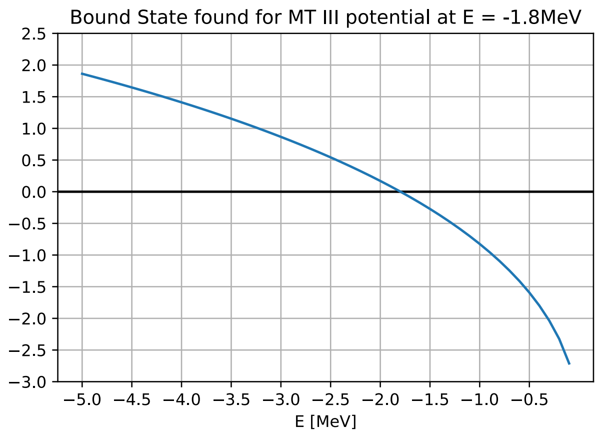

Figure 4: Plot of the Jost-like F function for MT-III. The root of this plot indicates the bound-state energy level, which we find to be -1.800 MeV.

Now with this energy, we plot our bound wavefunction, as shown in Figures 5 and 6.

While our Jost function and bound-state energy do not perfectly match the expected results, we are able to recognize the expected general form within our Jost and wave-function plots.

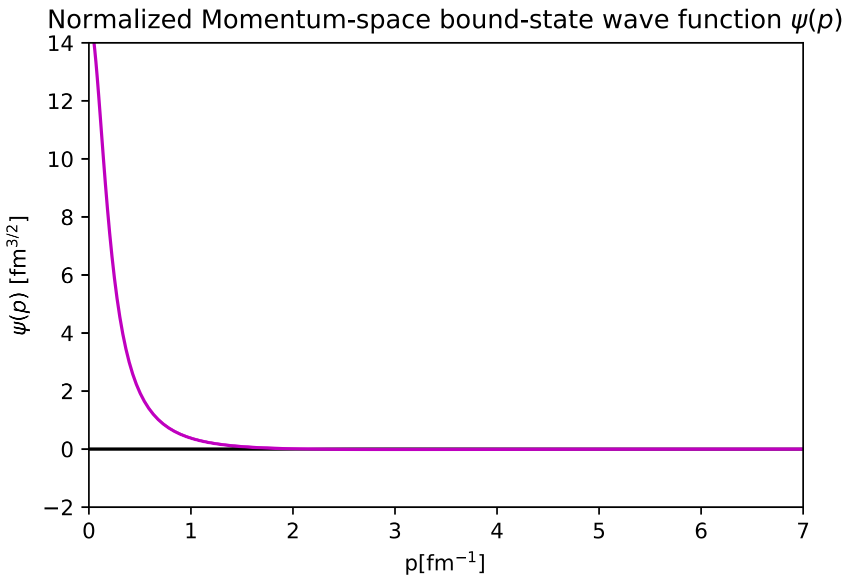

Figure 5: Plot of the bound-state wave-function for MT-III.

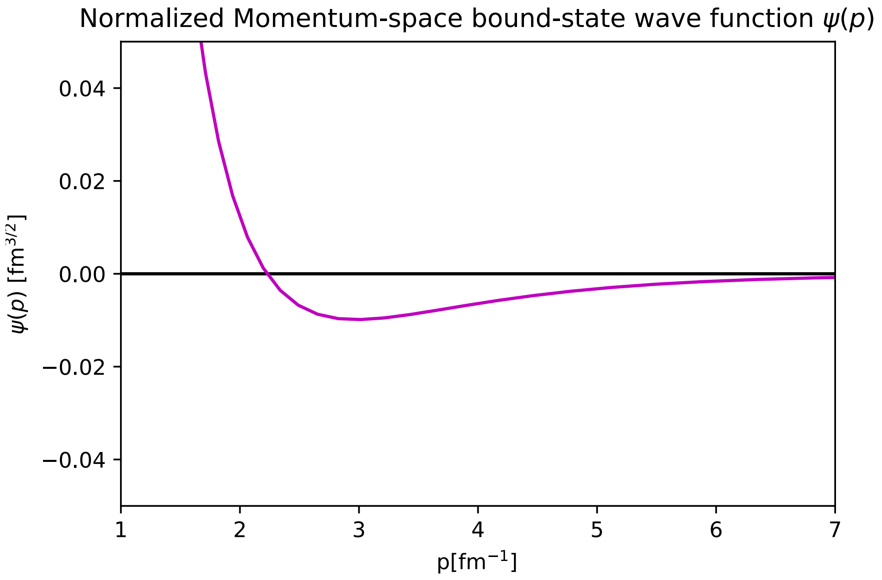

Figure 6: Here we see a more detailed view of our bound-state wave-function.