Claremont Colleges

Scholarship @ Claremont

Scripps Senior Theses

Scripps Student Scholarship

2020

Decay and Dissipation: Finding Energy Level Jumps in a Harmonic Oscillator System using Fortran and Fourier Analysis

Clara Chilton

Follow this and additional works at: https://scholarship.claremont.edu/scripps_theses

Part of the Quantum Physics Commons

Decay and Dissipation: Finding Energy Level Jumps in a Harmonic Oscillator System using Fortran and Fourier Analysis

A Thesis Presented By Clara Chilton

In Collaboration with Viðar Guðmundsson[1] - University of Iceland School of Engineering and Natural Sciences, Physics

To the Keck Science Department
Of Claremont McKenna, Pitzer, and Scripps Colleges

In partial fulfillment of
The Degree of Bachelor of Arts

Senior Thesis in Physics

Fall 2019

[1] Háskóli Íslands. Viðar Guðmundsson Available at: https://english.hi.is/staff/vidar. (Accessed: 3rd December 2019)

TABLE OF CONTENTS

Clara Chilton Decay and Dissipation

ABSTRACT

In this paper, I will look at a mass-spring system that can be described by a Hamiltonian. In most systems described by a Hamiltonian, the energy levels will be quantized, and the system will be able to jump between them. However, many methods of finding these jumps aren't well-suited to numerical analysis. I'll use a Markovian approximation (The Liouville von Neuman Equation), which allows me to use only the last time step to find the current one. Using this, I will analyze the system to find the time evolution of the probability density matrix – whose diagonal shows the probability of the object being in each energy state at a specific time. I will then repeat this process with an added dissipation energy added to the Hamiltonian, which makes it decay over time into its ground state. From this, and using Fortran coding and Fourier Analysis, I will find the most probable number of energy levels that the object will jump between when said dissipation is applied. The result of this is that I will see what information is lost in this approximation by comparing it with a different method of computing it that Vidar previously modeled.

INTRODUCTION

In an open quantum system with discrete energy states, an external perturbation can always induce quantum jumps in between energy levels. These jumps are seen in many places in nature, can be used in devices to harvest energy, and are useful to know about when building quantum computers and such since we first must understand the underlying quantum behavior before harnessing it.

In this paper, I will assume the object exhibits Markovian behavior – i.e., only the last state affects the current state and not the path it takes to get to the current state from the first state. Another way of saying this is that only the last time step affects the current state – the system has short-term memory). This model is useful because it is more convenient for numerical calculations than one that takes all previous states into effect. However, as we will see, some information, especially about upper-level energy states, is lost in this approximation. The question tackled in this paper is whether the information lost is critical to understanding the system's behavior. I will be comparing my results – with a Markovian approximation – to Viðar's – without the approximation – to see what information is lost.

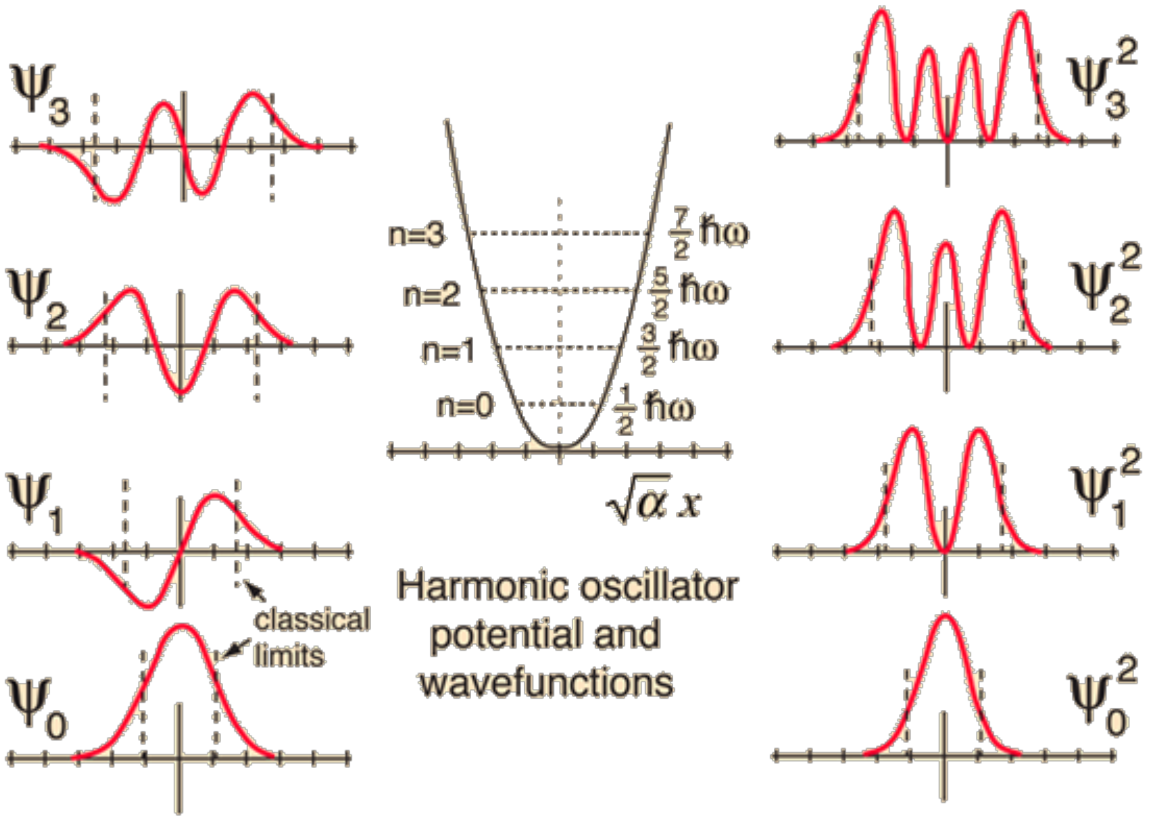
I. HARMONIC OSCILLATOR

An example of an energy level system that is both a good approximation of many systems seen in nature quantum harmonic oscillator. This is often used in physics classes due to its predictable behavior and ease of modeling. The Quantum Harmonic Oscillator can be modeled much like a conventional harmonic oscillator system, only with a Hamiltonian instead of a Lagrangian:

$$H_o = \frac{p^2}{2m} + \frac{1}{2}m\omega^2 x^2 \tag{1}$$

The basic format of a Hamiltonian is H=K+U, where K is the kinetic energy of the system and U is the potential. For comparison, in a Newtonian harmonic oscillator, the kinetic energy is $k = \frac{1}{2}\frac{p^2}{2m}$, and the potential is $U = \frac{1}{2}mx^2$, and the Lagrangian is L=K-U.

The main difference between the two is that, in a Quantum Harmonic Oscillator system, the object has quantized energy levels that it will jump between when disturbed by an outside force. These are energy levels separated by some ΔE, and the system can not have an energy level not equal to one of these levels. Each of these states, or energy levels, can be described as a wavefunction.



[2] Figure 1: Drawing illustrating the shapes of wavefunctions and their corresponding probability densities of a harmonic oscillator.

To model this, I will use Dirac notation (where |x> corresponds to a vector x and H – an operator - to a matrix H), which I will then convert to matrix-vector format for my calculations.

II. ENERGY LEVELS

The system described above has quantized energy levels, which means we visualize the oscillations between energy levels by plotting the probability density, or occupation of the states, as a function of time. Thus, we can view the behavior of the system by modeling how it oscillates between energy levels. At the end of this model, I will see how the system tends to move between these energy states when a perturbation is applied – either one energy level at a time or multiple.

It is also a quantum system, which means we have to model the particle in terms of the probability that it's in any one state since we can't know for sure which state it's in. This comes from the Uncertainty Principles.

[2] Wavefunctions. *Quantum Harmonic Oscillator* Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/hosc5.html. (Accessed: 4th December 2019)

Going into this project, I assumed the system would mostly only move to adjacent energy levels, but that there would be some probability of longer jumps.

III. LIOUVILLE-VON-NEWMAN EQUATION

In order to solve this system, I'll use the Liouville-von-Newman Equation:

$$i\hbar\dot{\rho}(t) = [H(t), \rho(t)] = i\Lambda[\rho(t)] \tag{4}$$

$$\rho(t_{n+1}) = \rho(t_n) + \frac{\Delta t}{2\hbar}(\Lambda[\rho(t_n)] + \Lambda[\rho(t_{n+1})] \tag{5}$$

Where ρ(t) is the probability density function – a matrix whose diagonal represents the amount of the system in any state n at time t. This is the same probability function as shown in figure 1. As said above, we can only say the probability that the particle is in any state, and not whether it is in that state. By solving this recursively, one can create a Markovian approximation of the system.

IV. FORTRAN

A large part of this project for me was devoted to learning Fortran. Although Fortran is often seen as an outdated language now[3], it is still used by physicists due to its speed. It is

[3] In fact, I had multiple people ask me why I was working with Fortran, or that they hadn't worked with it for decades.

approximately one-hundred times faster than Python, which aids in handling large arrays of numbers and has an amount of easily accessed open-source code. In this project, although I wasn't working with large enough arrays of numbers for the speed to be a factor, I did set up my code such that, in the future, I could work faster with more data at once. I did use open source code, such as a Fourier Transform Code. I developed the rest of the code after looking at simple examples from Viðar.

Much of my time was spent learning and implementing what I learned of Fortran to write readable code that was easy to edit (i.e., I wanted to be able to change a single line or variable to test out a new hypothesis). The final code that I wrote is in the Appendix, and the Fourier Transform code is linked.

V. HYPOTHESIS

Going into this project, I hypothesized that there would be some probability that the object would jump multiple energy levels at once, although it would mostly only jump one. I also thought that as I added more dissipation to the system, I would observe more multiple-energy-level jumps. I hypothesized this because when using a non-Markovian approximation, Viðar found longer jumps, and we both thought that my Markovian model would still pick up some of these.

Clara Chilton Decay and Dissipation

METHODS

I. FINDING HAMILTONIAN

Given a quantum system with discrete energy levels, one can write its energy in the form of the Hamiltonian. For the system in this paper, we will start with a Harmonic Oscillator Hamiltonian described above and convert notations to one dependent on the raising and lowering operators for ease of solving.

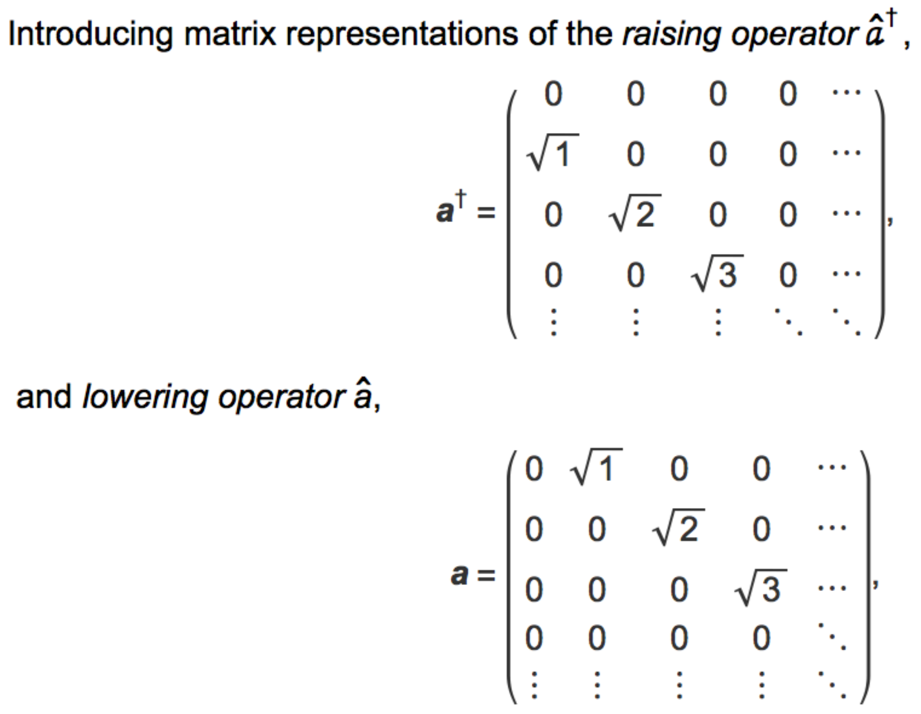
$$H(t) = \hbar\Omega\left\{a^{+} + a\right\}\theta(t) + H_o \tag{1}$$

$$H_o = \hbar\omega\left\{a^{+}a + \frac{1}{2}\right\} \tag{2}$$

Where a^{+} and a are the raising and lowering operators:

$$\begin{aligned} a|n> &= \sqrt{n}\,|n-1> \\ a_{+}|n> &= \sqrt{n+1}\,|n+1> \end{aligned} \tag{3}$$

Or, in matrix form:



Introducing matrix representations of the *raising operator* $\hat{a}^\dagger$,

$$\mathbf{a}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ \sqrt{1} & 0 & 0 & 0 & \cdots \\ 0 & \sqrt{2} & 0 & 0 & \cdots \\ 0 & 0 & \sqrt{3} & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{pmatrix},$$

and *lowering operator* $\hat{a}$,

$$\mathbf{a} = \begin{pmatrix} 0 & \sqrt{1} & 0 & 0 & \cdots \\ 0 & 0 & \sqrt{2} & 0 & \cdots \\ 0 & 0 & 0 & \sqrt{3} & \cdots \\ 0 & 0 & 0 & 0 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

[4]

This Hamiltonian was given to me as a starting point. The raising and lowering operators depend on the energy level n that the system is in at the moment, and dictate its behavior as it

[4] Physkid, PhyskidPhyskid 64044 silver badges88 bronze badges & ubpdqnubpdqn 51.7k22 gold badges4343 silver badges116116 bronze badges. odd matrix operation. *Mathematica Stack Exchange* (1967). Available at: https://mathematica.stackexchange.com/questions/146467/odd-matrix-operation. (Accessed: 4th December 2019)

changes energy levels over time. In a system with n states, I can represent these as matrices (operators) – such as H - acting on vectors (states) – such as n. These operators allowed me to form the hamiltonian by defining them instead of having to know the momentum, mass, and such. When I formed this Hamiltonian, the diagonal represented the energy levels of each state n.

The code used to form it is shown here:

```
        ALLOCATE(xmat(Nf,Nf),H0(Nf,Nf),Hmat(Nf,Nf),a_plus(Nf,Nf),a(Nf,Nf),x(Nf,Nf),
STAT=ierr)
        DO j=1,Nf
        DO i=1,Nf
        IF(ABS(i-j) .EQ. 1) xmat(i,j) = 0.5_dp*SQRT(REAL(i+j-1,dp))
        END DO
        END DO
        ! WRITE(12,FMT='(E15.8,1X,E15.8)') xmat(2,1)
        DO j = 1,Nf
         H0(j,j) = CMPLX((REAL(j-1,dp))+0.5_dp,0.0_dp,dp)
        END DO


         Hmat = Czero
         a_plus = Czero
         a = Czero


         DO j = 1,Nf
         DO i = 1,Nf
         IF ((ABS(i-j) .EQ. 1) .AND. (i .GT. j)) THEN
           a_plus(i,j)=xmat(i,j)
          ELSE IF ((ABS(i-j) .EQ. 1) .AND. (j .GT. i)) THEN
           a = xmat(i,j)
         END IF
         END DO
         END DO
         Hmat = H0+lam*xMat
        x = (a_plus + a)/SQRT(2.0_dp)
```

As shown, I used a loop to form a_plus and a, since they each have zeros everywhere except right above or below the diagonal, as seen above.

II. FINDING PROBABILITY DENSITY

Once I had the Hamiltonian, I could calculate the probability density function ρ(t), where ρ is a matrix whose diagonal lists the probability of finding the particle in each energy state at time t. This is the closest we can get to the location of the particle due to quantum uncertainty principles. In order to do this, we use the Liouville-von Neuman Equation described above:

$$i\hbar\dot{\rho}(t) = [H(t), \rho(t)] = i\Lambda[\rho(t)] \tag{4}$$

$$\rho(t_{n+1}) = \rho(t_n) + \frac{\Delta t}{2\hbar}(\Lambda[\rho(t_n)] + \Lambda[\rho(t_{n+1})] \tag{5}$$

Solving this equation recursively allows us to find ρ at every time t.

III. RECURSIVE SOLVING AND SOLUTION

To solve this, I first set both ρ(tn=0) and ρ(tn +1) to a zero matrix of dimension n by n. I then assigned the element of the diagonal, which represented the excited state of the system at t=0 to 1, and, using that, found a new ρ(tn +1) which I plugged in, keeping ρ(tn=0) the same. I repeated this process until the current ρ(tn +1) was reasonably close to the previous. To determine this, I took the trace of both and compared them, and when they had a difference of less than 1-6, I moved on to the next time step, setting ρ(tn) to the last value of ρ(tn +1) and repeating the process. I determined the value to test the traces through trial and error. I tried different values until I found one that minimized error, while not being so small that it slowed down the code.

In my code, the recursive sequence looks like:

```
          DO WHILE (abs(trace1-trace2)>1.0E-6_dp)

          trace1 = Czero
          do i = 1, Nf
           trace1 = trace1 + rho_in(i,i)
          end do

          trace2 = Czero
          do i = 1, Nf
           trace2 = trace2 + rho1(i,i)
          end do

 rho1 = rho0+(dt/(2.0_dp*ci))*((matmul(Hmat,rho0)-
matmul(rho0,Hmat))+((matmul(Hmat,rho_in)-matmul(rho_in,Hmat))))

           rho_in = rho1
           num = num+1
          END DO
```

Where Hmat is my Hamiltonian in terms of the raising and lowering operators, rho0 is my initial ρ value, rho1 is the value I'm trying to get, and rho_in is the recursion value (where I plug in the result of the last recursion until I'm within a margin of error).

IV. GRAPHING

Again, this is a matrix, this time with the diagonal representing the occupation density of each state.

With this, we can create a graph of ρ(t) with different lines representing the different energy states, and using the trace to check that the density functions of each state add up to one (figure 1).

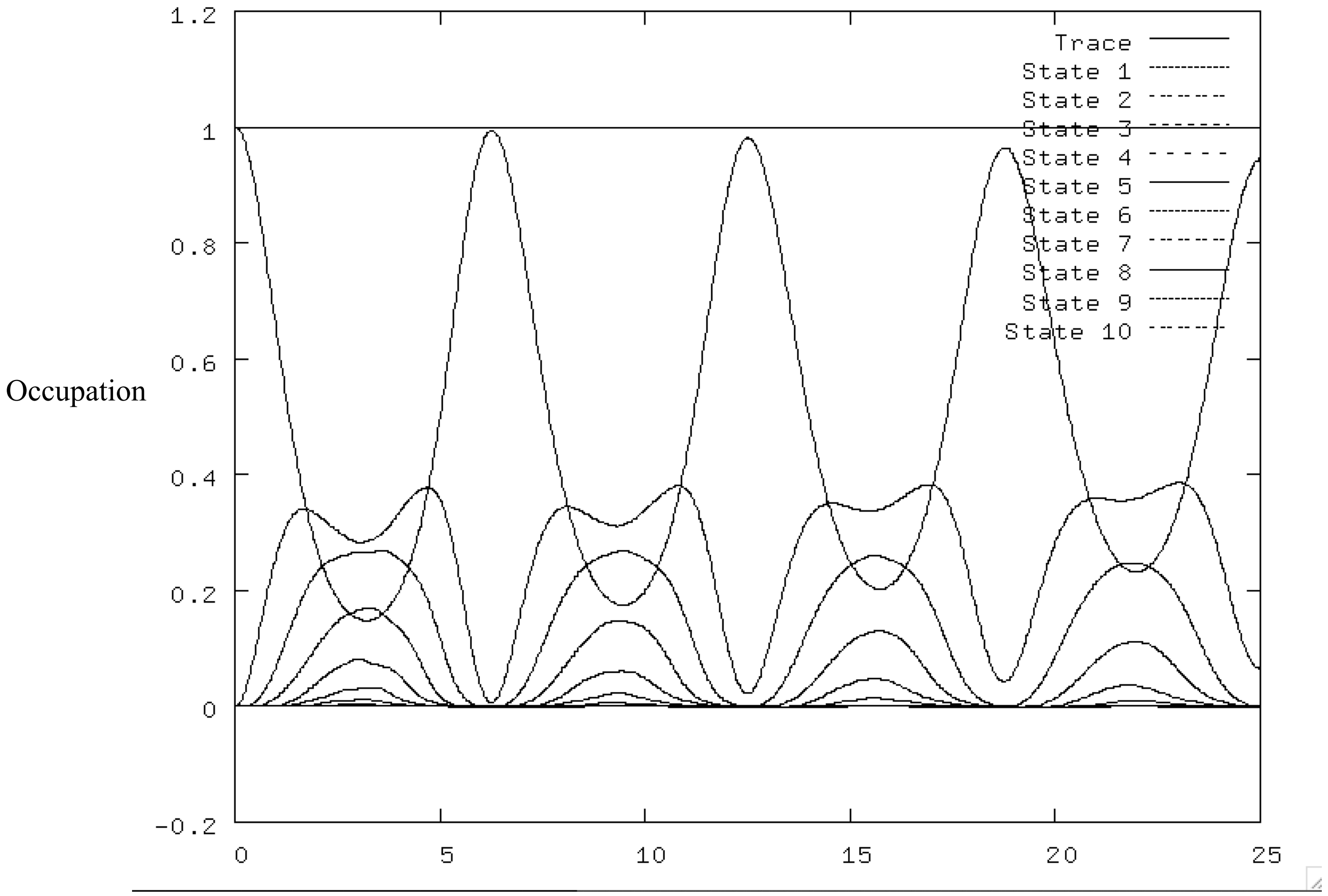


Figure 2: Graph showing the time evolution of the density function of the first ten energy states, with the curve with the highest amplitude as the ground state.

I used GnuPlot for this graph. I had to adjust it multiple times before getting the final version, as I kept having interference patterns showing up from errors in my raising and lowering operators.

V. CREATING THE OPEN SYSTEM

Up until now, we've dealt with a closed system. Now, we can add dissipation – opening the system up to the environment - to the system in the form of:

$$i\hbar\dot{\rho}(t) = [H(t), \rho(t)] - \frac{ik}{2}\left\{\left[a\rho(t), a^{+}\right] + \left[a, a^{+}\rho(t)\right]\right\} \tag{6}$$

This equation comes from the James-Cummings Model and adds an outside force that can be adjusted by the constant k. Much like the trace test value, I determined a value for this by testing different values and seeing what made my graphs both readable, which and corresponded to weakly coupled systems found commonly in nature. If it were too large, the dissipation would occur too quickly to see, and if it were too small, my program would time out before I saw results. I again used direct iteration of the Liouville-von Neuman Equation, this time with my new definition of [H(t),ρ(t)], and got a new ρ(t) matrix that we can graph. (figure 2).

When coding this, I had to be careful to get the notation right, as the Liouville Von Neuman Equation ends up being messy as one can see:

```
rho1 = rho0 + dt*0.5_dp*((-ci*(MATMUL(Hmat,rho0)-MATMUL(rho0,Hmat))&
+dis*0.5_dp*(MATMUL(MATMUL(a,rho0),a_plus)-
MATMUL(a_plus,MATMUL(a,rho0)) + &
 MATMUL(a,MATMUL(rho0,a_plus))-MATMUL(MATMUL(rho0,a_plus),a)))+ &
(-ci*(MATMUL(Hmat,rho_in)-MATMUL(rho_in,Hmat))&
+dis*0.5_dp*(MATMUL(MATMUL(a,rho_in),a_plus)-
MATMUL(a_plus,MATMUL(a,rho_in)) + &
MATMUL(a,MATMUL(rho_in,a_plus))-MATMUL(MATMUL(rho_in,a_plus),a))))
```

Although for the first graph, I started my system in the ground state, for this one, I had to start in a higher state since otherwise, the system showed no dissipation. Since it dissipates into the ground state, starting it in the ground state wouldn't give me any useful information. Again, I had to test different initial states, finding one that both had enough information but was readable, and settled on the fourth state. The fourth state gives it multiple states to jump through but keeps it simple enough that one can easily read the graph.

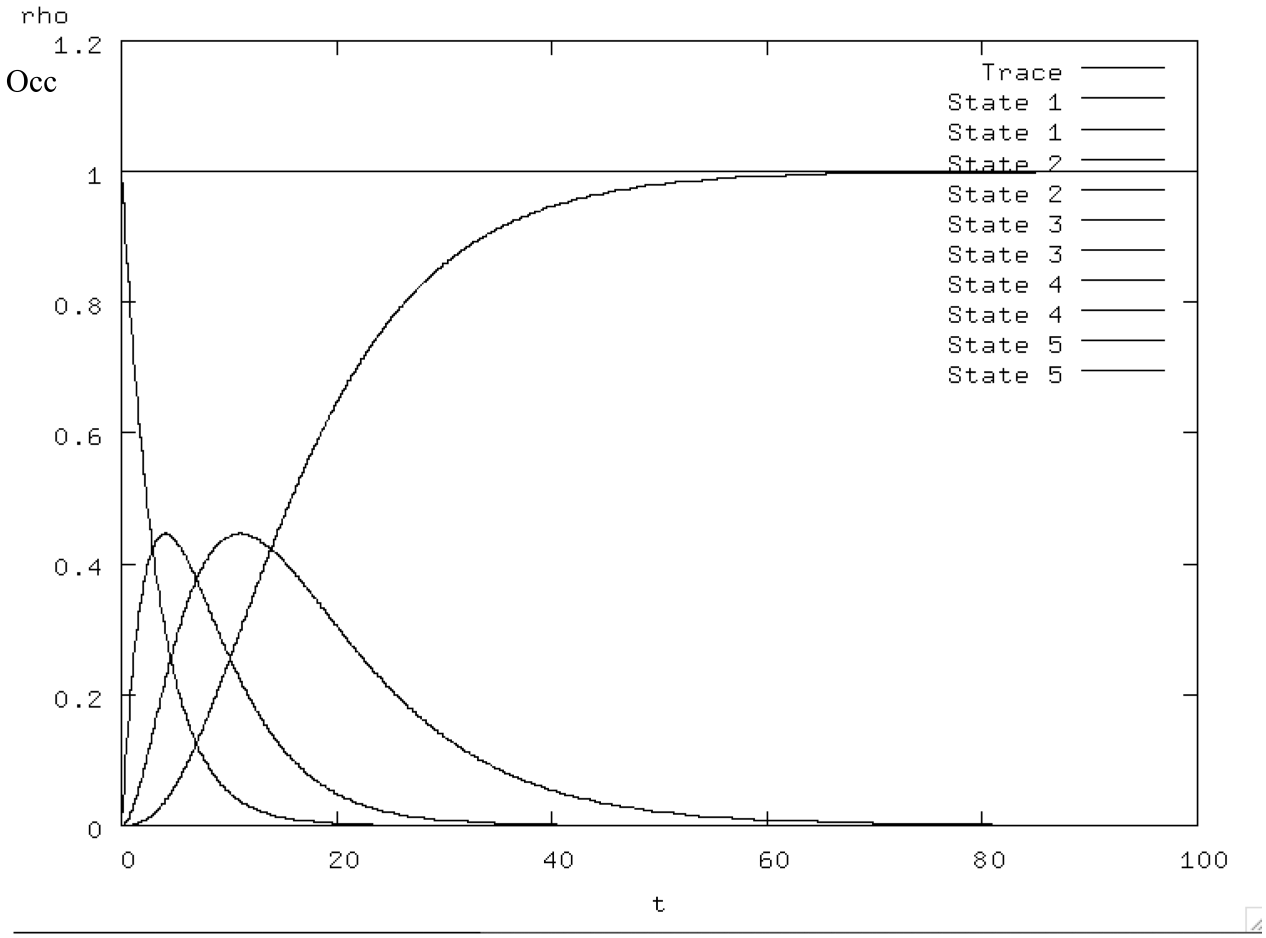


Figure 3: This shows the time evolution of our new density function, starting in state four and degrading until reaching equilibrium in the ground state.

My final code for the data graphed here is shown in Appendix I.

VI. MODELING TIME DEPENDENCE

Now, to find how the particle moves between each state, I defined a new function as

$$(a^{+} + a)\rho_{ss}(0) \tag{7}$$

Where ρss(0) is the equilibrium state, the graph in Figure 2 eventually reaches in matrix form. This graph allows us to observe the time-dependence of rho via the Quantum Regression Theorem. In this case, it ends up with most of the probability in the ground state and only a little bit in the rest. This equation then gave me figure 3, which shows the oscillation of the particle between energy states and how it reaches a steady-state:

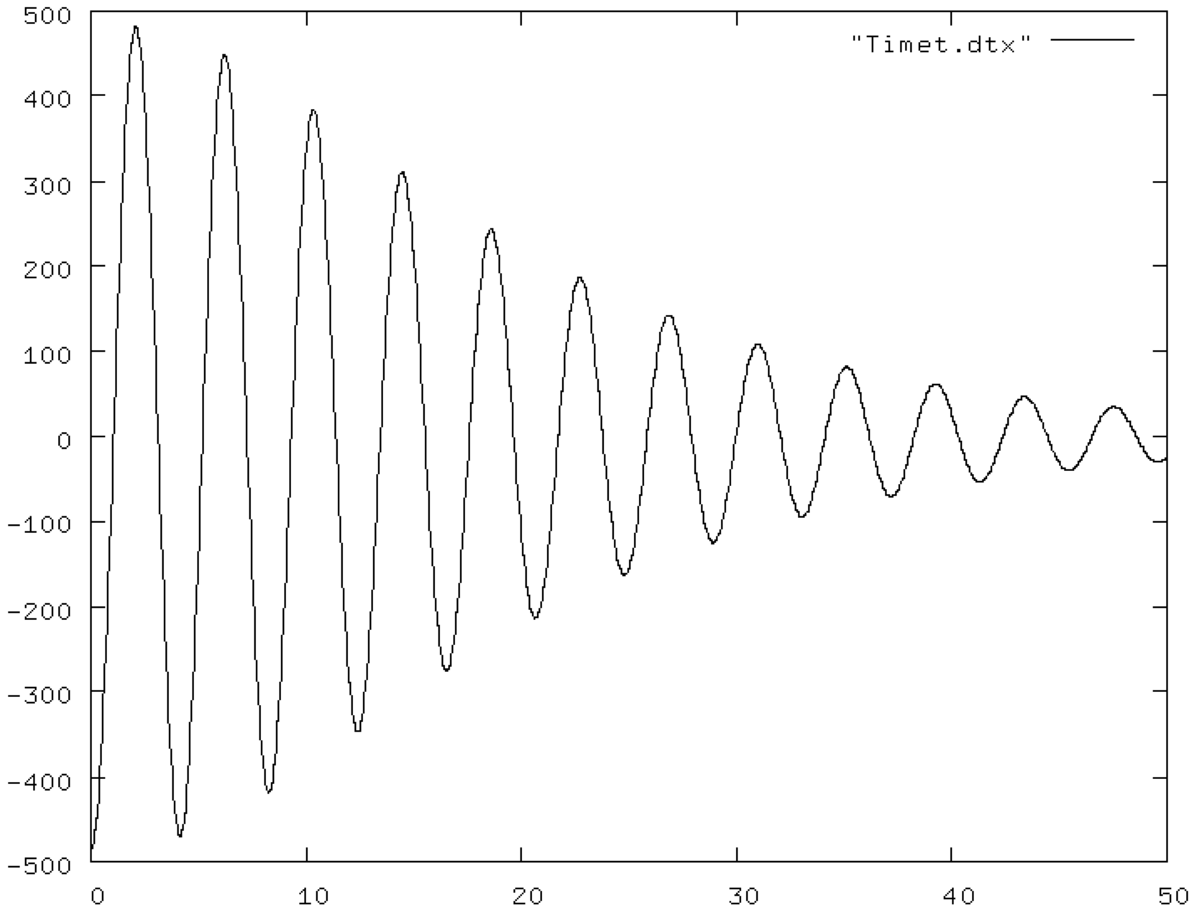


Figure 4: Graph showing the strength of the oscillations between energy levels as a function of time

My final code for the data graphed here is shown in Appendix II.

VII. USING FOURIER TRANSFORM

Lastly, I put the oscillation into a Fourier transform program on Fortran, which was an open-source program. I did this to show what wave-states contribute to the last graph (Figure 3), thus showing between which states the system is oscillating. Solving this gave me my final result:

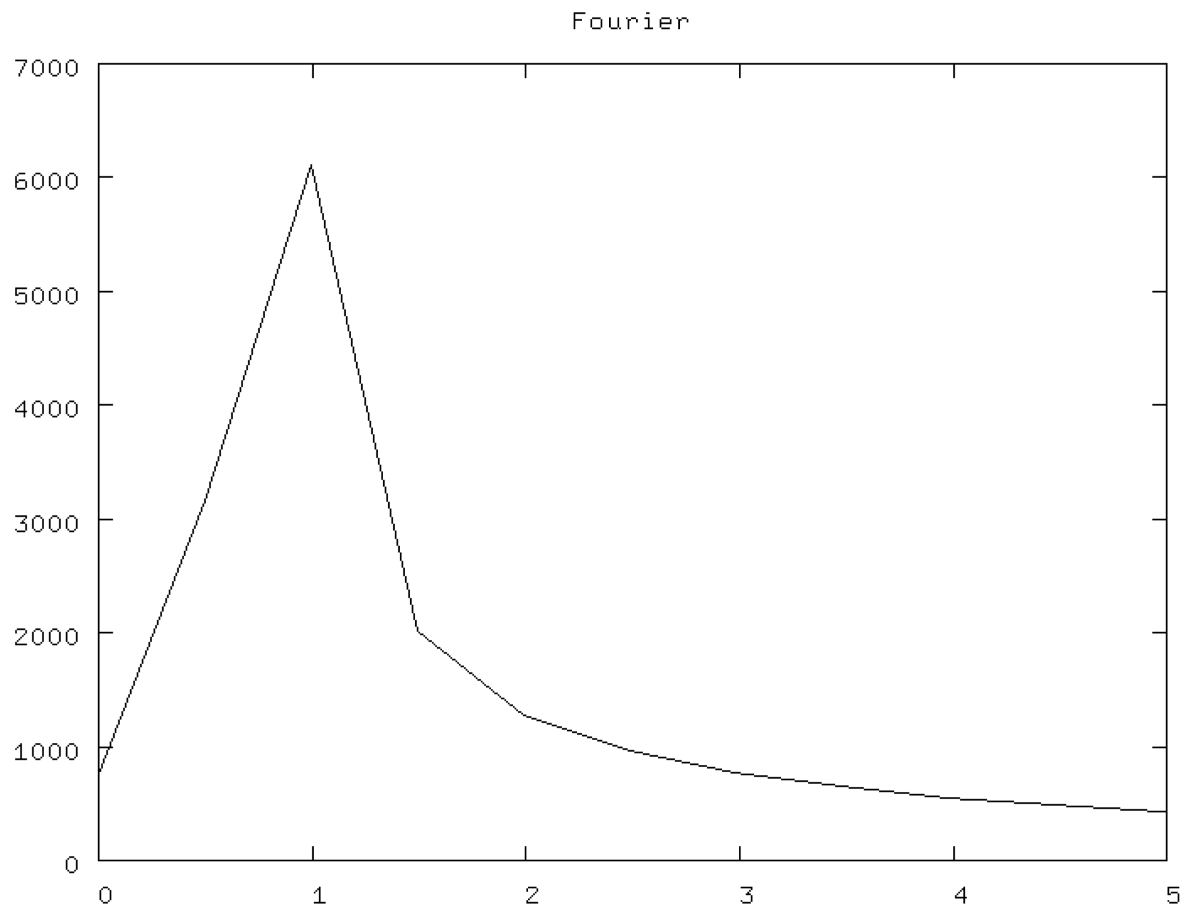


Figure 5: Graph showing the final Fourier analysis with respect to the energy states, scaled to reflect scales in my code.

This graph shows that the most common jump will be by one quantum unit - in this case, hbar*omega. There are no peaks seen for higher-level jumps, showing that there are none of these jumps present in this model.

Clara Chilton Decay and Dissipation

CONCLUSION

The final result of this work is that the object in the mass-spring system described will jump, at most, one energy level at a time when the system is coupled with the environment or dissipation is applied. This result went against my initial hypothesis that there would be some probability of the system jumping multiple energy levels at once, as well as Viðar's results when he modeled a non-Markovian system. This shows that it was most likely the Markovian approximation – only taking the previous time step into account and not all previous time steps – that caused information to be lost. Thus, the next step would be to test this model against one using integrals, which would give a more precise model of the dissipation over time and see if the results differ.

In the future, I could also adjust the dissipation model to see how more or less dissipation affects the behavior of the electron. I hypothesize that more dissipation would cause more drastic energy level changes. Using the code I have, (See Appendix), I would only have to change dis – the variable that governs how strong the dissipation field is.

Another way to continue this would be to start with a different Hamiltonian. I only used a basic Quantum Harmonic Oscillator, but this model would work with other systems as well, and, again, I would only have to edit a line of my code. This could give me a better idea of what information is lost with the Markovian approximation, as some systems might show longer energy level jumps.

Lastly, since I used Fortran, I could look at more energy levels at once and see if factoring in higher levels will affect my result. This was actually one of the main reasons it was used for this project – if I continue this work, I will be dealing with more complicated systems where the speed I can run the code at will make a difference.

Appendix

I. CODE FOR SOLVING PROBABILITY DENSITY

```
 PROGRAM Time

  USE omp_lib           ! For OpenMP parallel processing
  USE Mod_Precision       ! Module for setting double precision
  USE Mod_Init          ! Initial values
  USE Mod_Fields          ! Global variables

  USE lapack95          ! MKL-Subroutines
  USE blas95           ! MKL-Subroutines
USE omp_lib

  IMPLICIT NONE
!------- Local variables -----------------------------------
!
  INTEGER       :: i, j, ierr, k, num,Nt,it,initial
  REAL        :: trace, avg_x, trace1,trace2, scaler,delta_t,dt,t,
omega,omega_big,dis,lam


!
  OPEN(UNIT=11,FILE=    'Time.dtx'     ,STATUS='NEW')
    OPEN(UNIT=12,FILE=    'H.dtx'     ,STATUS='NEW')

  ierr = 0
  omega = 1.0_dp
  omega_big = 0.0_dp
  dt = 0.01_dp        ! time step
  Nt = 10000          ! Number of time points
  dis = 0.1_dp
  lam = 1.0_dp
  initial = 1
  ! build H
```

```
   ALLOCATE(xmat(Nf,Nf),H0(Nf,Nf),Hmat(Nf,Nf),a_plus(Nf,Nf),a(Nf,Nf),x(Nf,Nf),
STAT=ierr)
   DO j=1,Nf
   DO i=1,Nf
   IF(ABS(i-j) .EQ. 1) xmat(i,j) = 0.5_dp*SQRT(REAL(i+j-1,dp))
   END DO
   END DO
   ! WRITE(12,FMT='(E15.8,1X,E15.8)') xmat(2,1)
   DO j = 1,Nf
    H0(j,j) = CMPLX((REAL(j-1,dp))+0.5_dp,0.0_dp,dp)
   END DO


    Hmat = Czero
    a_plus = Czero
    a = Czero


    DO j = 1,Nf
    DO i = 1,Nf
    IF ((ABS(i-j) .EQ. 1) .AND. (i .GT. j)) THEN
     a_plus(i,j)=xmat(i,j)
     ELSE IF ((ABS(i-j) .EQ. 1) .AND. (j .GT. i)) THEN
     a = xmat(i,j)
    END IF
    END DO
    END DO
    Hmat = H0+lam*xMat
   x = (a_plus + a)/SQRT(2.0_dp)


                  WRITE(12,FMT='(E15.8,1X,E15.8)') a(2,3)


    ALLOCATE(rho0(Nf,Nf),rho_in(Nf,Nf),rho1(Nf,Nf),rhoa(Nf,Nf),rhoplus(Nf,Nf), &
    rhoa2(Nf,Nf),rhoplus2(Nf,Nf), test(Nf,Nf),STAT = ierr)
rho1 = Czero
! build rho(t=0)
  rho0 = Czero
  rho0(initial,initial) = Cunit
  rho_in = rho0
  t = 0.0_dp
  DO it = 0,Nt
    t = t + dt



trace1 = 1.0_dp


trace2 = 0.0_dp
num = 0

DO WHILE (abs(trace1-trace2)>1.0E-6_dp)

trace1 = Czero
do i = 1, Nf
  trace1 = trace1 + rho_in(i,i)
end do

trace2 = Czero
do i = 1, Nf
  trace2 = trace2 + rho1(i,i)
end do

rho1 = rho0 + dt*0.5_dp*((-ci*(MATMUL(Hmat,rho0)-MATMUL(rho0,Hmat))&
          +dis*0.5_dp*(MATMUL(MATMUL(a,rho0),a_plus)-
MATMUL(a_plus,MATMUL(a,rho0)) + &
                    MATMUL(a,MATMUL(rho0,a_plus))-MATMUL(MATMUL(rho0,a_plus),a)))+ &
(-ci*(MATMUL(Hmat,rho_in)-MATMUL(rho_in,Hmat))&
          +dis*0.5_dp*(MATMUL(MATMUL(a,rho_in),a_plus)-
MATMUL(a_plus,MATMUL(a,rho_in)) + &
                    MATMUL(a,MATMUL(rho_in,a_plus))-
MATMUL(MATMUL(rho_in,a_plus),a))))

                    test = MATMUL(MATMUL(a,rho0),a_plus)
                    !WRITE(12,FMT='(E15.8,1X,E15.8)') test(1,1)

  rho_in = rho1
  num = num+1
END DO
```

```
trace = Czero
do i = 1, Nf
  trace = trace + rho1(i,i)
end do
  xp = matmul(x,rho1)

avg_x = Czero
do i = 1, Nf
  avg_x = avg_x + xp(i,i)
end do
! print diagonals


 WRITE(11,FMT='(E15.8,1X,E15.8,2X,1000(1X,E15.8))') t,trace,(REAL(rho1(k,k)),k=1,Nf)

rho0 = rho1
rho_in = rho1
rho1 = Czero

END DO
END PROGRAM Time
```

II. CODE FOR FINDING OSCILLATIONS

```
PROGRAM Time


  IMPLICIT NONE


   INTEGER,       PARAMETER   :: dp = 8   ! Double precision parameter

   INTEGER,       PARAMETER    :: NumThreads    = 2 !4

   INTEGER,       PARAMETER    :: Nf  = 10  ! 128
   INTEGER,       PARAMETER    :: Nf2 = 9

!------------------------------------------------------------------------

   REAL(KIND=dp),    PARAMETER    :: pi   = 3.14159265358979324_dp
   REAL(KIND=dp),    PARAMETER    :: pi2i = 1.0_dp/(2.0_dp*pi)
   REAL(KIND=dp),    PARAMETER    :: pid2 = pi/2.0_dp
   REAL(KIND=dp),    PARAMETER    :: hbar = 1.05*10**(-34)

   COMPLEX(KIND=dp),  PARAMETER   :: ci    = CMPLX(0.0_dp, 1.0_dp)
   COMPLEX(KIND=dp),  PARAMETER   :: CUnit = CMPLX(1.0_dp, 0.0_dp)
   COMPLEX(KIND=dp),  PARAMETER   :: Czero = CMPLX(0.0_dp, 0.0_dp)

   CHARACTER(LEN=1), PARAMETER    :: JOB   = 'V', UPLO  = 'U'
   CHARACTER(LEN=1), PARAMETER    :: TRANS = 'N', RANGO = 'I'


   INTEGER                :: Nmax

   REAL(KIND=dp)           :: al

   COMPLEX(KIND=dp)         :: cl

   CHARACTER(LEN=1)         :: TransA, TransB
```

```
!---------------------------------------------------------------

  REAL(KIND=dp),      ALLOCATABLE, DIMENSION(:)     :: Eigval, E0,sum1,sum2, steady


  COMPLEX(KIND=dp),    ALLOCATABLE, DIMENSION(:,:)   ::  Eigvect,Eigvect0,
H0,Hmat, Cn,M1,rho0,rho1,rho2, &
  rho_in,Hadd, Hin, a_plus, a, x, xp, xmat


!------- Local variables -----------------------------------
!
  INTEGER      :: i, j, ierr, k, num,Nt,it,initial,k1
  REAL         :: trace, avg_x, trace1,trace2, scaler,delta_t,dt,t,
omega,omega_big,dis,lam,error


!
  OPEN(UNIT=11,FILE=     'Time2.dtx'     ,STATUS='NEW')
   OPEN(UNIT=12,FILE=     'H2.dtx'      ,STATUS='NEW')

  ierr = 0
  omega = 1.0_dp
  omega_big = 0.0_dp
  dt = 0.01_dp          ! time step
  Nt = 100000           ! Number of time points
  dis = 0.1_dp
  lam = 0.0_dp
  steady = (/0.10000000E+01, 0.10604975E-42, 0.33914169E-86, 0.32698460-130,
0.00000000E+00,&
   0.00000000E+00, 0.00000000E+00, 0.00000000E+00, 0.00000000E+00,
0.00000000E+00/)
  ! build H

  ALLOCATE(xmat(Nf,Nf),H0(Nf,Nf),Hmat(Nf,Nf),a_plus(Nf,Nf),a(Nf,Nf),x(Nf,Nf),
STAT=ierr)
  DO j=1,Nf
  DO i=1,Nf
  IF (ABS(i-j) .EQ. 1) xmat(i,j) = 0.5_dp*SQRT(REAL(i+j-1,dp))
  END DO
  END DO
  DO j = 1,Nf
   H0(j,j) = CMPLX((REAL(j-1,dp))+0.5_dp,0.0_dp,dp)
  END DO


   Hmat = Czero
   a_plus = Czero
   a = Czero


   DO j = 1,Nf
   DO i = 1,Nf
   IF (i==j+1) THEN
     a_plus(i,j) = SQRT(2.0_dp)*xmat(i,j)
  END IF

    IF (j==i+1) THEN
    a(i,j) = SQRT(2.0_dp)*xmat(i,j)
   END IF
   END DO
   END DO
   Hmat = H0+lam*xMat
    DO i=1,Nf
   !WRITE(12,FMT='(1000(1X,E15.8))') (REAL(a(i,k)),k=1,Nf)
   END DO
   !Hmat = omega*(matmul(a_plus,a)+0.5_dp)+omega_big*(a_plus+a)
  x = (a_plus + a)/SQRT(2.0_dp)




   ALLOCATE(rho0(Nf,Nf),rho_in(Nf,Nf),rho1(Nf,Nf),STAT = ierr)
rho1 = Czero
! build rho(t=0)
  rho0 = 0.0_dp
  DO i=1,Nf
   rho0(i,i) = steady(i)
END DO
  !rho0 = matmul(matmul((a_plus+a),(a_plus+a)),rho0)
  rho0 = matmul((a_plus+a),rho0)
  rho_in = rho0

  t = 0.0_dp
  DO it = 0,Nt
   t = t + dt



trace1 = 1.0_dp


trace2 = 0.0_dp
num = 0
k=1
DO WHILE (abs(trace1-trace2)>1.0E-20_dp)
!SDO k=1,10
!DO WHILE (SQRT(ABS(SUM(rho1-rho_in)))>1.0E-6_dp)

trace1 = Czero
do i = 1, Nf
 trace1 = trace1 + rho_in(i,i)
end do

trace2 = Czero
do i = 1, Nf
 trace2 = trace2 + rho1(i,i)
end do

rho1 = rho0 + dt*0.5_dp*((-ci*(MATMUL(Hmat,rho0)-MATMUL(rho0,Hmat))&
       +dis*0.5_dp*(MATMUL(MATMUL(a,rho0),a_plus)-
MATMUL(a_plus,MATMUL(a,rho0)) + &
             MATMUL(a,MATMUL(rho0,a_plus))-MATMUL(MATMUL(rho0,a_plus),a)))+ &
(-ci*(MATMUL(Hmat,rho_in)-MATMUL(rho_in,Hmat))&
       +dis*0.5_dp*(MATMUL(MATMUL(a,rho_in),a_plus)-
MATMUL(a_plus,MATMUL(a,rho_in)) + &
             MATMUL(a,MATMUL(rho_in,a_plus))-
MATMUL(MATMUL(rho_in,a_plus),a))))

             !WRITE(12,FMT='(E15.8,1X,E15.8)') test(1,1)
  Trace = Czero
  DO k = 1, Nf
```

```
     Trace = Trace+rho1(k,k)
    END DO
    error = Czero
    error = SQRT(ABS(SUM(rho1-rho_in)))

         IF(ABS(error) .LE. 1E-8) EXIT

   rho_in = rho1
   num = num+1
END DO


trace = 0.0_dp
do i = 1, Nf
  trace = trace + rho1(i,i
end do
  xp = matmul(x,rho1)

avg_x = Czero
do i = 1, Nf
  avg_x = avg_x + xp(i,i)
end do
! print diagonals

 !WRITE(11,FMT='(E15.8,1X,E15.8)') t*0.65820_dp,REAL(sum(rho1))
WRITE(11,FMT='(E15.8,1X,E15.8,2X,1000(1X,E15.8,1X,E15.8))') t,trace,(rho1(k,k),k=1,Nf)

rho0 = rho1
rho_in = rho1
rho1 = Czero

END DO

END PROGRAM Time
```

WORKS CITED

Háskóli Íslands. *Viðar Guðmundsson* Available at: https://english.hi.is/staff/vidar. (Accessed: 3rd December 2019)

Jaynes-Cummings Hamiltonian. *Stanford* Available at: https://web.stanford.edu/~ajlucas/Jaynes Cummings Hamiltonian.pdf. (Accessed: 3rd December 2019)

Elton, D. Why physicists still use Fortran. *More is Different* (2015). Available at: http://moreisdifferent.com/2015/07/16/why-physicsts-still-use-fortran/. (Accessed: 3rd December 2019)

FFTPACK. *NetLib* Available at: https://www.netlib.org/fftpack/. (Accessed: 3rd December 2019)

Wavefunctions. *Quantum Harmonic Oscillator* Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/hosc5.html. (Accessed: 4th December 2019)

Physkid, PhyskidPhyskid 64044 silver badges88 bronze badges & ubpdqnubpdqn 51.7k22 gold badges4343 silver badges116116 bronze badges. odd matrix operation. *Mathematica Stack Exchange* (1967). Available at: https://mathematica.stackexchange.com/questions/146467/odd-matrix-operation. (Accessed: 4th December 2019)

Viðar´s sources explaining the basic setup:
https://notendur.hi.is/vidar/Nam/TE/V01-2018.pdf
https://notendur.hi.is/vidar/Nam/TE/V02-2018.pdf