NUMERICAL SIMULATION OF A TWO-LEVEL ATOM IN A MODULATED STANDING WAVE

by

CYRUS BHARUCHA, B.A., B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF ARTS

THE UNIVERSITY OF TEXAS AT AUSTIN

August, 1994

NUMERICAL SIMULATION OF A TWO-LEVEL ATOM IN A MODULATED STANDING WAVE

APPROVED BY

SUPERVISORY COMMITTEE:

Supervisor: _____

Mark G. Raizen

_____

Greg O. Sitz

To all my teachers.

Acknowledgments

I am indebted to my advisor, Professor Mark Raizen, for all of his encouragement, guidance, and support for my training as a physicist. Dr. Paul Williams started our analyses of this system and wrote the initial versions of the software used for these calculations. Sincere thanks to Dr. Fred Moore for all his advice and generous help on this and many other projects. Thanks also go to John Robinson for being an unending source of useful discussions and for introducing me to Chocolate Infinity ice cream. Georgios Georgakis and Martin Fischer have my gratitude for their help in laying out this thesis.

And I am deeply grateful to my family for its love and support in all of my endeavors.

Cyrus Bharucha

The University of Texas at Austin

August, 1994

ABSTRACT

NUMERICAL SIMULATION OF A TWO-LEVEL ATOM IN A MODULATED STANDING WAVE

by

CYRUS BHARUCHA, M.A.

The University of Texas at Austin, 1994

SUPERVISOR: Mark G. Raizen

The motion of a particle in a one-dimensional moving potential well was modeled by integrating the Schroedinger equation for the system. This potential well is sinusoidal in space, ϕ, and its velocity is modulated in time, giving it the form $V(\phi, t) = k\cos(\phi - \lambda\sin(t))$.

This potential can be generated by placing a two-level atom in a standing wave of light whose propagating and counter-propagating components are differentially modulated in frequency. The calculation demonstrates that this system exhibits dynamical localization of the particle's momentum for large values of λ. This model was used in preliminary analyses of a system which has now been experimentally realized in our laboratory.

Table of Contents

Chapter 1

Introduction

In this thesis I will describe the numerical simulation of a quantum mechanical system which exhibits chaotic behavior in the classical limit. Such systems have been the subject of theoretical studies for the striking differences between their classical and quantum behaviors [1]. The system studied in this work is described by the one-dimensional, single particle Hamiltonian,

$$\hat{H} = \frac{\hat{p}^2}{2} - k\cos(\hat{\phi} - \lambda\sin(t)) \ . \tag{1.1}$$

Here $\hat{p}$ and $\hat{\phi}$ are the conjugate momentum and position of the particle, with the commutation relation $[\hat{p}, \hat{\phi}] = -i\hbar$. All quantities in this equation are in dimensionless units. The evolution of the wavefunction $\Psi(\phi, t)$ for this particle is governed by the Schroedinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi(\phi,t) = \left[-\frac{\hbar^2}{2}\frac{\partial^2}{\partial\phi^2} - k\cos(\phi - \lambda\sin(t))\right]\ \Psi(\phi,t). \tag{1.2}$$

In this representation the operator $\hat{p}$ is given by $-i\hbar\partial/\partial\phi$ and $\hat{\phi}$ is the scalar, ϕ.

The system's behavior is that of a particle in a sinusoidal potential well $V(\phi, t) = -k\cos(\phi - \lambda\sin(t))$. The well has an amplitude k and moves back and forth with a maximum excursion λ. The antinode originally at the origin moves from $\phi = 0$ to $\phi = \lambda$. It then turns around and moves to $\phi = -\lambda$, and it returns to $\phi = 0$ at $t = 2j\pi$ (j = integer). The other nodes and antinodes

of the well also oscillate sinusoidally in time around their central positions. The quantities k, λ, and $\bar{\hbar}$ are dimensionless parameters of the system. As can be seen from the Hamiltonian, k and λ indicate respectively the depth and excursion of the potential well, and $\bar{\hbar}$ is the rescaled Planck's constant. This system is of special interest because it can be realized experimentally with an atom in a modulated standing wave [2,3]. In the experimental realization, the standing wave consists of two counter-propagating beams of coherent light detuned from a transition frequency of the atom. In this environment, the atom can be modeled as a two-level system. When illuminated by this standing wave, the dipole interaction of the atom with the electric field induces a potential energy with an amplitude that is sinusoidal in displacement along the axis of the standing wave. If the frequencies of the two beams are differentially modulated, the standing wave moves back and forth along its axis, and the atom experiences a potential energy of the form seen in Eqn. 1.1.

The detuning of the light beams from the atomic transition can be made large enough to neglect dissipation due to spontaneous emission from the excited state of the transition. In addition, all the parameters that determine the system's behavior, namely the maximum excursion of the potential well (λ), the well depth (k), and the rescaled Planck's constant ($\bar{\hbar}$), can be experimentally adjusted.

Chapter 2

Integration of the Schroedinger Equation

The Schroedinger equation presented in the Introduction is the governing equation for the system described in Eqn. 1.1. For clarity, Schroedinger's equation is repeated here,

$$i\bar{k}\frac{\partial}{\partial t}\Psi(\phi,t) = \left[-\frac{\bar{k}^2}{2}\frac{\partial^2}{\partial\phi^2} - k\cos(\phi - \lambda\sin(t))\right]\ \Psi(\phi,t). \tag{2.1}$$

The purpose of the software developed for this work was to solve this equation for specific values of k, $\bar{k}$, and λ accessible by the experiment. This chapter presents the method used to derive a recursion relation to describe how $\Psi(\phi,t)$ evolves over a small step in time τ, assuming boundary conditions with the periodicity of the potential energy term.

The operator $\hat{p} = -i\bar{k}\partial/\partial\phi$ has eigenstates $\psi_m(\phi) = \frac{1}{\sqrt{2\pi}}e^{im\phi}$, with $\hat{p}\psi_m(\phi) = m\bar{k}\psi_m(\phi)$. The wavefunction $\Psi(\phi,t)$ can be expanded in these momentum eigenstates,

$$\Psi(\phi,t) = \sum_{m=-\infty}^{\infty} A_m(t)\ \psi_m(\phi) = \sum_{m=-\infty}^{\infty} A_m(t)\ \frac{e^{im\phi}}{\sqrt{2\pi}}\ . \tag{2.2}$$

The complex coefficients $A_m(t)$ represent the amplitudes of the momentum eigenstates which make up $\Psi(\phi,t)$. These coefficients are given by

$$A_m(t) = \langle\psi_m(\phi)|\Psi(\phi,t)\rangle = \int_0^{2\pi} d\phi\ \Psi(\phi,t)\ \frac{e^{-im\phi}}{\sqrt{2\pi}}. \tag{2.3}$$

Note that $|A_m(t)|^2$ is the probability at time t of the particle being in eigenstate $\psi_m(\phi)$. Note also that the above two equations identify the $A_m(t)$'s as the coefficients in a Fourier series expansion of $\Psi(\phi, t)$. As will be seen in the next chapter, these properties of the momentum coefficients are used extensively by the program.

Following the suggestion of Martin Schlautmann [4], we approximated the Hamiltonian, Eqn. 1.1, by discretizing the potential term. The new potential term,

$$V(\phi, t) = -k\cos(\phi - \lambda\sin(t)) \sum_{j=-\infty}^{\infty} \delta(\frac{t}{\tau} - j), \tag{2.4}$$

is zero except for delta-function pulses at discrete times $t = j\tau$, where j is an integer. This model assumes that the particle evolves freely except for pulses acting at time intervals separated by τ.

This assumption is equivalent to making an impulse approximation. In general, the change in the momentum of a particle over a time interval τ is given by the time integral of the force over the interval,

$$p(t+\tau) - p(t) = \int_t^{t+\tau} \left(-\frac{\partial V}{\partial \phi}\right) dt' \ . \tag{2.5}$$

In the impulse approximation, the force is assumed to be constant for the duration of the interval. The resulting expression for the change in momentum,

$$p(t+\tau) - p(t) = \left(-\frac{\partial V}{\partial \phi}\right)_{t+\tau} \tau \ , \tag{2.6}$$

is the same result as obtained by applying Eqn. 2.5 to the discretized potential in Eqn. 2.4.

Using the discretized potential, the Schroedinger equation becomes

$$i\bar{k}\frac{\partial}{\partial t}\Psi(\phi,t) = \left[-\frac{\bar{k}^2}{2}\frac{\partial^2}{\partial\phi^2} - k\cos(\phi - \lambda\sin(t))\sum_{j=-\infty}^{\infty}\delta(\frac{t}{\tau} - j)\right]\ \Psi(\phi,t)\ , \quad (2.7)$$

$$(\tau \ll 1)\ .$$

The evolution of the wavefunction over one time interval τ can be studied in two parts. The first part of the evolution occurs between successive impulses from $t_1 = (j\tau)^+$ to $t_2 = ((j+1)\tau)^-$. The second part occurs across an impulse from $t_2 = ((j+1)\tau)^-$ to $t_3 = ((j+1)\tau)^+ = t_1 + \tau$.

In the first part of the evolution of $\Psi(\phi,t)$ (*between* the impulses) the potential term of the Hamiltonian is zero, and Eqn. 2.7 becomes an equation for a particle undergoing free evolution. Substituting the Fourier expansion for $\Psi(\phi,t)$ (Eqn. 2.2), we get the following series equation for the momentum amplitudes A_m,

$$i\bar{k}\sum_{m=-\infty}^{\infty}\frac{\partial A_m(t)}{\partial t}\ \frac{e^{im\phi}}{\sqrt{2\pi}} = \sum_{m=-\infty}^{\infty}\frac{\bar{k}^2m^2}{2}\ A_m(t)\ \frac{e^{im\phi}}{\sqrt{2\pi}}\ . \quad (2.8)$$

By applying $\int_0^{2\pi} d\phi\ e^{-in\phi}$ to both sides the sum is eliminated, leaving an equation for $A_n(t)$,

$$i\bar{k}\frac{\partial}{\partial t}A_n(t) = \frac{\bar{k}^2n^2}{2}\ A_n(t)\ . \quad (2.9)$$

By dividing both sides of this equation by $A_n(t)$ and integrating over t from t_1 to t_2, we get an expression which gives the value of the coefficient A_n at t_2 in terms of its value at t_1,

$$A_n(t_2) = A_n(t_1)\ e^{-i\bar{k}n^2\tau/2}\ . \quad (2.10)$$

In the second part of the evolution, *across* a delta function pulse, the first term in the square brackets of the Schroedinger equation (Eqn. 2.7) is negligible compared to the pulse. The equation then becomes

$$i\hbar\frac{\partial}{\partial t}\Psi(\phi,t) = \left[-k\cos(\phi-\lambda\sin(t))\sum_{j=-\infty}^{\infty}\delta(\frac{t}{\tau}-j)\right]\ \Psi(\phi,t). \tag{2.11}$$

By dividing both sides of this equation by $\Psi(\phi,t)$ and integrating over t from $t_2=((j+1)\tau)^-$ to $t_3=((j+1)\tau)^+$, we get an expression for the value of $\Psi(\phi,t)$ after the pulse in terms of its value before the pulse,

$$\Psi(\phi,t_3) = \Psi(\phi,t_2)\ \exp\left(i\frac{k\tau}{k}\cos(\phi-\lambda\sin(t_3))\right). \tag{2.12}$$

This equation can be expanded in the momentum basis using the Fourier expansion in Eqn. 2.2:

$$\sum_{m=-\infty}^{\infty}A_m(t_3)\ \frac{e^{im\phi}}{\sqrt{2\pi}} = \sum_{m=-\infty}^{\infty}A_m(t_2)\ \frac{e^{im\phi}}{\sqrt{2\pi}}\ \exp\left(i\frac{k\tau}{k}\cos(\phi-\lambda\sin(t_3))\right). \tag{2.13}$$

To find the behavior of a particular A_m, the right side of this equation can first be rewritten so that its ϕ-dependence is expressed only in terms of $e^{in\phi}$'s. This expansion can be done with the identity $e^{ix\cos\theta}=\sum_{l=-\infty}^{\infty}i^lJ_l(x)e^{il\theta}$:

$$\sum_{m=-\infty}^{\infty}A_m(t_3)\ \frac{e^{im\phi}}{\sqrt{2\pi}}\ = \sum_{m,l=-\infty}^{\infty}A_m(t_2)\ \frac{e^{im\phi}}{\sqrt{2\pi}}\ i^lJ_l(\frac{k\tau}{k})e^{il\phi}\ e^{-il\lambda\sin(t_3)}\ . \tag{2.14}$$

By again applying $\int_0^{2\pi}d\phi\ e^{-in\phi}$ to both sides, the sum over l is eliminated, leaving an equation for $A_n(t)$ after the pulse in terms of the $A_m(t)$'s before the pulse,

$$A_n(t_3) = \sum_{m=-\infty}^{\infty}A_m(t_2)i^{n-m}J_{n-m}\left(\frac{k\tau}{k}\right)e^{-i(n-m)\lambda\sin(t_3)}\ . \tag{2.15}$$

Since t_3 and t_1 are instants in time after subsequent pulses ($t_3 = t_1 + \tau$), Equation 2.10 can be substituted into 2.15 to get a recursion relation which gives the value of the coefficients $A_n(t+\tau)$ in terms of their earlier values $A_m(t)$.

$$A_n(t+\tau) = \sum_{m=-\infty}^{\infty} M_{n,m}(t) A_m(t) \ , \tag{2.16}$$

$$M_{n,m}(t) = e^{-i\hbar m^2 \tau/2} \ i^{n-m} \ J_{n-m}\left(\frac{k\tau}{\hbar}\right) \ e^{-i(n-m)\lambda \sin(t+\tau)} \ . \tag{2.17}$$

This formulation, in principle, can be used to calculate the wavefunction for this system at any given time. Given an initial wavefunction $\Psi(\phi, t=0)$ with momentum amplitudes $A_m(t=0)$, the amplitudes at a time $t = j\tau$ later can be found by j iterations of Eqn. 2.16. Unfortunately, Eqn. 2.16 poses a computational difficulty: it requires an accurate evaluation of the Bessel functions J_m. This problem became apparent to us when the computation would not preserve probability as it stepped forward in time: as the iterations were performed, the sum $\sum_n |A_n(t)|^2$ would wander from its initial value of 1. The computational tools available to us did not provide sufficient accuracy of these functions at high orders m to correct the problem.

An alternative recursion relation was used which is not as straightforward to implement as Eqn. 2.16, but which is not as susceptible to errors in special functions. To derive this relation, we first define the functions $B_n(t)$ and $\beta(\phi, t)$:

$$B_n(t) = A_n(t) \ e^{-i\hbar n^2 \tau/2} \ , \tag{2.18}$$

$$\beta(\phi, t) = \sum_{n=-\infty}^{\infty} B_n(t) \ \frac{e^{in\phi}}{\sqrt{2\pi}} \ . \tag{2.19}$$

Then the equation describing the evolution of A_n between pulses (Eqn. 2.10) can be rewritten as

$$A_n(t_2) = B_n(t_1), \tag{2.20}$$

and the expansions for $\Psi(\phi,t)$ and $\beta(\phi,t)$ (Eqns. 2.2 and 2.19) can be combined into a relationship between $\Psi(\phi,t)$ and $\beta(\phi,t)$:

$$\begin{aligned} \Psi(\phi,t_2) &= \sum_{m=-\infty}^{\infty} A_m(t_2)\ \frac{e^{im\phi}}{\sqrt{2\pi}} \\ &= \sum_{m=-\infty}^{\infty} B_m(t_1)\ \frac{e^{im\phi}}{\sqrt{2\pi}} \\ &= \beta(\phi,t_1)\ . \end{aligned} \tag{2.21}$$

This relationship can be used to rewrite the equation describing the evolution of $\Psi(\phi,t)$ across a delta function pulse (Eqn. 2.12):

$$\Psi(\phi,t_3{=}t_1{+}\tau) = \beta(\phi,t_1)\ \exp\left(i\frac{k\tau}{k}\cos(\phi-\lambda\sin(t_3))\right)\ . \tag{2.22}$$

Using the definition of the A_m's (Eqn. 2.3) here leads to the first of two equations which relate the momentum amplitudes at a time $t{+}\tau$ to the amplitudes at t:

$$A_m(t+\tau) = \ \frac{1}{\sqrt{2\pi}}\int_0^{2\pi} d\phi\ \beta(\phi,t)\ e^{-im\phi}\ \exp\left(i\frac{k\tau}{k}\cos(\phi-\lambda\sin(t+\tau))\right)\ . \tag{2.23}$$

The second of the two equations results from combining the definitions of $B_n(t)$ and $\beta(\phi,t)$ (Equations 2.18 and 2.19) to get an expression for $\beta(\phi,t)$ as a function of $A_n(t)$,

$$\beta(\phi,t) = \frac{1}{\sqrt{2\pi}}\sum_{n=-\infty}^{\infty} A_n(t)\ e^{-i\hbar n^2\tau/2}\ e^{in\phi}\ . \tag{2.24}$$

Substituting this expression for $\beta(\phi, t)$ into the previous equation (2.23) provides the recursion relation for the coefficients A_n, making it possible to find $A_n(t+\tau)$ in terms of $A_n(t)$. Given an initial wavefunction $\Psi(\phi, t = 0)$ with momentum amplitudes $A_m(t = 0)$, the coefficients at a time $t = j\tau$ later can be found by j iterations of this process. As will be seen in the next chapter, the similarity of this recursion relationship to a Fourier transform makes it the heart of the computational scheme used in the program.

Chapter 3

Program Operation

This chapter describes how the programs use the recursion relationship derived in the last chapter to evolve the wavefuntion Ψ in time.

3.1 Input to the Program

The system parameters k, $\bar{k}$, and λ are read by the program from an input file. This file also contains the number of time steps τ to evolve the wavefunction. A second input file contains the initial values of the complex momentum amplitudes, $A_n(t=0)$.

Since these amplitudes are the coefficients of the momentum eigenstates in Eqn. 2.2, $|A_n(t)|^2$ is the probability at time t that the particle has a momentum $p=\bar{k}n$. The complex coefficients $A_n(0)$ therefore specify the initial momentum distribution of the particle wavepacket. Note that since the amplitudes are also the coefficients of the Fourier series for $\Psi(\phi,t)$, the vector $A_n(t)$ specifies the spatial distribution of the wavepacket as well as its momentum distribution.

3.2 Program Output

The program takes the initial A_n's as a vector of complex numbers and uses the recursion relation developed in the previous chapter to calculate their value at $t = \tau$. This calculation is repeated to evolve the wavepacket in time. After each iteration, information about the momentum distribution is recorded in an output file. In its current configuration, the program records the time t, the expectation value of the wavepacket's momentum, $\langle p \rangle$, and the RMS spread of its momentum, Δp. (The program can easily be modified to record other information about the A_n's after each time step.) These quantities are found by using $|A_n|^2$ as a probability distribution function:

$$\langle p \rangle = \sum_{n=-\infty}^{\infty} |A_n|^2 \; n\hbar\bar{k}, \tag{3.1}$$

$$\Delta p = \left[\sum_{n=-\infty}^{\infty} |A_n|^2 \; (\; n\hbar\bar{k} - \langle p \rangle)^2 \right]^{\frac{1}{2}} \tag{3.2}$$

As the program is executed, it creates an output file containing three columns of numbers. The first column indicates the elapsed time in the simulation. The next two columns give the values of $\langle p \rangle$ and Δp at those times. When the program is finished running, it creates a second output file containing the final momentum distribution. This file contains two columns: the first column is a list of m's, and the second column lists the corresponding $|A_m|^2$'s.

3.3 Truncation of the Wavepacket

One of the questions involved in a calculation like this is the number of terms to keep from the infinite series of A_n's which represents the wavepacket.

This number is a parameter in the program which can be chosen to make the calculation run in the minimum amount of time without sacrificing accuracy. In our runs so far we have typically chosen to keep $N = 64$ terms: keeping A_n's with $|n| > 32$ does not make the calculation more accurate since those terms do not get significantly populated for the values of k, $\bar{k}$, and λ we have been using.

Truncating the series of A_n's has the effect of changing the Fourier series in equations Eqns. 2.2 and 2.3 into a discrete Fourier transform pair,

$$\Psi(\phi_m, t) = \frac{1}{\sqrt{N}} \sum_{n=-N/2}^{N/2-1} A_n(t)\ e^{imn2\pi/N}\ , \tag{3.3}$$

$$A_n(t) = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} \Psi(\phi_m, t)\ e^{-imn2\pi/N}\ . \tag{3.4}$$

Here ϕ becomes the discrete variable ϕ_m,

$$\phi_m = m\Delta\phi\ , \qquad \left(\Delta\phi \equiv \frac{2\pi}{N}\ , \quad m = \text{integer}\right)\ . \tag{3.5}$$

This discrete Fourier transform pair preserves probability better than the previous truncated Bessel expansion.

3.4 Evolving the Wavepacket

Just as $|A_n(t)|^2$ is the probability of finding the particle in a momentum eigenstate $\psi_n(\phi)$, the quantity $|\Psi(\phi_m, t)|^2$ is the probability of finding the particle in the interval $\phi \in [\phi_m,\ \phi_m + \Delta\phi)$. In this discretized basis the integral over ϕ in the recursion relation 2.23 becomes a sum over ϕ_m's. The new

recursion relation for finding the $A_n(t+\tau)$'s in terms of the $A_n(t)$'s is then

$$A_n(t+\tau) = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} \beta(\phi_m, t)\ e^{-imn2\pi/N}\ \exp\left(i\frac{k\tau}{k} \cos(m\frac{2\pi}{N} - \lambda \sin(t+\tau)) \right), \tag{3.6}$$

where β is now given by

$$\beta(\phi_m, t) = \frac{1}{\sqrt{N}} \sum_{n=-N/2}^{N/2-1} A_n(t)\ e^{-in^2 k\tau/2}\ e^{imn2\pi/N}\ . \tag{3.7}$$

This algorithm preserves the phase coherence between the A_n's as it steps through time, so the vector $A_n(t)$ remains a complete description of the wavepacket.

These two equations express the vectors $A_n(t)$ and $\beta(\phi_m, t)$ as discrete Fourier transforms of each another with an additional phase term. They prescribe six steps for calculating the evolution of the momentum amplitudes over one step in time. These steps, which transform $A_n(t)$ into $A_n(t+\tau)$, are performed inside a program loop in the following manner.

1. Multiply each $A_n(t)$ by $e^{-in^2 k\tau/2}$.
2. Perform a fast Fourier transform (FFT) [5] on the A_n's to construct the $\beta(\phi_m, t)$'s.
3. Multiply each $\beta(\phi_m, t)$ by $\exp\left(i\frac{k\tau}{k} \cos(m\frac{2\pi}{N} - \lambda \sin(t+\tau)) \right)$.
4. Calculate the $A_n(t+\tau)$'s by an inverse FFT of the $\beta(\phi_m, t)$'s.
5. Record the time t and information on the new distribution in a file.
6. Increment the time t by τ, and loop back to step 1.

This process is repeated a given number of times as specified by the program's input. The time step we used for each iteration was typically $\tau = \frac{1}{300}$; test runs indicated that smaller values of τ did not seem to affect the calculations.

Chapter 4

Sample Runs

In this chapter I will present examples of data produced by the program under two sets of input conditions. The first set of conditions, with $k = 15.0$ and $\bar{k} = 1.58$, is the same as presented by Graham *et. al.* [2]. The second set of conditions, with $k = 0.36$ and $\bar{k} = 0.16$, corresponds to an experiment with Sodium atoms in a modulated standing wave which we recently performed [3].

For both simulations, the initial wavefunction had a Gaussian distribution in momentum with a mean momentum $\langle p \rangle = 0$ and an RMS momentum spread of $\Delta p = 0.5\bar{k}$. Figure 4.1 shows how $\langle p \rangle$ and Δp evolved under the first set of conditions. This plot was generated directly from the first output file of the program. The second output file was used to generate Figure 4.2, which shows the terminal momentum distribution $|A_n|^2$ for this simulation.

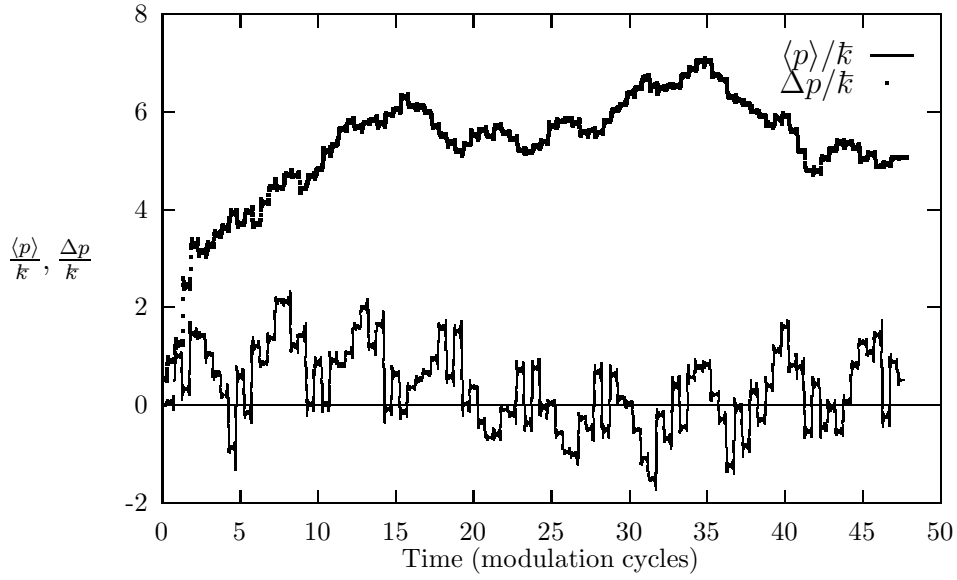

Figure 4.1: Plot of $\langle p \rangle$ and Δp versus time for the conditions $k = 15.0,\ \bar{k} = 1.58,\ \lambda = 85$.

Some interesting aspects of this system can be seen in Figure 4.1. The jagged appearance of the curve $\langle p \rangle$ vs. t indicates that the particle's momentum undergoes sharp changes at certain times. This phenomenon can be understood as "resonant kicks." When the well's velocity ($\lambda \cos t$) matches the velocity of the particle ($\dot{\phi}$), the particle "rolls" down the stationary potential well, experiencing a resonant kick. When the well's velocity is not the same as the particle's velocity the average force on the particle goes to zero. The system crosses this fundamental resonance ($\dot{\phi}(t) = \lambda \cos t$) twice during each period $T = 2\pi$, so the particle can receive two such kicks during each modulation period.

Figure 4.1 also shows a saturation of the momentum spread. After a rapid initial spread, this wavepacket slows its expansion in the momentum

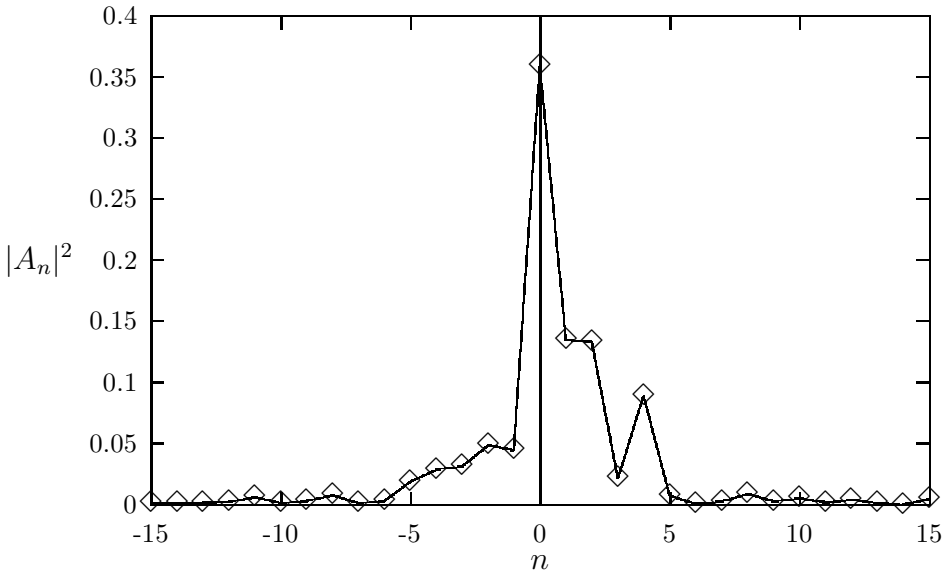

Figure 4.2: Momentum distribution of the wavepacket in Figure 4.1 at $t = 48$.

basis and ultimately reaches a maximum RMS spread of $\Delta p \approx 6\bar{k}$.

Figure 4.3 shows how the saturation of Δp depends on λ for the values of k and $\bar{k}$ used in this first simulation (15.0 and 1.58, respectively). For λ less than a critical value, the saturated Δp increases roughly linearly with λ. This behavior is because the resonant kick phenomenon described above does not occur for particles with a speed faster than the maximum speed of the well. According to the dimensionless Hamiltonian in Eqn. 1.1, this maximum speed is λ. A particle with speed greater than λ will not experience any further resonant kicks, so the speed of the particle does not increase much beyond the maximum speed of the well: $|\dot{\phi}|_{max} \approx \lambda$. Since Hamilton's equations for this system equate $\dot{\phi}$ and p ($\dot{\phi} \equiv \partial H/\partial p = p$), the values of p accessible by resonant kicks is limited to

$$|p|_{max} \approx \lambda. \tag{4.1}$$

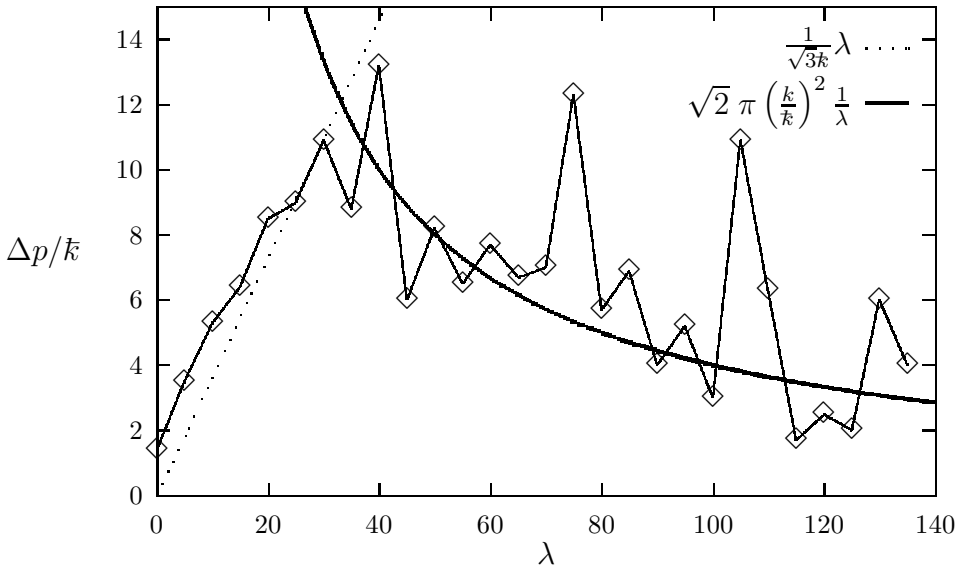

Figure 4.3: Saturation values of Δp as a function of λ for $k = 15.0$ and $\bar{k} = 1.58$.

If the wavepacket is uniformly distributed over this range of p, it will have an RMS spread in momentum of $\lambda/\sqrt{3}$. This value is represented by the dotted line in Figure 4.3; it approximates the behavior observed in the simulation for $\lambda < 35$.

For larger values of λ, other limits on the wavepacket's spread in momentum become apparent. The wavepacket is composed of Floquet states, the eigenstates of a time-dependent Hamiltonian. These Floquet states can be localized in p-space, where each Floquet state is constructed from a finite number of neighboring p-states. Once the initial conditions are set, this wavepacket never spreads to p-states outside those needed to construct the initial Floquet state distribution. Graham *et. al.* used a standard-map approximation of this system to estimate the saturation values of Δp in this regime. Their estimate of the saturated $\Delta p/\bar{k} = \sqrt{2}\pi(k/\bar{k})^2/\lambda$ is plotted in Figure 4.3. This exhibi-

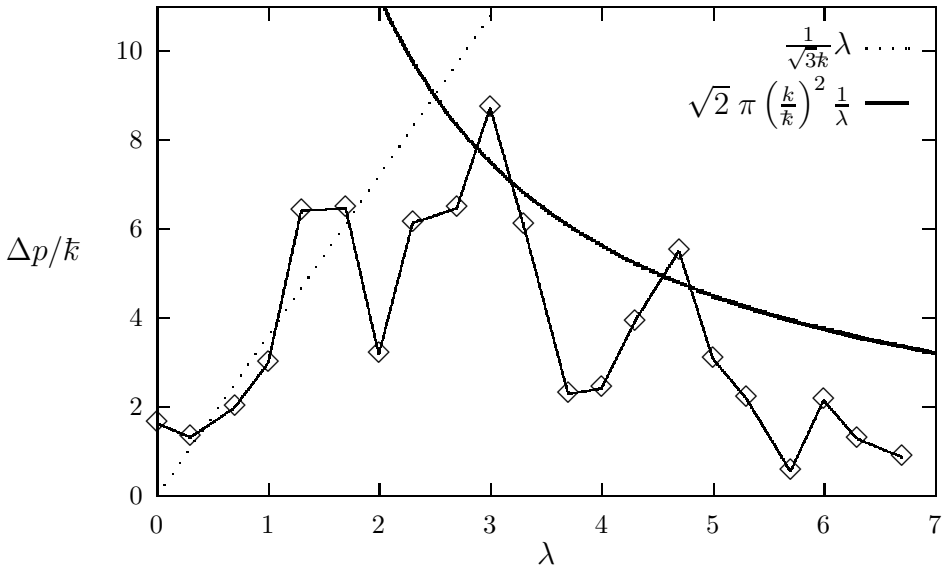

Figure 4.4: Saturation values of Δp as a function of λ for $k = 0.36$ and $\bar{k} = 0.16$.

tion of a saturation of Δp due to the Floquet distribution which makes up the wavepacket is an example of dynamical localization.

Figure 4.4 was generated in the same way as Figure 4.3, by finding the saturation values of Δp for various values of λ with k and $\bar{k}$ fixed. The k and $\bar{k}$ used here (0.36 and 0.16) correspond to the parameters used in our initial experimental realization of this system [3]. These values of k and $\bar{k}$ were experimentally accessible, as was the range of λ (0 to ≈ 7) needed to see the localization. In addition the peak value of Δp ($\approx 9\bar{k}$) was also large enough to make it visible over our background Δp of $\approx 2\bar{k}$. As indicated by this figure, the simulation predicted that an experiment done under these feasible conditions would demonstrate the same effects described above.

Appendix A

Relationship to the Experimental System

This section shows how an atom in a spatially modulated standing wave can be modeled as a particle obeying the Hamiltonian in Eqn. 1.1. We start with two counter-propagating laser beams with matched intensities and polarizations. The electric field along their axis $\vec{x}$ is then,

$$E(x,t) = E_o \cos(\omega_L t - k_L x + \theta_1) + E_o \cos(\omega_L t + k_L x + \theta_2) \ , \tag{A.1}$$

where E_o is the magnitude of each beam's electric field, $\frac{\omega_L}{2\pi}$ is the optical frequency, $k_L = \frac{\omega_L}{c}$, and θ_1 and θ_2 are the relative phases of the beams. If the beams are phase-modulated so that $\theta_1 = -\theta_2 = k_L \Delta L \sin(\omega_m t)$, then the field becomes

$$E(x,t) = 2E_o \cos(\omega_L t) \cos(k_L(x - \Delta L \sin \omega_m t)) \ . \tag{A.2}$$

Here $\frac{\omega_m}{2\pi}$ is the modulation frequency of the beams, and ΔL is the maximum displacement of the standing wave nodes from their central positions.

If ω_L is near a transition frequency of the atom, the atom can be modeled as a two-level system consisting of a ground ($|g\rangle$) and an excited ($|e\rangle$) state of the atom. In reference [2], Graham *et. al.* started with a spatially modulated standing wave with the above form. They used the dipole, two-level atom, and rotating wave approximations. They then perform an adiabatic elimination of the excited state amplitude to show that the Hamiltonian for an

atom in this field is approximated by,

$$H = \frac{p_x^2}{2M} - \frac{\hbar\Omega_{eff}}{8}\cos\left[2k_L(x - \Delta L\sin(\omega_m t))\right]. \tag{A.3}$$

The adiabatic elimination of the excited state is valid provided that the lasers are sufficiently detuned from the atom's resonance frequency. Here p_x is the momentum of the atom along the x-axis, M is its mass, and $\Omega_{eff} = (2\mu E_o)^2/\hbar^2\delta$ is the effective Rabi frequency. μ is the dipole matrix element of the transition between the ground and excited states of the atom, $|\langle g|ex|e\rangle|$, and $\delta/2\pi$ is the detuning of the laser frequency from the atomic transition.

To transform this Hamiltonian into the unitless form of Eqn. 1.1, we first make the following definitions of $k, \bar{k}$, and λ:

$$\begin{aligned} k &\equiv \frac{\hbar k_L^2}{2M\omega_m^2}\,\Omega_{eff} = \frac{\omega_r\Omega_{eff}}{\omega_m^2}, \\ \bar{k} &\equiv \frac{4k_L^2}{M\omega_m}\,\hbar = 8\,\frac{\omega_r}{\omega_m}, \\ \lambda &\equiv 2k_L\,\Delta L, \end{aligned} \tag{A.4}$$

where $\omega_r \equiv \frac{\hbar k_L^2}{2M}$ is the recoil frequency of the atom. Then by scaling the variables t, x, p_x, and H into their unitless counterparts,

$$\begin{aligned} t' &= \omega_m t\ , \\ \phi &= 2k_L x\ , \\ p &= \frac{2k_L}{M\omega_m}\,p_x\ , \\ H' &= \frac{4k_L^2}{M\omega_m^2}\,H\ , \end{aligned} \tag{A.5}$$

we can transform the Hamiltonian for the atom in the standing wave into the form of Eqn. 1.1:

$$H' = \frac{p^2}{2} - k\cos(\phi - \lambda\sin(t'))\ . \tag{A.6}$$

And multiplying the standard Schroedinger equation,

$$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi \ , \tag{A.7}$$

by $\frac{4k_L^2}{M\omega_m^2}$ leads to the unitless Schroedinger equation of Eqn. 1.2:

$$i\bar{k}\,\frac{\partial}{\partial t'}\Psi = \left[\frac{p^2}{2} - k\cos(\phi - \lambda\sin(t'))\right]\ \Psi \ . \tag{A.8}$$

Appendix B

Program Source Code

B.1 Main Routine

```
      PROGRAM qmapMain

c  The real work in this program is done by the subroutine 'qmap'.
c  'qmap' calculates the evolution of the momentum amplitudes
c  (contained in the vector Psi(n) ), and prints the output files.
c  This main routine serves merely as a humble interface between
c  'qmap' and the outside world.


cccccc  Information to be read from the input file          cccccc

      real*4  k, kbar, lambda
c                                 { Parameters of the system      }
      real*4  tMax
c                                 { Total time for the simulation }

      real*4  tau
c                                 { Amount of time to step in     }
c                                 { each iteration                }

      character*15 initCondFile
c                                 { File from which  to read      }
c                                 { initial conditions            }

      character*15 outputFileExt
c                                 { Extension for output files    }

      character*7 date
c                                 { Run date to use in marking    }
c                                 { the output files:  YYMonDD    }


cccccc  Information for the output file "finalProb.dat"     cccccc
```

```
      real*4 finalProb
c                                 { Sum of |psi(n)|*2 at the end  }
c                                 { of each run                   }


cccccc  Internally used variables                            cccccc

      integer simCount
c                                 { Counter for the number of     }
c                                 { different simulations to run  }

      character*15 inputFile, finalProbFile
      parameter(inputFile     = 'input.dat'     )
      parameter(finalProbFile = 'finalProb.dat' )
c                                 { Files to read input parameters}
c                                 { and to write the total final  }
c                                 { probability after each run.   }
c                                 { These files are only used here}
c                                 { in the main routine           }

      character*15 dummyString




cccccc  Read in the various simulation conditions            cccccc
cccccc  and perform the calculation for each of them.        cccccc

c  Read in a set of input parameters and perform the calculation
c  for them.  Record the finalProb returned from the subroutine
c  in the finalProbFile.  Repeat until you reach the end of the
c  input file.

      open(unit=1, file=inputFile, status='old')
      rewind(1)

      open(unit=2, file=finalProbFile, status='unknown')

      simCount = 0

 90   read(1,*,END=100) dummyString, k
         read(1,*)  dummyString, kbar
         read(1,*)  dummyString, lambda
         read(1,*)  dummyString, tMax
         read(1,*)  dummyString, tau
         read(1,*)  dummyString, initCondFile
```

```
         read(1,*)  dummyString, outputFileExt
         read(1,*)  dummyString, date

         nsteps = int(tMax/tau)

         call qmap(k, kbar, lambda, nsteps, tau,
     1            initCondFile, outputFileExt, date,
     2            finalProb)

         simCount = simCount + 1

         write(2,*)'totProb(Simulation #',simCount,')=',finalProb
         goto 90

 100     continue
      close(2)
      close(1)


      end
c         {main}

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

B.2 The qmap Subroutine

```
     SUBROUTINE QMAP (ko, kbar, lambda, nsteps, tau,
     1                initCondFile, outExt, date,
     2                finalProb)


ccccc  INPUT:                                                ccccc

      real*4 ko
C                     { ko = Nominal well depth in the Graham,   }
C                     {     Schlautmann, and Zoller Hamiltonian. }
C                     {     It's used by the subroutine kInTime  }
C                     {     to find the                          }
C                     {     instantaneous well depth k.          }

      real*4 kbar
C                     { kbar=   transformed h-bar in the         }
C                     {         GSZ Hamiltonian                  }
```

```
      real*4 lambda
C                     { lambda = maximum spatial displacement of }
C                     {          the potential well.             }

      integer nsteps
C                     { nsteps = total # of iteration steps      }

      real*4 tau
C                     { tau = amount of time to step in each     }
C                     {       iteration                          }

      character*15 initCondFile, outExt
C                     { name of the file holding the initial     }
C                     { amplitudes, and the extension to use on  }
C                     { the output files                         }

      character*7 date
C                     { used in the output file's headers        }


cccccc   OUTPUT:                                          cccccc


C     file '.n_vs_t.outExt'
C                     { <n> AND <n**2>-<n>**2 vs. (time in       }
C                     { modulation periods)                      }

C     file '.p2_vs_n.outExt'
C                     { |psi(n)|**2 vs. n  at the end of the     }
C                     { calculation                              }

      real*4 finalProb
C                     { finalProb = total |psi|**2 at the end of }
C                     { the calculation                          }



cccccc  DESCRIPTION                                       cccccc


C                                   QMAP


C  By Paul Williams and Cyrus Bharucha, 1993.

C       This subroutine iterates the quantum map for the chaotic
```

```
C  deflection experiment, using a spectral method.
C       It sets up the wave function in a momentum representation
C  as a 1-d array with successive entries being the real and
C  imaginary parts in order to facillitate the use of the  FFT
C  routine "four1" from Numerical Recipies.  The length of the
C  array will be set as a power of 2.  The initial conditions
C  are read from a the file specified in 'initCondFile'.  The
C  map is iterated for a total nuber of steps given as an
C  input parameter.  The iteration of the map is as follows.
C  First the map is iterated for free evolution of the
C  momentum between kicks.  Then an FFT of the wavefunction
C  to a space representation is computed, and the wave
C  function is evolved across the kick.  The wave function is
C  then transformed back to a momentum representation.
C  Other input parameters include the well depth, k, and the
C  modulation strength, lambda.
C       As the map is iterated, the values navg=<p>/kbar and
C  delta-n=delta-p/kbar are calculated and written to a file.
C  After the map is finished, the final probability distribution
C  in momentum is written to a second output file.
C       This subroutine evaluates k at each step in time using the
C  subroutine 'kInTime'.  'ko' is the nominal potential, and 'k'
C  is the instantaneous potential found by multiplying 'ko' and
C  the function given by 'kInTime'.



CCCCCC  Internally used variables                              CCCCCC


      parameter (nn=128)
C                     { the number of terms to keep in the       }
C                     { basis set of Psi                         }

      integer n
C                     { index for the vector psi(n)              }

      real *4 navg, n2avg
C                     { navg  = <p>/kbar                         }
C                     { n2avg = <p**2>/kbar**2                   }

      real *4 psi(2*nn), pstmpr, pstmpi
C                     { psi(n) is the complex vector which holds }
C                     { alternately                              }
C                     {    A(n, t) = momentum amplitudes, and    }
C                     {    Beta(phi(n), t)                       }
```

```
      real *4 psi2, sum
C                     { used for calculating |psi|**2           }

      real *4 pevolc(nn), pevols(nn), arg
C                     { intermediate values in the calculation  }

      real *4 k, k1
C                     { k =  instantaneous value of k, calculated}
C                     {      in each iteration.                 }
C                     { k1 = k*tau/kbar                         }

      integer printInterval
      parameter(printInterval = 20)
C                     { printInterval = how often to write      }
C                     { a line of output, in iteration steps    }

      parameter(psi2minimum=1.0e-30)


CCCCCC  Fundamental constants                                CCCCCC

      parameter(pi=3.1415927)
      parameter(twopi=2*pi)




CCCCC  open output files and write the headers               CCCCCC

      open(unit=4, file='n_vs_t.'//outExt, status='unknown')
         write(4,10) '# ',date
         write(4,11) '# k=      ',ko
         write(4,11) '# kbar=   ',kbar
         write(4,11) '# tau=    ',tau
         write(4,11) '# lambda= ',lambda
         write(4,12) '# File extension = '//outExt
         write(4,13) '# '
         write(4,14) '# T(mod cycles), <n>, sqrt(<n**2>-<n>**2)'
         write(4,14) '#----------------------------------------'
 10      format(a2,  a7)
 11      format(a10, 1G11.5)
 12      format(a34)
 13      format(a2)
 14      format(a41)
```

```
      open(unit=5, file='p2_vs_n.'//outExt, status='unknown')
         write(5,20) '# ',date
         write(5,21) '# k=      ',ko
         write(5,21) '# kbar=   ',kbar
         write(5,21) '# tau=    ',tau
         write(5,21) '# lambda= ',lambda
         write(5,22) '# File extension = '//outExt
         write(5,23) '# '
         write(5,24) '# n      |psi(n)|**2'
         write(5,24) '#--------------------'
 20      format(a2,  a7)
 21      format(a10, 1G11.5)
 22      format(a34)
 23      format(a2)
 24      format(a20)


CCCCCC  Read the initial condition from the input file       CCCCCC
c      write (*,*) 'aa'

      open(unit=7, file=initCondFile, status='old')
      rewind(7)
      do i=1,2*nn
         read(7,*) psi(i)
      end do
      close(7)


CCCCCC  Calculate the vectors to time evolve                 CCCCCC
CCCCCC  the free evolution
c      print *,'bb'

      do i = 1, nn
         n = i-nn/2
         pevolc(i)=cos(n**2*kbar*tau/2)
         pevols(i)=-sin(n**2*kbar*tau/2)
      end do


C  Do loop to iterate the map
c      print *,'cc'

      Do i= 1,nsteps
```

```
c         print *,'beginning time step',i


CCCCCC  Get the current value of k                          CCCCCC

c  ko     = the nominal well depth,
c  T      = the elapsed time (in modulation periods), and
c  Ttotal = the total time for the complete simulation.

         call kInTime(ko, k, i*tau/twopi, nsteps*tau/twopi)
         k1=k*tau/kbar


CCCCCC  Calculate the time evolution of the momentum state  CCCCCC
c      print *,'dd'

         do j = 1, 2*nn-1,2
           pstmpr=psi(j)
           pstmpi=psi(j+1)
           psi(j)=pstmpr*pevolc((j+1)/2)-pstmpi*pevols((j+1)/2)
           psi(j+1)=pstmpr*pevols((j+1)/2)+pstmpi*pevolc((j+1)/2)
         end do


CCCCCC  Now FFT to a postion representation, and then       CCCCCC
CCCCCC  calculate the evolution across the delta function.  CCCCCC
c      print *,'ee'

         call four1(psi,nn,1)


CCCCCC  Remove the oscillation due to the momentum offset   CCCCCC
c      print *,'ff'

         do j=1,2*nn-1,2
            cc=cos(pi*(j-1)/2.)
            psi(j)=psi(j)*cc
            psi(j+1)=psi(j+1)*cc
         end do


CCCCCC  Calculate the time evolution across                 CCCCCC
CCCCCC  the delta function
c      print *,'gg'

C  ... psi(theta=0)
```

```
         pstmpr=psi(1)
         pstmpi=psi(2)

         psi(1)=pstmpr*cos(k1*cos(-lambda*sin(i*tau)))
     *         - pstmpi*sin(k1*cos(-lambda*sin(i*tau)))
         psi(2)=pstmpr*sin(k1*cos(-lambda*sin(i*tau)))
     *         + pstmpi*cos(k1*cos(-lambda*sin(i*tau)))

C  ... psi(theta=(0..pi))

         do j=3,nn-1,2
            pstmpr=psi(j)
            pstmpi=psi(j+1)
            arg=k1*cos(twopi*(j-1)/(2.0*nn)-lambda*sin(i*tau))

            psi(j)=pstmpr*cos(arg) - pstmpi*sin(arg)
            psi(j+1)=+pstmpr*sin(arg) + pstmpi*cos(arg)

         end do

C  ... psi(theta=(-pi..0))

         do j=3,nn-1,2
            pstmpr=psi(j+nn)
            pstmpi=psi(j+nn+1)
            arg=k1*cos(twopi*(j-1-nn)/(2.0*nn)-lambda*sin(i*tau))

            psi(j+nn)=pstmpr*cos(arg) - pstmpi*sin(arg)
            psi(j+1+nn)=pstmpr*sin(arg) + pstmpi*cos(arg)

         end do

C  ... psi(theta=pi)

         pstmpr=psi(nn+1)
         pstmpi=psi(nn+2)

         psi(nn+1)=pstmpr*cos(k1*cos(pi-lambda*sin(i*tau)))
     *             - pstmpi*sin(k1*cos(pi-lambda*sin(i*tau)))
         psi(nn+2)=pstmpr*sin(k1*cos(pi-lambda*sin(i*tau)))
     *             + pstmpi*cos(k1*cos(pi-lambda*sin(i*tau)))


CCCCCC  Reintroduce momentum offset                          CCCCCC
c      print *,'hh'
```

```
         do j=1,2*nn-1,2
            cc=cos(pi*(j-1)/2.)
            psi(j)=psi(j)*cc
            psi(j+1)=psi(j+1)*cc
         end do


CCCCCC  Inverse FFT to momentum space                          CCCCCC
c      print *,'ii'

         call four1(psi,nn,-1)


CCCCCC  Renormalize psi                                        CCCCCC
c      print *,'jj'

         do j = 1,2*nn
            psi(j)=psi(j)/nn
         end do


CCCCCC  Done with calculating psi.  Now calculate the total  CCCCCC
CCCCCC  psi**2 average n, and average n**2                    CCCCCC

         if(i/printInterval*printInterval.eq.i) then
            navg=0.0
            n2avg=0.0
            sum=0.0
            psi2=0.0

            do j=1,2*nn-1,2
               psi2  = psi(j)*psi(j)+psi(j+1)*psi(j+1)
               n     = (j+1-nn)/2.0
               sum   = sum   + psi2
               navg  = navg  + psi2 * n
               n2avg = n2avg + psi2 * n**2
            end do

            write(4,*) i*tau/twopi, navg, sqrt( n2avg - navg**2),k

         end if


CCCCCC  End of time iteration                                  CCCCCC
c         print *,'ending time step',i
```

```
      end do

      finalProb=sum

CCCCCC  Write the terminal probability distribution          CCCCCC
c      print *,'ll'

      do j=1,2*nn-1,2
         psi2 = (psi(j)**2 + psi(j+1)**2)
         if (psi2 .lt. psi2minimum) then
            psi2 = psi2minimum
         end if
         n=(j+1-nn)/2.0
         write(5,*) n , psi2
      end do


CCCCCC  close output files and end                           CCCCCC
c      print *,'mm'

      close(4)
      close(5)
      close(6)

      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

B.3 Temporal Dependance of k

This subroutine allowed the potential well's depth k to be have a dependence on the elapsed time of the simulation. The implementation below was used to model our experimental conditions, which started with a linear rise in well depth over approximately one modulation period, maintained a constant value for some time, and then dropped back down to zero.

```
      SUBROUTINE kInTime(ko, k, T, Ttotal)


CCCCCC  INPUTS:                                              CCCCCC

      real*4 ko
```

```
c                          { nominal well depth                 }
      real*4 T
c                          { elapsed time in modulation cycles  }
      real*4 Ttotal
c                          { total time for the simulation in   }
c                          { modulation cycles                  }


CCCCCC  OUTPUT:                                                 CCCCCC

      real*4 k
c                          { the well depth at this iteration   }


CCCCCC  DEFINITION OF THE TEMPORAL BEHAVIOR OF K                CCCCCC

C  This function determines how k changes in time

C  In this implementation, k is initially zero.  It ramps up to
C  ko at in risetime, and during the last falltime of the
C  simulation it ramps back down to zero.

      real*4 risetime, falltime
      parameter (risetime = 1.0)
      parameter (falltime = 1.0)

      k = ko

      if (T .lt. risetime) then
         k = ko * T/risetime
      end if

      if ( (Ttotal-T) .lt. falltime) then
         k = ko * (Ttotal-T)/falltime
      end if


      end
```

BIBLIOGRAPHY

[1] L. E. Reichl, *The Transition to Chaos*, Springer-Verlag, 1988.

[2] R. Graham, M. Schlautmann, and P. Zoller, Dynamical localization of atomic-beam deflection by a modulated standing light wave, Physical Review A **45**(1), R19 (Jan. 1992).

[3] F. Moore, J. Robinson, C. Bharucha, P. Williams, and M. Raizen, Observation of Dynamical Localization in Atomic Momentum Transfer: A New Testing Ground for Quantum Chaos, Submitted to Physical Review Letters, June 1994.

[4] M. Schlautmann, On integrating Schroedinger's equation, Personal communication, Sept. 1992.

[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipies in C*, Cambridge University Press, second edition, 1981.

Vita

Following his only sibling, Jyoti, by fifteen minutes, Cyrus Bharucha was born to Farrokh and Laju Bharucha in Bristol on the thirteenth of August, 1967. Three years later, the family emigrated to the United States where the children were taught to value education in the tradition of their Indian heritage.

After graduating from Cardinal Gibbons High School in Fort Lauderdale, Cyrus earned his Bachelor's degrees in Biophysics and Electrical Engineering at Rice University. He entered the University of Texas at Austin in August 1991.

Permanent address: 7823 ~~Sandspoint~~ **bharucha.com**
Houston, Texas ~~77036~~

This thesis was typeset[1] with LaTeX by the author.

[1] LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for The University of Texas at Austin thesis format was written by Khe-Sing The and modified by Walter F. Buell and Martin C. Fischer.