PHYS 410: Computational Physics Fall 2024
Project 2—The Time Dependent Schrödinger Equation
Due: Monday, December 2, 11:59 PM
PLEASE report all bugs, comments, gripes etc. to Matt: `choptuik@physics.ubc.ca`

1. Problem 1

1.1 Introduction

In this part of the project you will solve the one-dimensional time-dependent Schrödinger equation using the method discussed in the lecture, which will be summarized here.

The continuum equation is (after non-dimensionalization)

$$i\psi(x,t)_t = -\psi_{xx} + V(x,t)\psi\,, \tag{1}$$

where the wavefunction, $\psi(x,t)$, is complex. The equation is to be solved on the domain

$$0 \le x \le 1\,, \qquad 0 \le t \le t_{\max},$$

subject to initial and boundary conditions

$$\psi(x,0) = \psi_0(x)\,, \tag{2}$$

$$\psi(0,t) = \psi(1,t) = 0\,. \tag{3}$$

The initial data function $\psi_0(x)$ can be complex in general, but in the cases we will consider will be real. We will also restrict attention to time-independent potentials, but carry the explicit dependence in the following to emphasize that, in principle, it is no more difficult to deal with a time-dependent potential than a time-independent one.

We note that ψ can be expressed in terms of its real and imaginary parts, $\psi_{\rm Re}$ and $\psi_{\rm Im}$, respectively as

$$\psi = \psi_{\rm Re} + i\psi_{\rm Im}$$

A useful diagnostic quantity is the "running integral", $P(x,t)$, of the probability density, $\rho = |\psi|^2 = \psi\psi^*$:

$$P(x,t) = \int_0^x \psi(\tilde{x},t)\psi^\star(\tilde{x},t)d\tilde{x} \tag{4}$$

If the wavefunction is properly normalized, then we will have

$$P(1,t) = 1$$

but even if it is not so normalized (and in our applications there will be no need to ensure normalization), we should have

$$P(1,t) = \text{conserved to level of solution error}$$

The quantity $\sqrt{\rho} = |\psi|$ is also a useful diagnostic. However, as you develop your code to solve the Schrödinger equation you should be prepared to plot the real and imaginary parts of ψ as well.

A family of exact solutions of (1) is

$$\psi(x,t) = e^{-im^2\pi^2 t}\sin(m\pi x) \tag{5}$$

where m is a positive integer.

1.2 Discretization

We discretize the continuum domain by introducing the discretization level, l, and the ratio of temporal to spatial mesh spacings

$$\lambda = \frac{\Delta t}{\Delta x}$$

Then

$$
\begin{aligned}
n_x &= 2^l + 1 \\
\Delta x &= 2^{-l} \\
\Delta t &= \lambda \Delta x \\
n_t &= \text{round}\,(t_{\max}/\Delta t) + 1
\end{aligned}
$$

We apply the Crank-Nicolson discretization approach to (1)–(3) yielding

$$
i\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = -\frac{1}{2}\left(\frac{\psi_{j+1}^{n+1} - 2\psi_j^{n+1} + \psi_{j-1}^{n+1}}{\Delta x^2} + \frac{\psi_{j+1}^{n} - 2\psi_j^{n} + \psi_{j-1}^{n}}{\Delta x^2}\right) + \frac{1}{2}V_j^{n+\frac{1}{2}}\left(\psi_j^{n+1} + \psi_j^n\right) \quad j = 2, 3, \ldots, n_x - 1\,, \quad n = 1, 2, \ldots n_t - 1 \tag{6}
$$

$$
\psi_1^{n+1} = \psi_{n_x}^{n+1} = 0\,, \quad n = 1, 2, \ldots n_t - 1 \tag{7}
$$

$$
\psi_j^1 = \psi_0(x_j)\,, \quad j = 1, 2, \ldots n_x \tag{8}
$$

1.3 Implementation

Implement your solution of (6)–(8) as a `MATLAB` function with the following header and arguments:

```
function [x t psi psire psiim psimod prob v] = ...
   sch_1d_cn(tmax, level, lambda, idtype, idpar, vtype, vpar)

%  Inputs
%
%     tmax:    Maximum integration time
%     level:   Discretization level
%     lambda:  dt/dx
%     idtype:  Selects initial data type
%     idpar:   Vector of initial data parameters
%     vtype:   Selects potential type
%     vpar:    Vector of potential parameters
%
%  Outputs
%
%     x:       Vector of x coordinates [nx]
%     t:       Vector of t coordinates [nt]
%     psi:     Array of computed psi values [nt x nx]
%     psire    Array of computed psi_re values [nt x nx]
%     psiim    Array of computed psi_im values [nt x nx]
%     psimod   Array of computed sqrt(psi psi*) values [nt x nx]
%     prob     Array of computed running integral values [nt x nx]
%     v        Array of potential values [nx]
```

The input parameters `idtype` and `vtype` are integers that select which initial data type and potential type, respectively, are to be used. Dependent on the type, elements of the associated parameter vector will be used to define the initial data or potential. Specifically, you are to implement options as follows:

Initial data types

- `idtype == 0`: Exact family (5)

$$
\psi(x, 0) = \sin(m\pi x)
$$

  - `idpar(1)`: m

- `idtype == 1`: Boosted Gaussian:

$$\psi(x,0) = e^{ipx} e^{-((x-x_0)/\delta)^2}$$

  - `idpar(1)`: x_0
  - `idpar(2)`: δ
  - `idpar(3)`: p

Potential types

- `vtype == 0`: No potential.

$$V(x) = 0$$

- `vtype == 1`: rectangular barrier or well.

$$V(x) = \begin{cases} 0 & \text{for} \quad x < x_{\min} \\ V_c & \text{for} \quad x_{\min} \le x \le x_{\max} \\ 0 & \text{for} \quad x > x_{\max} \end{cases}$$

  - `vpar(1)`: $x_{\min}$
  - `vpar(2)`: $x_{\max}$
  - `vpar(3)`: V_c

Notes

- The terminology "boosted Gaussian" comes from the fact that by pre-multiplying the (real-valued) Gaussian profile by e^{ipx}, we give the wave packet some momentum in the direction of p (which can have either sign).
- The constant V_c is positive for a barrier, negative for a well.
- Don't worry about the fact that, strictly speaking, the boosted Gaussian profile is incompatible with the boundary conditions. In practice we will try to center the Gaussian sufficiently far from the boundaries that the incompatibility is lost in the truncation error. You should always set the wave function to 0 at $x = 0$ and $x = 1$, including at the initial time.

Coding

- Write equations (6)–(7) as a complex (as in complex number) tridiagonal system for the advanced unknowns ψ_j^{n+1}. In your code, set up the tridiagonal system using `spdiags` as discussed in class and as illustrated in the code `diff_1d_imp.m` from Tutorial 6. and solve it using left division. Note that `MATLAB` has native support for complex numbers (the variables `i` and `j` are both initialized to the unit pure imaginary number, i) and that you should implement your solution directly using complex arithmetic. As observed above, there is no need to worry about normalization of the wave function.
- Once you've set up the tridiagonal matrix, say `A`, using `spdiags`, you can view the corresponding full matrix using the `MATLAB` command `full(A)`. This can be useful while debugging.
- Note that the `MATLAB abs(z)` command when applied to a complex number, z, returns its modulus, $|z|$.
- The integral in (4) can be computed to $O(h^2)$ using the trapezoidal formula. Recall that if we are given n approximate values f_i at values of x, x_i, then the trapezoidal approximation is given by

$$\int_{x_1}^{x_n} f(x)dx \approx \frac{1}{2} \sum_{i=1}^{n-1} (f_i + f_{i+1}) (x_{i+1} - x_i)$$

1.4 Convergence testing

Define a level l solution computed using `sch_1d_cn` by ψ^l. Note that ψ^l is a function of both the discrete time and space coordinates. Denote by $d\psi^l$ the quantity defined by

$$d\psi^l = \psi^{l+1} - \psi^l$$

where it is to be understood that the data defined by the grid function (array) ψ^{l+1} is 2:1 coarsened in both the time and space dimensions so that it has the same size as ψ^l. Then one way we can convergence test `sch_1d_cn` is to compute

$$\|d\psi^l\|_2(t^n) \tag{9}$$

where

$$\|\cdot\|_2$$

denotes the l-2 norm (RMS value) that has been defined before; namely, for any length-m vector v

$$\|v\|_2 = \sqrt{\frac{\sum_{j=1}^{m} |v^j|^2}{m}}$$

Observe that for complex numbers, $|v^j|$ is the modulus of the number. Note that (9) involves taking spatial norms of the pairwise subtraction of grid functions at two different levels. This results in a function of the discrete time, t^n, on the level-l grid.

Following the development we have seen for solutions of other finite difference equations, we note that since our FDA is $O(h^2)$ (where $\Delta x = h$ and $\Delta t = \lambda h$), we expect the solution to be $O(h^2)$ accurate, with ψ^l admitting an expansion of the form

$$\psi^l(x,t) = \psi(x,t) + h_l^2 e_2(x,t) + O(h_l^4) \tag{10}$$

where $e_2(x,t)$ is some error function. From (10) we can deduce that if we graph rescaled values of $\|d\psi^l\|_2$ on a single plot, then convergence is signalled by near-coincidence of the curves, with better agreement as we go to higher values of l. In particular, for a test with levels

$$l = l_{\min}, l_{\min}+1, \ldots, l_{\max}$$

we should plot

$$\|d\psi^{l_{\min}}\|_2\,,\ 4\|d\psi^{l_{\min}+1}\|_2\,,\ 4^2\|d\psi^{l_{\min}+2}\|_2\,,\ \ldots\,,\ 4^{l_{\max}-l_{\min}-1}\|d\psi^{l_{\max}-1}\|_2$$

Additionally, for the case when `idtype = 0`, so that we know the exact solution, we can compute the actual solution errors. Specifically, we can perform precisely the same type of convergence test just described, but where ψ^{l+1} is replaced with ψ_{exact}. Thus we define

$$\|E(\psi^l)\|_2(t^n) = \|\psi_{\text{exact}} - \psi^l\|_2(t^n)$$

and use exactly the same plotting strategy for $\|E(\psi^l)\|_2$ as we do for $\|d\psi^l\|_2$.

Convergence tests to perform

Ensure that at least one of the following convergence tests, including the plotting, can be performed by executing a script `ctest_1d`.

1. `idtype = 0`, `vtype = 0`

   Other parameters

   - `idpar = [3]`
   - `tmax = 0.25`
   - `lambda = 0.1`
   - $l_{\min} =$ `6`
   - $l_{\max} =$ `9`

   Perform the 4-level test and make convergence plots as described above for both $\|d\psi^l\|_2$ and $\|E(\psi^l)\|_2$. Include the plots in your report.

2. `idtype = 1`, `vtype = 0`

   Other parameters

   - `idpar = [0.50 0.075 0.0]`
   - `tmax = 0.01`

- `lambda = 0.01`
- $l_{\min}$= 6
- $l_{\max}$= 9

Perform the 4-level test and make a convergence plot as described above for $\|d\psi^l\|_2$. Include the plot in your report.

1.5 Numerical Experiments

Consider the discrete running integral of the probability density:

$$P_j^n = P(x_j, t^n)\,, \quad j = 1, 2, \ldots n_x\,, \quad n = 1, 2, \ldots n_t$$

Define the temporal average, $\bar{P}_j$, of the above quantity:

$$\bar{P}_j = \frac{\sum_{n=1}^{n_t} P_j^n}{n_t}$$

and note that the `MATLAB` command `mean` can be used to compute this. We will also want to ensure that $\bar{P}_j$ is properly normalized so that $\bar{P}_{n_x} = 1$. We can do this as follows:

$$\bar{P}_j := \bar{P}_j / \bar{P}_{n_x}\,,\ j = 1, 2, \ldots n_x$$

In the following we will assume that $\bar{P}_j$ *has* been properly normalized. Given two values of x, x_1 and x_2, satisfying $x_2 > x_1$, we can interpret the quantity

$$\bar{P}(x_2) - \bar{P}(x_1)$$

as the *fraction* of time our quantum particle spends in the interval $x_1 \le x \le x_2$. Here the notation $\bar{P}(x)$ is to be interpreted as $\bar{P}(x) = \bar{P}(x_j) = \bar{P}_j$ where x_j is the nearest grid point to x. Now, for a free particle—i.e. for $V = 0$—and for sufficiently long times, we expect that

$$\bar{P}(x_2) - \bar{P}(x_1) \to x_2 - x_1$$

That is, the fraction of time the particle spends in the interval is given simply by the width of the interval (this direct equality is due to the fact that we are solving the Schrödinger equation on the unit interval, $0 \le x \le 1$).

For the general case of a non-zero potential, we can then define the excess fractional probability that the particle spends in a given spatial interval as

$$\bar{F}_e(x_1, x_2) = \frac{\bar{P}(x_2) - \bar{P}(x_1)}{x_2 - x_1}$$

For the experiments described below, this quantity will span orders of magnitude so, particularly for the purposes of plotting, it will be convenient to compute its (natural) logarithm (recall that `MATLAB` uses `log` for the natural log).

$$\ln \bar{F}_e(x_1, x_2) = \ln \frac{\bar{P}(x_2) - \bar{P}(x_1)}{x_2 - x_1}$$

Also observe that although we have called $\bar{F}_e(x_1, x_2)$ the *excess* fractional probability, it can in fact satisfy $\bar{F}_e(x_1, x_2) < 1$, indicating that the particle is spending *less* time in the specified interval than a free particle would. Indeed, in the experiments that follow, you should find that $\bar{F}_e(x_1, x_2) < 1$ is the rule rather than the exception.

1.5.1 Experiment 1: Barrier Survey

In this investigation the particle will start to the left of a barrier whose height, V_0, is the control parameter for the experiment. You will then determine the dependence of $\ln(\bar{F}_e(x_1, x_2))$ on $\ln(V_0)$ where x_1 and x_2 span the region to the right of the barrier.

Parameters

- `tmax = 0.10`
- `level = 9`
- `lambda = 0.01`
- `idtype = 1` (boosted Gaussian)
- `idpar = [0.40, 0.075, 20.0]`
- `vtype = 1` (rectangular barrier)
- `vpar = [0.6, 0.8, VARIABLE > 0]`
- $x_1 = 0.8$
- $x_2 = 1.0$

Write a script called `barrier_survey` that uses `sch_1d_cn` to compute $\bar{F}_e(0.8, 1.0)$ for 251 uniformly spaced values of $\ln(V_0)$ ranging from -2 to 5. The script is to make a plot of $\ln(\bar{F}_e(0.8, 1.0))$ versus $\ln(V_0)$. Include the plot in your writeup and comment on what you can deduce from it. Since it will take some time for the 251 runs to complete, it only makes sense to debug your script using fewer values of $\ln(V_0)$.

1.5.2 Experiment 2: Well Survey

The second experiment is very much like the first, except that we now consider scattering of a particle off a potential *well*. Once again you will perform a survey to investigate the dependence of $\ln(\bar{F}_e(x_1, x_2))$ on $\ln(V_0)$ where x_1 and x_2 span the location of the well.

Parameters

- `tmax = 0.10`
- `level = 9`
- `lambda = 0.01`
- `idtype = 1` (boosted Gaussian, but note that $p = 0$)
- `idpar = [0.40, 0.075, 0.0]`
- `vtype = 1` (rectangular well)
- `vpar = [0.6, 0.8, VARIABLE < 0]`
- $x_1 = 0.6$
- $x_2 = 0.8$

Write a script called `well_survey` that uses `sch_1d_cn` to compute $\bar{F}_e(0.6, 0.8)$ for 251 uniformly spaced values of $\ln(|V_0|)$ ranging from 2 to 10. Note that V_0 is strictly less than 0. The script is to make a plot of $\ln(\bar{F}_e(0.6, 0.8))$ versus $\ln(|V_0|)$. Include the plot in your writeup and discuss what you can conclude from it.

2. Problem 2

2.1 Introduction

In this problem you will solve the two-dimensional Schrödinger equation using the ADI technique discussed in class for the case of the diffusion equation. *Indeed, as mentioned in lecture, and should you have time, I suggest that you consider first solving the 2d diffusion equation using ADI which will give you a good base from which to tackle the Schrödinger equation.*

The non-dimensionalized continuum equation is

$$i\psi(x,y,t)_t = -(\psi_{xx} + \psi_{yy}) + V(x,y)\psi$$

or, multiplying through by $-i$

$$\psi_t = i(\psi_{xx} + \psi_{yy}) - iV(x,y)\psi \tag{11}$$

In this last form, the similarity to a diffusion equation with an imaginary diffusion constant (and a source term) is apparent. Equation (11) is to be solved on the domain

$$0 \le x \le 1\,, \qquad 0 \le y \le 1\,, \qquad 0 \le t \le t_{\max}$$

subject to initial and boundary conditions

$$\psi(x,y,0) = \psi_0(x,y) \tag{12}$$

$$\psi(0,y,t) = \psi(1,y,t) = \psi(x,0,t) = \psi(x,1,t) = 0 \tag{13}$$

A family of exact solutions is given by

$$\psi(x,y,t) = e^{-i(m_x^2+m_y^2)\pi^2 t}\sin(m_x\pi x)\sin(m_y\pi y) \tag{14}$$

where m_x and m_y are positive integers.

2.2 Discretization and the ADI scheme

As for the 1d case, the continuum domain is discretized by introducing the discretization level, l, and the ratio of temporal to spatial mesh spacings

$$\lambda = \frac{\Delta t}{\Delta x} = \frac{\Delta t}{\Delta y}$$

Then

$$\begin{aligned} n_x &= n_y = 2^l + 1 \\ \Delta x &= \Delta y = 2^{-l} \\ \Delta t &= \lambda \Delta x \\ n_t &= \text{round}\,(t_{\max}/\Delta t) + 1 \end{aligned}$$

Defining the difference operators ∂^h_{xx} and ∂^h_{yy} by

$$\partial^h_{xx} u^n_{i,j} \equiv \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2}$$

$$\partial^h_{yy} u^n_{i,j} \equiv \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2}$$

the following is an ADI discretization of (11):

$$\left(1 - i\frac{\Delta t}{2}\partial^h_{xx}\right)\psi^{n+\frac{1}{2}}_{i,j} = \left(1 + i\frac{\Delta t}{2}\partial^h_{xx}\right)\left(1 + i\frac{\Delta t}{2}\partial^h_{yy} - i\frac{\Delta t}{2}V_{i,j}\right)\psi^n_{i,j}\,,$$
$$i = 2,3,\ldots,n_x - 1\,, \quad j = 2,3,\ldots,n_y - 1\,, \quad n = 1,2,\ldots,n_t - 1 \tag{15}$$

$$\left(1 - i\frac{\Delta t}{2}\partial^h_{yy} + i\frac{\Delta t}{2}V_{i,j}\right)\psi^{n+1}_{i,j} = \psi^{n+\frac{1}{2}}_{i,j}\,,$$
$$i = 2, 3, \ldots, n_x - 1\,, \quad j = 2, 3, \ldots, n_y - 1\,, \quad n = 1, 2, \ldots, n_t - 1 \tag{16}$$

Equations (15) and (16) are to be supplemented with the initial conditions

$$\psi^1_{i,j} = \psi_0(x_i, y_j)\,, \tag{17}$$

and the boundary conditions

$$\psi^n_{1,j} = \psi^n_{n_x,j} = \psi^n_{i,1} = \psi^n_{i,n_y} = 0\,. \tag{18}$$

2.3 Implementation

Implement your solution of (15)–(18) as a `MATLAB` function, `sch_2d_adi` with the following header and arguments:

```
function [x y t psi psire psiim psimod v] = ...
         sch_2d_adi(tmax, level, lambda, idtype, idpar, vtype, vpar)
%  Inputs
%
%     tmax:    Maximum integration time
%     level:   Discretization level
%     lambda:  dt/dx
%     idtype:  Selects initial data type
%     idpar:   Vector of initial data parameters
%     vtype:   Selects potential type
%     vpar:    Vector of potential parameters
%
%  Outputs
%
%     x:       Vector of x coordinates [nx]
%     y:       Vector of y coordinates [ny]
%     t:       Vector of t coordinates [nt]
%     psi:     Array of computed psi values [nt x nx x ny]
%     psire    Array of computed psi_re values [nt x nx x ny]
%     psiim    Array of computed psi_im values [nt x nx x ny]
%     psimod   Array of computed sqrt(psi psi*) values [nt x nx x ny]
%     v        Array of potential values [nx x ny]
```

As before, the integer arguments `idtype` and `vtype` encode the initial data and potential types, respectively, with `idpar` and `vpar` defining the parameters for the various options. In this case you are to implement options as follows:

Initial data types

- `idtype == 0`: Exact family (14)
$$\psi(x, y, 0) = \sin(m_x \pi x)\sin(m_y \pi y)$$
  - `idpar(1)`: m_x
  - `idpar(2)`: m_y
- `idtype == 1`: Boosted Gaussian
$$\psi(x, y, 0) = e^{ip_x x}e^{ip_y y}e^{-((x-x_0)^2/\delta_x^2 + (y-y_0)^2/\delta_y^2)}$$
  - `idpar(1)`: x_0
  - `idpar(2)`: y_0
  - `idpar(3)`: δ_x
  - `idpar(4)`: δ_y

- `idpar(5)`: p_x
- `idpar(6)`: p_y

As previously, don't worry about the fact that the Gaussian data is strictly speaking incompatible with the boundary conditions; just be sure to always impose the correct boundary conditions.

Potential types

- `vtype == 0`: No potential.

$$V(x,y) = 0$$

- `vtype == 1`: Rectangular barrier or well.

$$V(x,y) = \begin{cases} V_c & \text{for} \\ 0 & \text{otherwise} \end{cases} \quad (x_{\min} \le x \le x_{\max}) \text{ and } (y_{\min} \le y \le y_{\max})$$

  - `vpar(1)`: $x_{\min}$
  - `vpar(2)`: $x_{\max}$
  - `vpar(3)`: $y_{\min}$
  - `vpar(4)`: $y_{\max}$
  - `vpar(5)`: V_c

- `vtype == 2`: Double slit. Let $j' = (n_y - 1)/4 + 1$. Then

$$V_{i,j'} = V_{i,j'+1} = 0 \text{ for } [(x_1 \le x_i) \text{ and } (x_i \le x_2)] \text{ or } [(x_3 \le x_i) \text{ and } (x_i \le x_4)]$$

$$V_{i,j'} = V_{i,j'+1} = V_c \text{ otherwise}$$

$$V_{i,j} = 0 \text{ for } j \ne (j' \text{ or } j'+1)$$

That is, V is only non-zero for y-locations given by $j = j'$ or $j = j' + 1$ and for x-positions not coincident with one of the slits. This thus simulates a thin (two mesh points wide) plate at a fixed y-position, with adjustable slit openings, which span (x_1, x_2) and (x_3, x_4).

  - `vpar(1)`: x_1
  - `vpar(2)`: x_2
  - `vpar(3)`: x_3
  - `vpar(4)`: x_4
  - `vpar(5)`: V_c

Coding

- **IMPORTANT!!** Beware that the transpose operator `'` computes the *conjugate* transpose (i.e. it will complex-conjugate any complex numbers it encounters), so probably will not be what you want. Use the non-conjugating operator `.'` instead.

2.4 Convergence testing

Convergence test your code in the same manner that you did for the 1d case, taking into account the extra dimension. In particular, when performing a convergence test, compute the two-level deviation norms

$$\|d\psi^l\|_2(t^n) \tag{19}$$

as well as the deviations from the exact solution (when using initial data corresponding to an exact solution)

$$\|E(\psi^l)\|_2(t^n) = \|\psi_{\text{exact}} - \psi^l\|_2(t^n)$$

Note that for the 2d case, the spatial norm $\|\cdot\|_2$ involves a sum over both spatial dimensions, i.e. treat any 2d grid function as a vector of length $n_x \times n_y$.

Convergence test to perform

- `idtype = 0, vtype = 0`
  Other parameters
  - `idpar = [2, 3]`
  - `tmax = 0.05`
  - `lambda = 0.05`
  - $l_{\min} = 6$
  - $l_{\max} = 9$

Write a script called `ctest_2d` that performs the 4-level test and makes convergence plots as described above for both $\|d\psi^l\|_2$ and $\|E(\psi^l)\|_2$. Include the plots in your report.

2.5 Numerical experiments

In the true spirit of projects, you'll be mostly on your own here. At a minimum, make AVI movies of the following scenarios and include them in your submission. In all cases you should use Gaussian initial data, with or without a boost (in either and/or both directions) as you find convenient.

1. Scattering off a rectangular barrier (`vtype = 1`).
2. Scattering off a rectangular well (`vtype = 1`).
3. Scattering through a double slit (`vtype = 2`). Try to simulate discernible particle self-interference on the far side of the slit.

I recommend that you consider using filled contour plots generated with the `MATLAB contourf` command as the basis for making your movies. A useful attribute for `contourf` is the colormap which can be manipulated using the `colormap` command.

Final caution

Note that because your routine will be storing the entire time evolution of the solution (as well as several additional functions), memory requirements can get extreme for long run times. `MATLAB` will complain if you try to allocate an array that is "too large" but you may find your machine gets very sluggish before that point, so beware. At level 9 and below you shouldn't have too many difficulties, again provided that you don't try integrating to very long times. Report any undue difficulties to me as usual.