Excursions in Computing Science:
Book 11d. Forces and Invariants
Part II. Partial Slope Equations and Quantum Mechanics

T. H. Merrett*
McGill University, Montreal, Canada

August 1, 2021

Part I. Electrostatics and Electromagnetism

1. Central Forces.

2. Gravity vs. Electricity.

3. Energy and momentum scales.

4. Divergence, gradient and $\vec{\text{div}}$ $\vec{\text{grad}}$.

5. Electrodynamics departs from gravitation.

6. Invariants, cross-products and convention.

7. Electromagnetic waves.

I. Prefatory Notes

8. Partial Slope Equations: Laplace's Equation. In Part I, notably Notes 5 and 7, we expressed electromagnetism using the slope operators $\bigtriangledown^2 = \partial_x^2 + \partial_y^2 + \partial_z^2$ and ∂_t^2. These are *partial slope* operators, and the equations using them to describe electromagnetism are *partial slope equations* (conventionally called partial differential equations, or PDEs).

We must learn how to solve them, at least far enough that we can visualize the resulting fields.

We'll do this numerically, and only as precisely as is needed for visualization. Analytical solutions are possible in the simple cases we'll discuss, but numerical solutions give the pictures and animations we'll need to support our intuitions. And numerical solutions permit us in practice to go beyond the limited number of analytical solutions known. However, we'll see that we need to know these solutions in advance in order to set the "boundary conditions" for the numerical calculations,

In the next five Notes we'll work from simple examples to more complex. We'll discover that each example needs its own solution techniques.

We'll limit ourselves to two dimensions so that we can draw the pictures.

We start with Laplace's equation

$$\bigtriangledown^2 \phi = 0$$

*Copyleft ©T. H. Merrett, 2017, 2019, 2021. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation in a prominent place. Copyright for components of this work owned by others than T. H. Merrett must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to republish from: T. H. Merrett, School of Computer Science, McGill University, fax 514 398 3883.

which describes not only the central-force field potential (gravity, electrostatics) without sources or sinks, but also the temperature of a solid in thermal equilibrium and a number of other, apparently unrelated, physical and engineering phenomena.

The numerical expression of $\bigtriangledown^2\phi = \partial_x^2\phi + \partial_y^2\phi$ uses

$$\begin{aligned}
\partial_x^2\phi &= \partial_x\partial_x\phi \\
&\approx \partial_x \frac{\phi_{+\circ} - \phi_{\circ\circ}}{\Delta x} \\
&\approx \frac{1}{\Delta x}\left(\frac{\phi_{+\circ} - \phi_{\circ\circ}}{\Delta x} - \frac{\phi_{\circ\circ} - \phi_{-\circ}}{\Delta x}\right) \\
&= \frac{1}{(\Delta x)^2}(\phi_{+\circ} + \phi_{-\circ} - 2\phi_{\circ\circ})
\end{aligned}$$

and similarly

$$\partial_y^2\phi \approx \frac{1}{(\Delta y)^2}(\phi_{\circ+} + \phi_{\circ-} - 2\phi_{\circ\circ})$$

where the shorthand notation $\phi_{+\circ}$, $\phi_{-\circ}$, etc., mean

$$\begin{array}{lll}
\phi_{+\circ} = \phi(x+\Delta x, y) & \phi_{\circ\circ} = \phi(x,y) & \phi_{-\circ} = \phi(x-\Delta x, y) \\
\phi_{\circ+} = \phi(x, y+\Delta y) & & \phi_{\circ-} = \phi(x, y-\Delta y)
\end{array}$$

These combine to give, if we set $\Delta x = h = \Delta y$,

$$\bigtriangledown^2\phi = \frac{1}{h}(\phi_{+\circ} + \phi_{\circ+} + \phi_{-\circ} + \phi_{\circ-} - 4\phi_{\circ\circ})$$

The way to solve $\bigtriangledown^2\phi = 0$ can now be written as a matrix equation. Of course, we'll have to supply some previously-known values for ϕ. We'll start by supposing that the boundary is known. This would be the case if Laplace's equation were describing the temperature of a slab of material wth known temperatures on its boundary.

Let's try a 5-by-5 grid with known temperatures a_i, b_i, c_i as shown.

	1	2	3	4	5
1		b_1	a_1	c_1	
2	c_2				b_4
3	a_2				a_4
4	b_2				c_4
5		c_3	a_3	b_3	

Then $\bigtriangledown^2\phi = 0$ becomes the matrix equation

$$\left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)
\left(\begin{array}{c}
\phi_{22} \\ \phi_{23} \\ \phi_{24} \\ \phi_{32} \\ \phi_{33} \\ \phi_{34} \\ \phi_{42} \\ \phi_{43} \\ \phi_{44}
\end{array}\right)
=
\left(\begin{array}{c}
b_1 + c_2 \\ a_1 \\ c_1 + b_4 \\ a_2 \\ 0 \\ a_4 \\ b_2 + c_3 \\ a_3 \\ b_3 + c_4
\end{array}\right)$$

The matrix is 9-by-9 because there are 9 unknown values of ϕ, in the middle of the slab (we don't need to know the four corners to find these nine, so the corners are left blank).

We see the 4s on the diagonal, corresponding to the $4\phi_{\circ\circ}$ and up to four -1s in each row. Where a -1 is missing in the matrix, this is because the corresponding ϕ-value is known, and appears in the vector on the far right.

In the case of circular (spatial) symmetry, which holds for a gravitational field with a point source (not included in the equation $\bigtriangledown^2\phi = 0$) the a_i are all equal and the $b_i = c_i$ are all equal.

The above boundary conditions are not sufficient to capture a central-force potential field. This field, which is $-1/r$ in 3D and $\ln r$ in 2D, goes to $-\infty$ at the centre, where $r = 0$. We must tell the calculation about this, say by specifying also four values of the field at the closest points to the centre.

For a 5-by-5 grid the boundary conditions become (now using spherical symmetry)

	1	2	3	4	5
1		b	a	b	
2	b		c		b
3	a	c		c	a
4	b		c		b
5		b	a	b	

The matrix equation for the four values that are now unknown is

$$\begin{pmatrix} 4 & & & \\ & 4 & & \\ & & 4 & \\ & & & 4 \end{pmatrix} \begin{pmatrix} \phi_{22} \\ \phi_{24} \\ \phi_{444} \\ \phi_{22} \end{pmatrix} = \begin{pmatrix} 2b+2c \\ 2b+2c \\ 2b+2c \\ 2b+2c \end{pmatrix}$$

Of course this is very uninteresting in this toy example, but it becomes more interesting for a larger grid: under spherical symmetry there will be n possibly different known values around the outer border for a $2n+1$-by-$2n+1$ grid, with one further known value repeated four times around the centre, as c is above. The matrix of unknowns, without taking symmetry into account, will be $(2n-1)^2 - 5$ by $(2n-1)^2 - 5$.

Such matrices become very large for any reasonably-sized grid. They are also very sparse, so it is possible to solve the matrix equations iteratively rather than by brute force.

The method we'll use is Jacobi's. He observed, in the matrix equation for unknown u

$$Au = b$$

that a sparse A can be broken down into diagonal and non-diagonal components

$$A = D + D'$$

(where we expect the diagonal elements to be the largest as in the above examples).

So the equation to be solved becomes

$$Du = b - D'u$$

or

$$u = D^{-1}b - D^{-1}D'u$$

and this suggests a step between iterations j and $j+1$

$$u_{j+1} = D^{-1}(b - D'u_j)$$

Putting all this together for a $\ln r$ potential we need a suite of programs.

First it is handy to be able to generate the different values of r^2 for the boundary of the grid.

```
bdyvals = centralSym(5)
```
giving
```
bdyvals = [5,4,5]
```

	5	4	5	

Next we distribute the natural logarithm of r around the boundary

```
bdypro = symmBdyGen(log(bdyvals)/2)
```

1	2	b
1	3	a
1	4	b
4	1	b
3	1	a
2	1	b
5	4	b
5	3	a
5	2	b
2	5	b
3	5	a
4	5	b

	b	a	b	
b				b
a				a
b				b
	b	a	b	

a = 0.6931
b = 0.8047

Note that the output of `symmBdyGen()` is a *protor* (Book 11c, Note 4). This is the suitable form for sparse matrix calculations.

In order to capture the inner "boundary" values we can also use `symmBdyGen()` and then adjust.

```
bdyproC = symmBdyGen(1)
```

gives

bdyproC		
1	2	1
2	1	1
3	2	1
2	3	1

adjusted to 17-by-17

8	9	1
9	8	1
9	10	1
10	9	1

	8	9	10	
8		1		
9	1		1	
10		1		

And I've shown the adjustment to place these values at the centre of a 17-by-17 grid.

Of course, we'd be taking the log of the square root of that value, 1, which of course in 0. So now we should consider that the step size is not 1 but some very small number such as 1/400. The central part of the calculation becomes

```
bdyproC = symmBdyGen(log(1/400)/2)
```

giving, again for a 17-by-17 grid, with $a = -\ln 400 = -5.9915$

8	9	a
9	8	a
9	10	a
10	9	a

The final boundaries will just be the concatenation of the outer and inner result protors

```
bdypro = [bdypro;bdyproC]
```

Third we generate the matrix A and the vector b, also in protor form, which specify the desired grid and boundary values.

```
[mat,vec] = pdeMatVec(neighweigh,bdypro)
```

We discuss the inputs first. We need a protor to capture the numerical form of the slope operator. For

$$\bigtriangledown^2\phi = \frac{1}{h}(\pi_{+\circ} + \phi_{\circ+} + \phi_{-\circ} + \phi_{\circ-} - 4\phi_{\circ\circ})$$

this is

neighweigh		
0	0	4
1	0	−1
−1	0	−1
0	1	−1
0	−1	−1

And we need `bdypro` which, for this example, I'll take to be the 5-by-5 result of

```
bdypro = symBdyGen([5,4,5])
```

The results we want from the invocation of `pdeMatVec` are

- `mat` will be the protor version of the 9-by-9 matrix at the beginning of this Note

mat				
2	2	2	2	4
2	2	3	2	−1
:				
4	4	3	4	−1
4	4	4	3	−1

  where each index is the pair of integers labelling the corresponding cell in the grid.

- `vec` will be the 9-by-1 vector also given in that matrix equation for $\bigtriangledown^2\phi = 0$, also in protor form

vec		
2	2	10
2	3	4
2	4	10
3	2	4
3	4	4
4	2	10
4	3	4
4	4	10

  where the indices are again pairs, and we note that the 3 3 0 entry is omitted because of the 0.

What `pdeMatVec()` does is loop through all interior points of the grid (indices j, k), within these loops add each index pair of `neighweigh` to j and k, respectively (giving `j1, k1`) and look up `ji,k1` in `bdypro`. If it is there,and hence on the boundary, add the coefficient from `bdypro` to what (if anything) is already in `vec`; otherwise create a new row for `mat` of `j, k, j1, k1` and the coefficient.

Thus `pdeMatVec()` distributes the coefficents of `neighweigh` into matrix `mat` or vector `vec` (suitably weighted by `bdypro` in this case) depending on whether the point is interior or on the boundary.

Now that we have the matrix `mat` we must extract the diagonal and non-diagonal parts, and find the inverse of the diagonal.

```
[D,Dm1,LU] = undiagMat(mat)
```

where D is the diagonal, `Dm1` is its inverse, and `LU` (lower-upper triangles) is the non-diagonal part; all these are in the same paired-index protor form. For example

D					Dm1					LU				
2	2	2	2	4	2	2	2	2	0.25	2	2	3	2	−1
2	3	2	3	4	2	3	2	3	0.25	2	2	2	3	−1
:					:					:				

Finally we use protor routines (Book 11c, Notes 4 and 20) to iterate the Jacobi steps

$$\begin{aligned} u_1 &= D^{-1}b \\ u_{j-1} &= D^{-1}(b - D'u_j) \end{aligned}$$

where D^{-1} is what we called `Dm1` in the MATLAB programs, b is `vec` and D' is `LU`.

The matrix multiplication uses a paired-index version of `joinred()` (Book 11c, Note 4), e.g.,

```
U = joinred2(Dm1,[3,4],vec,[1,2])
```

The matrix subtraction uses `mergesum()` (Book 11c, Note 20) with numerical weights 1 and −1 in the first and third parameters respectively.

These are all combined into

```
result = pdeJacobi(Dm1,LU,vec)
```

To stop the iteration we need to find the relative difference between two successive steps. For this we write a new function

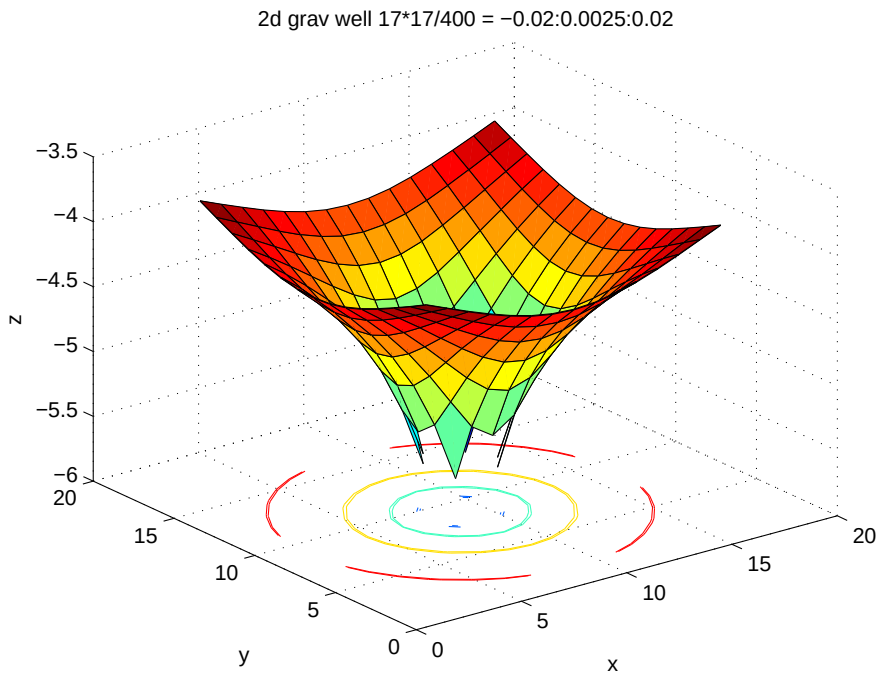
```
mergeProd(wt1,protor1,wt2,protor2)
```

whichis identical to `mergeSum()` except it finds the products of corresponding coefficients, in the form

$$< \texttt{protor1} >^{<\texttt{wt1}>} \times < \texttt{protor2} >^{<\texttt{wt2}>}$$

The code in `pdeJacobi()` stops the iteration when the minimum of these relation differences exceeds 0.0001, which gives enough accuracy for plotting.

Here is a result from a 17-by-17 grid with step size $h = 1/400$



9. The Wave Equation. In Note 7 of Part I we introduced the wave equation. Now it is time to "solve" it, at least to see how it describes waves.

You would think that we could just add a time dimension and use the same techniques as for the Laplace equation in Note 8, but this is not so. There will be boundary conditions, this time around the outer boundary in x-y space, but there will also be *initial* conditions for the time dimension—in fact, two layers of initial conditions because the slope with respect to time is second order.

In two spatial dimensions we can discretize

$$\frac{1}{c^2}\partial_t^2 u = \bigtriangledown^2 u$$

as

$$\frac{1}{(\Delta t)^2}(u_{+\circ\circ} - 2u_{\circ\circ\circ} + u_{-\circ\circ}) = c^2\left(\frac{1}{(\Delta x)^2}(u_{\circ+\circ} - 2u_{\circ\circ\circ} + u_{\circ-\circ}) + \frac{1}{(\Delta y)^2}(u_{\circ\circ+} - 2u_{\circ\circ\circ} + u_{\circ\circ-})\right)$$

and, if we set $\Delta x = \Delta y$ and $s = (c\Delta t/\Delta x)^2$, this is

$$u_{+\circ\circ} = -u_{-\circ\circ} + s(u_{\circ+\circ} + u_{\circ\circ+} + u_{\circ-\circ} + u_{\circ\circ-}) + 2(1-2s)u_{\circ\circ\circ}$$

The first subscript in this shorthand refers to time; the second and third to x and y respectively:

$$\begin{aligned} u_{+\circ\circ} &= u(t+\Delta t, x, y) \\ u_{-\circ\circ} &= u(t-\Delta t, x, y) \end{aligned}$$

etc.

We see from the above that we need the values of $u(t,x,y)$ at all spatial points for two previous time values in order to calculate $u_{+\circ\circ}$. That means that we must start with $u(t,x,y)$ at all spatial points for $t = 0$ and $t = \Delta t$: these are the initial conditions.

The MATLAB function

```
initMat = initWave2(c,delT,delX,delY,numXs,numYs,kx,ky,stanRtrav)
```

sets up either a standing wave or a travelling wave in the first two layers of the array (matrix)

```
initMat(1:2,1:numXs,1:numYs)
```

The parameters `kx, ky` give the direction of the wave. With `k` $= \sqrt{(\texttt{kx})^2 + (\texttt{ky})^2}$ here is the code for a standing wave.

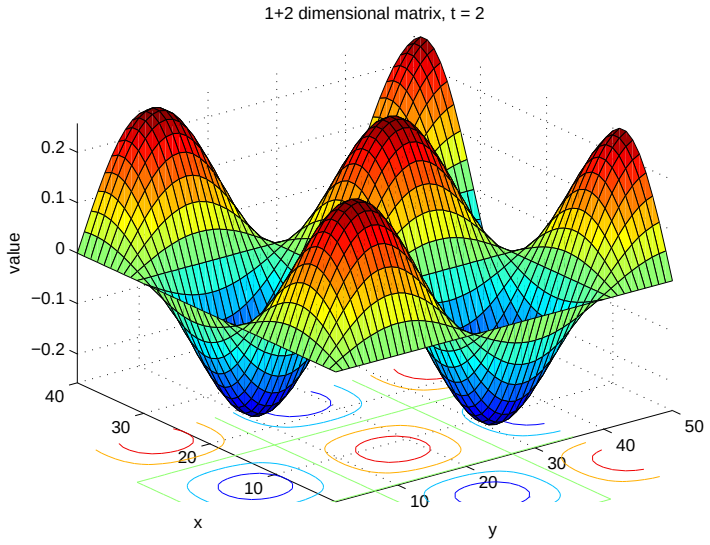
```
initMat(1,:,:)  =  zeros(numXs,numYs);
initMat(2,:,:)  =  sin(c*k*pi*delT)*
                   sin(kx*pi*Xvals)*
                   sin(ky*pi*Yvals)
```

And here is the code for the travelling wave.

```
[X,Y]           =  meshgrid(Xvals,Yvals)
Tdel            =  ones(size(X))*delT
initMat(1,:,:)  =  sin(pi*(kx*X + ky*Y - c*0))
initMat(2,:,:)  =  sin(pi*(kx*X + ky*Y - c*k*Tdel))
```

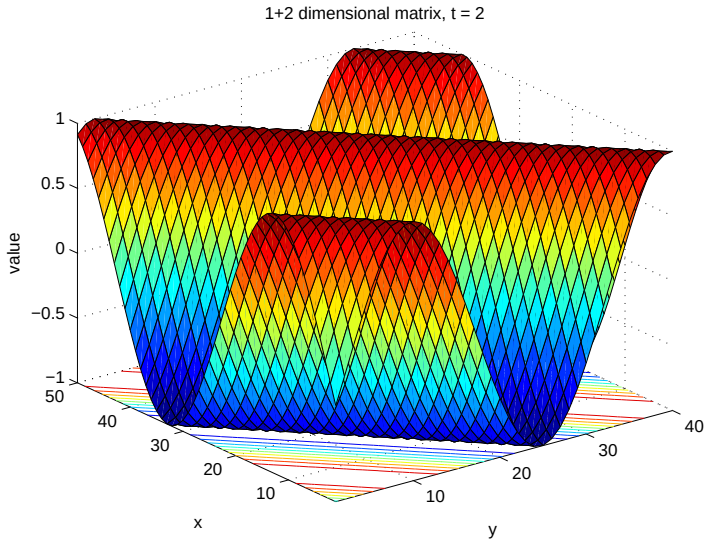
To illustrate both cases, here are plots of `initMat` for

```
initMat = initWave2(1,1/60,1/60,1/60,50,40,3,4,'s');
```



and

```
initMat = initWave2(1,1/60,1/60,1/60,50,40,3,4,'t');
```



As with the potential-well example of Laplace's equation, we must know the solution already, at least for the initial conditions. But we proceed with the numerical calculation because the spatial boundary conditions give rise to interesting simulations. There are two cases: cyclic and rigid.

Cyclic boundary conditions essentially make the boundary invisible:

$$\begin{aligned} u(t, \texttt{xMax}+1, y) &\quad \text{is matched up with} \quad u(t, 1, y) \\ u(t, -1, y) &\quad \text{is matched up with} \quad u(t, \texttt{xMax}, y) \\ u(t, x, \texttt{yMax}+1) &\quad \text{is matched up with} \quad u(t, x, 1) \\ u(t, x, -1) &\quad \text{is matched up with} \quad u(t, x, \texttt{yMax}) \end{aligned}$$

Effectively, the wave is treated as if it were in infinite space, and it behaves accordingly.

Rigid boundary conditions treat the wave as if it were in a bounded pool:

$$\begin{aligned} u(t, \texttt{xMax}+1, y) &= 0 \\ u(t, -1, y) &= 0 \\ u(t, x, \texttt{yMax}+1) &= 0 \\ u(t, x, -1) &= 0 \end{aligned}$$

If the initial wave is a standing wave, a change in boundary conditions makes no difference. But a travelling wave with rigid boundary conditions sloshes interestingly.

The wave equation solver essentially implements the equation for $u_{+\infty}$ at the beginning of this Note.

```
solnMat = pdeWaveDirich2D(initMat,c,delT,delX,delY,maxT,bc);
```

Four cases to illustrate the discussion above arise from two initial waves.

```
initMatS = initWave2(1/sqrt(2),1/60,1/60,1/60,41,31,3,4,'s');
initMatT = initWave2(1/sqrt(2),1/60,1/60,1/60,41,31,3,4,'t');
```

These are each supplied to `pdeWaveDirich2D()`, e.g.,

```
solnMatSC = pdeWaveDirich2D(initMatS,1/sqrt(2),1/60,1/60,1/60,55,'c')
solnMatTC = pdeWaveDirich2D(initMatT,1/sqrt(2),1/60,1/60,1/60,55,'c')
solnMatTR = pdeWaveDirich2D(initMatT,1/sqrt(2),1/60,1/60,1/60,55,'r')
```

The resulting 55-by-41-by-31 matrices can be plotted as an animation with

```
animMatrix(solnMat)
```

This uses `surfc()` to plot the initial values, pauses five seconds (using `pause(5)`) to give time to click on the figure, then uses `surfc()` repeatedly, followed by `pause(0.2)`, to display each subsequent step in turn.

The resolutions given reveal the numerical approximations in the code. Higher resolutions behave better.

10. The Schrödinger Equation I: Physics. Quantum mechanics needs the equation of motion

$$E = \frac{p^2}{2m} + V$$

(relating momentum, p, of a particle to its energy, E, via its mass m) to be formulated as *operators* on a *wavefunction* (Book 8c, Part IV, Note 35).

$$\begin{aligned} E &= i\hbar\partial_t \\ p &= -i\hbar\partial_x \qquad \text{in one dimension} \\ \vec{p} &= -i\hbar \vec{\text{grad}} \qquad \text{in } > 1 \text{ dimension} \end{aligned}$$

With wavefunction $u(t,x)$ (or $u(t,x,y)$ or $u(t,x,y,z)$) this becomes the time-dependent Schrödinger equation

$$i\hbar\partial_t u = -\frac{\hbar^2}{2m}\partial_x^2 u + Vu$$

(Note that the "operator" describing the effect of the potential on the wavefunction is just the product, $V(x,y,z)u$.)

Schrödinger's equation

$$i\hbar\partial_t u = -\frac{\hbar^2}{2m} \bigtriangledown^2 u + Vu$$

(whether $\bigtriangledown^2$ is in one, two or three spatial dimensions) is a partial slope equation, and our experience in Notes 8 and 9 is that we must have a pretty good idea of what such an equation is about before we can do the numerical calculations to visualize it. For Schrödinger's equation we are going to rely a great deal on the physics.

The first physical consideration we'll exploit is that an isolated system has a definite energy. So we can take E to be just a number and we can try

$$u(t,x,y,z) = e^{-iEt/\hbar}\psi(x,y,z)$$

as a decomposition of u into temporal and spatial parts. This of course is motivated by

$$\begin{aligned} i\hbar\partial_t e^{-iEt/\hbar} &= i\hbar\frac{-iE}{\hbar}e^{-iEt/\hbar} \\ &= Ee^{-iEt/\hbar} \end{aligned}$$

But we also saw it as the temporal part of the wavefunction in Week 7c, Note 2.

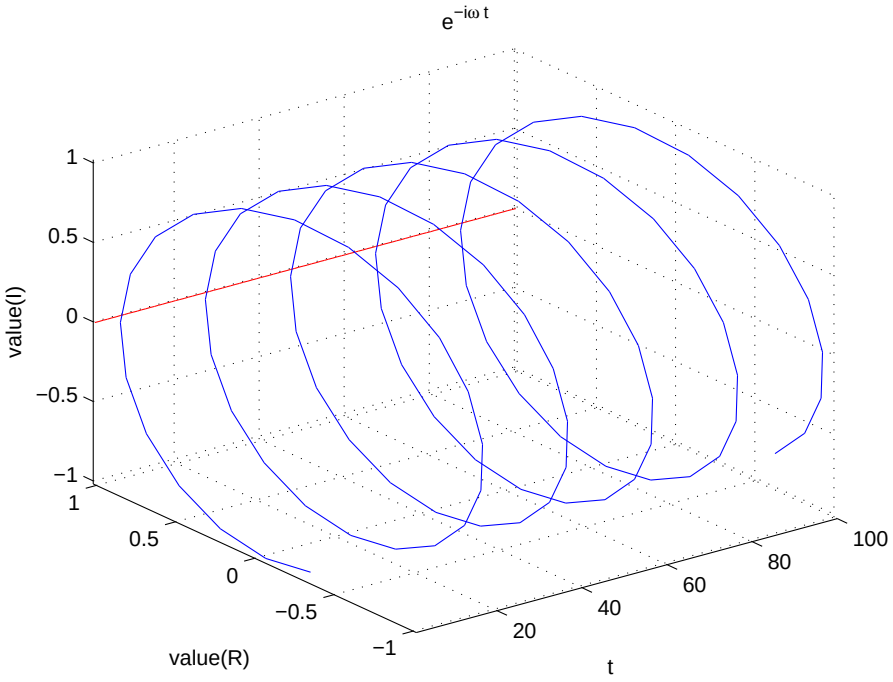
Out of this we find Schrödinger's time-independent equation for the spatial part, ψ,

$$E\psi = -\frac{\hbar^2}{2m} \bigtriangledown^2 \psi + V\psi$$

But we return to the behaviour with time. The exponential, $e^{-i\omega t}$, with $\omega = E/\hbar$, describes a helix in three dimensions, where one dimension represents the time, t, and the other two give the 2-number that is the result of

$$e^{-i\omega t} = \cos\omega t - i\sin\omega t$$

Here's a picture.



This figure labels the t-axis and the two values axes. R gives the $\cos(\omega t)$ component and I gives the $\sin(\omega t)$ component. The red line at $R = 1, I = 0$ is the absolute value

$$(\cos\omega t - i\sin\omega t)(\cos\omega t + i\sin\omega t) = 1$$

For a free particle, the potential V is constant and we can take it to be zero. Then

$$\psi = e^{ikx} = e^{ipx/\hbar}$$

(the spatial part in Note 2 of Week 9a) is a solution (in one dimension), giving

$$E\psi = -\frac{\hbar^2}{2m}\partial_x^2\psi = -\frac{\hbar^2}{2m}\left(\frac{ip}{\hbar}\right)^2 e^{ipx/\hbar} = \frac{p^2}{2m}\psi$$

the classical equation of motion, once we remove ψ from both sides.

This $\psi = e^{ikx}$ is also a helix, but in space. Its squared absolute value is interpreted as the *probability* that the particle is to be found at a position x. For the moment, with no constant c in front of the e^{ikx}—i.e., ce^{ikx}—that probability is 1 everywhere.

This is clearly impossible. Indeed, in infinite space $-\infty < x < \infty$, there is no finite constant c that can fit the interpretation.

The physics takes us back to square one.

We do not need to limit ourselves to a single energh E. The solution to Schrödinger's time-dependent equation can be a *superposition*—a linear combination—of many different energies, E_j, and the helical solution

$$e^{-E_j t/\hbar}$$

of each. Since $E_j = p_j^2/2m$, each of these enrgies gives rise to a different momentum, and we have a linear combination of the helices

$$e^{p_j x/\hbar}$$

If we use coefficients in this linear combination which follow a Gaussian bell curve (Week 9, Note 7) we have a Fourier transform something like

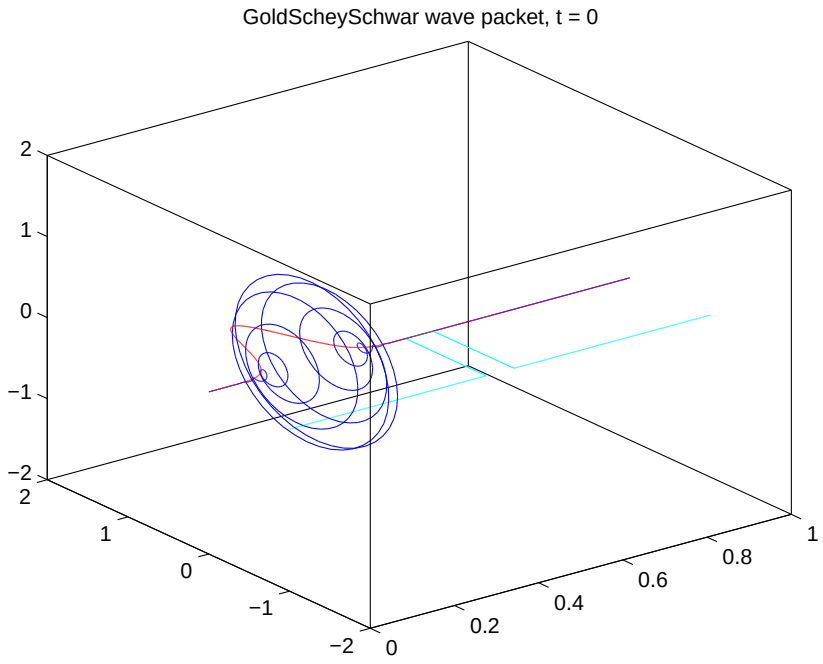
$$\sum_j e^{-p_j^2/2\delta^2} e^{ip_j x}$$

We saw in Note 7 of Week 9 that this transformation of a bell curve in p-space becomes a bell curve in x-space, with the net effect that we can also consider the x-values of the wave function to follow a bell curve such as

$$e^{-(x-x_0)^2/2\sigma^2}$$

(Here, σ is the standard deviation, a measure of the width of the bell curve. In p-space, δ, above, plays the same role.)

Putting this together with $e^{ip_0 x/\hbar}$ where p_0 is an average momentum of all the plane waves making up the packet, we have the following wavefunction in one spatial dimension.

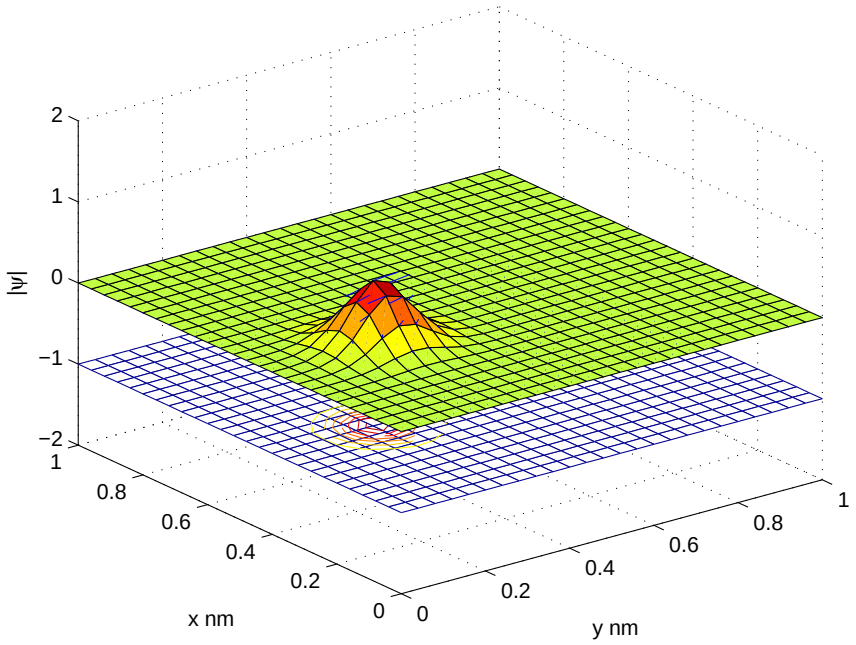


Again, the red line at $I = 0$ gives the squared absolute value, i.e., the probability in space, and we see that this is a *localized* "wave packet".

Not only does this give us something that actually looks like a particle, it also has the advantage that we can keep it far enough from the boundary when we evaluate the Schrödinger equation numerically that we can simply set the boundary conditions to zero.

In two spatial dimensions we must improvise to fit into a 3-D plot the two dimensions of the underlying space, plus the two dimensions of 2-numbers that describe the wave function. We do this by plotting the absolute value of the wavefunction (a function of two variables which needs three dimensions to plot) and add to this a horizontal arrow at each grid point, whose direction gives the phase of the 2-number wavefunction.

Here is such a plot: a wave packet in two dimensions.



In this case and for the case of one spatial dimension, to include a dimension for time t we just animate the plot.

11. The Schrödinger Equation II: Animating in 1D. In one spatial dimension the Schrödinger equation

$$\begin{aligned} i\hbar\partial_t u &= -\frac{\hbar^2}{2m}\partial_x^2 u + Vu \\ &= \hat{H}u \end{aligned}$$

where $\hat{H}$ is the Hamiltonian operator (see Ex. *The Hamiltonian*). This has the formal solution

$$u(t,x) = e^{-i(t-t_0)\hat{H}/\hbar}u(t_0,x)$$

in terms of some (initial) value of u at $t = t_0$.

When we change $u(t,x)$ from a continuous function to a discrete one, for numerical solution, and write $t_{n+1} = t_n + \delta$ for successive time steps, then $u_{n+1}^{\vec{}}$ and $u_n^{\vec{}}$ become vectors over all the space (x) locations, e.g., $u(t_n, x_j)$ becomes u_{nj}, and the operator $\hat{H}$ becomes a matrix $H_{jj'}$:

$$u_{n+1}^{\vec{}} = e^{-i\delta H/\hbar}u_n^{\vec{}}$$

H is a *hermitian* matrix (the 2-number analog of a symmetric matrix) $H_{jj'} = H^*_{j'j}$ where * is the complex conjugate operator (changing i to $-i$ everywhere). Written in matrix form:

$$H = H^\dagger$$

Because of this, the squared-magnitude of $\vec{u}$ does not change in time:

$$\begin{aligned} u_{n+1}^{*\vec{}}u_{n+1}^{\vec{}} &= \vec{u_n^*}e^{i\delta H^\dagger/\hbar}e^{-i\delta H/\hbar}\vec{u_n} \\ &= \vec{u_n^*}e^{i\delta(H^\dagger - H)\hbar}\vec{u_n} \\ &= \vec{u_n^*}I\vec{u_n} \\ &= \vec{u_n^*}\vec{u_n} \end{aligned}$$

Furthermore, the exponential $U = e^{-i\delta H/\hbar}$ is *unitary* (the 2-number analog of orthogonal):

$$U^\dagger U = e^{i\delta H^\dagger/\hbar} e^{-i\delta H/\hbar} = I$$

i.e., $U^\dagger = U^{-1}$.

If we can find a unitary approximation for

$$e^{-i\delta H/\hbar} = I - \frac{i\delta H}{\hbar} + \frac{1}{2}\left(\frac{i\delta H}{\hbar}\right)^2 + \frac{1}{3!}\left(\frac{i\delta H}{\hbar}\right)^3 + \cdots$$

the solution cannot blow up and we will not need to worry about stability.

The *Cayley approximation* gives what we need

$$e^{-i\delta H/\hbar} \approx \frac{I - i\delta H/2\hbar}{I + i\delta H/2\hbar}$$

This gives us

$$\vec{u_{n+1}} = \frac{I - i\delta H/2\hbar}{I + i\delta H/2\hbar}\vec{u_n}$$

or

$$(I + i\delta H/2\hbar)\vec{u_{n+1}} = (I - i\delta H/2\hbar)\vec{u_n}$$

We can expand $\partial_x^2 u = (u_{j+1} - 2u_j + u_{j-1})/\epsilon^2$ as usual for (partial) slope equations, where $\epsilon = x_{j+1} - x_j$ and where I've suppressed the time index n because it's true for any n.

Thus, with

$$Hu = Vu - \frac{1}{a}\partial_x^2 u$$

(I've renamed $2m/\hbar^2 \stackrel{\text{def}}{=} a$) we have

$$u_{n+1,j} + i\frac{\delta}{2\hbar}\left(V_j u_{n+1,j} - \frac{1}{a\epsilon^2}(u_{n+1,j+1} - 2u_{n+1,j} + u_{n+1,j-1})\right) =$$

$$u_{n,j} - i\frac{\delta}{2\hbar}\left(V_j u_{n,j} - \frac{1}{a\epsilon^2}(u_{n,j+1} - 2u_{n,j} + u_{n,j-1})\right)$$

Note that the potential, V, has different values at different x but does not depend on time.

We rearrange this using $\lambda \stackrel{\text{def}}{=} 2\hbar a\epsilon^2/\delta$ to

$$\begin{aligned} u_{n+1,j+1} + (i\lambda - a\epsilon^2 V_j - 2)u_{n+1,j} + u_{n+1,j-1} &= -u_{n,j+1} + (i\lambda + a\epsilon^2 V_j + 2)u_{n,j} - u_{n,j-1} \\ &\stackrel{\text{def}}{=} \Omega_{n,j} \end{aligned}$$

in which the latter definition is another notational convenience instead of writing out the long expression after the first = sign,

We can think of this result

$$u_{n+1,j+1} + (i\lambda - a\epsilon^2 V_j - 2)u_{n+1,j} + u_{n+1,j-1} = \Omega_{n,j}$$

as a matrix equation to be solved for $\vec{u_{n+1}}$, but there's a trick (the Thomas algorithm for solving a tridiagonal mattrix) which does it in only two passes.

We invent two auxiliary functions and suppose

$$u_{n+1,j+1} = e_{n,j}u_{n+1,j} + f_{n,j}$$

Putting the two equations together

$$e_{n,j}u_{n+1,j} + f_{n,j} + (i\lambda - a\epsilon^2 V_j - 2)u_{n+1,j} + u_{n+1,j-1} = \Omega_{n,j}$$

we get

$$u_{n+1,j} = (2 + a\epsilon^2 V_j - e_{n,j} - i\lambda)^{-1}u_{n+1,j-1} + (2 + a\epsilon^2 V_j - e_{n,j} - i\lambda)^{-1}(f_{n,j} - \Omega_{n,j})$$

Comparing this with the definition of $e_{n,j}$ and $f_{n,j}$

$$u_{n+1,j+1} = e_{n,j}u_{n+1,j} + f_{n,j}$$

we get iterations for both of these auxiliary functions

$$\begin{aligned} e_{n,j-1} &= (2 + a\epsilon^2 V_j - e_{n,j} - i\lambda)^{-1} \\ f_{n,j-1} &= e_{n,j-1}(f_{n,j} - \Omega_{n,j}) \end{aligned}$$

or

$$\begin{aligned} e_{n,j} &= 2 + a\epsilon^2 V_j - i\lambda - 1/e_{n,j-1} \\ f_{n,j} &= \Omega_{n,j} - f_{n,j-1}/e_{n,j-1} \end{aligned}$$

There is nothing in this result for $e_{n,j}$ which depends on time. so we can drop the time index, n, and just write

$$e_j = 2 + a\epsilon^2 V_j - i\lambda - 1/e_{j-1}$$

and the e-function can be calculated just once, before entering the time loop of the simulation we're getting to.

Both e_j and $f_{n,j}$ can be calculated for all j by starting at the $j = 0$ boundary condition $u_{n0} = 0$ (all n). We will justify this condition, as mentioned in Note 10, by making the initial wavefunction a Gaussian wavepacket, zero far enough away from its location and especially at the boundaries—and by keeping it far enough from the boundaries during simulation.

Thus

$$u_{n+1,2} = (2 + a\epsilon^2 V_1 - i\lambda)^{-1}u_{n+1,1} + \Omega_{n,1}$$

and so

$$\begin{aligned} e_1 &= 2 + a\epsilon^2 V_1 - i\lambda \\ f_{n,1} &= \Omega_{n,1} \end{aligned}$$

Calculating all these e_j and $f_{n.j}$ constitutes the forward pass of the algorithm.

The backward pass finds $u_{n+1,j}$ by starting at the maximum $j = J$ and using the other boundary conditions $u_{n,J} = 0$ all n.

$$0 = e_{J-1}u_{n+1,J-1} + f_{n,J-1}$$

so

$$u_{n+1,J-1} = -f_{n,J-1}/e_{J-1}$$

and

$$u_{n+1,j} = (u_{n+1,j+1} - f_{n.j})/e_j$$

for all $j < J - 1$ (and, as usual, all n).

This is the two-pass algorithm we must execute for all n ($n = 0 : N$) and all j (first $j = 2 : J - 1$, second $j = J - 1 : -1 : 1$).

But we must initialize the wavepacket as that Gaussian

$$u(0, x) = e^{ik_0 x}e^{-(x-x_0)^2/2\sigma_0^2}$$

which moves to the right with average momentum k_0. We set the starting location

$$x_0 = J\epsilon/4$$

and the standard deviation (width of the Gaussian bell)

$$\sigma_0 = J\epsilon/20$$

to keep $u(0,x)$ negligible at $x = 0$ (and at $x = J\epsilon$). To keep the wavepacket from going past $3J\epsilon/4$, three quarters of the simulation space, in the time $N\delta$ given for the full run, we have

$$\frac{J\epsilon}{2N\delta} \approx V = \frac{p}{m} = \frac{\hbar k_0}{m} = \frac{2k_0}{\hbar a}$$

so

$$k_0 = \frac{\hbar a}{2}\frac{J\epsilon}{2N\delta} = \frac{\hbar a J}{4N}\frac{\lambda}{2\hbar a\epsilon} = \frac{J\lambda}{8N\epsilon}$$

where we defined λ earlier as $2\hbar a\epsilon(\epsilon/\delta)$.

We must choose values for ϵ and δ: experiment is a guide. We can choose $J\epsilon$, the size of the space, to be 1nm (nanometers) and we can work in time units of fs (femtoseconds) and energy units of eV (electron volts). In these units, for instance, lightspeed is 300nm/fs and Plank's constant $\hbar = 0.66$eV-fs. The constant $a = 26/\text{eV-nm}^2$, and λ is dimensionless.

We finally pick $\lambda = 1$ and $k_0\epsilon = \pi/20$, i.e., 1/40 of a phase cycle per space-step in the iteration. Thus $5J = 2N\pi$ and we can experiment with various initial momenta by choosing reasonable numbers of spatial steps to simulate.

J	N	$\epsilon = 1/J$ nm	k_0	$E_0 = k_0^2/a$ eV
1000	800	10^-3	50π	941
1414	1130	$0,707 \times 10^-3$	70.7π	1883
2000	1600	0.5×10^-3	100π	3766

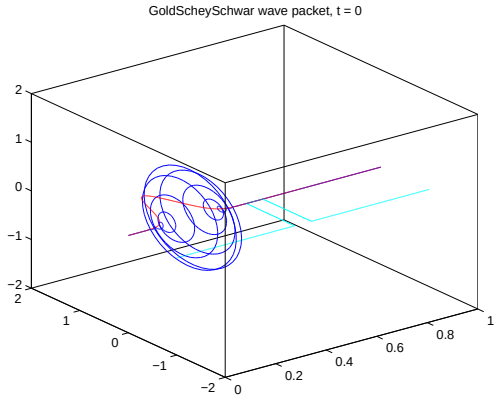
When we design potentials such as a barrier, we'll give them energies of the size of the middle of these three, so that we can experiment with energies less than, equal to, or more than the (barrier) energy.

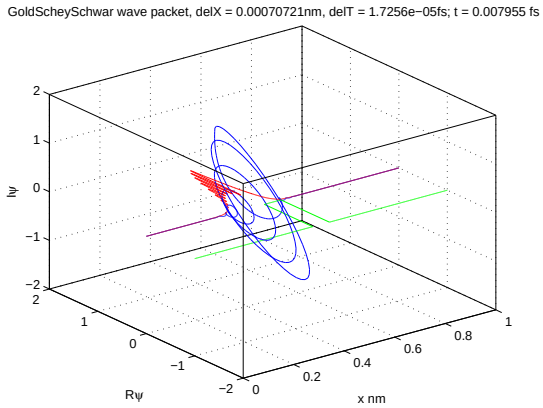
The MATLAB function `freeSchroeGauss(E,bw)` simulates the motions of a Gaussian wavepacket in one dimension with the above three possible energy levels ($E = 1, 2, 3$) and for various potential barriers or wells (`bw = 'f'` for free motion with no barrier or well; `bw = 'b'` for barrier height 1883 eV and width 0.0432nm; `bw = 'w'` for a well of the same depth and width; `bw = 'u'` for a step up (barrier) of the above height; `bw = 'd'` for a step down (well); `bw = 'h'` for a harmonic oscillator potential equalling the wave energy at $x = J\epsilon(1/4 - 1/10)$; `bw = 'ℓ'` for a Lennard-Jones potential with minimum at $x = J\epsilon/4$; and `bw = 'r'` for an inverse-r potential which is 0 at $x = J\epsilon/4$ and $x = 3J\epsilon/4$).

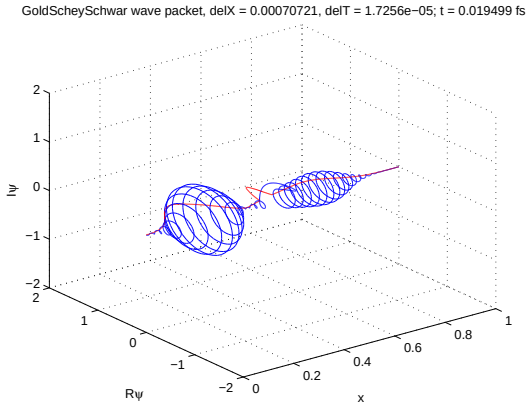
The function implements four preliminary steps followed by the double loop (solving the above two-pass boundary-condition problem for each time step) that does the simulation together with plotting the animation (`pause(0.01)` between time steps).

The preliminaries are, after initializing constructs $\hbar$, m (the electron mass, $0.5\text{MeV}/c^2$), a, x_0, σ_0: 1) setting J and N for the chosen energy level E; 2) creating the potential V for the chosen value of parameter `bw`; 3) doing the pass to create the auxiliary function e_j 4) initializing the wavefunction to the Gaussian $e^{ixk_0}e^{-(x-x_0)/2\sigma_0^2}$ (NB $x_0 = J\epsilon/4$ except for the harmonic potential `bw = 'h'` when $x_0 = J\epsilon/2$.)

Here are three extracts from the animation for a barrier (`bw = 'b'` shown in green) of the same energy as the wavepacket ($E = 2$).



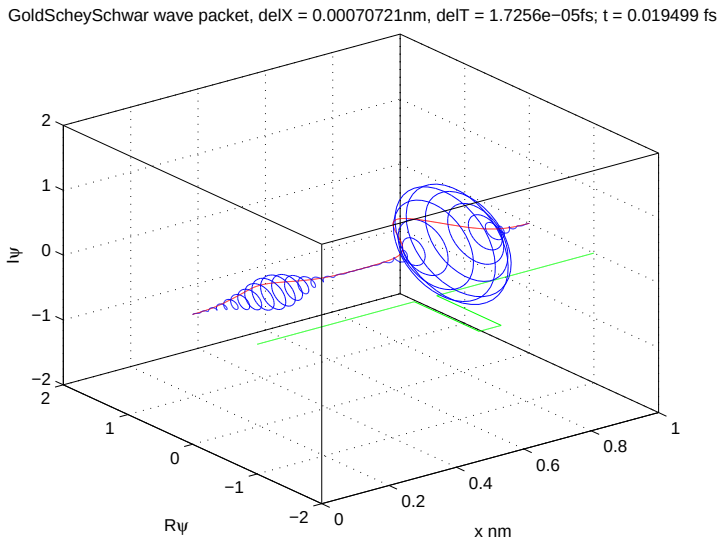




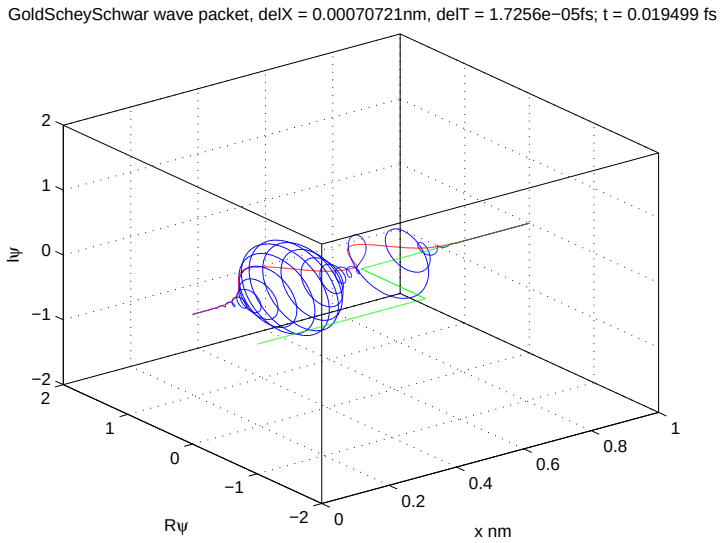
In the middle plot we can see the self-interference as the wavepacket hits the barrier. In the third plot we can see the "resonance" that persists inside the barrier for quite a long time after the wavepacket has partly reflected away from the barrier and partly transmitted through it.

The amplitudes of the wave, in red, give the square roots of the probabilities of finding the electron at given values of x—in this case, at the end, reflection has the greatest probability, then transmission, then a small chance that the electron is trapped *inside* the barrier before eventually leaking out.

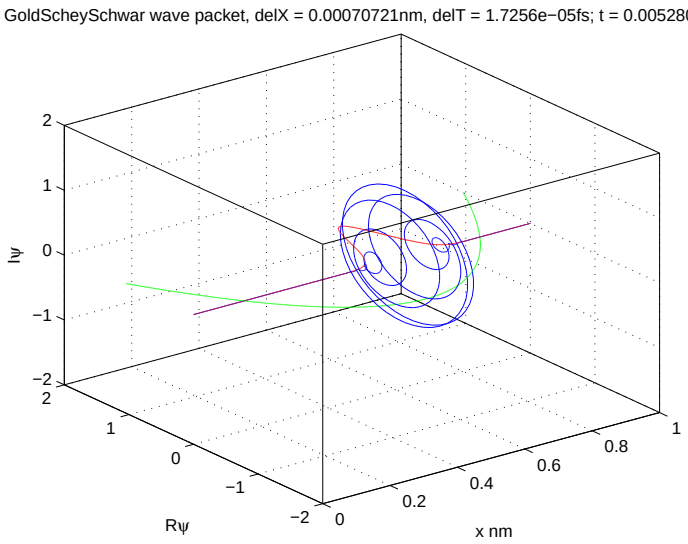
For a well (`E = 2, bw = 'w'`) the electron is more likely to be transmitted



For a step up (`E = 2, bw = 'u'`) the electron is entirely reflected, except for a resonance



For the harmonic oscillator (`E = 2, bw = 'h'`) the electron moves back and forth with no visible "jaggies" in the amplitude but the wavepacket is squeezed as it reflects



12. The Schrödinger Equation III: Animating in 2D. Most of the 1D discussion in Note 11 goes over to two dimensions. But the simple extension of $\bigtriangledown^2 u$, which gives a tridiagonal matrix in 1D, is no longer tridiagonal—see Note 8—and so the two-pass (Thomas) algorithm no longer works,

The trick is to do the two dimensions in two steps each time. Instead of

$$u_{n+1,j+1} + u_{n+1,j-1} - (2 + a\epsilon^2 V_j - i\lambda)u_{n+1,j} = \Omega_{n,j}$$

we use

$$u_{n+\frac{1}{2},j+1,k} + u_{n+\frac{1}{2},j-1,k} - (2 + a\epsilon^2 V_{jk} - 2i\lambda)u_{n+\frac{1}{2},j,k} = \Omega_{n,j,k}$$

followed by

$$u_{n+1,j+1,k+1} + u_{n+1,j-1,k+1} - (2 + a\epsilon^2 V_{jk} - 2i\lambda)u_{n+1,j,k} = \Omega_{n+\frac{1}{2},j,k}$$

with $\Omega_{n+\frac{1}{2}}$ defined as before in terms of u_n and Ω_n defined in terms of $u_{n+\frac{1}{2}}$. Note the doubling of λ because of each half step,

Each of these half steps in time involves a spatial boundary-value problem in only one dimension, which is still tridiagonal. So the two-pass method of Note 11 can be used.

The iterations become

$$\begin{aligned}
\Omega_{jk} &= -u_{j+1,k} - u_{j-1,k} + (2 + a\epsilon^2 V_{jk} + 2i\lambda)u_{jk} \\
e_{x}{}_{jk} &= \begin{cases} 2 + a\epsilon^2 V_{jk} - 2i\lambda & j = 1 \\ 2 + a\epsilon^2 V_{jk} - 2i\lambda - 1/e_{x}{}_{j-1,k} & j > 1 \end{cases} \\
f_{jk} &= \begin{cases} \Omega_{jk} & j = 1 \\ \Omega_{jk} + f_{j-1,k}/e_{x}{}_{j-1,k} & j > 1 \end{cases} \\
u_{jk} &= \begin{cases} -f_{j-1,k}/e_{x}{}_{j-1,k} & j = J \\ u_{j+1,k} - f_{j,k}/e_{x}{}_{j,k} & j < J \end{cases}
\end{aligned}$$

and

$$\Omega_{jk} = -u_{j,k+1} - u_{j,k-1} + (2 + a\epsilon^2 V_{jk} + 2i\lambda)u_{jk}$$

$$
\begin{aligned}
e_{yjk} &= \begin{cases} 2 + a\epsilon^2 V_{jk} - 2i\lambda & k = 1 \\ 2 + a\epsilon^2 V_{jk} - 2i\lambda - 1/e_{yj,k-1} & k > 1 \end{cases} \\
f_{jk} &= \begin{cases} \Omega_{jk} & k = 1 \\ \Omega_{jk} + f_{j,k-1}/e_{yj,k-1} & k > 1 \end{cases} \\
u_{jk} &= \begin{cases} -f_{j,k-1}/e_{yj,k-1} & k = J \\ u_{j,k+1} - f_{j,k}/e_{yj,k} & k < J \end{cases}
\end{aligned}
$$

It is this second u_{jk} that is plotted, after this second half-step. We do not need to store the time-dependence of u (i.e., $u_{n+1/2,j,k}$ and $u_{n+1,j,k}$) because we only need it plotted.

Note that we must distinguish e_x from e_y because these are evaluated once for all before the simulation loop. Because Ω, f and even u are workspaces, they can be shared across the two half-steps without damage.

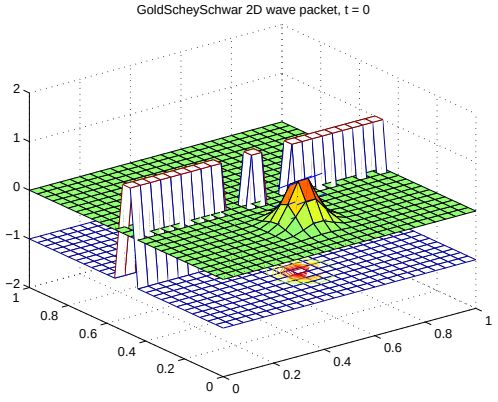
The MATLAB function `free2dSchroeGauss(E,bw,v,ang)` has the same structure as the 1D routine of Note 11, with the two-stage simulation loop depicted above. It has two new parameters: `ang` is the travelling direction for the wavepacket, in radians from the x-direction; `v` specifies a viewing direction for the 3D plots, which the program translates into azimuth and elevation for the MATLAB `view(az,el)` command. For instance, `v = '3'` translates into the default `view(−37.5,30)`. Depending on the view chosen the program also decides whether to plot the potential ("m̂esh" or "`cont`"our) and how to plot the wave ("`quiv`"er or "`surf`"ace)

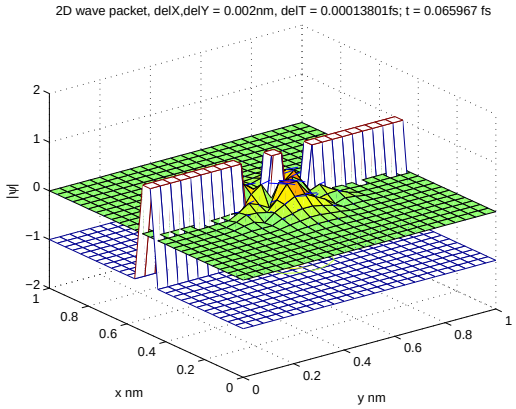
`v`	`az`	`el`	`mesh`	`cont`	`quiv`	`surf`
`'y'`	90	0	√		√	√
`'z'`	0	90		√	√	
`'3'`	−37.5	30	√		√	√

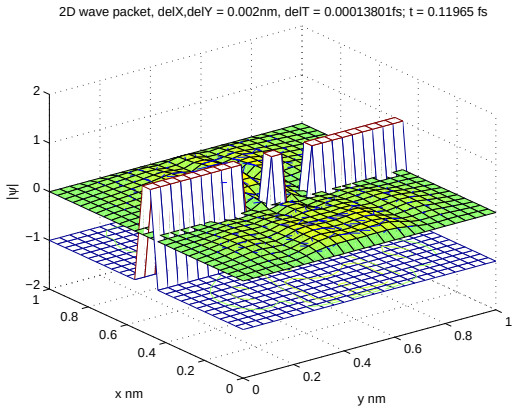
For the sake of speed the 2D routine uses half the number of x- and y- mesh points, $J \to J/2$. And the number of t-mesh points is doubled, $N \to 2N$.

The values for E remain 1,2,3. The values for `bw` describe a barrier (`'b'`), a well (`'w'`), a barrier with 1 or 2 slots (`'1'`, `'2'`, respectively), or a uniform gravitational field (`'g'`).

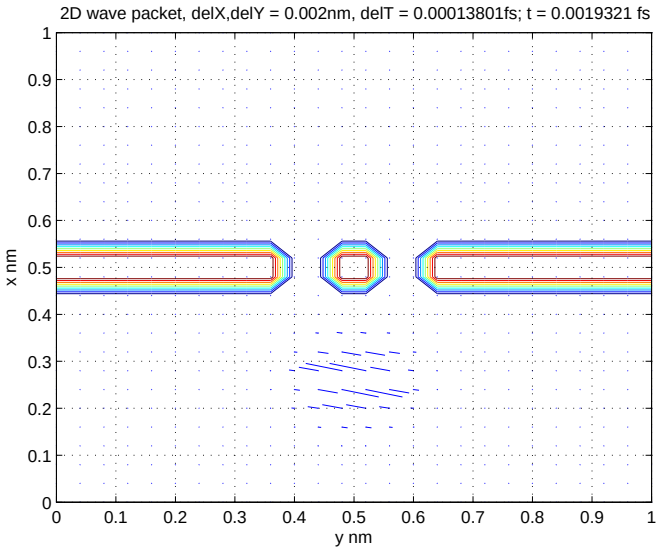
Here are the start and post-passage snapshots from `free2dSchroeGauss(1,'2','3',0)`



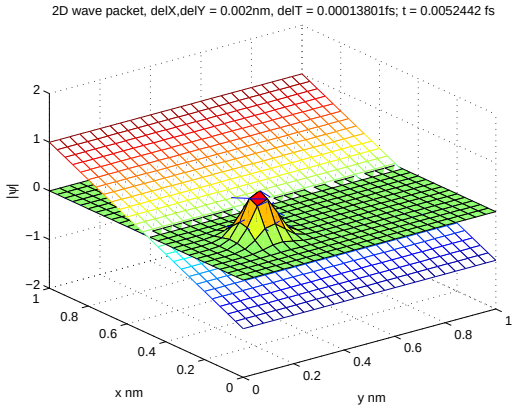


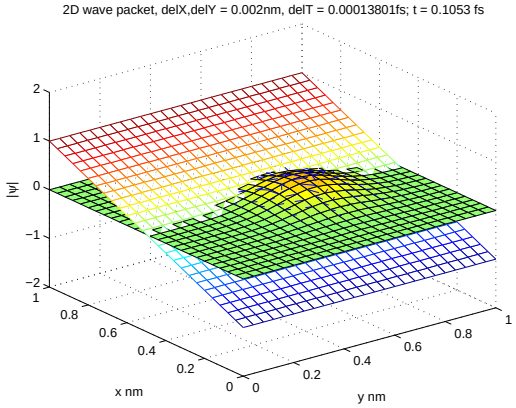


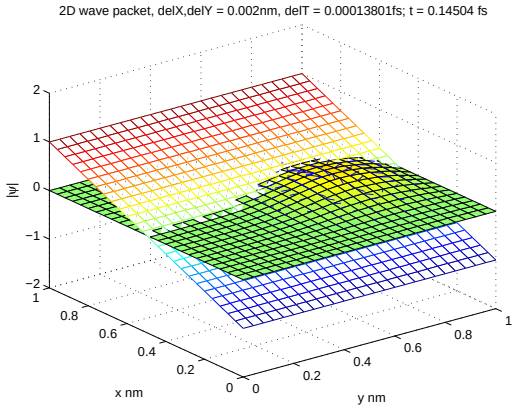
Here is an early snapshot from the `'z'` perspective, emphasizing the phase arrows.



Here are three snapshots of an electron being thrown upwards at 45^o in a uniform gravitational field.


2D wave packet, delX,delY = 0.002nm, delT = 0.00013801fs; t = 0.0052442 fs
|ψ|
x nm
y nm


2D wave packet, delX,delY = 0.002nm, delT = 0.00013801fs; t = 0.1053 fs
|ψ|
x nm
y nm


2D wave packet, delX,delY = 0.002nm, delT = 0.00013801fs; t = 0.14504 fs
|ψ|
x nm
y nm

The expected spread of the wavepacket is very evident here.

Part III. Quantum Electromagnetism

13. The electromagnetic Schrödinger equation.
14. Simulating a charged wavepacket moving near a current.
15. Links with geometry.
16. Local action versus action-at-a-distance.
17. Other symmetries, other forces.

Part IV. Quantum Field Theory: Matrix Quantum Mechanics

18. Introduction to Quantum Fields.
19. Small matrices.
20. Tensor products.
21. Spin.
22. Vectors and spinors,
23. Multiple and independent systems.
24. A simple field.
25. The Yukawa potential.
26. Perturbation approximations.
27. Fermions.
28. Slopes and antislopes of 2D numbers, etc.
29. Charge conservation and antimatter.
30. Relativistic quantum field theory redux, so far.

Part V. Functional Integrals

31. Path amplitudes.
32. Functionals.
33. Gaussian integrals.
33. Gaussian integrals.
34. Diagrams and QED.
35. Chirality and electroweak.
36. Green's functions.
37. Propagators.
38. Quantum Computing.
39. Binary Fourier transform.
40. Quantum Fourier transform.
41. Finding periods.
42. Quantum key distribution.
43. No cloning.
44. Database search.
45. Detecting and correcting errors.
46. Nonlocality: Einstein-Podolsky-Rosen.
47. Building a quantum computer.

II. **The Excursions**

You've seen lots of ideas. Now *do* something with them!

1. Write a program `demoJacobi(A,Dm1,Dp,b)` to test the Jacobian iterative solution of $Ax = b$ from Note 8. For example

$$A = \begin{pmatrix} 4 & -1 & \\ -1 & 4 & -1 \\ & -1 & 4 \end{pmatrix} \quad D = \begin{pmatrix} 4 & & \\ & 4 & \\ & & 4 \end{pmatrix} \quad D' = \begin{pmatrix} & -1 & \\ -1 & & -1 \\ & -1 & \end{pmatrix} \quad b = \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix}$$

   and $D^{-1} = I/4$

$$\begin{aligned} u_1 &= \frac{1}{4}\begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix} \quad Au_1 = \begin{pmatrix} 2 \\ -1 \\ 2 \end{pmatrix} \\ u_2 &= \frac{1}{4}\begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix} \quad Au_2 = \frac{1}{4}\begin{pmatrix} 7 \\ 0 \\ 7 \end{pmatrix} \\ &\vdots \\ u_{12} &= \begin{pmatrix} 0.5714 \\ 0.2857 \\ 0.5714 \end{pmatrix} \quad Au_{12} = \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix} \end{aligned}$$

   with Au_{12} being as shown to four decimal places in this example.
   (By the way, what does u_2 tell us about the exact solution?)

2. Note that the Jacobi iteration takes whole steps from u_1 to u_2 to .. We could alternatively adjust only one component of u_j per iteration. This is called the Gauss-Seidel iteration, and is faster and converges at least everywhere Jacobi does. Look up the Gauss-Seidel iterative solution of sparse matrix equations.

3. Write the MATLAB functions `initWave2(), pdeWaveDirich2D()` and `animMatrix()` used to illustrate the wave equation in Note 9.

4. **Stability.** In simulating the wave equation numerically, as we did in Note 9, we must be careful in choosing the temporal snd spatial step sizes. A wrong choice can lead to a system of equations which blows up numerically: it can be *unstable*.
   Here is a sketch of the stability issue in one spatial dimension. The wave equation $\partial_t^2 u = c^2 \partial_x^2 u$ can be expanded, with $s = (c\Delta t/\Delta x)^2$

$$u_{+\circ} - 2u_{\circ\circ} + u_{-\circ} = s(u_{\circ+} - 2u_{\circ\circ} + u_{\circ-})$$

   with the usual shorthand $u_{+\circ} = u(t + \Delta t, x)$, $u_{-\circ} = u(t - \Delta t, x)$, etc. Rearranging,

$$u_{+\circ} = 2(1 - s)u_{\circ\circ} + s(u_{\circ+} + u_{\circ-}) - u_{-\circ}$$

   And, writing the spatial part of u as a vector

$$\vec{u_+} = B\vec{u} - \vec{u_-}$$

   with

$$B = \begin{pmatrix} 2(1-s) & s & \cdots & \\ s & 2(1-s) & & \\ \vdots & & & \\ & & & 2(1-s) \end{pmatrix}$$

for rigid boundary conditions, and

$$B = \begin{pmatrix} 2(1-s) & s & \cdot\cdot & s \\ s & 2(1-s) & & \\ \vdots & & & \\ s & & & 2(1-s) \end{pmatrix}$$

for cyclic boundary conditions.
It seems that there may be a difference in behaviour of these systems as s increases from less than 1 to greater than 1. We can explore this behaviour independently of initial values u_0 and u_1. We define matrices B_n and A_n iteratively.

$$\begin{aligned} \vec{u_n} &= B_n\vec{u_1} + A_n\vec{u_0} \\ \vec{u_{n+1}} &= B\vec{u_n} - u_{n-1} \\ &= B(B_n\vec{u_1} + A_n\vec{u_0}) - (B_{n-1}\vec{u_1} + A_{n-1}\vec{u_0}) \\ &= (BB_n - B_{n-1})\vec{u_1} + (BA_n - A_{n-1})\vec{u_0} \end{aligned}$$

so

$$B_{n+1} = BB_n - B_{n-1}$$

and

$$A_{n+1} = BA_n - A_{n-1}$$

Since the iterations are identical we can focus on B_n. It starts

$$\begin{aligned} \vec{u_1} &= \vec{u_1} \\ \vec{u_2} &= B\vec{u_1} - \vec{u_0} \end{aligned}$$

so $B_1 = I$ and $B_2 = B$
Here are the first few matrices (As as well).

n	B_n	A_n
2	B	$-I$
3	$B^2 - I$	$-B$
4	$B(B^2 - 2I)$	$-(B^2 - I)$
5	$B^4 - 3B^2 + I$	$-B(B^2 - 2I)$
:		

Write a function which calculates B of various sizes (2-by-2, 3-by-3, etc.) for rigid or cyclic boundary conditions, and with s as a parameter.
Write a function which uses this B to calculate B_n and show that it becomes arbitrarily large when $s = 2$ but is manageable for $s \leq 1$.
Look up [Olv08, Sect.11.4] for a more extended clear treatment of this issue.
Why is the stability threshold in two dimensions, compared with $s = 1$ in one dimension, $s = 1/2$? Why did I set $c = 1/\sqrt{2}$ in the last two examples of Note 9?

5. **The Hamiltonian.** Quantum mechanics conventionally makes much use of the "Hamiltonian" operator, $\hat{H}$. This comes from a rearrangement of the time-independent Schrödinger equation of Note 10 from

$$E\psi = -\frac{\hbar^2}{2m} \bigtriangledown^2 \psi + v\psi$$

to

$$\hat{H}\psi = E\psi$$

with

$$\hat{H} = -\frac{\hbar^2}{2m} \bigtriangledown^2 + V$$

Thus the energies E are the eigenvalues (see, e.g., Week 3, Note 5) of the operator $\hat{H}$, if we were to think of $\hat{H}$ as a matrix and E as a diagonal matrix.

6. An alternative to using phase arrows to visualize the phase of a two-dimensional wave function (Note 10) is to use colour coding. Look up Bernd Thaller's [Tha00] for a systematic treatment along these lines: with Mathematica code but no discussion of the programming.

7. **Dispersion relations.** For a photon (see week 7a) which is an electromagnetic wave travelling at lightspeed c, show that $\omega = ck$ is the relationship between angular frequency ω and the angular momentum k. For a particle with mass m, $E = p^2/2m$ relates kinetic energy E and momentum p. Show that $\omega = k^2/2m$ is the ("dispersion") relationship for particle with mass.

8. Show that the Cayley approximation to $e^{-i\delta H/\hbar}$ of Note 11 is unitary.

$$\frac{I - i\delta H/2\hbar}{I + i\delta H/2\hbar} \frac{I + i\delta H^\dagger/2\hbar}{I - i\delta H^\dagger/2\hbar} = I$$

9. Show that the Cayley approximation expands to the series for $e^{-i\delta H/\hbar}$ correctly to order δ^2. (Use the binomial series for $(1+x)^{-1}$ in Note 7 of Week ii.)

10. I have followed slavishly the pedagogical initiative of Goldberg, Schey and Schwartz [GSS67] in setting up the simulations of Note 11. Compare their 1967 simulation (they had to take individual photos of their screen and merge them into a movie) with your own simulation.

11. Any part of the Prefatory Notes that needs working through.

References

[GSS67] Abraham Goldberg, Harry M. Schey, and Judah L. Schwartz. Computer-generated motion pictures of one-dimensional quantum-mechanical transmission and reflection phenomena. *Am. J. Phys. 35, 177 (1967)*, 35(3):177–86, March 1967. cited by Andrew Harris https://www.youtube.com/watch?v=Xj9PdeY64rA.

[Olv08] Peter J Olver. Numerical analysis lecture notes 11 Finite difference methods for partial differential equations. URL http://www.math.umn.edu/∼olver/num_/lnp.pdf (Last accessed 16/5/10), 2008.

[Tha00] Bernd Thaller. *Visual Quantum Mechanics: Selected Topics with Computer-Generated Animations of Quantum-Mechanical Processes*. Springer-Verlag, New York, 2000.