Partial differential equations and applications to Electromagnetism

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

2017

Famous PDEs

In the Natural Sciences we often encounter problems with many variables constrained by boundary conditions and initial values. Many of these problems can be modelled as partial differential equations. One case which arises in many situations is the so-called wave equation whose one-dimensional form reads

$$\frac{\partial^2 u}{\partial x^2} = A\frac{\partial^2 u}{\partial t^2}, \tag{1}$$

where A is a constant. The solution u depends on both spatial and temporal variables, viz. $u = u(x,t)$.

Famous PDEs, two dimension

In two dimension we have $u = u(x,y,t)$. We will, unless otherwise stated, simply use u in our discussion below. Familiar situations which this equation can model are waves on a string, pressure waves, waves on the surface of a fjord or a lake, electromagnetic waves and sound waves to mention a few. For e.g., electromagnetic waves we have the constant $A = c^2$, with c the speed of light. It is rather straightforward to extend this equation to two or three dimension. In two dimensions we have

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = A\frac{\partial^2 u}{\partial t^2},$$

Famous PDEs, diffusion equation

The diffusion equation whose one-dimensional version reads

$$\frac{\partial^2 u}{\partial x^2} = A\frac{\partial u}{\partial t}, \tag{2}$$

and A is in this case called the diffusion constant. It can be used to model a wide selection of diffusion processes, from molecules to the diffusion of heat in a given material.

Famous PDEs, Laplace's equation

Another familiar equation from electrostatics is Laplace's equation, which looks similar to the wave equation in Eq. (1) except that we have set $A = 0$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \tag{3}$$

or if we have a finite electric charge represented by a charge density $\rho(\mathbf{x})$ we have the familiar Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -4\pi\rho(\mathbf{x}). \tag{4}$$

Famous PDEs, Helmholtz' equation

Other famous partial differential equations are the Helmholtz (or eigenvalue) equation, here specialized to two dimensions only

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \lambda u, \tag{5}$$

the linear transport equation (in $2 + 1$ dimensions) familiar from Brownian motion as well

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0, \tag{6}$$

Famous PDEs, Schroedinger's equation in two dimensions

Schroedinger's equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + f(x, y)u = \imath\frac{\partial u}{\partial t}.$$

Famous PDEs, Maxwell's equations

Important systems of linear partial differential equations are the famous Maxwell equations

$$\frac{\partial \mathbf{E}}{\partial t} = \mathrm{curl}\mathbf{B},$$

and

$$-\mathrm{curl}\mathbf{E} = \mathbf{B}$$

and

$$\mathrm{div}\mathbf{E} = \mathrm{div}\mathbf{B} = 0.$$

Famous PDEs, Euler's equations

Similarly, famous systems of non-linear partial differential equations are for example Euler's equations for incompressible, inviscid flow

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\nabla\mathbf{u} = -Dp; \qquad \mathrm{div}\mathbf{u} = 0,$$

with p being the pressure and

$$\nabla = \frac{\partial}{\partial x}e_x + \frac{\partial}{\partial y}e_y,$$

in the two dimensions. The unit vectors are e_x and e_y.

Famous PDEs, the Navier-Stokes' equations

Another example is the set of Navier-Stokes equations for incompressible, viscous flow

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\nabla\mathbf{u} - \Delta\mathbf{u} = -Dp; \qquad \mathrm{div}\mathbf{u} = 0.$$

Famous PDEs, general equation in two dimensions

A general partial differential equation with two given dimensions reads

$$A(x,y)\frac{\partial^2 u}{\partial x^2} + B(x,y)\frac{\partial^2 u}{\partial x\partial y} + C(x,y)\frac{\partial^2 u}{\partial y^2} = F(x,y,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial y}),$$

and if we set

$$B = C = 0,$$

we recover the $1+1$-dimensional diffusion equation which is an example of a so-called parabolic partial differential equation. With

$$B = 0, \qquad AC < 0$$

we get the $2+1$-dim wave equation which is an example of a so-called elliptic PDE, where more generally we have $B^2 > AC$. For $B^2 < AC$ we obtain a so-called hyperbolic PDE, with the Laplace equation in Eq. (3) as one of the classical examples. These equations can all be easily extended to non-linear partial differential equations and $3+1$ dimensional cases.

Laplace's and Poisson's Equations

Laplace's equation reads

$$\nabla^2 u(\mathbf{x}) = u_{xx} + u_{yy} = 0.$$

with possible boundary conditions $u(x, y) = g(x, y)$ on the border $\delta\Omega$. There is no time-dependence. We seek a solution in the region Ω and we choose a quadratic mesh with equally many steps in both directions. We could choose the grid to be rectangular or following polar coordinates r, θ as well. Here we choose equal steps lengths in the x and the y directions. We set

$$h = \Delta x = \Delta y = \frac{L}{n+1},$$

where L is the length of the sides and we have $n + 1$ points in both directions.

Laplace's and Poisson's Equations, discretized version

The discretized version reads

$$u_{xx} \approx \frac{u(x+h, y) - 2u(x, y) + u(x-h, y)}{h^2},$$

and

$$u_{yy} \approx \frac{u(x, y+h) - 2u(x, y) + u(x, y-h)}{h^2},$$

which we rewrite as

$$u_{xx} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2},$$

and

$$u_{yy} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}.$$

Laplace's and Poisson's Equations, final discretized version

Inserting in Laplace's equation we obtain

$$u_{i,j} = \frac{1}{4}\left[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}\right]. \tag{7}$$

This is our final numerical scheme for solving Laplace's equation. Poisson's equation adds only a minor complication to the above equation since in this case we have

$$u_{xx} + u_{yy} = -\rho(x, y),$$

and we need only to add a discretized version of $\rho(\mathbf{x})$ resulting in

$$u_{i,j} = \frac{1}{4}\left[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}\right] + \frac{h^2}{4}\rho_{i,j}. \tag{8}$$

Laplace's and Poisson's Equations, boundary conditions

The boundary condtions read

$$u_{i,0} = g_{i,0} \quad 0 \leq i \leq n+1,$$

$$u_{i,L} = g_{i,0} \quad 0 \leq i \leq n+1,$$

$$u_{0,j} = g_{0,j} \quad 0 \leq j \leq n+1,$$

and

$$u_{L,j} = g_{L,j} \quad 0 \leq j \leq n+1.$$

With $n+1$ mesh points the equations for u result in a system of $(n+1)^2$ linear equations in the $(n+1)^2$ unknown $u_{i,j}$.

Scheme for solving Laplace's (Poisson's) equation

We rewrite Eq. (8)

$$4u_{i,j} = \left[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}\right] - h^2\rho_{i,j} = \Delta_{ij} - \tilde{\rho}_{ij}, \tag{9}$$

where we have defined

$$\Delta_{ij} = \left[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}\right],$$

and

$$\tilde{\rho}_{ij} = h^2\rho_{i,j}.$$

Scheme for solving Laplace's (Poisson's) equation

In order to illustrate how we can transform the last equations into a linear algebra problem of the type $\mathbf{Ax} = \mathbf{w}$, with $\mathbf{A}$ a matrix and $\mathbf{x}$ and $\mathbf{w}$ unknown and known vectors respectively, let us also for the sake of simplicity assume that the number of points $n = 3$. We assume also that $u(x,y) = g(x,y)$ on the border $\delta\Omega$.

The inner values of the function u are then given by

$$\begin{aligned}
4u_{11} - u_{21} - u_{01} - u_{12} - u_{10} &= -\,\tilde{\rho}_{11} \\
4u_{12} - u_{02} - u_{22} - u_{13} - u_{11} &= -\,\tilde{\rho}_{12} \\
4u_{21} - u_{11} - u_{31} - u_{22} - u_{20} &= -\,\tilde{\rho}_{21} \\
4u_{22} - u_{12} - u_{32} - u_{23} - u_{21} &= -\,\tilde{\rho}_{22}.
\end{aligned}$$

Scheme for solving Laplace's (Poisson's) equation

If we isolate on the left-hand side the unknown quantities u_{11}, u_{12}, u_{21} and u_{22}, that is the inner points not constrained by the boundary conditions, we can rewrite the above equations as a matrix $\mathbf{A}$ times an unknown vector $\mathbf{x}$, that is

$$Ax = b,$$

or in more detail

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{21} \\ u_{22} \end{bmatrix} = \begin{bmatrix} u_{01} + u_{10} - \tilde{\rho}_{11} \\ u_{13} + u_{02} - \tilde{\rho}_{12} \\ u_{31} + u_{20} - \tilde{\rho}_{21} \\ u_{32} + u_{23} - \tilde{\rho}_{22} \end{bmatrix}.$$

Scheme for solving Laplace's (Poisson's) equation

The right hand side is constrained by the values at the boundary plus the known function $\tilde{\rho}$. For a two-dimensional equation it is easy to convince oneself that for larger sets of mesh points, we will not have more than five function values for every row of the above matrix. For a problem with $n+1$ mesh points, our matrix $\mathbf{A} \in \mathbb{R}^{(n+1)\times(n+1)}$ leads to $(n-1)\times(n-1)$ unknown function values u_{ij}. This means that, if we fix the endpoints for the two-dimensional case (with a square lattice) at $i(j) = 0$ and $i(j) = n+1$, we have to solve the equations for $1 \geq i(j) len$.

Since the matrix is rather sparse but is not on a tridiagonal form, elimination methods like the LU decomposition discussed, are not very practical. Rather, iterative schemes like Jacobi's method or the Gauss-Seidel are preferred. The above matrix is also always diagonally dominant, a necessary condition for these iterative solvers to converge.

Scheme for solving Laplace's (Poisson's) equation using Jacobi's iterative method

In setting up for example Jacobi's method, it is useful to rewrite the matrix $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{D} + \mathbf{U} + \mathbf{L},$$

with $\mathbf{D}$ being a diagonal matrix with 4 as the only value, $\mathbf{U}$ is an upper triangular matrix and $\mathbf{L}$ a lower triangular matrix. In our case we have

$$\mathbf{D} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix},$$

and

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Scheme for solving Laplace's (Poisson's) equation, with Jacobi's method

We assume now that we have an estimate for the unknown functions u_{11}, u_{12}, u_{21} and u_{22}. We will call this the zeroth value and label it as $u_{11}^{(0)}$, $u_{12}^{(0)}$, $u_{21}^{(0)}$ and $u_{22}^{(0)}$. We can then set up an iterative scheme where the next solution is defined in terms of the previous one as

$$\begin{aligned} u_{11}^{(1)} &= \frac{1}{4}(b_1 - u_{12}^{(0)} - u_{21}^{(0)}) \\ u_{12}^{(1)} &= \frac{1}{4}(b_2 - u_{11}^{(0)} - u_{22}^{(0)}) \\ u_{21}^{(1)} &= \frac{1}{4}(b_3 - u_{11}^{(0)} - u_{22}^{(0)}) \\ u_{22}^{(1)} &= \frac{1}{4}(b_4 - u_{12}^{(0)} - u_{21}^{(0)}), \end{aligned}$$

where we have defined the vector

$$\mathbf{b} = \begin{bmatrix} u_{01} + u_{10} - \tilde{\rho}_{11} \\ u_{13} + u_{02} - \tilde{\rho}_{12} \\ u_{31} + u_{20} - \tilde{\rho}_{21} \\ u_{32} + u_{23} - \tilde{\rho}_{22} \end{bmatrix}.$$

Scheme for solving Laplace's (Poisson's) equation, final rewrite

We can rewrite the equations in a more compact form in terms of the matrices $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ as, after $r+1$ iterations,

$$\mathbf{x}^{(r+1)} = \mathbf{D}^{-1}\left(\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}^{(r)}\right), \tag{10}$$

where the unknown functions are now defined in terms of

$$\mathbf{x} = \begin{bmatrix} u_{11} \\ u_{12} \\ u_{21} \\ u_{22} \end{bmatrix}.$$

If we wish to implement Gauss-Seidel's algorithm, the set of equations to solve are then given by

$$\mathbf{x}^{(r+1)} = -(\mathbf{D} + \mathbf{L})^{-1}\left(\mathbf{b} - \mathbf{U}\mathbf{x}^{(r)}\right), \tag{11}$$

or alternatively as

$$\mathbf{x}^{(r+1)} = \mathbf{D}^{-1}\left(\mathbf{b} - \mathbf{L}\mathbf{x}^{(r+1)} - \mathbf{U}\mathbf{x}^{(r)}\right).$$

Jacobi Algorithm for solving Laplace's Equation

It is thus fairly straightforward to extend this equation to the three-dimensional case. Whether we solve Eq. (7) or Eq. (8), the solution strategy remains the same. We know the values of u at $i = 0$ or $i = n + 1$ and at $j = 0$ or $j = n + 1$ but we cannot start at one of the boundaries and work our way into and across the system since Eq. (7) requires the knowledge of u at all of the neighbouring points in order to calculate u at any given point.

Jacobi Algorithm for solving Laplace's Equation

The way we solve these equations is based on an iterative scheme based on the Jacobi method or the Gauss-Seidel method or the relaxation methods.

Implementing Jacobi's method is rather simple. We start with an initial guess for $u_{i,j}^{(0)}$ where all values are known. To obtain a new solution we solve Eq. (7) or Eq. (8) in order to obtain a new solution $u_{i,j}^{(1)}$. Most likely this solution will not be a solution to Eq. (7). This solution is in turn used to obtain a new and improved $u_{i,j}^{(2)}$. We continue this process till we obtain a result which satisfies some specific convergence criterion.

Jacobi Algorithm for solving Laplace's Equation, the algorithm

Summarized, this algorithm reads

1. Make an initial guess for $u_{i,j}$ at all interior points (i, j) for all $i = 1 : n$ and $j = 1 : n$
2. Use Eq. (7) to compute u^m at all interior points (i, j). The index m stands for iteration number m.
3. Stop if prescribed convergence threshold is reached, otherwise continue to the next step.
4. Update the new value of u for the given iteration
5. Go to step 2

Jacobi Algorithm for solving Laplace's Equation, simple example

A simple example may help in understanding this method. We consider a condensator with parallel plates separated at a distance L resulting in for example the voltage differences $u(x,0) = 200sin(2\pi x/L)$ and $u(x,1) = -200sin(2\pi x/L)$. These are our boundary conditions and we ask what is the voltage u between the plates?

Jacobi Algorithm for solving Laplace's Equation, to observe

The important part of the algorithm is applied in the function which sets up the two-dimensional Laplace equation. There we have a while statement which tests the difference between the temporary vector and the solution $u_{i,j}$. Moreover, we have fixed the number of iterations to a given maximum. We need also to provide a convergence tolerance. In the above program example we have fixed this to be 0.00001. Depending on the type of applications one may have to change both the number of maximum iterations and the tolerance.

Python code for solving the two-dimensional Laplace equation

The following Python code sets up and solves the Laplace equation in two dimensions.

```
# Solves the 2d Laplace equation using relaxation method

import numpy, math

def relax(A, maxsteps, convergence):
    """
    Relaxes the matrix A until the sum of the absolute differences
    between the previous step and the next step (divided by the number of
    elements in A) is below convergence, or maxsteps is reached.

    Input:
     - A: matrix to relax
     - maxsteps, convergence: Convergence criterions

    Output:
     - A is relaxed when this method returns
    """

    iterations = 0
    diff = convergence +1

    Nx = A.shape[1]
    Ny = A.shape[0]

    while iterations < maxsteps and diff > convergence:
        #Loop over all *INNER* points and relax
        Atemp = A.copy()
        diff = 0.0
        for y in xrange(1,Ny-1):
            for x in xrange(1,Ny-1):
                A[y,x] = 0.25*(Atemp[y,x+1]+Atemp[y,x-1]+Atemp[y+1,x]+Atemp[y-1,x])
                diff  += math.fabs(A[y,x] - Atemp[y,x])

        diff /=(Nx*Ny)
        iterations += 1
        print "Iteration #", iterations, ", diff =", diff;


def boundary(A,x,y):
    """
    Set up boundary conditions

    Input:
     - A: Matrix to set boundaries on
     - x: Array where x[i] = hx*i, x[last_element] = Lx
     - y: Eqivalent array for y

    Output:
     - A is initialized in-place (when this method returns)
    """

    #Boundaries implemented (condensator with plates at y={0,Lx}, DeltaV = 200):
    # A(x,0)   =  100*sin(2*pi*x/Lx)
    # A(x,Ly) = -100*sin(2*pi*x/Lx)
    # A(0,y)   = 0
    # A(Lx,y) = 0

    Nx = A.shape[1]
    Ny = A.shape[0]
    Lx = x[Nx-1] #They *SHOULD* have same sizes!
    Ly = x[Nx-1]


    A[:,0]     =   100*numpy.sin(math.pi*x/Lx)
    A[:,Nx-1] = - 100*numpy.sin(math.pi*x/Lx)
    A[0,:]     = 0.0
    A[Ny-1,:] = 0.0


#Main program

import sys

# Input parameters
Nx      = 100
Ny      = 100
maxiter = 1000

x = numpy.linspace(0,1,num=Nx+2) #Also include edges
y = numpy.linspace(0,1,num=Ny+2)
A = numpy.zeros((Nx+2,Ny+2))

boundary(A,x,y)
#Remember: as solution "creeps" in from the edges,
#number of steps MUST AT LEAST be equal to
#number of inner meshpoints/2 (unless you have a better
#estimate for the solution than zeros() )
relax(A,maxiter,0.00001)
```

```
# To do: add visualization
```

Jacobi's algorithm extended to the diffusion equation in two dimensions

Let us know implement the implicit scheme and show how we can extend the previous algorithm for solving Laplace's or Poisson's equations to the diffusion equation as well. As the reader will notice, this simply implies a slight redefinition of the vector **b** defined in Eq. (10).

To see this, let us first set up the diffusion in two spatial dimensions, with boundary and initial conditions. The $2+1$-dimensional diffusion equation (with dimensionless variables) reads for a function $u = u(x, y, t)$

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right).$$

Jacobi's algorithm extended to the diffusion equation in two dimensions

We assume that we have a square lattice of length L with equally many mesh points in the x and y directions. Setting the diffusion constant $D = 1$ and using the shorthand notation $u_{xx} = \partial^2 u/\partial x^2$ etc for the second derivatives and $u_t = \partial u/\partial t$ for the time derivative, we have, with a given set of boundary and initial conditions,

$$\begin{array}{cc}
u_t = u_{xx} + u_{yy} & x, y \in (0, L), t > 0 \\
u(x, y, 0) = g(x, y) & x, y \in (0, L) \\
u(0, y, t) = u(L, y, t) = u(x, 0, t) = u(x, L, t)0 & t > 0
\end{array}$$

Jacobi's algorithm extended to the diffusion equation in two dimensions, discretizing

We discretize again position and time, and use the following approximation for the second derivatives

$$u_{xx} \approx \frac{u(x+h, y, t) - 2u(x, y, t) + u(x-h, y, t)}{h^2},$$

which we rewrite as, in its discretized version,

$$u_{xx} \approx \frac{u^l_{i+1,j} - 2u^l_{i,j} + u^l_{i-1,j}}{h^2},$$

where $x_i = x_0 + ih$, $y_j = y_0 + jh$ and $t_l = t_0 + l\Delta t$, with $h = L/(n+1)$ and Δt the time step.

Jacobi's algorithm extended to the diffusion equation in two dimensions, the second derivative

The second derivative with respect to y reads

$$u_{yy} \approx \frac{u^l_{i,j+1} - 2u^l_{i,j} + u^l_{i,j-1}}{h^2}.$$

We use now the so-called backward going Euler formula for the first derivative in time. In its discretized form we have

$$u_t \approx \frac{u^l_{i,j} - u^{l-1}_{i,j}}{\Delta t},$$

resulting in

$$u^l_{i,j} + 4\alpha u^l_{i,j} - \alpha\left[u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1}\right] = u^{l-1}_{i,j},$$

where the right hand side is the only known term, since starting with $t = t_0$, the right hand side is entirely determined by the boundary and initial conditions. We have $\alpha = \Delta t/h^2$.

Jacobi's algorithm extended to the diffusion equation in two dimensions

For future time steps, only the boundary values are determined and we need to solve the equations for the interior part in an iterative way similar to what was done for Laplace's or Poisson's equations. To see this, we rewrite the previous equation as

$$u^l_{i,j} = \frac{1}{1+4\alpha}\left[\alpha(u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1}) + u^{l-1}_{i,j}\right],$$

or in a more compact form as

$$u^l_{i,j} = \frac{1}{1+4\alpha}\left[\alpha\Delta^l_{ij} + u^{l-1}_{i,j}\right], \qquad (12)$$

with $\Delta^l_{ij} = \left[u^l_{i,j+1} + u^l_{i,j-1} + u^l_{i+1,j} + u^l_{i-1,j}\right]$. This equation has essentially the same structure as Eq. (9), except that the function ρ_{ij} is replaced by the solution at a previous time step $l-1$. Furthermore, the diagonal matrix elements are now given by $1+4\alpha$, while the non-zero non-diagonal matrix elements equal α. This matrix is also positive definite, meaning in turn that iterative schemes like the Jacobi or the Gauss-Seidel methods will converge to the desired solution after a given number of iterations.

Wave Equation in two Dimensions

The $1+1$-dimensional wave equation reads

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2},$$

with $u = u(x,t)$ and we have assumed that we operate with dimensionless variables. Possible boundary and initial conditions with $L = 1$ are

$$\begin{array}{cc} u_{xx} = u_{tt} & x \in (0,1), t > 0 \\ u(x,0) = g(x) & x \in (0,1) \\ u(0,t) = u(1,t) = 0 & t > 0 \\ \partial u/\partial t|_{t=0} = 0 & x \in (0,1) \end{array}.$$

Wave Equation in two Dimensions, discretizing

We discretize again time and position,

$$u_{xx} \approx \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2},$$

and

$$u_{tt} \approx \frac{u(x,t+\Delta t) - 2u(x,t) + u(x,t-\Delta t)}{\Delta t^2},$$

which we rewrite as

$$u_{xx} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2},$$

and

$$u_{tt} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta t^2},$$

resulting in

$$u_{i,j+1} = 2u_{i,j} - u_{i,j-1} + \frac{\Delta t^2}{\Delta x^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right). \tag{13}$$

Wave Equation in two Dimensions

If we assume that all values at times $t = j$ and $t = j-1$ are known, the only unknown variable is $u_{i,j+1}$ and the last equation yields thus an explicit scheme for updating this quantity. We have thus an explicit finite difference scheme for computing the wave function u. The only additional complication in our case is the initial condition given by the first derivative in time, namely $\partial u/\partial t|_{t=0} = 0$. The discretized version of this first derivative is given by

$$u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j - \Delta t)}{2\Delta t},$$

and at $t = 0$ it reduces to

$$u_t \approx \frac{u_{i,+1} - u_{i,-1}}{2\Delta t} = 0,$$

implying that $u_{i,+1} = u_{i,-1}$.

Wave Equation in two Dimensions

If we insert this condition in Eq. (13) we arrive at a special formula for the first time step

$$u_{i,1} = u_{i,0} + \frac{\Delta t^2}{2\Delta x^2}\left(u_{i+1,0} - 2u_{i,0} + u_{i-1,0}\right). \tag{14}$$

We need seemingly two different equations, one for the first time step given by Eq. (14) and one for all other time-steps given by Eq. (13). However, it suffices to use Eq. (13) for all times as long as we provide $u(i,-1)$ using

$$u_{i,-1} = u_{i,0} + \frac{\Delta t^2}{2\Delta x^2}\left(u_{i+1,0} - 2u_{i,0} + u_{i-1,0}\right),$$

in our setup of the initial conditions.

Wave Equation in two Dimensions

The situation is rather similar for the $2+1$-dimensional case, except that we now need to discretize the spatial y-coordinate as well. Our equations will now depend on three variables whose discretized versions are now

$$\begin{array}{ll} t_l = l\Delta t & l \geq 0 \\ x_i = i\Delta x & 0 \leq i \leq n_x \\ y_j = j\Delta y & 0 \leq j \leq n_y \end{array},$$

and we will let $\Delta x = \Delta y = h$ and $n_x = n_y$ for the sake of simplicity. The equation with initial and boundary conditions reads now

$$\begin{array}{cc} u_{xx} + u_{yy} = u_{tt} & x, y \in (0,1), t > 0 \\ u(x,y,0) = g(x,y) & x, y \in (0,1) \\ u(0,0,t) = u(1,1,t) = 0 & t > 0 \\ \partial u/\partial t|_{t=0} = 0 & x, y \in (0,1) \end{array}.$$

Wave Equation in two Dimensions

We have now the following discretized partial derivatives

$$u_{xx} \approx \frac{u^l_{i+1,j} - 2u^l_{i,j} + u^l_{i-1,j}}{h^2},$$

and

$$u_{yy} \approx \frac{u^l_{i,j+1} - 2u^l_{i,j} + u^l_{i,j-1}}{h^2},$$

and

$$u_{tt} \approx \frac{u^{l+1}_{i,j} - 2u^l_{i,j} + u^{l-1}_{i,j}}{\Delta t^2},$$

which we merge into the discretized $2+1$-dimensional wave equation as

$$u^{l+1}_{i,j} = 2u^l_{i,j} - u^{l-1}_{i,j} + \frac{\Delta t^2}{h^2}\left(u^l_{i+1,j} - 4u^l_{i,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1}\right), \quad (15)$$

where again we have an explicit scheme with $u^{l+1}_{i,j}$ as the only unknown quantity.

Wave Equation in two Dimensions

It is easy to account for different step lengths for x and y. The partial derivative is treated in much the same way as for the one-dimensional case, except that we now have an additional index due to the extra spatial dimension, viz., we need to compute $u^{-1}_{i,j}$ through

$$u^{-1}_{i,j} = u^0_{i,j} + \frac{\Delta t}{2h^2}\left(u^0_{i+1,j} - 4u^0_{i,j} + u^0_{i-1,j} + u^0_{i,j+1} + u^0_{i,j-1}\right),$$

in our setup of the initial conditions.

Analytical Solution for the two-dimensional wave equation

We develop here the closed-form solution for the $2+1$ dimensional wave equation with the following boundary and initial conditions

$$\begin{array}{cc} c^2(u_{xx} + u_{yy}) = u_{tt} & x, y \in (0, L), t > 0 \\ u(x, y, 0) = f(x, y) & x, y \in (0, L) \\ u(0, 0, t) = u(L, L, t) = 0 & t > 0 \\ \partial u/\partial t|_{t=0} = g(x, y) & x, y \in (0, L) \end{array}.$$

Analytical Solution for the two-dimensional wave equation, first step

Our first step is to make the ansatz

$$u(x, y, t) = F(x, y)G(t),$$

resulting in the equation

$$FG_{tt} = c^2(F_{xx}G + F_{yy}G),$$

or

$$\frac{G_{tt}}{c^2G} = \frac{1}{F}(F_{xx} + F_{yy}) = -\nu^2.$$

Analytical Solution for the two-dimensional wave equation,

The lhs and rhs are independent of each other and we obtain two differential equations

$$F_{xx} + F_{yy} + F\nu^2 = 0,$$

and

$$G_{tt} + Gc^2\nu^2 = G_{tt} + G\lambda^2 = 0,$$

with $\lambda = c\nu$. We can in turn make the following ansatz for the x and y dependent part

$$F(x, y) = H(x)Q(y),$$

which results in

$$\frac{1}{H}H_{xx} = -\frac{1}{Q}(Q_{yy} + Q\nu^2) = -\kappa^2.$$

Analytical Solution for the two-dimensional wave equation, separation of variables

Since the lhs and rhs are again independent of each other, we can separate the latter equation into two independent equations, one for x and one for y, namely

$$H_{xx} + \kappa^2 H = 0,$$

and

$$Q_{yy} + \rho^2 Q = 0,$$

with $\rho^2 = \nu^2 - \kappa^2$.

Analytical Solution for the two-dimensional wave equation, separation of variables

The second step is to solve these differential equations, which all have trigonometric functions as solutions, viz.

$$H(x) = A\cos(\kappa x) + B\sin(\kappa x),$$

and

$$Q(y) = C\cos(\rho y) + D\sin(\rho y).$$

Analytical Solution for the two-dimensional wave equation, boundary conditions

The boundary conditions require that $F(x,y) = H(x)Q(y)$ are zero at the boundaries, meaning that $H(0) = H(L) = Q(0) = Q(L) = 0$. This yields the solutions

$$H_m(x) = \sin(\frac{m\pi x}{L}) \qquad Q_n(y) = \sin(\frac{n\pi y}{L}),$$

or

$$F_{mn}(x,y) = \sin(\frac{m\pi x}{L})\sin(\frac{n\pi y}{L}).$$

With $\rho^2 = \nu^2 - \kappa^2$ and $\lambda = c\nu$ we have an eigenspectrum $\lambda = c\sqrt{\kappa^2+\rho^2}$ or $\lambda_{mn} = c\pi/L\sqrt{m^2+n^2}$.

Analytical Solution for the two-dimensional wave equation, separation of variables and solutions

The solution for G is

$$G_{mn}(t) = B_{mn}\cos(\lambda_{mn}t) + D_{mn}\sin(\lambda_{mn}t),$$

with the general solution of the form

$$u(x,y,t) = \sum_{mn=1}^{\infty} u_{mn}(x,y,t) = \sum_{mn=1}^{\infty} F_{mn}(x,y)G_{mn}(t).$$

Analytical Solution for the two-dimensional wave equation, final steps

The final step is to determine the coefficients B_{mn} and D_{mn} from the Fourier coefficients. The equations for these are determined by the initial conditions $u(x,y,0) = f(x,y)$ and $\partial u/\partial t|_{t=0} = g(x,y)$. The final expressions are

$$B_{mn} = \frac{2}{L}\int_0^L\int_0^L dxdy f(x,y)\sin(\frac{m\pi x}{L})\sin(\frac{n\pi y}{L}),$$

and

$$D_{mn} = \frac{2}{L}\int_0^L\int_0^L dxdy g(x,y)\sin(\frac{m\pi x}{L})\sin(\frac{n\pi y}{L}).$$

Inserting the particular functional forms of $f(x,y)$ and $g(x,y)$ one obtains the final closed-form expressions.

Python code for solving the two-dimensional wave equation

The following Python code sets up and solves the two-dimensional wave equation for all three methods discussed.

```
#Program which solves the 2+1-dimensional wave equation by a finite difference scheme

from numpy import *
#Define the grid
N = 31
h = 1.0 / (N-1)
dt = .0005
t_steps = 10000
x,y = ndgrid(linspace(0,1,N),linspace(0,1,N),sparse=False)

alpha = dt**2 / h**2

#Initial conditions with du/dt = 0
u = sin(x*pi)*cos(y*pi-pi/2)
u_old = zeros(u.shape,type(u[0,0]))
for i in xrange(1,N-1):
    for j in xrange(1,N-1):
        u_old[i,j] = u[i,j] + (alpha/2)*(u[i+1,j] - 4*u[i,j] + u[i-1,j] + u[i,j+1] + u[i,j-1])
u_new = zeros(u.shape,type(u[0,0]))

#We don't necessarily want to plot every time step. We plot every n'th step where
n = 100
plotnr = 0

#Iteration over time steps
for k in xrange(t_steps):
    for i in xrange(1,N-1): #1 - N-2 because we don't want to change the boundaries
        for j in xrange(1,N-1):
            u_new[i,j] = 2*u[i,j] - u_old[i,j] + alpha*(u[i+1,j] - 4*u[i,j] + u[i-1,j] + u[i,j+1]

    #Prepare for next time step by manipulating pointers
    temp = u_new
    u_new = u_old
    u_old = u
    u = temp
#To do: Make movie
```