Physics 514 – Classical Few-Body Problems and Partial Differential Equations

Emanuel Gull

September 8, 2016

0.1 Sources and Literature

Most of this lecture closely follow a lecture given by M. Troyer at ETH Zurich in the summer of 2005/2006. Recommended books vary from chapter to chapter. Here are some general recommendations:

1. D. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge University Press (2000), ISBN 0521653665

2. J. M. Thijssen, *Computational Physics*, Cambridge University Press (1999), ISBN 013367230

3. H. Gould and J. Tobochnik, *An Introduction to Computer Simulation Methods*, 2nd edition, Addison Wesley (1996), ISBN 00201506041

4. Werner Krauth, *Statistical Mechanics: Algorithms and Computations*, Oxford university press (2006), 0198515367

Chapter 1

Classical Few-Body Problems

1.1 Ordinary Differential Equations

1.1.1 The Euler Method

The most simple ordinary differential equations are *initial value problems* for *first order* ordinary differential equations, which have the form

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, t) \tag{1.1}$$

$$\mathbf{y}(t_0) = \mathbf{y}_0 \tag{1.2}$$

$\mathbf{y}$ may be a vector and $\mathbf{f}$ a vector-valued function. A range of applications are based on the solution of ordinary differential equations, e.g. the radioactive decay:

$$\frac{dN}{dt} = -\lambda N, \tag{1.3}$$

where N is the number of particles and λ the radioactive decay constant, or "coffee cooling":

$$\frac{dT}{dt} = -\gamma(T - T_{\text{room}}), \tag{1.4}$$

where T is the temperature of a cup of coffee, T_{room} the room temperature, and γ the cooling rate.

For these problems an analytical solution can easily be found:

$$T(t) = T_{\text{room}} + (T(0) - T_{\text{room}}) \exp(-\gamma t). \tag{1.5}$$

In more complicated cases an analytic solution is not readily available and a numerical solution is needed. We obtain it numerically by performing a Taylor expansion of Eq. 1.1:

$$y(t_0 + \Delta t) = y(t_0) + \Delta t \frac{dy}{dt} = y_0 + \Delta t f(y_0, t_0) + O(\Delta t^2). \tag{1.6}$$

We then iterate this equation and introduce the notation $t_n = t_0 + n\Delta t$ and $y_n = y(t_n)$ to obtain the Euler algorithm:

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) + O(\Delta t^2) \tag{1.7}$$

This provides a general scheme to integrate any first order ordinary differential equation.

Exercise: Implement the Euler method and run it for the coffee cooling problem; examine convergence as a function of step size.

1.1.2 Higher order ODE integrators

Order of integration methods

The truncation of the Taylor expansion after the first term in the Euler method introduces an error of order $O(\Delta t^2)$ at each time step. In any simulation we need to control this error to ensure that the final result is not influenced by the size of the time step. We have two options if we want to reduce this numerical error: either we choose a smaller value of Δt in the Euler method, or we choose a *higher order method*.

A method which introduces an error of order $O(t^n)$ in a single time step is said to be locally of n-th order. Iterating a locally n-th order method over a fixed time interval T the discretization errors add up: we need to perform $T/\Delta t$ time steps and at each time step we pick up an error of order $O(\Delta t^n)$. The total error over time T is then:

$$\frac{T}{\Delta t} O(\Delta t^n) = O(\Delta t^{n-1}). \tag{1.8}$$

This means that a locally n-th order method is *globally* of order $n - 1$.

The Euler method is locally second order and *globally* first order, it is therefore usually called a *first order* method.

Predictor-Corrector methods

One straightforward idea to improve the Euler method is to evaluate the derivative not only at the initial time t_n but also at the next time t_{n+1}:

$$y_{n+1} \approx y_n + \frac{\Delta t}{2} \left[f(y_n, t_n) + f(y_{n+1}, t_{n+1}) \right] \tag{1.9}$$

of course the value y_{n+1} that enters f on the right hand side is not known. Instead of solving the equation numerically, we first *predict* a rough estimate $\tilde{y}_{n+1}$ using the Euler method:

$$\tilde{y}_{n+1} \approx \tilde{y}_n + \Delta t f(y_n, t_n) \tag{1.10}$$

We then use this estimate to *correct* the Euler estimate in a second step by using $\tilde{y}_{n+1}$ instead of the unknown y_{n+1} in Eq. 1.9:

$$y_{n+1} \sim y_n + \frac{\Delta t}{2} \left[f(y_n, t_n) + f(\tilde{y}_{n+1}, t_{n+1}) \right] \tag{1.11}$$

This corrector step can then be iterated using 1.9 with the new estimate until the estimate converges.

There are several variants of this method in common use: The *forward Euler* method is the one of Sec. 1.1.1. The *backward Euler* method uses only the estimated derivative at y_{n+1}, and the midpoint rule the averaged derivative of this section. The Backward Euler method is a so-called implicit method, meaning that a further iteration step is required to obtain the value y_{n+1}. In general implicit methods are more stable (see, e.g., L. Sander's lecture or your favorite numerics textbook). The order of the midpoint rule is globally quadratic (show why!), forward and backward Euler are globally of first order.

1.1.3 Runge Kutta methods

The Runge-Kutta methods are families of systematic higher order improvements over the Euler method. The key idea is to evaluate the derivative dy/dt not only at the end points t_n and t_{n+1} but also at intermediate points. Here is an example:

$$y_{n+1} = y_n + \Delta t f(t_n + \frac{\Delta t}{2}, y(t_n + \frac{\Delta t}{2})) + O(\Delta t^3). \tag{1.12}$$

The unknown solution $y(t_n + \Delta t/2)$ is approximated by an Euler step, giving the second order Runge Kutta algorithm:

$$k_1 = \Delta t f(t_n, y_n) \tag{1.13}$$

$$k_2 = \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \tag{1.14}$$

$$y_{n+1} = y_n + k_2 + O(\Delta t^3) \tag{1.15}$$

The general ansatz is

$$y_{n+1} = y_n + \sum_{i=1}^{N} \alpha_i k_i, \tag{1.16}$$

where the approximations k_i are given by

$$k_i = \Delta t f(y_n + \sum_{j=1}^{N-1} \nu_{ij} k_i, t_n + \sum_{j=1}^{N-1} \nu_{ij} \Delta t) \tag{1.17}$$

The constants α_i and ν_{ij} are chosen to obtain an N-th order method. Note that this choice is usually not unique.

The fourth-order Runge Kutta method is the most widely used ODE integrator:

$$k_1 = \Delta t f(t_n, y_n) \tag{1.18}$$
$$k_2 = \Delta t f(t_n + \Delta t/2, y_n + k_1/2) \tag{1.19}$$
$$k_3 = \Delta t f(t_n + \Delta t/2, y_n + k_2/2) \tag{1.20}$$
$$k_4 = \Delta t f(t_n + \Delta t, y_n + k_3) \tag{1.21}$$
$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5) \tag{1.22}$$

Again, there are variants of this: lower and larger order Runge Kutta methods, implicit and explicit ones, etc. Your favorite numerical mathematics textbook will provide more information and a derivation of the method and its convergence order.

1.2 Classical equations of motion

The most common ODEs you will encounter are Newton's equations of motion for N point particles with position x_i, mass m_i and velocity v_i:

$$m_i \frac{dv_i}{dt} = F_i(t, x_1, \cdots, x_N; v_1, \cdots, v_N) \tag{1.23}$$
$$\frac{dx_i}{dt} = v_i \tag{1.24}$$

The simplest method is again the forward Euler method:

$$v_{n+1} = v_n + a_n \Delta t \tag{1.25}$$
$$x_{n+1} = x_n + v_n \Delta t \tag{1.26}$$

The Euler method is unstable for oscillating systems, it should therefore be avoided in practice. It is surprisingly simple to stabilize the solution: using a backward difference instead of the forward difference, *i.e.* a Taylor expansion around x_{n+1} instead of x_n we obtain the implicit stable backward Euler method:

$$v_{n+1} = v_n + a_{n+1} \Delta t \tag{1.27}$$
$$x_{n+1} = x_n + v_{n+1} \Delta t, \tag{1.28}$$

where the new velocity v_{n+1} and accelerations are used in calculating the positions x_{n+1}. Note that in v and x at the new positions appear both on the left and on the right side of the equation. The propagation step therefore involves solving an equation, either explicitly or iteratively.

Many variants of these simple ODE integrators exist. Depending on your differential equation we can even construct integrators that are tailored to the

equation, see later in the chapter on the Schrödinger equation. Here are three more methods that are commonly used:

Almost as simple as the Euler methods but of second order in the positions the midpoint method:

$$v_{n+1} = v_n + a_n \Delta t \tag{1.29}$$

$$x_{n+1} = x_n + \frac{1}{2}(v_n + v_{n+1})\Delta t \tag{1.30}$$

The leap-frog method is often used for integrating classical equations of motion. It evaluates positions and velocities at different locations:

$$v_{n+1/2} = v_{n-1/2} + a_n \Delta t \tag{1.31}$$

$$x_{n+1} = x_n + v_{n+1/2}\Delta t. \tag{1.32}$$

as $v_{n-1/2}$ is not known at the first step, we have to jump-start the method with an Euler step (the method is not 'self-starting'):

$$v_{1/2} = v_0 + 1/2 a_0 \Delta t \tag{1.33}$$

It is interesting because it is fully time-reversal invariant, and therefore conserves total energy.

If the forces are velocity-dependent the second-order Euler-Richardson algorithm can be used:

$$a_{n+1/2} = a\left(x_n + \frac{1}{2}v_n\Delta t, v_n + \frac{1}{2}a_n\Delta t, t_n + \frac{1}{2}\Delta t\right) \tag{1.34}$$

$$v_{n+1} = v_n + a_{n+1/2}\Delta t \tag{1.35}$$

$$x_{n+1} = x_n + v_n\Delta t + \frac{1}{2}a_{n+1/2}\Delta t^2 \tag{1.36}$$

This algorithm evaluates the velocities and positions at times $n, n+1, \ldots$ and the forces or accelerations at time $n - 1/2, n + 1/2, \ldots$.

The most commonly used algorithm for classical equations of motion in molecular mechanics is the *velocity Verlet* algorithm, which is of second order:

$$x_{n+1} = x_n + v_n\Delta t + \frac{1}{2}a_n\Delta t^2 \tag{1.37}$$

$$v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\Delta t \tag{1.38}$$

As opposed to the leapfrog algorithm, the velocities and positions are known at the same time.

1.3 Boundary value problems and shooting

So far we have looked at *initial value problems*: given was the differential equation and a starting *initial* value (here for velocity and position) at time zero. Another type of problems is the boundary value problem where we specify an initial and a final condition instead of two initial conditions. For example:

- We launch a rocket from the surface of the earth and want it to enter space (defined as an altitude of 100km) after one hour. Here the initial and final positions are specified and the question is to estimate the required power of the rocket engine.
- We fire a cannon ball from Randall lab and want it to hit the football stadium. Known are the initial and final positions as well as the initial speed. The question is to determine the angle of the cannon barrel.

To solve this problem we

1. specify the initial position $x(0) = x_0$ and the final position $x(t) = x_f$
2. formulate the position as a function of time t and external parameter (barrel angle, power of the rocket) α as $x(t, \alpha)$
3. numerically solve the equation $x(t, \alpha) = x_f$.

To solve this we need to know about root solvers, i.e. how to find zeros of non-linear functions numerically.

1.4 Numerical Root solvers

The purpose of a root solver is to find a solution (a *root*) to the equation

$$\mathbf{f}(\mathbf{x}) = 0 \tag{1.39}$$

Numerical root solvers are well optimized and readily available in libraries. These routines should usually be preferred to hand-coded root solvers.

1.4.1 Bisection Method and Regula Falsi

Both the bisection method and the Regula Falsi require two starting values x_0 and x_1 surrounding the root, with $f(x_0) < 0$ and $f(x_1) > 0$ so that under the assumption of a continuous function f there exists at least one root between x_0 and x_1. The always find a solution (in contrast, e.g., to the Newton and Secant methods)

The *bisection method* performs the following iteration:

1. define a mid-point $x_m = (x_0 + x_1)/2$
2. if $sign(f(x_m)) = sign(f(x_0))$ replace $x_0 \leftarrow x_m$, otherwise replace $x_1 \leftarrow x_m$
3. iterate until a root is found.

The *Regula Falsi method* is similar:

1. Estimate the function f by a straight line from x_0 to x_1 and calculate the root of this linearized function:
$$x_2 = (f(x_0)x_1 - f(x_1)x_0)/(f(x_1) - f(x_0)) \tag{1.40}$$
2. if $signf(x_2) = signf(x_0)$ replace $x_0 \leftarrow x_2$, otherwise replace $x_1 \leftarrow x_2$

1.4.2 Newton and Secant algorithm

The newton method is one of the best known root solvers. It is not guaranteed to converge. The key idea is to start from a guess x_0, linearize the equation around that guess,

$$f(x_0) + (x - x_0)f'(x_0) = 0 \tag{1.41}$$

and solve this linearized equation to obtain a better estimate x_1. Iterating this procedure we obtain the *Newton method*:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1.42}$$

If the derivative f' is not known analytically, as is the case in our shooting problems, we can estimate it from the difference of the last two points:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{1.43}$$

Substituting this into the Newton method Eq. 1.42 we obtain the *Secant* method

$$x_{n+1} = x_n - (x_n - x_{n+1})\frac{f(x_n)}{f(x_n) - f(x_{n+1})}. \tag{1.44}$$

In more than one dimension the Newton method generalizes to:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - A^{-1}\mathbf{f}(\mathbf{x}) \tag{1.45}$$

If the derivatives A are not known analytically they can be estimated through finite differences:

$$A_{ij}(x) = \frac{f_i(\mathbf{x} + h_j\mathbf{e}_j) - f_i(\mathbf{x})}{h_j} \tag{1.46}$$

1.4.3 Optimizing a function

Root solvers can also be used to find an extremum (minimum or maximum) of a function $f(x)$, by looking at a root of the derivative:

$$\frac{df(x)}{dx} = 0 \tag{1.47}$$

These methods are not very efficient in higher dimensions.

A more efficient algorithm is this variant of the *steepest descent* method: we compute the gradient ∇f and use a one-dimensional minimizer to find the point which minimizes $f(\mathbf{x}_n + \lambda\nabla f(\mathbf{x_n}))$ as a function of λ. This determines the next guess:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda\nabla f(\mathbf{x}_n) \tag{1.48}$$

Imagine a situation where the function has a narrow valley, extended in one direction. The iteration will converge quickly in the direction perpendicular to the valley but very slowly in the direction along it. A method that locally approximates the function as a parabola will solve this problem in one step. The *Conjugate Gradient* or *CG* method implements this procedure. Again, as with ODE integrators: use reliable and fast library routines to implement this function.

A word of warning: All of these minimizers will only find local minima. If there is a global minimum cannot be decided without further knowledge of the function. It is therefore often necessary to start the iteration from a range of different initial starting guesses to check that no lower minimum can be found.

In the physics of glassy systems (more later) the energy landscape is rough and other methods, e.g. the Wang Landau method or parallel tempering methods, are used to tunnel among the various valleys in the energy landscape.

Chapter 2

Partial Differential Equations

This chapter will present algorithms for the solution of some simple but widely used partial differential equations (PDEs), and will discuss approaches for general partial differential equations. PDEs are an active research topic in numerical mathematics. Interested students should therefore consider taking a course in numerical mathematics at the department of mathematics.

2.1 Finite Differences

As in the solution of ordinary differential equations the first step in the solution of a PDE is to discretize space and time and to replace differentials by differences, using the notation $x_n = n\Delta x$. We already saw that a first order differential $\partial f/\partial x$ can be approximated in first order by

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_n)}{\Delta x} + O(\Delta x) = \frac{f(x_n) - f(x_{n-1})}{\Delta x} + O(\Delta x) \tag{2.1}$$

or to second order by the symmetric version

$$\frac{\partial f}{\partial x} = \frac{f(x_{n+1}) - f(x_{n-1})}{2\Delta x} + O(\Delta x^2) \tag{2.2}$$

We can get a second order derivative from these first order derivatives as

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x_{n+1}) + f(x_{n-1}) - 2f(x_n)}{\Delta x^2} + O(\Delta x^2) \tag{2.3}$$

for a general approximation for an arbitrary derivative to any given order use the ansatz

$$\sum_{k=-1}^{l} a_k f(x_{n+k}), \tag{2.4}$$

insert the Taylor expansion

$$f(x_{n+k}) = f(x_n) + \Delta x f'(x_n) + \frac{\Delta x^2}{2} f''(x_n) + \frac{\Delta x^3}{6} f'''(x_n) + \frac{\Delta x^4}{4!} f^{(4)}(x_n) + \cdots \tag{2.5}$$

and choose the values a_k such that all terms but the desired derivative vanish.

For example: to get the fourth order estimator for the second order derivative:

$$\frac{\partial^2 f}{\partial x^2} = \frac{-f(x_{n-2}) + 16f(x_{n-1}) - 30f(x_n) + 16f(x_{n+1}) - f(x_{n+2})}{12\Delta x^2} + O(\Delta x^4), \tag{2.6}$$

and to get the second order estimator for the third order derivative:

$$\frac{\partial^3 f}{\partial x^3} = \frac{-f(x_{n-2}) + 2f(x_{n-1}) - 2f(x_{n+1}) + f(x_{n+2})}{\Delta x^3} + O(\Delta x^2) \tag{2.7}$$

Extensions to higher dimensions and higher order derivatives are straightforward.

2.2 PDE solution as a matrix problem

By replacing differential operators by finite differences we convert the (non-) linear PDE to a system of (non-) linear equations. The first example to demonstrate this is determining the electrostatic or gravitational potential Φ given by the Poisson equation

$$\nabla^2 \Phi(\mathbf{x}) = -4\pi\rho(\mathbf{x}) \tag{2.8}$$

where ρ is the charge or mass density and the units are chosen such that the coupling constants are all unity. Discretizing this in three space dimensions we obtain the system of linear equations:

$$\begin{aligned} &\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) \\ +&\Phi(x_n, y_{n+1}, z_n) + \Phi(x_n, y_{n-1}, z_n) \\ +&\Phi(x_n, y_n, z_{n-1}) + \Phi(x_n, y_n, z_{n-1}) \\ &- 6\Phi(x_n, y_n, z_n) = -4\pi\rho(x_n, y_n, z_n)\Delta x^2, \end{aligned} \tag{2.9}$$

where the density $\rho(x_n, y_n, z_n)$ is defined to be the average density in the cube with linear extension Δx around the point (x_n, y_n, z_n).

The general method to solve a PDE is to formulate this linear system of equations as a matrix problem and then to apply a linear equation solver to solve this system of equations. For small linear problems you can use `Mathematica`, for larger ones `Matlab` or a `dsysv` (or `dgesv`) `LAPACK` call (in Python: numpy.linalg.solve).

For larger problems it is essential to realize that the matrices produced by the discretization of PDEs are usually very sparse, meaning that only $O(N)$

elements of the N^2 matrix elements are non-zero. For these sparse systems, optimized iterative numerical eigenvalues exist[1] and are implemented in numerical libraries.[2]

Solving PDEs as a matrix problem is the most general procedure and can be used in all cases, including for boundary value problems and for eigenvalue problems. The PDE eigenvalue value problem maps onto a matrix eigenvalue problem, and eigensolvers instead of linear solvers need to be used. Again, there are efficient implementations of iterative algorithms[3] for sparse matrix eigenvalue problems.

For non-linear problems, we can first linearize the problem and then solve it.

Instead of a brute-force solution of a matrix problem, many common PDEs allow to use a more optimized solver.

2.3 The relaxation method

For the Poisson equation a simple iterative method exists that can be obtained by rewriting Eq. 2.9 as:

$$\begin{aligned}
&\Phi(x_n, y_n, z_n) = \\
&\frac{1}{6}\Big[\Phi(x_{n+1}, y_n, z_n) + \Phi(x_{n-1}, y_n, z_n) \\
&+\Phi(x_n, y_{n+1}, z_n) + \Phi(x_n, y_{n-1}, z_n) \\
&+\Phi(x_n, y_n, z_{n-1}) + \Phi(x_n, y_n, z_{n-1}) + 4\pi\rho(x_n, y_n, z_n)\Delta x^2\Big],
\end{aligned} \tag{2.10}$$

The potential is just the average over the potential on the six neighboring sites plus a term proportional to the density ρ. We obtain a solution by iterating Eq. 2.10:

$$\begin{aligned}
&\Phi_{k+1}(x_n, y_n, z_n) \leftarrow \\
&\frac{1}{6}\Big[\Phi_k(x_{n+1}, y_n, z_n) + \Phi_k(x_{n-1}, y_n, z_n) \\
&+\Phi_k(x_n, y_{n+1}, z_n) + \Phi_k(x_n, y_{n-1}, z_n) \\
&+\Phi_k(x_n, y_n, z_{n-1}) + \Phi_k(x_n, y_n, z_{n-1}) + 4\pi\rho(x_n, y_n, z_n)\Delta x^2\Big],
\end{aligned} \tag{2.11}$$

[1] R. Barret, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods (SIAM, 1993)

[2] J.G. Siek, A. Lumsdaine and Lie-Quan Lee, Generic Programming for High Performance Numerical Linear Algebra in Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO98) (SIAM, 1998); the library is available on the web at: http://www.osl.iu.edu/research/itl/

[3] Z. Bai, J. Demmel and J. Dongarra (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide (SIAM, 2000)

2.3.1 Gauss - Seidel Overrelaxation

Convergence of this method from an arbitrary charge density to a converged solution is rather slow. We can use a convergence acceleration scheme by changing the charge density by a multiple of the proposed change

$$\Delta\Phi_{k+1}(x_n, y_n, z_n) = \tag{2.12}$$
$$\frac{1}{6}\Big[\Phi_k(x_{n+1}, y_n, z_n) + \Phi_k(x_{n-1}, y_n, z_n) + \Phi_k(x_n, y_{n+1}, z_n) + \Phi_k(x_n, y_{n-1}, z_n)$$
$$+\Phi_k(x_n, y_n, z_{n-1}) + \Phi_k(x_n, y_n, z_{n-1}) + 4\pi\rho(x_n, y_n, z_n)\Delta x^2 - \Phi_k(x_n, y_n, z_n)\Big],$$
$$\Phi_{k+1}(x_n, y_n, z_n) \leftarrow \Phi_k(x_n, y_n, z_n) + w\Delta\Phi_{k+1}(x_n, y_n, z_n) \tag{2.13}$$

The algorithm converges for $0 < w < 2$. If $w > 1$, w is called *overrelaxation* factor; $w < 1$ would be an *underrelaxation* factor (not used in this context but often useful to introduce additional damping of the convergence). For $w > 2$ the algorithm may diverge, the solution becomes unstable.

2.3.2 Multi-Grid methods

We still have the problem that convergence becomes slower and slower as the mesh size is decreased. We can cure this problem by using a multi-grid method. This dramatically accelerates the convergence of many iterative solvers. We start with a very coarse grid spacing $\Delta x = \Delta x_0$ and iterate:

1. solve the Poisson equation on a grid with spacing Δ_x
2. refine the grid $\Delta x \leftarrow \Delta x/2$
3. interpolate the potential at the new grid points
4. repeat until the desired final fine grid spacing Δx_f is reached.

Initially convergence is fast since we have a very small lattice. In the later steps convergence remains fast since we always start with a guess that is close to the converged solution.

2.4 Time dependent PDEs and the Method of Lines

Our next problem are differential equations with a first-order time derivative as well as spatial derivatives:

$$\frac{\partial f(x,t)}{\partial t} = F(f,t), \tag{2.14}$$

where F contains only spatial derivatives and the initial condition at time t_0 is given by $f(x, t_0) = u(x)$. A typical example is the heat equation (or diffusion equation):

$$\frac{\partial T(x,t)}{\partial t} = \frac{K}{C\rho}\nabla^2 T(x,t) + \frac{1}{C\rho}W(x,t) \tag{2.15}$$

here T is a temperature field, C the specific heat, ρ the particle density, and K the thermal conductivity. Temperature changes either via conduction to the neighboring sites (first term) or by an external heat source or sink W.

This and similar initial value problems can be solved by the *method of lines*: after discretizing the spatial derivatives we obtain a set of coupled ODEs which can be evolved for each point along the time line (hence the name) by standard ODE solvers.

For simplicity we limit ourselves to the one-dimensional case and discretize in x:

$$\frac{\partial T(x_n,t)}{\partial t} = \frac{K}{C\rho\Delta x^2}\Big[T(x_{n+1},t) + T(x_{n-1}t) - 2T(x_n,t)\Big] + \frac{1}{C\rho}W(x_n,t) \tag{2.16}$$

Discretizing in t and using a forward Euler algorithm (this is the ODE solver!) we finally obtain

$$\begin{aligned} T(x_n,t_{n+1}) &= T(x_n,t_n) \\ &+ \frac{K\Delta t}{C\rho\Delta x^2}\Big[T(x_{n+1},t_n) + T(x_{n-1},t_n) - 2T(x_n,t_n)\Big] + \frac{\Delta t}{C\rho}W(x_n,t_n) \end{aligned} \tag{2.17}$$

2.4.1 Stability

Unlike in the case of the electrostatic boundary value problem, stability is an important issue here. Great care has to be taken in choosing appropriate values of Δx and Δt, as too long time steps Δt immediately lead to instabilities. A careful mathematical analysis (see courses on PDEs!) shows that the heat equation solver is only stable for

$$\frac{K\Delta t}{C\rho\Delta x^2} < \frac{1}{2}. \tag{2.18}$$

Note in particular that Δt is not just proportional to Δx, but $\Delta t \ll \Delta x^2$. In PDEs it is extremely important to check for instabilities!

2.4.2 The Crank-Nicolson method

We can improve the scheme above by replacing the forward Euler scheme by something more accurate, for example by a midpoint rule:

$$\begin{aligned} T(x_n, t_{n+1}) = T(x_n, t_n) & \\ + \frac{K\Delta t}{2C\rho\Delta x^2}\left[\nabla^2 T(x_n, t_n) + \nabla^2 T(x_n, t_{n+1})\right] & \\ + \frac{\Delta t}{2C\rho}(W(x_n, t_n) + W(x_n, t_{n+1})) & \end{aligned} \tag{2.19}$$

Note that the value of T at time t_{n+1} appears both on the left and on the right side of this equation, meaning that an equation has to be solved *at each time step*. Crank-Nicholson is therefore an implicit method (compare to the chapter on ODEs). This is a rather time-consuming step but the resulting method has greatly improved stability.

We have discussed the Crank-Nicholson method only for the heat equation, but it can be applied to any time-dependent partial differential equation.

2.5 The Wave equation

The wave equation is another simple PDE. It has a second derivative in space and in time. For a function $y = y(x, t)$:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial y}{\partial x^2} \tag{2.20}$$

c describes the wave velocity, it is a material constant. This equation describes behavior that is fundamentally different from the heat equation: instead of dissipation we now have oscillatory behavior. This will lead to additional complications.

Analytic solutions of this wave equation are of the form

$$y = f_+(x + ct) + f_-(x - ct) \tag{2.21}$$

To solve the wave equation numerically we again discretize time and space and obtain, using the second order difference expressions for the second derivative:

$$\frac{y(x_i, t_{n+1}) + y(x_i, t_{n-1}) - 2y(x_i, t_n)}{\Delta t^2} \approx c^2 \frac{y(x_{i+1}, t_n) + y(x_{i-1}, t_n) - 2y(x_i, t_n)}{\Delta x^2}, \tag{2.22}$$

which can be transformed to

$$y(x_i, t_{n+1}) = 2(1 - \kappa^2)y(x_i, t_n) - y(x_i, t_{n-1}) + \kappa^2\Big[y(x_{i+1}, t_n) + y(x_{i-1}, t_n)\Big], \tag{2.23}$$

with $\kappa = c\Delta t/\Delta x$.

Again, we have to choose the values of Δt and Δx carefully. For $\kappa = 1$ we obtain the exact solution without any error (insert Eq. 2.20 into Eq. 2.22). This is an accident of the linear wave equation. Decreasing both Δt and Δx at the same time only increases the spatial resolution. For $\kappa < 1$ there will be solutions propagating faster than c, but they decrease with $1/x^2$. For $\kappa > 1$ the wave equation develops unphysical solutions, which diverge rapidly.

2.6 Finite Element method

2.6.1 The basic Finite Element Method

The finite difference method we used so far is simple and straightforward for regular mesh discretizations. It becomes very hard to apply to more complex problems such as:

- spatially varying constants, such as spatially varying dielectric constants in the Poisson equation
- irregular geometries such as airplane wings or turbine blades
- dynamically adapting geometries such as moving pistons

In such cases the finite element method has a big advantage over finite differences: it does not rely on a regular mesh discretization. We will discuss the finite element method using the one-dimensional Poisson equation

$$\phi''(x) = -4\pi\rho(x) \tag{2.24}$$

with boundary conditions $\phi(0) = \phi(1) = 0$. The first step is to expand the solution $\phi(x)$ in terms of basis functions $v_i, i = 1, \ldots, \infty$ of the function space:

$$\phi(x) = \sum_{i=1}^{\infty} a_i v_i(x) \tag{2.25}$$

For our numerical calculation the infinite basis set needs to be truncated. We choose a finite subset $u_i, i = 1, \ldots, N$ linearly independent but not necessarily orthogonal functions:

$$\phi_N(x) = \sum_{i=1}^{N} a_i u_i(x) \tag{2.26}$$

The usual choice are functions localized around some mesh points x_i, which in contrast to the finite difference method do not need to form a regular mesh. The coefficients $\mathbf{a} = (a_1, \ldots, a_N)$ are then chosen to solve the equation $0 = \phi_N''(x) + 4\pi\rho(x)$ in the truncated basis over the whole interval. Since we can choose N coefficients we can impose N conditions:

$$0 = g_i \int_0^1 \Big[\phi_N''(x) + 4\pi\rho(x)\Big] w_i(x) dx. \tag{2.27}$$

The weight functions $w_i(x)$ are usually chosen to be the same as the basis functions $w_i(x) = u_i(x)$. Methods of this type are called *Galerkin* methods.

In the case of a linear PDE this method results in a linear system of equations:

$$A\mathbf{a} = \mathbf{b} \tag{2.28}$$

with the matrix A given by

$$A_{ij} = -\int_0^1 u_i''(x) w_j(x) dx = \int_0^1 u_i'(x) w_j'(x) dx \tag{2.29}$$

$$b_i = 4\pi \int_0^1 \rho(x) w_i(x) dx \tag{2.30}$$

In the first line we have used integration by parts to circumvent problems with functions that are not twice differentiable.

A good and simple choice of local basis functions fulfilling the boundary conditions are local triangles centered over the points $x_i = i\Delta x$ with $\Delta x = 1/(n+1)$:

$$u_i(x) = \begin{cases} (x - x_{i-1})/\Delta x & \text{for } x \in [x_{i-1}, x_i] \\ (x_{i+1} - x)/\Delta x & \text{for } x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

Other choices, in particular higher order polynomials are also possible.

With this choice we obtain

$$A_{ij} = \begin{cases} 2/\Delta x & \text{for } i = j \\ -1/\Delta x & \text{for } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

and, choosing the charge density $\rho(x) = \frac{\pi}{4}\sin(\pi x)$

$$b_i = 4\pi \int_0^1 \rho(x) u_i(x) = \frac{1}{\Delta x}\Big(2\sin(\pi x_i) - \sin(\pi x_{i-1}) - \sin(\pi x_{i+1})\Big) \tag{2.33}$$

In the one-dimensional case the matrix A is tridiagonal and efficient linear solvers for this tridiagonal matrix can be found in the `LAPACK` library. In higher dimensions the matrices will usually be sparse band matrices and iterative solvers will be the method of choice.

2.6.2 Generalizations to arbitrary boundary conditions

Our 1d example had the boundary conditions $\phi(1) = \phi(0)$ built in by design: the finite element basis functions did not allow anything else. We can generalize this in two different ways: either by adding additional basis functions that are nonzero at the boundary or by starting from a generalized ansatz that automatically

ensures the correct boundary conditions, such as

$$\phi_N(x) = \phi_0(1-x) + \phi_1 x + \sum_{i=1}^{N} a_i u_i(x) \tag{2.34}$$

In this case, $\phi(0) = \phi_0$ and $\phi(1) = \phi_1$ by construction.

2.6.3 Generalizations to higher dimensions

Generalizations to higher dimensions are done by

- creating higher-dimensional meshes
- providing higher-dimensional basis functions, e.g. triangles

While the basic principles remain the same, stability problems may appear at sharp corners and edges and for time-dependent geometries. The creation of appropriate meshes and basis functions is an art in itself, and an important area of industrial research.

2.6.4 Nonlinear partial differential equations

The finite element method can also be applied to non-linear partial differential equations. Let us consider a simple example:

$$\phi(x)\frac{d^2\phi(x)}{dx^2} = -4\pi\rho(x) \tag{2.35}$$

Using the same ansatz, Eq. 2.26 as before and minimizing the residuals, Eq. 2.35 becomes

$$g_i = m\int_0^1 \Big[\phi\phi''(x) + 4\pi\rho(x)\Big] w_i(x)dx \tag{2.36}$$

but instead of having a linear equation we now have a nonlinear equation:

$$\sum_{ij} A_{ijk} a_i a_j = b_k \tag{2.37}$$

with

$$A_{ijk} = -\int_0^1 u_i(x) u_j''(x) w_k(x) dx \tag{2.38}$$

and b_k defined as before.

The only difference between the case of a linear and a non-linear partial differential equation is that the former gives a set of coupled linear equations, while the latter requires the solution of a set of coupled non-linear equations.

Instead of solving this non-linear equation we can try a *Picard* iteration: start with some simple guess ϕ_0 for the solution:

$$\phi_0(x)\frac{d^2\phi_1}{dx^2} = -4\pi\rho(x) \tag{2.39}$$

and solve this for ϕ_1. Then replace ϕ_1 by ϕ_2 and iterate by solving

$$\phi_n(x)\frac{d^2\phi_{n+1}}{dx^2} = -4\pi\rho(x) \tag{2.40}$$

The iteration this solution to convergence.

2.7 Maxwell's equations

The last linear partial differential equations we will consider in this section are Maxwell's equations for the electromagnetic field. We will first calculate the field created by a single charged particle and then solve Maxwell's equations for the general case.

2.7.1 Fields due to a single moving charge

The potential Φ at location $\mathbf{R}$ due to a single static charge q at the position $\mathbf{r}$ is

$$\Phi(R) = \frac{q}{|\mathbf{r}-\mathbf{R}|} \tag{2.41}$$

The electric field is $\mathbf{E} = -\nabla\Phi$ When calculating the fields due to moving charges one needs to take into account that the electromagnetic waves only propagate with the speed of light. It is thus necessary to find the retarded position

$$\mathbf{r}_{\text{ret}} = \mathbf{R} - \mathbf{r}(t_{\text{ret}}), \tag{2.42}$$

$$r_{\text{ret}} = |\mathbf{r}_{\text{ret}}| \tag{2.43}$$

and the retarded time

$$t_{\text{ret}} = t - \frac{r_{\text{ret}}(t_{\text{ret}})}{c} \tag{2.44}$$

so that the distance r_{ret} of the particle at time t_{ret} was just ct_{ret}. Given the path of the particle this just requires a root solver. Next, the potential can be calculated as

$$\Phi(\mathbf{R},t) = \frac{q}{r_{\text{ret}}(1-\hat{\mathbf{r}}_{\text{ret}}\cdot\mathbf{v}_{\text{ret}}/c)} \tag{2.45}$$

with the retarded velocity given by

$$\mathbf{v}_{\text{ret}} = \left.\frac{d\mathbf{r}_{\text{ret}}}{dt}\right|_{t=t_{\text{ret}}} \tag{2.46}$$

The 'hat' symbol denotes a unit vector $\hat{\mathbf{r}} = \frac{\mathbf{r}}{|\mathbf{r}|}$. Note that Eq. 2.45 contains a factor between zero and one which encodes the correction due to special relativity (see your EM class or Jackson).

From the potential (and the vector potential) you can then work out the electric and magnetic field:

$$\mathbf{E}(\mathbf{R},t) = \frac{q r_{\mathrm{ret}}}{\mathbf{r}_{\mathrm{ret}} \cdot \mathbf{u}_{\mathrm{ret}}} \Big[\mathbf{u}_{\mathrm{ret}}(c^2 - v_{\mathrm{ret}}^2) + \mathbf{r}_{\mathrm{ret}} \times (\mathbf{u}_{\mathrm{ret}} \times \mathbf{a}_{\mathrm{ret}}) \Big] \tag{2.47}$$

$$\mathbf{B}(\mathbf{R},t) = \hat{\mathbf{r}}_{\mathrm{ret}} \times \mathbf{E}(\mathbf{R},t) \tag{2.48}$$

with

$$\mathbf{a}_{\mathrm{ret}} = \left. \frac{d^2\mathbf{r}(t)}{dt^2} \right|_{t=t_{\mathrm{ret}}} \tag{2.49}$$

$$\mathbf{u}_{\mathrm{ret}} = c\hat{\mathbf{r}}_{\mathrm{ret}} - \mathbf{v}_{\mathrm{ret}} \tag{2.50}$$

2.7.2 The Finite Difference Time Domain algorithm

For a single charge it was possible to compute the field at a given time by propagating the charge back in time and determining the velocity and acceleration at the retarded position and time. This only required a root solver. In the general case of many particles it is easier to directly solve the time-dependent version of Maxwell's equations:

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E} \tag{2.51}$$

$$\frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{B} - 4\pi \mathbf{j} \tag{2.52}$$

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \mathbf{j} \tag{2.53}$$

in the volume of interest. This implies discretizing the charge densities, currents, and electric and magnetic fields. This method is also known as the Yee-Vischen algorithm. We start by dividing the volume into cubes of side length Δx and defining $\rho(\mathbf{x})$ as the total charge inside this cube.

Next we discretize the current density $\mathbf{j}$. The currents are most naturally defined as flowing perpendicular to the face of the cube. Defining as $j_x(\mathbf{x},t)$ the current flowing into the cube from the left, $j_y(\mathbf{x},t)$ the one from the front, and $j_z(\mathbf{x},t)$ the one from the bottom we obtain a discretized Kirchhoff law (continuity equation) Eq. 2.53:

$$\rho(\mathbf{x}, t + \Delta t/2) = \rho(\mathbf{x}, t - \Delta t/2) - \frac{\Delta t}{\Delta x} \sum_{f=1}^{6} j_f(\mathbf{x}, t). \tag{2.54}$$

where f enumerates the faces and we have

$$j_1(\mathbf{x}, t) = -j_x(\mathbf{x}, t) \tag{2.55}$$
$$j_2(\mathbf{x}, t) = -j_y(\mathbf{x}, t) \tag{2.56}$$
$$j_3(\mathbf{x}, t) = -j_z(\mathbf{x}, t) \tag{2.57}$$
$$j_4(\mathbf{x}, t) = j_x(\mathbf{x} + \Delta x \hat{\mathbf{e}}_x, t) \tag{2.58}$$
$$j_5(\mathbf{x}, t) = j_y(\mathbf{x} + \Delta x \hat{\mathbf{e}}_y, t) \tag{2.59}$$
$$j_6(\mathbf{x}, t) = j_z(\mathbf{x} + \Delta x \hat{\mathbf{e}}_z, t) \tag{2.60}$$

(note the directions of the currents and the signs when implementing this)

The equation for the time-derivative of the electric field contains a term proportional to the currents $\mathbf{j}$. We also define the electric field proportional to the faces, but we offset it by half a time step. The curl of the electric field needed in Eq. 2.51 is then most easily evaluated on the edges of the cube (careful with the signs!). The discretized equations then are:

$$\mathbf{E}(\mathbf{x}, t + \Delta t/2) = \mathbf{E}(\mathbf{x}, t - \Delta t/2) + \frac{\Delta t}{\Delta x}\left[\sum_{e=1}^{4} \mathbf{B}_e(\mathbf{x}, t) - 4\pi \mathbf{j}_e(\mathbf{x}, t)\right] \tag{2.61}$$

$$\mathbf{B}(\mathbf{x}, t + \Delta t) = \mathbf{B}(\mathbf{x}, t) - \frac{\Delta t}{\Delta x}\sum_{f=1}^{4} \mathbf{E}_f(\mathbf{x}, t + \Delta t/2) \tag{2.62}$$

These equations are stable if $\Delta t/\Delta x \leq 1/\sqrt{3}$.

The Yee Vischen algorithm is remarkable in the sense that no linear algebra problem needs to be solved. Therefore, almost arbitrarily large problems can be posed. Its main drawback comes from the finite difference scheme which requires the same spatial discretization everywhere, making it impossible to treat 'important' areas (e.g. a wave guide or a wire) differently from free space.

2.8 Hydrodynamics and the Navier Stokes equation

The Navier Stokes equations describe the flow of a classical Newtonian fluid.

The first equation describing the flow of the fluid is the continuity equation, describing the conservation of mass (or particles):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.63}$$

a change in the mass density is caused by the fluid flowing away (compare this to 2.53, the charge continuity equation). ρ is the mass density, $\mathbf{v}$ the velocity field of the fluid.

The second equation is the Navier-Stokes equation: the equation describing the conservation of momentum:

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot \Pi = \rho \mathbf{g}, \tag{2.64}$$

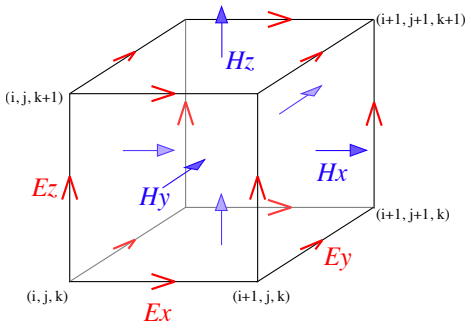

Figure 2.1: Definition of the Yee Vischen cell: Charges (not shown) are defined on the corners of the cell. Electric fields and currents go along the edges of the cell. Magnetic fields are defined through the faces of the cell. Charges and electric fields are given at half integer times, currents and magnetic fields are known at integer times.

where $\rho\mathbf{g}$ is an external source of momentum (g a force, e.g. the gravitational force) coupling to the mass density, and Π_{ij} is the momentum tensor

$$\Pi_{ij} = \rho v_i v_j - \Gamma_{ij}. \tag{2.65}$$

$$\Gamma_{ij} = \eta\Big[\partial_i v_j + \partial_j v_i\Big] + \left[\left(\zeta - \frac{2\eta}{3}\right)\nabla\cdot\mathbf{v} - P\right]\delta_{ij}. \tag{2.66}$$

The constant η describes the shear viscosity, ζ the bulk viscosity of the fluid, and P is the local pressure.

The third and final equation is the energy transport equation, describing conservation of energy:

$$\frac{\partial}{\partial t}\Big(\rho\epsilon + \frac{1}{2}\rho v^2\Big) + \nabla\cdot\mathbf{j}_e = 0 \tag{2.67}$$

where ϵ is the local energy density and the energy current $\mathbf{j}_e$ is defined as

$$\mathbf{j}_e = \mathbf{v}\Big(\rho\epsilon + \frac{1}{2}\rho v^2\Big) - \mathbf{v}\cdot\Gamma - \kappa\nabla k_B T, \tag{2.68}$$

where κ is the heat conductivity and T is the temperature.

This equation is linear except for the momentum tensor Π_{ij} in the Navier Stokes equation. This nonlinearity causes the fascinating and complex phenomenon of turbulent flow.

If you can solve the Navier-Stokes equation you will win a prize of a million US$.

2.8.1 Isothermal incompressible stationary flows

In the special case of an isothermal (constant T), static ($\partial/\partial t = 0$) flow of an incompressible fluid (rho constant) the Navier Stokes equations simplify to

$$\rho \mathbf{v} \cdot \nabla \mathbf{v} + \nabla P - \eta \nabla^2 \mathbf{v} = \rho \mathbf{g} \tag{2.69}$$

$$\nabla \mathbf{v} = 0 \tag{2.70}$$

In this stationary case there are no problems with instabilities and the Navier-Stokes equations can be solved by a linear finite-element or finite-differences method combined with a Picard iteration for the non-linear part.

2.8.2 Computational Fluid Dynamics (CFD)

Given the importance of solving the Navier-Stokes equation for engineering the numerical solution of these equations has become an important field of engineering called Computational Fluid Dynamics (CFD).

2.9 Solitons and Korteveg - de Vries equation

As the final application of partial differential equations from classical mechanics we will discuss the Korteveg - de Vries equations and solitons.

A soliton is a wave with special properties: It is a time-independent stationary solution of special non-linear wave equations. Remarkably, two solitons can pass through each other without interacting. Solitons can be used to transport signals in specially designed glass fibers over long distances without loss due to non-linear dispersion processes.

The Korteveg - de Vries (KdV) equation is famous for being the first equation for which a soliton solution exists. It is the non-linear wave equation

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u(x,t) \frac{\partial u(x,t)}{\partial x} + \mu \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{2.71}$$

where the spreading of wave packets due to dispersion (from the third term) and the sharpening due to shock waves (from the second non-linear term) combine to lead to time-independent solitons for certain parameter values.

Let us first consider these two effects separately. First, looking at a linear equation with a third order derivative

$$\frac{\partial u(x,t)}{\partial t} + c \frac{\partial u(x,t)}{\partial x} + \mu \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{2.72}$$

and solving it by the ansatz $u(x,t) = \exp(i(kx \pm \omega t))$ we find dispersion due to wave vector dependent velocities:

$$\omega = \pm ck \mp \beta k^3 \tag{2.73}$$

Thus a wave packet will spread over time. Next, looking at the non-linear term separately:

$$\frac{\partial u(x,t)}{\partial t} + \epsilon u(x,t)\frac{\partial u(x,t)}{\partial x} = 0 \tag{2.74}$$

The amplitude dependent derivative causes taller waves to travel faster than smaller ones, thus passing them and piling up to a large shock wave.

Balancing the dispersion caused by the third order derivative with the sharpening due to the nonlinear term we can obtain solitons!

The KdV equation can be solved analytically (see your favorite math textbook) using the ansatz $u(x,t) = f(x-ct)$, which yields (for $\mu = 1$ and $\epsilon = -6$):

$$u(x,t) = -\frac{c}{2}\mathrm{sech}^2\Big[\frac{1}{2}\sqrt{c}(x-ct)\Big] \tag{2.75}$$

Solving this equation instead using a finite difference method we obtain

$$\begin{aligned} u(x_i, t+\Delta t) = {} & u(x_i, t-\Delta t) \\ & - \frac{\epsilon}{3}\frac{\Delta t}{\Delta x}\Big[u(x_{i+1},t) + u(x_i,t) + u(x_{i-1,t})\Big]\Big[u(x_{i+1},t) - u(x_{i-1},t)\Big] \\ & - \frac{\mu}{2}\frac{\Delta t}{\Delta x^3}\Big[u(x_{i+2},t) + 2u(x_{i+1},t) - 2u(x_{i-1},t) - u(x_{i-2},t). \end{aligned} \tag{2.76}$$

This integrator is stable for

$$\frac{\Delta t}{\Delta x}\left[|\epsilon u| + 4\frac{|\mu|}{\Delta x^2}\right] \leq 1. \tag{2.77}$$