Numerics

Jonathan A. Pearson[*]
Centre for Particle Theory, Department of Mathematical Sciences,
Durham University, South Road, Durham, DH1 3LE, U.K.
(Dated: April 10, 2014)

Contents

I. INTRODUCTION

These notes contain descriptions of the numerics and algorithms.
The idea is to construct a code to evolve

$$\Box\Phi + m^2\Phi = 0, \tag{1.1}$$

where

$$\Box\Phi = g^{\mu\nu}\nabla_\mu\nabla_\nu\Phi, \tag{1.2}$$

in which the metric is

$$g_{\mu\nu}\mathrm{d}x^\mu\mathrm{d}x^\nu = a^2(\tau)\bigg[-(1+2V)\mathrm{d}\tau^2 + (1-2V)\mathrm{d}x_i\mathrm{d}x^i\bigg], \tag{1.3}$$

and V solves the Poisson equation,

$$\nabla^2 V = 4\pi G a^2\rho_{\mathrm{dust}}\bigg(|\Phi|^2 - 1\bigg), \qquad \rho_{\mathrm{dust}} = \bar{\rho}_{\mathrm{dust}}/a^3 \tag{1.4}$$

[*]Electronic address: jonathan.pearson@durham.ac.uk

A. Literature review

[1], [2], [3]

B. Discretization onto a lattice

Space will be discretized onto a lattice; the spacing between lattice sites is h. A quantity, Q, in 2D is discretized so that it lives on the lattice; $Q(x, y) \to Q_{i,j}$.

The discretization of the first derivative of a function $f(\mathbf{x})$ at lattice site "i" is

$$\frac{\partial f}{\partial x} = \frac{f_{i+1} - f_{i-1}}{2h}, \tag{1.5}$$

and the second derivative discretized to second order is

$$\frac{\partial^2 f}{\partial x^2} = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2}. \tag{1.6}$$

We could also discretize second derivatives to fourth order via

$$\frac{\partial^2 f}{\partial x^2} = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12h^2}. \tag{1.7}$$

II. ELLIPTIC PDES

Here we outline some methods of numerically solving elliptic partial differential equations. As an example, we will show how to solve the 2D Poisson equation,

$$\nabla^2 V = -\rho. \tag{2.1}$$

Here, $V = V(\mathbf{x})$ is a "potential", and $\rho = \rho(\mathbf{x})$ a source. Using the finite difference discretization scheme outlined in section I B, the discrete version of this is

$$V_{i+1,j} + V_{i-1,j} - 4V_{i,j} + V_{i,j+1} + V_{i,j-1} = -h^2\rho_{i,j}. \tag{2.2}$$

Here we used second order accurate derivative discretization. This can be rearranged to find

$$V_{i,j} = \frac{1}{4}\Big[V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} + h^2\rho_{i,j}\Big]. \tag{2.3}$$

There is clearly an issue: the value of the potential (which is to be found) relies on knowledge of the potential on its four neighbouring lattice sites. The remedy is to update the potential using (fictitious) time-steps;

$$V^{n+1}_{i,j} = \frac{1}{4}\Big[V^n_{i+1,j} + V^n_{i-1,j} + V^n_{i,j+1} + V^n_{i,j-1} + h^2\rho_{i,j}\Big]. \tag{2.4}$$

This is Jacobi's iterative method. As n increments, the values of the potential V converge onto those which solve the Poisson equation. There are clever ways of doing this, and each has different convergence properties.

A. Gauss-Seidel method

The Jacobi method (2.4) can be made faster by using the newly updated field values on the RHS,

$$V_{i,j}^{n+1} = \frac{1}{4}\left[V_{i+1,j}^{n} + V_{i-1,j}^{n+1} + V_{i,j+1}^{n} + V_{i,j-1}^{n+1} + h^2\rho_{i,j}\right]. \tag{2.5}$$

B. Successive over relaxation

Another method is to using a different linear combination,

$$V_{i,j}^{n+1} = (1-\omega)V_{i,j}^{n} + \frac{\omega}{4}\left[V_{i+1,j}^{n} + V_{i-1,j}^{n+1} + V_{i,j+1}^{n} + V_{i,j-1}^{n+1} + h^2\rho_{i,j}\right] \tag{2.6}$$

The parameter ω is the SoR parameter. Its value determines the convergence of the algorithm:

- Only convergent if $0 < \omega < 2$,
- Faster than Gauss-Seidel if $1 < \omega < 2$,
- Fastest on square lattice if $\omega \sim 2/(1+\pi/L)$, where L is the number of lattice points in each direction.

C. Checks against known analytic solutions

We want to check the convergence properties of the codes against known solutions.

$$\rho(x,y) = 2x^3 + 6xy(y-1), \qquad V(x,y) = x^3 y(1-y) \tag{2.7}$$

III. HYPERBOLIC PDES

Here we outline how to solve hyperbolic PDEs; that is, wave equations of the form

$$\ddot{\Phi} - \nabla^2\Phi + U'(\Phi) = S(\mathbf{x}). \tag{3.1}$$

Here, $S(\mathbf{x})$ is some source term (which will be important for the applications we have in mind), and $U(\Phi)$ is the scalar field potential; typically, $U = \frac{1}{2}m^2\Phi^2$ is a mass term. First, we trivially re-write this as

$$\ddot{\Phi} = \nabla^2\Phi - U' + S, \tag{3.2}$$

so that nothing on the RHS has time derivatives. We will write everything in 2D to reduce the number of indices needed to write the expressions down: the code is in 3D, and we shall explicitly point out any subtleties associates with going from 2D to 3D.

We define an "equation of motion" term

$$\mathcal{E}_{i,j}^{t} \equiv \nabla^2\Phi_{i,j}^{t} - U_{i,j}^{\prime t} + S_{i,j}^{t}, \tag{3.3}$$

where the Laplacian is discretized as

$$\nabla^2\Phi^t_{i,j} = \frac{\Phi^t_{i+1,j} + \Phi^t_{i-1,j} - 2\Phi^t_{i,j}}{h^2} + \frac{\Phi^t_{i,j+1} + \Phi^t_{i,j-1} - 2\Phi^t_{i,j}}{h^2}. \tag{3.4}$$

The second time derivative at a given location is discretized to second order as

$$\ddot{\Phi} = \frac{\Phi^{t+1}_{i,j} + \Phi^{t-1}_{i,j} - 2\Phi^t_{i,j}}{h_t^2}. \tag{3.5}$$

Hence,

$$\Phi^{t+1}_{i,j} = h_t^2\mathcal{E}^t_{i,j} - \Phi^{t-1}_{i,j} + 2\Phi^t_{i,j}. \tag{3.6}$$

This gives a rule to update the value of the field at each location.

A. Tests of the code

To check the code, we have implemented a few different types of initial conditions and potentials (this also helps to build intuition). To make sure that the code accesses arrays correctly, we begin by solving the gradient flow equation

$$\dot{\Phi} = \nabla^2\Phi - U'(\Phi), \tag{3.7}$$

for two different potentials:

$$U_{(1)}(\Phi) = \frac{1}{2}\Phi^2, \qquad U_{(2)}(\Phi) = \frac{1}{4}(\Phi^2-1)^2. \tag{3.8}$$

For a system endowed with the potential $U_{(1)}$, any homogeneous initial field configuration will evolve towards $\Phi = 0$, since those are the values which minimise the potential. Similarly, a system endowed with $U_{(2)}$ with homogeneous initial conditions will evolve towards $\Phi = \pm 1$.

The space- and time-step sizes must be carefully chosen: not all values give numerically stable evolutions. For the free wave equation, $\ddot{\Phi} - \nabla^2\Phi = 0$, one typically requires

$$h_t < h, \tag{3.9}$$

and for the gradient flow equation $\dot{\Phi} - \nabla^2\Phi = 0$ one typically requires

$$h_t < h^2. \tag{3.10}$$

Appendix A: Using the code

1. Klein-Gordon solver

The code is written in C^{++}, and requires `GSL` and `boost` libraries. To check that these are both installed correctly, nagivate to the `KGsolve` root directory, and type `make clean` then `make`. The code should compile without error. The source code is within the directory `src`.

The `field` is stored as a one-dimensional array; the value of the field at an array location is `field.vals[pos]` (usually, `field->vals` needs to be typed outside of the `main.cpp` file). To access the value of the field corresponding to component `c`, at time `t`, and position on the grid `i,j,k`, one should call the function `ind`:

`pos` = `ind(now,c,i,j,k,grid,field)`

[1] L. M. Widrow and N. Kaiser, *Using the Schroedinger Equation to Simulate Collisionless Matter*, *Astrophysical Journal Letters* **416** (Oct., 1993) L71.

[2] L. M. Widrow, *Modeling collisionless matter in general relativity: A New numerical technique*, *Phys.Rev.* **D55** (1997) 5997–6001, [`astro-ph/9607124`].

[3] C. Uhlemann, M. Kopp, and T. Haugg, *Schrödinger method as N-body double and UV completion of dust*, `arXiv:1403.5567`.