Computational Science & Engineering

The Definitive Guide

André Straubmeier

May 10, 2014

Contents

1 Introduction

Before we get started I would like to list all things you should already know before you start walking the road of Computational Science & Engineering. If you are not clear on any of the following things make sure to fire up Google immediately.

Math: This paper assumes you are well aware of the following mathematical topics:

- Solid understanding of linear algebra
- Big O notation

Computer Science: While computer science knowledge is not required for the understanding of this paper, it is essential to know basic computer architecture paradigms and performance optimization techniques if you ever plan to implement any of the discussed methods. These are the areas that you should be aware of before writing any simulation code:

- CPU Design:
  - CU (Control Unit) reads and executes instructions
  - ALU (Arithmetic Logic Unit) runs the actual computations
  - I/O (Input/Output) memory interface
  - CPU = CU + ALU + I/O
- CPU Problems:
  - Clock rate cannot be arbitrarily high
  - Architecture is inherently sequential (SISD)
  - The "von Neumann" Bottleneck: Speed of the memory interface is critical
- Cache Design:
  - Fully Associative
  - Direct Mapped
  - Set Associative
- Cache Problems:
  - Cache misses are very expensive
  - Aliasing: Cache lines are being trashed because every strided memory access gets mapped to the same cache line (especially problematic in a direct mapped cache)
  - Cache Coherence: Copies of the same cache line could reside in several CPU caches
- Pipelining:
  - Let N be the number of instructions and m the pipeline length

$$\text{Speedup} := \frac{T_{seq}}{T_{pipe}} = \frac{mN}{N+m-1}$$
$$\text{Throughput} := \frac{N}{T_{pipe}} = \frac{N}{N+m-1}$$

  - Warning: Avoid divisions or transcendental functions because they cannot be pipelined (pipeline bubble)
- Parrallelization:
  - Let P be the number of processes

$$\text{Speedup} := \frac{T_{seq}}{T_{par}}$$
$$\text{Efficiency} := \frac{T_{seq}}{P \cdot T_{par}}$$

2 Notation

Before we engage, let us define some important things that will be used throughout this paper. In general I will try to use upper case letters to denote matrices, lower case letters to denote vectors and Greek letters to denote scalars. Let A be a $n \times n$ matrix, x and b two vectors of length n. The equation $Ax = b$ would therefore be written as:

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & ... & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

There will however be exceptions to this rule, starting with a few reserved characters for domains, solutions and grids.

Domains

- $\Omega \rightarrow$ inner domain
- $\partial\Omega = \Gamma \rightarrow$ borders
- $\bar{\Omega} \rightarrow$ complete domain with borders
- $\Omega_h \rightarrow$ domain discretized by h

Solutions

- u exact solution
- u_i^n exact solution in the discretized domain at x_i and t^n
- $\tilde{u}_i^n$ computed solution
- $u - u_i^n$ discretization error
- $u_i^n - \tilde{u}_i^n$ algebraic error (computation/iteration error)
- $u - \tilde{u}_i^n$ total error (= discretization error + algebraic error)

Grids

- $N = (n+1)$ uniform 1D grid with $n+1$ points
- $N = (n+1) \times (n+1)$ uniform square 2D grid with $n+1$ points in each direction
- $N = (n+1) \times (n+1) \times (n+1)$ uniform square 3D grid with $n+1$ points in each direction
- $h = \frac{1}{n}$ distance between two grid points

3 Differential Equations

3.1 Classification

Consider the linear partial differential equation:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu + G = 0$$

It can be classified in the following ways:

ordinary	only derivatives with respect to one variable
partial	partial derivatives with respect to at least two variables
stationary	no derivatives with respect to t
instationary	derivatives with respect to t
linear	A-G only depend on x and y
quasi-linear	A-G also depend on u
nonlinear	A-G also depend on derivatives of u

elliptic	$B^2 - 4AC < 0$	e.g. $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$
parabolic	$B^2 - 4AC = 0$	e.g. $\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$
hyperbolic	$B^2 - 4AC > 0$	e.g. $\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2}$

3.2 Initial Conditions

Instationary differential equations require initial conditions for $t = t_0$:

- $\frac{\partial u}{\partial t} \rightarrow u(t_0) = c_1$
- $\frac{\partial^2 u}{\partial t^2} \rightarrow \frac{\partial u}{\partial t}(t_0) = c_2$
- ...

3.3 Boundary Conditions

Differential equations require boundary conditions for $x \in \Gamma$:

- Dirichlet: $u = g_1$
- Neumann: $\frac{\partial u}{\partial n} = n \cdot \nabla u = g_2$
- Robin: $\alpha u + \frac{\partial u}{\partial n} = \alpha u + n \cdot \nabla u = g_3$

3.4 Measurement of Errors

$$\text{Average absolute error} : \frac{1}{n+1}\sum_{k=0}^{n} |\tilde{u}_k - u_k|$$

$$\text{Mean squared error} : \frac{1}{n+1}\sum_{k=0}^{n} (\tilde{u}_k - u_k)^2$$

$$\text{Root mean squared error} : \sqrt{\frac{1}{n+1}\sum_{k=0}^{n} (\tilde{u}_k - u_k)^2}$$

4 Discretization Methods

4.1 Types

- Collocation methods: Find the solution at prescribed points (nodes) in the domain.
  Example: Finite Difference Method
- Spectral methods: Develop a representation of the solution in terms of basis functions.
  Example: Finite Element Method

4.2 Properties

- Consistency: Discretization error is proportional to grid discretization $h, \Delta t$
- Stability: Roundoff and iteration errors have an upper bound
- Convergence: $h, \Delta t \to 0 \Rightarrow \text{error} \to 0$
- Conservativity: Error affects only distribution of mass, conservative methods are more reliable
- Boundedness: If the exact solution is bounded, the numerical solution should be as well
- Accuracy: Relatively small error with relatively large $h, \Delta t$

4.3 Finite Difference Method

4.3.1 General Approach

Assume we want to discretize the following 1D equation:

$$a(x)\frac{\partial^2 u}{\partial x^2} + b(x)\frac{\partial u}{\partial x} + c(x)u = f(x) \qquad u(0) = u_0 \qquad u(1) = u_1$$

First we define $h = \frac{1}{n}$ and $x_k = k \cdot h$ and do a Taylor expansion of u around x:

$$u(x+h) = u(x) + \frac{h^1}{1!}u'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u'''(x) + O(h^4)$$
$$u(x-h) = u(x) - \frac{h^1}{1!}u'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + O(h^4)$$

Consider $u(x+h) - u(x-h)$:

$$u(x+h) - u(x-h) = 2hu'(x) + \frac{h^3}{3}u'''(x) + O(h^4)$$
$$\Rightarrow u'(x) = \frac{u(x+h) - u(x-h)}{2h} + O(h^2)$$

Consider $u(x+h) + u(x-h)$:

$$u(x+h) + u(x-h) = 2u(x) + h^2u''(x) + O(h^4)$$
$$\Rightarrow u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2)$$

One can therefore write:

$$a_k\frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} + b_k\frac{u_{k+1} - u_{k-1}}{2h} + c_k u_k = f_k$$

Now: Have a look at a simple 2D equation:

$$a(x,y)\frac{\partial^2 u}{\partial x \partial y} = f(x,y) \qquad u(0,y) = u_{y0} \quad u(1,y) = u_{y1} \quad u(x,0) = u_{x0} \quad u(x,1) = u_{x1}$$

By using $h_x = \frac{1}{n_x}$ and $x_k = k \cdot h_x$ we start with a discretization in x direction:

$$\frac{\partial u}{\partial x} = \frac{u(x+h,y) - u(x-h,y)}{2h_x} + O(h_x^2)$$

By using $h_y = \frac{1}{n_y}$ and $y_k = k \cdot h_y$ we discretize this resulting expression in y direction:

$$\begin{aligned}\frac{\partial^2 u}{\partial x \partial y} &= \frac{\frac{u(x+h,y+h)-u(x-h,y+h)}{2h_x} + O(h_x^2) - \frac{u(x+h,y-h)-u(x-h,y-h)}{2h_x} + O(h_x^2)}{2h_y} + O(h_y^2) \\ &= \frac{u(x+h,y+h) - u(x-h,y+h) - u(x+h,y-h) + u(x-h,y-h)}{4h_xh_y} + O(h_x^2) + O(h_y^2)\end{aligned}$$

Remark:

- The error order of the discretization is dominated by the lowest order rest term
- The error order resembles how fast the error converges towards zero
- The error order does **not** say anything about the absolute value of the error
- Second order approximations are usually a good tradeoff between accuracy and complexity

4.3.2 Neumann Boundary Conditions

$$a(x)\frac{\partial^2 u}{\partial x^2} + b(x)\frac{\partial u}{\partial x} + c(x)u = f(x) \qquad\qquad u(0) = u_0 \qquad \frac{\partial u}{\partial x}(1) = g_1$$

At the boundary the value $u(x+h)$ is unknown, for this reason central differences are not an option. One possibility is to discretize the boundary with backward differences:

$$\frac{\partial u}{\partial x} = \frac{u(x) - u(x-h)}{h} + O(h) \qquad \rightarrow \qquad \frac{u_n - u_{n-1}}{h} = g_1 + O(h)$$

However, this discretization is only first order and will drag down the complete algorithm to order $O(h)$. Therefore, higher order backward differences can be derived:

$$u(x-h) = u(x) - \frac{h^1}{1!}u'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + O(h^4)$$
$$u(x-2h) = u(x) - \frac{(2h)^1}{1!}u'(x) + \frac{(2h)^2}{2!}u''(x) - \frac{(2h)^3}{3!}u'''(x) + O(h^4)$$

Consider $4u(x-h) - u(x-2h)$:

$$4u(x-h) - u(x-2h) = 3u(x) - 2hu'(x) - 4h^3u'''(x) + O(h^4)$$
$$\Rightarrow u'(x) = \frac{3u(x) - 4u(x-h) + u(x-2h)}{2h} + O(h^2)$$

One can therefore write:

$$\frac{\partial u}{\partial x} = \frac{3u(x) - 4u(x-h) + u(x-2h)}{2h} + O(h^2) \qquad \rightarrow \qquad \frac{3u_n - 4u_{n-1} + u_{n-2}}{2h} = g_1 + O(h^2)$$

Remark: The resulting matrix is no longer tridiagonal, this can however easily be solved by manipulating the last row.

4.4 Timestep Discretization

4.4.1 Example Problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad\qquad x_i = i \cdot h \qquad t_j = j \cdot k$$

4.4.2 Explicit Euler Scheme

$$\frac{u_{i,j+1} - u_{i,j}}{k} + O(k) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2)$$

- Forward first order FD in time and central second order FD in space
- Explicit: only a matrix multiplication / time step
- Consistent but only stable for $k/h^2 \leq 1/2$

4.4.3 Implicit Euler Scheme

$$\frac{u_{i,j+1} - u_{i,j}}{k} + O(k) = \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{h^2} + O(h^2)$$

- Forward first order FD in time and central second order FD in space
- Implicit: solve a linear system / time step
- Consistent and stable

4.4.4 Richardson's Scheme

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(k^2) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2)$$

- Central second order FD in time and space
- Explicit: only a matrix multiplication / time step
- Consistent but unstable

4.4.5 Dufont-Frankil Scheme

A modification of Richardson's scheme in which the term $u_{i,j}$ is replaced by the time average.

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(k^2) = \frac{u_{i+1,j} - 2\left(\frac{u_{i,j+1}+u_{i,j-1}}{2}\right) + u_{i-1,j}}{h^2} + O(h^2) \quad = \frac{u_{i+1,j} - u_{i,j+1} - u_{i,j-1} + u_{i-1,j}}{h^2} + O(h^2)$$

- Central second order FD in time and space
- Explicit: only a matrix multiplication / time step
- Inconsistent but stable

4.4.6 Crank-Nicolson Scheme

A modification of Richardson's scheme in which the term u_{xx} is replaced by the time average.

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(k^2) = \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} + u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}}{2h^2} + O(h^2)$$

$$j - 1 \to j \qquad 2k \to k$$

$$\frac{u_{i,j+1} - u_{i,j}}{k} + O(k^2) = \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} + u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{2h^2} + O(h^2)$$

- Central second order FD in time and space
- Implicit: (matrix multiplication + linear system) / time step
- Consistent and stable

4.4.7 Consistency Analysis

A scheme is said to be consistent if it solves the equation for which it was constructed. The idea is to insert the Taylor series into the numerical scheme and see if unwanted factors that do not cancel out appear.

Example: Dufont-Frankil

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(k^2) = \frac{u_{i+1,j} - u_{i,j+1} - u_{i,j-1} + u_{i-1,j}}{h^2} + O(h^2)$$

By Taylor series:

$$u_{i,j+1} = u + ku_t + \frac{k^2}{2}u_{tt} + O(k^3) \qquad u_{i,j-1} = u - ku_t + \frac{k^2}{2}u_{tt} + O(k^3)$$
$$u_{i+1,j} = u + hu_x + \frac{h^2}{2}u_{xx} + O(h^3) \qquad u_{i-1,j} = u - hu_x + \frac{h^2}{2}u_{xx} + O(h^3)$$

This leads to:

$$\underbrace{\frac{ku_t + ku_t}{2k}}_{u_t} + O(k^2) = \underbrace{\frac{h^2u_{xx} - k^2u_{tt}}{h^2}}_{u_{xx} - \frac{k^2}{h^2}u_{tt}} + O(h^2)$$

The method is only consistent if $k \ll h$.

4.4.8 Stability Analysis

- Fourier approach (ignores the effect of the boundary conditions)
  1. Replace $u_{p,q}$ with $\lambda^q e^{i\alpha ph}$ in the numerical scheme
  2. Compute λ
  3. If $|\lambda| > 1$ the scheme is unstable, otherwise it is stable
- Matrix approach (boundary conditions included)

Example: Explicit Euler

$$\frac{u_{i,j+1} - u_{i,j}}{k} + O(k) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2)$$

The Fourier approach leads to:

$$u_{p,q+1} = u_{p,q} + \frac{k}{h^2}\left(u_{p-1,q} - 2u_{p,q} + u_{p+1,q}\right)$$
$$\lambda^{q+1}e^{i\alpha ph} = \lambda^q e^{i\alpha ph} + \frac{k}{h^2}\left(\lambda^q e^{i\alpha(p-1)h} - 2\lambda^q e^{i\alpha ph} + \lambda^q e^{i\alpha(p+1)h}\right)$$
$$\lambda = 1 + \frac{k}{h^2}\left(e^{-i\alpha h} - 2 + e^{i\alpha h}\right)$$

With some calculus this yields:

$$\lambda \leq +1 \qquad \text{always}$$
$$\lambda \geq -1 \qquad \text{if } \frac{k}{h^2} \leq \frac{1}{2}$$

The method is only stable if $\frac{k}{h^2} \leq \frac{1}{2}$.

4.4.9 Implementation Remarks

The complexity of the implemenation of time discretizations depends mainly on whether it is explicit or implicit.

Explicit Schemes:

1. Setup the solution of the LSE for t_0
2. Until t_n is reached:
   - Increment the time
   - Calculate the solution for t_i by *multiplying the result from t_{i-1} with a matrix* given by the discretization

     Remark: This works because the solution of t_i only depends on the solution of the previous time step, every point of the solution can therefore be directly calculated.

Implicit Schemes:

1. Setup the solution of the LSE for t_0
2. Until t_n is reached:
   - Increment the time
   - Calculate the solution for t_i by *solving the linear system of equations* given by the discretization

     Remark: As the solution of t_i depends on both the solution of the previous time step and some points of the solution of the current time step this can be viewed as a normal linear system of equations that has to be solved for every time step.

4.5 Finite Element Method

4.5.1 General Approach

1. **Finding the weak formulation of the problem with bilinear form a and linear form b**

$$\text{weak formulation: find } u \in V \text{ such that } a(u,v) = b(f,v) \quad \forall v \in V$$

a) Start with the weighted residual method

$$\mathcal{L}u = f$$
$$\int_\Omega \mathcal{L}u \cdot v \, \mathrm{d}x = \int_\Omega fv \, \mathrm{d}x \quad \forall v \in V$$

b) Use integration by parts [1] to transfer derivatives from u to v
Example: Poisson's equation, Dirichlet BCs (disappear b/c $v|_\Gamma = 0$, Neumann and Robin might not)

$$-\Delta u = f \quad u \in \Omega$$
$$u = g \quad u \in \Gamma$$
$$-\int_\Omega \Delta u \cdot v \, \mathrm{d}x = \int_\Omega fv \, \mathrm{d}x \quad \forall v \in V$$
$$\int_\Omega \nabla u \nabla v \, \mathrm{d}x = \int_\Omega fv \, \mathrm{d}x \quad \text{(using Green's identity)}$$
$$a(u,v) = b(f,v)$$

2. **Choosing the basis/shape functions**

one usually uses "tent" functions to get a sparse matrix in the resulting LSE (so-called "small support" of the basis)

for the 1D case: $\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases}$

3. **Deriving the matrix form of the problem**

- With discretized u, f and v...

$$u_h = \sum_{i=1}^{N} u_i \varphi_i \qquad f_h = \sum_{i=1}^{N} f_i \varphi_i \qquad v_h = \sum_{i=1}^{N} v_i \varphi_i$$

- By applying magic the weak formulation becomes...

$$\sum_{j=1}^{N} \sum_{i=1}^{N} u_i \cdot a(\varphi_i, \varphi_j) = \sum_{j=1}^{N} \sum_{i=1}^{N} f_i \cdot b(\varphi_i, \varphi_j)$$

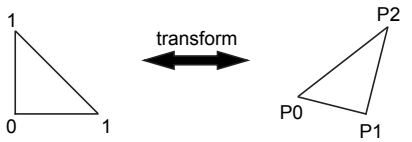
- Therefore one can define two matrices...

$$\text{stiffness matrix } A = (L_{ij}) \qquad \text{with } A_{ij} = a(\varphi_i, \varphi_j)$$
$$\text{mass matrix } M = (M_{ij}) \qquad \text{with } M_{ij} = b(\varphi_i, \varphi_j)$$

- And get the resulting LSE...

$$Au = Mf$$

4.5.2 Reference Triangle



reference triangle	**finite element**
$R : \hat{P}_0(0,0)\,\hat{P}_1(1,0)\,\hat{P}_2(0,1)$	$G_k : P_0(x_0,y_0)\,P_1(x_1,y_1)\,P_2(x_2,y_2)$
$\hat{\varphi}_i : R \to \mathbb{R}$	$\varphi_i : G_k \to \mathbb{R}$
$\hat{\varphi}_0 = 1 - \hat{x} - \hat{y}$	φ_0
$\hat{\varphi}_1 = \hat{x}$	φ_1
$\hat{\varphi}_2 = \hat{y}$	φ_2
$\hat{\varphi}_i(\hat{x},\hat{y}) = \varphi_i(T_k(\hat{x},\hat{y}))$	$\varphi_i(x,y) = \hat{\varphi}_i(T_k^{-1}(x,y))$

$$\begin{pmatrix} x \\ y \end{pmatrix} = T_k(\hat{x},\hat{y}) = P_0 + \hat{x}(P_1 - P_0) + \hat{y}(P_2 - P_0) = \begin{pmatrix} x_0 + \hat{x}(x_1 - x_0) + \hat{y}(x_2 - x_0) \\ y_0 + \hat{x}(y_1 - y_0) + \hat{y}(y_2 - y_0) \end{pmatrix}$$

$$J_{T_k} = \begin{pmatrix} \frac{\partial T_{k1}}{\partial \hat{x}} & \frac{\partial T_{k1}}{\partial \hat{y}} \\ \frac{\partial T_{k2}}{\partial \hat{x}} & \frac{\partial T_{k2}}{\partial \hat{y}} \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$$

4.5.3 Local Mass Matrix

$$M_{G_k} = \begin{pmatrix} b_{G_K}(\varphi_0,\varphi_0) & b_{G_K}(\varphi_0,\varphi_1) & b_{G_K}(\varphi_0,\varphi_2) \\ b_{G_K}(\varphi_1,\varphi_0) & b_{G_K}(\varphi_1,\varphi_1) & b_{G_K}(\varphi_1,\varphi_2) \\ b_{G_K}(\varphi_2,\varphi_0) & b_{G_K}(\varphi_2,\varphi_1) & b_{G_K}(\varphi_2,\varphi_2) \end{pmatrix}$$

$$\text{Poisson's equation in 2D: } b_{G_K}(\varphi_i,\varphi_j) = \int_{G_k} \varphi_i\varphi_j \,\mathrm{d}(x,y)$$

$$\begin{aligned} \int_{G_k} \varphi_i\varphi_j \,\mathrm{d}(x,y) &= \int_{G_k} \hat{\varphi}_i(T_k^{-1}(x,y)) \cdot \hat{\varphi}_j(T_k^{-1}(x,y)) \,\mathrm{d}(x,y) \quad (\rightarrow \text{change of coordinates}) \\ &= \int_R \hat{\varphi}_i(\hat{x},\hat{y}) \cdot \hat{\varphi}_j(\hat{x},\hat{y}) \cdot |\det J_{T_k}| \,\mathrm{d}(x,y) \end{aligned}$$

4.5.4 Local Stiffness Matrix

$$M_{G_k} = \begin{pmatrix} a_{G_K}(\varphi_0,\varphi_0) & a_{G_K}(\varphi_0,\varphi_1) & a_{G_K}(\varphi_0,\varphi_2) \\ a_{G_K}(\varphi_1,\varphi_0) & a_{G_K}(\varphi_1,\varphi_1) & a_{G_K}(\varphi_1,\varphi_2) \\ a_{G_K}(\varphi_2,\varphi_0) & a_{G_K}(\varphi_2,\varphi_1) & a_{G_K}(\varphi_2,\varphi_2) \end{pmatrix}$$

$$\text{Poisson's equation in 2D: } a_{G_K}(\varphi_i,\varphi_j) = \int_{G_k} \nabla\varphi_i \nabla\varphi_j \,\mathrm{d}(x,y)$$

$$\begin{aligned} \int_{G_k} \nabla\varphi_i \nabla\varphi_j \,\mathrm{d}(x,y) &= \int_{G_k} \nabla\hat{\varphi}_i(T_k^{-1}(x,y)) \cdot \nabla\hat{\varphi}_j(T_k^{-1}(x,y)) \,\mathrm{d}(x,y) \quad (\rightarrow \text{change of coordinates}) \\ &= \int_R \nabla\hat{\varphi}_i(\hat{x},\hat{y})^T \cdot J_{T_k}^{-1} \cdot (J_{T_k}^{-1})^T \cdot \nabla\hat{\varphi}_j(\hat{x},\hat{y}) \cdot |\det J_{T_k}| \,\mathrm{d}(x,y) \end{aligned}$$

5 Solving Linear Systems of Equations

Let us now discuss the case that want to solve $Ax = b$ with a sparse matrix A. For the case that A is a tridiagonal matrix, Gaussian elimination (or the Thomas algorithm as a simplification thereof) will lead to the exact solution within acceptable computation time. However, if A is for example a band matrix, Gaussian elimination leads to computation cost of $O(n^3)$ and storage cost of $O(n^2)$. All zeros between the left and right band of the matrix A are being destroyed one by one until we finally get a diagonal matrix. Therefore, when dealing with arbitrary sparse matrices, one has to look for an alternative way of solving the LSE.

5.1 Iterative Methods

Let x be the solution of $Ax = b$, the idea is to construct a succession of vectors that enjoy the property of convergence:

$$\lim_{k\to\infty} x^{(k)} = x$$

$$x^{(k+1)} = \alpha x^{(k)} \quad \begin{cases} \alpha < 1 \to 0 & \text{(contraction mapping)} \\ \alpha > 1 \to \infty \end{cases}$$

Start with additive splitting:

$$\begin{aligned} Ax &= b \qquad A = (P + N) \\ Px + Nx &= b \\ Px + (A - P)x &= b \\ Px &= (P - A)x + b \end{aligned}$$

Where P is called preconditioning matrix (approximates A^{-1}). We can then write:

$$\begin{aligned} x^{(k+1)} &= (I - P^{-1}A)x^{(k)} + P^{-1}b \\ x^{(k+1)} &= Bx^{(k)} + f \end{aligned}$$

Where B is called iteration matrix. Finally one can analyze the error at each iteration step:

$$\begin{aligned} e^{(k)} &= x - x^{(k)} \\ e^{(k+1)} &= x - x^{(k+1)} \\ x &= Bx + f \\ x^{(k+1)} &= Bx^{(k)} + f \end{aligned}$$

$$\begin{aligned} \Rightarrow e^{(k+1)} &= x - x^{(k+1)} = x - Bx^{(k)} - f = x - B(x - e^{(k)}) - f \\ &= \underline{x - Bx} + Be^{(k)} \underline{- f} = Be^{(k)} \end{aligned}$$

One can easily see that for the case that the iteration matrix B is a contraction mapping the iteration converges. This means that all eigenvalues of B must satisfy $|\lambda| < 1$ (smaller eigenvalues improve convergence, $|\lambda| > 1$ diverges).

5.2 Jacobi Method

Let L be a lower triangular matrix, D a diagonal matrix and U an upper triangular matrix.

$$Ax = b \qquad A = (L + D + U)$$

$$Dx = b - (L + U)x \qquad\qquad x = D^{-1}b - D^{-1}(L + U)x$$

Here $P = D$ is the preconditioning matrix and $B = D^{-1}(L + U)$ is the iteration matrix.

5.3 Gauß-Seidel Method

Let L be a lower triangular matrix, D a diagonal matrix and U an upper triangular matrix.

$$Ax = b \qquad A = (L + D + U)$$

$$(L + D)x = b - Ux \qquad\qquad x = (L + D)^{-1}b - (L + D)^{-1}Ux$$

Here $P = (L + D)$ is the preconditioning matrix and $B = (L + D)^{-1}U$ is the iteration matrix.

5.4 Successive Over-Relaxation

The SOR method is basically just a modified Gauß-Seidel method.

5.5 Examples

The difference between the Jacobi method and the Gauss-Seidel method can be demonstrated by a simple example:

$$\begin{bmatrix} 8 & 1 & -1 \\ 2 & 1 & 9 \\ 1 & -7 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \\ -4 \end{bmatrix}$$

We reorder this such that the absolute value of the product of the elements of the diagonal D is maximum. This leads to minimal eigenvalues of the iteration matrix:

$$\begin{bmatrix} 8 & 1 & -1 \\ 1 & -7 & 2 \\ 2 & 1 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -4 \\ 12 \end{bmatrix}$$

The Jacobi method leads to the following scheme:

$$\begin{aligned} x^{(k+1)} &= 1 - \tfrac{1}{8}y^{(k)} + \tfrac{1}{8}z^{(k)} \\ y^{(k+1)} &= \tfrac{4}{7} + \tfrac{1}{7}x^{(k)} + \tfrac{2}{7}z^{(k)} \\ z^{(k+1)} &= \tfrac{12}{9} - \tfrac{2}{9}x^{(k)} - \tfrac{1}{9}y^{(k)} \end{aligned}$$

The Gauß-Seidel method leads to the following scheme:

$$\begin{aligned} x^{(k+1)} &= 1 - \tfrac{1}{8}y^{(k)} + \tfrac{1}{8}z^{(k)} \\ y^{(k+1)} &= \tfrac{4}{7} + \tfrac{1}{7}x^{(k+1)} + \tfrac{2}{7}z^{(k)} \\ z^{(k+1)} &= \tfrac{12}{9} - \tfrac{2}{9}x^{(k+1)} - \tfrac{1}{9}y^{(k+1)} \end{aligned}$$

The question whether the Jacobi method or the Gauß-Seidel method is the better one depends on the application, however there are a few things to keep in mind:

- The Gauss-Seidel method is guaranteed to converge for M-matrices
  - In each row the sum of the absolute values of all off-diagonal entries is smaller than of equal to the absolute value of the diagonal entry
  - The matrix is non-singular
- FD discretizations are always M-matrices
- When does one stop iterating?
  - Generally speaking one should reduce the algebraic error to about 0.1 times the discretization error, however, the exact solution is often unknown
  - Use the residual $r = |A\tilde{x} - b|$ and stop when $r^{(k)} = |A\tilde{x}^{(k)} - b| < \epsilon |r^{(0)}|$
- When does one stop making the grid finer?
  - Use the error $|\tilde{x}_h - \tilde{x}_{h/2}|$ and stop when $|\tilde{x}_h - \tilde{x}_{h/2}| < \epsilon |\tilde{x}_h|$
- For dense matrices use Gaussian elimination, iterative methods are only suited for sparse matrices

6 Solving Nonlinear Systems of Equations

6.1 Example Problem

Consider the following boundary value problem:

$$\frac{\partial^2 u}{\partial x^2} = u^2 \qquad u(0) = 0 \qquad u(1) = 1$$

And the according FD discretization:

$$\frac{u_{k+1} - 2u_k + u_{k-1}}{h^2} = u_k^2 + O(h^2) \qquad u_0 = 0 \qquad u_n = 1$$

This leads to $n-1$ nonlinear algebraic equations to solve:

$$\begin{aligned} F_1 &= u_2 - 2u_1 - h^2 u_1^2 = 0 \\ F_k &= u_{k+1} - 2u_k + u_{k-1} - h^2 u_k^2 = 0 \qquad\qquad k = 2 \dots (n-2) \\ F_{n-1} &= 1 - 2u_{n-1} + u_{n-2} - h^2 u_{n-1}^2 = 0 \end{aligned}$$

6.2 Newton's Method

The nonlinear algebraic equation is being solved iteratively by Newton's method in the following way:

$$\begin{aligned} \vec{F}(\vec{u}) &= (F_1, F_2, \dots, F_{n-1}) = \vec{0} \\ \vec{F}(\vec{u}^{(k)}) &= J_{\vec{u}^{(k)}}(\vec{u}^{(k)} - \vec{u}^{(k-1)}) \end{aligned}$$

Where $J_{\vec{u}^{(k)}}$ is the Jacobian of $\vec{F}$ evaluated at $\vec{u}^{(k)}$

- The ith row of the tridiagonal Jacobian matrix is:

$$\begin{pmatrix} 0 & \dots & 0 & 1 & -2 - 2h^2 u_i^{(k)} & 1 & 0 & \dots & 0 \end{pmatrix}$$

- At each iteration of the Newton's method one has to solve a linear system
- The linear systems have the same structure as the FD stencil
- Newton iterations require an initial guess $\vec{u}^{(0)}$ - typically a simple function satisfying the boundary conditions
- In the example problem: $u_i^{(0)} = x_i$ satisfies $u(0) = 0$ and $u(1) = 1$

7 Conjugate Gradient Method

- Well suited for use with symmetric sparse matrices
- Convergence guaranteed for positive definite matrices ($x^T Ax > 0$ for any vector x)

7.1 Quadratic Form

$$f(x) = \frac{1}{2}x^T Ax - b^T x + \alpha$$
$$f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = \frac{1}{2}(A^T + A)x - b$$

A positive definite matrix A guarantees that the solution of $Ax = b$ is at the critical point $f'(x) = 0$. That means the quadratic form has exactly one critical point, the global minumum. For the case that A is symmetric ($A^T = A$), the equation simplifies even further:

$$f'(x) = Ax - b$$

7.2 Method of Steepest Descent

7.2.1 Derivation

- Start at a guess x_0
- Choose the direction in which f decreases fastest: $-f'(x_i) = b - Ax_i = r_i \rightarrow$ This is actually the residual!
- Choose the step length: $x_1 = x_0 + \alpha r_0 \rightarrow$ How to choose α?
  - Remember that our task is to find the global minimum of f
  - Therefore at every step, we want to minimize f along the line:

    $$x_1 = x_0 + \alpha r_0$$

  - From basic calculus we know that α minimizes f when:

    $$\frac{\partial f(x_1)}{\partial \alpha} = 0$$

  - By applying the chain rule:

    $$\frac{\partial f(x_1)}{\partial \alpha} = f'(x_1)^T \frac{\partial x_1}{\partial \alpha} = f'(x_1)^T r_0 = 0$$

  - We know that $f'(x_1) = -r_1$ and therefore we can write:

    $$r_1^T r_0 = 0$$

  - From this one can derive [4] that α should be:

    $$\alpha = \frac{r_0^T r_0}{r_0^T A r_0}$$

- The method of steepest descent therefore becomes:
  1. $r_i = b - Ax_i$
  2. $\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i}$
  3. $x_{i+1} = x_i + \alpha_i r_i$
- We can avoid recalculating r_i at every step by incrementing it in every iteration. By multiplying both sides of step 3 with $-A$ and adding b we get:

  $$r_{i+1} = r_i - \alpha_i A r_i$$

  However, this way the residual will get more and more inaccurate every step as numerical errors add up. It is therefore a good idea to periodically recalculate r_i from scratch using the equation from step 1 (e.g. every 10 iterations).

7.2.2 Convergence Analysis

Properties of eigenvalues and eigenvectors:

- All eigenvalues λ_i and eigenvectors v_i of a matrix A can be written as:

$$Av_i = \lambda_i v_i \qquad i = 1 \dots n$$

- If a matrix is symmetric, it has a set of orthonormal eigenvectors:

$$v_i^T v_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

- If a matrix is positive definite, all of its eigenvalues are positive:

$$v_i^T A v_i = \lambda_i v_i^T v_i > 0$$

A thought experiment: If the error is an eigenvector with eigenvalue λ_e the residual is also an eigenvector:

$$r_i = -Ae_i = -\lambda_e e_i$$

In this case one can observe instant convergence after one step of steepest descent:

$$\begin{aligned} e_{i+1} &= e_i + \frac{r_i^T r_i}{r_i^T A r_i} r_i \\ &= e_i + \frac{r_i^T r_i}{r_i^T \lambda_e r_i}(-\lambda_e e_i) \\ &= 0 \end{aligned}$$

Idea: Expressing the error as a linear combination n of eigenvectors and eliminating it component by component would lead to convergence after n steps:

$$e_i = \sum_{j=1}^{n} \xi_j v_j$$

Where ξ_j is the length of each component of the error. The residual can therefore be written as:

$$r_i = -Ae_i = \sum_{j=1}^{n} \xi_j \lambda_j v_j$$

With some linear algebra [5] we can write the error reduction as:

$$x_{i+1} = x_i + \alpha_i r_i \qquad \rightarrow \qquad e_{i+1} = e_i + \frac{\sum_{j=1}^{n} \xi_j^2 \lambda_j^2}{\sum_{j=1}^{n} \xi_j^2 \lambda_j^3} r_i$$

For the case that all eigenvalues are equal this again results in instant convergence:

$$\begin{aligned} e_{i+1} &= e_i + \frac{\lambda^2 \sum_{j=1}^{n} \xi_j^2}{\lambda^3 \sum_{j=1}^{n} \xi_j^2} r_i \\ &= e_i + \frac{1}{\lambda} r_i \\ &= 0 \end{aligned}$$

If the matrix A has several unequal eigenvalues, no choice of α_i will eliminate all the error components in one step. The choice therefore becomes a compromise - we use the weighted average of the eigenvalues:

$$e_{i+1} = e_i + \frac{\sum_{j=1}^{n} \xi_j^2 \lambda_j^2}{\sum_{j=1}^{n} \xi_j^2 \lambda_j^3} r_i$$

For this reason on a given iteration some of the error components may increase. The method of steepest descent (and the conjugate gradient method which we will discuss soon) are therefore called *roughers*, while the Jacobi method and the Gauß-Seidel method are known as *smoothers*.

Convergence analysis: The energy norm is defined as:

$$||x||_A = \sqrt{(x^T A x)}$$
$$||x||_A^2 = x^T A x$$

Minimizing the energy norm of the error is equivalent to minimizing f:

$$\begin{aligned} ||e_{i+1}||_A^2 &= e_{i+1}^T A e_{i+1} \\ &= (e_i^T + \alpha_i r_i^T) A (e_i + \alpha_i r_i) \qquad \text{[6]} \\ &= \omega^2 \, ||e_i||_A^2 \end{aligned}$$

$$\omega^2 = 1 - \frac{\left(\sum_{j=1}^n \xi_j^2 \lambda_j^2\right)^2}{\left(\sum_{j=1}^n \xi_j^2 \lambda_j^3\right)\left(\sum_{j=1}^n \xi_j^2 \lambda_j\right)}$$

For the 2D case ($n = 2$):

$$\begin{aligned} \omega^2 &= 1 - \frac{\left(\xi_1^2\lambda_1^2 + \xi_2^2\lambda_2^2\right)^2}{\left(\xi_1^2\lambda_1^3 + \xi_2^2\lambda_2^3\right)\left(\xi_1^2\lambda_1 + \xi_2^2\lambda_2\right)} \\ &= 1 - \frac{(\kappa^2 + \mu^2)^2}{(\kappa^3 + \mu^2)(\kappa + \mu^2)} \qquad \text{with } \kappa = \frac{\lambda_1}{\lambda_2} \geq 1 \text{ and } \mu = \frac{\xi_2}{\xi_1} \end{aligned}$$

The upper bound (for the worst-case starting points) is found by setting $\mu^2 = \kappa^2$:

$$\begin{aligned} \omega^2 &\leq 1 - \frac{4\kappa^4}{\kappa^5 + 2\kappa^4 + \kappa^3} \\ &= \frac{\kappa^5 - 2\kappa^4 + \kappa^3}{\kappa^5 + 2\kappa^4 + \kappa^3} \\ &= \frac{(\kappa - 1)^2}{(\kappa + 1)^2} \end{aligned}$$

$$\Rightarrow \omega \leq \frac{\kappa - 1}{\kappa + 1}$$

It can be shown that this is also valid for $n > 2$ if A is a symmetric, positive definite matrix. For this case the condition number is defined as $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ and the upper bound ($\mu^2 = \kappa^2$) for the error at step i becomes:

$$||e_i||_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i ||e_0||_A$$

This shows that the convergence is inverse proportional to the condition number κ, meaning the larger κ the slower the convergence. The method of steepest descent has the problem that steps are often being taken in the same direction as previous steps. This is easily understandable if one keeps in mind that every step is taken in an orthogonal direction of the previous step. For the 2D case this guarantees that every second step has the same direction.

Idea: Is it possible to guarantee that the error component in one direction is zero after one step? This would lead to convergence after n steps, as it would cut down the error term component by component.

7.3 Method of Conjugate Directions

7.3.1 Derivation

The method of conjugate directions is an extention of the method of steepest descent:

$$x_{i+1} = x_i + \alpha_i d_i$$

In the method of steepest descent d_i would be r_i. However, here we are looking for a d_i that, if walked along, completely cuts away one component of the error. If you imagine d_i being the coordinate axes, we would start walking along the first axis until the first component of our current solution is equal to the exact solution. This would be repeated for

every coordinate component and would yield the exact solution after n steps (n being the number of coordinates). This is equivalent to walking along d_i until it is orthogonal to the new error:

$$\begin{aligned} d_i^T e_{i+1} &= 0 \\ d_i^T (e_i + \alpha_i d_i) &= 0 \end{aligned}$$

This yields:

$$\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$$

However, the error is ususally unknown because the exact solution is unknown. The idea to circumvent this is to make the search direction A-orthogonal (conjugate):

$$d_i^T A d_j = 0 \qquad i \neq j$$

To get α_i we use the same technique we used in the method of steepest descent:

$$\begin{aligned} \frac{\partial}{\partial \alpha} f(x_{i+1}) &= 0 \\ f(x_{i+1})^T \frac{\partial}{\partial \alpha} x_{i+1} &= 0 \\ -r_{i+1}^T d_i &= 0 \\ d_i^T A e_{i+1} &= 0 \\ d_i^T A (e_i + \alpha_i d_i) &= 0 \end{aligned}$$

This yields:

$$\begin{aligned} \alpha_i &= -\frac{d_i^T A e_i}{d_i^T d_i} \\ &= -\frac{d_i^T r_i}{d_i^T d_i} \end{aligned}$$

The result of this expression can be calculated. Note that for $d_i = r_i$ this equation is the exact same as the equation for the method of steepest descent. However, in this case the search directions d_i are still unknown and have to be constructed in a way that they are A-orthogonal. This can be achieved by a method called the Gram-Schmidt conjugation, which will be discussed in the next but one section.

7.3.2 Convergence Analysis

To prove that the method of conjugate directions really computes x within n steps, express the error as a linear combination of search directions:

$$e_0 = \sum_{j=0}^{n-1} \delta_j d_j$$

To prove convergence after n steps:

$$\begin{aligned} e_0 + \sum_{j=0}^{n-1} \alpha_j d_j &= 0 \\ \sum_{j=0}^{n-1} \delta_j d_j + \sum_{j=0}^{n-1} \alpha_j d_j &= 0 \\ \sum_{j=0}^{n-1} \delta_j d_j &= -\sum_{j=0}^{n-1} \alpha_j d_j \end{aligned}$$

By multiplying both sides with $d_i^T A$ one can see:

$$\begin{aligned} \sum_{j=0}^{n-1} \delta_j d_i^T A d_j &= -\sum_{j=0}^{n-1} \alpha_j d_i^T A d_j \qquad \rightarrow d_i^T A d_j = 0 \text{ if } i \neq j \\ \delta_i d_i^T A d_i &= -\alpha_i d_i^T A d_i \\ \delta_i &= -\alpha_i \end{aligned}$$

So the process of building up our solution component by component can be seen as cutting down our error component by component. This also shows that the error is A-orthogonal to all previous search directions and the residual is orthogonal to all previous search directions.

7.4 Method of Conjugate Gradients

7.4.1 Derivation

The last thing missing is the construction of the search directions d_i in such a way that they are A-orthogonal. It was mentioned earlier that this can be achieved by the means of the Gram-Schmidt conjugation. Given a set of n linearly independent vectors (like the residuals), the Gram-Schmidt conjugation works as follows:

$$d_0 = u_0$$
$$d_1 = u_1 - \beta_{10} d_0 \qquad \beta_{ij} = -\frac{u_i^T A d_j}{d_j^T A d_j}$$
$$d_2 = u_2 - \beta_{20} d_0 - \beta_{21} d_1$$

When using this method it is required to store all previous search directions for the complete runtime of the algorithm, leading to a runtime of $O(n^3)$ (n^2 per iteration, n iterations to convergence) and immense storage costs. However, in the method of steepest descent the new residual has been constructed by:

$$r_{j+1} = r_j - \alpha_j A r_j$$

If one applies the idea of the method of conjugate directions to this equation (walking along d_j instead of r_j):

$$r_{j+1} = r_j - \alpha_j A d_j \qquad \alpha_j = -\frac{d_j^T r_j}{d_j^T d_j}$$
$$r_i^T r_{j+1} = r_i^T r_j - \alpha_j r_i^T A d_j$$
$$\alpha_j r_i^T A d_j = r_i^T r_j - r_i^T r_{j+1} \qquad r_i^T r_j = 0 \qquad i \neq j$$
$$r_i^T A d_j = \begin{cases} \frac{1}{\alpha_i} r_i^T r_i & i = j \\ -\frac{1}{\alpha_{i-1}} r_i^T r_i & i = j+1 \\ 0 & \text{otherwise} \end{cases}$$
$$\beta_{ij} = \begin{cases} \frac{1}{\alpha_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} & i = j+1 \\ 0 & i > j+1 \end{cases}$$

This shows that the residual r_{i+1} is already A-orthogonal to all previous search directions except d_i. It is no longer necessary to store all old search directions to ensure A-orthogonality of the new search direction, only the previous search direction is required! Both space complexity and time complexity per iteration are reduced from $O(n^2)$ to $O(m)$, where m denotes the number of non-zero entries of A. One last simplication in the notation of β can be done:

$$\beta_i = \beta_{i,i-1} = \frac{r_i^T r_i}{d_{i-1}^T r_{i-1}} = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}$$

Finally: The method of conjugate gradients:

1. $d_0 = r_0 = b - A x_0$
2. $\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}$
3. $x_{i+1} = x_i + \alpha_i d_i \qquad r_{i+1} = r_i - \alpha_i d_i$
4. $\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$
5. $d_{i+1} = r_{i+1} + \beta_{i+1} d_i$

7.4.2 Convergence Analysis

In practical problems, n iterations are not feasible. The upper bound of the error of the conjugate gradient method is:

$$||e_i||_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i ||e_0||_A$$

To reduce the norm of the error by a factor of ϵ:

$$||e_i|| \leq \epsilon||e_0||$$
$$i \geq \frac{1}{2}\sqrt{\kappa}\ln\left(\frac{2}{\epsilon}\right)$$

8 Multigrid Method

In the following sections A^h, u^h, f^h describe A, u, f on a grid with discretization h.

8.1 Basics

$$
\begin{aligned}
A^h u^h &= f^h && \textit{LSE} \\
r^h &= f^h - A^h u^h && \textit{residual} \\
e^h &= u^h - u^* && \textit{error} \\
(u^h)^{k+1} &= (u^h)^k + I_{2h}^h (A^{2h})^{-1} I_h^{2h} r^h && \textit{CGC (coarse grid correction)}
\end{aligned}
$$

Multigrid uses complementary features:

- Convergence of relaxation
  - Fast for high frequency errors
  - Slow for low frequency errors
- Convergence of course grid correction
  - Divergent for high frequency errors
  - Quickly convergent for low frequency errors

8.2 Properties

- Multigrid is still an iterative method
- Huge advantage: The convergence rate is independent of h
- Caution: r is small in norm does not mean e is small in norm, however, if r is zero e is also zero $\rightarrow$ residual equation: $Ae = r$

8.3 Coarse Grid Correction

1. Compute residual: $r^h = f^h - A^h u^h$
2. Restrict: $f^{2h} = I_h^{2h} r^h$
3. Solve: $A^{2h} c^{2h} = f^{2h}$
4. Correct: $u^h = u^h + I_{2h}^h c^{2h}$

Where 3 can be performed by:

- Relaxation
- Direct solver
- Other iterative schemes
- Recursion $\rightarrow$ multigrid

8.4 Grid Layout

- Vertex-Centered
- Cell-Centered

8.5 Smoother Layout

- Multicolor smoothing (e.g. red-black for 5-point-stencil, 4-color for 9-point-stencil)
- Line smoothing / zebra smoothing (like multicolor but linewise instead of checkerboard)
- Chaotic smoothing (randomized)

8.6 Coarsening Layout

- Standard coarsening
  - Take every second point in both directions
  - Every second line **and** row is being dropped
- Semi coarsening
  - Take all points of every second line **or** row
  - Every second line **or** row is being droppeit
  - example: wings
- Red-Black coarsening
  - Results in checkerboard
- Unstructured grids
  - More complex strategies

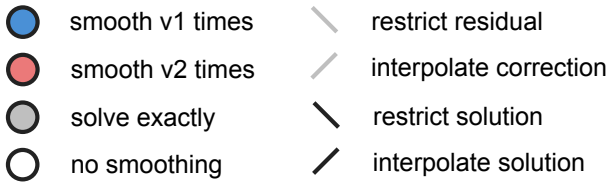
8.7 Restriction Strategies

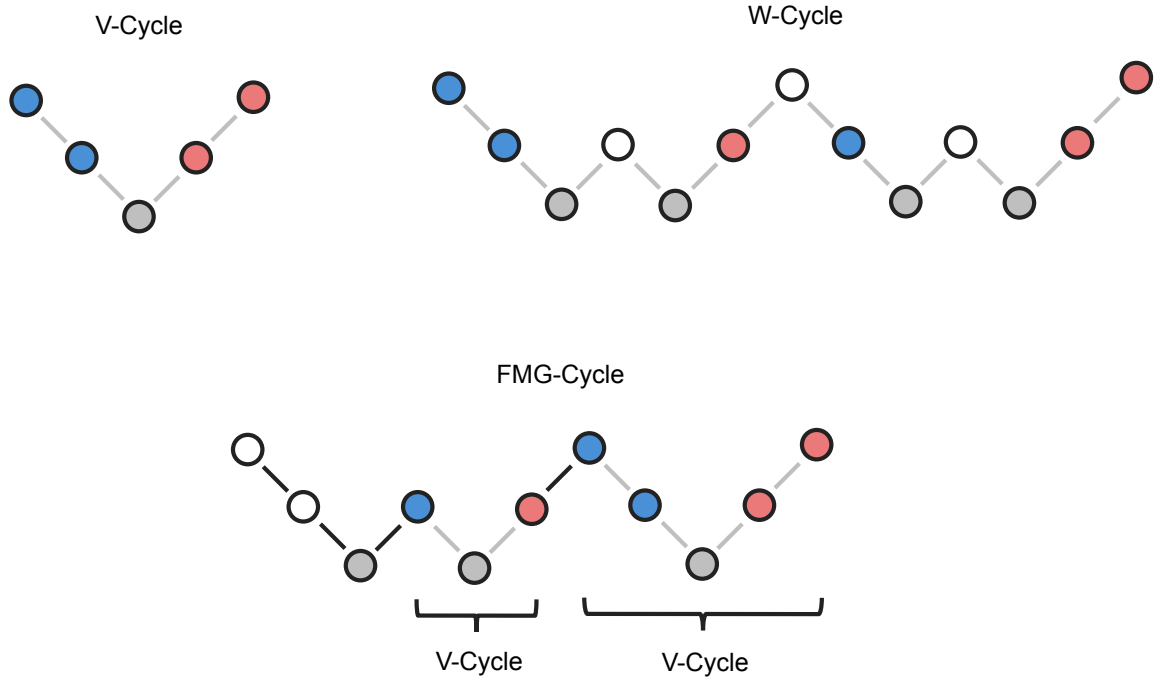
- Injection $(f^{2h})_{i,j} = (r^h)_{2i,2j}$
- Half injection $(f^{2h})_{i,j} = \frac{1}{2}(r^h)_{2i,2j}$
  $\rightarrow$ Useful for red-black smoother because it exploits the 0-residuals systematically
- Full weighting $f^{2h} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h^{2h} r^h$

8.8 Interpolation / Prolongation Strategies

- Linear interpolation
  $c^h = \frac{1}{2} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix}_{2h}^{h} c^{2h}$
  $c^h = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}_{2h}^{h} c^{2h}$
- Bilinear interpolation
  $c^h = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h}^{h} c^{2h}$

8.9 Multigrid Cycles





8.10 Detailed Complexity Analysis

$\gamma = 1$ for V-Cycle and 2 for W-Cycle
$c\,N$ complexity for one grid level

Computation:

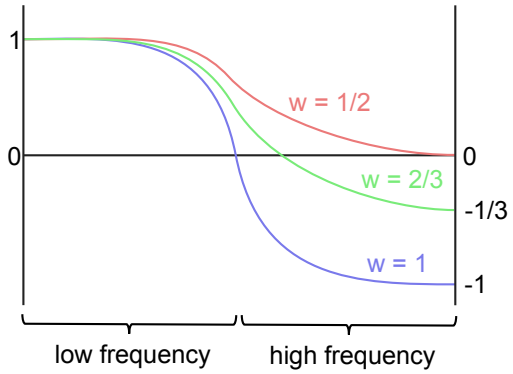
$$c\,(N + \frac{\gamma}{4}N + \frac{\gamma^2}{16}N + \frac{\gamma^3}{64}N + ...)$$
$$= c\,N(1 + \frac{\gamma}{4} + \frac{\gamma^2}{16} + \frac{\gamma^3}{64} + ...)$$
$$\approx c\,N\left(\frac{1}{1-\frac{\gamma}{4}}\right)$$

Storage:

8 bytes for u, f, r
$\Rightarrow c = N \cdot 24$ bytes
$24\,N\left(\frac{1}{1-\frac{\gamma}{4}}\right)$ bytes

8.11 Smoothing Factor

The value $max_{\frac{n}{2} \leq k \leq n-1} |\mu_k(\omega)|$ is called smoothing factor. It is ideal for $\omega = \frac{2}{3}$ and, in contrast to the convergence rate of the smoother, independent of h.



8.12 Boundary Conditions

- Dirichlet BCs
  1. Initialize grid
  2. Iterate **only** over the interiour grid points of the domain
- Neumann BCs (preferably used with cell-based grid)
  1. Add ghost layer
  2. Initialize grid
  3. Iterate over **all** grid points of the domain (except ghost layers)
  4. Loop over all Neumann BCs (copy to ghosts)

8.13 Extensions

- Full Approximation Storage (FAS)
- τ-Extrapolation

9 Störmer-Verlet Method

9.1 Properties

- Accuracy $O(\delta t^2)$
- Symplectic (energy conserving)
- Time reversible
- Simple basic algorithm but very complex to parallelize

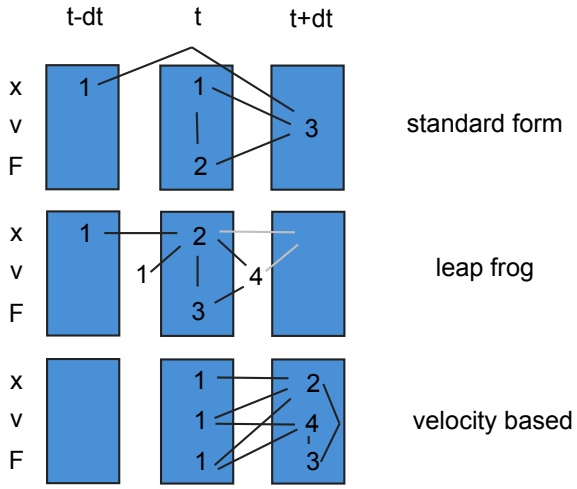
9.2 Boundary Conditions

- Periodic
- Reflecting
- Outflow
- Inflow
- Prescribed temperature

9.3 Complexity

For the interaction of N particles with each other MD has a complexity of $O(N^2)$ for each timestep. For some forces one can define cutoff distances to avoid having to calculate the interaction between all particles of the domain. This does not work for gravity because it only falls off with a factor of $\frac{1}{r^2}$, the sum of the gravity force over the sphere surface ($\sim r^2$) is independent of r.

9.4 Algorithm Variants



Velocity Störmer-Verlet

$$x_i^{n+1} = x_i^n + \Delta t\, v_i^n + \frac{F_i^n\, \Delta t^2}{2m_i}$$

$$v_i^{n+1} = v_i^n + \frac{(F_i^n + F_i^{n+1})\Delta t}{2m_i}$$

9.5 Extensions

- Cutoff radius
- Linked cell structure

10 Lattice-Boltzmann Method

10.1 Basics

LBM originated from the lattice gas automata (LGA), but it uses positive real numbers to represent some average amount of particles travelling at a lattice point in any of the lattice directions instead of the discrete positive integers of the LGA. Mathematically they are expected values of the statistical distribution functions for the lattice point.

Discrete Boltzmann equation:

$$f_i(x + c_i\Delta t, t + \Delta t) - f_i(x, t) = -w(f_i - f_i^{eq}) \qquad \text{with } w = \frac{1}{\tau} \text{ being the collision frequency}$$

Collision step:

$$\tilde{f}_i(x, t + \Delta t) = f_i(x, t) - w(f_i - f_i^{eq})$$

Streaming step:

$$f_i(x + c_i\Delta t, t + \Delta t) = \tilde{f}_i(x, t + \Delta t)$$

$$f_i^{eq} = t_i\rho\left(1 + \frac{3}{c^2}(c_i u) + \frac{9}{2c^4}(c_i u)^2 - \frac{3u^2}{2c^2}\right) \qquad \text{standard formulation}$$

$$f_i^{eq} = t_i\left(\rho + \frac{3}{c^2}(c_i u) + \frac{9}{2c^4}(c_i u)^2 - \frac{3u^2}{2c^2}\right) \qquad \text{special formulation for incompressible fluids}$$

with **density** $\rho = \sum_{i=0}^{N-1} f_i$ and **momentum density** $\rho u = \sum_{i=0}^{N-1} f_i c_i$

10.2 Properties

- Simple algorithm
- Easy to parallelize
- Easy to adapt for
  - Complicated geometries
  - Time-varying geometries
  - Free surfaces
  - Additional physical or chemical effects
- Very compute-intensive

10.3 Data Layouts

The LB method always requires two grids because the update of node n overwrites data that is still required for the following nodes. However, there are two possible data layouts, each with advantages and disadvantages.

- Collision optimized
  - Store all values of one cell consecutively
- Streaming optimized
  - Store one value of all cells consecutively

10.4 Streaming Techniques

- Stream-Collide
  - Pull node values
  - Irregular read access for collision-optimized data layout
  - Collide after pulling

- Collide-Stream
  - Push node values
  - Irregular write access for collision-optimized data layout (worse than irregular read)
  - Streaming before pushing

10.5 Boundary Conditions

- No-slip (bounce back)
- Moving no-slip (modified bounce back)

11 Proofs and Footnotes

Differential Operators

$$u : \text{scalar field}$$
$$v : \text{vector field}$$

Gradient / Nabla Operator

$$\text{grad } u = \nabla u = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$$
$$\text{grad } v = \nabla v = \begin{pmatrix} \nabla v_1 \\ \nabla v_2 \\ \nabla v_3 \end{pmatrix} = \begin{pmatrix} v_{1x} & v_{2x} & v_{3x} \\ v_{1y} & v_{2y} & v_{3y} \\ v_{1z} & v_{2z} & v_{3z} \end{pmatrix}$$

Laplace Operator

$$\Delta u = \nabla^2 u = \left(u_{xx} + u_{yy} + u_{zz} \right)$$
$$\Delta v = \nabla^2 v = \begin{pmatrix} \Delta v_1 \\ \Delta v_2 \\ \Delta v_3 \end{pmatrix} = \begin{pmatrix} v_{1xx} + v_{1yy} + v_{1zz} \\ v_{2xx} + v_{2yy} + v_{2zz} \\ v_{3xx} + v_{3yy} + v_{3zz} \end{pmatrix}$$

Divergence

$$\text{div } v = \nabla^T v = \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y} + \frac{\partial v_3}{\partial z}$$

Norms

Euclidean Norm

$$||x||_n = \sqrt[n]{\sum_i |x_i|^n}$$

$$\text{L0-norm by definition} : ||x||_0 = \sqrt[0]{\sum_i |x_i|^0}$$
$$\text{L0-norm in practice} : ||x||_0 = \#(i|x_i \neq 0) \text{ (the number of non-zero elements)}$$

$$\text{L1-norm} : ||x||_1 = \sum_i |x_i|$$
$$\text{absolute distance} : ||\tilde{x} - x||_1 = \sum_i |\tilde{x_i} - x_i|$$
$$\text{mean absolute error} : \frac{1}{n} \sum |\tilde{x_i} - x_i|$$

$$\text{L2-norm} : ||x||_2 = \sqrt{\sum_i x_i^2}$$
$$\text{squared distance} : ||\tilde{x} - x||_2 = \sum_i (\tilde{x_i} - x_i)^2$$
$$\text{mean squared error} : \frac{1}{n} \sum (\tilde{x_i} - x_i)^2$$

Maximum Norm

$$||x||_\infty = max_i(|x_i|)$$

Integration Theorems

Integration by Parts

$$\int_\Omega u'' v \, \mathrm{d}x = u' v|_\Omega - \int_\Omega u' v' \, \mathrm{d}x \tag{1}$$

Divergence Theorem

$$\int_\Omega \nabla u \, \mathrm{d}x = \int_\Gamma u \mathbf{n} \, \mathrm{d}s \tag{2}$$

Change of Coordinates

$$\int_{\varphi(x)} f(x) \, \mathrm{d}x = \int_x f(\varphi(x)) \cdot |\det J(x)| \, \mathrm{d}x \tag{3}$$

Conjugate Gradient Method

Line Search

$$\begin{aligned} r_1^T r_0 &= 0 \\ (b - Ax_1)^T r_0 &= 0 \\ (b - A(x_0 + \alpha r_0))^T r_0 &= 0 \\ (b - Ax_0 - \alpha A r_0)^T r_0 &= 0 \\ (r_0 - \alpha A r_0)^T r_0 &= 0 \\ r_0^T r_0 - (\alpha A r_0)^T r_0 &= 0 \\ r_0^T r_0 &= (\alpha A r_0)^T r_0 \\ r_0^T r_0 &= \alpha r_0^T A r_0 \\ \alpha &= \frac{r_0^T r_0}{r_0^T A r_0} \end{aligned} \tag{4}$$

Error Decomposition

$$\begin{aligned} r_i = -Ae_i &= \sum_{j=1}^{n} \xi_j \lambda_j v_j \\ x_{i+1} &= x_i + \alpha_i r_i \\ e_{i+1} &= e_i + \alpha_i r_i \end{aligned} \tag{5}$$

Because we have orthonormal eigenvectors:

$$v_j^T v_k = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$$

Therefore the equation for α becomes:

$$\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i}$$

$$= \frac{\left(\sum_{j=1}^n \xi_j \lambda_j v_j^T\right)\left(\sum_{j=1}^n \xi_j \lambda_j v_j\right)}{\left(\sum_{j=1}^n \xi_j \lambda_j v_j^T\right) A \left(\sum_{j=1}^n \xi_j \lambda_j v_j\right)} \qquad \to v_j^T v_k = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$$

$$= \frac{\sum_{j=1}^n \xi_j^2 \lambda_j^2}{\left(\sum_{j=1}^n \xi_j \lambda_j v_j^T\right) A \left(\sum_{j=1}^n \xi_j \lambda_j v_j\right)} \qquad \to v_j^T A v_k = \begin{cases} \lambda_j & j = k \\ 0 & j \neq k \end{cases}$$

$$= \frac{\sum_{j=1}^n \xi_j^2 \lambda_j^2}{\sum_{j=1}^n \xi_j^2 \lambda_j^3}$$

Plugging this into the equation for the error:

$$e_{i+1} = e_i + \alpha_i r_i = e_i + \frac{\sum_{j=1}^{n} \xi_j^2 \lambda_j^2}{\sum_{j=1}^{n} \xi_j^2 \lambda_j^3} r_i$$

Energy Norm

$$\begin{aligned}
||e_{i+1}||_A^2 &= e_{i+1}^T A e_{i+1} \\
&= (e_i^T + \alpha_i r_i^T) A (e_i + \alpha_i r_i) \\
&= e_i^T A e_i + 2\alpha_i r_i^T A e_i + \alpha_i^2 r_i^T A r_i \qquad \text{(by symmetry of } A\text{)} \\
&= ||e_i||_A^2 + 2\frac{r_i^T r_i}{r_i^T A r_i}(-r_i^T r_i) + \left(\frac{r_i^T r_i}{r_i^T A r_i}\right)^2 r_i^T A r_i \\
&= ||e_i||_A^2 - \frac{(r_i^T r_i)^2}{r_i^T A r_i} \\
&= ||e_i||_A^2 \left(1 - \frac{(r_i^T r_i)^2}{(r_i^T A r_i)(e_i^T A e_i)}\right) \\
&= ||e_i||_A^2 \underbrace{\left(1 - \frac{(\sum_j \xi_j^2 \lambda_j^2)^2}{(\sum_j \xi_j^2 \lambda_j^3)(\sum_j \xi_j^2 \lambda_j)}\right)}_{\omega^2} \\
&= \omega^2 ||e_i||_A^2
\end{aligned}$$

[6]

12 Literature and Sources

- "Simulation and Scientific Computing I" by Marc Avila
- "Simulation and Scientific Computing II" by Ulrich Rüde, Christoph Pflaum, Klaus Iglberger
- "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by Jonathan Richard Shewchuk
- "Jacobi Method" on Wikipedia
  http://en.wikipedia.org/wiki/Jacobi_method
- "Finite Difference Method" on Wikipedia
  http://en.wikipedia.org/wiki/Finite_difference_method
- "Finite Element Method" on Wikipedia
  http://en.wikipedia.org/wiki/Finite_element_method
- "Gauss-Seidel Method" on Wikipedia
  http://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method
- "Successive Over-Relaxation" on Wikipedia
  http://en.wikipedia.org/wiki/Successive_over-relaxation
- "Conjugate Gradient Method" on Wikipedia
  http://en.wikipedia.org/wiki/Conjugate_gradient_method
- "Multigrid Method" on Wikipedia
  http://en.wikipedia.org/wiki/Multigrid_method
- "Molecular Dynamics" on Wikipedia
  http://en.wikipedia.org/wiki/Molecular_dynamics
- "Lattice Boltzmann Method" on Wikipedia
  http://en.wikipedia.org/wiki/Lattice_Boltzmann_methods