

Physically Based Simulation

Alexander Schlögl

February 4, 2018

Contents

1	Introduction	3
2	Mass Spring Systems	3
2.1	External Forces f^{Ext}	3
2.2	Internal Forces f^{Int}	3
2.3	Spring Topologies and Parameters	4
2.4	Equations of Motion	4
2.5	Numerical Integration Methods	4
	Euler Method	5
	Heun's Method	5
	Midpoint Method	6
	Runge-Kutta Methods	6
	Verlet Method	7
	Leapfrog Method	7
	Implicit Euler	7
2.6	Test Equation	8
2.7	Collision Handling	8
2.8	Problems of Mass Spring Systems	8
3	Solving PDEs with Finite Elements	8
3.1	Types of Differential Equations	8
	Ordinary Differential Equations (ODEs)	8
	Partial Differential Equations (PDEs)	9
	Classification of Second Order PDEs	9
3.2	Initial and Boundary Conditions	9
3.3	Solving PDEs	9
3.4	Finite Element Method	9
	Piecewise Approximation	10
	Derivation of the Element Equation	11
	Linear Triangulation	11
	Assembling the Stiffness Matrix	12
	Right-Hand Side	12

4 Solving Systems of Equations	12
4.1 Jacobi Method	12
4.2 Solving an Equivalent Problem	13
Steepest Gradient Method	14
Conjugate Gradient Method	14
4.3 FEM summary	16

This is **my interpretation** of the lecture slides. I tried to be very verbose and explain everything, all while removing irrelevant parts from the lecture. Using this you should be able to pass the lecture easily. **However, I do not take responsibility for any bad results and will not be blamed from anyone. This was a lot of work and I did it to save others (especially students of following semesters) from having to do this themselves. Use this summary at your own responsibility.** If you have any feedback, feel free to create an issue on the git. I don't promise I will fix anything, but I will try.

1 Introduction

Physically based simulation is exactly what it sounds like. Creating accurate visualizations of real world objects by simulating the underlying physical equations. While doing so, a number of shortcuts is taken to speed up computation time. We do not need very accurate results, we need results that **look accurate**.

2 Mass Spring Systems

Mass Spring Systems are a simple way of simulating deformation, and they are often used for hair, clothes etc. The main steps of employing mass spring systems are:

1. Discretizing the object into mass points and connecting them with springs
2. Setting up the equations of motion
3. Discretizing the equations of motion in time
4. Determining internal and external forces acting on mass points
5. Solving the equations numerically

The location and mass distribution of the discretized points has a large influence on the accuracy of the simulation. The simplest approximation is just giving all points equal mass, but better methods exist. Each point has a position \mathbf{x}_i , a velocity \mathbf{v}_i and a mass m_i . The points are then connected with springs. The spring topology also heavily influences the accuracy, and certain heuristics exist. Each spring stores a stiffness k_q , rest length L_q and current length l_q .

2.1 External Forces f^{Ext}

External forces include gravity ($f_i^G = m_i [0 \ 0 \ 9.81]^T m/s^2$), friction and collisions.

2.2 Internal Forces f^{Int}

The internal forces are the forces exerted by the springs, as well as (viscous) damping. Spring force is given by Hooke's law:

$$f = k(L - l) \quad (1)$$

or in 3D

$$\mathbf{f}_{ij} = k(L - \|\mathbf{x}_i - \mathbf{x}_j\|) \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|} \quad (2)$$

for multiple springs connected to a single mass point, just sum up all the individual spring forces.

2.3 Spring Topologies and Parameters

In general springs are connected in three different ways. Structural springs connect neighboring points and oppose stretching; diagonal strings oppose shearing and interleaving springs oppose bending. For cloth or other objects that need to be bent, springs that skip multiple points can be used.

The stiffness for the springs can be calculated with existing heuristics or be set in such a way that they accurately represent known deformations. Usually manual tuning is required.

2.4 Equations of Motion

As the mass of the points, as well as the forces acting on them is known, we can simulate them using Newton's laws of motion. From Newton's second law follows

$$m_i \frac{d^2 \mathbf{x}_i(t)}{dt^2} = \mathbf{f}_i^{Int}(t) + \mathbf{f}_i^{Ext}(t) \quad (3)$$

with which we can calculate the acceleration and update the velocity of the mass point. Equation (3) is a first order ordinary differential equation (ODE), which we can solve and the numerically integrate. For the integration step we differentiate between explicit methods (where all required quantities are known) and implicit methods (where we need to solve a system of equations).

2.5 Numerical Integration Methods

As we actually need two integrations for our systems (one for \mathbf{v} and one for \mathbf{x}) we need two separate integration steps. For this we simply introduce a variable for velocity and integrate it to get the new position.

All integration methods are derived from Taylor series expansion, and some use multiple Taylor series or more parts of it to reach higher accuracy. As a reminder, the Taylor series for $y(x)$ about development point t is

$$y(x) = \sum_{n=0}^{\infty} \frac{y^{(n)}(t)}{n!} (x - t)^n \quad (4)$$

and if we evaluate this at point $t + h$ we get

$$y(t + h) = y(t) + y'(t)h + \frac{y''(t)}{2!}h^2 + \frac{y'''(t)}{3!}h^3 + \dots \quad (5)$$

From that we can derive our numerical integration methods and their accuracy.

Euler Method

Forward Euler is the simplest integration method. It is derived by using the first two summands of the Taylor series, giving us the following equation

$$y(t+h) = y(t) + y'(t)h + \mathcal{O}(h^2) \quad (6)$$

with our error being a function in the realm of $\mathcal{O}(h^2)$.

The following calculations are done for the Forward Euler:

- $\mathbf{x}(t+h) = \mathbf{x}(t) + h \cdot \mathbf{v}(t)$
- $\mathbf{f}(t) = \mathbf{f}^{Int}(t) + \mathbf{f}^{Ext}(t)$
- $\mathbf{a}(t) = \frac{1}{m}(\mathbf{f}(t) - \gamma \mathbf{v}(t))$ (where γ is the damping factor)
- $\mathbf{v}(t+h) = \mathbf{v}(t) + h \cdot \mathbf{a}(t)$

In order to improve behaviour of the Euler method, we can either directly reuse new positions

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t) \quad (7)$$

$$\mathbf{v}(t+h) = \mathbf{v}(t) + h\mathbf{a}(\mathbf{x}(t+h), \mathbf{v}(t)) \quad (8)$$

or directly reuse the new velocities

$$\mathbf{v}(t+h) = \mathbf{v}(t) + h\mathbf{a}(\mathbf{x}(t), \mathbf{v}(t)) \quad (9)$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t+h) \quad (10)$$

Using these improvements is known as the Symplectic Euler.

Heun's Method

This integrator uses more summands in its approximation:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \mathcal{O}(h^3) \quad (11)$$

As we don't know the second derivative, we approximate it with forward differences:

$$y''(t) = \frac{y'(t+h) - y'(t)}{h} + \mathcal{O}(h) \quad (12)$$

By plugging Equation (12) into Equation (11) we get

$$y(t+h) \approx y(t) + hy'(t) + \frac{h^2}{2} \left(\frac{y'(t+h) - y'(t)}{h} \right) \quad (13)$$

$$\approx y(t) + hy'(t) + \frac{h}{2}y'(t+h) - \frac{h}{2}y'(t) \quad (14)$$

$$\approx y(t) + \frac{h}{2}(y'(t) + y'(t+h)) \quad (15)$$

As we don't know $y'(t + h)$ yet when we evaluate this, we approximate it with a regular Euler step.

Heun's method uses the average of the current velocity and the next velocity for the update step

The implementation steps for Heun's method are the following

- $\mathbf{a}(t) = \frac{1}{m}(\mathbf{f}(t) - \gamma\mathbf{v}(t))$
- $\tilde{\mathbf{v}}(t + h) = \mathbf{v}(t) + h\mathbf{a}(t)$
- $\tilde{\mathbf{x}}(t + h) = \mathbf{x}(t) + h\tilde{\mathbf{v}}(t)$
- $\tilde{\mathbf{a}}(t + h) = \frac{1}{m}(\tilde{\mathbf{f}}(t + h) - \gamma\tilde{\mathbf{v}}(t + h))$
- $\mathbf{x}(t + h) = \mathbf{x}(t) + h\frac{\mathbf{v}(t) + \tilde{\mathbf{v}}(t + h)}{2}$
- $\mathbf{v}(t + h) = \mathbf{v}(t) + h\frac{\mathbf{a}(t) + \tilde{\mathbf{a}}(t + h)}{2}$

Midpoint Method

Heun's method averages $y'(t)$ and $y'(t + h)$. The midpoint method only determine y' at $t + h/2$, giving us the same accuracy with less work. We approximate $y'(t + h/2)$ with an Euler step.

For our mass spring use case we need to approximate both $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{a}}$ (and thus $\tilde{\mathbf{x}}$) for $t + h/2$.

Runge-Kutta Methods

These are more general cases of the Midpoint method. They increase accuracy by performing intermediate steps. The most common variant is RK4. Due to the increased accuracy, larger timesteps can be used, but this increases computational overhead.

RK4 works as follows:

$$k_1 = f(t, y(t)) \quad (16)$$

$$k_2 = f(t + h/2, y(t) + h/2k_1) \quad (17)$$

$$k_3 = f(t + h/2, y(t) + h/2k_2) \quad (18)$$

$$k_4 = f(t, y(t) + hk_3) \quad (19)$$

The value for $y(t + h)$ is calculated using a weighted average

$$y(t + h) = y(t) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (20)$$

Implementation of RK4 (or any RK method) should be pretty clear.

Verlet Method

For this method we start with two Taylor series expansions:

$$y(t+h) = y(t) + y'(t)h + \frac{y''(t)}{2!}h^2 + \frac{y'''(t)}{3!}h^3 + \mathcal{O}(h^4) \quad (21)$$

$$y(t-h) = y(t) - y'(t)h + \frac{y''(t)}{2!}h^2 - \frac{y'''(t)}{3!}h^3 + \mathcal{O}(h^4) \quad (22)$$

then, by summing them up we get

$$y(t+h) = 2y(t) - y(t-h) + h^2y''(t) + \mathcal{O}(h^4) \quad (23)$$

As you can see the velocity terms cancels out, leading to the typical assumption that the force term is independent of velocity (no damping). The Verlet method is a symplectic integrator (it has nice energy conservation properties in Hamiltonian systems) and is also time reversible.

The velocity term here is never explicitly calculated. If we do need it (e.g. for kinetic energy) we need to approximate it either via backward or central differences.

Leapfrog Method

The main goal of this method is to improve the velocity estimate. It is similar to the Verlet method but from the Taylor series for velocity:

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \mathbf{v}'(t)h + \frac{\mathbf{v}''(t)}{2!}h^2 + \frac{\mathbf{v}'''(t)}{3!}h^3 + \mathcal{O}(h^4) \quad (24)$$

$$\mathbf{v}(t-h) = \mathbf{v}(t) - \mathbf{v}'(t)h + \frac{\mathbf{v}''(t)}{2!}h^2 - \frac{\mathbf{v}'''(t)}{3!}h^3 + \mathcal{O}(h^4) \quad (25)$$

and by subtracting we get

$$\mathbf{v}(t+h) = \mathbf{v}(t-h) + 2\mathbf{v}'(t)h + \mathcal{O}(h^3) \quad (26)$$

The Leapfrog method then employs shifted updates, with \mathbf{v} being updated for $t + h/2$ and \mathbf{x} and \mathbf{a} being updated for t . This gives us an error of only $\mathcal{O}(h^3)$ with only one force evaluation. We need to initialize $\mathbf{v}(t_{1/2})$ with e.g. an Euler step.

Implicit Euler

The Implicit Euler formulates the update equation like as follows

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1}) \quad (27)$$

Due to this, we have to solve for an unknown in every update step. One option is the **fixed-point iteration**

$$y_{n+1}^{(k+1)} = y_n + h \cdot f(t_{n+1}, y_{n+1}^{(k)}) \quad (28)$$

which requires initialization such as $y_{n+1}^{(0)} = y_n$ or $y_{n+1}^{(0)} = y_n + h \cdot f(t_n, y_n)$.

Another option would be **Newton's Method**.

2.6 Test Equation

We can study stability behaviour of a solver via a linear test equation (Dahlquist's equation):

$$y'(t) = \lambda \cdot y(t) \quad (29)$$

which gives us the following solution for $y(t)$ (assuming $\lambda \in \mathbb{R}$)

$$y(t) = e^{\lambda t} y_0 \quad (30)$$

An example for the forward Euler: The update step is

$$y_{n+1} = y_n + h \cdot y'_n \quad (31)$$

by inserting the test equation we get

$$y_{n+1} = (1 + \lambda h) y_n \quad (32)$$

which with recursive evaluation gives us

$$y_{n+1} = (1 + \lambda h)^{n+1} y_0 \quad (33)$$

This shows that the forward Euler is unconditionally unstable if $\lambda > 0$ and stable iff $-2 < h\lambda < 0$.

2.7 Collision Handling

When using mass spring systems, a simple form of collision detection and handling can be required. For this, penalty forces are used. In the penalty forces approach any penetration of an object into another creates a spring force separating both objects. This approach is only an approximation, but can be very useful due to its simplicity.

2.8 Problems of Mass Spring Systems

The behaviour of the system is topology dependent. Also, we do not have an explicit notion of volume preservation, which can result in systems collapsing on themselves. This is again dependent on their topology.

3 Solving PDEs with Finite Elements

Differential equations relate an unknown functions with its derivatives. They are typically employed to describe physical laws and phenomena, e.g. growth or decay.

3.1 Types of Differential Equations

Ordinary Differential Equations (ODEs)

These describe an unknown function only based on derivatives with respect to a single variable.

Partial Differential Equations (PDEs)

Describe an unknown function based on its derivatives with respect to multiple variables. The **order** n of a PDE is given by its highest derivative. PDEs are called linear if the coefficients of the function and the derivatives are independent of the latter (e.g. no multiplication between them).

Classification of Second Order PDEs

Physical phenomena are often modeled as second order linear PDEs of the form

$$a \cdot u_{xx} + b \cdot u_{xy} + c \cdot u_x + e \cdot u_y + k \cdot u = g(x, y) \quad (34)$$

Then, based on the coefficients we differentiate three classes:

- **Hyperbolic:** $b^2 - 4ac > 0$
- **Parabolic:** $b^2 - 4ac = 0$
- **Elliptic:** $b^2 - 4ac < 0$

3.2 Initial and Boundary Conditions

In order to find a specific solution to the PDE we need to have some value to start from. This can be either done by **initial conditions**, which specify a certain physical state at start time, or **boundary conditions**, which impose a state on the boundary $\delta\Omega$ of the solution domain Ω .

3.3 Solving PDEs

Unfortunately, closed form analytical solutions exist only for very simple problems. This makes numerical analysis in order to get an approximate solution necessary. Typical solution methods are **finite differences** and **finite elements**. Only the finite element method is described in the lecture slides.

3.4 Finite Element Method

The key idea is to divide the domain into many small elements and then approximate the true solution in each element via a set of simple equations. After the individual equations are solved, they are combined into a global system for the final solution for the given boundary conditions.

Piecewise Approximation

In order to make the numerical solving easier, we approximate the target function via a linear combination of basis functions. Given a set of discrete evaluations of the function f , we want to find a way to get the values between those points for a continuous evaluation. As linear interpolation is a very intuitive form of doing so, we want a simple way of calculating this. This is done via "hat" functions, which can be seen in Figure 1.

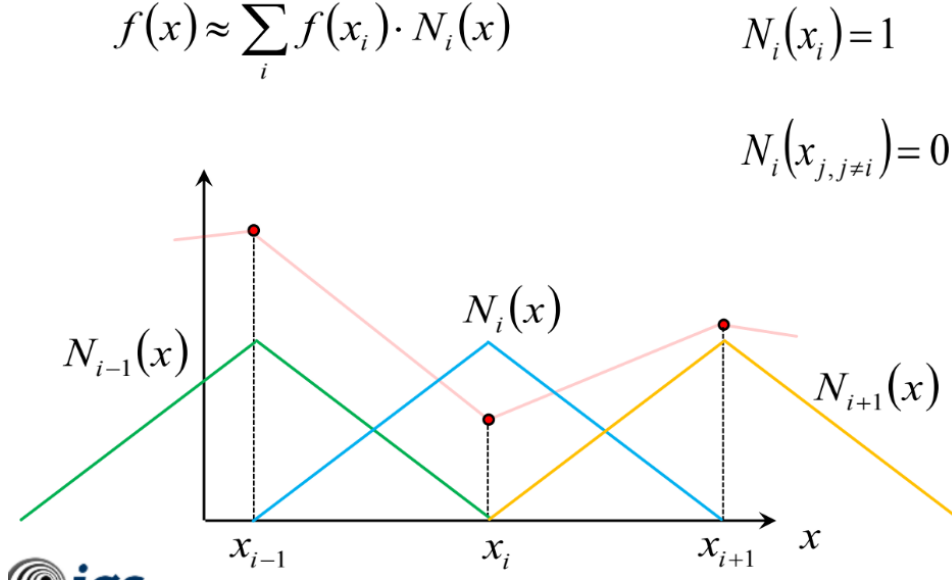


Figure 1: An example of interpolation via hat functions

For the finite element method we utilize the Ritz-Galerkin approach, which approximates the true solution with a linear combination of basis functions:

$$u \approx \sum_i u_i \cdot N_i \quad (35)$$

with unknowns u_i . We then select an arbitrary test function as linear combination of the basis functions

$$v \approx \sum_j N_j \quad (36)$$

which gives us the following approximations of the derivatives

$$\nabla u \approx \sum_i u_i \cdot \nabla N_i \quad \nabla v \approx \sum_j \nabla N_j \quad (37)$$

Derivation of the Element Equation

We start from the 2D Poisson equation on a domain Ω with Dirichlet boundary conditions (a fixed value on the domain boundary)

$$\Delta u(x, y) = f(x, y), \quad u(x, y) = 0 \text{ for } (x, y) \in \delta\Omega \quad (38)$$

Then we multiply with a sufficiently smooth (differentiable twice) test function $v(x, y)$ with $v(x, y) = 0$ on boundary $\delta\Omega$.

$$\Delta u(x, y)v(x, y) = f(x, y)v(x, y) \quad (39)$$

This we integrate over the domain

$$\int_{\Omega} \Delta u(x, y)v(x, y)d\Omega = \int_{\Omega} f(x, y)v(x, y)d\Omega \quad (40)$$

Then we derive the **weak form** of the problem by applying the Green-Gauss theorem (similar to integrating by parts)

$$\int_{\Omega} \nabla u(x, y) \nabla v(x, y)d\Omega = \int_{\Omega} f(x, y)v(x, y)d\Omega \quad (41)$$

After inserting the Ritz-Galerkin approximation into the weak form we get

$$\int_{\Omega} \sum_i u_i \cdot \nabla N_i \cdot \sum_j \nabla N_j d\Omega = \int_{\Omega} f \cdot \sum_j N_j d\Omega \quad (42)$$

$$\sum_i u_i \int_{\Omega} \nabla N_i \cdot \sum_j \nabla N_j d\Omega = \int_{\Omega} f \cdot \sum_j N_j d\Omega \quad (43)$$

and by splitting this into a set of single equations we get

$$\sum_i u_i \int_{\Omega} \nabla N_i \cdot \nabla N_j d\Omega = \int_{\Omega} f \cdot N_j d\Omega \quad \forall j = 1, \dots, n \quad (44)$$

We can write this system of equations as a matrix, which is symmetric, positive definite and banded. The so called **stiffness matrix** K can be solved with an iterative solver, e.g. conjugate gradient method. Due to the sparsity of the stiffness matrix it is easiest assembled element-wise.

Linear Triangulation

In 2D we want our linear triangular elements to be composed of three nodes with three linear basis functions:

$$N_i(x, y) = c_{i,0} + c_{i,1}x + c_{i,2}y \quad i = 1, 2, 3 \quad (45)$$

It is important to note that these basis functions have local support, ie. they are only non-zero for adjacent points.

Assembling the Stiffness Matrix

We want to evaluate the following equations

$$K_{ij} = \int_{\Omega} \nabla N_i \cdot \nabla N_j d\Omega \quad (46)$$

$$K_{ij} = \int_{\Omega} \left(\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y} \right) \cdot \left(\frac{\partial N_j}{\partial x}, \frac{\partial N_j}{\partial y} \right) dx dy \quad (47)$$

$$K_{ij} = \int_{\Omega} \frac{\partial N_i}{\partial x} \cdot \frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y} \cdot \frac{\partial N_j}{\partial y} dx dy \quad (48)$$

The basis function N_i extends over all elements adjacent to node i and is zero outside of Ω_i . $K_{ij} \neq 0$ if there exists an element (triangle) containing nodes i and j .

Right-Hand Side

We assemble the right hand side with similar assumptions as before:

$$f_j = \sum_{\text{adjacent}} \int_{\Omega_e} f \cdot N_j d\Omega_e \quad (49)$$

We can approximate the integral with a one point quadrature

$$\int_{\Omega_e} f \cdot N_j d\Omega_e \approx A_e \cdot f(x_c, y_c) \cdot N_j(x_c, y_c) \quad (50)$$

at barycenter (x_c, y_c) in order to save computation time.

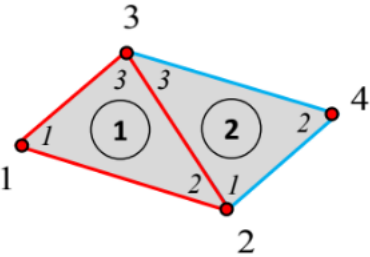
Figure 2 shows an example of a system setup.

4 Solving Systems of Equations

For calculating numerical solutions for our FEM systems we need an efficient way of solving systems of equations. These equations can be written as $\mathbf{Ax} = \mathbf{b}$. We could solve this by inverting the coefficient matrix \mathbf{A} , but that takes too long ($\mathcal{O}(n^3)$). To this end we use iterative solvers, which work more efficiently.

4.1 Jacobi Method

The Jacobi method only converges for diagonally dominant system matrices, but this is usually a given in physical applications. The basic approach of it is rearranging the



$$\begin{bmatrix}
 K_{11}^{(1)} & K_{12}^{(1)} & K_{13}^{(1)} & 0 \\
 K_{21}^{(1)} & K_{22}^{(1)} + K_{11}^{(2)} & K_{23}^{(1)} + K_{13}^{(2)} & K_{12}^{(2)} \\
 K_{31}^{(1)} & K_{32}^{(1)} + K_{31}^{(2)} & K_{33}^{(1)} + K_{33}^{(2)} & K_{32}^{(2)} \\
 0 & K_{21}^{(2)} & K_{23}^{(2)} & K_{22}^{(2)}
 \end{bmatrix}
 \begin{bmatrix}
 u_1 \\
 u_2 \\
 u_3 \\
 u_4
 \end{bmatrix}
 =
 \begin{bmatrix}
 f_1^{(1)} \\
 f_2^{(1)} + f_1^{(2)} \\
 f_3^{(1)} + f_3^{(2)} \\
 f_2^{(2)}
 \end{bmatrix}$$

K · u = f

Figure 2: Example setup

individual equations:

$$\sum_{j=1}^n a_{ij} x_j = b_i \quad (51)$$

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j \right) \quad (52)$$

From this we derive our update rule

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{(k)} \right) \quad (53)$$

or in matrix notation

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1} (\mathbf{b} - (\mathbf{A} - \mathbf{D}) \mathbf{x}^{(k)}) \quad (54)$$

4.2 Solving an Equivalent Problem

This method is especially effective for sparse matrices, and requires that the matrix \mathbf{A} be symmetric fulfill

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \quad \forall \mathbf{x} \neq \mathbf{0} \quad (55)$$

The method is then based on solving an equivalent problem, minimizing the function

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{b} \quad (56)$$

The first and second derivatives of the function are then

$$\nabla f(\mathbf{x}) = \mathbf{A} \mathbf{x} - \mathbf{b}, \quad \nabla^2 f(\mathbf{x}) = \mathbf{A} \quad (57)$$

The function f is a parabola with a single global minimum. This global minimum contains our solution, because there $\mathbf{A} \mathbf{x}^* - \mathbf{b} = 0$, so finding \mathbf{x}^* equals finding our solution. We try to find the solution via line searching.

Our iteration step is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)} \quad (58)$$

with initial guess $\mathbf{x}^{(0)}$, search direction $\mathbf{p}^{(k)}$ and step length $\alpha^{(k)}$. The question now is how to find the most efficient search direction and step length. We approximate the error in each step $\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$ with the residual of our approximation of the right-hand side

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A} \mathbf{x}^{(k)} \quad \rightarrow \quad \mathbf{r}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) \quad (59)$$

One can also note that $\mathbf{r}^{(k)} = \mathbf{A} \mathbf{e}^{(k)}$.

Steepest Gradient Method

The gradient determines the direction of greatest increase, so we step into the direction of the negative gradient

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)} \quad (60)$$

$$\mathbf{p}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) = \mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A} \mathbf{x}^{(k)} \quad (61)$$

We find the step size α by minimizing the next function value, which gives us the following formula for α

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T \mathbf{A} \mathbf{r}^{(k)}} \quad (62)$$

If the matrix \mathbf{A} is ill-conditioned, this method is slow converging. A better way of finding a search direction is needed.

Conjugate Gradient Method

The key idea here is to select a number of \mathbf{A} -conjugate directions in which to go. Two non-zero vectors are conjugate with respect to a matrix \mathbf{A} if $\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0$ with $\mathbf{p}_i \neq \mathbf{p}_j$. Instead of picking a set of vectors in the beginning, we pick a new one each iteration.

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k+1)} \mathbf{p}^{(k)}, \quad \mathbf{p}^{(0)} = \mathbf{r}^{(0)} \quad (63)$$

Obtaining β works as follows:

$$(\mathbf{p}^{(k+1)})^T \mathbf{A} \mathbf{p}^{(k)} = 0 \quad (64)$$

$$(\mathbf{r}^{(k+1)} + \beta^{(k+1)} \mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)} = 0 \quad (65)$$

$$(\mathbf{r}^{(k+1)})^T \mathbf{A} \mathbf{p}^{(k)} + \beta^{(k+1)} (\mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)} = 0 \quad (66)$$

$$\beta^{(k+1)} = -\frac{(\mathbf{r}^{(k+1)})^T \mathbf{A} \mathbf{p}^{(k)}}{(\mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)}} \quad (67)$$

From the update method

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)} \quad (68)$$

we get the equation for updating the residual

$$\mathbf{A} \mathbf{x}^{(k+1)} - \mathbf{b} = \mathbf{A} \mathbf{x}^{(k)} - \mathbf{b} + \alpha^{(k)} \mathbf{A} \mathbf{p}^{(k)} \quad (69)$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)} \mathbf{A} \mathbf{p}^{(k)} \quad (70)$$

$$\mathbf{A} \mathbf{p}^{(k)} = \frac{\mathbf{r}^{(k)} - \mathbf{r}^{(k+1)}}{\alpha^{(k)}} \quad (71)$$

Equation (71) gives us another way of calculating β

$$\beta^{(k+1)} = -\frac{(\mathbf{r}^{(k+1)})^T \mathbf{A} \mathbf{p}^{(k)}}{(\mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)}} = \frac{(\mathbf{r}^{(k+1)})^T \mathbf{r}^{(k+1)}}{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}} \quad (72)$$

The step length α is determined by

$$(\mathbf{r}^{(k+1)})^T \mathbf{r}^{(k)} = 0 \quad (73)$$

$$(\mathbf{r}^{(k)} - \alpha^{(k)} \mathbf{A} \mathbf{p}^{(k)})^T \mathbf{r}^{(k)} = 0 \quad (74)$$

$$(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)} - \alpha^{(k)} (\mathbf{A} \mathbf{p}^{(k)})^T \mathbf{r}^{(k)} = 0 \quad (75)$$

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)}} \quad (76)$$

Summary The iteration variables are calculated as follows:

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{p}^{(k)})^T \mathbf{A} \mathbf{p}^{(k)}} \quad (77)$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)} \quad (78)$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)} \mathbf{A} \mathbf{p}^{(k)} \quad (79)$$

$$\beta^{(k+1)} = \frac{(\mathbf{r}^{(k+1)})^T \mathbf{r}^{(k+1)}}{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}} \quad (80)$$

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k+1)} \mathbf{p}^{(k)} \quad (81)$$

We are guaranteed to find a solution after n steps for an $n \times n$ matrix, but can get reasonable approximations before that. In order to increase performance, one can use a precondition matrix $M^{-1}Ax = M^{-1}b$. The simplest choice is using the diagonal entries of A (Jacobi preconditioner).

4.3 FEM summary

The required steps for FEM are

1. Discretization of the domain
2. Construction of approximate solution over elements
3. Assembling elements into a global system
4. Imposing boundary conditions
5. Numerical solution of resulting equation system
6. Post-processing and visualization of results

FEM is widely applicable, easy to use for complex geometries, easy to adapt for better approximation and it is easy to impose boundary conditions. However, the solution is only an approximate, careful setup is required, the amount of pre-processing required can become large, and parallelization is not straight forward (due to the interdependencies between elements).