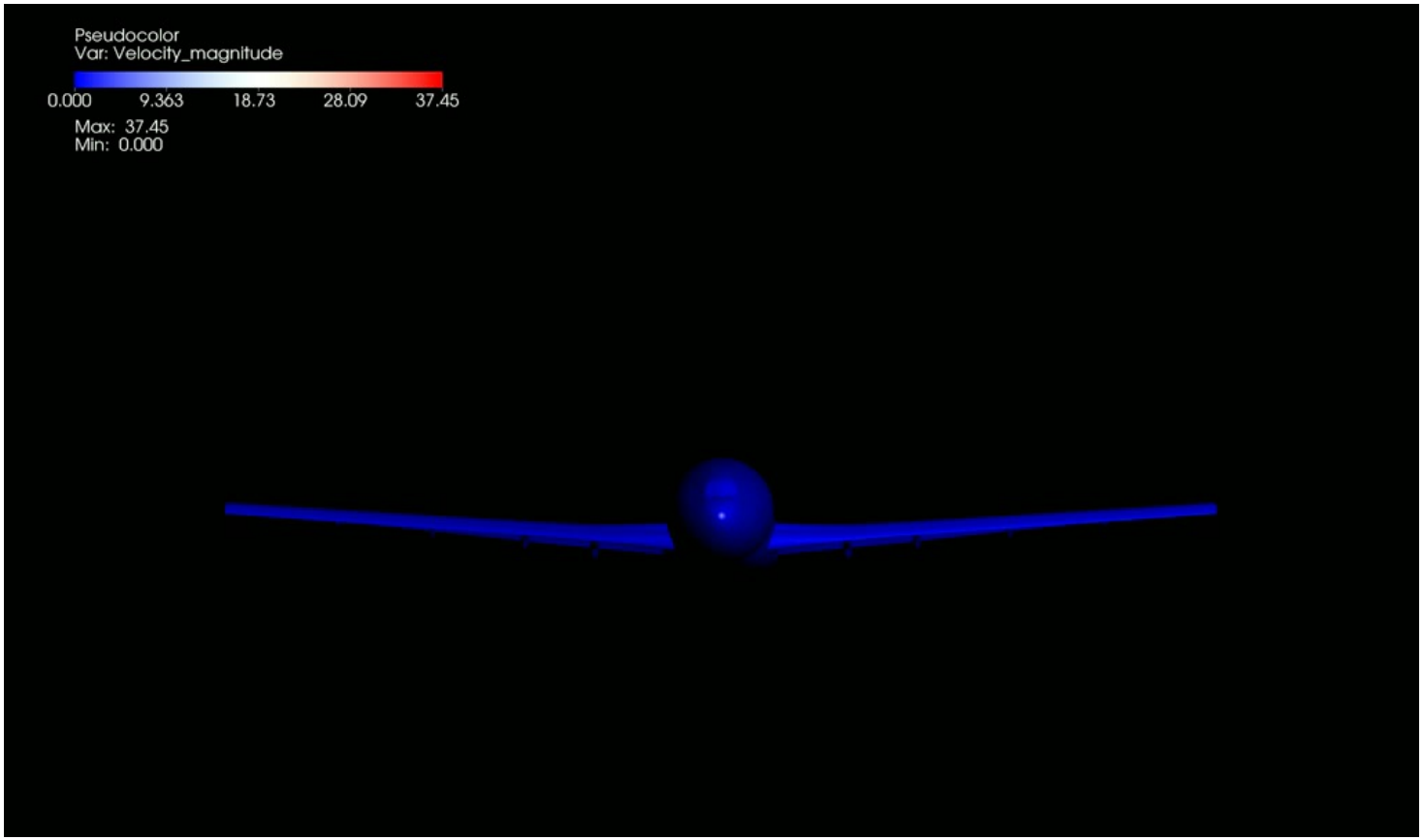
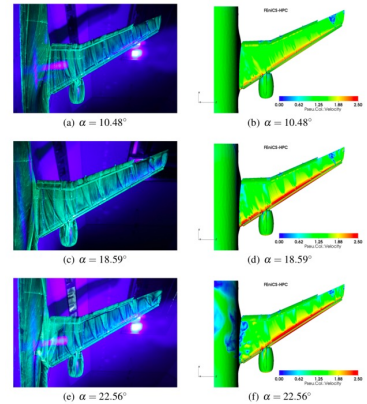
DD2363/2023 – Lecture 10
Systems of initial value problems (ch.14)

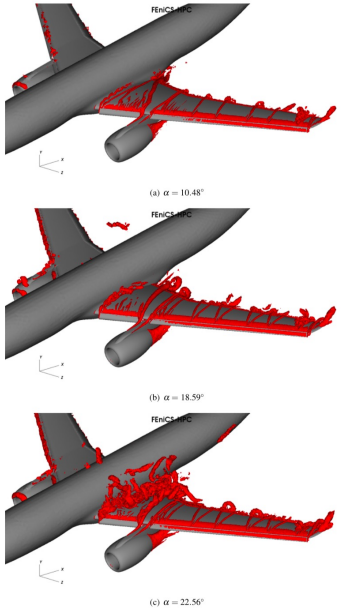
Johan Hoffman

Introduction:
scientific computing/computational science

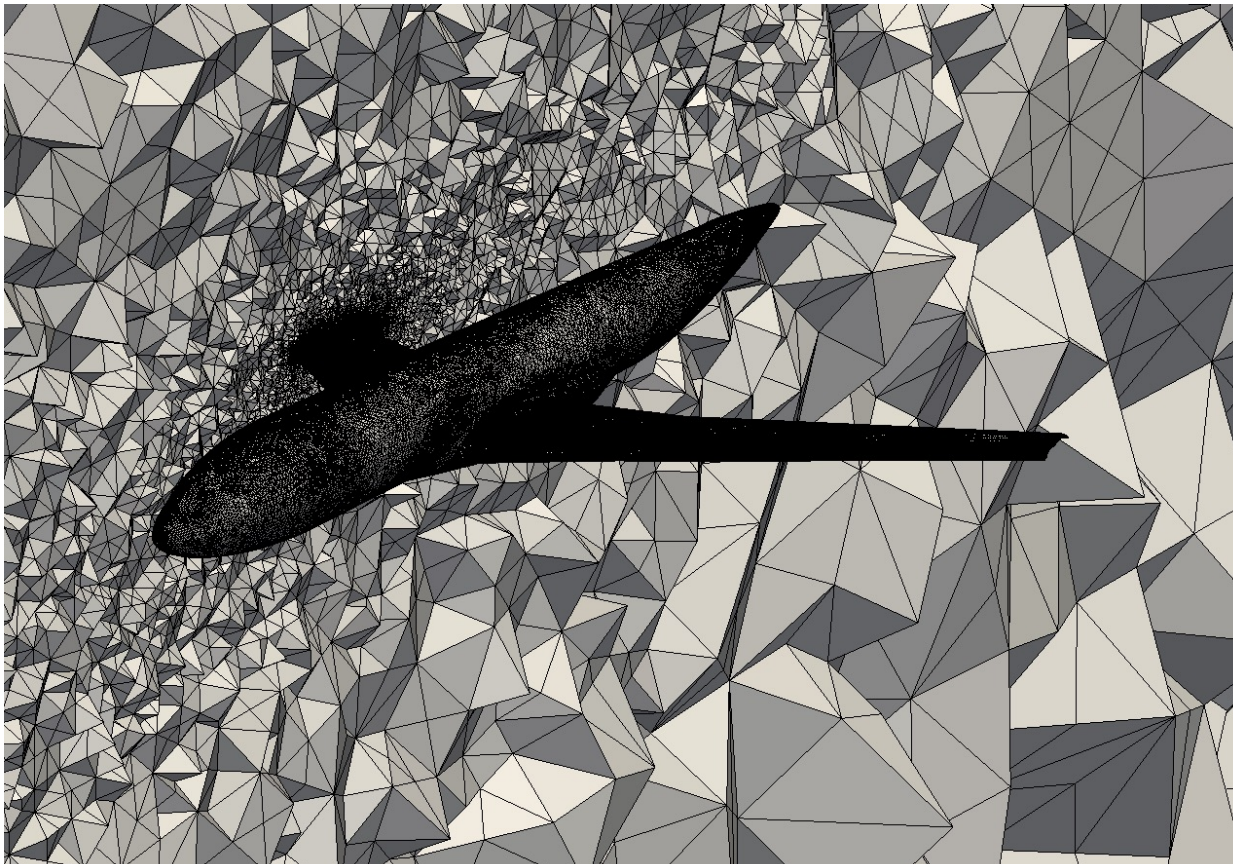
Simulation of airflow past airplane

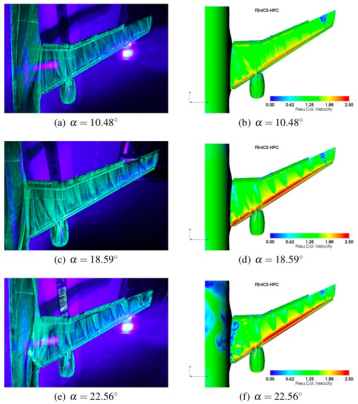


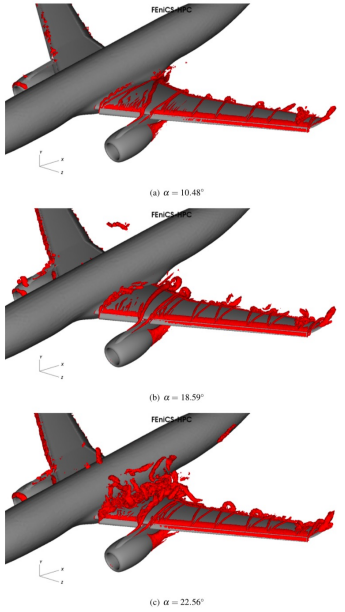




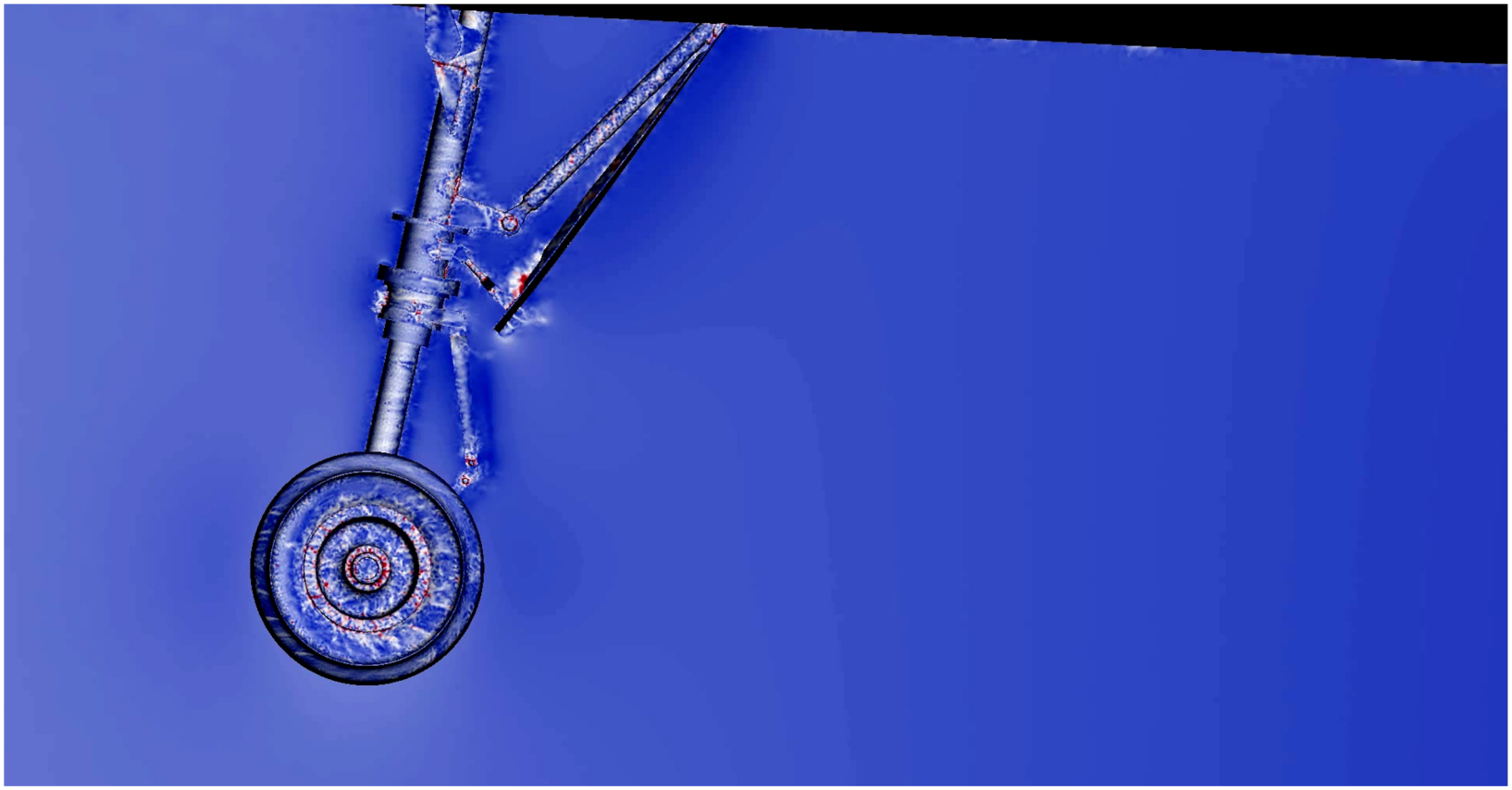
Discretization by a mesh



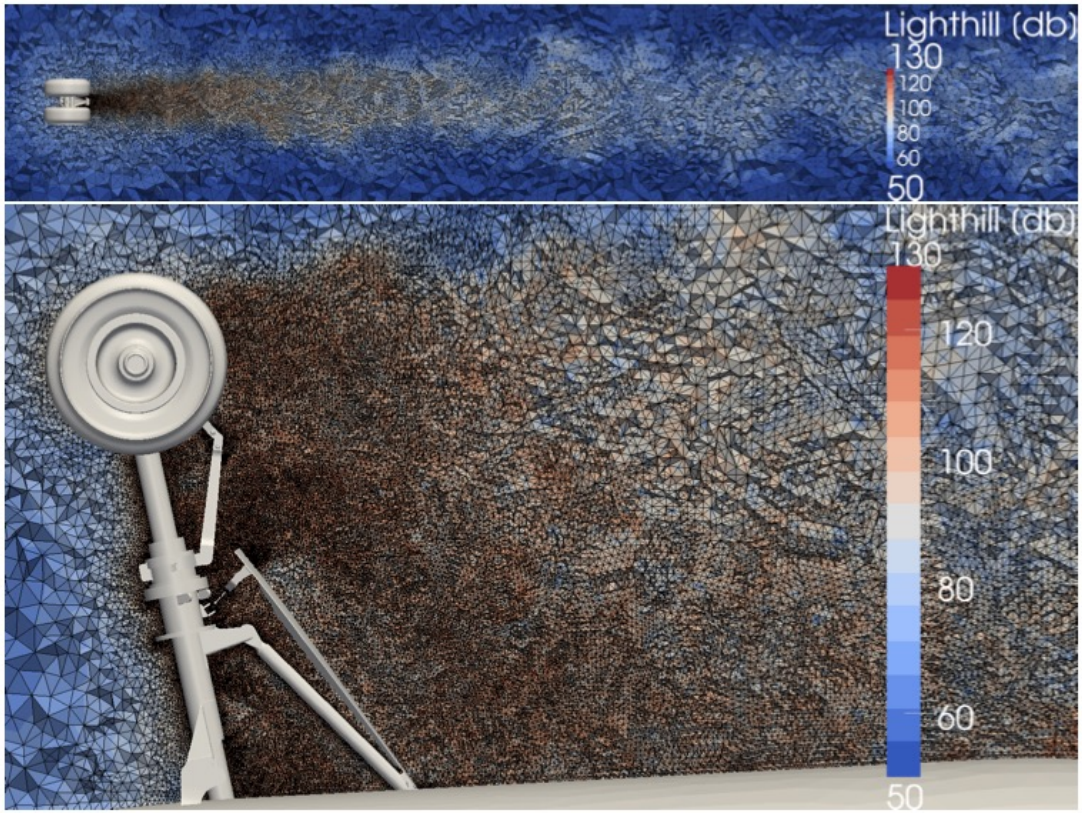


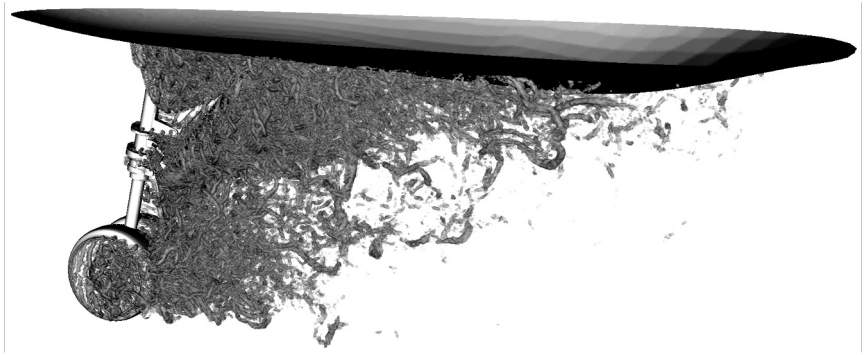


Simulation of airflow past landing gear

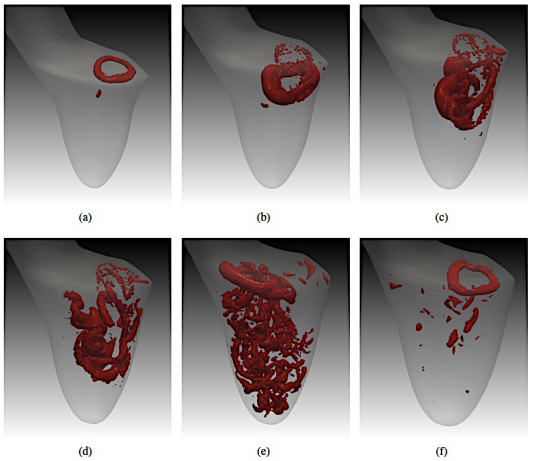


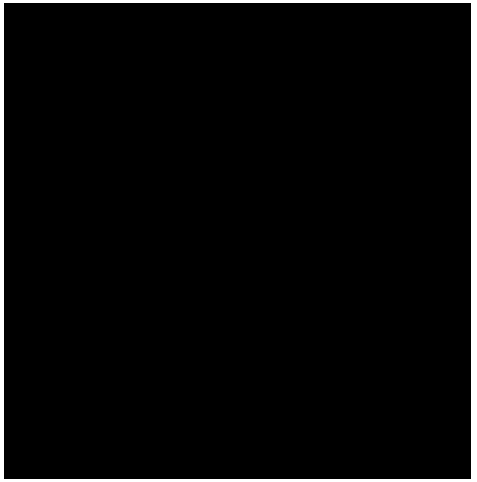
Acoustic sources and turbulent vortices

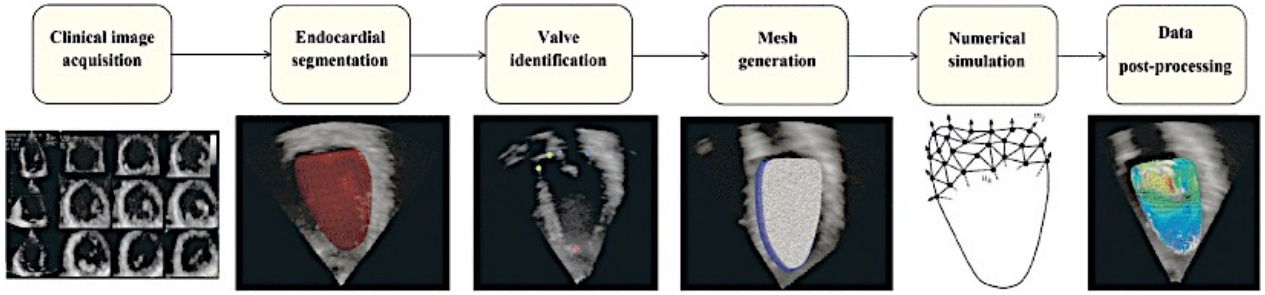




Heart simulation and data analysis







Time-stepping algorithm

ALGORITHM 3.6. (u, t) = time_step(T, u0, t0, dt).
Input: state transition function **T**, old state **u0**, old time **t0**, time step **dt**.
Output: new state **u**, new time **t**.

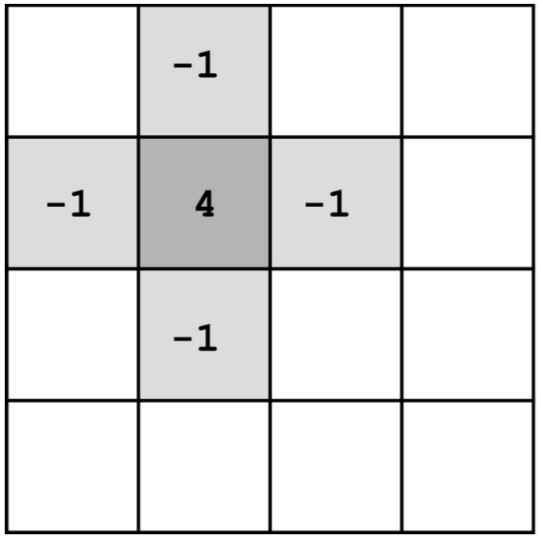
```
1: u = T(u0, t0, dt)
2: t = t0 + dt
3: return u, t
```

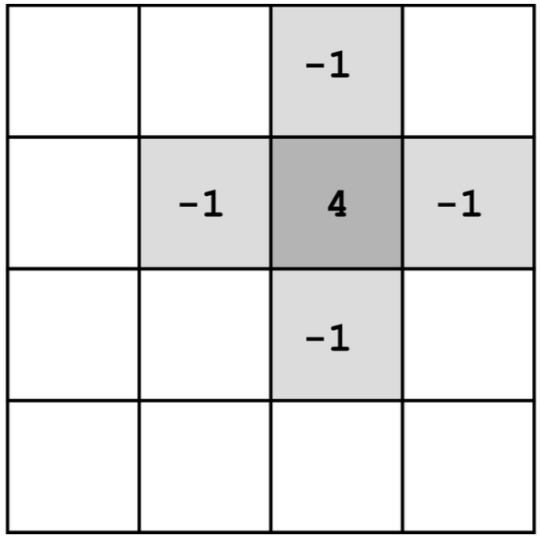
Convolution (filtering)

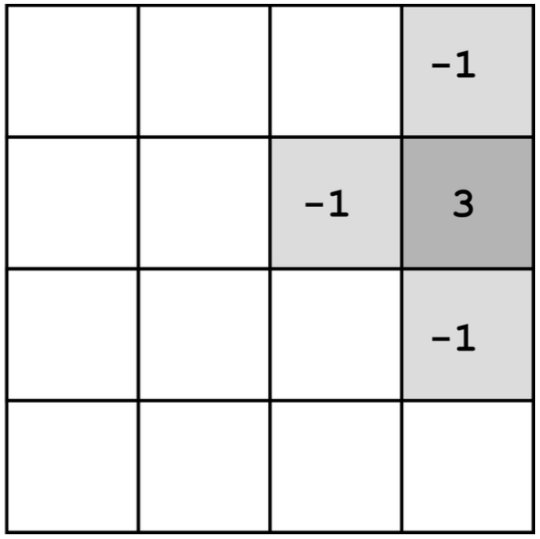
Convolution (filtering)

$$S = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad B = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution (filtering)





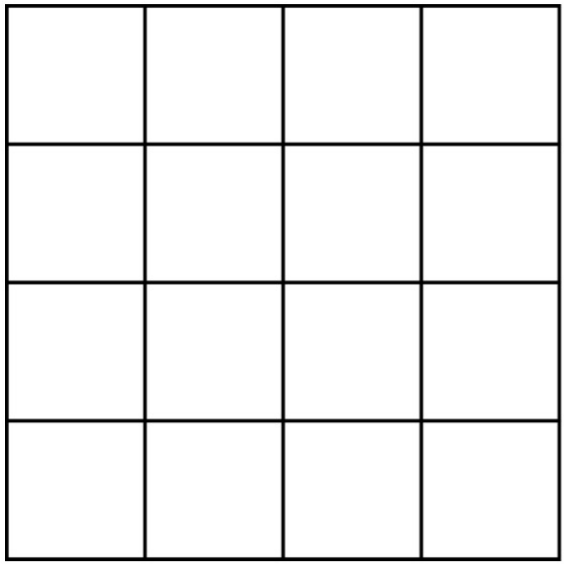


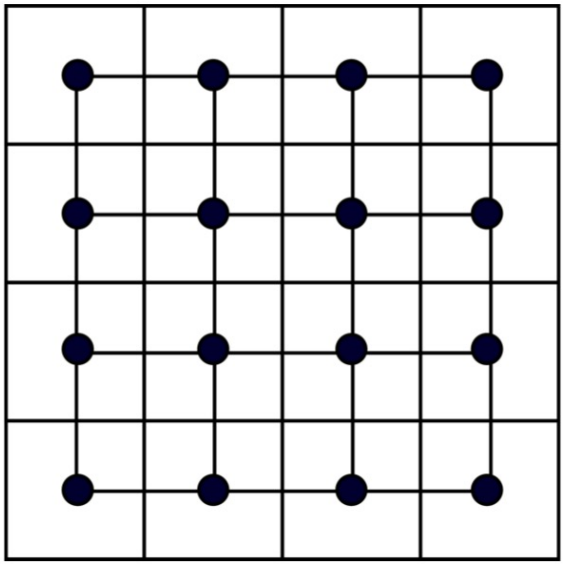
Convolution (filtering)

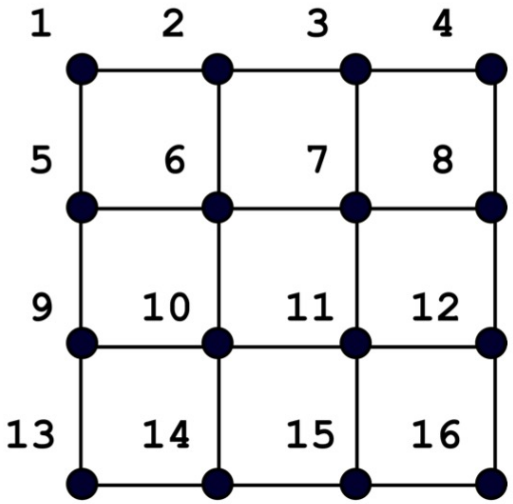
ALGORITHM 3.5. filtered_image = convolution(image, kernel).
Input: an array of pixel intensities **image**, and a **kernel**.
Output: an array of modified pixel intensities **filtered_image**.

```
1: for i=0:length(image)-1 do
2:     neighbors = kernel.get_neighbors(i)
3:     filtered_image[i] = 0
4:     for j=0:length(neighbor_pixels)-1 do
5:         filtered_image[i] = filtered_image[i] + kernel.weights(j) * image[neighbors[j]]
6:     end for
7: end for
8: return filtered_image
```

Vectorization







Convolution kernel -> sparse matrix

$$
\begin{bmatrix}
2 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 2
\end{bmatrix}
$$

Matrix-vector product

ALGORITHM 3.2. b = matrix_vector_product(A, x).
Input: an nxn matrix **A** and an n vector **x**.
Output: the matrix-vector product n vector **b** = Ax.

```
1: b = 0
2: for i=0:n-1 do
3:     for j=0:n-1 do
4:         b[i] = b[i]+A[i,j]*x[j]
5:     end for
6: end for
7: return  b
```

High performance computing

High performance computing

SUPERCOMPUTER FUGAKU - SUPERCOMPUTER FUGAKU, A64FX 48C 2.2GHZ, TOFU INTERCONNECT D

Site:	RIKEN Center for Computational Science
System URL:	https://www.r-ccs.riken.jp/en/fugaku/project
Manufacturer:	Fujitsu
Cores:	7,299,072
Memory:	4,866,048 GB
Processor:	A64FX 48C 2.2GHz
Interconnect:	Tofu interconnect D
Performance	
Linpack Performance (Rmax)	415,530 TFlop/s
Theoretical Peak (Rpeak)	513,855 TFlop/s
Nmax	20,459,520
HPCG [TFlop/s]	13,366.4

Methods in Scientific Computing

- Computing in science, and the science of computing
- Theory- and data-driven computational modeling and simulation
- Modern hardware and software platforms

Three key algorithms

- Time stepping
- Convolution
- Matrix-vector product

Chapter 14

Intial value problem

Consider the following *ordinary differential equation* (ODE) for a scalar function $u : [0, T] \to R$, with derivative $\dot{u} = du/dt$,

$$\dot{u}(t) = f(u(t), t), \quad 0 < t \leq T,$$
$$u(0) = u_0,$$

which we refer to as a *scalar initial value problem* (IVP), defined on the interval $I = [0, T]$ by the function $f : R \times R^+ \to R$, and the *initial condition* $u(0) = u_0$. Only in special cases can exact closed form solutions be found, instead approximation methods must be used in general.

Forward Euler method

ALGORITHM 13.1. f = explicit_euler_method(f, u0, t0, T, k).
Input: function **f**, initial data **u0**, initial time **t0**, final time **T**, time step **k**.
Output: approximation at final time **u**.

```
1: t = t0
2: while t < T do
3:     u = u0 + k*f(u0,t)
4:     u0 = u
5:     t = t + k
6: end while
7: return u
```

Backward Euler method

ALGORITHM 13.2. f = implicit_euler_method(f, u0, t0, T, k).
Input: function **f**, initial data **u0**, initial time **t0**, final time **T**, time step **k**.
Output: approximation at final time **u**.

```
1: t = t0
2: while t < T do
3:     u = newtons_method(u - u0 - k*f(u,t), u0)
4:     u0 = u
5:     t = t + k
6: end while
7: return u
```

Time stepping methods as quadrature rules

$$u(t_n) = u(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(u(t), t)\, dt,$$

from which we can construct various time stepping methods by using different quadrature rules to evaluate the integral in equation (13.3) at each time step I_n.

Euler methods as quadrature rules

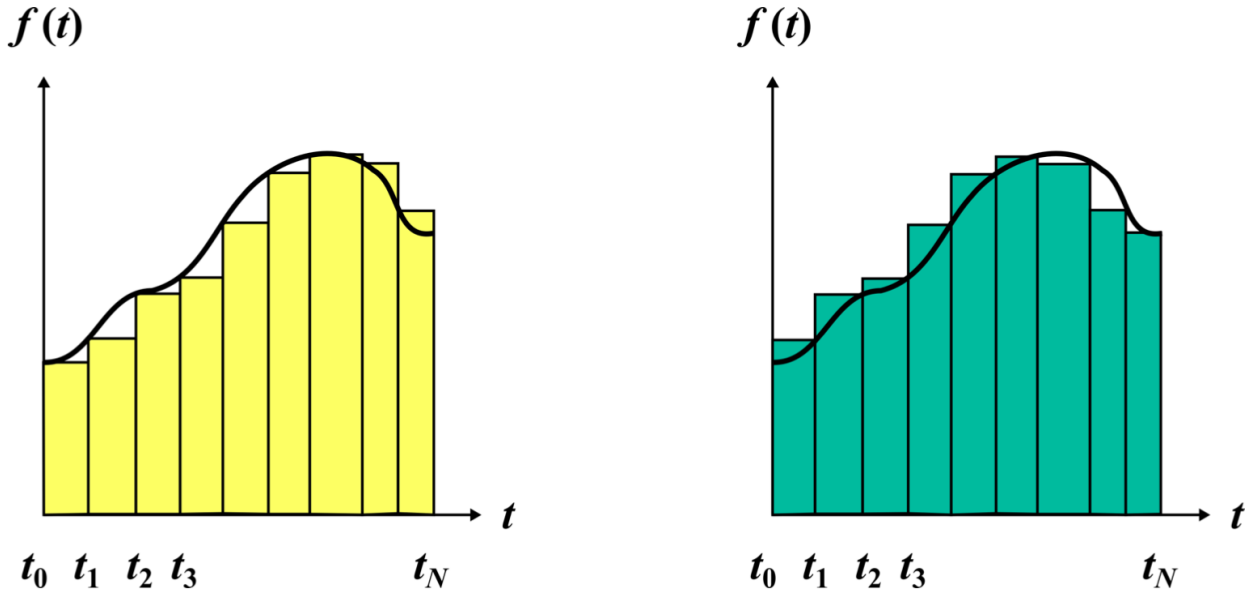


Figure 13.1. *Left (left) and right (right) Riemann sums which approximate the integral of* $f(t)$*, corresponding to the explicit and implicit Euler time stepping method for approximation of the IVP (13.1) with* $f(u(t), t) = f(t)$.

Midpoint and trapezoidal methods

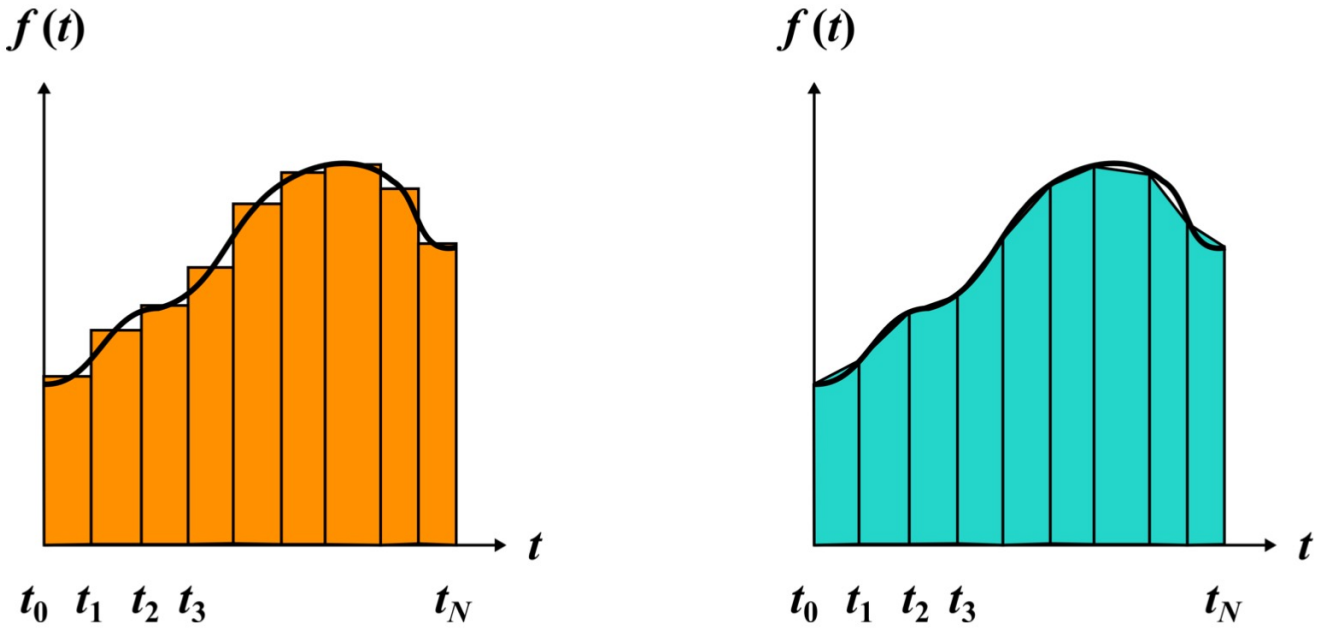


Figure 13.2. *The midpoint (left) and the trapezoidal (right) quadrature rules, corresponding to interpolation by a piecewise constant and piecewise linear function respectively.*

Theta method

ALGORITHM 13.3. f = theta_method(f, u0, t0, T, k, theta).
Input: function **f**, initial data **u0**, **theta**, final time **T**, time step **k**.
Output: approximation at final time **u**.

```
1: t = t0
2: while t < T do
3:     u = newtons_method(u - u0 - k*((1-theta)*f(u) + theta*f(u0)), u0)
4:     u0 = u
5:     t = t + k
6: end while
7: return u
```

Stability of initial value problems

The concept of *stability* of an initial value problem refers to how small perturbations propagate in time, and in the context of time stepping methods, how local approximation errors accumulate from one time step to the next. Specifically, for the initial value problem (13.1) we are interested in the stability of *equilibrium points* u^*, defined by the condition that

$$f(u^*, t) = 0, \quad \forall t \geq 0.$$

Stability of linear model problem

First assume that the model problem is homogeneous, that is, $g(t) = 0$. Then by equation (13.5) there is one equilibrium point $u^* = 0$, which we choose as the initial condition $u_0 = u^*$. We also introduce a problem with perturbed initial condition $u^* + \epsilon$ and solution $v(t)$,

$$\dot{u}(t) + \lambda u(t) = 0, \quad u(0) = u^*, \tag{13.7}$$

$$\dot{v}(t) + \lambda v(t) = 0, \quad v(0) = u^* + \epsilon, \tag{13.8}$$

with $\epsilon \neq 0$ a small perturbation. To study the propagation of the perturbation $\varphi(t) = v(t) - u(t)$, we subtract equation (13.7) from equation (13.8), to get the *perturbation equation*

$$\dot{\varphi}(t) + \lambda \varphi(t) = 0, \quad \varphi(0) = \epsilon,$$

with solution

$$\varphi(t) = \exp(-\lambda t)\epsilon.$$

The equilibrium point $u^* = 0$ is stable if $\lambda > 0$ and unstable if $\lambda < 0$, since then $\varphi(t)$ either decays to zero or grows exponentially. If $\lambda = 0$, then $u^* + \epsilon$ is a new equilibrium point.

Stability of linear model problem

For a complex number $z \in C$, we still have that

$$\frac{d}{dt}\exp(zt) = z\exp(zt),$$

and hence for $\lambda = a + ib$ a complex number, the solution to the perturbation equation is

$$\varphi(t) = \exp(-\lambda t)\epsilon = \exp(-at)\exp(-ibt)\epsilon = \exp(-at)(\cos(bt) - i\sin(bt))\epsilon.$$

We here again distinguish between three cases. If the real part $a < 0$ the equilibrium point u^* is unstable, because perturbations grow at exponential rate, whereas if $a > 0$ then u^* is stable since perturbations decay exponentially. For the pure imaginary case $a = 0$, perturbations oscillate around u^* with the same amplitude over time, in the sense that $|\exp(-ibt)| = 1$.

Linear stability analysis of general IVP

For the general nonlinear initial value problem (13.1), the stability of an equilibrium point u^* can be analyzed in a *linearized* perturbation equation, derived by truncating a Taylor expansion of the right hand side of equation (13.1). Assuming that the function $f(u(t), t)$ is continuously differentiable, and that $|v(t) - u(t)|$ is small, we drop the higher order terms which leads to the approximation

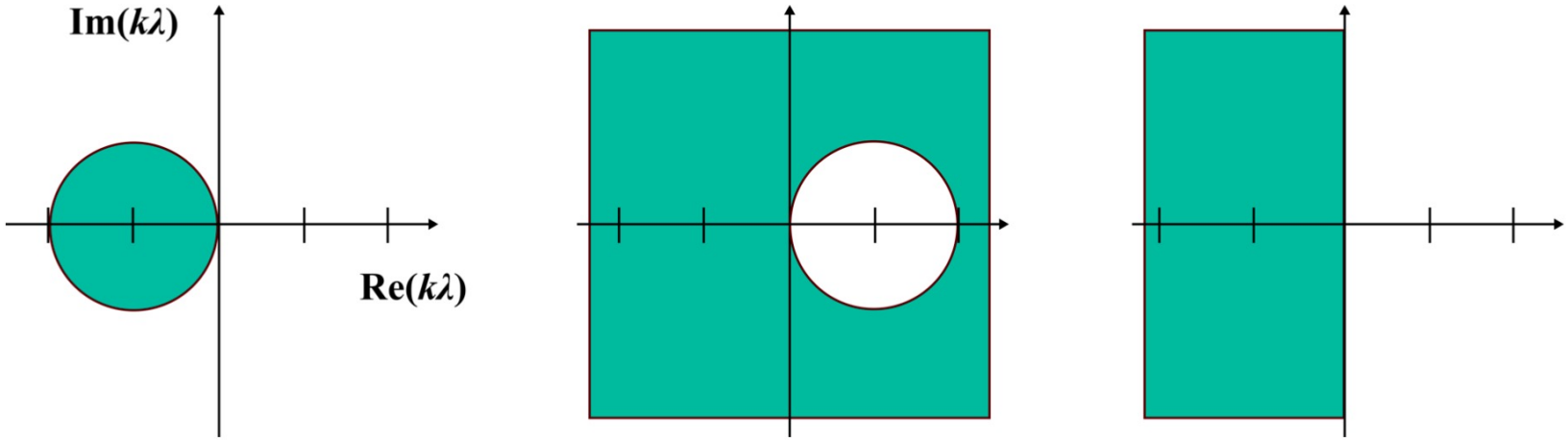
$$f(v(t), t) - f(u(t), t) \approx f'(u(t), t)(v(t) - u(t)),$$

where the derivative $f'(u(t), t)$ is taken with respect to the first argument of the function. By *linearization* at the equilibrium point $u(t) = u^*$, we get the *linearized perturbation equation*

$$\dot{\varphi}(t) - f'(u^*, t)\varphi(t) = 0, \quad \varphi(0) = \epsilon.$$

Stability regions for time stepping methods

In Figure 13.4 we visualize these conditions as *stability regions* in the complex plane. Since $\mathrm{Re}(\lambda) < 0$ and $k > 0$, the explicit Euler method is *conditionally stable*, provided $|1 + k\lambda| < 1$, whereas the implicit Euler method and the trapezoidal method are *unconditionally stable*.



A posteriori error estimation – general IVP

This type of *a posteriori* error analysis can be extended to the general initial value problem (13.1), with the linearized adjoint problem

$$-\dot{\varphi}(t) - \bar{f}'(u, U, t)\varphi(t) = \psi(t), \quad \varphi(T) = \varphi_T, \tag{13.18}$$

and the averaged derivative defined by equation (13.10).

$$\bar{f}'(v, u, t) = \int_0^1 f'(w(u, v; s), t)\, ds,$$

where $w(u, v; s) = sv(t) + (1 - s)u(t)$ is a convex combination of $v(t)$ and $u(t)$, and

$$f(v(t), t) - f(u(t), t) = \int_0^1 \frac{d}{ds} f(w(u, v; s), t)\, ds = \bar{f}'(v, u, t)\varphi(t).$$

A posteriori error estimation – general IVP

To derive the a posteriori error estimate (13.17) for the general initial value problem, we multiply the adjoint equation (13.18) by the error and integrate over the time interval $I = [0, T]$,

$$\begin{aligned}
\int_0^T e(t)\psi(t)\,dt &= \int_0^T e(t)(-\dot{\varphi}(t) - \bar{f}'(u, U, t)\varphi(t))\,dt \\
&= e(0)\varphi(0) - e(T)\varphi(T) + \int_0^T (\dot{e}(t) - \bar{f}'(u, U, t)e(t))\varphi(t)\,dt \\
&= e(0)\varphi(0) - e(T)\varphi(T) + \int_0^T (\dot{u} - f(u, t)) - (\dot{U}(t) - f(U, t))\varphi(t)\,dt \\
&= e(0)\varphi(0) - e(T)\varphi(T) + \int_0^T R(U(t))\varphi(t)\,dt,
\end{aligned}$$

by equation (13.11), from which we obtain the error representation (13.17), with the residual

$$R(U(t)) = f(U, t) - \dot{U}(t).$$

Adaptive time stepping methods

The *a posteriori* error estimate takes the form of an integral over the domain $I = [0, T]$, which can be split into a sum of integrals over the N subinterval $I_i = [t_{i-1}, t_i]$,

$$\int_0^T e(t)\psi(t)\,dt + e(T)\varphi_T = e(0)\varphi(0) + \sum_{i=1}^{N} \int_{I_i} R(U(t))\varphi(t)\,dt = e(0)\varphi(0) + \sum_{i=1}^{N} \mathcal{E}_i,$$

where $\mathcal{E}_i$ is the *error indicator* for subinterval I_i. The error indicators can be used to formulate *adaptive* time stepping algorithms, where, for example, in each iteration of the adaptive algorithm we bisect all subintervals I_i for which

$$\mathcal{E}_i > \frac{1}{N} \sum_{j=1}^{N} \mathcal{E}_j.$$

Systems of initial value problems

Consider now a system of initial value problems, where we seek a vector function

$$u : [0, T] \to R^N,$$

with derivative

$$\dot{u} = \frac{du}{dt} = \left(\frac{du_1}{dt}, ..., \frac{du_N}{dt} \right)^T,$$

such that for a function $f : R^N \times [0, T] \to R^N$,

$$\begin{aligned} &\dot{u}(t) = f(u(t), t), \quad 0 < t \leq T, \\ &u(0) = u_0. \end{aligned} \tag{14.1}$$

Stability of IVP systems

$$f(u^*, t) = 0, \quad \forall t \in [0, T].$$

The stability of an equilibrium point u^* can be analyzed by adding a small perturbation vector $\epsilon \in R^N$ to u^*, and then study how this perturbation evolves in time. More specifically, we introduce the perturbed initial value problem

$$\begin{aligned} \dot{v}(t) &= f(v(t), t), \quad 0 < t \leq T, \\ v(0) &= u^* + \epsilon, \end{aligned}$$

from which we subtract equation (14.1) with $u_0 = u^*$, to get an equation for the perturbation growth $\varphi(t) = v(t) - u(t)$,

$$\begin{aligned} \dot{\varphi}(t) &= f(v(t), t) - f(u(t), t), \quad 0 < t \leq T, \\ \varphi(0) &= \epsilon. \end{aligned}$$

Stability of linear IVP systems

If $f(u(t),t) = -Au(t)$, with $A \in R^{N\times N}$ a nonsingular matrix, then the only equilibrium point is $x^* = 0$, and the perturbation equation takes the form

$$\begin{aligned} &\dot{\varphi}(t) + A\varphi(t) = 0, \quad 0 < t \le T, \\ &\varphi(0) = \epsilon. \end{aligned} \tag{14.2}$$

The solution to (14.2) is $\varphi(t) = \exp(-At)\epsilon$, with $\exp(-At)$ the matrix exponential (7.21). If A is diagonalizable, we can express the matrix exponential as

$$\exp(-At) = Xe^{-\Lambda t}X^{-1} = X \begin{bmatrix} \exp(-\lambda_1 t) & & 0 \\ & \ddots & \\ 0 & & \exp(-\lambda_n t) \end{bmatrix} X^{-1},$$

Stability of nonlinear IVP systems

For a nonlinear, continuously differentiable function f, we can analyze the stability in a linearized problem with $A = f'(u^*, t)$, the Jacobian matrix differentiated with respect to the first argument and evaluated at the equilibrium point u^*, which follows from the approximation

$$f(v(t), t) - f(u^*, t) \approx f'(u^*, t)\varphi,$$

where we drop the higher order terms in the perturbation which are assumed to be small. For example, if the Jacobian $f'(u^*, t)$ is a symmetric positive definite matrix, then all eigenvalues are positive real numbers, which means that u^* is a stable equilibrium point.

Stability of nonlinear IVP systems

But linear stability analysis is not reliable if the Jacobian $f'(u^*, t)$ is a defective matrix, since then even if all eigenvalues have positive real parts, perturbations in the linearized system may grow for some time before the asymptotic exponential decay dominates. If this transient perturbation growth is strong enough, the asymptotic decay of the linearized system does not describe the true nonlinear system since the solution is too far from the equilibrium point.

Stability of nonlinear IVP systems

Example 14.1. The following linearized system describes the growth of perturbations of an equilibrium point in a certain nonlinear system,

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \end{bmatrix} = \begin{bmatrix} -\nu & \gamma \\ 0 & -\nu \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \end{bmatrix}, \tag{14.3}$$

where the initial perturbations are $\varphi_1(0) = 0$ and $\varphi_2(0) = \kappa$. The system matrix is defective, it has one stable eigenvalue $\lambda = -\nu$ of algebraic multiplicity two, but of geometric multiplicity only one. Since the eigenvalue is stable, we would expect the perturbations to decay to zero at exponential rate, but the solution to the system is

$$\begin{aligned} \varphi_1(t) &= \gamma\kappa t \exp(-\nu t), \\ \varphi_2(t) &= \kappa \exp(-\nu t). \end{aligned} \tag{14.4}$$

Therefore, before the perturbation decays to zero, there will be a linear growth of the perturbation φ_1, and if this linear growth is sufficiently strong the solution $u(t) = u^* + \varphi(t)$ can reach a new equilibrium point that may be unstable.

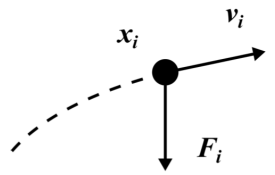
Particle models

Newton's 2nd law of motion for a particle states that mass times acceleration equals force,

$$m\ddot{x}(t) = F(t), \tag{14.5}$$

a second order initial value problem, which can be formulated as a system of first order initial value problems (14.1) for the position $x : [0, T] \to R^3$, the velocity $v(t) = \dot{x}(t)$, the acceleration $\dot{v}(t) = \ddot{x}(t)$, and the mass m of the particle,

$$u = \begin{bmatrix} v \\ x \end{bmatrix}, \quad f = \begin{bmatrix} F/m \\ v \end{bmatrix},$$



where $F : [0, T] \to R^3$ is the applied force. *Newton's first law* of motion states that the velocity of a particle is constant in the absence of a force, and is consistent with (14.5) for $F = 0$.

Example 14.2 (Gravitation). Gravitation is modelled by the force $F = (0, 0, -mg) = -mge_3$, with g the gravitation constant and e_3 a Cartesian basis vector in the vertical direction.

Harmonic oscillator

Example 14.3 (Hooke's law). By Hooke's law, the force in an elastic spring with stiffness (spring constant) k is modelled by $F = (-kx_1, 0, 0) = -kx_1 e_1$, with x_1 the extension of the spring in the e_1 direction.

Reduced to the e_1 direction, the motion of a particle which is attached to a Hookean spring is an example of a *harmonic oscillator*, where $x(t) = x_1(t)$ is the distance of the particle from its equilibrium position and $v(t) = v_1(t)$ its velocity in the e_1 direction,

$$\dot{x}(t) = v(t), \quad \dot{v}(t) = -\frac{k}{m}x(t). \tag{14.6}$$

The solution to the system is

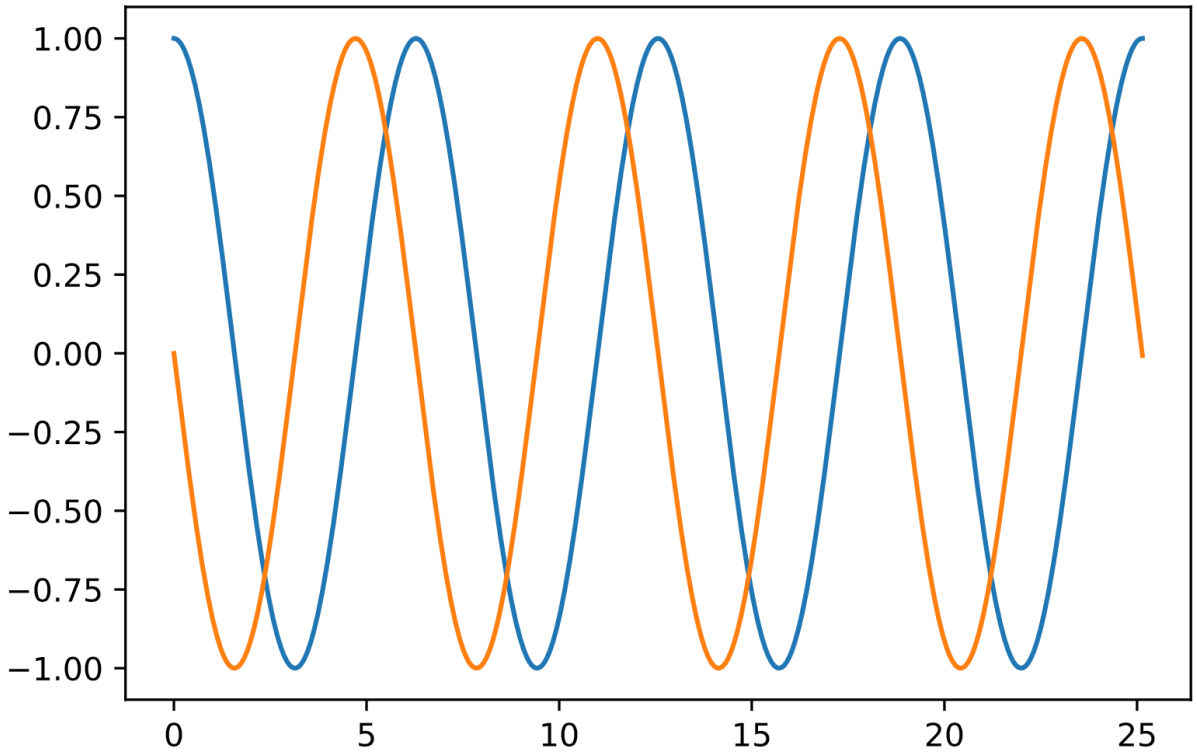
$$x(t) = A\sin(\omega t + \phi),$$
$$v(t) = A\omega\cos(\omega t + \phi),$$

where A is the *amplitude* and ϕ the *phase shift*, both determined from the initial conditions

$$x(0) = x_0, \quad v(0) = v_0.$$

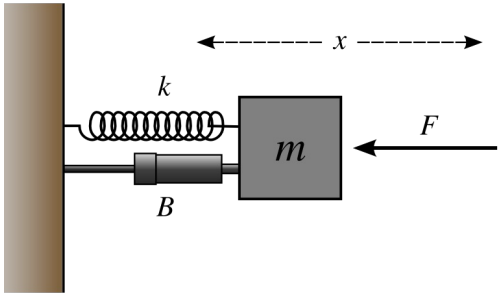
[Wikipedia]

Harmonic oscillator



[Wikipedia]

Harmonic oscillator



The *angular frequency* ω is a function of the mass and stiffness,

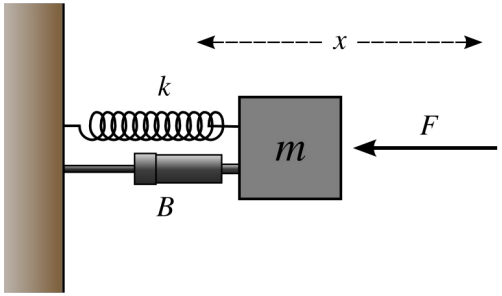
$$\omega = \sqrt{\frac{k}{m}},$$

and the *frequency* is related to the angular frequency as $f = \omega/2\pi$, connecting the oscillating motion with amplitude A to an angular motion along a circle with radius A. The *period* of both motions is $T = 1/f = 2\pi/\omega$. If we add also a friction force which is proportional to the velocity with a *viscous damping constant* c, the total force is $F(t) = -kx(t) - cv(t)$. With this new force, we can write equation (14.6) on matrix form as

$$\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix}. \tag{14.7}$$

[Wikipedia]

Harmonic oscillator



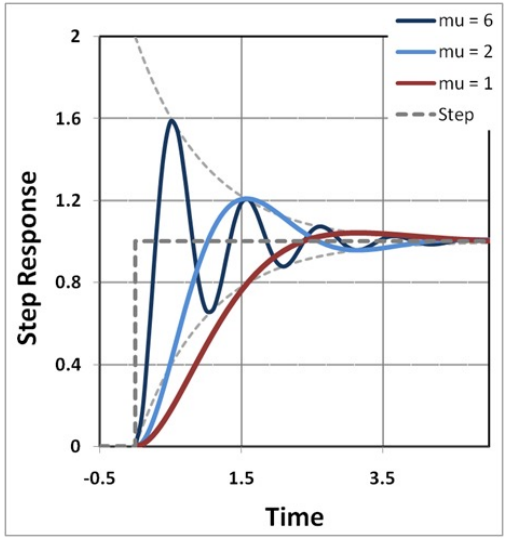
If $c = 0$, the matrix has two purely imaginary eigenvalues $\lambda = \pm\omega i$, and with $c \neq 0$ the eigenvalues are the roots of the characteristic polynomial $m\lambda^2 + c\lambda + k = 0$, with solutions

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4mk}}{2m} \tag{14.8}$$

from which we find that $\mathrm{Re}(\lambda) < 0$, and hence the viscous damping provides stability to the system. We distinguish between three cases based on equation (14.8),

$$c^2 - 4mk < 0, \text{ an underdamped system,}$$
$$c^2 - 4mk = 0, \text{ a critically damped system,}$$
$$c^2 - 4mk > 0, \text{ an overdamped system.}$$

For an underdamped system, corresponding to complex eigenvalues, the oscillations are damped by a factor $\exp(-ct/2m)$, whereas for a critical or an overdamped system where the eigenvalues all are real, no oscillations develop at all.



[Wikipedia]

Harmonic oscillator

Now multiply the first equation in (14.7) by $kx(t)$ and the second equation by $mv(t)$, followed by summation of the two equations, which leads to the *conservation law*

$$\frac{d}{dt}(K(t) + P(t)) = -D(t).$$

The conservation law expresses the evolution of the *total energy* as the sum of *kinetic energy* $K(t)$ and *potential energy* $P(t)$, with dissipation of energy from viscous damping $D(t)$, where

$$K(t) = \frac{m}{2}|v(t)|^2, \quad P(t) = \frac{k}{2}|x(t)|^2, \quad D(t) = \frac{c}{2}|v(t)|^2.$$

When we construct approximations, we want to respect conservation laws like this one. For $c = 0$, the θ-method for the harmonic oscillator, with time step length Δt, takes the form

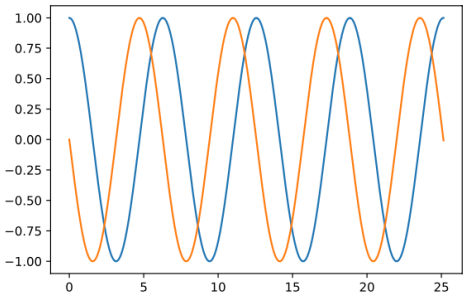
$$\begin{aligned} V_n &= V_{n-1} - \frac{k\Delta t}{m}((1-\theta)X_n + \theta X_{n-1}), \\ X_n &= X_{n-1} + \Delta t((1-\theta)V_n + \theta V_{n-1}). \end{aligned}$$

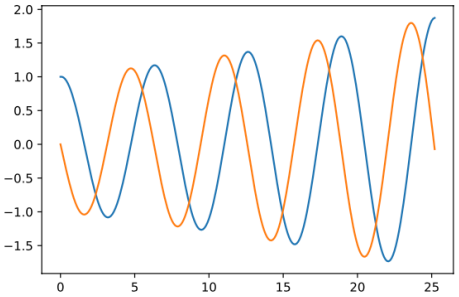
Harmonic oscillator

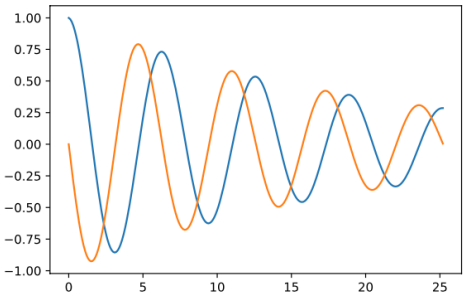
Neither the explicit Euler method ($\theta = 1$) or the implicit Euler method ($\theta = 0$) conserves the total energy over time, but with the trapezoidal rule ($\theta = 0.5$) the total energy is conserved for the approximate solution (V_n, X_n) at each time step,

$$\frac{m}{2}V_n^2 + \frac{k}{2}X_n^2 = \frac{m}{2}V_{n-1}^2 + \frac{k}{2}X_{n-1}^2, \tag{14.9}$$

which follows from summation of the two equations, after multiplication of the first equation by $(V_n + V_{n-1})/2$ and the second equation by $(X_n + X_{n-1})k/2m$.







N-body problem

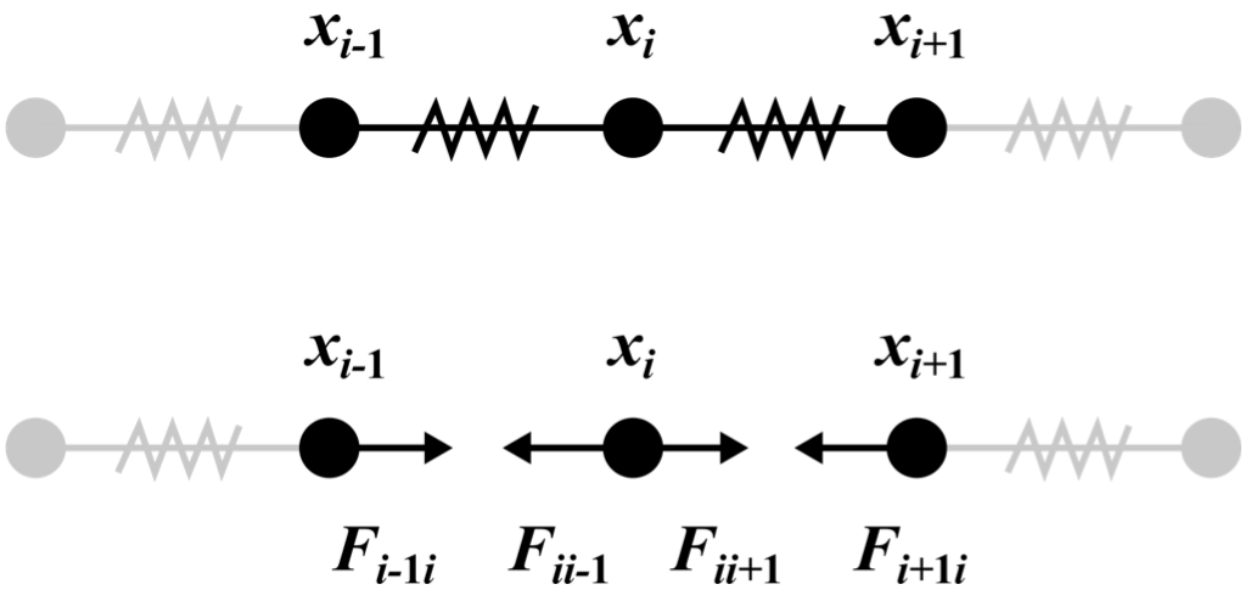
$$u = \begin{bmatrix} x_1 \\ \vdots \\ x_N \\ v_1 \\ \vdots \\ v_N \end{bmatrix}, \quad f = \begin{bmatrix} v_1 \\ \vdots \\ v_N \\ F_1/m_1 \\ \vdots \\ F_N/m_N \end{bmatrix},$$

where $x_i, v_i, F_i \in R^d$, and the total force on each particle p_i is the sum of pairwise interactions,

$$F_i = \sum_{j=1, j \neq i}^{N} F_{ij}.$$

To compute all pairwise force interactions in an N-body problem is an $\mathcal{O}(N^2)$ algorithm. But divide and conquer algorithms of complexity $\mathcal{O}(N \log N)$ and even $\mathcal{O}(N)$ can be constructed based on the idea to cluster forces from multiple particles at a distance.

N-body problem



Wave equation

Example 14.4 (Wave equation). Imagine now that we have N particles, each with mass m, connected by Hookean springs in an array of total mass $M = Nm$ and total length L, each at a distance of $h = L/N$ from each other, which represents an equilibrium state for the springs. The particles are connected to their two neighbors according to their numbering, so that particle p_i is connected to p_{i-1} from the left and p_{i+1} from the right, and we let x_i denote the distance of particle p_i from its equilibrium position. The Hookean force acting on particle p_i is then

$$F_i = k(x_{i+1} - x_i) - k(x_i - x_{i-1}) = k(x_{i+1} - 2x_i + x_{i-1}).$$

Because the springs are connected in series, the total spring constant of the particle system is $K = k/N$, so that $k/m = KN^2/M = KL^2/(Mh^2)$, and

$$F_i/m = \frac{KL^2}{M}\frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} \to \frac{KL^2}{M}x'', \text{ as } N \to \infty,$$

which by equation (14.5) leads to the *wave equation*

$$\ddot{x} - c^2 x'' = 0.$$

Wave equation

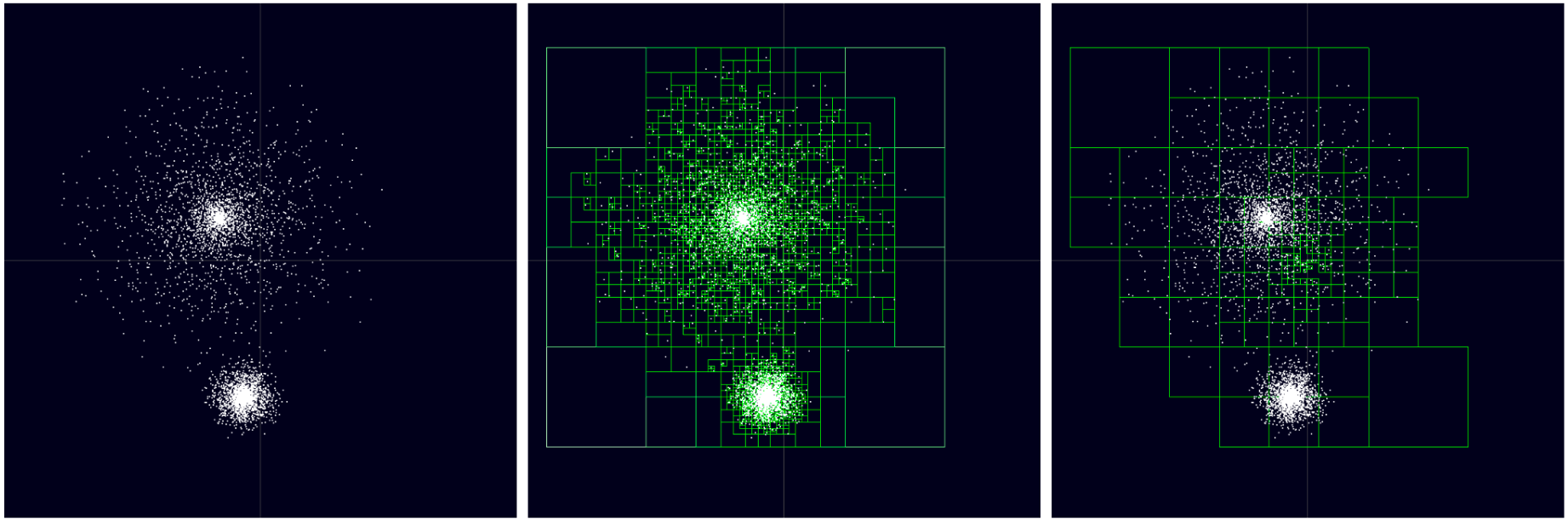
Barnes-Hut simulation

Example 14.5 (Barnes-Hut algorithm). In the $\mathcal{O}(N \log N)$ *Barnes-Hut algorithm*, the domain $\Omega \subset R^3$ is recursively subdivided into an octree of cubes that either contain one or zero particles. The nodes of the octree are either *internal nodes* with children, or *external nodes* without. Each internal node in the octree stores the total mass and the center of mass of all its children. In R^2 a quadtree is used analogously, see Figure 14.4. When computing the total force on a particle p_i, the octree is traversed from the root to sum up the contributions from all other particles. If the center of mass of an internal node N_j is sufficiently far away from the position of p_i, then the group of particles that represent children of N_j is treated as one macro particle with center of mass and total mass given by N_j. The threshold criterion can be formulated as

$$s_j < \theta d_j,$$

with s_j the side length of the cube represented by the node N_j in the octree, d_j the distance between the position of the particle p_i and the center of mass of N_j, and $\theta > 0$ a threshold parameter to be chosen.

Barnes-Hut simulation



Compartment models

A compartment model consists of a number of compartments, assumed to be homogeneous and connected in a network, which exchange, for example, data, energy or material with each other. Every compartment is described by a set of state variables $u(t)$, together with an input function $u_{in}(t)$ and an output function $u_{out}(t)$, so that the evolution of the state can be expressed as

$$\frac{du}{dt} = u_{in} - u_{out}.$$

The network architecture defines the relation between the input/output functions by conservation laws, with possible sinks and sources added. Compartment models are used extensively in areas such as biology, sociology and engineering.

Epidemiology

Example 14.10 (Epidemiology). In the SIR model of epidemiology three compartments are used, the susceptible $S(t)$, infectious $I(t)$, and recovered $R(t)$ individuals,

$$u = (u_1, u_2, u_3)^T = (S, I, R)^T.$$

Assuming that the recovered individuals are immune, the network which describes a population with $N = S(t) + I(t) + R(t)$ individuals exposed to an infectious disease is a directed graph

$$S \to I \to R,$$

which can be described by the following initial value problem,

$$\begin{aligned}
\dot{S} &= -\frac{\beta I S}{N}, \\
\dot{I} &= \frac{\beta I S}{N} - \gamma I = \left(R_0 \frac{S}{N} - 1 \right) \gamma I, \\
\dot{R} &= \gamma I,
\end{aligned}$$

where β is the contact rate and γ the recovery rate, and $R_0 = \beta/\gamma$ is the *basic reproduction number*. There can only be an epidemic outbreak if $R_0 > N/S(0)$, and once $R_0 < N/S(t)$ the number of infected individuals starts to decrease.

Epidemiology

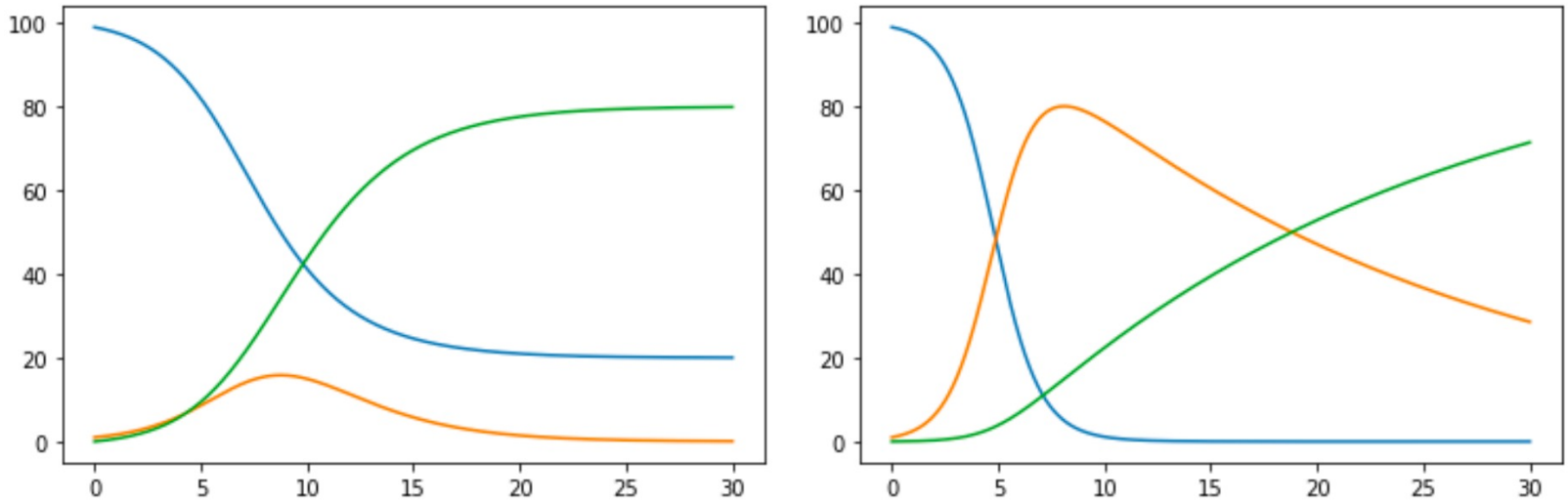


Figure 14.5. *SIR model of an infectious disease, showing the development over 30 days of the percentage of susceptible (blue), infectious (orange) and recovered (green) individuals, starting from 1% infected and 99% susceptible individuals: low R_0 which builds up protective herd immunity in the population (left), and high R_0 that leaves the population insufficient protection (right).*

Predator-Prey systems

Example 14.11 (Predator-prey systems). The *Volterra-Lotka equations* is a model that describes a predator-prey system of two spieces,

$$u = (u_1, u_2)^T,$$

with $u_1(t)$ the prey and $u_2(t)$ the predator. The initial value problem (14.1) then takes the form

$$\begin{bmatrix} \dot{u}_1(t) \\ \dot{u}_2(t) \end{bmatrix} = \begin{bmatrix} u_1(t)(\alpha - \beta u_2(t)) \\ u_2(t)(\delta u_1(t) - \gamma) \end{bmatrix}, \tag{14.14}$$

where $\alpha, \beta, \gamma, \delta > 0$ are positive parameters that models birth and death rates, and the predator-prey interactions. The system has two equilibrium points, the first one is $u_1 = u_2 = 0$, where both species vanish, and the second is

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \gamma/\delta \\ \alpha/\beta \end{bmatrix}.$$

Predator-Prey systems

The linearized Jacobian matrix for the Volterra-Lotka system (14.14) is

$$f'(u^*, t) = \begin{bmatrix} \alpha - \beta u_2^* & -\beta u_1^* \\ \delta u_2^* & \delta u_1^* - \gamma \end{bmatrix},$$

where the two equilibrium points result in the following Jacobian matrices

$$\begin{bmatrix} \alpha & 0 \\ 0 & -\gamma \end{bmatrix}, \qquad \begin{bmatrix} 0 & -\beta\gamma/\delta \\ \delta\alpha/\beta & 0 \end{bmatrix}.$$

The first matrix has eigenvalues $\lambda_1 = \alpha$ and $\lambda_2 = -\gamma$. Hence, $(0, 0)$ is an unstable equilibrium point. The second matrix has purely imaginary eigenvalues $\lambda_1 = i\sqrt{\alpha\gamma}$ and $\lambda_2 = -i\sqrt{\alpha\gamma}$, which corresponds to a periodic solution around the equilibrium point $(\gamma/\delta, \alpha/\beta)$. We illustrate the dynamics of the system in a *phase diagram*, where we plot the graph $(u_1(t), u_2(t))$ for $t \geq 0$.

Predator-Prey systems

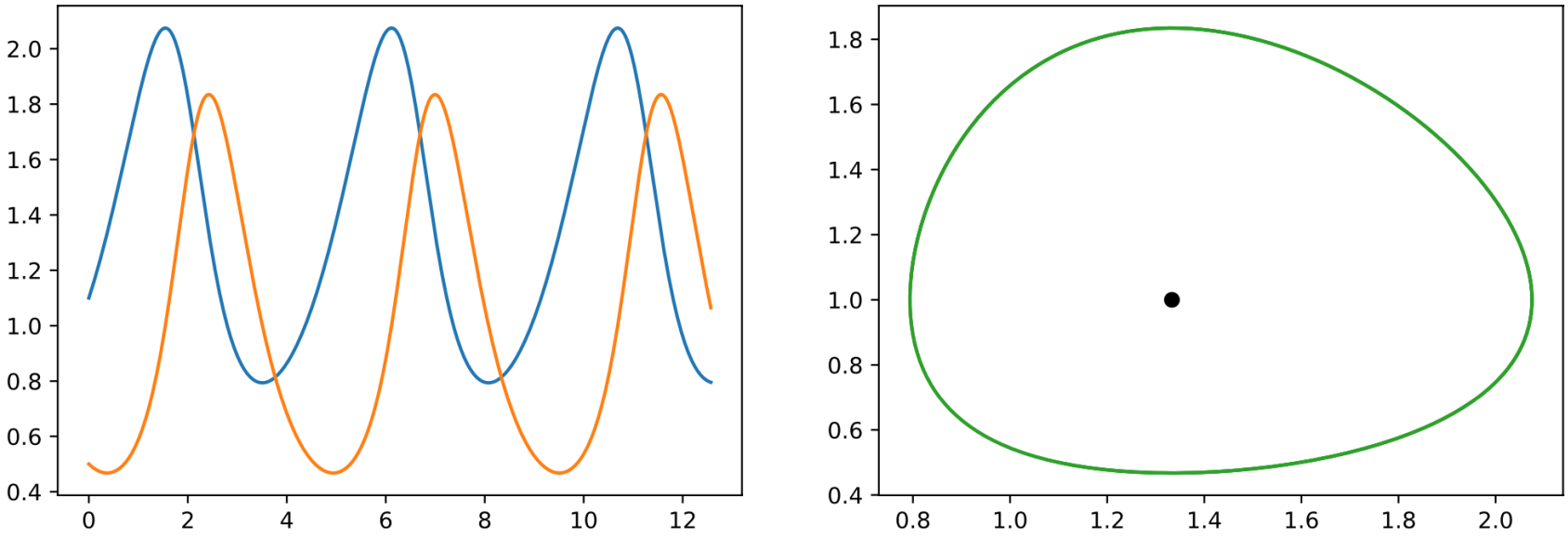


Figure 14.6. *Predator-prey system for* $(\alpha, \beta, \gamma, \delta) = (1, 1, 2, 1.5)$ *and* $u_0 = (1.1, 0.5)$*, showing the time evolution (left) of the number of prey (blue) and predators (orange), and the corresponding phase diagram (right) with the equilibrium point* $(\gamma/\delta, \alpha/\beta)$ *marked.*

Mass-spring systems

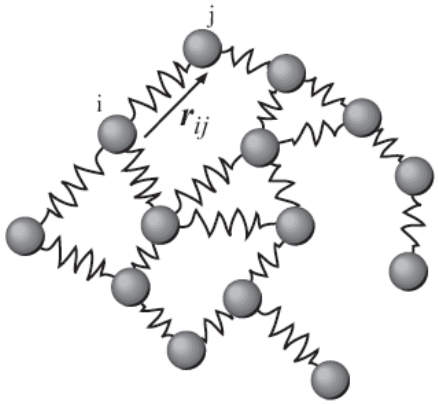
Example 14.13 (Mass-spring system). A *mass-spring system* is a lumped parameter model of an elastic material, in the form of particles which are connected to each other by springs in a mesh topology. That is, each particle is a node in the mesh which is connected to its neighbours by springs, represented by edges in the mesh. Hence, the one particle harmonic oscillator, and the array of masses and springs in Example 14.4 are both special cases of a mass-spring system. Forces in a mass-spring system are pairwise force interactions between adjacent particles in the mesh, where the force F_{ij} which acts on particle p_i by p_j, is given by

$$F_{ij} = -k_{ij}(\|x_i - x_j\| - l_{ij})\frac{x_i - x_j}{\|x_i - x_j\|},$$

with $k_{ij} = k_{ji}$ the spring constant, and l_{ij} its rest length. In a damped mass-spring model,

$$F_{ij} = -k_{ij}(\|x_i - x_j\| - l_{ij})\frac{x_i - x_j}{\|x_i - x_j\|} - c_{ij}(v_i - v_j),$$

with $c_{ij} = c_{ji}$ a damping coefficient. Such a damped mass-spring system can be used to model a viscoelastic material, with the spring forces representing the elastic response and the damping forces the viscous internal friction of the material.



[Wikipedia]

A posteriori error analysis

The a posteriori error analysis of a scalar initial value problem can be extended also to systems, which opens for efficient adaptive approximation algorithms. First consider the linear system

$$\dot{u}(t) + Au(t) = g(t), \quad u(0) = u_0, \quad 0 < t \leq T, \tag{14.16}$$

with $A \in R^{N \times N}$ a system matrix, g a given source term, and u_0 an initial condition. We seek to develop a strategy for adaptive selection of the time step length, based on the accuracy in the approximation. First introduce the corresponding adjoint problem,

$$-\dot{\varphi}(t) + A^T \varphi(t) = \psi(t), \quad \varphi(T) = \varphi_T, \tag{14.17}$$

by which we derive the following a posteriori error representation, analogous to equation (13.17),

$$\int_0^T (e(t), \psi(t))\, dt + (e(T), \varphi(T)) = (e(0), \varphi(0)) + \int_0^T (R(U(t)), \varphi(t))\, dt, \tag{14.18}$$

where $(\cdot, \cdot)$ is the R^N inner product, $U : [0, T] \rightarrow R^N$ is a continuous piecewise polynomial approximation over a subdivision of $[0, T]$, and $R(U(T)) = g(t) - AU(t) - \dot{U}(t)$ is the residual.

A posteriori error analysis

Example 14.17. Consider the SIR model of epidemiology, with solution vector

$$u = (u_1, u_2, u_3)^T = (S, I, R)^T,$$

representing the susceptible $S(t)$, infectious $I(t)$, and recovered $R(t)$ individuals. To plan the response during an epidemic it is important to estimate the number of infected individuals over time, hence, our output of interest is the average number of infected individuals over a time interval $[0, T]$. To control the error in this output of interest, we set the data in the adjoint problem to the following,

$$\psi(t) = (0, 1/T, 0)^T, \quad \varphi_T = 0,$$

which gives us the desired estimate,

$$\int_0^T (e(t), \psi(t))\, dt + (e(T), \varphi(T)) = \frac{1}{T} \int_0^T e_2(t)\, dt.$$

A posteriori error analysis

$$-\dot{\varphi}(t) - \bar{f}'(u, U, t)^T \varphi(t) = \psi(t), \quad \varphi(T) = \varphi_T,$$

with the averaged jacobian matrix defined by

$$\bar{f}'(u, U, t) = \int_0^1 \frac{d}{ds} f(w(s), t)\, ds, \quad w(s) = su(t) + (1-s)U(t),$$

so that

$$\begin{aligned} f(u(t), t) - f(U(t), t) &= \int_0^1 \frac{d}{ds} f(su(t) + (1-s)U(t), t)\, ds \\ &= \bar{f}'(u, U, t)(u(t) - U(t)). \end{aligned}$$

A posteriori error analysis

To derive the a posteriori error estimate, take the inner product of the adjoint equation and the error and then integrate over the time interval $I = [0, T]$, to get

$$\int_0^T (e(t), \psi(t))\, dt = \int_0^T (e(t), -\dot{\varphi}(t) - \bar{f}'(u, U, t)^T \varphi(t))\, dt$$

$$= (e(0), \varphi(0)) - (e(T), \varphi(T)) + \int_0^T (\dot{e}(t) - \bar{f}'(u, U, t)e(t), \varphi(t))\, dt$$

$$= (e(0), \varphi(0)) - (e(T), \varphi(T)) + \int_0^T ((\dot{u} - f(u, t)) - (\dot{U}(t) - f(U, t)), \varphi(t))\, dt$$

$$= (e(0), \varphi(0)) - (e(T), \varphi(T)) + \int_0^T (R(U(t)), \varphi(t))\, dt$$

Adaptive algorithms

$$\int_0^T (e(t), \psi(t))\, dt + (e(T), \varphi(T)) \leq (e(0), \varphi(0)) + \sum_n \mathcal{E}_n. \tag{14.23}$$

The error contribution from each time step I_n is estimated by the error indicator

$$\mathcal{E}_n = \|k_n R(U)\|_{L^2(I_n)} C_i \|\dot{\varphi}\|_{L^2(I_n)}.$$

Based on this error indicator, we can formulate adaptive algorithms to optimize the use of computational resources, and to be able to choose a desired tolerance TOL to bound the error in the output of interest.

Example 14.18. With $\psi = 0$ and $\varphi_T = 1$, we estimate the error at final time, and with V_N an approximation space of continuous piecewise polynomials, we can either adapt the time step length or the degree of the polynomial basis to improve the approximation.

Adaptive algorithms

ALGORITHM 14.1. f = adaptive_time_stepping(f, u0, t0, T, psi, phiT).
Input: function **f**, initial data **u0**, initial time **t0**, final time **T**, adjoint data **psi** and **phiT**.
Output: approximation at final time **u**.

```
1: k = generate_partition(0, T, error_indicators)
2: while estimated error > TOL do
3:     u = primal_problem_time_stepping(f, u0, t0, T, k)
4:     phi = dual_problem_time_stepping(psi, phiT, u, t0, T, k)
5:     error_indicators = error_estimation(u, phi, k)
6: end while
7: return u
```

Partial differential equation

A *partial differential equation* (PDE) is a differential equation which expresses relations between partial derivatives of a function of several variables. If we move all terms to one side of the equation, we can state any PDE on residual form as

$$R(u(x,t)) = 0,$$

which describes the evolution of a function

$$u : \Omega \times I \to R^M,$$

with $\Omega \subset R^d$ a spatial domain and $I = [0, T]$ a time interval. If $M = 1$, the solution to the PDE is a scalar valued function, otherwise it is vector valued.

Semi-discretization

Here we first discretize the spatial domain Ω, e.g. by a structured grid or an unstructured mesh, to obtain an approximation

$$u(x,t) \approx \sum_{i=1}^{N} U_i(t)\phi_i(x),$$

with $\{\phi\}_{i=1}^{N}$ a set of scalar or vector valued basis functions. The evolution of the degrees of freedom $\{U_i(t)\}_{i=1}^{N}$ can then be described by a system of NM initial values problems, assuming that we use N degrees of freedom for each vector component of the approximation. This system of initial value problems is then solved by a time stepping method.

Finite difference method

Example 14.19 (Finite difference method). Analogous to Example 5.9, consider now the time dependent convection-diffusion partial differential equation

$$\dot{u}(x,t) + u'(x,t) - \epsilon u''(x,t) = f(x), \quad u(x,0) = 0, \quad (x,t) \in \Omega \times I,$$

with a diffusion coefficient $\epsilon > 0$, spatial interval $\Omega = [0,1]$, and suitable conditions on $u(x,t)$ at the boundary points $x = 0$ and $x = 1$. With a finite difference method we can approximate the differential equation by the matrix equation

$$\dot{U}(t) + (D_h^-)U(t) - \epsilon(D_h^+ D_h^-)U(t) = b, \quad U(t) = (U_i(t)), \quad b = (b_i),$$

with $b_i = f(x_i)$, and $U_i(t) = U(x_i,t)$ the values at the nodes x_i of the grid.

Finite element method

Example 14.20 (Finite element method). Now we instead approximate the partial differential equation in Example 14.19 by a finite element method, based on the following variational formulation: for each $t \in I$ find $U(t) \in V_N$, such that

$$(\dot{U}(t), v) + a(U(t), v) = L(v), \quad \forall v \in V_N,$$

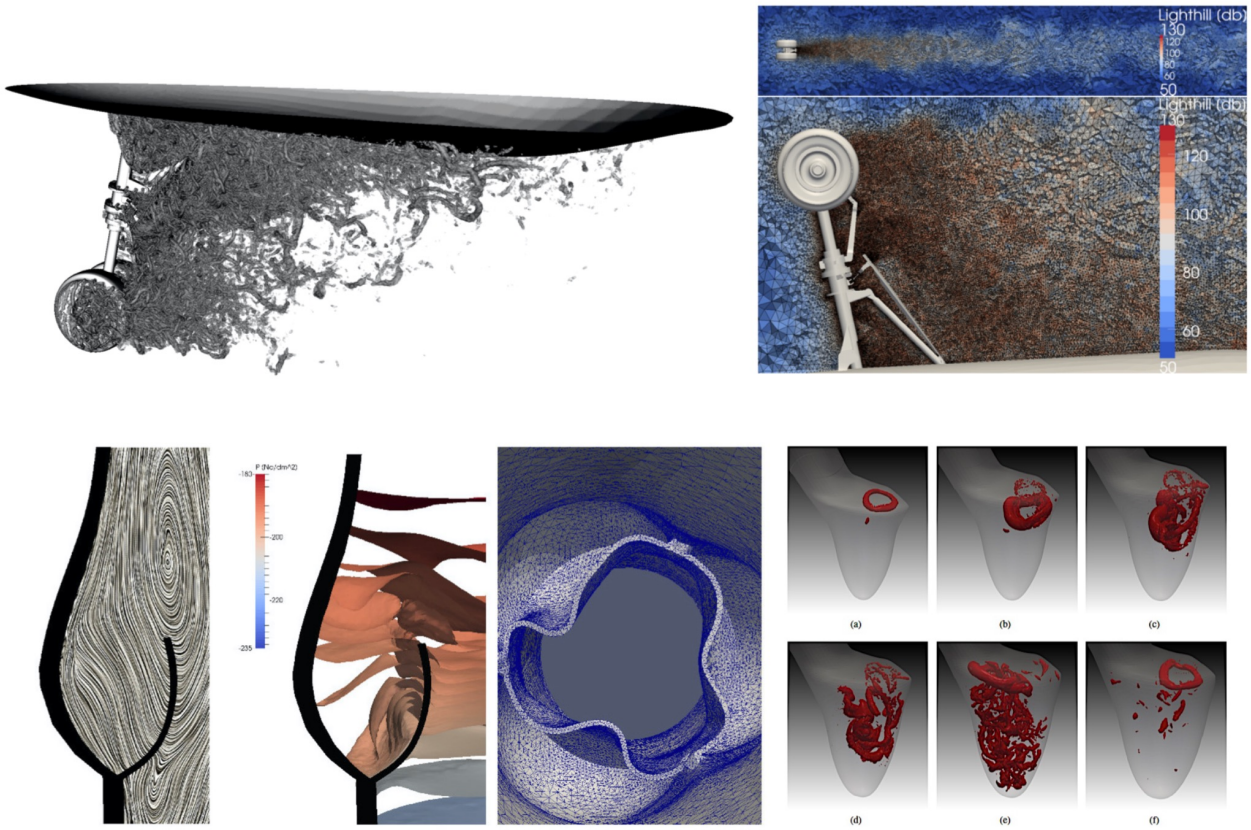
with V_N a piecewise polynomial approximation space defined over an unstructured mesh, and with $a(\cdot, \cdot)$ a bilinear form and $L(\cdot)$ a linear form. This is equivalent to a matrix equation for the vector $U(t) = (U_j(t))$, the degrees of freedom in the piecewise polynomial basis,

$$M\dot{U}(t) + AU(t) = b,$$

with M a mass matrix, A the matrix which corresponds to the bilinear form $a(\cdot, \cdot)$, and b vector generated by the linear form $L(\cdot)$. We can express the matrix equation on the form (14.1), as

$$\dot{U}(t) = f(U(t), t) = M^{-1}(b - AU(t)).$$

Finite element method



Extra (chapter 1)

Differential and integral operators

Subdivide the interval [0,1] into

$$0 = x_0 < x_1 < x_2 < ... < x_n = 1$$

with a constant interval length, or grid size, $h = x_i - x_{i-1}$ for all i, so that $x_i = x_0 + ih$. If $f(x_0) = f(0) = 0$, for each $x = x_i$ we may approximate the primitive function $F(x)$ by

$$F(x_i) = \int_0^{x_i} f(s)ds \approx \sum_{k=1}^{i} f(x_k)h \equiv F_h(x_i).$$

Equation (5.20) defines a function $F_h(x_i) \approx F(x_i)$ for all grid points x_i in $\mathcal{T}^h$, referred to as a Riemann sum. The function F_h can also be interpreted as a linear operator L_h that maps $y = (f(x_1), ..., f(x_n))^T$, the vector of sampled function values at the grid points of $\mathcal{T}^h$, to the vector $(F_h(x_1), ..., F_h(x_n))^T$. In component form, this is expressed as

Differential and integral operators

$$\text{For } f(x_0) = f(0) = 0, \;\; F(x_i) = \int_0^{x_i} f(s)ds \approx \sum_{k=1}^{i} f(x_k)h \equiv F_h(x_i)$$

$$L_h y = \begin{bmatrix} h & 0 & \cdots & 0 \\ h & h & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ h & h & \cdots & h \end{bmatrix} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} = \begin{bmatrix} f(x_1)h \\ f(x_1)h + f(x_2)h \\ \vdots \\ \sum_{k=1}^{n} f(x_k)h \end{bmatrix}$$

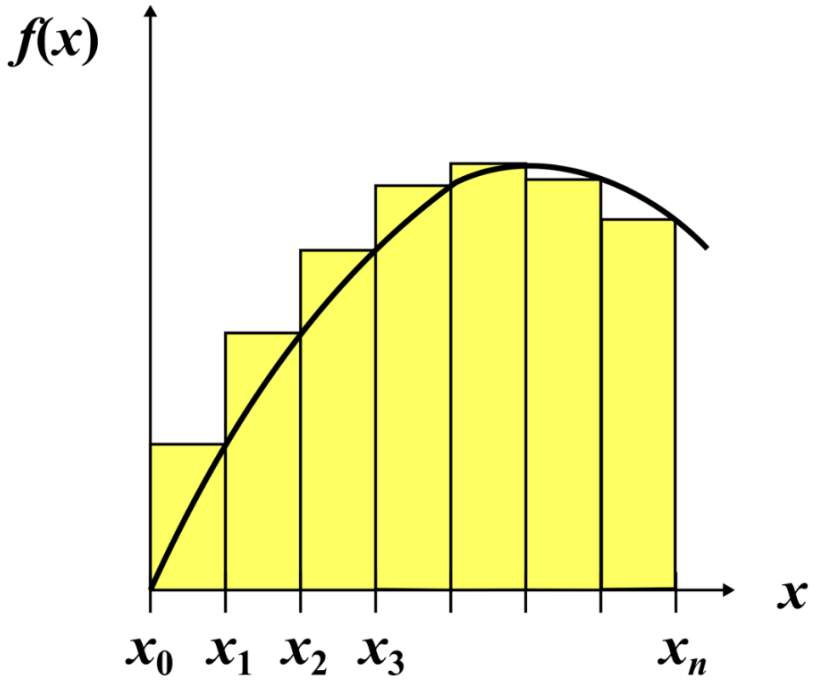
Differential and integral operators

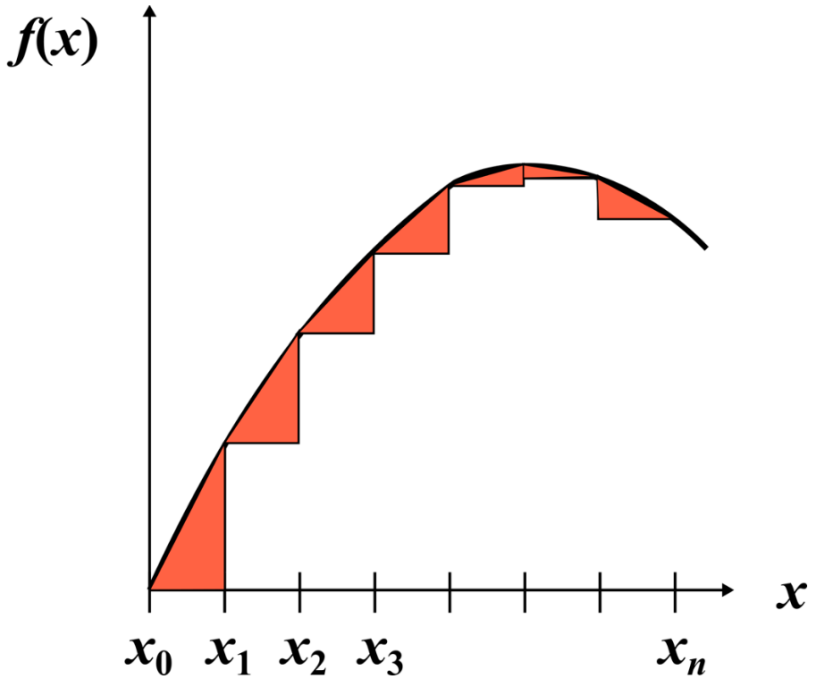
$$L_h = h \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \quad \Leftrightarrow \quad L_h^{-1} = h^{-1} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & -1 & 1 \end{bmatrix}$$

Differential and integral operators

$$L_h^{-1} y = h^{-1} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & -1 & 1 \end{bmatrix} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} = \begin{bmatrix} (f(x_1) - f(x_0))/h \\ (f(x_2) - f(x_1))/h \\ \vdots \\ (f(x_n) - f(x_{n-1}))/h \end{bmatrix}$$

Differential and integral operators





Differential and integral operators

Formally, as the interval length $h \to 0$, the summation and difference matrices converge to infinite dimensional integral and differential operators, so that for each $x \in (0, 1)$,

$$L_h y \to \int_0^x f(s)ds, \quad L_h^{-1} y \to f'(x),$$

under suitable assumptions on f. Further, as $h \to 0$, the product of the two matrices expresses the fundamental theorem of calculus,

$$y = L_h L_h^{-1} y \to f(x) = \int_0^x f'(s)ds.$$

Difference operators

The matrix L_h^{-1} in (5.21) is referred to as a *backward difference operator*, denoted by D_h^-, and similarly we can define a *forward difference operator* D_h^+,

$$D_h^- = h^{-1}\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & -1 & 1 & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}, \quad D_h^+ = h^{-1}\begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 \\ 0 & \cdots & 0 & 0 & -1 \end{bmatrix},$$

with a *central difference operator* obtained from $(D_h^+ + D_h^-)/2$.

Difference operators

$D_h^+ D_h^-$ is an approximation of a second order differential operator, and takes the form

$$D_h^+ D_h^- = h^{-2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -1 \end{bmatrix}.$$

Difference operators

Now consider a vector $y = (u(x_i))$, with $u(x_i)$ function values sampled at the grid points x_i of the structured grid $\mathcal{T}^h$. Then the ith row of the matrix $D_h^+ D_h^-$ corresponds to a *finite difference stencil* acting on the vector of function values, see Figure 5.9, and we can write the ith component of the matrix-vector product as

$$[(D_h^+ D_h^-)y]_i = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} = \frac{\dfrac{u(x_{i+1}) - u(x_i)}{h} - \dfrac{u(x_i) - u(x_{i-1})}{h}}{h}.$$

Analogously, the difference operators D_h^- and D_h^+ also correspond to finite difference stencils over the grid,

$$[(D_h^+)y]_i = \frac{u(x_{i+1}) - u(x_i)}{h}, \quad [(D_h^-)y]_i = \frac{u(x_i) - u(x_{i-1})}{h},$$

Finite difference method

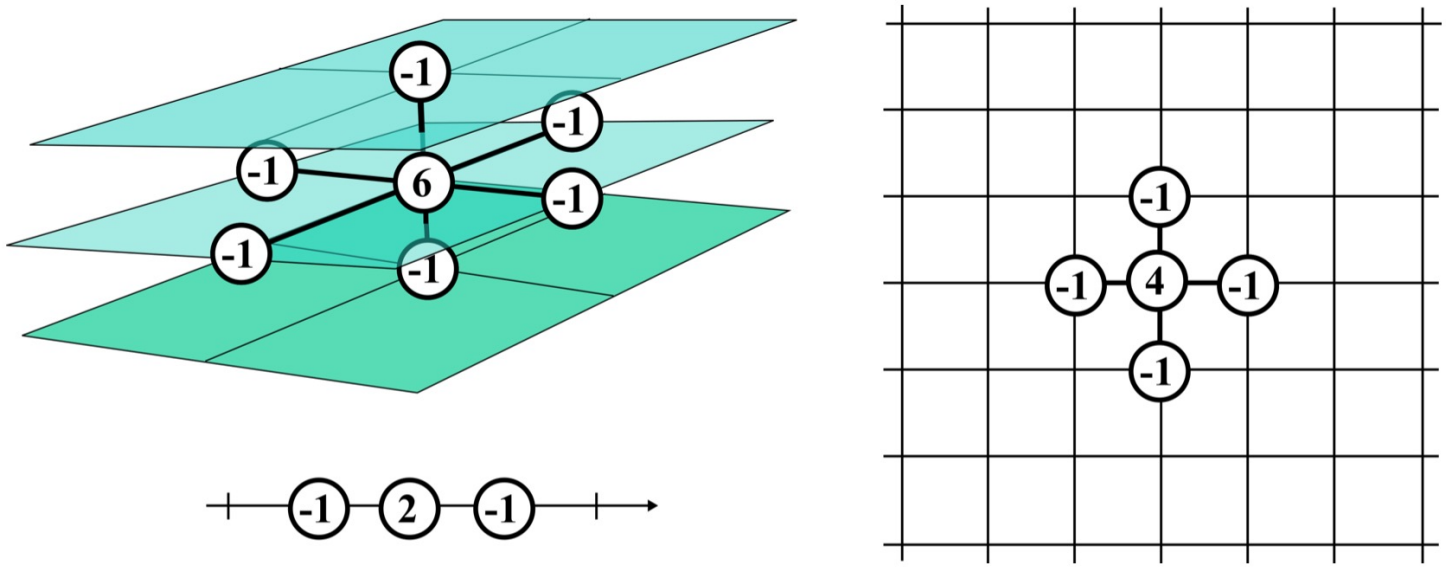


Figure 5.9. *Examples of finite difference stencils that correspond to the difference operator* $-(D_h^+ D_h^-)$ *over uniform grids in* R *(lower left),* R^2 *(right) and* R^3 *(upper left).*

Finite difference method

Example 5.9. Consider the one dimensional convection-diffusion differential equation

$$-\epsilon u''(x) + u'(x) = f(x),$$

which may describe the transport and diffusion of the concentration of air pollution $u(x)$ at a coordinate $x \in R$, with $\epsilon > 0$ a diffusion coefficient and $f(x)$ the source of pollution. By using the finite difference method with $y = (u(x_i))$, we can approximate the differential equation (5.26) by the matrix equation

$$-\epsilon(D_h^+ D_h^-)y + (D_h^-)y = b, \quad b_i = (f(x_i)).$$

Finite difference method

Example 5.10. The Poisson differential equation over the domain $\Omega = [0, 1]$, with *boundary conditions* at $x = 0$ and $x = 1$, takes the form

$$-u''(x) = f(x), \quad u(0) = u(1) = 0,$$

and describes diffusion phenomena such as heat conduction. If $f(x) = 0$, the equation is referred to as *Lapace equation*. By a finite difference method, the solution can be approximated by

$$y = A^{-1}b,$$

with $y = (u(x_i))$ and $b = (f(x_i))$, and where A is a second order difference matrix with a unique inverse A^{-1},

$$A = h^{-2}\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \quad \Leftrightarrow \quad A^{-1} = h^2/6 \begin{bmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 8 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 8 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}.$$

Convolution (filtering)

$$S = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad B = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution (filtering)