Numerical Approaches for Solving (Stiff) Differential Equations

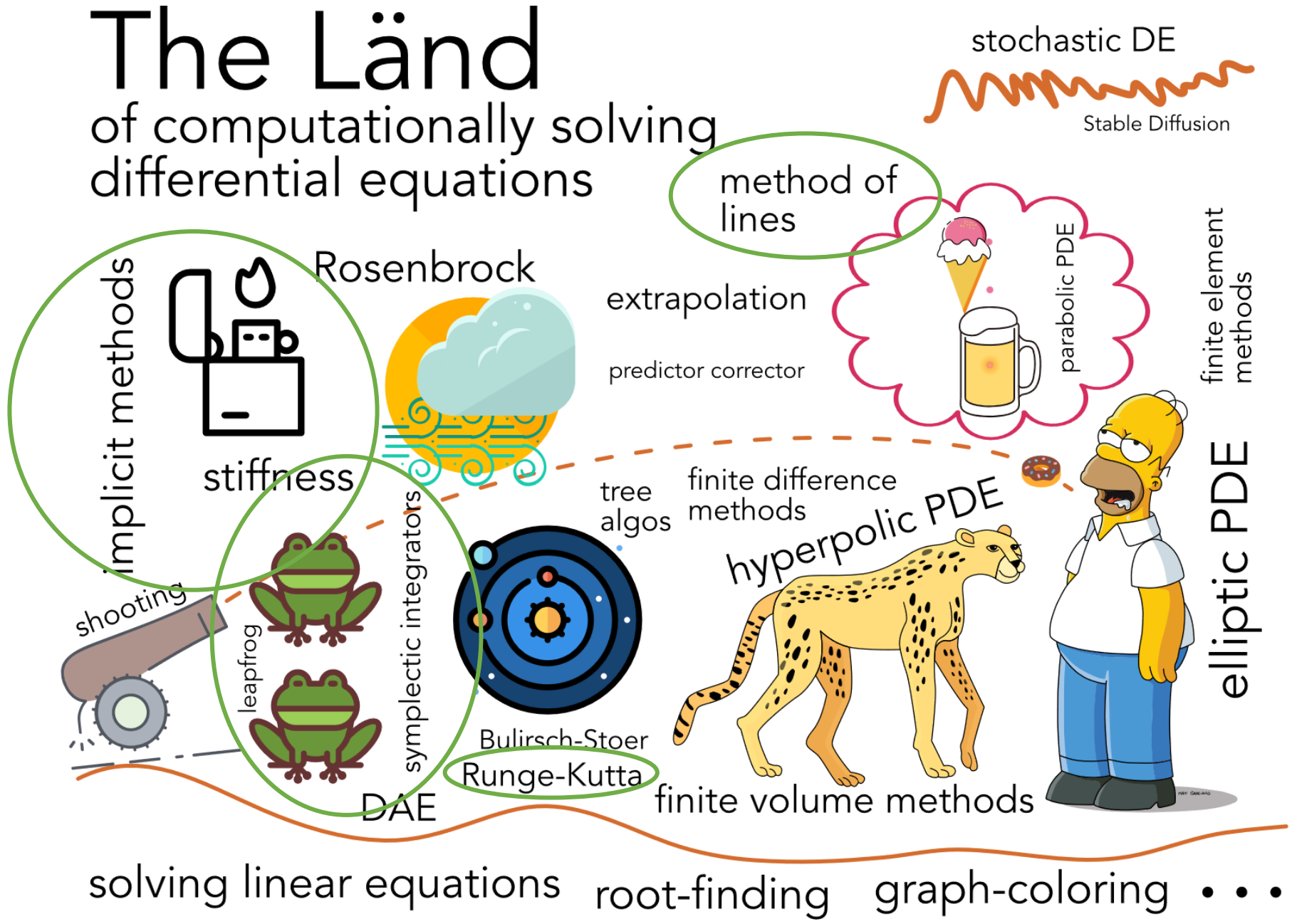
Introduction to basic and commonly used methods and their limitations.

Seminar talk in the Scientific Machine Learning Seminar by Dr. Tobias Buck, AstroAI-Lab

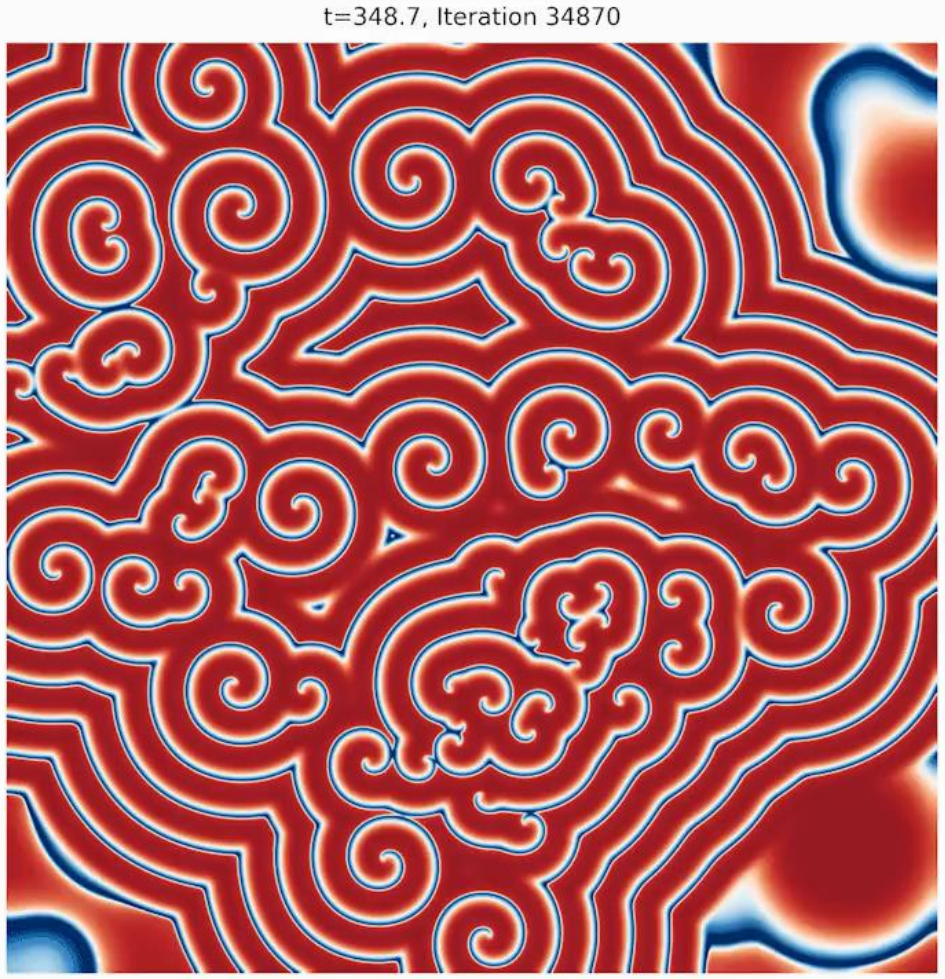
Heidelberg University, Department for Physics and Astronomy

by Daniel Richter and Leonard Storcks, 23.05.2023

It's big.


The Länd
of computationally solving
differential equations
stochastic DE
Stable Diffusion
method of lines
extrapolation
predictor corrector
parabolic PDE
finite element methods
Rosenbrock
implicit methods
stiffness
tree algos
finite difference methods
hyperpolic PDE
elliptic PDE
shooting
leapfrog
symplectic integrators
Bulirsch-Stoer
Runge-Kutta
DAE
finite volume methods
solving linear equations
root-finding
graph-coloring • • •

Motivation



Source: H-Psi-is-E-Psi, Brusselator Oscillations HQ Render, CC BY-SA 4.0

A beautiful oscillatory system involving chemistry and diffusion the Brusselator.

… and its stiff

Context of SciML

»Scientific machine learning is the burgeoning field combining techniques of machine learning into traditional scientific computing and mechanistic modeling.« [e. g. Zubov et al, 2021]

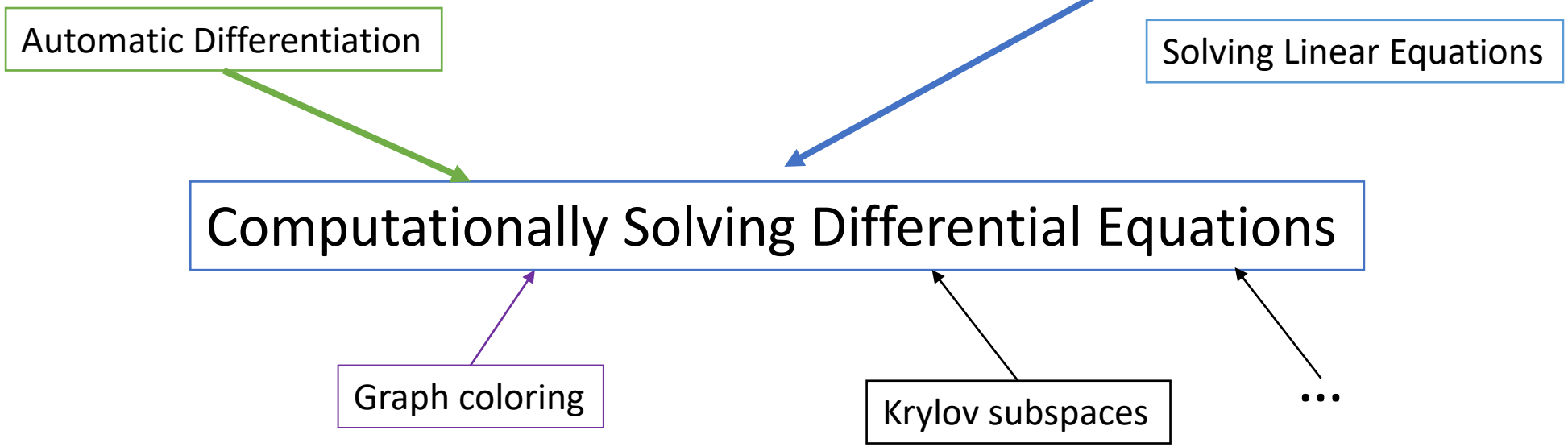


Table of Contents

main part

Some Classical Methods for Solving ODEs

Explicit Euler Method

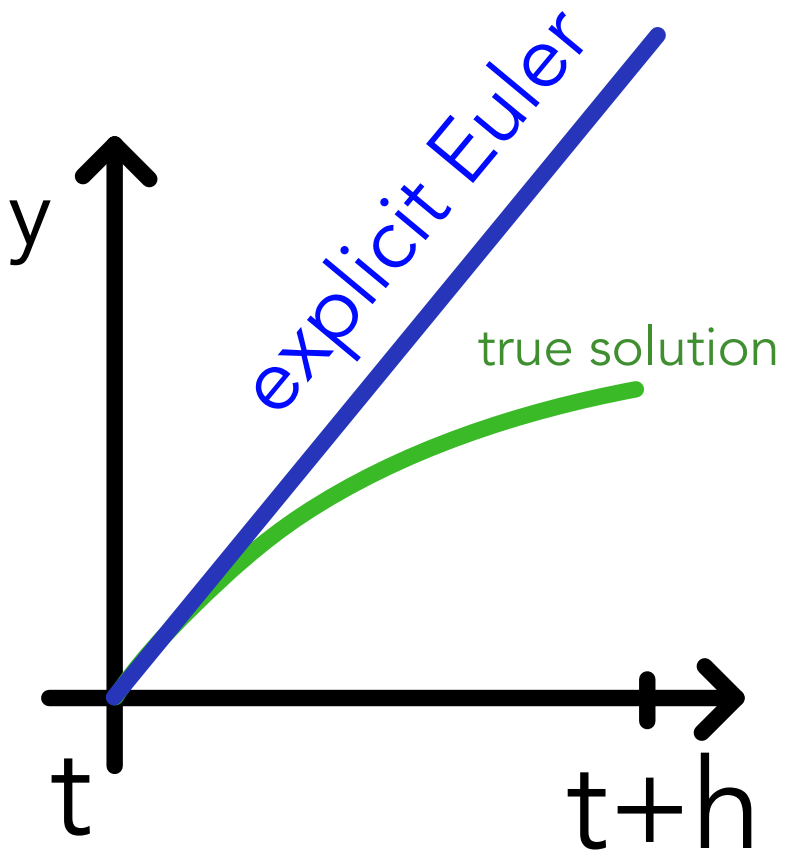
Euler Method for Solving ODEs $\partial_t \underline{y} = \underline{f}(\underline{y})$

$$\underline{y}^{(n+1)} = \underline{y}^{(n)} + \underline{f}\left(\underline{y}^{(n)}\right) \Delta t$$

Live coding stability analysis example

$$\frac{d\underline{y}}{dt} = -\alpha \underline{y}, \qquad \alpha > 0, \underline{y}(0) = \underline{y}_0$$

$$\text{known solution } \underline{y}(t) = \underline{y}_0 e^{-\alpha t}$$



What is the Order of Explicit Euler?

Taylor Expansion

$$\underline{y}_{n+1} = \underline{y}_n + \underline{y}'_n \, \Delta t + \mathcal{O}_s(\Delta t^2)$$

To simulate over a time T, we need $N_s = \frac{T}{\Delta t}$ steps, so the error over time scales with

$$N_s \mathcal{O}_s(\Delta t^2) = \mathcal{O}_T(\Delta t)$$

- first order accuracy.

Runge-Kutta-Method

Q: How would you construct a higher order scheme?

Runge-Kutta Idea: Weighted combination of simple derivatives

$$\underline{y}_{n+1} = \underline{y}_n + h \sum_{i=1}^{m} \beta_i \underline{k}_i$$

$$\underline{k}_i = f\left(\left(\underline{y}_n + h \sum_{l=1}^{m-1} \alpha_{i,l} \underline{k}_l\right), t_n + \gamma_i h\right), \qquad i = 1, \dots, m$$

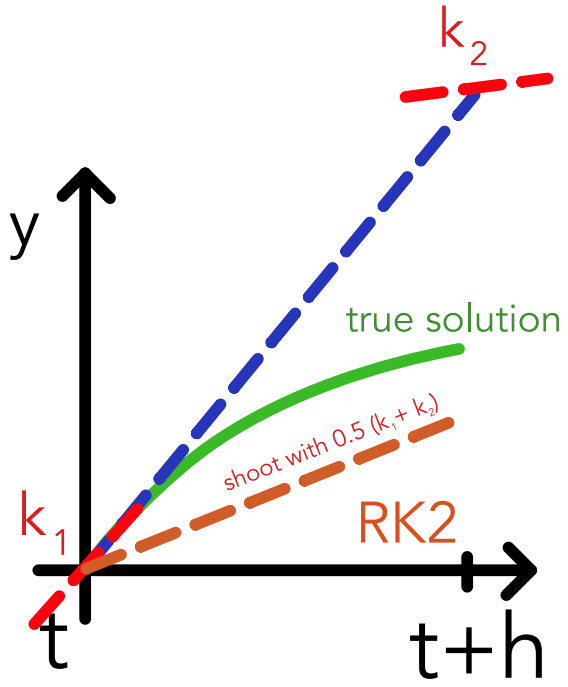
$$\sum_{l=1}^{m} \alpha_{i,l} = \gamma_i, \qquad \alpha_{i,l} = 0 \text{ for } l \geq i \rightarrow \text{explicit}$$

RK2

$$k_1 = f(y_n, t_n)$$

$$k_2 = f(y_n + \Delta t k_1, t_n + \Delta t)$$

$$y_{n+1} = y_n + \frac{\Delta t}{2}(k_1 + k_2) + \mathcal{O}(\Delta t^3)$$

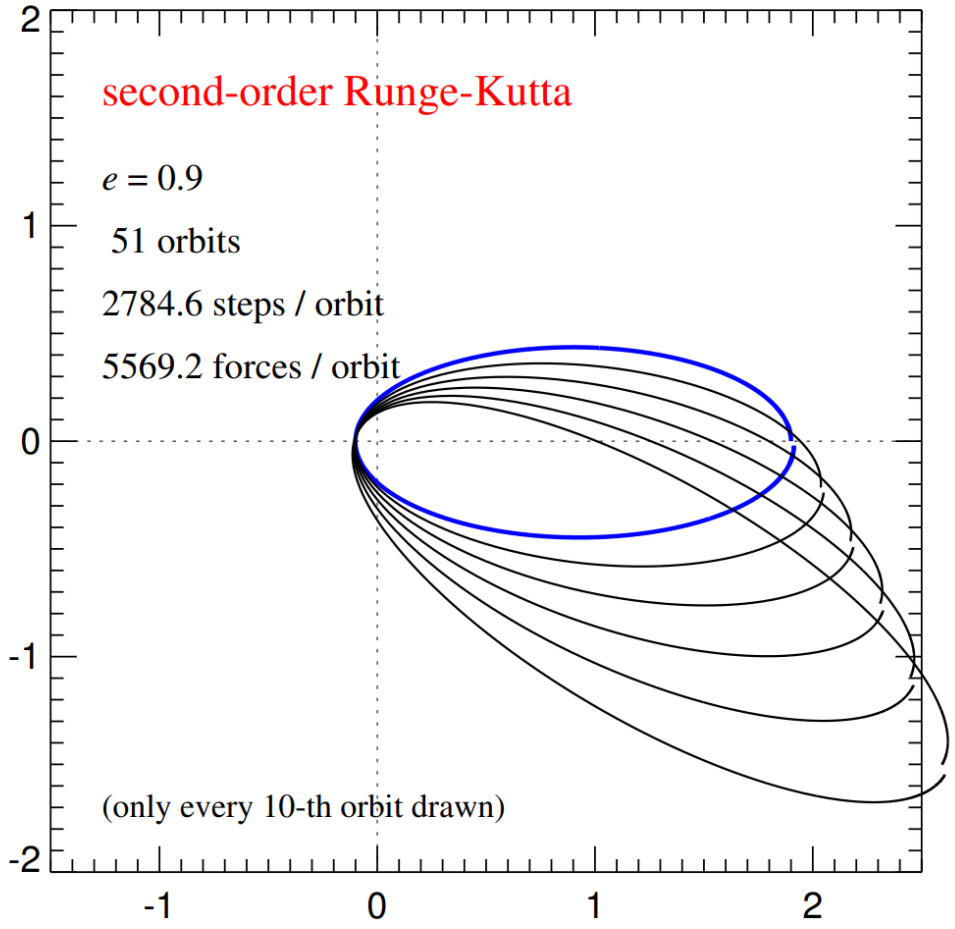


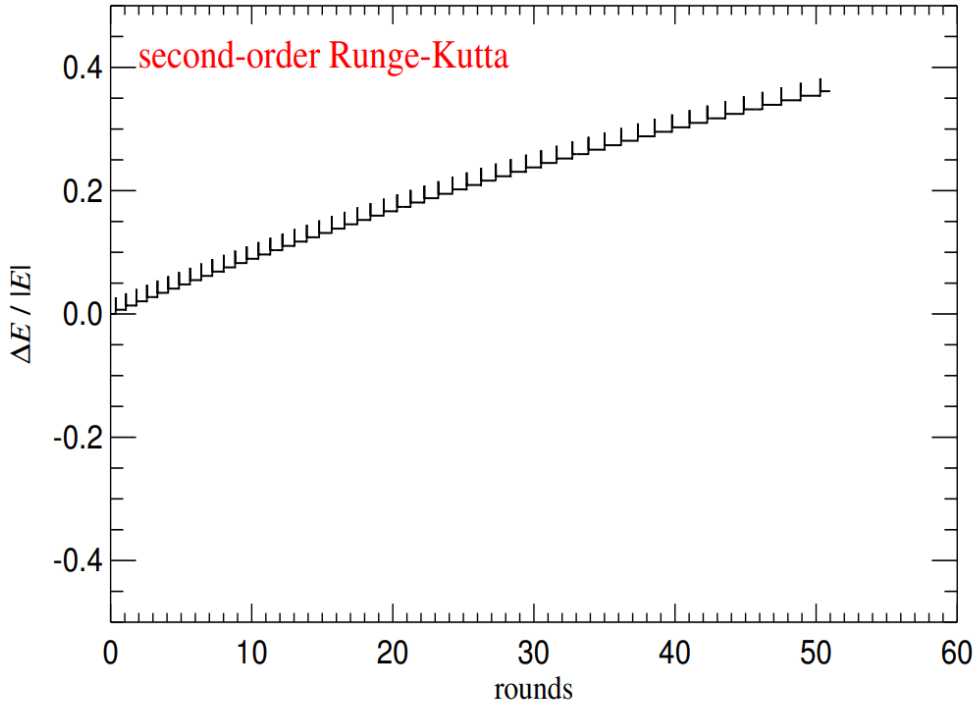
Outlook on further RK methods

- e. g. RK Fehlberg 4/5 with included error estimation

The Problem of Conserved Quantities

Solution of the two-body Problem

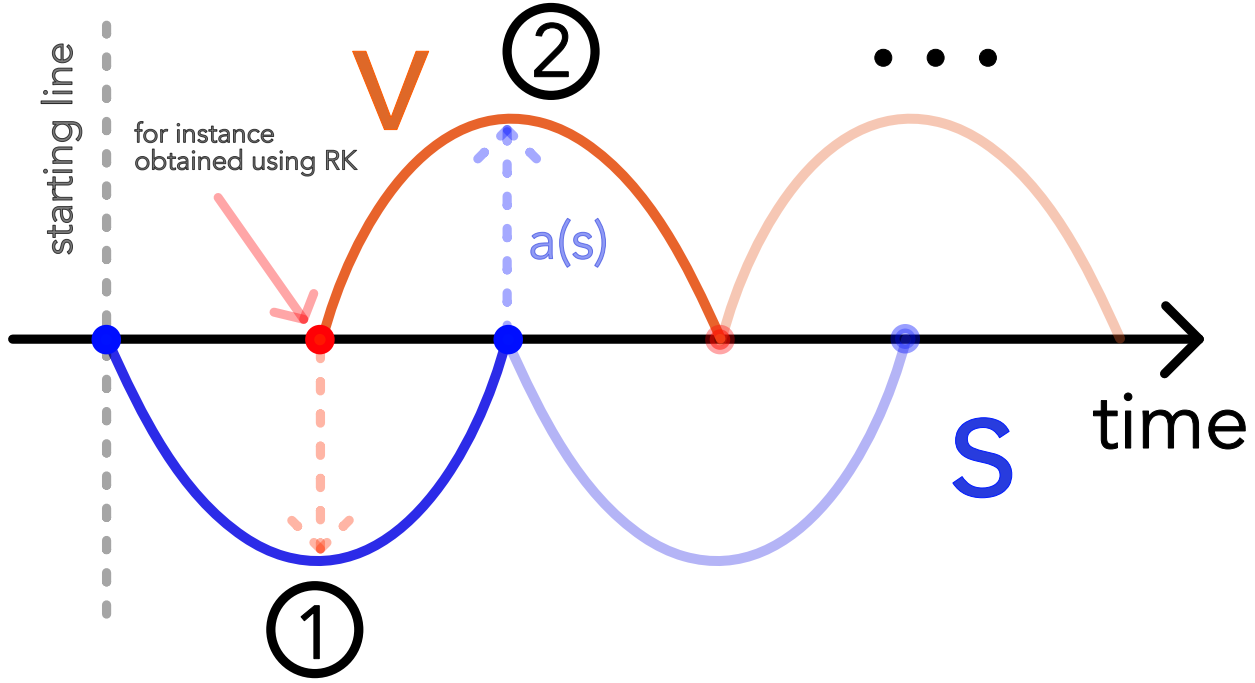




Energy error in each step!

[from the Fundamentals of Simulation Methods lecture notes by Springel et al., version 2022/2023]

Leapfrog



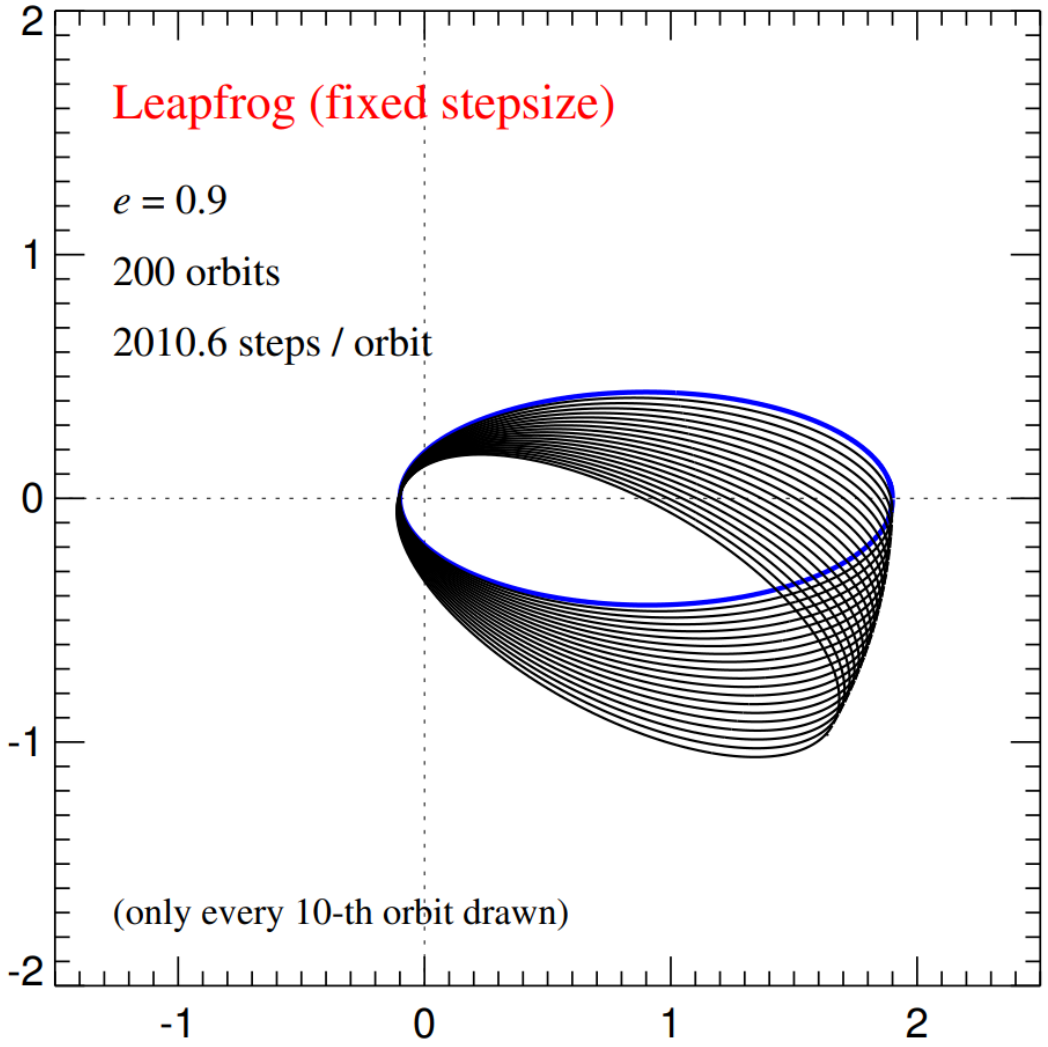
1. update location:

$$\underline{s}\left(t+\frac{1}{2}\Delta t\right)=\underline{s}\left(t-\frac{1}{2}\Delta t\right)+\underline{v}(t)\Delta t+\mathcal{O}(\Delta t^3)$$

2. update velocity:

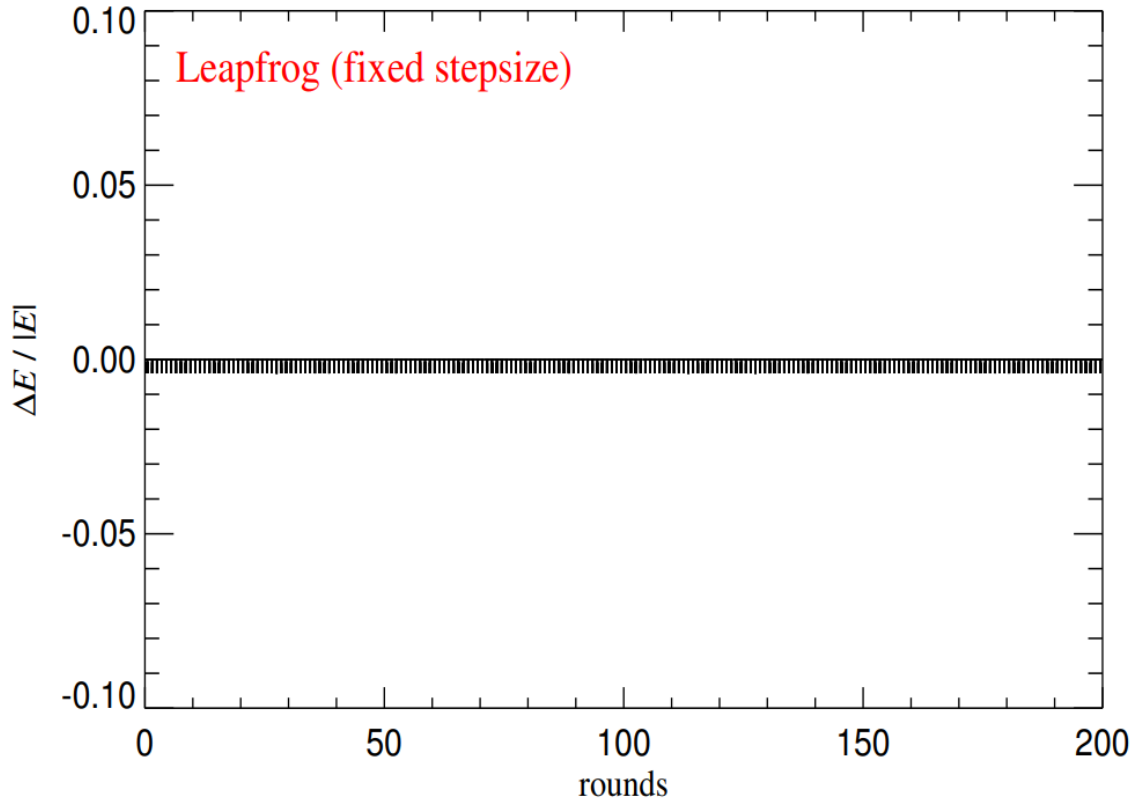
$$\underline{v}(t+\Delta t)=\underline{v}(t)+\underline{a}\left(t+\frac{1}{2}\Delta t\right)\Delta t+\mathcal{O}(\Delta t^3)$$

Symplectic integrator with good energy conservation properties as of time reversibility and exact angular momentum conservation.



Exact solution to $H_{leap} = H + H_{err}$,

$$H_{err} \propto \frac{\Delta t^2}{12}\left\{\{H_{kin}, H_{pot}\}, H_{kin} + \frac{1}{2}H_{pot}\right\} + \mathcal{O}(\Delta t^3)$$



[from the "Fundamentals of Simulation Methods" lecture notes by Springel et al., version 2022/2023]

Intuition of Symplecticity in 2D

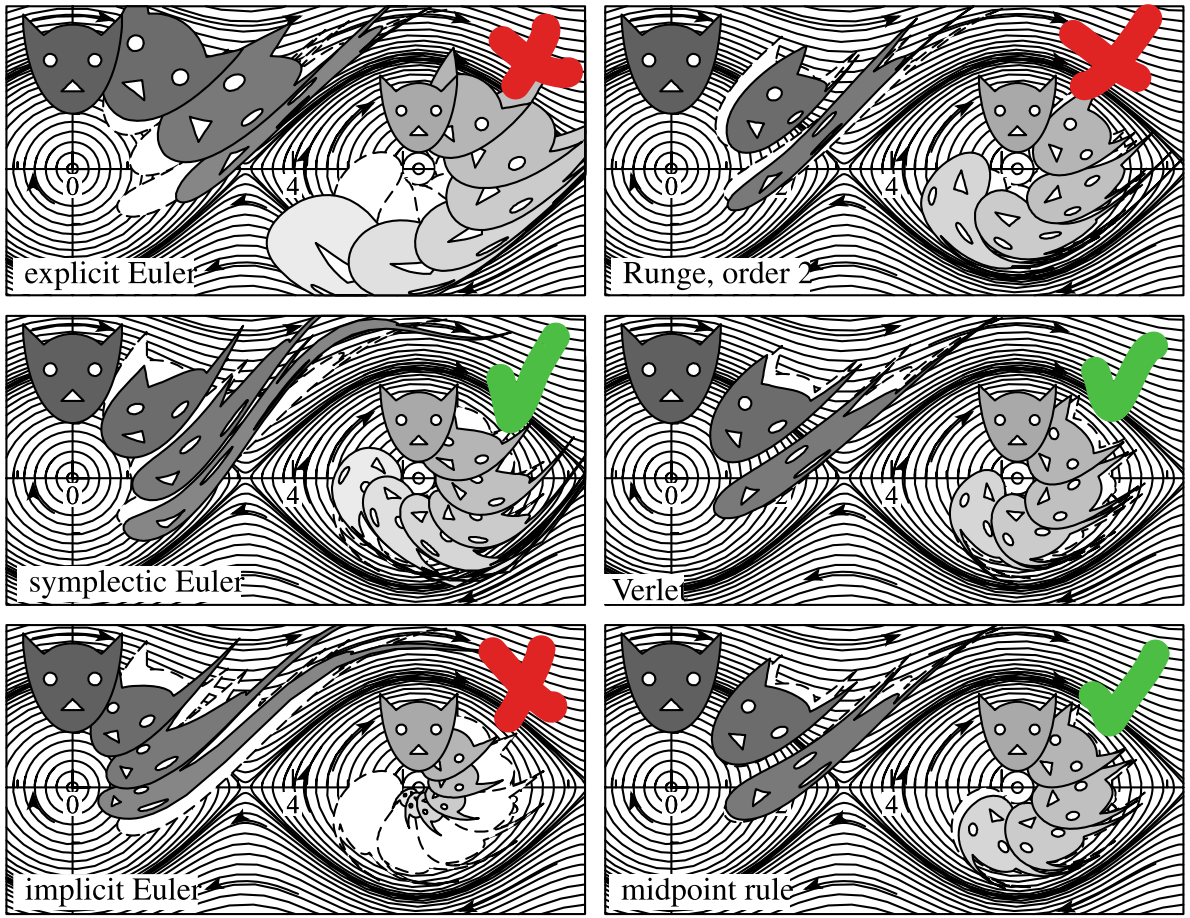
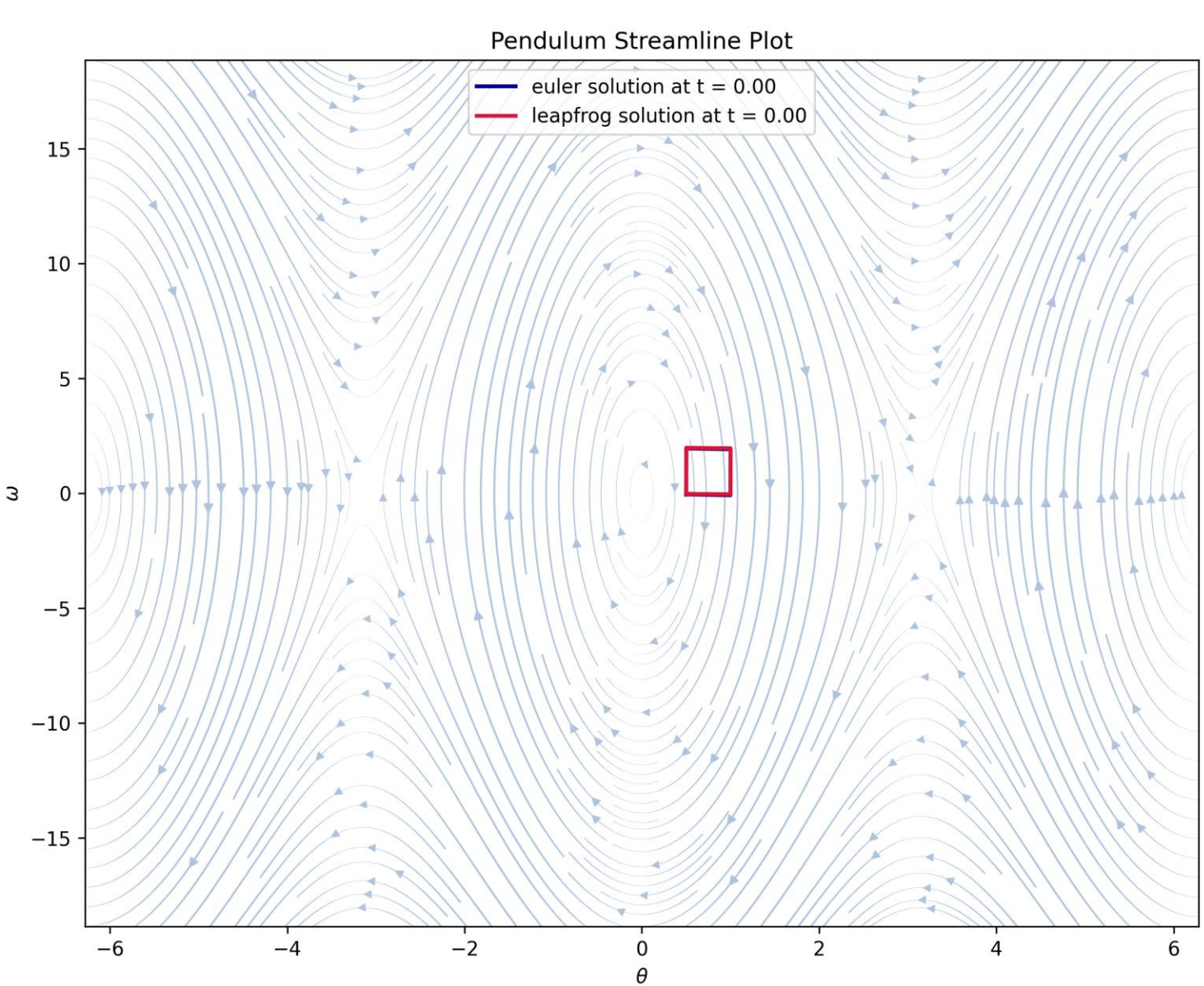


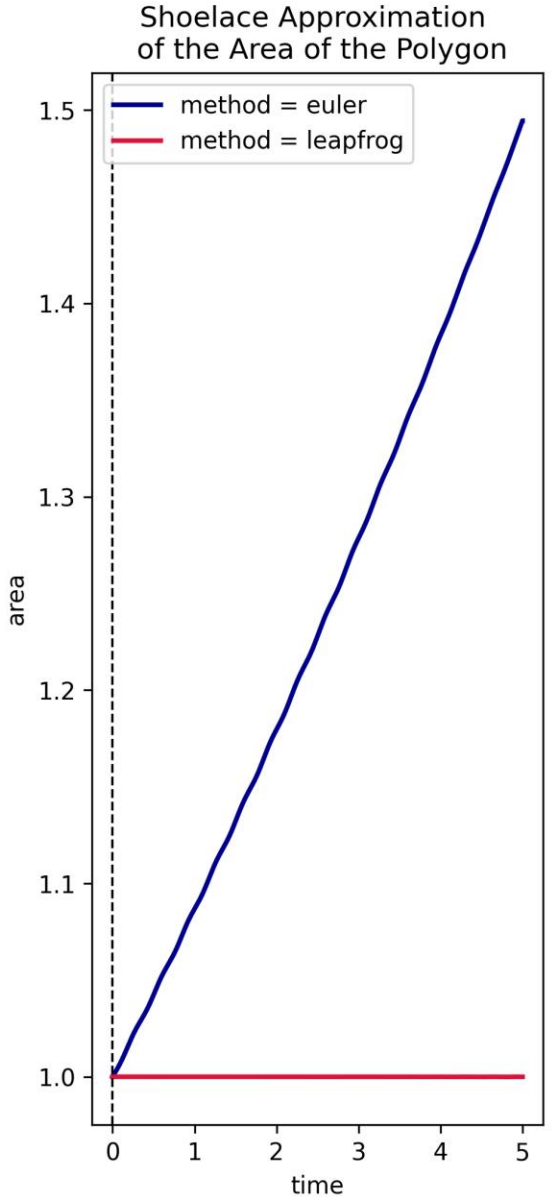
Fig. 3.1. Area preservation of numerical methods for the pendulum; same initial sets as in Fig. 2.2; first order methods (left column): $h = \pi/4$; second order methods (right column): $h = \pi/3$; dashed: exact flow

A symplectic transformation is area preserving.

- Note: No exact energy conservation, e. g. leapfrog exactly solves perturbed Hamiltonian
- Phase space conservation ⟺ long term energy error bounded, but floating-point error → small energy drift
- recommended for long-term integration, e. g. molecular dynamics, planetary orbitals, …

[from chapter 6 in "Geometrical Numerical Integration", Hairer, Lubich, Wanner, 2005]





(leapfrog in kick-drift-kick scheme)

The Problem of Stiffness

Linear ODE example

$$\partial_t u = 998u + 1998v, \qquad \partial_t v = -999u - 1999v, \qquad u(0) = 1, \qquad v(0) = 0$$

$$\partial_t \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}, \qquad \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Analytic solution

$$\begin{pmatrix} u \\ v \end{pmatrix} = \exp\left(\begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix} t \right) \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 2 \\ -1 \end{pmatrix} \exp(-1t) + \begin{pmatrix} -1 \\ 1 \end{pmatrix} \exp(-1000t)$$

ill-conditioned matrix

$$\text{stiffness ratio} = \frac{\text{Re}\,\lambda_{max}}{\text{Re}\,\lambda_{min}} = \frac{-1000}{-1}$$

decay on vastly different time-scales

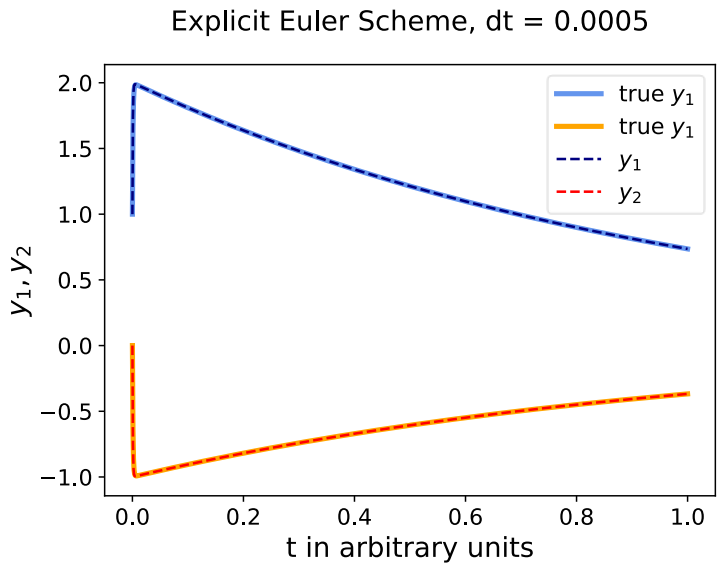
Example from the "Numerical Recipes" book

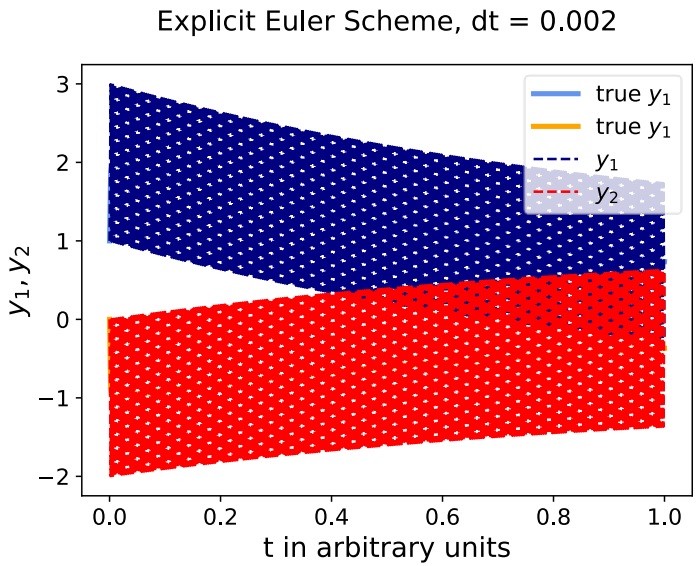
Explicit Schemes in Failure Mode

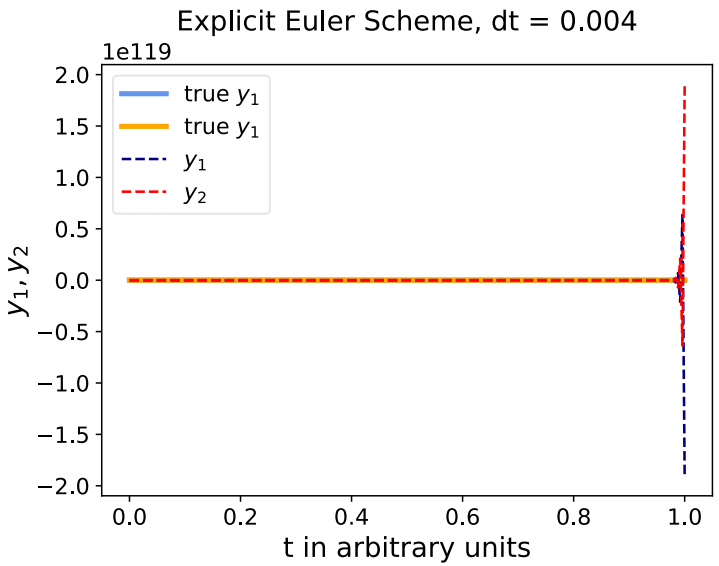
$$\Delta t < \frac{1}{1000} \rightarrow \text{stable}$$

$$\frac{1}{1000} < \Delta t \leq \frac{2}{1000} \rightarrow \text{oscillations}$$

$$\Delta t > \frac{2}{1000} \rightarrow \text{unstable}$$



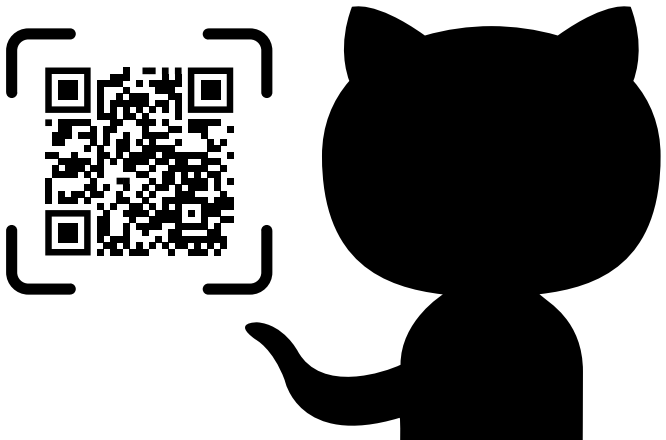




Timestep excessively small to large scale solution smoothness necessary in explicit method → **Stiffness** → more efficient approach?

Efficiently Solving Stiff ODEs

Overview on Solving Stiff ODEs



Implicit Euler Method for Solving Stiff ODEs $\partial_t \underline{y} = \underline{f}(\underline{y})$

$$\text{implicit step } \underline{y}^{(n+1)} = \underline{y}^{(n)} + \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right)$$

non-linear system

Formulate into root-finding problem

$$\underline{0} = \underline{y}^{(n+1)} - \underline{y}^{(n)} - \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right) =: \underline{g}(\underline{\xi})$$

and find the solution iteratively using Newton iteration

$$\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_g^{-1}(\underline{\xi}_k)\, \underline{g}(\underline{\xi}_k), \qquad \underline{\underline{J}}_g = \underline{\underline{1}} - \gamma(\Delta t)\, \underline{\underline{J}}_f$$

$$\text{alternatively quasi} - \text{Newton } \underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_g^{-1}(\underline{\xi}_0)\, \underline{g}(\underline{\xi}_k)$$

$$\rightarrow \text{linear equation } \underline{b} := \underline{g}(\underline{\xi}_k) = \underline{\underline{J}}_g(\underline{\xi}_k - \underline{\xi}_{k+1}) = \underline{\underline{J}}_g \underline{a}$$

sparse Jacobian

Calculate sparse Jacobian using sparse Differentiation (Autodiff with smart direction based on distance-1 graph coloring insights)

Solve linear equation using matrix decomposition $\underline{\underline{J}}_g = \underline{\underline{L}}\, \underline{\underline{R}}$. In Quasi-Newton reuse expensive $(\mathcal{O}(N^3))$ decomposition.

Option A for solving the linear system involving the Jacobian

Jacobian too large to store

Solve $\underline{\underline{J}}_g \underline{a} = \underline{b}$ without calculating $\underline{\underline{J}}_g$

Calculate only Jacobian-Vector products using Autodiff and solve the Linear equation using Krylov-Subspace methods.

Option B for solving the linear system involving the Jacobian

linear system

$$\partial_t \underline{y} = -\underline{\underline{C}}\, \underline{y}$$

Every step corresponds to solving a linear equation

$$\left(\underline{\underline{1}} + \Delta t\, \underline{\underline{C}}\right) \underline{y}_{n+1} = \underline{y}_n$$

Stable for all step-sizes.

Use LR decomposition

$$\left(\underline{\underline{1}} + \Delta t\, \underline{\underline{C}}\right) = LR$$

or iterative solver.

Ground level: Solve linear system

Introduction of the Brusselator | Note on PDEs

Non-linear chemical-reaction-diffusion partial differential equation in 1D

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$
$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}$$

fixed concentrations of substances A, B
x and t dependent concentrations u, v
diffusion constant α

Turn into ODE by method of lines

discretize all partial derivatives but 1

$$\partial_t u_i = 1 + u_i^2 v_i - 4u_i + \frac{\alpha}{(\Delta x)^2}(u_{i-1} - 2u_i + u_{i+1})$$
$$\partial_t v_i = 3u_i - u_i^2 v_i + \frac{\alpha}{(\Delta x)^2}(v_{i-1} - 2v_i + v_{i+1})$$
$$\text{grid index } i = 0, \dots, N+1$$

write compactly as

$$\underline{y} = \underline{f}(\underline{y}), \qquad \underline{y} = \begin{pmatrix} u_0 \\ \vdots \\ u_{N+1} \\ v_0 \\ \vdots \\ v_{N+1} \end{pmatrix}$$

$\underline{f}: \mathbb{R}^{2(N+2)} \to \mathbb{R}^{2(N+2)}$ from above

Boundary values and textbook solution

Boundary values

$$u_0(t) = u_{N+1}(t) = 1$$
$$v_0(t) = v_{N+1}(t) = 3$$

Initial conditions

$$u_i(0) = 1 + \sin(2\pi x_i)$$
$$v_i(0) = 3, i = 1, \dots, N$$

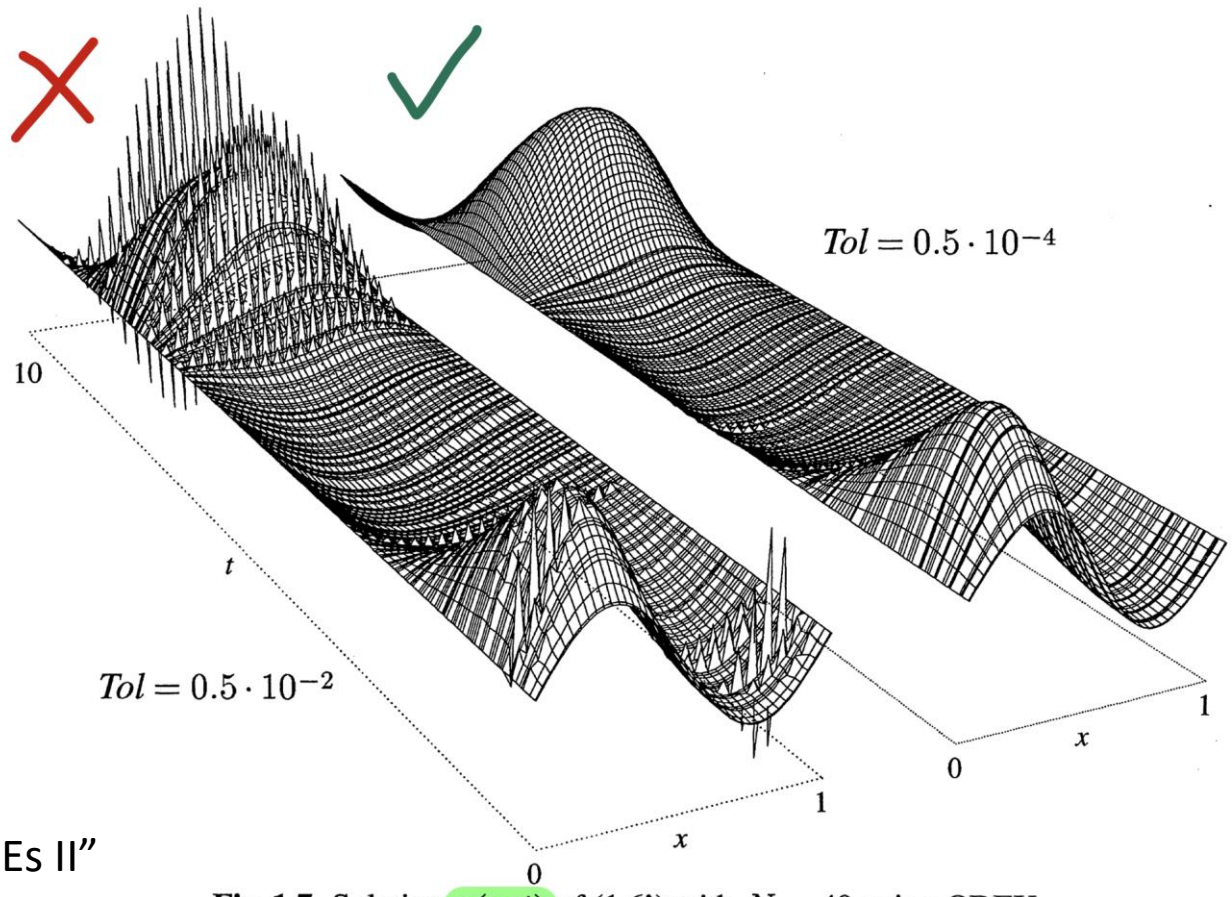
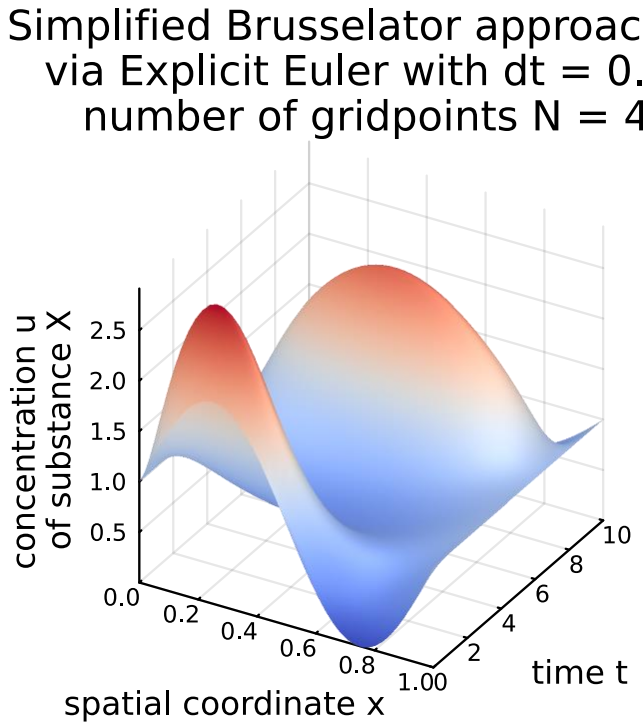


Fig. 1.7. Solution $u(x, t)$ of (1.6') with $N = 40$ using ODEX

Textbook solution from "Solving Stiff ODEs II" by E. Hairer and G. Wanner

Can we use and explicit method? | On Stiffness

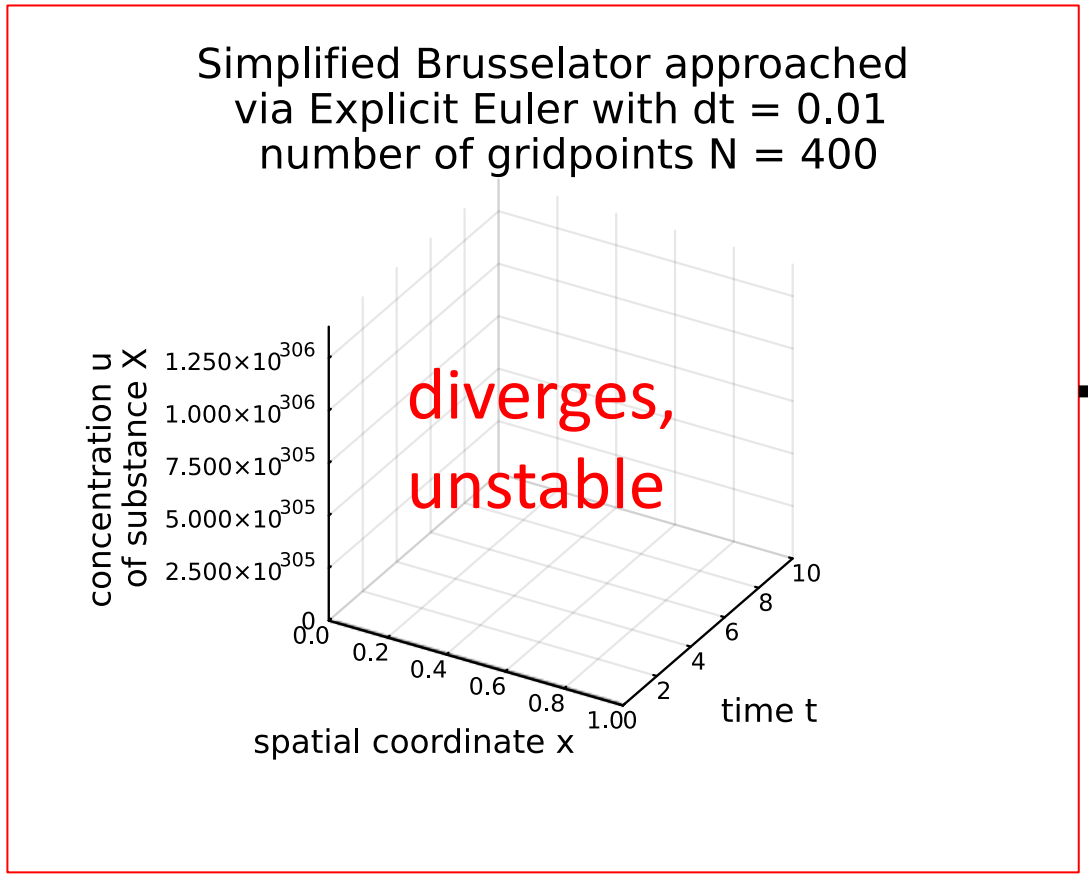
Explicit Euler Method: $\underline{y}^{(n+1)} = \underline{y}^{(n)} + \Delta t\, \underline{f}\left(\underline{y}^{(n)}\right)$



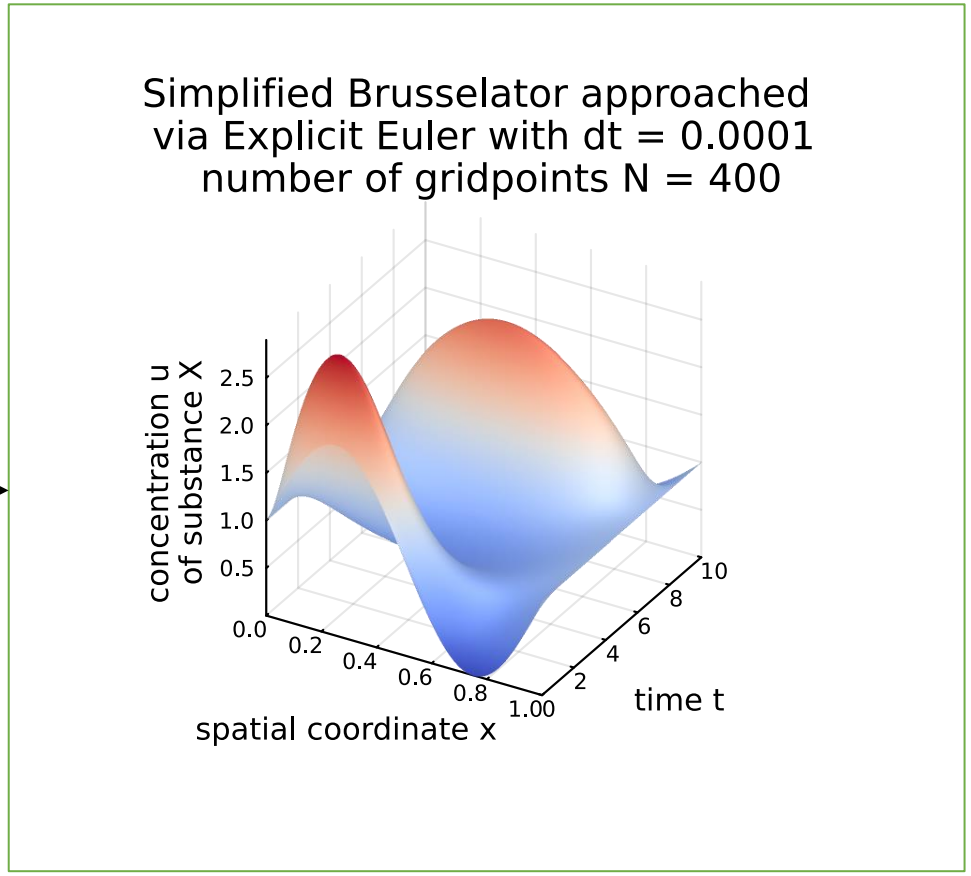
For $N = 40$ explicit Euler gives the correct result for a reasonably large stepsize.

The Occurrence of Stiffness

Increase N to 400

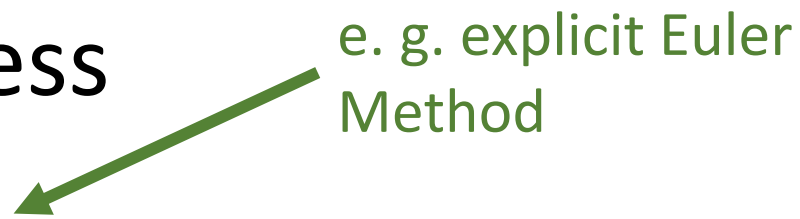


Decrease step-size



Default DifferentialEquations.jl Tsitouras 5/4 Runge-Kutta method: 220047 evaluations of $\underline{f}$

A Practical Definition of Stiffness

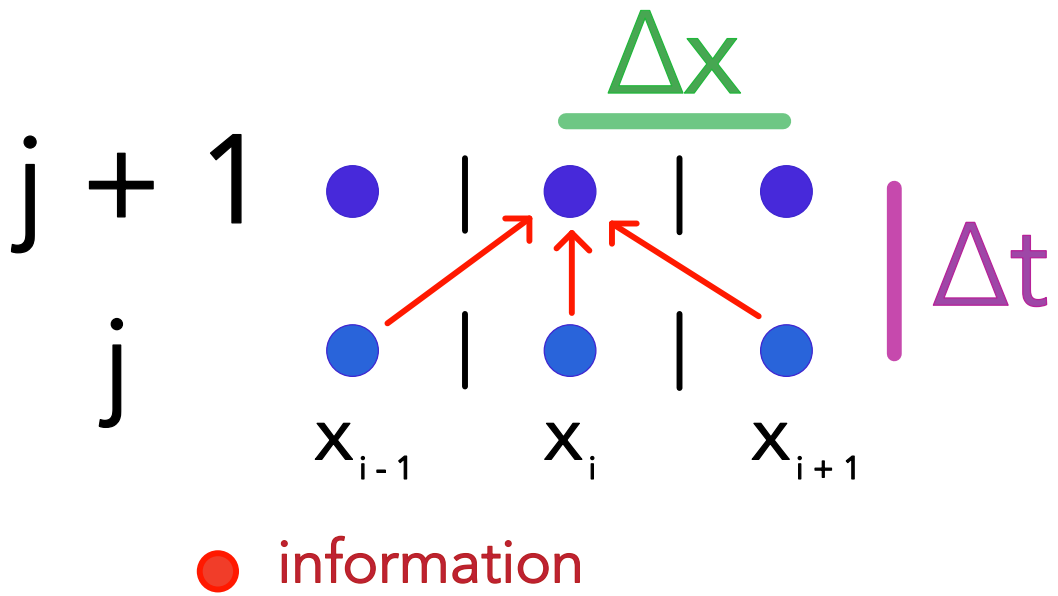


»If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a step length which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.«

J. D. Lambert, “Numerical Methods for ordinary Differential Systems”, chapter 6

Understanding Stiffness in a diffusive context

In every explicit step, only neighboring cells communicate.



Concentration spreads with $\sigma = \sqrt{2\alpha t}$

$$\Delta x > \sigma(\Delta t) = \sqrt{2\alpha \Delta t}$$

diffusion constant α

$$\Delta t < \frac{\Delta x^2}{2\alpha}, \qquad \Delta x = \frac{1}{N+1}$$

number N of gridpoints in $[0,1]$

Problem: Double resolution $\frac{\Delta x}{2}$ → need $\mathcal{O}(N^2)$ more steps to cover the timespan, every step $\mathcal{O}(N)$ more complex → $\mathcal{O}(N^3)$

For small Δx the explicit scheme takes excessively many steps.

Remember the discretized differential equation

$$\partial_t u_i = 1 + u_i^2 v_i - 4u_i + \frac{\alpha}{(\Delta x)^2}(u_{i-1} - 2u_i + u_{i+1})$$

$$\partial_t v_i = 3u_i - u_i^2 v_i + \frac{\alpha}{(\Delta x)^2}(v_{i-1} - 2v_i + v_{i+1})$$

typical stiffness indicator

Coefficients of vastly different magnitude occur for small Δx.

The Implicit Approach | Implicit Euler Method

Implicit Euler Method for Solving Stiff ODEs $\partial_t \underline{y} = \underline{f}(\underline{y})$

$$\text{implicit step } \underline{y}^{(n+1)} = \underline{y}^{(n)} + \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right)$$

Formulate taking **one timestep** as root-finding problem

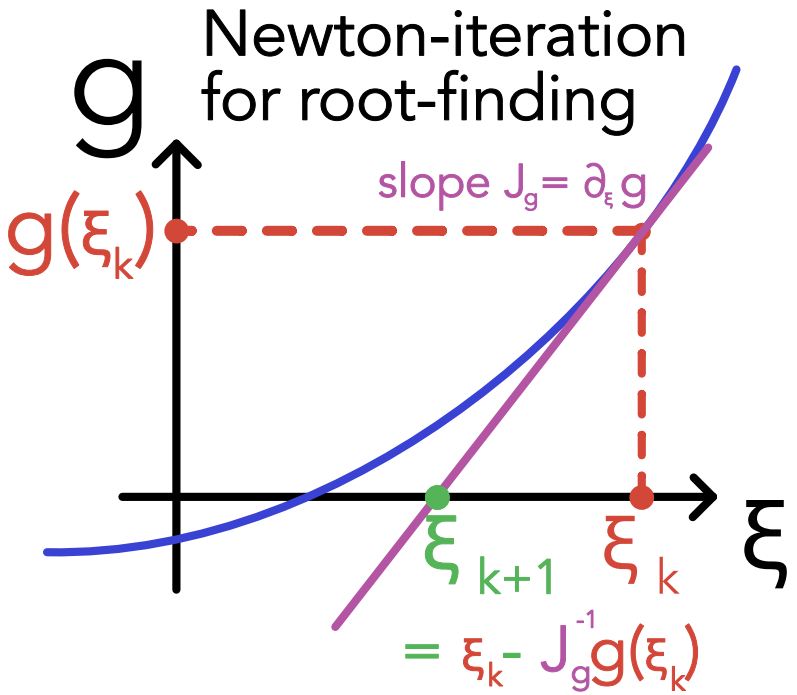
$$\underline{0} = \underline{y}^{(n+1)} - \underline{y}^{(n)} - \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right) =: \underline{g}(\underline{\xi})$$

Solve **one timestep** iteratively using Newton iteration

$$\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_{\underline{g}}^{-1}(\underline{\xi}_k)\, \underline{g}(\underline{\xi}_k), \qquad \underline{\underline{J}}_{\underline{g}} = \underline{\underline{1}} - \gamma(\Delta t)\, \underline{\underline{J}}_{\underline{f}}$$

$$\underline{\xi}_0 = \underline{y}^{(n)}, \qquad \underline{\xi}_m \xrightarrow[m\to\infty]{} \underline{y}^{(n+1)}$$

$$\text{alternatively quasi} - \text{Newton } \underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_{\underline{g}}^{-1}(\underline{\xi}_0)\, \underline{g}(\underline{\xi}_k)$$



In **each step of the Newton-iteration** calculate $\underline{u}_{k+1}$ from linear equation $\underline{b} := \underline{g}(\underline{\xi}_k) = \underline{\underline{J}}_{\underline{g}}(\underline{u}_k - \underline{u}_{k+1}) = \underline{\underline{J}}_{\underline{g}}\underline{a}$

How to solve the linear system $\underline{g}(\underline{\xi}_k) = \underline{\underline{J}}_g \underline{a}$?

Remember: Calculate a Jacobian Vector product (directional derivative)

$$\underline{\underline{J}}_g \underline{v} = (\underline{\nabla} \cdot \underline{g})\, \underline{v}, \qquad \underline{\underline{J}}_g \underline{\hat{e}}_j \text{ calculates } j-\text{th column of Jacobian}$$

to machine precision using one forward mode AD calculation

$$\underline{g}(\underline{\xi} + \epsilon \underline{v}) = \underline{g}(\underline{\xi}) + \epsilon\, \underline{\underline{J}}_g \underline{v}, \qquad \text{dual number} \underline{\xi} + \epsilon \underline{v}, \qquad \epsilon^2 = 0$$

Calculate $\underline{\underline{J}}_g$ efficiently and solve the linear system based on matrix decomposition.

Solve the system only based on Jacobian-vector-products.

Option A for solving the linear system involving the Jacobian

Option B for solving the linear system involving the Jacobian

Option A: Jacobi Calculation and Matrix Decomposition for $\underline{g}(\underline{\xi}_k) = \underline{\underline{J}}_g \underline{a}$

1. Calculate all columns $\underline{\underline{J}}_g \hat{\underline{e}}_j$ of the Jacobian

$$\underline{g}(\underline{\xi} + \epsilon \hat{\underline{e}}_j) = \underline{g}(\underline{\xi}) + \epsilon \, \underline{\underline{J}}_g \hat{\underline{e}}_j$$

$$j = 1, \dots, 2(N+2)$$

$$\rightarrow \mathcal{O}(N) \text{ evaluations of } \underline{f}$$

$$\rightarrow \mathcal{O}(N^2) \text{ complexity}$$

in Newton iteration in every iterative step

$$\underline{\xi} = \underline{\xi}_k$$

in Quasi-Newton once per timestep

$$\underline{\xi} = \underline{y}_n$$

2. Do matrix decomposition, e. g.

$$\underline{\underline{P}} \, \underline{\underline{J}}_g = \underline{\underline{L}} \underline{\underline{R}} \quad (\text{in } \mathcal{O}(N^3))$$

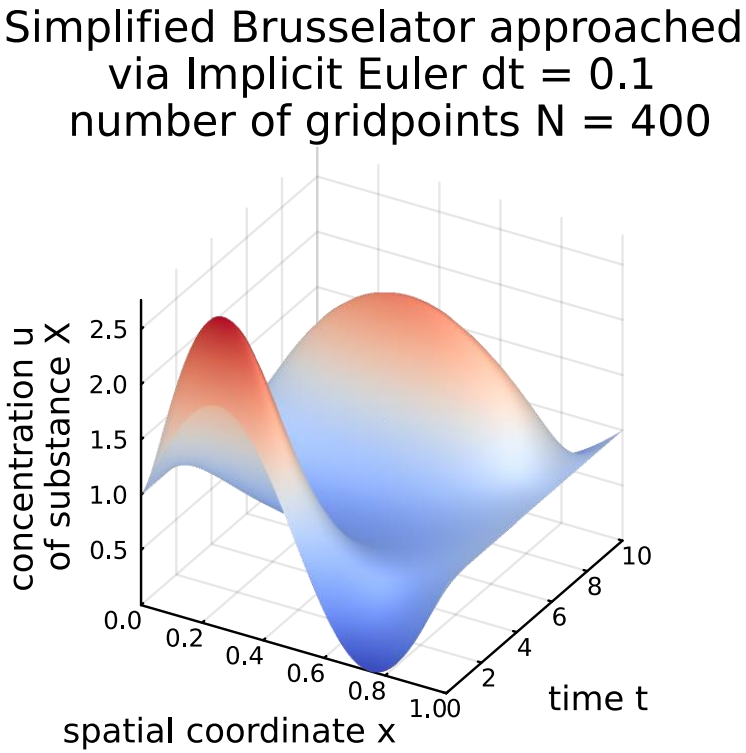
$$\underline{\underline{J}}_g \underline{a} = \underline{g} \Leftrightarrow \underline{\underline{L}} \, \underline{\underline{R}} \, \underline{a} = \underline{g}$$

Q: Why can't we simply invert $\underline{\underline{J}}_g$?

3. Solve for $\underline{a}$

1. solve $\underline{\underline{L}} \, \underline{y} = \underline{g}$ for $\underline{y}$ (forward substituttion)
2. solve $\underline{\underline{R}} \, \underline{a} = \underline{y}$ for $\underline{y}$ by forward (backward substitution)

Lower – left matrix $\underline{\underline{L}}$, Upper – right matrix $\underline{\underline{R}}$, permutation matrix $\underline{\underline{P}}$



Looks good and that at $\Delta t = 0.1$

Efficiency of our simple implementations

Explicit Euler, $dt = 0.0001$	Implicit Euler; Newton, $dt = 0.1$	Implicit Euler; Quasi-Newton, $dt = 0.1$
1.848 s (1447185 allocations: 5.95 GiB)	6.737 s (396055 allocations: 11.48 GiB)	3.144 s (197980 allocations: 5.57 GiB)

Problem I: Calculation of the Jacobian column-by-column is very expensive.

Problem II: The matrix factorization with $\mathcal{O}(\mathrm{N}^3)$ is also expensive but a highly optimized problem (→ LAPACK)

Comparison between Newton and Quasi-Newton

```
function newtons_method(f!, xi0, dt, tol, max_iter)
    """
    The root of g_zero (and therefore the next step
    in time in the implicit Euler scheme) is found
    using Newton's method, which is implemented here.
    """
    xi = copy(xi0)
    for i in 1:max_iter
        ###################################################
        # Here the LU decomposition is performed
        # the full Jacobian is constructed in every step
        luJ = lu(Jg(f!, xi, dt)) # Jg constructs
        #                          the Jacobian
        ###################################################
        a = luJ \ g_zero(f!, xi, xi0, dt)
        xi .= xi .- a
        if norm(g_zero(f!, xi, xi0, dt)) < tol
            return xi
        end
    end
    return xi
end
```

quadratic convergence $q = 2$ in $\lim_{n\to\infty} \frac{\left|\xi_{n+1} - y^{(n+1)}\right|}{\left|\xi_n - y^{(n+1)}\right|^q} = \mu$

(rate of convergence μ), but $\mathcal{O}(N^3)$ LU per iteration step

```
function quasi_newtons_method(f!, xi0, dt, tol, max_iter)
    """
    The root of g_zero (and therefore the next
    step in time in the implicit Euler scheme)
    is found using Newton's method,
    which is implemented here.
    """
    xi = copy(xi0)
    ###################################################
    # Here a matrix LU decomposition is performed once
    # per implicit Euler timestep
    luJ = factorize(Jg(f!, xi, dt))
    ###################################################
    for i in 1:max_iter
        a = luJ \ g_zero(f!, xi, xi0, dt)
        xi .= xi .- a
        if norm(g_zero(f!, xi, xi0, dt)) < tol
            return xi
        end
    end
    return xi
end
```

No quadratic convergence, but $\mathcal{O}(N^3)$ LU only once per implicit timestep

Intermezzo: Non-Overlapping differentiations

Consider e. g.

Twiggling at x_1 and x_2 has no cross effects thus separates additively

$$\underline{f}(\underline{x}) = \begin{pmatrix} x_1 + x_3 \\ x_2 x_3 \\ x_1 \end{pmatrix}, \qquad \underline{\underline{J}}_{\underline{f}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{pmatrix}, \qquad \text{sparsity } \underline{\underline{S}}_{\underline{f}} = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

where

$$\frac{\underline{f}(\underline{x} + h\hat{\underline{e}}_1) - \underline{f}(\underline{x})}{h} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \qquad \frac{\underline{f}(\underline{x} + h\hat{\underline{e}}_2) - \underline{f}(\underline{x})}{h} = \begin{pmatrix} 0 \\ x_3 \\ 0 \end{pmatrix}$$

and

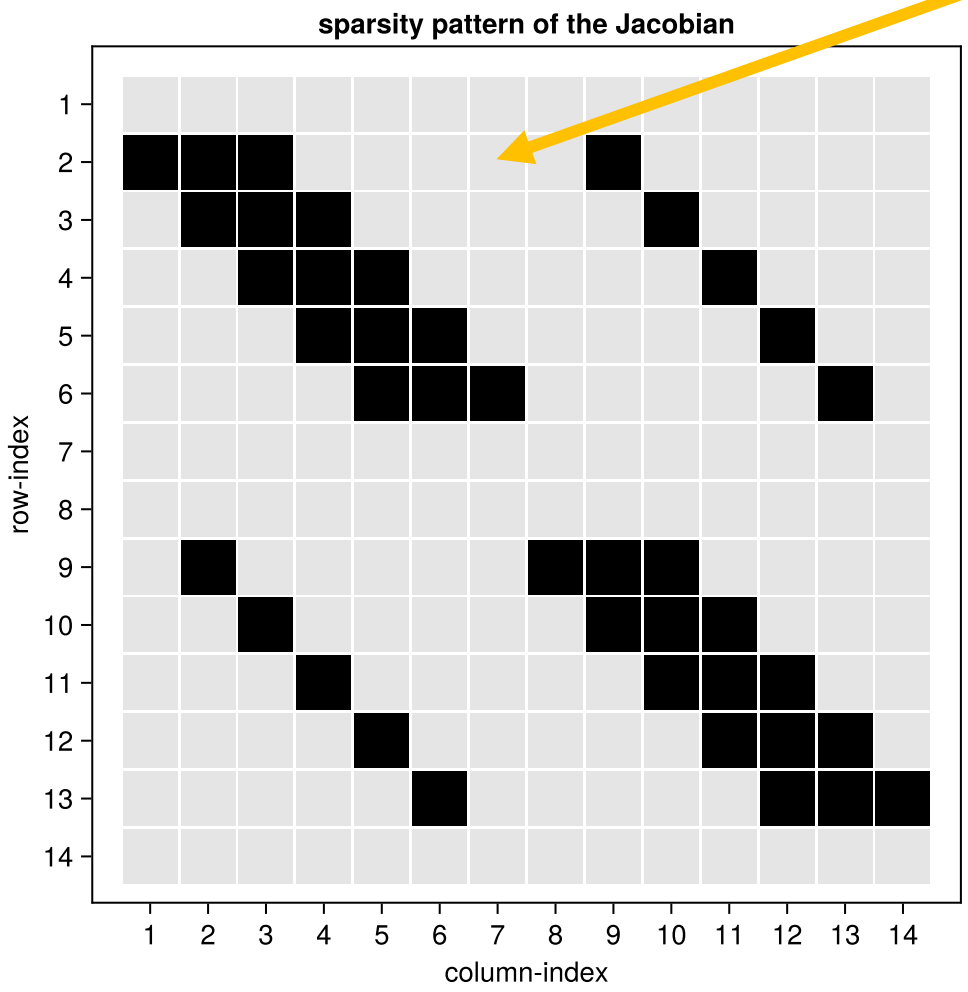
$$\frac{\underline{f}(\underline{x} + h\hat{\underline{e}}_1 + h\hat{\underline{e}}_2) - \underline{f}(\underline{x})}{h} = \begin{pmatrix} 1 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ x_3 \\ 0 \end{pmatrix} \xrightarrow[\text{using } \underline{\underline{S}}_{\underline{f}}]{} \underline{\underline{J}}_{\underline{f}} = \begin{pmatrix} 1 & 0 & \vdots \\ 0 & x_3 & \vdots \\ 1 & 0 & \vdots \end{pmatrix}$$

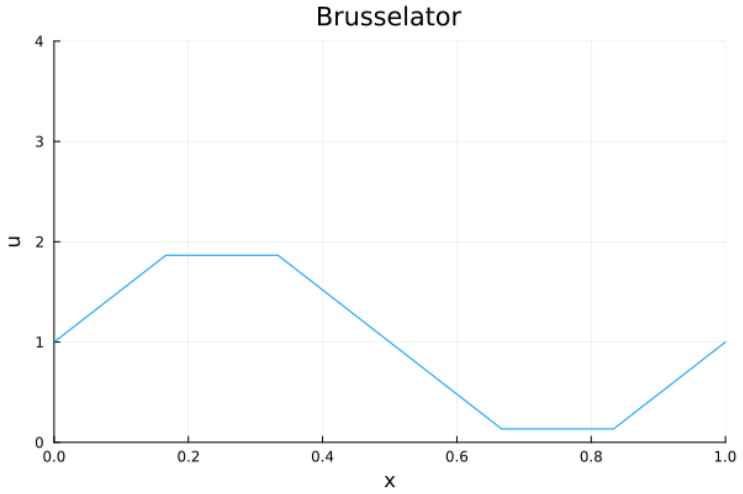
Changing x_2 neither effects f_1 nor f_3

Sparsity of the Jacobian

Examples for $N = 5$ for explainability.

$$\frac{\partial g_2}{\partial \xi_7} = 0$$



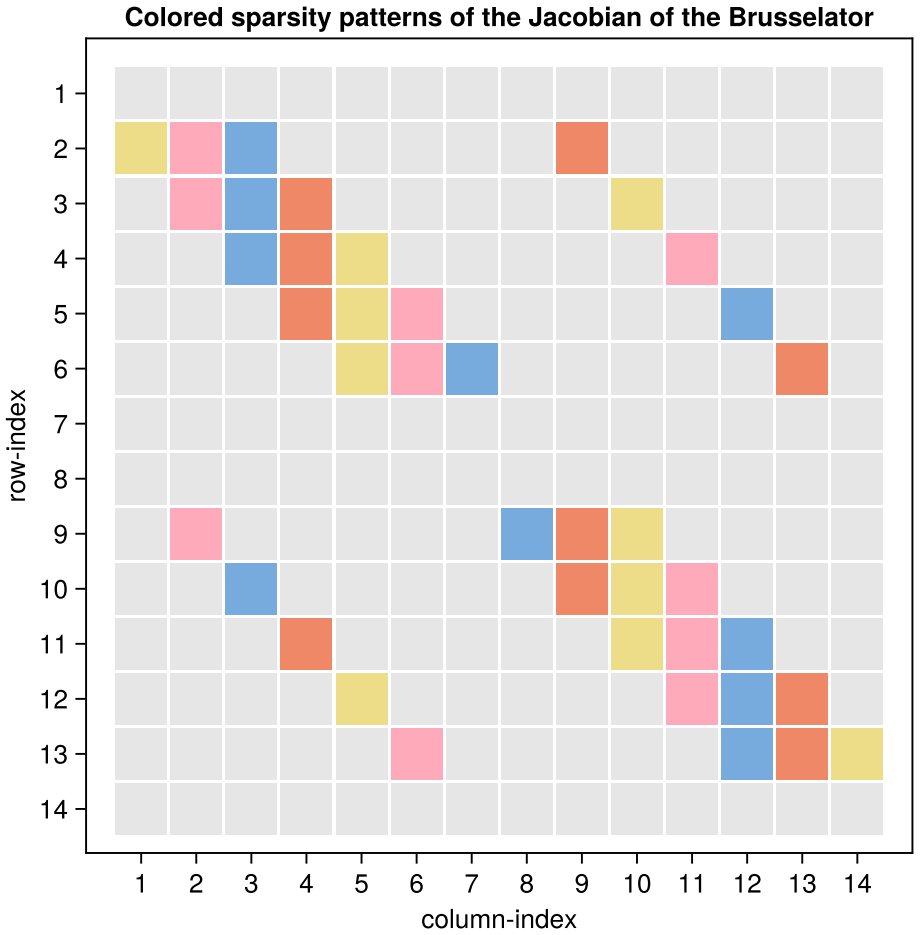


$$\mathrm{N} = 400: \frac{\#\ \text{non} - \text{zero elements}}{\#\ \text{all elements}} \approx 6 \cdot 10^{-5}$$

Q: How can we use this sparsity?

Idea: Calculate columns of the Jacobian in non-overlapping groups

Calculate e. g. the blue group in one forward pass $\underline{\underline{J_g}}(\underline{\hat{e}}_3 + \underline{\hat{e}}_7 + \underline{\hat{e}}_8 + \underline{\hat{e}}_{12}) =: \underline{JVP}$



$$= \begin{pmatrix} 0 \\ (JVP)_2 \\ (JVP)_3 \\ (JVP)_4 \\ (JVP)_5 \\ (JVP)_6 \\ 0 \\ 0 \\ (JVP)_9 \\ (JVP)_{10} \\ (JVP)_{11} \\ (JVP)_{12} \\ (JVP)_{13} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ (JVP)_2 \\ (JVP)_3 \\ (JVP)_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ (JVP)_{10} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ (JVP)_6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ (JVP)_9 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ (JVP)_5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ (JVP)_{11} \\ (JVP)_{12} \\ (JVP)_{13} \\ 0 \end{pmatrix}$$

Only 4 forward passes!

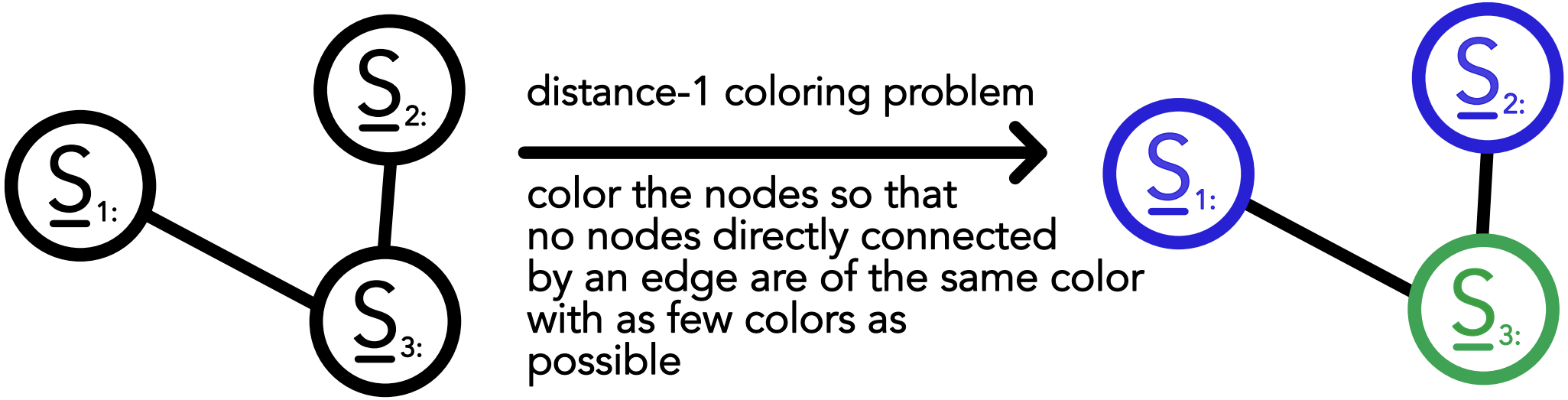
But how to find the grouping (coloring)?

Well known distance-1 graph coloring problem in disguise:

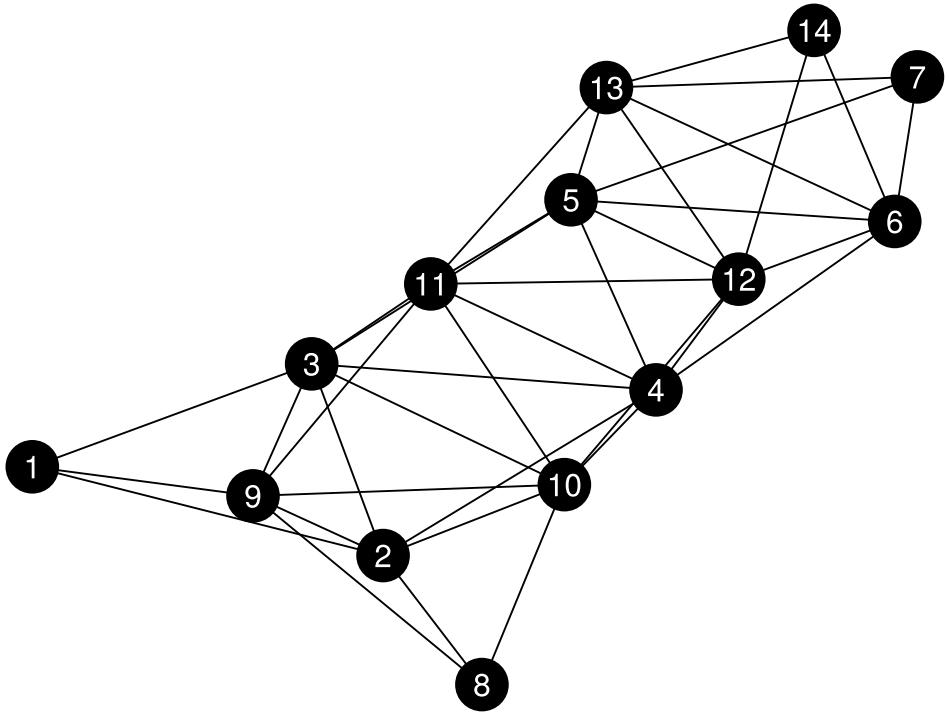
Nodes: Columns of the Sparsity of the Jacobian

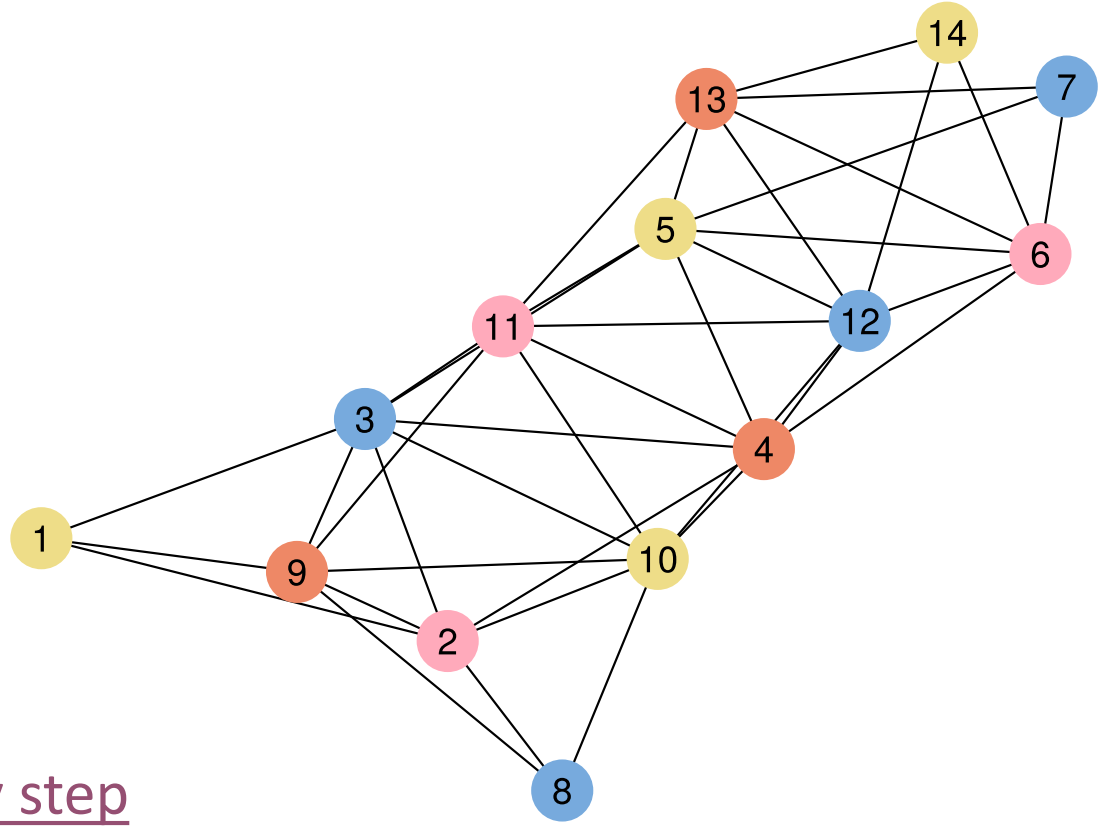
Edges: between overlapping columns

Simple example



NP-complete → use approximate Greedy algorithm



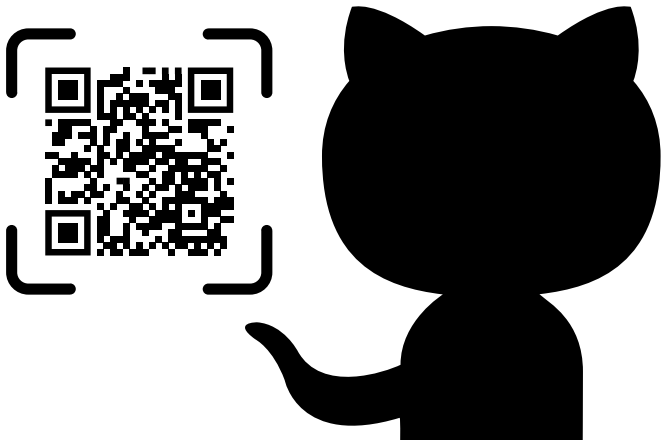


Greedy algorithm we used and step by step application in appendix :)

Results

Implicit Euler; Quasi-Newton, $\mathrm{dt} = 0.1$	Implicit Euler; Quasi-Newton, coloring, $\mathrm{dt} = 0.1$	Implicit Euler; Quasi-Newton, coloring, **sparse factorization** $\mathrm{dt} = 0.1$
3.144 s (197980 allocations: 5.57 GiB)	877.994 ms (17404 allocations: 1.51 GiB)	88.977 ms (25192 allocations: 200.19 MiB)

Overview on Solving Stiff ODEs



Implicit Euler Method for Solving Stiff ODEs $\partial_t \underline{y} = \underline{f}(\underline{y})$

$$\text{implicit step } \underline{y}^{(n+1)} = \underline{y}^{(n)} + \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right)$$

non-linear system

Formulate into root-finding problem

$$\underline{0} = \underline{y}^{(n+1)} - \underline{y}^{(n)} - \Delta t\, \underline{f}\left(\underline{y}^{(n+1)}\right) =: \underline{g}(\underline{\xi})$$

and find the solution iteratively using Newton iteration

$$\underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_g^{-1}(\underline{\xi}_k)\, \underline{g}(\underline{\xi}_k), \qquad \underline{\underline{J}}_g = \underline{\underline{1}} - \gamma(\Delta t)\, \underline{\underline{J}}_f$$

$$\text{alternatively quasi} - \text{Newton } \underline{\xi}_{k+1} = \underline{\xi}_k - \underline{\underline{J}}_g^{-1}(\underline{\xi}_0)\, \underline{g}(\underline{\xi}_k)$$

$$\rightarrow \text{linear equation } \underline{b} := \underline{g}(\underline{\xi}_k) = \underline{\underline{J}}_g(\underline{\xi}_k - \underline{\xi}_{k+1}) = \underline{\underline{J}}_g \underline{a}$$

sparse Jacobian

Calculate sparse Jacobian using sparse Differentiation (Autodiff with smart direction based on distance-1 graph coloring insights)

Solve linear equation using matrix decomposition $\underline{\underline{J}}_g = \underline{\underline{L}}\, \underline{\underline{R}}$. In Quasi-Newton reuse expensive $(\mathcal{O}(N^3))$ decomposition.

Option A for solving the linear system involving the Jacobian

Jacobian too large to store

Solve $\underline{\underline{J}}_g \underline{a} = \underline{b}$ without calculating $\underline{\underline{J}}_g$

Calculate only Jacobian-Vector products using Autodiff and solve the Linear equation using Krylov-Subspace methods.

Option B for solving the linear system involving the Jacobian

linear system

$$\partial_t \underline{y} = -\underline{\underline{C}}\, \underline{y}$$

Every step corresponds to solving a linear equation

$$\left(\underline{\underline{1}} + \Delta t\, \underline{\underline{C}}\right) \underline{y}_{n+1} = \underline{y}_n$$

Stable for all step-sizes.

Use LR decomposition

$$\left(\underline{\underline{1}} + \Delta t\, \underline{\underline{C}}\right) = LR$$

or iterative solver.

Ground level: Solve linear system

Why should I care? – my library does that all automatically

Give the solver more information on your problem.

DifferentialEquations.jl solve method **without sparsity pattern given**	DifferentialEquations.jl solve method **with sparsity pattern given**
251.900 ms (63311 allocations: 211.75 MiB)	14.961 ms (33879 allocations: 27.41 MiB)

Further improvements

- General
  - Adaptive timestep control in combination with higher order methods like Rosenbrock or Semi-Implicit Extrapolation

- For Jacobi-free variant
  - Preconditioning → decrease the condition number of the matrix being solved → faster convergence of GMRES

Breakdown of Classical Methods & Outlook

- Problem of strong turbulence: Reynolds number limited for turbulent flow simulations (unreasonably small grid would be necessary) → Idea: represent small scale phenomena based on NN [e. g. Kochkov et al is a nice read]
- Problem of different scales: Different scales can be very problematic (although there are adaptive methods, mesh free methods, tree algorithms etc.)
- Curse of dimensionality for high-dimensional PDEs (e. g. Schrödinger in Physics, Black-Scholes in finance): The computational cost for solving them goes up exponentially with the dimensionality. [Han et al, 2018]
- Need for inverse modeling
- How to blend differential equations with the vast data sets?
- …

TODO

Key insights

For integrations over long timespan where conservation properties should be obeyed, use Symplectic Integrators.

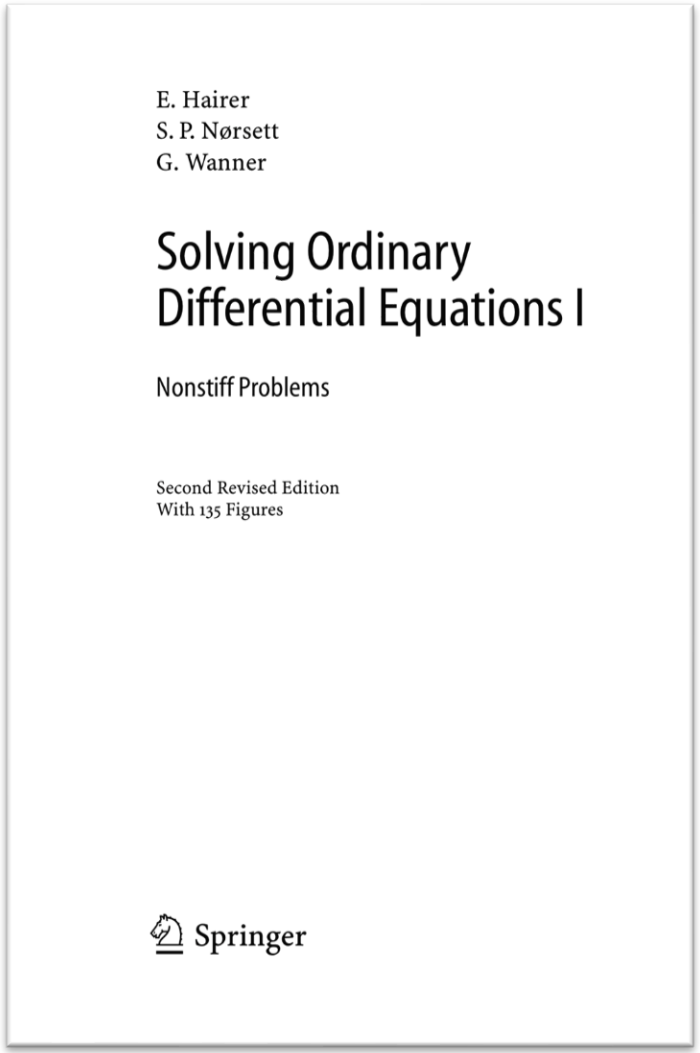
Stiffness is an efficiency problem.

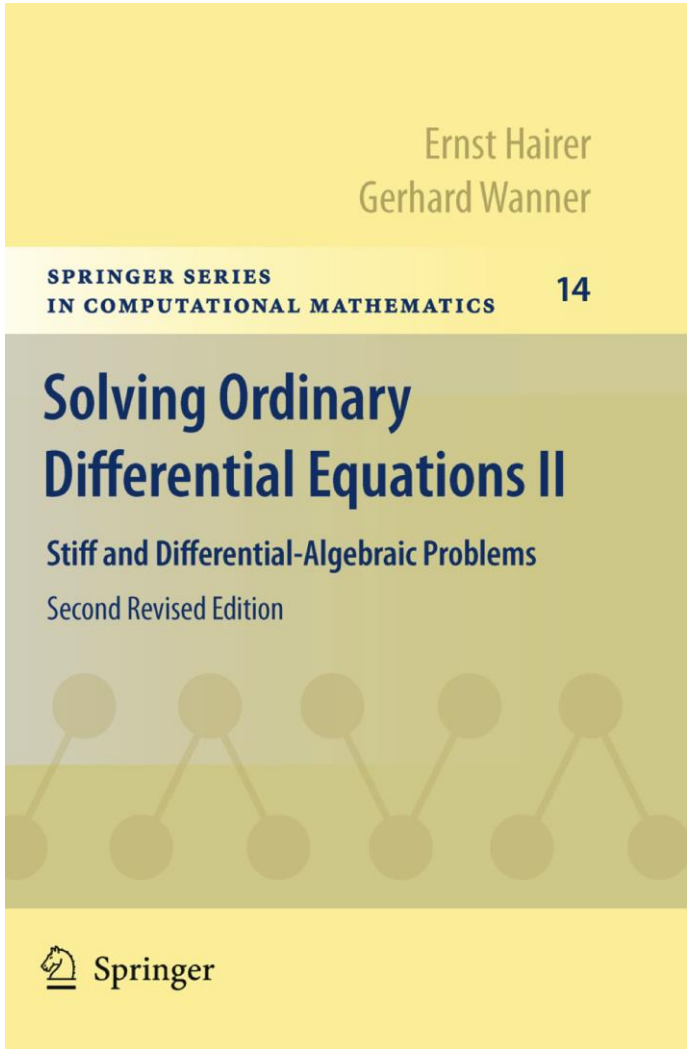
Stiff problems require special approaches where you need to mind the sparsity of jour Jacobian.

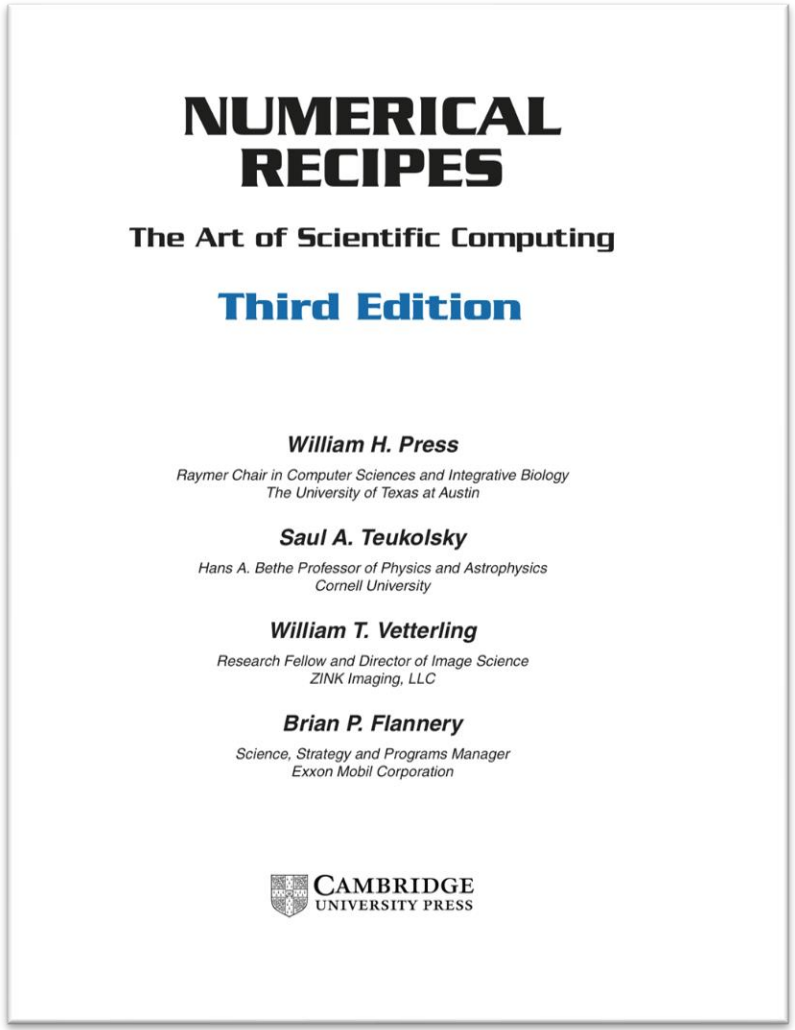
References

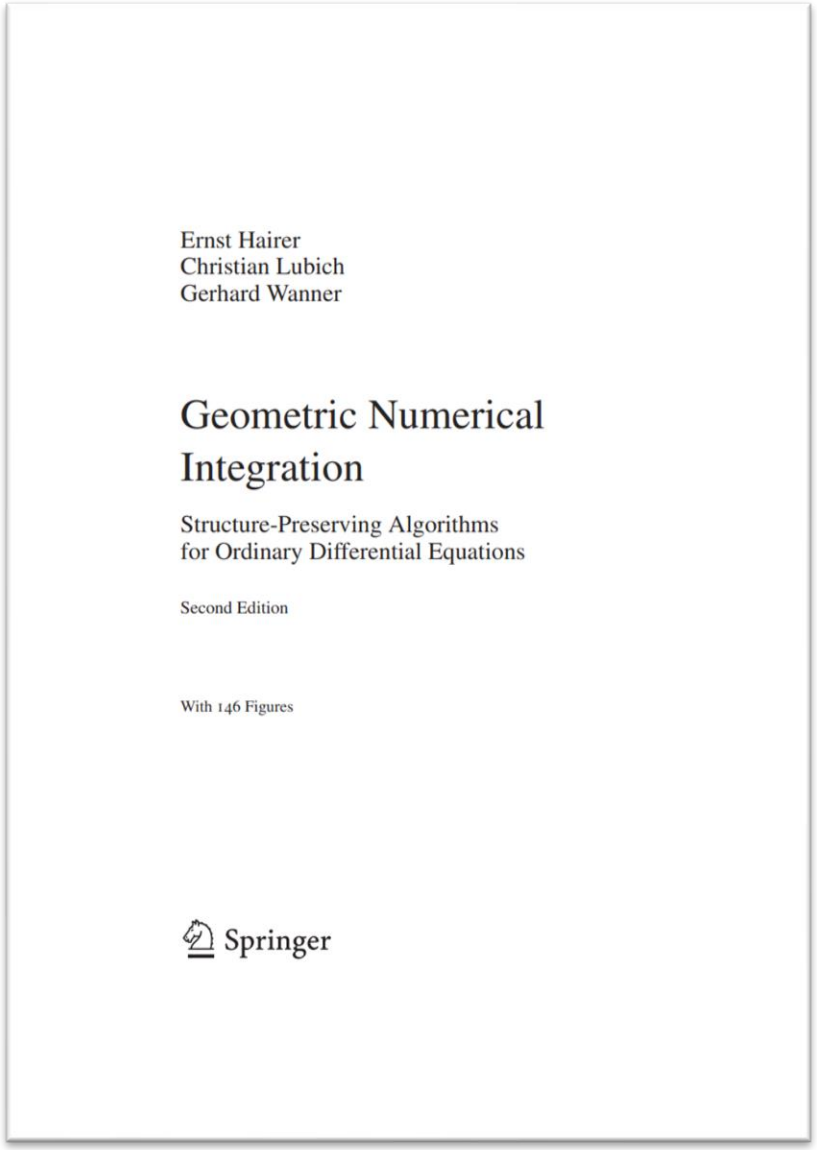
General Literature used

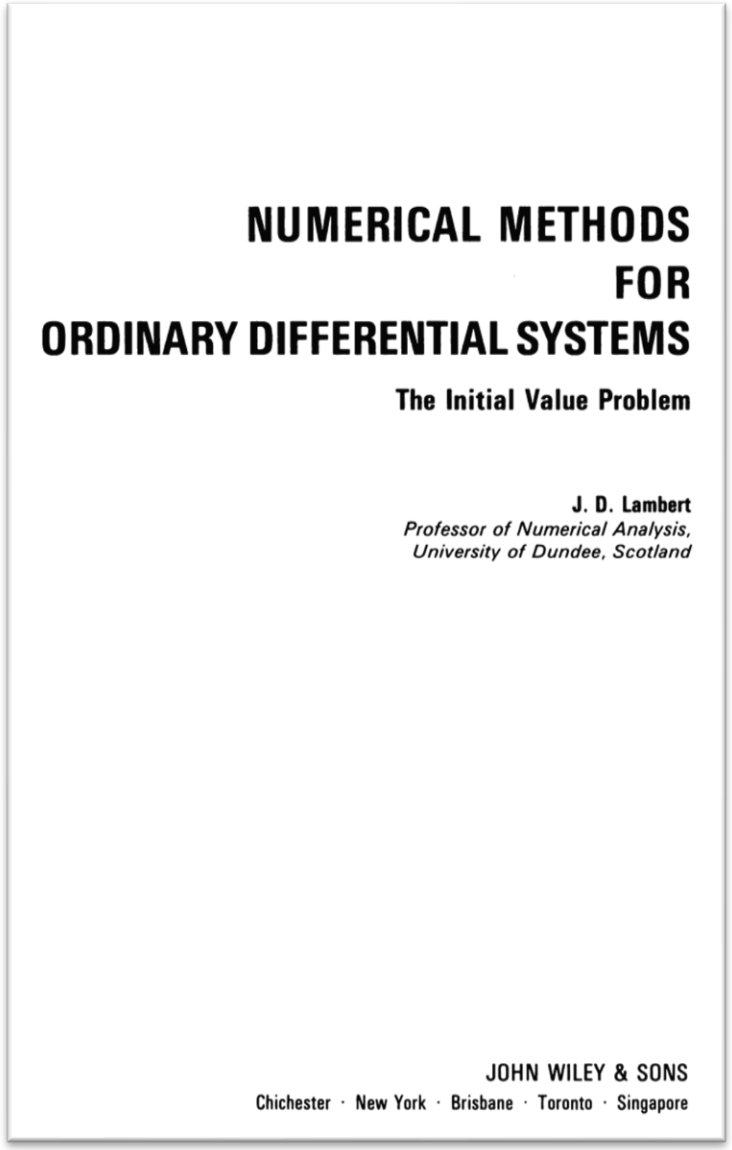
(see criticism)

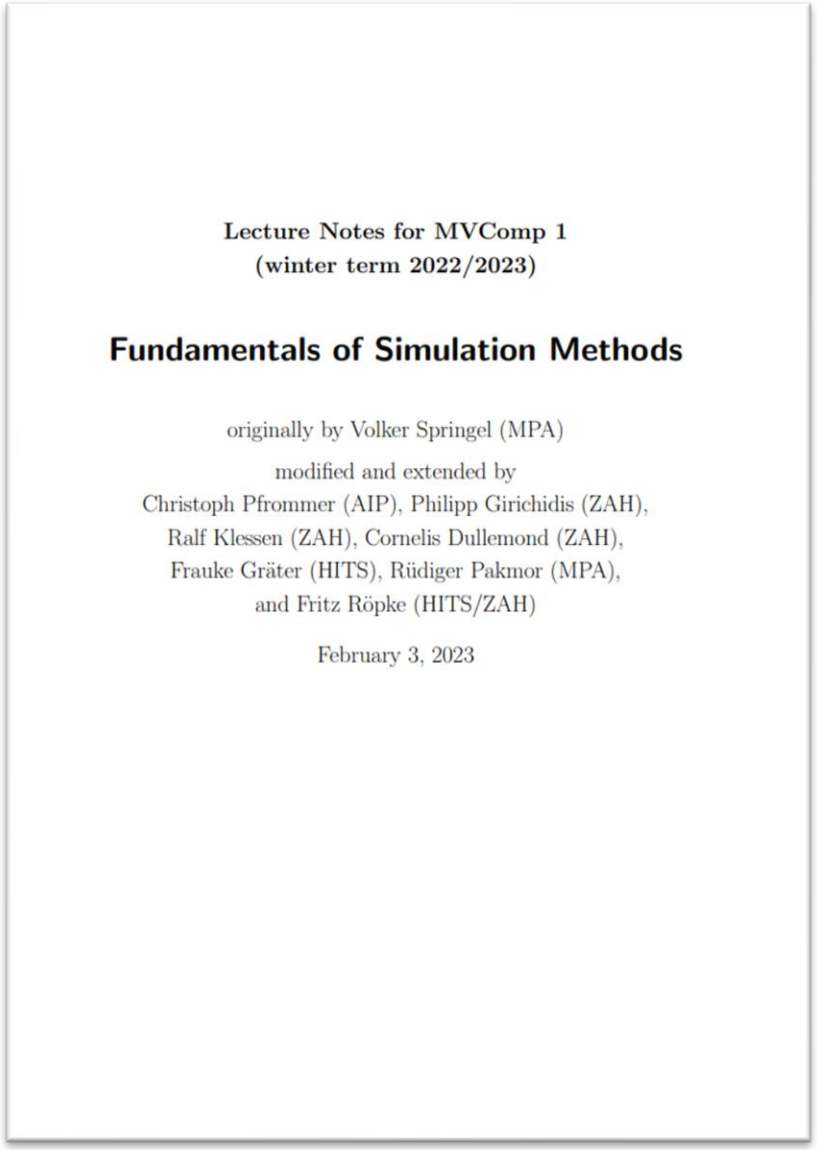







Ernst Hairer
Christian Lubich
Gerhard Wanner
Geometric Numerical Integration
Structure-Preserving Algorithms for Ordinary Differential Equations
Second Edition
With 146 Figures
Springer


NUMERICAL METHODS FOR ORDINARY DIFFERENTIAL SYSTEMS
The Initial Value Problem
J. D. Lambert
Professor of Numerical Analysis, University of Dundee, Scotland
JOHN WILEY & SONS
Chichester · New York · Brisbane · Toronto · Singapore


Lecture Notes for MVComp 1
(winter term 2022/2023)
Fundamentals of Simulation Methods
originally by Volker Springel (MPA)
modified and extended by
Christoph Pfrommer (AIP), Philipp Girichidis (ZAH),
Ralf Klessen (ZAH), Cornelis Dullemond (ZAH),
Frauke Gräter (HITS), Rüdiger Pakmor (MPA),
and Fritz Röpke (HITS/ZAH)
February 3, 2023

Main general resource

"Parallel Computing and Scientific Machine Learning (SciML): Methods and Applications", mainly chapter 9., lecture notes by Chris Rackaukas

Differential equations in Julia

Documentation of the DifferentialEquations.jl package

Papers

- Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34), 8505–8510. doi:10.1073/pnas.1718942115
- Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., & Hoyer, S. (2021). Machine learning–accelerated computational fluid dynamics. In Proceedings of the National Academy of Sciences (Vol. 118, Issue 21). Proceedings of the National Academy of Sciences. https://doi.org/10.1073/pnas.2101784118

Appendix

Option B: “Jacobian-free” method

Problem of option A: What if the Jacobian is too large to store?

Q: How to solve a linear system

$$\underline{\underline{A}}\underline{x} = \underline{b}$$

$$\text{in our case } \underline{\underline{J_g}}\underline{a} = \underline{g}$$

only based on Matrix (Jacobian)-Vector products?

Idea 1: Gradient descent on the quadratic form

$$\underline{\underline{J_g}}\underline{a} = \underline{g} \Leftrightarrow G(\underline{a}) \doteq \frac{1}{2}\underline{a}^T\underline{\underline{J_g}}\underline{a} - \underline{a}^T\underline{g} \text{ is minimal}$$

for $\underline{\underline{A}} \in \mathbb{R}^{n\times n}$ symmetric and positiv definite,

so $\underline{a}^T\underline{\underline{J_g}}\underline{a} > 0 \; \forall \; \underline{a} \in \mathbb{K}^n \setminus \{0\}$

$\rightarrow$ unique minimizer

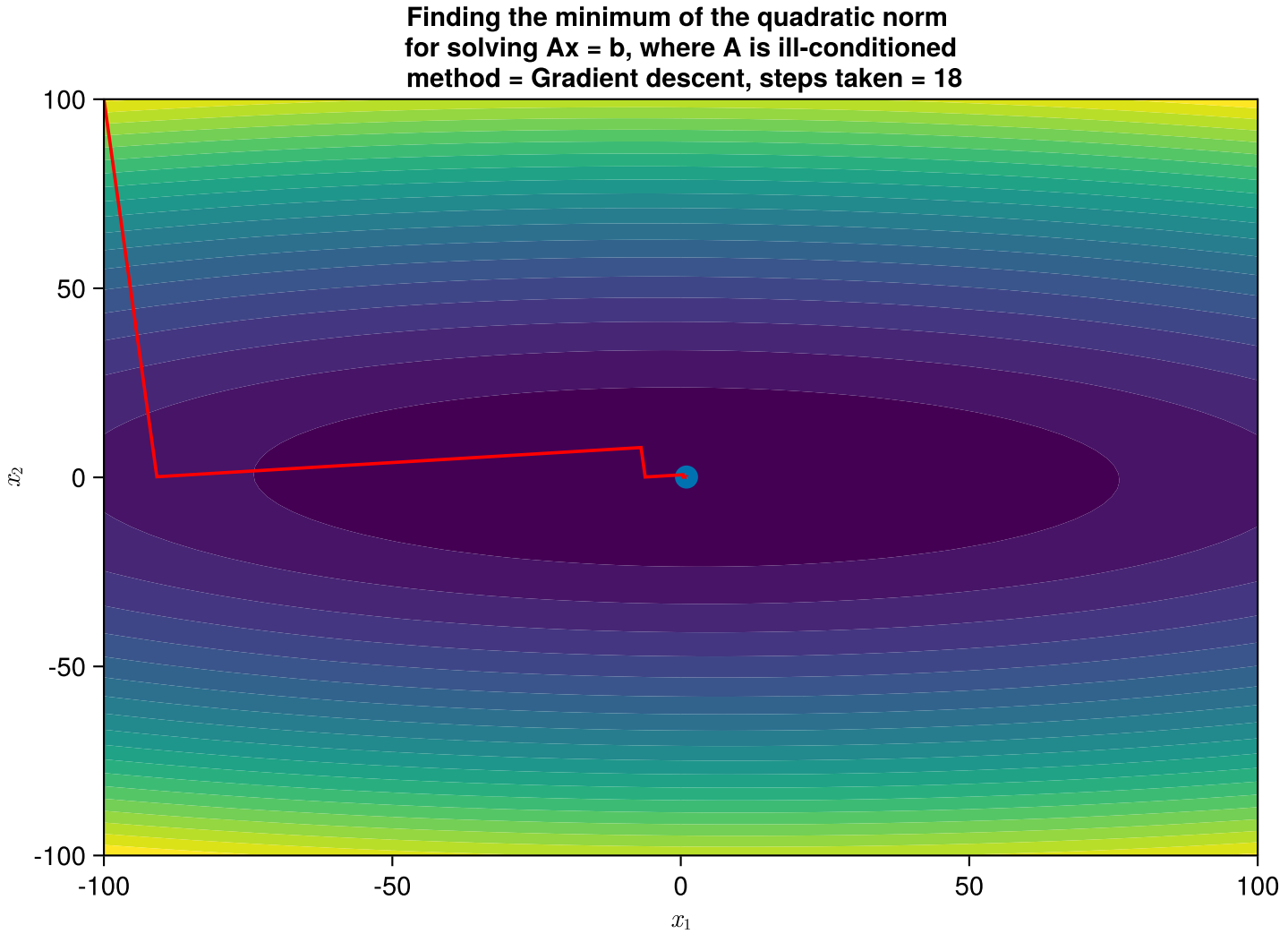
Gradient descent

$$\underline{a}_{k+1} = \underline{a}_k + \alpha(\underline{a}_k)\,\underline{r}_k$$

$$\text{residual } \underline{r}_k = \underline{g} - \underline{\underline{J_g}}\underline{a}_k$$

$$\text{with smart step} - \text{size } \alpha$$

Problem of gradient descent: Non-optimal sequence of search directions.



Consider the steps in Gradient descent for $\alpha = 1, \underline{a}_0 = \underline{0}, \underline{a}_1 = \underline{g}$

$$\underline{a}_2 = \underline{a}_1 + \left(\underline{g} - \underline{\underline{J}}_g \underline{a}_1\right) = 2\underline{g} - \underline{\underline{J}}_g \underline{g}$$

$$\underline{a}_3 = \underline{a}_2 + \left(\underline{g} - \underline{\underline{J}}_g \underline{a}_2\right) = 3\underline{g} - 3\underline{\underline{J}}_g \underline{g} + \underline{\underline{J}}_g^2 \underline{g}$$

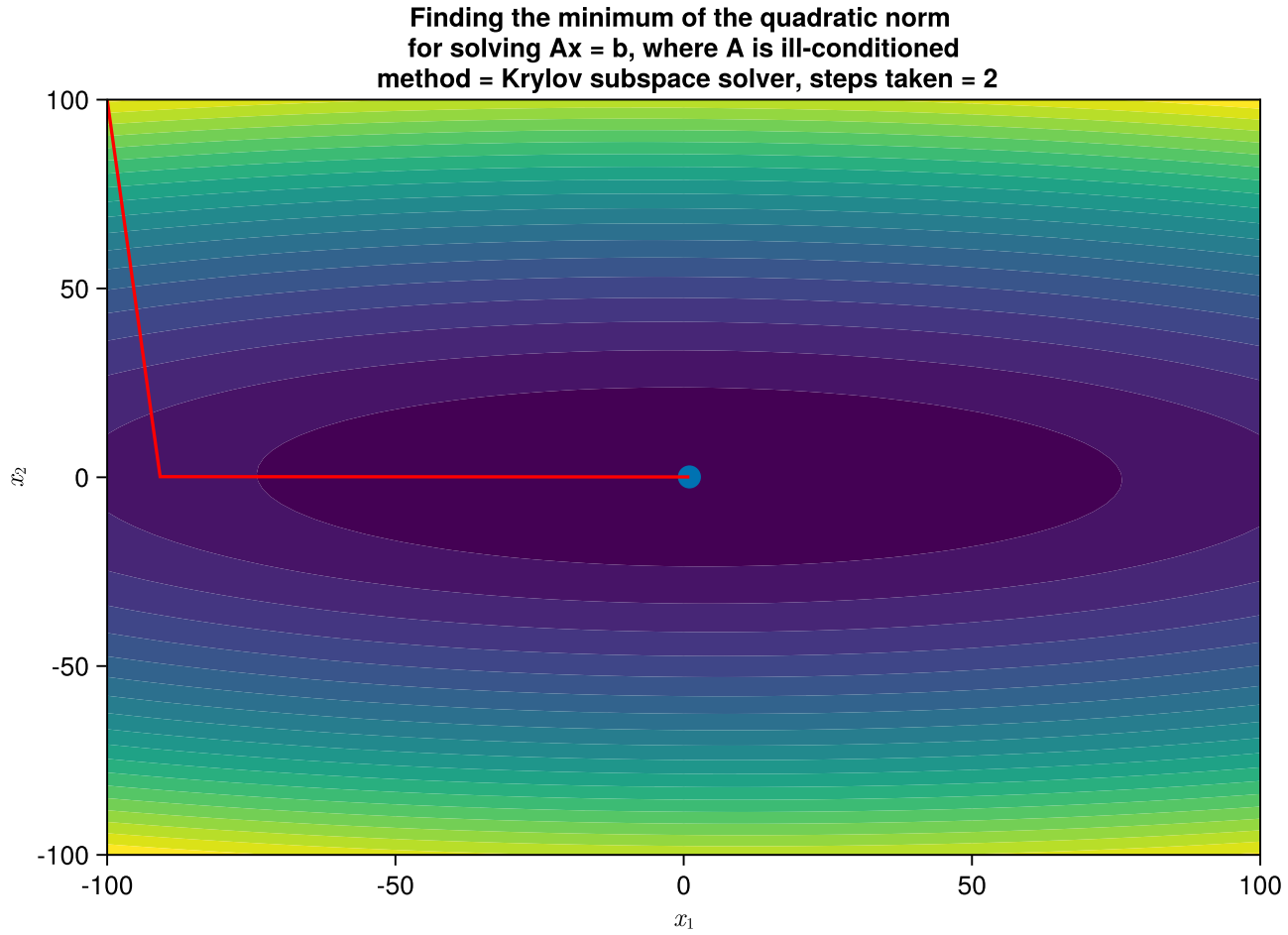
$$\vdots$$

Krylov subspace $\mathcal{K}_r := \mathcal{K}_r\left(\underline{\underline{J}}_g, \underline{g}\right) := \text{span}\left(\underline{b}, \underline{\underline{J}}_g \underline{g}, \underline{\underline{J}}_g^2 \underline{g}, \dots, \underline{\underline{J}}_g^{r-1} \underline{g}\right)$

$\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \cdots$, $\mathcal{K}_n$ spans $\mathbb{R}^n$ if $\underline{\underline{J}}_g$ is invertible

Krylov subspace can be easily constructed! → Are there better directions from $\mathcal{K}_r$?

Teaser: Conjugate Gradient method



Number of steps at most size of linear system.

Krylov subspace methods

Choose $\underline{a}_{k+1}$ so that residual $\underline{r}_{k+1} = \underline{g} - \underline{\underline{J}}_g \underline{a}_{k+1}$

a. is orthogonal to $\mathcal{K}_{k+1}$ → Conjugate Gradient method minimizing the quadratic form
b. has minimum norm for $\underline{a}_k \in \mathcal{K}_{k+1}$ → GMRES

works generally for $\underline{\underline{J}}_g$ invertible

Conjugate Gradient method only works if $\underline{\underline{J}}_g$ is symmetric positive definite → use GMRES for Brusselator problem

Proof: Jacobian-Vector products are directional derivatives

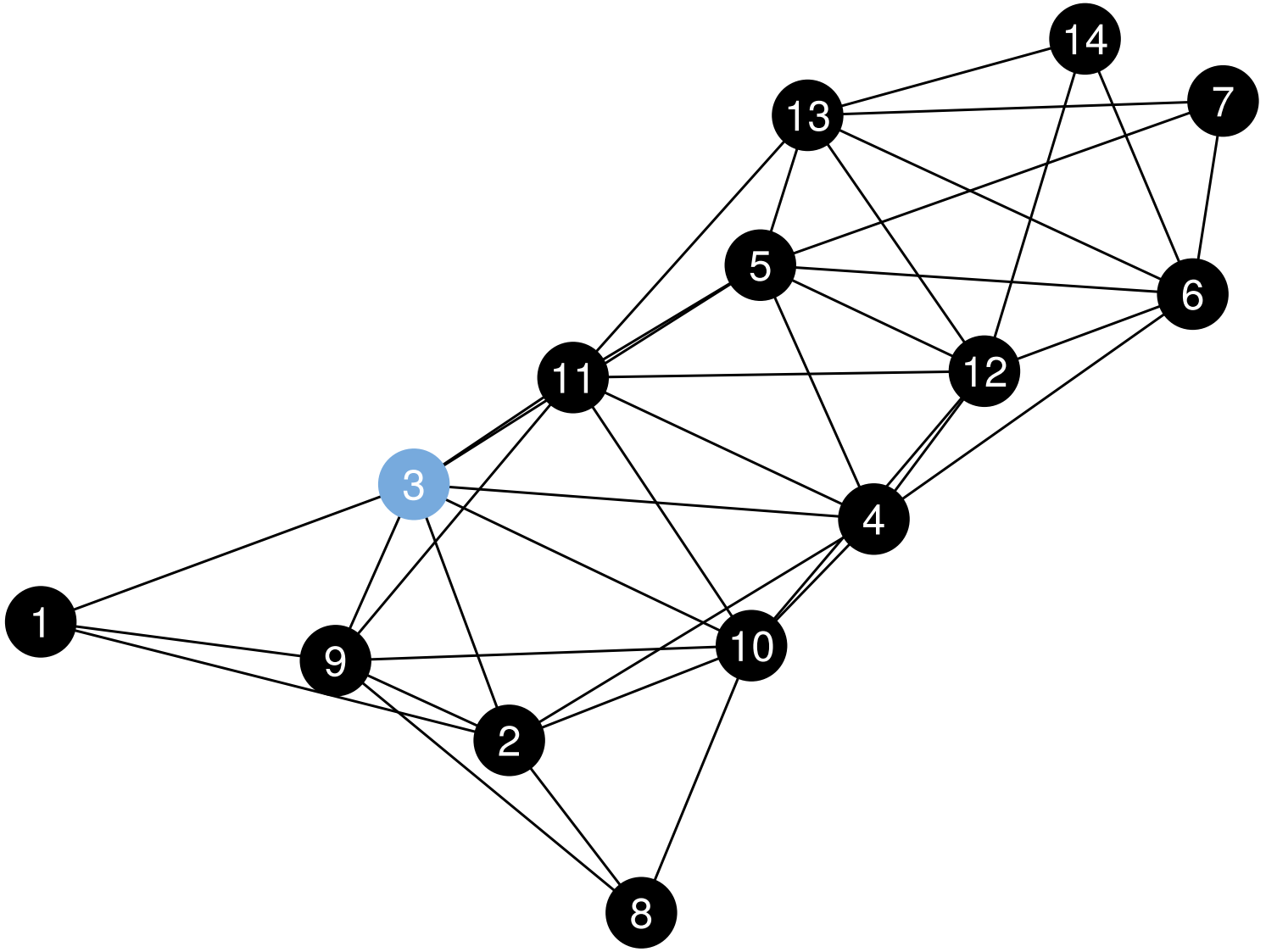
$$\underline{\underline{J}}_{\underline{g}}\,\underline{v} = \left(\underline{\nabla}\cdot\underline{g}\right)\underline{v} = \lim_{h\to 0}\frac{\underline{g}(\underline{x}+\underline{v}h)-g(\underline{x})}{h}$$

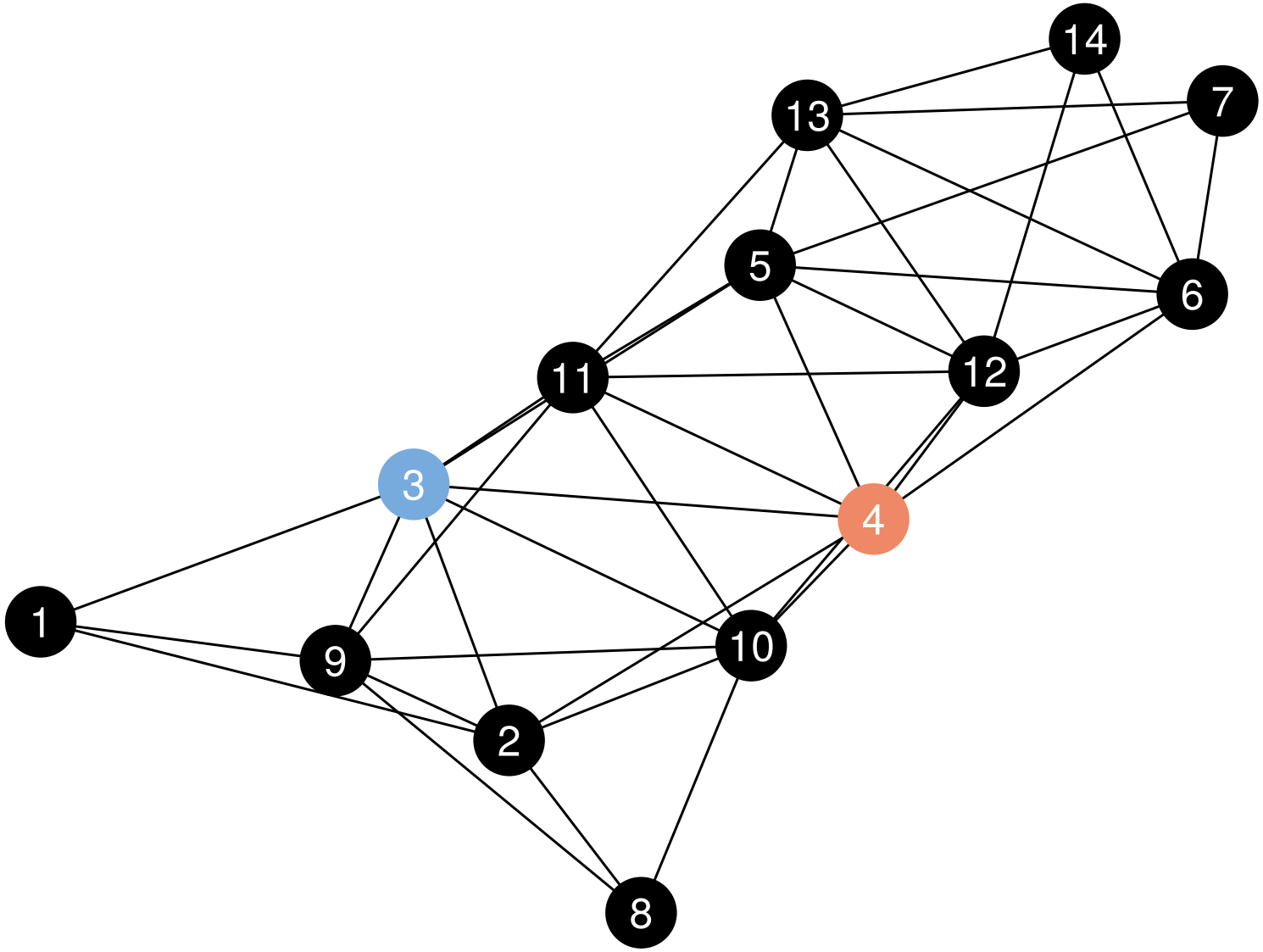
$$\left(\text{as } \left(\underline{\underline{J}}_{\underline{g}}\,\underline{v}\right)_i = \sum_{j=1}^{n}\left(J_g\right)_{ij}v_j = \sum_{j=1}^{n}\frac{\partial g_i}{\partial x_j}v_j = \left(\left(\underline{\nabla}\cdot\underline{g}\right)\underline{v}\right)_i\right)$$

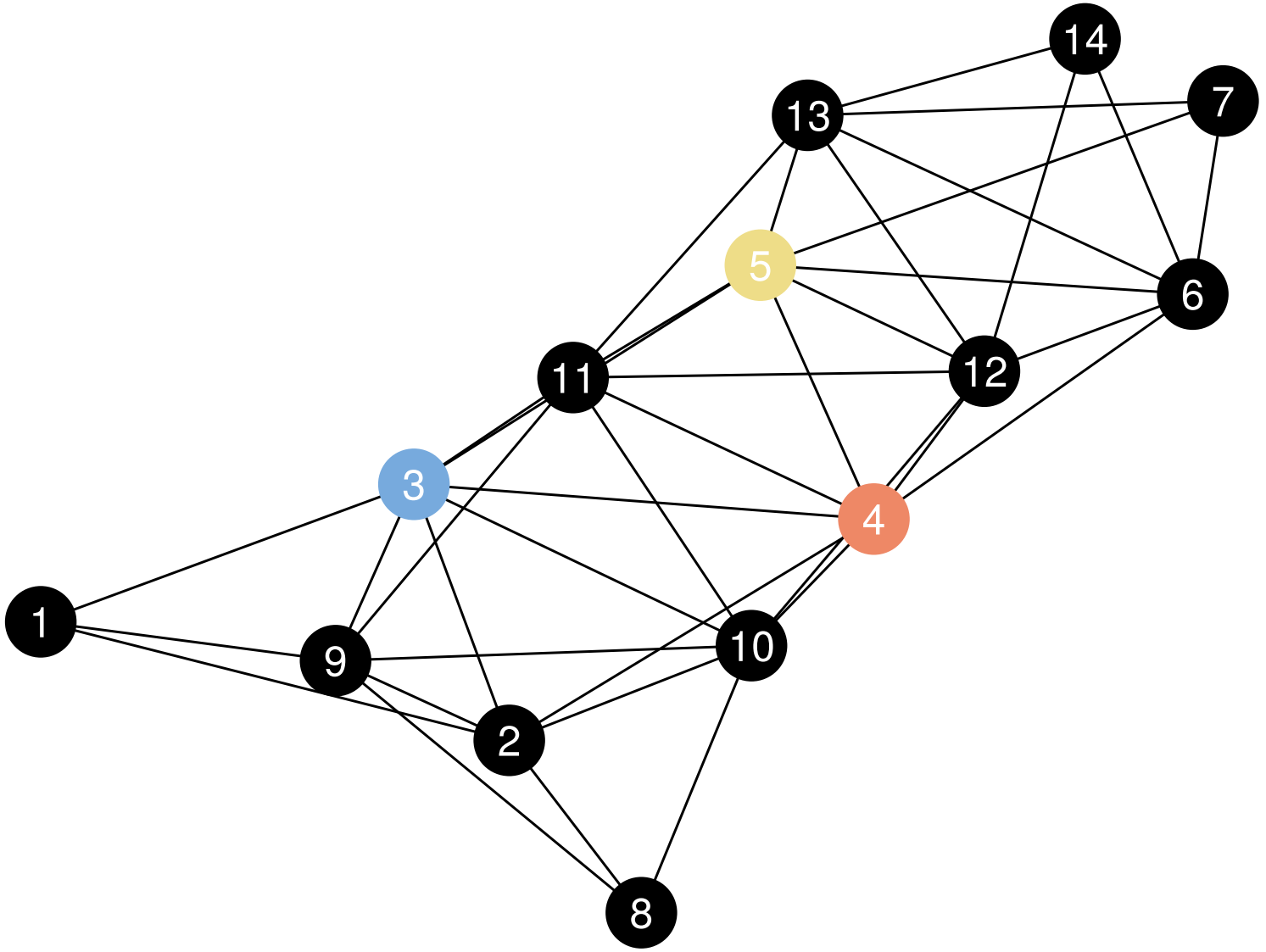
Greedy coloring

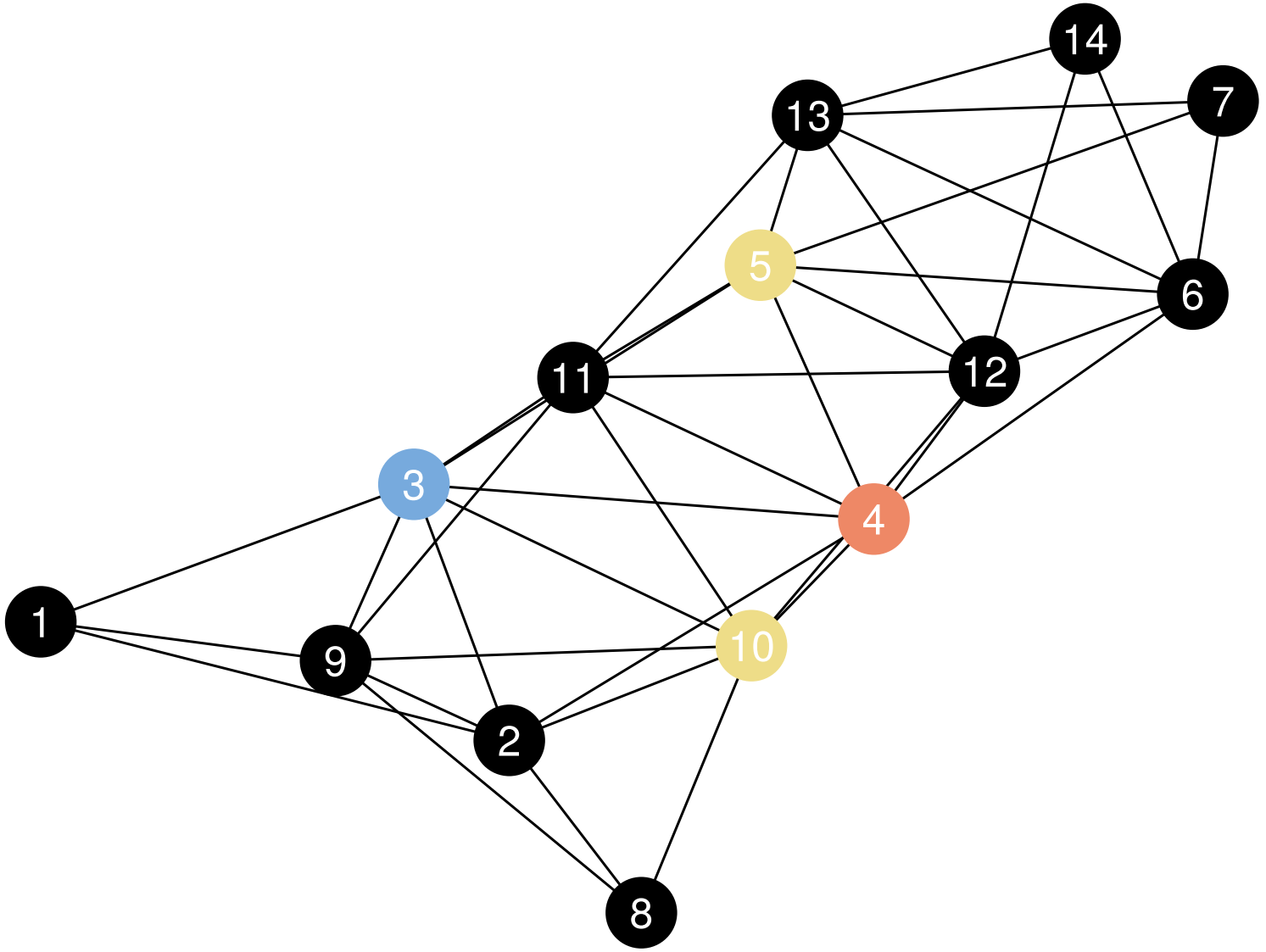
Practically we do not use the optimal solution but some good coloring found by a greedy algorithm (the following one worked the best for our problem)

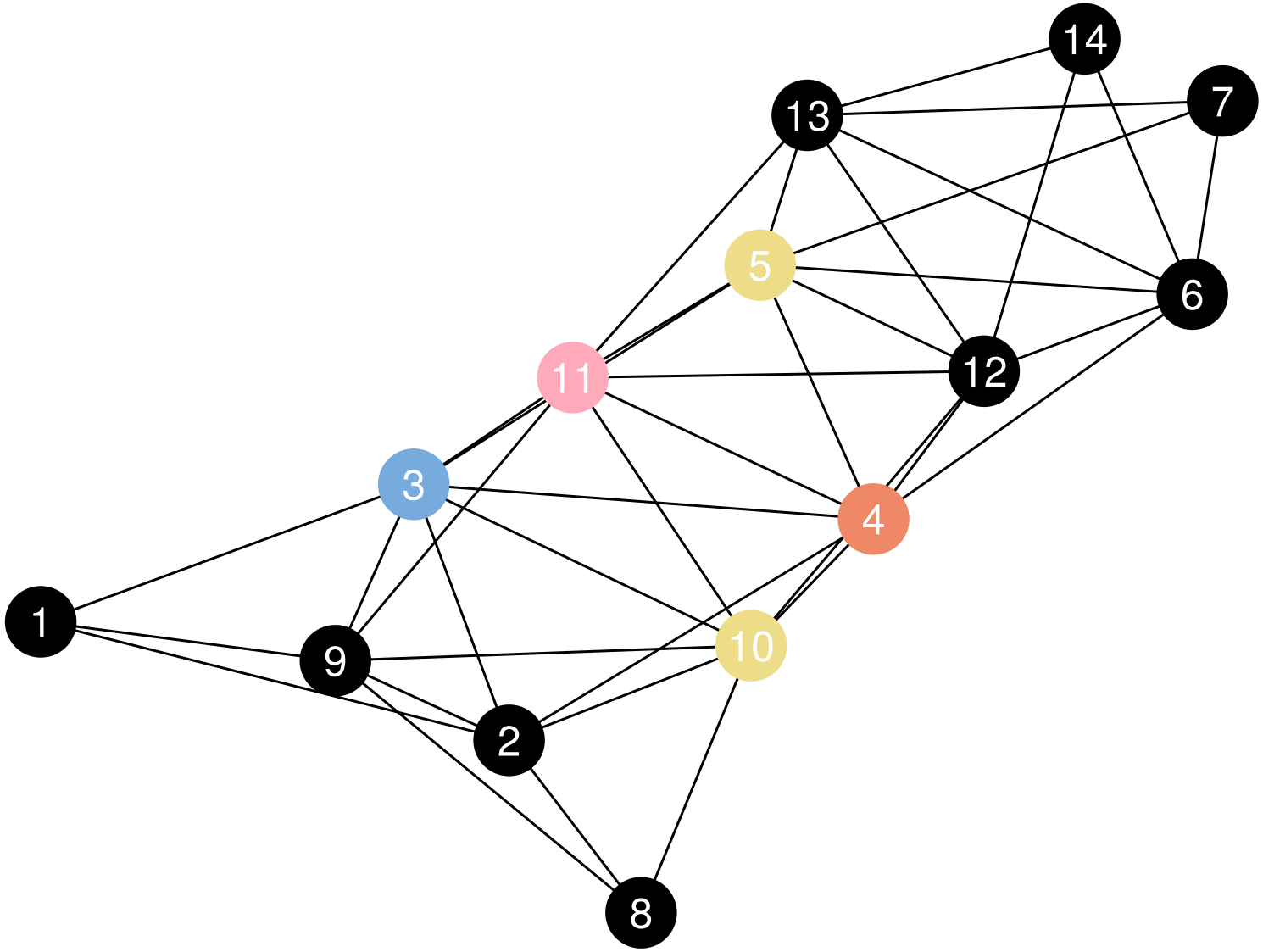
1. Generate a reverse-ordered list of the nodes by the node-degree, i. e. number of edges incident on a node
2. Go through this list (i. e. highest degree first) and choose the smallest color index possible for every node

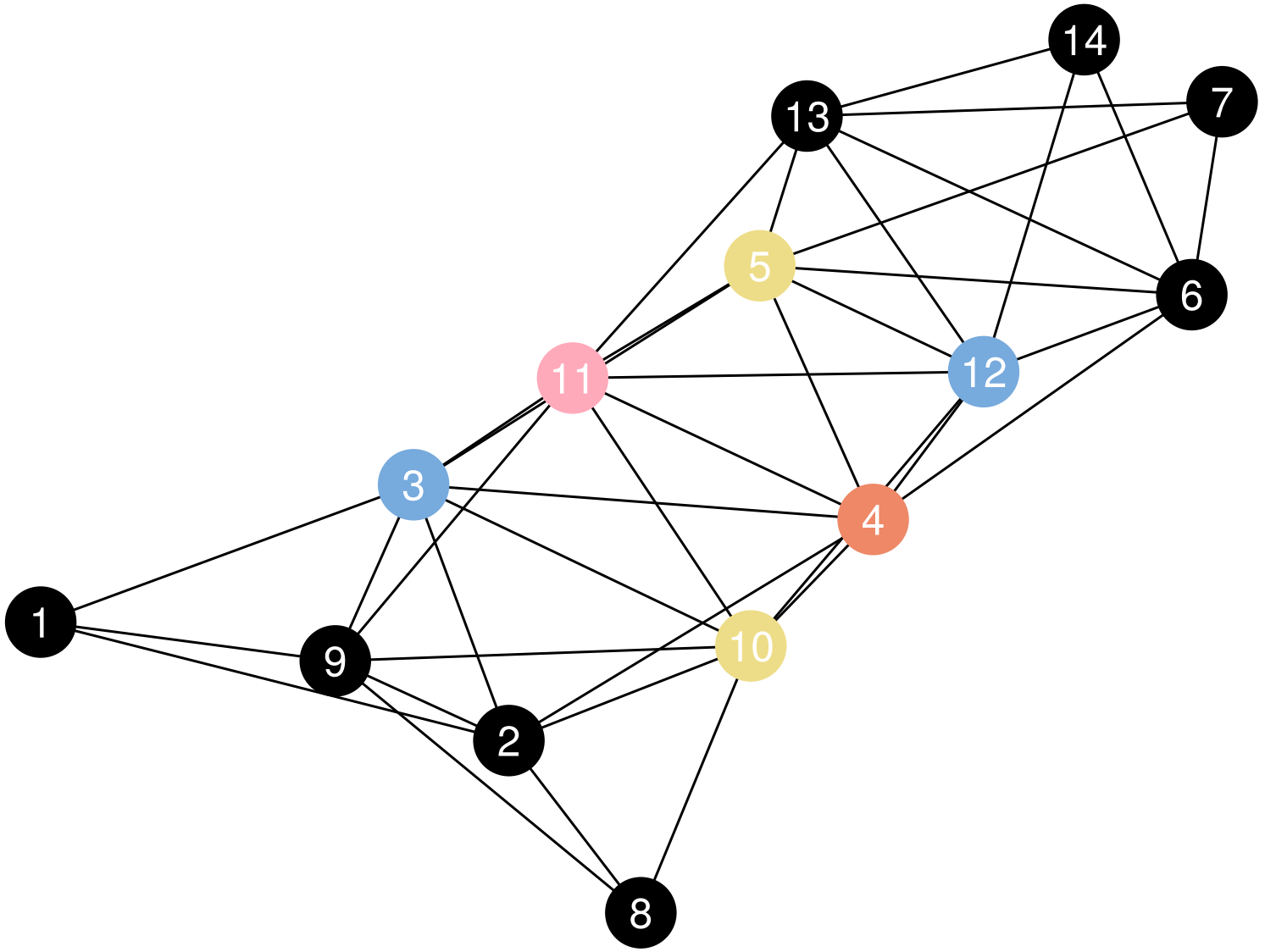

14
13
7
5
6
11
12
3
4
1
9
10
2
8

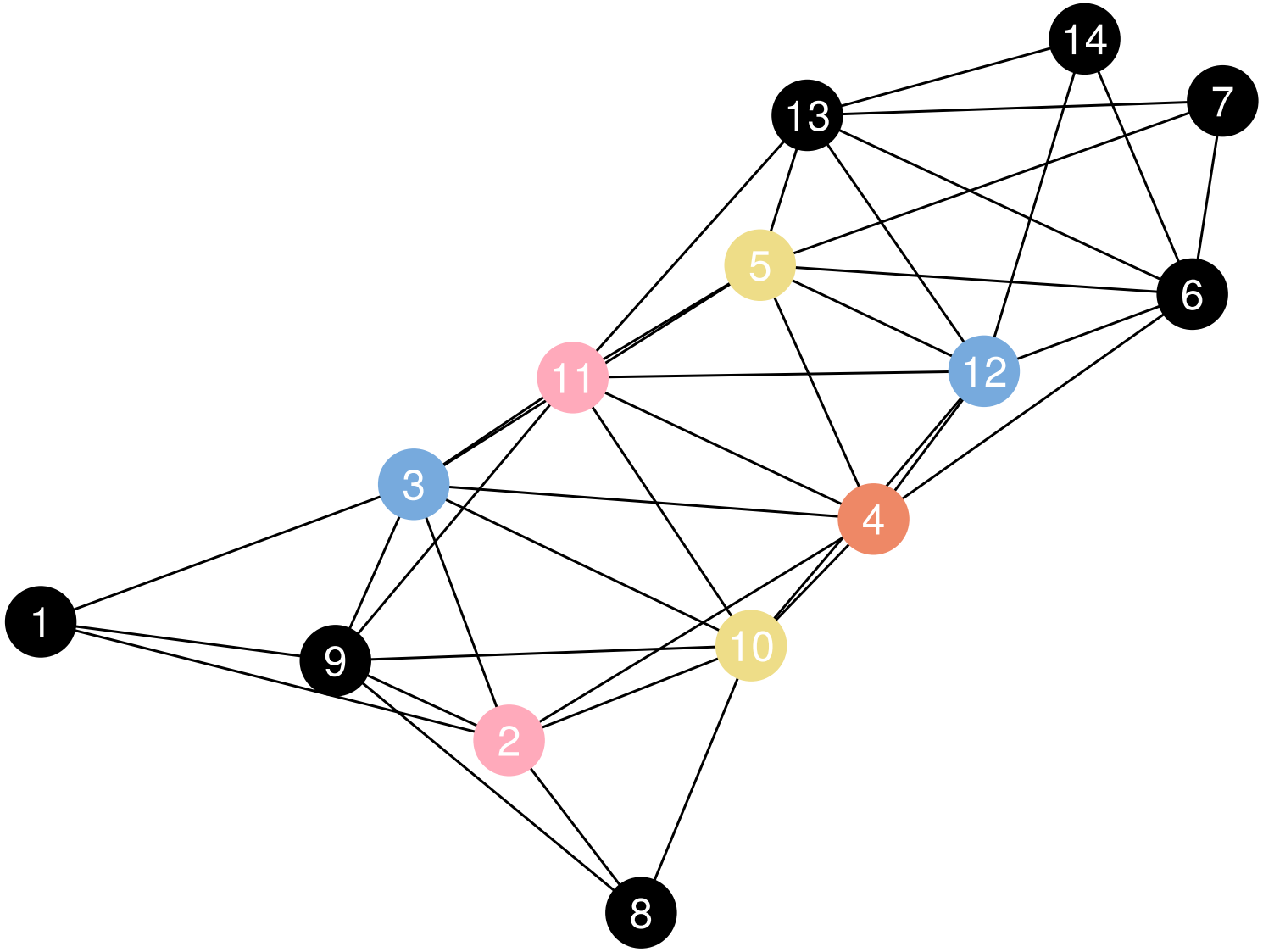

14
13
7
5
6
11
12
3
4
1
9
10
2
8

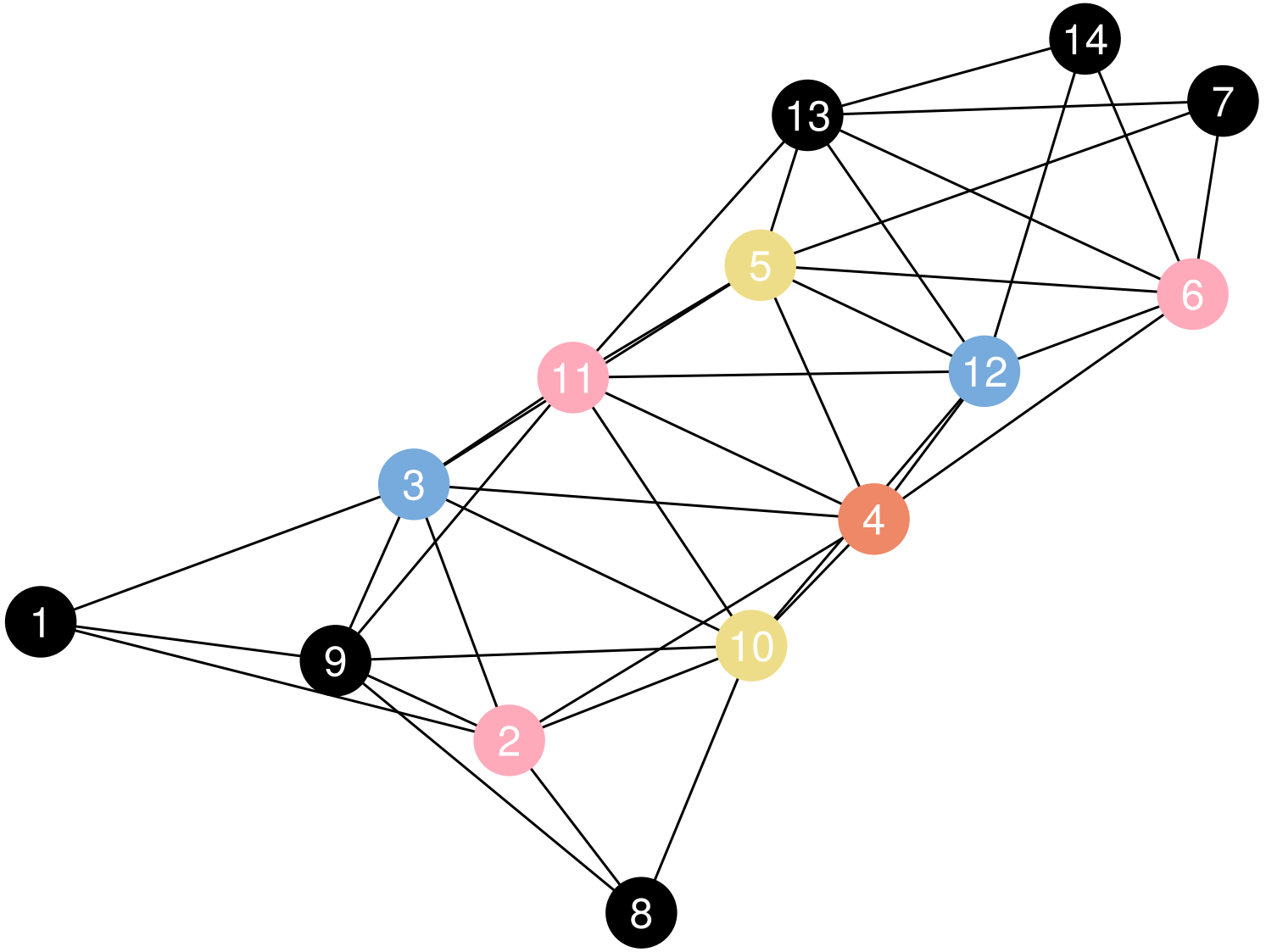

14
13
7
5
6
11
12
3
4
1
9
10
2
8

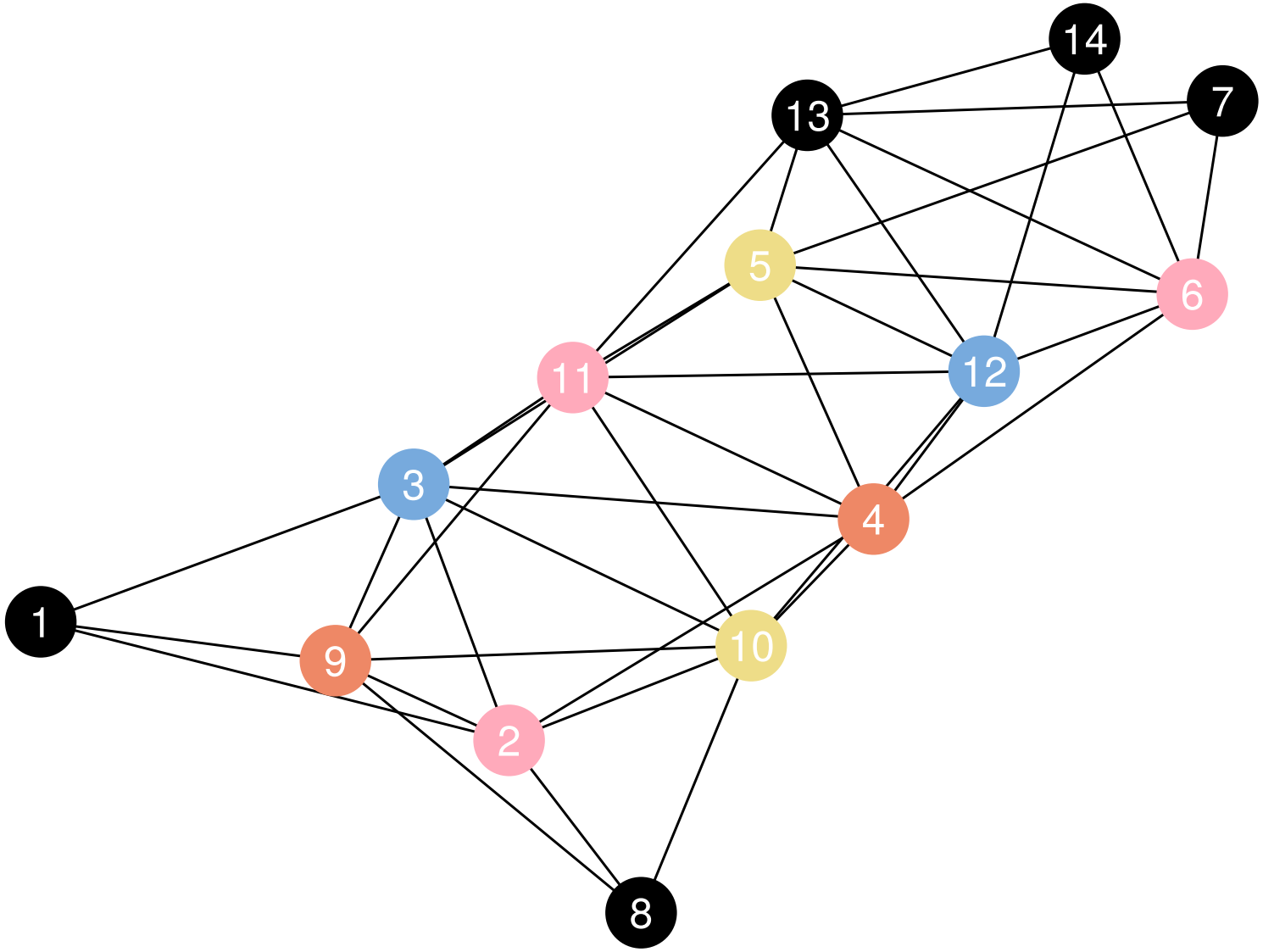

14
13
7
5
6
11
12
3
4
1
9
10
2
8

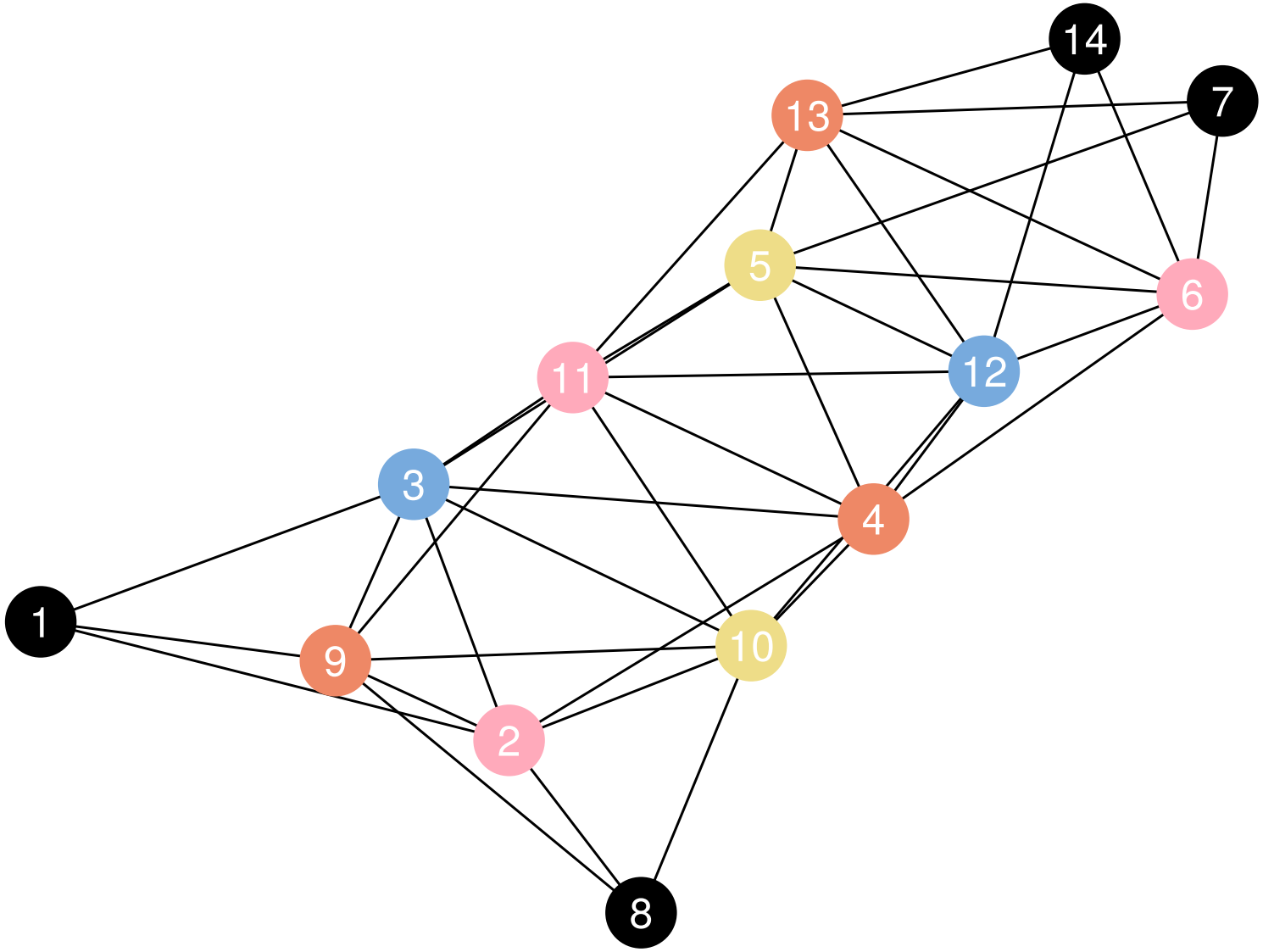

14
13
7
5
6
11
12
3
4
1
9
10
2
8

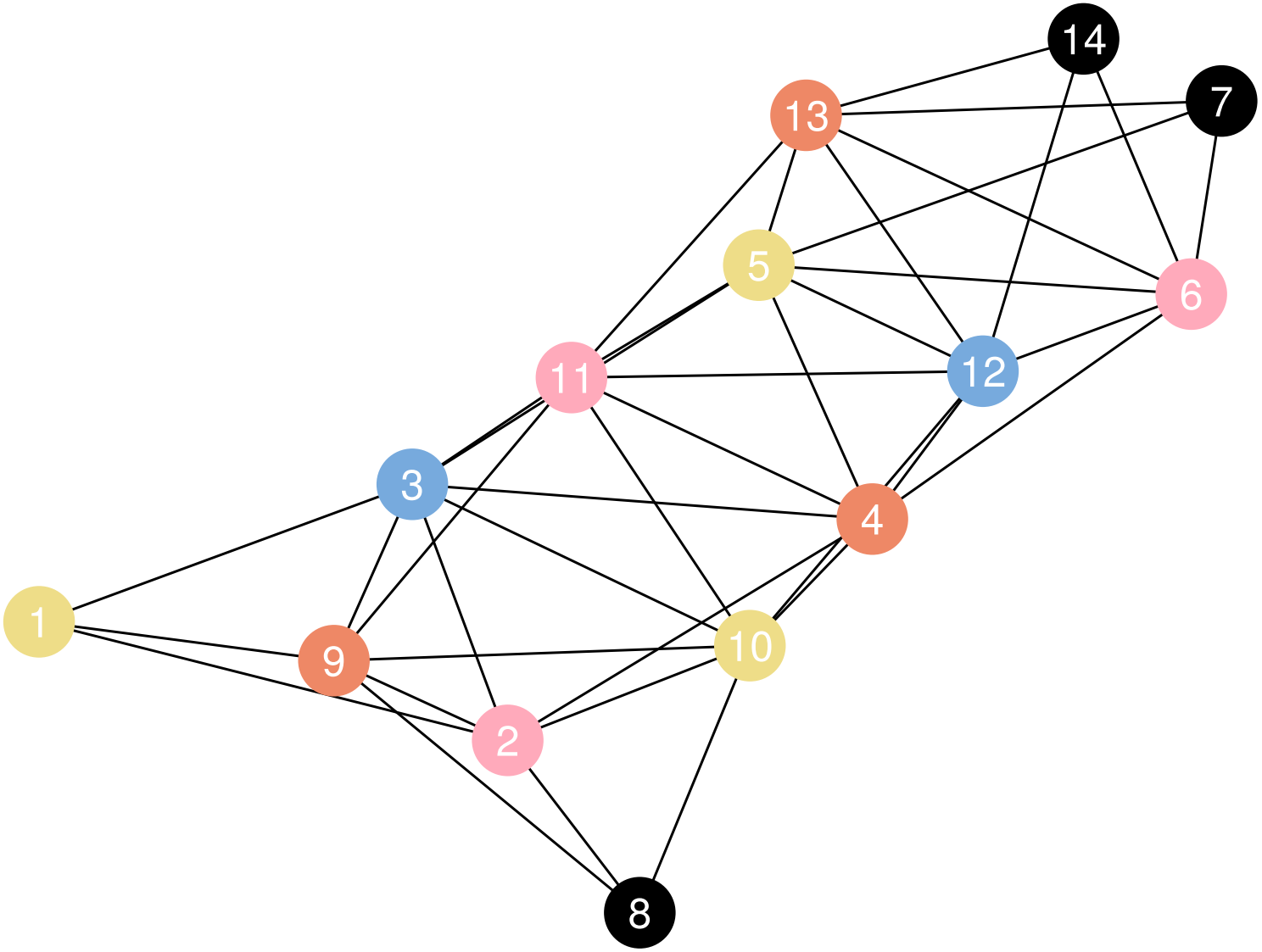

14
13
7
5
6
11
12
3
4
1
9
10
2
8

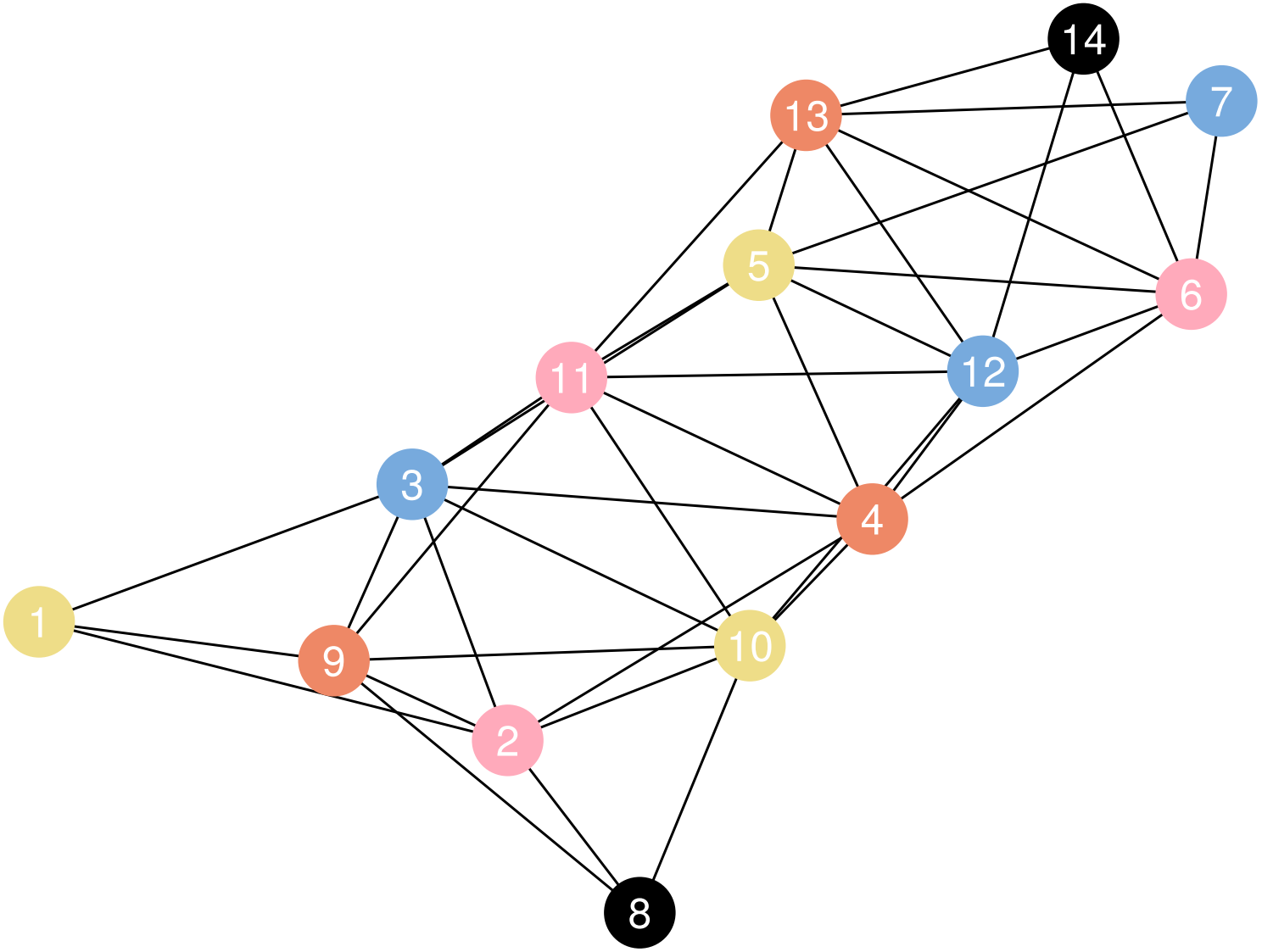

14
13
7
5
6
11
12
3
4
1
9
10
2
8

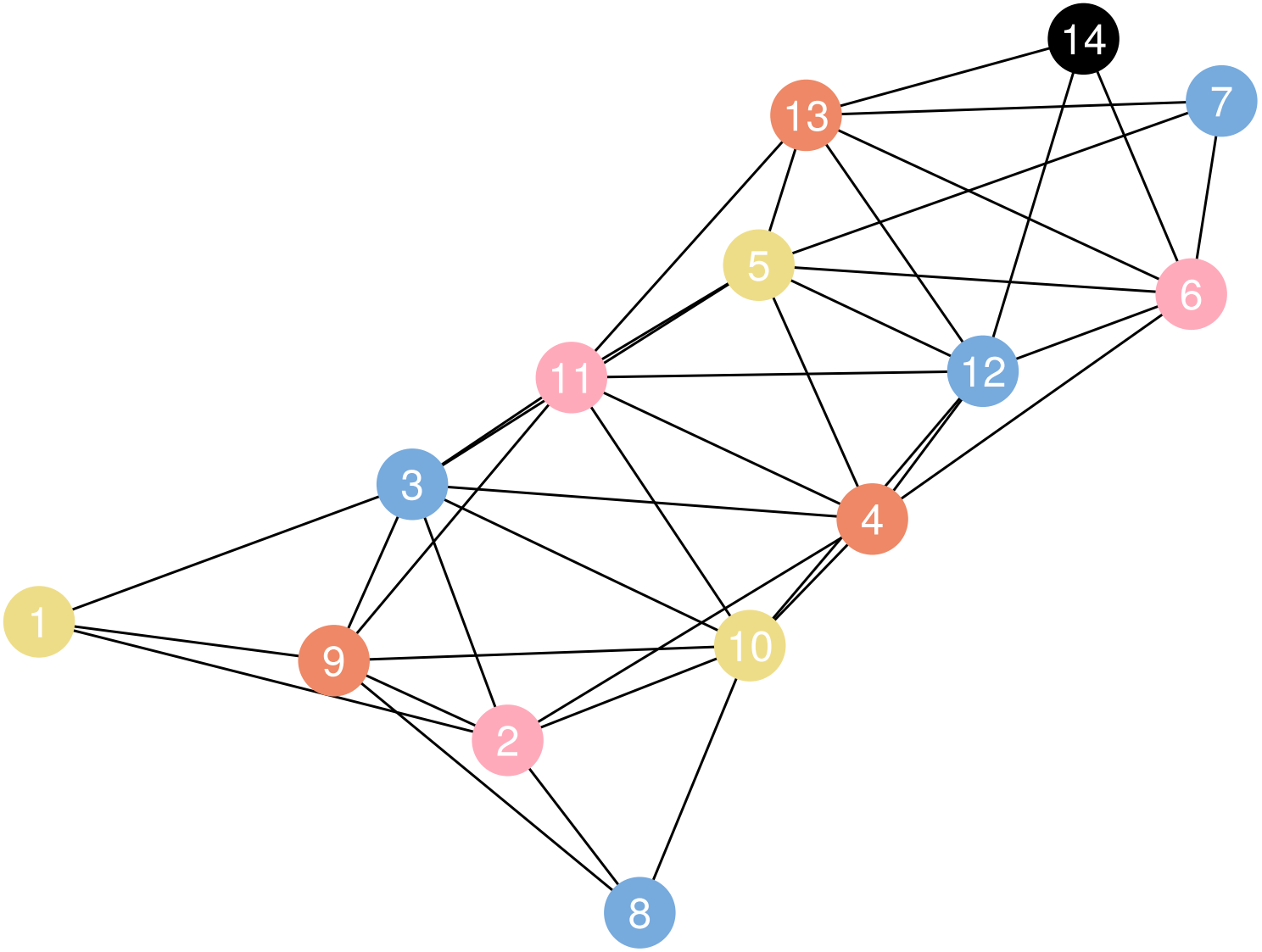

14
13
7
5
6
11
12
3
4
1
9
10
2
8

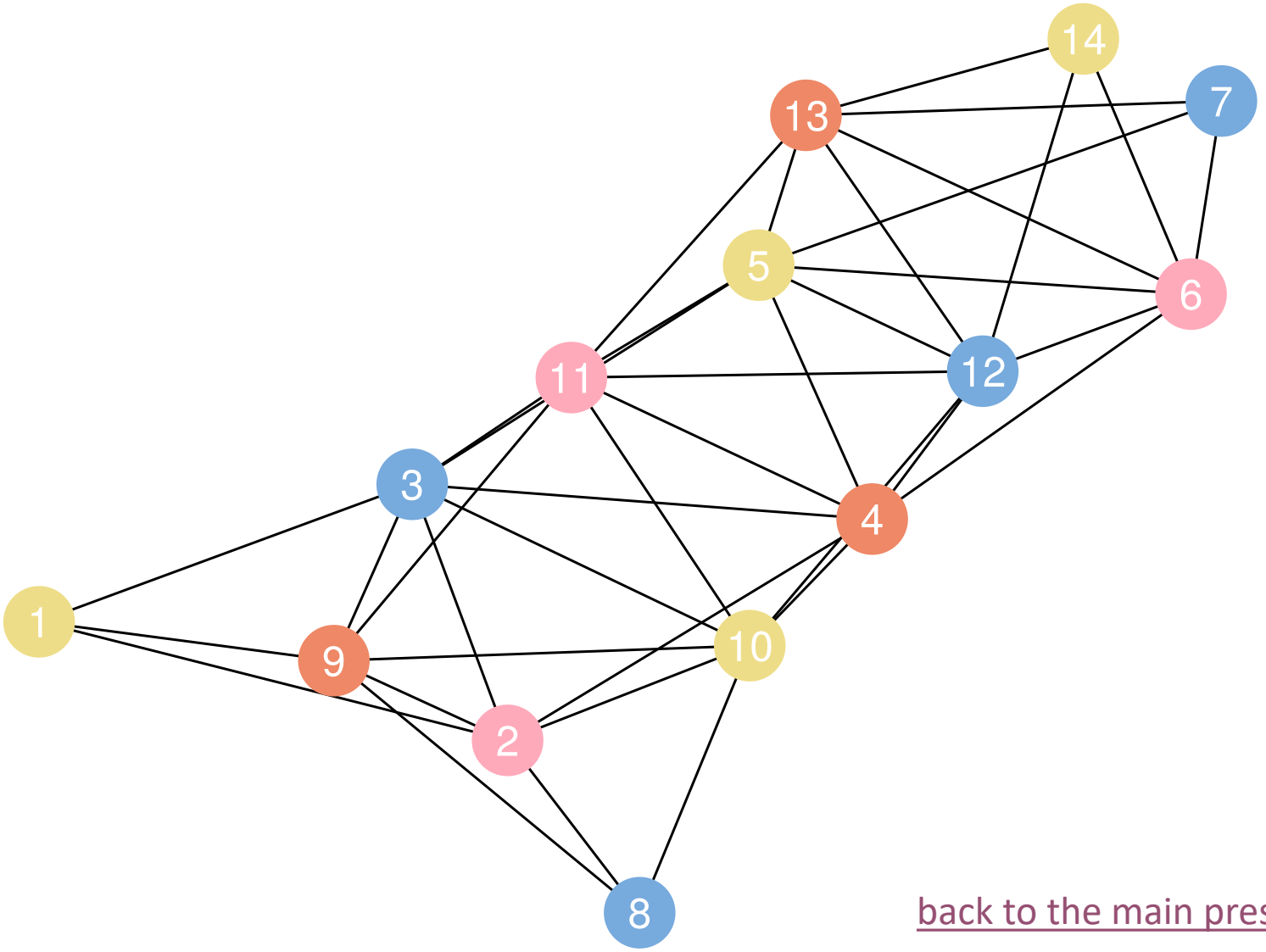

14
13
7
5
6
11
12
3
4
1
9
10
2
8


14
7
13
5
6
11
12
3
4
1
9
10
2
8


14
7
13
5
6
11
12
3
4
1
9
10
2
8


14
13
7
5
6
11
12
3
4
1
9
10
2
8


14
13
7
5
6
11
12
3
4
1
9
10
2
8



back to the main presentation :)

Symplectic integrators and Liouville's theorem

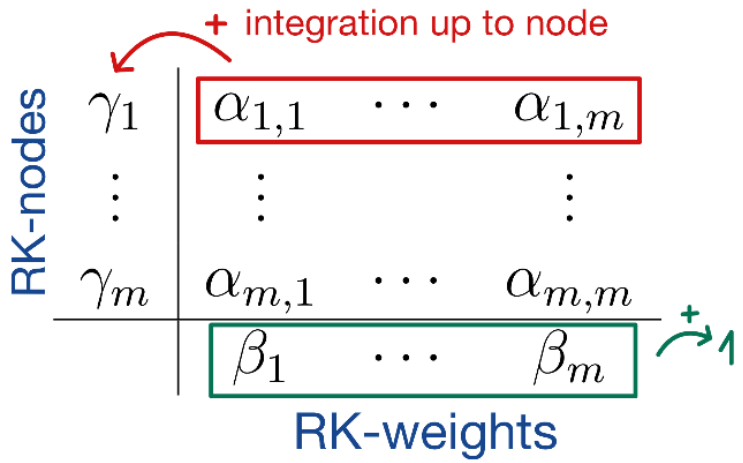
following texts taken from https://www.av8n.com/physics/liouville-intro.htm

It is interesting to look at the «area» in phase space occupied by our group of particles. Actually it's not really an area in the Euclidean sense, but rather a bivector in the Clifford Algebra sense. If you're not familiar with bivectors, to a good approximation you can cross out the word bivector every time you see it and replace it with the word "area" – but keep in mind that it is an *oriented* area. Just as two vectors can have the same length but different direction, two bivectors can have the same area and different orientation. In two dimensions, we reckon the bivector as negative or positive, depending on whether we go around the boundary in the clockwise or counterclockwise direction (respectively).

Liouville: For any group of points, the total bivector (in phase space) occupied by the group remains unchanged over time, as the group flows along in accordance with the laws of motion.

RK45 Fehlberg

Butcher Tableau



$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & & \\
\hline
\text{version A} & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0 & \\
\text{(same } k_j\text{)} & & & & & & & \\
\text{version B} & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55} &
\end{array}
$$

RKF 45

result from different versionsused for error estimate ε