Time Parallel Time Integration
Chapter 1: Introduction

Martin J. Gander
martin.gander@unige.ch

University of Geneva

Michigan, August 1st, 2022

Introduction: Historical Quotes

"The integration methods introduced in this paper are to be regarded as tentative examples of a much wider class of numerical procedures in which parallelism is introduced at the expense of redundancy of computation."

Jörg Nievergelt 1964

"Parallel algorithms for solving initial value problems for differential equations have received only marginal attention in the literature compared to the enormous work devoted to parallel algorithms for linear algebra. *It is indeed generally admitted that the integration of a system of ordinary differential equations in a step-by-step process is inherently sequential."*

Philippe Chartier and Bernard Philippe 1993

"La parallélisation qui en résulte se fait dans la direction temporelle ce qui est en revanche non classique."

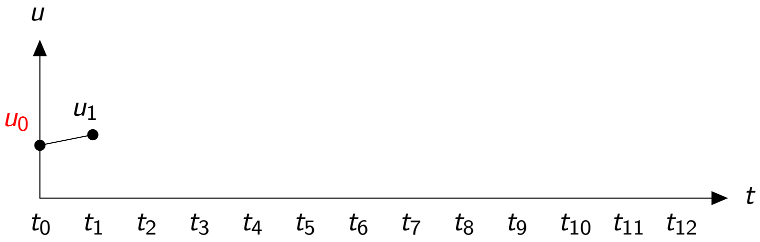
Jacques-Louis Lions, Yvon Maday and Gabriel Turinici 2001

Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1}) - u(t_n)}{\Delta t}$

$$u_1 = u_0 + \Delta t f(u_0)$$

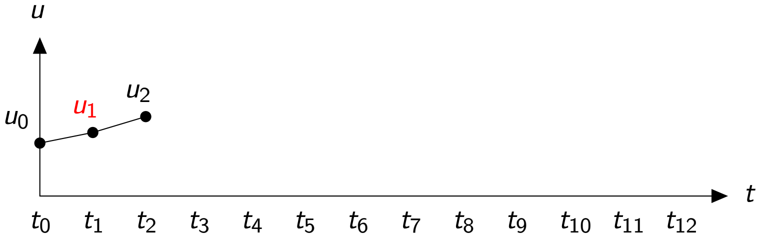


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1})-u(t_n)}{\Delta t}$

$$u_2 = u_1 + \Delta t f(u_1)$$

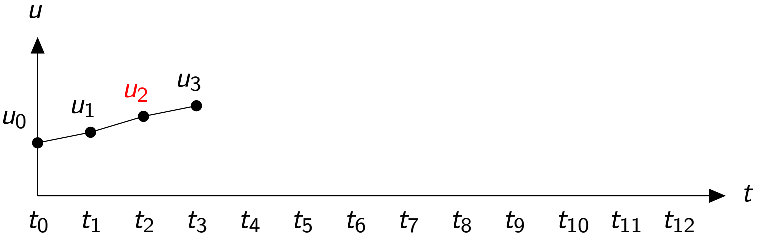


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1})-u(t_n)}{\Delta t}$

$$u_3 = u_2 + \Delta t f(u_2)$$

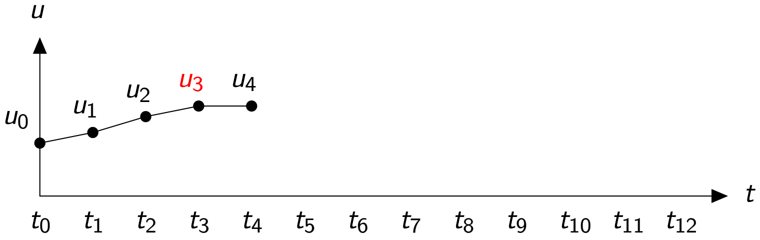


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1})-u(t_n)}{\Delta t}$

$$u_4 = u_3 + \Delta t f(u_3)$$

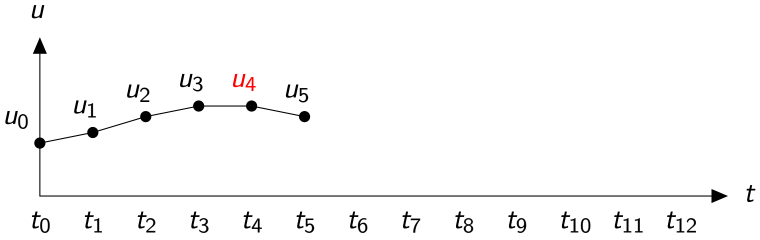


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1}) - u(t_n)}{\Delta t}$

$$u_5 = u_4 + \Delta t f(u_4)$$

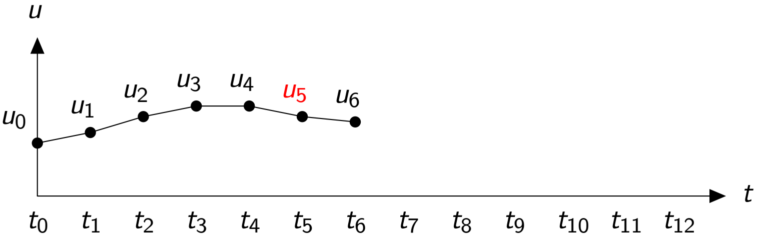


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1}) - u(t_n)}{\Delta t}$

$$u_6 = u_5 + \Delta t f(u_5)$$

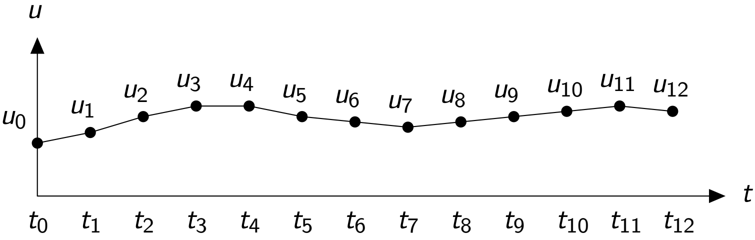


Causality Principle

The time direction is special for parallelization, because of the *causality principle*: the solution later in time is determined by the solution earlier in time, and never the other way round.

Example: $\frac{du}{dt} = f(u)$, $u(t_0) = u_0$, Euler: $\frac{du}{dt} \approx \frac{u(t_{n+1}) - u(t_n)}{\Delta t}$

$$u_{n+1} = u_n + \Delta t f(u_n)$$



Time Parallel Methods Over the Course of Time

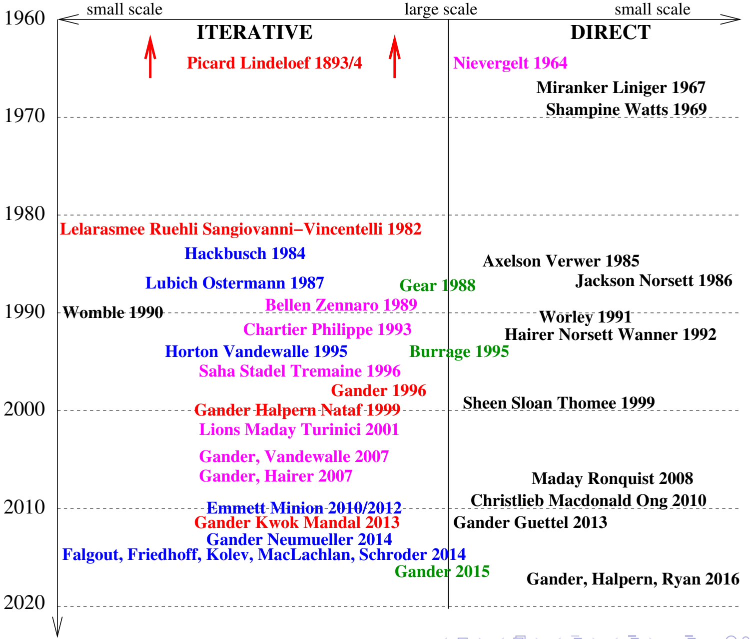


Table of Contents

1. Methods based on multiple shooting:

   Tuesday, Chapter 2

2. Methods based on domain decomposition and waveform relaxation:

   Wednesday, Chapter 3

3. Space-time multigrid methods:

   Thursday, Chapter 4

4. Direct time parallel methods:

   Friday, Chapter 5

Today: Chapter 1: Applications and Model ODEs and PDEs

Weather Prediction as a Typical Application

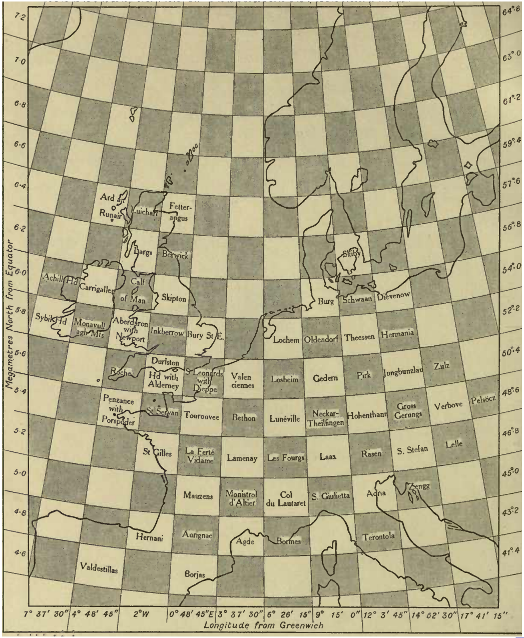
Theophrastus, c. 371 - c. 287 BC

> *"They are less certain when the moon is not full. If the moon looks fiery, it indicates breezy weather for that month, if dusky, wet weather; and, whatever indications the crescent moon gives, are given when it is three days old."*

Richardson (1922): Weather Prediction by Numerical Process (100 years!)

> *"Perhaps some day in the dim future it will be possible to advance the computations faster than the weather advances and at a cost less than the saving to mankind due to the information gained. But that is a dream."*

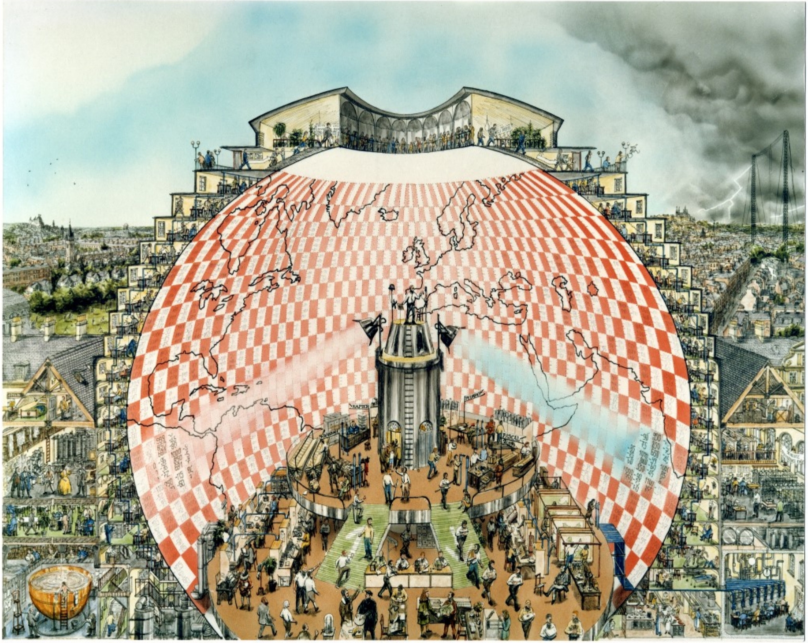
Weather prediction computation by Richardson



"An arrangement of meteorological stations designed to fit with the chief mechanical properties of the atmosphere"

"Pressure to be observed at the centre of each shaded chequer, velocity at the centre of each white chequer."

Richardson's vision (Stephen Conlin, 1986)



Edward N. Lorenz (1979)

On the prevalence of aperiodicity in simple systems

"As the lone meteorologist at a seminar of mathematicians, I feel that a few words regarding my presence may be in order. Let me begin with some remarks about the mathematics of meteorology.

One of the most familiar problems of interest to meteorologists is weather forecasting. Mathematically this is an initial-value problem. The atmosphere and its surroundings are governed by a set of physical laws which in principle can be expressed as a system of integro-differential equations.

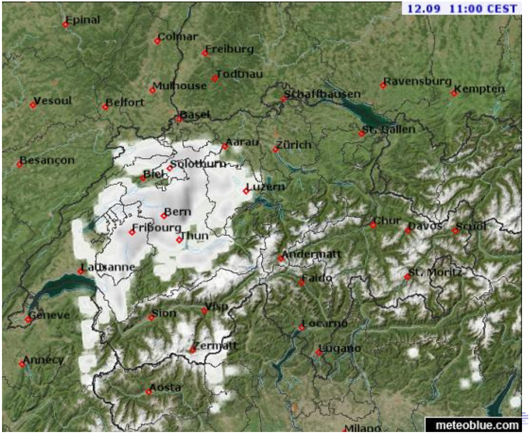
At the turn of the century, the forecast problem was identified by Bjerknes as the problem of solving these equations, using initial conditions obtained from observations of current weather. Detailed numerical procedures for solving these equations were formulated during World War I by Richardson, but the practical solution of even rather crude approximations had to await the advent of computers."

Weather Prediction Equations

$$
\begin{aligned}
\partial_t \boldsymbol{u}(\boldsymbol{x},t) &= \mathcal{L}(\boldsymbol{u}(\boldsymbol{x},t)) && \text{in } \Omega \times (0,T], \\
\boldsymbol{u}(\boldsymbol{x},0) &= \boldsymbol{u}_0(\boldsymbol{x}) && \text{in } \Omega, \\
\mathcal{B}(\boldsymbol{u}(\boldsymbol{x},t)) &= \boldsymbol{g}(\boldsymbol{x},t) && \text{on } \partial\Omega,
\end{aligned}
$$

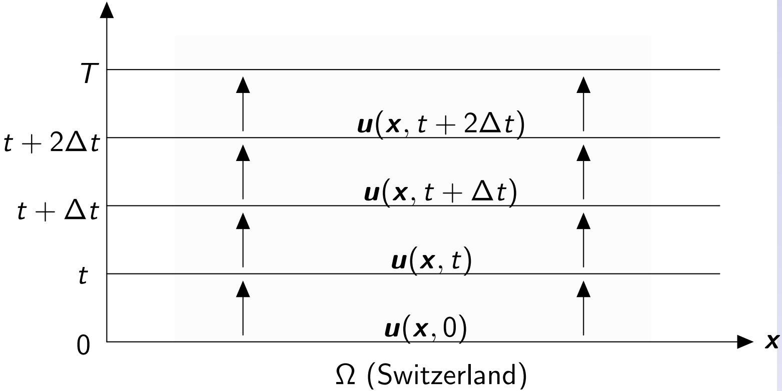
$\boldsymbol{u}$ contains the wind vector, temperature, pressure etc.

If $\boldsymbol{u}_0$ is known over Switzerland, we can predict the weather, using $\boldsymbol{g}$ from a European weather model.



Computing Weather Prediction Step by Step

The solution $\boldsymbol{u}(\boldsymbol{x}, t + \Delta t)$ depends on the solution $\boldsymbol{u}(\boldsymbol{x}, t)$ for any $\Delta t > 0$:



This is again an entirely sequential process, impossible to use effectively all processors to compute $\boldsymbol{u}(\boldsymbol{x}, t + \Delta t)$ from $\boldsymbol{u}(\boldsymbol{x}, t)$, there are simply too many processors.

Evolution of Computing Systems

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	361,760	19,590.0	25,326.3	2,272
4	Japan Agency for Marine-Earth Science and Technology Japan	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz ExaScaler	19,860,000	19,135.8	28,192.0	1,350
5	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209

Top 500 list in November 2017

Evolution of Computing Systems

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942
4	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
5	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438

Top 500 list in June 2022

Ordinary Differential Equations

General system of ordinary differential equations (ODEs)

$$\begin{aligned} \partial_t \boldsymbol{u}(t) &= \boldsymbol{f}(t, \boldsymbol{u}(t)) \qquad t \in (0, T], \\ \boldsymbol{u}(0) &= \boldsymbol{u}_0, \end{aligned}$$

Edward N. Lorenz (1979): On the prevalence of aperiodicity in simple systems

> *"The first task was to find a suitable system of equations to solve. In principle any nonlinear system might do, but a system with some resemblance to the atmospheric equations offered the possibility of some useful by-products."*

Lorenz Equations: for parameters $\sigma, r, b \in \mathbb{R}$:

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y & x(0) = x_0, \\ \dot{y} &= -xz + rx - y & y(0) = y_0, \\ \dot{z} &= xy - bz & z(0) = z_0, \end{aligned}$$

Fixed Points and Trajectories

$$\begin{aligned}
\dot{x} &= -\sigma x + \sigma y & x(0) &= x_0, \\
\dot{y} &= -xz + rx - y & y(0) &= y_0, \\
\dot{z} &= xy - bz & z(0) &= z_0,
\end{aligned}$$

Fixed points: when $\dot{x} = \dot{y} = \dot{z} = 0$,

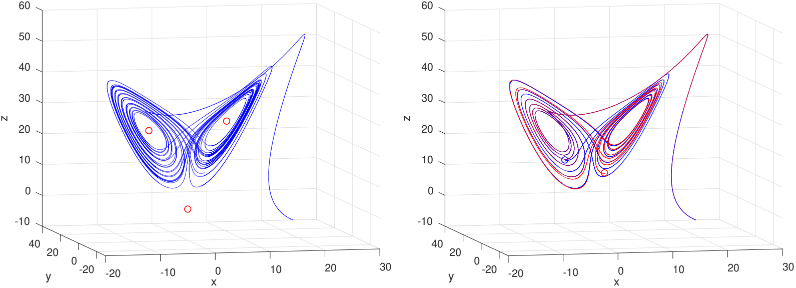
1. $x = y = z = 0$,
2. $x = y = \pm\sqrt{b(r-1)}$, $z = r - 1$.

Fixed Points and Trajectories

$$
\begin{aligned}
\dot{x} &= -\sigma x + \sigma y & x(0) &= x_0, \\
\dot{y} &= -xz + rx - y & y(0) &= y_0, \\
\dot{z} &= xy - bz & z(0) &= z_0,
\end{aligned}
$$

Fixed points: when $\dot{x} = \dot{y} = \dot{z} = 0$,

1. $x = y = z = 0$,
2. $x = y = \pm\sqrt{b(r-1)}$, $z = r - 1$.



The Butterfly Effect

Edward N. Lorenz (1979): On the prevalence of aperiodicity in simple systems

> *"Two states differing by imperceptible amounts may eventually evolve into two considerably different states . . . If, then, there is any error whatever in observing the present state – and in any real system such errors seem inevitable – an acceptable prediction of an instantaneous state in the distant future may well be impossible. . . . In view of the inevitable inaccuracy and incompleteness of weather observations, precise very-long-range forecasting would seem to be nonexistent."*

This led to the famous saying that the flapping of the wings of a butterfly in Europe can change the weather completely in the US.

Numerical Approximations

In the general system of ordinary differential equations (ODEs)

$$\begin{aligned} \partial_t \boldsymbol{u}(t) &= \boldsymbol{f}(t, \boldsymbol{u}(t)) \qquad t \in (0, T], \\ \boldsymbol{u}(0) &= \boldsymbol{u}_0, \end{aligned}$$

one approximates the time derivative by a finite difference,

$$\partial_t \boldsymbol{u} \approx \frac{\boldsymbol{u}(t+\Delta t) - \boldsymbol{u}(t)}{\Delta t}$$

The two most simple methods are Euler methods:

- **Forward Euler**: $\frac{\boldsymbol{u}_{n+1} - \boldsymbol{u}_n}{\Delta t} = \boldsymbol{f}(t_n, \boldsymbol{u}_n)$,
- **Backward Euler:**: $\frac{\boldsymbol{u}_{n+1} - \boldsymbol{u}_n}{\Delta t} = \boldsymbol{f}(t_{n+1}, \boldsymbol{u}_{n+1})$.

Forward Euler is much easier to use than Backward Euler!

A Matlab Implementation

```
sigma=10;r=28;b=8/3;
f=@(t,x) [sigma*(x(2)-x(1)); r*x(1)-x(2)-x(1)*x(3); x(1)*x(2)-b*x(3)];
T=30;N=30000;dt=T/N;
x=[20;5;-5];
for i=1:N
  x(:,i+1)=x(:,i)+dt*f(i*dt,x(:,i));             % Forward Euler step
  if mod(i,100)==0                               % plot only every 100th
    plot3(x(1,:),x(2,:),x(3,:),'-b');            % for animation speed
    axis([-20 30 -30 40 -10 60]); view([-13,8]);
    xlabel('x'); ylabel('y'); zlabel('z');
    grid on
    pause
  end
end
hold on
xf=sqrt(b*(r-1)); yf=sqrt(b*(r-1)); zf=r-1;
plot3(xf,yf,zf,'or'); plot3(-xf,-yf,zf,'or');    % plot fixed points
plot3(0,0,0,'or');
hold off
```

A Better Matlab Implementation

```
function [t,u]=ForwardEuler(f,tspan,u0,N);
% FORWARDEULER solves system of ODEs using the Forward Euler method
%   [t,u]=ForwardEuler(f,tspan,u0,N) solves du/dt=f(t,u) with initial
%   value u0 on the time interval tspan doing N steps of Forward
%   Euler. Returns the solution in time and space in the matrix u, and
%   also the corresponding time points in the column vector t.

dt=(tspan(2)-tspan(1))/N;
t=(tspan(1):dt:tspan(2))';
u(1,:)=u0(:);                          % colon to make column vector
for n=1:N,
  u(n+1,:)=u(n,:)+dt*f(t(n),u(n,:));
end;
```

Properties of Discretizations

Local truncation error: error the method makes after one step,

$$\tau := ||\boldsymbol{u}(\Delta t) - \boldsymbol{u}_1||,$$

where any suitable vector norm can be chosen. For Euler

$$\begin{aligned} \tau_{FE} &= ||\boldsymbol{u}(\Delta t) - \boldsymbol{u}_1|| \\ &= ||\boldsymbol{u}(0) + \boldsymbol{u}'(0)\Delta t + O(\Delta t^2) - (\boldsymbol{u}_0 + \Delta t f(0, \boldsymbol{u}_0))|| \\ &= O(\Delta t^2) \end{aligned}$$

Global truncation error: error the method makes after many steps,

$$E := ||\boldsymbol{u}(T) - \boldsymbol{u}_N||, \quad N\Delta t = T.$$

The global truncation error is one order less than the local truncation error, Theorem 10.2 in Gander-Gander-Kwok 2014.

Better methods, like Runge-Kutta or linear multistep methods, are in Chapter 10, Gander-Gander-Kwok 2014.

Dahlquist Test Equation

$$\dot{u} = \lambda u, \quad u(0) = u_0, \quad \lambda \in \mathbb{C} \quad \Longrightarrow \quad u(t) = u_0 e^{\lambda t}.$$

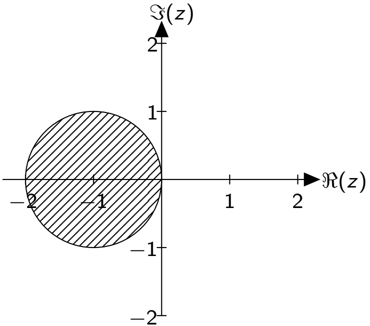
Growing if $\Re(\lambda) > 0$, and decaying if $\Re(\lambda) < 0$.

What happens when we use Forward Euler?

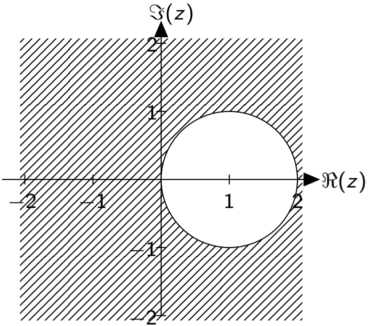
$$u_{n+1} = u_n + \Delta t \lambda u_n = (1 + \Delta t \lambda) u_n =: R_{FE}(z) u_n,$$

with $z := \Delta t \lambda \in \mathbb{C}$.

Region of absolute stability: $|R_{FE}(z)| < 1$



Forward Euler



Backward Euler

Heat Equation

Parabolic partial differential equation, studied by *Joseph Fourier* in 1822

$$
\begin{aligned}
\partial_t u(\boldsymbol{x},t) &= \Delta u(\boldsymbol{x},t) + f(\boldsymbol{x},t) && \text{in } \Omega \times (0,T], \\
u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) && \text{in } \Omega, \\
u(\boldsymbol{x},t) &= g(\boldsymbol{x},t) && \text{on } \partial\Omega \times (0,T].
\end{aligned}
$$

Example: A nail between ice cubes

$$
\begin{aligned}
\partial_t u(x,t) &= \partial_{xx} u(x,t) && \text{in } (0,L) \times (0,T], \\
u(x,0) &= u_0(x) && \text{in } (0,L), \\
u(0,t) &= 0 && \text{in } (0,T], \\
u(L,t) &= 0 && \text{in } (0,T].
\end{aligned}
$$

Numerical Approximation

Spatial mesh with grid points $x_j := j\Delta x$, $j = 0, 1, \ldots, J$, $\Delta x := L/J$, approximates the Laplace operator by

$$\partial_{xx} u(x,t) \approx \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2},$$

This leads to the system of ordinary differential equations

$$\partial_t u_j(t) = \frac{u_{j+1}(t) - 2u_j(t) + u_{j-1}(t)}{\Delta x^2},$$

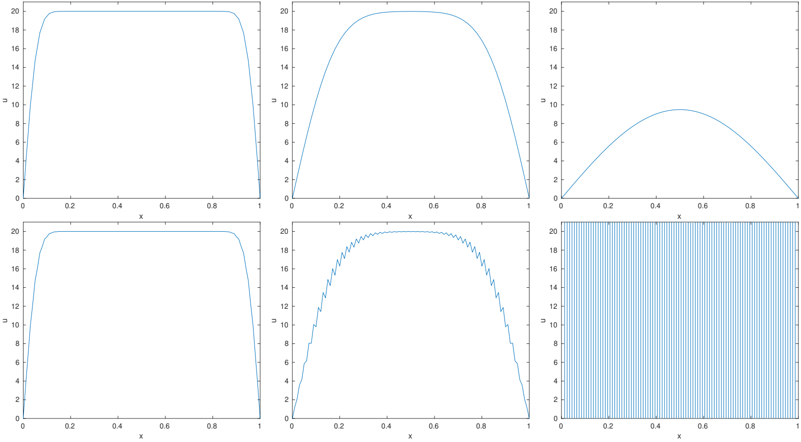
Using now forward Euler in time gives

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2},$$

and with Backward Euler, we obtain

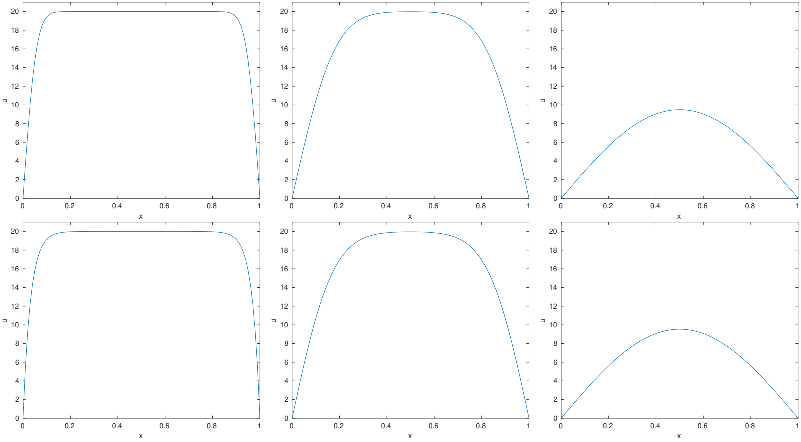
$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2}.$$

Approximate solution of the heat equation: FE



Top row: approximate solutions at $t = 0.001$, $t = 0.01$ and $t = 0.1$ using $\Delta x = 0.01$ and $\Delta t = 0.00005$.
Bottom row: results when using $\Delta t = 0.000050505\ldots$.

Approximate solution of the heat equation: BE



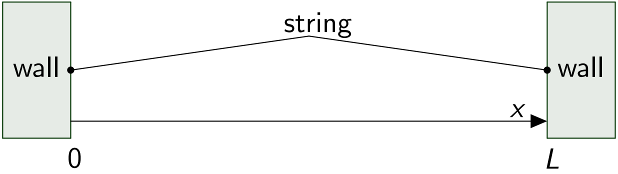
Top row: approximate solution at $t = 0.001$, $t = 0.01$ and $t = 0.1$ using $\Delta x = 0.01$ and $\Delta t = 0.000050505\ldots$.
Bottom row: results when using $\Delta t = 0.001$.

The Second Order Wave Equation

Hyperbolic partial differential equation, studied by Jean-Baptiste le Rond d'Alembert in 1747:

$$\begin{aligned}
\partial_{tt} u(\boldsymbol{x},t) &= c^2 \Delta u(\boldsymbol{x},t) + f(\boldsymbol{x},t) && \text{in } \Omega \times (0,T],\\
u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) && \text{in } \Omega,\\
u_t(\boldsymbol{x},0) &= \tilde{u}_0(\boldsymbol{x}) && \text{in } \Omega,\\
u(\boldsymbol{x},t) &= g(\boldsymbol{x},t) && \text{on } \partial\Omega \times (0,T].
\end{aligned}$$

Example: vibration of a string fixed between two walls



$$\begin{aligned}
\partial_{tt} u(x,t) &= \partial_{xx} u(x,t) && \text{in } (0,L) \times (0,T],\\
u(x,0) &= u_0(x) && \text{in } (0,L),\\
\partial_t u(x,0) &= 0 && \text{in } (0,L),\\
u(0,t) &= 0 && \text{in } (0,T],\\
u(L,t) &= 0 && \text{in } (0,T].
\end{aligned}$$

Numerical Approximation

Can use the same finite difference approximation in time and space

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}.$$

Need the values at $t = 0$,

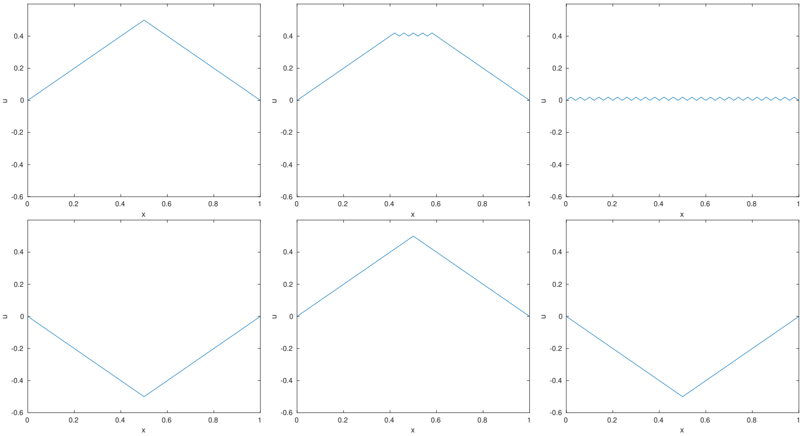
$$u_j^0 = u(x_j, 0),$$

and also at $t = \Delta t$ to start the three step recurrence. One can use for this a Forward Euler approximation,

$$\frac{u_j^1 - u_j^0}{\Delta t} = \tilde{u}(x_j).$$

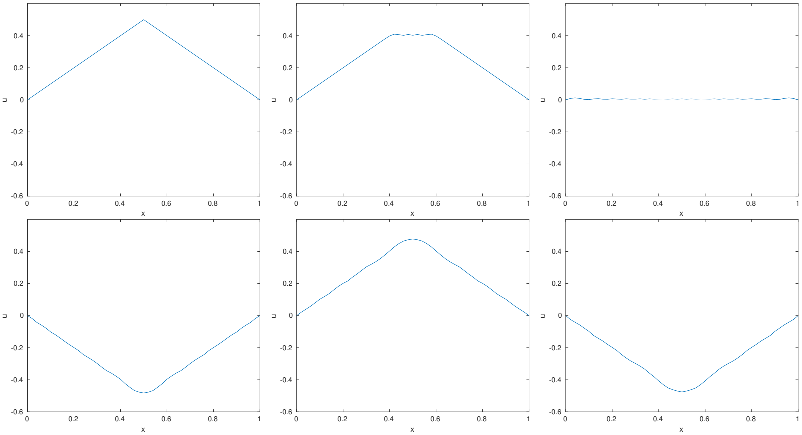
Now how does a string of a musical instrument oscillate ?

Approximate Solution



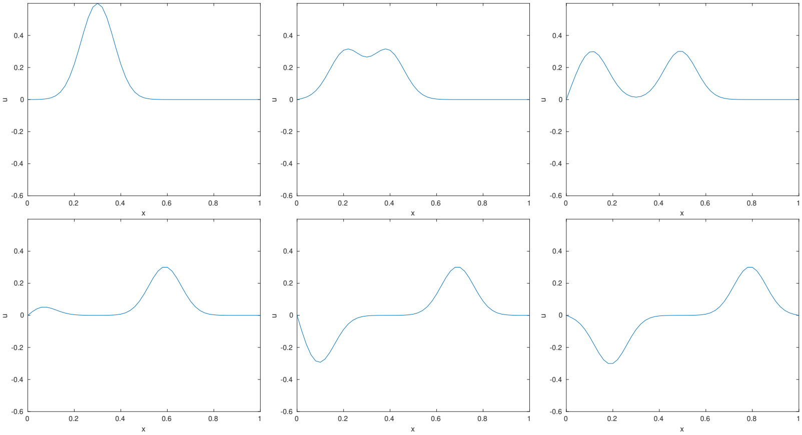
Snapshots of the approximate solution of the string using the classical wave equation at $t = 0, 0.1, 0.5, 1, 2, 3$ using for $L = 1$, $\Delta x = \Delta t = 0.02$.

Approximate Solution with smaller time step



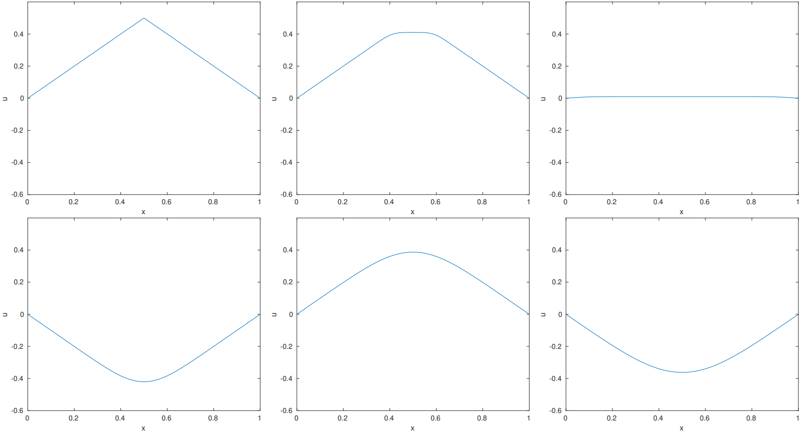
No sine shape developing like for a guitar for example!

Starting with a different initial condition



Visco-Elastic Damping

$$\partial_{tt} u(x,t) = \partial_{xx} u(x,t) + \gamma \partial_{xxt} u(x,t)$$



Snapshots of the approximate solution of the true string at $t = 0, 0.1, 0.5, 1, 2, 3$.

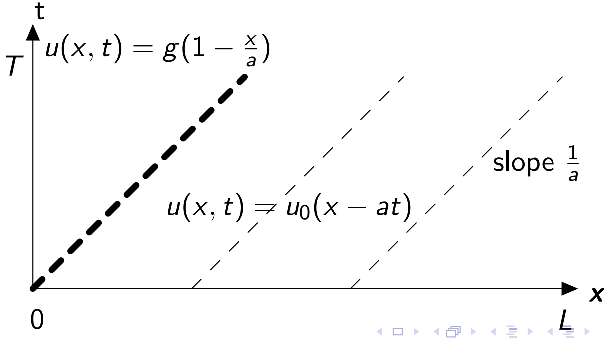
The Transport or Advection Equation

Simplest hyperbolic partial differential equation,

$$\begin{aligned} \partial_t u(x,t) + a\partial_x u(x,t) &= f(x,t) \quad \text{in } (0,L)\times(0,T], \\ u(x,0) &= u_0(x) \quad \text{in } (0,L), \\ u(0,t) &= g(t) \quad \text{in } (0,T], \end{aligned}$$

where $a > 0$ is the transport speed. For $f = 0$, the solution is of the form $u(x,t) = G(x - at)$ as one can see by inspection:

$$\partial_t u(x,t) + a\partial_x u(x,t) = G'(x-at)(-a) + aG'(x-at) = 0.$$



Numerical Approximations

Introduce the finite difference operators

$$D^+_{\Delta x} u^n_j := \frac{u^n_{j+1}-u^n_j}{\Delta x}, \quad D^-_{\Delta x} u^n_j := \frac{u^n_j-u^n_{j-1}}{\Delta x},$$
$$D^+_{\Delta t} u^n_j := \frac{u^{n+1}_j-u^n_j}{\Delta t}, \quad D^-_{\Delta t} u^n_j := \frac{u^n_j-u^{n-1}_j}{\Delta t}.$$

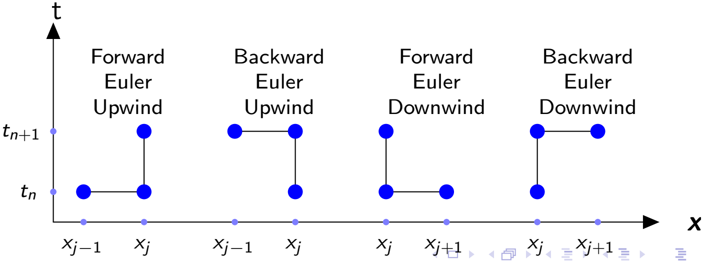
Four possibilities to discretize the transport equation:

Forward Euler Upwind: $(D^+_{\Delta t} + aD^-_{\Delta x})u^n_j = f(x_j, t_n)$

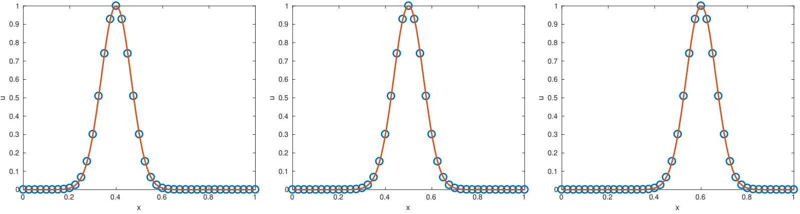
Backward Euler Upwind: $(D^-_{\Delta t} + aD^-_{\Delta x})u^{n+1}_j = f(x_j, t_{n+1})$

Forward Euler Downwind: $(D^+_{\Delta t} + aD^+_{\Delta x})u^n_j = f(x_j, t_n)$

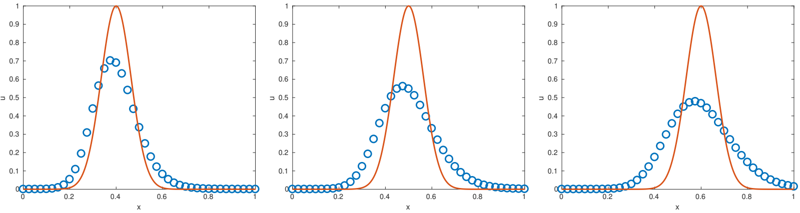
Backward Euler Downwind: $(D^-_{\Delta t} + aD^+_{\Delta x})u^{n+1}_j = f(x_j, t_{n+1})$



Which scheme should one use ?

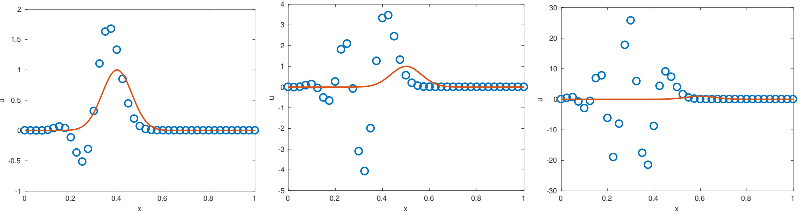


Forward Euler Upwind

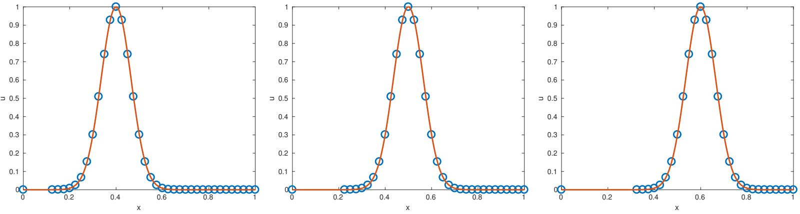


Backward Euler Upwind

Which scheme should one use ?

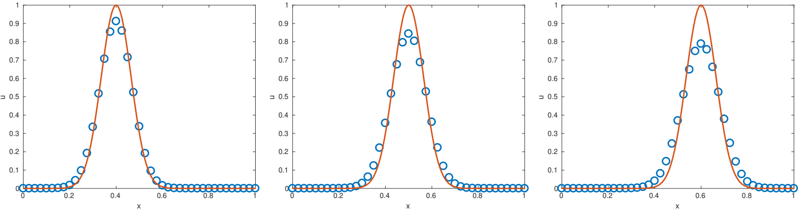


Forward Euler Downwind

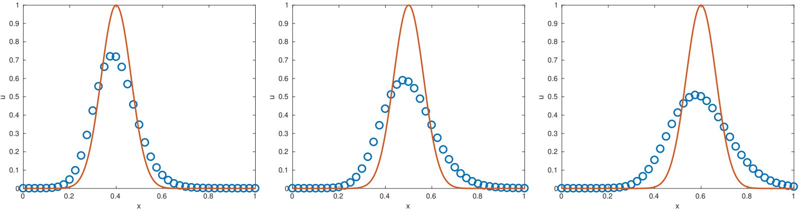


Backward Euler Downwind

Using a smaller time step

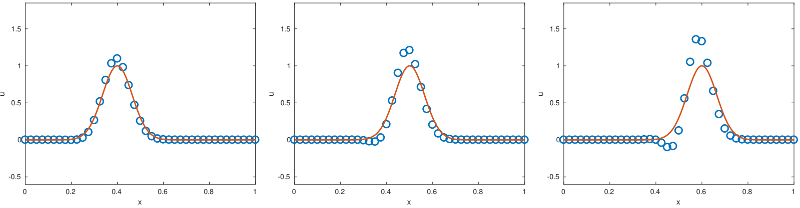


Forward Euler Upwind

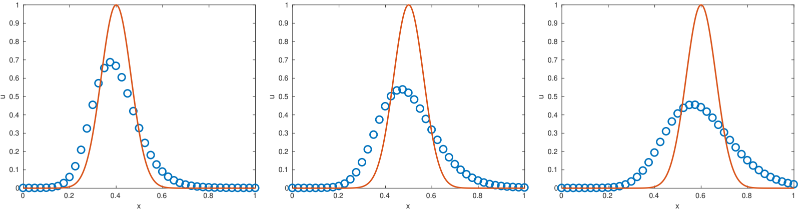


Backward Euler Upwind

Using a larger time step



Forward Euler Upwind

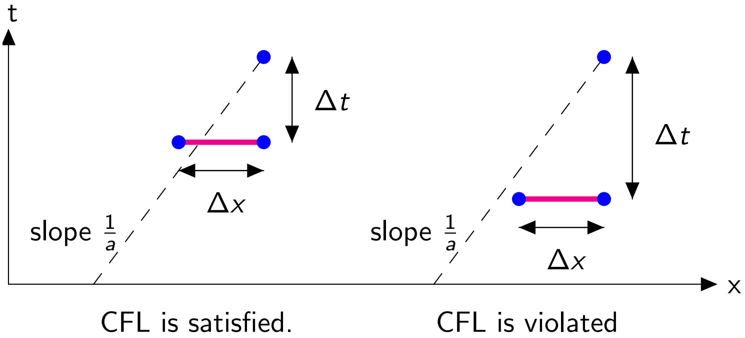


Backward Euler Upwind

The CFL Condition

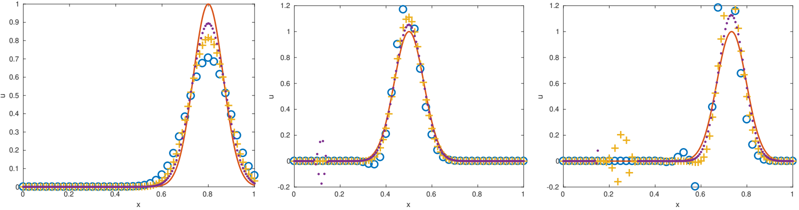
Courant, Friedrich and Levy (1928): "If the domain of dependence of the numerical scheme does not include the domain of dependence of the exact solution, then it can not converge."

Example: Forward Euler Upwind



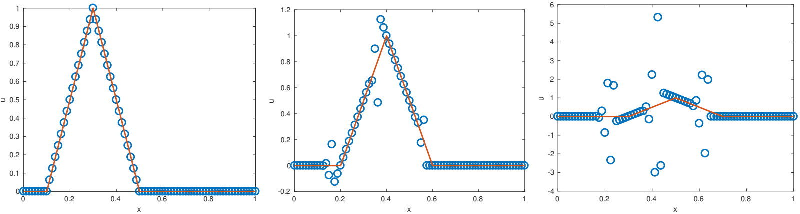
$$\frac{\Delta t}{\Delta x} \leq \frac{1}{a} \quad \text{which means} \quad a\frac{\Delta t}{\Delta x} \leq 1$$

What happens if the CLF is violated?

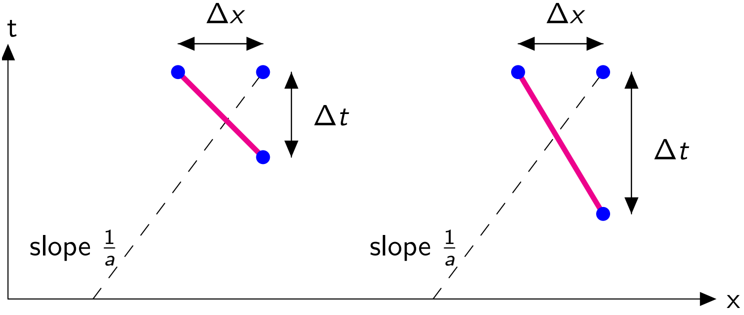


Forward Euler Upwind under mesh refinement (left satisfying the CLF, middle and right not)

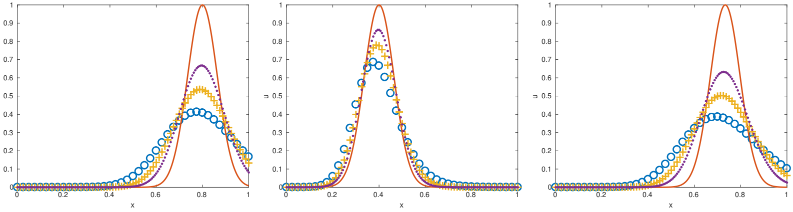
Below with less regular data not satisfying the CFL



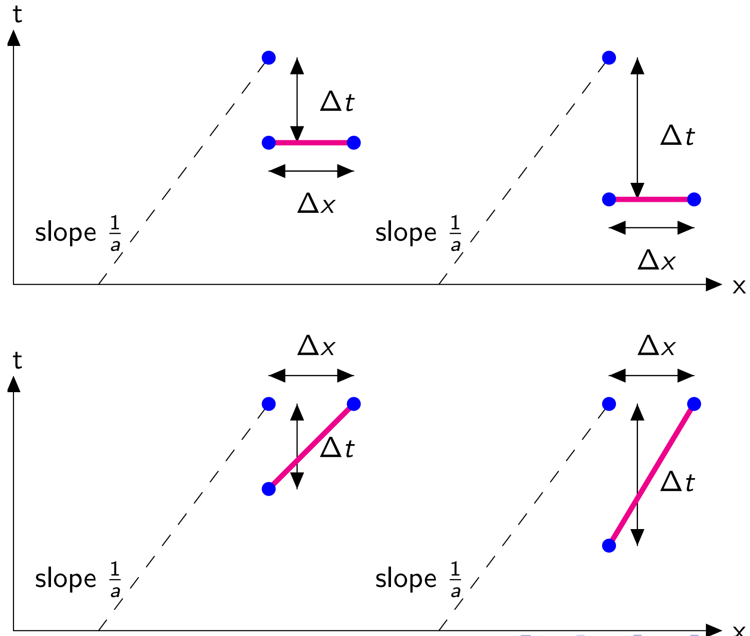
The CFL for Backward Euler Upwind



The domain of dependence of this scheme always includes the characteristics, so the scheme should always work:



What happens to downwind schemes ?



Advection Reaction Diffusion Equations

Combinations of the terms we have seen in the earlier partial differential equations

$$
\begin{array}{rcll}
\partial_t u + \boldsymbol{a} \cdot \nabla u & = & \nu \Delta u - \eta u + f & \text{in } \Omega \times (0, T], \\
u(\boldsymbol{x}, 0) & = & u_0(\boldsymbol{x}) & \text{in } \Omega, \\
u(\boldsymbol{x}, t) & = & g(\boldsymbol{x}, t) & \text{on } \partial\Omega \times (0, T]
\end{array}
$$

$u = u(\boldsymbol{x}, t)$ is the solution sought

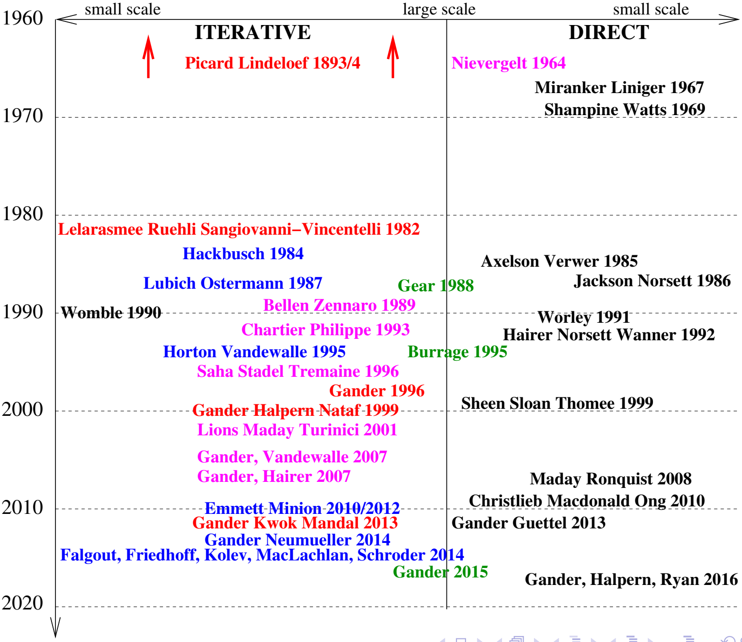
$\boldsymbol{a} = \boldsymbol{a}(\boldsymbol{x}, t)$ represents the transport direction, which could even depend on the solution u itself (e.g. in the case of the Navier Stokes equations)

$\nu > 0$ is the diffusion coefficient

η is the reaction coefficient of the linear reaction term in this model

$f(\boldsymbol{x}, t)$ is a source term, which could also depend on the solution u

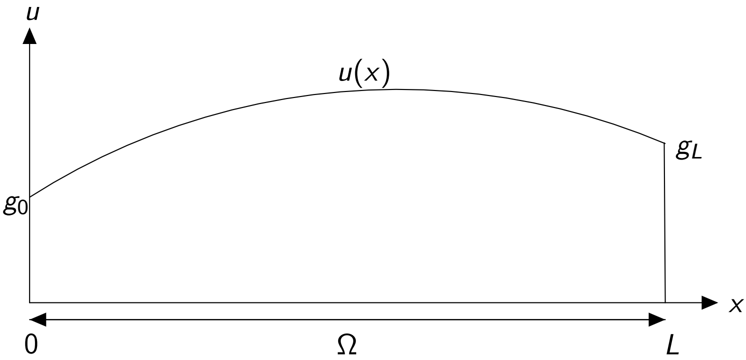
Getting back to the 4 Classes of Methods



Methods based on multiple shooting

Model problem: a semilinear partial differential equation in one spatial dimension:

$$\begin{aligned} \partial_{xx} u(x) &= f(u(x)) \quad \text{in } (0, L), \\ u(0) &= g_0, \\ u(L) &= g_L, \end{aligned}$$

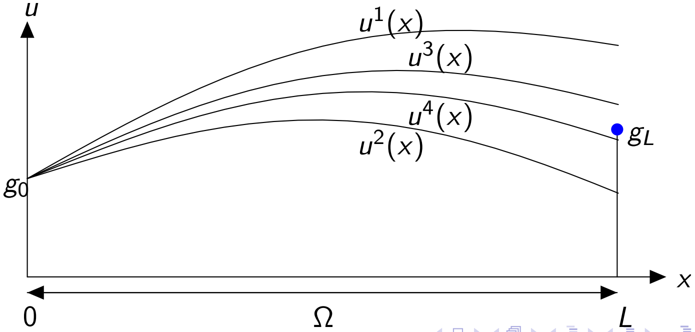


Idea of Shooting Methods

Transform the semilinear boundary value problem into a simpler initial value problem

$$\begin{aligned} \partial_{xx} u(x) &= f(u(x)) \quad \text{in } (0, L), \\ u(0) &= g_0, \\ \partial_x u(0) &= U, \end{aligned}$$

which one can easily solve numerically using for example Forward or Backward Euler. Choice of the shooting parameter U?



Systematic Shooting

We have to solve the non-linear equation

$$F(U) := u(L, U) - g_L = 0.$$

This is usually done using Newton's method

$$U^{k+1} = U^k - (F'(U^k))^{-1} F(U^k).$$

To calculate $F'(U)$, we just differentiate the ODE,

$$\begin{aligned} \partial_{xx} u(x, U) &= f(u(x, U)) \quad \text{in } (0, L), \\ u(0, U) &= g_0, \\ \partial_x u(0, U) &= U \end{aligned}$$

and obtain for $F'(U) = u_U(x, U)$ the linear ODE

$$\begin{aligned} \partial_{xx} u_U(x, U) &= f'(u(x, U)) u_U(x, U) \quad \text{in } (0, L), \\ u_U(0, U) &= 0, \\ \partial_x u_U(0, U) &= I \end{aligned}$$

However the problems we are interested in for time parallelization are already initial value problems, there is no target!

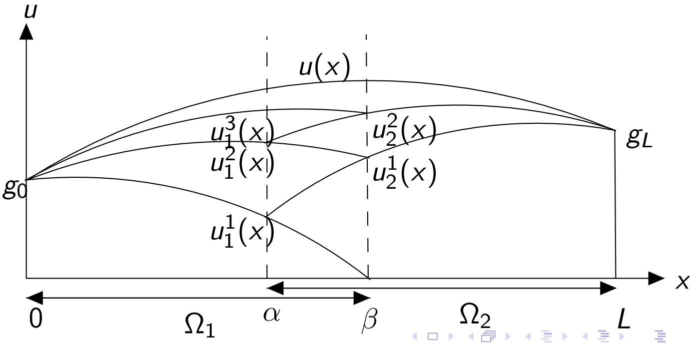
Methods based on domain decomposition

Example: Schwarz domain decomposition for

$$\begin{aligned}\partial_{xx}u(x) &= f(u(x)) \quad \text{in } \Omega := (0,L),\\ u(0) &= g_0,\\ u(L) &= g_L,\end{aligned}$$

Decompose Ω into $\Omega_1 := (0,\beta)$ and $\Omega_2 := (\alpha, L)$, and iterate

$$\begin{aligned}\partial_{xx}u_1^k(x) &= f(u_1^k(x)) \quad \text{in } \Omega_1, & \partial_{xx}u_2^k(x) &= f(u_2^k(x)) \quad \text{in } \Omega_2,\\ u_1^k(\beta) &= u_2^{k-1}(\beta), & u_2^k(\alpha) &= u_1^k(\alpha),\end{aligned}$$



Methods based on multigrid

Consider for simplicity a linear model problem

$$\begin{aligned} \partial_{xx} u(x) &= f(x) \quad \text{in } (0, L), \\ u(0) &= 0, \\ u(L) &= 0. \end{aligned}$$

Discretization with centered finite differences:

$$A\boldsymbol{u} := \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_J \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_J \end{pmatrix} =: \boldsymbol{f},$$

Introduce the matrix splitting $A = L + D + U$, $D = \frac{-2}{h^2} I$ and

$$L = \frac{1}{h^2}\begin{pmatrix} & & \\ 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}, \quad U = \frac{1}{h^2}\begin{pmatrix} & 1 & \\ & & \ddots \\ & & & 1 \\ & & & \end{pmatrix}.$$

Smoothers for multigrid

Jacobi stationary iterative method

$$D\boldsymbol{u}^{k+1} = -(L+U)\boldsymbol{u}^k + \boldsymbol{f},$$

Gauss-Seidel stationary iterative method

$$(L+D)\boldsymbol{u}^{k+1} = -U\boldsymbol{u}^k + \boldsymbol{f}.$$

Edouard Stiefel (1952):

> *"sodass der positive Residualberg mit dem Löffel statt mit einer Baggermaschine abgetragen wird!"*

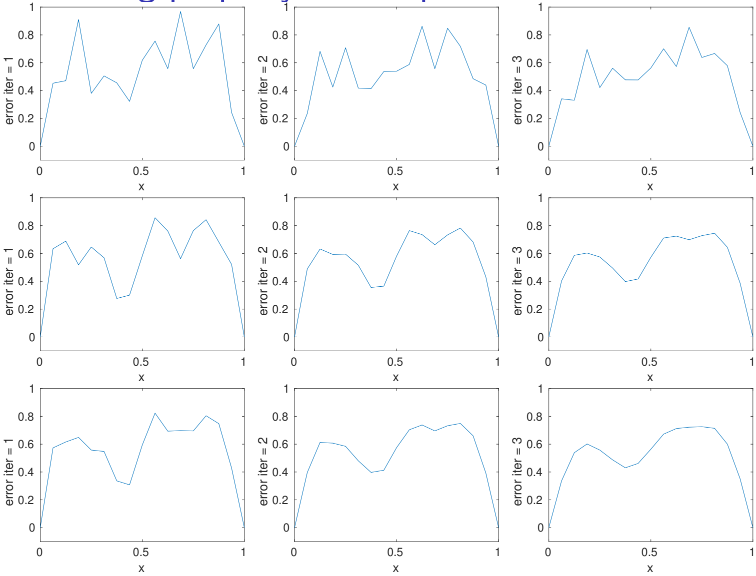
Also for damped Jacobi, obtained from

$$\boldsymbol{u}^{k+1} = -D^{-1}(L+U)\boldsymbol{u}^k + D^{-1}\boldsymbol{f} = \boldsymbol{u}^k + D^{-1}(\boldsymbol{f} - A\boldsymbol{u}^k),$$

and then adding the damping parameter ω,

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \omega D^{-1}(\boldsymbol{f} - A\boldsymbol{u}^k).$$

Smoothing property of damped Jacobi

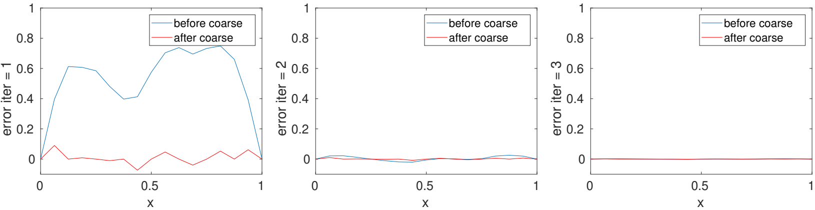


Error after one, two and three damped Jacobi steps.
$\omega = 1$ (top), $\omega = \frac{2}{3}$ (middle), $\omega = \frac{1}{2}$ (bottom)

Multigrid idea: correct on coarser grid

For $k = 0, 1, \ldots$ compute

$$\begin{aligned}
\boldsymbol{u}^{k+\frac{1}{3}} &= S(\boldsymbol{f}, \boldsymbol{u}^k, \nu_1)\,; && \text{\% presmoothing}\\
\boldsymbol{u}^{k+\frac{2}{3}} &= \boldsymbol{u}^{k+\frac{1}{3}} + PA_c^{-1}R(\boldsymbol{f} - A\boldsymbol{u}^{k+\frac{1}{3}}) && \text{\% coarse correction}\\
\boldsymbol{u}^{k+1} &= S(\boldsymbol{f}, \boldsymbol{u}^{k+\frac{2}{3}}, \nu_2)\,; && \text{\% postsmoothing}
\end{aligned}$$



Error before and after the coarse correction for the first, second and third two grid iterations with two damped Jacobi steps used as a presmoother with damping parameter $\omega = \frac{2}{3}$.

For multigrid, one uses this idea recursively for A_c^{-1}!

Direct time parallel methods

Take for example

$$\partial_t u = f(t,u) \text{ in } (0,T], \quad u(0) = u_0$$

Backward Euler discretization

$$u_{n+1} = u_n + \Delta t f(t_n, u_{n+1}),$$

normally solved by forward substitution

$$\begin{array}{rcl} u_1 & = & u_0 + \Delta t f(t_1, u_1) \\ u_2 & = & u_1 + \Delta t f(t_2, u_2) \\ u_3 & = & u_2 + \Delta t f(t_3, u_3) \\ \vdots & \vdots & \vdots \end{array}$$

All at once system:

$$\boldsymbol{F}(\boldsymbol{u}) = \begin{pmatrix} u_1 - u_0 - \Delta t f(t_1, u_1) \\ u_2 - u_1 - \Delta t f(t_2, u_2) \\ \vdots \\ u_N - u_{N-1} - \Delta t f(t_N, u_N) \end{pmatrix} = 0.$$

Direct solution without iteration

Such non-linear systems must in general be solved by iterative methods, for example Newton's method.

In the linear case, when $f(t,u) = au + g(t)$, the all at once system becomes

$$\begin{pmatrix} 1-\Delta ta & & & \\ -1 & 1-\Delta ta & & \\ & \ddots & \ddots & \\ & & -1 & 1-\Delta ta \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} \Delta tg(t_1) + u_0 \\ \Delta tg(t_2) \\ \vdots \\ \Delta tg(t_N) \end{pmatrix}$$

Direct solution by Gaussian elimination ? Just forward substitution again!

Direct time parallel methods solve such systems faster than by forward substitution using many processor, without iteration!