THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 10: Systems of ODEs and PDEs

Sergei V. Kalinin

Previous lecture

- Differentiating functions: numerical precision
- Ordinary differential equations (ODEs)
  - Where do equations come from
  - Initial and boundary value problems
- Solving ODEs:
  - Euler methods
  - Runge-Kutta methods
  - Instability of forward schemes
- Systems of ODEs
- Partial differential equations (PDEs)

Systems of Ordinary Differential Equations

System of N first-order ODE

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(x_1, \dots, x_N, t), \\
\frac{dx_2}{dt} &= f_2(x_1, \dots, x_N, t), \\
&\dots \\
\frac{dx_N}{dt} &= f_N(x_1, \dots, x_N, t).
\end{aligned}$$

Vector notation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t).$$

- all the methods we covered have the same structure when applied for systems of ODEs
- apply component by component

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Systems of Ordinary Differential Equations

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t).$$

- Euler method

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\,\mathbf{f}[\mathbf{x}(t), t]\ .$$

- RK2

$$\begin{aligned}
\mathbf{k_1} &= h\,\mathbf{f}(\mathbf{x}, t), \\
\mathbf{k_2} &= h\,\mathbf{f}(\mathbf{x} + \mathbf{k_1}/2, t + h/2), \\
\mathbf{x}(t+h) &= \mathbf{x}(t) + \mathbf{k_2}\ .
\end{aligned}$$

- RK4

$$\begin{aligned}
\mathbf{k_1} &= h\,\mathbf{f}(\mathbf{x}, t), \\
\mathbf{k_2} &= h\,\mathbf{f}(\mathbf{x} + \mathbf{k_1}/2, t + h/2), \\
\mathbf{k_3} &= h\,\mathbf{f}(\mathbf{x} + \mathbf{k_2}/2, t + h/2), \\
\mathbf{k_4} &= h\,\mathbf{f}(\mathbf{x} + \mathbf{k_3}, t + h), \\
\mathbf{x}(t+h) &= \mathbf{x}(t) + \frac{1}{6}(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4})\ .
\end{aligned}$$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Systems of Ordinary Differential Equations

```python
def ode_euler_multi(f, x0, t0, h, nsteps):
    """Multi-dimensional version of the Euler method.
    """

    t = np.zeros(nsteps + 1)
    x = np.zeros((len(t), len(x0)))
    t[0] = t0
    x[0,:] = x0
    for i in range(0, nsteps):
        t[i + 1] = t[i] + h
        x[i + 1,:] = ode_euler_step(f, x[i], t[i], h)
    return t,x
```

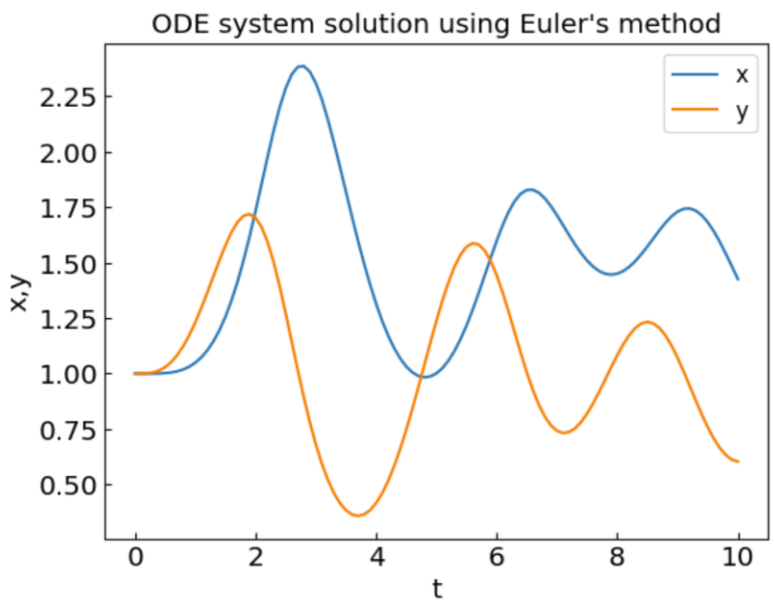
```python
def ode_rk2_multi(f, x0, t0, h, nsteps):
    """Multi-dimensional version of the RK2 method.
    """

    t = np.zeros(nsteps + 1)
    x = np.zeros((len(t), len(x0)))
    t[0] = t0
    x[0,:] = x0
    for i in range(0, nsteps):
        t[i + 1] = t[i] + h
        x[i + 1] = ode_rk2_step(f, x[i], t[i], h)
    return t,x
```

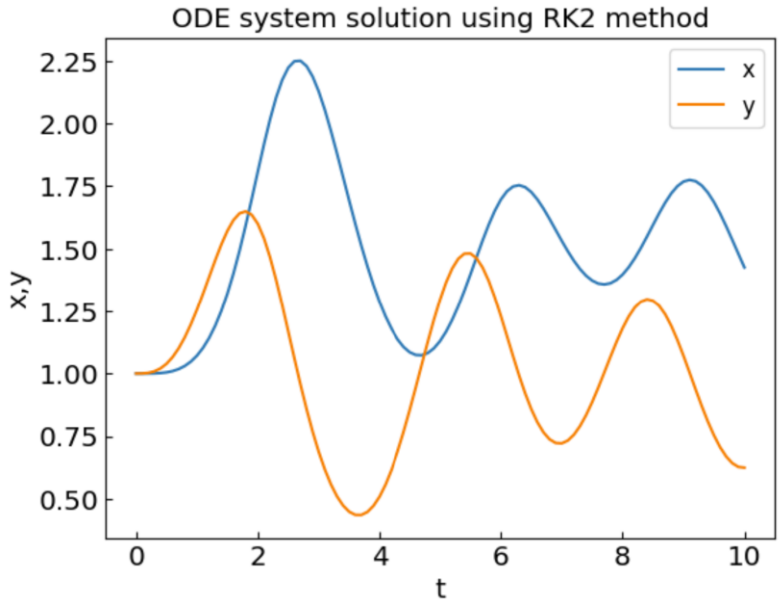
```python
def ode_rk4_multi(f, x0, t0, h, nsteps):
    """Multi-dimensional version of the RK4 method.
    """

    t = np.zeros(nsteps + 1)
    x = np.zeros((len(t), len(x0)))
    t[0] = t0
    x[0,:] = x0
    for i in range(0, nsteps):
        t[i + 1] = t[i] + h
        x[i + 1] = ode_rk4_step(f, x[i], t[i], h)
    return t,x
```

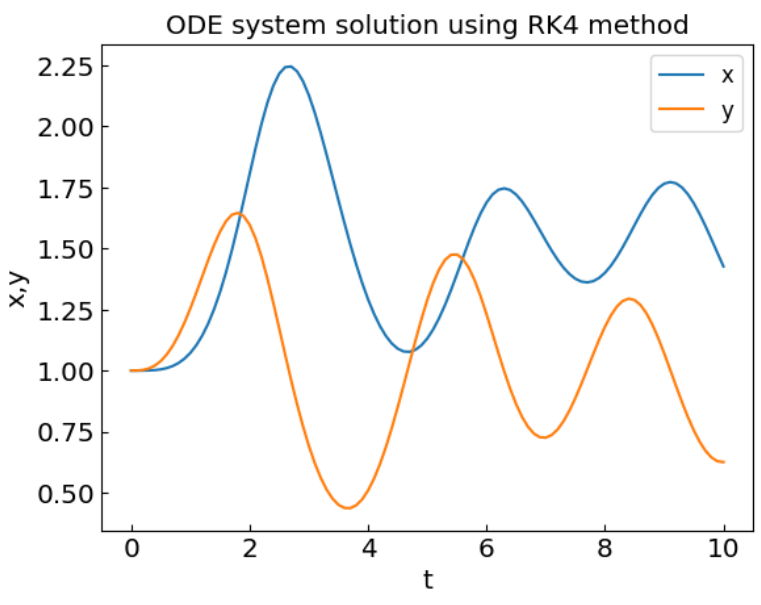
From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Systems of Ordinary Differential Equations: Example

$$\frac{dx}{dt} = xy - x,$$
$$\frac{dy}{dt} = y - xy + (\sin t)^2$$







From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Systems of 2nd-order ODEs

Newton/Lagrange equations of motion are 2nd order systems of ODE $m_i \frac{d^2 x_i}{dt^2} = F_i(\{x_j\}, \{dx_j/dt\}, t)$

A system of *N* second-order ODEs

$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{f}(\mathbf{x}, d\mathbf{x}/dt, t),$$

can be written as a system of *2N* first-order ODEs by denoting $\frac{d\mathbf{x}}{dt} = \mathbf{v}$

$$\frac{d\mathbf{x}}{dt} = \mathbf{v},$$
$$\frac{d\mathbf{v}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{v}, t),$$

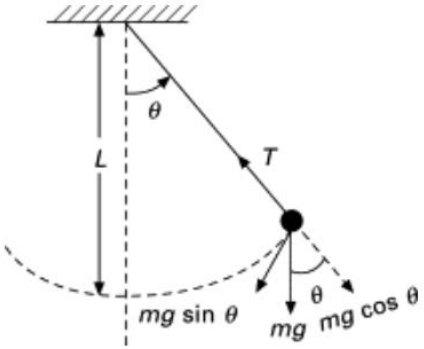
and can be solved for **x**(t) and **v**(t) using standard methods

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Example: Simple pendulum

The equation of motion for a simple pendulum reads

$$mL\frac{d^2\theta}{dt^2} = -mg\sin\theta.$$



denote $\frac{d\theta}{dt} = \omega$ and write a system of two first-order ODE

$$\frac{d\theta}{dt} = \omega,$$
$$\frac{d\omega}{dt} = -\frac{g}{L}\sin\theta\,,$$

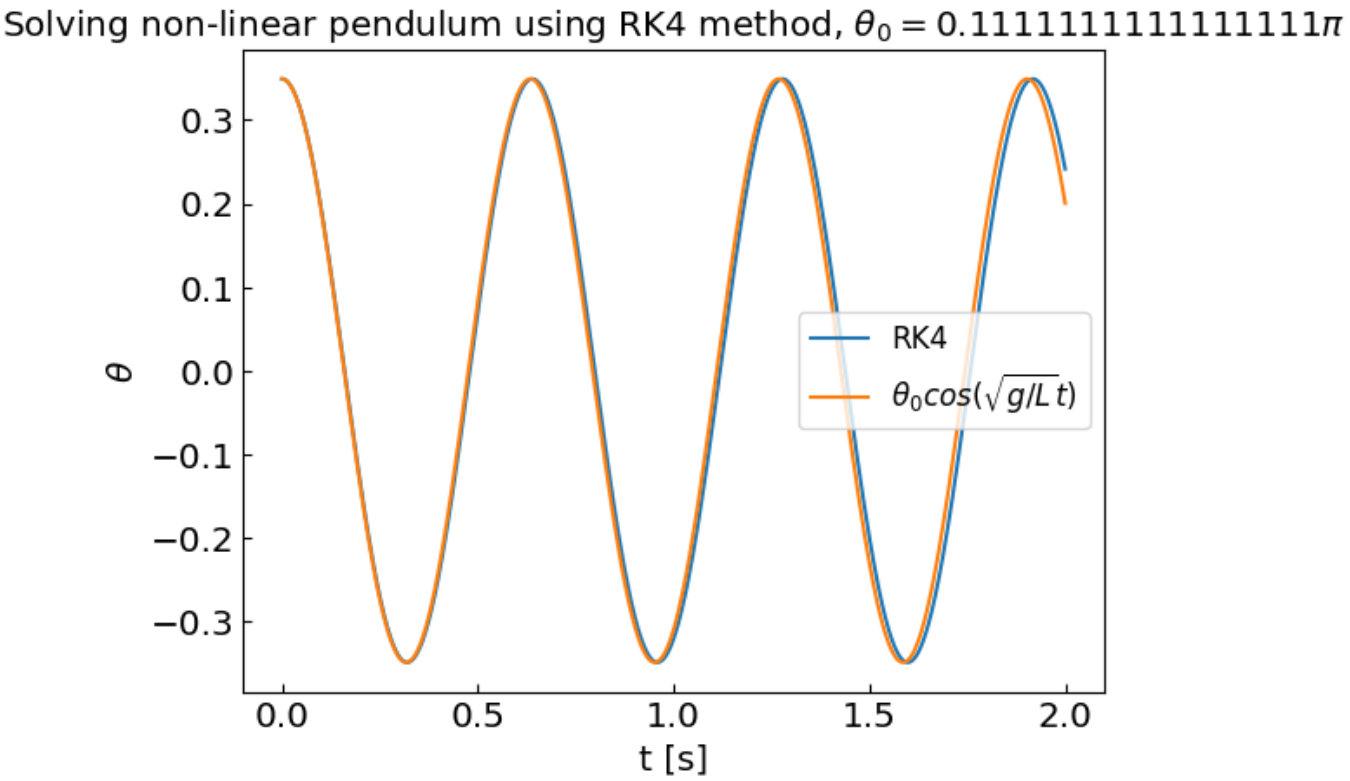
For small angles $\sin\theta \approx \theta$, an analytic solution exists

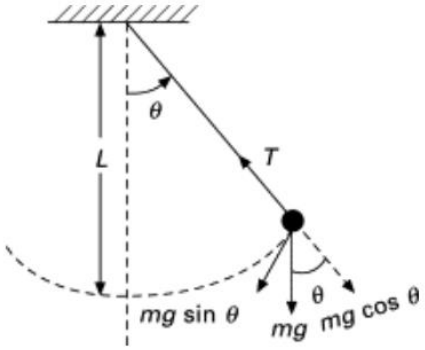
$$\theta(t) \approx \theta_0 \cos\left(\sqrt{\frac{g}{L}}t + \phi\right)$$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Example: Simple pendulum

Initially at rest at angle $\theta_0 = 20° \approx 0.111\pi$ L=0.1 m, g=9.81 m/s^2



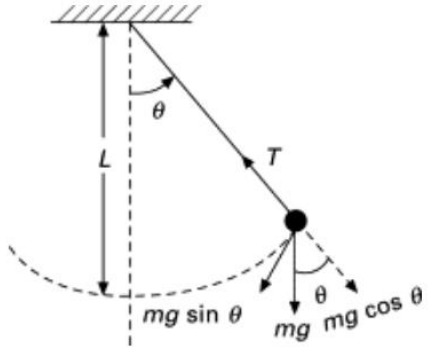


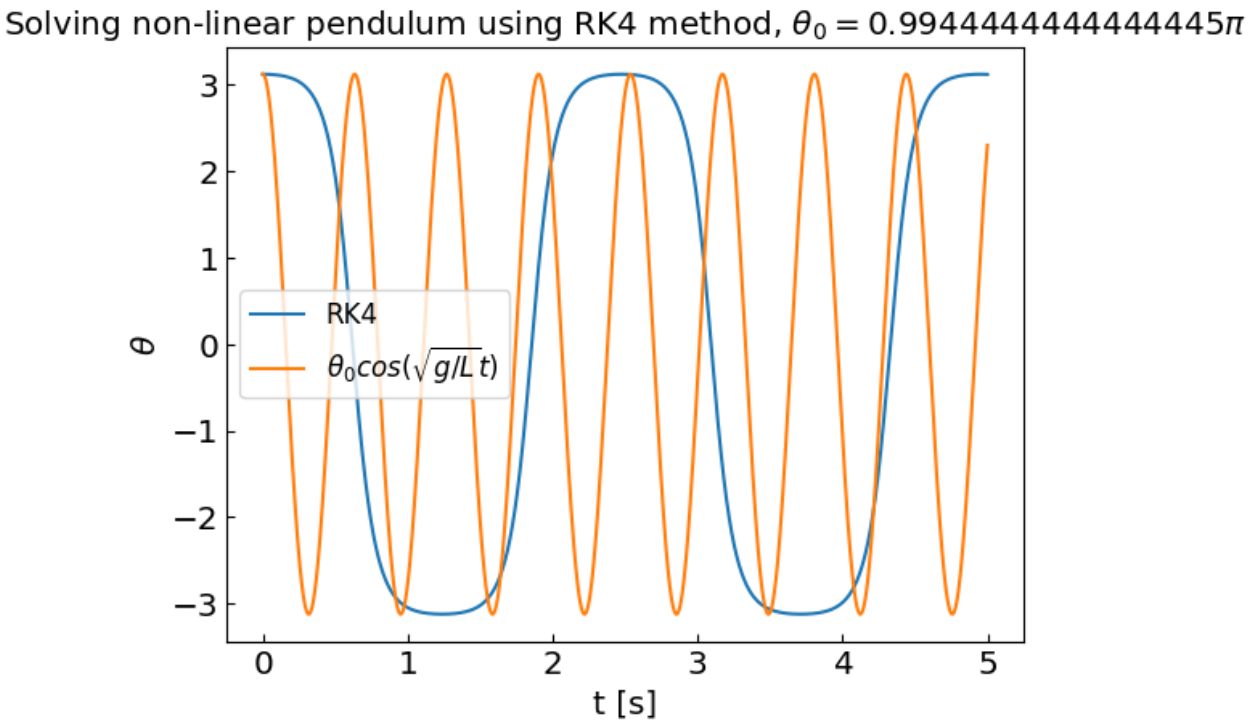
Linear regime at small angles

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Example: Simple pendulum

Initially at rest at angle $\theta_0 = 179° \approx 0.994\pi$ L=0.1 m, g=9.81 m/s^2



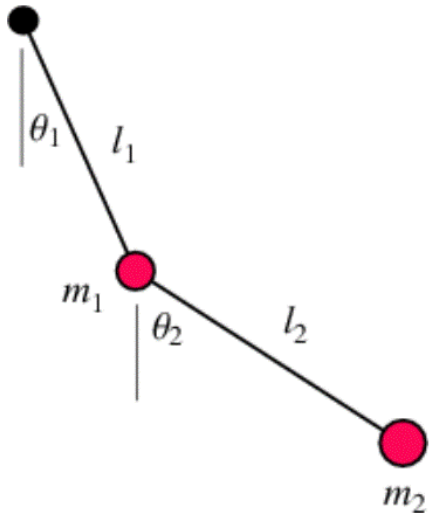


Non-linear regime at large angles, approximate analytic solution fails

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Double pendulum

Double pendulum is the simplest system exhibiting chaotic motion – *deterministic chaos*



Two degrees of freedom: the displacement angles θ_1 and θ_2.

$$
\begin{aligned}
x_1 &= l_1 \sin(\theta_1), \\
y_1 &= -l_1 \cos(\theta_1), \\
x_2 &= l_1 \sin(\theta_1) + l_2 \sin(\theta_2), \\
y_2 &= -l_1 \cos(\theta_1) - l_2 \cos(\theta_2).
\end{aligned}
$$

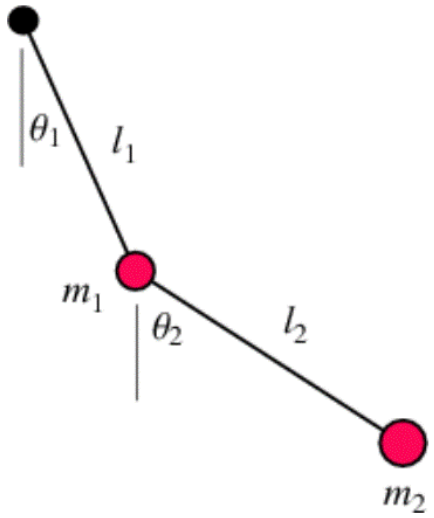
$$T = \frac{m_1 \dot{x}_1^2}{2} + \frac{m_2 \dot{x}_2^2}{2} = \frac{1}{2} m_1 l_1^2 \dot{\theta}_1 + \frac{1}{2} m_2 \left[l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2), \right]$$ kinetic energy

$$V = m_1 g y_1 + m_2 g y_2 = -(m_1 + m_2) g l_1 \cos(\theta_1) - m_2 g l_2 \cos(\theta_2).$$ potential energy

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Double pendulum

Double pendulum is the simplest system exhibiting chaotic motion – *deterministic chaos*



Two degrees of freedom: the displacement angles θ_1 and θ_2.

$$
\begin{aligned}
x_1 &= l_1 \sin(\theta_1), \\
y_1 &= -l_1 \cos(\theta_1), \\
x_2 &= l_1 \sin(\theta_1) + l_2 \sin(\theta_2), \\
y_2 &= -l_1 \cos(\theta_1) - l_2 \cos(\theta_2).
\end{aligned}
$$

$$
T = \frac{m_1 \dot{x}_1^2}{2} + \frac{m_2 \dot{x}_2^2}{2} = \frac{1}{2} m_1 l_1^2 \dot{\theta}_1 + \frac{1}{2} m_2 \left[l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2), \right]
$$

kinetic energy

$$
V = m_1 g y_1 + m_2 g y_2 = -(m_1 + m_2) g l_1 \cos(\theta_1) - m_2 g l_2 \cos(\theta_2).
$$

potential energy

The Lagrange equations of motion read

$$
\begin{aligned}
(m_1 + m_2) l_1 \ddot{\theta}_1 + m_2 l_2 \cos(\theta_1 - \theta_2) \ddot{\theta}_2 &= -m_2 l_1 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) - (m_1 + m_2) g \sin(\theta_1), \\
m_2 l_1 \cos(\theta_1 - \theta_2) \ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 &= m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) - m_2 g \sin(\theta_2).
\end{aligned}
$$

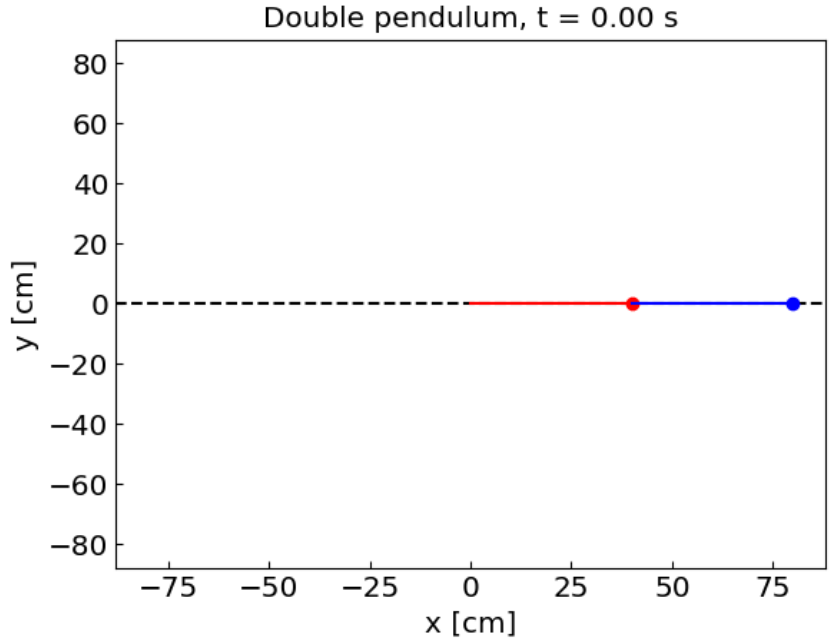
This is a system of two linear equation for $\ddot{\theta}_{1,2}$ that can be solved easily.

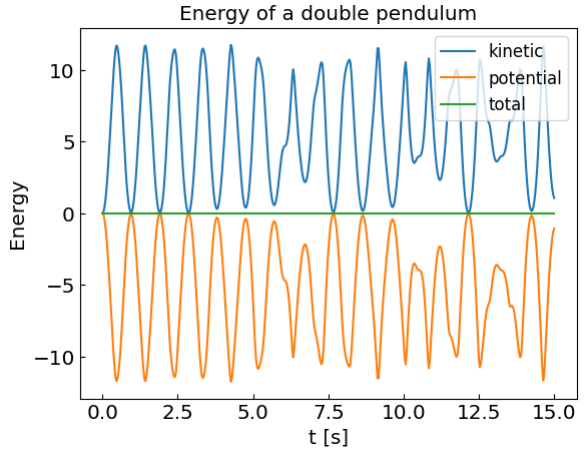
From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Double pendulum

```python
g = 9.81
l1 = 0.4
l2 = 0.4
m1 = 1.0
m2 = 1.0

def fdoublependulum(xin, t):
    global f_evaluations
    f_evaluations += 1

    theta1 = xin[0]
    theta2 = xin[1]
    omega1 = xin[2]
    omega2 = xin[3]
    a1 = (m1 + m2)*l1
    b1 = m2*l2*np.cos(theta1 - theta2)
    c1 = m2*l2*omega2*omega2*np.sin(theta1 - theta2) + (m1 + m2)*g*np.sin(theta1)
    a2 = m2*l1*np.cos(theta1 - theta2)
    b2 = m2*l2;
    c2 = -m2*l1*omega1*omega1*np.sin(theta1 - theta2) + m2*g*np.sin(theta2) # + k
    domega1 = - ( c2/b2 - c1/b1 ) / ( a2/b2 - a1/b1 )
    domega2 = - ( c2/a2 - c1/a1 ) / ( b2/a2 - b1/a1 )
    return np.array([omega1,
                     omega2,
                     domega1,
                     domega2
                    ])
```

```python
sol = bulirsch_stoer(fpendulum, x0, t0, 1, tend, eps, error_definition_pendulum, 10)
```

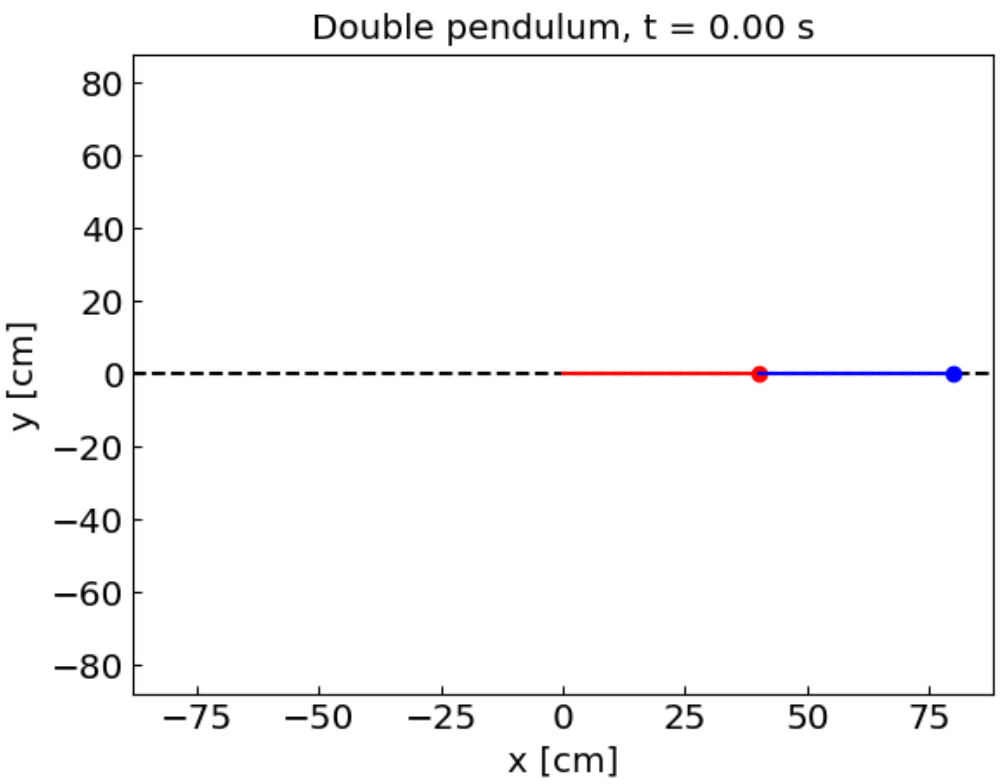




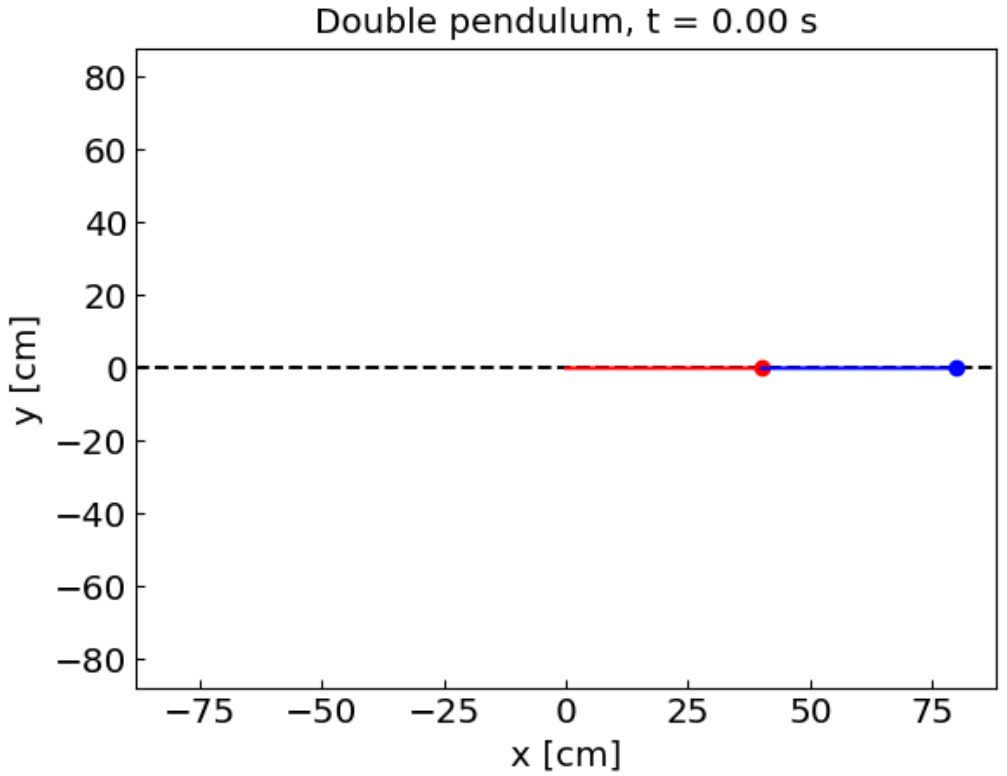
From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Double pendulum: chaotic behavior

$$\theta_1^0 = \theta_2^0 = \pi/2$$



$$\theta_1^0 = \theta_2^0 = \pi/2 + 10^{-4}$$

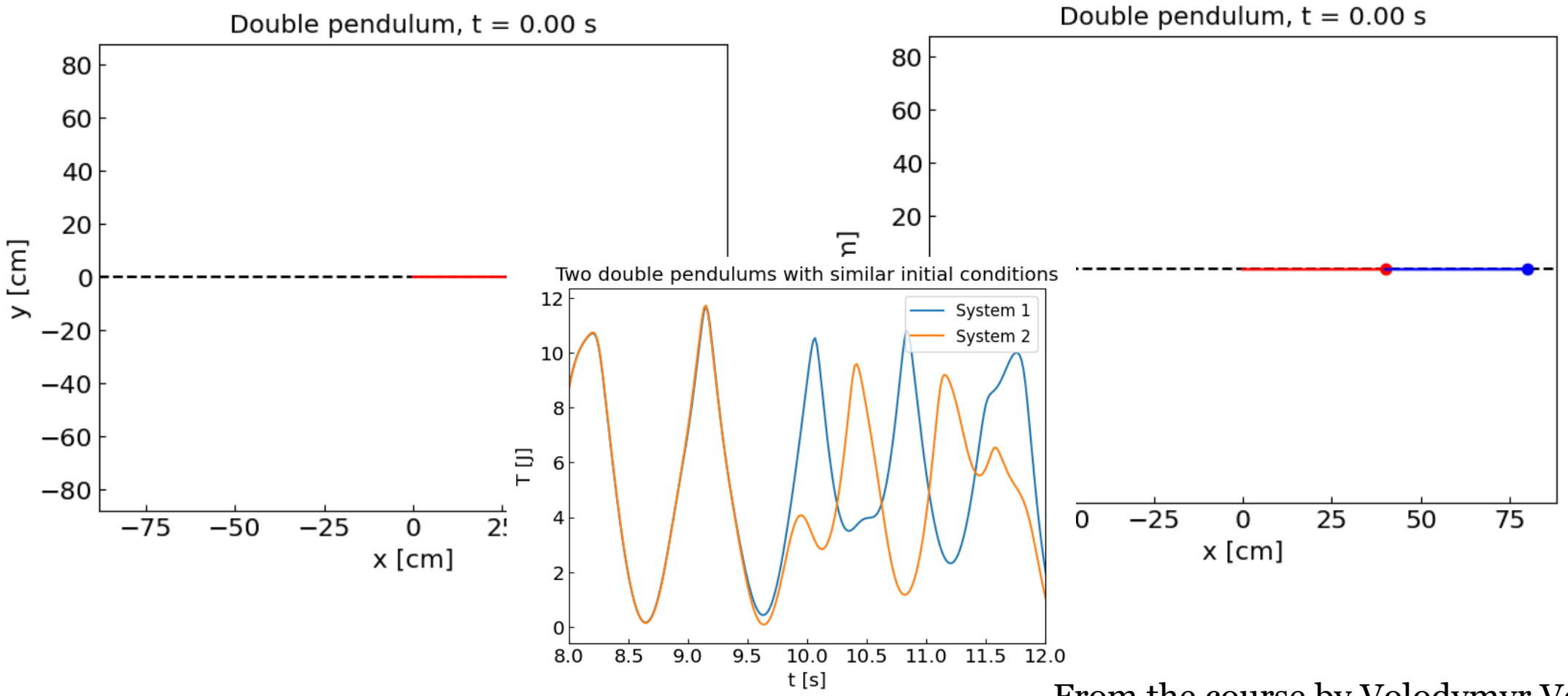


From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Double pendulum: chaotic behavior

$\theta_1^0 = \theta_2^0 = \pi/2$

$\theta_1^0 = \theta_2^0 = \pi/2 + 10^{-4}$



From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Three-body problem

Three stars interacting through gravitational force

The equations of motion are

$$\frac{d^2\mathbf{r}_1}{dt} = Gm_2\frac{\mathbf{r}_2-\mathbf{r}_1}{|\mathbf{r}_2-\mathbf{r}_1|^3} + Gm_3\frac{\mathbf{r}_3-\mathbf{r}_1}{|\mathbf{r}_3-\mathbf{r}_1|^3},$$
$$\frac{d^2\mathbf{r}_2}{dt} = Gm_1\frac{\mathbf{r}_1-\mathbf{r}_2}{|\mathbf{r}_1-\mathbf{r}_2|^3} + Gm_3\frac{\mathbf{r}_3-\mathbf{r}_2}{|\mathbf{r}_3-\mathbf{r}_2|^3},$$
$$\frac{d^2\mathbf{r}_3}{dt} = Gm_1\frac{\mathbf{r}_1-\mathbf{r}_3}{|\mathbf{r}_1-\mathbf{r}_3|^3} + Gm_2\frac{\mathbf{r}_2-\mathbf{r}_3}{|\mathbf{r}_2-\mathbf{r}_3|^3}.$$

Name	Mass	x	y
Star 1	150.	3	1
Star 2	200.	-1	-2
Star 3	250.	-1	1

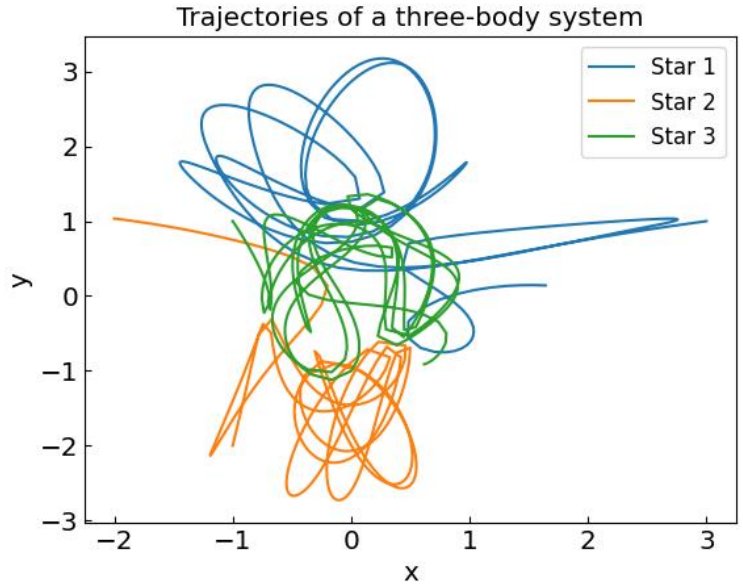
Cannot be solved analytically!

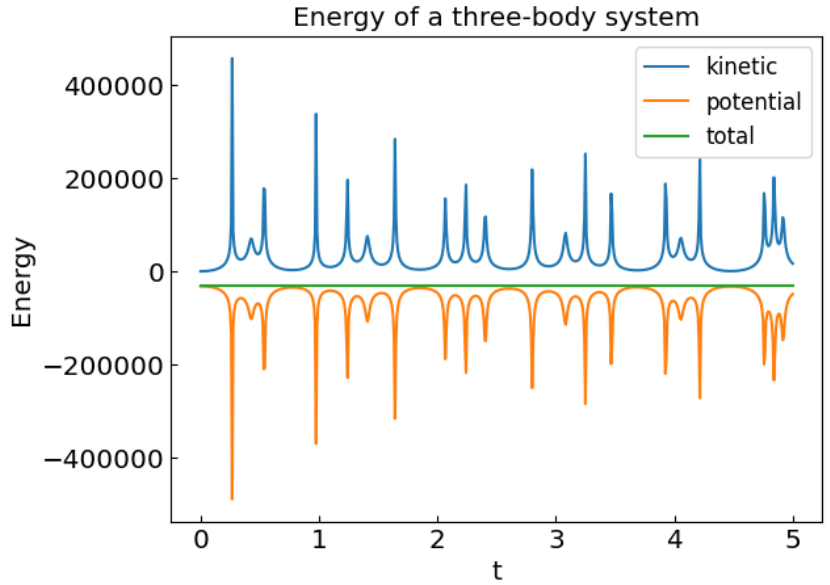
Take *G = 1* (dimensionless)

Initially at rest and move in plane z = 0

Initial coordinates: $\mathbf{r}_1 = (3,1)$, $\mathbf{r}_2 = (-1,-2)$, $\mathbf{r}_3 = (-1,1)$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics



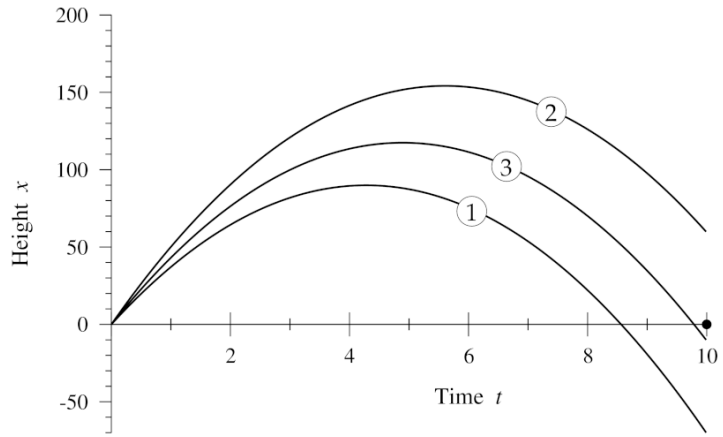


Boundary value problems and the shooting method

Sometimes we have equations, such as vertically thrown object

$$\frac{dx}{dt} = v,$$
$$\frac{dv}{dt} = -g,$$

and boundary conditions, e.g. x(0) = 0 and x(10) = 0 instead of initial conditions v(0) = v_0.



How to solve this problem?

In the shooting method one takes trial values of v_0 until finding the one where the solution satisfies the boundary condition x(10) = 0.

To find v_0 efficiently one combines numerical ODE method (e.g. RK4) with non-linear equation solver (e.g. bisection method).

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Partial differential equations

Describe functions of more than one variable

In physics, this commonly corresponds to fields $\phi(x, y, z)$

Examples:

- Electrostatic potential $\phi(x, y, z)$ (Poisson's equation)

$$\Delta\phi(x, y, z) = -\frac{\rho(x, y, z)}{\epsilon_0}$$

- Density or temperature profiles (diffusion/heat equation)

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} = D\Delta u(\mathbf{x}, t)$$

- Displacement (amplitude) profile (wave equation)

$$\frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} = c^2\Delta u(\mathbf{x}, t)$$

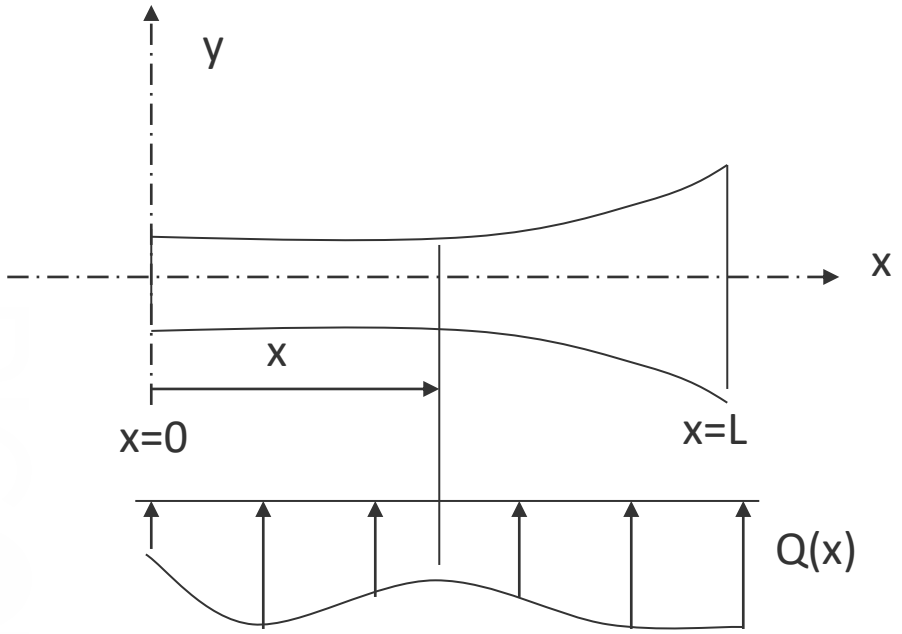
- Fluid dynamical fields (flow velocity) -- e.g. Navier-Stokes equations

$$\frac{\partial}{\partial t}(\rho\,\mathbf{u}) + \nabla\cdot(\rho\,\mathbf{u}\otimes\mathbf{u}) = -\nabla p + \nabla\cdot\boldsymbol{\tau} + \rho\,\mathbf{g}$$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

From ODEs to PDEs

Heat conduction in a fin



A(x) - cross section at x
Q(x) - heat input per unit length per unit time [J/sm]
k(x) - thermal conductivity [J/^{o}C ms]
T(x) - temperature of the fin at x

Differential equation governing the response of the fin

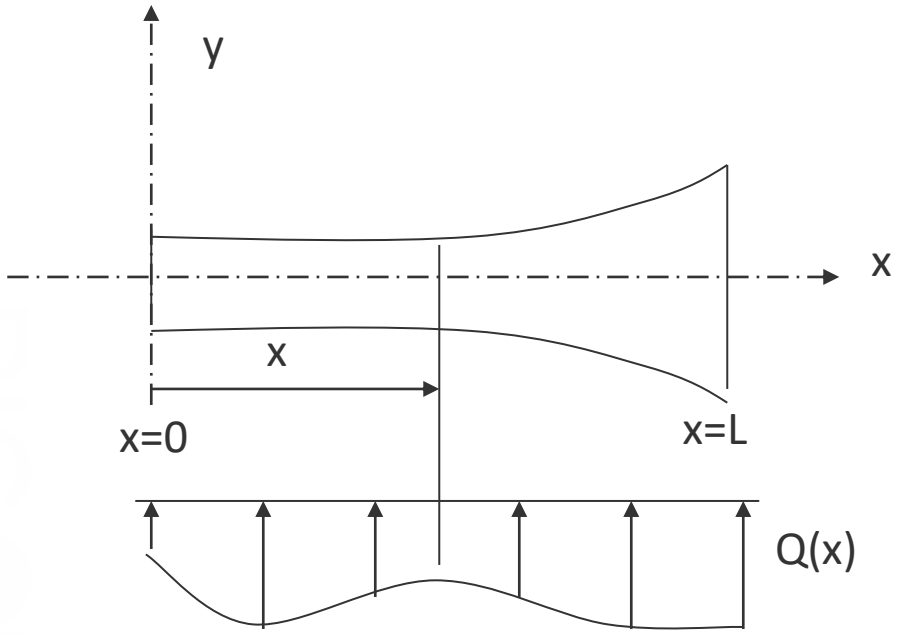
$$\frac{d}{dx}\left(Ak\frac{dT}{dx} \right) + Q = 0; \qquad 0 < x < L$$

Second order differential equations
Requires 2 boundary conditions for solution

Adapted from slides by Prof. Suvranu De

From ODEs to PDEs

Heat conduction in a fin



A(x) - cross section at x
Q(x) - heat input per unit length per unit time [J/sm]
k(x) - thermal conductivity [J/^{o}C ms]
T(x) - temperature of the fin at x

Boundary conditions (examples)

$T = 0 \quad at \quad x = 0$ Dirichlet/ displacement bc

$-k\frac{dT}{dx} = h \quad at \; x = L$ Neumann/ force bc

Adapted from slides by Prof. Suvranu De

Table 4.1 Examples of second-order differential equations

Differential equation	Physical problem	Quantities	Constitutive law
$\frac{d}{dx}\left(Ak\frac{dT}{dx}\right)+Q=0$	One-dimensional heat flow	T = temperature A = area k = thermal conductivity Q = heat supply	Fourier $q=-k\,dT/dx$ q = heat flux
$\frac{d}{dx}\left(AE\frac{du}{dx}\right)+b=0$	Axially loaded elastic bar	u = displacement A = area E = Young's modulus b = axial loading	Hooke $\sigma=E\,du/dx$ σ = stress
$S\frac{d^2w}{dx^2}+p=0$	Transversely loaded flexible string	w = deflection S = string force p = lateral loading	
$\frac{d}{dx}\left(AD\frac{dc}{dx}\right)+Q=0$	One-dimensional diffusion	c = iron concentration A = area D = diffusion coefficient Q = ion supply	Fick $q=-D\,dc/dx$ q = ion flux
$\frac{d}{dx}\left(A\gamma\frac{dV}{dx}\right)+Q=0$	One-dimensional electric current	V = voltage A = area γ = electric conductivity Q = electric charge supply	Ohm $q=-\gamma\,dV/dx$ q = electric charge flux
$\frac{d}{dx}\left(A\frac{D^2}{32\mu}\frac{dp}{dx}\right)+Q=0$	Laminar flow in pipe (Poiseuille flow)	p = pressure A = area D = diameter μ = viscosity Q = fluid supply	$q=-(D^2/32\mu)\,dp/dx$ q = volume flux q = mean velocity

1. All the cases we considered lead to very similar differential equations and boundary conditions.
2. In 1D it is easy to analytically solve these equations
3. Not so in 2 and 3D especially when the geometry of the domain is complex: need to solve **approximately**
4. Numerical methods can transfer from 1D to 2- and higher D cases

Adapted from slides by Prof. Suvranu De

Laplace's equation

$$\frac{\partial^2 u(x,y,z)}{\partial x^2} + \frac{\partial^2 u(x,y,z)}{\partial y^2} + \frac{\partial^2 u(x,y,z)}{\partial z^2} = 0$$

Used to describe the steady state distribution of heat in a body.

Also used to describe the steady state distribution of electrical charge in a body.

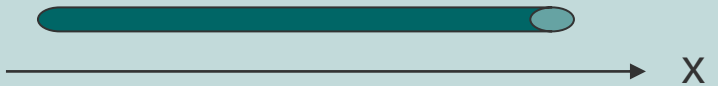
Heat equation

$$\frac{\partial u(x,y,z,t)}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

The function u(x,y,z,t) is used to represent the temperature at time *t* in a physical body at a point with coordinates (x,y,z)

α is the thermal diffusivity. It is sufficient to consider the case **α = 1**.

$$\frac{\partial\, T(x,t)}{\partial t} = \frac{\partial^2\, T(x,t)}{\partial x^2}$$



T(x,t) represents the temperature at time *t* at the point *x* of the thin rod.

Wave equation

$$\frac{\partial^2 u(x,y,z,t)}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

The function *u(x,y,z,t)* is used to represent the displacement at time t of a particle whose position at rest is *(x,y,z)* .

The constant *c* represents the propagation speed of the wave.

Classification of PDEs

Linear Second order PDEs are important sets of equations that are used to model many systems in many different fields of science and engineering.

Classification is important because:

- Each category relates to specific engineering problems.
- Different approaches are used to solve these categories.

Linear second order PDEs

A second order linear PDE (2-independent variables)

$$A\,u_{xx} + B\,u_{xy} + C\,u_{yy} + D = 0,$$

A, B, and C are functions of x and y

D is a function of x, y, u, u_x, and u_y

is classified based on $(B^2 - 4AC)$ as follows:

$B^2 - 4AC < 0$	Elliptic
$B^2 - 4AC = 0$	Parabolic
$B^2 - 4AC > 0$	Hyperbolic

Linear second order PDEs

Laplace Equation $\dfrac{\partial^2 u(x,y)}{\partial x^2} + \dfrac{\partial^2 u(x,y)}{\partial y^2} = 0$

$A = 1, B = 0, C = 1 \Rightarrow B^2 - 4AC < 0$

$\Rightarrow$ Laplace Equation *is Elliptic*

One possible solution: $u(x,y) = e^x \sin y$

$u_x = e^x \sin y, \quad u_{xx} = e^x \sin y$

$u_y = e^x \cos y, \quad u_{yy} = -e^x \sin y$

$u_{xx} + u_{yy} = 0$

Linear second order PDEs

Heat Equation $\alpha \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial u(x,t)}{\partial t} = 0$

$A = \alpha,\ B = 0,\ C = 0 \Rightarrow B^2 - 4AC = 0$

$\Rightarrow$ Heat Equation *is Parabolic*

Wave Equation $c^2 \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial^2 u(x,t)}{\partial t^2} = 0$

$A = c^2 > 0,\ B = 0,\ C = -1 \Rightarrow B^2 - 4AC > 0$

$\Rightarrow$ Wave Equation *is Hyperbolic*

Boundary conditions for PDEs

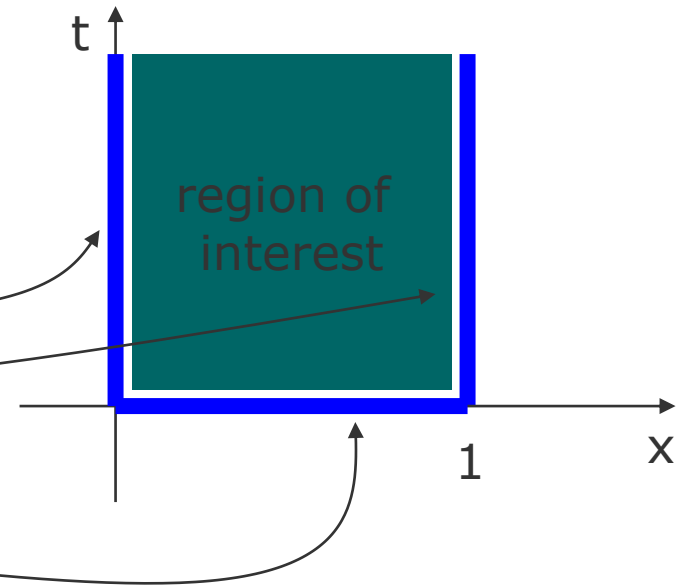
- To specify a solution to the PDE, a set of boundary conditions are needed.
- Both regular and irregular boundaries are possible.

$$\text{Heat Equation}: \alpha \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial u(x,t)}{\partial t} = 0$$

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

$$u(1,t) = 0$$

$$u(x,0) = \sin(\pi x)$$



Solution methods for PDEs

- Analytic solutions are possible for simple and special (idealized) cases only.
- To make use of the nature of the equations, different methods are used to solve different classes of PDEs.
- The methods discussed here are based on the finite difference technique.

Common methods for PDEs

- Finite difference method
  - Approximate the derivatives by finite differences
  - Easier to implement than other methods
  - Works best for regular (rectangular) shapes

- Finite element method
  - Subdivide the system into smaller parts -- finite elements
  - Boundary value problems in 2/3 dimensions
  - Works well for irregular shapes

- Finite volume method
  - Convert surface integrals around each mesh point into volume integrals
  - Conserves the mass by design, good for solving fluid dynamical equations

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = \mathbf{0}.$$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem

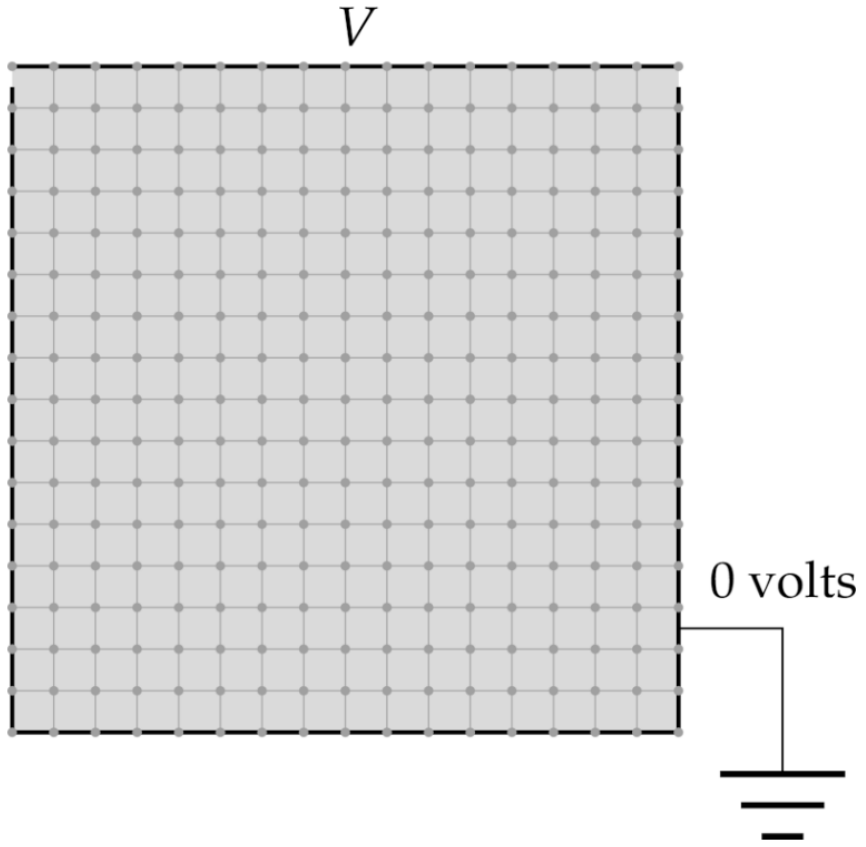
Consider Laplace's equation in two dimensions

$$\frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} = 0.$$

Boundary conditions

$$\begin{aligned} \phi(x, L) &= V, \\ \phi(x, 0) &= 0, \\ \phi(0, y) &= 0, \\ \phi(L, y) &= 0. \end{aligned}$$

How to find the profile $\phi(x, y)$?



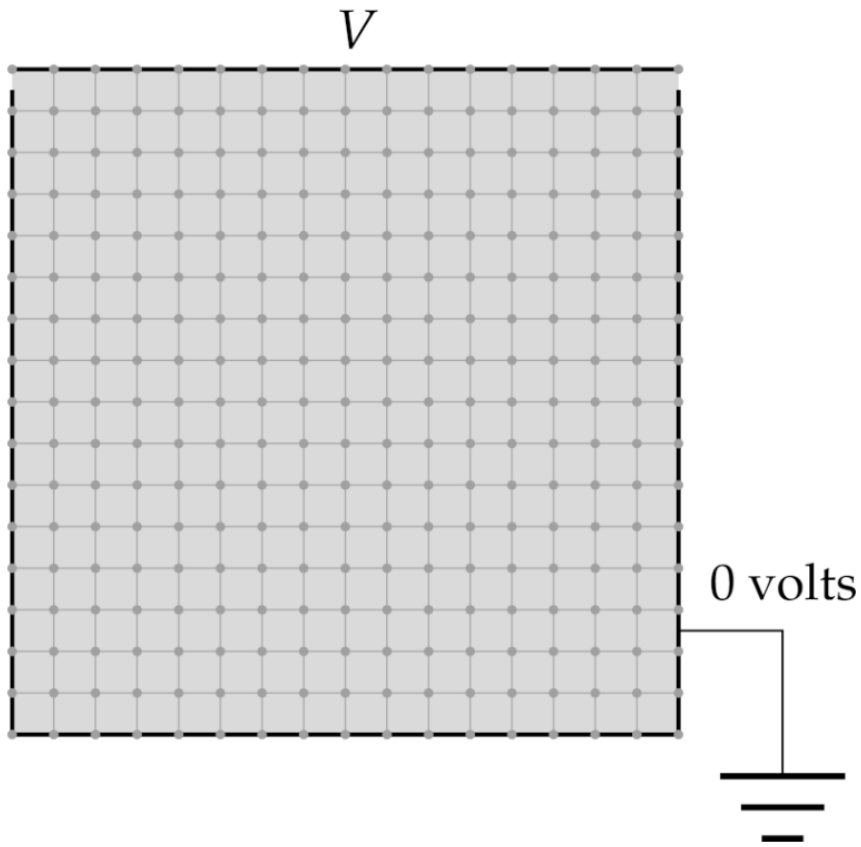
Source: M. Newman, Computational Physics

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Finite difference method

$$\frac{\partial^2\phi(x,y)}{\partial x^2} + \frac{\partial^2\phi(x,y)}{\partial y^2} = 0.$$

$$\begin{aligned}\phi(x,L) &= V,\\ \phi(x,0) &= 0,\\ \phi(0,y) &= 0,\\ \phi(L,y) &= 0.\end{aligned}$$



Source: M. Newman, Computational Physics

Discretize the space onto a grid MxM of length a=L/M

Apply central difference to the derivatives

$$\frac{\partial^2\phi(x,y)}{\partial x^2} = \frac{\phi(x+a,y) - 2\phi(x,y) + \phi(x-a,y)}{a^2},$$

$$\frac{\partial^2\phi(x,y)}{\partial y^2} = \frac{\phi(x,y+a) - 2\phi(x,y) + \phi(x,y-a)}{a^2}.$$

Laplace's equation becomes

$$\phi(x+a,y) + \phi(x-a,y) + \phi(x,y+a) + \phi(x,y-a) - 4\phi(x,y) = 0.$$

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Jacobi method

$$\phi(x+a,y)+\phi(x-a,y)+\phi(x,y+a)+\phi(x,y-a)-4\phi(x,y)=0.$$

is a system of M^2 linear equations

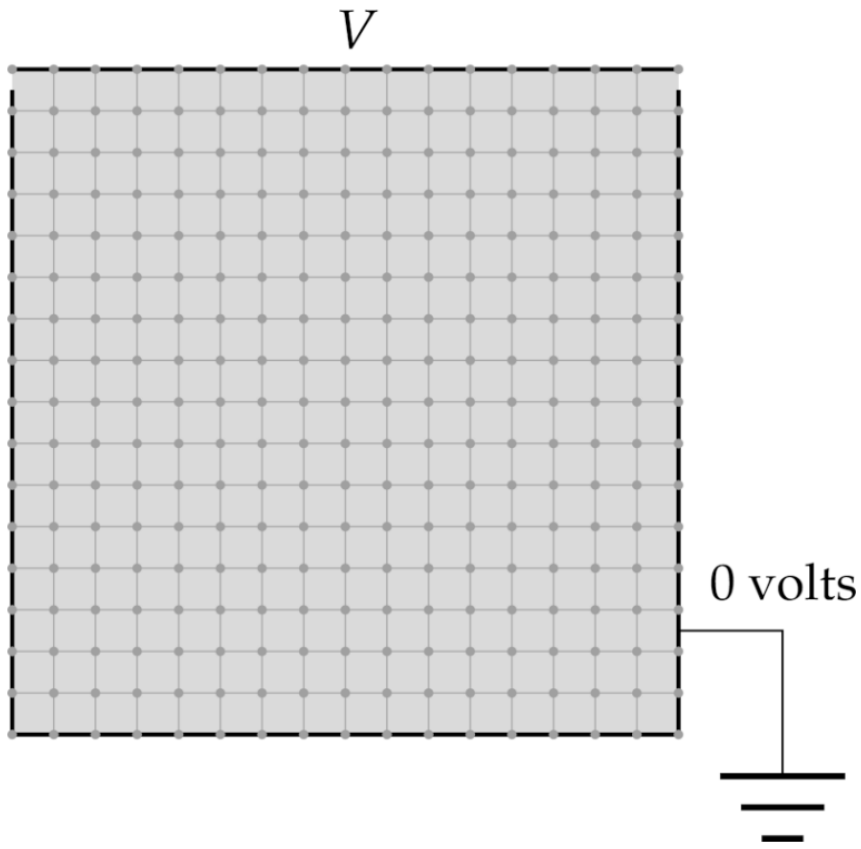
The general solution is $O(M^6)$

Jacobi method:

Perform the solution iteratively

$$\phi_{n+1}(x,y)=\frac{\phi_n(x+a,y)+\phi_n(x-a,y)+\phi_n(x,y+a)+\phi_n(x,y-a)}{4}$$

until the desired accuracy is met



Source: M. Newman, Computational Physics

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Jacobi method

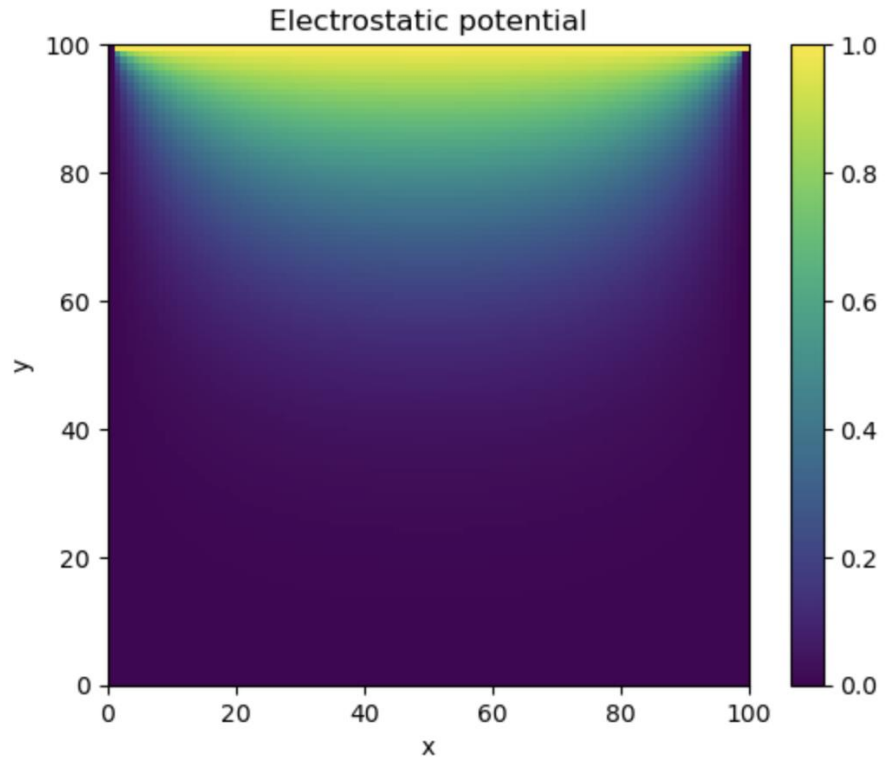
```python
import numpy as np

# Single iteration of the Jacobi method
# The new field is written into phinew
def iteration_jacobi(phinew, phi):
    M = len(phi) - 1

    # Boundary conditions
    phinew[0,:] = phi[0,:]
    phinew[M,:] = phi[M,:]
    phinew[:,0] = phi[:,0]
    phinew[:,M] = phi[:,M]

    for i in range(1,M):
        for j in range(1,M):
            phinew[i,j] = (phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + phi[i,j-1])/4

    delta = np.max(abs(phi-phinew))

    return delta

def jacobi_solve(phi0, target_accuracy = 1e-6, max_iterations = 100):
    delta = target_accuracy + 1.
    phi = phi0.copy()
    for i in range(max_iterations):
        delta = iteration_jacobi(phi, phi0)
        phi0, phi = phi, phi0

        if (delta <= target_accuracy):
            print("Jacobi method converged in " + str(i+1) + " iterations")
            return phi0

    print("Jacobi method failed to converge to a required precision in " + str(max_iterations) + " iterations")
    print("The error estimate is ", delta)

    return phi
```

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Jacobi method

```
# Constants
M = 100         # Grid squares on a side
V = 1.0         # Voltage at top wall
target = 1e-4   # Target accuracy

# Initialize with zeros
phi = np.zeros([M+1,M+1],float)
# Boundary condition
phi[0,:] = V
phi[:,0] = 0

phi = jacobi_solve(phi, target, 10000)

# Plot
import matplotlib.pyplot as plt

plt.title("Electrostatic potential")
plt.xlabel("x")
plt.ylabel("y")
CS = plt.imshow(phi, vmax=1., vmin=0.,origin="upper",extent=[0,M,0,M])
plt.colorbar(CS)
plt.show()
```

```
Jacobi method converged in 1909 iterations
```



From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Gauss-Seidel with overrelaxation

Gauss-Seidel method: Use newly computed ϕ_{n+1} whenever available

$$\phi_{n+1}(x,y) = \frac{\phi_n(x+a,y) + \phi_{n+1}(x-a,y) + \phi_n(x,y+a) + \phi_{n+1}(x,y-a)}{4}$$

Overrelaxation:

$$\phi_{n+1}(x,y) = \phi_n(x,y) + (1+\omega)\Delta_n\phi(x,y),$$

Combined:

$$\phi_{n+1}(x,y) = (1+\omega)\frac{\phi_n(x+a,y) + \phi_{n+1}(x-a,y) + \phi_n(x,y+a) + \phi_{n+1}(x,y-a)}{4} - \omega\,\phi_n(x,y),$$

Stable for w < 1, much faster

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Gauss-Seidel method

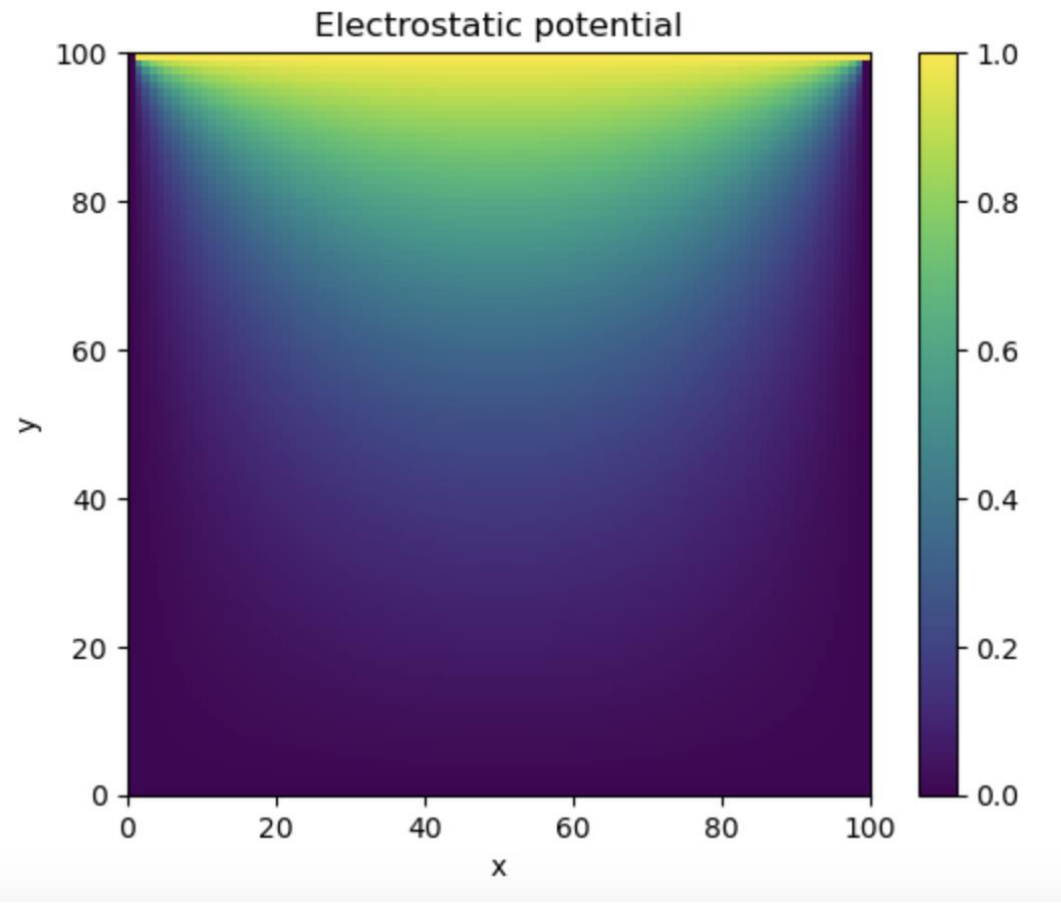
```python
import numpy as np

# Single iteration of the Jacobi method
# The new field is written into phinew
# omega >= 0 is the overelaxation parameter
def gaussseidel_iteration(phi, omega = 0):
    M = len(phi) - 1

    delta = 0.

    # New iteration
    for i in range(1,M):
        for j in range(1,M):
            phiold = phi[i,j]
            phi[i,j] = (1. + omega) * (phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + phi[i,j-1])/4 - omega * phi[i,j]
            delta = np.maximum(delta, abs(phiold - phi[i,j]))

    return delta

def gaussseidel_solve(phi0, omega = 0, target_accuracy = 1e-6, max_iterations = 100):
    delta = target_accuracy + 1.
    phi = phi0.copy()
    for i in range(max_iterations):
        delta = gaussseidel_iteration(phi, omega)

        if (delta <= target_accuracy):
            print("Jacobi method converged in " + str(i+1) + " iterations")
            return phi

    print("Jacobi method failed to converge to a required precision in " + str(max_iterations) + " iterations")
    print("The error estimate is ", delta)

    return phi
```

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Boundary value problem: Gauss-Seidel method

```
# Constants
M = 100         # Grid squares on a side
V = 1.0         # Voltage at top wall
target = 1e-4   # Target accuracy

# Initialize with zeros
phi = np.zeros([M+1,M+1],float)
# Boundary condition
phi[0,:] = V
phi[:,0] = 0

omega = 0.9
phi = gaussseidel_solve(phi, omega, target, 1000)

# Plot
import matplotlib.pyplot as plt

plt.title("Electrostatic potential")
plt.xlabel("x")
plt.ylabel("y")
CS = plt.imshow(phi, vmax=1., vmin=0.,origin="upper",extent=[0,M,0,M])
plt.colorbar(CS)
plt.show()
```

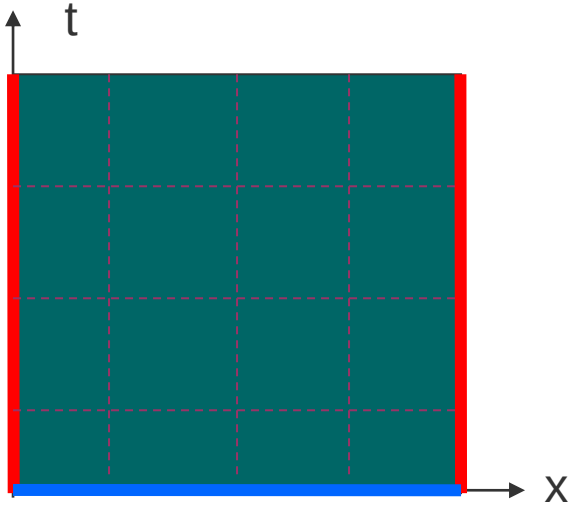
```
Jacobi method converged in 202 iterations
```



From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Finite Difference Methods

- Divide the interval *x* into sub-intervals, each of width *h*
- Divide the interval *t* into sub-intervals, each of width *k*
- A grid of points is used for the finite difference solution
- $T_{i,j}$ represents $T(x_i, t_j)$
- Replace the derivates by finite-difference formulas



Finite Difference Methods

Replace the derivatives by finite difference formulas

Central Difference Formula for $\frac{\partial^2 T}{\partial x^2}$:

$$\frac{\partial^2 T(x,t)}{\partial x^2} \approx \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{(\Delta x)^2} = \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{h^2}$$

Forward Difference Formula for $\frac{\partial T}{\partial t}$:

$$\frac{\partial T(x,t)}{\partial t} \approx \frac{T_{i,j+1} - T_{i,j}}{\Delta t} = \frac{T_{i,j+1} - T_{i,j}}{k}$$

Solution of the heat equation

1. Explicit Method:

   Simple, Stability Problems.

2. Crank-Nicolson Method:

   Involves the solution of a Tridiagonal system of equations, Stable.

Explicit Method

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2}$$

$$\frac{T(x,t+k)-T(x,t)}{k} = \frac{T(x-h,t)-2T(x,t)+T(x+h,t)}{h^2}$$

$$T(x,t+k)-T(x,t) = \frac{k}{h^2}\left(T(x-h,t)-2T(x,t)+T(x+h,t)\right)$$

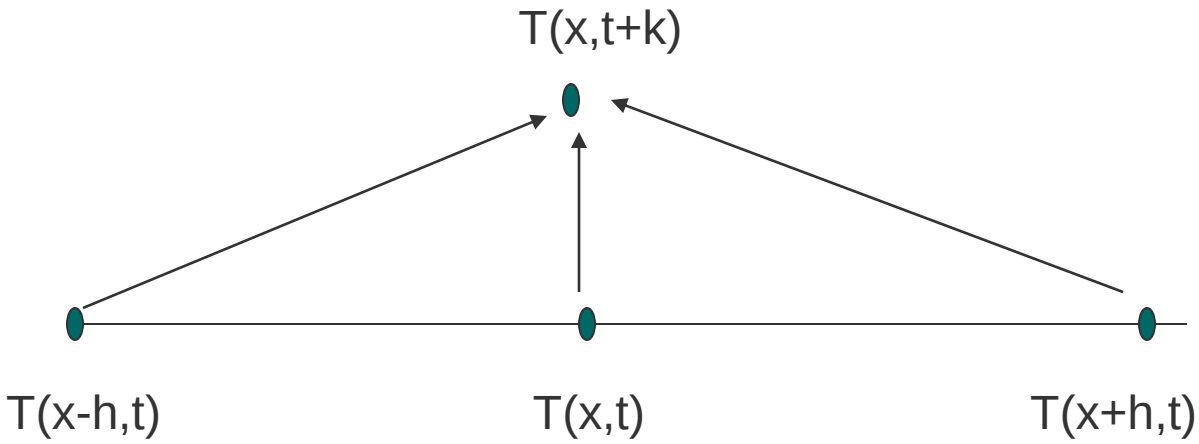
$$\textit{Define} \quad \lambda = \frac{k}{h^2}$$

$$T(x,t+k) = \lambda\ T(x-h,t)+(1-2\lambda)\ T(x,t)+\lambda\ T(x+h,t)$$

Explicit Method

$$T(x,t+k) = \lambda\, T(x-h,t) + (1-2\lambda)\, T(x,t) + \lambda\, T(x+h,t)$$

means



Convergence and Stability

$T(x,t+k)$ can be computed directly using:

$$T(x,t+k) = \lambda\ T(x-h,t) + (1-2\lambda)\ T(x,t) + \lambda\ T(x+h,t)$$

Can be unstable (errors are magnified)

To guarantee stability, $(1-2\lambda) \geq 0 \Rightarrow \lambda \leq \frac{1}{2} \Rightarrow k \leq \frac{h^2}{2}$

This means that k is much smaller than h

This makes it slow.

Convergence and Stability

- Convergence

  The solutions converge means that the solution obtained using the finite difference method approaches the true solution as the steps Δx and Δt approach zero.

- Stability:

  An algorithm is stable if the errors at each stage of the computation are not magnified as the computation progresses.

Crank – Nicolson method

The method involves solving a Tridiagonal system of linear equations.

The method is stable (No magnification of error).

$\rightarrow$ We can use larger h, k (compared to the Explicit Method).

Based on the finite difference method

1. Divide the interval x into subintervals of width h
2. Divide the interval t into subintervals of width k
3. Replace the first and second partial derivatives with their *backward* and *central difference* formulas respectively:

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x,t) - u(x,t-k)}{k}$$

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

Crank – Nicholson method

$$\text{Heat Equation}: \frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \text{ becomes}$$

$$\frac{u(x-h,t)-2u(x,t)+u(x+h,t)}{h^2} = \frac{u(x,t)-u(x,t-k)}{k}$$

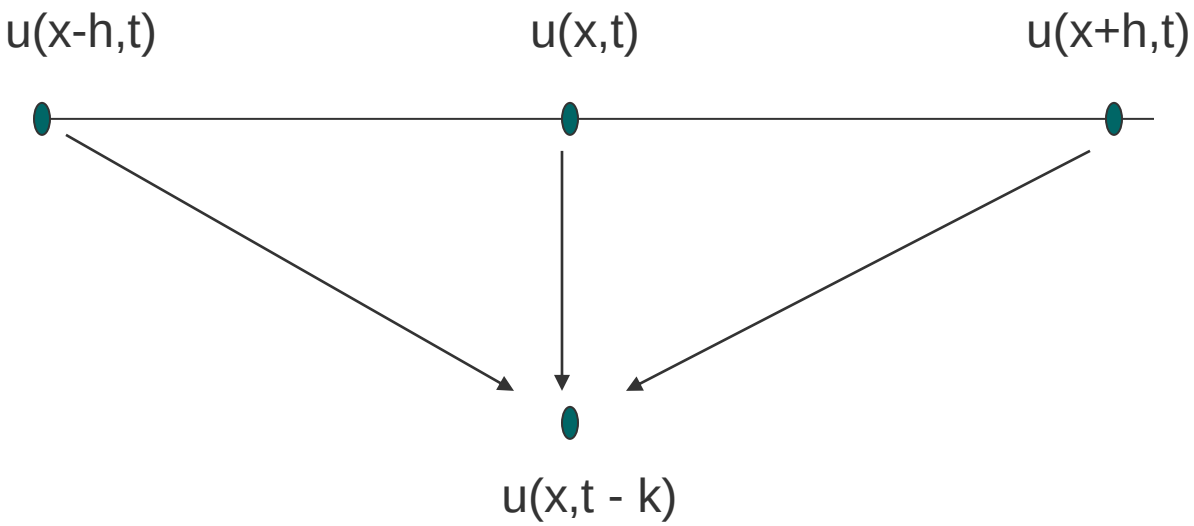
$$\frac{k}{h^2}\left(u(x-h,t)-2u(x,t)+u(x+h,t)\right) = u(x,t)-u(x,t-k)$$

$$-\frac{k}{h^2}u(x-h,t)+\left(1+2\frac{k}{h^2}\right)u(x,t)-\frac{k}{h^2}u(x+h,t) = u(x,t-k)$$

Crank – Nicholson method

Define $\lambda = \frac{k}{h^2}$ then Heat equation becomes :

$$-\lambda\, u(x-h,t)+(1+2\lambda)\, u(x,t)-\lambda\, u(x+h,t)=u(x,t-k)$$



Crank – Nicholson method

The equation :

$$-\lambda\, u(x-h,t)+(1+2\lambda)\, u(x,t)-\lambda\, u(x+h,t)=u(x,t-k)$$

can be rewritten as :

$$-\lambda\, u_{i-1,\,j}+(1+2\lambda)\, u_{i,\,j}-\lambda\, u_{i+1,\,j}=u_{i,\,j-1}$$

and can be expanded as a system of equations (fix $j=1$) :

$$-\lambda\, u_{0,1}+(1+2\lambda)\, u_{1,1}-\lambda\, u_{2,1}=u_{1,0}$$

$$-\lambda\, u_{1,1}+(1+2\lambda)\, u_{2,1}-\lambda\, u_{3,1}=u_{2,0}$$

$$-\lambda\, u_{2,1}+(1+2\lambda)\, u_{3,1}-\lambda\, u_{4,1}=u_{3,0}$$

$$-\lambda\, u_{3,1}+(1+2\lambda)\, u_{4,1}-\lambda\, u_{5,1}=u_{4,0}$$

Crank – Nicholson method

$$-\lambda\, u(x-h,t)+(1+2\lambda)\, u(x,t)-\lambda\, u(x+h,t)=u(x,t-k)$$

can be expressed as a Tridiagonal system of equations :

$$\begin{bmatrix} 1+2\lambda & -\lambda & & \\ -\lambda & 1+2\lambda & -\lambda & \\ & -\lambda & 1+2\lambda & -\lambda \\ & & -\lambda & 1+2\lambda \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{4,1} \end{bmatrix} = \begin{bmatrix} u_{1,0}+\lambda\, u_{0,1} \\ u_{2,0} \\ u_{3,0} \\ u_{4,0}+\lambda\, u_{5,1} \end{bmatrix}$$

where $u_{1,0}, u_{2,0}, u_{3,0}$, and $u_{4,0}$ are the initial temperature values at $x = x_0 + h,\ x_0 + 2h,\ x_0 + 3h,$ and $x_0 + 4h$

$u_{0,1}$ and $u_{5,1}$ are the boundary values at $x = x_0$ and $x_0 + 5h$

Crank – Nicholson method

The solution of the tridiagonal system produces :

The temperature values $u_{1,1}, u_{2,1}, u_{3,1}$, and $u_{4,1}$ at $t = t_0 + k$

To compute the temperature values at $t = t_0 + 2k$

Solve a second tridiagonal system of equations ($j = 2$)

$$\begin{bmatrix} 1+2\lambda & -\lambda & & \\ -\lambda & 1+2\lambda & -\lambda & \\ & -\lambda & 1+2\lambda & -\lambda \\ & & -\lambda & 1+2\lambda \end{bmatrix} \begin{bmatrix} u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ u_{4,2} \end{bmatrix} = \begin{bmatrix} u_{1,1} + \lambda u_{0,2} \\ u_{2,1} \\ u_{3,1} \\ u_{4,1} + \lambda u_{5,2} \end{bmatrix}$$

To compute $u_{1,2}, u_{2,2}, u_{3,2}$, and $u_{4,2}$

Repeat the above step to compute temperature values at $t_0 + 3k$, etc.

General comments

The Explicit Method:

- One needs to select small k to ensure **stability.**
- Computation per point is very simple but many points are needed.

Cranks Nicolson:

- Requires the solution of a **Tridiagonal** system.
- Stable (Larger k can be used).