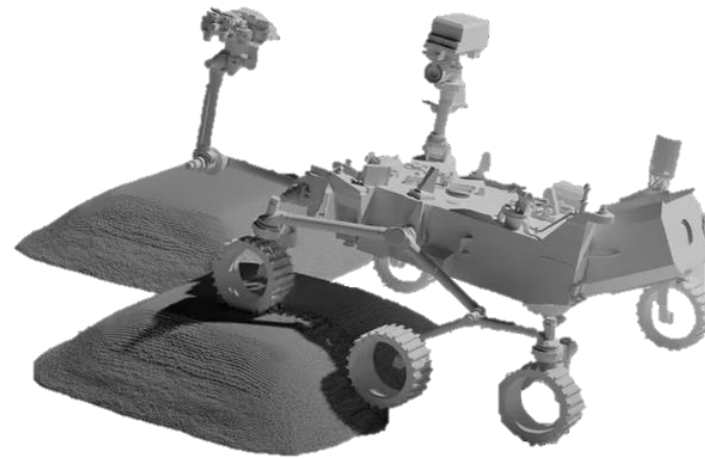


SIMULATING DYNAMIC SYSTEMS

An introduction to using Scipy in Python



UNIVERSITY OF
WATERLOO

LEARNING OUTCOMES

- Simulate a system of ordinary differential equations using Scipy
- Plot simulated dynamic trajectories using Matplotlib
- Be familiar with the basic concepts behind ODE integrators

RESOURCES

- APMonitor course
 - <https://apmonitor.com/pdc/index.php/Main/SolveDifferentialEquations>
- Documentation
 - <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html>
- Many tutorials freely available

SIMULATING DYNAMIC SYSTEMS

- Simulating dynamic systems is vital for enabling engineering applications



Curiosity Rover



Batch Reactor

- Simulate using **numerical methods to approximate dynamics** (e.g., (partial) differential equations)
- Enables us to **computationally experiment** and implement **automation**

ANALYTICAL VS. NUMERICAL METHODS

Analytical Methods

- Separate and integrate

$$g(y) \frac{dy}{dx} = h(x) \quad \Rightarrow \quad \int g(y) dy = \int h(x) dx + C$$

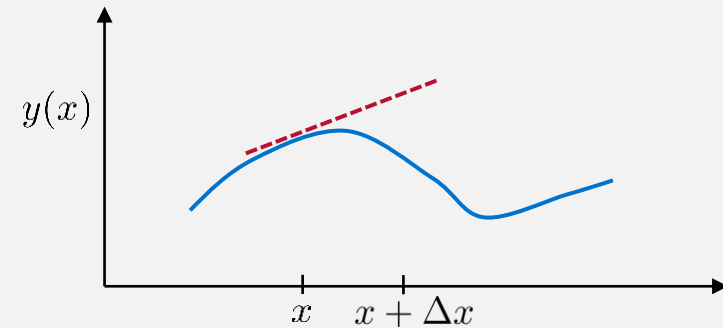
- ODEs of special forms

$$\frac{dy}{dx} = \frac{x+y}{x-y} \quad \Rightarrow \quad \frac{1}{2} \log \left(\frac{y^2}{x^2} + 1 \right) - \tan^{-1} \left(\frac{y}{x} \right) = C - \log(x)$$

- Solving general ODEs is often difficult or not possible

Numerical Methods

- Seek to numerically approximate the solution
- **Finite difference methods** are common



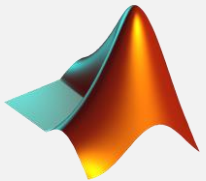
- **ODE integrators** typically use a range of advanced finite difference techniques

COMMON SIMULATION TOOLS

ODE Integrators

- Common in scripting languages
- Provide numerical solutions to ODE systems

julia



Symbolic Solvers

- Can provide analytic solutions when possible
- Typically, not used for large problems



Mathcad®

Optimization Tools

- Can incorporate differential equations when solving optimization problems



Problem Specific

- Simulate dynamics for particular systems



OpenFOAM®



EXPLICIT EULER (FORWARD DIFFERENCE)

Methodology

- Consider a 1st order ODE

$$\frac{dy(t)}{dt} = f(y(t), t)$$

$$y(0) = y_0$$

- Define time steps Δt

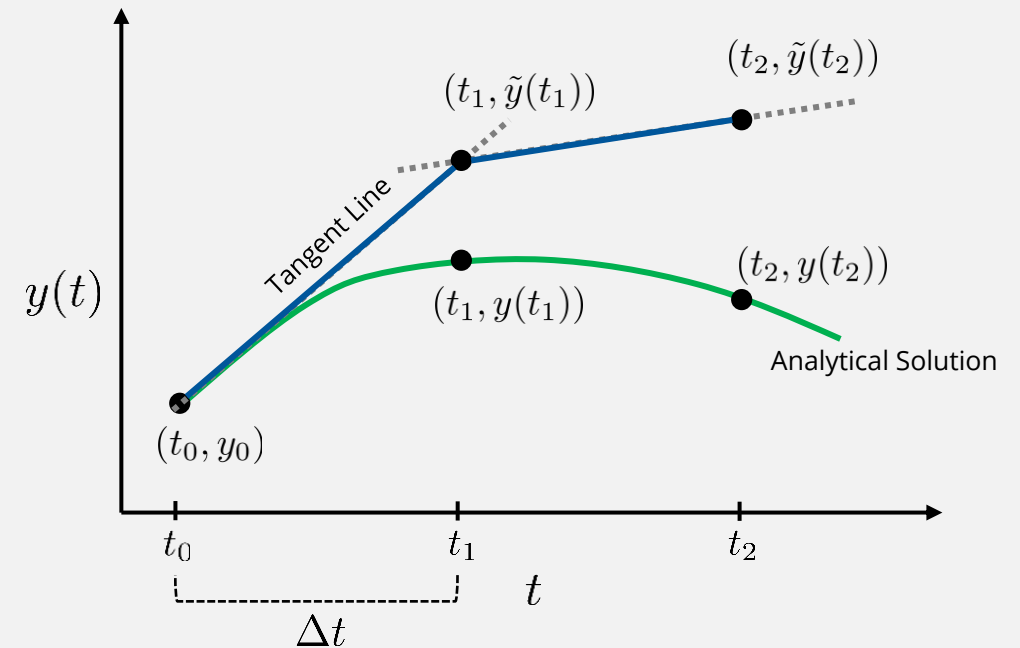
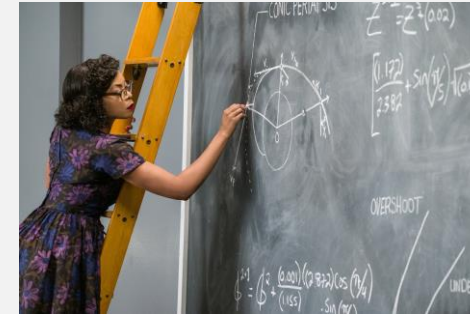
$$t \in [t_0, t_f] \quad t_k = t_0 + k\Delta t$$

- Approximate derivative as finite difference

$$\left. \frac{dy(t)}{dt} \right|_{t_k} \approx \frac{\tilde{y}(t_{k+1}) - \tilde{y}(t_k)}{\Delta t}$$

- Define update rule

$$\tilde{y}(t_{k+1}) = \tilde{y}(t_k) + f(y(t_k), t_k)\Delta t$$



PROPERTIES: ERROR

Local Truncation Error (LTE)

- Recall update rule

$$\tilde{y}(t_{k+1}) = \tilde{y}(t_k) + f(y(t_k), t_k) \Delta t$$

- Taylor series expansion of analytic solution

$$y(t_k + \Delta t) = y(t_k) + \Delta t \left. \frac{dy(t)}{dt} \right|_{t_k} + O(\Delta t^2)$$

- Difference w/ explicit Euler

$$y(t_k + \Delta t) - \tilde{y}(t_{k+1}) = O(\Delta t^2)$$

- Hence, the error incurred after one step is

$$O(\Delta t^2)$$

Global Truncation Error (GTE)

- The number of steps

$$\frac{t - t_0}{\Delta t} \propto \frac{1}{\Delta t}$$

- Multiplying the LTE, we get GTE that is

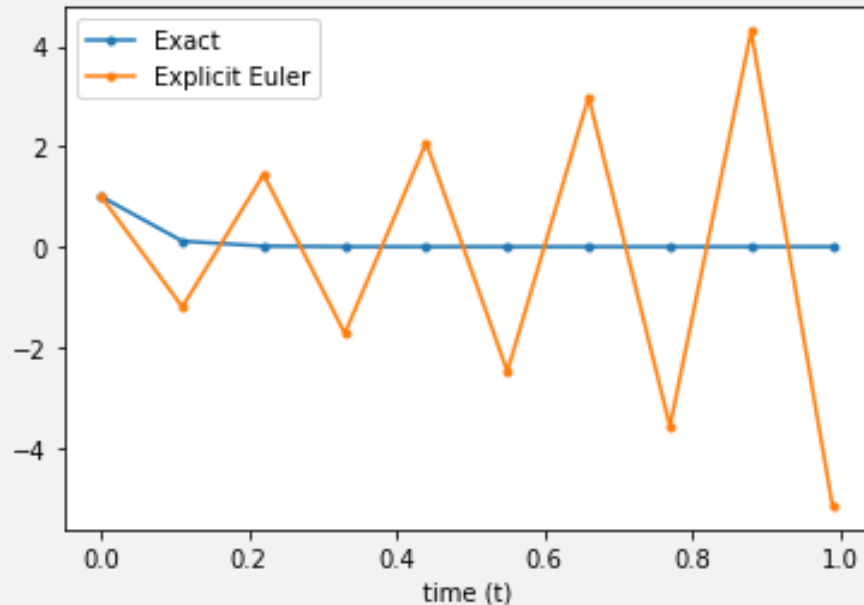
$$O(\Delta t)$$

- Hence, explicit Euler is a **first order method**

- **Higher order methods are available**

PROPERTIES: STABILITY

Example



Stiff ODEs

- Systems that exhibit **numerical instability**
- Precise mathematical definition is nontrivial
- Common with **reaction systems**
 - Coexistence of small and large rate constants
- So, what can we do? → Use **implicit methods**

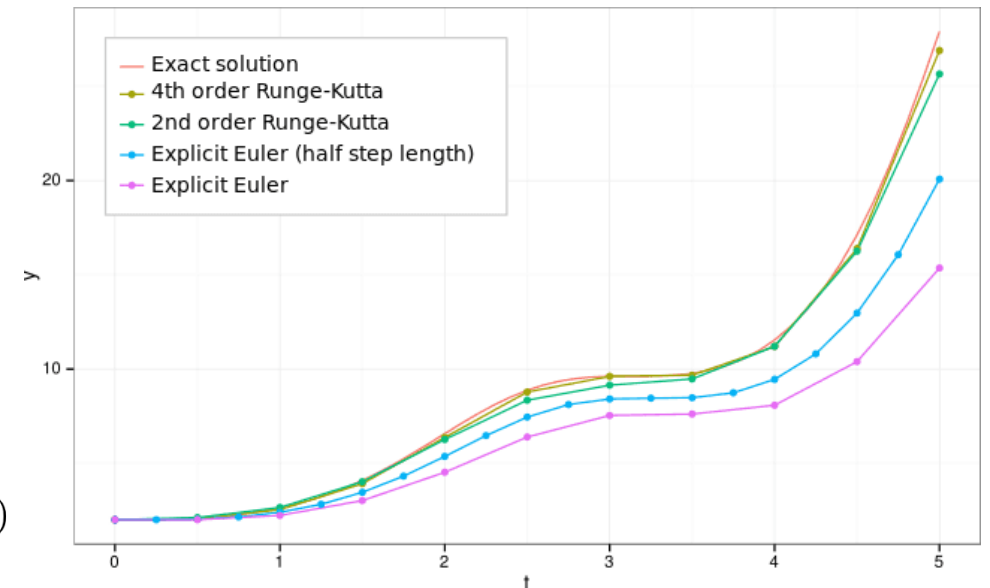
$$\tilde{y}(t_{k+1}) = \tilde{y}(t_k) + f(\tilde{y}(t_{k+1}), t_{k+1})\Delta t$$

ODE INTEGRATION

- Numerically approximate system of ODEs by taking a series of steps
- Higher order methods are typical
- Runge-Kutta (RK) methods are common

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i,$$
$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + c_2 h, y_n + (a_{21} k_1)h), \\ k_3 &= f(t_n + c_3 h, y_n + (a_{31} k_1 + a_{32} k_2)h), \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + (a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1})h) \end{aligned}$$

- Linear multistep methods are also popular
 - Utilizes linear combination of previous steps



USING SCIPY

scipy.integrate.odeint

```
scipy.integrate.odeint(func, y0, t, args=(), Dfun=None, col_deriv=0, full_output=0,  
ml=None, mu=None, rtol=None, atol=None, tcrit=None, h0=0.0, hmax=0.0, hmin=0.0, ixpr=0,  
mxstep=0, mxhnil=0, mxordn=12, mxords=5, printmessg=0, tfirst=False) \[source\]
```

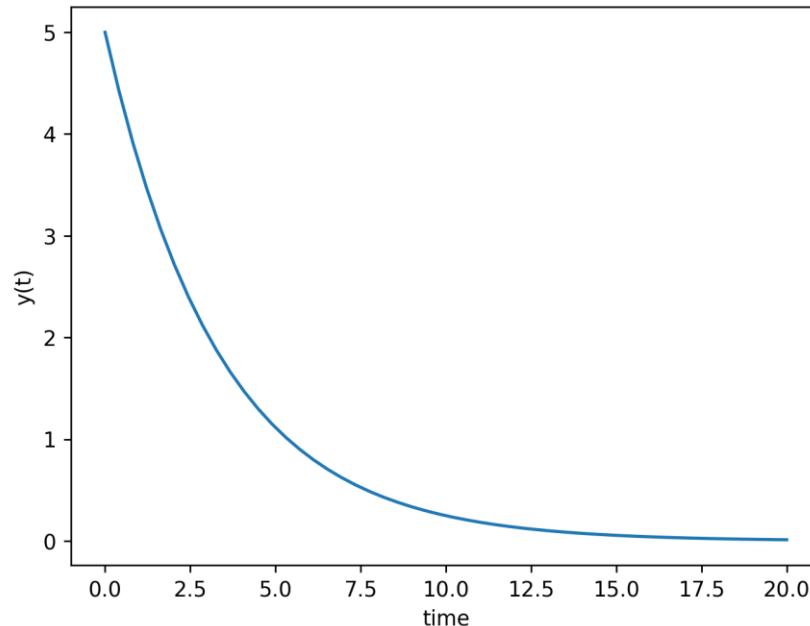
Integrate a system of ordinary differential equations.

- We will use `odeint` instead of `solve_ivp`
 - For control simulations, we need to control the time steps
- Reformulate ODEs into standard form
- Make a function that returns the LHS
- Requires the LHS function, initial values, and time steps
- Automatically detects stiffness and chooses numerical method

BASIC EXAMPLE

- Simulate the following with Scipy and plot with Matplotlib
 - Let $k = 0.3$

$$\frac{dy(t)}{dt} = -k y(t) \quad y(0) = 5$$

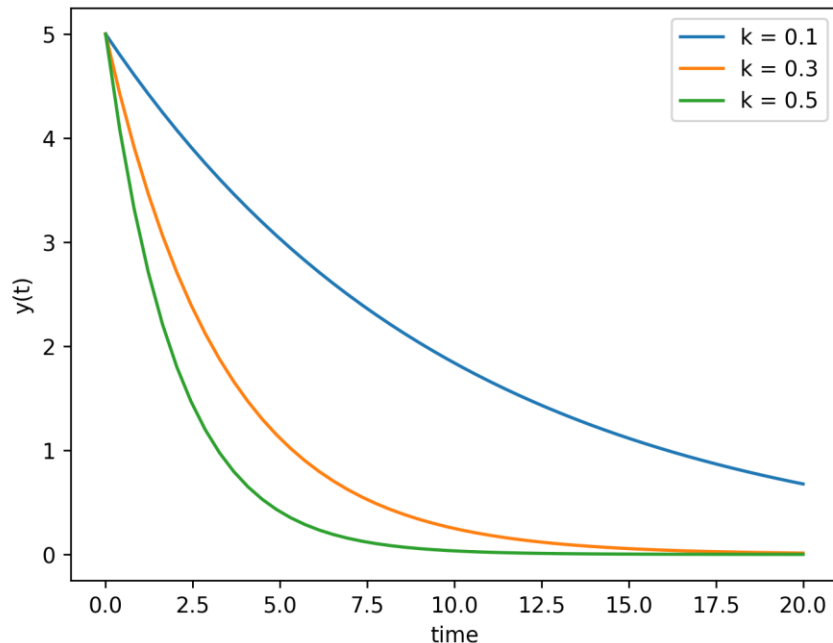


```
1  √ from scipy.integrate import odeint
2      import numpy as np
3      import matplotlib.pyplot as plt
4
5      # function that returns dy/dt
6  √ def model(y, t):
7      |     k = 0.3
8      |     dydt = -k * y
9      |     return dydt
10
11     # set the parameters
12     y0 = 5
13     ts = np.linspace(0, 20)
14
15     # solve ODE
16     ys = odeint(model, y0, ts)
17
18     # plot results
19     plt.plot(ts, ys)
20     plt.xlabel('time')
21     plt.ylabel('y(t)')
22     plt.show()
```

USING PARAMETER ARGUMENTS

- Simulate the following with Scipy and plot with Matplotlib
 - Plot $k \in \{0.1, 0.3, 0.5\}$

$$\frac{dy(t)}{dt} = -k y(t) \quad y(0) = 5$$



```
1  from scipy.integrate import odeint
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # function that returns dy/dt
6  def model(y, t, k):
7      dydt = -k * y
8      return dydt
9
10 # set the parameters
11 y0 = 5
12 ts = np.linspace(0, 20)
13 ks = [0.1, 0.3, 0.5]
14
15 # solve ODEs and plot
16 plt.figure()
17 for k in ks:
18     ys = odeint(model, y0, ts, args=(k,))
19     plt.plot(ts, ys, label='k = '+str(k))
20
21 # Show the finalized plot
22 plt.legend(loc='best')
23 plt.xlabel('time')
24 plt.ylabel('y(t)')
25 plt.show()
```

SIMPLE EXERCISES: WHAT IS THE LONG-TERM STEADY STATE?

- Problem 1

$$\begin{aligned}\frac{dy(t)}{dt} &= -y(t) + 1 \\ y(0) &= 0\end{aligned}$$

- Problem 2

$$\begin{aligned}5 \frac{dy(t)}{dt} &= -y(t) + u(t) \\ y(0) &= 1\end{aligned} \quad u(t) = \begin{cases} 0 & t < 10 \\ 2 & t \geq 10 \end{cases}$$

SIMPLE EXERCISES

- Problem 3

$$\begin{aligned}\frac{dx(t)}{dt} &= 3 \exp(-t) & \frac{dy(t)}{dt} &= 3 - y(t) \\ x(0) &= y(0) = 0\end{aligned}$$

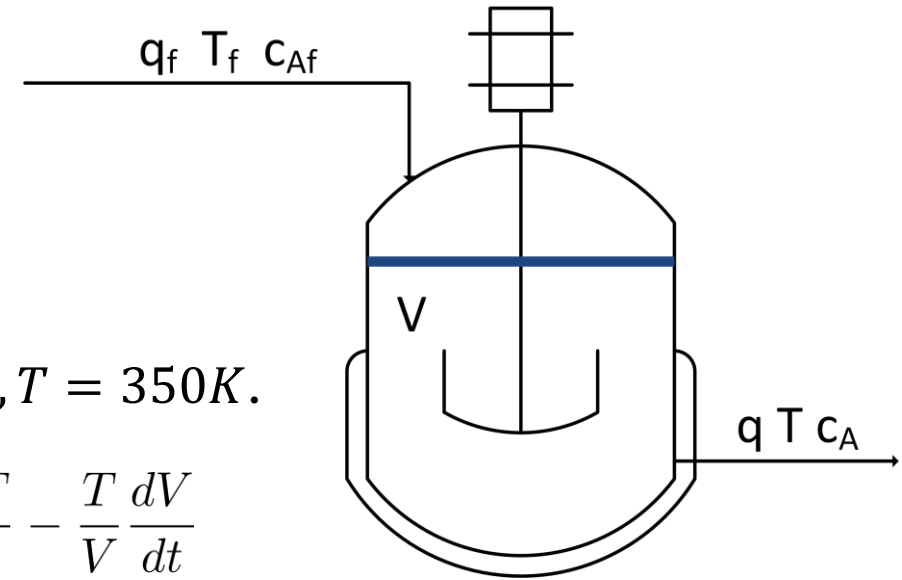
- Problem 4

$$\begin{aligned}2 \frac{dx(t)}{dt} &= -x(t) + u(t) & u(t) &= \begin{cases} 0 & t < 5 \\ 2 & t \geq 5 \end{cases} \\ 5 \frac{dy(t)}{dt} &= -y(t) + x(t) & x(0) &= y(0) = 0\end{aligned}$$

CASE STUDY: MIXER

- Recall the mixer model we developed
 - Initial conditions for the vessel are $V = 1.0L$, $c_A = 0.0 \frac{\text{mol}}{L}$, $T = 350K$.

$$\frac{dV}{dt} = q_f - q \quad \frac{dc_A}{dt} = \frac{q_f c_{Af} - q c_A}{V} - \frac{C_A}{V} \frac{dV}{dt} \quad \frac{dT}{dt} = \frac{q_f T_f - q T}{V} - \frac{T}{V} \frac{dV}{dt}$$



BEFORE NEXT TIME

- Quiz 2: Due at 11:59 pm via Learn
- Assignment 1: Due Monday at 1:30 pm via Learn
- Prepare for lecture (~1 hr)
 - <https://apmonitor.com/pdc/index.php/Main/ModelLinearization>
 - <https://apmonitor.com/pdc/index.php/Main/FirstOrderSystems>
 - <https://apmonitor.com/pdc/index.php/Main/FirstOrderGraphical>
 - <https://apmonitor.com/pdc/index.php/Main/FirstOrderPlusDeadTime>