ELEN90097
Modelling and Analysis for AI

Prof Tansu Alpcan

THE UNIVERSITY OF MELBOURNE

ELEN90097 Modelling and Analysis for AI

Module 2 – System Modelling
Lesson 01

Prof Tansu Alpcan

This lesson contains

- Modelling engineering and physical systems using ODEs
- Solving ODEs numerically, Runge-Kutta method
- Simulating physical systems
- A quick overview/refresher on system behaviour and control

Suggested Reading

- Links and a more extensive list is on Canvas > Online Resources
- Feedback Systems: An Introduction for Scientists and Engineers by Karl J. Åström and Richard M. Murray.
- Modeling and Simulation in Python by Allen B. Downey.

System Modelling

System Modelling Fundamentals

- A model is a mathematical representation of a physical, biological, or information system.
- Models help us to reason about a system and make predictions about system behaviour.
- All models are wrong! **Models approximate the underlying system**.

Philosophical perspective

- There are degrees of right and wrong, see e.g. Asimov's great essay *"The Relativity of Wrong"*
  https://hermiene.net/essays-trans/relativity_of_wrong.html
- And this is not restricted to technical things. Even novelists use some type of modelling!
  http://helenlowe.info/blog/2013/05/19/a-book-quote-for-sunday-from-ursula-le-guin/

Historical Perspective

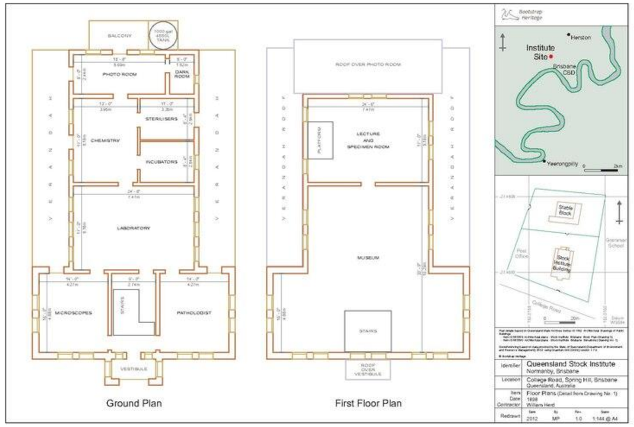
Disciplines and centuries

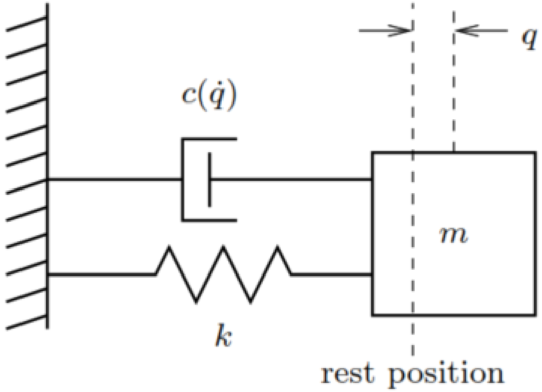
<18th civil engineering

19th **mechanical** engineering

20th **electrical** engineering

21st **computing**



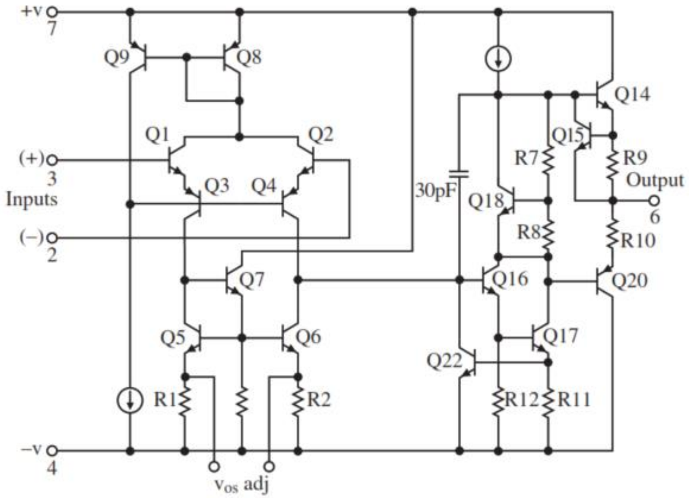


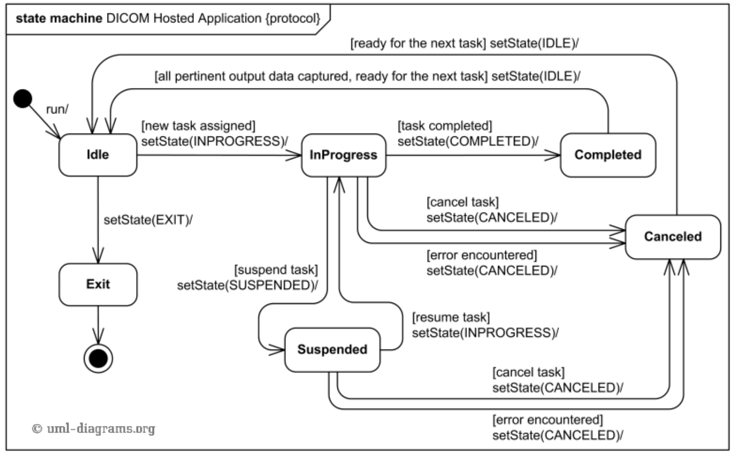
(top left): architectural drawing

(top right): spring-mass system

(bottom left): circuit diagram

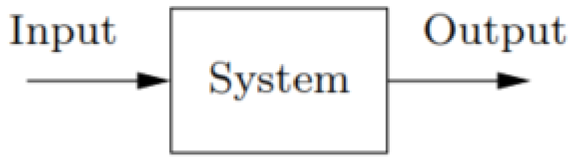
(bottom right): finite state machine of a (computing) protocol

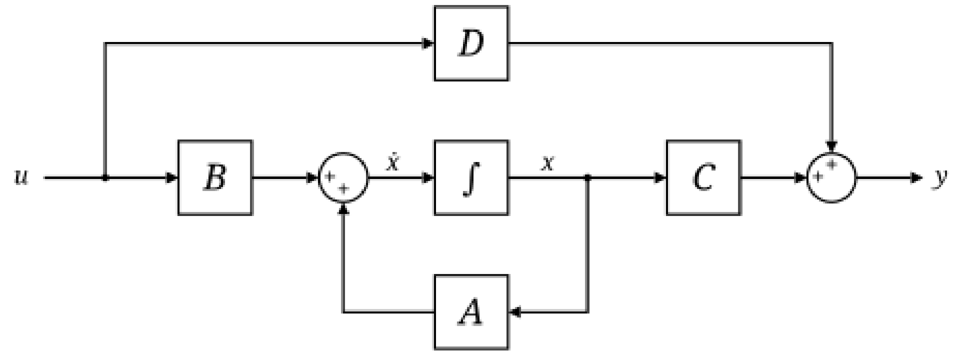




State Space Models

- A **state-space representation** is a mathematical model of a **physical system** specified as a set of input, output, and state variables.
- The state of a system is a collection of (state) variables that summarize the past of a system.
  - Purpose is predicting the future of the system.
  - This is essentially Markovian. All history is captured by the value of state variables.
- State variables can be combined into a state vector.
- System behaviour is described by differential equations or difference equations that relate input, output, and state variables.





$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}$$

ODEs as State Space Models

- Ordinary Differential (Difference) Equations are very widely used as state-space models of physical systems (mech, electrical, chemical, bio).
- These powerful models
  - are descriptive (explanatory)
  - predict future behaviour of systems
  - enable system control and optimisation

$$\frac{dx}{dt} = f(x, u), \qquad y = h(x, u)$$

$$\frac{dx}{dt} = Ax + Bu, \qquad y = Cx + Du$$

$$x[k+1] = f(x[k], u[k]), \qquad y[k] = h(x[k], u[k])$$

$$x[k+1] = Ax[k] + Bu[k], \qquad y[k] = Cx[k] + Du[k]$$

Common Assumptions

Widely used ODE models commonly make the following assumptions:

- Linear vs nonlinear
- Time (or shift) invariance
- Variables are real, in Euclidean space $\mathbb{R}^n$
- Time is continuous (in majority of models) or simply uniformly discretised (k integer)
- Linear algebra plays a significant role in linear models (A, B, C, D)
- Inputs (u) and outputs (y) are well-defined
- State assumption holds (no chaotic systems or long-range dependence)
- Often deterministic or additive uncertainty “noise” under i.i.d. assumptions on random variables.

$$\frac{dx}{dt} = f(x,u), \qquad y = h(x,u)$$

$$\frac{dx}{dt} = Ax + Bu, \qquad y = Cx + Du$$

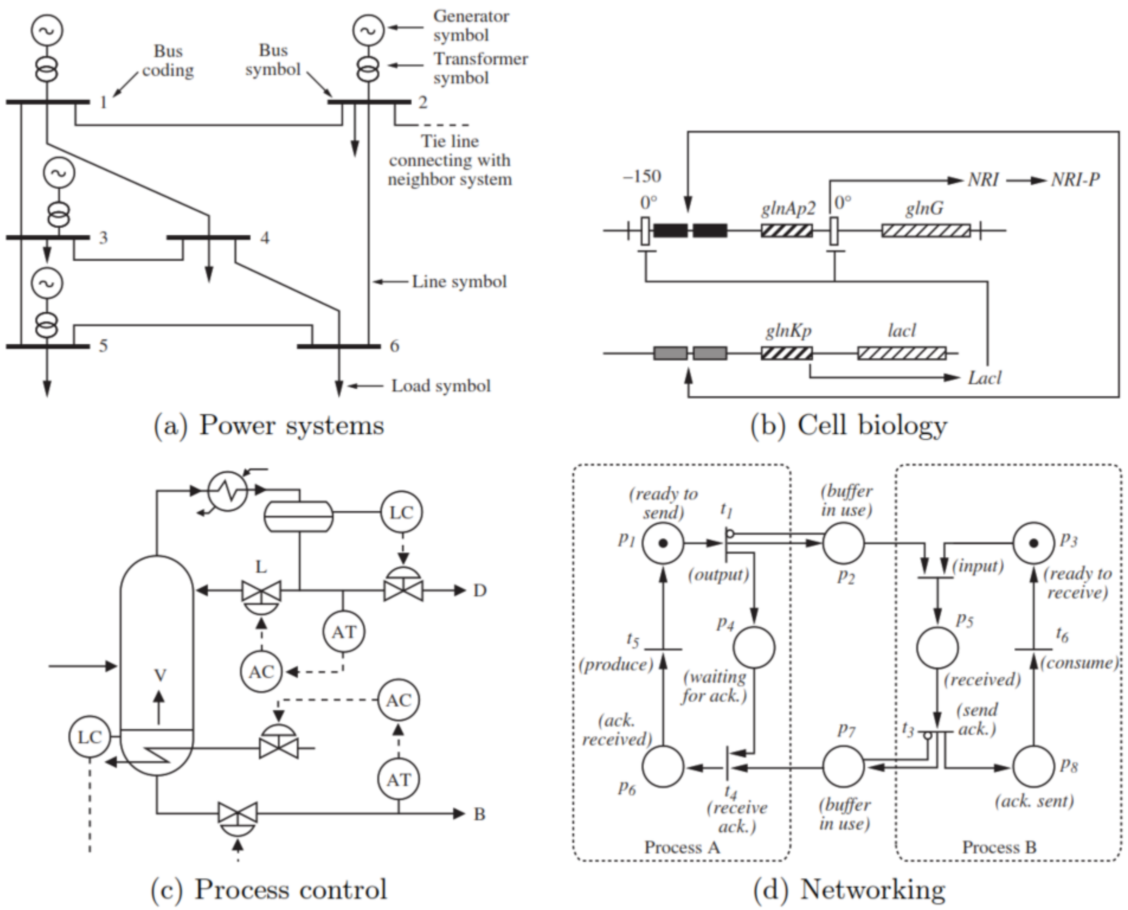
$$x[k+1] = f(x[k], u[k]), \qquad y[k] = h(x[k], u[k])$$

$$x[k+1] = Ax[k] + Bu[k], \qquad y[k] = Cx[k] + Du[k]$$

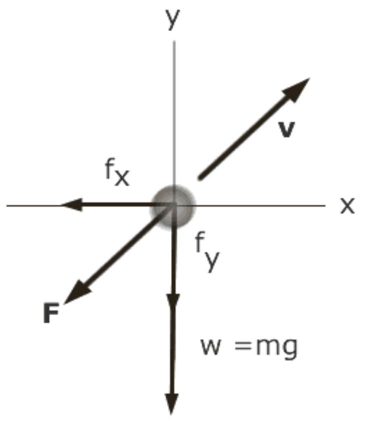
- *Note that a “mechanistic” worldview underpins these assumptions.*
- *Mathematical tools are mainly from linear algebra, multivariate calculus, real/functional analysis*

Visualisation: block diagrams

- Block diagrams are widely used in engineering and computing disciplines to visualise systems.
- They usually emphasise information flow and provide a high-level view by hiding details of the system.
- There are even visual programming languages/software turn diagrams functional (e.g. Simulink, LabView)



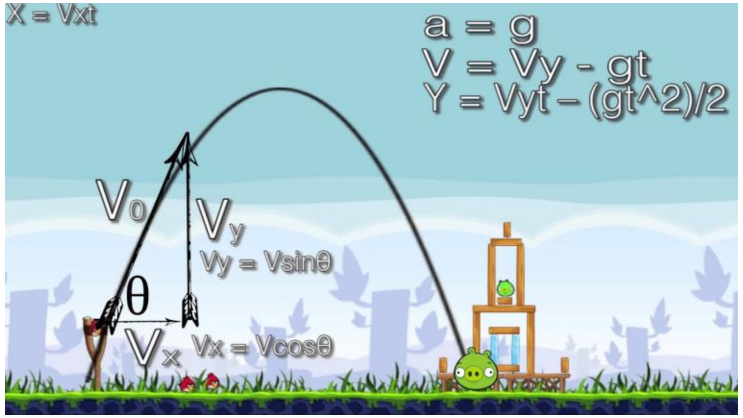
ODE Example – projectile motion



Let’s consider a system modelled by this very general ODE

$$y'(t) = f\big(t, y(t)\big)$$

Consider the **projectile motion in 2 dimensions with air drag**.
This is a very simple mechanical model.



Specifically, we have two variables: x and y (two dimensions) and define the vector $u = (x, y)$.

The ODE that we are going to solve is:

$$u'' = -\frac{k}{m}u' + g$$

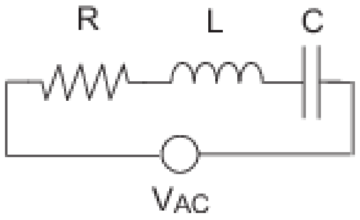
with an initial velocity $u' = (x0, y0)$.

https://rjallain.medium.com/projectile-motion-with-linear-drag-3c489b8045d7
https://ipython-books.github.io/123-simulating-an-ordinary-differential-equation-with-scipy/
https://www.youtube.com/watch?app=desktop&v=fXyuzo0NRvY

Derivation

ODE Example – Series RLC circuit

- Using Kirchoff’s Voltage and Current Laws (KVL, KCL), we can model the simple series RLC circuit shown.
- The resulting ODEs can be written in matrix form and easily solved by hand or numerically (e.g. in Python).
- Matlab Simulink is one possible software for this in addition to many circuit simulators available.



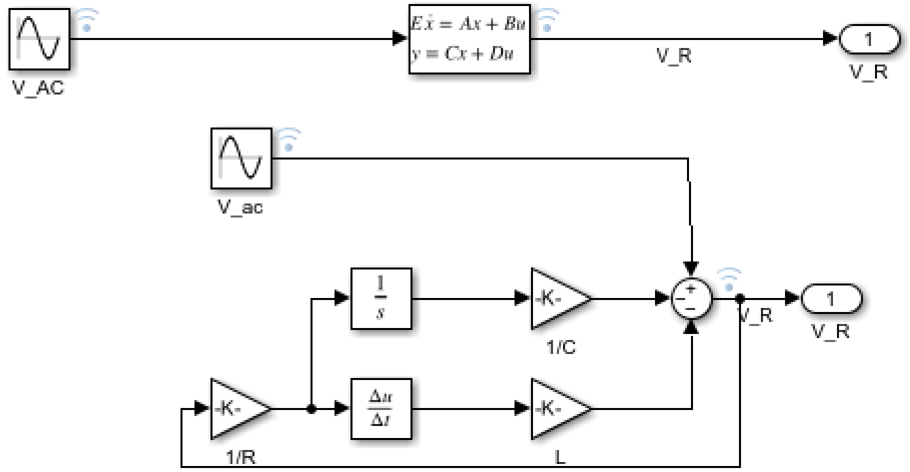
https://au.mathworks.com/help/simulink/slref/model-series-rlc-circuit.html
https://www.intmath.com/differential-equations/8-2nd-order-de-damping-rlc.php

$$V_R = I(t)\,R$$

$$V_L = L\dot{I}_L \text{ or } \dot{I}_L = \frac{1}{L}V_L$$

$$V_C = V_{AC}(0) + \int_0^t I_C(\tau)\,d\tau \text{ or } \dot{V}_c = \frac{1}{C}I_c$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{V}_C \\ \dot{V}_L \\ \dot{V}_R \\ \dot{I}_L \\ \dot{I}_{AC} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{C} & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & R & 0 \\ 0 & \frac{1}{L} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} V_C \\ V_L \\ V_R \\ I_L \\ I_{AC} \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} V_{AC}$$



Derivation

System Simulation and Control

Solving ODEs: Euler's method

- Euler method is a first-order **numerical procedure for solving ODEs with a given initial value**.
- It is the most basic explicit method for numerical integration of ordinary differential equationsh.
- It is an iterative algorithm.
- Step sizes could be uniform $t_{n+1} - t_n = h$ or varying.
- The Euler method can be derived in several ways: Taylor series expansion, geometrically, finite-difference formula for derivatives, etc.

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0$$

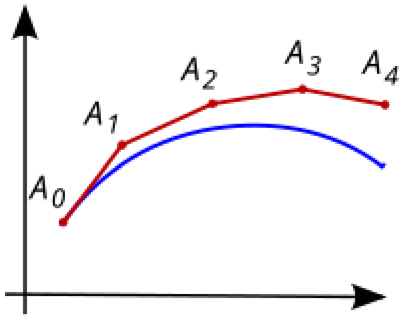
$$\left.\frac{dy}{dt}\right|_{t=t_0} = f(t_0, y_0)$$

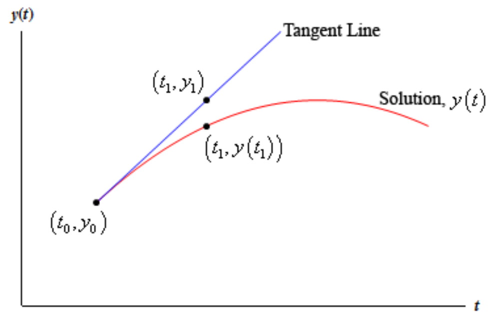
$$y_1 = y_0 + f(t_0, y_0)(t_1 - t_0)$$

$$y_2 = y_1 + f(t_1, y_1)(t_2 - t_1)$$

$$y_{n+1} = y_n + f_n \cdot (t_{n+1} - t_n)$$

1. **define** $f(t, y)$.
2. **input** t_0 and y_0.
3. **input** step size, h and the number of steps, n.
4. **for** j from 1 to n **do**
   (a) $m = f(t_0, y_0)$
   (b) $y_1 = y_0 + h * m$
   (c) $t_1 = t_0 + h$
   (d) Print t_1 and y_1
   (e) $t_0 = t_1$
   (f) $y_0 = y_1$
5. **end**

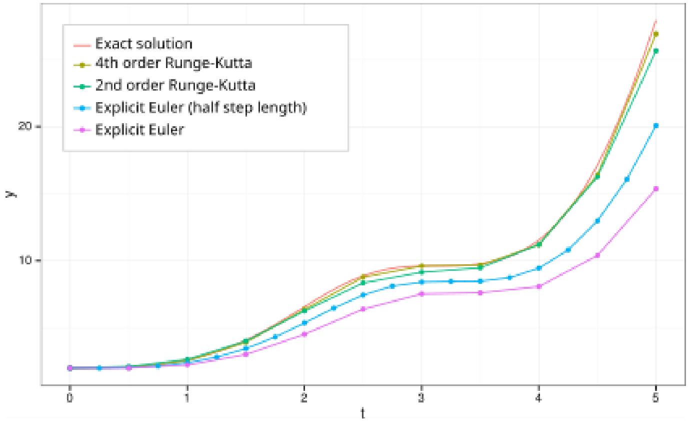




https://en.wikipedia.org/wiki/Euler_method
https://tutorial.math.lamar.edu/classes/de/eulersmethod.aspx
https://www.intmath.com/differential-equations/11-eulers-method-des.php
https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter22.03-The-Euler-Method.html

Solving ODEs: Runge-Kutta (RK) methods

- Runge Kutta (RK) methods are one of the most widely used methods for solving ODEs.
- More accurate as it uses more than one point to extrapolate the solution .
- Fourth Order Runge Kutta (RK4) is widely used.



$$y(x+h) = y(x) + \frac{1}{2}(F_1 + F_2)$$

$$F_1 = hf(x,y)$$

RK2

$$F_2 = hf(x+h, y+F_1)$$

$$y(x+h) = y(x) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)$$

$$F_1 = hf(x,y)$$

RK4

$$F_2 = hf\left(x + \frac{h}{2}, y + \frac{F_1}{2}\right)$$

$$F_3 = hf\left(x + \frac{h}{2}, y + \frac{F_2}{2}\right)$$

$$F_4 = hf(x+h, y+F_3)$$

https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods
https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter22.05-Predictor-Corrector-Methods.html
https://www.intmath.com/differential-equations/12-runge-kutta-rk4-des.php
https://en.wikipedia.org/wiki/Heun%27s_method

Books
https://math.libretexts.org/Workbench/Numerical_Methods_with_Applications_(Kaw)
https://nm.mathforcollege.com/chapter-08-04-runge-kutta-4th-order-method/
https://jonshiach.github.io/ODEs-book/intro.html

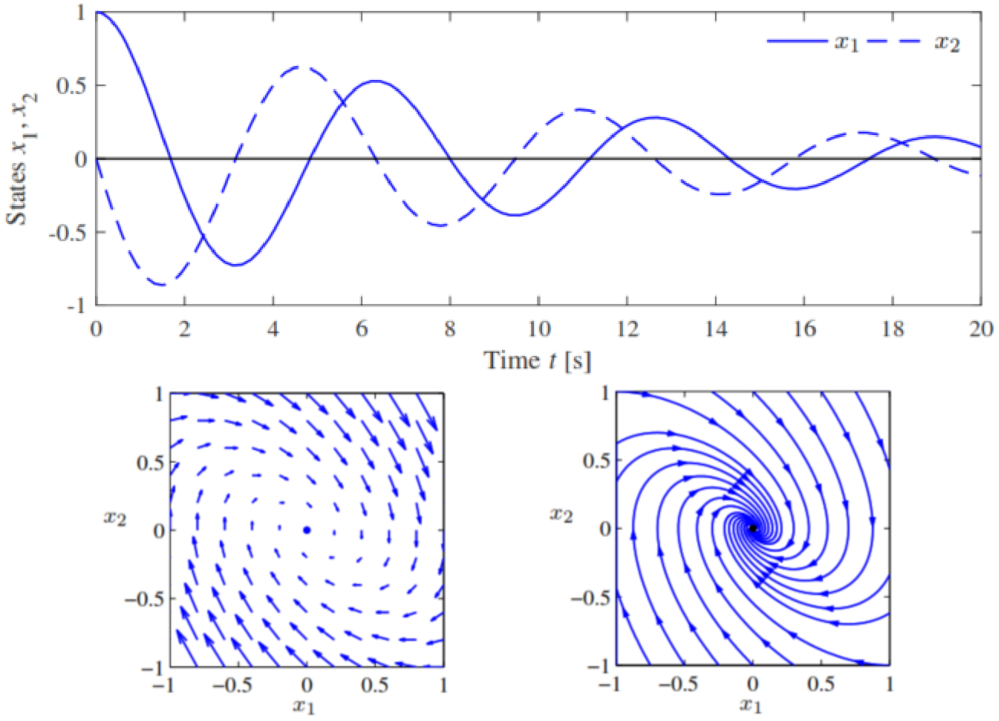
Derivation

System Simulation

- ODE models of the systems can be solved (numerically) to simulate system behaviour.
- Simulators (power, circuit, mechanical) use ODEs and system rules to calculate/predict system behaviour.
- Remember **model ≠ real system**!
- Still, very useful to investigate system evolution
  - with different parameters and starting points
  - by visualising and qualitatively
  - and identify properties like equilibrium points.

$x(t)$ is a solution of the differential equation

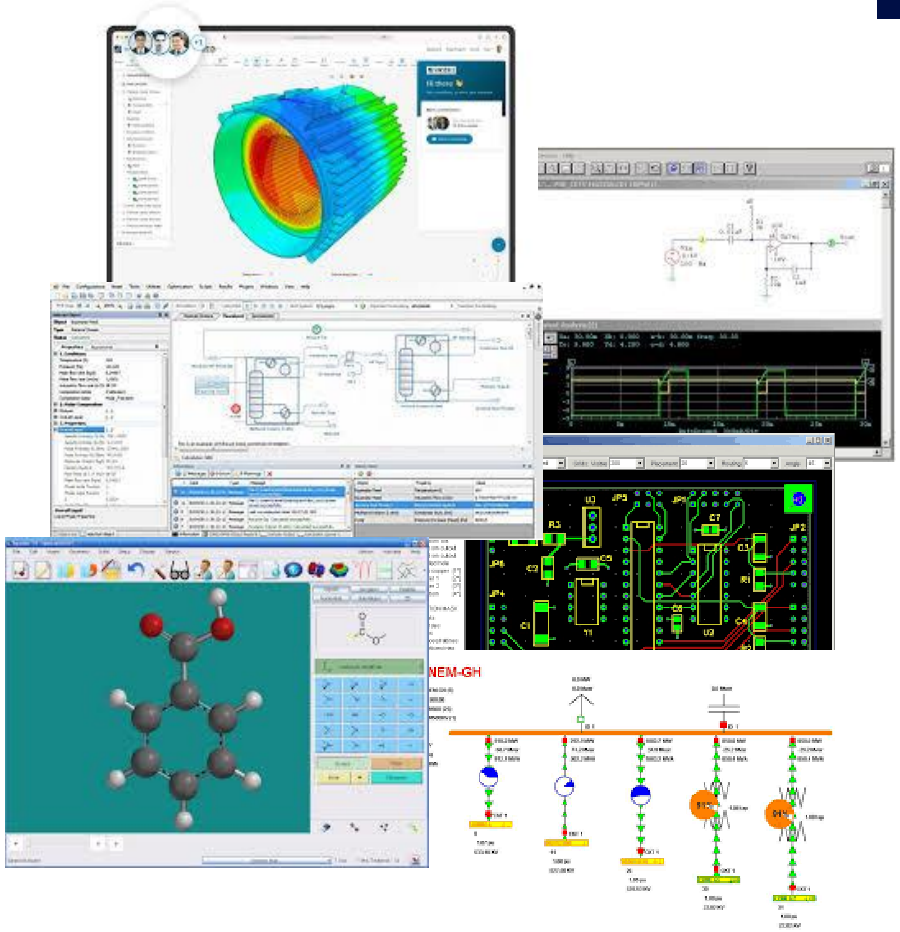
$$x(t_0) = x_0 \quad \text{and} \quad \frac{dx(t)}{dt} = F(x(t)) \quad \text{for all } t_0 < t < t_f.$$



(a) Vector field (b) Phase portrait

Simulators

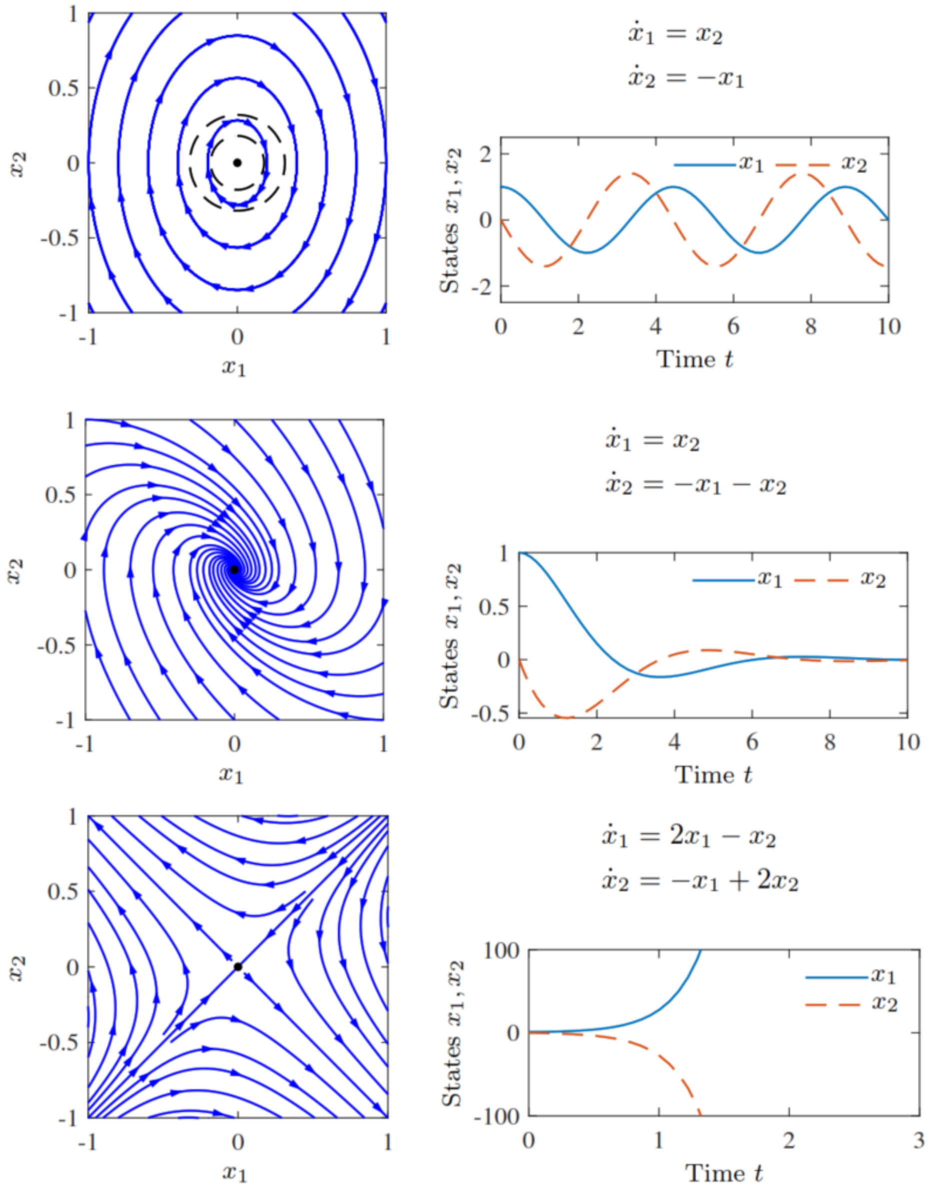
- Modern engineering is heavily reliant on simulations
- They are very complex, very specialised software that may take years to master
- Simulations generate synthetic data!
- Simulators combine engineering and computing!
- Design software may have built-in simulation, but they are not the same thing.
- Circuit, power system, electromagnetic, mechanical, chemical, discrete-event simulators, etc.
- *Simulators do not always give the correct answer, especially in edge cases*.



https://en.wikipedia.org/wiki/List_of_computer_simulation_software

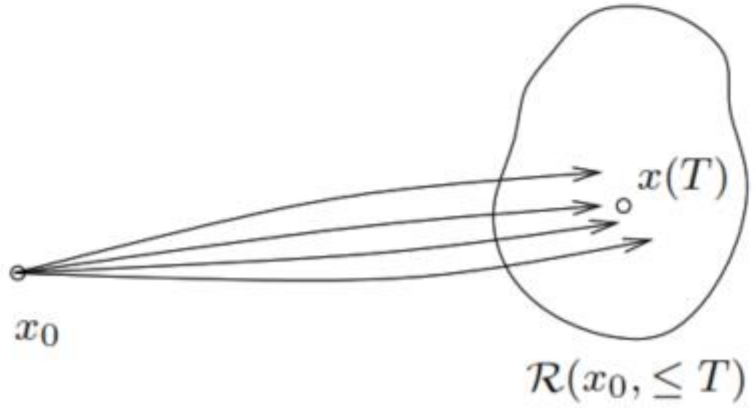
System Stability

- Stability of a solution [trajectory] of an ODE (**model** of a system) determines whether or not other nearby solutions [trajectories] remain close, get closer, or move further away.
  $x(t;a)$ *is stable, if* $\forall\, \epsilon > 0, \exists \delta > 0$ *such that* $\|b-a\| < \delta \;\Rightarrow \|x(t;a) - x(t;b)\| < \epsilon\; \forall t > 0.$
- Special case if a solution is an equilibrium point, $x(t;a) = x_e$
- Asymptotically stable is the same thing but every nearby trajectory eventually approaches the solution
  $$\lim_{t\to\infty} \|x(t;a) - x(t;b)\| = 0.$$
- A solution is unstable if it is not stable.
- Lyapunov functions can be used to show stability.



Control - Reachability

- One of the fundamental properties of a control system is **what set of points in the state space can be reached through the choice of a control input**.
- Define a reachable set from any initial conditions under allowed control inputs.



See Astrom FBS Chapter 7

Definition 7.1 (Reachability). A linear system is *reachable* if for any $x_0, x_f \in \mathbb{R}^n$ there exists a $T > 0$ and $u\colon [0,T] \to \mathbb{R}$ such that if $x(0) = x_0$ then the corresponding solution satisfies $x(T) = x_f$.

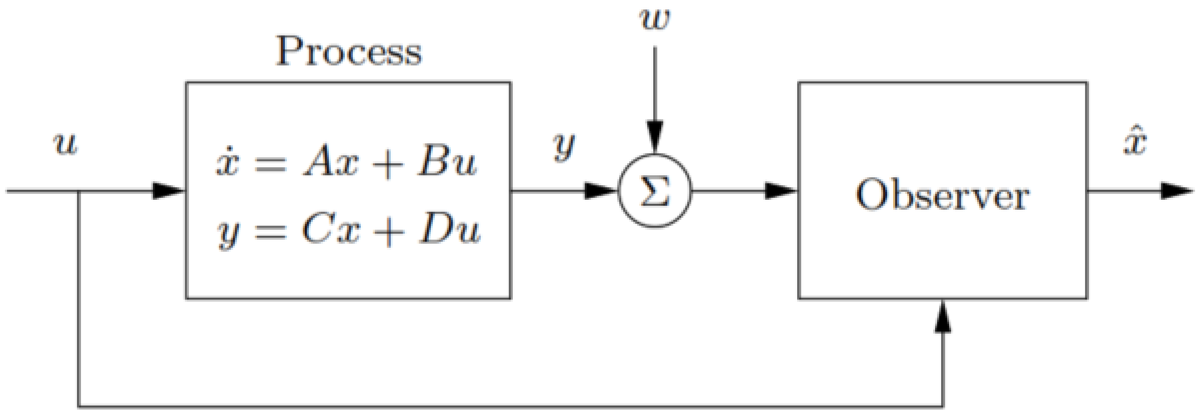
$$\frac{dx}{dt} = Ax + Bu$$

Theorem 7.1 (Reachability rank condition). *A linear system of the form* (7.1) *is reachable if and only if the reachability matrix W_r is invertible (full column rank).*

$$W_r = \begin{pmatrix} B & AB & \cdots & A^{n-1}B \end{pmatrix}$$

Control - Observability

- For many systems, it is highly unrealistic to assume that all the states are measured.
- How can we estimate system states from outputs/observations?
- The observer uses the process measurements (output y) and the inputs u to estimate the current state of the process, $\hat{x}$



See Astrom FBS Chapter 7

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

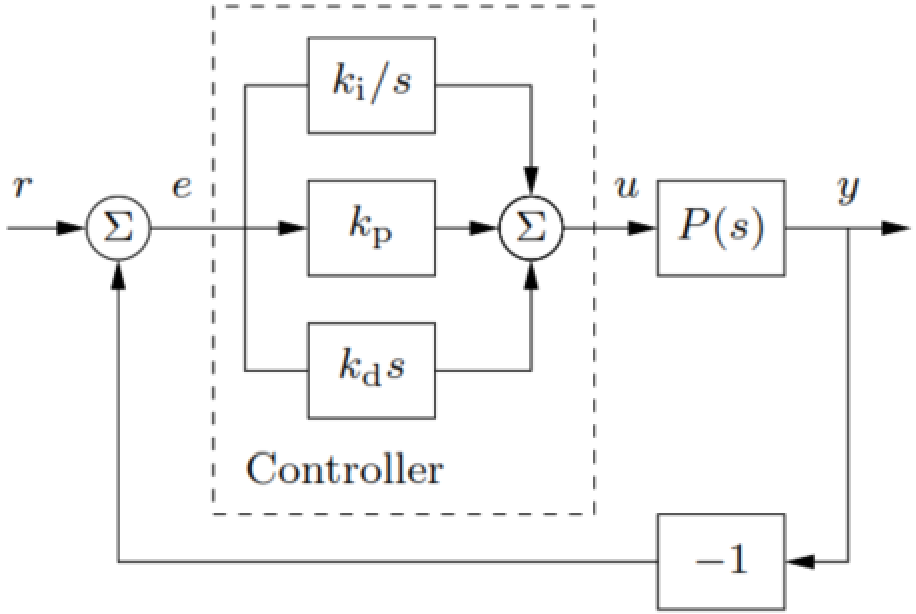
Definition 8.1 (Observability). A linear system is *observable* if for every $T > 0$ it is possible to determine the state of the system $x(T)$ through measurements of $y(t)$ and $u(t)$ on the interval $[0, T]$.

Theorem 8.1 (Observability rank condition). *A linear system of the form (8.1) is observable if and only if the observability matrix W_o is full row rank.*

$$W_o = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

Feedback Control

- We can use feedback control to
  - stabilise a system, e.g. state feedback
  - reduce a cost function, e.g. Linear Quadratic Regulator *(FBS Chapter 7)*
  - track a reference signal, e.g. PID control *(FBS Chapter 11)*
- We may need observers or Kalman filters to estimate system states based on observations *(FBS Chapter 8)*
- Frequency domain analysis is often used (Laplace, Z, Fourier transforms) to analyse control systems. *(FBS Chapters 9,10)*
- PID control is very widely used in industry!



Control Theory - comments

Many positives

- Feedback is at the heart of control theory: “*Feedback is control and control is feedback!*”
- Purely model-based approach, very successful with physical linear (LTI) systems.
- Nonlinear, robust, adaptive, network, hybrid, model-predictive control variants successfully expand upon linear system fundamentals.

Limitations

- Too much focus on abstract models that don’t match modern systems anymore.
- Too much emphasis on theory and applied mathematics rather than real-world systems. Disconnect between theoreticians and practitioners.
- General disdain for computational methods that cannot be easily analysed.
- Linear system thinking shapes the field and limited success with complex, high-dimensional, computational systems.

After this lesson, you should know about

- How to model engineering systems using ODEs
- Numerical solution of ODEs, basics of RK method
- Simulating engineering systems
- System and control theory basic concepts