Power System Dynamic Modelling and Simulation in Python

Julius Susanto and Alireza Fereidouni

Agenda

9:00 – 10:30	Types of models in power systems Mathematical representation of dynamical systems *Exercise 1: Predator-prey equations*
10:30 – 11:00	**Morning Tea**
11:00 – 12:30	Transfer functions and control block diagrams Common primitive blocks and their step response *Exercise 2: Step responses of primitive blocks*
12:30 – 13:30	**Lunch**
13:30 – 15:00	Types of control systems Initialisation of dynamic models *Exercise 3: Initialisation of a hydro turbine governor*
15:00 – 15:30	**Afternoon Tea**
15:30 – 17:30	Structure of an RMS simulation program *Exercise 4: Dynamic performance of a synchronous machine*

Types of models in power systems

There are typically three broad approaches to modelling power systems, each intended to investigate specific power system phenomenon:

1. Steady-state (or static) models

- In steady-state models, the power system is represented at an equilibrium operating point where there is power balance between generation and loads.
- Network branch and shunt elements such as transformers, cables, shunt capacitor banks, etc. are modelled in terms of elementary impedance (R, X) and admittance (G, B) values evaluated at nominal system frequency (or at multiples of nominal frequency for harmonic studies).
- Generators and loads are typically modelled as fixed active and reactive power injections into the system.
- Voltage and power quantities are always represented as root-mean squared (RMS) phasors (i.e. a quantity is described by a magnitude and phase angle or as a complex number such as P + jQ).
- Steady-state models are used for load flow, short circuit and harmonic analyses.

Types of models in power systems

2. RMS (or stability) models

- RMS models are used for dynamic time-domain simulations with RMS phasor quantities to investigate the dynamic electro-mechanical response of a power system after it is subjected to a disturbance or perturbation (e.g. fault, generator trip, motor start, etc.).
- Like in steady-state simulations, network branch and shunt elements in an RMS model are modelled as impedance and admittance values evaluated at nominal system frequency (50 Hz).
- This assumption holds as long as the system frequency remains relatively close to 50 Hz throughout the simulation.
- Generators are modelled as power injections into the system, but with magnitudes that are governed by sets of differential equations (e.g. swing equation and synchronous machine equations).
- Control elements such as AVRs, governors, automatic tap changers, STATCOMs, etc. are also modelled dynamically.
- RMS models are used for transient stability, motor starting, dynamic voltage stability and load rejection analyses.

Types of models in power systems

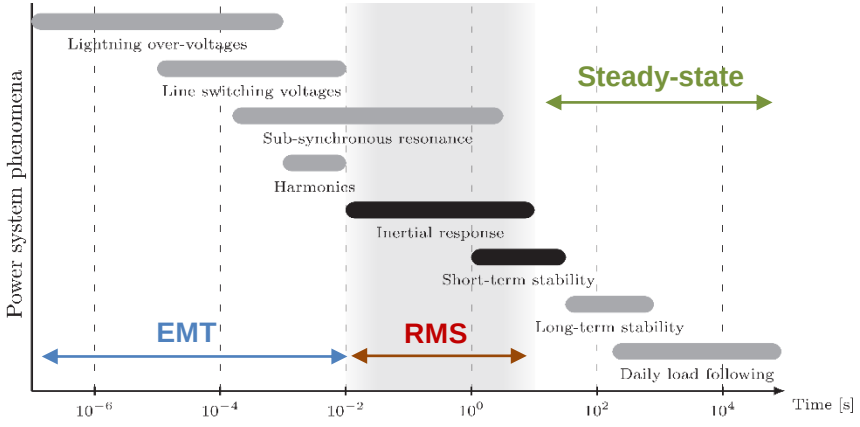
3. Electromagnetic Transient (EMT) models

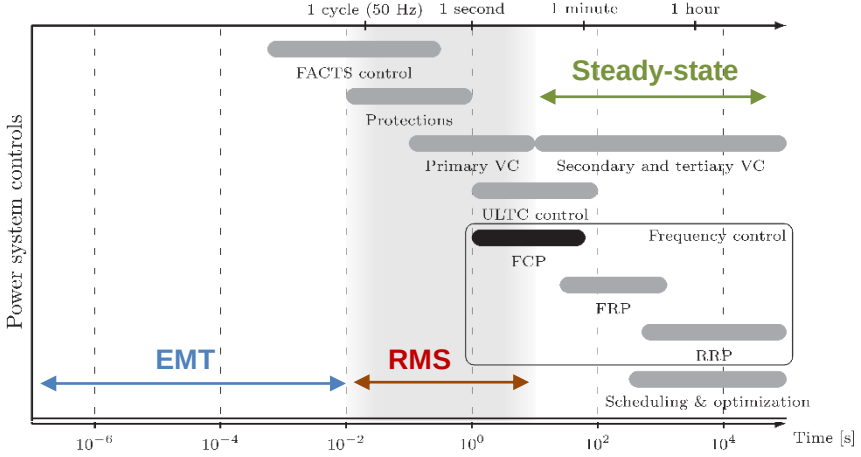
- EMT models are used for dynamic time-domain simulations natural three-phase quantities (e.g. voltage is represented as instantaneous three phase voltages Va, Vb and Vc).
- Unlike RMS models, no assumptions are made about frequency. As a result, network branch and shunt elements are modelled as differential equations (e.g. the voltage and current across an inductor is modelled as $V=L\ di/dt$).
- This allows for the investigation of phenomena such as lightning strikes, switching transients and resonance effects, all of which occur at a broad range of frequencies.
- This is not possible using RMS studies because of the inherent modelling assumption of a near-nominal frequency, i.e. RMS studies can only look at the dynamic behaviour of the system around 50 Hz.
- Depending on the time scale of the study, control elements such as AVRs, governors, etc. may or may not need to be modelled (e.g. a switching or lightning transient is typically too fast for control elements to react and influence the results).

Types of models in power systems

Summary of power system models

Model Type	Quantities	Modelling Requirements	Application Examples
Steady-state	RMS phasors	Simple (impedances and power injections)	Load flow, short circuit, harmonic analysis
RMS / Stability	RMS phasors	Moderately detailed (requires dynamic controller models)	Transient stability, voltage stability, motor starting
Electromagnetic Transients (EMT)	Natural (instantaneous values)	Very detailed	Lightning and switching transient overvoltages, ferroresonance, variable frequency drive ramp up



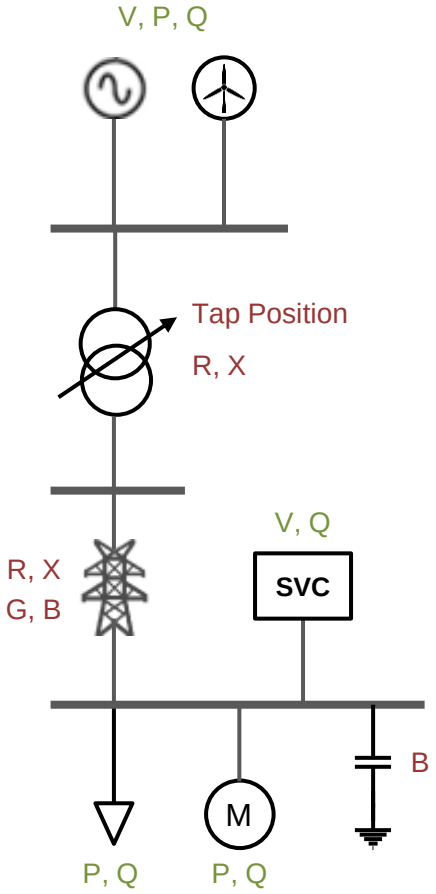


Source: Tielens P., Van Hertem D., “The relevance of inertia in power systems”, Renewable and Sustainable Energy Reviews, Volume 55, March 2016

Dynamic RMS power system models: Steady-state representation

In the steady-state, a power system is typically modelled as follows (i.e. load flow):

- **Generation elements:** are characterised by active and reactive power injections and/or voltage setpoints
- **Transformers:** are passive elements represented by series (and optionally shunt) impedances (that is adjusted by the tap position)
- **Transmission lines and cables:** are passive elements represented by series (R + jX) and shunt (G + jB) impedances
- **Shunt reactive plant:** such as capacitors and reactors are passive elements represented by shunt impedances
- **SVCs and FACTS devices:** are characterised by reactive power injections and/or voltage setpoints
- **Loads:** such as static loads and induction motors are characterised by active and reactive power absorptions

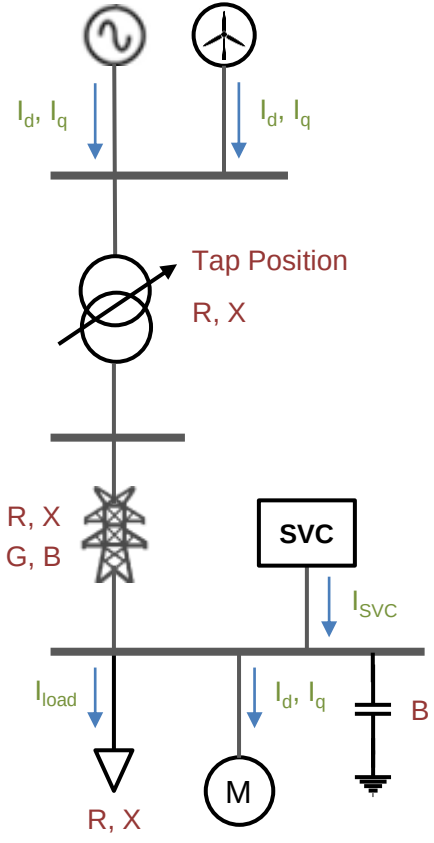


Dynamic RMS power system models: Dynamic representation

In dynamic simulations, the power flows and voltages in the power system change with respect to time (if they weren't, it would be considered steady-state)

Elements in the power system can be categorised as either passive or active elements depending on whether or not they can be represented as impedances (passive) or as dynamic current injections that can change over time or be controlled (active)

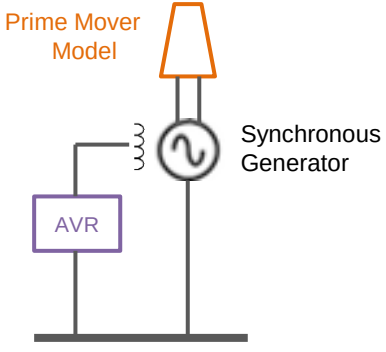
Element	Representation	Controllable	Dynamic RMS Model
Synchronous machine	Active	Yes	Stator magnetic and electrical equations, mechanical (swing) equation
Induction machine	Active	Yes (if generator)	Stator and rotor equations, mechanical load / prime mover equation
Converter-interfaced generation	Active	Yes	Voltage source behind impedance, controllable current injection
Transformer	Passive	Yes	Series / shunt impedances
Lines and cables	Passive	No	Series / shunt impedances
Shunt reactive plant	Passive	No	Shunt impedance
SVC / FACTS	Active	Yes	Either voltage source behind impedance or equivalent shunt impedance
Load	Passive or Active	No	Passive: shunt impedance Active: current injection behind shunt impedance

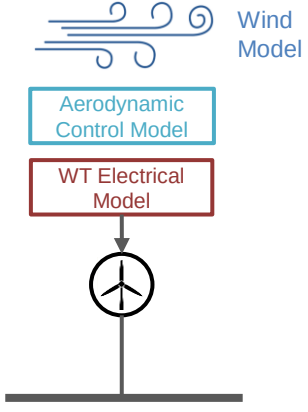


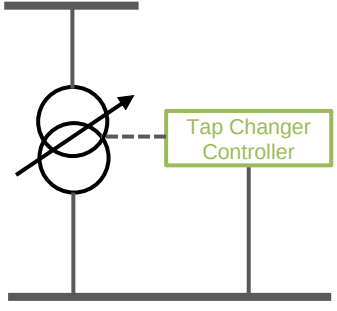
Dynamic RMS power system models: Controllable elements

Power system elements such as synchronous machines, wind turbines, solar PV plants, SVCs and power transformers are “controllable” in the sense that their current injections in a dynamic simulation can be adjusted via a control system.

Controllers are represented by their own dynamic models (more on this later).







Mathematical representation of dynamical systems

In time-domain (RMS) simulations, the most convenient mathematical representation for power system dynamics is a set of differential-algebraic equations (DAE):

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \qquad \mathbf{0} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t)$$

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \vdots \\ \frac{dx_n}{dt} \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n, t) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n, t) \end{bmatrix} \qquad \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} g_1(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n, t) \\ \vdots \\ g_n(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n, t) \end{bmatrix}$$

Differential equations **Algebraic equations**

$$\boldsymbol{x}(t_n) = \begin{bmatrix} x_1(t_n) \\ \vdots \\ x_n(t_n) \end{bmatrix} \qquad \boldsymbol{y}(t_n) = \begin{bmatrix} y_1(t_n) \\ \vdots \\ y_n(t_n) \end{bmatrix} \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 = \begin{bmatrix} x_1(t_0) \\ \vdots \\ x_n(t_0) \end{bmatrix} \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 = \begin{bmatrix} y_1(t_0) \\ \vdots \\ y_n(t_0) \end{bmatrix}$$

State variables **Algebraic variables** **Initial values**

Mathematical representation of dynamic RMS power system models

The dynamic RMS power system model is most commonly formulated with a current injection model* as the algebraic equations (representing the interaction between the network and the dynamic models):

Differential equations: $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v})$

Algebraic equations: $\mathbf{0} = [Y]\boldsymbol{v} - \boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$

$\boldsymbol{x}$ is a vector of state variables

$[Y]$ is the network nodal admittance matrix

$\boldsymbol{v}$ is a vector of nodal voltages (algebraic variables)

$\boldsymbol{i}$ is a vector of (net) nodal current injections

Except for very simple systems, a closed-form analytical solution for this DAE system is not possible and it must be solved using **numerical integration**.

(*) Alternatively, a power injection model of the form $\mathbf{0} = diag(\boldsymbol{v})[Y]\boldsymbol{v}^* - diag(\boldsymbol{v})\,\boldsymbol{i}^*(\boldsymbol{x}, \boldsymbol{v})$ can also be used.

Numerical Integration: Explicit Methods

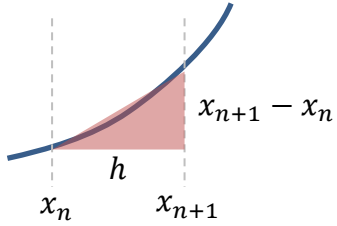
Consider the first-order differential equation: $\frac{dx}{dt} = f(x,t)$

Let's first discretise time such that $t = t_0, t_0 + h, t_0 + 2h, \ldots, t_0 + nh$, where h is the integration step size:

$$\frac{dx_n}{dt} = f(x_n, t_n)$$

The derivative at any time step can be approximated by a simple tangent:

$$\frac{dx_n}{dt} = \frac{(x_{n+1} - x_n)}{h}$$



We can re-arrange this approximate derivative to solve for x at the next time step (*):

$$x_{n+1} = x_n + h\frac{dx_n}{dt} = x_n + h\, f(x_n, t_n)$$

This formulation is called the ***[Naïve]* Euler Method** and is the simplest of the **explicit** numerical integration methods (named as such because we can explicitly formulate the solution to the next time step given an initial condition).

(*) Alternatively, we can derive this from the first two terms of a Taylor series expansion: $x(t_n + h) = x(t_n) + h\frac{dx}{dt} + \frac{h^2}{2!}\frac{d^2x}{dt^2} + \cdots \approx x_n + h\, f(x_n, t_n)$

Numerical Integration: Stability and Accuracy of Explicit Methods

Consider the first-order differential equation: $\frac{dx}{dt} = -\lambda x(t)$ where λ is a real number and $\lambda \geq 0$

Given the initial condition $x(0) = c$, the exact solution is:

$$x(t) = ce^{-\lambda t}$$

Applying the naïve Euler method to solve this ODE:

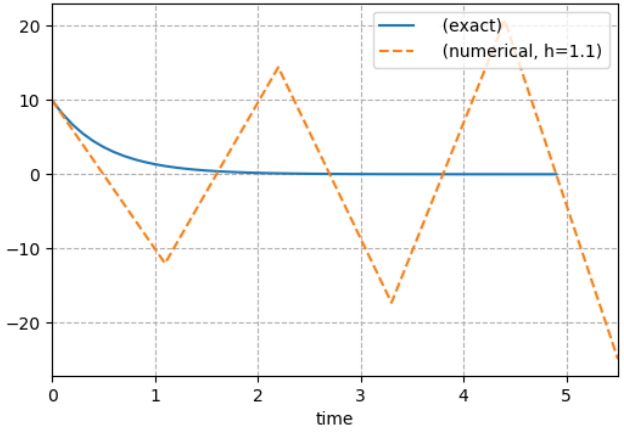
$$\begin{aligned} x_{n+1} &= x_n + h\, f(x_n, t_n) \\ &= x_n - h\lambda\, x_n \\ &= x_n(1 - h\lambda) \\ &= x_{n-1}(1 - h\lambda)^2 \\ &= x_0(1 - h\lambda)^n \end{aligned}$$

Clearly, x_{n+1} is only bounded if $|1 - h\lambda| \leq 1$, or the integration time step $h \leq \frac{2}{\lambda}$

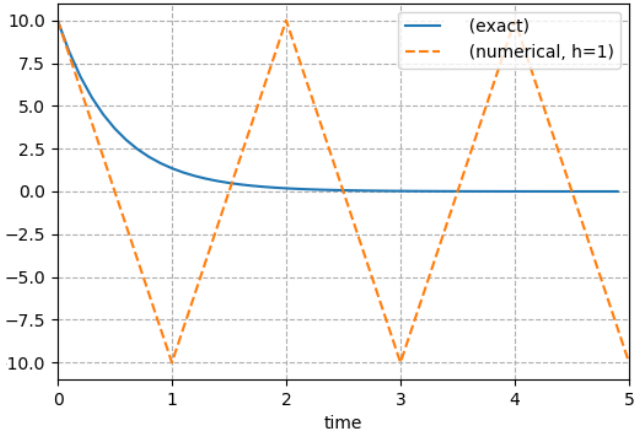
Conclusion: with explicit integrators, the time step must be sufficiently small, otherwise errors will compound at each time step and cause numerical instability.

However, there is a difference between **numerical stability** and **solution accuracy**.

Numerical Integration: Stability and Accuracy of Explicit Methods



Numerically unstable



Numerically stable, but a wild solution

Parameters:

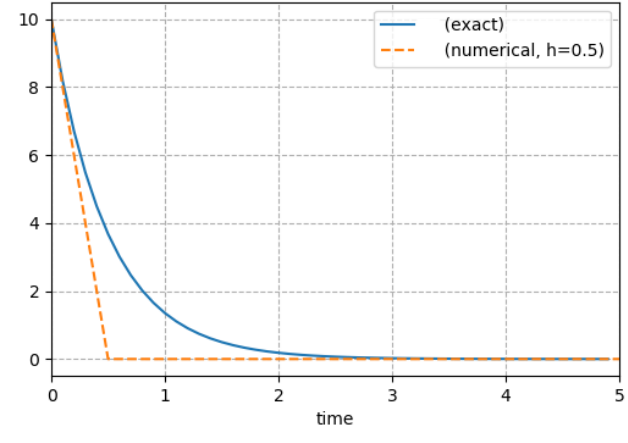
$\lambda = 2, y(0) = 10$

Exact solution:

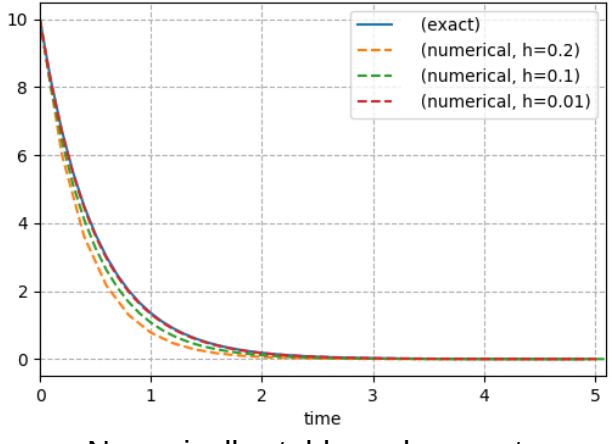
$x(t) = 10e^{-2t}$

For numerical stability:

$h \leq \frac{2}{\lambda} \leq 1$



Numerically stable, but inaccurate

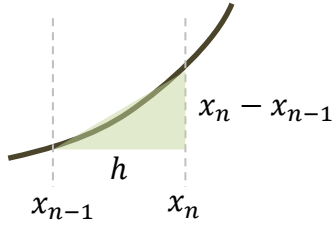


Numerically stable and accurate

Numerical Integration: Implicit Methods

Suppose we changed the way of calculating the derivative by using the previous time step (rather than the next time step):

$$\frac{dx_n}{dt} = \frac{(x_n - x_{n-1})}{h}$$



Given the first-order differential equation $\frac{dx}{dt} = f(x,t)$, we can re-arrange this approximate derivative as follows:

$$x_n = x_{n-1} + h\, f(x_n, t_n)$$

The next time step can be found by setting $n = n + 1$:

$$x_{n+1} = x_n + h\, f(x_{n+1}, t_{n+1})$$

This formulation is called the **Backward Euler Method** and is referred to as an **implicit** method because x_{n+1} appears on both sides of the equation.

Implicit methods typically require an iterative algorithm to solve each time step and are thus have higher computational burden, but they are attractive because of their inherent numerical stability.

Numerical Integration: Stability and Accuracy of Implicit Methods

Again consider the first-order differential equation: $\frac{dx}{dt} = -\lambda x(t)$ where λ is a real number and $\lambda \geq 0$

Applying the Backward Euler method to solve this ODE:

$$x_{n+1} = x_n - h\lambda x_{n+1}$$

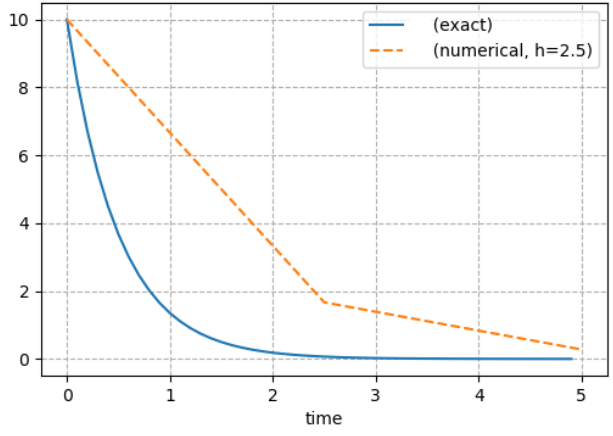
$$= \frac{x_n}{1+h\lambda}$$

$$= \frac{x_{n-1}}{(1+h\lambda)^2}$$

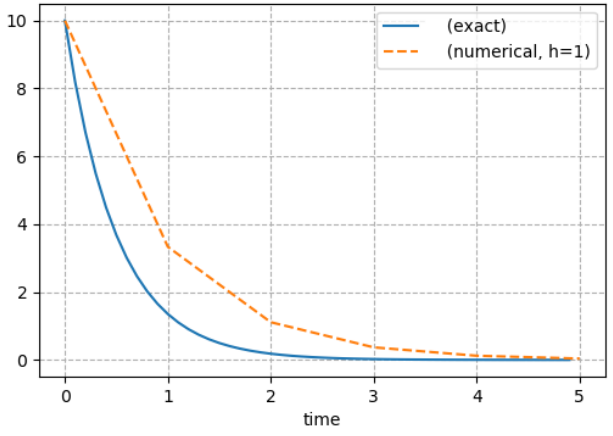
$$= \frac{x_0}{(1+h\lambda)^{n+1}}$$

x_{n+1} is only bounded if $1 + h\lambda \geq 1$ and since $\lambda \geq 0$, then x_{n+1} is stable for any step size $h \geq 0$

Numerical Integration: Stability and Accuracy of Implicit Methods



Numerically stable, but inaccurate



Numerically stable, but inaccurate

Parameters:

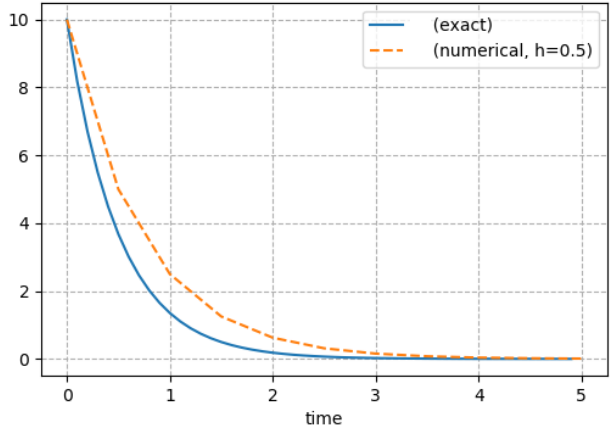
$\lambda = 2, x(0) = 10$

Exact solution:

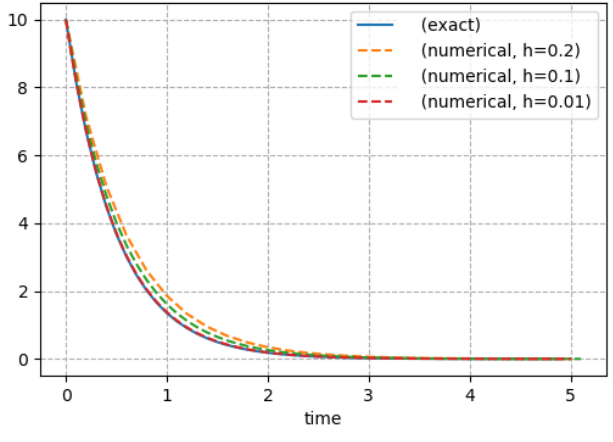
$x(t) = 10e^{-2t}$

For numerical stability:

$h \geq 0$



Numerically stable, but inaccurate

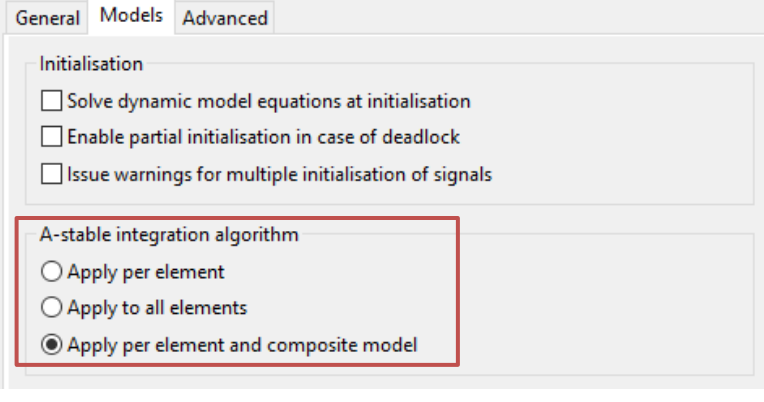


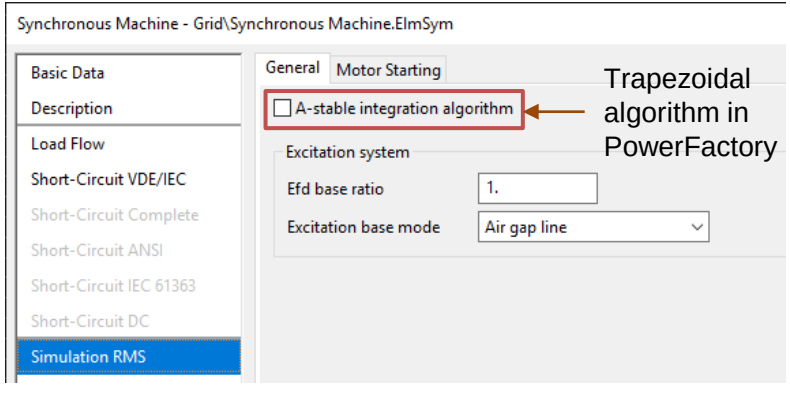
Numerically stable and accurate

Numerical Integration: Summary

Summary of numerical integration approaches

	Explicit Methods	Implicit Methods
Numerical Stability	Stable only at small enough step sizes	Inherently stable (“A-stable”)
Solution Accuracy	Accurate at small enough step sizes, i.e. relative to the largest negative eigenvalue in the system or as a rule of thumb, less than half of the fastest time constant	
Computational Burden	Low – Medium	High (unless problem can be re-arranged to become explicitly formulated)
Algorithm Examples	• Naïve Euler • Heun (predictor-corrector) • (Explicit) Runge-Kutta	• Backward Euler • Trapezoidal • Implicit Runge-Kutta

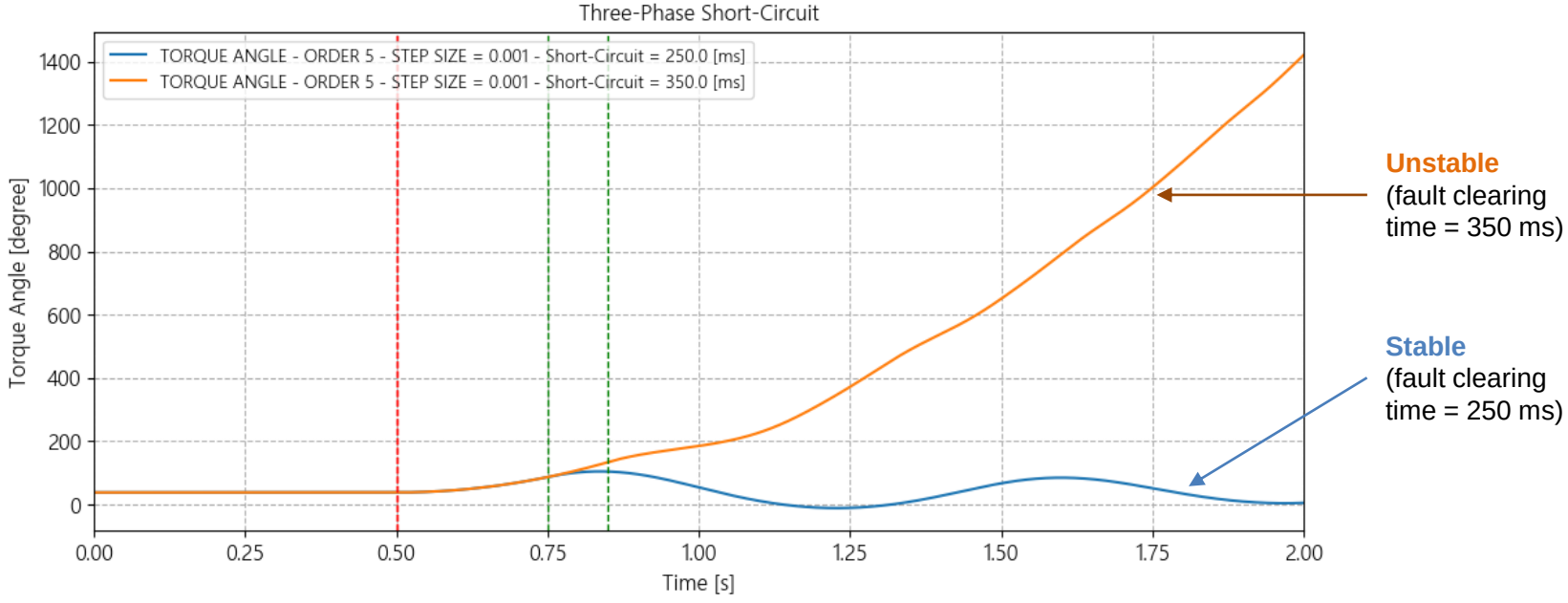




Physical vs Numerical Instability

In dynamic modelling, we need to be careful to make the distinction between physical or numerical instability.

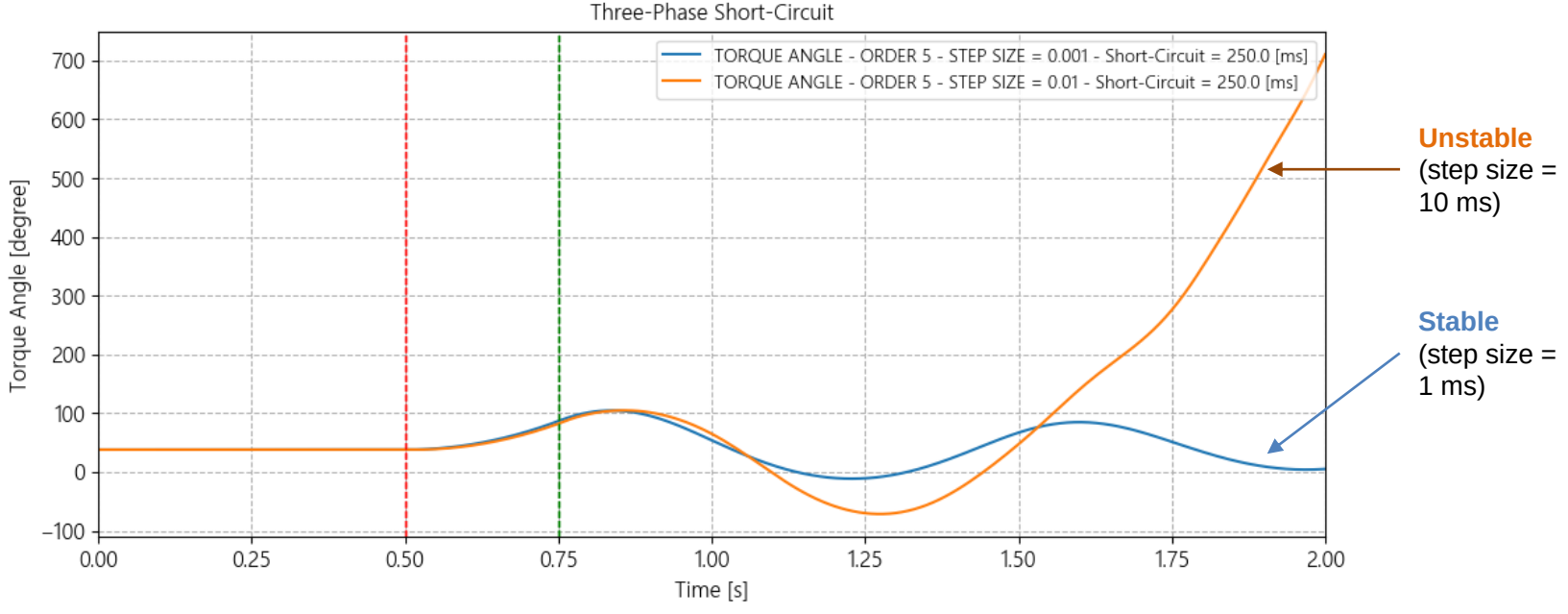
For example, consider the rotor angle stability of a synchronous machine connected to an infinite bus (SMIB) under fault conditions. When the short circuit is applied for longer than the critical fault clearing time (CFCT), the machine loses synchronism with the system, i.e. it is **physical unstable** in that this would happen in real life.



Physical vs Numerical Instability

However, a different result can arise for the same set of conditions by modifying the integration step size.

Below is the same SMIB short circuit simulation with a 250 ms fault clearing time. From the previous slide, we know that this would be physically stable, but by changing the integration step size of the Naïve Euler explicit integrator, we can get a result that is **numerically unstable**.



Exercise 1: Predator-Prey (Lotka-Volterra) Equations

The Lotka-Volterra equations are a pair of differential equations that are often used to describe how natural populations of predators (e.g. foxes) and prey (e.g. rabbits) rise and fall over time.

The basic logic is as follows:

1. The foxes eat the rabbits
2. When there are lots of rabbits, the fox population increases
3. But as the fox population increases, they eat more of the rabbits and the rabbit population decreases
4. After a while, the rabbit population becomes too small to support the fox population and the foxes start dying off
5. As the fox population dwindles, there are less predators for the rabbits and their population rises
6. The cycle starts all over again

Exercise 1: Predator-Prey (Lotka-Volterra) Equations

The Lotka-Volterra equations are as follows:

$$\frac{dx}{dt} = \alpha x - \beta xy \qquad \frac{dy}{dt} = \delta xy - \gamma y$$

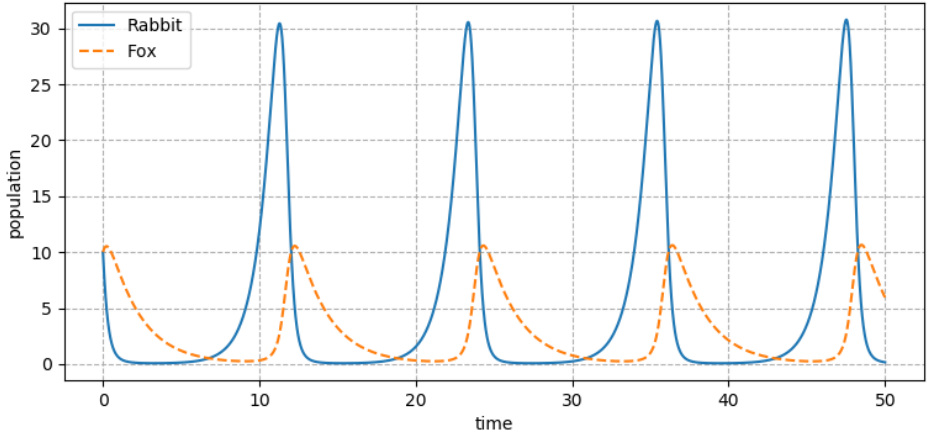
where x is the population of rabbits, y is the population of foxes, α and β are the growth and death rates of the rabbits, and δ and γ are the growth and death rates of the foxes

Exercise

i. Solve the Lotka-Volterra equations with the naïve Euler method

ii. What is a suitable integration step size for this problem?

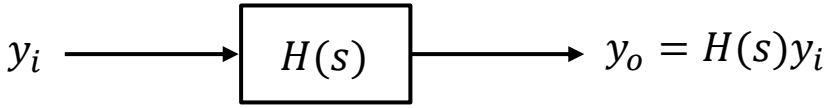
iii. Solve this problem again with the Backward Euler method



$x0 = 10, y0 = 10, \alpha = 1.1, \beta = 0.4, \delta = 0.1, \gamma = 0.5$

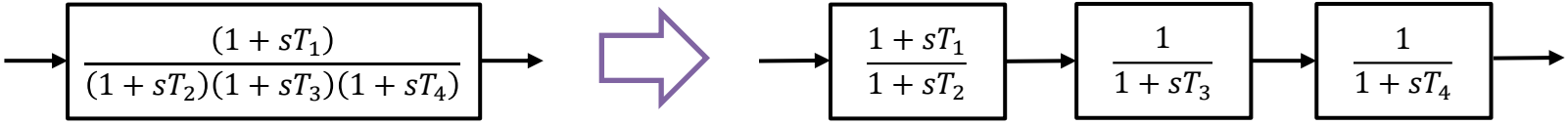
Transfer functions and control block diagrams

In linear time-invariant systems, the **transfer function** $H(s)$ defines the relationship between the input y_i and output y_o of the system in the frequency domain:



where $s = \frac{d}{dt}$ is the differential or Laplacian operator.

In principle, the transfer function can be any arbitrary function, but is usually broken down into more standard “primitive” function blocks for practical purposes and readability:

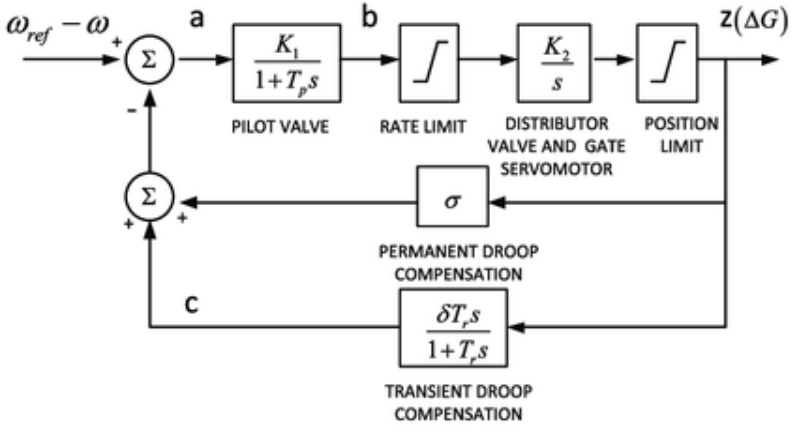


Transfer functions and control block diagrams

Mathematically speaking, a control block diagram is a directed graph where the nodes / vertices are transfer function blocks and edges / arcs are signal flows.

More plainly, a control block diagram is a graphical representation of how primitive blocks are connected together and how they are related to (and combined with) the inputs and outputs of the system.

The control block diagram representation can be used for any arbitrary set of ODEs, but are typically used to describe controllers such as excitation systems, prime mover / governors, converter control systems, etc.

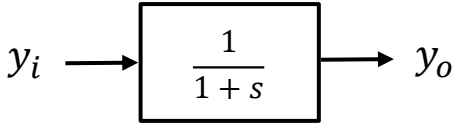


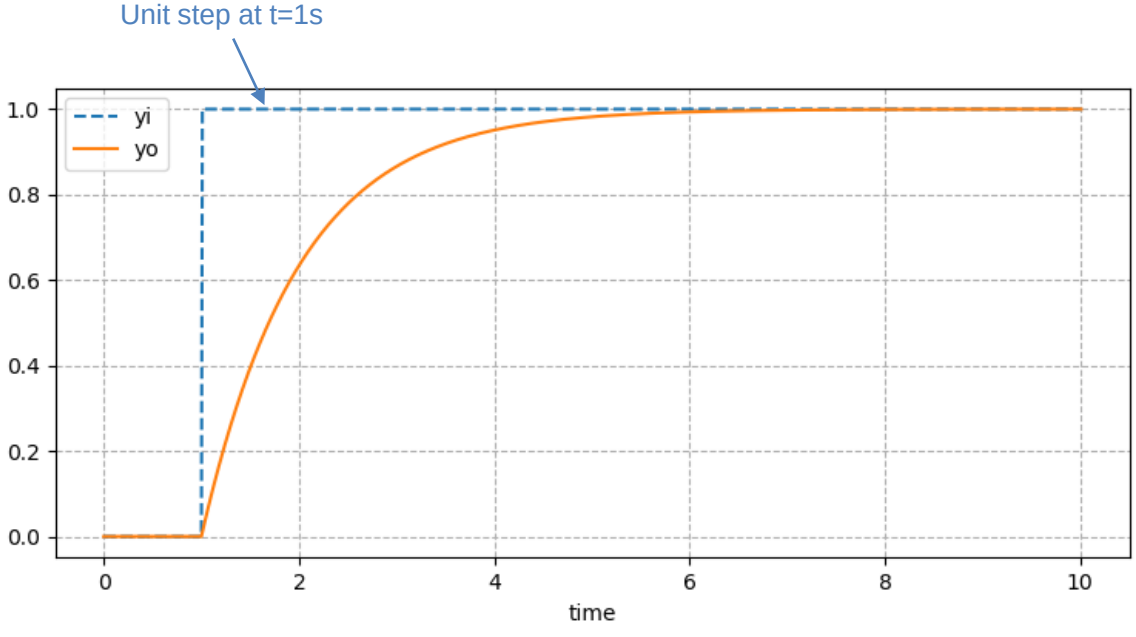
Control block diagram for a hydro turbine governor

Common primitive blocks and their step responses

The step response is a dynamic system's response to a unit step function input.

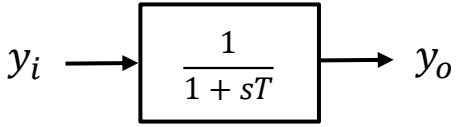
For example, consider the step response for a simple lag block:

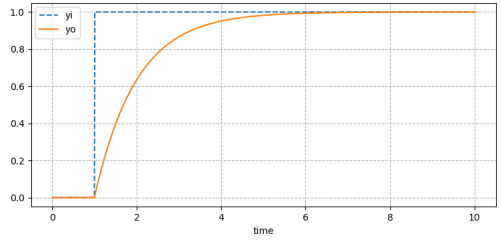




Common primitive blocks and their step responses

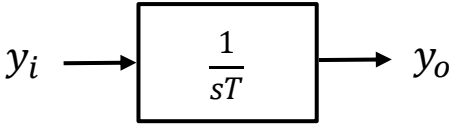
Lag block

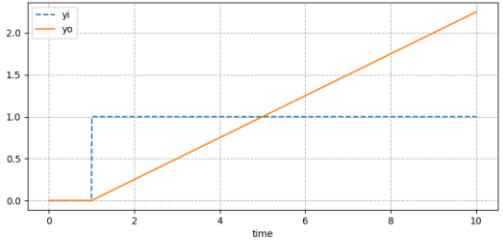




Low pass filter

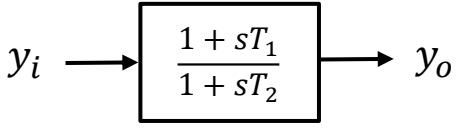
Integrator block

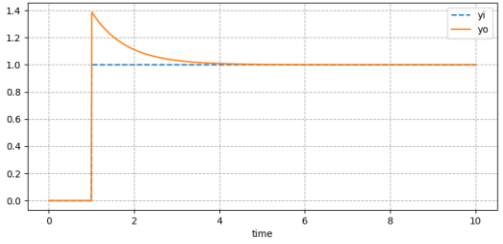




Accumulator

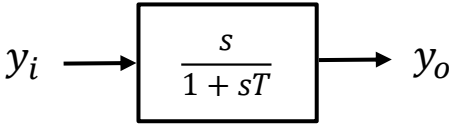
Lead-lag block

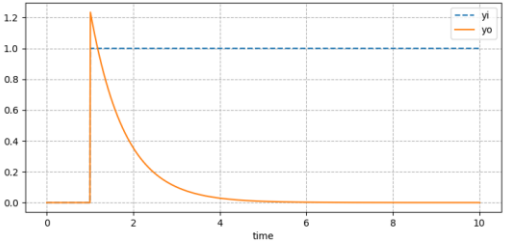




Phase-lead compensation

Washout block (lag-differentiator)

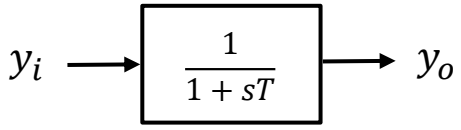




High pass filter

Time-domain equations for primitive blocks: Lag block

Consider the primitive lag block:



Mathematically, this block can be broken down as follows:

$$y_o = \frac{y_i}{(1+sT)}$$

$$y_o(1+sT) = y_i$$

$$sy_o = \frac{y_i - y_o}{T}$$

Recalling that $s = \frac{d}{dt}$, then the lag block is simply the first-order differential equation: $\frac{dy_o}{dt} = \frac{y_i - y_o}{T}$

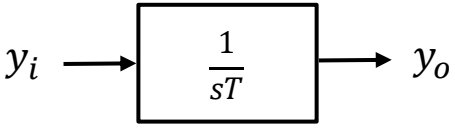
For the sake of completeness, let us define x as the state variable. The lag block can be described as:

$$\frac{dx}{dt} = \frac{y_i - x}{T}$$

where y_i is the input variable and $y_o = x$ is the output variable

Time-domain equations for primitive blocks: Integrator block

Consider the primitive integrator block:



Mathematically, this block can be broken down as follows:

$$y_o = \frac{y_i}{sT}$$

$$sy_o = \frac{y_i}{T}$$

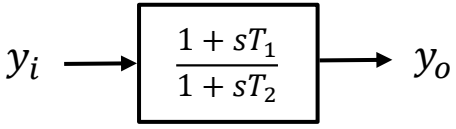
Defining x as the state variable, the integrator block can be described as:

$$\frac{dx}{dt} = \frac{y_i}{T}$$

where y_i is the input variable and $y_o = x$ is the output variable

Time-domain equations for primitive blocks: Lead-lag block

Consider the primitive lead-lag block:



Mathematically, this block can be broken down as follows:

$$y_o = \left(\frac{1+sT_1}{1+sT_2}\right) y_i$$

$$= \frac{y_i}{1+sT_2} + \frac{sy_iT_1}{1+sT_2}$$

Define x such that: $x = \frac{y_i}{1+sT_2}$ then it follows that: $sx = \frac{y_i - x}{T_2}$ or $\frac{dx}{dt} = \frac{y_i - x}{T_2}$

and $y_o = x + sxT_1$

Therefore, with x as the state variable, the lead-lag block can be described as:

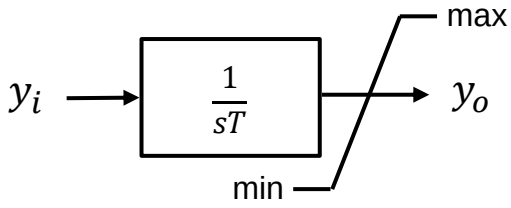
$\frac{dx}{dt} = \frac{y_i - x}{T_2}$ where y_i is the input variable and $y_o = x + \left(\frac{y_i - x}{T_2}\right) T_1$ is the output variable

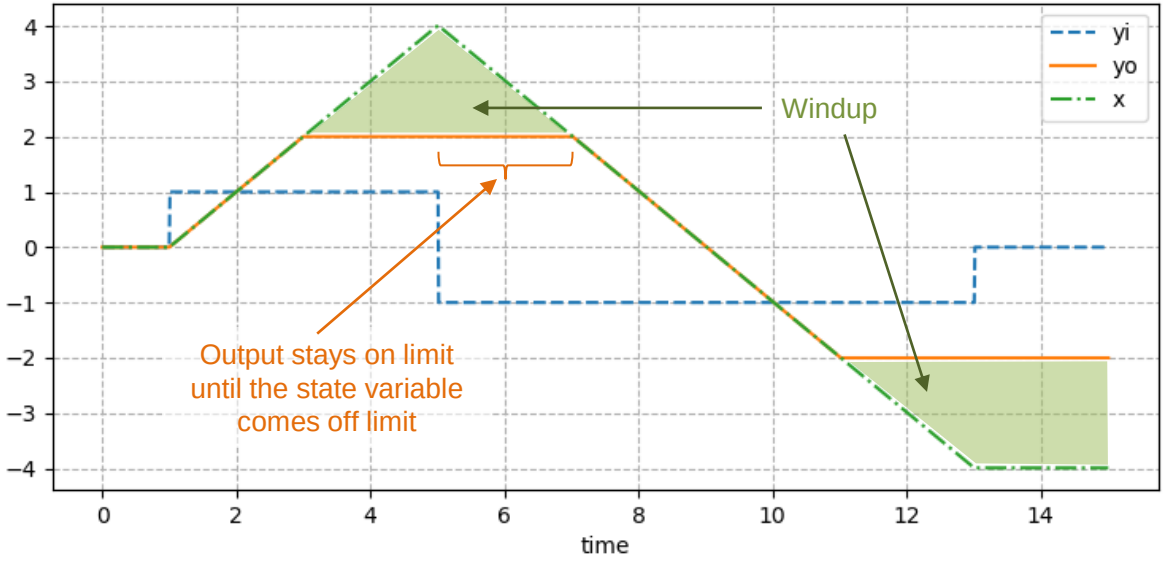
Windup and non-windup limiters

The distinction between windup and non-windup (or anti-windup) limiters is important for limiters associated with integrator, lag and lead-lag blocks. This is because the state variables can accumulate (windup).

Consider the integrator with a windup limiter (with state variable x):

$$\frac{dx}{dt} = \frac{y_i}{T} \qquad y_o = \begin{cases} min & , \quad x < min \\ x & , \quad min \le x \le max \\ max & , \quad x > max \end{cases}$$





Parameters:

T=1

$min = -2$

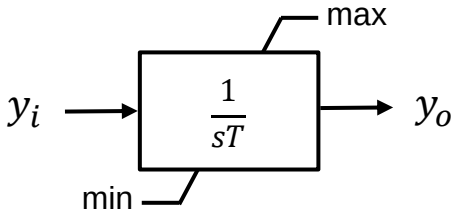
$max = 2$

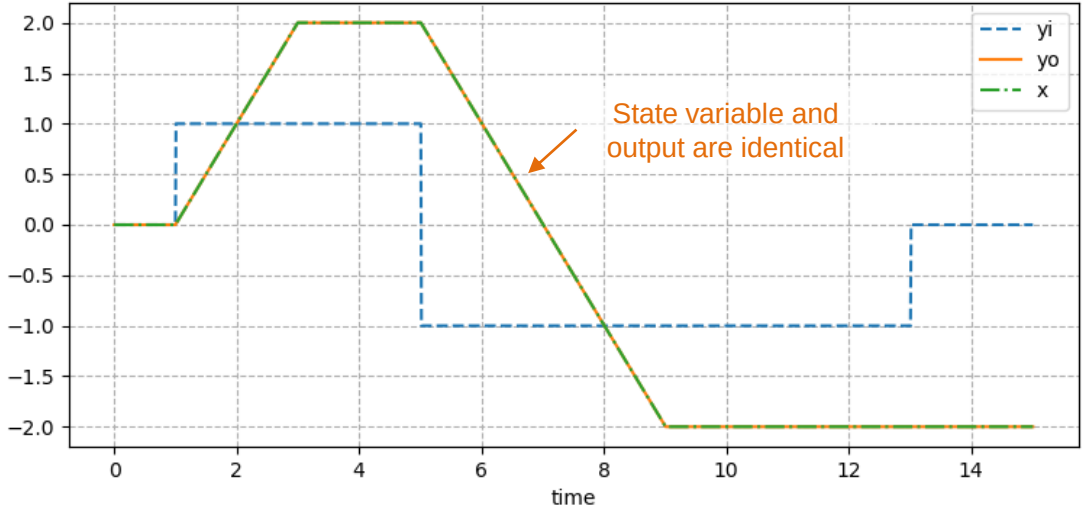
Windup and non-windup limiters

This windup behaviour may not be desirable, particularly if there is a large accumulation of windup (meaning that the output stays on limit for longer).

A non-windup (or anti-windup) limiter can be used that prevents this windup behaviour from occurring by essentially limiting the state variable as well as the output:

$$\frac{dx}{dt} = \begin{cases} 0 & , \quad x < min \\ \frac{y_i}{T} & , \quad min \le x \le max \\ 0 & , \quad x > max \end{cases} \qquad y_o = \begin{cases} min & , \quad x < min \\ x & , \quad min \le x \le max \\ max & , \quad x > max \end{cases}$$





Parameters:

T=1

$min = -2$

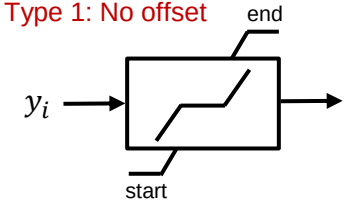
$max = 2$

Dead Bands

A dead band (sometimes called a neutral zone or dead zone) is a band of input values in the domain of a transfer function in a control system or signal processing system where the output is zero (the output is 'dead' - no action occurs).

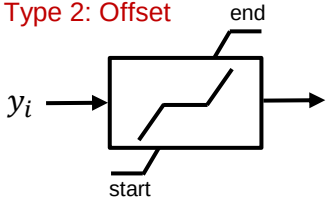
Example: The dead band of a generating unit for their droop control in South-West Interconnected System (SWIS) is 50 (mHz). This means that no droop actions occur when the system frequency is within this range.

Type 1: No offset

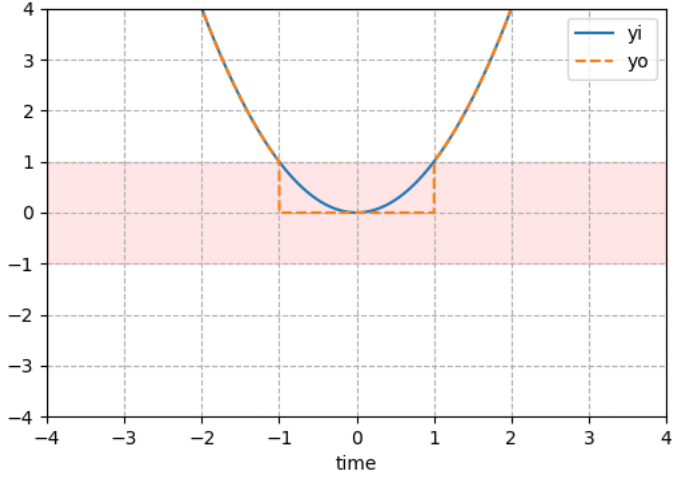


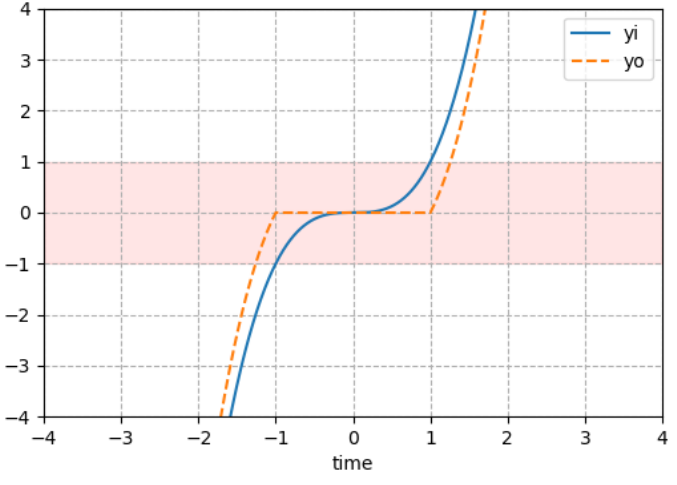
$$y_o = \begin{cases} yi & , \quad yi < start \\ 0 & , \quad start \leq yi \leq end \\ yi & , \quad x > max \end{cases}$$

Type 2: Offset



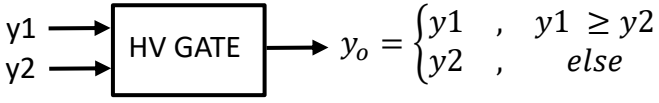
$$y_o = \begin{cases} yi - start & , \quad yi < start \\ 0 & , \quad start \leq yi \leq end \\ yi - end & , \quad x > max \end{cases}$$



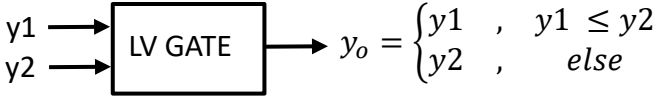


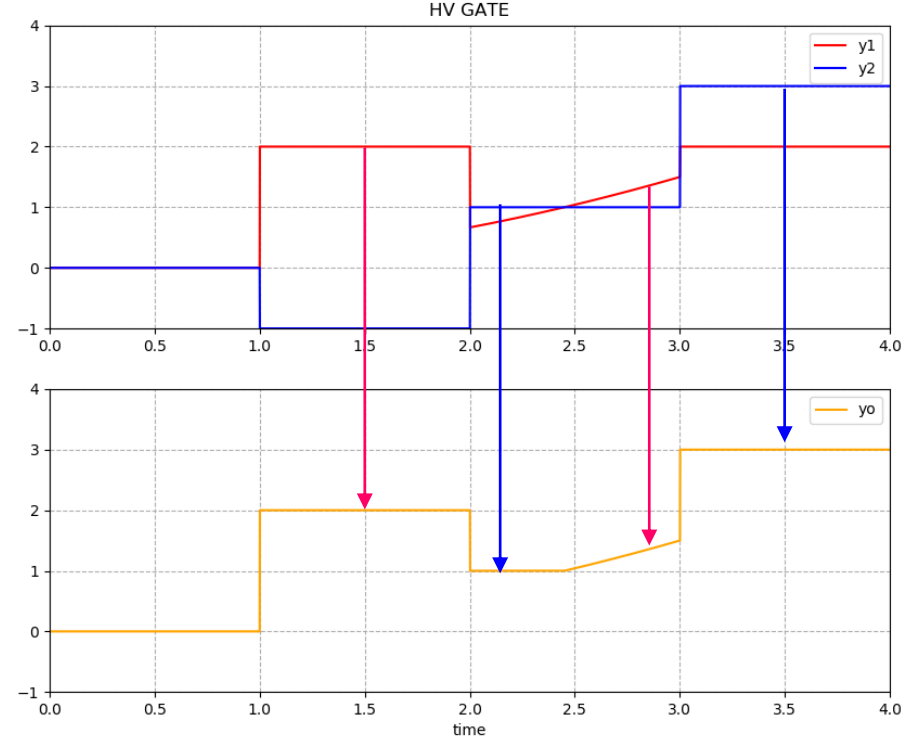
HV and LV Gates

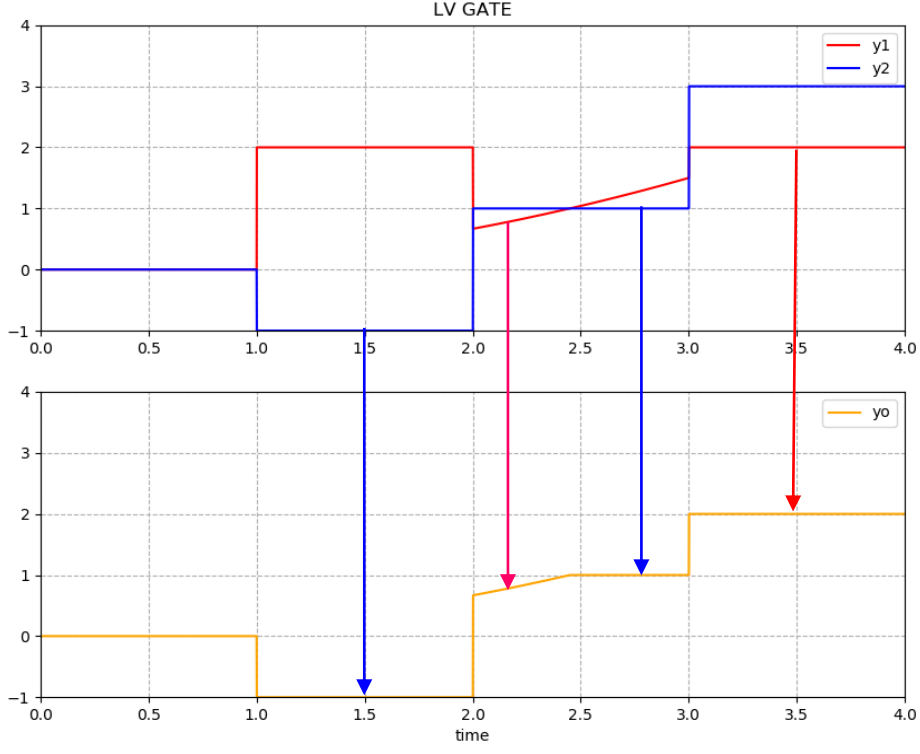
HV Gate: Higher input is passed (Example: UEL).



LV Gate: Lower input is passed (Example: OEL).



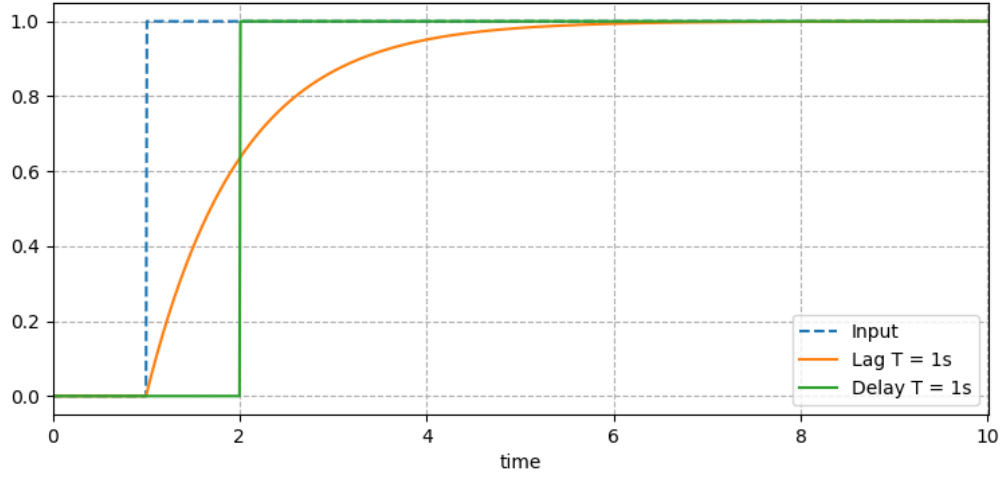




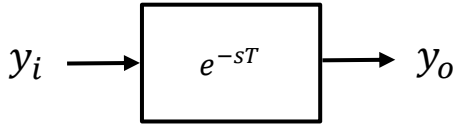
Lags vs time delays

In practical dynamic systems, there is often a time delay (or dead time) between a change in input and the corresponding change in output, e.g. signal communication / propagation time.

This time delay is distinct from a lag block in that a delay is actually a discontinuity in the time domain:



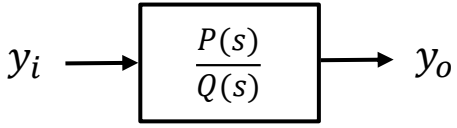
In the frequency domain, the transfer function for a time delay is an exponential function:



Lags vs time delays

From a simulation perspective, time delays are easy to model as $e^{-sT}F(s)$ is simply a time shift in the time domain $\mathrm{f}(\mathrm{t}-T)$, so all that is required is to skip time steps equivalent to T (provided the integration time step $\mathrm{h} < T$)

However, there are applications such as root-locus or Nyquist plot analyses where the poles and zeros of the transfer function are required, i.e. it needs to be represented as a rational function:



And the time delay transfer function e^{-sT} has no poles or zeros (since it is discontinuous in the time domain).

A common technique for overcoming this problem is to approximate e^{-sT} as a continuous-time function that is rational in the frequency domain. This can be done with Padé approximations, which is a method for finding rational function approximations of any function:

$$R(s) = \frac{a_0 + a_1 s + a_2 s^2 + \cdots + a_2 s^m}{1 + b_1 s + b_2 s^2 + \cdots + b_2 s^n}$$

where $R(s)$ is the Padé approximant of order m, n with a_i and b_i the coefficients computed with an estimation algorithm.

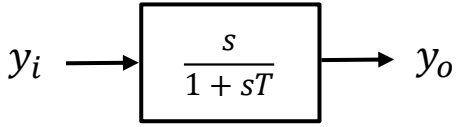
Exercise 2: Step responses of primitive blocks

This exercise is intended to develop intuition on the structure of primitive control blocks and their step responses (e.g. shape and behaviour).

Exercise

i. Adjust the primitive block time constants to answer the following:
   a) For a lag block, what time constant T is required to get a response of 90% within 2s?
   b) For an integrator block, what time constant T is required to reach a value of 20 within 5s?
   c) For a lead-lag block, what time constants T_a and T_b are required to achieve a phase-lead and phase-lag response?

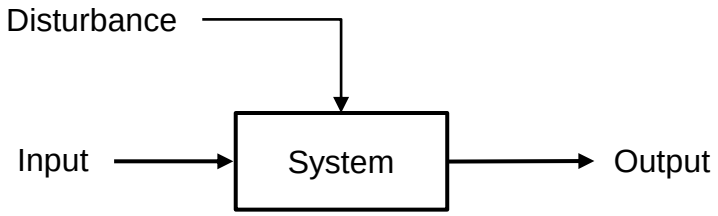
ii. Implement the primitive washout block:
   a) Determine the time-domain equations for a washout block (hint: follow the same logic as the lead-lag formulation)
   b) Generate a step response with $T = 0.8$
   c) Modify the standard washout block with a gain, determine values of K and T for unity gain and different cut-off frequencies



Types of control systems

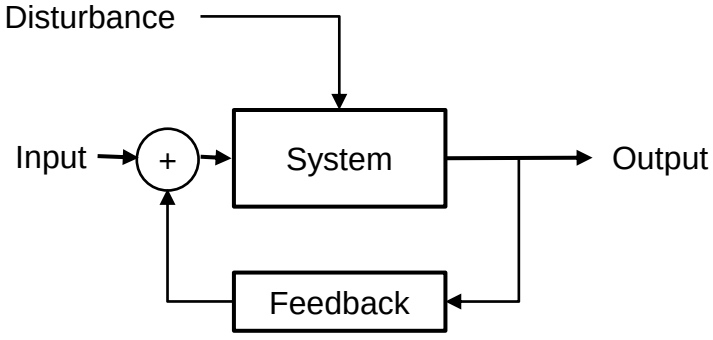
1. Open loop control

- Input is directly converted to an output
- Does not take into account natural / random disturbances
- Can be prone to error if disturbances are large
- Useful for simple systems where the output is predictable, e.g. time-based sprinkler system



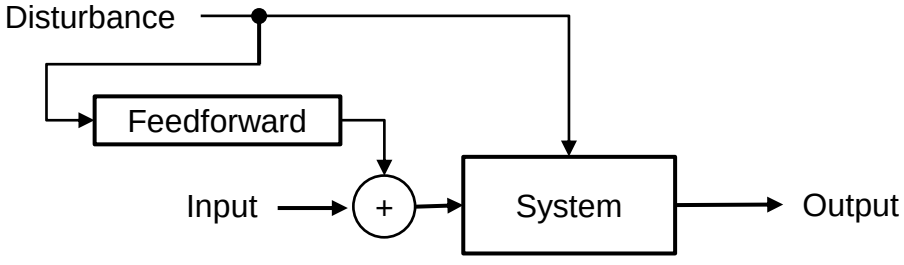
2. Feedback control

- Input is combined with the output (transformed or otherwise)
- Effects of disturbances are manifested in the output signal so the control system only needs to react and does not need to have any a priori knowledge about the nature of the disturbance
- Most common for moderately simple systems



3. Feedforward control

- Input is combined with the disturbance (or a proxy measurement) and the control system compensates for the effects of the disturbance
- Need to know something about the nature of the disturbance and how it affects the system
- Used in complex systems or those with higher-order dynamics, e.g. model-predictive control

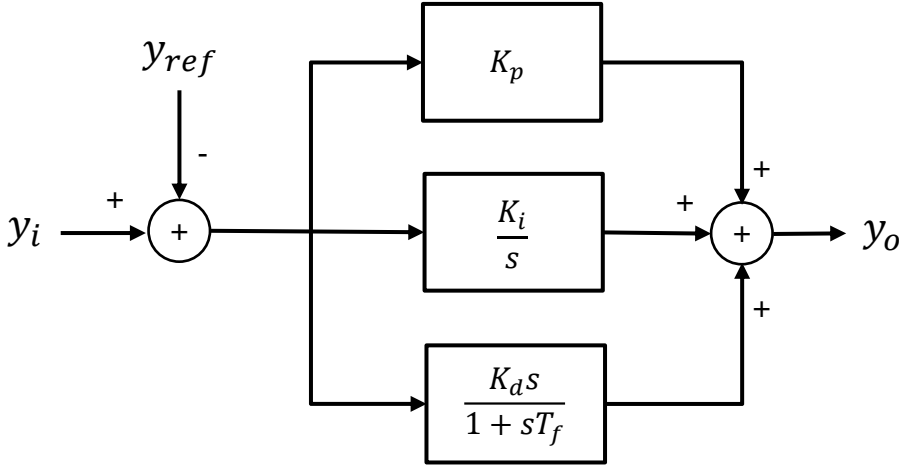


Feedback control systems: PI(D) control

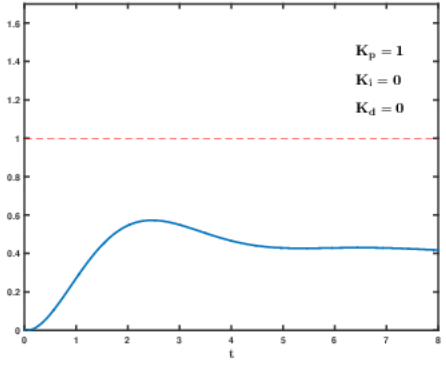
Feedback control systems are the most common type of control found in power systems, and of these systems, the two most common control structures are as follows:

1. PI(D) Control

- Proportional-Integral-(Derivative) control regulates an output to a reference signal
- Generally works well for linear systems, but can fail in non-linear systems, e.g. if there are significant saturation effects
- Derivative term is often neglected, particularly for signals with a lot of high frequency noise (as the derivative term will amplify the noise)

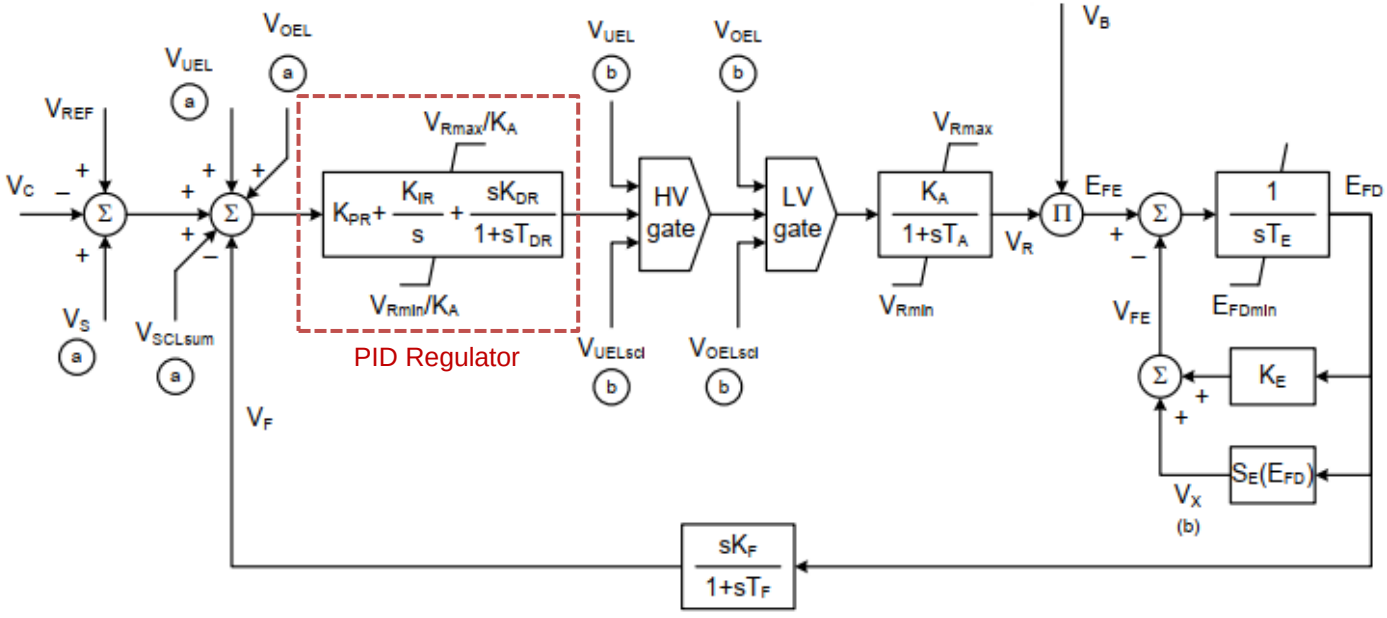


Term	Control Impact of Increasing Value
Proportional	Increases speed of response, but also increases overshoot
Integral	Eliminates steady-state error, but causes oscillations (increased settling time)
Derivative	Decreases overshoot and settling time



Feedback control systems: PI(D) control

Example: Type DC4C dc commutator exciter

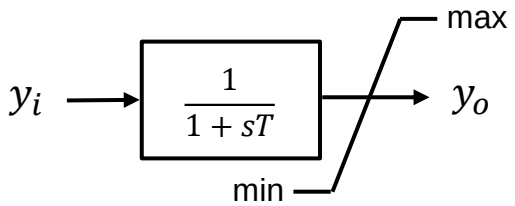


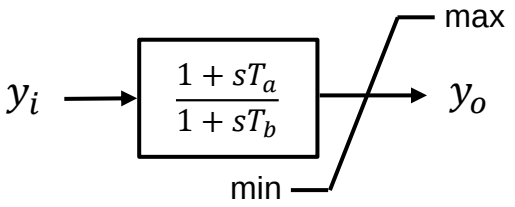
Source: IEEE Std 421.5-2016, IEEE Recommended Practice for Excitation System Models for Power System Stability Analysis

Feedback control systems: Lead-Lag control

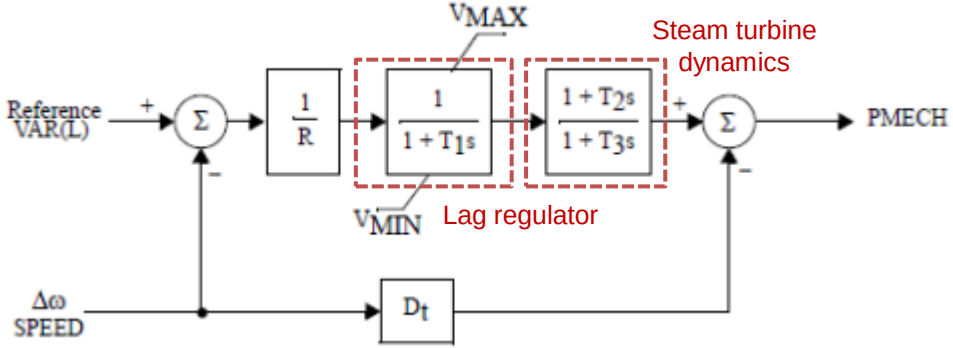
2. Lag or Lead-Lag Control

- Uses combinations of lag and/or lead-lag blocks to regulate an output to an input (reference) value
- Time constants of lead-lag block can be selected to provide phase lead or lag compensation
- Steady-state input into the controller is equal to the output (unlike PI(D) controllers where steady-state input is zero)





Example: Type TGOV1 steam turbine model



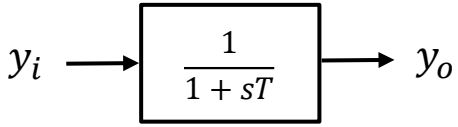
Source: IEEE PES-TR1, Dynamic Models for Turbine-Governors in Power System Studies, Technical Report, 2013

Initialisation of dynamic models

Initialising a dynamic model means setting the state variables such that the model begins in the steady-state or "**flat starts**", i.e. the model outputs do not move around without a disturbance being applied

Mathematically, this means that all derivatives are zero, i.e. $\frac{dx}{dt} = 0$ or $s = 0$

For example, consider the primitive lag block:



We know that the time-domain equations can be written as:

$$\frac{dx}{dt} = \frac{y_i - x}{T} \qquad \text{and} \qquad y_o = x$$

Setting $\frac{dx}{dt} = 0$, we can initialize the model as follows:

$$0 = \frac{y_i - x}{T}$$

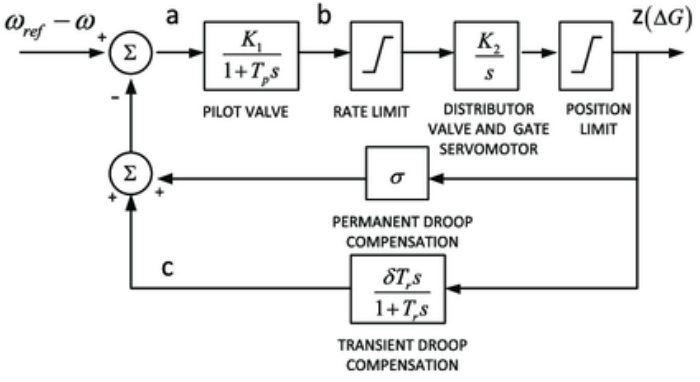
$$x = y_i = y_o$$

Initialisation of dynamic models

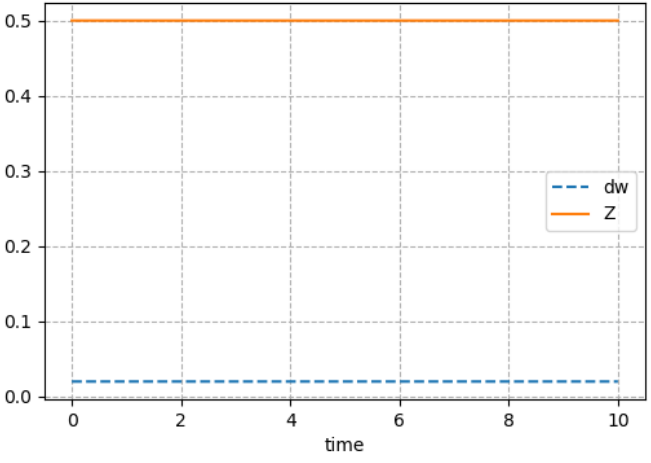
It is good practice to check that your model flat starts by running the simulation without any disturbances / events and verifying that the outputs show no changes over time, i.e. are flat curves.

A flat start indicates that the state variables have been correctly initialised.

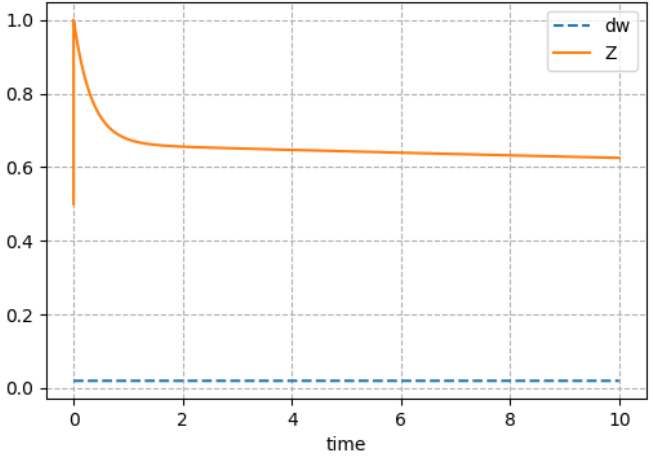
Consider a flat and non-flat start of the hydro turbine governor model introduced earlier:



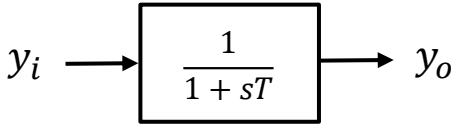
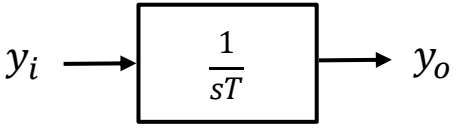
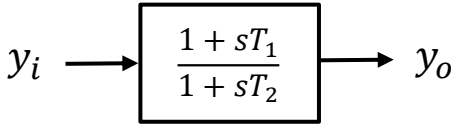
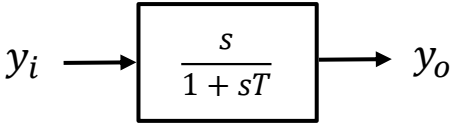
Flat Start



Non-Flat Start



Initialisation of primitive blocks

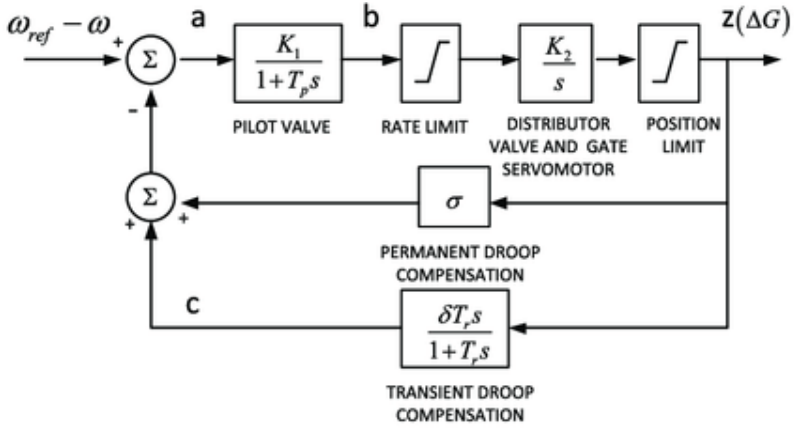
		Input / output initialisation	State variable initialisation (*)
Lag block		$y_i = y_o = y_{o,ss}$	$\frac{dx}{dt} = \frac{y_i - x}{T}$ $x = y_i$
Integrator block		$y_i = 0$ $y_o = y_{o,ss}$	$\frac{dx}{dt} = \frac{y_i}{T}$ $x = y_o$
Lead-lag block		$y_i = y_o = y_{o,ss}$	$\frac{dx}{dt} = \frac{y_i - x}{T_2}$ $x = y_i$
Washout block (lag-differentiator)		$y_i = y_{i,ss}$ $y_o = 0$	$\frac{dx}{dt} = \frac{y_i - x}{T}$ $x = y_i$

(*) Based on the specific formulation of the differential equations.

Exercise 3: Initialisation of a hydro turbine governor

Exercise

i. For the control block diagram of a hydro turbine governor on the right:
   a) Calculate on paper the initial values for the state variables in the model
   b) Test the initial values with the model in Python and check that it flat starts



Control block diagram for a hydro turbine governor

Parameter	Symbol	Value
Pilot valve gain	K_1	1
Pilot valve time constant	T_p	0.6
Rate limiter minimum	r_{min}	0
Rate limiter maximum	r_{max}	1
Distributor valve gain	K_2	1
Position limiter minimum	p_{min}	0
Position limiter maximum	p_{max}	1
Permanent droop compensation	σ	0.04
Transient droop gain	δ	1
Transient droop time constant	T_r	0.5

Initial Value	Symbol	Value
Input	$d\omega = \omega_{ref} - \omega$	0.02
Output	Z	0.5

Structure of an RMS simulation program

As described earlier, an RMS simulation model has the following set of differential-algebraic equations:

Differential equations: $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v})$

Algebraic equations: $\boldsymbol{0} = [Y]\boldsymbol{v} - \boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$

$\boldsymbol{x}$ is a vector of state variables

$[Y]$ is the network nodal admittance matrix

$\boldsymbol{v}$ is a vector of nodal voltages (algebraic variables)

$\boldsymbol{i}$ is a vector of (net) nodal current injections

The **differential equations** represent all of the modelled system (RMS) dynamics, e.g. synchronous machines and their controllers (AVRs, governors, PSS, etc), converter-based generation, SVC/FACTS devices, transformer tap control, etc.

The **algebraic equations** represent the static (passive) elements of the network, e.g. line and cable impedances, fixed shunt and series reactive plant and loads. For a network with n nodes:

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} Y_{11} & \cdots & Y_{1n} \\ \vdots & \ddots & \vdots \\ Y_{n1} & \cdots & Y_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} - \begin{bmatrix} i_1 \\ \vdots \\ i_n \end{bmatrix}$$

The Ybus matrix defines the network structure / topology where diagonal elements are shunt admittances at each node and off-diagonal elements are series admittances between nodes

Bus voltages at each node

Net current injection at each node

Structure of an RMS simulation program: Solving the model

A stylised program flow for a **partitioned solution approach** is shown in the flow chart on the right. In this approach, the differential equations and algebraic network equations are solved separately (i.e. partitioned) and are interfaced within a single time step.

An alternative approach is to solve the differential and algebraic equations simultaneously in each time step, but this is a more complicated approach from a programming perspective (e.g. treatment of non-linear functions).

In RMS simulations, the dynamic models are initialised using steady-state network variables (e.g. voltage, power, etc) from a power flow simulation.

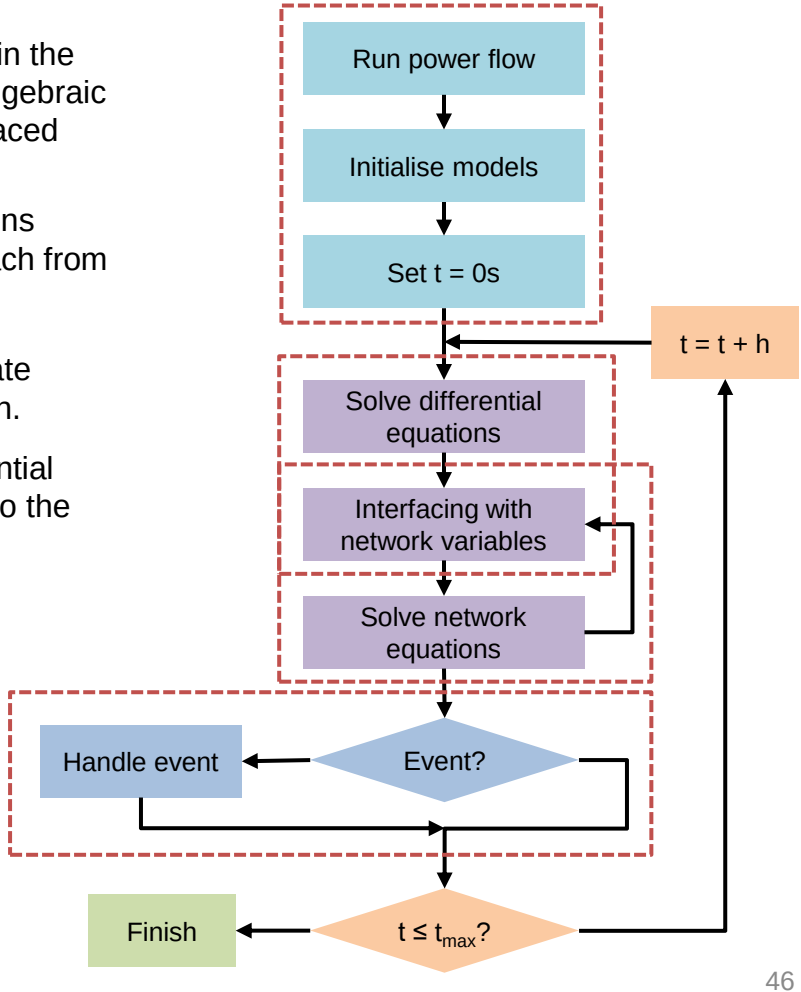
At each time step, the state variables are updated by solving the differential equations. The dynamic models of active elements are then interfaced to the network as current injections (more on this later)

The network equations $\mathbf{0} = [Y]\boldsymbol{v} - \boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$ are solved by calculating current injections $\boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$ using the latest state variables and directly solving for the bus voltages:

$$\boldsymbol{v} = [Y]^{-1}\boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$$

Since the current injections of dynamic models also depend on bus voltages, the network interfacing and network solution steps are done iteratively until the differences between the last two iterations is small.

Events (e.g. faults, contingencies, etc) are handled and may require changes to the Ybus matrix, e.g. after network reconfiguration.



Structure of an RMS simulation program: Network interfacing

Network interfacing refers to the interfacing of dynamic active elements (e.g. generators, SVC/FACTS devices, induction machines, etc) to the network as current injections. This typically involves two steps:

1. Reference frame transformation

- Voltages and currents of active elements such as synchronous machines and converters are typically expressed in dq coordinates with an individual (local) reference frame.

$$\begin{bmatrix} V_r \\ V_i \end{bmatrix} = \begin{bmatrix} \cos\delta & -\sin\delta \\ \sin\delta & \cos\delta \end{bmatrix} \begin{bmatrix} V_q \\ V_d \end{bmatrix}$$

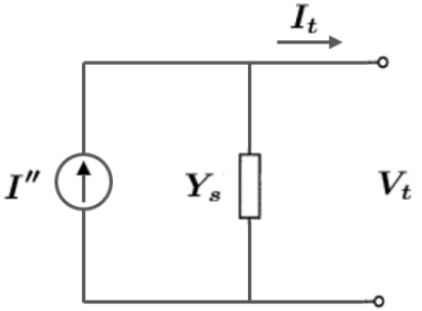
- However, it is convenient to express network quantities in a common synchronously rotating reference frame (e.g. real-imaginary phasor coordinates)

$$\begin{bmatrix} V_q \\ V_d \end{bmatrix} = \begin{bmatrix} \cos\delta & \sin\delta \\ -\sin\delta & \cos\delta \end{bmatrix} \begin{bmatrix} V_r \\ V_i \end{bmatrix}$$

- Therefore, a reference frame transformation is necessary to interface active elements to the network

2. Current injection via Norton equivalent circuit

- Active elements are typically interfaced to the network via a source impedance that is incorporated directly into the Ybus matrix as a source admittance $\boldsymbol{Y}_s$



- This is depicted as a Norton equivalent circuit where $\boldsymbol{V}_t$ is the bus voltage, $\boldsymbol{I}_t$ is the current injection from the active element (based on its state variables) and $\boldsymbol{I}''$ is the total current injection considering the source admittance (“pseudo-current”):

$$\boldsymbol{I}'' = \boldsymbol{I}_t + \boldsymbol{Y}_s \boldsymbol{V}_t$$

Structure of an RMS simulation program: Treatment of loads

The Ybus matrix is inherently near-singular (i.e. non-invertible) and is actually singular when there are no shunt elements connected to the network (e.g. shunt reactive plant, lines with shunt branches, etc).

As a result, solving the network equations directly by $\boldsymbol{v} = [Y]^{-1}\boldsymbol{i}(\boldsymbol{x}, \boldsymbol{v})$ would not possible and it is desirable to make the Ybus matrix less ill-conditioned and more numerically stable by adding shunt elements to the main diagonal.

This can be done by interfacing active elements via a source admittance (as described previously), but another common approach is to also incorporate PQ loads directly into the Ybus matrix as constant impedance (Z) shunt elements.

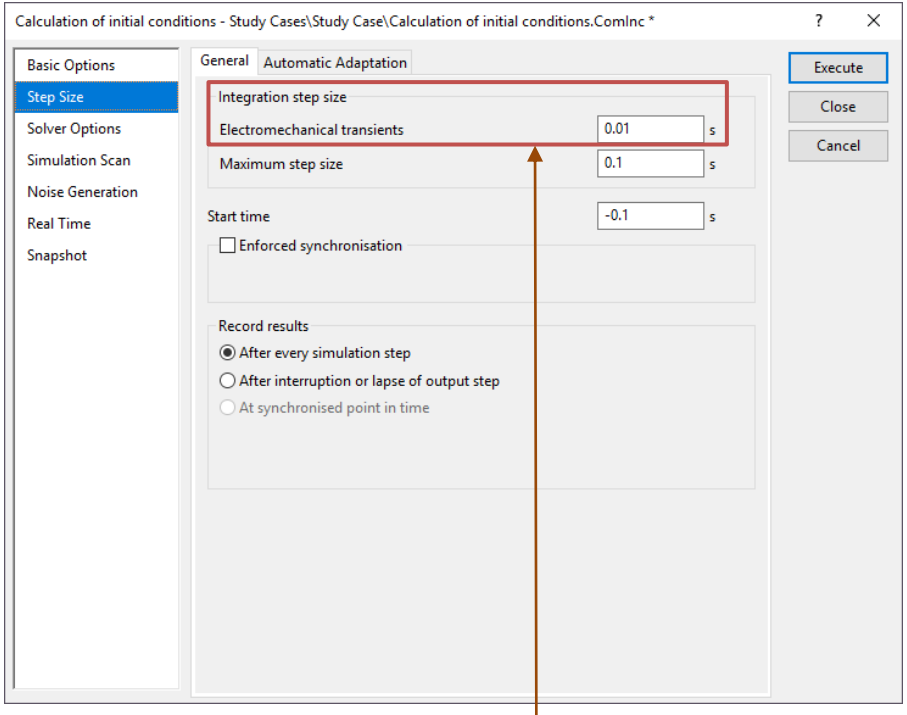
This is normally done after the power flow initialisation by the following calculation for a load connected to bus i:

$$\boldsymbol{Y}_{load} = \frac{P_i - jQ_i}{\boldsymbol{v}_i^2}$$

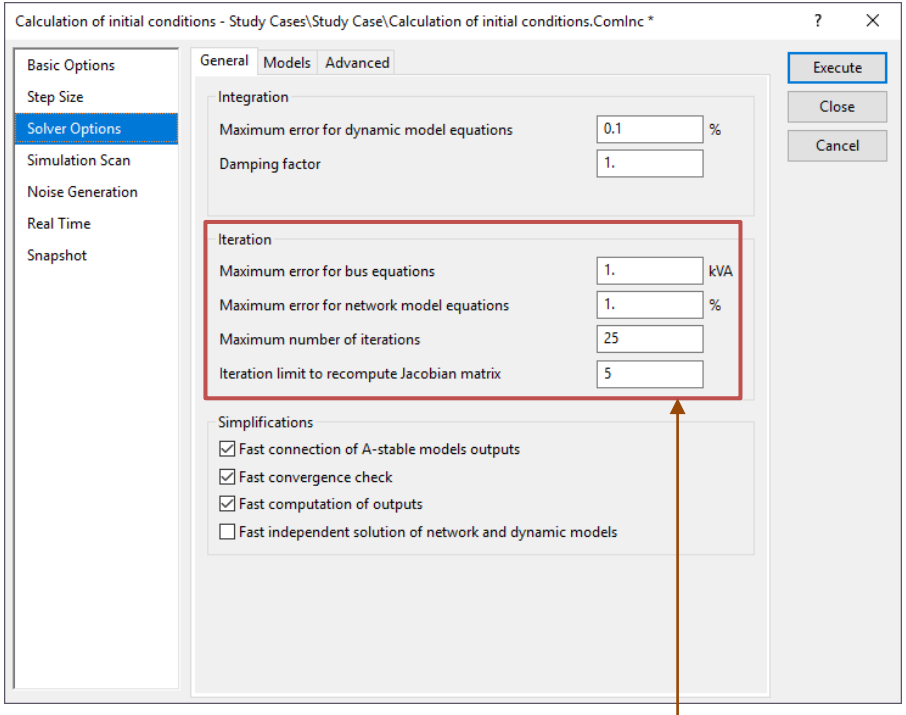
Key implications: this is a standard approach in virtually all RMS simulation programs (e.g. PowerFactory, PSSE, ETAP, etc) and it is important to note that loads defined as PQ in the power flow are no longer treated as constant power, but constant impedance in the RMS simulation.

What this means practically is that in an RMS simulation, loads will tend to decrease when voltages fall (and vice versa), i.e. voltage dependencies need to be built in specifically for loads.

RMS simulation options in PowerFactory

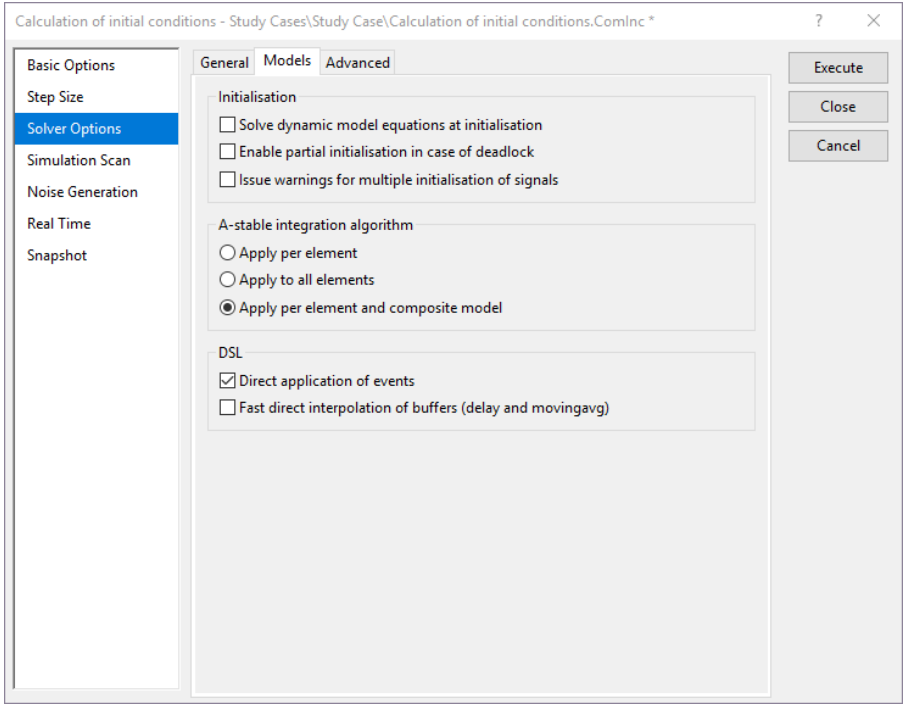


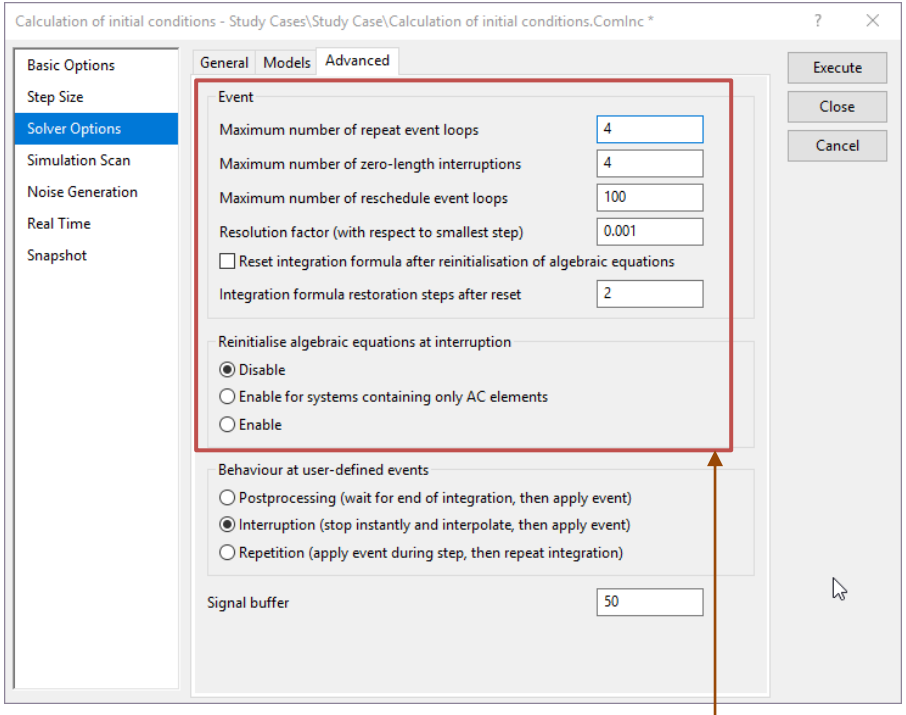
Integrator step size h



Parameters for interfacing with network variables and solving network equations

RMS simulation options in PowerFactory



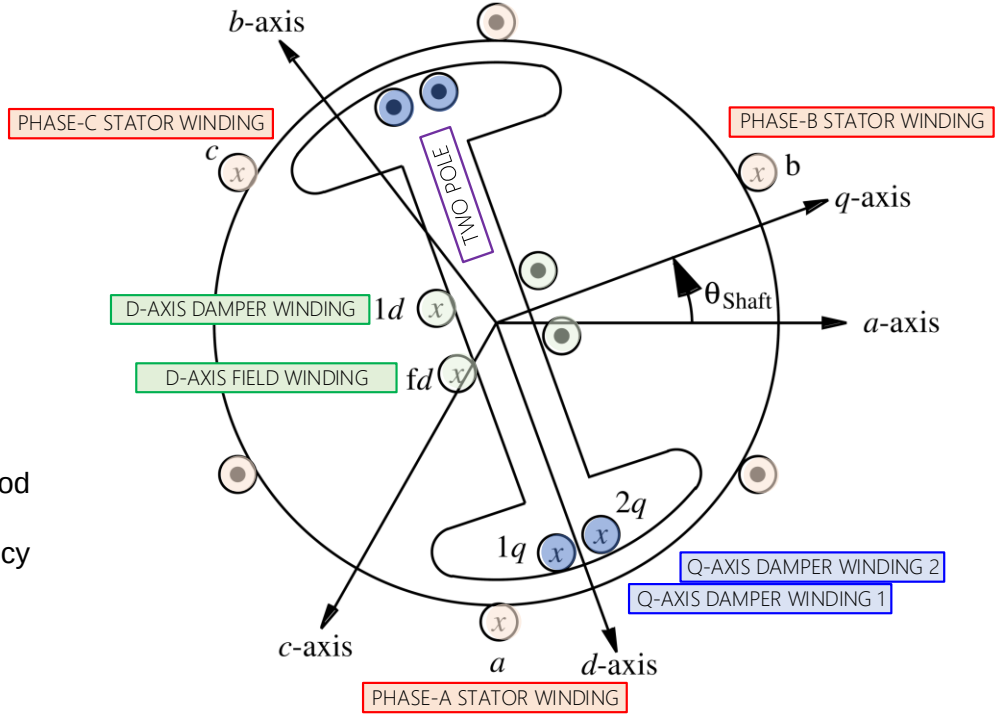


Parameters to set algorithm behaviour after an event

Exercise 4 – Dynamic Performance of Synchronous Generator

This exercise aims to simulate the dynamic behavior of a Steam Turbine Synchronous Generator using Park's equations* for the events specified below:

- **Analysis 1:**
  Step Change in Mechanical Torque
- **Analysis 2:**
  3-Phase Fault at the Terminals – Full Order
- **Analysis 3:**
  3-Phase Fault at the Terminals – Reduced Order



Notes:

1. To solve the differential equations, the **Euler Explicit** Method is used.
2. The machine is connected to an infinite bus whose frequency and voltage are constant during the simulation.
3. The equations used in this section are written in the Rotor Reference Frame known as Park's equations.
4. In Analysis 3, the transients of the stator windings are neglected.

*R. H. Park, "Two-Reaction Theory of Synchronous Machines-Generalized Method of Analysis, Part I, AIEE Transactions", Vol. 48, July 1929, pp. 716-727.

Exercise 4 – Dynamic Performance of Synchronous Generator

Machine Nameplate:

$$X_{mq} = X_q - X_{ls} = 1.303\ [\Omega]$$
$$X_{md} = X_d - X_{ls} = 1.303\ [\Omega]$$
$$X_{kq1} = X_{mq} + X_{lkq1} = 1.960\ [\Omega]$$
$$X_{kq2} = X_{mq} + X_{lkq2} = 1.379\ [\Omega]$$
$$X_{fd} = X_{md} + X_{lfd} = 1.417\ [\Omega]$$
$$X_{kd} = X_{md} + X_{lkd} = 1.368\ [\Omega]$$

Rating: 835 MVA
Line-to-line voltage: 26 kV
Power factor: 0.85
Poles: 2
Speed: 3600 r/min
Combined inertia of generator and turbine:

$J = 0.0658 \times 10^6\ \mathrm{J \cdot s^2}$, or $WR^2 = 1.56 \times 10^6\ \mathrm{lbm \cdot ft^2}$ $\quad H = 5.6\ \mathrm{s}$

Parameters in ohms and per unit:

$$r_s = 0.00243\ \Omega, 0.003\ \mathrm{pu}$$
$$X_{ls} = 0.1538\ \Omega, 0.19\ \mathrm{pu}$$

$X_q = 1.457\ \Omega, 1.8\ \mathrm{pu}$	$X_d = 1.457\ \Omega, 1.8\ \mathrm{pu}$
$r'_{kq1} = 0.00144\ \Omega, 0.00178\ \mathrm{pu}$	$r'_{fd} = 0.00075\ \Omega, 0.000929\ \mathrm{pu}$
$X'_{lkq1} = 0.6578\ \Omega,\ \mathrm{pu}\ 0.8125\ \mathrm{pu}$	$X'_{lfd} = 0.1145\ \Omega, 0.1414\ \mathrm{pu}$
$r'_{kq2} = 0.00681\ \Omega, 0.00841\ \mathrm{pu}$	$r'_{kd} = 0.01080\ \Omega, 0.01334\ \mathrm{pu}$
$X'_{lkq2} = 0.07602\ \Omega, 0.0939\ \mathrm{pu}$	$X'_{lkd} = 0.06577\ \Omega, 0.08125\ \mathrm{pu}$

Exercise 4 – Dynamic Performance of Synchronous Generator

The differential equations to be numerically solved by the **Euler Explicit Method** are given below*:

$$\frac{d\psi_{qs}}{dt} = \omega_b \left[v_{qs} - \frac{\omega_r}{\omega_b}\psi_{ds} + r_s\left(a_{11}\psi_{qs} + a_{12}\psi_{kq1} + a_{13}\psi_{kq2}\right) \right]$$

q-axis - Stator Winding Equation

$$v_{qs} = \cos(\delta)$$

$$\frac{d\psi_{ds}}{dt} = \omega_b \left[v_{ds} + \frac{\omega_r}{\omega_b}\psi_{qs} + r_s\left(b_{11}\psi_{ds} + b_{12}\psi_{fd} + b_{13}\psi_{kd}\right) \right]$$

d-axis - Stator Winding Equation

$$v_{ds} = \sin(\delta)$$

$$\frac{d\psi_{0s}}{dt} = \omega_b \left[v_{0s} + \frac{r_s}{X_{ls}}\psi_{0s} \right]$$

0-sequence - Stator Winding Equation

$$v_{0s} = 0$$

$$\frac{d\psi_{kq1}}{dt} = \omega_b \left[v_{kq1} + r_{kq1}\left(b_{21}\psi_{qs} + b_{22}\psi_{kq1} + b_{23}\psi_{kq2}\right) \right]$$

q-axis - Rotor 1st Damper Winding Equation

$$v_{kq1} = 0$$

$$\frac{d\psi_{kq2}}{dt} = \omega_b \left[v_{kq2} + r_{kq2}\left(b_{31}\psi_{qs} + b_{32}\psi_{kq1} + b_{33}\psi_{kq2}\right) \right]$$

q-axis - Rotor 2nd Damper Winding Equation

$$v_{kq2} = 0$$

P. C. Krause, O. Wasynczuk, S. D. Sudhoff, “Analysis of Electric Machinery and Drive systems”, IEEE Press, 2002.

Exercise 4 – Dynamic Performance of Synchronous Generator

The differential equations to be numerically solved by the **Euler Explicit Method** are given below:

$$\frac{d\psi_{fd}}{dt} = \omega_b r_{fd}\left[\frac{e_{xfd}}{X_{md}} - \left(b_{21}\psi_{ds} - b_{22}\psi_{fd} - b_{23}\psi_{kd}\right)\right]$$

→ ***d-axis - Rotor Field/Excitation Winding Equation***

$$\frac{d\psi_{kd}}{dt} = \omega_b\left[v_{kd} - r_{kd}\left(b_{31}\psi_{ds} - b_{32}\psi_{fd} - b_{33}\psi_{kd}\right)\right]$$

→ ***d-axis - Rotor Damper Winding Equation***

$v_{kd} = 0$

$$\frac{d\delta}{dt} = \omega_r - \omega_b$$

→ ***Torque Angle Equation***

$$\frac{d\omega_r}{dt} = \frac{\omega_b}{2H}\left[T_m - T_e\right]$$

→ ***Swing Equation***

Exercise 4 – Dynamic Performance of Synchronous Generator

Initialisation or Steady-State Quantities:

$$|I_{as}| = \frac{|S|}{3\,|V_{as}|} = \frac{835}{3 \times 26/\sqrt{3}} = 18.54 \text{ [kA]}$$

Stator Current

$$\tilde{I}_{as} = 18.54 \angle -31.8^\circ \text{ [kA]}$$

Stator Current – Phasor Representation

$$\tilde{E}_a = \tilde{V}_{as} + \left[r_s + j\frac{\omega_e}{\omega_b}X_q\right] = \frac{26}{\sqrt{3}} \angle 0^\circ + \left[0.00234 + j(1)(1.457)\right] \times 18.54 \angle -31.8^\circ = 37 \angle 38^\circ \text{ [kV]}$$

Internal Voltage

$$\delta = 38^\circ$$

Torque Angle

$$\begin{aligned} I_{ds} &= -\sqrt{2}\left|\tilde{I}_{as}\right| \sin[\theta_{ei}(0) - \theta_{ev}(0) - \delta] \\ &= -\sqrt{2} \times (18.54) \times \sin(-31.8^\circ - 0 - 38^\circ) \\ &= 24 \text{ [kA]} \end{aligned}$$

d-axis Stator Current

$$\begin{aligned} E_{xfd} &= \frac{\omega_e}{\omega_b}\left[\sqrt{2}\left|\tilde{E}_a\right| + \frac{\omega_e}{\omega_b}(X_d - X_q)I_{ds}\right] \\ &= \sqrt{2} \times (37) = 52 \text{ [kV]} \end{aligned}$$

Excitation Voltage

$$\begin{aligned} I_{qs} &= -\sqrt{2}\left|\tilde{I}_{as}\right| \cos[\theta_{ei}(0) - \theta_{ev}(0) - \delta] \\ &= -\sqrt{2} \times (18.54) \times \cos(-31.8^\circ - 0 - 38^\circ) \\ &= 9 \text{ [kA]} \end{aligned}$$

q-axis Stator Current

$$\begin{aligned} T_e &= \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)\left(\frac{1}{\omega_b}\right)(\psi_{ds}I_{qs} - \psi_{qs}I_{ds}) \\ &= 1.88 \times 10^6 \text{ [N.M]} \end{aligned}$$

Electrical Torque

Exercise 4 – Dynamic Performance of Synchronous Generator

Initialisation or Steady-State Quantities:

$$\psi_{qs} = -X_q I_{qs} + X_{mq} I_{kq1} + X_{mq} I_{kq2} = -13.15 \text{ [kV]}$$ → ***q-axis Stator Winding Flux***

$$\psi_{kq1} = -X_{mq} I_{qs} + X_{kq1} I_{kq1} + X_{mq} I_{kq2} = -11.76 \text{ [kV]}$$ → ***d-axis Stator Winding Flux***

$$\psi_{kq2} = -X_{mq} I_{qs} + X_{mq} I_{kq1} + X_{kq2} I_{kq2} = -11.76 \text{ [kV]}$$ → ***q-axis 1st Rotor Damper Winding Flux***

$$\psi_{ds} = -X_d I_{ds} + X_{md} I_{fd} + X_{md} I_{kd} = 16.73 \text{ [kV]}$$ → ***q-axis 2nd Rotor Damper Winding Flux***

$$\psi_{fd} = -X_{md} I_{ds} + X_{fd} I_{fd} + X_{md} I_{kd} = 25.14 \text{ [kV]}$$ → ***d-axis Rotor Field Winding Flux***

$$\psi_{kd} = -X_{md} I_{ds} + X_{md} I_{fd} + X_{kd} I_{kd} = 20.51 \text{ [kV]}$$ → ***d-axis Rotor Damper Winding Flux***

$$I_{kq1} = I_{kq2} = 0$$

$$I_{kd} = 0$$

Exercise 4 – Dynamic Performance of Synchronous Generator

Base Quantities:

$$S_b = S = 835 \text{ [MVA]}$$ → **Powers**

$$V_b = \sqrt{2}V_{as} = \sqrt{2} \times \frac{26}{\sqrt{3}} = 21 \text{ [kV]}$$ → **Voltages**

$$I_b = 2\frac{S_b}{3V_b} = 2\frac{835}{3 \times 21} = 26 \text{ [kA]}$$ → **Currents**

$$Z_b = \frac{V_b}{I_b} = \frac{21}{26} = 0.8 \text{ [}\Omega\text{]}$$ → **Impedances**

$$\omega_b = 2 \times \pi \times f = 2 \times \pi \times 60 = 377 \text{ [rad/s]}$$ → **Speeds**

$$T_b = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)\left(\frac{1}{\omega_b}\right)V_b I_b = 2.2 \times 10^6 \text{ [N.m]}$$ → **Torques**

$$\Psi_b = V_b = 21 \text{ [kV]}$$ → **Fluxes**

Per Unit System:

$$F_{pu} = \frac{F_{actual}}{F_b}$$

Exercise 4 – Dynamic Performance of Synchronous Generator

Fluxes and Currents:

$$i_{qs} = a_{11}\psi_{qs} + a_{12}\psi_{kq1} + a_{13}\psi_{kq2}$$

q-axis Stator Current

$$i_{ds} = b_{11}\psi_{ds} + b_{12}\psi_{fd} + b_{13}\psi_{kd}$$

d-axis Stator Current

$$i_{kq1} = a_{21}\psi_{qs} + a_{22}\psi_{kq1} + a_{23}\psi_{kq2}$$

q-axis 1st Damper Current

$$i_{fd} = b_{21}\psi_{ds} + b_{22}\psi_{fd} + b_{23}\psi_{kd}$$

d-axis Excitation Current

$$i_{kq2} = a_{31}\psi_{qs} + a_{32}\psi_{kq1} + a_{33}\psi_{kq2}$$

q-axis 2nd Damper Current

$$i_{kd} = b_{31}\psi_{ds} + b_{32}\psi_{fd} + b_{33}\psi_{kd}$$

d-axis Damper Current

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \frac{1}{D_q} \begin{bmatrix} X_{kq1}X_{kq2} - X_{mq}^2 & -X_{mq}X_{kq2} + X_{mq}^2 & -X_{mq}X_{kq1} + X_{mq}^2 \\ X_{mq}X_{kq2} - X_{mq}^2 & -X_q X_{kq2} + X_{mq}^2 & X_q X_{kmq} - X_{mq}^2 \\ X_{mq}X_{kq1} - X_{mq}^2 & X_q X_{mq} - X_{mq}^2 & -X_q X_{kq1} + X_{mq}^2 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \frac{1}{D_q} \begin{bmatrix} X_{kq1}X_{kq2} - X_{mq}^2 & -X_{mq}X_{kq2} + X_{mq}^2 & -X_{mq}X_{kq1} + X_{mq}^2 \\ X_{mq}X_{kq2} - X_{mq}^2 & -X_q X_{kq2} + X_{mq}^2 & X_q X_{kmq} - X_{mq}^2 \\ X_{mq}X_{kq1} - X_{mq}^2 & X_q X_{mq} - X_{mq}^2 & -X_q X_{kq1} + X_{mq}^2 \end{bmatrix}$$

$$D_q - X_{mq}^2(X_q - 2X_{mq} + X_{kq1} + X_{kq2}) - X_q X_{kq1} X_{kq2}$$
$$D_d = X_{md}^2(X_d - 2X_{md} + X_{fd} + X_{kd}) - X_d X_{fd} X_{kd}$$

Exercise 4 – Dynamic Performance of Synchronous Generator

Python Coding - hints:

1. Create dictionaries for the actual, base and per unit parameters. Please see the examples given below:

```
actual_dict = {}
actual_dict = {"S": 835}

base_dict = {}
base_dict = {"Sb": 835}

pu_dict = {}
pu_dict = {"S": 1}
```

2. Initialisations – use per unit values:

```
Te = pu_dict["Te"]
```

3. Before solving the differential equations, create a dictionary for results – parameters of interest:

```
results_dict = {}
results_dict = {"wr": []}
```

4. Use a "while" loop for simulation for t_sim [s]:

```
t = 0
h = STEP SIZE
while t < t_sim:
    dfdt = refer to differential equations
    f = f + h x df # Euler Explicit method
    results["f"].append(f) # saving the results
    t = t + h
```

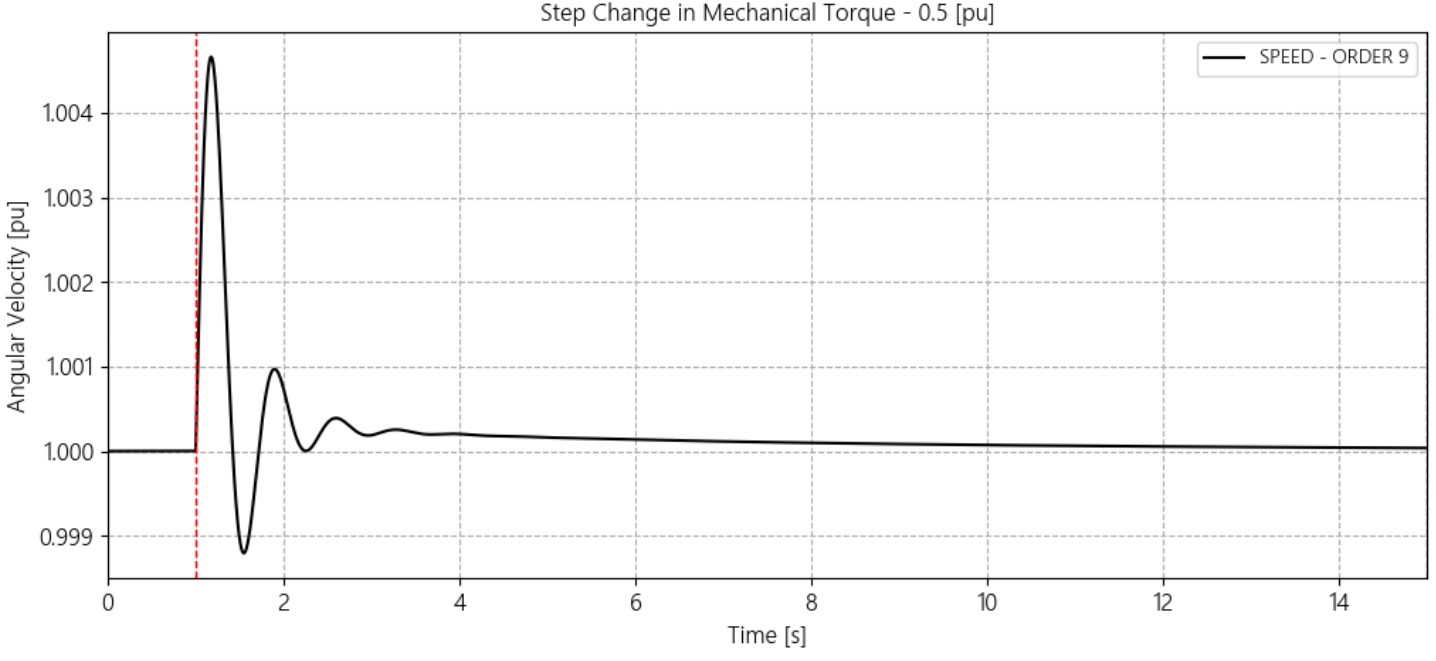
Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 1 = Step Change in Mechanical Torque

- *Simulation Time* = 15 [s]
- *Step Size* = 0.00001 [s]
- *Event Time* = 1 [s]
- *Input Step Change* = $T_m = 0.5\ [\text{pu}]$

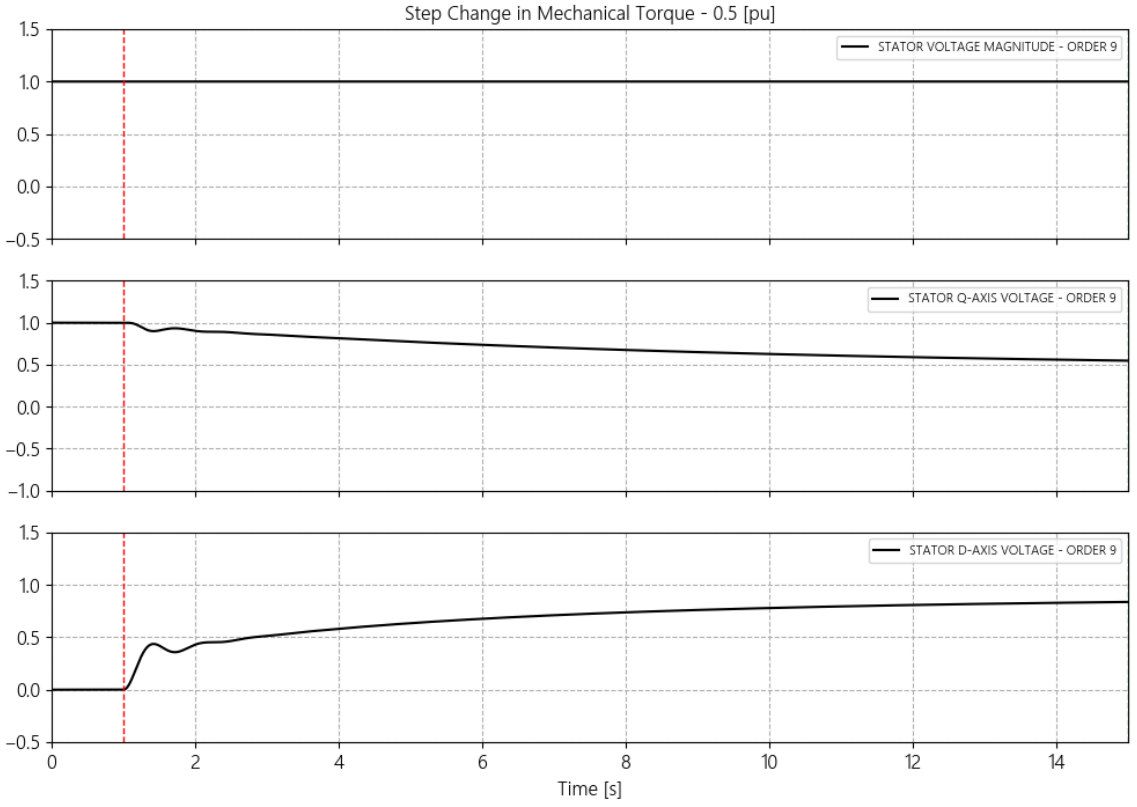
Steady-State

$$T_m = T_e = I_{as} = 0$$



Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 1 = Step Change in Mechanical Torque



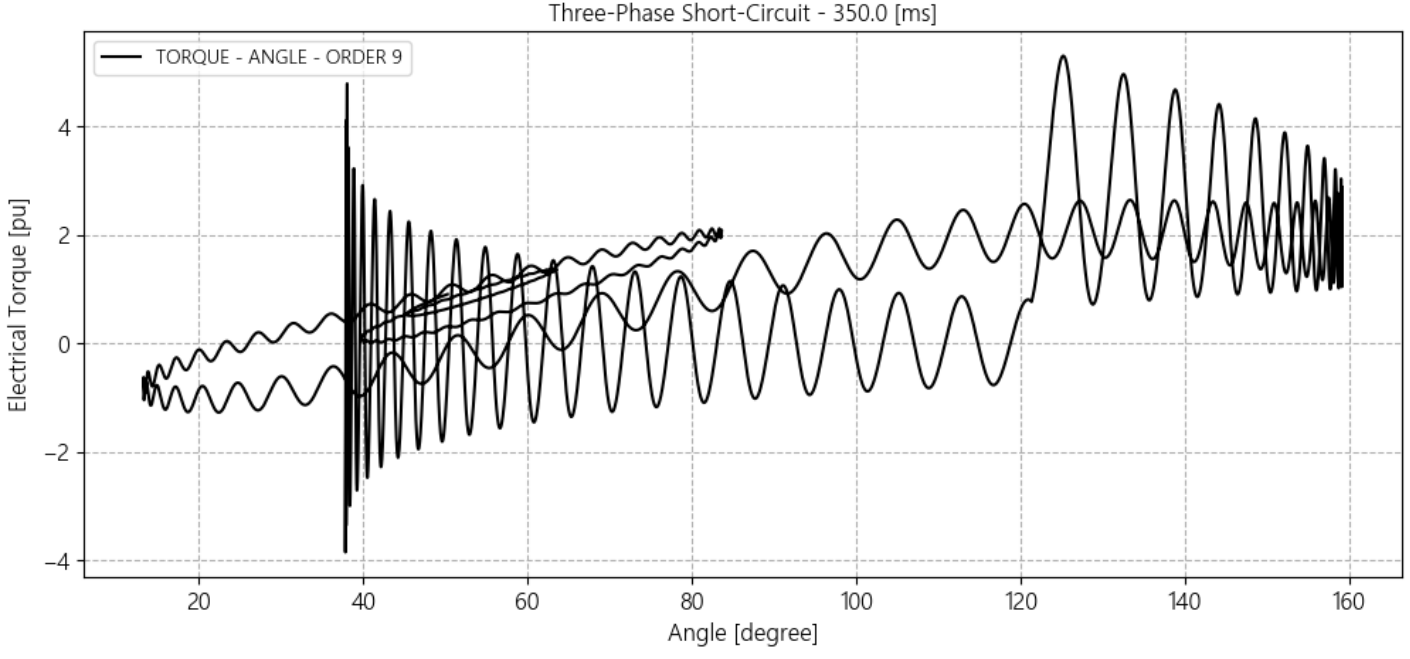
Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 2 = 3-Phase Short-Circuit Fault at the Terminals – Full Order

- *Simulation Time = 3 [s]*
- *Step Size = 0.00001 [s]*
- *Event Time = 0.5 [s]*
- *Faulted Terminal Voltage = 0 [pu]*
- *Fault Duration = 350 [ms]*

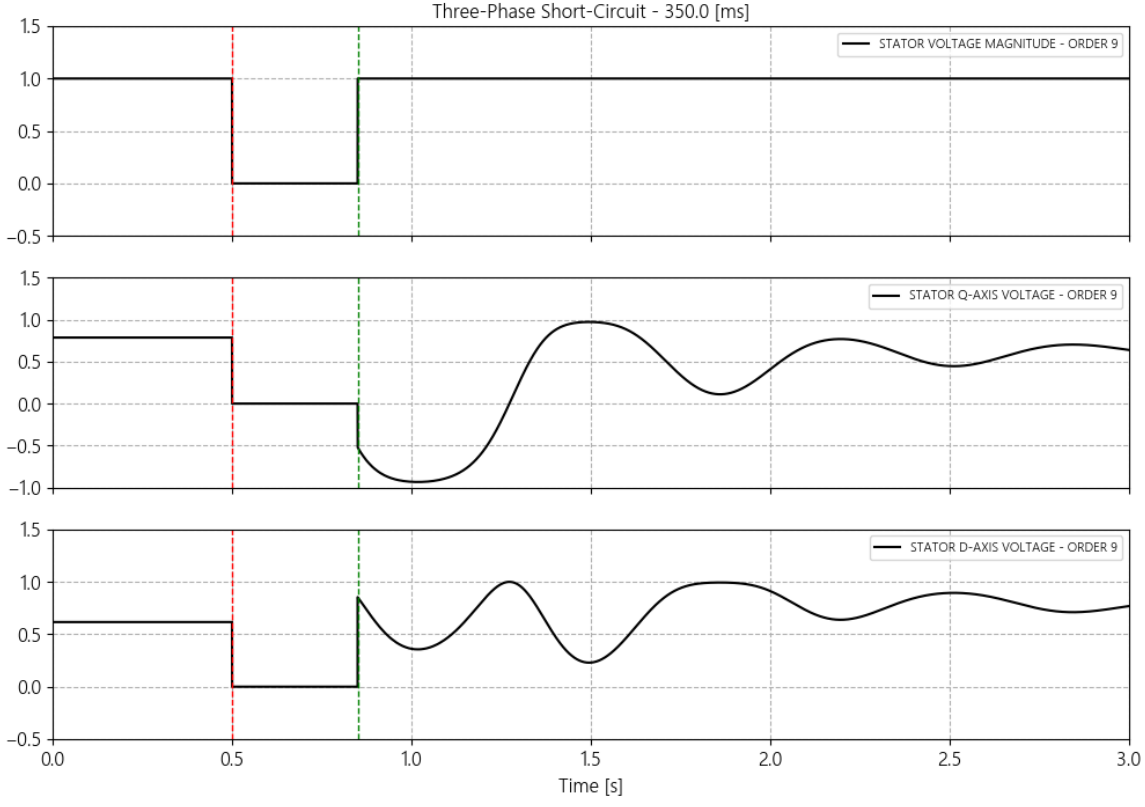
During Fault

$$v_{qs} = v_{ds} = 0$$



Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 2 = 3-Phase Short-Circuit Fault at the Terminals – Full Order

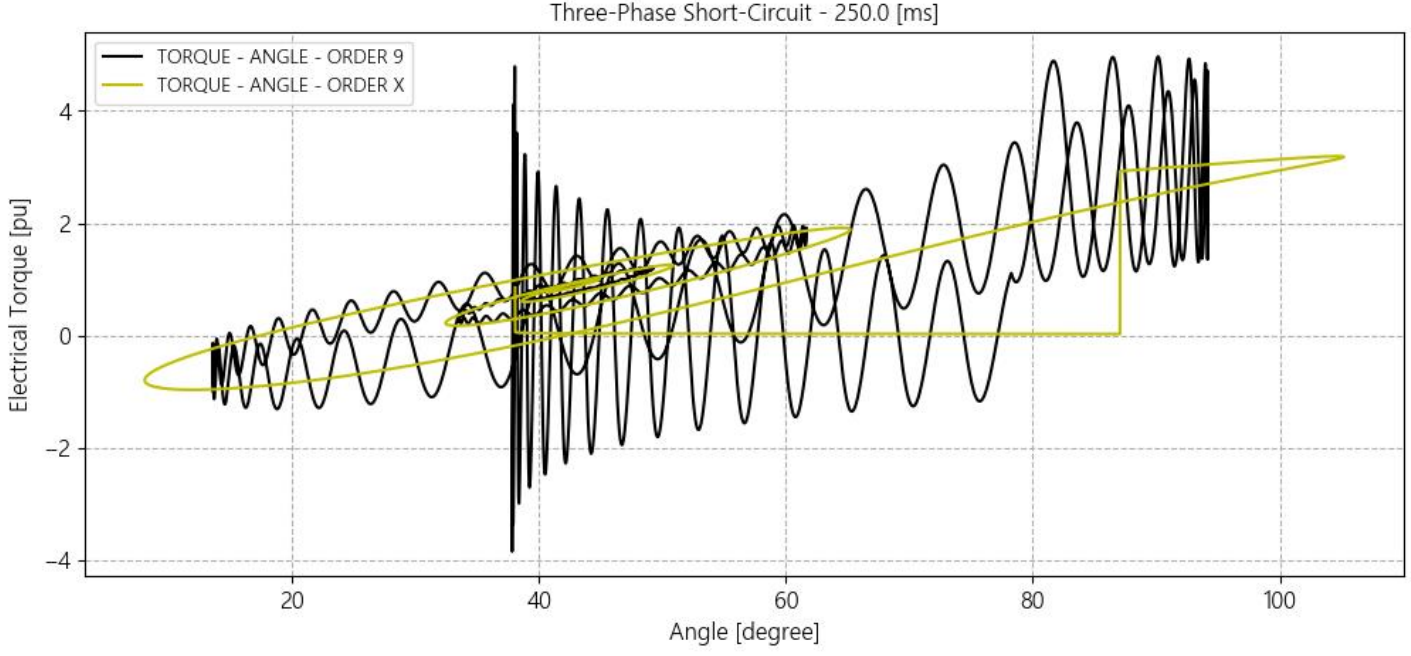


Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 3 = 3-Phase Short-Circuit Fault at the Terminals – Reduced Order

- *Simulation Time* = 3 [s]
- *Step Size* = 0.00001 [s]
- *Event Time* = 0.5 [s]
- *Faulted Terminal Voltage* = 0 [pu]
- *Fault Duration* = 250 [ms]

During Fault	Stator Transients
$v_{qs} = v_{ds} = 0$	$\frac{d\psi_{qs}}{dt} = \frac{d\psi_{ds}}{dt} = 0$



Exercise 4 – Dynamic Performance of Synchronous Generator

Results – Analysis 3 = 3-Phase Short-Circuit Fault at the Terminals – Reduced Order