

SciML for PS

Scientific Machine Learning (SciML) for Power Systems (PS)

Focus — dynamic modeling, stability assessment, and system identification

Last updated: October 14, 2026 at 11:50 AM (UTC+00:00) (ab5797a)

Content: CC BY-NC-SA 4.0 · Code: MIT

Note: This PDF export is experimental. Interactive figures appear as static snapshots or may be omitted. Prefer the online notes when something looks wrong.

Under development until v1.0 — public preview v0.1 (v0.x series): chapters may change between versions. Contributions from everyone are welcome; see sciml4ps.albertpoulose.com/contribute.

Albert Poulose

© 2026 Albert Poulose. Book content licensed under Creative Commons Attribution–NonCommercial–ShareAlike 4.0 International (CC BY-NC-SA 4.0). Teaching code licensed under the MIT License. Online: sciml4ps.albertpoulose.com

Contents

Front matter	1
Start here	2
How to use this book	4
Advice to readers	7
Preview this book locally	9
Running notebooks	12
Your first lab	14
Book map	17
Full outline	19
Learning paths	25
Related open resources	28
Join and contribute	30
Status and versions	32
Feedback	34
License	35
Preparation	36
Preparation overview	37
Programming from zero	39
Power systems for ML people	41
Machine learning for power-systems people	43
Prep 01 — NumPy basics	45
Prep 02 — ODE integration	46
Prep 03 — Tiny neural regression	47
Prep 04 — Gradients	48
Foundations	49
Foundations overview	50
SciML method families (landscape)	52
Problem framing	55
SMIB dynamics	58
Reference simulation	60
Data and surrogates	62
Automatic differentiation	64
Forward PINNs	66

Classical estimation baselines	69
Identifiability	71
Inverse PINNs	73
Neural ODEs and hybrid models	75
Dynamics	77
Dynamics pillar overview	78
Stability	80
Stability pillar overview	81
Stability labels and CCT	83
Surrogate models for stability tasks	86
ANDES as network reference	89
Direct methods (equal-area and energy ideas)	92
Identification	94
Identification pillar overview	95
Measurements and excitation	97
Physical vs equivalent parameters	99
Differentiable simulation	101
Advanced topics	104
Advanced topics overview	105
From assessment to decision	107
Laboratories	109
Laboratories	110
Lab 1 — Reference simulation and scientific dataset	111
Lab 2 — Supervised surrogate versus physics residual (PINN)	113
Lab 3 — Inertia and damping estimation	114
Lab 4 — Generalization of a learned dynamic model	116
Lab 5 — Stability and CCT	117
Reference	119
Reading list	120
Glossary	122
Notation	127

Front matter

Start here

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

These notes are a meeting place. People who already work in machine learning and people who already work in power systems can learn enough of the other field to study the same plant, write the same labs, and review each other’s chapters. Students, teachers, engineers, utility workers, and researchers share one book. You may open any chapter. Each chapter states what it assumes in a **Before you start** box, and the [glossary](#) translates a word into both fields.

Power-system stability language used here follows published definitions [Kundur et al., 2004, Hatzia-rygiou et al., 2021]. The learning methods sit in the physics-informed machine learning line [Kar-niadakis et al., 2021].

Choose your entrance

Answer two questions. The closest row is enough. You can change your mind and jump later. Every route is written out on [Learning paths](#).

Your background.	Closest description	Start with
	New to programming and new to power sys-tems	Programming from zero , then your first
	Comfortable with machine learning, new to power systems	Power systems for ML people
	Comfortable with power systems, new to ma-chine learning	Machine learning for power-systems people
	Comfortable with both	Full outline . Use each page’s Before start box

Your role.	Role	How to use the notes
	Student	Follow the background route on Learning paths , and keep the labs beside the prose
	Teacher	Use the course overlays on Learning paths. Share a week plan through Join and contribute
	Practitioner or engineer	Go to the application you already work on: Dy-namics , Stability , or Identification . Read that page’s Before you start box and fill only the gaps
	Researcher	Start from SciML method families , Identifiabil-ity , and Related open resources

Background and role together

Background	Student	Teacher	Practitioner	Researcher
New to both	Beginner path	Teacher overlays after you have run Lab 1	First lab , then the pillar you work on	Both primers, then the researcher path
From machine learning	From ML	Same overlays; assign the power-systems primer in week 1	Power-systems primer , then your pillar	Primer, then landscape
From power systems	From power systems	Same overlays; assign the ML primer in week 1	ML primer , then your pillar	Primer, then identifiability
Both	Outline	Teacher overlays	Pillar page, then its lab	Researcher path

You can jump

The sidebar order (Preparation, Foundations, then the three applications) is one useful route. It is optional.

- [Full outline](#) lists every page, what it needs, and which lab sits beside it.
- [Learning paths](#) gives shorter routes by background and by role.
- [Glossary](#) and [notation](#) work from any page.
- [Advice to readers](#) is short and worth reading once.

Run a lab today

[Your first lab](#) runs Lab 1 in a browser, with numbered steps and no install. Local install remains the canonical path for assessed work: [Running notebooks](#).

How to use this book

What this is

A Jupyter Book 2 / MyST computational textbook for **Scientific Machine Learning (SciML) for Power Systems (PS)** — short form **SciML for PS**. Focus: dynamic modeling, stability assessment, and system identification. Maintained by **Albert Poulou** for teaching and research training. The notes are written so machine-learning readers and power-systems readers can enter at their own level, including readers who are new to both. Pages follow a common pattern (goals → reading → media → example → lab → misconceptions).

You can enter anywhere

[Start here](#) asks two questions (your background, your role) and links to a route. You may also open the [full outline](#) and jump to a chapter.

Three things make that safe:

- **Before you start** on each chapter names what the page assumes, with a hint for ML readers, a hint for power-systems readers, and a hint if both are new.
- The [glossary](#) gives the plain meaning, the power-systems reading, and the ML reading of the same word. [Notation](#) is the symbol table.
- [Learning paths](#) writes the routes out in order, including what you can skip.

The sidebar order (Preparation, then Foundations, then Dynamics, Stability, or Identification) is one useful route. It is the `path_core` overlay for a first course. It is optional if your background already covers part of it.

How SciML methods appear

These notes are organized by **power-system question**, not by a catalogue of architectures.

1. Name the engineering question: [Dynamics](#), [Stability](#), or [Identification](#).
2. Use the same plant: [Lab 1](#) (and Foundations SMIB chapters).
3. Keep a classical or supervised baseline, then meet an early physics-informed path: residual / inverse PINN in Labs 2–3.
4. Meet other families when the same plant needs them — Neural ODEs and operators in [Lab 4](#), differentiable simulation under Identification, frontiers under [Advanced](#). They do not get a second table of contents.
5. Keep five assessed labs. New methods deepen Labs 1–5 or sit in Advanced; they do not become Lab 6.

Family map: [SciML method families](#). Practice table: [Laboratories](#).

Browser

Prefer **Google Chrome**, **Microsoft Edge**, or another Chromium-based browser for the live notes. Interactive figures and long pages scroll more reliably there than in Safari.

If you are new to programming

Use [Programming from zero](#) and [Your first lab](#). That walkthrough runs Lab 1 in the browser. Install nothing until you want a copy on your own computer.

Software

For a local copy of the labs (the canonical path once you are past the browser walkthrough):

- Core Python stack: `envs/course_core.yml` (Labs 1–5 default). Setup steps: [Running notebooks](#).
- Network / ANDES path (after SMIB CCT): `envs/course_andes.yml` — see [ANDES as network reference](#)

Preview the book on your machine

Step-by-step commands and helper scripts, for people who want the HTML site locally:

→ [Preview this book locally](#)

Short path (Windows):

```
.\scripts\setup_env.ps1
conda activate sciml_ps
.\scripts\preview.ps1
```

Then open <http://localhost:3000/> and leave that terminal running.

Running a single lab notebook is a smaller job than previewing the whole site. Beginners should use [Your first lab](#) first.

What not to expect yet

This book is a **public preview v0.1** (v0.x series) and remains **under development until v1.0**. Stub pages and thin [Preparation](#) notebooks in the sidebar are intentional. Prefer the live HTML site over the experimental PDF. Version labels: [Status and versions](#). How to run lab notebooks (local = canonical; Binder = primary online; Colab = secondary): [Running notebooks](#).

PDF download

The navbar **PDF** control opens a snapshot of the notes in a **new browser tab**, built each time the site is published. It may lag slightly behind the HTML pages. Treat the export as experimental. If you preview the book locally, see [Preview this book locally](#) for rebuilding the PDF.

Prefer the live site for labs

For executable laboratories and notebooks, use the HTML book and [Running notebooks](#), not the PDF snapshot.

Feedback and collaboration

See [Feedback](#) (GitHub issues for now; Google Form planned). To help improve the notes, translate a term into the other field, or express interest as a reviewer/section editor: [Join and contribute](#). Neighbors and positioning: [Related open resources](#).

License

Book content is **CC BY-NC-SA 4.0** (free to share and adapt for non-commercial teaching with credit). Teaching code is **MIT**. Full terms: [License](#).

For instructors and maintainers

Course overlays live on Learning paths. Previewing, editing chapters, wiring the TOC, and growing parts over many years is documented in the git repo under the `docs/guides/` folder (maintainer handbook; not a book page). Hosting/PDF notes: `docs/guides/site_hosting.md`. Legal checklist: `docs/guides/legal_and_licensing_considerations.md`. Governance: root `GOVERNANCE.md`.

Advice to readers

How to read a page

Most chapter pages follow the same order: learning goals → reading → worked example → lab → misconceptions. Skim the goals and the **Before you start** box first. If a word is unfamiliar, open the [glossary](#). Then read the example and open the lab notebook. Misconceptions are worth reading even when the rest felt easy.

You may skip a chapter when the Before you start box is already satisfied. [Start here](#) and [Learning paths](#) list routes by background and by role.

If you are new to programming

Run the notebook before you try to understand every line. [Your first lab](#) is numbered and says what you should see. Restarting the kernel and running all cells from the top fixes many early errors.

If you come from machine learning

Respect units, per-unit bases, and operating limits. A trajectory error is one score. Test on a disturbance the model did not train on, and report stability labels and CCT beside that error when the chapter is about security. Inertia and damping are named parameters with bounds. They are separate from network weights. Do not treat every neural method as a PINN — read the [SciML method families](#) map first.

If you come from power systems

A low training loss is a fit, and a held-out disturbance is the check. Ask whether a fitted (H, D) is the physical constant in the model or an equivalent value that absorbed model error. Classical least squares stays beside the neural estimator so you can see what the extra machinery changed. In these notes, a PINN means the swing-equation residual enters the loss (Labs 2–3). Lab 4 and Lab 5 are other SciML jobs on the same plant — learned dynamics and CCT margins — not more PINN chapters by another name.

Run the notebook while you read

These notes are meant to be used with code open. Open prep notebooks from the **Preparation** sidebar (they may still be stubs) and laboratory notebooks from **Laboratories**. Execute cells as you go. Reading alone is not enough for Labs 1–5. Browser steps: [Your first lab](#). Local, Binder, and Colab: [Running notebooks](#).

State units, bounds, and test disturbances

Every fitted parameter and every stability label needs a contract: units (or per-unit base), bounds, and which disturbance or clearing time was used for training versus testing. A low trajectory error without that contract is not a result this course accepts.

Prefer the live site for labs

Use the HTML book for notebooks. The PDF is a snapshot; interactive figures and runnable cells may flatten. Prefer a Chromium-based browser (Chrome, Edge, or similar) for smoother scrolling and interactive figures.

Ask for help early

Typos, unclear steps, and broken links: [Feedback](#). Do not paste confidential research data or grades into public issues. A useful early contribution is a glossary row that translates a term into the other field: [Join and contribute](#).

Honesty about metrics

Report what you measured (RMSE, CCT error, false-safe count, held-out disturbance) against a named reference simulator. Do not claim physical parameter recovery or protection-grade CCT from a single training loss.

Preview this book locally

Use these commands from the **repository root** (the folder that contains `myst.yml`). After the server starts, open **one book URL**: <http://localhost:3000/>.

Jupyter Book 2 also starts a file/content server (usually port **3100**). That is part of the same preview, not a second book. Prefer **Chrome or Edge** for the navbar **PDF** link (editor-embedded browsers often show a blank PDF). Or open `exports/sciml_ps_notes.pdf` on disk after `scripts/build_pdf.ps1`.

1. One-time environment setup

The course environment installs **Python**, **Jupyter Book 2**, and **Node.js** (required by JB2's MyST site server).

```
conda env create -f environment.yml
conda activate sciml_ps
```

On Windows PowerShell you can instead run:

```
.\scripts\setup_env.ps1
conda activate sciml_ps
```

If the environment already exists and dependencies changed (for example after Node.js was added):

```
conda env update -f environment.yml - --prune
conda activate sciml_ps
```

Confirm both tools:

```
jupyter book - --help
node - --version
```

2. Live preview (recommended while reading or editing)

Option A — helper script (Windows).

```
conda activate sciml_ps
.\scripts\preview.ps1
```

Optional port or notebook execution at build time:

```
.\scripts\preview.ps1 -Port 3001
.\scripts\preview.ps1 -Execute
```

Option B — helper script (macOS / Linux).

```
conda activate sciml_ps
chmod +x scripts/preview.sh scripts/build.sh    # once
./scripts/preview.sh

./scripts/preview.sh - --port 3001
./scripts/preview.sh - --execute
```

Option C — raw Jupyter Book commands. Use `HOST=localhost` with `-keep-host` so the theme UI and the content/PDF server agree on hostnames (PDF links redirect to `http://localhost:3100/...`). Helper scripts already do this.

```
conda activate sciml_ps
$env:PYTHONUTF8 = "1"
$env:HOST = "localhost"
jupyter book start - -port 3000 - -keep -host
```

```
conda activate sciml_ps
export HOST=localhost
jupyter book start - -port 3000 - -keep -host
```

```
jupyter book start - -port 3001 - -keep -host    # only if 3000 is busy
jupyter book start - -execute - -keep -host
```

Edit any page under `book/`, save, and refresh the browser.

Open in the browser: <http://localhost:3000/>

The preview process must stay running in that terminal (do not close it). Do not start a second preview while one is running.

3. Static HTML build (no live server)

Windows.

```
conda activate sciml_ps
.\scripts\build.ps1
```

macOS / Linux.

```
conda activate sciml_ps
./scripts/build.sh
```

Raw command.

```
jupyter book build
```

Open the `index.html` path printed by the CLI (often under `_build/`).

4. What you should see

- Sidebar parts: Front matter, Preparation, Foundations, Dynamics, Stability, Identification, Advanced, Laboratories, Reference
- This page under Front matter
- Stub chapters until prose is filled

5. Troubleshooting

Symptom	Fix
Page will not load / connection refused	Start the server first (<code>.\scripts\preview.ps1</code>) and leave that terminal open
Confused by port 3100	Normal JB2 content/static server for the same preview; bookmark only <code>http://localhost:3000/</code>
Navbar PDF is blank in an editor-embedded browser	Open in Chrome/Edge, or open <code>exports/sciml_ps_notes.pdf</code> on disk
Navbar PDF is old / 404	Run <code>.\scripts\build_pdf.ps1</code> , then <code>python scripts/build_pdf.py -sync-only</code> if preview is already up; hard-refresh
<code>jupyter</code> / <code>jupyter</code> book not found	<code>conda activate sciml_ps</code> (or re-run <code>.\scripts\setup_env.ps1</code>)
<code>node</code> not found / JB asks to install Node	<code>conda activate sciml_ps</code> then <code>conda env update -f environment.yml -prune</code>
Unicode / emoji crash in Windows console	Scripts set <code>PYTHONUTF8=1</code> ; or run <code>\$env:PYTHONUTF8=1</code> before <code>jupyter book start</code>
Port already in use	Stop the other preview, or <code>.\scripts\preview.ps1 -Port 3001</code> (then use that port in the URL)
New page missing from sidebar	Add the file under <code>project.toc</code> in <code>myst.yml</code>
Theme download slow first time	Wait; JB2 caches templates under <code>_build/templates/</code>
CLI flag unknown	Run <code>jupyter book -help</code> — JB2 names can change slightly

Maintainer detail (PDF engine, cover provenance, hosting): `docs/guides/site_hosting.md`. Editing / long-term growth: `docs/guides/` (not book pages).

Running notebooks

Under development until v1.0 — public preview v0.1

Launch paths and notebooks may change while the notes grow. Contributions are welcome — help harden Binder, Colab setup cells, or new lab notebooks: [Join and contribute](#). Version labels: [Status and versions](#).

New to notebooks? Follow the numbered steps in [Your first lab](#) (Colab, then Binder, then a local install). This page is the full comparison of the three launch paths.

These living notes keep every assessed notebook in the [GitHub repository](#). You can run them in three ways. Use them in this order of trust:

1. **Local install** — canonical (always works once the repo is cloned)
2. **Binder** — primary online launch (temporary JupyterLab from this repo)
3. **Google Colab** — secondary convenience (personal copy in Google’s cloud)

Binder and Colab need a public GitHub repository. Until `albert8943/SciML_PS` is public, use **local install** only. Session edits in Binder or Colab do **not** write back to GitHub; to improve the book, open a pull request ([Contribute](#)).

1. Local install (canonical)

Best for assessed work and for decades of reproducible teaching.

```
git clone https://github.com/albert8943/SciML_PS.git
cd SciML_PS
conda env create -f environment.yml
conda activate sciml_ps
pip install -e .
```

Then open the lab notebook (for example `book/900_labs/lab_01_reference_sim/lab01_reference_sim.ipynb`) with Jupyter.

To preview the **book site** locally (HTML), see [Preview this book locally](#). That author path also uses `environment.yml` and needs Node.js for Jupyter Book 2.

Optional network / ANDES path (Lab 5 extension only): use `envs/course_andes.yml` on your machine — not Binder.

2. Binder (primary online)

Opens a **temporary** JupyterLab built from this repository. Your edits live only for that session.

- First launch after a change can take **several minutes** while Binder builds the image.
- Uses the lean student env under `binder/` (not the author Jupyter Book stack).
- Does **not** include ANDES.

Example — Lab 1:

Lab pages that already ship a notebook also carry a Binder badge under **Materials**.

3. Google Colab (secondary)

Opens a **personal copy** of the notebook on Google’s servers. Convenient for a quick try; still not a substitute for local install on assessed work.

Example — Lab 1:

Run the notebook’s first setup cell so `sciml_ps` installs from GitHub when needed.

Which labs?

Lab	Local	Binder / Colab
Lab 1	Yes	Yes (badges on the lab page)
Lab 3	Yes	Yes
Labs 2 and 4	When notebooks arrive	Same — badges arrive with the notebook
Lab 5 SMIB path	Yes when the SMIB notebook ships	Planned with that notebook
Lab 5 ANDES network extension	Local only (<code>course_andes</code>)	Not on Binder / Colab

Invite collaboration

If Binder builds fail, Colab install cells break, or you want to add launch badges for a new notebook, please [Join and contribute](#). Improving these launch paths is a welcome contribution to this public preview.

Your first lab

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Under development (v0.x public preview)

Launch buttons and the notebook may change while the notes grow. Help improve these steps: [Join and contribute](#).

This page runs [Lab 1](#) with no programming background. The notebook simulates a classical single-machine swing model [Kundur and Malik, 2022]. Plan on about 15 minutes. You will simulate one generator after a short fault and see a plot of rotor angle against time.

Three ways to run it, in the order a new reader should try:

1. **Google Colab** below (browser, Google account, no install)
2. **Binder** if you prefer not to use Google (browser, no install, slower the first time)
3. **Local install** when you want the canonical copy on your own computer

Assessed coursework should use local install once you are comfortable. The full comparison is [Running notebooks](#). Binder and Colab need the GitHub repository to be public. Until then, use the local steps.

Before you click

A notebook is a page of boxes called cells. Text cells explain. Code cells compute. You run a code cell with its play button. The computer remembers the cells you have already run until you restart the session.

Route A. Google Colab

You need a free Google account.

1. Open the Lab 1 notebook in Colab: [Open Lab 1 in Colab](#). What you should see: a Colab page titled like the lab, with a column of gray and white cells. The first cell is an explanation. The next cell is code.
2. If Colab asks you to sign in, sign in. If it warns you that the notebook comes from GitHub, continue. That warning is expected.
3. Click the play button on the **first code cell** (it starts with `import sys`). What you should see: Colab may spend a minute installing the teaching package. The cell finishes with the line `imports ok`. A green check, or a number in brackets beside the cell, means that cell finished.
4. Click the play button on the **next code cell** (it starts with `# Same physics, two numerical settings`). What you should see: four printed lines. Two say whether a fine and a coarse simulation failed. Two say whether each result was stable.

5. Run the **following code cell** (it starts with `fig, ax = plt.subplots`). What you should see: a plot. The horizontal axis is time in seconds. The vertical axis is rotor angle in radians. Vertical lines mark the fault and the clearing time.
6. Run the remaining code cells in order. What you should see: a short table of scenarios, the words `train` and `test`, and a line that starts with `wrote` and ends with `lab01_manifest.json`.

To run every cell without clicking each one: menu **Runtime**, then **Run all**. Wait until the last cell finishes.

Your edits stay in your Colab copy. They do not change the book. To suggest a fix, use [Feedback](#).

Route B. Binder

Binder builds a temporary JupyterLab from this repository. The first launch can take several minutes. Your edits disappear when you close the tab.

1. Open [Launch Lab 1 on Binder](#). What you should see: a waiting page, then Jupyter with the Lab 1 notebook open.
2. Menu **Run**, then **Run All Cells**. What you should see: the same `imports ok` line, the same four printed lines, and the same angle plot as in Route A. Binder uses the copy of the code already in the repository, so the setup cell should print `Using local src:`.

Route C. On your own computer

Do this when Colab and Binder are unavailable, or when a teacher asks for a local run. Install [Miniconda](#) first and allow the installer to add `conda` to your PATH. Then get the repository either by downloading the ZIP from the GitHub **Code** button or by cloning it.

Windows. In PowerShell, from the folder that contains `environment.yml`:

```
.\scripts\setup_env.ps1
conda activate sciml_ps
pip install -e .
jupyter lab
```

What you should see: a browser tab opens with a file list. Open `book`, then `900_labs`, then `lab_01_reference_sim`, then `lab01_reference_sim.ipynb`. Menu **Run**, then **Run All Cells**. The setup cell should print `sciml_ps` already importable or `Using local src:`.

Mac or Linux. The Windows script is a small wrapper around the same commands. In a terminal, from the folder that contains `environment.yml`:

```
conda env create -f environment.yml
conda activate sciml_ps
pip install -e .
jupyter lab
```

If the environment already exists, use `conda env update -f environment.yml -prune` instead of `conda env create`. Then follow the same Jupyter clicks as on Windows.

Previewing the whole website (the HTML book) is a separate, optional step: [Preview this book locally](#).

Troubleshooting

What happened	What to do
The setup cell says it is installing from GitHub, then fails	Binder and Colab need a public repository. Until this repository is public, use Route C
A later cell says <code>sciml_ps</code> is not defined, or a name is missing	Restart the kernel (Colab: Runtime → Restart session . Jupyter: Kernel → Restart). Run all cells again from the top
Binder has been on a waiting page for a long time	The first build often takes several minutes. Leave the tab open. If it errors, use Route A or Route C
Colab shows a red error in a cell	Read the last line of the error. If it is still unclear, copy that last line into a feedback issue . Leave out private data
You see no plot	Run the cell that starts with <code>fig, ax = plt.subplots</code> after the simulation cell has finished
You changed a cell and the result looks stale	Run all cells from the top again. The kernel only knows what it has run in this session

After the plot

Read the Lab 1 goals on the [lab page](#), then [SMIB dynamics](#) for the equation behind the plot. If the power-system words are new, read [Power systems for ML people](#). If the learning words are new, read [Machine learning for power-systems people](#).

Book map

How the notes are organized. Use the left sidebar for day-to-day navigation; this page is the big picture. For every chapter and lab on one page, see the [Full outline](#).

Section	What you find here
Front matter	Start here , how to use, local preview, running notebooks , your first lab , this map, full outline , learning paths , related open resources , contribute , status , feedback, license
Preparation	Overview, three bridge primers (programming, power systems for ML, ML for power systems), plus four diagnostic notebooks (stubs today)
Foundations	Shared SMIB model and core SciML ideas (losses, forward/inverse PINN, baselines)
Dynamics	Deeper dynamic modeling and simulation (SciML for trajectories / vector fields)
Stability	Stability assessment (labels, CCT, surrogate stability; ANDES network bridge after SMIB)
Identification	Parameter estimation and system identification
Advanced topics	Synthesis (assessment → decision) plus optional extensions (operators, UQ, converters)
Laboratories	Assessed Labs 1–5
Reference	Reading list, glossary , and notation

Read **Preparation** → **Foundations**, then deepen along **Dynamics**, **Stability**, or **Identification**, when you want the long route. Other entrances (by background and by role) are on [Start here](#) and [Learning paths](#). Any chapter can be a starting page when its **Before you start** box is satisfied. Labs sit beside the notes and cross-link from chapters.

Methods appear when an application needs them. PINN is an early deep path (Labs 2–3); other SciML families sit under Lab 4, Identification options, or Advanced. Labs 1–5 are the fixed assessed set — deepen notebooks rather than adding Lab 6. Reader summary: How SciML methods appear.

This top-level layout is intentional and meant to stay stable as the notes grow: shared foundations, three applications, labs beside the prose, and a separate reference section. Semester week-by-week paths belong under [Learning paths](#), not as a rewrite of these sections.

What each application produces

Application	Engineering question	What you produce
Dynamic modeling and simulation	How will the system evolve under specified inputs, parameters, and disturbances?	State trajectories, learned vector fields, hybrid dynamic models, parameterized response maps
Stability assessment	Will the system remain stable, and how close is it to a stability boundary?	Stable/unstable decisions, critical clearing time (CCT), empirical stability boundaries
Parameter estimation and system identification	Which parameters and dynamic relationships explain the measurements?	Inertia/damping estimates, controller parameters, learned missing terms, uncertainty estimates

In the sidebar these appear as **Dynamics**, **Stability**, and **Identification**.

Full outline

Every page of the notes on one page, in reading order. The [Book map](#) explains *why* the book is organized this way; this page lists *what* is in each section. **Needs** says what to read if you jump here directly. Entrances by background and role: [Start here](#).

Maturity follows the labels on each page: **Stub** (outline and goals only), **Draft** (readable prose, still growing), **In review**, **Stable**. Details: [Status and versions](#). Want to write, review, or extend a chapter? See [Join and contribute](#).

Front matter

Page	What it covers
Start here	Choose an entrance by background and by role
How to use this book	What these notes are, how pages are laid out, how to get help
Advice to readers	How to read a page, work the labs, and avoid common traps
Preview this book locally	Build and serve the notes on your own machine
Running notebooks	Local (canonical), Binder (primary online), Colab (secondary)
Your first lab	Lab 1 in the browser, with numbered steps and no install
Book map	Big-picture layout and what each application produces
Full outline	This page
Learning paths	Routes by background and by role, plus course overlays
Related open resources	Neighbouring courses, notes, and research codes
Join and contribute	How to report, write, review, or join as a section editor
Status and versions	Public preview v0.1 (until v1.0), versioning, maturity labels
Feedback	Typos, broken links, unclear pages
License	CC BY-NC-SA 4.0 content, MIT code, contribution sign-off

Preparation

Page	What it covers	Maturity
Preparation overview	Self-check before Foundations; optional maths refreshers	Draft
Programming from zero	Notebooks, cells, and a kernel, for readers new to programming	Draft
Power systems for ML people	Grid, swing equation, faults, and CCT in machine-learning language	Draft
Machine learning for power-systems people	Loss, training, and generalization in power-system language	Draft
Prep 01: NumPy basics	Array workflows	Stub
Prep 02: ODE integration	Numerical integration of small ODEs	Stub
Prep 03: Tiny neural regression	A minimal neural network fit	Stub
Prep 04: Gradients	Gradients and automatic differentiation by hand	Stub

Foundations

Chapter	What it covers	Maturity	Lab	Needs
Foundations overview	Shared SMIB spine used throughout the book	Draft		Either bridge primer
SciML method families	Map of SciML families used for power-system dynamics and stability	Draft		Glossary
Problem framing	Forward dynamics, stability assessment, and inverse estimation as distinct tasks	Draft	Lab 1	Glossary: state, parameter
SMIB dynamics	Classical swing equations in consistent units	Draft	Lab 1	PS primer if the machine is new
Reference simulation	A transparent numerical SMIB reference with an event schedule	Draft	Lab 1	SMIB dynamics; first lab
Data and surrogates	Input/output contracts for supervised trajectory surrogates	Draft	Lab 2	Problem framing; ML primer if supervised learning is new
Automatic differentiation	Finite differences versus reverse-mode AD for residual losses	Draft		ML primer or Prep 04
Forward PINNs	Data and physics losses for a known ODE	Draft	Lab 2	Data and surrogates; automatic differentiation
Classical estimation baselines	Simulation-based least squares for named, bounded parameters	Draft	Lab 3	Reference simulation
Identifiability	Noise, poor excitation, and structural non-identifiability	Draft	Lab 3	Classical estimation
Inverse PINNs	Inverse PINNs with trainable named parameters	Draft	Lab 3	Forward PINNs; classical estimation
Neural and ODEs hybrid models	Collocation PINNs versus solver-based learned dynam-	Draft	Lab 4	Forward PINNs

Dynamics (application 1)

Chapter		What it covers	Maturity	Lab	Needs
Dynamics overview	pillar	How will the system evolve under inputs, parameters, and disturbances? Planned chapters	Stub	Lab 4	SMIB dynamics; Lab 1

Stability (application 2)

Chapter		What it covers	Maturity	Lab	Needs
Stability overview	pillar	Will the system remain stable, and how close is it to the boundary?	Draft	Lab 5	Lab 1; SMIB dynamics
Stability and CCT	labels	Rotor-angle security, labels, and critical clearing time by bisection	Draft	Lab 5	Stability overview
Surrogate models for stability tasks		Binary labels, margin estimates, and trajectory surrogates as different products	Draft	Lab 5	Data and surrogates; CCT
ANDES as network reference		Moving from SMIB to a network simulator after the SMIB work	Draft	Lab 5	Lab 5 SMIB path
Direct methods		Equal-area and energy ideas as an independent CCT check	Draft	Lab 5	CCT; SMIB dynamics

Identification (application 3)

Chapter	What it covers	Maturity	Lab	Needs
Identification pillar overview	Which parameters and dynamics explain the measurements?	Draft	Lab 3	Classical estimation
Measurements and excitation	Observation channels, noise models, and window lengths	Draft	Lab 3	Identifiability
Physical vs equivalent parameters	Physical versus equivalent (effective) fitted parameters	Draft	Lab 3	Identifiability
Differentiable simulation	Gradients with respect to (H, D) through a numerical solver	Draft	Lab 3	Automatic differentiation; classical estimation

Advanced topics

Chapter	What it covers	Maturity	Lab	Needs
Advanced topics overview	Optional extensions (operators, uncertainty, converters)	Stub		Three pillar overviews
From assessment to decision	How the three applications connect in one operating-point loop	Draft		Three pillar overviews

Laboratories

Lab	What you deliver	Maturity	Needs
Laboratories index	How the five assessed labs fit together	Draft	Start here
Lab 1: Reference simulation and scientific dataset	SMIB simulator, event schedule, dataset manifest, convergence check	Draft	First lab
Lab 2: Supervised surrogate versus physics residual (PINN)	Matched comparison with a physics-term ablation	Stub	Data and surrogates; Lab 1
Lab 3: Inertia and damping estimation	Least squares versus inverse PINN on the same measurements	Draft	Lab 1; classical estimation
Lab 4: Generalization of a learned dynamic model	Hybrid/neural ODE or DeepONet tested on a held-out domain	Stub	Neural ODEs; Labs 1–3
Lab 5: Stability and CCT	Reference CCT brackets, then a learned model near the boundary	Draft	Lab 1; CCT

Reference

Page	What it covers
Reading list	Core textbooks, simulators, and how per-chapter references work
Glossary	Plain meaning plus power-systems and ML readings of the same word
Notation	Symbols and unit conventions

Learning paths

Maturity: Draft

Routes below are usable now. Week counts stay blank until an institution packages a semester. See [Status and versions](#) and [Join and contribute](#).

One book, one table of contents. The routes on this page are overlays. They are not separate books. [Start here](#) helps you pick a route. On any chapter, the **Before you start** box says whether you can skip ahead.

Complete beginner

For readers new to programming and new to power systems.

1. [Programming from zero](#)
2. [Your first lab](#) (Lab 1 in the browser)
3. [Power systems for ML people](#)
4. [Machine learning for power-systems people](#)
5. [Problem framing](#)
6. [SMIB dynamics](#)
7. The assessed write-up on [Lab 1](#)

You can skip [Running notebooks](#) until you want a local install. Prep notebooks are optional drills while they are still stubs.

From machine learning

1. [Power systems for ML people](#)
2. Glossary (machine and grid)
3. [Problem framing](#)
4. [SMIB dynamics](#)
5. [Reference simulation](#) and [Lab 1](#)
6. Continue at [Data and surrogates](#) and the [Foundations overview](#) order

You can skip [Automatic differentiation](#) if you already use reverse-mode AD, and skip Prep 03–04. Still read the SMIB units before Lab 3.

From power systems

1. [Machine learning for power-systems people](#)
2. [Your first lab](#) if you have not run a notebook before. Otherwise open [Lab 1](#) locally
3. [Problem framing](#)

4. Skim [SMIB dynamics](#) for the per-unit form this book uses
5. [Data and surrogates](#)
6. [Classical estimation](#), then [Forward PINNs](#)

You can skip the power-systems primer. You can skip a full re-derivation of the swing equation after the units skim.

For teachers

Use one book. Assign an overlay. Students who already know one side read the matching primer in week 1 and then join the same labs.

Path ID	Audience	Suggested parts	Notes
<code>path_core</code>	First offering	Preparation → Foundations → Labs 1–3	Week count TBD when an institution schedules delivery
<code>path_stability</code>	Stability / CCT focus	Foundations → Stability → Lab 5	Assumes Lab 1
<code>path_identification</code>	Parameter estimation focus	Foundations → Identification → Lab 3	Classical least squares beside the inverse PINN

Share a week-by-week overlay through [Join and contribute](#). Overlays stay on this page so the sidebar order does not fork into a second book.

For practitioners

Go to the question you already work on. Read that page’s **Before you start** box and fill only the gaps. The [full outline](#) lists a short “Needs” note on each row.

Question you already ask	Open
How will the system evolve?	SMIB dynamics , Lab 1 , Dynamics
Will it stay in step, and how close is the boundary?	Stability , CCT , Lab 5
Which parameters explain the measurements?	Identification , Lab 3

If the learning vocabulary is new, read [Machine learning for power-systems people](#) first. If the machine vocabulary is new, read [Power systems for ML people](#) first.

For researchers

1. [SciML method families](#)
2. [Identifiability](#) and [Physical vs equivalent parameters](#)
3. The pillar you intend to extend: [Dynamics](#), [Stability](#), or [Identification](#)

4. [From assessment to decision](#)
5. [Related open resources](#) for neighboring notes and codes

Cross-domain review of a chapter (one ML reader and one power-systems reader) is a useful contribution: [Join and contribute](#).

Related open resources

This page situates **SciML for PS** among nearby open materials. It is a positioning note for readers and collaborators, not a claim of uniqueness.

Positioning (honest wording)

To our knowledge, there is **not yet** an established open, collaborative *living textbook* that organizes scientific machine learning around power-system **dynamics**, **stability assessment**, and **system identification** as three engineering applications with a shared plant, classical baselines, and an assessed lab spine.

Related research codes, short courses, general SciML notes, and energy-system Jupyter Books **do** exist. This book aims to fill the **teaching-product** gap: one long-lived narrative you can study, cite by version, and help improve. If you know a resource we have missed, please [tell us](#) so this page stays fair.

Neighbors worth knowing

	Resource	Role relative to SciML for PS
Product patterns (how living notes are built).	EECS 245 course notes (Rampure)	Living M CC BY- ment <i>pat</i>
	Physics-based Deep Learning (Thuerey et al.)	Method ideas cite SMIB
	The Turing Way , Dive into Deep Learning , Project Pythia	Models f cutable n
General SciML (methods, not power-system applications).		
	Introduction to Scientific Machine Learning (Bilionis, Purdue)	
	Cornell Virtual Workshop: SciML JHU SciML Course (Drgoa)	

Power systems + AI / data (adjacent open teaching).

Resource	
IEEE course notebooks: AI for Power and Energy Systems (Fran Li)	
Data Science for Energy System Modelling (TU Berlin)	
Power Systems Optimization	
Causality in Electricity Markets	
MIT OCW: Seminar in Electric Power Systems	

PINN / SciML research code for power dynamics (cite; do not treat as a textbook).

The DTU group around Spyros Chatzivasileiadis has published foundational papers and open code for PINNs on power-system dynamics and identification, plus toolboxes and tutorials (Colab demos, assignments, [PowerPINN](#), [PINNSim](#)). See [downloads](#) and the [PSCC 2024 tutorial](#) on power-system PINNs and trustworthy AI. These are **research and tutorial artifacts**, not a collaborative living course book with the Dynamics / Stability / Identification spine used here.

Curated paper and tool lists such as [awesome-ml-power-systems](#) still point learners mainly to **general** ML textbooks; that gap is part of why this resource exists.

How we use neighbors

- **Cite and link** public literature and docs; rewrite in teaching voice.
- **Do not** paste sealed research metrics or unpublished campaign results into student pages.
- **Do not** copy ND-licensed or unlicensed course text (EECS 245, Math 124 patterns are link-only).
- ANDES and similar simulators are **installed dependencies**, not redistributed sources.

Attribution modes and register: maintainer file `docs/references/attribution_register.md`. Licence summary: [License](#).

Suggest a missing neighbor

If you know another open book, course, or long-lived notes site that covers this niche, open a [Feedback](#) issue or discussion with the URL. We will update this page when the comparison changes.

Join and contribute

These notes are a **collaborative teaching resource** under active development. Experts in power-system dynamics, stability, scientific machine learning, and system identification — and careful students — are welcome to help over the long term.

Under development until v1.0

Treat the book as a **public preview v0.1** (v0.x series): structure and goals are intentional; many pages are still thickening until the first stable edition (**v1.0**). See [Status and versions](#).

Cross-domain collaboration

This book is a bridge. A chapter is stronger when a machine-learning reader and a power-systems reader both review it.

Contribution	Why it helps
A glossary row that translates one term into the other field	Gives the next reader a shared word
A “Coming from ML” or “Coming from power systems” sentence on a chapter you know	Lets people jump in without reading everything above
A paired review (one reader from each field) on a pull request	Catches jargon that only one community understands
A teacher overlay (week plan) for Learning paths	Lets another course reuse the same labs
Deepening Labs 1–5, a landscape example, or a Lab 4 option — not a new assessed Lab 6 for a new architecture	Keeps one spine; maintainers: <code>docs/guides/lab_spine_policy.md</code>

A glossary row or a translation sentence is a good first contribution. A single row is enough.

How you can help

Role	Examples
Reader	Report typos, broken links, unclear explanations via Feedback
Contributor	Fix a page, add an exercise, improve a lab notebook (pull request with DCO sign-off)
Reviewer	Comment on chapter proposals and pull requests in your area
Section editor	Own Dynamics, Stability, Identification, or Labs once appointed (GOVERNANCE.md)

Full process: [CONTRIBUTING.md](#).

Conduct: [CODE_OF_CONDUCT.md](#).

Credit: [CONTRIBUTORS.md](#).

What we ask

1. Use **published** sources only — no sealed research metrics on student pages.
2. Write in teaching voice; cite neighbors honestly ([Related open resources](#)).
3. Sign every commit (`git commit -s`) so contributions come in under the same licences as the book (content CC BY-NC-SA 4.0, code MIT).
4. Keep the three applications — dynamics, stability, identification — as the core unless a governance discussion says otherwise.

Interest in a longer role

Open a GitHub issue with the **Collaboration / section editor** template, or start a Discussion in the Collaborators category once Discussions are enabled. We expect to appoint section editors gradually as the preview matures.

Licence reminder

By contributing you agree that your content and code may be distributed under this project's dual licence. Summary: [License](#).

Status and versions

Public preview (v0.x)

SciML for PS is released early on purpose so learners and collaborators can use and improve it.

Say both when you cite or announce it: this is a **public preview (currently v0.1)** in the **v0.x** series, and it stays **under development until v1.0**. A site-wide under-development notice remains until that first stable edition.

Stage	Meaning
v0.1 (current)	First public preview tag (v0.1.0). Usable, incomplete.
v0.x	Preview series. Goals and TOC are stable; many chapters are Stub or Draft.
v1.0	First stable edition (see criteria below). Site-wide warning may be removed.
v2.0+	Later editions after major restructures.

Roadmap: [ROADMAP.md](#).

Release notes: [CHANGELOG.md](#).

Governance (v1.0 bar): [GOVERNANCE.md](#).

How we bump versions

Maintainers and collaborators use this when tagging a release. Milestone *what* lives in the roadmap; this section is *when* to bump.

Change	When to ship	Example
v0.N → v0.N+1 (e.g. 0.1 → 0.2)	The matching checklist in ROADMAP.md is honestly done	Lab 1 runnable + early Foundations → v0.2
v0.N.P → v0.N.P+1 (patch) → v1.0	Fixes only; no new roadmap milestone First stable edition criteria in GOVERNANCE.md (and the v1.0 list in the roadmap)	Broken Binder, typos, bad links → v0.1.1 Foundations + Labs 1/3/5 + pillar cores Stable and reviewed
→ v2.0+	Major restructure of scope or TOC	New edition shape

Practice

1. Open the next v0.N checklist in the roadmap.
2. When those items are done, tag v0.N.0, update [CHANGELOG.md](#), and deposit Zenodo when you use DOIs.
3. Between milestones, use patch tags only.

4. Do **not** invent unused minors (no forced path through 0.6–0.9). Skip numbers that have no milestone.
5. Do **not** call it **v1.0** only because calendar time passed.

Chapter labels (**Stub** / **Draft** / **In review** / **Stable**) are per page. The book **version** is a whole tagged release.

Chapter maturity labels

Authors mark pages near the top with one of:

Label	Meaning
Stub	Outline and goals only; little runnable content
Draft	Readable teaching prose; examples and labs still growing
In review	Proposed as near-complete; awaiting outside or editor review
Stable	Reviewed for the current edition; still may receive errata

After v1.0, per-page labels remain so individual chapters can improve without pretending the whole book is frozen.

How to cite

Prefer a **versioned** citation when you rely on a specific wording or figure:

1. Use the site URL and the PDF cover **Last updated** stamp (git commit), or
2. Cite a **tagged release** and Zenodo DOI once published (see root `CITATION.cff`).

Suggested form (update the version when you pin a release):

Albert Poullose, *Scientific Machine Learning (SciML) for Power Systems (PS)*,
 version 0.1 (public preview; under development until v1.0),
<https://sciml4ps.albertpoullose.com>, licensed under CC BY-NC-SA 4.0.

Prefer the live notes

The HTML book is the primary product. The navbar **PDF** is an experimental snapshot; interactive material may flatten. Prefer linking the live page for teaching.

Feedback

Help improve these notes: typos, broken links, unclear explanations, and accessibility issues.

Interim channel (active now)

Open a GitHub issue on the SciML_PS repository with:

- The page **title** and browser URL (copy from the address bar)
- What is wrong or unclear
- Optional suggested fix

Planned Google Form

A Google Form (EECS 245-style) will replace the interim GitHub channel once a form URL is ready.

Do not paste confidential research data or student grades into public issues.

Accessibility

We aim for **WCAG 2.1 AA** so readers can use the notes with assistive technology and without relying on colour alone. If something blocks access — missing figure alt text, low contrast, an uncaptioned video or plot, or a keyboard trap — report it through the GitHub channel above (page title, URL, and what you hit).

Authors: every figure needs an `:alt:` description; every important plot needs a short text caption; do not encode meaning only in colour. Checklist: `book/chapter_template.md`.

License terms for reusing these notes: [License](#).

License

©2026 Albert Poulouose.

This book uses **two** licenses on purpose.

Book content — [CC BY-NC-SA 4.0](#)

Applies to the notes in `book/` (text, notebook prose, and figures), `slides/`, `media/`, and the generated website and PDF.

You may:

- read, share, and adapt this material for **non-commercial** teaching and learning
- do so only with **attribution** to Albert Poulouose / SciML for PS
- release any adapted version under the **same** license (ShareAlike)

You may **not** use the content commercially (paid courses, training products, or printed editions for sale) without **written permission** from the copyright holder.

Full legal text: [LICENSES/CC-BY-NC-SA-4.0.txt](#) in the repository.

Code — [MIT](#)

Applies to `src/sciml_ps/`, `scripts/`, `tests/`, `templates/`, and code cells in notebooks.

You may use, modify, and redistribute the code, including in commercial software, provided you keep the copyright notice and MIT permission notice.

Full legal text: [LICENSES/MIT.txt](#) in the repository.

Contributions

Outside contributions are accepted under the [Developer Certificate of Origin \(DCO\)](#) with **inbound** = **outbound** licensing: content you submit is licensed under CC BY-NC-SA 4.0, and code under MIT, matching this repository. See [CONTRIBUTING.md](#) and [Join and contribute](#).

Suggested attribution

Albert Poulouose, *Scientific Machine Learning (SciML) for Power Systems (PS)*,
<https://sciml4ps.albertpoulouose.com>, licensed under CC BY-NC-SA 4.0.

Prefer a tagged release or Zenodo DOI once published (see root `CITATION.cff` and [Status and versions](#)).

Third-party software and citations

Installed tools keep their own licenses (for example ANDES is GPL-3.0; we install and call it, we do not redistribute its source here). Citations and reuse of external teaching assets are recorded for maintainers in the repository attribution register.

Details for the whole repository: root [LICENSE](#).

Preparation

Preparation overview

Maturity: Draft

This overview is readable; the four notebooks below are still stubs. See [Status and versions](#) and [Join and contribute](#).

Learning goals

- Confirm Python/NumPy fluency for array workflows
- Integrate a simple ODE and plot trajectories
- Fit a small neural regression model and compute gradients

Choose your bridge

Preparation is optional when you already know the missing side. Pick the primer that matches you, then use the notebooks below as drills.

If this is you	Read
New to programming	Programming from zero , then Your first lab
Machine learning, new to power systems	Power systems for ML people
Power systems, new to machine learning	Machine learning for power-systems people
Choosing a longer route	Start here and Learning paths

Prep package (four notebooks)

The four notebooks are also in the left sidebar under **Preparation**. Today they are **stubs** (scaffold cells only); expand them before relying on this section for a full course offering.

Notebook	Focus	Status
Prep 01 — NumPy basics	Arrays, broadcasting, plotting	Stub
Prep 02 — ODE integration	<code>solve_ivp</code> on a damped oscillator	Stub
Prep 03 — Tiny neural regression	Tiny MLP curve fit (NumPy or PyTorch)	Stub
Prep 04 — Gradients	Manual vs AD gradients	Stub

Download a local copy: use the notebook page's download control when the theme shows one, or open the files from the repository folder `book/200_prep/`. Lab launch paths (local / Binder / Colab) are documented under [Running notebooks](#). Prep notebooks stay stubs for now — Binder/Colab badges for Prep arrive when those notebooks are filled.

Diagnostic

After the four notebooks, simulate a damped oscillator and fit a simple regression model to its samples. If that is painful, work through [Programming from zero](#) and [Your first lab](#), then return to Lab 1.

Maths refresher (optional, external)

If linear algebra or multivariable calculus feels rusty, these **external** sections from the [EECS 245 course notes](#) (Suraj Rampure, University of Michigan; **CC BY-NC-ND 4.0**) are useful background. We **link only** — we do not copy or adapt their text (ND forbids distributing derivatives). Read them on their site; then return here for power-system examples.

Topic	EECS 245 section	Used in SciML for PS
Partial derivatives	Partial derivatives	Automatic differentiation
Least squares	Least squares	Classical estimation , Lab 3
Gradients and gradient descent	Gradient descent , convexity	Training and inverse fits
Eigenvalues	Eigenvalues and eigenvectors	Small-signal / modal ideas (later)
SVD / low-rank / PCA	Low-rank PCA approximation,	Identifiability , model reduction

Source: Rampure.

What is not in Preparation

ANDES (network power-system simulator) is **not** part of this section. Lab 1 uses a small SMIB helper without ANDES. You meet ANDES later if a stability / CCT lab or network chapter needs it (separate environment). Tool docs: [Reference — Reading list](#).

Next

- [Foundations overview](#)

Programming from zero

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

This page is for readers who have not programmed before. You can run the first laboratory without installing anything. A longer book on data and dynamical systems, when you want one, is Brunton and Kutz [2022].

Words you will see

Word	Meaning in these notes
Python	The programming language used in the labs
Notebook	A web page made of text boxes and code boxes. You read the text and run the code
Cell	One box. A text cell explains. A code cell computes
Run	Ask the computer to carry out that one cell. Use the play button, or press Shift+Enter
Kernel	The live Python session behind the notebook. Restarting the kernel forgets earlier cells, so you run the notebook again from the top
Package	Reusable code with a name. The teaching package in this book is called <code>sciml_ps</code>
Error	A red message. Read the last line. It usually says what was missing

The [first-lab walkthrough](#) points at each one while you use it.

Run one notebook today

1. Open [Your first lab](#).
2. Follow the Google Colab steps (a free browser session). Binder is the fallback if you prefer not to use a Google account.
3. Stop when you see a plot of rotor angle against time. That means the notebook ran.

What you should see after the setup cell: a line that ends with `imports ok`.

Practice after that

The four notebooks under [Preparation](#) are the next practice (arrays, a small differential equation, a tiny neural fit, gradients). They are still stubs. Use Lab 1 for a complete run, and treat Preparation as optional drills while those notebooks grow.

If a later chapter feels hard only because of the code, come back to this page and to the walk-through's troubleshooting list.

Where to go next

If you also need	Read
The power-system story	Power systems for ML people
The learning story	Machine learning for power-systems people
A map of the whole book	Start here

Power systems for ML people

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

You already know models, data, and losses. This page names the power-system objects those tools will touch. The textbook treatment is Kundur and Malik [2022]. Stability words follow the IEEE/CIGRE definitions [Kundur et al., 2004, Hatziargyriou et al., 2021]. Symbol-by-symbol translations sit in the glossary.

The grid

Generators produce electrical power. Loads consume it. Transmission lines connect them. The machines must stay in step: their electrical frequency (50 Hz or 60 Hz, depending on the grid) is that shared rhythm. A generator's **rotor angle** is how far its spinning shaft has drifted from the rhythm. **Rotor speed** is how fast that angle is changing.

In machine-learning language, angle and speed are a short **state** sequence. A fault is an input event. The resulting time series is a trajectory you may later predict.

One machine, one infinite bus

Almost every lab uses a **SMIB** plant: a single machine connected to an infinite bus. The infinite bus is a grid so large that its voltage and frequency stay fixed while the one machine moves. That keeps the equations small enough to learn on, and the same equations reappear in simulation, surrogates, stability labels, and parameter estimation.

The **swing equation** is Newton's law for the rotor, written in electrical units:

- **Inertia** resists a change in speed.
- **Damping** pulls the speed back toward the grid.
- The gap between mechanical power (from the turbine) and electrical power (sent to the grid) sets the acceleration.

In machine-learning language, inertia H and damping D are **parameters**. Forward labs treat them as known inputs. Identification labs treat them as the unknowns to estimate. They are named and bounded. They are separate from the neural network's weights.

Faults, clearing, and the boundary

A **fault** (a short circuit) suddenly reduces the electrical power the machine can send. The rotor speeds up. **Clearing** means protective breakers remove the fault. If clearing is late, the angle runs away and the machine loses synchronism (it falls out of step with the grid).

Critical clearing time (CCT) is the latest clearing time that still leaves the machine in step, for one stated fault and one study window. In machine-learning language, CCT is a margin: a distance to a decision boundary. Stable or unstable is a label on that scenario. A small error in

the trajectory can still flip the label when the case sits near the boundary. That is why this book reports CCT error and false-safe decisions together with trajectory error.

Per-unit

Engineers scale power, voltage, and time to a chosen base so the numbers sit near 1. Labs here use per-unit power and radian angles. The dataset manifest records the base. A model trained in one base and tested in another is a different problem.

Why a physics term appears in the loss

Labeled recordings from a real grid are scarce, and a wrong stability call has a safety cost. A **physics residual** asks the network to satisfy the swing equation, alongside matching the samples. That is the reason a data loss and a physics loss appear together. The chapter that writes both losses is [Forward PINNs](#).

Where to go

1. [Problem framing](#) (three tasks on this one plant)
2. [SMIB dynamics](#) (the equation)
3. [Your first lab](#) if you want to see a trajectory before reading further

If programming itself is new, read [Programming from zero](#) first.

Machine learning for power-systems people

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

You already know models, measurements, and simulation. This page names the learning words these notes use, in power-system language. Training and neural networks: Bishop and Bishop [2024]. Data and dynamical systems: Brunton and Kutz [2022]. Physics-informed training: Karniadakis et al. [2021]. Both readings of each word are in the glossary.

A model is a function with knobs

A **neural network** is a flexible function. Its knobs are called **weights**. Weights are free numbers inside the function. They are separate from inertia H and damping D . When a chapter says the estimands are (H, D) , those two stay named parameters with units and bounds. The network is only there to represent a trajectory or a missing term.

Data

One **example** is one simulation run or one measured window. Inputs might be time, a clearing time, or a parameter value. Outputs might be rotor angle and speed over that window, or a stable/unstable label. A table of many runs is the dataset. The [Lab 1](#) manifest is that table, written down.

Loss, in the language of least squares

A **loss** is a single number that is small when the model agrees with the evidence you named. Least squares between a simulated angle and a measured angle is a loss. Training moves the knobs to reduce that number.

Gradient descent takes a step downhill on the loss. **Automatic differentiation** computes that downhill direction for you. One pass through the training examples is an **epoch**. You will see these words in [Automatic differentiation](#) and [Classical estimation](#). Classical least squares remains the baseline beside every neural estimator in this book.

Generalization

A fit can match the runs used to train it and fail on a new disturbance. **Overfitting** means the model memorized the training runs. These notes split data by **scenario** (a different clearing time or a different fault), and they ask you to report the scenarios that were held out. A low loss on the training window is a training result. A held-out disturbance is the check.

What “physics-informed” adds

A physics-informed neural network keeps the data loss and adds a **residual** of the differential equation. The swing equation you already trust becomes part of the number being minimized. The

residual is small when the network's angle and speed satisfy that equation at a set of time points. Details: [Forward PINNs](#).

Where to go

1. [Data and surrogates](#)
2. [Classical estimation](#) (the baseline you already know how to judge)
3. [Forward PINNs](#)

If you have never run a notebook, start with [Programming from zero](#) and [Your first lab](#). If the machine model itself should be refreshed, skim [SMIB dynamics](#) for the units this book uses.

Prep 01 — NumPy basics

Status: stub — scaffold cells only; expand before relying on this notebook for a course offering. Practice array creation, slicing, and a simple plot.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x) + 0.1 * np.random.randn(x.size)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, alpha=0.3)
plt.show()
print(x.shape, float(y.mean()))
```

Prep 02 — ODE integration

Status: stub — scaffold cells only; expand before relying on this notebook for a course offering.

Integrate a damped oscillator with `scipy.integrate.solve_ivp`.

```
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

def rhs(t, z, zeta=0.2, w0=2.0):
    x, v = z
    return [v, -2 * zeta * w0 * v - w0**2 * x]

sol = solve_ivp(rhs, (0, 10), [1.0, 0.0], max_step=1e-2)
plt.plot(sol.t, sol.y[0])
plt.xlabel('t')
plt.ylabel('x')
plt.grid(True, alpha=0.3)
plt.show()
```

Prep 03 — Tiny neural regression

Status: stub — scaffold cells only; expand before relying on this notebook for a course offering.

Fit $y = \sin(x)$ with a small MLP when PyTorch is available.

```
try:
    import torch
    from torch import nn
    x = torch.linspace( -3, 3, 200).unsqueeze(1)
    y = torch.sin(x)
    model = nn.Sequential(nn.Linear(1, 32), nn.Tanh(), nn.Linear(32, 1))
    opt = torch.optim.Adam(model.parameters(), lr=1e -2)
    for _ in range(1000):
        opt.zero_grad()
        loss = ((model(x) - y) ** 2).mean()
        loss.backward()
        opt.step()
    print('torch loss', float(loss))
except ImportError:
    print('PyTorch not installed in this env  use course_core with pytorch.')
```

Prep 04 — Gradients

Status: stub — scaffold cells only; expand before relying on this notebook for a course offering. Compare a hand derivative to automatic differentiation when PyTorch is available.

```
import numpy as np
w, t = 2.0, 0.3
print('analytic', w * np.cos(w * t))
try:
    import torch
    tt = torch.tensor(t, requires_grad=True)
    f = torch.sin(w * tt)
    f.backward()
    print('autograd', float(tt.grad))
except ImportError:
    print('Install PyTorch in the course env for AD practice.')
```

Foundations

Foundations overview

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: a bridge primer if either field is new | **Coming from ML:** skim [Power systems for ML people](#), then [Problem framing](#) | **Coming from power systems:** skim [Machine learning for power-systems people](#), then [Problem framing](#) | **New to both:** [Programming from zero](#), then both primers

Shared method core before the three application pillars. Use a classical **SMIB** (single-machine infinite-bus) system repeatedly so the same equations appear as forward simulation, surrogate learning, stability labels, and parameter estimation.

Shared SMIB spine

Foundations reuse one classical **SMIB** model so forward simulation, surrogates, stability labels, and parameter estimation share the same equations and notation.

Coming from another field

Chapters in this part can be read out of order when the **Before you start** box on the page is already satisfied.

If this is you	Do this first
Machine learning, new to power systems	Power systems for ML people , then Problem framing
Power systems, new to machine learning	Machine learning for power-systems people , then Problem framing
New to both	Programming from zero and Your first lab
Comfortable with both	Use the outline and open the chapter you need

Word-by-word translations: [Glossary](#). Routes by role: [Learning paths](#).

Foundations teach the shared plant and an early physics-informed path (supervised baseline, then forward and inverse PINNs). [Dynamics](#), [Stability](#), and [Identification](#) deepen the three engineering questions. Other SciML families (Neural ODEs, operators) arrive when needed — see [SciML method families](#) and [Lab 4](#).

Learning outcomes

After this part you should be able to:

- State forward, inverse, and stability-assessment tasks with clear inputs and outputs

- Distinguish **states** (time-varying) from **parameters** (usually fixed unknowns)
- Build or use a transparent numerical SMIB reference with documented events
- Explain data loss vs physics residual loss in a PINN
- Contrast classical least squares with inverse PINNs for named parameters
- Know when Neural ODEs / operator learning are a different SciML branch

Suggested reading order

1. [Problem framing](#) — tasks, state vs parameter, forward vs inverse
2. [SMIB dynamics](#)
3. [Reference simulation](#) then [Lab 1](#)
4. [Data and surrogates](#)
5. [Automatic differentiation](#)
6. [Forward PINNs](#)
7. [Classical estimation](#)
8. [Identifiability](#)
9. [Inverse PINNs](#) then [Lab 3](#)
10. [Neural ODEs and hybrids](#)

Optional landscape: [SciML method families](#).

Pillars after this part

Pillar	Engineering question
Dynamics	How will the system evolve?
Stability	Stable or not, and how close to the boundary?
Identification	Which parameters explain the measurements?

SciML method families (landscape)

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: names of PINN and Neural ODE in the glossary | **Coming from ML:** these are the families this book uses on one grid model | **Coming from power systems:** a PINN is a neural model trained with a residual of the equations you already write | **New to both:** read both primers on [Start here](#), then use this page as a map

Reader path

Practice order on the shared plant: [Lab 1](#) (reference) → [Lab 2](#) (supervised surrogate versus PINN residual) → [Lab 3](#) (inverse) → [Lab 4](#) (Neural ODE / operator options) → [Lab 5](#) (CCT). Pick the physics-coupling level from the engineering question and the data contract, not from climbing the ladder below. The table on this page says where each family lives in the book.

Learning goals

- Name major SciML families used for power-system dynamics and TSA
- Say what each family **learns** (trajectory, vector field, operator, ...)
- Avoid treating every neural method as “a PINN”

Reading

- Physics-informed machine learning overview [Karniadakis et al., 2021]
- [Overview and models](#) chapter of Thuerey et al. [2021]

Plain-words map

SciML here means: neural models with **physics structure** for dynamics or margins — not only black-box stable/unstable classifiers.

Family	Learns (roughly)	Teaching role in this book
Supervised surrogate	Map features $\rightarrow$ trajectories / labels	Mandatory baseline (Lab 2)
PINN (collocation)	$\hat{x}(t)$ with residual loss	Core forward / inverse path
Neural ODE / hybrid	Vector field (+ optional known physics)	Lab 4 option
Operator learning (DeepONet / FNO)	Function $\rightarrow$ function	Advanced / Lab 4 option; not inverse named- θ default
Differentiable simulator	Sensitivities through a solver	Identification (differentiable simulation); Lab 3 option
Classical simulation	Reference trajectories	Lab 1 and all baselines

For **transient stability**, families differ mainly by whether they output trajectories, decisions, or operators — and how CCT search sits on top. Details: [Stability pillar](#).

How tightly is physics coupled?

A second way to sort the same families is by **how tightly the physics is built into training**. This ordering follows the method textbook by Thuerey et al. [2021]. Here it is restated for the SMIB swing equation.

Level	Where the swing equation enters	Gradient comes from	In this book
1. Supervised surrogate	Only through simulator-generated labels	Data mismatch	Data and surrogates , Lab 2 baseline
2. Physics loss (PINN)	Residual r_δ, r_ω at collocation points	Data mismatch + residual (soft constraint)	Forward PINN , inverse PINN , Labs 2–3
3. Differentiable simulator	A numerical integrator (e.g. RK4) rolled out in an AD framework	Mismatch of the simulated trajectory, differentiated through every solver step	Differentiable simulation , Lab 3 option; hybrid models in Neural ODE / hybrid

Plain words: moving down the table, the model is allowed to break the physics less, but each training step costs more and needs a solver you can differentiate.

Tighter coupling is not automatically better. Pick the level from the engineering question and the data contract, not from the ladder.

Further reading

- Neural ODEs [Chen et al., 2018] and universal differential equations [Rackauckas et al., 2020], developed in [Neural ODEs and hybrid models](#)

Common misconceptions

- “DeepONet is a PINN variant”
- Ranking methods by hype instead of task fit
- Skipping classical simulation because a neural method exists
- “The most tightly coupled method is always the best one”

Problem framing

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: the glossary entries for state and parameter | **Coming from ML:** three different prediction problems on one plant | **Coming from power systems:** forward simulation, stability, and estimation stay separate questions | **New to both:** open [Start here](#), then this chapter

Learning goals

- Formulate forward dynamics, stability assessment, and inverse estimation as distinct tasks
- Distinguish **states** from **parameters**
- Contrast forward simulation with inverse identification

Reading

- Power-system context: Kundur and Malik [2022]; SciML landscape: Karniadakis et al. [2021]
- [Book map](#) for the three applications

Three tasks (same SMIB, different questions)

Task	Given	Infer	Evaluate
Forward dynamics	Model, initial state, inputs, parameters θ	Trajectories $x(t)$	Trajectory / rollout error
Stability assessment	Operating point, contingency, clearing rule	Stable/unstable or CCT	False-safe rate, CCT error
Inverse estimation	Measurements (+ known inputs)	Parameters θ (and often $x(t)$)	Recovery, identifiability, held-out events

Do not mix metrics: a model with low trajectory NRMSE can still be **false-safe** near a stability boundary, or recover the **wrong** parameters.

State vs parameter

State estimation asks: *what is the system doing over time?*

Examples: rotor angle $\delta(t)$, speed $\omega(t)$. States **change with time**.

Parameter estimation asks: *what are the fixed unknown constants in the model?*

Teaching examples: inertia H , damping D . (Converter virtual inertia / droop appear later as electives — keep estimands explicit.)

Analogy:

- **State** = movie frames
- **Parameter** = camera / machine settings

A network can draw a beautiful trajectory while guessing the wrong settings. Nice plots are not enough; later labs need recovery checks and held-out disturbances.

Forward vs inverse

FORWARD: known θ + inputs / disturbance $\rightarrow$ trajectories $x(t)$
 INVERSE: measured trajectories (+ known inputs) $\rightarrow$ unknown θ

Style	Typical goal
Forward PINN / surrogate	Fast trajectories when θ is known or assumed
Inverse PINN	Calibrate named θ from PMU-like data
Classical simulator (e.g. ANDES later)	High-fidelity forward reference

Changing H in a simulator and re-running is **forward**. Given frequency after a load step and asking for H is **inverse**.

Worked example

Specify one SMIB fault-on/clearing case in writing:

1. What is known? What is unknown?
2. Is the goal a trajectory, a stability label/CCT, or a parameter vector?
3. Which metric would decide success?

Lab

- Foundations continue into [Lab 1 — reference simulation](#)

Further reading

- [SciML method families](#) for how the three tasks map onto method choices
- [Identifiability](#) for why inverse estimation needs more than a good fit

Common misconceptions

- “Good waveform fit proves parameters are correct.” — No.
- “PINN replaces the need to define the engineering task.” — No; the task defines the loss and metrics.
- Treating converter GFM knobs and classical (H, D) as interchangeable without saying so.

SMIB dynamics

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Problem framing](#) helps; the symbols are defined on this page | **Coming from ML:** read [Power systems for ML people](#); angle and speed are the state, H and D are parameters | **Coming from power systems:** this is the classical swing model in per-unit form | **New to both:** read the [power-systems primer](#), then this page

Learning goals

- Write classical SMIB swing equations in consistent units
- Relate inertia and damping to post-disturbance trajectories
- Name prefault, fault-on, and postfault stages for teaching experiments

Reading

- Swing equations and transient stability framing [Kundur and Malik, 2022]
- [Lab 1 — reference simulation](#) uses this model

Classical swing model (teaching form)

A common classical-machine SMIB model (per-unit power, radian angles) is:

$$\begin{aligned}\dot{\delta} &= \omega - \omega_0, \\ \frac{2H}{\omega_0} \dot{\omega} &= P_m - P_e(\delta) - D(\omega - \omega_0),\end{aligned}\tag{1}$$

with electrical power of the form $P_e = \frac{EV}{X} \sin \delta$ (details and bases documented in the lab notebook).

Teaching estimands in Lab 3: inertia H and damping D . Keep units and the per-unit base in the dataset manifest.

Event stages

Stage	Network / driving	Typical use
Prefault	Intact network, equilibrium	Initial condition
Fault-on	Reduced P_e capability (e.g. bolted fault)	Energy input / acceleration
Postfault	Cleared topology	Stability vs loss of synchronism

Critical clearing time (CCT) later asks: *how long can fault-on last before postfault trajectories are unstable under a stated rule?* See [Stability and CCT](#).

Worked example

For fixed H , plot qualitative $\delta(t)$ after a short fault vs a long fault. Which parameter mainly scales the “inertia” of the swing?

Lab

- [Lab 1 — Reference simulation](#)

Further reading

- Stability vocabulary used later in the book [Kundur et al., 2004]
- [Stability labels and CCT](#) for what happens after the fault is cleared

Common misconceptions

- Calling every fitted coefficient “physical inertia” without a model map
- Mixing SI and per-unit in the same residual without conversion
- Using converter (M_v, D_v, m_p) language in classical SMIB labs without an explicit elective flag

Reference simulation

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [SMIB dynamics](#) and a way to run code ([Your first lab](#)) | **Coming from ML:** the simulator is the data-generating process for later labs | **Coming from power systems:** document the event schedule the way you document a study case | **New to both:** follow [Your first lab](#)

Learning goals

- Build a transparent numerical SMIB reference with an event schedule
- Document solver settings, failures, and scenario-based data splits
- Treat the simulator as scientific ground truth for later ML comparisons

Reading

- [Lab 1 — reference simulation](#)
- Numerical ODE notes in [Preparation](#)

Why a reference first

Before PINNs or surrogates, you need:

1. Equations and parameters in one place
2. An event log (fault inception, clearing, horizon)
3. A convergence / failure policy (do not silently delete failed runs)
4. A **manifest**: scenario IDs, seeds, solver tolerances, units

Machine learning cannot repair an undocumented dataset.

Minimum deliverables (Lab 1)

- SMIB integrator (teaching code in `src/sciml_ps`)
- At least **two** numerical settings compared (e.g. step size or method)
- Dataset split **by scenario**, not by randomly shuffling time points across events
- Flagged numerical failures retained as records

Worked example

Compare trajectories for the same physical scenario at two time steps. Report max $|\delta|$ difference and whether the stability label (if defined) flips.

Lab

- [Lab 1 — Reference simulation and scientific dataset](#)

Further reading

- [Lab 1](#) protocol and dataset record fields

Common misconceptions

- Deleting “ugly” trajectories that failed to converge
- Splitting every time sample randomly into train/test (leaks the same event)
- Calling ANDES optional until network labs — classical SMIB does not require ANDES

Data and surrogates

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Problem framing](#) | **Coming from ML:** a surrogate is supervised learning on simulator trajectories | **Coming from power systems:** read the loss section of [Machine learning for power-systems people](#) | **New to both:** read the [ML primer](#), then this chapter

Learning goals

- Define matched input/output contracts for a supervised trajectory surrogate
- Separate scenario splits from within-trajectory splits
- State what a surrogate can and cannot claim about physics and stability

Reading

- [Lab 2 — surrogate versus PINN](#); neural-network training basics [Bishop and Bishop, 2024]

Supervised surrogate

Given features (e.g. time, operating point, clearing time, or a parameter vector) predict trajectories or a reduced output. Training uses labeled simulator data only — **no** residual term unless you add one (then you are moving toward a PINN).

Strength	Limitation
Simple baseline; clear budget	Needs enough labeled scenarios
Fast evaluation	Extrapolation outside the training distribution is unsafe
Easy ablation	Does not enforce physics unless you add constraints

Data contracts

Document:

- Channels: which states/outputs are supervised
- Normalization fit **on training scenarios only**
- Held-out scenarios chosen **before** training

Worked example

Train a small MLP to predict $\delta(t)$ on a grid of clearing times. Report error on a held-out clearing time near the stability boundary vs far from it.

Lab

- [Lab 2 — Surrogate versus PINN](#) (after Lab 1)

Further reading

- [Supervised training discussion](#) in Thuerey et al. [2021] as method background
- Data-driven dynamics [Brunton and Kutz, 2022]

Common misconceptions

- Low average trajectory error implies correct CCT decisions
- More capacity always fixes distribution shift
- Using test events to tune architecture without saying so

Automatic differentiation

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: Prep 04, or the training section of the [ML primer](#) | **Coming from ML:** this is reverse-mode automatic differentiation applied to a residual | **Coming from power systems:** the software computes the derivative of the loss with respect to H and D | **New to both:** read the [ML primer](#); use Prep 04 when that notebook is filled

Learning goals

- Contrast finite differences with reverse-mode AD for residual losses
- Explain why AD enables physics-informed training
- Name practical pitfalls (discontinuous events, stiff residuals)

Reading

- Automatic differentiation survey [Baydin et al., 2015]; backpropagation intuition [Bishop and Bishop, 2024]

Why AD matters for PINNs

Physics residuals need $\partial\hat{x}/\partial t$ (and often parameter sensitivities). Finite differences are noisy and scale poorly. **Automatic differentiation** gives exact derivatives of the computational graph (up to machine precision) for the operations you coded.

In a typical collocation PINN:

1. Network maps $t \mapsto \hat{x}(t)$
2. AD supplies $\dot{\hat{x}}$
3. Residual $r(\hat{x}, \dot{\hat{x}}, u, \theta)$ enters $\mathcal{L}_{\text{phys}}$

Forward vs reverse mode (intuition)

- **Forward mode:** efficient with few inputs, many outputs
- **Reverse mode (backprop):** efficient with many inputs (weights), few outputs (scalar loss)
— the usual deep-learning case

PINN training almost always uses reverse mode on a scalar total loss.

Gradient scaling

AD gives exact gradients, but exact gradients can still point in an unhelpful direction when the loss terms have very different scales.

In the classical SMIB of [SMIB dynamics](#), ω is in rad/s. It sits near $\omega_0 = 2\pi \cdot 60 \approx 377$, while its swing is a few rad/s. The angle δ is of order 1 rad. Unscaled, r_ω and the ω data term can dominate the total loss, so training mostly fits speed and neglects angle. Inverse problems have the same issue in parameter space: $H \approx 5$ and $D \approx 0.1$ have very different sensitivities.

Practical fixes, in the order to try them:

1. **Non-dimensionalize states.** Train on $\Delta\omega = \omega - \omega_0$ (or $\Delta\omega/\omega_0$) and $\delta - \delta_0$, not raw ω .
2. **Normalize each loss term** by a typical magnitude, or report per-term losses so imbalance is visible.
3. **Rescale parameters.** Optimize $\log H$ and $\log D$, or values divided by a nominal, inside stated bounds.
4. **Weight terms explicitly** (λ_{phys} and per-state weights) and report the weights used.

Plain words: the optimizer follows whichever number is biggest. Make the numbers comparable before trusting the fit.

Further reading: Thuerey et al. [2021] treat this problem in depth, including [scale-invariant physics training](#) and [half-inverse gradients](#) demonstrated on coupled oscillators. Those algorithms are beyond this course; the fixes above are enough for the labs.

Worked example

Differentiate $\sin(\omega t)$ w.r.t. t with AD in PyTorch/`torch.autograd` and compare to $\omega \cos(\omega t)$.

Lab

- Used implicitly in Labs 2–3 when you implement residual losses

Further reading

- Framework AD docs; course prep notebook on gradients
- Maths background (external, link only): [gradient descent](#) and [convexity](#) in the EECS 245 notes [Rampure] (CC BY-NC-ND 4.0)

Common misconceptions

- “AD is approximate like finite differences.” — For smooth elementary ops it is not.
- Differentiating through discontinuous fault switches without careful event handling
- Forgetting `create_graph=True` when higher-order derivatives are needed
- Feeding raw $\omega \approx 377$ rad/s into a loss next to $\delta \approx 1$ rad and blaming the optimizer

Forward PINNs

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Data and surrogates](#) and [Automatic differentiation](#) | **Coming from ML:** a data loss plus a residual of a known ODE | **Coming from power systems:** the swing equation enters the training objective | **New to both:** read the [ML primer](#) and [SMIB dynamics](#)

Learning goals

- Write data and physics losses for a known ODE fragment
- Explain the role of the physics weight λ_{phys}
- Contrast forward PINNs with supervised surrogates

Reading

- Original PINN formulation [Raissi et al., 2019]

Three pieces

1. **Neural network** — usually input t (and maybe known parameters / controls); output $\hat{x}(t)$
2. **Data loss** — match measurements or labeled simulator trajectories

$$\mathcal{L}_{\text{data}} \approx \|\hat{x}(t) - x^{\text{meas}}(t)\|^2 \quad (2)$$

3. **Physics loss** — collocation residual of the ODE/DAE fragment

$$\mathcal{L}_{\text{phys}} \approx \|r(\hat{x}, \dot{\hat{x}}, u, \theta)\|^2 \quad (3)$$

Total loss:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda_{\text{phys}} \mathcal{L}_{\text{phys}} + \dots \quad (4)$$

Plain words: the network is pulled toward the data **and** toward obeying physics. λ_{phys} sets how hard the physics pull is.

Forward-style use

In a **forward** PINN, parameters θ inside the residual are **known and fixed**. The network learns trajectories (or a surrogate solution) consistent with physics and any available labels.

If $\lambda_{\text{phys}} = 0$	If you ignore data
You may fit waveforms that violate physics	You may get a residual-minimizing curve that misses measurements

Worked example

For the classical swing equations with known (H, D) , place collocation points in time and write r_δ and r_ω explicitly. Train with a small labeled window plus residual.

Strengths and limits of physics losses

The points below follow the discussion of physical losses in Thuerey et al. [2021], rewritten for the swing equation.

Strength	Limit
Needs no solver: only the network, AD, and the residual r_δ, r_ω	Soft constraint. The residual is only <i>penalized</i> , so a trained $\hat{x}(t)$ still violates the swing equation by some amount. Report the residual next to trajectory error
Works with sparse or partial measurements (e.g. only ω measured)	Collocation cost. The residual is only checked at collocation points. Fault switching at t_f and t_c needs points concentrated around the events
Same code pattern for forward and inverse problems	Loss weighting. $\mathcal{L}_{\text{data}}$, r_δ and r_ω have different scales, so λ_{phys} (and per-term weights) change the answer. See gradient scaling
Smooth, differentiable output in t	Weak extrapolation. A network trained on $t \in [0, 2]$ s is not a simulator. Outside the training window, or for a new clearing time, it must be re-checked against Lab 1

Plain words: a PINN is a curve fitted to data and nudged toward the physics. It is not a solver that guarantees the physics.

Lab

- [Lab 2 — Surrogate versus PINN](#)
- Optional further practice (does not replace Lab 2): [ANDES PINN surrogate gallery](#)

Further reading

- Thuerey et al. [2021], [physical losses](#) and [discussion of physical losses](#) (Burgers examples; same ideas); [Lab 2](#)

Common misconceptions

- Calling every residual network an “inverse PINN”
- Reporting only $\mathcal{L}_{\text{phys}}$ without trajectory metrics
- Assuming a trained forward PINN is a stability certificate

Classical estimation baselines

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Reference simulation](#) | **Coming from ML:** least squares against the simulator, before any neural inverse method | **Coming from power systems:** the curve fit you know, with named bounds on H and D | **New to both:** read the loss section of the [ML primer](#) and finish [Lab 1](#)

Learning goals

- Set up simulation-based least squares for named parameters with bounds
- Use classical estimators as mandatory baselines before neural inverse methods
- Keep teaching estimands explicit: classical (H, D)

Reading

- [Lab 3 — inertia and damping identification](#); system identification references [Ljung, 1999, Brunton and Kutz, 2022]

Least squares against a simulator

Given measurements $y^{\text{meas}}(t)$ and a simulator $y(t; \theta)$:

$$\hat{\theta} = \arg \min_{\theta \in \Theta} \sum_i \|y^{\text{meas}}(t_i) - y(t_i; \theta)\|^2 \quad (5)$$

with explicit bounds Θ (e.g. positive H , non-negative D).

Strength	Limitation
Interpretable; strong baseline	Needs a reliable simulator in the loop
Bounds and units are clear	Local minima; excitation-dependent
Easy to plot 2-D loss surfaces for (H, D)	Costly if each loss eval is a long TDS

Teaching rule

Lab 3 compares **simulation-based LS** with **inverse PINN** on the **same** measurements, windows, and bounds. Do not silently swap in converter GFM (M_v, D_v, m_p) unless the elective says so.

Worked example

Fix a load-step experiment. Plot a coarse grid of $\mathcal{L}(H, D)$ and mark the true (H^*, D^*) and the LS minimizer.

Lab

- [Lab 3 — Inertia and damping estimation](#)

Further reading

- [Lab 3](#); [identifiability](#) chapter next
- Gradient-based version of the same LS problem: [differentiable simulation](#)
- Maths background (external, link only): [least squares](#) in the EECS 245 notes [Rampure] (CC BY-NC-ND 4.0)

Common misconceptions

- Skipping the classical baseline because “PINN is modern”
- Comparing LS and PINN on different noise seeds or windows
- Reporting percent error without stating the true θ and bounds

Identifiability

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Classical estimation](#) | **Coming from ML:** a low loss can still leave H and D ambiguous | **Coming from power systems:** poor excitation and model mismatch, written for the estimation labs | **New to both:** read the glossary section

Learning goals

- Separate noise, poor excitation, and structural non-identifiability
- Design a simple two-parameter loss-surface or sensitivity diagnostic
- Distinguish **physical** vs **equivalent** fitted parameters

Reading

- Identifiability and experiment design [Ljung, 1999]
- [Physical versus equivalent parameters](#)

Three failure modes

Cause	Symptom	Mitigation
Noise	Scatter around a identifiable minimum	Longer windows, filtering, repeats
Poor excitation	Flat loss valleys	Richer disturbances; multiple events
Structural	Many θ give the same y	Reparametrize; measure more channels; change model

Physical vs equivalent parameters

A fitted coefficient is **physical** when it matches a device/control constant under a consistent model map and informative data. It is **equivalent** (effective) when it mainly absorbs mismatch, weak excitation, or an inconsistent residual — even if trajectory error looks small.

Low NRMSE does **not** certify physical meaning.

Worked example

For (H, D) on one load step, plot $\mathcal{L}(H, D)$. If the valley is elongated along D , damping is weakly informed by that experiment — redesign excitation before trusting $\hat{D}$.

Lab

- Diagnostics required in [Lab 3](#)
- Deeper measurement design: [Measurements and excitation](#)

Further reading

- [Lab 3](#) success criteria; [physical versus equivalent parameters](#) in the identification part
- Maths background (external, link only): [low-rank approximation](#) / [PCA](#) in the EECS 245 notes [Rampure] (CC BY-NC-ND 4.0) — useful intuition for flat directions in a loss surface

Common misconceptions

- “Unique minimizer of training loss $\Rightarrow$ unique physical θ ”
- Bound-clamped estimates reported as if interior physical optima
- Ignoring that a second disturbance may invalidate an equivalent fit

Inverse PINNs

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Forward PINNs](#) and [Classical estimation](#) | **Coming from ML:** network weights and named parameters are different knobs | **Coming from power systems:** H and D become trainable beside the residual | **New to both:** read the [ML primer](#) and [Classical estimation](#)

Learning goals

- Formulate an inverse PINN with trainable named parameters
- Compare to classical LS on the same data contract
- Keep network inputs, known signals, and unknowns in separate buckets

Reading

- Inverse problems section of Raissi et al. [2019]
- [Lab 3 — inertia and damping identification](#)

Where parameters enter

Style	Network	Parameters θ
Forward PINN	Learns $\hat{x}(t)$	Fixed inside residual
Inverse PINN	Learns $\hat{x}(t)$	Trainable variables in residual

The optimizer updates neural weights **and** θ (teaching Lab 3: H , D).

	Bucket	Examples	Role
Buckets (do not confuse).	Network input	t	Feeds the NN
	Known signals	measured $P(t)$, setpoints	Residual / forcing
	Unknowns	NN weights + θ	Optimized

Measured power is **not** a parameter; it is a known signal or observation channel.

Comparison protocol (Lab 3)

1. Same measurements, noise model, window, and bounds
2. Classical LS baseline
3. Inverse PINN
4. Loss-surface or sensitivity diagnostic
5. Test fitted θ on a **held-out** disturbance

Worked example

Write $\mathcal{L}_{\text{data}} + \lambda_{\text{phys}} \mathcal{L}_{\text{phys}}$ with $\theta = (H, D)$ trainable. List which tensors require `requires_grad=True`.

Strengths and limits for identification

The forward-PINN limits (strengths and limits of physics losses) carry over. Inverse use adds three of its own.

- **Shared blame.** Network weights and (H, D) are fitted together. A flexible network can absorb model error that should have shown up in θ . So a low loss does not mean θ is right.
- **Weighting moves θ .** Changing λ_{phys} can shift the estimated (H, D) , not only the fit quality. Report estimates for more than one weight.
- **Residual consistency is not trajectory consistency.** Check the fitted θ by re-simulating with the reference solver (Lab 1). Compare that trajectory with the measurements, not the network output.

A tighter alternative puts the solver itself in the loop: [differentiable simulation](#). There, θ is the only unknown and every candidate trajectory obeys the model exactly.

Lab

- [Lab 3 — Inertia and damping estimation](#)

Further reading

- [Identifiability](#); identification pillar overview
- Thuerey et al. [2021], [discussion of physical losses](#) and [differentiable physics versus physics-informed training](#)

Common misconceptions

- Calling any PINN with parameters in the residual a solved identification problem
- Optimizing θ without bounds or units
- Using GFM triad language without declaring an elective model

Neural ODEs and hybrid models

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Forward PINNs](#) | **Coming from ML:** a numerical solver in the loop, distinct from collocation | **Coming from power systems:** the integrator stays visible; a neural term can sit inside the ODE | **New to both:** read the [ML primer](#) and [Reference simulation](#)

Learning goals

- Contrast collocation PINNs with solver-based learned dynamics
- State when Neural ODEs or hybrid/UDE models are appropriate teaching choices
- Keep operator learning (DeepONet/FNO) as a **different SciML branch**

Reading

- Neural ODEs [Chen et al., 2018]
- Universal differential equations [Rackauckas et al., 2020]

Collocation PINN vs Neural ODE

	Collocation PINN	Neural ODE / hybrid
Dynamics	Residual at collocation points	Integrate a learned or hybrid vector field with a solver
Strength	Flexible collocation; easy physics residual	Respects solver structure; natural rollouts
Watch-outs	Residual $\neq$ solution quality	Long-rollout drift; adjoint cost

Hybrid / UDE idea: known physics terms plus a neural correction for missing dynamics. The correction can absorb parameter error — treat fits as possibly **equivalent**.

Operator learning (awareness only)

DeepONet / FNO learn maps between **functions** (e.g. load profile $\rightarrow$ trajectory). That is not the same as collocation inverse PINN for a few named scalars. See [SciML landscape](#) and Lab 4 options.

Worked example

Define a held-out longer time horizon before training a Neural ODE on SMIB data. Report error at train horizon vs extrapolated horizon.

Lab

- [Lab 4 — Learned dynamics generalization](#)

Further reading

- [Lab 4 — learned dynamics](#)

Common misconceptions

- Treating DeepONet as “a type of PINN”
- Claiming generalization without a pre-registered held-out domain
- Using hybrid corrections as proof of physical parameter recovery

Dynamics

Dynamics pillar overview

Maturity: Stub

This section is under development. The engineering question and lab links are stable; chapter depth grows after Foundations and Labs 1–2.

Before you start

Needs: [SMIB dynamics](#) and [Lab 1](#) | **Coming from ML:** the target is a trajectory or a vector field | **Coming from power systems:** the same study question, with learned models beside the simulator | **New to both:** [Your first lab](#), then this page

Engineering question: How will the system evolve under specified inputs, parameters, and disturbances?

Planned depth (stubs later)

- Surrogate rollouts and event interfaces
- Hybrid physics–ML dynamic models
- Multimachine trajectory tasks (after SMIB mastery)

Labs

- Lab 1 (reference data), Lab 2 (surrogate vs PINN), Lab 4 (learned dynamics)

Further practice (optional)

- After Labs 1–2: [ANDES PINN surrogate gallery](#) (SMIB + symbolic DAE path) — does not replace course labs
- Renewables / frequency projects: [low-inertia IEEE 14 gallery](#)
- Network reference overview: [ANDES as network reference](#)

Model reduction and long-horizon rollouts

A learned dynamics model is usually trained on short windows but used for long rollouts. Two ideas help:

- **Reduced state.** Learn the dynamics in a small latent state (an encoder compresses the measured state, a learned map advances it, a decoder returns physical variables). For SMIB the full state is already small; the idea matters for multimachine and converter-rich systems.
- **Rollout error growth.** One-step accuracy does not guarantee multi-step accuracy, because errors feed back into the next prediction. Lab 4 should report error **versus horizon**, not one number.

Method background: Thuerey et al. [2021], [model reduction and time series](#) (fast-forward overview with pointers to external code; check each linked implementation’s own license).

Discussion: what Dynamics methods can and cannot claim

Can claim (with evidence)	Cannot claim
Trajectory error on stated test disturbances, horizons, and operating points	Accuracy on disturbances, horizons, or operating points outside the test set
Speed-up over the reference simulator for the same task	That a fast surrogate replaces the simulator for new topologies
Physics residual size, if a residual is reported	That a low residual means the physics is exactly satisfied

Rule: every Dynamics result names its reference simulator (Lab 1), its test disturbances, and its horizon.

Status

Overview stub only — grow independently after foundations.

Stability

Stability pillar overview

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Lab 1](#) and [SMIB dynamics](#) | **Coming from ML:** labels and margins, reported beside trajectory error | **Coming from power systems:** CCT and operational labels on the shared SMIB | **New to both:** read the fault section of the [power-systems primer](#), then Lab 1

Engineering question: Will the system remain stable, and how close is it to a stability boundary?

Focus

Transient rotor-angle stability and **critical clearing time (CCT)** under stated fault/clearing rules. Finite-horizon simulation labels are **operational**, not asymptotic certificates.

Learning path

1. [Stability labels and CCT](#) — IEEE/CIGRE placement, three cases, CCT metrics
2. [Surrogate models for stability tasks](#) — classifier vs trajectory vs margin
3. [Direct methods](#) — equal-area / energy as classical check
4. [Lab 5 — Stability and CCT](#) (SMIB first)
5. [ANDES as network reference](#) — after SMIB CCT works; Kundur gallery as linked depth

Relation to foundations

Use the same SMIB reference from Lab 1. Trajectory accuracy alone does not imply correct CCT or safe decisions near the boundary. Network CCT (ANDES / Kundur) reuses the same SciML evaluation ideas on a larger plant — it does not replace Foundations.

Discussion: what Stability methods can and cannot claim

Can claim (with evidence)	Cannot claim
CCT error and false-safe count against the reference simulator, under a stated fault, clearing rule, and study window	A stability certificate beyond the study window or the tested faults
Agreement with the equal-area criterion on the classical SMIB	That a learned margin is protection-grade near the boundary
Ranking of surrogates under the same stability criterion	That a better trajectory fit implies a better CCT

Rule: report false-safe errors separately from average error. Near the boundary, one optimistic mistake matters more than many small ones.

Status

Stability narrative deepened (classification, surrogates, direct methods). Executable SMIB CCT notebooks come after Lab 1–2 tooling is stable. Network path: summarize-then-link to ANDES; pin `envs/course_andes.yml`.

Stability labels and CCT

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Stability pillar overview](#) | **Coming from ML:** CCT is a margin for one scenario | **Coming from power systems:** IEEE/CIGRE placement, plus a bisection definition of CCT | **New to both:** read CCT in the glossary

Learning goals

- Place transient rotor-angle security in the IEEE/CIGRE stability picture
- Define stable/unstable labels under an explicit horizon and criterion
- Define **critical clearing time (CCT)** as a scenario property (a security margin)
- Report signed CCT error and false-safe rate — not accuracy alone

Reading

- [Lab 5 — stability and CCT](#)
- IEEE/CIGRE stability classification [Kundur et al., 2004, Hatziargyriou et al., 2021]; transient stability chapters of Kundur and Malik [2022]

Dynamic security (short)

Dynamic security is the ability of the grid, in its **present operating condition**, to remain stable after credible disturbances such as short circuits. Assessment checks whether machines (and, later, converters) keep synchronism and whether a **margin** such as CCT remains adequate.

That framing comes from IEEE/CIGRE stability and security language [Kundur et al., 2004, Hatziargyriou et al., 2021].

Where this sits in power system stability

In the IEEE/CIGRE framework, post-fault **transient rotor-angle** behaviour belongs under **rotor-angle stability**. The 2020/2021 revision adds **converter-driven** and **resonance** stability. These categories place TSA and CCT in a wider picture; SciML for PS Lab 5 starts with classical rotor-angle / SMIB CCT before converter electives.

Power system stability classification, redrawn for this book after the IEEE/CIGRE task-force definitions [Kundur et al., 2004, Hatziargyriou et al., 2021]. Teaching focus here: **rotor-angle** stability and its **transient** branch; converter-driven stability appears in later electives.

Transient stability — three qualitative cases

Transient stability is the ability to maintain **synchronism** after a **severe transient disturbance**. Rotor angles undergo large excursions; the outcome depends on the initial operating state and the severity of the disturbance.

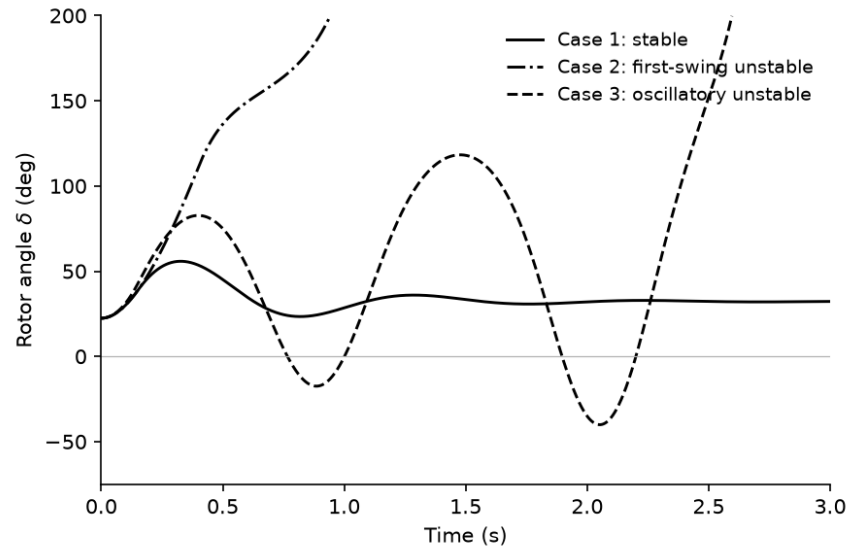


Figure 1: *

Rotor-angle response to a three-phase fault on the classical SMIB model (`sciml_ps.smib`, regenerated by `scripts/make_figures.py`). Case 2 clears the fault too late; Case 3 uses slightly negative damping to mimic poorly damped controls.

1. **Stable.** The angle rises to a maximum, then falls and oscillates with decreasing amplitude toward a new steady state.
2. **First-swing instability.** The angle keeps increasing until synchronism is lost (often insufficient synchronizing torque).
3. **Oscillatory instability.** The first swing looks acceptable, but growing oscillations later drive loss of synchronism.

Classical multimachine TSA study windows are often about **3–5 seconds** (longer, ~ 10 s, when slow interarea modes dominate). With converters, the critical post-fault interval can be shorter, and assessment may need converter-driven phenomena as well as rotor-angle behaviour [Hatziairgyriou et al., 2021]. Finite-horizon simulation labels used in labs are **operational**, not asymptotic Lyapunov certificates.

Critical clearing time

CCT is the longest fault-on duration for which the system remains transiently stable after clearing, under a **stated** stability rule (angle threshold, simulator flag, horizon, ...).

Clearing later than CCT leads to instability (e.g. loss of synchronism) after clearance. CCT is a property of the **operating point + contingency + model + criterion**, not of a neural network.

TSA asks whether synchronism is kept after a disturbance. Operations also need a **margin**: how long can a fault remain before stability is lost? CCT is one such measure.

Operational search. Practitioners often estimate CCT by bisection or grid search over clearing time, assuming a monotone stable→unstable transition inside a bracket. If no valid bracket exists, **label the case** — do not invent a CCT.

Metrics

Metric	Meaning
Signed CCT error	Positive values overestimate available clearing time (dangerous)
Absolute CCT error	Magnitude of mistake
False-safe rate	Fraction of truly unstable cases predicted stable

Accuracy alone can hide an unacceptable false-safe imbalance. Test **boundary-adjacent** scenarios separately from easy cases.

Lab

- [Lab 5 — Stability and CCT](#)
- Classical check when assumptions allow: [Direct methods](#)

Further reading

- [Direct methods](#) for an independent equal-area check on CCT
- [Surrogate models for stability tasks](#) for learned labels and margins

Common misconceptions

- Calling a finite-horizon label a Lyapunov proof
- Using trajectory NRMSE as a proxy for CCT quality
- Ignoring bracket saturation in bisection
- Treating CCT as a property of a trained network rather than of the scenario + criterion

Surrogate models for stability tasks

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Data and surrogates](#) and [Stability labels and CCT](#) | **Coming from ML:** a classifier, a margin model, and a trajectory model are different outputs | **Coming from power systems:** a close trajectory and a correct CCT ranking can disagree | **New to both:** read the [ML primer](#) and the CCT chapter

Learning goals

- Separate **binary labels**, **margin estimates**, and **trajectory surrogates** as different products
- Separate trajectory error from boundary / CCT decision error
- Place a learned surrogate inside an operational CCT search carefully
- Require classical simulation baselines on the same scenarios

Reading

- [Lab 2](#) and [Lab 5](#)
- [Stability labels and CCT](#); [Forward PINNs](#)

What the model is asked to produce

Approach	Typical output	Operational use
Binary classifier	Stable / unstable	Fast screen only — no clearing margin
Margin regressor	CCT-like scalar	Needs careful labels and honesty about extrapolation
Trajectory surrogate	$\delta(t)$ (and related states), then search	Reuse one model for nominal TSA and CCT search if the same criterion is used

Binary stable/unstable labels are not enough for risk-informed decisions that need **how close** the system is to the boundary. Margin regression and trajectory-based search aim at that gap; both still require a stated criterion and a reference simulator.

One trajectory model, two assessment tasks

A useful teaching pattern (planning-oriented screening):

Time -domain reference (e.g. Lab 1 SMIB / later ANDES)

```

 $\rightarrow$  train trajectory surrogate (supervised net or forward PINN)
 $\rightarrow$  TSA screen at nominal clearing
 $\rightarrow$  CCT search by bisection on clearing time

```

The same trained checkpoint can serve both tasks **only if** the stability rule (horizon, threshold, persistence window, ...) matches the rule used on the reference trajectories. Mixing criteria between surrogate and reference invalidates the comparison.

Forward surrogates for TSA

A **forward** surrogate predicts trajectories quickly for many clearing times or operating points. CCT search may call the surrogate instead of the full simulator — but then CCT error and false-safe rate must be reported against the **reference** simulator.

Safe practice	Unsafe practice
Same fault, clearing action, horizon, criterion	Mixing criteria between surrogate and reference
Boundary-focused test set	Only easy stable cases
Signed CCT error + false-safe	Accuracy-only leaderboard
Report that better trajectories need not yield protection-grade CCT	Treating low trajectory RMSE as CCT certification

Improved trajectory fidelity helps screening and comparative studies; it does **not** automatically deliver protection-grade CCT estimation.

Worked example

On SMIB Lab 1 scenarios, compare CCT from dense reference simulation vs CCT from a surrogate with a coarse time grid. Where do rankings disagree? Keep the stability rule identical.

Lab

- [Lab 5](#); trajectory models from [Lab 2](#)

Further reading

- [Forward PINNs](#); [SciML landscape](#)
- Optional published SMIB trajectory-surrogate example (Further reading only — not a course lab): Poulou

Common misconceptions

- “Fast surrogate implies trustworthy CCT”
- “Binary accuracy is enough for operational margins”
- Retraining on the CCT test bracket without disclosure
- Skipping classical energy-function or TDS baselines when available

ANDES as network reference

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: the SMIB path in [Lab 5](#) | **Coming from ML:** the same evaluation protocol on a larger simulator | **Coming from power systems:** network time-domain simulation after the SMIB CCT study | **New to both:** finish [Lab 1](#) and the [stability overview](#) first

After you can run **SMIB** stability and CCT work ([Lab 5](#)), the next plant fidelity step is a **network** simulator. This course uses [ANDES](#) [Cui et al., 2021, ANDES developers] as that reference: load a case, run power flow and time-domain simulation, then apply the **same SciML evaluation ideas** (scenarios, operational labels, CCT brackets, manifests) you already used on SMIB.

ANDES is open-source (GPL v3). We summarize what you need here and point to the official docs for full tutorials and derivations — we do not republish the ANDES manual in this book.

SMIB first

Foundations and assessed Labs 1–5 stay on classical SMIB. Treat this page as the bridge to network depth, not a replacement for Foundations.

Install (short)

Use the course ANDES environment from the repository root:

```
conda env create -f envs/course_andes.yml
conda activate sciml_ps_andes
```

Course pin: **andes==2.0.0** in `envs/course_andes.yml`. After create/activate, run `andes prep` once if prompted. Do not mix ANDES 1.x and 2.x in the same lab narrative.

Optional notebook (Lab 5 extension): [lab05_network_cct_andes.ipynb](#).

Details: [ANDES installation](#).

Concepts you will touch (summaries)

Idea in this course	Meaning for SciML labs	Details
Power flow (PFlow)	Find a steady operating point before dynamics	Power flow
Time-domain (TDS)	Simulate faults and clearing; get trajectories for labels / CCT	Time-domain simulation
Case file	Documented network + machines + events	Data and formats
Plots / channels	Extract angles, speeds, etc. for your metrics	Plotting

Minimal workflow in words: load case → `PFlow.run()` → configure fault/clearing → `TDS.run()` → read trajectories → apply **your** stability rule and CCT search. Full API options stay in the tutorials above and the [reference](#).

Preferred class network case: Kundur

Item	Choice
Case	<code>andes.get_case("kundur/kundur_full.xlsx")</code>
Why	Two-area multimachine plant; matches course stage C after SMIB
Official walkthrough	Critical Clearing Time Mapping

That gallery sweeps fault clearing times across buses, classifies stable / unstable / non-convergent outcomes, and maps CCT. Use it to learn the **ANDES CCT workflow**. Your course extension (when assigned) still reports **SciML metrics**: signed CCT error, false-safe rate, and comparison to a learned surrogate — same spirit as Lab 5 on SMIB.

Start smaller if needed

On a first pass, fix **one** fault bus and a short clearing-time grid before running a full bus-wise map.

Self-study path (ANDES), then return here

1. [Installation](#)
2. [First simulation](#)
3. [Power flow](#)
4. [Time-domain](#)
5. [Data and formats](#)

6. Plotting

Then return to Lab 5 / this network path. Tutorials 07+ (eigenvalue, sweeps, contingency, ...), the [Modeling guide](#), and [Verification](#) are **further practice**, not assessed core.

Other gallery examples (optional)

Full index: [ANDES Gallery](#).

Gallery	How it connects to this book
PINN surrogate (SMIB)	Further practice after Labs 1–2 — does not replace them
Low-inertia frequency (IEEE 14)	Project / Advanced (frequency / RoCoF), not the CCT spine
Forced oscillation (WECC 179)	Large-system elective / project
RL oscillation damping (SMIB)	Outside the core three pillars; curiosity only

Honesty

Network simulation labels and CCT values are **operational** under a stated criterion and model set. They are not automatic certificates of real-world security. Always record ANDES version, case name, fault/clearing definitions, and the stability rule in your manifest.

Citation

When you use ANDES in a report, cite Cui et al. [2021]. Tool links: [documentation](#), [GitHub](#). License: GPL v3.

Direct methods (equal-area and energy ideas)

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Stability labels and CCT](#) and [SMIB dynamics](#) | **Coming from ML:** a classical check beside the learned margin | **Coming from power systems:** equal-area when the assumptions of that criterion hold | **New to both:** read the [power-systems primer](#) and the CCT chapter

Learning goals

- Use a **classical** CCT check (equal-area / energy idea) when assumptions hold, before trusting only bisection or a surrogate
- State when those assumptions fail (damping, multi-swing, converters)
- Keep converter energy-function TSA as an **elective**, not Lab 5 default

Reading

- Equal-area and energy chapters of Kundur and Malik [2022] (check the edition before class)
- [Stability labels and CCT](#)
- [Lab 5 — stability and CCT](#)

Why direct methods belong in a SciML course

SciML surrogates speed trajectories and CCT **search**. Direct methods give an independent **physics-based** estimate of clearing margin under simplified models. Comparing search-based CCT to a classical estimate (when valid) catches criterion bugs and overconfident surrogates.

Equal-area idea (classical SMIB sketch)

For a highly idealized SMIB (constant mechanical power, no damping, classical “before / during / after” network strengths), the **equal-area criterion** balances accelerating and decelerating areas in the P – δ plane. The critical clearing angle (and thus CCT, given fault-on dynamics) is the clearing angle at which the decelerating area just matches the accelerating area.

Teaching use:

1. State the assumptions explicitly.
2. Compute or look up the classical critical clearing angle / time for a chosen SMIB case.
3. Run Lab 5-style bisection with the **same** stability rule as elsewhere.

4. If assumptions hold and the rule is first-swing oriented, expect rough agreement; investigate large disagreements.

Do **not** treat equal-area as a general certificate for damped, multi-machine, or converter-rich cases.

Energy-function / Lyapunov ideas (preview)

Lyapunov-style **transient energy functions** compare “energy” after clearing to a critical energy associated with a stability boundary. They motivate **direct** TSA and CCT without a long time-domain sweep — under a stated model and energy function.

For this course:

- Classical energy ideas deepen understanding of **why** clearing early helps.
- **GFM/GFL converter energy-function CCT** is a research-strength topic and belongs as a later **Advanced / elective** case study, not as the assessed Lab 5 plant.

Relation to Labs and surrogates

Tool	Role
Reference TDS + bisection	Operational CCT under a stated rule (Lab 5 core)
Equal-area / classical energy	Independent check when assumptions allow
Trajectory surrogate + same rule	Fast repeat of the search (Lab 2 → Lab 5)

Same rule across all three. Trajectory RMSE alone does not validate CCT.

Lab

- [Lab 5 — Stability and CCT](#): after SMIB bisection works, optionally compare to equal-area under documented assumptions
- Surrogates: [Surrogate models for stability tasks](#)

Further reading

- Classification context [Kundur et al., 2004, Hatziargyriou et al., 2021]
- Energy-function TSA for converter-dominated grids: see the [Advanced](#) part

Common misconceptions

- “Equal-area replaces the need to state a simulation criterion.”
- “Energy methods always apply to GFM/GFL without new models.”
- Using direct-method CCT from one model to certify a surrogate trained on another plant.

Identification

Identification pillar overview

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Classical estimation](#) | **Coming from ML:** an inverse problem with a measurement contract | **Coming from power systems:** which parameters explain the records | **New to both:** read the [ML primer](#) and finish [Lab 1](#)

Engineering question: Which parameters and dynamic relationships explain the measurements?

Focus

Identifiability, measurement contracts, excitation, and classical vs physics-informed estimators. Teaching Lab 3 uses classical inertia and damping (H, D). Converter-parameter electives (GFM/GFL reduced-order knobs) stay optional and must name estimands explicitly.

Learning path

1. Foundations: [classical estimation](#), [identifiability](#), [inverse PINN](#)
2. [Measurements and excitation](#)
3. [Physical vs equivalent parameters](#)
4. [Differentiable simulation](#) — gradients through the solver
5. [Lab 3 — Inertia and damping](#)

Honesty rule

Low trajectory error does not certify physical parameter recovery. Report bounds, excitation, and a held-out disturbance test.

Discussion: what Identification methods can and cannot claim

Can claim (with evidence)	Cannot claim
Recovery of a known θ in a synthetic test with stated noise and excitation	Recovery on real data without a model-map argument
Held-out disturbance error with frozen $\hat{\theta}$	That $\hat{\theta}$ is a physical constant when the model is mismatched (it may be equivalent)
Which parameter the data inform weakly (loss surface, sensitivity)	That a faster or gradient-based estimator removes weak identifiability

Rule: least squares, differentiable simulation, and inverse PINN are compared on the same data contract, and every estimate is re-simulated with the reference solver.

Measurements and excitation

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Identifiability](#) | **Coming from ML:** features, noise, and window length are part of the data contract | **Coming from power systems:** channels and excitation aimed at H and D | **New to both:** read the glossary

Learning goals

- Specify observation channels, noise models, and window lengths
- Design excitation so target parameters are informative
- Tie measurement contracts to Lab 3 protocols

Reading

- [Lab 3](#); foundations [identifiability](#) chapter; excitation design [Ljung, 1999]

Measurement contract

Before fitting, write down:

Field	Example (SMIB teaching)
Channels	$\delta(t)$ and/or $\omega(t)$ (or PMU-like proxies)
Sampling	Δt , window $[t_0, t_1]$
Noise	e.g. i.i.d. Gaussian on ω with stated σ
Known inputs	P_m , event times
Unknowns	(H, D) inside bounds

Excitation

A single mild load step may leave D nearly flat on the loss surface. Prefer:

- Clear fault-on/clearing swings (Lab 1 events), or
- Multiple magnitudes / clearing times, with at least one **held-out** event for validation

Worked example

Compare a 2-D (H, D) loss surface for a short clearing time vs a longer (still stable) clearing time. Which experiment tightens the valley along D ?

Lab

- [Lab 3](#)

Common misconceptions

- Using the same event for training and “validation” without saying so
- Estimating two weakly excited parameters and claiming both are physical
- Changing noise between LS and PINN comparisons

Physical vs equivalent parameters

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Identifiability](#) | **Coming from ML:** a fitted number is a physical constant only when the model matches the plant | **Coming from power systems:** an equivalent parameter absorbs model error and still has a stated test | **New to both:** read physical and equivalent parameter in the glossary

Learning goals

- Define physical vs equivalent (effective) fitted parameters
- Give examples of mismatch that force equivalence
- Require recovery checks beyond trajectory NRMSE

Reading

- Model structure and parameter interpretation [Ljung, 1999]
- [Identifiability](#)

Definitions

- **Physical:** coefficient matches a device/control constant under a consistent model map and informative data.
- **Equivalent:** coefficient mainly absorbs unmodeled dynamics, bad excitation, or residual mismatch — even if waveforms look good.

Practical checks

1. Loss-surface / sensitivity along each parameter
2. Bound activity (clamped estimates are suspicious)
3. Predict a **held-out** disturbance with frozen $\hat{\theta}$
4. Optional: synthetic recovery with known true θ before real data

Converter elective note

Reduced-order GFM/GFL parameters are a different estimand family from classical (H, D) . If you introduce them, document the simulator map explicitly — do not relabel foreign block constants as if they were identical.

Lab

- Required discussion prompt in [Lab 3](#)

Common misconceptions

- “We estimated D , therefore we know machine damping.”
- Treating bound-saturated values as interior physical optima

Differentiable simulation

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Automatic differentiation](#) and [Classical estimation](#) | **Coming from ML:** gradients through the numerical solver | **Coming from power systems:** the simulator stays the model; the fit uses its derivatives | **New to both:** read the [ML primer](#) and [Classical estimation](#)

Learning goals

- Explain how a gradient with respect to (H, D) can flow through a numerical solver
- Compare differentiable simulation with simulation-based least squares and inverse PINNs on the same data contract
- State what the method does **not** fix: identifiability, model mismatch, and bound activity

Reading

- [Classical estimation](#), [automatic differentiation](#), [inverse PINN](#)
- Thuerey et al. [2021], [differentiable physics](#) and [differentiable physics versus physics-informed training](#) (method background with fluid examples)

The idea

Simulation-based least squares already puts the simulator in the loop: pick (H, D) , simulate, compare with measurements. What it lacks is a cheap **direction** to move (H, D) . A grid search simulates many candidates, and finite differences need extra simulations per parameter.

Differentiable simulation writes the integrator itself in an automatic-differentiation framework. Each solver step is an ordinary differentiable operation, so reverse-mode AD returns

$$\frac{\partial \mathcal{L}}{\partial \theta}, \quad \mathcal{L}(\theta) = \frac{1}{N} \sum_k (\hat{\omega}_k(\theta) - \omega_k^{\text{meas}})^2, \quad \theta = (H, D), \quad (6)$$

from a single forward rollout and one backward pass, where $\hat{\omega}_k(\theta)$ is the **simulated** speed at sample k .

How the gradient flows

For the classical SMIB of [SMIB dynamics](#), one RK4 step maps the state $x_k = (\delta_k, \omega_k)$ to

$$x_{k+1} = \Phi_{\Delta t}(x_k; \theta, X_k), \quad (7)$$

where X_k is the network reactance for the current stage (prefault, fault-on, postfault). The trajectory is a chain $x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_N$, and the loss gradient is the chain rule along that chain. Backpropagation computes it automatically, carrying the sensitivity of each state to H and D backward through every step.

Two practical conditions make this work in the teaching model:

1. **Fault switching depends on time only.** The reactance changes at fixed t_f and t_c , so put both on the step grid. Each step then sits inside one stage, and nothing non-differentiable happens inside a step. State-triggered events (protection acting on δ) need event-aware solvers and are beyond this course.
2. **Scales are comparable.** Fit $\log H$ and $\log D$, and compare speed deviation $\omega - \omega_0$, as in gradient scaling.

Course helper: `sciml_ps.smib_torch.rollout_smib` (fixed-step RK4 in PyTorch, tensors (batch, time, 2)). `parity_error` checks it against the SciPy reference solver used in Lab 1.

Three estimators, one data contract

	Simulation LS (grid or derivative-free)	Differentiable simulation	Inverse PINN
Unknowns	θ	θ	NN weights and θ
Trajectory obeys the model?	Exactly (solver)	Exactly (solver)	Approximately (residual penalty)
Gradient source	None or finite differences	AD through solver steps	AD through network and residual
Cost per iteration	One simulation per candidate	One rollout + backward pass	One network pass over collocation points
Needs	Any simulator	Differentiable solver	Network, collocation design, loss weights
Main failure mode	Coarse grid; many simulations	Long or stiff rollouts give poor gradients; solver must be rewritten in AD	Network absorbs model error; weight-shifting shifts θ

The comparison rule from Lab 3 still holds: **same measurements, noise, window, and bounds** for every estimator, and a held-out disturbance for every estimate.

What it does not fix

- **Identifiability.** Exact gradients of a flat loss surface are still flat. Weak excitation leaves D poorly informed whichever estimator you use ([identifiability](#)).
- **Model mismatch.** The fitted θ is the best value *for the model you differentiated*. If the data come from a different model, $(\hat{H}, \hat{D})$ become **equivalent** parameters ([physical vs equivalent](#)).
- **Bounds.** Report the bounds and whether an estimate ended on one.

Where it leads

Differentiable simulators are also the basis of **hybrid models**: a neural term trained *inside* the solver, next to the known swing equation. See [Neural ODE / hybrid](#). That is how SciML models can later plug into established simulation workflows instead of replacing them.

Lab

- [Lab 3 — Inertia and damping estimation](#), optional section “Differentiable simulation”

Honesty rule

Fast convergence is not recovery. Report the true or reference θ when known, the estimate, the bounds, and the held-out disturbance error, as for every other estimator.

Common misconceptions

- “Exact gradients mean the estimate is exact.”
- “Differentiable simulation is a kind of PINN.” It has no network and no residual penalty; the solver enforces the model.
- Differentiating through a state-triggered switch as if it were smooth

Advanced topics

Advanced topics overview

Maturity: Stub

This section is under development. Start with [assessment to decision](#); other Advanced topics thicken after Labs 1–5.

Before you start

Needs: a skim of the Dynamics, Stability, and Identification overviews | **Coming from ML:** operator learning and uncertainty sit here after a SMIB PINN is familiar | **Coming from power systems:** multimachine and converter pointers sit here after the SMIB labs | **New to both:** follow a route on [Learning paths](#)

Start here

1. [From assessment to decision](#) — how Identification, Dynamics, and Stability connect toward operating-point decisions

Planned directions (stubs later)

- Operator learning (DeepONet / FNO) as a **forward** function→function branch — not a PINN variant
- Uncertainty quantification for estimated parameters
- Multimachine / network scaling after SMIB mastery (ANDES / Kundur path under Stability)
- Selected converter synchronization case studies (after Labs 1–5; VSM–swing bridge first)

Fast-forward topics

Short pointers to directions this book does not teach yet. Each needs Labs 1–5 first. Method background is Thuerey et al. [2021]; linked external code carries its own license.

- **Probabilistic learning for uncertainty.** Generative models (diffusion, flow matching) learn a *distribution* of trajectories or parameters instead of one answer. In this book’s terms: a spread of plausible (H, D) or a probability of instability, instead of a point estimate. Background: [probabilistic learning introduction](#).
- **Graph architectures for networks.** Multimachine systems are graphs (buses and lines). Graph neural networks and graph-based generative models respect that structure and are a natural next step after the Kundur path. Background: [network architectures](#), [graph-based diffusion models](#).
- **Reinforcement learning for control.** Learning control actions (e.g. damping) from interaction. It is outside the three pillars of this book; treat it as curiosity. Background: [reinforcement learning introduction](#); applied example: the ANDES RL gallery entry below.

Linked further practice (ANDES gallery)

Summarize-then-link only — not core TOC spines:

- [Forced oscillation \(WECC 179\)](#) — large-system elective
- [RL oscillation damping \(SMIB\)](#) — outside the three pillars; curiosity
- Gallery index: <https://docs.andes.app/en/stable/gallery/index.html>

Status

Synthesis page filled; other Advanced topics remain stubs until Labs 1–5 deepen.

From assessment to decision

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: the three pillar overviews | **Coming from ML:** one operating-point loop joining prediction, labels, and estimation | **Coming from power systems:** how the three studies connect toward a decision | **New to both:** read the [book map](#) first

Learning goals

- See how the three book applications connect in one operating-point loop
- Distinguish **finished teaching labs** from a **long-horizon** digital-twin programme
- Avoid claiming a finished twin or protection-grade automation

Reading

- [Identification overview](#)
- [Dynamics overview](#)
- [Stability overview](#)

The three applications in one loop

Identification	$\rightarrow$	calibrated parameters / model
Dynamics	$\rightarrow$	fast trajectories (simulator or SciML surrogate)
Stability	$\rightarrow$	TSA label and CCT (or other) margin
Decision	$\rightarrow$	contingency screen, planning study, operator attention

Labs practice pieces of this loop on a transparent SMIB plant:

Piece	Typical lab
Reference trajectories	Lab 1
Surrogate vs physics-informed forward model	Lab 2
Parameter recovery (H, D)	Lab 3
Learned dynamics generalization	Lab 4
CCT and false-safe honesty	Lab 5

Building toward a measurement-informed twin

A **grid digital twin**, in this teaching sense, is a **living model of the dynamics**, not a 3D plant replica:

Grid / PMU -like measurements

```

    ${\rightarrow}$ parameter / model update (Identification)
    ${\rightarrow}$ dynamic model
    ${\rightarrow}$ physics -informed or hybrid surrogate (Dynamics)
    ${\rightarrow}$ security margins such as CCT (Stability)
    ${\rightarrow}$ operator -facing or planning decision
  
```

That programme is a **long-term research** direction. Published teaching evidence in this book is strongest on Foundations + classical SMIB SciML tasks. Treat the twin loop as **aspiration and synthesis**, not as a claim that the course already ships a twin.

Honesty rules

- Finite-horizon CCT searches are planning-oriented unless otherwise certified.
- Surrogates inherit the criterion and plant of their training / reference data.
- Converter-rich and cyber-physical extensions belong in later electives after Labs 1–5 run.

Further reading

- Stability: [CCT](#), [surrogates](#), [direct methods](#)
- Stability definitions [Kundur et al., 2004, Hatziaargyriou et al., 2021]; optional research code [Poulou]

Common misconceptions

- “Digital twin means a visualization of substations.”
- “Finishing Lab 5 means the twin loop is done.”
- Collapsing binary TSA accuracy with operational margin decisions.

Laboratories

Laboratories

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Five assessed laboratories aligned to the course plan. This **is** the full lab set for SciML for PS — we deepen these rather than adding Lab 6+.

Lab	Topic	Status
1	Reference simulation and scientific dataset	Notebook + <code>sciml_ps.smib</code> helper
2	Supervised surrogate versus physics residual (PINN)	Brief only (after Lab 1)
3	Inertia and damping estimation	LS + loss-surface notebook; PINN template
4	Learned dynamics generalization	Brief only
5	Stability and CCT	Protocol ready; SMIB path + optional Kundur/ANDES notebook

Where SciML families are practiced

Family	Lab
Classical reference simulation	Lab 1
Supervised surrogate and PINN residual (with physics ablation)	Lab 2
Classical least squares and inverse PINN	Lab 3
Neural ODE / hybrid or operator learning (held-out domain)	Lab 4
Stability labels and CCT	Lab 5

New architectures should deepen these five labs or sit under [Advanced topics](#) / optional further practice. They do not become a sixth assessed laboratory. Method map: [SciML method families](#).

Install helpers from the repository root: `pip install -e .`

How to run notebooks (local = canonical; Binder = primary online; Colab = secondary): [Running notebooks](#). Online launch is under development and needs a public GitHub repo; contributions welcome via [Join and contribute](#).

Suggested path when building the course: get **Lab 1** solid, then **Lab 3**, then **Lab 5**; fill **2** and **4** after those run.

Books, simulators, and optional external packages live under [Reference — Reading list](#), not as a second lab menu.

Lab 1 — Reference simulation and scientific dataset

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [SMIB dynamics](#) helps; programming is optional if you use the walkthrough | **Coming from ML:** this lab builds the simulator that later labs learn from | **Coming from power systems:** an event schedule, two solver settings, and a dataset manifest | **New to both:** follow [Your first lab](#)

Goals

Deliver an SMIB simulator, event schedule, dataset manifest, and convergence check. Compare at least two numerical settings. Explain scenario-based splits. Retain numerical failures as flagged records.

Materials

Under development

Online launch (Binder / Colab) is a public-preview convenience and may change. Canonical path: local install. Help improve these paths: [Join and contribute](#). Full guide: [Running notebooks](#).

- Notebook: [lab01_reference_sim.ipynb](#) ↗ [Running notebooks](#)
- ↗
- Teaching helper: `src/sciml_ps/smib.py` (`simulate_smib`, `SMIBParams`, `FaultEvent`)

Setup (local — canonical)

```
conda activate sciml_ps
cd /path/to/SciML_PS # repository root
pip install -e .
jupyter notebook book/900_labs/lab_01_reference_sim/lab01_reference_sim.ipynb
```

Binder and Colab need the **public** GitHub repo; until then use local install. No ANDES required.

Deliverables

1. Plot comparing two solver settings on one fault scenario

2. Manifest JSON with scenario IDs, solver metadata, and train/test **scenario** split
3. Short note: whether stability labels flipped between solver settings

Linked chapters

- [Reference simulation](#)
- [SMIB dynamics](#)

Lab 2 — Supervised surrogate versus physics residual (PINN)

Maturity: Stub

This laboratory page is under development. Goals below are the contract; the executable notebook arrives in a later pass.

Before you start

Needs: [Data and surrogates](#) and [Lab 1](#) | **Coming from ML:** a matched supervised baseline beside a residual loss | **Coming from power systems:** read the [ML primer](#), then this comparison | **New to both:** read the [ML primer](#); the executable notebook is still a stub

Why this lab names PINN

The teaching contrast is **data-only** versus **data plus a physics residual** on the same SMIB plant, inputs, and training budget. A collocation PINN makes that residual easy to write for the swing equation and easy to remove in an ablation. This lab is not a survey of every SciML method. Neural ODEs, hybrids, and operator learning belong in [Lab 4](#) and on the [SciML method families](#) map.

Goals

Matched input/output definitions and a documented training budget. Vary labeled-data amount. Report trajectory errors, residuals, and error near event interfaces. Include a controlled ablation removing the physics term.

Further practice (optional)

After this lab exists in course form, the [ANDES PINN surrogate gallery](#) is optional further practice (symbolic DAE extraction). It does **not** replace Lab 2.

Materials

Binder / Colab badges will appear with the executable notebook. Until then use [Running notebooks](#) (local install is canonical). Contributions welcome: [Join and contribute](#).

Status

Stub — no executable notebook yet.

Lab 3 — Inertia and damping estimation

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Lab 1](#), [classical estimation](#), [identifiability](#), and [inverse PINNs](#) | **Coming from ML:** H and D are named parameters with bounds, separate from network weights | **Coming from power systems:** least squares beside an inverse PINN on the same records | **New to both:** read the [ML primer](#) and finish Lab 1

Goals

Compare **simulation-based least squares** with an **inverse PINN** (or collocation residual estimator) on the same measurements and bounds. Vary noise, window length, and excitation. Plot a two-parameter loss surface or local sensitivity diagnostic. Test fitted (H, D) on another disturbance.

Teaching estimands

Classical SMIB **inertia and damping** (H, D) . Do not silently substitute research GFM converter knobs.

Prerequisites

- [Lab 1](#) completed
- Chapters: [classical estimation](#), [identifiability](#), [inverse PINN](#), [physical vs equivalent](#)

Materials

Under development

Online launch (Binder / Colab) is a public-preview convenience and may change. Canonical path: local install. Help improve these paths: [Join and contribute](#). Full guide: [Running notebooks](#).

- Starter notebook: `lab03_inertia_damping_id.ipynb` ù [Running notebooks](#)
- ù
- Simulator: `sciml_ps.simulate_smib` (same classical SMIB as Lab 1)
- Optional differentiable rollout: `sciml_ps.smib_torch.rollout_smib` (needs `torch`; install with `pip install -e ".[torch]"` locally)

Local (canonical): `conda activate sciml_ps → pip install -e . → open the notebook`. Binder/Colab need the **public** GitHub repo.

Protocol

1. Generate a “truth” trajectory with known (H^*, D^*) and additive noise on ω
2. Grid or optimize LS loss vs (H, D) with documented bounds
3. Train inverse PINN / residual estimator with the same data contract
 - * *Optional*: fit (H, D) by [differentiable simulation](#) (gradient through an RK4 rollout) and add it to the table
4. Plot $\mathcal{L}(H, D)$ or slices; discuss physical vs equivalent
5. Freeze $\hat{\theta}$ and simulate a **held-out** clearing time; compare to truth

Deliverables

- * Loss-surface figure
- * Table: true θ , LS $\hat{\theta}$, PINN $\hat{\theta}$, held-out trajectory error
- * Short paragraph: which parameter was weakly informed, and why

Status

Executable notebook provided for the LS + loss-surface path and the optional differentiable-simulation fit; inverse PINN cell is a structured template to complete in class or homework.

Lab 4 — Generalization of a learned dynamic model

Maturity: Stub

This laboratory page is under development. Goals below are the contract; the executable notebook arrives after Labs 1–3 mature.

Before you start

Needs: [Neural ODEs and hybrid models](#) and Labs 1–3 | **Coming from ML:** define the held-out domain before training | **Coming from power systems:** report error against rollout length and unseen inputs | **New to both:** read the [ML primer](#); the executable notebook is still a stub

Goals

Choose either hybrid/neural ODE modeling or DeepONet. Define the held-out domain before training. Test longer rollouts or unseen input functions as appropriate. A simple baseline is mandatory.

Note

DeepONet is operator learning (function→function), a different SciML branch from collocation PINNs.

Materials

Binder / Colab badges will appear with the executable notebook. Until then use [Running notebooks](#) (local install is canonical). Contributions welcome: [Join and contribute](#).

Status

Stub — no executable notebook yet.

Lab 5 — Stability and CCT

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

Before you start

Needs: [Lab 1](#) and [Stability labels and CCT](#) | **Coming from ML:** report false-safe counts and CCT error beside trajectory error | **Coming from power systems:** build the reference CCT by search, then judge a learned model | **New to both:** read the fault section of the [power-systems primer](#) and finish Lab 1

Goals

Create reference labels and CCT brackets on classical SMIB (Lab 1 tooling), then evaluate a learned surrogate or forward PINN near the boundary. Report signed and absolute CCT errors, false-safe decisions, and one case where trajectory ranking and CCT ranking disagree.

Prerequisites

- * [Lab 1](#)
- * Prefer also [Lab 2](#)
- * Chapters: [CCT](#), [surrogate stability](#), optional [direct methods](#)

Reference CCT (no ML yet)

Using `simulate_smib` / `angle_stable`:

1. Fix prefault state and fault inception
2. Sweep or bisect clearing time until the operational label flips
3. Record bracket width and tolerance
4. Optional: if the case matches equal-area assumptions, compare search CCT to the classical estimate ([direct methods](#))

Learned CCT (after Lab 2)

Replace the inner trajectory call with a surrogate; keep the **same** stability rule. Compare $\hat{t}_{\text{CCT}}$ to the reference CCT.

Deliverables

- * Reference CCT table for ≥ 3 operating / fault settings
- * Signed CCT errors for the learned method
- * False-safe count on a boundary-focused set
- * One disagreement case write-up

Materials (launch paths)

Under development

Online launch (Binder / Colab) for the **SMIB** Lab 5 notebook will appear when that executable notebook ships (after Lab 2). Canonical path remains local install. Help improve launch paths: [Join and contribute](#). Guide: [Running notebooks](#).

- * Assessed SMIB path: use Lab 1 tooling locally until the dedicated SMIB CCT notebook is added; then Binder/Colab badges will match Labs 1 and 3
- * **ANDES network extension is local only** (not on Binder / Colab)

Optional network extension (after this lab)

Assessed Lab 5 stays on **classical SMIB**. When SMIB CCT works, the preferred network follow-on is Kundur via ANDES (**not** a sixth assessed lab):

- * Notebook: [lab05_network_cct_andes.ipynb](#) — one-bus CCT sweep + signed error / false-safe metrics (**local / envs/course_andes.yml only**)
- * Helper: `sciml_ps.andes_cct` (requires `envs/course_andes.yml`, **andes==2.0.0**)
- * Bridge page: [ANDES as network reference](#)
- * Official full map (link only): [Critical Clearing Time Mapping](#)

Status

Narrative and protocol ready; full SMIB surrogate CCT notebook follows Lab 2. Network extension notebook + ANDES pin (2.0.0) are in place.

Reference

Reading list

How references work in this book

Each chapter has its own **References** section at the bottom of the page, listing only the works that chapter cites. Use those lists when you study a topic; this page keeps only the few sources the whole book leans on.

All entries come from one shared bibliography (`references.bib` in the [repository](#)), so a source has the same citation everywhere. Missing a key paper, or found a wrong DOI? Please [tell us](#) or [contribute](#) a fix.

Core textbooks

Textbook	Used for
Kundur and Malik [2022]	Power-system dynamics and transient stability
Brunton and Kutz [2022]	Data-driven dynamics
Bishop and Bishop [2024]	Neural networks, training, and backpropagation
Thuerey et al. [2021]	Physical losses and differentiable physics
Ljung [1999]	Classical estimation, excitation, and identifiability

Chapter or section numbers are given on chapter pages only where an edition has been checked.

Simulators

Tool	Role
ANDES [Cui et al., 2021, ANDES developers]	Network simulation reference (GPL-3.0). Course bridge: ANDES as network reference
ANDES Gallery	Optional cases; preferred network CCT example: Kundur CCT mapping

Maths refresher (external, link only)

The [EECS 245 course notes](#) [Rampure] give optional linear-algebra, gradient, and SVD background. They are **CC BY-NC-ND 4.0**, so this book links to them and does not adapt them. Topic table: [Preparation overview](#).

Where this book reuses Physics-based Deep Learning

Thurey et al. [2021] is a method textbook with fluid-dynamics examples. This book takes its ideas and rewrites them for the SMIB swing equation; it does not copy its figures or notebooks.

PBDL chapter	Used in
Overview and models	SciML landscape § How tightly is physics coupled?
Supervised training discussion	Data and surrogates ; pillar Discussion sections
Physical losses, discussion	Forward PINN , inverse PINN
Differentiable physics, DP versus PINN	Differentiable simulation ; Lab 3 option
Improved gradients, half-inverse gradients	Automatic differentiation § Gradient scaling
Model reduction and time series	Dynamics overview ; Lab 4
Probabilistic learning, graph diffusion, reinforcement learning	Advanced overview § Fast-forward topics

Fluids-only examples (Navier-Stokes, airfoils, Lagrangian and meshless methods, GANs) are not part of this course.

Neighbouring courses and code

Other open courses, tutorials, and research codes are listed on [Related open resources](#). Research code repositories are further reading, not redistributable course material; assessed work is only Labs 1–5 under [Laboratories](#).

Glossary

Maturity: Draft

Readable teaching prose; examples, figures, and exercises are still growing. See [Status and versions](#) and [Join and contribute](#).

The same word often means one thing in a power-system course and another in a machine-learning course. This page gives both readings. Symbols and unit rules are on [Notation](#). Textbook depth: Kundur and Malik [2022] for the machine, Bishop and Bishop [2024] for training.

Contributions that add a row, or that correct a translation, are welcome: [Join and contribute](#).

The machine and the grid

Term	Plain meaning	Power-systems view	ML view	First used on
Grid	Generators, loads, and lines that must stay in step	The network under study	The plant that generates the data	Power systems for ML people
Generator	A machine that converts mechanical rotation into electrical power	The device whose rotor angle we track	The source of the state trajectory	SMIB dynamics
Infinite bus	A grid so large its voltage and frequency stay fixed	The far end of a SMIB model	A constant boundary condition	SMIB dynamics
SMIB	Single machine, infinite bus	The shared teaching plant in every lab	The data-generating system reused across tasks	Foundations overview
Frequency	How fast the shared electrical rhythm repeats (50 Hz or 60 Hz)	A system operating quantity	A slow variable beside rotor speed	Power systems for ML people
Fault	A short circuit or similar sudden event	The contingency in a stability study	An input event that changes the dynamics	Stability labels and CCT
Clearing	Breakers remove the fault at a chosen time	A protection action and a study parameter	An event time in the input	Lab 1
Synchronism	The machine stays in step with the grid	The operational stability outcome	The positive class, when labels are used	Stability labels and CCT

States, parameters, and units

Term	Plain meaning	Power-systems view	ML view	First used on
State	A quantity that moves in time	Rotor angle and speed	The sequence the model predicts	Problem framing
Parameter	A quantity treated as fixed during one simulation	Inertia, damping, network data	A named unknown, separate from network weights	Problem framing
Rotor angle δ	How far the shaft has drifted from the grid rhythm	A state, in radians	One channel of the trajectory	SMIB dynamics
Rotor speed ω	How fast the angle is changing	A state	The other channel of the trajectory	SMIB dynamics
Inertia H	Resistance to a change in speed	A machine constant, in seconds	A named estimand in Lab 3	SMIB dynamics
Damping D	A term that pulls speed back toward the grid	A machine constant in the swing equation	The other Lab 3 estimand	SMIB dynamics
Per-unit	A scaling so powers and voltages sit near 1	The unit system of the labs	A preprocessing choice that must match at test time	Notation
Operating point	Power, voltage, and equilibrium before the event	The prefault condition	A feature, or a conditioner, of the example	Problem framing
Trajectory	Angle and speed over a time window	A simulation result	The regression target	Reference simulation
Disturbance	The event you apply (fault, clearing time, input)	A study scenario	The axis you split train and test on	Lab 1

Learning and losses

Term		Plain meaning	Power-systems view	ML view	First used on
Neural network		A flexible function with many knobs	A curve-fit or a dynamics model, depending on the lab	The parameterized function being trained	Machine learning for power-systems people
Weights		The knobs inside that function	Distinct from H and D	The parameters gradient descent updates by default	Machine learning for power-systems people
Loss		One number that is small when the model agrees with the evidence you named	Comparable to a weighted squared error	The training objective	Classical estimation
Data loss		The part of the loss that matches samples	Fit to the measured or simulated trajectory	Supervised error	Forward PINNs
Physics residual		The swing equation evaluated on the network's prediction	The equation you already trust, written as an error	An extra term in the loss	Forward PINNs
Training		Adjusting knobs so the loss falls	An estimation procedure	Gradient descent for a set number of epochs	Automatic differentiation
Epoch		One pass through the training examples	One sweep of the identification records	One optimization epoch	Machine learning for power-systems people
Surrogate		A learned stand-in for the simulator	A faster model of trajectories or labels	A supervised model	Data and surrogates
PINN		A network trained with data loss and a physics residual	The equation is part of the fit	Collocation-style physics-informed training	Forward PINNs
Neural ODE		A numerical solver whose right-hand side includes a learned piece	The integrator stays visible	A different SciML branch from collocation PINNs	Neural ODEs and hybrid models
Overfitting		A close fit on the training data	A model that fails on new data	Memorization of the training data	Machine learning for power-systems people

Checks and honesty

Term	Plain meaning	Power-systems view	ML view	First used on
CCT	Latest clearing time that still leaves the machine in step, for one scenario	A security margin from a stated fault and window	A decision margin for that scenario	Stability labels and CCT
False-safe	Calling a case stable when the reference says it is unstable	The dangerous direction of a stability error	A one-sided classification error	Stability labels and CCT
Identifiability	Whether the measurements can determine the named parameters	Excitation, noise, and model structure	Whether the loss has a unique parameter answer	Identifiability
Physical parameter	The constant in the model you wrote down	Nameplate or model-sheet meaning, with units	Keeps the nameplate meaning when the model matches the plant	Physical vs equivalent parameters
Equivalent parameter	The number that fits the records when the model is incomplete	An effective value, still useful inside its test	A fitted value whose meaning is the fit, not the nameplate	Physical vs equivalent parameters
Scenario split	Train on some disturbances and test on others	Held-out faults or clearing times	The generalization check this book asks for	Lab 1

Notation

Plain-language meanings, with a power-systems reading and a machine-learning reading of the same word, are in the [glossary](#). This page is the symbol table.

Conventions (teaching draft)

Symbol	Meaning
δ, ω	Rotor angle, speed (classical SMIB)
H, D	Inertia, damping (Lab 3 estimands)
P_m, P_e	Mechanical / electrical power
θ	Generic parameter vector
$\hat{x}(t)$	Network-predicted state
$\mathcal{L}_{\text{data}}, \mathcal{L}_{\text{phys}}$	Data and physics losses
λ_{phys}	Physics loss weight
CCT	Critical clearing time

Rules

- * Prefer **per-unit** classical SMIB in labs; document the base in the manifest.
- * Tensor batches in code examples: (**batch**, **time**, **dim**) when tensors appear.
- * Distinguish **physical** vs **equivalent/effective** parameters in prose.
- * Converter GFM symbols (M_v, D_v, m_p) appear only in electives with an explicit model map.

Equation authoring across MyST and manuscripts is documented for maintainers in the repository guides (portable LaTeX equations).

Bibliography

- ANDES developers. Andes documentation. <https://docs.andes.app/en/stable/>. URL <https://docs.andes.app/en/stable/>. Python library for power system modeling and numerical analysis (GPL-3.0).
- A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey, 2015. URL <https://arxiv.org/abs/1502.05767>.
- C. M. Bishop and H. Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024. ISBN 9783031454684. doi: 10.1007/978-3-031-45468-4. URL <https://www.bishopbook.com/>.
- S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, second edition, 2022. ISBN 9781009098489. doi: 10.1017/9781009089517. URL <https://www.databookuw.com/>.
- R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations, 2018. URL <https://arxiv.org/abs/1806.07366>.
- H. Cui, F. Li, and K. Tomsovic. Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis. *IEEE Transactions on Power Systems*, 36(2):1373–1384, 2021. doi: 10.1109/TPWRS.2020.3017019.
- N. Hatziargyriou, J. Milanovic, C. Rahmann, V. Ajjarapu, C. Canizares, I. Erlich, D. Hill, I. Hiskens, I. Kamwa, B. Pal, P. Pourbeik, J. Sanchez-Gasca, A. Stankovic, T. Van Cutsem, V. Vittal, and C. Vournas. Definition and Classification of Power System Stability – Revisited & Extended. *IEEE Transactions on Power Systems*, 36(4):3271–3281, 2021. doi: 10.1109/TPWRS.2020.3041774.
- G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. doi: 10.1038/s42254-021-00314-5.
- P. Kundur and O. P. Malik. *Power System Stability and Control*. McGraw-Hill Education, second edition, 2022. ISBN 9781260473544. URL <https://openlibrary.org/isbn/9781260473544>.
- P. Kundur, J. Paserba, V. Ajjarapu, G. Andersson, A. Bose, C. Canizares, N. Hatziargyriou, D. J. Hill, A. Stankovic, C. Taylor, T. Van Cutsem, and V. Vittal. Definition and Classification of Power System Stability IEEE/CIGRE Joint Task Force on Stability Terms and Definitions. *IEEE Transactions on Power Systems*, 19(3):1387–1401, 2004. doi: 10.1109/TPWRS.2004.825981.
- L. Ljung. *System Identification: Theory for the User*. Prentice Hall PTR, second edition, 1999. ISBN 0136566952. URL <https://openlibrary.org/isbn/0136566952>.

- A. Poulose. Unified physics-informed neural surrogate SMIB TSA/CCT benchmark. URL <https://github.com/albert8943/unified-physics-informed-neural-surrogate-smib-tsa-cct-benchmark>. Research code; optional further reading, not a course lab.
- C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal Differential Equations for Scientific Machine Learning, 2020. URL <https://arxiv.org/abs/2001.04385>.
- M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: 10.1016/j.jcp.2018.10.045.
- S. Rampure. Eecs 245 course notes. <https://notes.eecs245.org/>. URL <https://notes.eecs245.org/>. University of Michigan; CC BY-NC-ND 4.0, linked only.
- N. Thuerey, B. Holzsuh, P. Holl, G. Kohl, M. Lino, Q. Liu, P. Schnell, and F. Trost. Physics-based Deep Learning, 2021. URL <https://physicsbaseddeeplearning.org/>. Online book; cited chapters are linked in the text.