An Introduction to Statistical and Variational Methods for Assimilation and Real Time Modeling

Miguel Alfonso Mendez

mendez@vki.ac.be

Table of Contents

What is a Digital Twin ? Industry 4.0

A **digital twin** is a *dynamic* digital representation of a physical object or system, using sensors and real-time data to predict *in real time* its real-world counterpart's behavior, maintenance needs, and performance, enabling optimization, decision-making and troubleshooting.

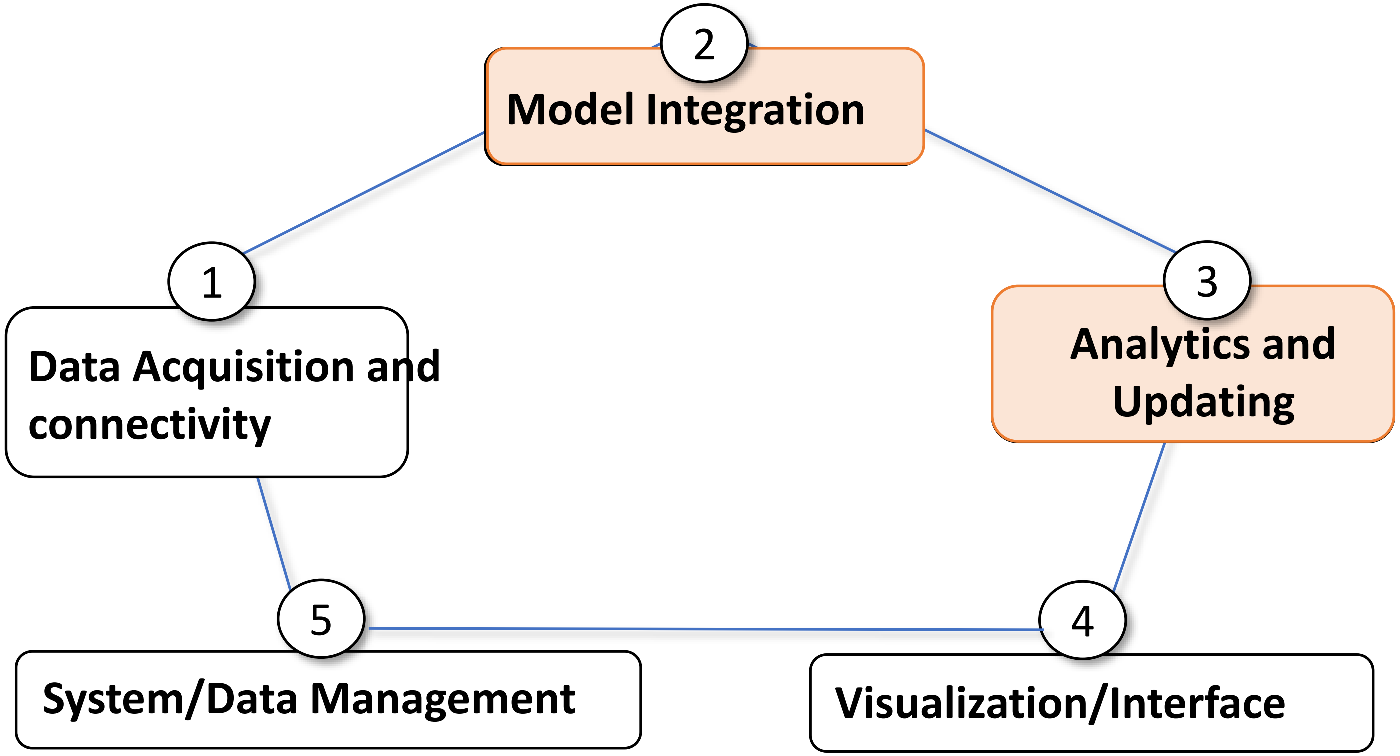
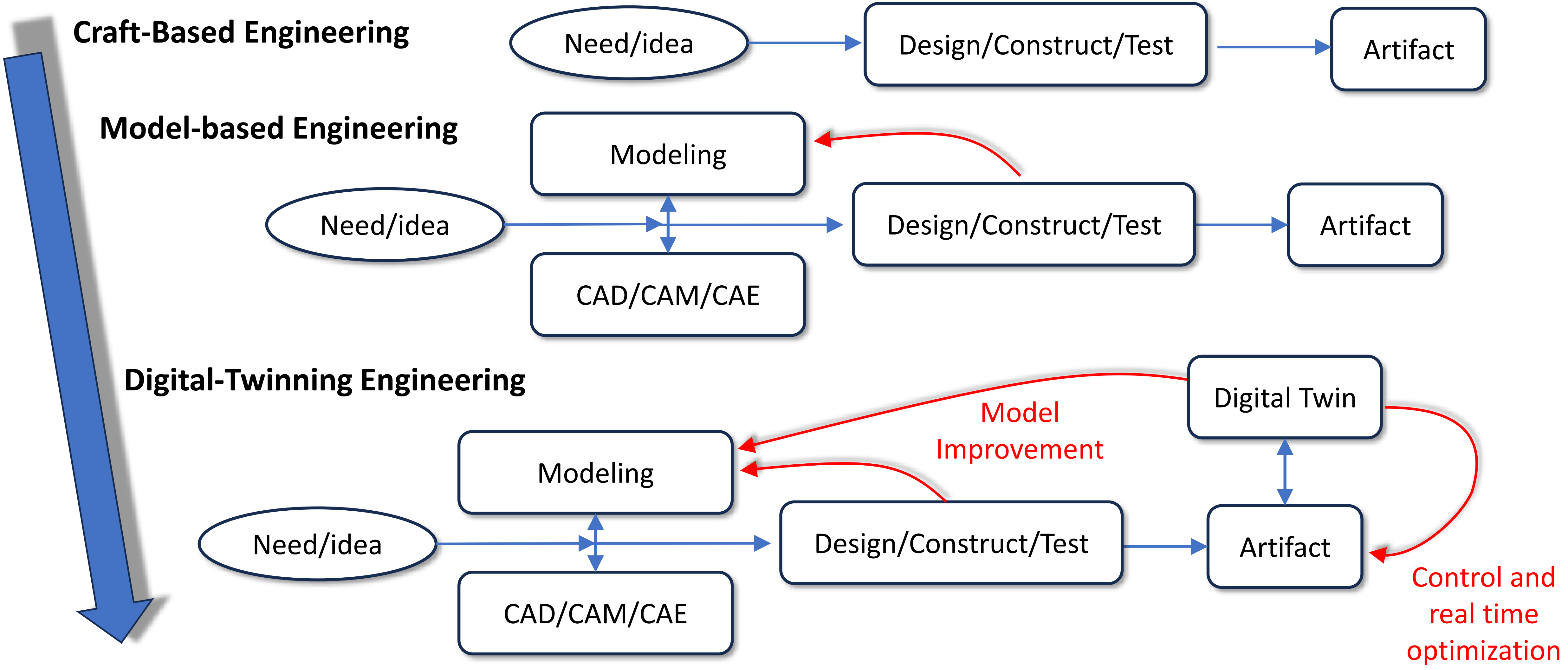


Image generated with ChatGPT4

Perspectives: a change of paradigm?



Inspired by "*On the Notion of Digital Twins: A Modeling Perspective*", Bedir Tekinerdogan

The key enablers

1. Sensor technology: cost and availability

Light

~0.01 €

Temperature

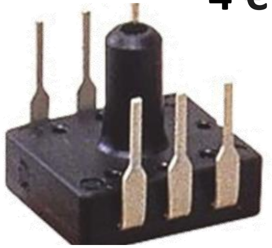
~0. 7 c€

Currents

~3. 5€

Pressure

~4 €



Acceleration

~6 €

Camera (3MP)+ objective

~25 € + 20 €

Micro controller (wifi)

~30 €

2. Connectivity and Internet of Things

3. Machine Learning Engineering

Machine Learning + Domain knowledge + Data Assimilation and Inverse Modeling + Control Engineering

Table of Contents

A general setting for modeling

Consider a dynamical system which depends on some parameters.
These can be link to the system's state by a closure law:

State vector and its time derivative

$$\boldsymbol{s}(t), \dot{\boldsymbol{s}}(t) \in \mathbb{R}^{n_s}$$

'engineering' model parameters

$$\boldsymbol{p}(t) \in \mathbb{R}^{n_p}$$

$$\begin{cases} \dot{\boldsymbol{s}}(t) & = f(\boldsymbol{s}(t), \boldsymbol{p}) \\ \boldsymbol{p}(t) & = \tilde{g}(\boldsymbol{s}(t); \boxed{\boldsymbol{w}_p}) \\ \boldsymbol{s}(0) & = \boldsymbol{s}_0 \,. \end{cases}$$

'surrogate' model parameters

$$\boldsymbol{w}_p \in \mathbb{R}^{n_w}$$

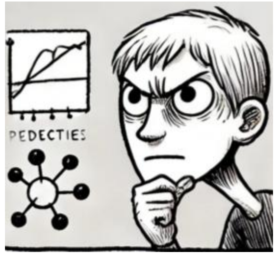
(preferably from physics)

$$f : \mathbb{R}^{n_s \times n_p} \to \mathbb{R}^{n_s}$$

closure law
(machine learning)

$$\tilde{g} : \mathbb{R}^{n_s \times n_w} \to \mathbb{R}^{n_p}$$

If you have $\boldsymbol{w}_p$ $\xrightarrow{\tilde{g}}$ You can predict $\boldsymbol{p}(t)$ $\xrightarrow{f}$ You can predict $\boldsymbol{s}(t)$



Modelists

We look for the model parameters that minimize
the discrepancy with the available data.

$$\mathcal{J}_p(\boldsymbol{w}_p)$$

A general setting for control

Consider a dynamical system which depends on some parameters.
These can be link to the system's state by a closure law:

State vector and its time derivative

$$\boxed{\boldsymbol{s}(t), \dot{\boldsymbol{s}}(t) \in \mathbb{R}^{n_s}}$$

$$\begin{cases} \dot{\boldsymbol{s}}(t) & = f(\boldsymbol{s}(t), \boldsymbol{a}) \\ \boldsymbol{a}(t) & = \pi(\boldsymbol{s}(t); \boxed{\boldsymbol{w}_a}) \\ \boldsymbol{s}(0) & = \boldsymbol{s}_0 \, . \end{cases}$$

(preferably from physics)

$$\boxed{f : \mathbb{R}^{n_s \times n_p} \to \mathbb{R}^{n_s}}$$

Control Action

$$\boxed{\boldsymbol{a}(t) \in \mathbb{R}^{n_a}}$$

Policy Parameters

$$\boxed{\boldsymbol{w}_a \in \mathbb{R}^{n_\pi}}$$

Action policy

$$\boxed{\pi : \mathbb{R}^{n_s \times n_\pi} \to \mathbb{R}^{n_a}}$$

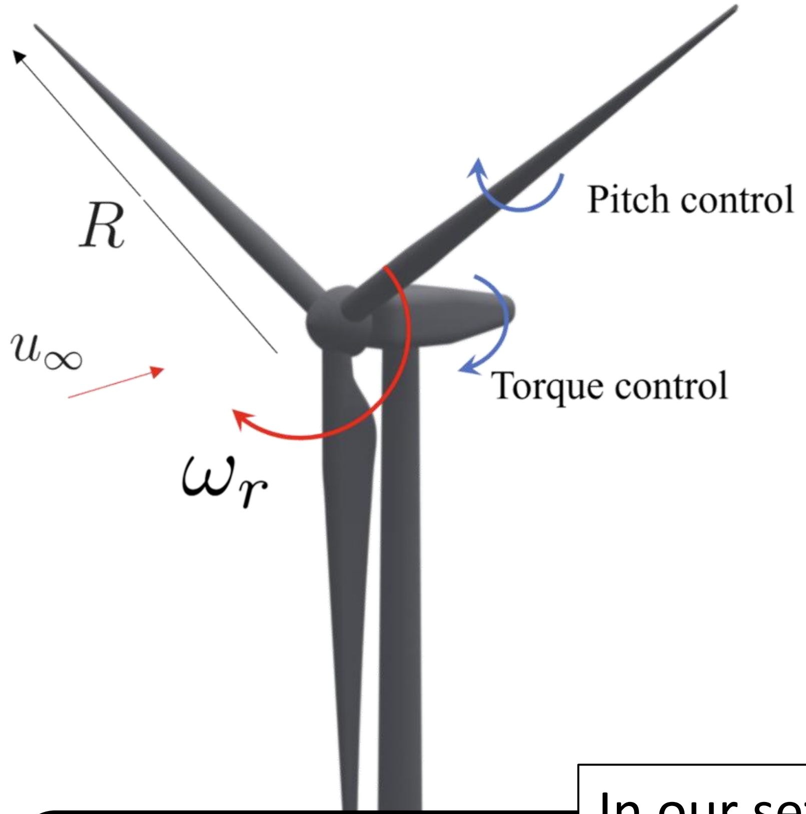
If you have $\boldsymbol{w}_a \xrightarrow{\pi}$ You have actions $\boldsymbol{a}(t) \xrightarrow{f}$ You can control $\boldsymbol{s}(t)$

$$\boxed{\mathcal{J}_a(\boldsymbol{w}_a)}$$

We look for the policy parameters that drive the systems to the desired trajectory

Controlists

An example: Wind turbine modeling/control



The rotor's angular velocity gives:

$$\dot{\omega}_r = \frac{1}{J}\left(\tau_a - \boxed{\tau_g} N_g\right)$$

Gearbox ratio

Generator's torque

Rotor's moment of inertia

Aerodynamic Torque

Power coefficient

Blade Pitch angle

$$\tau_a = \frac{1}{2}\rho A_r \frac{\boxed{C_p(\lambda_r, \boxed{\beta})}}{\omega_r} u_\infty^3$$

$$\lambda_r = \frac{\omega_r R}{u_\infty}$$

In our settings:

$\boldsymbol{s}(t) = \omega(t)$ $\boldsymbol{p}(t) = C_p(t)$ $\boldsymbol{a}(t) = [\tau_g(t), \beta(t)]$

Control goal: maximize energy production without overloading the blades

Table of Contents

Challenges and tools in this short course

1. Training a machine learning model embedded in a dynamical system requires an efficient strategy to compute the gradients riving the learning

→ The Adjoint Method

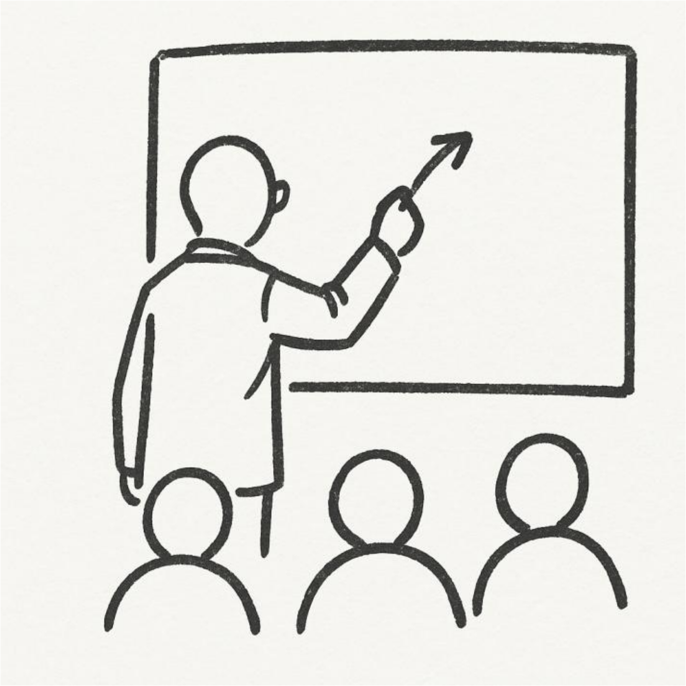
This lecture

2. Sometimes both our measurement and models are incomplete, and the system might be too sensitive to incomplete and noisy measurements

→ The Kalman Filter

Next Lecture

Table of Contents



Blackboard time

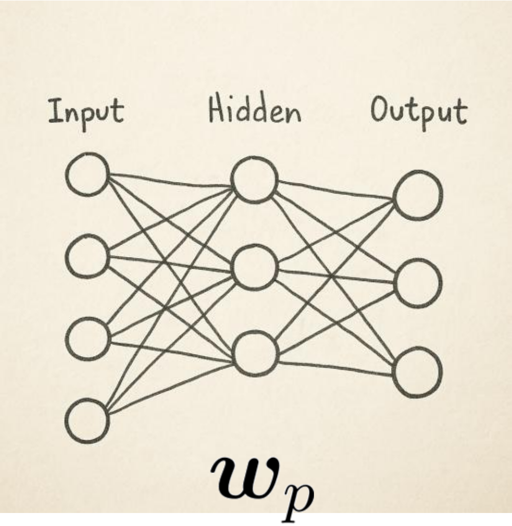
- Make sure you follow the derivation for

1. *Baseline: gradients in the training of a stand-alone ML problem*
2. *Gradients in a dynamical system integration*
3. *The forward problem*
4. *The backward problem*

The problem set

$\boldsymbol{s}(t)$

$\boldsymbol{s}(t)$



$\boldsymbol{w}_p$

$\boldsymbol{p}(t)$

$$
\begin{cases}
\dot{\boldsymbol{s}}(t) &= f(\boldsymbol{s}(t), \boldsymbol{p}) \\
\boldsymbol{p}(t) &= \tilde{g}(\boldsymbol{s}(t); \boldsymbol{w}_p) \\
\boldsymbol{s}(0) &= \boldsymbol{s}_0 \, .
\end{cases}
$$

ODE solver

Relevant quantities

$\boldsymbol{s} \in \mathbb{R}^{n_s}$

$\boldsymbol{p} \in \mathbb{R}^{n_p}$

$\boldsymbol{w}_p \in \mathbb{R}^{n_w}$

We want to minimize the following cost function (this is the simplest possible!)

$$
\mathcal{J}(\boldsymbol{w}_p) = \frac{1}{T}\int_0^T ||\check{\boldsymbol{s}}(t) - \boldsymbol{s}(\boldsymbol{p}(\boldsymbol{w}_p))||^2 dt = \frac{1}{T}\int_0^T \mathcal{L}(\boldsymbol{s}(\boldsymbol{p}(\boldsymbol{w}_p)))dt
$$

The forward approach

Given

$$\mathcal{J}(\boldsymbol{w}_p) = \frac{1}{T}\int_0^T ||\check{\boldsymbol{s}}(t) - \boldsymbol{s}(\boldsymbol{p}(\boldsymbol{w}_p))||^2 dt = \frac{1}{T}\int_0^T \mathcal{L}(\boldsymbol{s}(\boldsymbol{p}(\boldsymbol{w}_p)))dt$$

The gradient is

$$\frac{d\mathcal{J}}{d\boldsymbol{w}_p} = \frac{1}{T}\int_0^T \frac{d\mathcal{L}}{d\boldsymbol{w}_p} dt = \frac{1}{T}\int_0^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}}\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{p}}\right)\frac{d\boldsymbol{p}}{d\boldsymbol{w}_p} dt$$

Were the sensitivities are solution of the forward problem

$$\begin{cases} \dfrac{d}{dt}\left(\dfrac{\partial \boldsymbol{s}}{\partial \boldsymbol{p}}\right) = \dfrac{\partial f}{\partial \boldsymbol{s}}\dfrac{\partial \boldsymbol{s}}{\partial \boldsymbol{p}} + \dfrac{\partial f}{\partial \boldsymbol{p}} \\ \dfrac{\partial \boldsymbol{s}}{\partial \boldsymbol{p}}(0) = \mathbf{0} \end{cases}$$

Note:

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{p}} \in \mathbb{R}^{n_s \times n_p}$$

The more parameters, the larger the forward problem

The backward (the adjoint!) approach

Step 1: Forward integration (state equation)

Integrate the state equation forward in time:

$$\frac{d\boldsymbol{s}}{dt} = f(\boldsymbol{s}, \boldsymbol{p}), \quad \boldsymbol{s}(0) = \boldsymbol{s}_0,$$

to obtain $\boldsymbol{s}(t)$ for $t \in [0, T]$

Step 2: Backward integration (adjoint equation)

Solve the adjoint equation backward in time:

$$\frac{d\boldsymbol{\lambda}}{dt} = -\left(\frac{\partial f}{\partial \boldsymbol{s}}\right)^{\top} \boldsymbol{\lambda} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}}, \quad \boldsymbol{\lambda}(T) = \mathbf{0},$$

to obtain the adjoint variable $\boldsymbol{\lambda}(t) \in \mathbb{R}^{n_s}$

Step 3: Gradient evaluation

Evaluate the gradient of the cost function with respect to $\boldsymbol{w}_p$ as

$$\frac{d\mathcal{J}}{d\boldsymbol{w}_p} = \frac{1}{T} \int_0^T \boldsymbol{\lambda}^{\top} \frac{\partial f}{\partial \boldsymbol{p}} \frac{d\boldsymbol{p}}{d\boldsymbol{w}_p} \, dt.$$

The size of the backward problem does not depend on the number of parameters

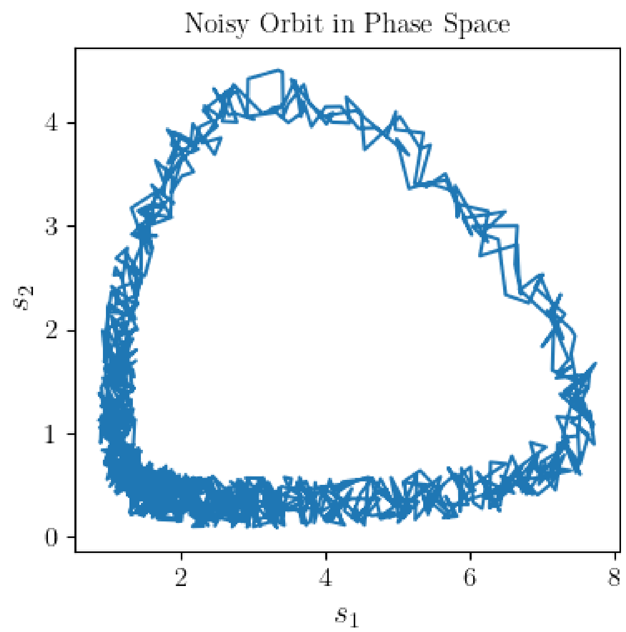
Table of Contents

A toy exercise

We consider the classical Lotka–Volterra predator–prey model:

$$\frac{ds_1}{dt} = as_1 - bs_1s_2$$
$$\frac{ds_2}{dt} = -cs_2 + ds_1s_2$$

The system state is defined as $\boldsymbol{s}(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix}$



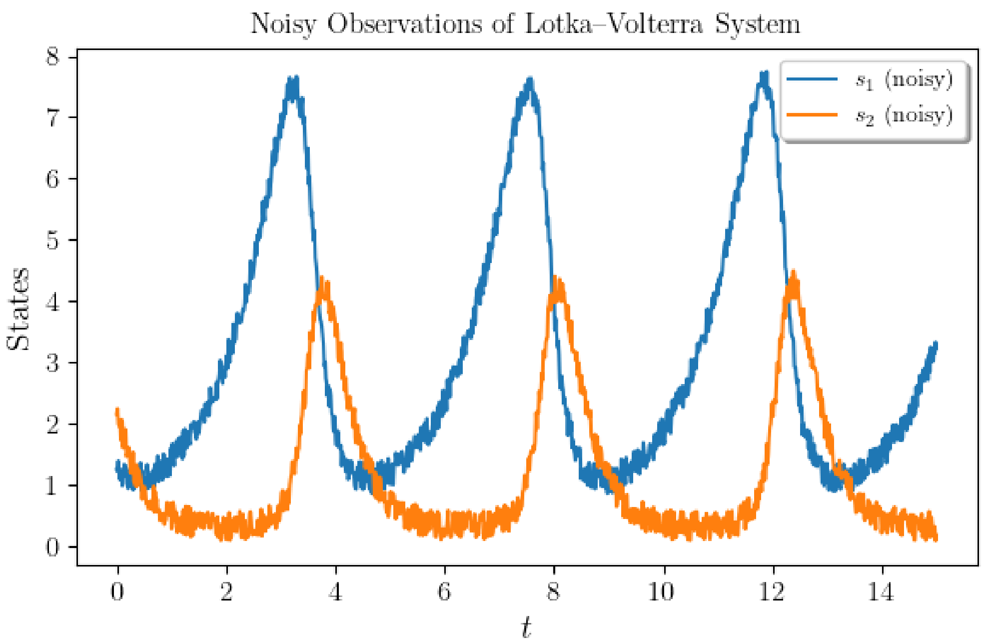
The set of parameters is $\boldsymbol{p}^\star = [a, b, c, d] = [1, 1, 3, 1]$

We observe over the interval $t \in [0, 15]$

The system starts at $\boldsymbol{s}(0) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Observations are simulated by adding random perturbations:

$$\check{\boldsymbol{s}}(t_i) = \boldsymbol{s}(t_i) + \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{U}(\boldsymbol{0}, \delta \boldsymbol{I})$$



Study carefully the four .ipynb files

1_Lotka_Volterra_Data_Generation.ipynb

2_Adam_Optimization_Demo.ipynb

3_Forward_Sensitivity_LotkaVolterra.ipynb

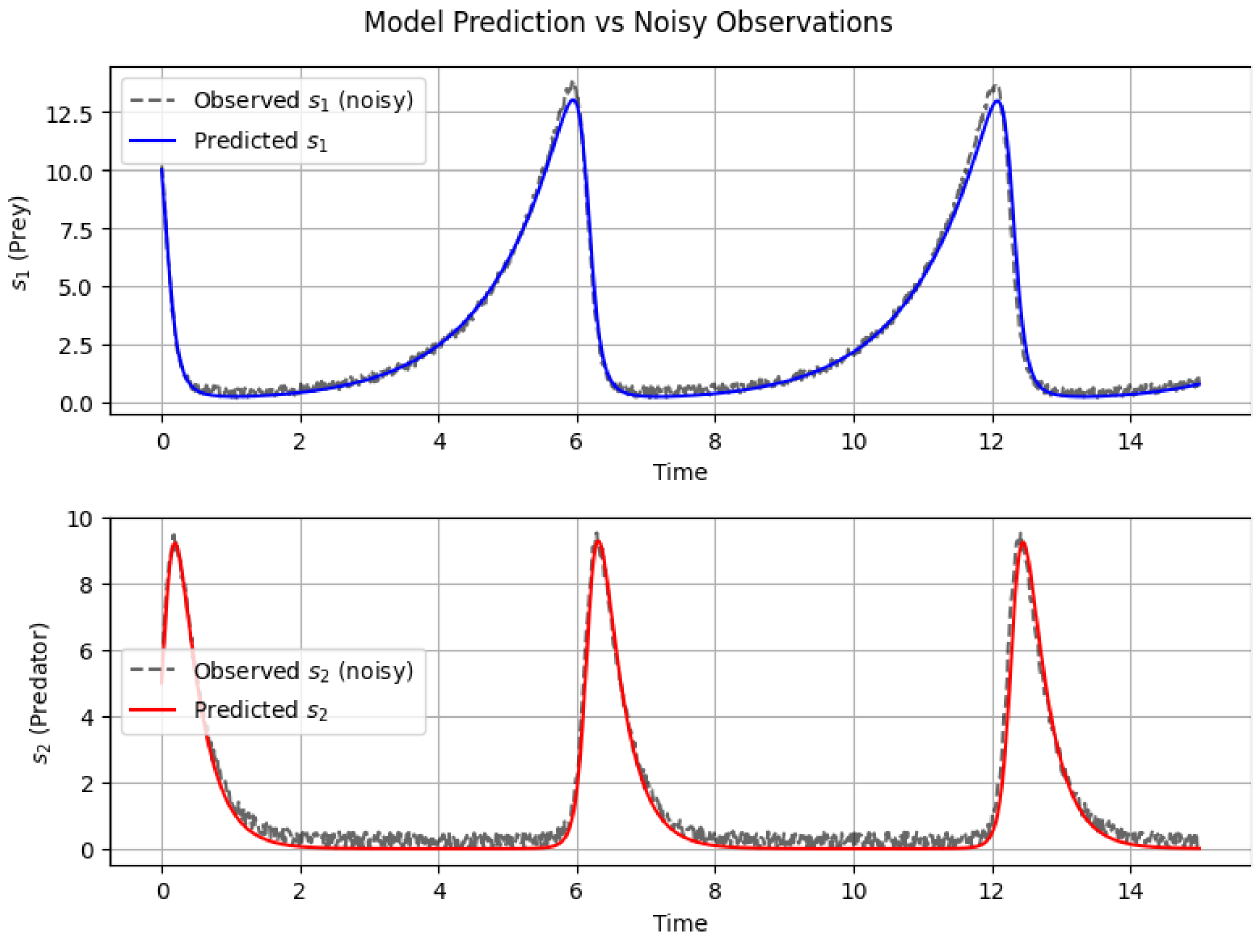
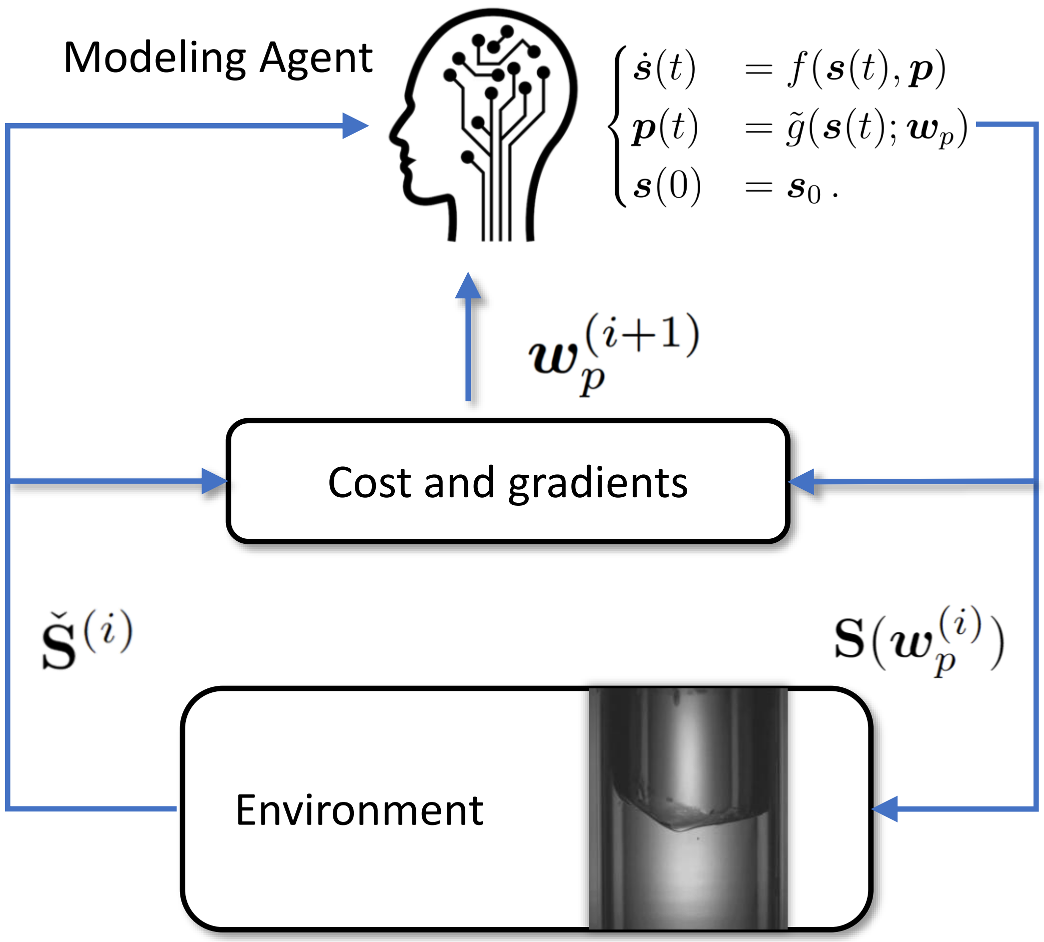


Table of Contents

Ingredients for the "twinning"



void loop () {

1. Collect Observations $\check{\mathbf{S}}^{(i)}$

2. Predict $\mathbf{S}(\boldsymbol{w}_p^{(i)})$

Ensemble averaged,
Adjoint-based

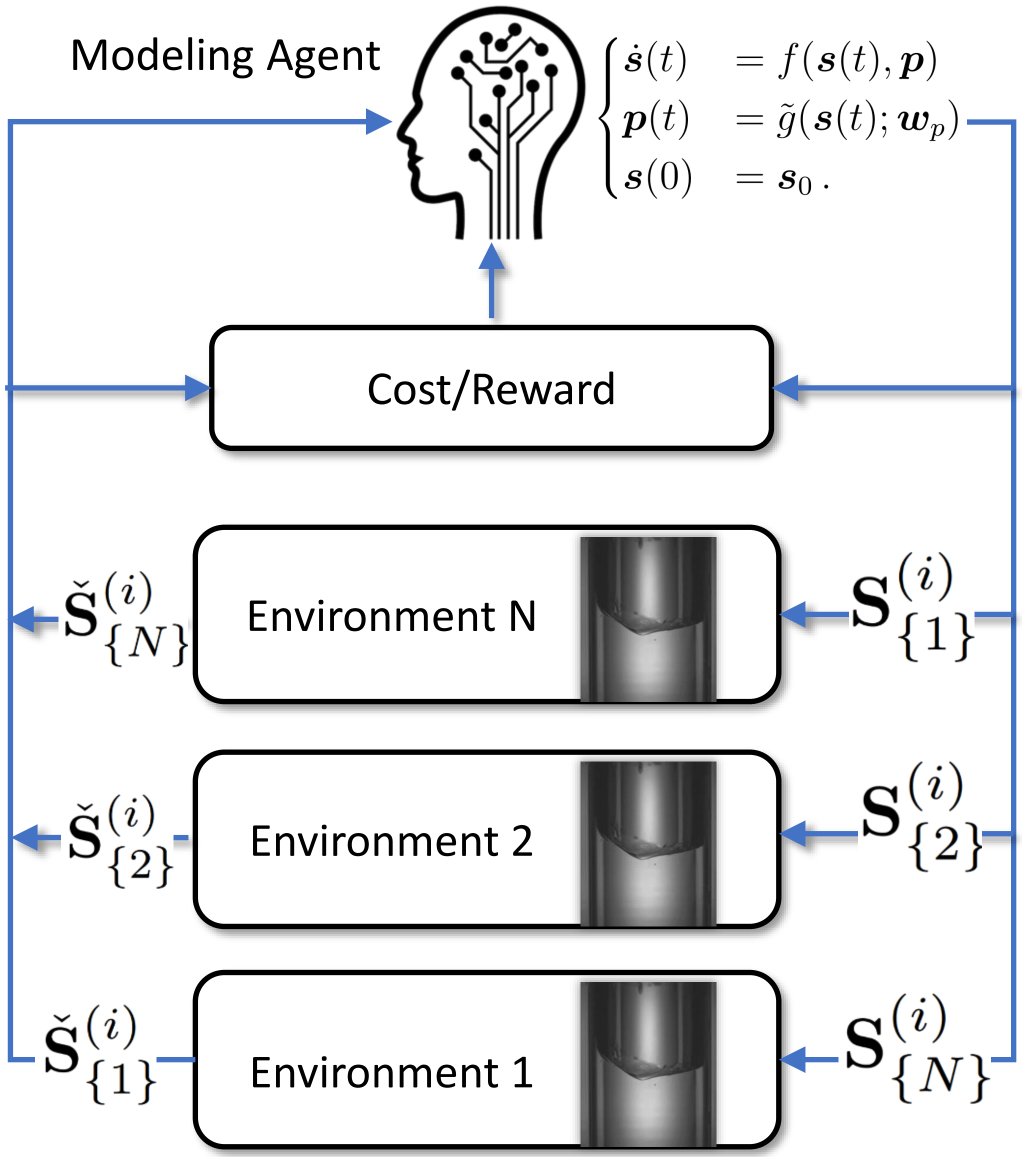
3. Get Performance and Gradients $\mathcal{J}_p(\boldsymbol{w}_p)$ $\boxed{d\mathcal{J}/d\boldsymbol{w}_p}$

4. Update parameters

$$\boldsymbol{w}_p^{(i+1)} \leftarrow \mathrm{Update}(\boldsymbol{w}^{(i)}, d\mathcal{J}^{(i)}/d\boldsymbol{w}_p)$$

}

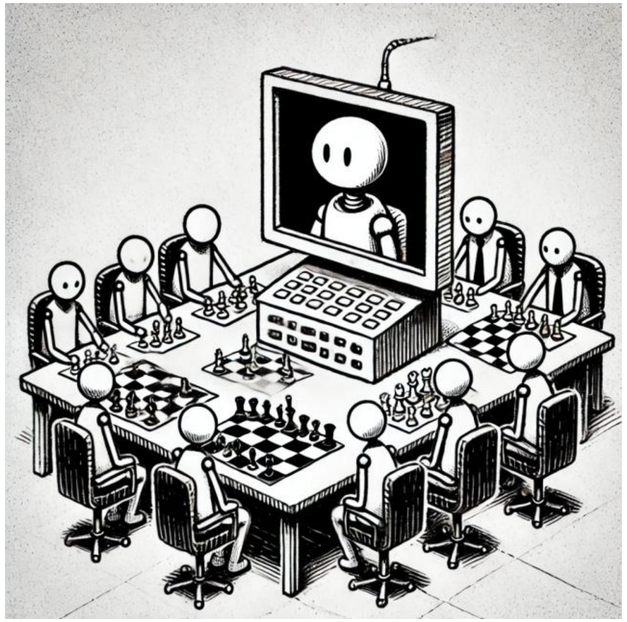
Collaborative Twinning



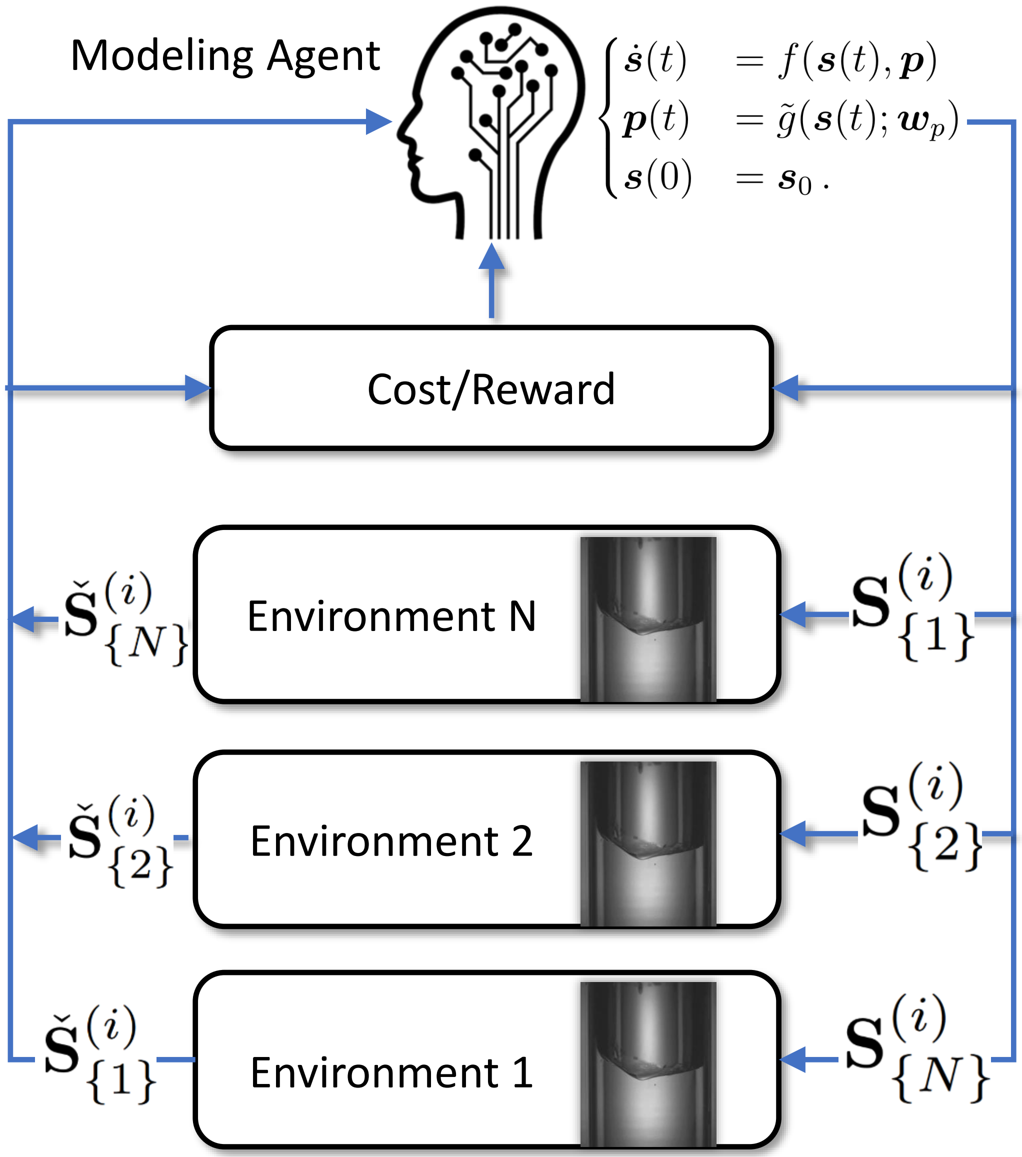
The total cost is the average of the cost in each environment:

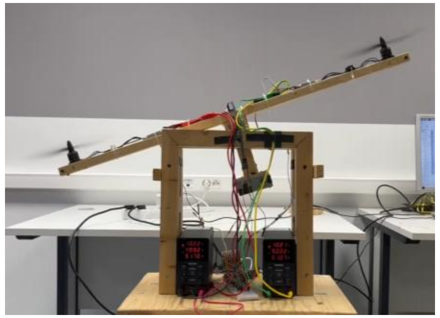
$$\mathcal{J}(\boldsymbol{w}_p) = \frac{1}{N} \sum_{n=1}^{T_o} \mathcal{J}_{p,\{n\}}$$

$$\frac{d\mathcal{J}}{d\boldsymbol{w}_p} = \frac{1}{N} \sum_{n=1}^{T_o} \frac{d\mathcal{J}}{d\boldsymbol{w}_p}\bigg|_{\{n\}}$$



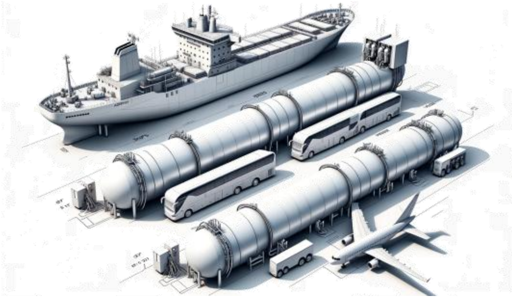
Collaborative Twinnin



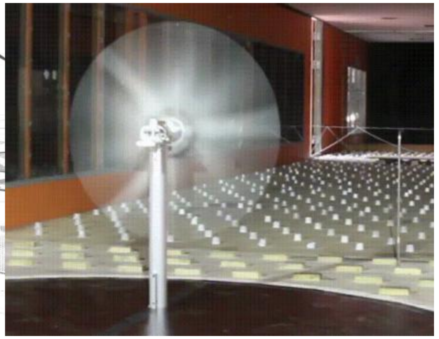


Drones

Cryogenic Storage Units



Wind Turbine Control



Re-Twist:
Reinforcement
Twinning Systems

Summary and Outlook

You have learned that how to propagate gradients through dynamical systems. You should now be able to implement both forward and backward techniques

1. The forward approach offers a robust tool for computing sensitivities
2. The backward approach avoids sensitivities and gives the gradient in two integrations.
3. Combined with quasi newton methods, they provide extremely fast convergence
4. Both methods could be combined with more complex solvers (e.g. FEM or FVM)

On the other hand, you should consider that…

1. The forward problem is unfeasible when the number of parameters is too large
2. The backward time integration is very fragile.
3. Computing Jacobians for large scale numerical problems is difficult !
4. Inverse modeling is ill posed: strong sensitivity to the initial guess and many local minima.

   (try different initial guesses!)

Thank You !