Parameter Estimation for Differential Equations with Julia

Steffen Plunder

November 25, 2024

Statistics and optimisation theory provide rich theories to find optimal model parameters for given datapoints. These methodologies adapt well to ODE and PDE models. Depending on the number of unknown parameters and available data, different strategies are optimal. We will discuss optimisation based parameter estimation, forward and adjoint sensitivity analysis and automatic differentiation, which are all techniques to obtain accurate gradients.

The lecture is complemented by code examples in the programming language Julia.

Recommended literature

Ideal parameter estimation techniques depend critically on the application setting. Here are some references: A good introduction, especially for ODE models in mathematical biology, is [2, Chapter 3] and [6]. For PDE models, adjoint equations are of key importance, where one might consult [5, 4].

These notes largely follow [4] and the lecture given 2018 at the Technical Univeristy Kaiserslautern (now called RPTU Kaiserslautern) by Prof. Rene Pinnau.

Notation

Before we start, a quick note about notations. Let X, Y be real Banach spaces. For a function $f : X \times Y \to \mathbb{R} : (x, y) \mapsto f(x, y)$ we will write $\partial_x f$ to denote the derivative of f with respect to the first argument x here, and $\partial_y f$ for the second argument. To denote the total derivative of an expression, we will write $\frac{\mathrm{d}}{\mathrm{d}x} f(x, y(x))$.

The derivative evaluated at a point is a linear map $\partial_x f(x, y) : X \to \mathbb{R}$ and for a direction $v \in X$ we write $\partial_x f(x, y)[v]$ to denote the directional derivative in

direction ν.

A function $f : X \to Y$ is Gâteaux differentiable at a point $x \in X$, if the limit $\lim_{h\to 0} \frac{f(x+h\nu)-f(x)}{h}$ exists for all $\nu \in X$.
A function is Fréchet differentiable at $x \in X$, in which case there existis a bounded, linear operator $A(x) : X \to Y$ such that $\lim_{\|\nu\|\to 0} \frac{\|f(x+\nu)-f(x)-A(x)\nu\|}{\|\nu\|} = 0$. We denote the Fréchet derivative as $\partial_x f(x) := A$. A function is continously differentiable, if $x \mapsto A(x)$ is continous with respect to the operator norm $\|\partial_x f(x)\| := \sup_{\nu\in X} \frac{\|\partial_x f(x)[\nu]\|_Y}{\|\nu\|_X}$. For more notes on nonlinear analysis in infinite dimensional spaces, we refer to [1] or the first chapter of [4].

1 Introduction

Most of the theory can be developed similar for ODE and PDE models, which we will do in the next chapter. To give a brief overview, we first consider the most typical ODE setting. Given an ODE model such as

$$\begin{aligned} \partial_t u(t) &= f(u(t), p) \quad \text{for } t \in (0, T), \\ u(0) &= u_0, \end{aligned}$$

where $T > 0$ denotes the terminal time, $u_0 \in \mathbb{R}^{n_x}$ the inital data and $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ is a parameterized vector field such that the ODE is well-posed for any parameter $p \in \mathbb{R}^{n_p}$. We denote the solution as $u(t, p)$ to make the dependency on the parameters explicit.

For given times $0 < t_1 \leq \cdots \leq t_m \leq T$ and data points $u_1, \ldots, u_m \in \mathbb{R}^{n_x}$ we seek to find parameters p which minimize the distance of the ODE model to the data. For this task, we might introduce the reduced (discrete) cost function as

$$j(p) := \sum_{k=0}^{m} \|u(t_k, p) - u_k\|^2 + \gamma \|p\|^2$$

where γ is a regularisation parameter which should be small but might help numerical minimisation attempts.

The parameter estimation problem can then be written as an optimisation

$$p^* \in \operatorname*{arg\,min}_{p\in\mathbb{R}^{n_p}} j(p),$$

turning the parameter optimisation into a regular minimisation problem.

One aspect which is specific for parameter optimisation is that the evaluation of $j(p)$ requires the solution of a differential equations, which might be computationally expensive and numerically inaccurate. Hence, we need fitting optimisation methods and ideally a robust way to approximate the gradients $\partial_p j(p)$.

Therefore, the main topics of these lecture notes are:

- choices of the cost function,
- strategies to compute the gradients $\partial_p j(p)$ effectively,
- a quick overview of (global) optimisation methods.

One word of caution: The performance of parameter estimation depends often on the quality of the data, the choice of the mathematical model and the cost function. Rather than a magic tool to find all unknown parameters for a model, one might need adapt the techniques to the concrete model.

Exercise 1. *Find a differential equation for $s(t) := \partial_p u(t,p)$, for this task, calculate $\partial_t s(t)$ in terms of $u(t,p)$. How can $s(t)$ be used to compute $\partial_t j(p)$?*

2 Parameter Estimation as a Constrained Optimisation Problem

2.1 From differential equations to optimisation problems

In an abstract setting, we will consider differential equations of the form

$$\begin{cases} \partial_t u = f(u,p) & \text{for } t \in (0,T), \\ u(0) = u_0. \end{cases} \tag{1}$$

where $u \in U$ is the solution, $u_0 \in X$ the initial condition, $T > 0$ the terminal time and $p \in P$ are free parameters. The spaces U, X, P are some Banach spaces which we specify later. We might denote the solution as $u(t,p)$ (or $u(x,t,p)$ in the PDE case) and we assume that the equation is always well-posed (see assumptions below), such that we can define the solution map

$$S : P \mapsto U : p \mapsto u(\cdot, p).$$

Next, we consider an abstract cost function

$$J : U \times P \to \mathbb{R}$$

and its reduced counterpart

$$j(p) := J(S(p), p).$$

Our aim is to find

$$p^* \in \operatorname*{arg\,min}_{p \in P} j(p). \tag{2}$$

For our task, it is convenient to write the differential equation in its implicit form. Let us introduce the state equation as

$$F : U \times P \mapsto V$$

for some suitable Banach space V such that it defines the differential equation implicitly, i.e.,

$$F(u, p) = 0 \quad \Leftrightarrow \quad \text{eq. (1).}$$

We can then rewrite the unconstrained optimisation problem in eq. (2) as

$$\min_{p \in P} J(u, p) \tag{3}$$

$$\text{such that } F(u, p) = 0. \tag{4}$$

Assumptions While these notes do not put special emphasize on mathematical rigour. Here are some assumptions which ensure that our constrained optimisation problem would be well-posed. We refer to [4] for the technical details.

- U, X, P are reflexive Banach spaces, V is a Banach space.
- $J : U \times P \to V$ is continuous.
- For each $p \in P$ there is one unique solution $u \in U$ such that $F(u, p) = 0$. Moveover, the state equation $F(u, p) = 0$ has a bounded solution map $S : P \to U$.
- $F : U \times P \mapsto V$ is continuous under weak convergence.
- J is sequentially weakly lower semicontinuous.
- J, F are continuously (Fréchet)-differentiable.
- $\partial_u F(S(p), p)) : U \to V : \eta_u \mapsto \partial_u F(S(p), p)[\eta_u]$ is continuously invertible.

Under these assumptions, the constrained minimisation problem in eqs. (3) and (4) has a unique solution, see [4, Thm. 1.45].

Let us discuss two examples to make the abstract formalism more concrete.

2.1.1 ODE case

In the ODE setting, we can pick

$$\begin{aligned} U &= H^1([0, T], \mathbb{R}^{n_x}), \\ P &= \mathbb{R}^{n_p}, \\ V &= L^2([0, T], \mathbb{R}^{n_x}) \times \mathbb{R}^{n_x}. \end{aligned}$$

and define the state equation including the initial condition as

$$F(u, p) \coloneqq \begin{pmatrix} \partial_t u - f(u, p) \\ u(0) - u_0 \end{pmatrix}$$

where $\partial_t u - f(u, p)$ is a shorthand notation for the function $t \mapsto \partial_t u(t) - f(u(t), p)$ and H^1 is the Sobolev space of functions with one time derivative in L^2.

2.1.2 PDE case

In the PDE setting, the used spaces depend highly on the setting. We will consider here . For the model equation

$$\begin{cases} \partial_t u(x,t) = \Delta_x u(x,t) + f(x,p) & \text{in } \Omega \times (0,T), \\ u(\cdot,0) = u_0 & \text{at } t = 0, \\ \partial_x u(x,t)[n] = 0 & \text{on } \partial\Omega \times (0,T), \end{cases}$$

for some compact, non-empty domain $\Omega \subset \mathbb{R}^{n_x}$ with smooth boundary with normal n. The construction of suitable function spaces is out-of-scope here, but we just outline that one can pick

$$U = W([0,T], L^2(\Omega), H^1(\Omega)), \quad V = L^2([0,T], H^{-1}(\Omega)) \times L^2(\Omega)$$

where $H^1(\Omega)$ denotes the Sobolev space over Ω with one derivative in L^2, H^{-1} it's dual space and W denotes a space of once weakly differentiable functions such as $t \mapsto u(\cdot,t) \in H^1(\Omega)$ such that $t \mapsto u(\cdot,t)$ and $t \mapsto \partial_x u(\cdot,t)$ are Bochner integrable with finite L^2 norm over time. See [4, Section 1.3] for details.

Similar to the ODE setting, we can then define the PDE implicitly with the map

$$F(u,p) = \begin{pmatrix} t \mapsto \partial_t u(\cdot,t) - \Delta_x u(\cdot,t) - f(\cdot,t) \in H^{-1}(\Omega) \\ u(\cdot,0) - u_0 \in L^2(\Omega) \end{pmatrix}$$

where the upper line would correspond to the weak-form of the heat equation.[1]

Julia examples

In the following, we use Julia to show (simplified) implementations of the presented theoretical methods. Please note that we might skip technical details which would improve the implementation, and rather focus on mathematical clearity.

We start with an simple example of solving the ODE $\dot{x} = -\alpha x$, just to demonstrate the interface.

```
using OrdinaryDiffEq, Plots

# Define right-hand side of ODE as a in-place function
```

[1] The upper line is using the dual notation, but $F = 0$ will imply that for any test function $\eta_u \in H^1(\Omega)$ we have $\int_0^T (\partial_t u - f)\eta_u + \langle \nabla_x u, \nabla_x \eta_u \rangle_{\mathbb{R}^{n_x}} \, \mathrm{d}t = 0$.

```julia
function ode(du, u, p, t)
    du[1] = p[1] * u[1]
end

tspan = (0.0, 10.0) # time interval
p = [-1.0]          # parameters
u0 = [1.0]          # initial value

odeprob = ODEProblem(ode, u0, tspan, p)
odesol = solve(odeprob, Tsit5())

plot(odesol, labels = ["x(t)"], title = "First-order ODE")
```

Listing 1: Solving ODE models with Julia.

Exercise 2 (Lotka-Volterra Predator-Prey model). *Solve and visualize the differential equation*

$$\begin{aligned} \partial_t x &= \alpha x - \beta xy, \\ \partial_t y &= -\gamma y + \delta xy \end{aligned}$$

with initial data $x(0) = 1, y(0) = 1$ *for* $t \in [0, 10]$ *and* $\alpha = \frac{3}{2}, \beta = 1, \gamma = 3, \delta = 1$.

Exercise 3. *Write your own ODE solver into the function 'solver(fun, u0, tspan, dt, p)' such that the following code works. As a numerical method, you can choose the explicit Euler method*

$$u(t + \Delta t) \coloneqq u(t) + \Delta t f(u(t), p, t).$$

```julia
using Plots

function solver(fnc, u0, tspan, dt, p)
    # write your code here
end

# test code which should run
function ode(du, u, p, t)
    du[1] = p[1] * u[2]
    du[2] = -p[2] * u[1]
end

tspan = (0.0, 10.0) # time interval
dt = 0.01
p = [1.0, 1.0]          # parameters
u0 = [1.0]          # initial value

odesol = solver(ode, u0, tspan, dt, p)

plot(odesol.t, odesol.u[1,:])
```

```
# should show the first component of the solution

@time solver(ode, u0, tspan, dt, p)
@profview solver(ode, u0, tspan, dt, p)
# measure the performance of your solver as well
```

Listing 2: Writing your own ODE soler.

As an extra challenge: Can you write the solver such that only very allocations occur (let's say less than 10 allocations)?

Exercise 4. *Implement the 2D Laplace operator for Neumann boundary conditions in Julia.*

2.2 Cost functionals

We now outline the most typical choices for cost functionals.

2.2.1 Continuous cost function

Given a target trajectory $y(t)$, we define

$$J^{\text{cont}}(u,p) = \frac{1}{2}\|u-y\|_U^2 = \frac{1}{2}\int_0^T \|u(t,p)-y(t)\|_{\mathbb{R}^d}^2 \,\mathrm{d}t$$

2.2.2 Discrete cost function

The most common case is that at given time points $0 \le t_1 \le t_2 \dots t_M \le T$ we have experimental data $u_1, \dots, u_M \in U$ and weights $w_1, \dots, w_M \ge 0$. Then, a least square loss function reads

$$J^{\text{discr}}(u,p) = \sum_{k=1}^{M} w_k (u(t_k,p)-u_k)^2.$$

2.2.3 Regularization

One issue with L^2-type cost functions is that there is not guarantee that the resulting cost function $j(p)$ has a unique global minimizer. Especially in applications, it might even be unclear how close a model can fit the provided data in the first place. The simplest workaround is a Tikhonov type regularisation, by simply adding a quadratic term such as

$$J^{L_2-\text{reg}}(u,p) = J(u,p) + \gamma\|p\|_2^2.$$

For large γ, the regularized cost function might be convex, which allows numerical algorithms to converge easily, but at the cost of potentially missing out the true minimizers of $j(p)$.

Julia examples

One of the advantages of Julia is, that we can define the cost function generically, without compromising performance.

Exercise 5. *Define your own favourite cost function for the Lotka-Volterra Predetor-Prey model.*

Exercise 6. *Use the package 'DiffEqParamEstim' to define a discrete L^2 loss function for the Lotka-Volterra model, similar to the example below:*

```
using DiffEqParamEstim

ts = [0.0, 5.0, 7.0, 9.0]
xs = @. exp( -0.2 * ts )

cost_function = build_loss_objective(odeprob, Tsit5(),
                    L2Loss(ts, xs),
                    Optimization.AutoForwardDiff(),
                    maxiters = 10000)

# calling the function:
cost_function([-0.2])
cost_function([0.2])
```

Listing 3: Writing your own ODE soler.

3 Computing gradients

The schemes discussed in the previous section, all require the gradient of the cost function $\partial_p j(p)$ or the directional derivatives $\partial_p j(p)[v]$.

There are two main strategies to obtains these derivatives. The most cannonical approach is the forward-sensitivity analysis which is based on the chain rule. The backward counterpart is called adjoint-sensitivity analysis which is favourable for large-scale problems [3].

3.1 Forward-sensitivity analysis

If we directly differentiate $j(p) = J(S(p), p)$, we obtain

$$\partial_p j(p) = \partial_u J(S(p), p)\, \partial_p S(p) + \partial_p J(S(p), p). \tag{5}$$

The derivative of the solution with respect to the parameters $\partial_p S(p)$ are called the *sensitivities*. We denote the sensitivities as $s = \partial_p S(p)$. Since $S(p)$ is defined

as the unique solution $u = S(p)$ such that $F(u, p) = 0$, we can use implicit differentiation to obtain an equation for the sensitivities s via

$$\partial_u F(S(p), p)[\partial_p S(p)] = \partial_p F(S(p), p). \tag{6}$$

The above equation is overly abstract, but applies to a wide range of problems.

3.1.1 ODE case

Let us consider the ODE example

$$F(u, p) = \begin{pmatrix} \partial_u - f(u, p) \\ u(0) - u_0 \end{pmatrix}$$

where $V = V^* = L^2([0, T], \mathbb{R}^{n_x}) \times \mathbb{R}^{n_x}$.

The linearization of this map around a solution u with $F(u, p) = 0$ leads to

$$\partial_u F(u, p)[\eta_u] = \begin{pmatrix} \partial_t \eta_u - \partial_u f(u, p)\eta_u \\ \eta_u(0) \end{pmatrix}$$

for $\eta_u \in U$ and

$$\partial_p F(u, p) = \begin{pmatrix} \partial_p f(u, p) \\ 0 \end{pmatrix}.$$

Hence, the sentitivities simply solve the *parameter variational equation*

$$\begin{aligned} \partial_t s(t) &= \partial_u f(u(t), p)s(t) + \partial_p f(u(t), p) \\ s(0) &= 0. \end{aligned}$$

We can compute $u(t)$ and $s(t)$ at the same time and then obtain the gradient of the solution by using the chain rule in eq. (5). For the discrete cost functional J^{discr}, this yields

$$\partial_p j(p) = \sum_{k=1}^{M} w_k(u(t_k, p) - u_k)s(t_k)$$

and for the continous cost functional one obtains

$$\partial_p j(p) = \int_0^T (u(t, p) - y(t))s(t) \, \mathrm{d}t$$

which can be approximated via quadrature rules.

3.1.2 PDE case

Forward-sensitivity analysis is very similar in both the ODE and the PDE setting. Considering the heat equation as previously, we obtain the parameter variational equation

$$\partial_t s(x,t) = -\Delta_x s(x,t) + \partial_p f(x,p).$$

The major difference between the ODE case and the PDE case will show up later for adjoint sensitivities.

3.1.3 Finite differnces

The most straightforward approach are finite differences. We can approximate the derivative in direction $v \in Y$ as

$$s(t)[v] = \partial_p u(t,p)[v] \approx \frac{u(t,p) - u(t,p-hv)}{h}$$

for a suitable stepsize $h > 0$. To approximate $\partial_p u(t,p)$ via finite differences, we therefore need n_Y additional solutions.

An advantage of this approach is the relatively easy implementation and especially for $n_Y < 200$ one might also get a very performant gradient evaluation.

However, finding the ideal parameter step-size h_p can be a challenge, as too large or too small values lead to numerical approximation or round-off errors.

3.1.4 Automatic differentiation

A convenient way to get derivatives without the need to explicitly setup the parameter variational equation, is by using automatic differentiation.

The key idea is that computer programs are essentially a composition of functions for which the derivative is known.

Blackboard...

3.2 Adjoint-senstivity analysis

As mentioned before, for large-scale problems the application of the chain rule is suboptimal, as it requires us to compute the sensitivities whereas the the gradient $\partial_p j$ in principle does not depend on the dimension of the differential equation.

We recall that our original aim is to solve $\min_p j(p)$ which we could alternatively formulate as a constrained optimisation problem

$$\begin{aligned} &\min_{p \in P} J(u,p) \\ &\text{such that } F(u,p) = 0. \end{aligned}$$

Instead of using the solution map, we could instead introduce a Lagranian function with a Lagrangian multiplier $\lambda \in V^*$ for the constraints as follows

$$\mathcal{L}(u,p,\lambda) = J(u,p) + \langle \lambda, F(u,p) \rangle_{V^*,V}$$

which leads to equivalent[2] minimisation problem

$$\min_{p \in P, \lambda \in V^*} \mathcal{L}(u,p,\lambda).$$

The first-order optimality conditions for this problem read

$$\begin{aligned} 0 &= \partial_u J(u,p)[\eta_u] + \langle \lambda, \partial_u F(u,p)[\eta_u] \rangle_{V^*,V} && \forall \eta_u \in U, \\ 0 &= \partial_p J(u,p)[\eta_p] + \langle \lambda, \partial_p F(u,p)[\eta_p] \rangle_{V^*,V} && \forall \eta_p \in P, \\ 0 &= \langle \eta_\lambda, F(u,p) \rangle_{V,V^*} && \forall \eta_\lambda \in V. \end{aligned}$$

Let's break down these three equations one-by-one.

Let's us fix a solution u, p, λ of the above system [ref TODO]. Since V is a Hilbert space and V^* it's dual space, the last equation simply states that the differential equation is satisfied

$$F(u,p) = 0.$$

Next, we consider the first equation, which we can rewrite by using the adjoint of the linearized state equation

$$\begin{aligned} 0 &= \partial_u J(u,p)[\eta_u] + \langle v, \partial_u F(u,p)[\eta_u] \rangle_{V^*,V} \\ &= \partial_u J(u,p)[\eta_u] + \langle \partial_u F(u,p)^* \lambda, \eta_u \rangle_{U^*,U} \\ &= \langle \partial_u J(u,p) + \partial_u F(u,p)^* \lambda, \eta_u \rangle_{U^*,U}, \end{aligned}$$

which has to hold for all $\eta_u \in U$. In other words, the Lagrangian multiplier solves the adjoint state equation

$$\partial_u F(u,p)^* \lambda = -\partial_u J(u,p).$$

[2]Notice that whenever $F(u,p) \neq 0$ the minimum would be $-\inf$ which is not a proper minimum.

Inserting this into the second equation, we obtain

$$\begin{aligned}
0 &= \partial_p J(u,p)[\eta_p] + \langle \lambda, \partial_p F(u,p)[\eta_p] \rangle_{V^*,V} \\
&= \partial_p J(u,p)[\eta_p] - \langle \partial_u F(u,p)^{-*} \partial_u J(u,p), \partial_p F(u,p)[\eta_p] \rangle_{V^*,V} \\
&= \partial_p J(u,p)[\eta_p] - \partial_u J(u,p) \partial_u F(u,p)^{-1} \partial_p F(u,p)[\eta_p] \\
&= \partial_p j(p)[\eta_p].
\end{aligned}$$

Hence, we can use the adjoint state λ to compute the cost gradient as

$$\partial_p j(p) = \partial_p J(u,p) - \partial_p F(u,p)^* \lambda.$$

3.2.1 ODE example

Returning to our running example, we again consider the setting

$$\begin{aligned}
F(u,p) &= \begin{pmatrix} \partial_t u - f(u,p) \\ u(0) - u_0 \end{pmatrix}, \\
\partial_u F(u,p)[\eta_u] &= \begin{pmatrix} \partial_t \eta_u - \partial_u f(u,p)\eta_u \\ \eta_u(0) \end{pmatrix}, \\
\partial_p F(u,p) &= \begin{pmatrix} \partial_p f(u,p) \\ 0 \end{pmatrix}
\end{aligned}$$

where $V = V^* = L^2([0,T], \mathbb{R}^{n_x}) \times \mathbb{R}^{n_x}$. However, this time we derive the adjoint equations. For a given $\eta_u \in U$ and $\lambda = (\lambda_u, \lambda_0) \in V^*$, we compute

$$\begin{aligned}
\partial_u F(u,p)^*[\lambda][\eta_u] &= \langle \partial_u F(u,p)^* \lambda, \eta_u \rangle \\
&= \langle \lambda, \partial_u F(u,p) \eta_u \rangle \\
&= \int_0^T \lambda_u(t) \cdot \Big(\partial_t \eta_u(t) - \partial_u f(u(t),p) \eta_u(t) \Big) \,\mathrm{d}t + \lambda_0 \eta_u(0) \\
&= -\int_0^T \Big(\partial_t \lambda_u(t) - \partial_u f(u(t),p) \lambda_u \Big) \cdot \eta_u(t) \,\mathrm{d}t \\
&\qquad + \lambda(T)\eta_u(T) - \lambda_u(0)\eta_u(0) + \lambda_0 \eta_u(0).
\end{aligned}$$

where we applied the chain rule in the last step. At this stage, we can already see that we are going to obtain a differential equation for the adjoint states λ_u with a terminal condition for $\lambda_u(T)$ and a free boundary at $\lambda_u(0) = \lambda_0$ which can take any value. (Since λ_0 is not important in the further analysis, we will in the following just write $\lambda = \lambda_u$.)

Unlike the forward sensitivies, we cannot write down the differential equations for the adoint states without reference to the cost functional.

Let us therefore consider the example of a continuous cost function

$$J^{\text{cont}}(u,p) = \frac{1}{2}\int_0^T \|u(t) - y(t)\|^2 \,\mathrm{d}t$$

which then leads to the following adjoint equations

$$\begin{aligned} \partial_t \lambda &= \partial_u f(u,p)\lambda - (u(t) - y(t)) \quad \text{for } t \in (0,T), \\ \lambda(T) &= 0 \end{aligned}$$

which form a terminal problem. This system can be understood as backtracking the changes of the cost functional rather than forwarding the derivations of the solutions.

Once we computed the adjoint state, we obtain the cost gradient as

$$\partial_p j(p) = -\int_0^T \partial_p f(u(t),p)\lambda(t) \,\mathrm{d}t.$$

3.2.2 Computational efficiency between forward-sensitivities and adjoint-sensitivities

To recapture, we now have two strategies to obtain the cost gradients $\partial_p j(p)$. Let us compare their computational demand for the ODE setting with $\dim(u(t)) \in \mathbb{R}^{n_x}$ and $\dim(P) = n_p$. In both cases, one first computes the solution $u = S(p)$ for the current parameter value p.

- **Forward-sensitivities.**
  - Compute sensitivities $s(t) \in \mathbb{R}^{n_p \times n_x}$ by solving the parameter variational equation.
  - Apply the chain rule, i.e. $\partial_p j = \partial_p J + \partial_u J\, s$.
  - *Runtime* $\approx \mathcal{O}(n_p^2 \cdot n_x)$.
- **Adjoint-sensitivities.**
  - Compute the adjoint states $\lambda(t) \in \mathbb{R}^{n_x}$.
  - Use the adjoint gradient equation $\partial_p j = \partial_p J - \partial_p F^* \lambda$.
  - *Runtime* $\approx \mathcal{O}(n_p \cdot n_x)$.

In short, by avoiding the chain rule we can speed-up the gradient computation by a linear factor in $\dim(P)$. If one only computes the directional derivatives $\partial_p j(p)[v]$, then the runtimes change accordingly from $\mathcal{O}(n_p \cdot n_x)$ to $\mathcal{O}(n_x)$.[3]

[3]However, we have to note that this anaysis is theoretical in the sense that solving a larger system with adaptive time-stepping methods might lead to hard-to-compute nonlinear effects on the runtime.

Exercise 7. *Derive the adjoint equation for the case that the parameters describe the initial condition, i.e.*

$$F(u,p) = \begin{pmatrix} \partial_t u - f(u) \\ u(0) = p \end{pmatrix}.$$

Does the adjoint state depend on p? Explain why it depends on p or why not.

Exercise 8. *Consider the usual ODE setup, with with the discrete cost functional J^{discr}. Compute the adjoint equations. The adjoint state $\lambda(t)$ will not be continuous, explain in which sense one can obtain well-posedness and uniqueness for the adjoint state.*

3.2.3 PDE case

To be continued...

Julia implementation for adjoint sensitivity analysis

Fortunaltly, the heavy lifting for adjoint sensitivity analysis can be automated, which is part of the Julia package `SciMLSensitivities`. In the example below, we need to specify explicitly the gradients at the discrete times $(t_k)_k$, which are for the discrete L^2 loss given by

$$\partial_p J^{\text{distr}}(u,p)(t_k) = u(t_k,p) - u_k.$$

The following Julia code allows us to compute the gradient of the cost function using adjoint-sensitivities.

```
using SciMLSensitivities

ts = LinRange(0, 1.0, 10)
data = rand(10)  # some random fitting data here...

# we need to define the discrete gradients
dg(out,u,p,t,i) = (out.=data[i].+u)

res = adjoint_sensitivities(sol,Tsit5();
                            t=ts,
                            dg_discrete=dg)
```

Listing 4: Writing your own ODE soler.

4 Optimisation methods

Once the cost function is selected, we are "only" left with the task of finding a minimizer of a function $j : P \to \mathbb{R}$.

In general, we cannot assume that the function j is convex. We will focus on the following on the methods which are used, rather than their underlying mathematical theory.

4.1 Local minimisation methods

Starting from an initial parameter guess p_0 one could seek a minimum of g by applying

$$p_{j+1} = p_j - \alpha_j \nabla j(p_j)$$

where α_j is either a constant step-size or the solution of a linear search problem.

Since the first order optimality condition for **??** reads

$$\partial j(p) = 0,$$

we might stop the iterations once the stopping condition

$$\| \partial j(p_j) \|_\infty \leq \texttt{abstol} + \texttt{reltol} |j(p_j)|$$

is satisfied.[4]

To be completed...

Julia examples

Exercise 9 (Finally, Parameter Estimation)**.** *Continuing the running example of the Lotka-Volterra model. Combine all the techniques to find optimal paramters for the datasets* `data_⋆.csv`*. In each case, the amount of noise increases. Try also some global optimisation solvers from the package* `OptimizationBBO`*.*

5 Practical considerations

Modelling and parameter estimation are in practice often mixed:

- Wrong models, might make a statisfying parameter estimation impossible,
- the model itself might imply conditions on the parameters (non-negativity or bounds).

This section collects some theoretically "boring" but in practice relevant tips.

[4] Notice that here the sup-norm is chosen to ensure that a system with more parameters does not converge faster just because more components of the cost functional gradient as zero.

5.1 Parameter bounds

Global optimisation methods usually require parameter bounds to enclose the search area. For a parameter which has to remain positive, one can apply the transformation

$$p' \mapsto e^p$$

which also has the advantage that optimisation solvers might take automatically smaller steps for small parameter values and larger steps for large parameters values.

Notice that all techniques in the previous sections do not suffer from using a nonlinear paramter transformation, as most likely the automatic differentiation will compute all required gradients for us.

5.2 Choice of ODE solvers

Numerical stiffness can dramatically change depending on the model parameters, which simple equations like $\dot{x} = -\gamma x$ might require different solvers for different values of γ, for $\gamma > 0$ the equation becomes stiff and implicit solvers are required, whereas for too large γ the solutions might grow too quickly.

One way around this issue is to use automatic stiffness detection, which adaptively switches between a fast explicit solvers and a slower but more robust stiff solver. In Julia we can use for example the algorithm `AutoTsit5(Rosenbrock23())`.

5.3 Application specific loss functions

In some cases, the application has specific features which are not well captured by L^2 loss functions. The following challenge consideres a model for electric signals in neurons. The solutions show steep peeks. The challenge is to find a loss function which gives enough weight to the peaks such that optimisation routines can capture the dynamics well.

Exercise 10 (FitzHugh–Nagumo neuron spike model)**.** *We consider the model*

$$\begin{aligned}\frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{1}{\varepsilon}(f(v,\alpha) - wI_{\text{app}}),\\ \frac{\mathrm{d}w}{\mathrm{d}t} &= -v + \gamma w\end{aligned}$$

with $v(0) = 1, w(0) = 0$ *and* $t \in [0, 10]$*. The parameters are* $\varepsilon = 0.01$*,* $\alpha = 0.2, \gamma = 0.3$ *and* $I_{\text{app}} = 0.8$ *and*

$$f(v,\alpha) = v(1-v)(v-\alpha).$$

Fit this model to the (real) dataset 'spikes.csv'. Notice that since the data is experimental data, the model will not perfectly fit and there are no true parameters which perfectly fit the data! Good luck.

References

[1] P. G. Ciarlet. *Linear and Nonlinear Functional Analysis with Applications*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2013. 846 pp. DOI: 10.1137/1.9781611972597. URL: https://epubs.siam.org/doi/book/10.1137/1.9781611972597 (visited on 11/22/2024).

[2] P. Deuflhard and S. Röblitz. *A Guide to Numerical Modelling in Systems Biology*. Vol. 12. Texts in Computational Science and Engineering. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-20059-0. URL: https://link.springer.com/10.1007/978-3-319-20059-0 (visited on 02/04/2024).

[3] F. Fröhlich, B. Kaltenbacher, F. J. Theis, and J. Hasenauer. "Scalable Parameter Estimation for Genome-Scale Biochemical Reaction Networks". *PLOS Computational Biology* 13.1 (2017). Ed. by J. Stelling, e1005331. DOI: 10.1371/journal.pcbi.1005331.

[4] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, eds. *Optimization with PDE Constraints*. Mathematical Modelling: Theory and Applications 23. New York: Springer, 2009. 270 pp.

[5] A. Manzoni, A. Quarteroni, and S. Salsa. *Optimal Control of Partial Differential Equations: Analysis, Approximation, and Applications*. Vol. 207. Applied Mathematical Sciences. Cham: Springer International Publishing, 2021. DOI: 10.1007/978-3-030-77226-0. URL: https://link.springer.com/10.1007/978-3-030-77226-0 (visited on 02/02/2024).

[6] A. Raue, M. Schilling, J. Bachmann, A. Matteson, M. Schelke, D. Kaschek, S. Hug, C. Kreutz, B. D. Harms, F. J. Theis, U. Klingmüller, and J. Timmer. "Lessons Learned from Quantitative Dynamical Modeling in Systems Biology". *PLOS ONE* 8.9 (2013), e74335. DOI: 10.1371/journal.pone.0074335.