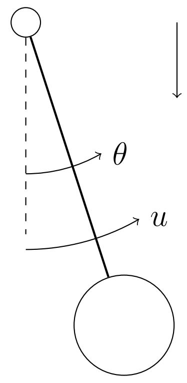
Numerical Optimization - Exercise 6

Benjamin Biemond, Lorenzo Stella

1 Nonlinear pendulum

Consider the motion of the following pendulum system.



Here, the angular position is θ, the actuator torque applied is u and $g = 9.81$ is the gravitational force exerted on the pendulum with unit inertia. In addition, a damping torque $-c\dot{\theta}$ is present. Introducing a state vector

$$x = \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix},$$

a discrete-time mapping, using a crude first-order integration scheme with time-step h, represents the system's motion as:

$$x_{k+1} = f(x_k, u_k) := x_k + h \begin{pmatrix} x_{k,2} \\ -g\sin(x_{k,1}) - cx_{k,2} + u_k \end{pmatrix}, \tag{1}$$

with x^0 a known initial condition. To design a optimal control input sequences $u^0, u^1, \dots u^{N-1}$ to obtain fast convergence of this system to the origin for the next N time instances, one can solve

$$\text{minimize } J^N(\bar{x}, \bar{u}) = \sum_{k=1}^{N} x_k^T Q x_k + u_{k-1}^T R u_{k-1}, \tag{2}$$

$$\text{subject to } x_{k+1} = f(x_k, u_k), \quad k = 0, \dots, N-1, \tag{3}$$

where we introduced vectors

$$\bar{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}, \quad \bar{u} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix}.$$

The positive definite matrix $Q \in \mathbb{R}^{2\times 2}$ puts a penalty on the states deviating from the origin, and $R > 0$ measures the cost of the control action. The derivatives of J^N must be computed in order to apply optimization algorithms to the solution of (2).

Tasks:

1. Given an initial condition x^0, express the cost function J^N, for $N = 2$, as a function of u^0 and u^1 only, and compute the derivative analytically
2. We need $\nabla_u J^N$ for $N = 50$, and for this purpose, will use algorithmic differentiation. Focusing on $N = 3$ first, construct the list of elementary manipulations φ_i as introduced in Chapter 9 of the lecture notes, and the matrix C as given on page 69.
3. Give the partial derivatives of these elementary manipulations (for $N = 3$).
4. Function J^N is implemented in `pendulum.m` for

$$Q = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}, \quad R = 1.$$

   It returns the value of J^N and the sequence $x_1, \dots, x_n$ associated with the given inputs u. Implement an analogous function, in `pendulum_forward.m`, that also returns $\Delta u^T \nabla_u J^N$, the directional derivative along direction Δu. Do this via forward algorithmic differentiation.
   Hint: implement Algorithm 5 in the lecture notes. Also refer to Example 9.2.
5. To obtain the full gradient $\nabla_u J^N$, complete function `pendulum_gradient.m` by invoking forward AD N times, one for each coordinate direction.
6. Solve the optimisation problem using the gradient descent method, by completing the script `ex6_pendulum.m` and plot the resulting trajectory X and control action u. You can use fixed stepsize, or use `line_search.m` from sessions 3.

2 Nonlinear least squares

We want to find a curve that fits some experimental data, $(t_i, y_i) \in \mathbb{R}^2$ for $i = 1, \dots, m$, and from the data and our knowledge of the application we know that the signal has exponential and oscillatory behaviour of some type. We then look for functions of the form

$$\phi(t, x) = x_1 + x_2 e^{-(x_3 - t)^2 / x_4} + x_5 \cos(x_6 t),$$

where $x_1, \dots, x_6 \in \mathbb{R}$ are the parameters of the model. To fit the data we minimize the sum of the squared residuals as follows:

$$\text{minimize}_x \sum_{i=1}^{m} (y_i - \phi(t_i, x))^2. \tag{4}$$

For each data point (t, y) we can compute each square residual by computing the following auxiliary variables:

$$\begin{aligned}
x_7 &= (x_3 - t)^2 \\
x_8 &= x_7 / x_4 \\
x_9 &= e^{-x_8} \\
x_{10} &= x_2 x_9 \\
x_{11} &= \cos(x_6 t) \\
x_{12} &= x_5 x_{11} \\
x_{13} &= x_{10} + x_{12} \\
x_{14} &= x_1 + x_{13} \\
x_{15} &= (y - x_{14})^2
\end{aligned}$$

so that $x_{15} = (y - \phi(t, x))^2$.

Tasks:

1. Look into `nls.m`, where the objective function of problem (4) is computed by means of auxiliary variables $x_7, \dots, x_{15}$. Fill in `nls_backward.m` appropriately, so to return also the *gradient* of the objective function computed by means of the backward AD algorithm.
   Hint: refer to Section 9.3 of the lecture notes, in particular Algorithm 6 and Example 9.3.

2. The script called `ex6_nls.m` loads a data set and fits it by solving problem (4). The problem is solved using MATLAB's `fminunc`, first using the function `nls.m` and computing gradients with finite differences, and then using `nls_backward.m`. Run the script, check whether the two results agree, and look into the output of `fminunc` to verify the number of function calls in the two cases.

3. Write your own routine that, using function `nls_backward.m`, solves problem (4) using BFGS.
   Hint: you can take inspiration from `minimize_bfgs.m` and `line_search.m`, from session 3, on how to structure the functions.