Hands-on CasADi course on optimal control, Yacoda

Exercise: Fitting

In this exercise, we will study the dynamics of a simple predator-prey system.
We consider a discrete-time system for the number or rabbits ($x \in \mathbb{R}$) and foxes ($y \in \mathbb{R}$) in a closed ecosystem. Note that our rabbits and foxes are real numbers..
The population dynamics is given modeled by:

$$x_{k+1} = (a x_k - \alpha x_k y_k)/(1 + \gamma x_k),$$
$$y_{k+1} = (b y_k + \beta x_k y_k)/(1 + \delta y_k),$$

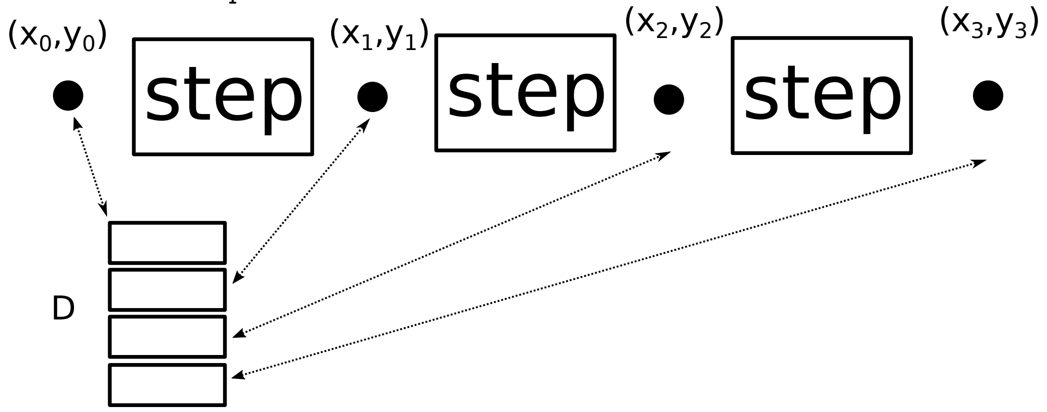
where k indicates the amount of years passed after some specific date. Let's summarize these unknown parameters as $\begin{bmatrix} a & b & \alpha & \beta & \gamma & \delta \end{bmatrix}^T$ as $p \in \mathbb{R}^6$.
We are provided with a raw estimate of p:

$$\hat{p} = \begin{bmatrix} 1.43 & 0.99 & 0.22 & 0.022 & 0.022 & 0.011 \end{bmatrix}^T$$

1 Alternative formulations

Tasks:

1. Starting from $x_0 = 10, y_0 = 0.1$, simulate the estimated population dynamics 10 steps forward in time, using numerical $\hat{p}$. Verify that you obtain `[13.1929 1.2829]`. Don't use any CasADi functionality; just a for-loop operating on plain numeric data-types
2. Locate `data.mat`, which represents population data $D \in \mathbb{R}^{100\times 2}$ recorded over 100 years. Load it in with `np.loadtxt('data.mat')`.



   As a metric to quantify the deviation between simulated population dynamics and data D, we propose the sum of squared deviations. For the same 10 steps as ex 1.1, compute this metric. Verify that you obtain `56.893`. Again, no CasADi features needed here.

3. Instead of feeding your simulation for-loop with numeric values $\hat{p}$, feed it with decision variables of an Opti problem. Minimize our metric over a horizon of 100. If you perform the estimation correctly, the fit will be exact and deliver nicely rounded parameters.

2 CasADi solvers

1. Reformulate your implementation such that your objective reads:

   ```
   opti.minimize(F.T @ F)
   ```

   with F a column vector.

2. We will pass a custom Hessian H to IPOPT as follows:

   ```
   sigma = MX.sym('sigma')
   opts["hess_lag"] = Function('hess_lag',...
     [opti.x, opti.p, sigma, opti.lam_g],...
     [sigma*triu(H)])
   opti.solver('ipopt',opts)
   ```

   Declare a Gauss-Newton Hessian H consistent with the objective function. A correct implementation will lead to convergence in 10 steps.

   In the IPOPT output, inspect the regularization column. How did it change from the previous exercise?

3. Undo the last changes. Reformulate your implementation such that your objective reads:

   ```
   F = vec(S(p)-D)
   opti.minimize(F.T @ F)
   ```

   where `S` is a CasADi Function that returns a $\mathbb{R}^{100\times 2}$ matrix with simulation results.

4. Inject an outlier into the data: set the number of rabbits at year 10 to 1. You will get an estimate for the parameters that deviates from that in the previous task. Simulate the population dynamics using these parameters (make use of `S`). Is the fit still good?

5. It is a classic result from convex programming that a 1-norm is more robust against outliers when doing fitting problems. Let's check if the result extends for our non-convex problem.

   Use a smooth reformulation.

3 (extra) Fitting with other solvers

There exist many codes for unconstrained optimization:

$$\underset{x}{\text{minimize}} \quad f(x), \tag{1}$$

as well as codes for nonlinear least-squares fitting:

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|F(x)\|_2^2. \tag{2}$$

Most codes expect a numerically evaluatable $f(x)$ or $F(x)$ and can optionally be provided with sensitivity information.
Choose your favourite code (e.g. `scipy.optimize.leastsq`) and redo exercise 1.6 while using CasADi to provide sensitivity information.