Numerical Optimal Control Course at IfA, ETH Zurich
on Feb-May 2011, by Prof. M. Diehl (K.U. Leuven)

Exercise 3: Gauss-Newton vs. BFGS, and Reverse Differentiation

Moritz Diehl and Robin Vujanic

In this third exercise we use again our Runge-Kutta simulator within a sequential approach. We make our model a bit nonlinear. Second, we compare the Gauss-Newton algorithm with a BFGS algorithm. Third, if there is still time, we write a new routine for the Jacobian calculation based on the reverse mode of algorithmic differentiation.

1. Throughout this exercise sheet, we make our controlled oscillator slightly nonlinear by making it a pendulum and setting

$$\frac{d}{dt}\begin{bmatrix} p(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ -C\sin(p(t)/C) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t), \quad t \in [0, T]. \tag{1}$$

with $C := 180/\pi$. We again abbreviate the ODE as $\dot{x} = f(x, u)$ with $x = (p, v)^T$, and choose again the fixed initial value $x_0 = (10, 0)^T$ and $T = 10$. Note that p now measures the deviation from the equilibrium state in degrees.

We regard again the optimal control problem from the last two exercise sheets

$$\min_{U \in \mathbb{R}^N} \|U\|_2^2 \quad \text{subject to} \quad g_{\text{sim}}(U) = 0 \tag{2}$$

and we use again RK4 and do again $N = 50$ integrator steps to obtain the terminal state x_N as a function of the controls $u_0, \ldots, u_{N-1}$.

Run again your Gauss-Newton scheme from the last exercise sheet, i.e. use in each iteration finite differences to compute the Jacobian matrix

$$J_{\text{sim}}(U) := \frac{\partial g_{\text{sim}}}{\partial U}(U)$$

and iterate

$$U_{k+1} = U_k - \begin{bmatrix} \mathbb{I} & 0 \end{bmatrix} \begin{bmatrix} 2\mathbb{I} & J_{\text{sim}}(U_k)^T \\ J_{\text{sim}}(U_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2U_k \\ g_{\text{sim}}(U_k) \end{bmatrix}$$

How many iterations do you need now, with the nonlinear oscillator? Plot the vector U_k and the resulting trajectory of p in each Gauss-Newton iteration so that you can observe the Gauss-Newton algorithm at work.

2. Modify your Gauss-Newton scheme so that you also obtain the multiplier vectors, i.e. iterate with $B_k = 2\mathbb{I}$ as follows:

$$\begin{bmatrix} U_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} U_k \\ 0 \end{bmatrix} - \begin{bmatrix} B_k & J_{\text{sim}}(U_k)^T \\ J_{\text{sim}}(U_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2U_k \\ g_{\text{sim}}(U_k) \end{bmatrix}$$

Choose as your stopping criterion now that the norm of the residual

$$\text{KKTRES}_k := \left\| \begin{bmatrix} \nabla_U \mathcal{L}(U_k, \lambda_k) \\ g_{\text{sim}}(U_k) \end{bmatrix} \right\| = \left\| \begin{bmatrix} 2U_k + J_{\text{sim}}(U_k)^T \lambda_k \\ g_{\text{sim}}(U_k) \end{bmatrix} \right\|$$

shall be smaller than a given tolerance, $\text{TOL} = 10^{-5}$. Store the values KKTRES_k and plot their logarithms against the iteration number k. What do you see?

3. BFGS method: now use a different Hessian approximation, namely the BFGS update, i.e. start with a unit Hessian, $B_0 = \mathbb{I}$ and then update the Hessian according to

$$B_{k+1} := B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k}.$$

with $s_k := U_{k+1} - U_k$ and $y_k := \nabla_U \mathcal{L}(U_{k+1}, \lambda_{k+1}) - \nabla_U \mathcal{L}(U_k, \lambda_{k+1})$. Devise your BFGS algorithm so that you need to evaluate the expensive Jacobian $J_{\text{sim}}(U_k)$ only once per BFGS iteration. Tipp: remember the old Jacobian $J_{\text{sim}}(U_k)$, then evaluate the new one $J_{\text{sim}}(U_{k+1})$, and only then compute B_{k+1}.

4. Observe the BFGS iterations and regard the logarithmic plot of KKTRES_k. How many iterations do you need now? Can you explain the form of the plot?

*** In the remainder of this exercise sheet, we want to compute the Jacobian $J_{\text{sim}}(U)$ in a more efficient way, inspired by the reverse mode of algorithmic differentiation (AD). This part of the exercise sheet is optional and you should only do it if you still have time and energy for it. ***

5. For a start, save your old routine for $J_{\text{sim}}(U)$ in a separate folder to be able to compare the results of your new routine with it later.

6. Then, note that the RK4 integrator step can be summarized in a function Φ so that the last state x_N, i.e. the output of the function $g_{\text{sim}}(U)$, is obtained by the recursion

$$x_{k+1} = \Phi(x_k, u_k), \quad k = 0, \ldots, N-1.$$

Along the simulated trajectory $\{(x_k, u_k)\}_{k=0}^{N-1}$, this system can be linearized as

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k, \quad k = 0, \ldots, N-1.$$

where the matrices

$$A_k := \frac{\partial \Phi}{\partial x}(x_k, u_k) \quad \text{and} \quad B_k := \frac{\partial \Phi}{\partial u}(x_k, u_k).$$

can be computed by finite differences. Note that we use the symbol B_k here for coherence with the notation of linear system theory, but that this symbol B_k here has nothing to do with the Hessian matrix B_k used in the other questions.

To become specific: modify your integrator so that

- Your RK4 step is encapsulated in a single function `[xnew]=RK4step(x,u)`
- You also write a function `[xnew,A,B]=RK4stepJac(x,u)` using finite differences with a step size of $\delta = 10^{-4}$
- your integrator stores and outputs both the trajectory of states $\{x_k\}_{k=0}^{N-1}$ and the trajectory of matrices $\{(A_k, B_k)\}_{k=0}^{N-1}$. Use three dimensional tensors like `Atraj(i,j,k)`.

The interface of the whole routine could be `[x,Atraj,Btraj]=forwardsweep(U)`

7. Now, using the matrices A_k, B_k, we want to compute $J_{\text{sim}}(U)$, i.e. write a routine with the interface `[Jsim]=backwardsweep(Atraj,Btraj)`. For this aim we observe that

$$\frac{\partial g_{\text{sim}}}{\partial u_k}(U) = \underbrace{(A_{N-1}A_{N-2}\cdots A_{k+1})}_{=:G_{k+1}} B_k$$

In order to compute all derivatives $\frac{\partial g_{\text{sim}}}{\partial u_k}(U)$ in an efficient way, we compute the matrices $G_{k+1} = (A_{N-1}A_{N-2}\cdots A_{k+1})$ in reverse order, i.e. we start with $k = N-1$ and then go down to $k = 0$. We start by $G_N := \mathbb{I}$ and then compute

$$G_k := G_{k+1}A_k, \quad k = N-1, \ldots, 0$$

8. Combining the forward and the backward sweep from the previous two questions, and write a new function for $J_{\text{sim}}(U)$. It is efficient to combine it with the computation of $g_{\text{sim}}(U)$, i.e. have the interface `[gsim,Jsim]=gsimJac(U)`. Compare the result with the numerical Jacobian calculation from before by taking `norm(Jsimold-Jsimnew)`.

9. How do the computation times of old and the new Jacobian routine scale with N? This question can be answered without numerical experiments,just by thinking.

10. Now run your Gauss-Newton algorithm again and verify that it gives the same solution and same number of iterations as before.