Automatic Differentiation

For optimization purposes the sensitivities of the function $f(x) = x_1 \sin(x_2) - 5x_3$, with $x = [x_1 \ x_2 \ x_3]^T$, shall be calculated.

a) Complete the graph for the function $f(x)$ and write down the derivatives along the arcs.

\[
\frac{\partial x_4}{\partial x_3} = 
\]

\[
x_4 = 
\]

\[
\begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
x_4 = \\
\end{array}
\]
Firstly, use the **forward mode** to compute the gradient of the function.

b) Give the index sets $J_i$ for your introduced intermediate and output quantities.

c) Compute the gradients for all input and intermediate quantities.

d) Give the gradient of the function $f(x)$ in terms of the input quantities $x = [x_1 \ x_2 \ x_3]^T$. 
Use also the **reverse mode** to compute the gradient of the function.

e) Give the index sets $I_i$ for the input quantities and the introduced intermediate quantities.

f) Compute the scalar gradients $\bar{x}_j$ for the output and intermediate quantities.

g) Give the gradient of the function $f(x)$ in terms of the input quantities $x = [x_1 \ x_2 \ x_3]^T$.

h) Check your result by direct differentiation of the function $f(x)$. 
Tool 1: CasADi for MATLAB

CasADi\(^1\) is an open-source MATLAB tool for nonlinear optimization and automatic differentiation. It is written in C++ but is most conveniently used via full-featured interfaces to MATLAB or Python. Applications range from academic teaching to fields such as optimal control, robotics, and aerospace. In order to use CasADi, an appropriate version can be downloaded\(^2\) and added to the MATLAB search path.

1. \texttt{import casadi.*} % add CasADi toolbox
2. \texttt{x = SX.sym('x', 2);} % create 2-D variable x
3. \texttt{f = x(1)*x(2) - sin(x(2));} % define the function expression
4. \texttt{gf = jacobian(f,x);} % calculate gradient expression
5. \texttt{Hf = jacobian(gf,x);} % calculate Hessian expression
6. \texttt{gf_fun = Function('gf',{x},{gf});} % create functions for ...  
7. \texttt{Hf_fun = Function('Hf',{x},{Hf});} % ... evaluation of expressions
8. \texttt{x0 = [1, 0];} % define evaluation point
9. \texttt{y1 = gf_fun(x0)} % evaluate gradient
10. \texttt{y2 = Hf_fun(x0)} % evaluate Hessian

Output: \n\begin{align*}
y1 &= [[0, 0]] \quad \% \text{value of gradient at } x0 \\
y2 &= [[0, 1], [1, 0]] \quad \% \text{value of Hessian at } x0
\end{align*}

Tool 2: autograd in Python

Autograd\(^3\) is a Python package that can automatically differentiate native Python and Numpy code, the typical Python package for scientific computing. The main intended application of Autograd is gradient-based optimization. It can be installed via the package manager pip:

\texttt{pip install autograd}

1. \texttt{import autograd.numpy as np} # import numpy
2. \texttt{from autograd import jacobian} # import jacobian for AD
3. \texttt{def f(x):} # define the function
4. \hspace{1em} \texttt{return x[0]*x[1] - np.sin(x[1])} # calculate gradient expression
5. \texttt{gf_fun = jacobian(f)} # calculate Hessian expression
6. \texttt{Hf_fun = jacobian(gf_fun)} # define evaluation point
7. \texttt{x0 = np.array([1., 0.])} # evaluate gradient
8. \texttt{print(gf_fun(x0))} # evaluate Hessian
9. \texttt{print(Hf_fun(x0))}

Output: \n\begin{align*}
[0., 0.] \quad \% \text{value of gradient at } x0 \\
[[0., 1.], [1., 0.]] \quad \% \text{value of Hessian at } x0
\end{align*}

A very similar alternative developed from Autograd but also supporting, e.g., just-in-time compilation of gradients and functions is JAX\(^4\).

---


\(^2\) https://web.casadi.org/get/

\(^3\) https://github.com/HIPS/autograd

\(^4\) https://github.com/google/jax