Jacobian matrix, partial and total derivatives: Symbolic Toolbox examples

This script worked with Matlab **R2021b**

**Code, comments and related materials in description*

Video-presentation: http://personales.upv.es/asala/YT/V/derivsmlEN.html

Objectives: illustrating with a Matlab Symbolic Toolbox example how to operate with partial derivatives, chain rule and total derivatives.

Table of Contents

Preliminaries: functions we will work with....1
Partial derivatives and jacobian matrix (symbolic toolbox)....2
Composite function and (multivariable) chain rule....3
Total derivatives....5
  Computation of derivatives of q via chain rule....6
Conclusions....7

Preliminaries: functions we will work with

Let us define a couple of ordinary "numeric" functions in Matlab as:

```
f=@(x) [x(1)*x(2); ...
        1-x(1)^2; ...
        sin(x(2))/(1+x(2)^2)] %input is a vector with 2 elements
```

```
f = function_handle with value:

    @(x)[x(1)*x(2);1-x(1)^2;sin(x(2))/(1+x(2)^2)]
```

```
g=@(y) [cos(y(1)); ...
        sin(y(1)+y(2))] %input is a vector with 2 elements
```

```
g = function_handle with value:

    @(y)[cos(y(1));sin(y(1)+y(2))]
```

f maps $\mathbb{R}^2 \mapsto \mathbb{R}^3$, and g maps $\mathbb{R}^2 \mapsto \mathbb{R}^2$

```
f([2;2])
```

```
ans = 3×1
    4.0000
   -3.0000
    0.1819
```

```
g([3;-1])
```

```
ans = 2×1
   -0.9900
    0.9093
```

Arguments of f and g are numerical **vectors**, or any "object" that can be multiplied, divided, added and has *sine* and *cosine* functions defined upon.

As ordinary "plain code" in a given programming language, does not admit "derivatives" (well, they can be done numerically --approximately-- or with so called "automatic differentiation" code, but that's not in the scope of this material), then we will create their symbolic toolbox equivalents, by letting the functions be evaluated with "symbolic" arguments:

```
syms x1 x2 y1 y2 real
f_sym=f([x1;x2]) %now, f is a symbolic expression
```

```
f_sym =
```

$$\begin{pmatrix} x_1\,x_2 \\ 1-{x_1}^2 \\ \dfrac{\sin(x_2)}{{x_2}^2+1} \end{pmatrix}$$

```
g_sym=g([y1;y2])
```

```
g_sym =
```

$$\begin{pmatrix} \cos(y_1) \\ \sin(y_1+y_2) \end{pmatrix}$$

Partial derivatives and jacobian matrix (symbolic toolbox)

The partial derivatives of each component of f with respect to, say, x_2 can be computed one by one as, say:

```
diff(f_sym(3),x2)
```

```
ans =
```

$$\frac{\cos(x_2)}{{x_2}^2+1}-\frac{2\,x_2\sin(x_2)}{\left({x_2}^2+1\right)^2}$$

The command "diff" actually evaluates derivatives of each element of its first input with respect to the second one:

```
diff(f_sym,x1)
```

```
ans =
```

$$\begin{pmatrix} x_2 \\ -2\,x_1 \\ 0 \end{pmatrix}$$

But arranging all of them in a matrix (called Jacobian matrix, being a 3×2 matrix) yields

```
[diff(f_sym,x1) diff(f_sym,x2)]
```

```
ans =
```

$$\begin{pmatrix} x_2 & x_1 \\ -2\,x_1 & 0 \\ 0 & \frac{\cos(x_2)}{x_2^2+1} - \frac{2\,x_2\sin(x_2)}{\left(x_2^2+1\right)^2} \end{pmatrix}$$

which, actually can be computed with:

```
jac_f=jacobian(f_sym)
```

```
jac_f =
```

$$\begin{pmatrix} x_2 & x_1 \\ -2\,x_1 & 0 \\ 0 & \frac{\cos(x_2)}{x_2^2+1} - \frac{2\,x_2\sin(x_2)}{\left(x_2^2+1\right)^2} \end{pmatrix}$$

Same for g, of course (Jacobian is now a 2×2 matrix):

```
jac_g=jacobian(g_sym)
```

```
jac_g =
```

$$\begin{pmatrix} -\sin(y_1) & 0 \\ \cos(y_1+y_2) & \cos(y_1+y_2) \end{pmatrix}$$

Note: the order of the variables (each is associated to a given column) is alphabetic, capital letters first, and the command differentiates with respect to ALL symbolic variables.

If we wished a different ordering, we need to explicitly tell Matlab:

```
jac_g_reordered=jacobian(g_sym,[y2 y1]) %colums are now swapped
```

```
jac_g_reordered =
```

$$\begin{pmatrix} 0 & -\sin(y_1) \\ \cos(y_1+y_2) & \cos(y_1+y_2) \end{pmatrix}$$

The "jacobian" command does not take derivatives of other variables apart from the ones in its second argument. This is helpful to, say, avoid derivatives with respect to "constants" such as mass, etc. in our equations. Thus, using the second argument of "jacobian" is perhaps advisable to make things clearer for you and your readers, when lots of symbols appear in your expressions.

Composite function and (multivariable) chain rule

Let us now consider $z(y) = f(g(y))$, i.e., the symbolic expression:

```
z=f(g([y1;y2]))
```

z =

$$\begin{pmatrix} \sin(y_1+y_2)\cos(y_1) \\ 1-\cos(y_1)^2 \\ \dfrac{\sin(\sin(y_1+y_2))}{\sin(y_1+y_2)^2+1} \end{pmatrix}$$

Then, z maps $\mathbb{R}^2 \mapsto \mathbb{R}^3$, and its Jacobian is a 3×2 matrix

```
jac_z=simplify(jacobian(z))
```

jac_z =

$$\begin{pmatrix} \cos(2\,y_1+y_2) & \cos(y_1+y_2)\cos(y_1) \\ \sin(2\,y_1) & 0 \\ \sigma_1 & \sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = \frac{\cos(\sin(y_1+y_2))\cos(y_1+y_2)}{\sin(y_1+y_2)^2+1} - \frac{2\sin(\sin(y_1+y_2))\cos(y_1+y_2)\sin(y_1+y_2)}{\left(\sin(y_1+y_2)^2+1\right)^2}$$

However, multivariable chain rule says that this Jacobian (obtained "directly", after substitution) coincides with the product of the Jacobian matrices of f and g:

```
chainrule_tmp = jacobian(f_sym)*jacobian(g_sym) % Warning: not yet correct, we need to
```

chainrule_tmp =

$$\begin{pmatrix} x_1\cos(y_1+y_2)-x_2\sin(y_1) & x_1\cos(y_1+y_2) \\ 2\,x_1\sin(y_1) & 0 \\ \sigma_1 & \sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = \cos(y_1+y_2)\left(\frac{\cos(x_2)}{x_2^2+1} - \frac{2\,x_2\sin(x_2)}{\left(x_2^2+1\right)^2}\right)$$

```
chainrule_finalform = simplify(subs(chainrule_tmp, {'x1','x2'}, {g_sym(1),g_sym(2)}))
```

chainrule_finalform =

$$\begin{pmatrix} \cos(2\,y_1 + y_2) & \cos(y_1 + y_2)\cos(y_1) \\ \sin(2\,y_1) & 0 \\ \sigma_1 & \sigma_1 \end{pmatrix}$$

where

$$\sigma_1 = \cos(y_1 + y_2)\left(\frac{\cos(\sin(y_1 + y_2))}{\sin(y_1 + y_2)^2 + 1} - \frac{2\sin(\sin(y_1 + y_2))\sin(y_1 + y_2)}{\left(\sin(y_1 + y_2)^2 + 1\right)^2}\right)$$

```
simplify(chainrule_finalform-jac_z)
```

```
ans =
```

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Note: beware of variable ordering... if the symbolic inputs to f were named "(Bob,Alice)", then Jacobian would have derivatives with respect to "(Alice, Bob)" so the product above would mess up and provide an incorrect answer. If you are in doubt, you are advised to put the variables explicitly `jacobian(f_sym,[x1 x2])*jacobian(g_sym,[y1 y2])`.

Summary: "chain rule" for composition of functions says that "substitution first, derivatives later" (i.e., the obvious direct way) gets the same result as "derivatives first, product of jacobians, substitution last".

Total derivatives

This is a naming popular in physics. Let us imagine that the actual inputs y to g are functions of "time" t. Then

```
syms t real
y1_t=-t; y2_t=t^2; %y1 and y2 are no longer arbitrary symbolic variables.
y_t=[y1_t; y2_t]
```

```
y_t =
```

$$\begin{pmatrix} -t \\ t^2 \end{pmatrix}$$

```
dydt=diff(y_t) %Vector of "velocities"
```

```
dydt =
```

$$\begin{pmatrix} -1 \\ 2\,t \end{pmatrix}$$

Let us now define a function which depends on time both "implicitly" (via (y_1, y_2) if they were functions of time, as they actually are now), and "explicitly":

```
q_yt=g([y1;y2])+[t;-t]
```

q_yt =

$$\begin{pmatrix} t+\cos(y_1) \\ \sin(y_1+y_2)-t \end{pmatrix}$$

Hence, once the actual trajectory $(y_1(t), y_2(t))$is substituted above (instead of arbitrary symbols y_1 and y_2), results in:

```
q_t=g([y1_t;y2_t])+[t;-t]
```

q_t =

$$\begin{pmatrix} t+\cos(t) \\ -t-\sin(t-t^2) \end{pmatrix}$$

Our goal is obtaining the velocities (time derivatives) of $q(t)$. In this case "total" and "partial" derivatives are the same (well, maybe even the terminology is meaningless or at the least misleading), as q depends only on time.

Hence, trivially, the time derivatives of q are:

```
the_speed_of_q=diff(q_t)
```

the_speed_of_q =

$$\begin{pmatrix} 1-\sin(t) \\ \cos(t-t^2)\ (2\,t-1)-1 \end{pmatrix}$$

Computation of derivatives of q via chain rule

Now, "`the_speed_of_q`" above is called "total" derivative, but chain rule gets only "partial" derivatives.

Partial derivatives of q_yt (a function of y_1, y_2, and t) are:

```
symvar(q_yt) %note that variables are ordered alphabetically, this will be the ordering
```

ans = $\begin{pmatrix} t & y_1 & y_2 \end{pmatrix}$

```
jac_q_yt=jacobian(q_yt,[y1 y2 t]) %let us operate in a different order of variables, to
```

jac_q_yt =

$$\begin{pmatrix} -\sin(y_1) & 0 & 1 \\ \cos(y_1+y_2) & \cos(y_1+y_2) & -1 \end{pmatrix}$$

So, in order to compute the total derivative "`the_speed_of_q`" we must apply chain rule in the two first columns, as above:

```
dqdt_tmp=jacobian(q_yt,[y1 y2])*dydt + diff(q_yt,t) %not yet correct, we need to substi
```

dqdt_tmp =

$$\begin{pmatrix} \sin(y_1)+1 \\ 2\,t\cos(y_1+y_2)-\cos(y_1+y_2)-1 \end{pmatrix}$$

Note that `jacobian(q_yt,[y1 y2])` only evaluates partial derivatives with respect to the variables in its second argument... it depends on a third symbol (t) but it skips it.

So when we evaluate the above `dqdt_tmp` replacing `y1` and `y2` (coming from the Jacobian evaluation) with the actual trajectory, we get:

```
dqdt_substitutionmade=subs(dqdt_tmp, {'y1','y2'}, {y1_t,y2_t})
```

```
dqdt_substitutionmade =
```

$$\begin{pmatrix} 1-\sin(t) \\ 2\,t\cos(t-t^2)-\cos(t-t^2)-1 \end{pmatrix}$$

And the result is, obviously, the same as "`the_speed_of_q`" we previously obtained by substitution "before" taking derivatives.

```
simplify(dqdt_substitutionmade-the_speed_of_q)
```

```
ans =
```

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Conclusions

We showed how to manipulate Matlab functions, convert them to symbolic expressions, and verified that chain rule works, including in the total/partial derivative case usual in physics jargon.