

Sensitivity Analysis, Automatic Differentiation, and Subset Selection

Adam Attarian

September 14, 2009

Sensitivity Analysis

Given a parameter dependent ODE,

$$\dot{x} = f(t, x(t, q), q), \quad x \in \mathbb{R}^n, q \in \mathbb{R}^m$$

the sensitivities are given by how the state changes wrt the parameters. Differentiating wrt q we obtain

$$\frac{d}{dt} \frac{\partial x}{\partial q_j} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial q_j} + \frac{\partial f}{\partial q_j},$$

which are the sensitivity equations. Computing the sensitivities requires solving $mn + n$ differential equations: n from the original states and mn sensitivities.

Sensitivity Analysis

- ▶ Sensitivity analysis is a local concept – the model is evaluated at a specific, fixed parameter value.
- ▶ You see how the model changes sensitivity to parameters over time – this can be important.
- ▶ The derivatives $\frac{\partial x}{\partial q_j}$ are solved for by the integrator, the other derivatives must be provided by some other method.
- ▶ Could use finite difference, analytic solutions if you got'em, or automatic differentiation.

Sensitivity Analysis

Once the sensitivities are computed, we compute the *relative ranking*, given by

$$\left\| \frac{\partial x_i}{\partial q_j} \right\|_2 = \left[\frac{1}{t_f - t_0} \int_{t_0}^{t_f} \left(\frac{\partial x_i}{\partial q_j} \frac{q_j}{x_i} \right)^2 dt \right]^{1/2}.$$

- ▶ Multiplying by $\frac{q_j}{x_i}$ essentially non-dimensionalizes the results allowing comparison.
- ▶ This allows a determination to be made as to which are they most sensitive parameters in a model.
- ▶ To numerically compute this, use a trapezoidal method. In Matlab, it's `trapz(X,Y)`. Do **NOT** use `trapz(Y)`. This assumes unit spacing and is *bad*.

Spring Example

Consider the spring mass model that has been normalized with respect to mass,

$$\ddot{x} + C\dot{x} + Kx = 0.$$

A change of variables gives us the system

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -Cx_2 - Kx_1$$

Now we differentiate the system with respect to the parameters...

Spring Sensitivities

And then manually, by hand taking the partial derivatives we have

$$\frac{d}{dt} \frac{\partial x_1}{\partial C} = \frac{\partial x_2}{\partial C}$$

$$\frac{d}{dt} \frac{\partial x_1}{\partial K} = \frac{\partial x_2}{\partial K}$$

$$\frac{d}{dt} \frac{\partial x_2}{\partial C} = -x_2 - C \frac{\partial x_2}{\partial C} - K \frac{\partial x_1}{\partial C}$$

$$\frac{d}{dt} \frac{\partial x_2}{\partial K} = -C \frac{\partial x_2}{\partial K} - x_1 - K \frac{\partial x_1}{\partial K}.$$

So you can see how for large compartment models with many states and parameters this is a long, tedious, error prone task. Better way: Automatic Differentiation.

Automatic Differentiation (AD)

- ▶ AD is essentially the application of the chain rule on an expression.
- ▶ AD uses the product rule, quotient rule, it knows the derivative of \sin is \cos , etc. A big table lookup.
- ▶ AD breaks down a function into its basic constituent operations, and then differentiates.
- ▶ Not a numerical approximation to the derivative, but rather machine-precision exact.
- ▶ Free AD for Matlab: myAD by Martin Fink. Available on the Matlab File Exchange [here](#).

AD Example

Suppose we want to take the derivative of a vector valued function, $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ at the point $(1, 1)$, where

$$f(x_1, x_2) = \begin{bmatrix} 2x_1 + x_2^2 \\ x_1^3 + x_1x_2 \end{bmatrix}.$$

The derivative is given by the Jacobian matrix:

$$f'(x_1, x_2) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2 & 2x_2 \\ 3x_1^2 + x_2 & x_1 \end{bmatrix} \Big|_{(1,1)} = \begin{bmatrix} 2 & 2 \\ 4 & 1 \end{bmatrix}.$$

And to code this up in AD...

AD Example

```
1 function adex
2
3 x=[1;1];
4
5 bigX=myAD(x); % convert to a myAD variable
6 out=fun(bigX); % evaluate with the myAD variable
7
8 dfdx=getderivs(adOut); % get the jacobian
9
10 function f=fun(x)
11 x=[2*x(1)+x(2)^2;
12 x(1)^3+x(1)*x(2)];
```

Running the code returns the result that we expect.

AD & Sensitivities

The book keeping on computing sensitivities on many states and parameters becomes a problem, so we wrote a code `tssolve.m` to do the hard parts. We'll apply `tssolve` to the spring model in a moment, but we first need to modify the spring ODE as follows.

```
1 function dx=springode(t,x,q)
2
3 K=q(2);
4 C=q(1);
5
6 dx=[ (q(1) ./q(1)) .*x(2);
7      -K.*x(1)-C.*x(2) ];
```

⇒ each state needs to have a parameter in it, and myAD only operates on scalar operations (!)

AD & Sensitivities

`tssolve.m` is available from [here](#), as is the [manual](#) (read the manual or read the help!). To compute just relative sensitivities, here is how to use the code.

```
1 q0=[4;5];
2 x0=[1 1];
3 t=linspace(0,5,100);
4 sens=tssolve(@springode,x0,q0,t,1);
```

`sens` is a Matlab structure, so it has lots of things in it. The important ones are

- ▶ `sens.t`: the time vector
- ▶ `sens.relSens`: a 3D array containing the relative sensitivities

AD & Sensitivities

RS.relsens is ordered in 3D arrays as (time, state, parameter), so the sensitivity of state one with parameter 5 would be `relsens(:,1,5)`.

In the spring example, $q = [q_1, q_2] = [C, K]$, so we have

$$\frac{\partial x_1}{\partial C} \quad \text{sens.relsens}(:,1,1)$$

$$\frac{\partial x_1}{\partial K} \quad \text{sens.relsens}(:,1,2)$$

$$\frac{\partial x_2}{\partial C} \quad \text{sens.relsens}(:,2,1)$$

$$\frac{\partial x_2}{\partial K} \quad \text{sens.relsens}(:,2,2)$$

Read the `tssolve.m` documentation for more info. Computes lots of other things if you want.

Subset Selection

- ▶ Subset selection allows you to determine a set of parameters that are most identifiable.
- ▶ Many times it is preferable to optimize over a smaller set of parameters rather than a larger set.
- ▶ The ones that are deemed less identifiable can be fixed to some nominal value.
- ▶ Use a QR decomposition with column pivoting applied to a sensitivity matrix.

Subset Selection

We'll define the sensitivity matrix to be all of the sensitivities of the model output. On the spring model, if we only observe velocity (N observations) then

$$S = \left[\begin{array}{cc} \frac{\partial x_1}{\partial C} & \frac{\partial x_1}{\partial K} \end{array} \right]_{N \times 2}.$$

Then the parameters C, K are identifiable iff $\text{rank}(S) = m$, the number of unknown parameters. Equivalently,

$$\det(S^T S) \neq 0.$$

So the “more linearly independent” columns of $S^T S$ are more identifiable. How do determine which is more linearly independent?

Subset Selection

We use a QR factorization with column pivoting. We factor $S^T S$ as

$$S^T S \Pi = QR,$$

where Π is a permutation matrix. QR with pivoting puts the most linearly independent columns of $S^T S$ to the left.

- ▶ The permutation matrix encodes which parameters were moved.
- ▶ You can essentially read it off from the matrix which parameters are most identifiable.
- ▶ In Matlab: `[Q,R,P]=qr(A)`

Subset Selection

Suppose you're analyzing three parameters, and after doing QR with pivoting you obtain a permutation matrix that looks like

$$\Pi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

This says, essentially, the third parameter is the most identifiable, followed by the first, and lastly the second. For a more quantitative take on things, take a look at the eigenvalues of $S^T S$.