Optimization, Auto-Differentiation, and Tomlab

May 8, 2017

Derivatives and Numerical Methods

There are two general types of algorithms for optimizers/solvers/etc.:

1. Derivative-free:
   - e.g. Simplex and Nelder-Mead. This is Matlab's `fminsearch`
   - Also, "costly global function" optimization
   - Avoid at all costs (though sometimes don't have a choice)

2. Derivatives
   - Pretty much every other algorithm, especially for large number of variables/constraints
   - Including global optimization techniques (which use derivatives locally)

Key derivatives to calculate are:

- Gradient of objective
- Hessian of objective (nonlinear least squares and some algorithms only use gradient)
- Jacobian of constraints

Calculating Derivatives

How to calculate derivatives for the objective and constraints?

1. Calculate by hand
   - Sometimes, though not always, the most accurate and fastest option
   - But algebra is error prone for non-trivial setups
   - (note: many optimizers have a way to check your analytical derivatives)

2. Finite-differences:
   - $\boldsymbol{\partial}_{x_i} f(x_1, \ldots x_N) \approx \frac{f(x_1, \ldots x_i + \Delta, \ldots x_N) - f(x_1, \ldots x_i, \ldots x_N)}{\Delta}$
   - Evaluates function at least N extra times to get a gradient
   - $\#$ evaluations for Jacobians with M constraints even worse
   - Large Δ is numerically stable but inaccurate, small Δ is unstable
   - **Avoid like the plague!** (and is what matlab does out of the box)

3. Auto-differentiation
   - Not a form of finite-differences or numeric differentiation
   - Essentially analytical. Repeated use of the chain-rule
   - Does not work for every function, but only evaluates $f(\cdot)$ once if it works

Auto-differentiation (adapted from Wikipedia)

- Remember the chain rule: $\frac{dy}{dx} = \frac{dy}{dw}\frac{dw}{dx}$
- Consider functions composed of calculations with fundamental operations (with known analytical derivatives)
- For example, consider function: $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$

Operations to compute value	Operations to compute $\frac{df(x_1,x_2)}{dx_1}$
$w_1 = x_1$	$\frac{dw_1}{dx_1} = 1$ (seed)
$w_2 = x_2$	$\frac{dw_2}{dx_1} = 0$ (seed)
$w_3 = w_1 \cdot w_2$	$\frac{dw_3}{dx_1} = w_2 \cdot \frac{dw_1}{dx_1} + w_1 \cdot \frac{dw_2}{dx_1}$
$w_4 = \sin w_1$	$\frac{dw_4}{dx_1} = \cos w_1 \cdot \frac{dw_1}{dx_1}$
$w_5 = w_3 + w_4$	$\frac{dw_5}{dx_1} = \frac{dw_3}{dx_1} + \frac{dw_4}{dx_1}$

- Generalizes to multiple variables. AD takes source code and generates the derivatives at the same time (i.e. doesn't increase with # variables)

Implementations of AD

- A field unto itself. Do not implement directly
- Implementation is language dependent. Two approaches:
  - *Source code transformation:* utility (outside of the language itself) reads in the code for your function, and generates a function which calculates value and derivative. Rerun if you change your code
  - *Operator Overloading:* Takes your existing functions, and passes variables that act like numbers, but are actually recording and tracing the chain rule steps/etc. Can be magical, or infuriating
- Implementation depends on the language:
  - *Fortran:* usually needs SCT. Many choices: e.g. `http://tapenade.inria.fr:8080/tapenade/index.jsp`
  - *Python:* `https://github.com/LowinData/pyautodiff` and `https://pythonhosted.org/algopy/`
  - *C++:* overloading `http://www.fadbad.com/fadbad.html`,...
  - *Matlab:* open source SCT (e.g. AdiMat) not very good. Use Tomlab/MAD instead, coupled with the Tomlab optimizer.

Getting Derivatives Directly in Tomlab

```
%Example function
f = @(x) 3 * x  + exp(x);

%Evaluating function
x_val = 2.1;
f(x_val)

%Evaluating with derivative at x_val
x = fmad(x_val, 1); %Seed, since dx/dx = 1
f_val = f(x)

%Extract (both calculated at same time)
getvalue(f_val)
getderivs(f_val)
```

Black Magic? Is it Always so Easy?

- Auto-differentiation works seamlessly for functions composed of an arbitrarily complicated # of simple functions
  - Just need analytical derivatives for the lowest-level functions
  - Functions of vector and matrices are no problem at all. In fact, the field was designed for large numbers of variables/constraints and sparsity
- Can you call other functions (with operator overloading)?
  - Depends on how they were written. Often no problem at all
  - If the functions assume arguments are numbers, there can be problems
  - Sometimes can fix the underlying code to make more generic (example)
- Verboten: Iterations and fixed-points *within a function*
  - e.g. it can't differentiate an optimization step within a function
  - However, many algorithms can be re-written without nesting (e.g. nested fixed-point vs. MPEC for discrete-choice estimation)
  - Possible that simulation could be embedded (e.g. mixed-logit)...

Fixing the Common Problem

Bla...

Adding a New Function

What if I know the derivative of a function which isn't there?

Sparsity

Bla...

-