Optimization, Auto-Differentiation, and Tomlab

May 8, 2017

Auto-Differentiation(AD)

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—i.e. $O(1)$ not $O(N \times M)$ for $f : \mathbb{R}^N \to \mathbb{R}^M$

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`,...
  - *R:* `https://://cran.r-project.org/web/packages/madness/index.html`
  - *Matlab:* open source SCT (e.g. AdiMat) not very good. Use Tomlab/MAD instead, coupled with the Tomlab optimizer.

Sparsity

Sparse Matrices and Methods

- Many algorithms are specialized for matrices (or Jacobians or Hessians) with many 0s—e.g. Gaussian elimination
- Only store non-zero values, but $0 \neq 0.0$ for optimizers
- Not (usually) for storage, but rather specialized algorithms
- For Jacobians and Hessians, can solve enormous (e.g. hundreds of thousands or millions) of variable systems
  - But the more non-zeros, the more likely dense methods are preferable.
- For example, $f : \mathbb{R}^N \to \mathbb{R}^N$ with $f(x) = \sqrt{x}$ point-wise
  - Jacobian has N non-zeros, while dense has N^2
  - Optimizers/solvers can use this to step in the right direction
  - Auto-differentiation will figure out the sparsity pattern of derivatives—i.e., which values are always 0 for all inputs

Sparse Matrices in Matlab

```
%First, can convert dense matrix, and it drops the 0's.
X = [1.0 0   0;
     2.0 1.0 0];
S = sparse(X)
%S =
%(1,1)        1
%(2,1)        2
%(2,2)        1

%Or can take lists of indices and values,
x_indices = [1; 2; 2];
y_indices = [1; 1; 2];
values = [1; 2; 1];
S2 = sparse(x_indices, y_indices, values)

%Or can preallocate and just reference in loops/etc.
S3 = sparse(0,3);
S3(1,1) = 1;
S3(2,1) = 2;
S3(2,2) = 1;
```

Tomlab

What is in Tomlab?

- Sadly, the Operations Research community keeps the best implementations closed-source
- Collection of sparse/dense linear/nonlinear local/global constrained/unconstrained continuous/mixed-integer optimizers
- Nonlinear methods have built in auto-differentiation
- Repackages and resells state-of-the-art commercial products, and adds a few of its own which are high quality
- Several methods to solve the same type of problem, because you never know which one will work best. Easy to swap

What Types of Problems?

See http://tomopt.com/docs/TOMLAB_QUICKGUIDE.pdf.

- Programming = Optimizer in OR
- Linear Programming (LP) and Mixed-Integer LP (MILP)
- Constrained Nonlinear Programming (NLP)
- Unconstrained Global Optimization (glb)
- Linear Least Squares (LLS)
- Nonlinear Least Squares (NLLS)
- Solving systems of equations generally uses NLLS
- ...and many others (semi-definite, quadratic, etc.)

Most have sparse vs. dense algorithms, and constrained vs. unconstrained

- Read docs to find best fit for your particular problem
- Always use appropriate constraints (none, box-bounded, linear, etc.)
- For borderline sparse problems, sometimes dense works better

Purchased Packages

See `http://tomopt.com/tomlab/products/`

- Stanford Systems Optimization Laboratory (SOL): SNOPT, NPSOL, NLSSOL, LSSOL...
- Knitro. Good for big problems, and complementarity conditions
- MAD (auto-differentiation)
- LGO and CGO (global optimizers, costly and otherwise)
- For a given problem type, tomlab will list available algorithms

Linear Least Squares (i.e. Regression)

$$\min_x \frac{1}{2} \|Cx - d\|_2$$
$$\text{s.t. } x_L \leq x \leq x_U$$
$$b_L \leq Ax \leq b_U$$

- Little benefit over stata until problems get large or sparse
- Though if there are linear constraints, A, may be helpful
- Major benefit for large, sparse C
- See "Section 11. LLS Problem" in `http://tomopt.com/docs/TOMLAB_QUICKGUIDE.pdf`

Example: Two-way fixed Effect

- Use student, i, and instructor, j fixed effects with observables

$$\text{grade}_{ij} = \text{observables}_{ij} + \text{student}_i + \text{instuctor}_j + \epsilon_{ij}$$

- 300K observations for 36K students and 945 instructors
- Took about a day to solve in Stata
- See `teacher_student_fixed_effect.m` example for generating sparse matrix. Given `id1` student id, and `id1` instructor id. Key code:

```
%Preallocate a sparse matrix
%Total number of observables with indicators for the two types.
X = sparse(N_observations, N_observables + N_students + N_teachers);

%Filling in indicators for the matches
for i=1:N_observations
X(i, N_observables + id1(i)) = 1; %set student indicator
X(i, N_observables + N_students + id2(i)) = 1; %sets instructor indicator
end
```

Solving LLS in Tomlab

- Given X and y such that $\min_\beta \|X\beta - y\|_2$
- Ensure X loaded sparse, with each row having 2 indicators

```
%linear least squares, can pass in sparse matrices or use dense
% call XXXAssign for problem type XXX
Prob = llsAssign(X, y, [], [], 'LLS Example');

%Can change settings.  See tomlab documentation
Prob.optParam.MaxIter = 5000; %optional, increase iterations
Prob.PriLevOpt  = 1; %optional, gives more information if higher

%Run, passing in the algorithm type
%Takes about 10-20 seconds to run, instead of a day.  Not even tweaked
Result = tomRun('Tlsqr', Prob); %intended for sparse unconstrained LLS
beta = Result.x_k;


%Tried alternative methods.  Easy to swap
%Result = tomRun('snopt', Prob); %Tlsqr works much better here
```

AD with Tomlab

Using AD Directly in Tomlab

- Keep in mind that optimizers/solvers in Tomlab do this automatically
- But if having trouble with optimizer calls, can test function separately

```
%Example function.  Also works fine with separate files/function defs
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 graph 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)...

Keep Functions Generic

- Remember, MAD replaces arguments with things that look like variables. Keep everything generic, don't overwrite with other types
- Some internal matlab functions do this sort of thing
- Sometimes can copy/paste others sourcecode and tweak

```
madinitglobals; %Need to run for auto-differentiation to work
x_val = [2.1;3.0];
x = fmad(x_val, eye(2,2)); %Seeds with derivatives

%Extract (both calculated at same time)
f_val = f_func(x)

function y = f_func(x)
%x = 1; %Don't do this!!!!!!
%y = zeros(1,1) %Don't do this!!!!!
%y(1) = x.^2; %careful not to preallocate as double
y = x.^2; %This leaves x, y generic
end
```

Missing Function (with Analytical Derivative)

- See MAD manual, `http://tomopt.com/docs/TOMLAB_MAD.pdf`
- See `Adding Functions to the fmad Class`
- Example, `normcdf` isn't there, could add something like (untested):

```
function y=normcdf(x)
        y = x; %Needs to copy
        y.value= normcdf(x.value); %Evaluate given double
        y.deriv = normpdf(x.value) .* x.deriv; %Note chain rule
end
```