Automatic Differentiation

Lecture 18b

What is "Automatic Differentiation"

- We want to compute both f(x), f'(x), given a program for f(x):=exp(x^2)*tan(x)+log(x).

- We could do this via…

$$\frac{d\left(e^{x^2}\tan x + \log x\right)}{dx} = 2\,x\,e^{x^2}\,\tan x + e^{x^2}\,\sec^2 x + \frac{1}{x}$$

But this is too much work.

- All we really want is a way to compute, for a specific numeric value of x, f(x), and f'(x). Say the value is x=2.0. We could evaluate those two expressions, or we could evaluate a taylor series for f around the point 2.0, whose coefficients would give us f(2) and f'(2)/2!
- Or we could do something cleverer.

ADOL, ADOL-C, Fortran (77, 90) or C "program differentiation"

- Rewrite (via a compiler?) a more-or-less arbitrary Fortran program that computes f(c) into a program that computes $\langle f(c), f'(c) \rangle$ for a real value c.
- How to do this?
- Two ways, "forward" and "reverse"

Forward automatic differentiation

- In the program to compute f(c) make these substitutions:
  - Where we see the expression x, replace it with ⟨c,1⟩
    - the rule d(u) = 1 if u=x.
  - Where we see a constant, e.g. 43, use ⟨43,0⟩
    - the rule d(u)=0 if u does not depend on x
  - Where we see an operation ⟨a,b⟩ +⟨r,s⟩ use ⟨a+r,b+s⟩
    - the rule d(u+v) = du+dv
  - Where we see an operation ⟨a,b⟩ *⟨r,s⟩ use ⟨a*r,b*r+a*s⟩
    - the rule d(u*v)= u*dv+v*du
  - Where we see cos(⟨r,s⟩) use ⟨cos(r),-sin(r)*s⟩
    - the rule d(cos(u))=-sin(u)du
  - Etc.

Two ways to do forward AD

- Rewrite the program so that every variable is now TWO variables, var and var_x, the extra var_x is the derivative.
- Leave the "program" alone and overlay the meaning of all the operations like +, *, cos, with generalizations on pairs $\langle\alpha, \beta\rangle$

Does this work "hands off"? (Either method..)

- Mostly, but not entirely.
- Some constructions are not differentiable. Floor, Ceiling, decision points.
- Some constructions don't exist in normal fortran (etc.) e.g. "get the derivative" needed for use by
  - Newton iteration: $z_{i+1} := z_i - f(z_i)/f'(z_i)$
- Is it still useful? Apparently enough to build up a minor industry in enhanced compilers. see www.autodiff.org for comprehensive references
- If the programs start in Lisp, the transformations are clean and short. See www.cs.berkeley.edu/~fateman/papers/overload-AD.pdf

Where is this useful?

- A number of numerical methods benefit from having derivatives conveniently available: minimization, root-finding, ODE solving.
- Generalized to F(x,y,z), and partial derivatives of order 2, 3, (or more?)

What about reverse differentiation?

- The reverse mode computes derivatives of dependent variables with respect to intermediate variables and propagates from one statement to the <u>previous</u> statement according to the chain rule.
- Thus, the reverse mode requires a "reversal" of the program execution, or alternatively, a stacking of intermediate values for later access.

Reverse..

- Given independent vars **x** = $\{x_1, x_2, \ldots\}$
- And dependent vars **y** = $\{y_1(\mathbf{x}), y_2(\mathbf{x}) \ldots\}$, and also temporary vars…
- (read grey stuff upward…)
- A=f(x1,x2)
- B=g(A,x3) … we know d/dx1 of B is dG/d1 * dA/dx1
- R=A+B … we know d/dx1 of R is dA/dx1+dB/dx1, what are they?

Which is better?

- Hybrid methods have been advocated to take advantage of both approaches. Reverse has a major time advantage for many vars; it has the disadvantage of non-linear memory use– depends on program flow, loops iterations must be stacked.
- Numerous conferences on this subject.
- Lots of tools, substantial cleverness.