AD-like techniques in global optimization

A. Neumaier
University of Vienna
Austria

Reliable global optimization

requires not only access to function values, gradients and Hessians of a function at given points, but also a variety of properties conveying

global information

that can be constructed in an AD-like fashion from a computational graph:

- ranges of functions and constraints
- linear and nonlinear relaxations
- interval slopes
- etc.

Part of what we discovered in this context may also be useful in other contexts, such as

- solutions of nonlinear systems
- local optimization
- algorithm-independent modeling

In this talk, I'll emphasize the latter aspects.

Overview.

1. Slopes

- they generalize derivatives and AD
- they are more robust than derivatives, when applicable

2. FMathL

- A modeling and documentation language for mathematics, under development at our institute in Vienna

3. AD-like propagation methods

- Taylor series and generalizations
- Linear relaxations
- Bounds on variables
- and much else...

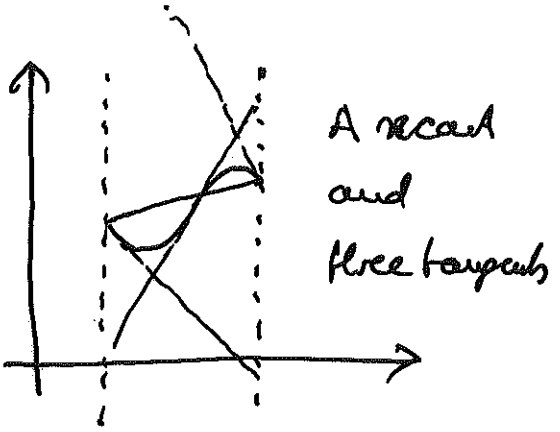
4. Global optimization on graphs

5. Inclusion- and Exclusion regions

for zeros of systems of equations

} (if time permits)

Slopes I



Slopes are a generalization of derivatives with <u>much</u> better global approximation properties.

There are problems whose <u>definition</u> already requires derivatives, such as bifurcation problems, finding stationary points, etc. Here derivatives are indispensible.

But often, slopes can be used in place of derivatives, and are much <u>more robust</u> in certain contexts.

Slopes can be computed cheaply by AD, and at the <u>same cost</u> as derivatives. + probably easy to adapt

Essentially all properties of derivatives have an extension to slopes. Proofs of these properties are even simpler (since no limit is needed).

Slopes show their real power in <u>global</u> problems:

- nonexistence of a solution
- existence proofs
- inclusion and exclusion regions

Slopes II

Slopes generalize well-known formulas such as

$$x^2 - z^2 = (x+z)(x-z)$$

$$\sqrt{x} - \sqrt{z} = \frac{x-z}{\sqrt{x}+\sqrt{z}}$$

to general multivariate, vector-valued functions $F: \mathbb{R}^n \to \mathbb{R}^m$.
A <u>slope</u> at $x \in \mathbb{R}^n$ is a matrix $F[z,x] \in \mathbb{R}^{m \times n}$ such that

$$F(x) = F(z) + F[z,x](x-z)$$

If F is continuously differentiable, one usually imposes the requirement of continuity. Then

$$F'(x) = F[x,x]$$

is the limiting slope at two equal arguments.

$f(x) = x^2$	$f[z,x] = x+z$	$f[x,x] = 2x = f'(x)$
$f(x) = \sqrt{x}$	$f[z,x] = \frac{1}{\sqrt{x}+\sqrt{z}}$	$f[x,x] = \frac{1}{2\sqrt{x}} = f'(x)$

The mean value theorem implies that (for c.d. F)

$$F[z,x] := \int_0^1 dt\, F'(z+t(x-z))$$

is a slope, but this is useless in practice.

For $n > 1$, slopes are <u>not unique</u>:

$$x_1x_2 - z_1z_2 = (x_2, z_2)\begin{pmatrix} x_1 - z_1 \\ x_2 - z_2 \end{pmatrix} = (z_2, x_1)\begin{pmatrix} x_1 - z_1 \\ x_2 - z_2 \end{pmatrix}$$

So $f(x) = x_1x_2 \Rightarrow F[x,z] = (x_2, z_1)$ or (z_2, x_1) are essentially different slopes

Slopes III

Slopes by finite differences:

$$F[z,x] = \frac{F(x) - F(z)}{z - x} \qquad \text{for } n = 1$$

requires 2 function values

(for $n > 1$ similar formulas with $n+1$ function values)

But:

Numerically unstable when $z \approx x$

$\Rightarrow$ not recommended

Slopes by symbolic manipulation:

by example: $F(x) = x + \sqrt{2x+1}$

$$F(x) - F(z) = x - z + \sqrt{2x+1} - \sqrt{2z+1}$$

$$= x - z + \frac{(2x+1) - (2z+1)}{\sqrt{2x+1} + \sqrt{2z+1}}$$

$$\Rightarrow F[z,x] = 1 + \frac{2}{\sqrt{2x+1} + \sqrt{2z+1}}$$

numerically stable

But:

expressions are unwieldy when F is complicated

(like in symbolic differentiation which results for $z = x$)

$\Rightarrow$ not recommended

Slopes by AD-like approach:

recommended

stable

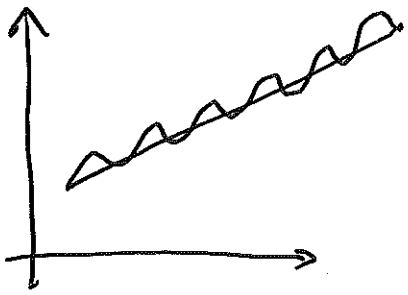
cheap (≤ 3 #Ops for f, for any n)

But needs some preparation ... (we first discuss uses, however)

Slopes IV

Why slopes?

- accurate at 2 points rather than only one
  ⇒ better global properties
- provides identities rather than truncated series
  ⇒ full control over errors
- has a smoothing effect
  ⇒ insensitive to small but high-frequency oscillations or discontinuities
  (as frequent in numerical adaptive codes)



whereas derivatives become completely meaningless!

Example $f(x) = e^{i\omega x}$

$$f[x,z] = \frac{e^{i\omega x} - e^{i\omega z}}{x-z} = e^{i\omega z}\,\frac{e^{i\omega(x-z)}-1}{x-z}$$

$$= e^{i\omega z}\, i\omega\, \exp_1(i\omega(x-z))$$

where $\exp_1(t) = \frac{e^t-1}{t}$ is analytic
(and easy to evaluate in terms of Padé approximants for small t, directly for larger t)

$\exp_1(0)=1 \Rightarrow f[x,x] = e^{i\omega x} i\omega = O(\omega)$

$f[z,x] = O(\omega)$ for $x-z = O(\omega^{-1})$ only
$f[z,x] = O(h^{-1})$ for $x-z = O(h)$, $\omega^{-1} \ll h$

Thus large frequencies ($\omega \gg 1$) affect the derivative but not the slope (except at very short distances where it is appropriate)

Slopes V

The secant method for $F(x)=0$ in $\mathbb{R}^n$

Define $N_z(x) := z - F[z,x]^{-1} F(z)$

If $F(\hat{x})=0$ then $F(z) = F(\hat{x}) - F[z,\hat{x}](\hat{x}-z)$
$= F[z,\hat{x}](z-\hat{x})$

therefore $F(\hat{x})=0 \Rightarrow N_z(\hat{x}) = \hat{x}$

If $z \approx \hat{x}$ then $N_z(x)$ is a contraction operator near $\hat{x}$

$\Rightarrow$ secant method $x_{k+1} = x_k - F[x_k, x_{k-1}]^{-1} F[x_k]$

converges superlinearly (near $\hat{x}$)

(with AD!) cost = cost of Newton's method
speed = slightly less (order 1.6, not 2)
but robustness increased

In practice, needs line search along $p_k = -F[x_k, x_{k-1}]^{-1} F[x_k]$

sparsity = same as Newton's method

Slopes VI

Quasi-Newton updates for optimization using slopes

For a quadratic function $f(x) = \gamma + c^T x + \frac{1}{2} x^T G x$ (G symmetric)

the gradient is $f'(x)^T = c + Gx$

while the slope is $f[z, x] = \left(c + \frac{1}{2} G(z+x)\right)^T$

so we have the quadratic model

$$f(x) = f(x_k) + f[x_k, x_{k-1}](x - x_k) + \frac{1}{2}(x - x_{k-1})^T G (x - x_k) + O(\|x - x_k\| \, \|x - x_{k-1}\| \, \|x - x_{k-2}\|)$$

where the error term is just a guess for nonquadratic functions (after enough updates)

From two successive slopes one can calculate the <u>quasi-Newton condition</u>

$$G(x_k - x_{k-2}) = 2\left(f[x_k, x_{k-1}] - f[x_{k-1}, x_{k-2}]\right)^T$$

so that the Hessian of a quadratic function can be fully accumulated from $n+2$ linearly independent search steps, using the standard SR1 update (and a trust-region approach)

The quasi-Newton step from x_k becomes the minimizer of the quadratic model, which is at

$$x_{k+1} = \frac{x_k + x_{k-1}}{2} - G^{-1} f[x_k, x_{k-1}]^T$$

if the trust region is not active

- no implementation yet
- finite termination on quadratics $\Rightarrow$ probably superlinear convergence
- should be insensitive to oscillations

slopes VII

composition formulas for slopes

$F = \alpha G \pm \beta H \Rightarrow F[z,x] = \alpha G[z,x] \pm \beta H[z,x]$

$F = GH \Rightarrow F[z,x] = G[z,x] H(z) + G(x) H[z,x]$
if G or H is scalar

$F = G/H \Rightarrow F[z,x] = \left(G[z,x] - F(z) H(z,x)\right) / H(x)$
if H is scalar

$F = G \circ H \Rightarrow F[z,x] = G[H(z),H(x)]\ H[z,x]$ chain rule

(Krawczyk & Neumaier 1985)

For multiplication and division, there is a second formula with (x) and (z) interchanged, which (for $n > 1$) typically gives different results. The same holds for the formula for implicit functions

F defined by $G(F(x),H(x)) = 0$ for $x \in z$
$\Rightarrow F[z,x] = -G[F(z),F(x)](H(x))^{-1} G(F(z))[H(z),H(x)] H(z,x)$

Here $G[u,v](w)$ is the slope of $G(u,w)$ w.r.t. u
$G(u)[v,w]$ " " " $G(u,v)$ w.r.t. v

This formula seems to be new, and implies application to slopes of solution of linear systems, inverses, and eigenvalue problems, which can all be stated as implicit functions

As observed by Blick (1992), $c^T F[z,x]$ can be propagated cheaply in a backward fashion, using the standard trick of AD for adjoint computations

$\Rightarrow$ slopes of scalar functions at the cost of 3 ops for f
slopes of arbitrary functions at same cost as derivatives

slopes VIII

Slopes for initial-value problems

The parameterized IVP

$$\dot{u} = F(u,x) \qquad \text{(generally vector-valued)}$$
$$u(0) = u_0(x)$$

defines $u(t)$ as a function of $t \in [0,T]$.

The slope $W = u[z,x]$ then satisfies the

variational equation

$$\dot{w} = F[u(z),u(x)](z)\, w + F(u(x))[z,x] \qquad \text{(and a second alternative formula)}$$
$$w(0) = u_0[z,x]$$

which can be used in forward mode to calculate the slope if there are few parameters

To get corresponding adjoint equations, we work

$$\langle c, u \rangle_t = \int_t^T dt\, c(t)^T u(t) + b^T u(T).$$

To compute the slope of the linear functional $\langle c, u \rangle_0$, we construct functions $v(t)$, $v_0(t)$ such that

$$\langle c, u(x) - u(z) \rangle_t = v(t)^T (u(x) - u(z))\Big|_t + v_0(t)^T (x - z)$$

Comparing the derivative w.r.t. t of both sides we see that we must satisfy

$$c^T (u(x) - u(z)) = \dot{v}^T (u(x) - u(z)) + v^T (\dot{u}(x) - \dot{u}(z)) + \dot{v}_0^T (x - z)$$

This is satisfied if the slope satisfies the equation

$$c^T w[x,x] = \dot{v}^T w[z,x] + v^T \dot{w}[z,x] + \dot{v}_0^T$$

Slopes VIII, ctd.

Using the variational equation, we can match the two sides if we solve the coupled adjoint equations (backward in time)

$$\dot{V} = c - F[u(t), u(x)](z)^T V$$

$$V(T) = b$$

and

$$\dot{V}_0 = -F(u(x))[z, x]^T V$$

$$V_0(T) = 0$$

(the initial conditions at $t = T$ are obvious.)

Slopes IX

Verifying existence or nonexistence of zeros

$F(x) = 0$, $x \in \mathbb{X}$ ($\mathbb{X}$ a nonempty, closed, convex set, in practice, a box: $\mathbb{X} = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^n$)

We know that any zero $\hat{x}$ is a fixed point of

$$N(x) = z - F[z, x]^{-1} F(z)$$

Using appropriate fixed-point theorems, we can conclude existence of a solution provided that a set theoretic closure condition is satisfied; typically the fixed point operator must map the region of interest into itself, and some other technical conditions must hold.

This requires bounds on ranges over sets, which are generally computed using interval arithmetic. In the simplest case, one just substitutes intervals for the variables and computes an enclosure. (If the result is pessimistic, a more sophisticated analysis is called for; there are a number of pitfalls that must be recognized and overcome.)

Using intervals in the computation of the slopes and solving (with an appropriate algorithm — not Gauss elimination!) the resulting linear interval equation, one gets an enclosure $N(\mathbb{X}) = z - F[z, \mathbb{X}]^{-1} F(z)$ of all images $N(x)$ with $x \in \mathbb{X}$.

Slopes IX, ctd.

$N(\mathbb{X})$ is called a Newton operator applied to the box $\mathbb{X}$.
Because of the fixed-point property of $N(x)$, the Newton operator has the important property

> Any zero $\hat{x} \in \mathbb{X}$ satisfies $\hat{x} \in N(x)$ and hence $\hat{x} \in N(\mathbb{X}) \cap \mathbb{X}$

Thus applying the Newton operator and intersecting with the box never loses a zero (at least with rigorously implemented outward-rounding interval arithmetic)

As a corollary,

> $N(\mathbb{X}) \cap \mathbb{X} = \emptyset \Rightarrow \mathbb{X}$ contains no zero

This (and its generalization) nonexistence result has no analogue in traditional numerical analysis – there is no way to verify nonexistence of solutions by traditional numerical methods (except with symbolic packages)

From the Kakutani fixed-point theorem (or the Leray-Schauder fixed point theorem), one can also deduce

> $N(\mathbb{X}) \subseteq \mathrm{int}\,\mathbb{X} \Rightarrow \mathbb{X}$ contains some zero

This existence result is valid rigorously (in a rigorous implementation) and is the basis of techniques for computer-assisted proofs in analysis

Slopes X

Interval arithmetic, together with associated existence and nonexistence results (of which we only showed one example each) is indispensable in reliable global optimization which is based on pruning the search space by discarding regions which are proved to contain no solution of the problem better than the best already known solution.

The global optimization package BARON is the current state of the art in reliable global optimization and outperforms stochastic (unreliable) search methods on problems up to a few hundred variables. (For larger problems, the search space is often so large that pruning takes too much time, and the extra guarantee of having found the solution is lost if the pruning process is stopped due to a time limit.)

The authors of BARON won in 2006 the Beale-Orchard-Hayes prize, the most prestigious prize awarded by the Mathematical Programming Society.

Slopes XI

The interval Newton method says that

$$\tilde{N}(X) = z - F'(X)^{-1} F(z)$$

has the same properties as the Newton operator discussed here. In fact it predates the work on slopes by about 20 years. Note that

$$F[z,X] \subseteq F[X,X] = F'(X)$$

so that the formula with the slope is always more accurate.

For example, if $F(x) = x^2$ and $x \in [1,3]$, $z = 2$

then $F'(X) = 2X = [2,6]$ has width 4

while $F[z,X] = z + X = [3,5]$ has only width 2

It can be proved that asymptotically (for narrow boxes) the width of $F[z,X]$ tends to half the width of $F'(X)$, while for larger boxes the gain is often (but not always) much more conspicuous.

These slopes are an essential improvement over derivatives in interval-based applications

Slopes XII

To compute interval slopes recursively, one needs enclosures for the slopes of the elementary functions φ. Kolev (1997) noticed that for convex and concave functions φ,

$$\varphi[z, X] = \square\{\varphi[z, \underline{X}], \varphi[z, \overline{X}]\}, \qquad (*)$$

where $\square$ denotes the interval hull. For other functions, a more complicated analysis is needed, but the final result is usually not much more costly than (*).

The real slopes may be computed by difference quotients (using directed rounding to get the outward enclosure property). However, for $z \approx \underline{X}$ or $z \approx \overline{X}$ this may give pessimistic bounds due to cancellation, followed by division by a small number. Therefore it is advisable to intersect the computed result from (*) with $\varphi'(X)$ which is an enclosure (and, as we have seen, overestimates by a factor of 2) but does not suffer from stability problems

Slopes XIII

Higher order derivatives can also be extended to slopes, using

$$f[x_1,x_2,x_3] = f[x_1][x_2,x_3],$$
$$f[x_1,x_2,x_3,x_4] = f[x_1,x_2][x_3,x_4], \text{ etc}$$

These <u>higher order slopes</u> appear as coefficients in the multivariate generalization of <u>Newton's interpolation formula</u>,

$$f(x) = f[x_1] + f[x_1,x_2](x-x_1) + \cdots + f[x_1,\cdots,x_n](x-x_1)\cdots(x-x_{n-1}) + f[x_1,\cdots,x_n,x](x-x_1)\cdots(x-x_n).$$

We have used successfully <u>second order slopes</u> for finding good inclusion and exclusion regions for zeros of nonlinear systems. (Schichl & Neumaier 2004)

This generalizes the existence and nonexistence results obtainable by (first order) slopes, and sometimes increases the region where an existence or nonexistence decision can be found by orders of magnitude.

Higher order slopes can be computed in an AD-like fashion in full analogy to the case of higher derivatives.

Slopes XIV

Some references

My textbook

Introduction to Numerical Analysis
Arnold Neumaier
Cambridge University Press 2001

covers the standard topics of such a book, together with forward and reverse automatic differentiation and interval techniques

(It is probably the only NA textbook covering A.D.)

My older text

Interval methods for systems of equations
Cambridge Univ. Press 1990

covers interval methods in depth but has only forward slopes (and nothing on the now important linear relaxation techniques discussed below).

The book can still be ordered via print-on-demand directly from C.U.P. (I still get royalties for about 30 copies each year.)

FMathL I

FMathL (= Formal Mathematical Language) is the working title for a modeling and documentation language for mathematics to be developed in a 3-year project at Vienna University which just started.

Thus all I say here is about plans, to be realized within the next few years.

The design of FMathL is based on our experience with

- modeling languages
- numerical programming systems
- test problem collections for various problem types
- real applications, and
- wrong starts in the past

FMath II

The goal of the FMathL project is to

- combine the advantages of LaTeX for writing and viewing mathematics,
- the user-friendliness of mathematical modeling systems such as AMPL for the flexible definition of large-scale numerical analyses,
- the universality of the mathematical language to describe completely arbitrary problems,
- the high-level discipline of the CVX system for solving convex programming problems and enforcing their semantic correctness, and
- the semantic clarity of the Z notation for the precise specification of concepts and statements.

We believe that this goal is reachable, and that an easy-to-use such system will change the way modeling is done in practice.

FMathL III

Key characteristics of the future language:

- Separate completely
  modeling aspects (declarative) and
  algorithmic aspects (imperative)
- The model is (almost) the documentation
  the latter can be automatically generated
  from the model in "publishable" quality
- The model is specified close to how it would
  be communicated informally when describing it
  in a lecture or paper (except for suppressed details)
- easy exchange of problems on a high level
- very transparent use, fully modular
- user extensible
- public domain
- mathematicians friendly
  (write as in a textbook...)
- any mathematical problem can be specified
  (though not necessarily passed to a solver)
  They later extensions only affect the interface
  not the problem structure, and can be done by software
  contributed by third parties

FMathL IV

Specifications in FMathL

- specify problems in their natural mathematical form with functions, sets, operations, measures, cases rather than loops, indices, branches
- specify algorithms by their requirements (input conditions, output conditions) rather than a piece of program
- specify strategies for combining algorithms to solve problems from a given category
- document which programs satisfy which algorithmic specifications
  - such that in principle (and perhaps in a following project) program verification is possible
- From these specifications, automatically create the structure needed to interface with existing programming systems (LAPACK, Matlab, --) and solvers (for optimization problems, ODEs, PDEs, etc), and with LaTeX for documentation.
- As part of the tasks, include automatic differentiation capabilities on the highest level:
  differentiate specifications rather than codes.

Propagation I

What can be propagated successfully through a computational graph?

Traditionally:

- (non)standard data types: scalar, double, complex, variable precision, interval, polynomial, etc.
- gradients and variants
  $F'(x)u$, $F'(x)^T v$, $f''(x)u$
- Taylor series and higher derivatives
- sparsity patterns
- work counts for all these tasks
- function data type (creating the DAG if no branches are encountered)

In addition, it is possible to propagate a large number of other information ...

Propagation II

Konstantad

easy to do in forward mode

- ranges (interval data type)
- norm bounds (for error estimation, quantitative)
  e.g. $\| \frac{d^k f}{dx^k} \|_\infty \le k! \, \|f\|_k$ where $(fg)_k = \sum f_j g_{k-j}$
  etc. in any dimension
- Lipschitz constants $\|F(x) - F(y)\| \le L \|x - y\|$
- bounds on linear functionals (integrals, high order derivatives, etc.) using Cauchy's integral theorem
  needs ranges on products of discs
  or products of boundaries of rectangles $\subseteq \mathbb{C}$

easy to do in backward mode

- error bounds for approximations, e.g. Taylor series
- asymptotic expansions
  bounds on unbounded domains
- linear, quadratic, or convex relaxations
  also more general structured relaxations such as
  separable quadratic, semiseparable (= quadratic + separable), biseparable (= sum of bivariates)
- convexity determination
  Fourer: Dr. AMPL ; Ye + Wright: CVX
- slopes

Notes:
- overestimation problem:
  on $[-1,1]$, Lipschitz const of $x - x$ is 2
  range of $x - x$ is $[-2, 2]$
- quality / time tradeoff
  - work harder to reduce overestimation
  - extra effort for payoff in rounding

Propagation III

The derivation of formulas in the nonstandard case becomes fairly easy once one has seen the basic underlying principle. We begin with a traditional backward evaluation to motivate the principle.

Why AD-generated gradients are cheap

Standard evaluation of $f(x)$ at an arbitrary point x gives the values of all intermediate results y_i (where $y_{1:n} = x$) and as final result $y_N = f(x)$.
We can write $f(x)$ in many ways as a function of the intermediate results $y_{1:N}$, for example as

$$f(x) = f_N(y) := y_N.$$

Recursively, we can get a new representation $f_{k-1}(y)$ of $f(x)$ by substituting the defining relation for y_k into $f_k(y)$.
Clearly, $f_k(y)$ depends only on $y_{1:k}$.
In particular, $f_n(y)$ depends only on $y_{1:n} = x$, and indeed, it is the original expression for f as a function of x.

Now suppose that we know the intermediate results y_i^0 for an evaluation at x^0. Then we can approximate each $f_k(y)$ as a truncated Taylor series $t_k(y)$ centered around y^0. The substitution process $f_k \to f_{k-1}$ then clearly amounts to getting $t_{k-1}(y)$ from $t_k(y)$ by substituting the defining relation for y_k into $t_k(y)$ and Taylor expanding the few terms affected by this substitution.

In particular, to get gradients, a linear Taylor expansion suffices. In this case, the only term affected by the substitution is a term $c_k y_k$, which after substitution can be linearized trivially with $O(N) \le 3N$ operations. Therefore the whole substitution process computes $t_n(y) = f(y^0) + f'(x^0)(x - x^0)$ and hence the gradient in $O(N) \le 3N$ operations, where N is the number of operations to get $f(x)$ in the first place. Thus gradients are cheap.

To get Hessians, one needs a quadratic Taylor expansion $\gamma + c^T y + \frac{1}{2}(y-y^0)^T G (y-y^0)$ [illegible] Now the terms to be substituted are of the form $c_k y_k$, $G_{kk} y_k y_k$ ($k < k$) and $\frac{1}{2} G_{kk} y_k^2$. Although typically only a few of these terms are nonzero, one sees that the worst case complexity is significantly worse than $O(N)$.

Propagation IV

Using Newton-polynomials in place of truncated Taylor polynomials, an identical procedure produces slopes in the linear case, and higher-order slopes in the case of Newton polynomials of higher degree.

Only the details of the expansion change!

Once the generality of the principle is recognized, one can apply it to generate all sorts of desired information.

We show how to obtain a linear relaxation of $f(x)$ on a box X, i.e., a linear function $r(x)$ such that

$$r(x) \le f(x) \text{ for all } x \in X.$$

In an initial forward pass, we compute enclosures Y_i for the ranges of all intermediate results y_i. (Actually, this can be improved using constraint propagation, see below.)

We construct $r(x)$ as $r_n(y)$, where $r_N(y) = y_N$, and each $r_{i-1}(y)$ is obtained from $r_i(y)$ by substituting one of the and subsequent linear relaxation. As in the case of gradient and slope, the cost is $O(N)$ per step, since we only need to relax the term obtained by substituting into $c_i \diamond y_k$. Thus linear relaxation is cheap.

To be able to program this, we just need to have symbolic linear relaxations for a few expressions, namely

$$\pm(u \circ v) \text{ for } \circ = +, -, *, /, \wedge, \text{atan2}$$
$$\pm\varphi(u) \text{ for } \varphi = \text{sqrt}, \exp, \log, \sin, \cos, \ldots$$

Side conditions for the (nonunique) relaxation of these basic expressions are

either a minimal worst case error on the rectangular domain of u, v, resp. the interval for u,

or a minimal error at a target point u^0, v^0, obtained by a forward evaluation.

Figuring out an optimal relaxation is now in each case a straightforward exercise.

Propagation V

Constraint propagation (CP)

The propagation of ranges can be done in more sophisticated ways when the application is a global problem (for finding a feasible point of a constraint satisfaction problem, or an optimal point of a global optimization problem).

The reason is that in this case the constraints on the results of a function, or an upper bound on the objective given by the best known feasible point, can be used to improve the ranges of intermediate results — though they might be outside the improved ranges for an arbitrary input in the given box, they must be within the range for any feasible input, and only the latter counts.

There are some range information propagated backwards from results to intermediates and ultimately to the original variables. This process is called constraint propagation.

On a computational dag, we have always relations of the form

$z = x \circ y$ or $z = \varphi(x)$

Forward evaluation of ranges requires bounds on the given bounds on x and y, and is standard interval arithmetic, together with intersection.

Backward evaluation of ranges requires bounds on x or y given bounds on z (and y or x), and is interval arithmetic on the corresponding inverse function. Complications arise because of the multi-valuedness and because of non-invertibility in a number of cases ($\varphi = \sin$, resp. $\circ = *$), but correct formulas are known in each case.

Ranges that improved significantly are propagated again, also in the forward direction, resulting in a somewhat chaotic propagation scheme that repeatedly traverses the relevant regions of the DAG until improvements become insignificant and a near fixed-point is reached.

Higher consistency techniques, based on "slacks" and other tricks are more expensive but give better results on compound expressions.

Linear relaxation

We discussed already how to create linear relaxation by a backward AD-like scheme.

Alternatively, and realized, e.g., in BARON, are linear relaxation techniques that work directly on each relation of the DAG rather than on the original variables only.

BARON (following McCormick (1976)) uses optimal linear inequalities for the standard operations,

e.g. $z = xy$, $x \in [\underline{x}, \overline{x}]$, $y \in [\underline{y}, \overline{y}]$

$\Rightarrow z \geq \underline{x}y + x\underline{y} - \underline{x}\underline{y}$ since the difference is $(x-\underline{x})(y-\underline{y}) \geq 0$,

and three other such inequalities are obtained from $(x-\underline{x})(y-\overline{y}) \leq 0$

$(x-\overline{x})(y-\underline{y}) \leq 0$

$(x-\overline{x})(y-\overline{y}) \geq 0$

Refined analysis for particular compound operations occurring frequently in chemical engineering (the original target of BARON), such as xyz, xy/z, etc. yields better linear inequalities involving more variables.

Each analysis of a compound expression is a symbolic global optimization problem in few variables which, once solved, can be incorporated into the solver for any software containing such a compound subexpression.

However, these symbolic problems get harder with the complexity (rapidly) of the compound.

The family of all (linear) relaxations generated is then made the basis of a linear program whose solution yields selected ranges, frequently with extraordinary success compared to pure CP. The cost of solving these large scale LPs is amortized in BARON by the drastic reduction in the size of the branch-and-bound tree that needs to be considered.

Optimization problems as graphs

$\min_{\max} f(x)$ equalities $\hat{=} \in [0,0]$

s.t. $x \in \mathbb{X}$, $F(x) \in \mathbb{F}$ inequalities $\hat{=} \in [0,\infty]$

Introducing variables for slacks and objective

$\min \pm x_n$ with more variables

s.t. $x \in \mathbb{X}$, $F(x) = 0$

Enforcing sparse Jacobians and Hessians by substitution and tearing

$$f(x) = \sum \varphi_i(x_i) + \left(\sum \psi_i(x_i)\right)^2$$ dense Hessian

$$u = \sum \varphi_i(x_i)$$
$$v = \sum \psi_i(x_i)$$
$$w = u + v \; [= f(x)]$$

} sparse representation: Hessian of Lagrangian is diagonal

Dense rows in Jacobians kill in the least square Hessian term in penalty formulations and interior point codes:
The Hessian of $\frac{1}{2}\|Ax-b\|^2$ involves A^TA which is a dense matrix if A has a dense row: $| \cdot \cdots = $ ▦

Tearing splits an equation

(*) $\sum f_i(x_{I_i}) = 0$ leading to a dense row

into $\sum u_i = 0$, $f_i(x_{I_i}) - u_i = 0$

which is sparse.

Figuring out the optimal way to tear a constraint into the form (*) is nontrivial

Relation graphs I

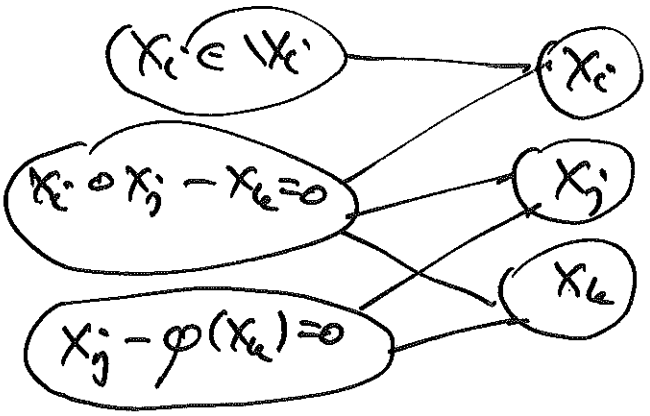
Carrying the substitution process to the extreme ends in an optimization problem where every intermediate term is a variable and all constraints are simple ones, binary or ternary relations of the form

$$x_i \in \mathbb{X}_i$$
$$\varphi(x_i) - x_j = 0$$
$$x_i \circ x_j - x_k = 0$$

Essentially we get an undirected version of the computational graph.

Substitution and its converse, elimination, may then be viewed as transformation of an undirected relation graph, consisting of the union of all edges defined on the disjoint vertex set with relations and variables as vertices, and edges between a variable and a relation if the variable appears in the relation.

E.g.,



By interpreting a variable node as an equality relation between all incoming edges, we can think of all vertices as relations, and all edges as variables

Relation graphs II

In this interpretation (vertices = relations, edges = variables) all transformations involving substitution and/or elimination can be accommodated very naturally.

By focusing on certain types of canonical relations one can bring any optimization problem (with certain restrictions) into a canonical form most useful for applications.

For example, GloptLab (a Matlab global solver under development) requires a canonical form consisting of quadratic constraints only. Any algebraic problem can be transformed to this form by removing divisions and removing integer (or rational) powers and roots

e.g.. $x/y = z \iff x = yz,\ y \neq 0$

$\sqrt{x} = y \iff x = y^2,\ y \geq 0$

Also many trigonometric problems can be transformed in this way

$$\left.\begin{array}{l} x = r\cos\varphi \\ y = r\sin\varphi \\ r \geq 0,\ \varphi \in [0, 2\pi[\end{array}\right\} \iff \left\{\begin{array}{l} x^2+y^2 = r^2 \\ \varphi = \operatorname{atan2}(x,y) \end{array}\right.$$

$$\left.\begin{array}{l} x = \cos 2\varphi \\ y = \sin(\varphi+\alpha) \\ z = \cos\alpha \end{array}\right\} \iff \left\{\begin{array}{l} x = c^2 - s^2, \quad c^2+s^2 = 1 \\ y = sw + cz, \quad w^2+z^2 = 1 \end{array}\right.$$

if φ, α are eliminated

But $\varphi + \sin\varphi = 0$ is not reducible to an algebraic problem.

Problem

$$
\begin{aligned}
\min\ & f = (4x_1 - x_2x_3)(x_1x_2 + x_3), \\
\text{s.t.}\ & x_1^2 + x_2^2 + x_1x_2 + x_2x_3 + x_2 = 0, \\
& \exp(x_1x_2 + x_2x_3 + x_2 + \sqrt{x_3}) \in [-1, 1], \\
& x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \in [-1, 8].
\end{aligned}
$$

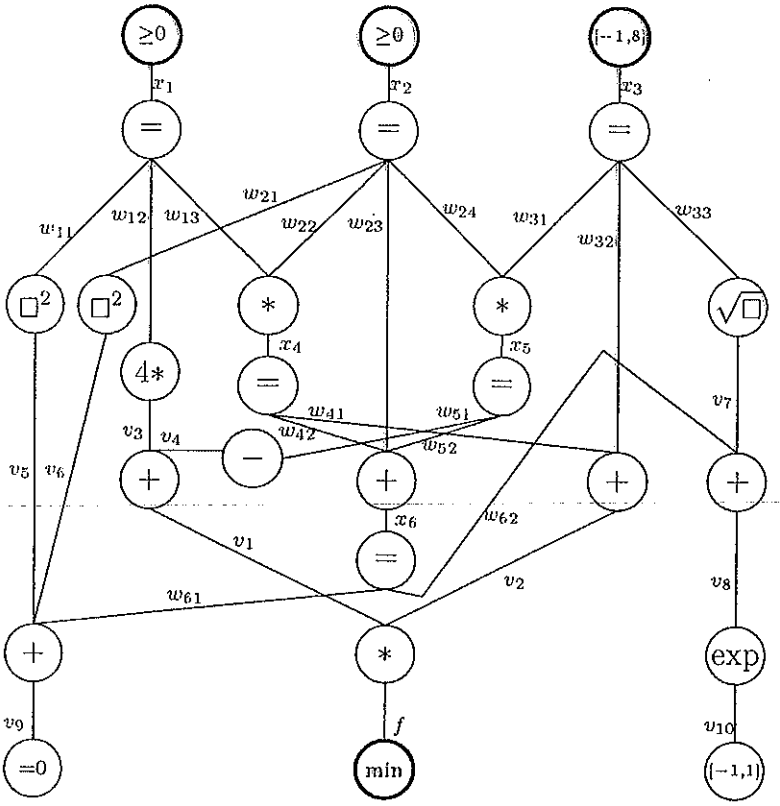
Codelist

$$
\begin{aligned}
\min\ & f \\
\text{s.t.}\ & f = v_1v_2 \\
& v_1 = v_3 + v_4 \\
& v_2 = x_4 + x_3 \\
& v_3 = 4x_1 \\
& v_4 = -x_5 \\
& v_5 = x_1^2 \\
& v_6 = x_2^2 \\
& v_7 = \sqrt{x_3} \\
& v_8 = x_6 + v_7 \\
& v_9 = v_5 + v_6 + x_6 \\
& v_{10} = \exp(v_8) \\
& x_4 = x_1x_2 \\
& x_5 = x_2x_3 \\
& x_6 = x_2 + x_4 + x_5 \\
& x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \in [-1, 8] \\
& v_9 = 0, \quad v_{10} \in [-1, 1].
\end{aligned}
$$

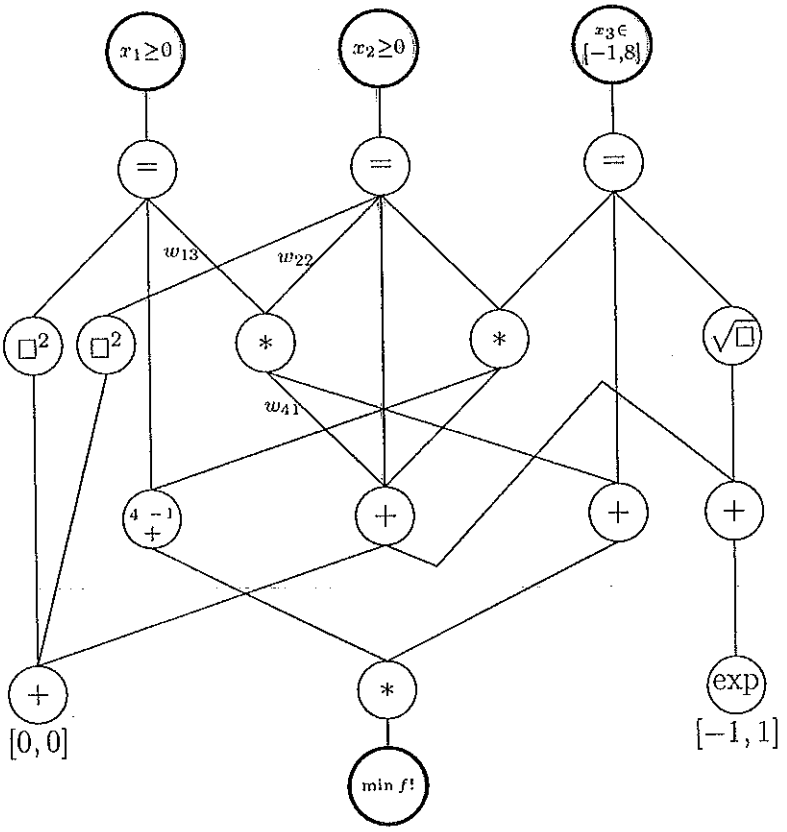
relaxed

$$
\begin{aligned}
\min\ & f \\
\text{s.t.}\ & x_1 = w_{11} = w_{12} = w_{13} \\
& x_2 = w_{21} = w_{22} = w_{23} = w_{24} \\
& x_3 = w_{31} = w_{32} = w_{33} \\
& x_4 = w_{41} = w_{42} \\
& x_5 = w_{51} = w_{52} \\
& x_6 = w_{61} = w_{62} \\
& f = v_1 v_2 \\
& v_1 = v_3 + v_4 \\
& v_2 = w_{41} + w_{32} \\
& v_3 = 4 w_{12} \\
& v_4 = -w_{51} \\
& v_5 = w_{11}^2 \\
& v_6 = w_{21}^2 \\
& v_7 = \sqrt{w_{33}} \\
& v_8 = w_{62} + v_7 \\
& v_9 = v_5 + v_6 + w_{61} \\
& v_{10} = \exp(v_8) \\
& x_4 = w_{13} w_{22} \\
& x_5 = w_{24} w_{31} \\
& x_6 = w_{23} + w_{42} + w_{52} \\
& x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \in [-1, 8] \\
& v_9 = 0, \quad v_{10} \in [-1, 1].
\end{aligned}
$$

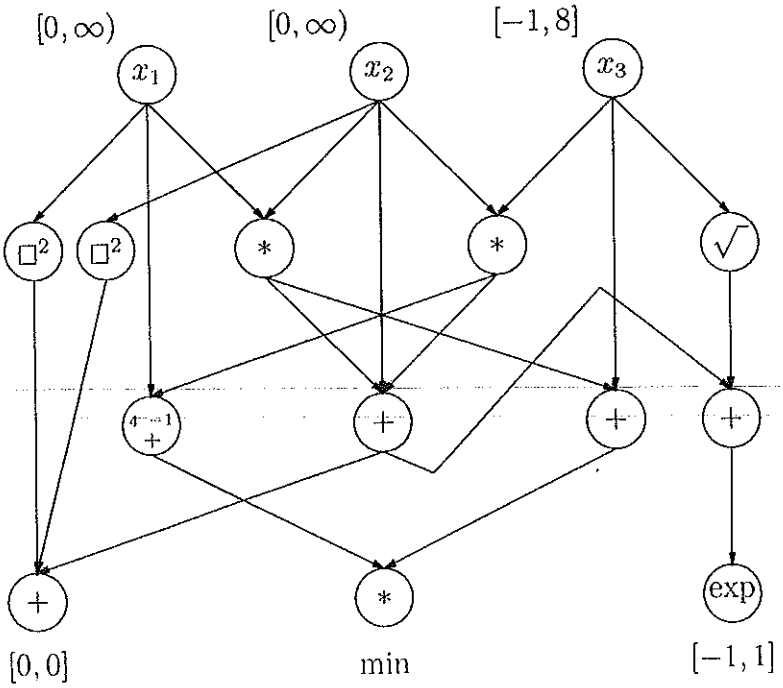
relation graph



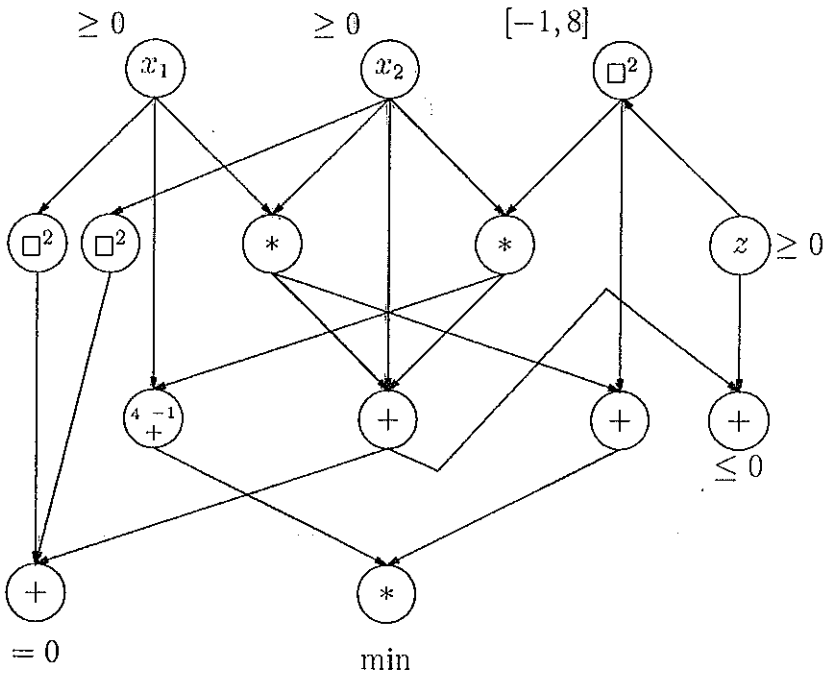
simplified



Oriented as DAG



remove sqrt (different DAG)

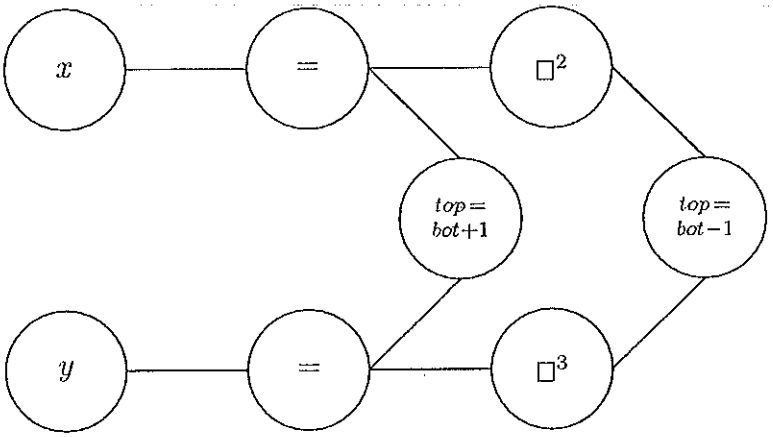


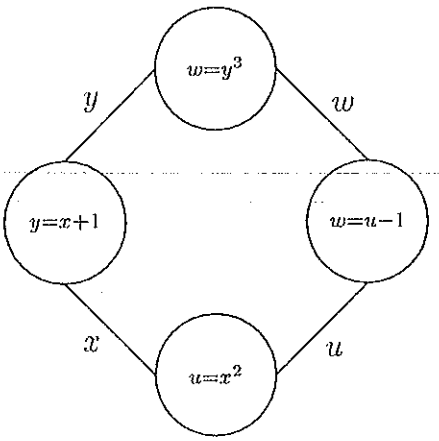
print problem

$$
\begin{aligned}
\min\ & f = (4x_1 - x_2 z^2)(x_1 x_2 + z^2), \\
\text{s.t.}\ & x_1^2 + x_2^2 + x_1 x_2 + x_2 z^2 + x_2 = 0, \\
& x_1 x_2 + x_2 z^2 + x_2 + z \le 0, \\
& x_1 \ge 0, \quad x_2 \ge 0, \quad z^2 \in [-1, 8], \quad z \ge 0.
\end{aligned}
$$

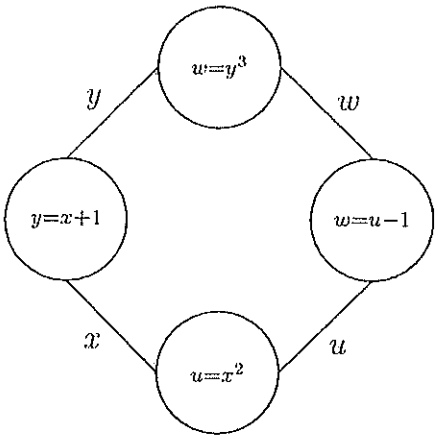
Graph transformation

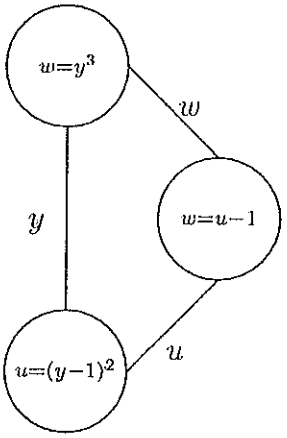
$$y^3 = x^2 - 1$$
$$y = x + 1$$

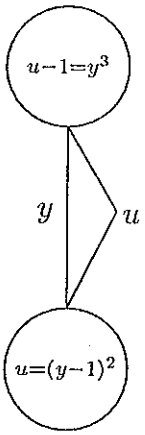


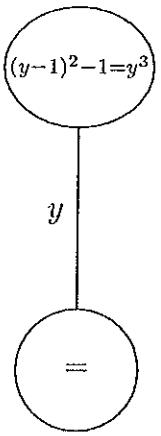


elimination of x





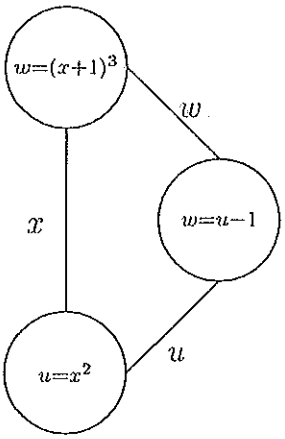


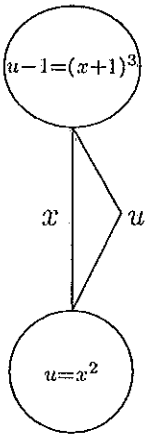


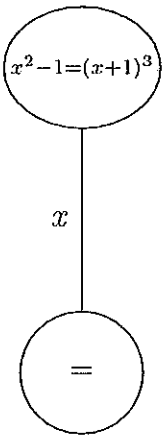
print problem

$$y^3 = (y-1)^2 - 1$$

alternative elimination of 4







$$x^2 = (x+1)^3 - 1$$

Relation graphs III

In a relation graph representation in which all intermediate results are made to variables, differentiation is trivial, as is in the more compact representation by

semiseparable constraints

$$\frac{1}{2} x^T G x + \sum \varphi_i(x_i) = 0$$

(each φ_i a univariate expression)

What happened to the A.D concerns?
The problems of AD on the original problem simply transmuted into problems of sparse matrix analysis!

To see this, note that a DAG evaluation of $z = F(x)$ is nothing else than a solution of the extended sparse triangular nonlinear system $E(x,y) = 0$ for the vector y of intermediate variables, and the projection $z = Py$ which extracts the results from y. By construction, the Jacobian $J = \frac{\partial E}{\partial y}(x,y)$ of this extended system is extremely sparse and lower triangular. Differentiating, we find $\frac{\partial E}{\partial x} + \frac{\partial E}{\partial y}\frac{\partial y}{\partial x} = 0$, hence

$$F'(x) = \frac{\partial z}{\partial x} = P\frac{\partial y}{\partial x} = -P\underbrace{\left(\frac{\partial E}{\partial y}\right)^{-1}}_{\text{backward Jacobian (adjoint)}}\underbrace{\frac{\partial E}{\partial x}}_{\text{forward Jacobian}} \qquad (*)$$

The formula $(*)$ is just the Schur complement of the sparse matrix

$$\begin{pmatrix} \frac{\partial E}{\partial y} & \frac{\partial E}{\partial x} \\ P & 0 \end{pmatrix}$$

Thus AD is just the art of computing certain sparse Schur complements in an optimal way!

Relation graphs IV

Now a Jacobian (or Hessian, which naturally transmute into Schur complement computations) are never an end in itself.
In (local) optimization problems, they are just used to find good search directions or new trial points, by using them in the context of (the) linear algebra. Thus viewing the Jacobian computation as part of the full problem only may introduce extra flexibility.

This flexibility is naturally provided by the relation graph representation, where the differentiation part and the combinatorial handling part are seamlessly integrated.

For example, the AD problem of scarcity is naturally transmuted into the graph problem of scalarizing out unwanted dense pieces, and handling linear pieces within the graph in a special fashion.

On the relation graph level, AD techniques and sparse matrix technology are likely to complement each other in a natural way, and it will be interesting to explore this connection more deeply.

Relation graphs V

The problem transformations enabled by the relation graph framework in a natural and automatic way may even make the difference between a tractable and an intractable problem.

A case demonstrating this is the solution (negative) of Wilkinson's conjecture, stating that the growth factor in Gaussian elimination with complete pivoting on an $n \times n$-matrix is bounded by n. (In fact it grows beyond this conjectured bound)

This is an unconstrained optimization problem, but in this form the problem is intractable for values such as $n = 12$. (The number of variables is n^2.)
By introducing variables for all intermediate elimination values, the problem size increases ($O(n^3)$ variables), but it becomes sparse. Although now a constrained problem, the solution for $n = 12$ was found easily by Conn, Gould and Toint with their (at that time) new LANCELOT optimization package. The optimal value was slightly above n, thus refuting the conjecture.

Upon analysis the deeper reason for the success reveals that the expanded constrained problem is much less sensitive to perturbations than the original problem.

In general, scaling and stability issues are easier handled in the sparser formulations.