Third Order Methods using slices of the Tensor and AD developments

Robert Gower
collaborators: **Artur Gower**, Margarida P Mello
my supervisor: Jacek Gondzio

Edinburgh Research Group in Optimization

gowerrobert@gmail.com

What's to come

- Third Order information can be used in practical nonlinear solvers
  - Automatic Differentiation (AD) methods that calculate third-order information at the same cost of the Hessian.
  - A family of third order methods that requires solving two linear systems.
  - Large-Scale tests comparing to Newton

Overview

Why not third order

Unconstrained minimization of $f : \mathbb{R}^n \to \mathbb{R}$.

First-order

$$d = -\alpha Df(x)$$

$$(Df(x) \equiv \text{The gradient})$$

Second-order

$$D^2 f(x) \cdot d + Df(x) = 0$$

$$(D^2 f(x) \equiv \text{The Hessian matrix})$$

$$n(x) = -(D^2 f(x))^{-1} Df(x).$$

Why stop here?

Why not third order

Unconstrained minimization of $f : \mathbb{R}^n \to \mathbb{R}$.
First-order

$$d = -\alpha Df(x)$$

$$(Df(x) \equiv \text{The gradient})$$

Second-order

$$D^2 f(x) \cdot d + Df(x) = 0$$

$$(D^2 f(x) \equiv \text{The Hessian matrix})$$

$$n(x) = -(D^2 f(x))^{-1} Df(x).$$

Why stop here? It's hard to solve these n

$$\frac{1}{2} D^3 f(x) \cdot (d, d) + D^2 f(x) \cdot d + Df(x) = 0.$$

Can we get third order convergence with **only** linear systems?

Can we get third order convergence with **only** linear systems?

Halley's Method

$$(D^2 f(x)) \cdot d + Df(x) = 0.$$

Can we get third order convergence with **only** linear systems?

Halley's Method

$$\Big(\underbrace{\tfrac{1}{2}D^3 f(x)\cdot n(x)}_{\text{A matrix}} + D^2 f(x)\Big)\cdot d + Df(x) = 0.$$

Can we get third order convergence with **only** linear systems?

Halley's Method

$$\Big(\underbrace{\frac{1}{2}D^3 f(x)\cdot n(x)}_{\text{A matrix}} + D^2 f(x)\Big)\cdot d + Df(x) = 0.$$

Chebyshev's Method

$$D^2 f(x)\cdot d + Df(x) = 0.$$

Can we get third order convergence with **only** linear systems?

Halley's Method

$$(\underbrace{\frac{1}{2}D^3f(x)\cdot n(x)}_{\text{A matrix}}+D^2f(x))\cdot d+Df(x)=0.$$

Chebyshev's Method

$$D^2f(x)\cdot d+Df(x)+\underbrace{\frac{1}{2}D^3f(x)\cdot(n(x),n(x))}_{\text{A vector}}=0.$$

Can we get third order convergence with **only** linear systems?

Halley's Method

$$(\underbrace{\frac{1}{2}D^3 f(x) \cdot n(x)}_{\text{A matrix}} + D^2 f(x)) \cdot d + Df(x) = 0.$$

Chebyshev's Method

$$D^2 f(x) \cdot d + Df(x) + \underbrace{\frac{1}{2}D^3 f(x) \cdot (n(x), n(x))}_{\text{A vector}} = 0.$$

Why exactly these red pieces? Order 3 local convergence

Convex $\lambda \in [0,1]$ combinations Halley-Chebyshev family

$$(1-\lambda)\left(\tfrac{1}{2}D^3f(x)\cdot n(x) + D^2f(x)\right)\cdot d + Df(x)\Big) = 0$$

$$+$$

$$\lambda\left(D^2f(x)\cdot d + Df(x) + \tfrac{1}{2}D^3f(x)\cdot (n(x))^2\right) = 0$$

$$=$$

$$\left(D^2f(x) + \frac{(1-\lambda)}{2}D^3f(x)\cdot n(x)\right)\cdot d + Df(x) + \frac{\lambda}{2}D^3f(x)\cdot (n(x))^2 = 0$$

Convex $\lambda \in [0,1]$ combinations Halley-Chebyshev family

$$(1-\lambda)\left(\tfrac{1}{2}D^3f(x)\cdot n(x) + D^2f(x)\right)\cdot d + Df(x)\Big) = 0$$

$$+$$

$$\lambda\left(D^2f(x)\cdot d + Df(x) + \tfrac{1}{2}D^3f(x)\cdot(n(x))^2\right) = 0$$

$$=$$

$$\left(D^2f(x) + \frac{(1-\lambda)}{2}D^3f(x)\cdot n(x)\right)\cdot d + Df(x) + \frac{\lambda}{2}D^3f(x)\cdot(n(x))^2 = 0$$

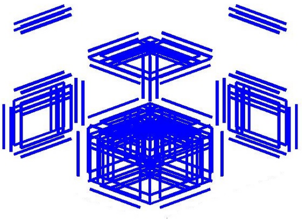
- Halley-Chebyshev family [Gutierrez, 1997]
- Implicit form [Steihaug, 2012]
- Convex combination ⇒ Order-3 convergence (My homepage)
- Higher order generalizations possible!

Handling third order derivative

Problem: $D^3 f(x)$ is cube.

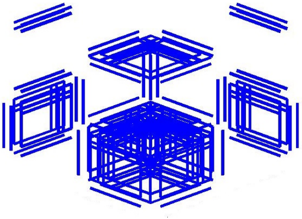
Handling third order derivative

Problem: $D^3 f(x)$ is cube.



Handling third order derivative

Problem: $D^3 f(x)$ is cube.

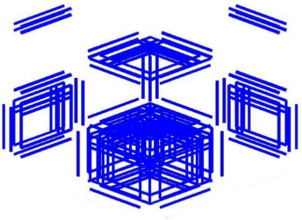


Current approach (Gundersen and Steihaug 2012):

- Data structures that balance Sparsity $\times$ Access time.
- Faster contractions $D^3 f(x) \cdot n(x)$.

Handling third order derivative

Problem: $D^3 f(x)$ is cube.



Current approach (Gundersen and Steihaug 2012):

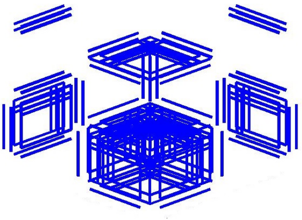
- Data structures that balance Sparsity $\times$ Access time.
- Faster contractions $D^3 f(x) \cdot n(x)$.

But we only need

$$\frac{d}{dt} D^2 f\left(x + t \cdot n(x)\right)|_0 \\ = D^3 f(x) \cdot n(x).$$

Handling third order derivative

Problem: $D^3 f(x)$ is cube.



Current approach (Gundersen and Steihaug 2012):

- Data structures that balance Sparsity $\times$ Access time.
- Faster contractions $D^3 f(x) \cdot n(x)$.

But we only need

$$\frac{d}{dt} D^2 f\,(x + t \cdot n(x))\,|_0 \\ = D^3 f(x) \cdot n(x).$$

Automatic **D**ifferentiation solution.

Why High Order AD?

- High order optimization methods
- Calculating quadratures (Vinay Kariwala 2012, G. F. Corliss, A. Griewank 1997)
- bifurcations and periodic orbits (J. Guckenheimer and B. Meloon 2000)
- Classifying Degenerate singularities and equilibria.

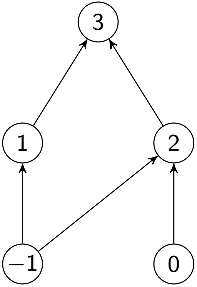
- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$

$$f(h(x_{-1}), g(x_{-1}, x_0))$$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$

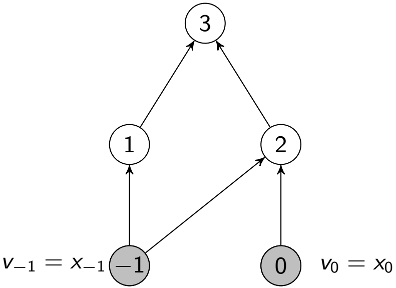
$$
\begin{aligned}
&f(h(x_{-1}), g(x_{-1}, x_0)) \\
&v_{-1} = x_{-1} \\
&v_0 = x_0 \\
&v_1 = h(v_{-1}) \\
&v_2 = g(v_{-1}, v_0) \\
&v_3 = f(v_2, v_1)
\end{aligned}
$$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.



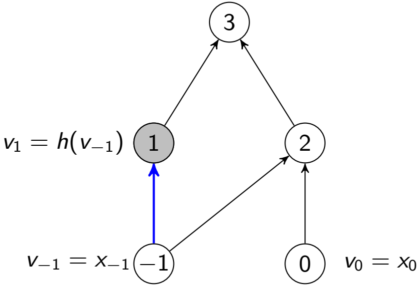
$$f(h(x_{-1}), g(x_{-1}, x_0))$$
$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = h(v_{-1})$$
$$v_2 = g(v_{-1}, v_0)$$
$$v_3 = f(v_2, v_1)$$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.



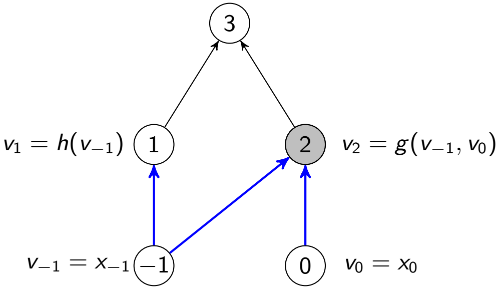
$$
\begin{aligned}
&f(h(x_{-1}), g(x_{-1}, x_0))\\
&v_{-1} = x_{-1}\\
&v_0 = x_0\\
&v_1 = h(v_{-1})\\
&v_2 = g(v_{-1}, v_0)\\
&v_3 = f(v_2, v_1)
\end{aligned}
$$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.



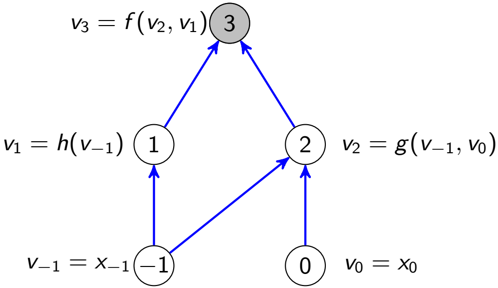
$f(h(x_{-1}), g(x_{-1}, x_0))$
$v_{-1} = x_{-1}$
$v_0 = x_0$
$v_1 = h(v_{-1})$
$v_2 = g(v_{-1}, v_0)$
$v_3 = f(v_2, v_1)$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.



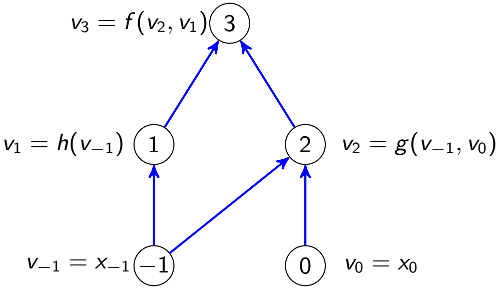
$$
\begin{aligned}
&f(h(x_{-1}), g(x_{-1}, x_0)) \\
&v_{-1} = x_{-1} \\
&v_0 = x_0 \\
&v_1 = h(v_{-1}) \\
&v_2 = g(v_{-1}, v_0) \\
&v_3 = f(v_2, v_1)
\end{aligned}
$$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.



$$f(h(x_{-1}), g(x_{-1}, x_0))$$
$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = h(v_{-1})$$
$$v_2 = g(v_{-1}, v_0)$$
$$v_3 = f(v_2, v_1)$$

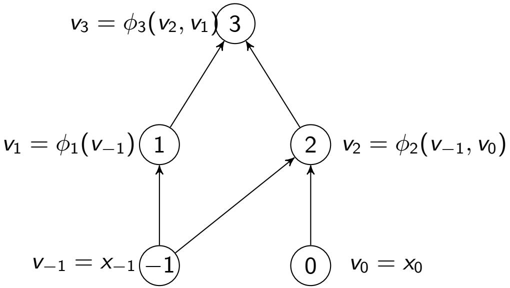
- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.



$f(h(x_{-1}), g(x_{-1}, x_0))$
$v_{-1} = x_{-1}$
$v_0 = x_0$
$v_1 = h(v_{-1})$
$v_2 = g(v_{-1}, v_0)$
$v_3 = f(v_2, v_1)$

- Indices of matrices and vectors shifted by $-n$.
  $y \in \mathbb{R}^m$: $y = (y_{1-n}, \ldots, y_{m-n})^T$
- Unravel function into simpler functions.
- Node numbering is in order of evaluation.
- (j is a predecessor of i) $\equiv j \in P(i)$.

Millions of nodes are common (This one has just 150)



$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

- Standardize function names ϕ_i
- In general case might have many *intermediate functions*

Standardized Function Evaluation

Input: $v_{i-n} = x_{i-n}$, for $i = 1, \dots n$
for $i = 1 \dots \ell$ **do**
$\quad v_i = \phi_i(v_{P(i)})$
end
Output: $f(x) = v_\ell$

- Nodes for *Independent variables:*
$$v_{i-n} = x_{i-n}, \quad \text{for } i = 1, \dots, n$$
- Nodes for *Intermediate variables:*
$$v_i = \phi_i(v_{P(i)}), \quad \text{for } i = 1, \dots, \ell.$$

Each ϕ_i a *elemental* function with derivatives coded.
AD packages transform users functions to standard form.

Differentiating standardized function

- How do we differentiate our Standardized function?
- How do we differentiate an algorithm?

Differentiating standardized function

- How do we differentiate our Standardized function?
- How do we differentiate an algorithm?
- Solution: represent as a composition of operators.
- We know how to differentiate operators.

State transformation

Make an operator that calculates a single node

State transformation

Make an operator that calculates a single node
Big vector of all values

State transformation

Make an operator that calculates a single node
Big vector of all values

$$v := (v_{1-n}, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_\ell)$$

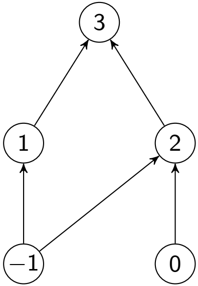
State transformation

Make an operator that calculates a single node
Big vector of all values

$$v := (v_{1-n}, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_\ell)$$

The ith State Transformation (Griewank)

$$\begin{aligned}\Phi^i : \mathbb{R}^{n+\ell} &\to \mathbb{R}^{n+\ell}, \\ v &\mapsto (v_{1-n}, \dots, v_{i-1}, \phi_i(v_{P(i)}), v_{i+1}, \dots, v_\ell),\end{aligned}$$



$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

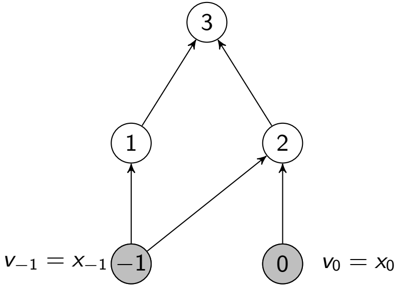
$$v_{-1} = x_{-1}$$

$$v_0 = x_0$$

$$v_1 = \phi_1(v_{-1})$$

$$v_2 = \phi_2(v_{-1}, v_0)$$

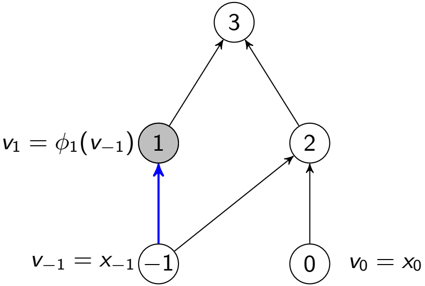
$$v_3 = \phi_3(v_2, v_1)$$



$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$
$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

$$\mathcal{I}x$$

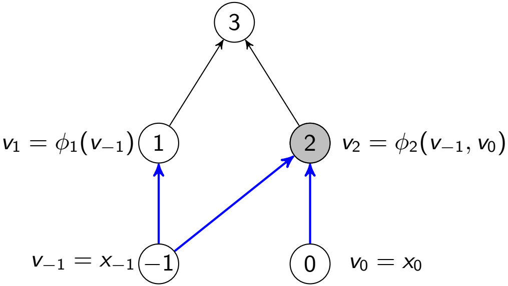
$$(v_{1-n}, v_0, 0, 0, 0)$$



$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$
$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

$$\Phi^1 \circ \mathcal{I}x$$

$$(v_{1-n}, v_0, v_1, 0, 0)$$

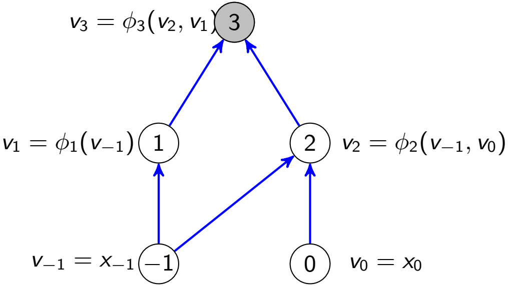


$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

$$\Phi^2 \circ \Phi^1 \circ \mathcal{I}x$$

$$(v_{1-n}, v_0, v_1, v_2, 0)$$

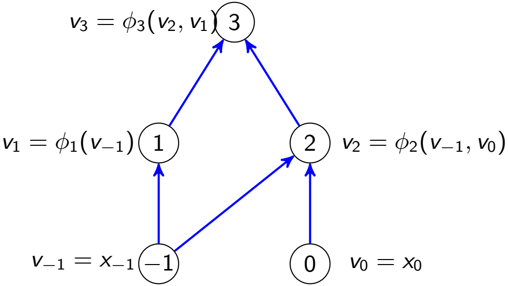


$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

$$
\begin{aligned}
v_{-1} &= x_{-1} \\
v_0 &= x_0 \\
v_1 &= \phi_1(v_{-1}) \\
v_2 &= \phi_2(v_{-1}, v_0) \\
v_3 &= \phi_3(v_2, v_1)
\end{aligned}
$$

$$\Phi^3 \circ \Phi^2 \circ \Phi^1 \circ \mathcal{I}x$$

$$(v_{1-n}, v_0, v_1, v_2, v_3)$$

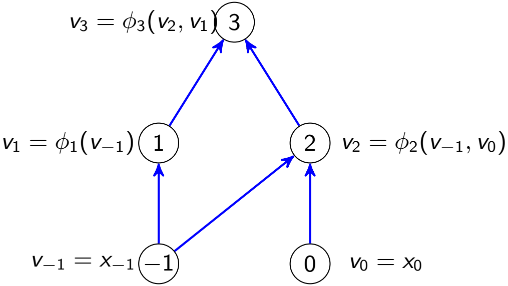


$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

$$f(x) = e_{3+n}^T \Phi^3 \circ \Phi^2 \circ \Phi^1 \circ \mathcal{I} x$$

$$v_3 = e_{3+n}^T (v_{1-n}, v_0, v_1, v_2, v_3)$$



$$\phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

$$v_{-1} = x_{-1}$$
$$v_0 = x_0$$
$$v_1 = \phi_1(v_{-1})$$
$$v_2 = \phi_2(v_{-1}, v_0)$$
$$v_3 = \phi_3(v_2, v_1)$$

$$f(x) = e_{3+n}^T \Phi^3 \circ \Phi^2 \circ \Phi^1 \circ \mathcal{I}x$$

$$v_3 = e_{3+n}^T (v_{1-n}, v_0, v_1, v_2, v_3)$$

We can differentiate compositions of operators

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^{\ell} \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I} x$$

Chain-rule says:

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^\ell \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I} x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^T D\Phi^\ell \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^\ell \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I} x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^T D\Phi^\ell \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

$$\bar{v}^T \leftarrow e_{\ell+n}^T$$

Reverse Gradient

$$f(x) = e_{\ell+n}^{T} \Phi^{\ell} \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^{1} \circ \mathcal{I} x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^{T} D\Phi^{\ell} \cdot D\Phi^{\ell-1} \cdots D\Phi^{1} \cdot \mathcal{I}$$

$$\bar{v}^{T} \leftarrow \bar{v}^{T} D\Phi^{\ell}$$

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^{\ell} \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I} x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^T D\Phi^{\ell} \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

$$\bar{v}^T \leftarrow \bar{v}^T D\Phi^{\ell-1}$$

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^{\ell} \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I}x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^T D\Phi^{\ell} \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

initialization: $\bar{v} = e^{\ell+n}$
for $i = \ell, \ldots, 1$ **do**
$\quad \bar{v}^T \leftarrow \bar{v}^T D\Phi^i$
end
Output: $Df(x) = \bar{v}$

- Implemented version $O(\text{eval}(f))$ **independent of** n!

Reverse Gradient

$$f(x) = e_{\ell+n}^T \Phi^{\ell} \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I}x$$

Chain-rule says: Multiply the Jacobians

$$Df = e_{\ell+n}^T D\Phi^{\ell} \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

initialization: $\bar{v} = e^{\ell+n}$
for $i = \ell, \ldots, 1$ **do**
    $\bar{v}^T \leftarrow \bar{v}^T D\Phi^i$
end
Output: $Df(x) = \bar{v}$

- Implemented version $O(\text{eval}(f))$ **independent of** n!

Reverse Gradient

$$f(x) = e^T_{\ell+n}\Phi^\ell \circ \Phi^{\ell-1} \circ \cdot \circ \Phi^1 \circ \mathcal{I}x$$

Chain-rule says: Multiply the Jacobians

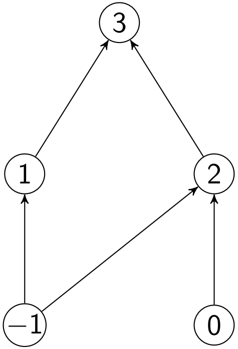
$$Df = e^T_{\ell+n} D\Phi^\ell \cdot D\Phi^{\ell-1} \cdots D\Phi^1 \cdot \mathcal{I}$$

---

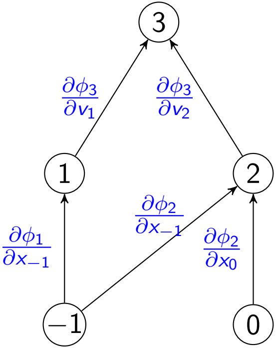
initialization: $\bar{v} = e^{\ell+n}$
for $i = \ell, \ldots, 1$ **do**
$\quad \bar{v}^T \leftarrow \bar{v}^T D\Phi^i$
end
Output: $Df(x) = \bar{v}$

- Implemented version $O(\text{eval}(f))$ **independent of** n!
- Back on the graph (where calculations actually take place)

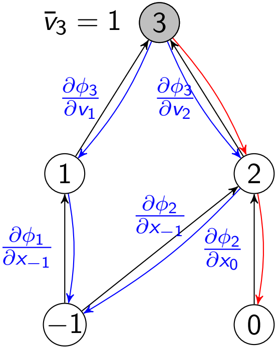
$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



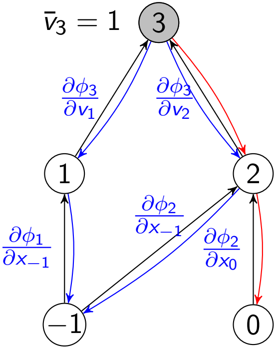
$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



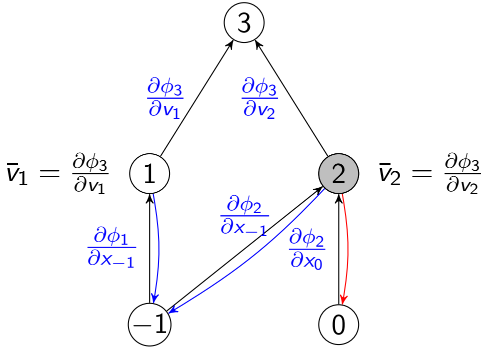
$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



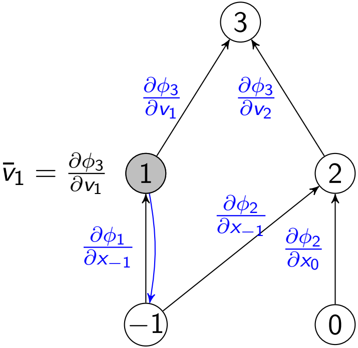
$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$

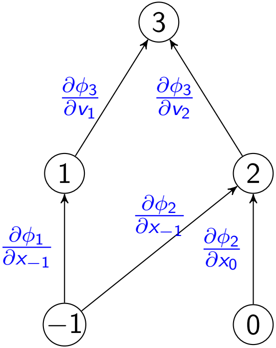


$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



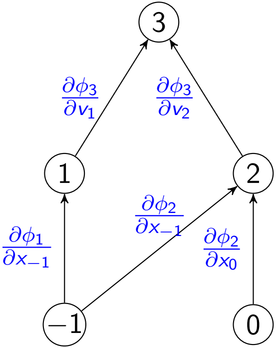
$$\bar{v}_{-1} = \frac{\partial \phi_2}{\partial x_{-1}} \bar{v}_2 \qquad \bar{v}_0 = \frac{\partial \phi_2}{\partial x_0} \bar{v}_2$$

$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



$$\bar{v}_{-1} = \frac{\partial \phi_1}{\partial x_{-1}} \bar{v}_1 + \frac{\partial \phi_2}{\partial x_{-1}} \bar{v}_2 \quad \bar{v}_0 = \frac{\partial \phi_2}{\partial x_0} \bar{v}_2$$

$$f(x) = \phi_3(\phi_1(x_{-1}), \phi_2(x_{-1}, x_0))$$



$$\bar{v}_{-1} = \frac{\partial\phi_1}{\partial x_{-1}}\bar{v}_1 + \frac{\partial\phi_2}{\partial x_{-1}}\bar{v}_2 \quad \bar{v}_0 = \frac{\partial\phi_2}{\partial x_0}\bar{v}_2$$

$$\frac{\partial f}{\partial x_{-1}} = \bar{v}_{-1} \quad \frac{\partial f}{\partial x_0} = \bar{v}_0$$

$$D^2 f =$$

$$D^2 f = \text{Differentiating again gets messy}$$

$$D^2 f = \text{Differentiating again gets messy}$$

Use induction instead

$$f(x) = e_{\ell+n}\Phi^2 \circ \Phi^1(x)$$

$$D^2 f = \text{Differentiating again gets messy}$$

Use induction instead

$$f(x) = e_{\ell+n}\Phi^2 \circ \Phi^1(x)$$

$$D^2 f = e_{\ell+n}^T D^2\Phi^2 \cdot (D\Phi^1, D\Phi^1) + D\Phi^2 \cdot D^2\Phi^1.$$

$$D^2 f = \text{Differentiating again gets messy}$$

Use induction instead

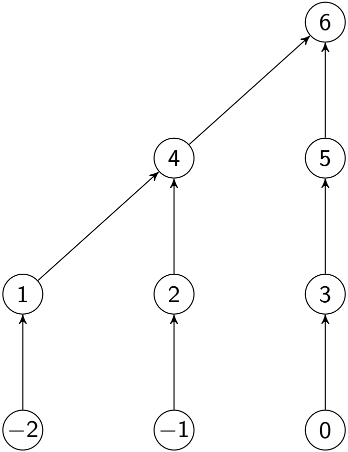
$$f(x) = e_{\ell+n}\Phi^2 \circ \Phi^1(x)$$

$$D^2 f = e_{\ell+n}^T D^2\Phi^2 \cdot (D\Phi^1, D\Phi^1) + D\Phi^2 \cdot D^2\Phi^1.$$

$$D^3 f \cdot d = e_{\ell+n}^T D^3\Phi^2 \cdot (D\Phi^1, D\Phi^1, D\Phi^1 d) + e_{\ell+n}^T D\Phi^2 \cdot D^3\Phi^1 d$$
$$+ e_{\ell+n}^T D^2\Phi^2 \cdot \left((D\Phi^1, D^2\Phi^1 d) + (D^2\Phi^1 d, D\Phi^1) + (D^2\Phi^1, D\Phi^1 d)\right)$$

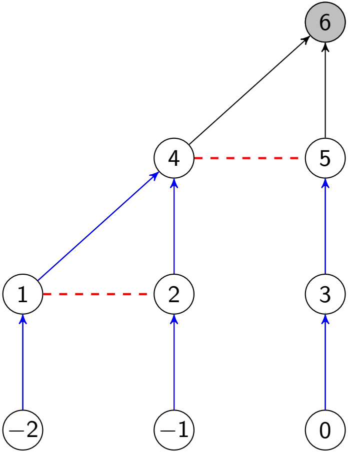
Solve in reverse, apply inductively, solve a case with ℓ compositions.

Reverse Hessian on Graph



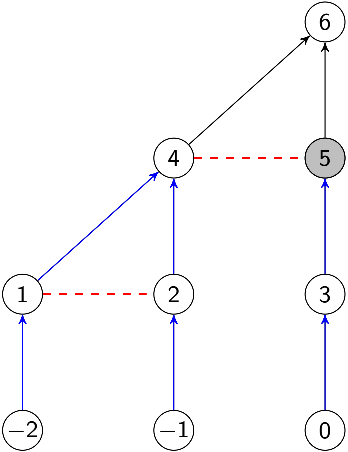
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



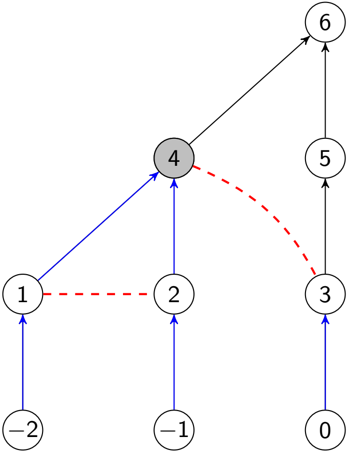
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3 v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



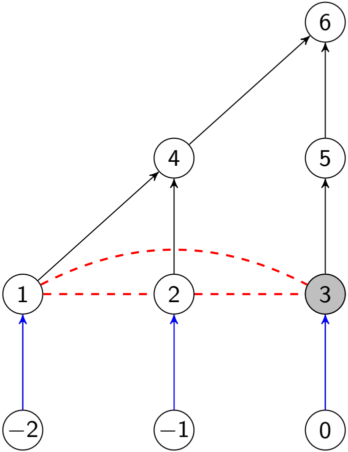
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



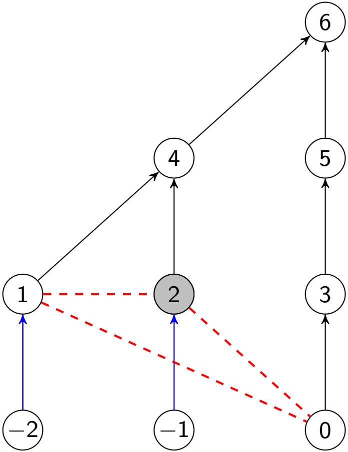
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



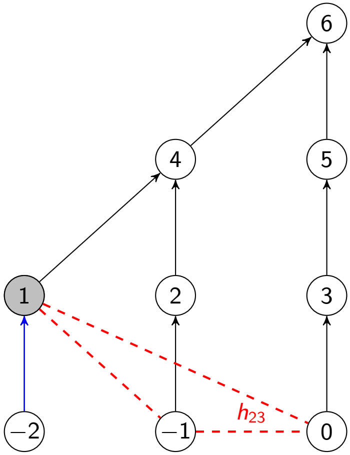
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



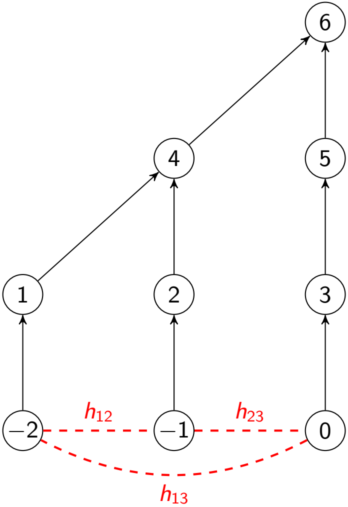
$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$
$v_1 = v_{-2} + 1$
$v_2 = v_{-1} + 1$
$v_3 = v_0 + 1$
$v_4 = v_1 v_2$
$v_5 = 3v_3$
$v_6 = v_4 v_5$

Reverse Hessian on Graph



$$f(x) = (x_{-2}+1)(x_{-1}+1)3(x_0+1)$$

$$
\begin{aligned}
v_1 &= v_{-2} + 1\\
v_2 &= v_{-1} + 1\\
v_3 &= v_0 + 1\\
v_4 &= v_1 v_2\\
v_5 &= 3v_3\\
v_6 &= v_4 v_5
\end{aligned}
$$

$$D^2 f = \begin{pmatrix} 0 & h_{12} & h_{13} \\ h_{12} & 0 & h_{23} \\ h_{13} & h_{12} & 0 \end{pmatrix}$$

Reverse Hessian Results

- Implemented version average 9s for Hessian matrices $10^6 \times 10^6$ from CUTE.
- Faster then state-of-the-art graph coloring based methods, Gebremedhin, Manne, Pothen, Walther, Tarafdar

$D^2 f(x)$ $D^2 f(x) S$

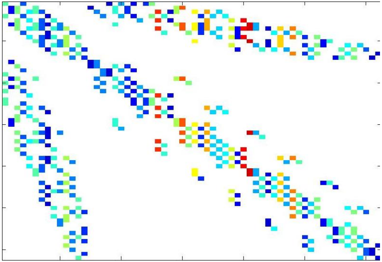
$\Rightarrow$

Robert Gower, M P Mello (2012)
A new framework for the computation of Hessians
Optimization Methods and Software 2(27), 251–2738.

Reverse Hessian Results

- Implemented version average 9s for Hessian matrices $10^6 \times 10^6$ from CUTE.
- Faster then state-of-the-art graph coloring based methods, Gebremedhin, Manne, Pothen, Walther, Tarafdar

$D^2 f(x)$ $\qquad\qquad\qquad$ $D^2 f(x) S$



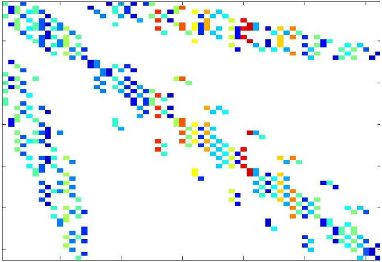
$\Rightarrow$

Robert Gower, M P Mello (2012)
A new framework for the computation of Hessians
Optimization Methods and Software 2(27), 251–2738.

Reverse Hessian Results

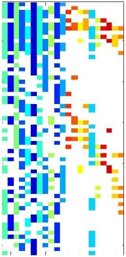
- Implemented version average 9s for Hessian matrices $10^6 \times 10^6$ from CUTE.
- Faster then state-of-the-art graph coloring based methods, Gebremedhin, Manne, Pothen, Walther, Tarafdar

$D^2 f(x)$



$\Rightarrow$

$D^2 f(x) S$



Robert Gower, M P Mello (2012)
A new framework for the computation of Hessians
Optimization Methods and Software 2(27), 251–2738.

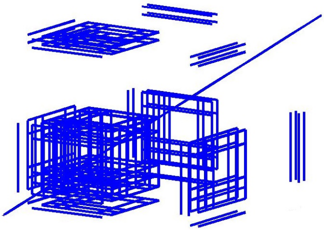
Differentiating operators works for higher orders

Same induction technique, design reverse method for

Differentiating operators works for higher orders

Same induction technique, design reverse method for

$D^3 f(x) =$



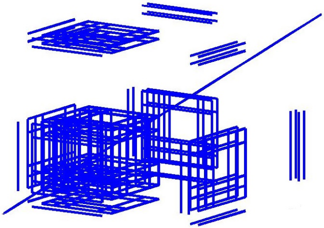
A sparse symmetric cube

Differentiating operators works for higher orders

Same induction technique, design reverse method for

$$\frac{d}{dt} D^2 f(x+td)|_0 = D^3 f(x) \cdot d =$$

$D^3 f(x) =$



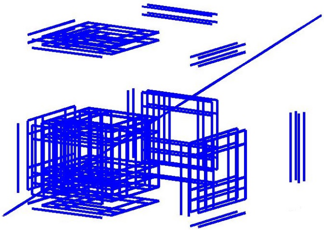
A sparse symmetric cube

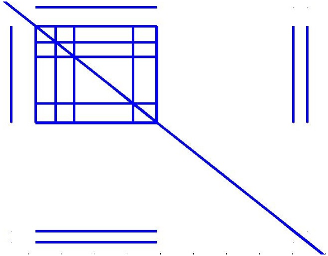
Differentiating operators works for higher orders

Same induction technique, design reverse method for

$\frac{d}{dt} D^2 f(x+td)|_0 = D^3 f(x) \cdot d =$

$D^3 f(x) =$



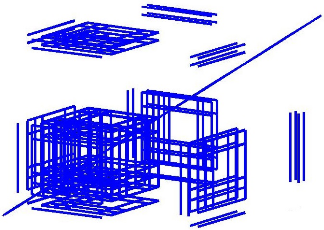


A sparse symmetric cube

Differentiating operators works for higher orders

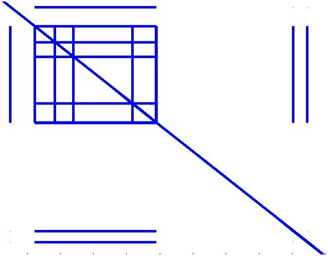
Same induction technique, design reverse method for

$D^3 f(x) =$



A sparse symmetric cube

$\frac{d}{dt} D^2 f(x + td)|_0 = D^3 f(x) \cdot d =$



A sparse symmetric matrix. **No cube ever formed**. Pseudocode

Tests calculating $D^3 f(x) \cdot v$ is fast

Average 10 seconds for dimension $= 10^6 \times 10^6$

Costs 1.08% of $D^2 f(x)$, on average

Tests calculating $D^3 f(x) \cdot v$ is fast

Average 10 seconds for dimension $= 10^6 \times 10^6$

Costs 1.08% of $D^2 f(x)$, on average

name	Pattern	nnz/n	$D^3 f(x) \cdot v + D^2 f(x)$
cosine	B 3	3.0000	5.25
chainwood	B 3	1.4999	7.22
morebv	B 3	3.0000	9.44
scon1dls	B 3	0.7002	8.12
bdexp	B 5	0.0004	3.86
pspdoc	B 5	4.9999	5.97
augmlagn	5×5 diagonal blocks	4.9998	9.28
brybnd	B 11	12.9996	38.79
chainros_trigexp	B 3 + D 6	4.4999	12.87
toiqmerg	B 7	6.9998	8.89
arwhead	arrow	3.0000	6.78
nondquar	arrow + B 3	4.9999	5.61
sinquad	frame + diagonal	4.9999	10.01
bdqrtic	arrow + B 7	8.9998	19.62
noncvxu2	irregular	6.9998	9.55
ncvxqp3	irregular	6.9997	6.48
heavey_band	B 39	38.9995	61.27

Large dimensional tests become possible

Sometimes Newton is faster

$$\text{Arrow head}(x) = \sum_{i=1}^{n-1} \left(-4x_i + 3.0 + (x_i^2 + x_{n-1}^2)^2\right)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	5 (7s)	5(14s)	5(11s)	5(11s)

$$\text{Broyden banded}(x) = \sum_{i=1}^{n} \left(x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j) \right)^2$$

$J_i = \{j \in 1 \cdots n : \max(1, i-1) \leq j \leq \min(n, i+5)\}$, for $i = 1, \ldots, n$.

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	125 (290s)	117 (768s)	117 (770s)	118 (791s)

Large dimensional tests become possible

Sometimes Newton is faster

$$\text{Arrow head}(x) = \sum_{i=1}^{n-1} \left(-4x_i + 3.0 + (x_i^2 + x_{n-1}^2)^2\right)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	5 (7s)	5(14s)	5(11s)	5(11s)

$$\text{Broyden banded}(x) = \sum_{i=1}^{n} \left(x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j) \right)^2$$

$J_i = \{j \in 1 \cdots n : \max(1, i-1) \leq j \leq \min(n, i+5)\}$, for $i = 1, \ldots, n$.

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	125 (290s)	117 (768s)	117 (770s)	118 (791s)

Sometimes Halley-Chebyshev is better

λ makes a difference

$$\text{Chain Wood}(x) = \sum_{i=1}^{n/2-2} \Big(100(x_{2i} - x_{2i-1}^2)^2 + (1.0 - x_{2i-1})^2 \\ + 90(x_{2i+2} - x_{2i+1}^2)^2 + (1.0 - x_{2i+1})^2 \\ + 10(x_{2i} + x_{2i+2} - 2.0)^2 + (x_{2i} - x_{2i+2})^2/10\Big)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	NC	NC	NC	18 (54.7s)

$$\text{bdqrtic}(x) = \sum_{i=1}^{n-4} \left((-4x_i + 3.0)^2 + (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2\right)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
100	2683(2.6s)	62 (0.2 s)	62 (0.2s)	51 (0.2s)
10^4	$10^5 >$ (20min)	94 (40s)	94 (40s)	93 (40s)

Sometimes Halley-Chebyshev is better

λ makes a difference

$$\text{Chain Wood}(x) = \sum_{i=1}^{n/2-2} \big(100(x_{2i} - x_{2i-1}^2)^2 + (1.0 - x_{2i-1})^2 + 90(x_{2i+2} - x_{2i+1}^2)^2 + (1.0 - x_{2i+1})^2 + 10(x_{2i} + x_{2i+2} - 2.0)^2 + (x_{2i} - x_{2i+2})^2/10\big)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
$2 \cdot 10^5$	NC	NC	NC	18 (54.7s)

$$\text{bdqrtic}(x) = \sum_{i=1}^{n-4} \big((-4x_i + 3.0)^2 + (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2\big)$$

n	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
100	2683(2.6s)	62 (0.2 s)	62 (0.2s)	51 (0.2s)
10^4	$10^5 >$ (20min)	94 (40s)	94 (40s)	93 (40s)

Conclusions & Contributions

- method for designing high order AD algorithms
- new third-order methods for large-scale

  Robert Gower, Artur Gower (2013)
  Higher-order Reverse Automatic Differentiation with emphasis on the third-order
  submitted www.maths.ed.ac.uk/ERGO/

- Automatic derivatives $\Rightarrow$ Empirical comparisons of high order methods possible.
- Too many failures for Halley-Cheby on general nonlinear. Step size & damping required.

Conclusions & Contributions

- method for designing high order AD algorithms
- new third-order methods for large-scale

  Robert Gower, Artur Gower (2013)
  Higher-order Reverse Automatic Differentiation with emphasis on the third-order
  submitted `www.maths.ed.ac.uk/ERGO/`

- Automatic derivatives $\Rightarrow$ Empirical comparisons of high order methods possible.
- Too many failures for Halley-Cheby on general nonlinear. Step size & damping required.
- What is possible if high order information is not so expensive?

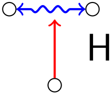
References

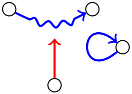
J.M. Gutiérrez and M.a. Hernández (1997)
A family of Chebyshev-Halley type methods in Banach spaces
Bulletin of the Australian Mathematical Society 1(55), 113–133.

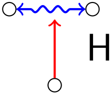
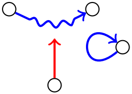
Geir Gundersen and Trond Steihaug (2012)
On diagonally structured problems in unconstrained optimization using an inexact super Halley method
Journal of Computational and Applied Mathematics 15(236), 3685–3695.

W. Hock and K. Schittkowski (1980)
Test examples for nonlinear programming codes
Journal of Optimization Theory and Applications, 30, pp. 127–129.

Ladislav Luksan and Jan Vlcek (2003)
Test problems for unconstrained optimization


HANK


U

 HANK  U

QUESTIONS?

We have hand-picked sixteen problems from the CUTE collection, augm- lagn from [Hock, 1980], toiqmerg (Toint Quadratic Merging problem) and chainros trigexp (Chained Rosenbrook function with Trigonometric and exponential constraints) from [Vlcek, 2003] for the experiments. We have also created a function

$$\text{heavey\_band}(x, band) = \sum_{i=0}^{n-band} \sin\left(\sum_{j=0}^{band} x_{i+j}\right)$$

For our experiments, we tested heavey band(x, 20). Back

Preliminary tests Halley-Chebychev $\times$ Newton

Halley-Chebyshev iteration costs 2 to 3 X Newton step

- Large dimensional tests become possible
- Some cases Halley-Chebyshev better
- Some cases Newton is better
- λ makes a difference!

Name:dimension	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
cosine:30	FAIL	74	74	74
cragglevy:10	FAIL	218	218	225
chainwood:2.10^5	FAIL	FAIL	FAIL	18
brybnd:10^5	125	117	117	118
arwhead:2.10^5	5	5	5	5
sinquad:2.10^5	29	18	18	20
bdqrtic:100	2683	62	62	51
bdqrtic:10^4	$10^5 >$	94	94	93

Table for time

Preliminary tests Halley-Chebychev $\times$ Newton

Halley-Chebyshev iteration costs 2 to 3 X Newton step

- Large dimensional tests become possible
- Some cases Halley-Chebyshev better
- Some cases Newton is better
- λ makes a difference!

Name:dimension	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
cosine:30	FAIL	74	74	74
cragglevy:10	FAIL	218	218	225
chainwood:2.10^5	FAIL	FAIL	FAIL	18
brybnd:10^5	125	117	117	118
arwhead:2.10^5	5	5	5	5
sinquad:2.10^5	29	18	18	20
bdqrtic:100	2683	62	62	51
bdqrtic:10^4	$10^5 >$	94	94	93

Table for time

Preliminary tests Halley-Chebychev $\times$ Newton

Halley-Chebyshev iteration costs 2 to 3 X Newton step

- Large dimensional tests become possible
- Some cases Halley-Chebyshev better
- Some cases Newton is better
- λ makes a difference!

Name:dimension	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
cosine:30	FAIL	74	74	74
cragglevy:10	FAIL	218	218	225
chainwood:2.10^5	FAIL	FAIL	FAIL	18
brybnd:10^5	125	117	117	118
arwhead:2.10^5	5	5	5	5
sinquad:2.10^5	29	18	18	20
bdqrtic:100	2683	62	62	51
bdqrtic:10^4	$10^5 >$	94	94	93

Table for time

Preliminary tests Halley-Chebychev $\times$ Newton

Halley-Chebyshev iteration costs 2 to 3 X Newton step

- Large dimensional tests become possible
- Some cases Halley-Chebyshev better
- Some cases Newton is better
- λ makes a difference!

Name:dimension	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
cosine:30	FAIL	74	74	74
cragglevy:10	FAIL	218	218	225
chainwood:2.10^5	FAIL	FAIL	FAIL	18
brybnd:10^5	125	117	117	118
arwhead:2.10^5	5	5	5	5
sinquad:2.10^5	29	18	18	20
bdqrtic:100	2683	62	62	51
bdqrtic:10^4	$10^5 >$	94	94	93

Table for time

Name:dimension	Newton	HallC $\lambda = 0$	HallC $\lambda = 0.5$	HallC $\lambda = 1.0$
cosine:30	FAIL	0.03	0.02	0.02
cragglevy:10	FAIL	0.05	0.03	0.04
chainwood:$2 \cdot 10^5$	FAIL	FAIL	FAIL	54.7
brybnd:10^5	289.96	767.51	769.36	790.02
arwhead:$2 \cdot 10^5$	6.31	13.54	10.79	10.98
sinquad:$2 \cdot 10^5$	22.67	39.56	39.65	44.07
bdqrtic:100	2.58	0.18	0.16	0.14
Bdqrtic:10^4	$1492 >$	39.57	39.78	39.34

Back

Algorithm 1: Reverse Hessian Directional Derivative

initialization: $\dot{\mathrm{v}}^1 = d, \overline{\mathrm{v}} = y, W = Td = 0 \in \mathbb{R}^{m_\ell \times m_\ell}$

for $i = 1, \ldots, \ell$ **do**

  $\dot{\mathrm{v}}^i \leftarrow D\Phi^i \cdot \dot{\mathrm{v}}^{i-1}$

end

for $i = \ell, \ldots, 1$ **do**

  $Td \leftarrow Td \cdot (D\Phi^i, D\Phi^i)$

  $Td \leftarrow Td + W \cdot \big((D\Phi^i, D^2\Phi^i 1{,}1dot \dot{\mathrm{v}}^{i-1}) + (D^2\Phi^i \cdot \dot{\mathrm{v}}^{i-1}, D\Phi^i)\big)$

  $Td \leftarrow Td + W \cdot (D^2\Phi^i, D\Phi^i \cdot \dot{\mathrm{v}}^{i-1})$

  $Td \leftarrow Td + \overline{\mathrm{v}}^T D^3\Phi^i \cdot \dot{\mathrm{v}}^{i-1}$

  $W \leftarrow W \cdot (D\Phi^i, D\Phi^i) + \overline{\mathrm{v}}^T D^2\Phi^i$

  $\overline{\mathrm{v}}^T \leftarrow \overline{\mathrm{v}}^T D\Phi^i$

end

Output: $y^T D^3F(x) \cdot d \leftarrow Td,\ y^T D^2F \leftarrow W,\ y^T DF \leftarrow \overline{\mathrm{v}}^T$

Back