Automatic Differentiation Applied to QR Iteration

Peter Solfest

University of Minnesota
Department of Computer Science and Engineering

Research Group Meeting 11/11/2014

Outline

Automatic Differentiation (AD)

Big Picture

- At its core, computer code consists of arithmetic and logical operations
- AD applies elementary differentiation rules to the arithmetic operations

Can overload operators to achieve this.
Examples:

$$
\begin{aligned}
a + b &\to a + b, \partial a + \partial b \\
a \cdot b &\to a \cdot b, \partial a \cdot b + a \cdot \partial b \\
\sin(a) &\to \sin(a), \cos(a) \cdot \partial a \\
&\dots
\end{aligned}
$$

Forward Mode

- Calculate effect on output due to perturbed inputs
- Input perturbation direction is provided to seed the calculation
- Notation: $\dot{w}$ is the derivative w.r.t. input perturbation

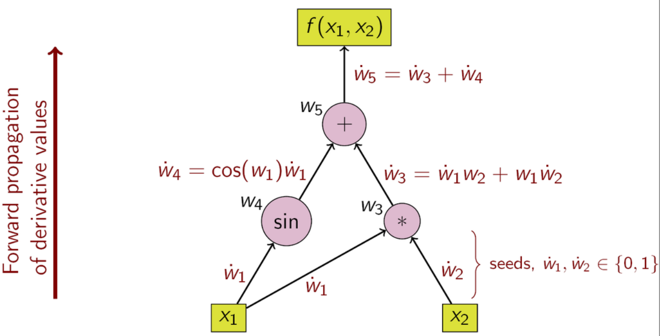


Figure: The subscripts represent intermediate steps[1]

[1] Figure from wikipedia's Automatic Differentiation page

Reverse Mode

- Calculate what input perturbation creates output perturbation
- Output perturbation direction is provied to seed the calculation
- Notation: $\bar{w}$ is the derivative w.r.t. input perturbation

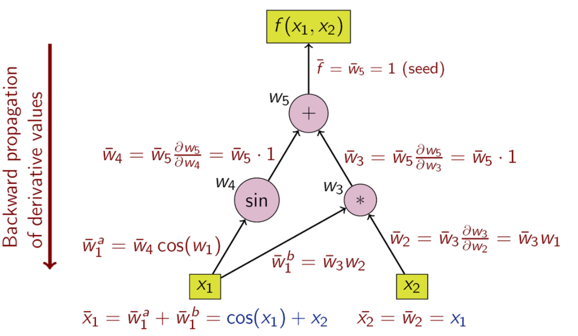


Figure: The subscripts represent intermediate steps[2]

[2] Figure from wikipedia's Automatic Differentiation page

Abstract Formulation

- An algorithm can be viewed as a map

$$f : \mathbb{R}^m \to \mathbb{R}^n$$

- Forward Mode calculates directional derivatives of f (the directions are the seeds)
- Reverse Mode calculates the gradient of f (if seeded with 1, otherwise is a scaled gradient)

Forwards vs. Backwards

For an algorithm $\vec{f} : \mathbb{R}^m \to \mathbb{R}^n$

Forward Mode	Backwards Mode
simple memory access	complicated memory access
1-2 function evaluations for $\nabla \vec{f} \cdot \vec{d}$	1-10 function evaluations for ∇f_i
m sweeps to compute $\nabla \vec{f}$	n sweeps to compute $\nabla \vec{f}$

Will focus on forward mode

Schur Decomposition AD

For $A \in \mathbb{R}^{n \times n}$, the Schur decomposition is

$$\begin{aligned} A &= QSQ' \\ Q'Q &= I \end{aligned}$$

with S upper triangular[3]
Thus

$$\begin{aligned} \dot{A} &= \dot{Q}SQ' + Q\dot{S}Q' + QS\dot{Q}' \\ Q'\dot{A}Q &= PS + SP' + \dot{S} \\ Q'\dot{A}Q &= PS - SP + \dot{S} \end{aligned}$$

Where $P = Q'\dot{Q}$, and the orthonormality of Q yields that $P = -P'$

[3] except 2x2 blocks on diagonal

Schur Decomp AD Algorithm

- Treat $\dot{A}$ as the seeded input, define $B = Q'\dot{A}Q$
- Noting $S, \dot{S}$ are upper triangular and P is skew symmetric, note

$$\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} P_{11} & -P'_{21} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix} - \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix} \begin{bmatrix} P_{11} & -P'_{21} \\ P_{21} & P_{22} \end{bmatrix} + \begin{bmatrix} \dot{S}_{11} & \dot{S}_{12} \\ 0 & \dot{S}_{22} \end{bmatrix}$$

- The (21) block yields $B_{21} = P_{21}S_{11} - S_{22}P_{21} + 0$, use triangular sylvester equation solver[4]
- Recursively solve the (11) and (22) blocks
- Compute $\dot{Q} = QP$ and $\dot{S} = B - PS + SP$

[4] e.g. Octave's `syl`. Or use the same splitting trick

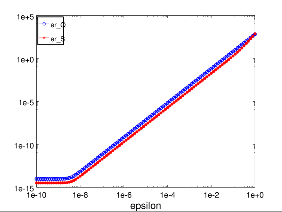
Schur Decomp AD Performance

Given a Schur decomposition $A = Q * S * Q'$, and seed $\dot{A}$:

$$A + \varepsilon \dot{A} = (Q + \varepsilon \dot{Q})(S + \varepsilon \dot{S})(Q + \varepsilon \dot{Q})' + O(\varepsilon^2)$$

$$(Q + \varepsilon \dot{Q})'(Q + \varepsilon \dot{Q}) = I + O(\varepsilon^2)$$

Following is the difference (error) observed as measured in the Frobenius norm.



On 10 random matrices in $\mathbb{R}^{400\times 400}$, octave's built in `schur` function took 3.78 seconds (cumulative), and the schurAd algorithm spent 3.47 seconds (cumulative) calculating the derivatives.

Implicit multishift QR and Enhancements

Multishift QR

Multishift QR sweep:

function $\text{MQRSWEEP}(H,\vec{\sigma})$

form householder vector from 1^{st} column of $\prod_{i=1}^{s}(H - \sigma_i I)$

Apply householder transformation to H using this vector

Reduce H to hessenberg form

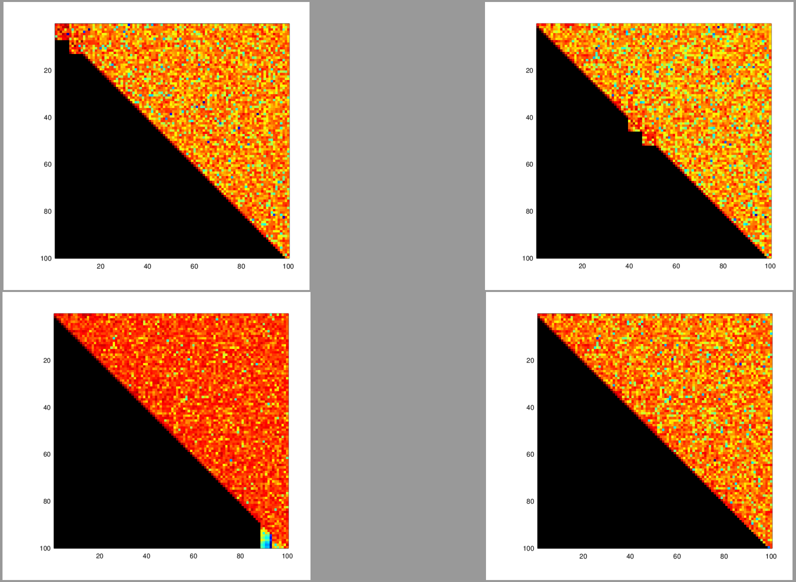
Deflate H if small entry on subdiagonal

return H, deflated eigenvalues

end function

In practice for $s > 5$, add the shifts in 5 at a time rather than all at once.[]

One 'Sweep'

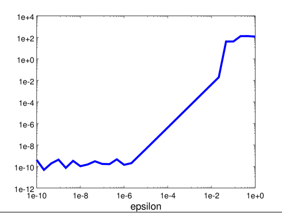


QRI AD Performance

Consider $f : \mathbb{R}^s \to \mathbb{R}^{n\times n}$, where the inputs are the shifts $\vec{\sigma}$, and the output is the matrix after a sweep of QRI.
Then we have in the direction $\vec{d}$

$$f(\vec{\sigma}) + \varepsilon \dot{f}(\vec{\sigma}) = f(\vec{\sigma} + \varepsilon \vec{d}) + O(\varepsilon^2)$$

Below is the difference (error) observed as measured in the Frobenius norm.



On 10 random matrices in $\mathbb{R}^{400\times 400}$, using 20 shifts, a QRI sweep took 8.74 seconds (cumulative), and with 1 seed it took 22.47 seconds (cumulative).

AD Definitions

- the shifts $\vec{\sigma}$ form a natural set of inputs
- $f : \mathbb{R}^s \to \mathbb{R}^t$
- define $J_{i,j}(\vec{\sigma}) = \dfrac{\partial f_i}{\partial \sigma_j}(\vec{\sigma})$, so $J \in \mathbb{R}^{t \times s}$
- Assume we want to find $\vec{\sigma}$ such that $f(\vec{\sigma}) = \vec{0}$
- Newton's method provides an update procedure $\vec{\sigma}^{(n+1)} = \vec{\sigma}^{(n)} - J^{-1}(\vec{\sigma}^{(n)})f(\vec{\sigma}^{(n)})$

AD Variation on standard QRI

Standard Algorithm:

repeat

$\sigma \leftarrow eig(H_{n-s:n,n-s:n})$

$H \leftarrow MQRSweep(H, \sigma)$

until length(H) < 2

AD Variation:

Choose the output of f to be the s bottommost subdiagonals

Choose σ

repeat

$\dot{H} \leftarrow MQRSweep(H, \sigma)$ for seeded $I_{s \times s}$

Deflate H if possible

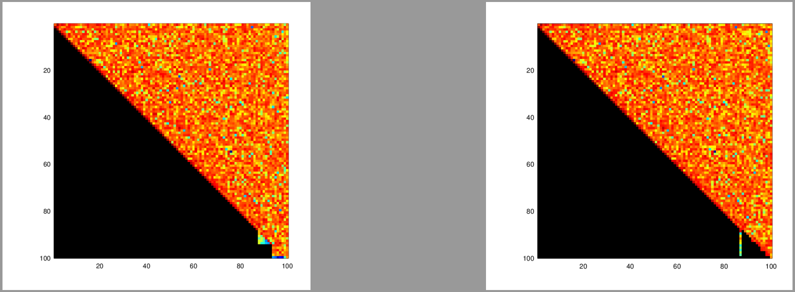
Form $J(\vec{\sigma})$

$\vec{\sigma} \leftarrow \vec{\sigma} - J^{-1}(\vec{\sigma})f(\vec{\sigma})$

until length(H) < 2

Aggresive Early Deflation

- drive the bulges through the matrix via QRI
- perform a Schur decomposition on the bottom portion containing the bulges, introducing a spike
- deflate matrix if spike values are below a tolerance



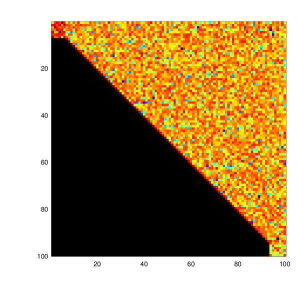
For the AD version, the outputs are the bottom s values on the spike.

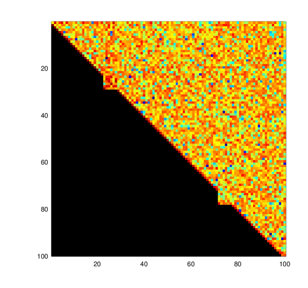
Middle Deflation

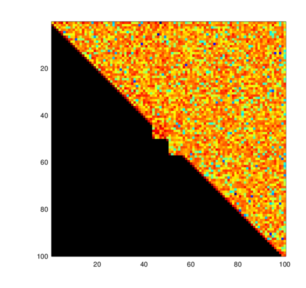
1. Bulges are driven from both ends
2. When they meet, a Schur decomposition is performed, generating 2 spikes[]
3. If the tips of the spikes have enough zeros, can split matrix in the middle

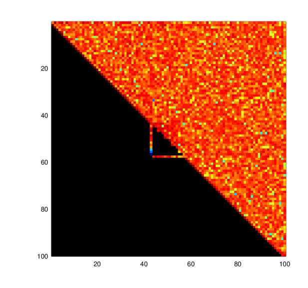
- Performing this in the middle of QRI was proposed in Braman
- the outputs for the AD algorithm will be the tips of the spikes

One 'Sweep' to middle









Implementation Details

- Schur decomposion is not unique since the eigenvalues can be reordered. Thus they were reordered to yield minimum value in the spike tips []
- For aggressive early deflation, a second spike on the deflated matrix without another QR iteration can yield further deflatable eigenvalues which are deflated[]

Results

- Run on matrices generated by normal distribution of random numbers
- the Jacobian becomes singular very quickly.
- Using a low-rank approximation to the Jacobian, causing shifts to converge when deflation isn't possible
- Re-initializing shifts when this happens leads to convergence to another non-deflatable minima

Bibliography