Seed Matrices

Laurent Hascoët
Paul Hovland
Jan Hückelheim
Sri Hari Krishna Narayanan

SIAM AN24 Student Days

Seed Matrices: Theory

Recall:
Forward mode computes

$$JS$$

at cost proportional to number of columns in S.

Reverse mode computes

$$W^T J$$

at cost proportional to number of columns in W.

Seed Matrices: Practice

In reality, it is rarely that simple.

- data structure pragmatics
- multiple output variables
- multiple input variables
- Jacobian compression

Seed Matrices: data structure pragmatics

Depending on

- programming language (row-major vs. column-major arrays)
- choice of AD tool

the J, S, W matrices might be stored in a transposed layout or as part of a struct and the derivative dimension might be flattened.

Consequently, what is logically $\dot{X}_{ij}$ might be stored as `dX(j,i)` or `x[i].grad[j]`.

Seed Matrices: multiple scalar outputs, scalar input

- Consider function `foo(in:x,out:f,out:g)`
- Then, forward mode AD computes

$$\dot{f} = \frac{\partial f}{\partial x}\dot{x} \quad \text{and} \quad \dot{g} = \frac{\partial g}{\partial x}\dot{x}$$

- Initializing $\dot{x} = 1$ yields the full (logical) Jacobian, split over two data structures.

$$J = \begin{bmatrix} \dot{f} \\ \dot{g} \end{bmatrix}$$

Seed Matrices: multiple scalar outputs, vector input

- Consider function `foo(in:x(N),out:f,out:g)`
- That is, x is now a vector of length N.
- Then, vector forward mode AD computes

$$\dot{f} = \frac{\partial f}{\partial x}\dot{x} \quad \text{and} \quad \dot{g} = \frac{\partial g}{\partial x}\dot{x}$$

- Initializing $\dot{x}$ to the identity matrix yields the full Jacobian, at a cost proportional to N.
- Alternatively, one could loop over $\dot{x} = e_i$, computing one column of the Jacobian at a time.

Seed Matrices: multiple scalar outputs, vector input

- Consider function `foo(in:x(N),out:f,out:g)`
- Then, reverse mode AD computes

$$\overline{x} = \overline{f}\,\frac{\partial f}{\partial x} + \overline{g}\,\frac{\partial g}{\partial x}$$

- Initializing $\overline{f} = 1$ and $\overline{g} = 0$ yields $\overline{x} = \frac{\partial f}{\partial x}$
- Initializing $\overline{f} = 0$ and $\overline{g} = 1$ yields $\overline{x} = \frac{\partial g}{\partial x}$

Seed Matrices: multiple scalar outputs, vector input

- Initializing $\overline{f} = 1$ and $\overline{g} = 0$ yields $\overline{x} = \frac{\partial f}{\partial x}$
- Initializing $\overline{f} = 0$ and $\overline{g} = 1$ yields $\overline{x} = \frac{\partial g}{\partial x}$
- Thus, we are computing $W^T J$ for each of

$$W = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } W = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- In vector reverse mode, we can let $\overline{f} = [1\ 0]$ and $\overline{g} = [0\ 1]$ and compute the full Jacobian all at once, using $W = I$.

Seed Matrices: multiple scalar inputs, scalar output

- Now, consider function `foo(in:x,in:y,out:f)`
- Then, reverse mode AD computes

$$\overline{x} = \overline{f}\,\frac{\partial f}{\partial x} \quad \text{and} \quad \overline{y} = \overline{f}\,\frac{\partial f}{\partial y}$$

- Initializing $\overline{f} = 1$ yields the full Jacobian, or gradient, split over two data structures.

$$J = \left[\begin{array}{cc} \overline{x} & \overline{y} \end{array}\right]$$

Seed Matrices: multiple scalar inputs, scalar output

- Consider function `foo(in:x,in:y,out:f)`
- Then, forward mode AD computes

$$\dot{f} = \frac{\partial f}{\partial x}\,\dot{x} + \frac{\partial f}{\partial y}\,\dot{y}$$

- Initializing $\dot{x} = 1$ and $\dot{y} = 0$ yields $\dot{f} = \frac{\partial f}{\partial x}$
- Initializing $\dot{x} = 0$ and $\dot{y} = 1$ yields $\dot{f} = \frac{\partial f}{\partial y}$

Seed Matrices: multiple scalar inputs, scalar output

- Initializing $\dot{x} = 1$ and $\dot{y} = 0$ yields $\dot{f} = \frac{\partial f}{\partial x}$
- Initializing $\dot{x} = 0$ and $\dot{y} = 1$ yields $\dot{f} = \frac{\partial f}{\partial y}$
- Thus, we are computing JS for each of

$$S = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } S = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

- In vector forward mode, we can let $\dot{x} = [1\ 0]$ and $\dot{y} = [0\ 1]$ and compute the full Jacobian all at once, using $S = I$.

Seed Matrices: multiple vector inputs

- Consider `foo(in:x(3),in:y(3),out:f)`

- Then, to compute the full Jacobian using vector forward mode AD, we must initialize

$$\dot{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and

$$\dot{y} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Seed Matrices: Jv and $J^T v$ products

- Numerical algorithms may only need Jv and/or $J^T v$ products
- Can initialize S or W to v and compute this product at a small multiple of function cost
- AD tool may not support vector forward or vector reverse mode
- Can iterate through columns of S or W

Seed Matrices: Jacobian compression

Suppose the Jacobian is tridiagonal. That is,

$$J = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix}$$

Seed Matrices: Jacobian compression

Suppose the Jacobian is tridiagonal. Then, initializing the seed matrix S to

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Seed Matrices: Jacobian compression

yields

$$JS = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Seed Matrices: Jacobian compression

yields

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{34} & a_{32} & a_{33} \\ a_{44} & a_{45} & a_{43} \\ a_{54} & a_{55} & a_{56} \\ 0 & a_{65} & a_{66} \end{bmatrix}$$

Seed Matrices: Jacobian compression

- We can compute a tridiagonal Jacobian using a seed matrix with 3 columns, independent of N
- For general sparse J, we can perform a distance-2 coloring of the bipartite graph representing J to identify the *structurally orthogonal* columns of J
- Cost of computing the compressed Jacobian is proportional to the number of colors
- Special colorings for sparsity arising from stencil-based discretizations

Seed Matrices: Jacobian compression

- We can compute a tridiagonal Jacobian using a seed matrix with 3 columns, independent of N
- Cost of computing the compressed Jacobian is proportional to the number of colors

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{34} & a_{32} & a_{33} \\ a_{44} & a_{45} & a_{43} \\ a_{54} & a_{55} & a_{56} \\ 0 & a_{65} & a_{66} \end{bmatrix}$$