Sparsity Detection in Matlab

Cranfield UNIVERSITY

11th European Workshop on Automatic Differentiation
Cranfield University, Shrivenham
9th December 2010

Shaun Forth[1] and Anand Chaturvedi[2]

[1]Applied Mathematics & Scientific Computing, Cranfield University
[2]Indian Institute of Technology Rookee

Cranfield UNIVERSITY

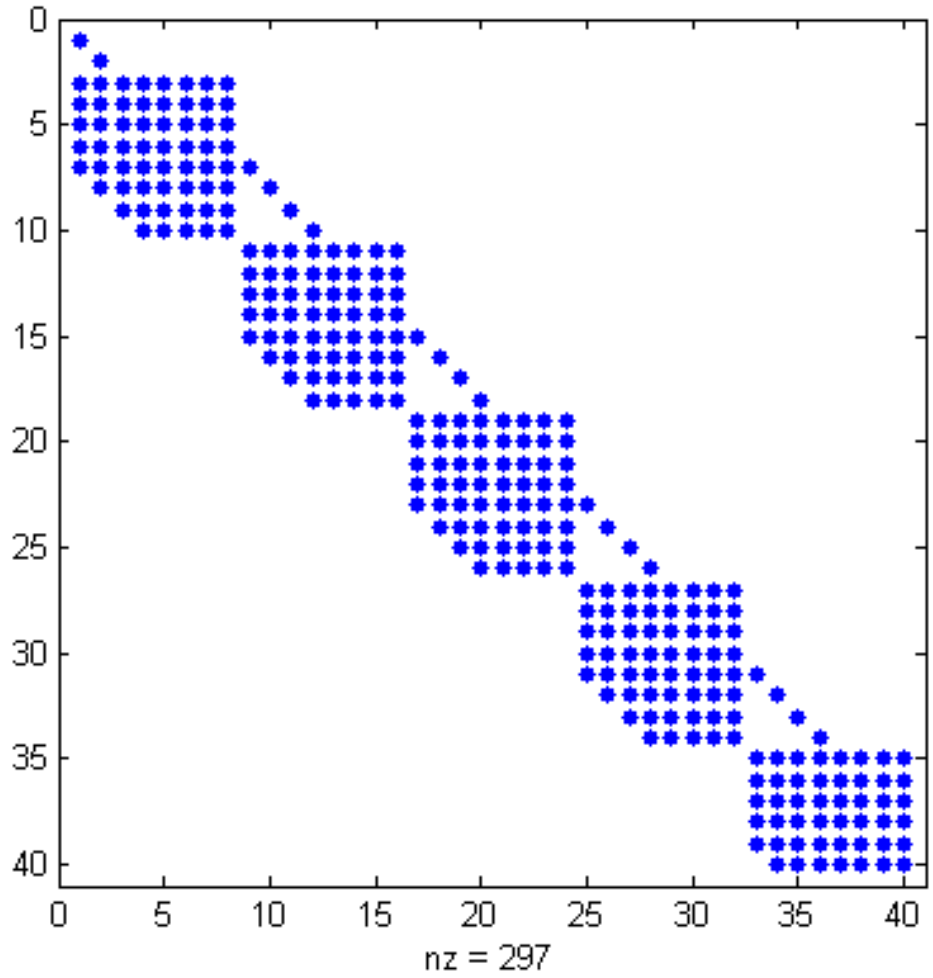
Outline

Introduction

- For a large sparse Jacobian compression techniques can greatly reduce the cost of determining a Jacobian's entries by AD [GW08, Chap.8].
- Such techniques require the Jacobian's sparsity pattern - or a tight over-estimate.
- Sophisticated approaches in ADOL-C [WG04] and TAF [GK06] for C/C++ and Fortran.
- By-product of use of sparse storage in forward mode AD [BKBC96, FK04].

Cranfield UNIVERSITY

Example (Minpack FIC Sparsity Pattern [ACMX92])



Sparsity pattern = location of Jacobian entries

Cranfield UNIVERSITY

Problem Definition

Definition (Sparsity Estimation)

Given a function $f : \mathbf{x} \in \mathbf{R}^{\mathbf{n}} \rightarrow \mathbf{y} \in \mathbb{R}^{\mathbf{m}}$ with Jacobian $\mathbf{Jf} \in \mathbb{R}^{\mathbf{m} \times \mathbf{n}}$ and **f** defined by a computer program then the sparsity estimation problem is to compute the locations (i, j) of all non-structurally zero elements (a.k.a. entries) of **Jf**.

- $\mathbf{Jf}_{i,j}$ is structurally zero if it is zero for all valid **x**.
- If the sparsity pattern is not valid for all valid independent variables **x** then this should be signalled.

Cranfield UNIVERSITY

Example (Structurally Zero)

Consider the trivial problem $\mathbf{y} = \mathbf{f}(\mathbf{x})$ with,

$$f_i = x_i^2,$$

and

$$\mathbf{Jf}_{i,j} = \begin{cases} 2x_i, & i = j, \\ 0, & i \neq j. \end{cases}$$

- Sparsity pattern is clearly the identity matrix.
- But if $x_i \equiv 0$ then $\mathbf{Jf}_{i,i} = 0$.
- Element $\mathbf{Jf}_{i,j}, i \neq j$ is structurally zero.
- Element $\mathbf{Jf}_{i,i}$ is not structurally zero.
- Element $\mathbf{Jf}_{i,i}$ may *accidentally* be zero when $x_i \equiv 0$.

n.b. Branching can give similar effects.

Cranfield UNIVERSITY

Approaches to Sparsity Detection

- By-Product of Sparse Forward Mode AD
- Using Bit Operations
- Verma's Matlab Implementation

Cranfield UNIVERSITY

By-Product of Sparse Forward Mode AD

ADiFor, MAD, ADO2 [BKBC96, FK04, PR98] may propagate derivative vectors using sparse storage.

- For each active variable's derivatives ∇v_k store only the non-zeros as a data structure[1],

$$\nabla v_k = \left\{ \left(j, \frac{\partial v_k}{\partial x_j} \right) : \frac{\partial v_k}{\partial x_j} \neq 0 \right\}.$$

- Such methods typically do not store a derivative if it is *accidentally* zero.
- Sparsity pattern may be an underestimate - unsafe.
- Work-around: randomize **x** to reduce likelihood of *accidents* [FK04] - resulting **Jf** of no practical use.

[1] or similar!

Cranfield UNIVERSITY

Using Bit Operations

The sparsity pattern $\hat{v}_k$ for any variable v_k may be defined by a bit pattern,

$$\hat{v}_k = \left\{ \hat{v}_k{}^j, j = 1, \ldots, n \in \{0,1\}^n \right\},$$

with

$$\hat{v}_k{}^j = \begin{cases} 0 & \text{if } \frac{\partial v_k}{\partial x_j} \text{ structurally zero} \\ 1 & \text{if } \frac{\partial v_k}{\partial x_j} \text{ not structurally zero} \end{cases}.$$

e.g. for $n = 3$ then

$$\begin{aligned} \hat{x}_1 &= \{1,0,0\} \\ \hat{x}_2 &= \{0,1,0\} \\ \hat{x}_3 &= \{0,0,1\} \\ \hat{v}_1 : v_1 = x_1 + x_2 &= \{1,1,0\} \end{aligned}$$

Cranfield UNIVERSITY

Using Bit Operations (ctd.)

For active scalars x, y and constant a [GK06],

- If $v = x \circ y$ with $\circ = +, -, *, /$ then,

$$\hat{v} = \hat{x}|\hat{y},$$

with $|$ the bit-wise or operation

$$\hat{v}^j = \begin{cases} 1 & \text{if } \hat{x}^j = 1 \text{ or } \hat{y}^j = 1, \\ 0 & \text{otherwise.} \end{cases}$$

- For $v = a \circ x$, or $x \circ a$, $a \neq 0$,

$$\hat{v} = \hat{x}.$$

Cranfield UNIVERSITY

Using Bit Operations (ctd.)

- Such bit-wise operations exist in C, C++, Fortran and Matlab for (unsigned) integer types.
- With 32 bit (4 byte) integers the sparsity pattern for up to 32 directional derivatives may be stored in a single integer.
- For n independent variables a vector of length $n/32$ can hold the entire sparsity pattern for a scalar.
- Just $n/32$ bit-wise or operations are required to combine such patterns in $\hat{x}|\hat{y}$.
- For 64 bit integers (8 bytes) use $n/64$ in above!
- TAF can produce sparsity patterns 2 orders of magnitude faster than Jacobians [GK06].

Cranfield UNIVERSITY

Verma's Matlab Implementation

Verma [Ver98] realised that sparsity propagation rules must be adapted to vector/matrix operations.

Example ($\mathbf{z} = \mathbf{I_n x}$)

- In Matlab `z = eye(n)*x`.
- If we don't account for zeros in $\mathbf{I_n}$ then Jacobian sparsity pattern is the overestimate,

$$\hat{\mathbf{J}}_{\mathbf{z}} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

- In operation $\mathbf{z} = \mathbf{A x}$ account for sparsity of $\mathbf{A}$.

Cranfield UNIVERSITY

Verma's Matlab Implementation (ctd.)

- What if matrix is active $\mathbf{z} = \mathbf{Y}\mathbf{x}$? Verma used sparsity of $\mathbf{Y}$'s value - but this might have entries that are accidentally zero.
- Verma stored sparsities using full arrays - performance overhead.

Cranfield UNIVERSITY

Preliminary MAD Implementations

- Account for Sparsity of Intermediate Variables
- Logical Arrays for Derivative Sparsity
- Bitwise Storage for Derivative Sparsity
- Branch Detection

Cranfield UNIVERSITY

Account for Sparsity of Intermediate Variables

Both our implementations have three components/properties:

- `value` - stores an object's numerical value.
- `entries` - a logical array indicating the location of structural non-zeros.
- `sparsity` - storage of derivative sparsity pattern - see later.

The `entries` component is updated as one might expect:

operation	entries operation
`z = x + y`	`z.entries = x.entries | y.entries`
`z = x .* y`	`z.entries = x.entries & y.entries`
`z = x * y`	`z.entries = logical(x.entries*y.entries)`

Cranfield UNIVERSITY

Logical Matrices for Derivative Sparsity

- MAD's `derivvec` class stores pattern for an object `v` of size $d_1 \times d_2 \times \cdots \times d_D$ as a $(d_1 \cdot d_2 \cdots d_D) \times n$ logical matrix.
- The logical matrix may be full or sparse.
- Storage requirement is
  - full: $4 \cdot d_1 \cdot d_2 \cdots d_D \cdot n$ bytes
  - sparse: $< 12n$ `nnz(v.entries)` bytes (underlying CSC).
- Sparsity propagation a la Verma but using an object's `entries` and not its `value`.

operation	entries operation
`z = x + y`	`z.sparsity = x.sparsity | y.sparsity`
`z = x .* y`	`z.sparsity = x.entries & y.sparsity` `| x.sparsity & y.entries`

Conformability - `derivvec` *replicates* `entries` n times.

Cranfield UNIVERSITY

Bitwise Storage for Derivative Sparsity

- `bitvec` class stores pattern for a $d_1 \times d_2 \times \cdots \times d_D$ object `v` as a $(d_1 \cdot d_2 \cdots d_D) \times \text{ceil}(n/32)$ `uint32` matrix.
- The integer matrix must be full - no sparse support.
- Storage requirement is $4 \cdot d_1 \cdot d_2 \cdots d_D \cdot \text{ceil}(n/32)$ bytes.
- Matlab R2010a has insufficient support for `uint64`.
- Sparsity propagation as for logical arrays.
- Conformability - `bitvec` *replicates* `entries` to ceil(n/32) `uint32` integers as required.

Cranfield UNIVERSITY

Branch Detection

Basic facility [Men10].

- Logs results of logical operations on active variables expected to control branching.
- Logs can be examined or successive logs compared for changes in control flow.
- Not fully tested.

Cranfield UNIVERSITY

Results

All test cases from the Minpack Collection [ACMX92].

- Flow in Channel - FIC
- Swirling Flow between Disks - SFD
- Combustion of Propane - CPF/CPR

Cranfield UNIVERSITY

Flow in Channel - FIC

	problem size n		
technique	8	64	2400
fmad	0.5	1.4	120.3
logical derivvec	0.5	1.4	65.6
bitvec	0.5	1.4	42.4

Cranfield UNIVERSITY

Swirling Flow between Disks - SFD

	problem size n			
technique	14	700	1400	2800
fmad	0.9	26.3	67.5	257.5
logical derivvec	0.9	24.6	51.8	110.7
bitvec	0.9	24.5	48.6	100.5

Cranfield UNIVERSITY

Combustion of Propane - CPF/CPR

	problem and size *n*	
	CPR	CPF
technique	5	11
fmad	0.31	0.33
logical derivvec	0.26	0.28
bitvec	0.25	0.28

Cranfield UNIVERSITY

Conclusions & Further Work

- It is possible to forward propagate sparsity patterns about 2.5 to 4 times faster than derivatives for moderate sized problems in Matlab.
- Presently, bitwise operations are slightly faster.
- Robustness requires:
  - Storage of an objects entries as well as its derivative sparsity pattern.
  - Branch detection.
- Scope for further improvements:
  - For efficiency of indexing and `|` and `&` operations logical `derivvec` internal storage should be transposed.
  - Improved Matlab support for `uint64`.
  - Hybrid approach using sparse bitwise storage - C/C++/Fortran coding?

Cranfield UNIVERSITY

References I

Brett M. Averick, Richard G. Carter, Jorge J. Moré, and Guo-Liang Xue.
The MINPACK-2 test problem collection.
Preprint MCS–P153–0692, ANL/MCS–TM–150, Rev. 1, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992.
See `ftp://info.mcs.anl.gov/pub/MINPACK-2/tprobs/P153.ps.Z`.

Christian H. Bischof, Peyvand M. Khademi, Ali Bouaricha, and Alan Carle.
Efficient computations of gradients and Jacobians by dynamic exploitation of sparsity in automatic differentiation.
Optimization Methods and Software, 7:1–39, 1996.

Shaun A. Forth and Robert Ketzscher.
High-level interfaces for the MAD (Matlab Automatic Differentiation) package.
In P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer, editors, *4th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS)*, volume 2. University of Jyväskylä, Department of Mathematical Information Technology, Finland, Jul 24–28 2004.
ISBN 951-39-1869-6.

Cranfield UNIVERSITY

References II

Ralf Giering and Thomas Kaminski.
Automatic Sparsity Detection implemented as a source-to-source transformation.
In Alexandrov, VN and VanAlbada, GD and Sloot, PMA and Dongarra, J, editor, *COMPUTATIONAL SCIENCE - ICCS 2006, PT 4, PROCEEDINGS*, volume 3994 of *LECTURE NOTES IN COMPUTER SCIENCE*, pages 591–598, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2006. SPRINGER-VERLAG BERLIN.
6th International Conference on Computational Science (ICCS 2006), Reading, ENGLAND, MAY 28-31, 2006.

Andreas Griewank and Andrea Walther.
Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.
Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

Marina Menshikova.
Uncertainty estimation using the moments method facilitated by automatic differentiation in Matlab.
PhD thesis, Cranfield University, Cranfield Defence and Security, Defence Academy of the UK, Shrivenham, Swindon, SN6 8LA, UK, 2010.
.

Cranfield UNIVERSITY

References III

J.D. Pryce and J.K. Reid.
ADO1, a Fortran 90 code for automatic differentiation.
Technical Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 OQX, England, 1998.
Available via `ftp://matisa.cc.rl.ac.uk/pub/reports/prRAL98057.ps.gz`.

Arun Verma.
ADMAT: Automatic differentiation in MATLAB using object oriented methods.
In *SIAM Interdisciplinary Workshop on Object Oriented Methods for Interoperability*, pages 174–183, Yorktown Heights, New York, Oct 21-23 1998. SIAM, National Science Foundation.

Andrea Walther and Andreas Griewank.
ADOL-C: computing higher-order derivatives and sparsity patterns for functions written in c++.
In *Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering - ECCOMAS 2004*, July 2004.
.