Developments in the MAD package

Shaun Forth

Cranfield University (Shrivenham Campus)
S.A.Forth@cranfield.ac.uk

1^{st} European Workshop on Automatic Differentiation,
Nice, France, April $14^{th} - 15^{th}$ 2005

Outline

1. MAD's Forward Mode [For04]
2. Differentiating Object-Oriented Code
3. Integration into TOMLAB
4. Sparse Matrices
5. Roadmap & Conclusions
6. References

Present MAD Release Contains

- `fmad` class - forward mode AD by operator overloading
- `derivvec` class - for storage and combination of multiple directional derivatives.
- `madutil` - directory of utility functions
- `madrecode` - directory of recoded MATLAB functions
- `usefulbits` - directory with sample `startup.m` initialisation file

fmad class constructor

- e.g. `x=fmad([1.1 2 3],[4 5 6]);`
- Defines `fmad` object with
  - `value` component - row vector `[1.1 2 3]`
  - `deriv` component - single directional derivative `[4 5 6]`
- Perform overloaded operations, e.g., element-wise multiplication via `times`

```
z=x.*x
value =
    1.2100    4.0000    9.0000
derivatives =
    8.8000   20.0000   36.0000
```

fmad class times function for z=x.*y

```
function z=times(x,y)
% FUNCTION: TIMES
% SYNOPSIS: elemental multiplication z=x.*y of one or more
if isa(x,'fmad')&isa(y,'fmad')
    z.value=x.value.*y.value;
    z.deriv=y.value.*x.deriv+x.value.*y.deriv;
elseif isa(x,'fmad')
    z.value=x.value.*y;
    z.deriv=y.*x.deriv;
else
    z.value=x.*y.value;
    z.deriv=x.*y.deriv;
end
z=class(z,'fmad');
```

Working with multiple directional derivatives

- What if we want the Jacobian?
- Seed derivatives with identity $\mathbf{I}_3$

```
x=fmad([1.1 2 3],eye(3));
```

- Overloaded operation with same times function gives

```
value =
    1.2100    4.0000    9.0000
Derivatives
Size = 1   3
No. of derivs = 3
derivs(:,:,1) = 2.2000         0         0
derivs(:,:,2) =      0     4     0
derivs(:,:,3) =      0     0     6
```

The fmad and derivvec classes

```
function xad=fmad(x,dx)
% FUNCTION: FMAD
% SYNOPSIS: Class constructor for forward Matlab AD objects
xad.value=x;
sx=size(xad.value);
sd=size(dx);
  if prod(sx)==prod(sd)
     xad.deriv=reshape(dx,sx);
  else
     xad.deriv=derivvec(dx,size(xad.value));
  end
```

If number of elements of supplied derivatives and value don't match then pass derivatives and value's size to derivvec

The derivvec class

- Store derivatives as a matrix with each directional derivative "unrolled" into a column.
- e.g. `derivvec(eye(3),[1 3])` derivatives stored as,

$$\begin{bmatrix} \text{direc 1} & \text{direc 2} & \text{direc 3} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The `times` operation of the `derivvec` class

- e.g. Need to calculate,

$$\begin{bmatrix} 1.1 & 2 & 3 \end{bmatrix} .* \begin{bmatrix} \text{direc 1} & \text{direc 2} & \text{direc 3} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

  with multiplication of each of the 3 directional derivatives.
- Convert value to column matrix and replicate 3 times

$$\begin{bmatrix} 1.1 & 1.1 & 1.1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} .* \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1.1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

  columns give required directional derivatives.

Accessor Functions

- Getting the value

```
getvalue(z)
ans =
      1.2100      4.0000      9.0000
```

- Getting "external representation" of derivatives

```
getderivs(z)
ans(:,:,1) =
      2.2000      0      0
ans(:,:,2) =
      0      4      0
ans(:,:,3) =
      0      0      6
```

Accessor Functions (ctd)

- Getting unrolled internal representation

```
>> getinternalderivs(z)
ans =
     2.2000         0         0
     0         4.0000         0
     0         0         6.0000
```

madutil functions

e.g.

- getvalue, getderivs, getinternalderivs for objects of class double.
- High-level interface functions for use in stiff ODE and optimisation solvers [FK04].
- MADcolor, MADgetseed and MADgetcompressedJac for colouring (row compression) a sparse Jacobian, generating the seed matrix and "uncompressing" the compressed Jacobian.

madrecode functions

Used for 2 reasons

- 1: Builtin MATLAB function is too complicated to manipulate its value and derivative components e.g. `filter` function
  - Code as MATLAB to let `fmad` differentiate it directly
  - Place in `madrecode` directory e.g. `madrecode/filter_mad`
  - Create `fmad` class function `filter` to call `filter_mad`
- 2: MATLAB supplied function not differentiable by `fmad`
  - Usually due to assignment of `fmad` object to part of double array e.g. `splncore`
  - Place copy of MATLAB function in `madrecode` directory e.g. `madrecode/splncore_mad`
  - Edit to ensure `fmad` can differentiate it
  - Create `fmad` class function `splncore` to call `splncore_mad`

Differentiating Object-Oriented Code

- User's code with classes, objects, overloaded operations?
- e.g., polynomials $p_1 = x^3 + 2x^2 + 3x + 4$ and $p_2 = 3x + 4$ and $p_3 = a_1 * p_1 + a_2 * p_2$ via `polynom` objects

```
a1=1; a2=2; x=1;
p1=polynom([1 2 3 4]); % class constructor call
p2=polynom([3 4]); % class constructor call
p3=a1*p1+a2*p2 % overloaded arithmetic
y=polyval(p3,x) % accessor function
```

- and gives

```
p3 =  x^3 + 2*x^2 + 9*x + 12
y =    24
```

Jacobian w.r.t. a_1, a_2

- Set derivatives of a_1, a_2 to be rows of I_2

```
a1=fmad(1,[1 0]); a2=fmad(2,[0 1]);
...
p3=a1*p1+a2*p2;
```

- Gives error

```
??? Function 'times' is not defined for values of class
Error in ==> times at 18
  [varargout{1:nargout}] = builtin('times', varargin{:}
Error in ==> fmad.mtimes at 38
           z.value=xval.*y;
```

- In `a1*p1` since first object is `fmad` then uses `fmad mtimes` operation.

Setting object precedence

- Must use `polynom` class operations ahead of `fmad` ones.
- Modify `polynom` class constructor

```
function p = polynom(a)
% some coding removed
   p.c = a(:).';
   p = class(p,'polynom');
superiorto('fmad') % added this line
```

- Now we get

```
p3=a1*p1+a2*p2; y=polyval(p3,x)
value = 24
Derivatives
Size = 1  1 No. of derivs = 2
derivs(:,:,1) = 10
derivs(:,:,2) = 7
```

Object-oriented code

- Technique of overloading user's objects or AD library objects used by other AD tools e.g. C++ templating in FADBAD [BS96].
- Success with `fmad` in differentiating one industrial application from chemical industry involving 6 classes.
- Need graceful exception handling.

TOMLAB

- TOMLAB [HE04, HGE04] is general purpose development environment in MATLAB for solution of optimisation problems.
- TOMLAB supplies:
  - MATLAB-coded solver algorithms
  - State-of-the-art optimisation software e.g., SNOPT [GMS05]
  - External solvers are distributed as compiled MEX files.
- MAD distributed as package [FE04] - single user academic license \$110 + \$22 / year support/upgrade

Integration into TOMLAB

- Integration performed by Kenneth Holmström and Marcus Edvall of TOMLAB
- Always uses `fmad`'s forward mode with sparse storage of derivatives
  - No need for sparsity pattern to be supplied or calculated.
  - Good performance over a wide range of problem sizes
- If user supplies gradient code then `fmad` can calculate the Hessian.
- For the user everything is automatic (provided it works!)

The Brown Problem

$$\text{minimise } f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[(x_i^2)^{x_{i+1}^2+1} + (x_{i+1}^2)^{x_i^2+1} \right],$$

with $n = 1000$ and $x_0 = [-1, 1, -1, 1, \ldots, 1]$ and supplied Hessian sparsity pattern.

Coding

```
load brownhstr; % Hessian sparsity pattern
n=1000; x_0=-ones(n,1); x_0(2:2:n,1)=1;
% set up TOMLAB Problem specification for FD
Prob = conAssign('brownf',[],[], Hstr,[],[],...
                  'Brown Problem', x_0);
disp('Using FD gradient')
ResultFD=tomRun('ucsolve',Prob,[],2);
% now use AD
Prob.ADObj=1; % turns on AD for 1st derivatives
disp('Using AD gradient')
ResultAD=tomRun('ucsolve',Prob,[],2);
```

Results

Using BFGS algorithm with default convergence conditions on Pentium IV laptop

Derivative Technique	Nonlinear Iterations	CPU time (s)
FD	7	26.1
`fmad`(sparse)	7	4.0

Sparse Matrices

- Sparse matrices are intrinsic to MATLAB
- Created by `sparse` function,

```
S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
S =
    (3,1)        1
    (2,2)        2
    (3,2)        3
    (4,3)        4
    (1,4)        5
```

- `fmad` uses `sparse` objects to store derivatives.
- Only recently enabled `fmad` to posses `sparse` values.

Dropping Problem

```
>> x=[3;4]; % create values
>> s=sparse([1;2],[1;2],x,2,2) % diagonal matrix
s =(1,1)        3
   (2,2)        4
>> s(1,1)=s(1,1)-3  % subtract 3 from element 1,1
s =
   (2,2)        4   % element is dropped
>> [i,j,val]=find(s)  % find nonzero elements
i = 2   j = 2   val = 4
```

The zero value is dropped

Differentiating the dropping problem

```
>> x=fmad([3;4],speye(2));
>> s=sparse([1;2],[1;2],x,2,2);
>> s(1,1)=s(1,1)-3;
>> [i,j,val]=find(s);
Warning: value and derivatives have different sparsity
> In fmad.find at 54
>> getvalue(i) = [1 2]'
>> getvalue(j) = [1 2]'
>> getvalue(val) = [0 4]'
>> getinternalderivs(val) =
   (1,1)          1
   (2,2)          1
```

Return zero values where we have nonzero derivatives.

- Similar to branching problem defining $y = f(x) = x$ by

```
if x==0
   y=0
else
   y=x
end
```

  for which AD gives $dy/dx = 0$ for $x = 0$.
- Dropping problem: for each element x_{ij} of the sparse matrix

```
if x(i,j)==0
   storage for x(i,j) is removed
end
```

- But `fmad` does not remove corresponding ∇x_{ij} (unless zero).
- Ramifications for reverse mode!

Roadmap

MAD

- Improve documentation.
- Complex valued functions w.r.t real dependents [PBC95]- engineering analysis.
- Reverse mode

MSAD

- Source transformation extension of MAD
- Uses optimised derivative operations of `fmad/derivvec`
- But with further efficiency advantages of source-transformation
- See Rahul Kharche's talk

Conclusions

- `fmad`/`derivvec` classes provide easy to use, efficient implementation of forward mode AD for first derivatives.
- Can handle coding of some complexity e.g. response surface fitting [RF04], racing car trajectory optimisation [Bra04], nonlinear control [CAS03].
- It appears users' object oriented code can be easily differentiated
- Robust enough to be commercially distributed

The 2^{nd} European AD Workshop

- Thursday November 17^{th} – Friday November 18^{th}
- Whitworth Conference Centre
  Royal Military College of Science
  Cranfield's Shrivenham Campus
  Shrivenham, Swindon
  Oxfordshire, UK
- 20 miles south of Oxford, 40 miles west of Heathrow
- Special sessions: AD in computational engineering, + ?
- Enquiries:
  - Admin `amor@rmcs.cranfield.ac.uk`
  - Programme `S.A.Forth@cranfield.ac.uk`

Damien Bradshaw.
The use of numerical optimisation to determine on-limit handling behaviour of race cars.
PhD thesis, School of Engineering, Department of Automotive, Mechanical and Structural Engineering, Cranfield University, Bedfordshire, MK43 0AL, UK, 2004.

Claus Bendtsen and Ole Stauning.
FADBAD, a flexible C++ package for automatic differentiation.
Technical Report IMM-REP-1996-17, Technical University of Denmark, IMM, Departement of Mathematical Modeling, Lyngby, 1996.

Y. Cao and R. Al-Seyab.
Nonlinear model predictive control using automatic differentiation.
In *European Control Conference (ECC 2003)*, pages CD–ROM, Cambridge, UK, September 2003.

Shaun A Forth and Marcus M. Edvall.
User Guide for MAD - MATLAB Automatic Differentiation Toolbox TOMLAB/MAD, Version 1.1 The Forward Mode.
TOMLAB Optimisation Inc., 855 Beech St 12, San Diego, CA 92101, USA, Jan 2004.
See `http://tomlab.biz/products/mad`.

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.

Shaun A. Forth.
An efficient overloaded implementation of forward mode automatic differentiation in MATLAB.

Submitted ACM TOMS, 2004.

Philip E. Gill, Walter Murray, and Michael A. Saunders.
SNOPT: An SQP algorithm for large-scale constrained optimization.
SIAM Review, 47(1):99–131, March 2005.

Kenneth Holmström and Marcus M. Edvall.
Chapter 19: The TOMLAB optimization environment.
In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, APPLIED OPTIMIZATION 88, ISBN 1-4020-7547-2, Boston/Dordrecht/London, January 2004. Kluwer Academic Publishers.

Kenneth Holmström, Anders O. Göran, and Marcus M. Edvall.
User's guide for TOMLAB 4.3.
TOMLAB Optimisation Inc., 855 Beech St 12, San Diego, CA 92101, USA, April 2004.
See `http://www.tomlab.biz`.

Gordon D. Pusch, Christian Bischof, and Alan Carle.
On automatic differentiation of codes with COMPLEX arithmetic with respect to real variables.
Technical Memorandum ANL/MCS-TM-188, Argonne National Laboratory, Mathematics and Computer Science Division, 9700 South Casss Avenue, Argonne, IL 60439, June 1995.

Trevor J. Ringrose and Shaun A. Forth.
Simplifying multivariate second order response surfaces by fitting constrained models using automatic differentiation.
Technometrics, Accepted, 2004.