A Matlab Implementation of the Minpack-2 Test Problem Collection

Dr Shaun Forth
S.A.Forth@cranfield.ac.uk

AD2016 - $12^{th} - 15^{th}$ **September 2016**

Outline

Introduction

1. Introduction
   - AD Packages in Matlab
   - The Minpack-2 Test Problem Collection

Introduction

- Grad [Rich and Hill, 1992] - forward mode AD on a Matlab string via Turbo-C
- ADMAT [Verma, 1999] - forward and reverse (via tape) mode AD using OO features of Matlab (+ second order derivatives and sparsity detection)
- ADiMat [Bischof et al., 2003] - source transformed forward/reverse mode
- MAD [Forth, 2006] - optimised derivatives storage class `derivvec` to give improved overloaded forward mode performance over ADMAT
- MSAD [Kharche and Forth, 2006] - source-transformation by specialising and inlining MAD's derivative objects.
- ADiGator [Patterson et al., 2013, Weinstein and Rao, 2015] - source-transformed, sparsity-exploiting forward mode AD (vertex elimination with forward ordering [Griewank and Reese, 1991]).

Need for testcases.

The Minpack-2 Test Problem Collection [Averick et al., 1991]

- Describes 24 optimisation problems, including Fortran 77 source code, of three problem types:

unconstrained minimisation	- objective function, gradient, Hessian
least squares minimisation	- residual, Jacobian
systems of nonlinear equations	- residual, Jacobian

- For large-scale problems source code for Hessian/Jacobian sparsity pattern and Hessian/Jacobian-vector products also provided.
- Appears 82 times in Scopus
- Widely used for AD tool validation, eg, Bischof et al. [1996], Walther and Griewank [2004], Naumann and Utke [2005], Giering and Kaminski [2006], Shin and Hovland [2007], Forth et al. [2004].

Matlab Implementation of Minpack-2 Problems

2. Matlab Implementation of Minpack-2 Problems
   - Converting Minpack-2 to Matlab
   - Re-coding Minpack-2 in Matlab

Converting Minpack-2 to Matlab

- Lenton [2005] hand-converted all the Minpack problems to Matlab
  - Changes of syntax, array constructors
  - Arrays must have lower index 1 in Matlab (also affects loop indices)
- Validated by
  - Fortran program creates random vectors $\mathbf{x}$, calls Fortran Minpack, writes $f(\mathbf{x})$, ∇f, ... to formatted text file.
  - Text file read into Matlab and results compared to those from Matlab version of Minpack calls
  - Results agree to within i/o and floating point round-off

Re-coding Minpack-2 in Matlab

- Lenton's conversion satisfactory for small-scale, fixed *n* problems.
- For large scale problems Lenton's Fortran-based coding uses loops and subscripting operations.
- Large number of overloaded function calls in overloaded AD.
- Re-coded Minpack-2 problems using array operations to give a vectorised version.
- Uniform interface to all functions with problem specific parameters supplied in a structure.

Re-coding Minpack-2 in Matlab (ctd.)

- Separate functions for:
  - Setting problem parameters - constants, standard start vector, etc
  - Function evaluation only + vectorised version where appropriate.
  - Function + Gradient/Jacobian
  - Gradient/Jacobian only
  - Jacobian/Hessian-vector product
  - Sparsity pattern.
- Use Matlab's Code Analyzer to eliminate: dead code, unused variables.
- Hand-coded adjoint for gradients.

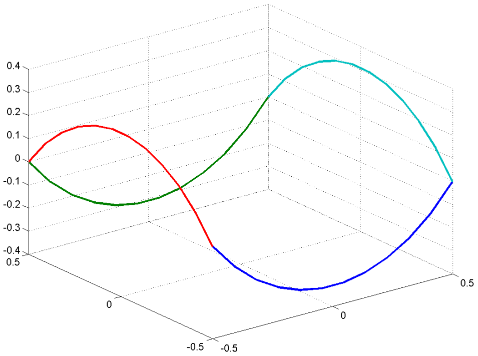
Example

3 Example

- Minimal Surface Area (MSA) Problem

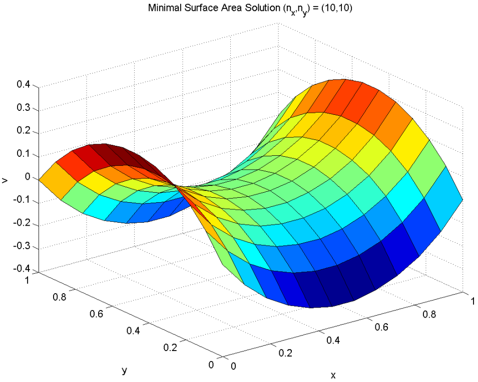
Minimal Surface Area (MSA) Problem

- Supply height on x and y boundaries of $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$



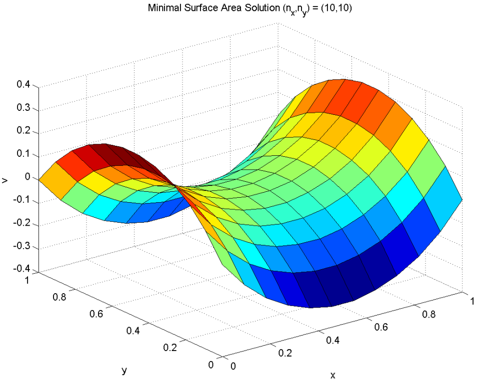
Minimal Surface Area (MSA) Problem

- Supply height on x and y boundaries of $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$
- Determine surface $u(x, y)$ of minimal surface area with given boundaries.

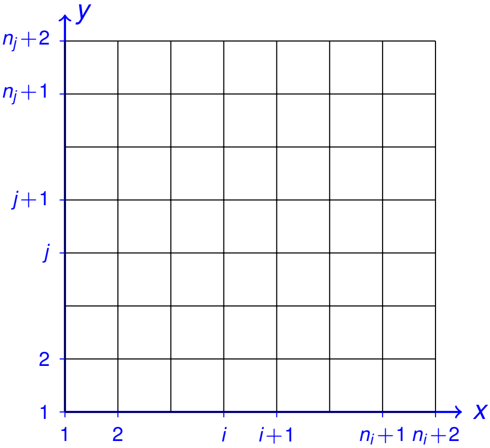


Minimal Surface Area (MSA) Problem

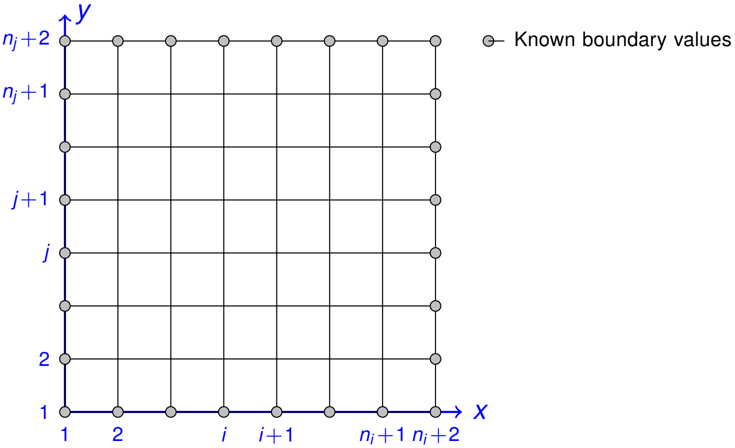
- Supply height on x and y boundaries of $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$
- Determine surface $u(x, y)$ of minimal surface area with given boundaries.
- Known analytic solution due to Enneper.



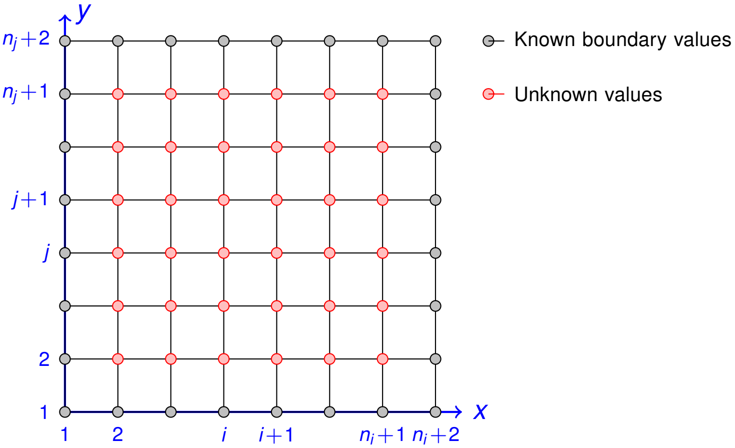
MSA - Finite Difference Formulation



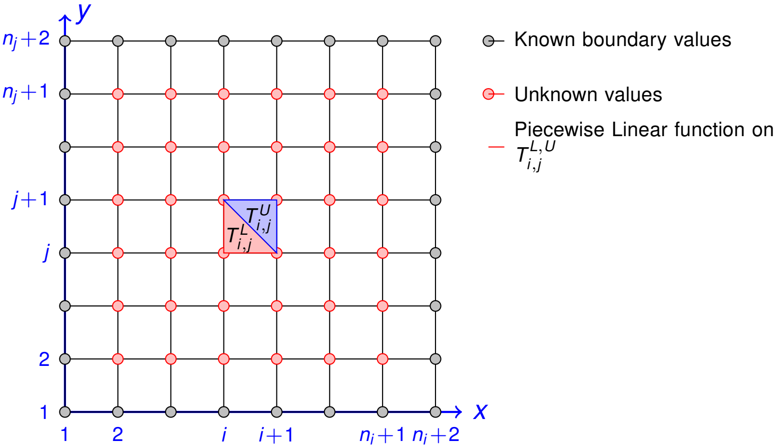
MSA - Finite Difference Formulation



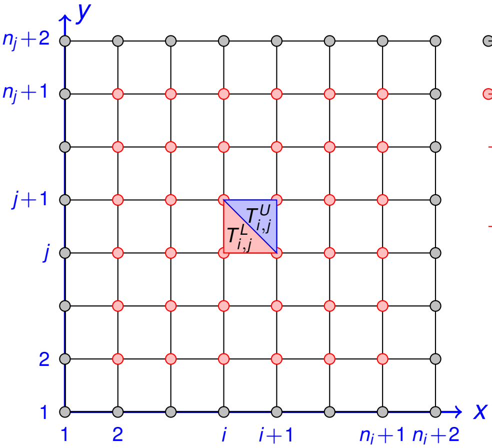
MSA - Finite Difference Formulation



MSA - Finite Difference Formulation



MSA - Finite Difference Formulation



- Known boundary values
- Unknown values
- Piecewise Linear function on $T^{L,U}_{i,j}$
- Min $f(u) = \sum_{i=1}^{n_i+1} \sum_{j=1}^{n_j+1} \left(f^L_{i,j} + f^U_{i,j} \right)$

MSA - Finite Difference Formulation (ctd.)

Minimise,

$$f(u) = \sum_{i=1}^{n_i+1} \sum_{j=1}^{n_j+1} \left(f_{i,j}^L + f_{i,j}^U \right),$$

with $f_{i,j}^{L,U}$ the surface area on the lower/upper triangles:

$$\begin{aligned} f_{i,j}^L &= \frac{h_x h_y}{2} \left\{ 1 + \left(\frac{u_{i+1,j} - u_{i,j}}{h_x} \right)^2 + \left(\frac{u_{i,j+1} - u_{i,j}}{h_y} \right)^2 \right\}^{\frac{1}{2}} \\ f_{i,j}^U &= \frac{h_x h_y}{2} \left\{ 1 + \left(\frac{u_{i+1,j+1} - u_{i,j+1}}{h_x} \right)^2 + \left(\frac{u_{i+1,j+1} - u_{i+1,j}}{h_y} \right)^2 \right\}^{\frac{1}{2}} \end{aligned}$$

and we only consider $u_{i,j}$ with $i = 2, \ldots, n_i + 1$, $j = 2, \ldots, n_j + 1$.

MSA - Fortran Coding (much abbreviated)

```
      subroutine dmsafg(nx,ny,x,f,fgrad,task,bottom,top,left,right)
:
c     function and gradient over the lower triangular elements.
       do 50 j = 0, ny
          do 40 i = 0, nx
             k = nx*(j-1) + i ! 1-D indexing
             if (i .ge. 1 .and. j .ge. 1) then
                v = x(k) ! first vertex in triangle
             else
                if (j .eq. 0) v = bottom(i+1)
:
            if (i .lt. nx .and. j .gt. 0) then
               vr = x(k+1) ! right vertex
:
            dvdx = (vr-v)/hx
            dvdy = (vt-v)/hy
            fl = sqrt(one+dvdx**2+dvdy**2)
            if (feval) f = f + fl
           if (geval) then
```

MSA - Lenton's Matlab Conversion

```
function varargout=dmsafg(nx,ny,x,task,bottom,top,left,right)
:
switch task
    case {'F','G','FG'}
        for j = 0:ny
            for i = 0:nx
                k = nx*(j-1) + i; % 1-D indexing
                if i >= 1 && j >= 1
                    v = x(k); % first vertex in triangle
                else if j == 0
                        v = bottom(i+1);
                :
                if i < nx && j > 0
                    vr = x(k+1); % right vertex
                :
                dvdx = (vr-v)/hx;
                dvdy = (vt-v)/hy;
                fl = sqrt(1+dvdx^2+dvdy^2);
                if geval
```

MSA - Re-Coding the Problem Definition

Structure `Prob` used to store all problem data

```
function [Prob,nuse]=MinpackMSA_Prob(varargin)
%   [Prob,nuse]=MinpackMSA_Prob(nx,ny,bottom,top,left,right,compat)
% check/set boundary conditions
if isempty(bottom)
    bottom=Enneper('bottom',nx,ny);
elseif ~(isvector(bottom)&&length(bottom)==nx+2)
    error(['MinPackMSA_Prob: bottom must be a length nx+2 = ',...
end
:
% Compute the standard starting point
x_0 = reshape((top(2:nx+1)*alpha + bottom(2:nx+1)*(1-alpha) +...
Prob.x_0=x_0;
Prob.user.nx=nx;
:
nuse=nx*ny;

function bcvec=Enneper(bc,nx,ny)
:
```

MSA - Re-Coding the Function Definition

Regularise the interface to use `Prob` structure

```
function f = MinpackMSA_F(x,Prob)
:
bottom=Prob.user.bottom;
top   =Prob.user.top;
left  =Prob.user.left;
right =Prob.user.right;
```

otherwise similar to Lenton coding with loops and branching.

MSA - Vectorising the Function Definition

```
function f = MinpackMSA_Fvec(x,Prob)
:
bottom=Prob.user.bottom;
:
% transfer interior values x to entire grid v
v = zeros(nx+2,ny+2,'like',x); % 'like' ensures v has class of x
v(2:nx+1,2:ny+1) = reshape(x,nx,ny);
% apply boundary conditions
v(:,1)    = bottom;
:
% computer dvdx and dvdy on each edge of the grid
dvdx = (v(2:nx+2,:)-v(1:nx+1,:))/hx;
dvdy = (v(:,2:ny+2)-v(:,1:ny+1))/hy;
% quadratic term over lower and upper elements
fL=sqrt(1+dvdx(1:nx+1,1:ny+1).^2+dvdy(1:nx+1,1:ny+1).^2);
fU=sqrt(1+dvdx(1:nx+1,2:ny+2).^2+dvdy(2:nx+2,1:ny+1).^2);
f = area*(sum(sum(fL+fU)));
```

No loops or branches!

MSA - Vectorised Gradient

From vectorised function - easy (!) to write vectorised gradient

```
function fgrad = MinpackMSA_Gvec(x,Prob)
% coding as for MinpackMSA_Fvec
:
% quadratic term over lower and upper elements
fL=sqrt(1+dvdx(1:nx+1,1:ny+1).^2+dvdy(1:nx+1,1:ny+1).^2);
fU=sqrt(1+dvdx(1:nx+1,2:ny+2).^2+dvdy(2:nx+2,1:ny+1).^2);
% gradient just use interior points
i=2:nx+1;
j=2:ny+1;
fgrad=area*(...
    (1/hx)*(1./fL(i-1,j)+1./fU(i-1,j-1)).*dvdx(i-1,j)...
    -(1/hx)*(1./fL(i,j)+1./fU(i,j-1)).*dvdx(i,j)...
    +(1/hy)*(1./fL(i,j-1)+1./fU(i-1,j-1)).*dvdy(i,j-1)...
    -(1/hy)*(1./fL(i,j)+1./fU(i-1,j)).*dvdy(i,j));
fgrad=fgrad(:);
```

Testing

4 Testing

Testing

- Unit testing describe at Euro AD Workshop in Paderborn
- Validate Matlab coding against original Fortran coding.

Interfacing with Optimization Toolboxes

5 Interfacing with Optimization Toolboxes
- Interfacing with TOMLAB
- Interfacing with Matlab Optimization Toolbox

Interfacing with TOMLAB [Kenneth Holmström et al., 2010]

- Set up the Minpack problem

```
n = 100;
Prob = MinpackMSA_Prob(n,'Tomlab');
```

- Using the Minpack Function and Gradient

```
Result = ucSolve(Prob);
```

- Or, using the Minpack Function and MAD Gradient

```
Prob.FUNCS.g=[];
options = struct; options.derivatives = 'automatic';
Result = ucSolve(Prob,options);
```

- Obtain the optimal value

```
x = Result.x_k;
f = Result.f_k;
```

Interfacing with Matlab Optimization Toolbox [Mathworks, 2016]

- Set up the Minpack problem

```
n = 100;
Prob = MinpackMSA_Prob(n,'Tomlab');
```

- Using the Minpack Function and Gradient

```
options = optimoptions('fminunc','Algorithm',...
    'quasi-Newton','SpecifyObjectiveGradient',true);
[x,f]=fminunc(@(x)MinpackMSA_FG(x,Prob),Prob.x_0,options);
```

- Or, using the Minpack Function and MAD Gradient [Forth and Ketzscher, 2004]

```
options=optimset(@fminunc);
options = optimsetMAD(options,'GradObj','fmadsparse');
[x,f]=fminuncMAD(@(x)MinpackMSA_F(x,Prob),...
     Prob.x_0,options);
```

Results

6 Results

- MSA Problem
- MSA - Minimization Performance

Results

- Processor - Intel Core i5-4300U CPU (1.9-2.5GHz) with 4 GB RAM
- OS - Windows 7
- Matlab - release 2016a
- Multi-threading not exploited.

MSA Problem - Run time Ratios

			Problem size		
what	how	vectorized	100	400	1600
F	Minpack	no	1	1	1
F	Minpack	yes	1.00	0.75	0.38
F+G	Minpack	no	3.00	2.00	1.75
F+G	Minpack	yes	4.00	1.50	0.81
F+G	`fmad-sparse`	no	16000.00	24250.00	23125.00
F+G	`fmad-sparse`	yes	220.00	145.00	106.25

- Function CPU times = $0.078ms$, $0.078ms$ and $0.25ms$
- Minpack timings averaged over 1000 evaluations, `fmad` over 1 and `fmad-sparse` over 50.

MSA - Minimization Performance

Chose solvers with BFGS updates to inverse Hessian:

- Tomlab `ucSolve` [Kenneth Holmström et al., 2010]
- Matlab Optimization Toolbox `fminunc` [Mathworks, 2016]
- MAD interface `fminuncMAD` to `fminunc` [Forth and Ketzscher, 2004]

All use Minpack standard start point, default tolerances and converge to same solution.

MSA - Minimization Performance Run Times (s)

			Problem size		
solver	gradient	vectorized	100	400	1600
`ucSolve`	Minpack	no	0.50	0.91	41.31
`ucSolve`	Minpack	yes	0.16	0.87	38.78
`ucSolve`	FD	no	0.41	3.49	49.64
`ucSolve`	FD	yes	0.62	2.98	31.48
`ucSolve`	`fmad-sparse`	no	0.36	3.34	51.03
`ucSolve`	`fmad-sparse`	yes	0.44	3.01	31.25
`fminunc`	Minpack	no	0.07	0.28	6.93
`fminunc`	Minpack	yes	0.05	0.23	6.77
`fminunc`	FD	no	0.20	2.26	54.52
`fminunc`	FD	yes	0.25	1.86	29.00
`fminuncMAD`	`fmad-sparse`	no	182.71	-	-
`fminuncMAD`	`fmad-sparse`	yes	3.70	19.53	329.41

Conclusions

7 Conclusions

Conclusions

- We have a validated (well nearly) Matlab implementation of all problems in the Minpack-2 Test Problem Collection.
- Re-coded version with uniform interface and an implementation of the function alone to allow testing of AD for
  - First Derivatives for Jacobians and gradients
  - Sparsity Patterns for Jacobians and Hessians
  - Jacobian-vector and Hessian-vector products
  - Impact of code vectorisation.
- Code vectorisation will have a major impact on efficiency of overloaded and source transformed AD codes - including compile-time costs for source transformation.
- Code vectorisation almost always beneficial to performance and crucial for overloaded AD.

Further Work

8 Further Work

Further Work

- Package up our Matlab version of the Minpack-2 problems for users in AD and Optimization.
- Present `fmad` class uses obsolete Matlab OO coding practices - recoding should allow JIT acceleration to be applied.
- Reverse mode AD in MAD?

References I

B. M. Averick, R. G. Carter, and J. J. More. The Minpack-2 Test Problem Collection. Technical Memorandum TM 150, Argonne National Laborartory, Mathematics and Computer Science Division, 9700 South Cass Avenue, Argonne, Illinois 60439, USA, May 1991. URL `ftp://ftp.mcs.anl.gov/pub/MINPACK-2/tprobs.92/P153.ps.Z`.

C. Bischof, P. Khademi, A. Bouaricha, and A. Carle. Efficient computation of gradients and jacobians by dynamic exploitation of sparsity in automatic differentiation. *Optimization Methods and Software*, 7(1):1–39, 1996. ISSN 1055-6788.

C. Bischof, B. Lang, and A. Vehreschild. Automatic Differentiation for MATLAB Programs. *Proceedings in Applied Mathematics and Mechanics*, 2(1):50–53, Mar. 2003. ISSN 1617-7061. doi: 10.1002/pamm.200310013. URL `http://onlinelibrary.wiley.com/doi/10.1002/pamm.200310013/abstract`.

S. Forth, M. Tadjouddine, J. Pryce, and J. Reid. Jacobian code generated by source transformation and vertex elimination can be as efficient as hand-coding. *ACM Transactions on Mathematical Software*, 30(3):266–299, 2004. ISSN 0098-3500. doi: 10.1145/1024074.1024076.

S. A. Forth. An efficient overloaded implementation of forward mode automatic differentiation in MATLAB. *ACM Trans. Math. Softw.*, 32(2):195–222, June 2006. ISSN 0098-3500. doi: 10.1145/1141885.1141888. URL `http://doi.acm.org/10.1145/1141885.1141888`.

References II

S. A. Forth and R. Ketzscher. High-Level Interfaces for the MAD (Matlab Automatic Differentiation) Package. In P. Neittaanmaki, T. Rossi, S. Korotov, E. Onate, and J. Periaux, editors, *4th European Congress on Computational Methods in Applied Sciences & Engineering (ECCOMAS)*, Jyvaskyla, Finland, July 2004. URL `http://www.mit.jyu.fi/eccomas2004/proceedings/pdf/641.pdf`.

R. Giering and T. Kaminski. *Automatic Sparsity Detection implemented as a source-to-source transformation*, volume 3994 LNCS - IV of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2006. ISBN 3540343857.

A. Griewank and S. Reese. On the Calculation of Jacobian Matrices by the Markowitz Rule. In A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms - Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, PA, 1991. ISBN 0–89871–284–X.

Kenneth Holmström, Anders O. Göran, and Marcus M. Edvall. Users Guide for TOMLAB 7. Technical report, TOMLAB Optimization, May 2010. URL `http://tomopt.com/docs/TOMLAB.pdf`.

R. Kharche and S. Forth. Source transformation for MATLAB automatic differentiation. In V. Alexandrov, G. van Albada, P. Sloot, and J. Dongarra, editors, *Computational Science – ICCS 2006*, volume 4 of *Lecture Notes in Computer Science*, pages 558–565. 2006. ISBN 978-3-540-34385-1. URL `http://link.springer.com/chapter/10.1007%2F11758549_77`.

References III

K. Lenton. *An Efficient, Validated Implementation of the MINPACK-2 Test Problem Collection in MATLAB*. MSc Dissertation, Cranfield University, Applied Mathematics \& Operational Research Group, Engineering Systems Department, RMCS Shrivenham, Swindon SN6 8LA, UK, 2005.

Mathworks. Optimization Toolbox - User's Guide. Technical Report R2016a, The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098, 2016. URL `http://www.mathworks.com/help/pdf_doc/optim/optim_tb.pdf`.

U. Naumann and J. Utke. Optimality-preserving elimination of linearities in Jacobian accumulation. *Electronic Transactions on Numerical Analysis*, 21:134–150, 2005. ISSN 1068-9613.

M. A. Patterson, M. Weinstein, and A. V. Rao. An efficient overloaded method for computing derivatives of mathematical functions in MATLAB. *ACM Trans. Math. Softw.*, 39(3):17:1–17:36, May 2013. ISSN 0098-3500. doi: 10.1145/2450153.2450155. URL `http://doi.acm.org/10.1145/2450153.2450155`.

L. Rich and D. Hill. Automatic differentiation in MATLAB. *Applied Numerical Mathematics*, 9(1):33–43, 1992. ISSN 0168-9274. doi: 10.1016/0168-9274(92)90065-L.

J. Shin and P. Hovland. Comparison of two activity analyses for automatic differentiation: Context-sensitive flow-insensitive vs. context-insensitive flow-sensitive. *Proceedings of the ACM Symposium on Applied Computing*, pages 1323–1329, 2007. doi: 10.1145/1244002.1244287.

References IV

A. Verma. ADMAT: Automatic Differentiation in MATLAB Using Object Oriented Methods. In M. E. Henderson, C. R. Anderson, and S. L. Lyons, editors, *Object Oriented Methods for Interoperable Scientific and Engineering Computing: Proceedings of the 1998 SIAM Workshop*, pages 174–183. SIAM, Philadelphia, 1999.

A. Walther and A. Griewank. ADOL-C: Computing higher-order derivatives and sparsity pattern for functions written in C/C++. 2004.

M. J. Weinstein and A. V. Rao. ADiGator, a Toolbox for the Algorithmic Differentiation of Mathematical Functions in MATLAB Using Source Transformation via Operator Overloading. *ACM Transactions on Mathematical Software*, Submitted, 2015. URL `http://vdol.mae.ufl.edu/SubmittedJournalPublications/adigator-CALGO.pdf`.

Cranfield
Defence and
Security

www.cranfield.ac.uk

T: +44 (0)1793 785810

@cranfielduni
@cranfielduni
/cranfielduni