A Matlab Implementation of the Minpack-2 Test Problem Collection

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

AD2016 - 12\textsuperscript{th} – 15\textsuperscript{th} September 2016

www.cranfield.ac.uk
Outline

1. Introduction
2. Matlab Implementation of Minpack-2 Problems
3. Example
4. Testing
5. Interfacing with Optimization Toolboxes
6. Results
7. Conclusions
8. Further Work
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

- 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 \( f(\mathbf{x}), \nabla f, \ldots \) 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.
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

- Minimal Surface Area (MSA) Problem
Supply height on $x$ and $y$ boundaries of $\left[-\frac{1}{2}, \frac{1}{2}\right] \times \left[-\frac{1}{2}, \frac{1}{2}\right]$
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.
Known boundary values
Unknown values

\[ T_L^{i,j}, \quad T_U^{i,j} \]

Piecewise Linear function on 

\[ \text{Min} f(u) = \sum_{i=1}^{n_i+1} \sum_{j=1}^{n_j+1} (f_L^{i,j} + f_U^{i,j}) \]
Known boundary values
Known boundary values

Unknown values

Min $f(u) = \sum_{i=1}^{n_i+1} \sum_{j=1}^{n_j+1} (f_{L_{ij}} + f_{U_{ij}})$
MSA - Finite Difference Formulation

[Diagram with grid and labels for indices and boundary conditions]

- Known boundary values
- Unknown values
- Piecewise Linear function on $T_{i,j}^{L,U}$
Known boundary values

Unknown values

Piecewise Linear function on $T_{i,j}^{U}$

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

\[ f(u) = \sum_{i=1}^{n_{i+1}} \sum_{j=1}^{n_{j+1}} (f_{i,j}^L + f_{i,j}^U) , \]

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

\[
f_{i,j}^L = \frac{h_x h_y}{2} \left\{ 1 + \left( \frac{u_{i+1,j} - u_{i,j}}{h_x} \right)^2 + \left( \frac{u_{i,j+1} - u_{i,j}}{h_y} \right)^2 \right\}^{1/2}
\]

\[
f_{i,j}^U = \frac{h_x h_y}{2} \left\{ 1 + \left( \frac{u_{i+1,j+1} - u_{i,j+1}}{h_x} \right)^2 + \left( \frac{u_{i+1,j+1} - u_{i+1,j}}{h_y} \right)^2 \right\}^{1/2}
\]

and we only consider \( u_{i,j} \) with \( i = 2, \ldots, n_i + 1 \), \( j = 2, \ldots, n_j + 1 \).
subroutine dmsafg(nx,ny,x,f,fgrad,task,bottom,top,left,right)

! 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)
    end if
    if (i .lt. nx .and. j .gt. 0) then
      vr = x(k+1) ! right vertex
    end if
    dvdx = (vr-v)/hx
    dvdy = (vt-v)/hy
    fl = sqrt(one+dvdx**2+dvdy**2)
    if (feval) f = f + fl
    if (geval) then
      ! code
    end if
  end do 40
end do 50
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);
                end
                else if j == 0
                    v = bottom(i+1);
                end
                if i < nx && j > 0
                    vr = x(k+1); % right vertex
                end
                dvdx = (vr-v)/hx;
                dvdy = (vt-v)/hy;
                fl = sqrt(1+dvdx^2+dvdy^2);
                if geval

15
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)
:
Regularise the interface to use `Prob` structure

```matlab
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.
```
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!
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

- Unit testing described at Euro AD Workshop in Paderborn
- Validate Matlab coding against original Fortran coding.
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
  
  \[
  n = 100; \\
  \text{Prob} = \text{MinpackMSA\_Prob}(n, 'Tomlab'); \\
  \]

- Using the Minpack Function and Gradient
  
  \[
  \text{Result} = \text{ucSolve}(\text{Prob}); \\
  \]

- Or, using the Minpack Function and MAD Gradient
  
  \[
  \text{Prob}\_\text{FUNCS}\_\text{g}=[]; \\
  \text{options} = \text{struct}; \text{options}\_\text{derivatives} = '\text{automatic}'; \\
  \text{Result} = \text{ucSolve}(\text{Prob}, \text{options}); \\
  \]

- Obtain the optimal value
  
  \[
  x = \text{Result}\_x\_k; \\
  f = \text{Result}\_f\_k; \\
  \]
Interfacing with Matlab Optimization Toolbox [Mathworks, 2016]

- Set up the Minpack problem
  
  ```matlab
  n = 100;
  Prob = MinpackMSA_Prob(n,'Tomlab');
  ```

- Using the Minpack Function and Gradient
  
  ```matlab
  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]
  
  ```matlab
  options=optimset(@fminunc);
  options = optimsetMAD(options,‘GradObj’,’fmadpsparse’);
  [x,f]=fminuncMAD(@(x)MinpackMSA_F(x,Prob),...
   Prob.x_0,options);
  ```
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

<table>
<thead>
<tr>
<th>what</th>
<th>how</th>
<th>vectorized</th>
<th>Problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>F</td>
<td>Minpack</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>Minpack</td>
<td>yes</td>
<td>1.00</td>
</tr>
<tr>
<td>F+G</td>
<td>Minpack</td>
<td>no</td>
<td>3.00</td>
</tr>
<tr>
<td>F+G</td>
<td>Minpack</td>
<td>yes</td>
<td>4.00</td>
</tr>
<tr>
<td>F+G</td>
<td>fmad-sparse</td>
<td>no</td>
<td>16000.00</td>
</tr>
<tr>
<td>F+G</td>
<td>fmad-sparse</td>
<td>yes</td>
<td>220.00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>solver</th>
<th>gradient</th>
<th>vectorized</th>
<th>100</th>
<th>400</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucSolve</td>
<td>Minpack</td>
<td>no</td>
<td>0.50</td>
<td>0.91</td>
<td>41.31</td>
</tr>
<tr>
<td>ucSolve</td>
<td>Minpack</td>
<td>yes</td>
<td>0.16</td>
<td>0.87</td>
<td>38.78</td>
</tr>
<tr>
<td>ucSolve</td>
<td>FD</td>
<td>no</td>
<td>0.41</td>
<td>3.49</td>
<td>49.64</td>
</tr>
<tr>
<td>ucSolve</td>
<td>FD</td>
<td>yes</td>
<td>0.62</td>
<td>2.98</td>
<td>31.48</td>
</tr>
<tr>
<td>ucSolve</td>
<td>fmad-sparse</td>
<td>no</td>
<td>0.36</td>
<td>3.34</td>
<td>51.03</td>
</tr>
<tr>
<td>ucSolve</td>
<td>fmad-sparse</td>
<td>yes</td>
<td>0.44</td>
<td>3.01</td>
<td>31.25</td>
</tr>
<tr>
<td>fminunc</td>
<td>Minpack</td>
<td>no</td>
<td>0.07</td>
<td>0.28</td>
<td>6.93</td>
</tr>
<tr>
<td>fminunc</td>
<td>Minpack</td>
<td>yes</td>
<td>0.05</td>
<td>0.23</td>
<td>6.77</td>
</tr>
<tr>
<td>fminunc</td>
<td>FD</td>
<td>no</td>
<td>0.20</td>
<td>2.26</td>
<td>54.52</td>
</tr>
<tr>
<td>fminunc</td>
<td>FD</td>
<td>yes</td>
<td>0.25</td>
<td>1.86</td>
<td>29.00</td>
</tr>
<tr>
<td>fminuncMAD</td>
<td>fmad-sparse</td>
<td>no</td>
<td>182.71</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fminuncMAD</td>
<td>fmad-sparse</td>
<td>yes</td>
<td>3.70</td>
<td>19.53</td>
<td>329.41</td>
</tr>
</tbody>
</table>
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
Further Work

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


