R. Hiptmair
A. Moiola

Numerical Methods for CSE

ETH Zürich
D-MATH

Examination – Solutions

August 11th, 2011

Total points: 90 = 16+25+16+13+20

Problem 1. Structured matrix-vector product [16=2+6+2+2+4 pts]

(1a) Matrix–vector multiplication: quadratic dependence $O(n^2)$.

(1b) For every j we have $y_j = \sum_{k=1}^{j} kx_k + j\sum_{k=j+1}^{n} x_k$, so we precompute the two terms for every j only once.

```
function y = multAmin(x)
% O(n), slow version
n = length(x);
y = zeros(n,1);
v = zeros(n,1);
w = zeros(n,1);

v(1) = x(n);
w(1) = x(1);
for j = 2:n
    v(j) = v(j-1)+x(n+1-j);
    w(j) = w(j-1)+j*x(j);
end
for j = 1:n-1
    y(j) = w(j) + v(n-j)*j;
end
y(n) = w(n);

%    To check the code, run:
% n=500; x=randn(n,1); y = multAmin(x);
% norm(y - min(ones(n,1)*(1:n), (1:n)'*ones(1,n)) * x)
```

Better version, using `cumsum` to avoid the for loops:

```
function y = multAmin2(x)
% O(n), no-for version
n = length(x);
v = cumsum(x(end:-1:1));
w = cumsum(x.*(1:n)');
y = w + (1:n)'.*[v(n-1:-1:1);0];
```

(1c) Linear dependence: $O(n)$.

(1d)

$$\mathbf{B} := \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 1 \end{pmatrix}$$

Notice the value 1 in the entry (n, n).

(1e) It is easy to verify with Matlab (or to prove) that $\mathbf{B} = \mathbf{A}^{-1}$. The last line of `multAB.m` prints the value of $\|\mathbf{A}\,\mathbf{B}\,\mathbf{x} - \mathbf{x}\| = \|\mathbf{x} - \mathbf{x}\| = 0$. The returned values are not exactly zero due to roundoff errors.

Problem 2. Modified Newton method [25=3+3+9+7+3 pts]

(2a) If $\mathbf{F}(\mathbf{x}^{(k)}) = \mathbf{0}$ then $\mathbf{y}^{(k)} = \mathbf{x}^{(k)} + \mathbf{0} = \mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)} = \mathbf{y}^{(k)} - \mathbf{0} = \mathbf{x}^{(k)}$.
So, by induction, if $\mathbf{F}(\mathbf{x}^{(0)}) = \mathbf{0}$ then $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} = \mathbf{x}^{(0)}$ for every k.

(2b) Code:

```
function x1 = ModNewtStep (x0, F, DF)
y0 = x0 + F(x0) / DF(x0);     % version for scalar eqs only
x1 = y0 - F(y0) / DF(x0);

% y0 = x0 + DF(x0) \ F(x0);  % version for scalar and vector eqs
% x1 = y0 - DF(x0) \ F(y0);
```

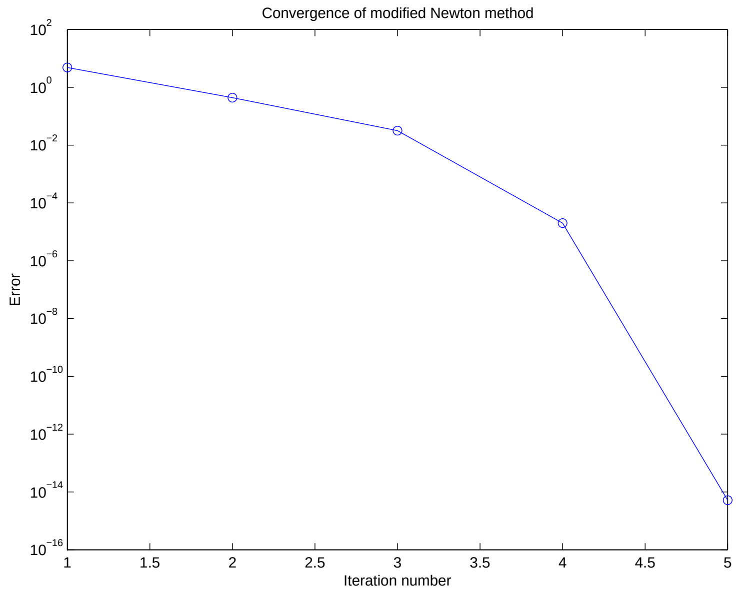
(2c) The order of convergence is approximately 3.

```
function ModNewtOrder
a = 0.123 ;
F = @(x) atan(x) - a ;
DF = @(x) 1./(1 + x.^2) ;
x_exact = tan(a);
% x_exact = fzero(F,x0);
x0 = 5;

x = x0;
err = abs(x0 - x_exact);
it = 0;
while (err(end) > eps   &&   it < 100)
    x_new = ModNewtStep (x(end), F, DF);
    x = [x, x_new];
    err = [ err, abs(x_new - x_exact) ];
    it = it+1;
end
% logarithm of the error and ratios to estimate order of conv.:
log_err = log(err);
emp_orders = (log_err(3:end) - log_err(2:end-1)) ./ ...
    (log_err(2:end-1) - log_err(1:end-2));

close all; figure;
semilogy(1:length(err), err,'o-');
title('Convergence of modified Newton method');
xlabel('Iteration number'); ylabel('Error');
print -depsc2  'ModNewtOrder.eps';
print -djpeg95  'ModNewtOrder.jpg';

disp('Errors and empirical orders of conv.:')
[err', [emp_orders';0;0]]
```



(2d) Code:

```
function x = ModNewtSys( A, c, tol )
c = c(:);          % make sure it is column
F = @(x) A*x + c .* exp(x);
DF = @(x) A + diag( c .* exp(x) );
x0 = zeros( size(c) );

x = x0;
it = 0;
RelIncr = 1;
res = norm(F(x));
while ( RelIncr > tol  &&  it < 100 )
    y = x(:,end) + DF(x(:,end)) \ F(x(:,end));
    x_new = y - DF(x(:,end)) \ F(y);
    RelIncr = norm( x(:,end) - x_new ) / norm(x_new);
    x = [ x, x_new ];
    it = it + 1;
    res = [res, norm(F(x(:,end))) ];
end
disp('Sequence of residuals:')
res

% return only final values and forget history
x = x(:, end);

%        test:
%close all; plot(ModNewtSys(gallery('poisson',20),(1:400),1e-10));
```

Even better with only one LU decomposition for the solution of the 2 linear systems.

(2e) The two systems share the same matrix $D\mathbf{F}(\mathbf{x}^{(k)})$. The LU decomposition of the matrix $D\mathbf{F}(\mathbf{x}^{(k)})$ can be computed once and used twice in order to solve both linear systems. The computation of the LU decomposition is more expensive ($O(n^3)$) than the solution of the two triangular systems ($O(n^2)$).

This method is taken from:
S. Amat, C. Bermudez, S. Busquier, S. Plaza, *On a third-order Newton-type method free of bilinear operators*, Numerical Lin. Alg. Appl. 17, (2010), no. 4, 639–653. DOI: 10.1002/nla.654.

Problem 3. Quadrature plots [16=8+8 pts]

(3a) Plot #1 — Quadrature rule C, Composite 2-point Gauss:
algebraic convergence for every function, about 4^{th} order for two functions.

Plot #2 — Quadrature rule B, Composite trapezoidal:
algebraic convergence for every function, about 2^{nd} order.

Plot #3 — Quadrature rule A, Global Gauss:
algebraic convergence for one function, exponential for another one, exact integration with 8 evaluations for the third one.

(3b) Curve 1 red line and small circles — f_C polynomial of degree 12:
integrated exactly with 8 evaluations with global Gauss quadrature.

Curve 2 blue continuous line only — f_A smooth function:
exponential convergence with global Gauss quadrature.

Curve 3 black dashed line — f_B non smooth function:
algebraic convergence with global Gauss quadrature.

Problem 4. System of ODEs [13=9+4 pts]

(4a) The second order IVP can be rewritten as a first order one by introducing $\mathbf{v}$:

$$\begin{aligned}
\dot{u}_i &= v_i \qquad i = 1, \ldots, n \ , \\
2\dot{v}_1 - \dot{v}_2 &= u_1(u_2 + u_1) \ , \\
-\dot{v}_{i-1} + 2\dot{v}_i - \dot{v}_{i+1} &= u_i(u_{i-1} + u_{i+1}) \qquad i = 2, \ldots, n-1 \ , \\
2\dot{v}_n - \dot{v}_{n-1} &= u_n(u_n + u_{n-1}) \ , \\
u_i(0) &= u_{0,i} \qquad i = 1, \ldots, n \ , \\
v_i(0) &= v_{0,i} \qquad i = 1, \ldots, n \ .
\end{aligned}$$

The ODE system can be written in vector form

$$\dot{\mathbf{u}} = \mathbf{v} \ , \qquad \mathbf{A}\dot{\mathbf{v}} = \mathbf{g}(\mathbf{u}) := \begin{pmatrix} u_1(u_2 + u_1) \\ u_i(u_{i-1} + u_{i+1}) \\ u_n(u_n + u_{n-1}) \end{pmatrix} \ ,$$

where $\mathbf{A} \in \mathbb{R}^{n,n}$ is

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \ .$$

In order to use `ode45`, collect $\mathbf{u}$ and $\mathbf{v}$ in a $(2n)$-dimensional vector $\mathbf{y} = (\mathbf{u}; \mathbf{v})$ and the system reads

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) := \begin{pmatrix} y_{n+1} \\ \vdots \\ y_{2n} \\ \mathbf{A}^{-1}\mathbf{g}(y_1, \ldots, y_l) \end{pmatrix} \ .$$

```
function [Tout, Uout] = MyOde ( T, u0, v0 )

n = length(u0);
y0 = [u0(:); v0(:)];

% create ODE function
A = spdiags( [-ones(n,1),2*ones(n,1),-ones(n,1)], [-1:1], n, n );
```

```
f = @(t,y) [ y(n+1:2*n); A\(y(1:n).*([y(1);y(1:n-1)] + [y(2:n);y(n)])) ];

[ Tout, Yout ] = ode45( f, [0,T], y0 );
Uout = Yout( : , 1:n );
```

(4b) Function, see Figure 2 in the question sheet.

```
function PlotOde
n = 5;
T = 1;
u0 = (1:n)'/n;
v0 = -ones(n,1);

[Tout, Uout] = MyOde ( T, u0, v0 );

close all; figure;
plot(Tout, Uout, 'linewidth',2);
title('Trajectories u_1,...,u_n of second order ODE system');
legend('u1','u2','u3','u4','u5','location','eo');

print -depsc2  'PlotOde.eps';
print -djpeg95 'PlotOde.jpg';
```

Problem 5. Least squares fitting of a quadratic functional [20=4+4+4+8 pts]

(5a) If we denote the cartesian entries of the vectors $\mathbf{z}_i$ as $\mathbf{z}_i = (z_{1,i}, z_{2,i})^T$, we can choose:

$$\mathbf{A} = \begin{pmatrix} z_{1,1}^2 & 2z_{1,1}z_{2,1} & z_{2,1}^2 \\ \vdots & \vdots & \vdots \\ z_{1,i}^2 & 2z_{1,i}z_{2,i} & z_{2,i}^2 \\ \vdots & \vdots & \vdots \\ z_{1,N}^2 & 2z_{1,N}z_{2,N} & z_{2,N}^2 \end{pmatrix} \in \mathbb{R}^{N,3}, \qquad \mathbf{b} = \mathbf{y} \in \mathbb{R}^N, \qquad \mathbf{x} = \begin{pmatrix} M_{1,1} \\ M_{1,2} \\ M_{2,2} \end{pmatrix} \in \mathbb{R}^3 .$$

Indeed,

$$\begin{aligned} \sum_{i=1}^{N} \big(\Phi_{\mathbf{P}}(\mathbf{z}_i) - y_i\big)^2 &= \sum_{i=1}^{N} \big(\mathbf{z}_i^T \mathbf{M} \mathbf{z}_i - y_i\big)^2 \\ &= \sum_{i=1}^{N} \Bigg(\sum_{j,k=1}^{2} z_{j,i} M_{j,k} z_{k,i} - y_i \Bigg)^2 \\ &= \Big\| \mathbf{A}\mathbf{x} - \mathbf{b} \Big\|_2^2 . \end{aligned}$$

(5b) Function:

```
function M = QuadFit( Z, y )

A = [ Z(1,:).^2; 2*Z(1,:).*Z(2,:); Z(2,:).^2]'  ;
x = A \ y(:);
M = [x(1), x(2); x(2), x(3)];

%          TEST:
% clear all;  n=200; M=rand(2); M=M+M'; Z=randn(2,n);
% norm(QuadFit(Z, diag(Z'*M*Z).*(1+1e-5*randn(n,1)))-M)/norm(M)
```

(5c) We represent a general upper triangular matrix as

$$\mathbf{R} = \begin{pmatrix} R_{1,1} & R_{1,2} \\ 0 & R_{2,2} \end{pmatrix}, \quad \text{thus} \quad \mathbf{R}^T\mathbf{R} = \begin{pmatrix} R_{1,1}^2 & R_{1,1}R_{1,2} \\ R_{1,1}R_{1,2} & R_{1,2}^2 + R_{2,2}^2 \end{pmatrix} .$$

Then, the i-th component of the function $\mathbf{F}$ can be written as

$$\big(\mathbf{F}(\mathbf{R})\big)_i = \mathbf{z}_i^T\mathbf{R}^T\mathbf{R}\mathbf{z}_i - y_i = z_{1,i}^2 R_{1,1}^2 + 2z_{1,i}z_{2,i}R_{1,1}R_{1,2} + z_{2,i}^2(R_{1,2}^2 + R_{2,2}^2) - y_i\ .$$

(5d) If $\mathbf{R}$ is represented with the vector $\mathbf{r} = (R_{1,1}, R_{1,2}, R_{2,2})^T$, the gradient of $\mathbf{F}_i$ is

$$\mathbf{grad}F_i = \Big(2z_{1,i}^2 R_{1,1} + 2z_{1,i}z_{2,i}R_{1,2}\ , \quad 2z_{1,i}z_{2,i}R_{1,1} + 2z_{2,i}^2 R_{1,2}\ , \quad 2z_{2,i}^2 R_{2,2}^2\Big)\ .$$

```
function   R = NlQuadFit( Z, y, R0 )

r = [R0(1,1); R0(1,2); R0(2,2)];
F = @(r) (Z(1,:).^2*r(1)^2+2*Z(1,:).*Z(2,:)*r(1)*r(2)...
          +Z(2,:).^2*(r(2)^2+r(3)^2))' - y;
DF = @(r) [ 2*Z(1,:).^2*r(1) + 2*Z(1,:).*Z(2,:)*r(2);
            2*Z(1,:).*Z(2,:)*r(1) + 2*Z(2,:).^2*r(2);
            2*Z(2,:).^2*r(3)  ]';

it = 0;
N_incr = 1;
while ( N_incr > 1e-10  && it < 20)
    incr =  DF(r)\F(r);
    N_incr = norm(incr);
    r = r - incr;
    it = it+1;
    disp('Iteration and norm of GN increment:')
    [it, N_incr]
end
R = [r(1), r(2); 0, r(3)];

%   TEST:
% clear all; n=200; Z=randn(2,n); M = [3,1;1,4];
% R=NlQuadFit(Z,diag(Z'*M*Z).*(1+1e-5*randn(n,1)),[1 0;0 1]); norm(R'*R-M)
```