Numerical Methods for CSE

Examination - Solutions

February 10th, 2011

Total points: 65 = 8+14+16+16+11 (These are not "master solutions" but just a reference!)

Problem 1. Advantages of conjugate gradient method [8 pts]

- In case the system matrix is large and sparse;
- in case a matrix-vector product routine is available;
- in case a good initial guess is available;
- in case only an approximated solution requested;
- ...

Problem 2. Cholesky and QR [14 pts]

(2a) $\mathbf{A}^T\mathbf{A}$ has to be s.p.d.:
$(\mathbf{A}^T\mathbf{A})^T = \mathbf{A}^T(\mathbf{A}^T)^T = \mathbf{A}^T\mathbf{A} \Rightarrow$ symmetric
$\mathrm{rank}(\mathbf{A}) = n \Rightarrow \mathbf{A}$ injective $\Rightarrow \forall \mathbf{v} \in \mathbb{R}^n$ s.t. $\mathbf{v} \neq 0$ it holds $\mathbf{A}\mathbf{v} \neq 0 \Rightarrow \mathbf{v}^T\mathbf{A}^T\mathbf{A}\mathbf{v} = \|\mathbf{A}\mathbf{v}\|_2^2 > 0 \Rightarrow$ positive definite.

(2b) 3 things have to be verified:

- $\mathbf{R} \in \mathbb{R}^{n,n}$ upper triangular, from definition of Cholesky decomposition;
- $\mathbf{Q} = (\mathbf{R}^{-T}\mathbf{A}^T)^T = \mathbf{A}\mathbf{R}^{-1} \in \mathbb{R}^{m,n}$; $\mathbf{R}^T\mathbf{R} = \mathbf{A}^T\mathbf{A} \Rightarrow \mathbf{R}^{-T}\mathbf{A}^T\mathbf{A}\mathbf{R}^{-1} = \mathbf{Id} \Rightarrow \mathbf{Q}^T\mathbf{Q} = \mathbf{R}^{-T}\mathbf{A}^T\mathbf{A}\mathbf{R}^{-1} = \mathbf{Id}$, $\mathbf{Q}$ has orthogonal columns;
- $\mathbf{Q}\mathbf{R} = (\mathbf{R}^{-T}\mathbf{A}^T)^T\mathbf{R} = \mathbf{A}\mathbf{R}^{-1}\mathbf{R} = \mathbf{A}$.

(2c) $\mathbf{A}$ has rank 2 but

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 1+\frac{1}{4}\epsilon & 1 \\ 1 & 1+\frac{1}{4}\epsilon \end{pmatrix} \approx \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

has rank one in machine arithmetics. So, the matrix is symmetric but non positive definite, the hypothesis needed for the Cholesky decomposition are not satisfied.

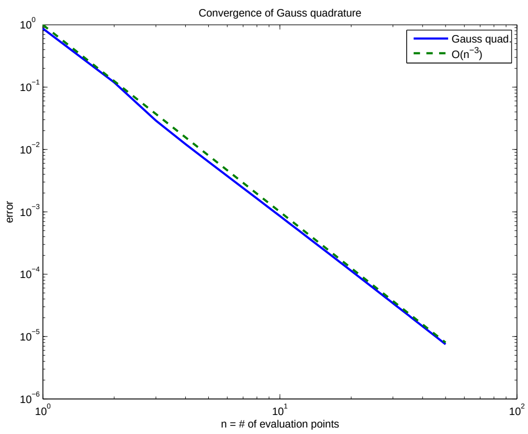
Problem 3. Quadrature [16 pts]

(3a) Routine :

```
function GaussConv(f_hd)
if nargin<1;    f_hd = @(t) sinh(t);    end;
I_exact = quad(@(t) asin(t).*f_hd(t),-1,1,eps)

n_max = 50;  nn = 1:n_max;
err = zeros(size(nn));
for j = 1:n_max
    [x,w] = gaussquad(nn(j));
    I = dot(w, asin(x).*f_hd(x));
    % I = GaussArcSin(f_hd,nn(j));    % using pcode
    err(j) = abs(I - I_exact);
end

close all; figure;
loglog(nn,err,[1,n_max],[1,n_max].^(-3),'--','linewidth',2);
title('Convergence of Gauss quadrature');
xlabel('n = # of evaluation points'); ylabel('error');
legend('Gauss quad.','O(n^{-3})');
print -depsc2 'GaussConv.eps';
```



(3b) Algebraic convergence. (Not requested: approxim. $O(n^{-3})$, expected $O(n^{-2.7})$)

(3c) With the change of variable $t = \sin(x)$, $\mathrm{d}t = \cos x \mathrm{d}x$

$$I = \int_{-1}^{1} \arcsin(t)\ f(t)\,\mathrm{d}t = \int_{-\pi/2}^{\pi/2} x\ f(\sin(x))\cos(x)\,\mathrm{d}x.$$

(the change of variable has to provide a smooth integrand on the integration interval)

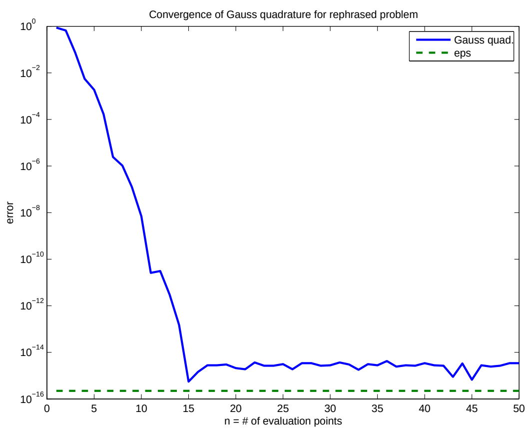
(3d) Routine:

```
function GaussConvCV(f_hd)
if nargin<1;    f_hd = @(t) sinh(t);    end;
g = @(t) t.*f_hd(sin(t)).*cos(t);
I_exact = quad(@(t) asin(t).*f_hd(t),-1,1,eps)
```

```
%I_exact = quad(@(t) g(t),-pi/2,pi/2,eps)

n_max = 50;   nn = 1:n_max;
err = zeros(size(nn));
for j = 1:n_max
    [x,w] = gaussquad(nn(j));
    I = pi/2*dot(w, g(x*pi/2));
   % I = GaussArcSinCV(f_hd,nn(j));    % using pcode
    err(j) = abs(I - I_exact);
end

close all; figure;
semilogy(nn,err,[1,n_max],[eps,eps],'--','linewidth',2);
title('Convergence of Gauss quadrature for rephrased problem');
xlabel('n = # of evaluation points'); ylabel('error');
legend('Gauss quad.','eps');
print -depsc2 'GaussConvCV.eps';
```



(3e) The convergence is now exponential. The integrand of the original integrand belongs to $C^0([-1,1])$ but not to $C^1([-1,1])$ because the derivative of the $\arcsin$ function blows up in ± 1. The change of variable provides a smooth integrand: $x\cos(x)\sinh(\sin x) \in C^\infty(\mathbb{R})$. Gauss quadrature ensures exponential convergence only if the integrand is smooth (C^∞). This explains the algebraic and the exponential convergence.

Problem 4. Exponential integrator [16 pts]

(4a) Given $\mathbf{y}(0)$, a step of size t of the method gives

$$\begin{aligned}
\mathbf{y}_1 &= \mathbf{y}(0) + t\,\varphi\big(t\,\mathbf{Df}(\mathbf{y}(0))\big)\,\mathbf{f}(\mathbf{y}(0)) && = \mathbf{y}(0) + t\,\varphi\big(t\,\mathbf{A}\big)\,\mathbf{A}\,\mathbf{y}(0)\\
&= \mathbf{y}(0) + t\,\big(\exp(\mathbf{A}t) - \mathbf{Id}\big)\,(t\mathbf{A})^{-1}\,\mathbf{A}\,\mathbf{y}(0) && = \mathbf{y}(0) + \exp(\mathbf{A}t)\,\mathbf{y}(0) - \mathbf{y}(0)\\
&= \exp(\mathbf{A}t)\,\mathbf{y}(0) && = \mathbf{y}(t).
\end{aligned}$$

(4b) Function:

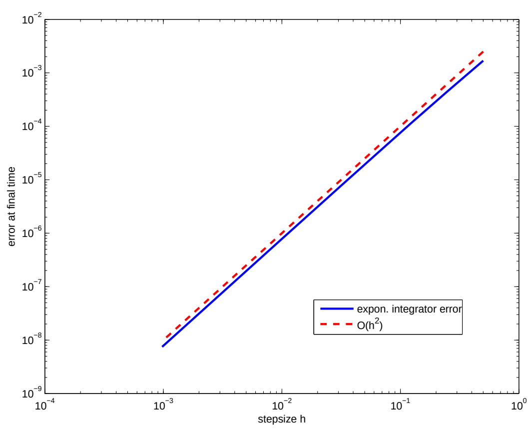
```
function y1 = ExpEulStep(y0,f,df,h)
y1 = y0(:) + h * phim( h * df(y0(:))) * f(y0(:) );
```

(4c) Error $= O(h^2)$.

```
function ExpIntOrder
% estimate order of exponential integrator on scalar logistic eq.
t0 = 0;
T = 1;
y0 = 0.1;

% logistic f, derivative and exact solution:
f = @(y) y.*(1-y);
df = @(y) 1 - 2*y;
exactY = @(t,y0) y0./(y0+(1-y0).*exp(-t));
exactYT = exactY(T-t0,y0);

kk = 2.^(1:10);          % different numbers of steps
err = zeros(size(kk));
for j=1:length(kk)
    k = kk(j);
    h = (T-t0)/k;
    y = y0;
    for st = 1:k         % timestepping
        y = y + h * phim(h*df(y)) * f(y);
    end
    err(j) = abs(y - exactYT);
end
hh = (T-t0)./kk;
close all; figure;
loglog(hh, err, hh, hh.^(2)/100, '--r', 'linewidth',2);
legend('expon. integrator error','O(h^2)','location','best');
xlabel('stepsize h'); ylabel('error at final time');
print -depsc2 'ExpIntOrder.eps';
```



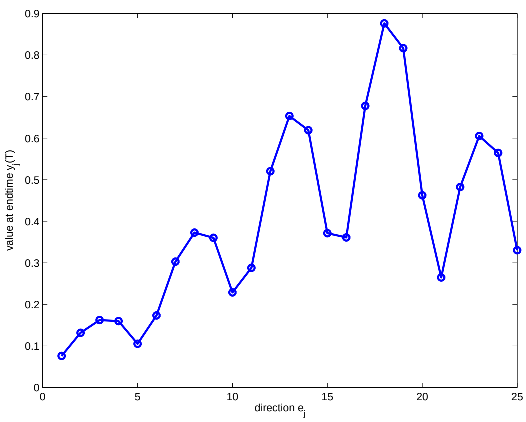
(4d) Function:

```
function yOut = ExpIntSys(n,N,T)
% exponential integrator for system y' = -Ay + y.^3
if nargin<1;
    n = 5;
```

```
    N = 100;
    T = 1;
end
d = n^2;            % system dimension
t0 = 0;
y0 = (1:d)/d;       % rhs

% build ODE
A = gallery('poisson',n);
%% Octave version:
% B = spdiags([-ones(n,1),2*ones(n,1),-ones(n,1)],[-1,0,1],n,n);
% A = kron(B,speye(n))+kron(speye(n),B);
f  = @(y) -A*y(:) + y(:).^3;
df = @(y) -A + diag(3*y.^2);

% exponential integrator
h = (T-t0)/N;
%tOut = linspace(t0,T,N+1)';
yOut = zeros(N+1, d);
yOut(1,:) = y0(:).';
for st = 1:N     % timestepping
      yOut(st+1, :) = ExpEulStep( yOut(st,:), f, df, h );
end
close all;
plot((1:d),yOut(end,:),'-o','linewidth',2);
xlabel('direction e_j'); ylabel('value at endtime y_j(T)');
print -depsc2 'ExpIntSys.eps';
```



(4e) Routine:

```
function ExpIntErr
n = 5;
d = n^2;            % dimension
t0 = 0;
T = 1;
y0 = (1:d)/d;       % RHS
N = 100;            % number of expon. integrator steps

```

```
% build ODE
A = gallery('poisson',n);
%% Octave version:
% B = spdiags([-ones(n,1),2*ones(n,1),-ones(n,1)],[-1,0,1],n,n);
% A = kron(B,speye(n))+kron(speye(n),B);
f  = @(y) -A*y(:) + y(:).^3;
% solve with ode45 and Exponential Euler Integrator:
[t45,y45] = ode45(@(t,y) f(y),[t0,T],y0);
yOut = ExpIntSys(n,N,T);
% relative error in 2-norm at time T
RELERR = norm(y45(end,:)-yOut(end,:)) / norm(y45(end,:))

%% extra: plot, lines = exponential Euler, circles = ode45
% figure; plot((T-t0)*(0:N)/N+t0,yOut); hold on; plot(t45,y45,'o');
```

Problem 5. Matrix least squares in Frobenius norm [11 pts]

(5a) We call $\mathbf{m}^* \in \mathbb{R}^{n^2}$ the vector obtained by the concatenation of the rows of $\mathbf{M}^*$. Then the functional to be minimized is just the 2-norm of $\mathbf{m}^*$, i.e., $\mathbf{A} = \mathbf{Id}_{n^2}$ and $\mathbf{b} = \mathbf{0}$. The constraint $\mathbf{M}^*\mathbf{z} = \mathbf{g}$ can be written as $\mathbf{C}\mathbf{m}^* = \mathbf{d}$ choosing $\mathbf{d} = \mathbf{g}$ and the $n \times n^2$ matrix $\mathbf{C}$ as the Kronecker product of the $n \times n$ identity matrix and $\mathbf{z}^T$. In symbols

$$\mathbf{C} = \begin{pmatrix} (z_1, \dots, z_n) & 0 & \cdots & 0 \\ 0 & \mathbf{z}^T & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \underbrace{0}_{n} & \cdots & 0 & \mathbf{z}^T \end{pmatrix}.$$

Then it is possible to obtain $\mathbf{m}^*$ from the solution of the extended normal equations linear system:

$$\begin{pmatrix} \mathbf{Id}_{n^2} & \mathbf{C}^T \\ \underbrace{\mathbf{C}}_{n^2} & \underbrace{\mathbf{0}}_{n} \end{pmatrix} \begin{pmatrix} \mathbf{m}^* \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{g} \end{pmatrix}.$$

(5b) Function:

```
function M = MinFrob(z,g)
n = size(z,1);

C =  kron( eye(n), z.' );
x = [eye(n^2), C'; C, zeros(n,n)] \ [zeros(n^2,1); g];
m = x(1:n^2);
M = reshape(m,n,n)';

% test, n=10 --> norm(M,'fro') = 0.1611...     :
%z = (1:n)'; g = ones(n,1); M = MinFrob(z,g);
%[norm(M*z-g),norm(M,'fro'),norm(g)/norm(z),norm(M-g*z'/norm(z)^2)]
```

(from Matlab : $\mathbf{M} = \mathbf{g}\mathbf{z}^T / \|\mathbf{x}\|_2^2$, $\quad \|\mathbf{M}\|_F = \|\mathbf{g}\| / \|\mathbf{z}\|$)