R. Hiptmair
J. Šukys

Fall term 2011

Numerical Methods for CSE

ETH Zürich
D-MATH

Examination – Solutions

August 14th, 2012

Total points: 150 = 7 + 8 + 25 + 30 + 30 + 20 + 30.

Problem 1. Advantages of higher order quadrature rules

Efficiency: higher order rules provide smaller increase in the number of evaluation of the integrand function f (effort) for a prescribed reduction of the error, provided that f is smooth enough.

Problem 2. Cubic spline interpolation

Things to check for complete spline interpolation:

- $\mathcal{P}_3 \subset \mathcal{S}_{3,\mathcal{M}}$
- Uniqueness of complete cubic spline interpolant
- Given polynomial p satisfies all interpolation conditions

each item.

Natural spline interpolation: $s''(a) = s''(b) = 0$ is *not* necessarily satisfied by a given polynomial p.

Problem 3. Newton's method

(3a) Using chain rule, we obtain

$$D\left(\|\mathbf{x}\|_2^2\,\mathbf{x}\right) = D\left(\|\mathbf{x}\|_2^2\right)\mathbf{x}^\top + \|\mathbf{x}\|_2^2\, D(\mathbf{x}) = 2\mathbf{x}\mathbf{x}^\top + \|\mathbf{x}\|_2^2\,\mathbf{I}.$$

Hence

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{F(\mathbf{x}^{(k)})}{DF(\mathbf{x}^{(k)})}$$

with

$$F(\mathbf{x}^{(k)}) = \left(\mathbf{A} + \|\mathbf{x}\|_2^2\,\mathbf{I}\right)\mathbf{x} - \mathbf{b}, \qquad DF(\mathbf{x}^{(k)}) = \mathbf{A} + 2\mathbf{x}\mathbf{x}^\top + \|\mathbf{x}\|_2^2\,\mathbf{I}.$$

$F(\mathbf{x}^{(k)})$, $DF(\mathbf{x}^{(k)})$, full correct formulation.

(3b) Function:

```
function [x, res] = newton(n)

rtol = 1e-6;
atol = 1e-8;

A = gallery('poisson',n);
b = ones(n*n,1);

x0 = A\b;

I = eye(n*n);

F = @(x) (A + norm(x)^2*I) * x - b;
```

```
14
15 DF = @(x) A + 2*x*x' + norm(x)^2*I;
16
17 [x, res] = dampnewton(x0, F, DF, rtol, atol);
```

setup, correct use of `dampnewton` results and arguments,
`res(end) = 3.246799898324143e-10`,
`norm(x) = 2.103052392050431`.

Problem 4. Zero finding

(4a) Using the notation $\mathbf{y} = (\mathbf{x}^\top, \lambda)^\top$, the nonlinear systems is equivalent to

$$F(\mathbf{y}) = \begin{pmatrix} (\mathbf{A} + \lambda\mathbf{I})\mathbf{x} - \mathbf{b} \\ \|\mathbf{x}\|_2^2 - \lambda \end{pmatrix} = 0.$$

The Newton iteration is hence

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} - \frac{F(\mathbf{y}^{(k)})}{DF(\mathbf{y}^{(k)})}$$

with

$$DF(\mathbf{y}^{(k)}) = \begin{pmatrix} \mathbf{A} + \lambda\mathbf{I} & \mathbf{x} \\ 2\mathbf{x}^\top & -1 \end{pmatrix}.$$

$F(\mathbf{y}^{(k)})$, each correct entry in $DF(\mathbf{y}^{(k)})$.

(4b) Function Ψ is obtained by solving the first equation of (2) for $\mathbf{x}$ and substituting into the second equation:

$$\mathbf{x} = (\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{b} \quad \Longrightarrow \quad \Psi(\lambda) = \lambda - \left\|(\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{b}\right\|^2. \tag{1}$$

Function:

```
1 function y = psi(A,b,x)
2
3 I = eye(size(A));
4 y = x - norm((A+x*I)\b)^2;
```

(4c) Function:

```
1  function plot_psi(A,b)
2
3  ezplot(@(x)psi(A,b,x), [-10 10]);
4  xlabel('{\bf lambda}');
5  ylabel('{\bf Psi}');
6  title('');
7  print -depsc 'plot_psi.eps'
8
9  ezplot(@(x)psi(A,b,x), [3 8]);
10 xlabel('{\bf lambda}');
11 ylabel('{\bf Psi}');
12 title('');
13 print -depsc 'plot_psi_zoom.eps'
```

(4d) From the plots in subproblem (5c) we see that $\Psi(\lambda)$ is continuous in interval $[2, 10]$ and that it has a zero, for instance, in interval $[3, 8]$. Hence, we choose $\lambda_a = 3$ and $\lambda_b = 8$.

Function:

```
1 function lambda = psi_zero(A,b)
2
3 lambda = fzero(@(x)psi(A,b,x),[3 8]);
```

setup, determining interval boundaries, correct use of `fzero`, `lambda = 4.4228`.

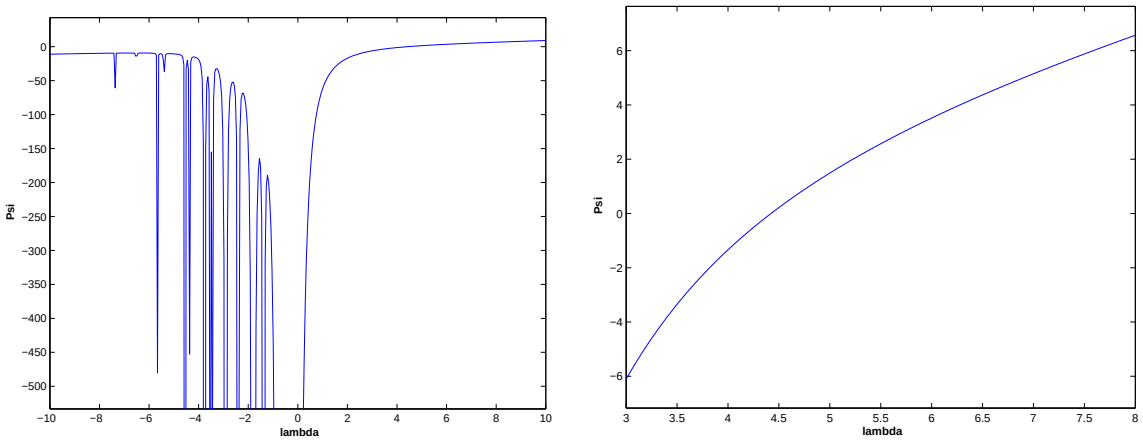

Figure 1: Plot for (5b).

Problem 5. The leapfrog single step method for 2nd-order ODEs

(5a) Function:

```
function yfinal = lf (f, y0, T, N)

h = T/N;
y = y0;
v = 0 + h/2*f(y);

for i=1:N
    y = y + h*v;
    v = v + h*f(y);
end

yfinal = y;
```

setup, each line in `for`-loop.

(5b) Function:

```
function yfinal = lfode45(f,y0,T)

n = length(y0);
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-10);
[~, yfinal] = ode45(@(t,y)[y(n+1:2*n);f(y(1:n))], [0, T], [y0; zeros(n,1)],options);
yfinal = yfinal(end,1:n);
```

absolute and relative tolerances, each correct argument in `ode45`, correct output.

if non-vector version was implemented.

(5c) Function:

```
function order = lforder (f,y0,T)

Ns = 2.^(4:9);

y_exact = lfode45(f,y0,T);

err = [];
for N=Ns
    err = [err, norm(y_exact - lf(f,y0,T,N))]; %#ok<AGROW>
end
```

```
11
12 loglog(Ns, err);
13 p = polyfit (log(Ns), log(err), 1);
14
15 order = - p(1);
```

Should return: `order = 1.99`.
setup, exact solution, `for`-loop, usage of `lf`, error computation, order computation.

Problem 6. Matrix equation

(6a) Let $\mathbf{A} = \mathbf{R}_\mathbf{A}^\top \mathbf{R}_\mathbf{A}$ and $\mathbf{S} = \mathbf{R}_\mathbf{S}^\top \mathbf{R}_\mathbf{S}$ be Cholesky decompositions of $\mathbf{A}$ and $\mathbf{S}$. Then $\mathbf{X}^\top \mathbf{R}_\mathbf{A}^\top \mathbf{R}_\mathbf{A} \mathbf{X} = \mathbf{R}_\mathbf{S}^\top \mathbf{R}_\mathbf{S}$. By the uniqueness of the Cholesky decomposition, we have $\mathbf{R}_\mathbf{A}\mathbf{X} = \mathbf{R}_\mathbf{S}$, hence $\mathbf{X} = \mathbf{R}_\mathbf{A}^{-1}\mathbf{R}_\mathbf{S}$.

(6b) Function:

```
1 function X = mateqsolve(A,S)
2
3 R_A = chol(A);
4 R_S = chol(S);
5
6 X = R_A \ R_S;
```

each `chol`, last line.

to (5a) if the solution for (5b) clearly reveals that the idea to use Cholesky decomposition has been grasped, but this is not included in the answer for sub-problem (5a).

(6c) `mateqsolve` involves Cholesky decomposition, which attains asymptotic computational complexity of $O(n^3/3)$, i.e. half of the standard LU-decomposition.

The final matrix-matrix multiplication is also $O(n^3)$.

correct answer, explanation or numerical experiment.

Problem 7. Special matrices

(7a) Set $\{(i+j) : i,j = 1,\ldots,n\}$ consists of numbers $2,\ldots,2n$, hence $2n-1$ different entries.

(7b)

$$\mathbf{P} = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix} \tag{2}$$

(7c) Matrix-vector multiplication is $O(n^2)$.

(7d) Function:

```
1 function y = multA(n,f,x)
2
3 a = [f((n+1:2*n)/n) 0 f((2:n)/n)]';
4 A = fft(a);
5 X = fft([x; zeros(n,1)]);
6 Y = A .* X;
7 y = ifft(Y);
8 y = flipud(y(1:n));
```

assembly of `a`, `fft(a)`, `ifft(Y)`, each other line in code.

`norm(y) = 154.5270.`