R. Hiptmair
J. Šukys

Fall term 2011

Numerical Methods for CSE

ETH Zürich
D-MATH

Examination

January 31st, 2012

Instructions.

Duration of examination: 180 minutes. **Total points: 180.**

Concise answers are desirable, but any “yes” or “no” answer requires explaining.

Write Matlab codes as simple as possible and add essential comments. Features of a code that have not been asked for will not earn extra points.

Only the Matlab files that are requested in the problem statement will be corrected. The theoretical parts of the problems have to be solved **on paper**.

All the requested `.m` and `.eps` files (with the correct file names) have to be saved in the folders
`~/resources/Matlab/Task*/`

Do not save or modify any file outside these folders.

The course scripts are available in `~/resources/NumCSE11(_ext).pdf`

Look at the number of points awarded for sub-problems to gauge the desired level of detail for the answer.

Good luck!

Problem 1 Structured linear system

[35 points]

Let two vectors $\mathbf{a} = (a_1, \dots, a_n)^T \in \mathbb{R}^n$, $\mathbf{b} = (b_1, \dots, b_n)^T \in \mathbb{R}^n$ be given. Consider the $n \times n$ linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ where the matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \begin{pmatrix} a_1 & 2a_1 & 3a_1 & \cdots & n\,a_1 \\ 0 & a_2 & 2a_2 & \cdots & (n-1)a_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1} & 2a_{n-1} \\ 0 & \cdots & \cdots & 0 & a_n \end{pmatrix}. \tag{1}$$

(1a) [2 points] Give necessary and sufficient conditions on the vector $\mathbf{a}$ such that the matrix $\mathbf{A}$ is non-singular.

(1b) [8 points] Write a Matlab function

```
function  A = AstructMat(a)
```

that creates the matrix $\mathbf{A}$ from the column vector $\mathbf{a}$.

(1c) [3 points] What is the expected asymptotic complexity (with respect to the size n) of the solution of the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ by the Matlab direct solver (“\”)?

(1d) [22 points] Write an efficient Matlab function

```
function  x = AstructLSE(a, b)
```

that first checks whether the linear system with system matrix $\mathbf{A}$ from (1) (defined by the column vector $\mathbf{a}$) and right hand side $\mathbf{b}$ is **uniquely** solvable, and, if it is, solves it with **linear** complexity with respect to the problem size parameter n.

HINT: Using Matlab , study the structure of $\mathbf{A}^{-1}$. In case you did not complete sub-problem (b), a scrambled Matlab implementation of `AstructMat` is provided as the p-file `AstructMatP.p`.

Problem 2 Best rank-k approximation

[15 points]

Given a regular, square, dense matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ and an integer k such that $0 < k < n$, we are interested in a best rank-k approximation of the inverse of $\mathbf{A}$:

$$\mathbf{B} := \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{n,n},\ \mathrm{rank}(\mathbf{M})=k} \left\|\mathbf{A}^{-1} - \mathbf{M}\right\|_2^2 .$$

(2a) [13 points] Write a (short) Matlab function

```
B = kRankInv(A, k)
```

that computes the matrix $\mathbf{B}$.

You are **not** allowed to invert matrix $\mathbf{A}$, e.g. backslash "\", `inv`, `^(-1)`, `QR` and `LU` commands are **not** allowed.

(2b) [2 points] Is $\mathbf{B}$, a best rank-k approximation of the inverse of $\mathbf{A}$, unique for every invertible matrix $\mathbf{A}$? Explain your answer.

Problem 3 Solving an eigenvalue problem with Newton's method

[35 points]

Given a symmetric positive-definite matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, solving

$$\mathbf{F}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix} = 0 \quad \text{for} \quad \mathbf{F}\begin{pmatrix}\mathbf{x}\\ \lambda\end{pmatrix} := \begin{pmatrix}\mathbf{A}\mathbf{x} - \lambda\mathbf{x}\\ 1 - \frac{1}{2}\|\mathbf{x}\|^2\end{pmatrix},$$

amounts to finding an eigenvector $\mathbf{x}$ and associated eigenvalue λ for $\mathbf{A}$.

Thus, a numerical method for computing one eigenvalue/eigenvector of $\mathbf{A}$ is the application of Newton's method to find a zero of the vector-valued function $\mathbf{F} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$.

(3a) [10 points] Compute the Jacobian $\mathbf{DF}\binom{\mathbf{x}}{\lambda}$ of $\mathbf{F}$ at $\binom{\mathbf{x}}{\lambda} \in \mathbb{R}^{n+1}$.

(3b) [5 points] State the Newton iteration for solving $\mathbf{F}(\mathbf{x}) = \mathbf{0}$.

(3c) [20 points] Implement a Matlab function

```
function [eigvec,eigval] = eignewton(A, x, rtol, atol)
```

which computes one eigenvector/eigenvalue of the given matrix $\mathbf{A}$ using the Newton method from the previous sub-problem. Use relative and absolute error tolerances passed in `rtol` and `atol` for termination. The vector argument `x` passes an initial guess for the eigenvector. The corresponding initial guess for the eigenvalue is to be computed by means of the Rayleigh quotient.

HINT: Test your code with some small matrix $\mathbf{A}$ and compare with the output of the Matlab built-in function `eig`.

Problem 4 Matrix ODE

[55 points]

We consider the initial value problem

$$\dot{\mathbf{Y}} = -(\mathbf{Y} - \mathbf{Y}^\top)\mathbf{Y} =: f(\mathbf{Y}) \quad , \quad \mathbf{Y}(0) = \mathbf{Y}_0 \in \mathbb{R}^{n,n} \; , \tag{2}$$

whose solution is a matrix valued function $t \mapsto \mathbf{Y}(t) \in \mathbb{R}^{n,n}$.

(4a) [5 points] Show that all explicit Runge-Kutta methods applied to (2) produce constant solutions, if

$$\mathbf{Y}_0 = \mathbf{Y}_0^\top .$$

(4b) [15 points] Write a MATLAB function

```
function YT = matode(Y0,T)
```

that solves (2) over $[0, T]$ for the initial value passed in `Y0` using Matlab 's standard integration routine `ode45` with absolute tolerance 10^{-10} and relative tolerance 10^{-8}. The return value should be an approximation of $\mathbf{Y}(T) \in \mathbb{R}^{n,n}$.

(4c) [10 points] Show that $t \mapsto \mathbf{Y}^\top(t)\mathbf{Y}(t)$ is constant for the exact solution $\mathbf{Y}(t)$ of (2).

HINT: This is equivalent to showing $\frac{d}{dt}(\mathbf{Y}^\top(t)\mathbf{Y}(t)) = 0$, which can first be rephrased using the product rule.

(4d) [5 points] Write a MATLAB function

```
function checkinvariant(Y0,T)
```

that is meant to verify (numerically) the assertion of sub-problem (c) at time $t = T$ for the output of `matode` from sub-problem (b). The arguments are the same as for `matode`.

HINT: In case you did not complete sub-problem (b), a scrambled Matlab implementation of `matode` is available as the p-file `matodeP.p`.

(4e) [10 points] The so-called discrete gradient rule for (2) is a single step method defined by

$$\mathbf{Y}_* = \mathbf{Y}_k + \tfrac{1}{2}h_k f(\mathbf{Y}_k) \quad , \quad \mathbf{Y}_{k+1} = (\mathbf{I} + \tfrac{1}{2}h_k(\mathbf{Y}_* - \mathbf{Y}_*^\top))^{-1}(\mathbf{I} - \tfrac{1}{2}h_k(\mathbf{Y}_* - \mathbf{Y}_*^\top))\mathbf{Y}_k, \tag{3}$$

where $\mathbf{Y}_k$ denotes the approximated solution at time $t = t_k$ and h_k is the integration step length, i.e. $h_k = t_{k+1} - t_k$.

Write a Matlab script

```
function YT = matodespr (Y0,T,N)
```

which approximately solves the ODE (2) using the discrete gradient rule (3). The input and output parameters are the same as for `matode` in sub-problem (b); the additional parameter `N` is the number of equidistant integration steps.

(4f) [10 points] Write a Matlab function

```
function matodecvg ()
```

that determines the order of convergence of the discrete gradient rule in a numerical experiment.

Use N equidistant integration steps

$$N \in \{10, 20, 40, 80, 160, 320, 640, 1280\}$$

for solving (2) approximately. Create an appropriate error vs. number of integration steps plot.

As a substitute for an exact solution, use the solution produced by `matode` from sub-problem (b).

Set time horizon to $T = 1$. As initial value $\mathbf{Y}_0$ use the *orthogonal* matrix `Y0` generated by

```
[Y0,dummy] = qr(magic(3)).
```

HINT: In case you did not complete sub-problem (e), a scrambled Matlab implementation of `matodespr` is available as the p-file `matodesprP.p`.

Problem 5 Legacy routine

[40 points]

Some legacy Matlab code contains the poorly documented routine `gse` listed below.

Listing 1: MATLAB function `gse`

```
function se = gse(A,B,tol,maxit)
% A should be a symmetric positive matrix of size n*n,
% B should be a handle of type @(x) to a routine that realizes a mapping R^n to R^n.
if (nargin < 4), maxit = 100; end

z = A(:,1); z = z/norm(z);
rho = 0;
for i=1:maxit
  rho_old = rho;
  v = A*z;
  rho = dot(v,z);
  if (abs(rho-rho_old) < tol*abs(rho)), break; end
  r = v - rho*z;
  z = z - B(r);
  z = z/norm(z);
end
se = rho;
```

(5a) **[5 points]** Consider the following use of `gse`:

$$\texttt{gse(A, @(x) pcg(A,x,0,m), 0.01)} \tag{4}$$

with `A` containing a symmetric positive definite matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, and $m \in \mathbb{N}$. What is the asymptotic computational complexity for a single step of the iteration of `gse` in terms of m and n, if $\mathbf{A}$ is known to have at most five non-zero entries per row?

(5b) **[15 points]** Write a Matlab function

```
function gsecvg ()
```

that investigates the convergence of the iterations in `gse` for the function call as in (4). Use

```
A = gallery('poisson',100);
```

Use `tol=1E-12`. For $m \in \{1, 2, 3, 4\}$, function `gsecvg` should plot the error `|se-se_exact|` vs. number of iterations `maxit`. Use the appropriate scaling of axes. What kind of convergence do you observe?

HINT: the Matlab code for `gse` is provided in `gse.m`. The "exact solution" `se_exact` for convergence plots can be obtained by executing `gse` function with large number of iterations and small tolerance; for instance, set

`maxit = 10000`, `tol = 1e-14`, and `B = @(x) A\x`.

HINT: Function `gse` does not need to be modified.

(5c) **[20 points]** Which algorithm is carried out by the following invocation of `gse`

```
gse(A,@(x) (A\x), 0.01)
```

for a symmetric positive definite matrix $\mathbf{A}$ stored in `A`?

What is the meaning of the value returned?

Explain your answers.