Numerical Mathematics I, 2015/2016, Lab session 5

Keywords: rootfinding, nonlinear equations, fixed point iteration, Newton method

Remarks

- Make a new folder called `NM1_LAB_5` for this lab session, save all your functions in this folder.
- Whenever a new `MATLAB` function is introduced, try figuring out yourself what this function does by typing `help <function>` in the command window.
- Make sure that you have done the preparation before starting the lab session. The answers should be worked out either by pen and paper (readable) or with any text processing software (LaTeX, Word, etc.).

1 Preparation

1.1 Fixed point iteration

1. Study (Textbook, Section 2.4 & 2.5).
2. Consider the fixed point iteration given by
$$x^{(k+1)} = \phi(x^{(k)}), \quad \alpha = \phi(\alpha). \tag{1}$$
Show that the following relationship holds for the error $e^{(k)} = \alpha - x^{(k)}$
$$e^{(k+1)} = \phi'(x^{(k)})e^{(k)} + \frac{\phi''(\eta^{(k)})}{2}(e^{(k)})^2, \tag{2}$$
for some $\eta^{(k)} \in I_{\alpha,x^{(k)}}$. (For an arbitrary iteration function ϕ.)
3. Consider solving the scalar root-finding problem $f(\alpha) = 0$ using the iteration function ϕ_a, which is given by
$$\phi_a(x^{(k)}) = x^{(k)} + af(x^{(k)}). \tag{3}$$
Suppose you are given $f'(\alpha)$, how would you choose the *constant* a to obtain the "best" convergence? What if you are given $f'(x^{(0)})$?
4. Explain how the Newton method follows from (2) and (3) if we allow a new value of a to be chosen at each iteration.
5. Consider the Aitken extrapolation formula (Textbook, Equation 2.28). Suppose we are given some fixed point iteration function ϕ, we define $\phi_\Delta(x)$ as
$$\phi_\Delta(x) = \frac{x\phi(\phi(x)) - [\phi(x)]^2}{\phi(\phi(x)) - 2\phi(x) + x}.$$
This is the first extrapolation of ϕ. The convergence order of $\phi_\Delta(x)$ is improved as compared to that of ϕ, as follows from (Textbook, Theorem 2.2). However, $\phi_\Delta(x)$ by itself is again an iteration function. Hence we can recursively define
$$\phi_\Delta^{(i+1)}(x) = \frac{x\phi_\Delta^{(i)}(\phi_\Delta^{(i)}(x)) - [\phi_\Delta^{(i)}(x)]^2}{\phi_\Delta^{(i)}(\phi_\Delta^{(i)}(x)) - 2\phi_\Delta^{(i)}(x) + x},$$

with $\phi_\Delta^{(0)}(x) := \phi(x)$, and hence $\phi_\Delta^{(1)}(x) = \phi_\Delta(x)$. What are, according to the textbook, the orders of convergence of $\phi_\Delta^{(1)}$, $\phi_\Delta^{(2)}$ and $\phi_\Delta^{(3)}$ if ϕ converges linearly to a simple root of f? What if ϕ converges quadratically to a simple root of f?

6. *Optional.* Consider using Newton's method on a scalar rootfinding problem $f(\alpha) = 0$. Suppose we replace $f'(x)$ by an approximation
$$\delta f(x) = f'(x) + C(x)h^p, \tag{4}$$
for some order p approximation to $f'(x)$. Show that (neglecting $\mathcal{O}((\alpha - x)^2)$ and $\mathcal{O}(h^{p+1})$)
$$\phi' = (\alpha - x)\frac{f'f'' + h^p(f'C' + f''2C)}{(f')^2} + \frac{Ch^p}{f'}.$$

1.2 The Newton method for systems

1. Study (Textbook, Section 2.3).
2. The scalar equation $x^2 = b$ has multiple solutions if $b \neq 0$. A similar result holds for the square root of a matrix. A square root of a matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ is defined as a matrix $\mathbf{X} \in \mathbb{C}^{n\times n}$ such that
$$\mathbf{X}^2 = \mathbf{B}.$$
Show that if $\mathbf{B}$ has n distinct nonzero eigenvalues, then $\mathbf{B}$ has 2^n square roots. Hint: If a matrix has distinct eigenvalues, it is always diagonalisable.

2 Lab experiments

2.1 Fixed point iteration

Introduction

In this exercise you will investigate the convergence properties of several fixed point methods. As indicated in the preparation, the idea of Aitken extrapolation leads to a family of iterative methods defined by its "parent" iteration function $\phi = \phi_\Delta^{(0)}$. Such a family of iterative methods is implemented in the given function `aitkenExtrap.m`. Read the comments of this function and figure out how it works. The input `parentIter` should be a function handle to the parent iteration function $\phi = \phi_\Delta^{(0)}$.

Experiment

Write a parent iteration function `staticParent.m` for the static iteration (1), the header of your function should look like

```
% INPUT
% f          function of rootfinding problem
% a0         static parameter: xnew = x + a0*f(x)
% x0         initial guess
% tol        desired tolerance
% maxIt      maximum number of iterations
% OUTPUT
% root       root of f
% flag       if 0: attained desired tolerance
%            if 1: reached maxIt nr of iterations
% convHist   convergence history
function [root, flag, convHist] = staticParent(f, a0, x0, tol,...
    maxIt)
```

Also write a parent iteration function `newtonParent.m` for the Newton method. Instead of the input `a0`, it should have an input `df` which is the function name or function handle to the derivative of f. Here is an example of how you could use `aitkenExtrap.m`

```
func = @(x) x^2 - 2; a0 = -1 / 12; x0 = 6;
phi = @(x) staticParent(func, a0, x, 0, 1);
tol = 1E-8; maxIt = 15; nrExtrap = 2;
[root, flag, convHist] = aitkenExtrap(phi, x0, tol, maxIt, nrExtrap),
```

where `func` is a function handle to the function of which we want to find the root.

Test the function `aitkenExtrap.m` for both parent functions on the following rootfinding problem

$$\text{Find } \alpha \text{ such that } f(\alpha) = 0, \text{ where } f(x) = \log(xe^x).$$

Use as initial guess $x^{(0)} = 3$. Use the following parameters for `aitkenExtrap.m`: `tol = 1E-12`, `maxIt = 25`. Let `a0` be such that $\phi'(x^{(0)}) = 0$ when using the static parent method. Compare the convergence histories for the two standard parent methods (here you can use `maxDepth = 0`) as well as the first three extrapolations $\phi_\Delta^{(1)}$, $\phi_\Delta^{(2)}$ and $\phi_\Delta^{(3)}$ for *both* parent iteration functions (hence in total you will get at least 8 convergence histories). Plot the logarithm of the error estimate against the iteration number. Whenever possible, calculate the convergence order.

2.2 The Newton method for systems

Introduction

Not every matrix is diagonalisable, however we can factorise any square matrix $\mathbf{A}$ in the following form*

$$\mathbf{A} = \mathbf{PBP}^{-1},$$

where $\mathbf{B}$ is a block-diagonal matrix with blocks $\mathbf{B}_i$, the diagonal of $\mathbf{B}$ contains the eigenvalues of $\mathbf{A}$. Each $\mathbf{B}_i$ is of the following form

$$\mathbf{B}_i = \begin{pmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix}.$$

It can be shown that each block $\mathbf{B}_i$ has an eigenspace of dimension one, hence it cannot be diagonalised (unless it is the 1×1 matrix). So even if we have $\mathbf{P}$ and $\mathbf{B}$, we still need some alternative way of computing the square root of $\mathbf{B}_i$ whenever it is of size larger than one.

*The matrix $\mathbf{B}$ is called the Jordan normal form of $\mathbf{A}$.

In the following experiment you will use Newton's method for computing the square root of such a matrix $\mathbf{B}_i$, which we will denote by $\mathbf{B}(\lambda_i)$ (for $\lambda_i \neq 0$). One way of approaching this problem is to compute the root of the function $\mathbf{F} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ defined by

$$\mathbf{F}(\mathbf{X}) = \mathbf{X}^2 - \mathbf{B}(\lambda). \tag{5}$$

However, the way Newton's method is usually formulated is for functions of the form $\mathbf{F} : \mathbb{R}^{m\times 1} \to \mathbb{R}^{m\times 1}$. Hence we rewrite (5) by introducing the following notation (the vector representation of a matrix)

$$\mathrm{vec}(\mathbf{M}) := \begin{pmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{pmatrix} \in \mathbb{R}^{n^2},$$

where $\mathbf{m}_i$ are the columns of $\mathbf{M}$. Hence we also define the function $\mathbf{f} := \mathrm{vec}(\mathbf{F}) : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$ as

$$\mathbf{f}(\mathrm{vec}(\mathbf{X})) := \mathrm{vec}(\mathbf{X}^2) - \mathrm{vec}(\mathbf{B}(\lambda)).$$

In `MATLAB` you can transform $\mathbf{X}$ to $\mathrm{vec}(\mathbf{X})$ by using `X = reshape(X, n^2, 1)`, and reverse this transformation by `X = reshape(X, n, n)`. The Jacobian matrix $\mathbf{J}$ of $\mathbf{f}$ is then defined as

$$J_{ij}(\mathbf{X}) := \frac{\partial f_i}{\partial(\mathrm{vec}(\mathbf{X})_j)}(\mathbf{X}),$$

for $i, j = 1, \dots, n^2$.

Experiment

Write a `MATLAB` implementation of the Newton method for systems (you could use as a starting point the `newtonParent.m` function for the scalar rootfinding problem). The header of your function should look like

```matlab
% INPUT
% f          function of rootfinding problem
% df         function returning the Jacobian
% x0         initial guess
% tol        desired tolerance
% maxIt      maximum number of iterations
% OUTPUT
% root       if succesful, root is an approximation to the
%            root of the nonlinear equation
% flag       flag = 0: tolerance attained, flag = 1: reached maxIt
% iter       the number of iterations
% convHist   convergence history
function [root, flag, iter, convHist] = newton(f, df, x0, tol,...
    maxIt)
```

Write a `MATLAB` function `matFunc.m` that evaluates $\mathrm{vec}(\mathbf{F})$ given the input $\mathrm{vec}(\mathbf{X})$. The header of your function should look like

```matlab
% INPUT
% vecX       vector (n^2 x 1) representing a matrix X of size
%            n x n
% B          matrix B of size n x n
% OUTPUT
% vecF       vector (n^2 x 1) representing the matrix
%            F = X^2 - B
function vecF = matFunc(vecX , B)
```

You are given the function `matJac.m` that evaluates the Jacobian matrix $\mathbf{J}$ either exactly, or by using a finite difference approximation (depending on the input). The input for `newton.m` should be of the following form when using it to compute a square root of $\mathbf{B}$.

```
dfFunc = @(x) matJac(x, 'exact', [], B);
vecRoot = newton(@(x) matFunc(x, B), dfFunc, x0, tol, maxIt);
```

Test the consistency of the finite difference approximation of the Jacobian. Denote by $\mathbf{J}^h$ the finite difference approximation to $\mathbf{J}$, compute the following maximum

$$\max_{i,j}\left(\left|(\mathbf{J}(\mathbf{x})^h - \mathbf{J}(\mathbf{x}))_{ij}\right|\right),$$

for $h = 10^{-l}$, $l = 0, \ldots, 14$. Note that this is very similar to the consistency test you performed in lab session 2. Find the optimal value h_{opt} for which the error is smallest.

Consider the matrix $\mathbf{B}(\lambda)$ given by

$$\mathbf{B}(\lambda) = \begin{pmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{pmatrix},$$

use as initial guess the 3×3 identity matrix and let $\lambda = 2$. Compare the convergence histories for using the exact Jacobian and the approximation using $h = h_{\text{opt}}$. Also try $h = 10^{-1}$. Use a Newton tolerance of 10^{-10} and a maximum number of iterations of 25.

Optional

Computing (an approximation to) the Jacobian is very expensive and hence we would like to avoid this. If we solve the linear system during the Newton iteration using an iterative method (like the one considered in lab session 4) we only need the matrix-vector product $\mathbf{J}(\mathbf{x})\mathbf{v}$. Therefore we can replace this matrix-vector product by a finite difference approximation to the directional derivative $\nabla_{\mathbf{v}}\mathbf{f}(\mathbf{x})$ (like you did in lab session 2). Implement this approach in a function called `newtonIter.m`, this function should call `iterMethod.m` [†] at each Newton iteration. Think about what tolerance you set for the linear solve (perhaps it should decrease during Newton iteration), let the maximum number of iterations of the linear solve be given by the size of the of linear system that is being solved.

Consider the analogous problem for $\mathbf{B}(\lambda) \in \mathbb{R}^{n\times n}$, where $n = 5, 10, 50$. Compare the efficiency of this implementation to the standard implementation. Also try using a second-order central approximation.

3 Discussion

3.1 Fixed point iteration

1. Do the found convergence orders agree with the theory?

[†] If you haven't done so already, read the optional exercise of lab session 4 to be able to use a function instead of a matrix as input for `iterMethod.m`.

2. What are the pros and cons when considering static iteration with the first Aitken extrapolation versus the “standard” Newton method?

3. If two methods, say method A and method B, both attain the same tolerance but A does so in less iterations. Does this always mean that method A is more efficient? Why, or why not?

3.2 The Newton method for systems

1. Consider (4), does the Newton method maintain quadratic convergence if h is chosen too large? Did your experiments confirm this?

2. If an iterative method is used for solving the linear system of equations, is it necessary to have (an approximation to) the *full* Jacobian matrix $\mathbf{J}$ or is it sufficient to have a function that can evaluate (an approximation to) the matrix-vector product $\mathbf{Jv}$ for some vector $\mathbf{v}$? Explain.

3. *Optional.* How did you choose the tolerance for the iterative method at each step of Newton iteration? Explain.

4. *Optional.* Discuss the efficiency of `newtonIter.m` by comparing it to `newton.m`. Explain why the second-order central approximation works so well for this application.