NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 11
Iterative Methods for Non-Linear Systems of Equations

P. Bansal

Submission deadline - 20th December.

Problem 11.1: Newton's method for $F(x) := \arctan x$

The merely local convergence of Newton's method can be problematic. In Example 8.4.54 in the lecture notes, the consequences of this local convergence are discussed. One of them is the possible failure to converge because of the Newton correction overshoot. We look at this issue in this problem.

(a) Find an equation satisfied by the smallest positive initial guess $x^{(0)}$ for which Newton's method **does not converge** when applied to $F(x) = \arctan x$.

Hint: Find out when the Newton method oscillates between two values, consider the function graph to get insight.

(b) Implement a C++ function:

```
double newton_arctan(double x0_);
```

which uses Newton's method to find an approximation of such $x^{(0)}$. Here `x0_` is the initial guess.

Problem 11.2: Modified Newton method

In this problem, we look at a modified version of the Newton method.

Given $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$, such that $\mathcal{J}_\mathbf{F}(\mathbf{x}^{(0)})$ is regular, the non-linear system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ can be solved using the following iterative method:

$$\mathbf{y}^{(k)} = \mathbf{x}^{(k)} + \mathcal{J}_\mathbf{F}^{-1}(\mathbf{x}^{(k)})\ \mathbf{F}(\mathbf{x}^{(k)})\ ,$$
$$\mathbf{x}^{(k+1)} = \mathbf{y}^{(k)} - \mathcal{J}_\mathbf{F}^{-1}(\mathbf{x}^{(k)})\ \mathbf{F}(\mathbf{y}^{(k)})\ ,$$

where $\mathcal{J}_\mathbf{F}(\mathbf{x}) \in \mathbb{R}^{n,n}$ is the Jacobian matrix of $\mathbf{F}$ evaluated at $\mathbf{x}$.

(a) Show that the iteration is consistent with $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, i.e. show that $\mathbf{x}^{(k)} = \mathbf{x}^{(0)}$, $\forall k \in \mathbb{N}$, if and only if $\mathbf{F}(\mathbf{x}^{(0)}) = \mathbf{0}$.

(b) Implement a C++ function

```
template <typename Scalar, class Function, class Jacobian>
Scalar mod_newt_step_scalar(const Scalar& x,
                            const Function& f,
                            const Jacobian& df);
```

that computes a step of the modified Newton method for a *scalar* function $\mathbf{F}$, that is, for the case $n = 1$.

Here, `f` is a function object of type `Function` passing the function $F : \mathbb{R} \mapsto \mathbb{R}$ and `df` a function object of type `Jacobian` passing the derivative $F' : \mathbb{R} \mapsto \mathbb{R}$. Both require an appropriate lambda function.

(c) Implement a C++ function `void mod_newt_ord()` which uses the function `mod_newt_step` and a good termination criteria to solve:

$$\arctan(x) - 0.123 = 0\ ;$$

Use $x_0 = 5$ as initial guess. Determine empirically the order of convergence.

Problem 11.3: Solving a quasi-linear system

In this problem, we implement a simple fixed point iteration and Newton's method to solve a so-called quasi-linear system of equations. **Template:** `quasilin.cpp`

Consider the *nonlinear* (quasi-linear) system:

$$\mathbf{A}(\mathbf{x})\mathbf{x} = \mathbf{b}\;, \tag{11.1}$$

where $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{A} : \mathbb{R}^n \to \mathbb{R}^{n,n}$ is a matrix-valued function:

$$\mathbf{A}(\mathbf{x}) := \begin{bmatrix} \gamma(\mathbf{x}) & 1 & & & & \\ 1 & \gamma(\mathbf{x}) & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & \gamma(\mathbf{x}) & 1 \\ & & & & 1 & \gamma(\mathbf{x}) \end{bmatrix}, \quad \gamma(\mathbf{x}) := 3 + \|\mathbf{x}\|_2.$$

Here $\|\cdot\|_2$ is the Euclidean norm.

A fixed point iteration can be obtained from Eq. (11.1) by the "frozen argument technique": for the step '$k+1$', evaluate the matrix valued function from the previous step 'k' and solve a linear system for the '$(k+1)$-th' iterate.

(a) Derive the fixed point iteration for Eq. (11.1) and implement a C++ function;

```
template <class func, class Vector>
void fixed_point_step(const func& A, const Vector & b,
                      const Vector & x, Vector & x_new);
```

to advance from $\mathbf{x}^{(k)}$ to $\mathbf{x}^{(k+1)}$.

Important Remark: The lambda function for `func A` is provided in the template.

(b) Implement a C++ function:

```
template <class func, class Vector>
void fixed_point_method(const func& A, const Vector& b,
    const double atol, const double rtol, const int max_itr);
```

which computes the solution $\mathbf{x}^*$ using `fixed_point_step`. The input arguments are the absolute tolerance `atol`, relative tolerance `rtol` and maximum iterations `max_itr`. Use $\mathbf{x}^{(0)} = \mathbf{b}$ as initial guess.

(c) Derive the Newton iteration for Eq. (11.1) and implement a C++ function as in subproblem(a):

```
template <class func, class Vector>
void newton_step(const func& A, const Vector & b,
                 const Vector & x, Vector & x_new);
```

Note that the Jacobian will be a rank 1 modification. So, ideally, you can use the Sherman-Morrison-Woodbury formula to compute the inverse of the Jacobian optimally.

(d) Implement a C++ function similar to `fixed_point_method`:

```
template <class func, class Vector>
void newton_method(const func& A, const Vector& b,
    const double atol, const double rtol, const int max_itr);
```