THE FIXED POINT ITERATION ALGORITHM - LINEAR/NONLINEAR ALGEBRAIC EQUATIONS

JOHAN JANSSON NINNI CARLSUND LEVIN

1. TO READ

Time-stepping Newton's equations of motion (17.1)

Contraction Mapping (ch. 76)

Newton's method (ch. 77)

Solving Linear Algebraic Systems (94)

2. GOALS

2.1. **Understanding.**

- Understand that implicit time-stepping (Trapezoid for example) requires solving a (possibly non-linear) algebraic equation system.
- If $g(x)$ is contraction mapping then the fixed-point iteration $x = g(x)$ converges.
- Understand Newton's method as the best fixed-point method.
- Understand that standard iterative methods (like Jacobi, Steepest Descent and Conjugate Gradient) for linear algebraic systems are fixed-point iteration methods.

2.2. **Skills.**

- Be able to formulate an algebraic equation as $f(x) = 0$ and a fixed-point iteration $x = g(x)$ ($x = x - \alpha f(x)$ as template case).
- Be able to derive the condition for convergence of a fixed-point iteration (definition of contraction mapping).
- Be able to derive Newton's method for solving an algebraic equation system.

- Be familiar with some basic iterative (fixed-point) methods (at least Jacobi's method) for solving linear algebraic systems.

3. Software Interfaces

Follow the interface specifications in the template Python file in the "python" subdirectory. The file **fixedpoint.py** includes the functions (some of them to be finished) needed for this module, continue adding your own code to the functions. When you're finished or want to check how well you're doing, submit your "fixedpoint.py" (must be named exactly that) to the Web-CAT system at http://webcat.csc.kth.se.

You will also find a file **main.py** where you can implement the different test cases specified in the questions below.

4. Examination

The examination of this module consists of the questions below. Each question gives 1 point (a question with two sub-questions gives $0.5+0.5 = 1$ point). To pass the module 2/3 points are necessary.

5. Questions

5.1. **Fixed-point iteration/Newton's method.**

- Formulate a fixed-point iteration ($x = g(x)$) for the non-linear equation $x^2 - 4x + 3 = 0$ and derive the conditions for convergence (contraction mapping).

  Compute a solution using your fix-point iteration. You may use the function `fixedpoint()` (or write your own).
  In order to submit to Web-CAT, complete the implementation (or implement your own solution) in `square_equation()`.

- Solve the following system

$$\begin{pmatrix} x^2 - y \\ x - y^2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

by Newton's method (you may use the function `newton()`, or write your own). The system has one root in $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, compute the other root.
Complete the implementation of `square_system()` to be able to submit to Web-CAT.

5.2. **Implicit time-stepping.**

- Make a new version of the `solve()` function from module 3 (still using the `timestep()` function from module 3), and now implement the fixed-point iteration (you may use `fixedpoint()`).
- Extend your new `solve()` function to Newton's method (you may use `newton()` and `fixedpoint_adapter()`). Verify that your solver now can take larger time steps than the fixed-point variant allows.

5.3. **Linear systems.** We wish to solve the linear system of equations, $Ax = b$, by:

Jacobi:

Jacobi's method can be described as:

$$Ax = b \Rightarrow [A = D + M] \Rightarrow x = D^{-1}(-Mx + b) = g(x)$$

where the matrix D is the diagonal of A and $M = A - D$.

Implement Jacobi's method for solving linear systems of equations. Complete the function `jacobi(A, b)` to be able to submit to Web-CAT. You may use `fixedpoint()` if you wish.

Steepest Descent (hard):

The Steepest Descent method can be described as:

$$x = x - \alpha(Ax - b) = g(x) \quad (r = Ax - b)$$

$$\alpha = \frac{(r, r)}{(r, Ar)}$$

Complete the implementation of Steepest Descent in the function `steepest_descent(A, b)`. You can use the FEM linear system from `generate_data()` as a test case if you wish.

Conjugate Gradient (hard):

The Conjugate Gradient method is very similar to Steepest Descent, but has much faster convergence.

Test the implementation of Conjugate Gradient given in the function `conjugate_gradient(A, b)` (very similar to Steepest Descent) and verify that the convergence is much faster (fewer iterations). You can use the FEM linear system from `generate_data()` as a test case if you wish.