Responsible TA:
Jiwoo Oh
jiwoo.oh@chem.ethz.ch

1 Fixed point iterations

Consider the following nonlinear equation

$$f(x) = x\text{exp}(x) - 1 = 0 \tag{1}$$

Like in the lecture, we consider the three fixed point equations

$$x = \phi_1(x) \text{ with } \phi_1(x) = \text{exp}(-x) \tag{2}$$

$$x = \phi_2(x) \text{ with } \phi_2(x) = \frac{x^2\text{exp}(x) + 1}{\text{exp}(x)(1 + x)} \tag{3}$$

$$x = \phi_3(x) \text{ with } \phi_3(x) = x + 1 - x\text{exp}(x) \tag{4}$$

1. Show analytically that the three fix point equations are consistent with (1) by rearranging the equations (2) to (4) to the form of (1).

2. For each of the iterative formulas (2)-(4) try to find a fixed point using an iteration of the form $x^{k+1} = \phi_i(x^k)$ with $i = 1, 2, 3$ and k denoting the k-th iteration :
   - Use a starting guess x_0 between 0 and 1
   - Loop `while abs(x`$_k$ `- x`$_{k-1}$`) > 1e-8` calculate the next `x` value
   - Store all values that you calculate in a vector `xvec`
   - Also terminate the while-loop if `1e5` iterations are exceeded

3. For each formula, say if the fixed point iteration converges or not? Provide the answers using an `if` block in your code.

4. Compare your results with those, which can be obtained by using fsolve for each of the cases (1) to (4). Could you improve your results by using a different maximal error (tolerance level)?

5. Estimate the convergence orders p and the rates of convergence C for the formulas which have a fixed point (keep in mind that the following formulas cannot be applied to the first and last elements of your x^k vectors).
   - Define `xstar` based on the last iteration value or the solution from `fsolve`
   - Calculate the vector `eps = abs(xvec - xstar)`
   - Use the results in to determine p and C according to (5) and (6), plot the results (p and C vs k) using `subplot` and interpret them.

$$p = \frac{\log(\epsilon^{k+1}) - \log(\epsilon^k)}{\log(\epsilon^k) - \log(\epsilon^{k-1})} \tag{5}$$

$$C = \frac{\epsilon^{k+1}}{(\epsilon^k)^p} \tag{6}$$

2 Nonlinear Equations

The steady state heat flux Q of a CSTR for a first order, irreversible reaction is given by

$$\begin{aligned} Q &= \frac{\eta\kappa(\theta)}{1+\kappa(\theta)} + 1 - \theta + K^C(\theta^C - \theta) = 0 \\ \kappa(\theta) &= \kappa_0 \mathrm{exp}\left(-\frac{\alpha}{\theta}\right) \end{aligned} \tag{7}$$

1. Plot the total heat flow Q from and to the reactor (7) vs. the dimensionless reactor temperature θ , for θ between 0.9 and 1.25
   - Use α=49.46; κ_0=2.17×10^{20}; K^C=0.83; η=0.33; θ^C=0.9;
2. Implement and use the secant method in a function to find the three steady state temperatures of the CSTR.
   - Your function file header should read something like `function [x, xvec] = secantRoot(f, x0)` where `f` is a function handle to the function that is to be solved, and `x0` is an initial guess.
   - Store and return all x-values calculated in a vector `xvec`.
   - The calculation steps of the secant method read

     $$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} \\ f'(x_k) &\approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \end{aligned}$$

   - The secant method requires two starting points, use $x_1 = (1+\epsilon)x_0$ as a second point. Suggest a value for ϵ (not too small, why?).
   - Loop `while abs(`x_k` - `x_{k-1}`) > 1e-8` **and** `abs(f(`x_k`)) > 1e-6` **and** `n < 1e5`
   - You will have to work with three x-values at any given iteration, that is x_{k+1}, x_k and x_{k-1}
   - In what range of `x0` can you converge to the intermediate solution? What feature of the function determines which solution is found?
3. Repeat the task of 2. by using the Newton method, i.e. by using the analytical solution instead of the approximation.
4. Repeat the task of 2. by using the finite difference approximation of the derivative such that $f'(x_k) = (f(x_k + h) - f(x_k))/h$, where two values for h, 1e-6 and 1e-16, are used.
5. Use the resulting `xvec` to estimate the convergence order and rate of convergence of the three methods. What do you observe regarding the algorithmic performance of each method? (**Hint:** you can use the built-in keywords `tic` and `toc`)

3 Systems of Nonlinear Equations

The steady state concentrations of a CSTR with two second order reactions taking place reads

$$
\begin{aligned}
0 &= (x_1^{in} - x_1) + \tau(-k_1 x_1 x_2) \\
0 &= (x_2^{in} - x_2) + \tau(-k_1 x_1 x_2 - k_2 x_2 x_3) \\
0 &= -x_3 + \tau(k_1 x_1 x_2 - k_2 x_2 x_3) \\
0 &= -x_4 + \tau(k_2 x_2 x_3)
\end{aligned} \tag{8}
$$

1. Write down the analytical Jacobian matrix for the system of equations (8).
2. Implement the basic Newton method.
   - The multi-dimensional Newton iteration formula reads

     $$x_{k+1} = x_k - \mathbf{J}^{-1}(x_k) f(x_k)$$

   - Your function file header should read something like `function [x, info] = newtonMethod(f, J, x0, tol)` where `f` is a function handle to the function you want to solve, `J` is a function handle that returns the Jacobian matrix, `x0` is an initial guess and `tol` is a vector of tolerances for stopping criteria (relative, absolute errors and number of iterations).
   - As in with the secant method, use a while loop to find the solution.
   - Suggest stopping criteria and failure checks. When can the Newton method fail in general?
   - Use left division $\backslash$ to solve the linear system at every iteration (do not use `inv(J)`!)
   - Let `info` be a `struct` you can use to return additional information, like reason of termination and number of steps needed.
3. Use your Newton algorithm to solve the steady state CSTR numerically.
   - Create a main file and two function files; one that calculates the CSTR equations (8) as functions of x, and one that calculates the analytical Jacobian as a function of x.
   - Use $k_1 = 0.5$, $k_2 = 10$, $x_1^{in} = 1.5$ and $\tau = 5$
   - What is the total conversion of A (x_1) to D (x_4)?
   - Compare your result to what `fsolve()` finds. Try different starting guesses. Can you find more than one solution?
4. (Optional) Find online (same place where you found the exercise sheet) the function `jacobianest`. It is part of a user-made toolbox for estimating derivatives numerically, the DERIVEST suite which can be found on the MatlabCentral.
   - Modify your Newton algorithm so that it uses `jacobianest` to approximate the Jacobian if the input **J** is empty (use `isempty(J)` to check). To provide an empty input, use [] in the call.
   - How many steps are required with the analytical Jacobian for this specific case compared to the numerical Jacobian? Which algorithm takes longer?