

Theme: Soft soils and nonlinear equations Part II

Rules:

- Each student should hand in an individually completed report at latest **November 24**.
 - You may discuss the problem among fellow students. If you receive considerable help from someone, say so in your solutions.
 - Do not copy solutions or code from others. Do not lend your solution or your code to other students.
-

Computer exercises

1. Write a Matlab function that returns the function values and the Jacobian matrix associated with the soil problem given in Part I. The function head should be

```
function [f J] = soilf_J(a, r, p, h)
```

Output parameters are the 3-vector of function values f and the 3-by-3 Jacobian matrix J . Input parameters are: a vector a containing parameter values a_1 , a_2 , and a_3 ; a vector r containing three disk radii; a vector p containing the three pressure values associated with corresponding components in r ; and a nonnegative scalar parameter h . When $h = 0$, the function should return the exact Jacobian matrix, as computed by hand in Part I of the theme. When $h > 0$, the function should return a finite-difference approximation of the Jacobian computed using the step length h . Specify how you have tested that your implementation is correct.

2. Implement Newton's method for the solution of the system of nonlinear equations associated with determining parameters a_1 , a_2 , and a_3 . Your implementation should use the function implemented in task 1, and terminate when current iterate $a^{(n)}$ satisfies $\|f(a^{(n)})\| \leq \delta$ for a user-specified tolerance δ .

Assume that plates of radii 25, 50, and 75 mm require the pressures 69, 83, and 103 Pa to be displaced to the same depth. Using your code with $h = 0$ (exact Jacobians), find corresponding values of constants a_1 , a_2 , and a_3 . Test your code with the following starting guesses: (1, 10, 1), (1, 1, 1), (1, 1, 1000), and (10, 10, 1000).

3. Run the above problem with $h > 0$ (finite-difference approximations of the Jacobian) using the starting value (1, 10, 1). Examine the robustness of your algorithm with respect to the choice of h :
 - (a) How large can h at most be (approximately) for the algorithm not to require more iteration than when using exact Jacobians?
 - (b) What is the smallest value of h (approximately) for which the algorithm works. Why does very small values of h give problems?

Remark. Good choices of h are highly dependent on the scaling of the problem at hand and of the properties of the function f , so the good range of values for h you computed above is unfortunately only relevant for this particular problem!

4. The Newton method you just have implemented could be called the *naive Newton method*. Based on your experiences from using it, list at least two problems with the naive Newton method!
5. A much more robust implementation of Newton's method is contained in the routine `fsolve`, which is available in Matlab's optimization toolbox. Type `doc fsolve` to survey the features of the routine.

Use the following syntax to call `fsolve` for the present problem:

```
func = @(a)soilf_J(a, r, p, 0);  
options = optimset('Jacobian','on');  
a = fsolve(func,a0,options);
```

The `@(a)` in the first row defines `func` as a *function handle* for the Matlab function `soilf_J`, where `a` is regarded as the independent variable for `soilf_J` as a mathematical function.

Use `fsolve` instead of your own implementation of Newton's method to determine a_1 , a_2 , and a_3 . Try the same initial guesses as in problem 2, that is, $(1, 10, 1)$, $(1, 1, 1)$, $(1, 1, 1000)$, and $(10, 10, 1000)$. Comment on differences and similarities!