Theme: Soft soils and nonlinear equations

Soft soil stiffness model

Assume that a layer of soft homogeneous soil of depth $D$ lies on top of harder soil or rock. We wish to determine the pressure needed to sink a large, stiff, and circular plate of radius $r$ a distance $d < D$ into the soil. A simplified model for the required pressure $p$ is

$$p = a_1 e^{a_2 r} + a_3 r,$$  \hspace{1cm} (1)

where, $a_1$, $a_2$, and $a_3$ are constants with $a_2 > 0$. The constants depend on the material properties of the soil, the depth $d$, but not on the radius $r$ of the object. To determine parameters $a_1$, $a_2$, and $a_3$, three small plates with different radii are sunk to the same depth, and the pressures required for this sinkage are measured. Substituting the measurements data into expression (1) leads to a system of nonlinear equations for parameters $a_1$, $a_2$, and $a_3$. Once the parameters have been determined, formula (1) can then be used, for instance, to estimate the minimal size of a plate required to sustain a large load.

Theoretical exercises

1. Write down the nonlinear system of equations (on the form $f(a) = 0$) that emerges from the above strategy using three measurements to determine parameters $a_1$, $a_2$, and $a_3$ in equation (1). Also write down the Jacobian matrix.

2. Let a sequence of real numbers be defined by

$$x_{k+1} = x_k (2 - d x_k), \quad k = 0, 1, 2, \ldots$$

where $d \neq 0$ is a given real number and $x_0$ is given. Assume that the sequence converges to a nonzero number $x^* \in \mathbb{R}$.

   (a) To what number does the sequence converge?
   (b) Show that the convergence rate is quadratic.

Computer exercises

Note: Exercises marked with * are "optional". (These exercises are good for future reference but fall outside of the scope of what will be examined in the present course.)

1. Write a Matlab function that returns the function values and the Jacobian matrix for the nonlinear system you wrote down in task 1 of the theoretical exercises above. The function head should be

   function $[f \ J] = \text{soilf\_J}(a, r, p, h)$

Output arguments are the 3-vector of function values $f$ and the 3-by-3 Jacobian matrix $J$. Input arguments 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 the problem of determining parameters $a_1$, $a_2$, and $a_3$. Your implementation should use the function implemented in task 1, and terminate when the current iterate $a^{(n)}$ satisfies $\|f(a^{(n)})\| \leq \delta$ for a user-specified tolerance $\delta$.

Assume that plates of radii $2.5 \times 10^{-2}$, $5.0 \times 10^{-2}$, and $7.5 \times 10^{-2}$ m require the pressures 69, 83, and 109 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 $a_3$. (Note: use the SI units above, that is, m for radii and Pa for pressures.) Test your code with the following starting guesses $(10, 10, 1000)$, $(1, 10, 1)$, $(1, 1, 1)$, and $(1, 1, 1000)$ for the parameter vector $(a_1, a_2, a_3)$.

3. Solve the problem of determining the parameters $a_1$, $a_2$, and $a_3$ by using your Newton implementation with $h > 0$ (finite-difference approximations of the Jacobian) using the starting value $(10, 10, 1000)$ for $(a_1, a_2, a_3)$. 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 help fsolve or doc fsolve to survey the features of the routine. Use the following syntax to call fsolve for the present problem:

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

The $\Theta(a)$ 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. That is, in the code snippet above, $func$ is defined to be a function such that when called with an argument, say $func([1, 1, 1])$, in reality, the command $soilf_J([1, 1, 1], r, p, 0)$ will be executed, using for the values of $r$ and $p$ the values that these variables contained at the moment when $func = \Theta(a)soilf_J(a, r, p, 0)$ was executed. The above construction allows us to “clean up” $soilf_J$ into a function $func$ with only one independent variable $a$ and hiding the other input variables to $soilf_J$. 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!
6. In the above exercises, we used the bare minimum of only three measurements to determine parameters $a_1$, $a_2$, and $a_3$. In a real-world scenario, it is recommended to use more plates of various radii and determine the coefficients in a (nonlinear) least-squares sense. Assume that we are given $N$ measurements, the parameter estimation problem can be formulated as a least-squares problem, that is

$$
\min_a \frac{1}{2} \sum_{i=1}^{N} f_i(a)^2.
$$

(2)

where $N$ is the number of measurements. (Note the functions $f_i$ are of the same form as previously.) Modify your MATLAB function soilf_J so that given parameters $a$; a vector $r$ containing $N$ disk radii; a vector $p$ containing the $N$ pressure values associated with corresponding components in $r$; this function returns the vector

$$
(f_1(a), f_2(a), \ldots, f_N(a))^T
$$

and matrix

$$
\begin{pmatrix}
\frac{\partial f_1}{\partial a_1} & \frac{\partial f_2}{\partial a_1} & \ldots & \frac{\partial f_N}{\partial a_1} \\
\frac{\partial f_1}{\partial a_2} & \frac{\partial f_2}{\partial a_2} & \ldots & \frac{\partial f_N}{\partial a_2} \\
\frac{\partial f_1}{\partial a_3} & \frac{\partial f_2}{\partial a_3} & \ldots & \frac{\partial f_N}{\partial a_3}
\end{pmatrix}.
$$

7. Assume that plates of radii $2.5 \times 10^{-2}$, $3.0 \times 10^{-2}$, $5.0 \times 10^{-2}$, $6.0 \times 10^{-2}$, and $7.5 \times 10^{-2}$ m require the pressures 69, 74, 83, 94, and 109 Pa to be displaced to the same depth. Solve problem (2) with the Levenberg–Marquardt method by invoking the routine lsqnonlin from MATLAB’s Optimization Toolbox. Use the following syntax to call lsqnonlin

```matlab
func = @(a)soilf_J(a, r, p, 0);
options = optimset('Jacobian','on','Display','testing',
                  'Algorithm','levenberg-marquardt');
k = lsqnonlin(func,a0,[],[],options);
```

where $a_0$ is your initial guess. The options above specify that gradients are available, and that we want to invoke the Levenberg–Marquardt method. Try the same initial guesses as in problem 2, that is, $(1,10,1)$, $(1,1,1)$, $(1,1,1000)$, and $(10,10,1000)$, and record carefully the number of iterations and function evaluations. Using the option 'Display', 'testing', you can read off the values from the function counter.

Note. In some older version of Matlab you have to use the following options instead:

```matlab
options = optimset('Jacobian','on','Display','testing',
                  'LevenbergMarquardt','on','LargeScale','off');
```