School of Mathematics and Statistics
MAST30028 Numerical Methods & Scientific Computing
Week 9

Drag and drop the folder `Week9` from `L: \MAST30028` to `C:\...\MATLAB` and include it in the path. Now MATLAB knows how to find the files in `Week9`.

Recall: there are 2 common ways to solve Linear Least Squares problems — the normal equations

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

or preferably by using the QR factorization of $\mathbf{A}$. From $\mathbf{A} = \mathbf{QR}$, we solve the triangular system

$$\mathbf{R}\mathbf{x} = \mathbf{Q}^T\mathbf{b}$$

This is what the MATLAB \ command does in the case that $\mathbf{A}$ is rectangular with $m > n$ (overdetermined).

Exercise Set 1

This relates to material in Lecture 15.

Sensitivity of the least squares problem

Run the script `ShowLSqV2.m` and enter `10 4 7` as input values. This solves 2 least squares problems using a 10×4 design matrix with condition number of 10^7, first as a zero residual problem and next as a non-zero residual problem.

This illustrates how the sensitivity of least squares problems depends on the residual of the fit. The relative error is magnified by $\approx \kappa(A)$ for small-residual problems, and by $\approx \kappa(A)^2$ for large-residual problems.

QR factorization

There are 2 ways to do the QR factorization. Each repeatedly applies orthogonal transformations to the matrix $\mathbf{A}$ to create zeroes under the diagonal, leaving an upper triangular matrix $\mathbf{R}$. Suitable transformations can be rotations — called Givens rotations — or reflections — called Householder reflections.

a. Run the script `ShowQR` to see how Givens rotations are used to create zeroes under the diagonal. You don't need to worry about how the Givens matrices are constructed i.e. how `Rotate.m` works.

b. MATLAB actually uses Householder reflections. Run the function `qrExample.m` which works through the example on p. 150 of Moler, illustrating the use of QR factorization to find the least squares fit to a quadratic model. You don't need to worry about how the Householder matrices are constructed i.e. how `qrsteps.m` works.

Four ways

The system

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 89 \\ 67 \\ 53 \\ 35 \\ 20 \end{bmatrix}$$

has the least squares solution $(35.125, 32.5, 20.625)^T$.

Obtain this solution:

a. solving the normal equations by Cholesky factorization

b. using the QR factorization, and using the matrix $\mathbf{Q}$

c. using \

d. by using the Singular Value Decomposition (SVD) of $\mathbf{A}$ to get

$$\mathbf{x} = \mathbf{V}\mathbf{y}$$

where

$$\mathbf{\Sigma}\mathbf{y} = \mathbf{U}^{\mathbf{T}}\mathbf{b}$$

Which methods give the smallest error for this simple problem?

Exercise Set 2

This relates to material in Lecture 17.

Steepest descent

The simplest optimization method using gradient information is *steepest descent*. It can be applied to any objective function $f(\mathbf{x})$, not just the least squares case $f = \frac{1}{2}\mathbf{R}(\mathbf{x})^T\mathbf{R}(\mathbf{x})$. The method uses the descent direction $-\nabla f$ and must choose a stepsize to ensure f decreases sufficiently at each step. It is not necessary to solve this line search problem exactly i.e. accurately.

The method can slow down on certain functions with narrow steep valleys, of which the Rosenbrock 'banana' function is the prime example.

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

a. To see how steepest descent fails on such a function, run `bandem.m` in MATLAB R2015b or earlier, which comes from the Optimization Toolbox. In the GUI which pops up, click Info to see a description of the problem. Then click on the 3rd button Steepest to run steepest descent. It fails to find the minimum after 250 function evaluations. Rosenbrock's function has the form of a sum of squares, so we can apply methods designed for nonlinear least squares to it. Click on the last button L-M to run the Levenberg-Marquardt method for this problem. It finds the minimum after 34 function evaluations.

b. Run the script `ShowSDbanana` to see how steepest descent with line search performs in 40 iterations. Then zoom in on the trajectory to see the characteristic zig-zag trajectory that happens when steepest descent stalls.

Note: If you're running MATLAB R2016a or later, instead investigate the example Banana Function Minimization in the MATLAB documentation Examples → Optimization Toolbox → Nonlinear Optimization. Click on the Open Script button and run the first three cells. The third cell runs steepest descent and fails to find the minimum after 600 function evaluations.

Table 1: Data for Gauss-Newton

t	0	1	2	3
y	2.0	0.7	0.3	0.1

Gauss-Newton method

Here we try the Gauss-Newton method on a simple fitting problem from Heath, *Scientific Computing*, 2nd ed. The task is to fit the data above
to the exponential model

$$y \sim x_1 e^{x_2 t}$$

a. I wrote the code `GNdriver.m` to show how undamped Gauss-Newton method performs, both for a good initial guess and a poor one. By undamped, we mean that we compute the GN direction by solving

$$\mathbf{Js} = -\mathbf{R}$$

and update $\mathbf{x}$ by $\mathbf{x} \to \mathbf{x} + \gamma \mathbf{s}$, with $\gamma = 1$ i.e. we take a full GN step.

b. it is more common to instead step along the GN direction a fraction $\gamma < 1$ of the full step, to ensure a minimum reduction in the cost function i.e. avoid overshoot of the minimum along the search direction. The damping factor is chosen by an approximate linesearch along the GN direction. An implementation by Kelley is given by `gaussn.m`, with a driver for the same problem given in `kelleyGNdriver`. You don't need to understand the linesearch used in `gaussn.m`.

How does each method perform for good and poor initial guesses?