Graded Homework 1 Numerical Optimization Fall 2023

The solutions are due by 11:59PM on Friday October 6, 2023

Programming Exercise

Problem 1.1

1.1.a Overview

Your general task is to implement two basic approaches to solving the linear least squares problem

$$\min_{x \in \mathbb{R}^k} \|b - Ax\|_2$$

given $b \in \mathbb{R}^n$ and the matrix $A \in \mathbb{R}^{n \times k}$ with linearly independent columns.

The two algorithms are:

1. Either the approach based on the normal equations (LDL^T or Cholesky factorization may be used) **or** the approach based on the problem transformation

$$H_k H_{k-1} \cdots H_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad H_k H_{k-1} \cdots H_1 b = \begin{pmatrix} c \\ d \end{pmatrix},$$

   where H_i, $1 \leq i \leq k$ are Householder reflectors, $R \in \mathbb{R}^{k \times k}$ is a nonsingular upper triangular matrix with positive elements on the diagonal, $c \in \mathbb{R}^k$, and $d \in \mathbb{R}^{n-k}$.

2. The incremental algorithm for updating the solution

$$x_{min}^{(n)} = \operatorname*{argmin}_{x \in \mathbb{R}^k} \|b_n - A_n x\|_2$$

   to

$$x_{min}^{(n+1)} = \operatorname*{argmin}_{x \in \mathbb{R}^k} \|b_{n+1} - A_{n+1} x\|_2$$

   where all of the matrices are as defined in the class notes. You may implement the weighted version if you wish.

All algorithms should be implemented to be as efficient as possible in space and operations. Your solutions must be clear and concise about justifying your design.

1.1.b Tasks

Task 1

Design, execute and summarize tests to demonstrate that your codes are correct. This should use specific problems with particular properties to make a point, sets of problems to show overall statistical behavior of the accuracy for particular classes of problem, problems with a range of dimensions. You should also show that your incremental code produces consistent results with your "full problem" code.

You may use state-of-the-art libraries and problem solving environments, e.g., Matlab, to assist in designing, testing, and organizing your results. For example, you may use Matlab to check the solutions by comparing it to yours or to generate test problems. As noted in the syllabus, you may also use Matlab or related languages to implement your codes. However, your codes must be written in such a way that the data structures and control structures are clear and easily translated into a compiled and typed language.

Task 2

Consider regularized linear least squares of the form

$$\min_{x\in\mathbb{R}^n} \|b-x\|_2^2 + \lambda\|Lx\|_2^2$$
$$x,\ b\in\mathbb{R}^n,\quad L\in\mathbb{R}^{n-1\times n}$$

$$e_i^T L e_i = 1,\quad e_i^T L e_{i+1} = -1,\ \text{ and all other elements of } L \text{ are } 0.$$

For example, if $n=10$, we have $L\in\mathbb{R}^{9\times 10}$

$$L = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}.$$

This is a simple way to mitigate noise in the observations of some value. In this case $b = x_{true} + w$ where $x_t rue$ contains $\xi_1^{true},\ldots,\xi_n^{true}$ that are n samples of the signal $x_{true}(t_i)$ at time values $t_1 = 0, t_2, \ldots, t_{n-1}, t_n = 4$, and w contains $\omega_1^{Gauss},\ldots,\omega_n^{Gauss}$ that are n samples of a standard Gaussian distribution representing noise on the observed signals, i.e., $e_i^T b = \beta_i$ is what is received as the i-sample rather than ξ_i^{true}. To improve the quality of the sample β_i the regularized least squares problem is solved to obtain $x_{RLS}\in\mathbb{R}^n$ which should be a better

approximation of x_{true} than b if $\lambda > 0$ is chosen correctly and assuming the simple smoothing penalty term is sufficient for the noise properties. The idea behind L is that it assumes that the two successive elements of x_{true} are close in value. Of course, that is increasingly false if the samples of x_{true} grow in separation in t, i.e., fewer samples.

Consider the values $\lambda = 1, 10, 100, 1000$ and $n = 100, 200, 300, 400, 500$. For each pair (λ, n) run several instances of generating x_{true}, contaminating it with a randomly generated w to get b from which you recover x_{RLS}. Organize and discuss the quality of the reconstruction x_{RLS} vs x_{true} in terms of n and λ.

Use the following sinusoid as the underlying signal

$$x_{true}(t) = \sin(t) + t\cos^2(t), \quad 0 \leq t \leq 4$$

and use $t_1 = 0$, $t_n = 4$ for n evenly separated time values.

You can use either of your linear least squares codes for this task but you are encouraged to use the incremental version since this is a particularly important approach for real-time signal processing.

You may, of course, explore the problem further with other $x_{true}(t)$, noise variances, magnitudes and distributions.

1.1.c Organization of Results

One of the main goals of this assignment is for you to learn how to organize and present your design, execution and analysis of observations. As noted above, you may use graphics, statistical, and algorithmic support from established libraries and environments such as Matlab. The algorithmic support, of course, may not be used as a substitute for your codes but it can be used for problem generation, i.e., random number generation, computation of sums for comparison, and computation of various values useful for presentation of results. Make use of specific tests, i.e., particular sums of a particular length, and its results to make points, e.g., verifying very well or very ill conditioned examples as well as large groups of sums that are some how related to make a point, e.g., mean and variance of accuracy of the computed sum. Histograms, graphs and appropriate tables should be used to compress data and support conclusions. Your description of your codes and results must also be clear as to how you used any environment or library you did not write as part of your solutions.

Simply running a few examples and comparing the answers to the exact sums is not acceptable and will not prepare you for more complicated algorithm evaluation later in your use of optimization for research or other classes.

1.1.d Comments on Test Problems

For the linear least squares codes and experiments in Task 1, you should include for various values of n, k, and b problems for at least the three situations:

1. $n = k$, i.e., a square nonsingular matrix A where $x_{min} = A^{-1}b$.

2. $n > k$ and $Ax = b$ for $b \in \mathbb{R}^n$ and $b \in \mathcal{R}(A)$ i.e., a rectangular matrix A with full column rank and a vector b that define a consistent set of overdetermined equations.

3. $n > k$ and $b \in \mathbb{R}^n$ and $b \notin \mathcal{R}(A)$ i.e., a rectangular matrix A with full column rank and a vector $b = b_1 + b_2$, $b_1 \in \mathcal{R}(A)$, $b_2 \notin \mathcal{R}(A)$, $b_2 \neq 0$, that define a linear least squares problem with a nonzero residual $r_{min} = b_2 = b - Ax_{min}$

This requires careful construction of the test problems. See the discussion below.

When generating test problems, you may use library routines available in whatever language environment you are using. You may also use its factorizations, e.g., to compute the QR factors of a given matrix $A = QR$ where $A, Q \in \mathbb{R}^{n\times k}$, $Q^TQ = I_k$ and $R \in \mathbb{R}^{k\times k}$ is nonsingular, upper triangular and has positive diagonal elements. Also note $\mathcal{R}(A) = \mathcal{R}(Q)$ as we have discussed in class.

Note you can also generate the matrices of the types required for this assignment, e.g., nonsingular matrices, full rank rectangular matrices, i.e., $A \in \mathbb{R}^{n\times k}$ for $n \geq k$ with linearly independent columns, and isometries, i.e., $Q \in \mathbb{R}^{n\times k}$ for $n \geq k$ such that $Q^TQ = I_k$. The techniques are reviewed below. If there is confusion on how to generate these matrices ask in class or set up an appointment to discuss them.

A nonsingular $n \times n$ matrix, A, can easily be generated from a random $n \times n$ matrix G by adding an $n \times n$ diagonal matrix, D. (The nonzero diagonal elements of D can be positive or negative.) The magnitude of the nonzero elements of D are taken so that the matrix $G + D$ is strictly diagonally dominant. Since you may want the test matrix not to be strictly diagonally dominant for some tests, you can postprocess $G + D$ by applying random row and column permutations $A = P_{rows}(G + D)P_{cols}$. This preserves nonsingularity while generically destroying diagonal dominance.

An $n \times k$ matrix, A, with linearly independent columns can be generated using the technique described above to form an $n \times n$ nonsingular matrix. Selecting k columns of an $n \times n$ nonsingular matrix yields an $n \times k$ matrix, A, with linearly independent columns. You may also select k rows to get a $k \times n$ matrix and then take A to be the transpose.

A technique that avoids producing an $n \times n$ matrix in an effort to avoid forming large dense matrices, starts by forming a set of k linearly independent vectors by defining a lower trapezoidal matrix, $L \in \mathbb{R}^{n\times k}$ with nonzero diagonals. The columns must be linearly independent due to the locations of the 0's and the nonzeros on the diagonal. To create the matrix A that does not have the upper triangular part 0 simply postmultiply by a $k \times k$ nonsingular matrix. This can be created as above or more simply by generating a random $k \times k$ upper triangular matrix with nonzeros on the diagonal. Note that this technique also generates an $n \times n$ nonsingular matrix when $k = n$. As before random permutations can also be applied to scramble the matrices.

Routines in Matlab or any state-of-the-art library can be used to verify that the columns of the matrix A generated are sufficiently linearly independent, i.e., not close to being dependent.

In addition to using built-in primitives of MATLAB or similar environments to generate orthogonal matrices it is possible to generate them via simple techniques. For example, an

$n \times n$ rotation matrix can be easily defined by considering a random index pair (i, j) and random angle θ. The matrix, Z, that is the identity everywhere except positions (i, i), (j, j), (i, j), (j, i), where it is taken to be

$$\cos\theta = e_i^T Z e_i, \quad \cos\theta = e_j^T Z e_j, \sin\theta = e_i^T Z e_j, \quad -\sin\theta = e_i^T Z e_j$$

is a plane rotation and orthogonal, as we have discussed in class. Selecting many, say s, random index pairs (i, j) and random angles θ and multiplying all of the rotations they define together yields an orthogonal matrix

$$Q = Z_1 Z_2 \cdots Z_s.$$

Of course, you need to select enough pairs and angles, s, so that the matrix is dense.

Similarly, one can use reflectors to generate an orthogonal matrix Q. Simply choose several random vectors, v_i, $i = 1, \ldots, s$ for a large s. Then for each v_i form an elementary reflector

$$Q_i = I - 2u_i u_i^T, \quad u_i = \frac{1}{\|v_i\|_2} v_i$$

and $Q = Q_1 Q_2 \cdots Q_s$. Taking $s >> n$ is a good way of scrambling the directions defining Q.

Once an $n \times n$ orthogonal Q is computed selecting randomly k of the n columns yields an $n \times k$ matrix $\tilde{Q}$ that has orthonormal columns, i.e., $\tilde{Q}^T \tilde{Q} = I_k$. Of course, it is not necessary to form the $n \times n$ orthogonal Q to take k selected columns. After selecting the indices of the columns of Q you plan to use, then rather than computing all of Q by taking the products of the s rotations or reflectors you can simply apply each one in turn to a set of k vectors that are initialized to the standard basis elements defined by the randomly selected column indices.

For example, suppose you want columns 2, 10 and 50 of Q and Q is defined as the product of s some set of reflectors. The isometry $\tilde{Q} \in \mathbb{R}^{n \times 3}$ is efficiently computed using

$$\tilde{Q} = \begin{pmatrix} e_2 & e_{10} & e_{50} \end{pmatrix}$$

$$for \quad i = 1, \ldots, s \quad \tilde{Q} \leftarrow Q_i \tilde{Q} \quad end$$

Of course, there is no reason to use only reflectors or rotations. A mix of reflectors and rotations can also be used.

When using either rotations, reflectors or a combination, it is necessary to compute the the matrix-matrix product or matrix-vector products efficiently so as not to take huge amounts of time when creating test problems since you must run many of them. Make sure you exploit all of the structure available to gain computational and storage efficiency. You should of course point out anything you do along these lines in your solution.

You should verify that the matrix computed has orthonormal columns to at least single precision accuracy. This computation of $Q^T Q$ or $\tilde{Q}^T \tilde{Q}$ and comparison to I_n or I_k should be performed in double precision.

A final simple way to generate a problem, a basis and an orthonormal basis is to generate a random matrix $A \in \mathbb{R}^{n\times k}$ and then use a robust state-of-the-art library algorithm, e.g., in Matlab, to compute the QR factors of a given matrix $A = QR$ where $A, Q \in \mathbb{R}^{n\times k}$, $Q^TQ = I_k$ and $R \in \mathbb{R}^{k\times k}$ is nonsingular, upper triangular and has positive diagonal elements. We have that $\mathcal{R}(A) = \mathcal{R}(Q)$. The matrix Q can be ysed as discussed below along with A to generate particular problems. Once again your should check the matrix R to make sure it is sufficiently nonsingular by examining its diagonal elements or computing its condition number using a state-of-the-art library algorithm.

For the three situations mentioned earlier, you should run problems where you have created the problem with a known solution and those for which you do not know the solution. **You must consider each of these classes of problems in your report.**

To form a consistent overdetermined set of equations given A with a known solution, z, simply set b to $b = Az$. These computations should also be done in double precision especially if your code is in single precision.

A linear least squares problem with known solution, x_{min}, and nonzero residual $r_{min} = b - Ax_{min} \neq 0$ can be created by modifying a consistent overdetermined system with known solution as follows:

1. Choose your desired solution x_{min}
2. Compute $b_1 \in \mathbb{R}^n$ with $b_1 \in \mathcal{R}(A)$ by $b_1 = Ax_{min}$.
3. Set b to $b = b_1 + b_2$ where b_2 is any vector that is chosen to be orthogonal to $\mathcal{R}(A)$.

The linear least squares problem

$$\min_{x\in\mathbb{R}^k} \|b - Ax\|_2$$

therefore has solution x_{min} and residual $r_{min} = b_2$. Generating $b_2 \in \mathcal{R}^{\perp}(A)$ requires some more effort.

To generate b_1 and b_2 in their appropriate spaces, it is convenient therefore to generate an orthonormal basis, $Q \in \mathbb{R}^{n\times k}$, and then generate $A \in \mathbb{R}^{n\times k}$ so that $\mathcal{R}(A) = \mathcal{R}(Q)$. Given Q, A can be generated by computing $A = QM$ where $M \in \mathbb{R}^{k\times k}$ is a random nonsingular matrix possbily with convenient structure, e.g., upper triangular with positive diagonal elements so systems involving $Mw = v$ are easily solved. As noted above you can also generate a random $A \in \mathbb{R}^{n\times k}$ and recover a Q and R from a state-of-the-art library.

Generating b_1 and b_2 is a bit more complicated. Given any vector $v \in \mathbb{R}^n$ we have

$$v = v_1 + v_2, \quad v_1 \in \mathcal{R}(Q) \quad v_2 \in \mathcal{R}^{\perp}(Q)$$
$$v_1 = Q(Q^Tv) \quad v_2 = v - v_1$$

So one way of generating these vectors is to take a random vector v and compute v_1 and v_2 as above. Of course, a priori you will not know the relative magnitudes of v_1 and v_2. It is possible that the random vector v will be almost entirely in $\mathcal{R}(Q)$ or almost orthogonal to it. So you may have to try several random v until you get two reliable directions v_1 and v_2.

In any case, you should check that $\cos\theta_{1,2} = v_1^T v_2/(\|v_1\|_2\|v_1\|_2)$ is suitably small to verify that finite precision has not caused difficulties. You should, in fact, try several such pairs of various dimensions to test your code. Additionally, to analyze your code's performance, given any such pair, you can create multiple b vectors by taking various combinations of v_1 and v_2 in a controlled manner, i.e., set

$$b = b_1 + b_2 = \alpha_1 v_1 + \alpha_2 v_2$$

keeping $\|b\|_2$ constant. When α_1 is large relative to α_2 the system is closer to consistent than when α_2 dominates.

Note that this technique can also be used when you have already selected b_1 as mentioned above when designing a problem with a known solution. Taking a random vector v and computing $v_2 = v - Q(Q^T v) \in \mathcal{R}^{\perp}(Q)$ provides the v_2 to use in the expression

$$b = b_1 + b_2 = b_1 + \alpha_2 v_2$$

to preserve the chosen solution of the problem and b_1 while allowing α_2 to be used to adjust the dominance or lack thereof for b_1 in b.

When you generate problems for which you do not know the solution a priori, you should think about how you would determine if the solution is reasonable. For example, you should examine the residual carefully to make sure it satisfies all required conditions. Also note, since the solution is supposed to minimize the norm of the residual over all x, you can also check residuals generated for randomly selected x vectors and compare their norms to the norm of the residual generated by x_{min}. Finally, as noted above, you can compare your results to other libraries or routines available to you.