Math 6397 — Applied Inverse Problems

Problem Set 2

Due on October 26, at 2:30 PM

To complete this assignment, put all functions you implement into a single folder and submit the content of this folder to blackboard. Each main question will have a script to run the code. How you implement the optimization algorithms is up to you. If you choose to work on the templates I provide, just put them into your local folder and modify them. The first thing to do is implement the objective function and make sure the derivative check works. If not, you have a bug in your objective function.

1 Background

1.1 Line Search Methods

We are going to consider the numerical solution of unconstrained optimization problems of the general form

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x}), \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the unknown and $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function. The type of methods we are going to use fall into the category of so-called iterative line search methods. These methods are of the general form $\mathbf{x}_{k+1} = \mathbf{x} + t_k\mathbf{s}_k$, $k = 1, 2, 3 \ldots$, with line search parameter $t_k > 0$ and search direction $\mathbf{s}_k \in \mathbb{R}^n$. More specifically, we will consider approaches of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k\mathbf{B}_k\nabla f(\mathbf{x}_k), \tag{2}$$

where $\nabla f \in \mathbb{R}^n$ denotes the gradient of the objective function f and $\mathbf{B}_k \in \mathbb{R}^{n,n}$ is a scaling matrix. (We are going to consider different choices for $\mathbf{B}_k$ below.) The key questions we are going to address are how to select the line search parameter t_k to obtain a sufficient decrease in the objective function f and how to compute adequate search directions $\mathbf{s}_k = -\mathbf{B}_k\nabla f(\mathbf{x}_k)$. We consider a search direction to be a descent direction if the directional derivative satisfies $\mathbf{s}_k^{\mathsf{T}}\nabla f(\mathbf{x}_k) < 0$. An outline of the code can be found in Alg. 1.

Notice that iterative schemes like the one presented in (2) require an initial guess $\mathbf{x}_0$. The scripts referred to below provide an adequate initial guess. In general, the selection of a good initial guess is a challenging problem by itself, especially when considering non-convex problems.

1.2 Derivative Check

One of the many traps in optimization is working with an erroneous derivative. The following test provides a simple way of checking the implementation of a derivative. To this end, let f be a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ and let $\mathbf{v} \in \mathbb{R}^n$ be an arbitrary vector in the Taylor expansion

$$f(\mathbf{x} + h\mathbf{v}) = f(\mathbf{x}) + h\nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{v} + \mathcal{O}(h^2).$$

Algorithm 1 Generic line search method for solving an unconstrained optimization problem of the form (1). This pseudo-code is implemented in optimization/runOptimizer.m.

$x_1 \leftarrow \mathbf{0}$, $k \leftarrow 0$, tol $\leftarrow$ 1e−4, maxit $\leftarrow$ 1000
$[f_0, \mathbf{g}_0] \leftarrow \text{fun}(\mathbf{x}_0)$
while not converged **do**
    $\mathbf{s}_k \leftarrow \text{getSearchDir}(\text{fun}, \mathbf{x}_k, \mathbf{g}_k)$
    $t_k \leftarrow \text{doLineSearch}(\text{fun}, \mathbf{x}_k, \mathbf{s}_k)$
    if $t_k == 0$ **then**
        break
    end if
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + t_k \mathbf{s}_k$, $k \leftarrow k + 1$
    $[f_k, \mathbf{g}_k] \leftarrow \text{fun}(\mathbf{x}_k)$
    converged $\leftarrow \text{checkConvergence}(\mathbf{g}_0, \mathbf{g}_k, k, \text{tol}, \text{maxit})$
end while

The vector g is the derivative of f if and only if the difference

$$|f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x}) - h\mathbf{g}^\mathsf{T}\mathbf{v}|$$

is essentially quadratic in h. Similarly, we can check the implementation of the Hessian based on the Taylor expansion

$$f(\mathbf{x} + h\mathbf{v}) = f(\mathbf{x}) + h\nabla f(\mathbf{x})^\mathsf{T}\mathbf{v} + \frac{1}{2}h^2\mathbf{v}^\mathsf{T}\nabla^2 f(\mathbf{x})\mathbf{v} + \mathcal{O}(h^3).$$

Accordingly, the matrix $\mathbf{H}$ is the second derivative of f if and only if the difference

$$|f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x}) - h\mathbf{g}^\mathsf{T}\mathbf{v} - 0.5h^2\mathbf{v}^\mathsf{T}\mathbf{H}\mathbf{v}|$$

is essentially cubic in h. The function that implements this test is optimization/checkDerivative.m.

As you can see, a common choice for the steps size is `h = logspace(-1,-10,10)`. Notice that you should use random vectors to make sure that the derivative check generalizes. The "output" of the derivative check is the following: A plot of the trend of the error (first, second, and third order) versus the step size h. More importantly, the values will be printed in your command window. Inspect the trend of the exponent of the different errors. You will see the predicted behavior (if the derivatives are correct) up until approximately machine precision, i.e., $\mathcal{O}(1e{-}16)$. If you do not see the expected trend, you have a bug in your code.

This derivative check can handle Hessians in matrix form and implemented as function handles that apply the matrix to a vector.

1.3 Optimization Problems

1.3.1 Problem (A): Rosenbrock Function

The first optimization problem we are going to consider is given by

$$\underset{\mathbf{x}\in\mathbb{R}^2}{\text{minimize}} \;\; (a - x_1)^2 + b(x_2 - x_1^2)^2. \tag{3}$$

This objective function is called the Rosenbrock function. It is a non-convex function. Typical parameters for a and b are 1 and 100, respectively.

1.3.2 Problem (B): Regularized Least Squares Problem

We consider the (unconstrained) regularized least squares problem of the general form

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}\|\mathbf{K}\mathbf{x}-\mathbf{y}^\delta\|_2^2+\frac{\alpha}{2}\|\mathbf{L}\mathbf{x}\|_2^2, \tag{4}$$

with regularization parameter $\alpha > 0$, regularization operator $\mathbf{L} \in \mathbb{R}^{n,n}$, $\mathbf{K} \in \mathbb{R}^{m,n}$, $\mathbf{y}^\delta \in \mathbb{R}^m$, and $\mathbf{x} \in \mathbb{R}^n$.

2 Assignments

1. Implement an iterative solver to solve the optimization problem (A) in (3). A template for implementing the objective function is kernel/objFunRB.m. If you run this function without any input arguments it will perform a derivative check. You can find a working implementation of an objective function for a least squares problem in kernel/objFunLSQ.m. The main script to execute this iterative solver is scripts/optprb/scSolveRB.m. This script sets up the appropriate functions to solve the problem. A template for the iterative solver described in Alg. 1 is implemented in optimization/runOptimizer.m.

   a) Implement a backtracking line search to compute the line search parameter t_k. In particular, we will accept a step size $t > 0$ if the so called Armijo or sufficient decrease condition

   $$f(\mathbf{x}_k + t\mathbf{s}_k) \leq f(\mathbf{x}_k) + ct\nabla f(\mathbf{x}_k)^\mathsf{T}\mathbf{s}_k, c \in (0,1)$$

   is satisfied. A template for the implementation of your line search algorithm can be found in optimization/doLineSearch.m. Notice, that this template has an upper bound for the number of backtracking steps to avoid an infinite loop. If you cannot find a decrease after 24 steps, set the line search parameter t to zero (to indicate that the line search failed).

   b) Implement a function to compute the search direction $\mathbf{s}_k \in \mathbb{R}^n$. Consider three methods for computing the search direction: (*i*) gradient descent for which $\mathbf{B}_k$ is equal to an $n \times n$ identity matrix, (*ii*) limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS), and (*iii*) Newton's method with $\mathbf{B}_k = (\nabla^2 f(\mathbf{x}_k))^{-1}$. For the latter, you can compute the Newton step $\mathbf{s}_k$ by solving the linear system

   $$\mathbf{H}\mathbf{s}_k = -\mathbf{g}_k,$$

   where $\mathbf{H} := \nabla^2 f(\mathbf{x}_k) \in \mathbb{R}^n$ and $\mathbf{g}_k := \nabla f(\mathbf{x}_k) \in \mathbb{R}^{n,n}$. Use a direct solver (in particular, Matlab's backslash command) to do so. A template for computing the search direction can be found in optimization/getSearchDir.m.

   To switch between these two methods, you can either set the flag `method` to `'gdsc'` for gradient descent, to `'bfgs'` for L-BFGS, or `'newton'` for Newton's method, respectively, in the main script scripts/optprb/scSolveRB.m.

   c) The final task is to determine when to terminate the iterative solver. To do so, implement a function to check a set of convergence criteria. A template for this function is provided in optimization/checkConvergence.m. We are going to terminate the solver if one of the following

conditions is satisfied:

$$(a) \quad \|\nabla f(\mathbf{x}_k)\|_2 \leq \epsilon_g \|\nabla f(\mathbf{x}_0)\|_2$$
$$(b) \quad \|\nabla f(\mathbf{x}_k)\|_2 \leq 1\text{e}{-6}$$
$$(c) \quad k \geq k_{\max}$$

Here, $k_{\max} \in \mathbb{N}$ is a user defined upper bound on the maximum number of iterations and $\epsilon_g > 0$ is a user defined tolerance for the relative reduction of the gradient. Both of these parameters are input arguments to the function optimization/checkConvergence.m and are set in optimization/runOptimizer.m.

d) Once you have implemented all functions, run the script scripts/optprb/scSolveRB.m. Switch between the implemented optimization methods by setting the flags `method` accordingly. Make sure the solvers converge to an adequate solution.

2. We are going to consider problem (B) next. This assignment serves as a first step towards developing a matrix-free Newton solver large-scale inverse problems. The basic building blocks of the algorithm introduced in the former section should work without any modifications. The problem we are going to consider first has only three unknowns. The operators for the optimization problem (4) are as follows: We select $\mathbf{L}$ for the Tikhonov regularization to be a 3×3 identity matrix and

$$\mathbf{K} = \begin{bmatrix} 2 + 1\text{e}{-3} & 3 & 4 \\ 3 & 5 + 1\text{e}{-3} & 7 \\ 4 & 7 & 10 + 1\text{e}{-3} \end{bmatrix}.$$

The matrix $\mathbf{K}$ can be constructed via $\mathbf{K} = \mathbf{B}^\mathsf{T}\mathbf{B} + \gamma \mathbf{I}_3$, $\gamma = 1\text{e}{-3}$, where

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}.$$

The right hand side of the linear system $\mathbf{K}\mathbf{x} = \mathbf{y}$ is computed by applying the matrix $\mathbf{K}$ to a vector $\mathbf{x}_{\text{true}}$ (the true solution of our problem). Subsequently, we can compare the computed solution $\mathbf{x}_\alpha$ to this true solution. To make the problem more complicated, we perturb the right hand side $\mathbf{y}$ by a random vector $\eta \in \mathbb{R}^n$ to obtain $\mathbf{y}^\delta = \mathbf{y} + \delta\eta$, $\delta > 0$, $\eta_i \propto \mathcal{N}(0, 1)$. The script to run this problem is scripts/optprb/scSolveRLSQ.m.

a) Implement the objective function. To do so, you need to compute the gradient and the Hessian matrix for this optimization problem. A template is given in kernel/objFunRLSQ.m. Make sure the derivative check provides an adequate result.

b) Execute gradient descent, L-BFGS, and Newton's method. What are your observations?

c) Replace the direct solver (i.e., Matlab's backslash) in optimization/getSearchDir.m for computing the Newton step with an iterative solver, more precisely, an inexact Newton method. Use PCG with a quadratic forcing sequence. We will supply PCG only with an expression for the action of a matrix on a vector (i.e., an expression for the matrix-vector-product ("matvec")). To do so, we will

provide a "fake" matrix-free version of the Hessian matvec by replacing the stored Hessian matrix in your objective with a function handle. That is, instead of returning `d2f = H`, where **H** is your expression for the Hessian matrix, you return a function handle `d2f = @(x)H*x` that applies the $n \times n$ matrix **H** you have formed to a vector. In a first step, use Matlab's supplied `pcg` algorithm. Subsequently, write your own version of a CG algorithm that includes a check for the curvature. A template for implementing your CG algorithm can be found in optimization/runCG.m. To switch between the inexact, matrix-free PCG version of your Newton method and the direct solver in optimization/getSearchDir.m you can use the boolean check `isa(d2fc,'function_handle')` that returns `true`, if `d2fc` is a function handle.

3. Next, we consider a two-dimensional problem of the form $\mathbf{Kx} = \mathbf{y}$ you have already worked on in your first homework assignment. Instead of directly solving the optimality system (iteratively), we are now going to use line search methods to solve the regularized least-squares formulation in (4). The script that with the setup of this problem is given in scripts/optprb/scSolveRLSQDCV.m.

   a) The first step is to implement the objective function. A template can be found in kernel/objFunRLSQDCV.m. Use the same matrix-free approach you considered in your first homework to evaluate the objective function, the gradient and the Hessian. That is, to evaluate this matrix vector product $\mathbf{Kx}$, we will apply one-dimensional blurring operators $\mathbf{K}_1$ and $\mathbf{K}_2$ along the individual coordinate directions. This results in significant savings in terms of memory-requirements. Use the same kernel **K** you used in your former homework for the 2D case. You can use the implementation provided in kernel/getKernel2D.m to construct this kernel.

   One additional difficulty is that this matrix-free version of applying **K** to **x** is defined for a two-dimensional layout of **x** (i.e., you assume that your vector **x** is stored as a matrix of size, say, $m \times m$. The optimization algorithm you have implemented assumes that your vector **x** is stored as a column vector (lexicographical ordering), i.e., $\mathbf{x} \in \mathbb{R}^n$ with $n = m^2$. Consequently, to be able to apply this solver to this two-dimensional problem you need to switch between these layouts. You can use Matlab's built in `reshape` function to switch between an $m \times m$ matrix layout needed to apply $\mathbf{K}_1$ and $\mathbf{K}_2$ (and to visualize the data) and an $n \times 1$ (i.e., $m^2 \times 1$) lexicographical layout. Once you implemented the objective function, perform the derivative check to make sure your derivative is correct.

   b) After implementing the objective function, execute gradient descent, L-BFGS, and Newton's method to solve this problem. What are your observations?

4. TBA