Advanced Programming for Scientific Computing

Luca Formaggia and Alberto Artoni

23 jan 2025
A.Y. 2023 – 2024

General rule

Provide the code and all other relevant material in a zip, gzip or tgz file named **name-surname** and upload it on WeBeep. Make sure that the upload has been successful. The file should not contain compiled code, object files or compiled libraries, only text files (source code + possible data file or solution files). I should be able to compile the code using make, and there should be a `README.md` file with simple instructions on how to run the test cases. *Comments in the source are appreciated. You can choose to do thing differently than what proposed here, just justify your choices in the `README.md` file.*

Setting

An interesting technique for constrained minimization problems often used in machine learning is the *Augmented Lagrangian*, also known as methods of multipliers. The problem it addresses is a constrained minimization of the form[1]: find $\mathbf{x} \in \mathbb{R}^n$ such that

$$\begin{cases} \mathbf{x} = \operatorname{argmin}_{\mathbf{y}\in\mathbb{R}^n} f(\mathbf{y}), \\ \mathbf{b}(\mathbf{x}) = \mathbf{g} \end{cases} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a (typically convex) function and $\mathbf{b} : \mathbb{R}^n \to \mathbb{R}^d$, with $d < n$, is a set of linear or non-linear constraints, and $\mathbf{g} \in \mathbb{R}^d$. More details on the solvability of the problem may be found in any good textbook and are omitted here.
The augmented Lagrangian technique rewrites the problem by introducing the dual variable $\boldsymbol{\lambda} \in \mathbb{R}^d$ (Lagrange multiplier), and solve the minmax problem

$$(\boldsymbol{\lambda}, \mathbf{x}) = \operatorname{argmax}_{\boldsymbol{\mu}\in\mathbb{R}^d} \min_{\mathbf{y}\in\mathbb{R}^n} f(\mathbf{y}) + \frac{\rho}{2}\|\mathbf{b}(\mathbf{y}) - \mathbf{g}\|^2 + \boldsymbol{\mu}^T(\mathbf{b}(\mathbf{y}) - \mathbf{g}), \tag{2}$$

where $\rho > 0$ is a penalization parameter. Here and in the following, $\|\mathbf{x}\|$ denotes the 2-norm:

$$\|\mathbf{x}\| = \sqrt{\sum x_i^2}$$

[1]The general setting for the Augmented Lagrangian technique may be found in any good text on numerical optimization.

The *method of multipliers* for solving (2) is as follows: Given $\boldsymbol{\lambda}^{(0)}$ (often just null vectors), a tolerance ϵ (and a maximal number of iterations) do, for $k = 0, \ldots,$

- Solve for $\mathbf{x}$: $\mathbf{x}^{(k+1)} = \text{argmin}_{\mathbf{y}\in\mathbb{R}^n} f(\mathbf{y}) + \frac{\rho}{2}\|\mathbf{b}(\mathbf{y}) - \mathbf{g}\|^2 + {\boldsymbol{\lambda}^{(k)}}^T(\mathbf{b}(\mathbf{y}) - \mathbf{g})$
- Update $\boldsymbol{\lambda}$: $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \rho(\mathbf{b}(\mathbf{x}^{(k+1)}) - \mathbf{g})$
- Stop if either $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \epsilon$ or maximal number of iteration exceeded. You ay also want to control the convergence of the dual variable or the satisfaction of the constraints.

Quadratic cost function and linear constraints

This is the general setting. Here we will first focus on a special case where f is a quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{d}\|^2 = \frac{1}{2}(A\mathbf{x} - \mathbf{d})^T(A\mathbf{x} - \mathbf{d}) \tag{3}$$

where $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, $d \in \mathbb{R}^m$. While, the constraints are taken linear, that is

$$\mathbf{b}(\mathbf{x}) = B\mathbf{x}, \tag{4}$$

with $B \in \mathbb{R}^{d\times n}$, with $d < n$. The problem is well posed if $\ker(A) \cap \ker B = \emptyset$ and B has full rank. In this case the minimization problem in the first step of the method of multipliers reduces to solving a linear system

$$\boxed{\begin{aligned}(A^TA + \rho B^TB)\mathbf{x}^{(k+1)} &= A^T\mathbf{d} + B^T(\rho\mathbf{g} - \boldsymbol{\lambda}^{(k)}), \qquad (5)\\ \boldsymbol{\lambda}^{(k+1)} &= \boldsymbol{\lambda}^{(k)} + \rho(B(\mathbf{x}^{(k+1)}) - \mathbf{g})\end{aligned}}$$

where the matrix in the left-hand side is symmetric positive definite thanks to the solvability condition, while the update step remains unchanged.
This simplified problem is quite common in some machine learning algorithms, where $A\mathbf{x} = \mathbf{d}$ represents an overdetermined set of linear equations for the parameters $\mathbf{x}$ (produced, for instance, from several samples), while $B\mathbf{x} - \mathbf{g} = \mathbf{0}$ is a set of constraints that we need to satisfy exactly.

Application to differential problems

Augmented Lagrangian techniques can be used also to impose conditions on a differential problem. We consider here a very simple case: find the function $u : [0, 1] \to \mathbb{R}$ solution of

$$\begin{cases} -u'' = f & \text{in } (0,1) \\ u(0) = g_0, \quad u(1) = g_1. \end{cases} \tag{6}$$

It is equivalent to a constrained minimization problem (not reported here), which may be recast in the augmented Lagrangian form as

$$(\mathbf{u}, \boldsymbol{\lambda}) = \text{argsup}_{\boldsymbol{\mu}\in\mathbb{R}^2} \inf_w \frac{1}{2}\int_0^1 |w'|^2 + \frac{\rho}{2}(|w(0) - g_0|^2 + |w(1) - g_1|^2) + \boldsymbol{\mu}^T \begin{bmatrix} w(0) - g_0 \\ w(1) - g_1 \end{bmatrix} + \int_0^1 fw \tag{7}$$

Setting the variations to zero we can then obtain the method of multipliers for this case as

- Solve for u:

$$\int_0^1 u'v' + \rho[(u(0)-g_0)v(0) + (u(1)-g_1)v(1)] + \boldsymbol{\lambda}^{(k)^T}\begin{bmatrix} v(0) \\ v(1) \end{bmatrix} = 0, \quad \forall v \in H^1(0,1).$$

- Update $\boldsymbol{\lambda}$: $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \rho \begin{bmatrix} u(0)-g_0 \\ u(1)-g_1 \end{bmatrix}$ where $\boldsymbol{\lambda} \in \mathbb{R}^2$

It can be found that if we discretize the previous relations using linear finite elements on a uniform grid of n *nodes*, and we indicate with $\mathbf{x} \in \mathbb{R}^n$ the discrete solution, we can write an iterative scheme of the same form as seen in the first part (only the matrices are slightly different). Namely,

Starting from a given $\boldsymbol{\lambda}^{(0)}$ (usually zero) we need to perform

$$(A + \rho B^T B)\mathbf{x}^{(k+1)} = \mathbf{f} - B^T\boldsymbol{\lambda}^{(k)} + \rho B^T\mathbf{g}, \tag{8}$$

while the update is

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \rho(B\mathbf{x}^{(k+1)} - \mathbf{g}). \tag{9}$$

Here,

$$A = \frac{1}{h}\begin{bmatrix} 1 & -1 & 0 & \cdots & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \vdots & \vdots & 0 \\ & & \vdots & & & & \\ & \cdots & & & -1 & 2 & -1 \\ & \cdots & & & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad B = \begin{bmatrix} 1 & \cdots\cdots & 0 \\ 0 & \cdots\cdots & 1 \end{bmatrix} \in \mathbb{R}^{2\times n},$$

$$\mathbf{g} = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} \quad \text{and } h = \frac{1}{n+1}$$

while $\mathbf{f}$ discretizes the forcing term f. For a *constant* f, it is just

$$\mathbf{f} = h\begin{bmatrix} 1/2 \\ 1 \\ \vdots \\ 1 \\ 1/2 \end{bmatrix} \in \mathbb{R}^n.$$

Note that in this setting to have an appropriate scaling we should take

$$\rho = \frac{\hat{\rho}}{h}, \quad \text{for a } \hat{\rho} > 0,$$

that is ρ should scale inversely with h. The stop criterion should be performed in a suitable norm. For instance one may choose a stopping criteria based on the H^1 norm of the solution increment, which may be approximated by with

$$\text{Stop if } (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})^T(A + hI)(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) \le \epsilon^2$$

where I is the identity matrix.

Exam

The objective of the exam is to build a library for constrained minimization via the method of multipliers, focusing on the quadratic setting.

Please read all points before starting, and provide a working Makefile.
The library should be built with the following components:

1. (4 points) A class for the general method of multipliers
   (a) (4 points) It should delegate the actual implementation of the two steps (solve and update) to an instance of a user provided polymorphic class (see next point).
2. (14 points) A class hierarchy that provides the policy for solve and update.
   (a) (3 points) A base class that exposes as virtual methods the ones necessary in the class developed in the point above. It serves only to provide the common interface.
   (b) (4 points) A derived class that specializes the methods for the quadratic case and linear constraints. You should use the Eigen library for linear algebra operations.
   (c) (4 points) The specialized class should enable to use different methods for solving the linear problem for the calculation of $\mathbf{x}$ among the ones provided by Eigen (for instance by passing the Eigen solver via a template parameter), and also do thing efficiently, avoiding recomputing the factorization of the matrix for the different right-hand side corresponding to different iterations (look at this Eigen tutorial on how to use linear system solvers for dense matrices.)
   (d) (3 points) Write a main to test the code with a very simple problem where

$$A = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 5 \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad B = [-1, 1], \quad \mathbf{g} = [0],$$

   whose exact solution (with six significant digits) is $x_1 = x_2 = 0.189189$. Use null initial condition, $\rho = 1$, $\epsilon = 10^{-6}$ and an error indicator based on the maximum difference between two iterates of $\mathbf{x}$. Write in the file `sol.txt` the number of iterations taken by the method to reach convergence, and the value of the residual $\|A\mathbf{x}^* - \mathbf{d}\|$, where $\mathbf{x}^*$ is the found solution.
3. (12 points) Tackle the differential problem (6) with the method of multipliers:
   (a) (4 points) Do a second specialization of the policy class hierarchy that tackles problem (6) via the shown method of multipliers,
      1. It should have a constructor (or setters) for all the quantity needed (the constant f, g_0 and g_1), all aggregated in a **struct**,
      2. It should have a setter for the chosen number of nodes for the discretization, that also sets the various matrices. Note that you do not really need to build B explicitly if you wish *and that you could use here sparse Eigen matrices* (2 bonus points if you do).
   (b) (3 points) Run a test case where $f = 1$, $g_0 = 0$, $g_1 = 1$, using $\hat{\rho} = 0.5$, $\epsilon = 10^{-5}$, for $h = 16$, 32 and 64 nodes. Use the H1 norm of the increment as stopping criteria using the formula given above[2].
   (c) (3 points) Knowing that $u_{ex} = \frac{1}{2}(3x - x^2)$ is the exact solution, write a function to estimate the actual error in the $L^2(0,1)$ norm, as follows
      - Create a vector of N equispaced points $(y_0, \dots, y_{N-1})$ in the interval $[0, 1]$, where N is at least $10n$ (to have enough sampling points).
      - Write a small function (or a lambda), that takes the computed solution $\mathbf{x}$ and a generic point $y \in [0, 1]$ and returns the value provided by the piecewise linear interpolant, Π_1 (think about it, it is simple, you need only to find in which interval y falls and then in the single interval you may use std::lerp).

[2] In `Eigen` the identity the identity matrix is created with Eigen::MatrixXd::Identity(n,n).

- Compute the error as $e\sqrt{h\sum_{i=1}^{n}(u_{ex}(y_i) - \Pi_1(y_1))^2}$ and print the value on the screen.

(d) (2 point bonus points) Use gnuplot-iostream to plot the solution directly from the main program.

Notes: Marks are only indicative. Depending on the quality of the code I may decide to give extra points on top of the maximum. For students of the 10 credit version the mark is capped at 25 by multiplying the outcome by 5/6. Points taken with the challenges will be added.

As a final note I want to recall that the method of multipliers is not the best method to solve the differential problem (6), but it becomes more interesting for more general or complex constraints.