PCG-ILU

Instructor: Seongjai Kim*

1 Introduction

Let $\Omega = (a_x, b_x) \times (a_y, b_y) \subset \mathbb{R}^2$ with its boundary $\Gamma = \partial\Omega$. The problem we will solve in the project is

$$\begin{aligned} -(au_x)_x - (au_y)_y + cu &= f(x,y), \quad (x,y) \in \Omega, \\ au_\nu + \beta u &= g(x,y), \quad (x,y) \in \Gamma, \end{aligned} \tag{1.1}$$

where the subscripts x and y respectively denote the x-derivative and y-derivative, the coefficients a, c, and β is functions defined on the domain (they are not constants in practice), and ν is the outer normal derivative from Ω defined by

$$\begin{aligned} -au_x + \beta u &= g, \quad \text{if } (x,y) = (a_x, y), \\ au_x + \beta u &= g, \quad \text{if } (x,y) = (b_x, y), \\ -au_y + \beta u &= g, \quad \text{if } (x,y) = (x, a_y), \\ au_y + \beta u &= g, \quad \text{if } (x,y) = (x, b_y). \end{aligned} \tag{1.2}$$

We will apply the centered **finite difference method** (FDM) to construct the algebraic system

$$A\mathbf{u} = \mathbf{b} \tag{1.3}$$

which has the truncation error $\mathcal{O}(h^2)$, where h is the grid size.

The problem (1.1) is being widely applied to simulate physical quantities in various sciences and industries, and there are still many interesting numerical aspects to be considered and solved. The problems is considered in e.g. contaminant transport in underground water, secondary/tertiary oil production, high speed semiconductor simulation, and electromagnetic wave applications.

The problem is known to be relatively easy to solve for certain types of the problem coefficients: a, c, and β. However, if $c = \beta = 0$, then the problem admits infinitely many solutions. In fact, if u is a solution of (1.1), so is $u + \gamma$ for all $\gamma \in \mathbb{R}$. Which implies that the corresponding algebraic system (1.3) is singular, and therefore there are infinitely many numerical solutions.

*Department of Mathematics, University of Kentucky, Lexington, KY 40506-0027 USA Email: skim@ms.uky.edu

Through the project, you will implement the **conjugate gradient** (CG) method to iteratively solve the algebraic system of more or less million unknowns. CG is known to converge with the iteration number being proportional to $\mathcal{O}\left(\sqrt{\kappa(A)}\right)$, where $\kappa(A)$ is the condition number of the matrix A. Since the condition number of A can become arbitrarily large (depending on the problem coefficients), the CG method may converge after a very large number of iterations. To speed it up, we need to precondition the matrix by a matrix M such that $\kappa(M^{-1}A) \ll \kappa(A)$. The preconditioned system

$$M^{-1}A\mathbf{u} = M^{-1}\mathbf{b} \tag{1.4}$$

is to be solved by the CG method. The technique is called **preconditioned conjugate gradient** (PCG) method and it is a core part of the modern scientific computation. Now the question is: how can we choose the preconditioner M? First of all, it should approximate the original matrix A well. Secondly, it should not be computationally expensive to invert. We will try the most beloved preconditioners: **incomplete LU** (ILU) factorizations and **incomplete Cholesky factorizations (ICF)**.

2 Krylov subspace methods

2.1 Introduction

Let A be a symmetric positive definite $n \times n$ matrix. We consider iterative methods for solving

$$\boldsymbol{Ax = b}, \quad \boldsymbol{A} \in \mathbb{R}^{\boldsymbol{n\times n}}, \ \boldsymbol{b} \in \mathbb{R}^{\boldsymbol{n}}. \tag{2.1}$$

Given an initial guess $x_0 \in \mathbb{R}^n$, find successive approximations $x_k \in \mathbb{R}^n$ of the form

$$\boldsymbol{x_{k+1} = x_k + \alpha_k p_k}, \quad \boldsymbol{k = 0, 1, \cdots}, \tag{2.2}$$

where p_k is the **search direction** and $\alpha_k > 0$ is the **step length**. Different methods differ in the choice of the search direction and the step length. In this note, we consider the gradient method (also known as steepest descent method, or Richardson's method), the conjugate gradient (CG) method, and preconditioned CG method.

Note that (2.1) admits a unique solution $x \in \mathbb{R}^n$ equivalently characterized by

$$\min_{\eta\in\mathbb{R}^n} f(\eta), \quad \text{where } f(\eta) = \frac{1}{2}\eta \cdot A\eta - b \cdot \eta, \tag{2.3}$$

where $a \cdot b = a^T b$.

2.2 Steepest descent method

We denote the **gradient** and **Hessian** of f by f' and f'', respectively. It is easy to see that

$$f'(\eta) = A\eta - b, \quad f''(\eta) = A.$$

Given x_{k+1} as in (2.2), we have by Taylor's formula

$$f(x_{k+1}) = f(x_k + \alpha_k p_k) = f(x_k) + \alpha_k f'(x_k) \cdot p_k + \frac{\alpha_k^2}{2} p_k \cdot f''(\xi) p_k, \text{ for some } \xi.$$

Since the element of f'' is bounded (As a matter of fact, we assume it!),

$$f(x_{k+1}) = f(x_k) + \alpha_k f'(x_k) \cdot p_k + \mathcal{O}(\alpha_k^2), \quad \text{as } \alpha_k \to 0.$$

The goal is to find p_k and α_k such that $f(x_{k+1}) < f(x_k)$, which can be achieved if

$$f'(x_k) \cdot p_k < 0 \tag{2.4}$$

and α_k is sufficiently small. Due to this motivation, p_k is called the **descent direction** of f if (2.4) holds. In particular, (2.4) holds if we choose

$$\boldsymbol{p_k = -f'(x_k) = b - Ax_k =: r_k} \tag{2.5}$$

and if $f'(x_k) \neq 0$.

Now, let us choose the step length: we may determine α_k such that

$$f(x_k + \alpha_k p_k) = \min_\alpha f(x_k + \alpha p_k),$$

in which case α_k is said to be **optimal**. If α_k is optimal, then

$$\begin{aligned} 0 &= \left.\frac{\mathrm{d}}{\mathrm{d}\alpha} f(x_k + \alpha p_k)\right|_{\alpha=\alpha_k} = f'(x_k + \alpha_k p_k) \cdot p_k \\ &= (A(x_k + \alpha_k p_k) - b) \cdot p_k = (Ax_k - b) \cdot p_k + \alpha_k p_k \cdot Ap_k. \end{aligned}$$

So,

$$\boldsymbol{\alpha_k = \frac{r_k \cdot p_k}{p_k \cdot Ap_k}.} \tag{2.6}$$

The steepest descent method consists of the search direction $p_k = r_k$ and the step length α_k in (2.6). For the method, the following is known

$$\| x - x_k \|_2 \leq \left(1 - \frac{1}{\kappa(A)}\right)^k \| x - x_0 \|_2;$$

the number of iterations required to reduce the error by a factor of ε is in the order of the condition number of A:

$$\boldsymbol{k \;\geq\; \kappa(A) \log \frac{1}{\varepsilon}.}$$

2.3 Conjugate gradient method

In this method the search directions p_k are conjugate, i.e.,

$$p_i \cdot Ap_j = 0, \quad i \neq j,$$

and the step length α_k is chosen to be optimal.

CG Algorithm, V.1

select $\boldsymbol{x_0}$, $\boldsymbol{\varepsilon}$;
$\boldsymbol{r_0 = b - Ax_0}$, $\boldsymbol{p_0 = r_0}$;
do $\boldsymbol{k = 0, 1, \cdots}$
$\quad \boldsymbol{\alpha_k = r_k \cdot p_k / p_k \cdot Ap_k}$; (CG1)
$\quad \boldsymbol{x_{k+1} = x_k + \alpha_k p_k}$; (CG2)
$\quad \boldsymbol{r_{k+1} = r_k - \alpha_k Ap_k}$; (CG3)
$\quad$ if $\|\boldsymbol{r_{k+1}}\|_2 > \boldsymbol{\varepsilon}$, continue;
$\quad \boldsymbol{\beta_k = -r_{k+1} \cdot Ap_k / p_k \cdot Ap_k}$; (CG4)
$\quad \boldsymbol{p_{k+1} = r_{k+1} + \beta_k p_k}$; (CG5)
enddo

Notes for the CG Algorithm:

- α_k in (CG1) is designed such that $r_{k+1} \cdot p_k = 0$. You may easily verify it using r_{k+1} in (CG3).
- $r_k = b - Ax_k$, by definition. So,

  $$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha_k p_k) = b - Ax_k - \alpha_k Ap_k = r_k - \alpha_k Ap_k,$$

  which is (CG3).
- β_k in (CG4) is determined such that $p_{k+1} \cdot Ap_k = 0$. Verify it using p_{k+1} in (CG5).
- The CG method finds the iterate $x_k \in x_0 + \text{span}\{r_0, Ar_0, \cdots, A^{k-1}r_0\}$ so that $(x - x_k) \cdot A(x - x_k)$ is minimized.

Theorem 2.1. *For* $m = 0, 1, \cdots,$

$$\text{span}\{p_0, \cdots, p_m\} = \text{span}\{r_0, \cdots, r_m\} = \text{span}\{r_0, Ar_0, \cdots, A^m r_0\}.$$

Theorem 2.2. *The search directions and the residuals satisfy*

$$p_i \cdot Ap_j = 0; \quad r_i \cdot r_j = 0, \quad i \neq j.$$

Theorem 2.3. *For some $m \le n$, we have $Ax_m = b$ and*

$$\| x - x_k \|_A \le 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \| x - x_0 \|_A.$$

So the required iteration number to reduce the error by a factor of ε is

$$k \ge \frac{1}{2}\sqrt{\kappa(A)} \log\frac{2}{\varepsilon}.$$

Proofs of the above theorems are not so difficult; however, I think it is out of the scope of the class. We may further simplify the algorithm by using the properties and identities included in the method. The following is the most popular form of the CG method:

CG Algorithm, V.2

select $\boldsymbol{x_0}$, $\boldsymbol{\varepsilon}$;
$\boldsymbol{r_0 = b - Ax_0}$, $\boldsymbol{p_0 = r_0}$, compute $\boldsymbol{\rho_0 = r_0 \cdot r_0}$;
do $\boldsymbol{k = 0, 1, \cdots}$
    $\boldsymbol{\alpha_k = \rho_k / p_k \cdot A p_k}$;
    $\boldsymbol{x_{k+1} = x_k + \alpha_k p_k}$;
    $\boldsymbol{r_{k+1} = r_k - \alpha_k A p_k}$;
    if $\boldsymbol{\| r_{k+1} \|_2 > \varepsilon}$, continue;
    $\boldsymbol{\rho_{k+1} = r_{k+1} \cdot r_{k+1}}$;
    $\boldsymbol{\beta_k = \rho_{k+1} / \rho_k}$;
    $\boldsymbol{p_{k+1} = r_{k+1} + \beta_k p_k}$;
enddo

As shown in Theorem 2.3, the condition number of A is the critical point for the convergence of the CG method. If we can find a matrix M such that

$$M \approx A$$

and it is easy to invert, we may try to apply the CG algorithm to the following system

$$M^{-1}Ax = M^{-1}b.$$

Since $\kappa(M^{-1}A) \ll \kappa(A)$ (hopefully, $\kappa(M^{-1}A) \approx 1$), the CG algorithm will converge much faster. In practice, we do not have to multiply M^{-1} to the original algebraic system. The idea can be implemented equivalently as follows:

Preconditioned CG Algorithm

select $\boldsymbol{x_0}$, $\boldsymbol{\varepsilon}$;
$\boldsymbol{r_0 = b - Ax_0}$, $\boldsymbol{Mz_0 = r_0}$;
$\boldsymbol{p_0 = z_0}$, compute $\boldsymbol{\rho_0 = z_0^* r_0}$;
do $\boldsymbol{k = 0, 1, \cdots}$
    $\boldsymbol{\alpha_k = \rho_k / p_k^* A p_k}$;
    $\boldsymbol{x_{k+1} = x_k + \alpha_k p_k}$;
    $\boldsymbol{r_{k+1} = r_k - \alpha_k A p_k}$;
    if $\| \boldsymbol{r_{k+1}} \|_2 > \boldsymbol{\varepsilon}$, continue;
    $\boldsymbol{M z_{k+1} = r_{k+1}}$;
    $\boldsymbol{\rho_{k+1} = z_{k+1}^* r_{k+1}}$;
    $\boldsymbol{\beta_k = \rho_{k+1} / \rho_k}$;
    $\boldsymbol{p_{k+1} = z_{k+1} + \beta_k p_k}$;
enddo

As the choice of preconditioners M, **incomplete LU (ILU)** factorizations and **incomplete Cholesky factorizations (ICF)** are very popular in modern scientific computation. I will provide stabilized versions of the factorizations, as a part of the basis code package.