Lecture 10b: iterative and direct linear solvers

MATH0504: Mathématiques appliquées

X. Adriaens, J. Dular

Université de Liège

November 2018

Table of contents

The goal of this lecture is to investigate numerically different ways of solving a linear system

$$Ax = b$$

where

- $A \in M^{n\times n}(\mathbb{R})$ is a known matrix,
- $x \in \mathbb{R}^n$ is an unknown vector
- and $b \in \mathbb{R}^n$ is a known vector.

Direct methods

LU decomposition (without pivoting) I

Consider

$$Ax = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} x = b$$

Gaussian elimination consists in substituting (in A and b)

$$l_2 \to l_2 - 2l_1$$
$$l_3 \to l_3 - 4l_1$$

In matrix notation this operation can be written

$$L_1 A = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & 3 & 5 \end{pmatrix}$$

LU decomposition (without pivoting) II

Repeating this operation gives

$$L_2 L_1 A = \begin{pmatrix} 1 & & \\ & 1 & \\ & -3 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & 3 & 5 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & & 2 \end{pmatrix} = U$$

which is the LU decomposition provided that $L^{-1} = L_2 L_1$ is lower triangular.

Fortunately it is and there is a simple link between the components of L and the components of L_1 and L_2 (*cfr* MATH0006-3 Introduction to numerical analysis) which yields

$$A = LU \quad \text{with} \quad L = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & & 2 \end{pmatrix}$$

LU decomposition (without pivoting) III

Now that the decomposition $A = LU$ is known, the solution to $Ax = LUx = b$ is computed in two steps

1. Determine the intermediate vector y through $Ly = b$.
   Because L is lower triangular, the system can be solved by determining successively the $[y]_i$ starting from $[y]_1$ (downward substitution).

2. Determine the solution x through $Ux = y$.
   Because U is upper triangular, the system can be solved by determining successively the $[x]_i$ starting from $[x]_n$ (upward substitution).

The `Matlab` command `A\b` uses a direct method which is a variant of LU decomposition in some cases.

Full matrix inversion

One can wonder why computing the LU decomposition instead of simply computing A^{-1}.

Actually, one of the most efficient way to compute A^{-1} is

1. Compute the decomposition $A = LU$
2. Solve $LUx_j = e_j$ with $e_j = \left(\underbrace{0 \dots 0}_{j-1} 1 \underbrace{0 \dots 0}_{n-j}\right)^T \ \forall j \in [1, n]$.
3. Construct

$$A^{-1} = \left(x_1 \middle| x_2 \middle| \dots \middle| x_{n-1} \middle| x_n \right)$$

because one has $AA^{-1} = LUA^{-1} = I$.

This algorithm is more expensive than the previous one as step 2 involves $2n$ more triangular system solutions. Moreover A^{-1} can be dense and the product $A^{-1}b$ is then expensive.

Practice

- Open `methodComparison.m`
- Comment the part `ITERATIVE TECHNIQUES`
- Compare

Full inversion:

```
x_inv = inv(A)*b;
```

LU **decomposition:**

```
[L, U] = lu(A);
y = L\b;
x_lu = U\y;
```

`Matlab` **direct solver:**

```
x_Ab = A\b;
```

for

1. a random dense matrix A
2. a random SDP dense matrix A
3. a sparse (5-diagonal) matrix A

by running `methodComparison.m`

Iterative methods

Iterative versus direct method

- An iterative method is an algorithm that computes approximate solutions x_k successively until the norm of the residuals $\|r_k\| = \|Ax_k - b\|$ is small enough. Consequently, an iterative method does not always give the exact solution, even if there is no numerical error.

- At the opposite, if there is no numerical errors, a direct method gives the exact solution (in $\mathcal{O}(n^3)$ operations).

- However an approximate solution is often sufficient if it can be obtained quickly (*i.e* in particular in less than $\mathcal{O}(n^3)$ operations).

Stationary iterative methods

Update equation

Stationary method have update equations of the form

$$K\left(x_{k+1}-x_k\right)=b-Ax_k\left(=r_k\right).$$

They are called stationnary because K is the same for all iterations.

The matrix K is expected to

1. be close to A. Indeed, taking $K=A$ yields $Ax_{k+1}=b$.
2. be such that $K\Delta x_{k+1}=r_k$ can be solved easily.

The most famous choices are

$$\begin{aligned} K &= \text{diag}\left(A\right) && \text{(Jacobi)} \\ K &= \text{diag}\left(A\right)+\text{lower}\left(A\right) && \text{(Gauss-Seidel)} \end{aligned}$$

The solution to $K\Delta x_{k+1}=r_k$ then only involves explicit operations for Jacobi and an upward substitution for Gauss-Seidel.

Practice

- Comment part `DIRECT TECHNIQUES` and uncomment `ITERATIVE TECHNIQUES`
- Create a function `jacobi.m`
- Create a function `gaussSeidel.m`
- Compare them for
  1. a random dense matrix A
  2. a random diagonally dominant dense matrix A
  3. a sparse (5-diagonal) A
  4. a sparse (5-diagonal) diagonally dominant matrix A

Practice: Jacobi

```
function [x, iteration] = jacobi(A, b, relTol, maxIteration)
    x = zeros(length(b), 1);
    r = b-A*x; r0 = norm(r);
    Kinv = diag(1./diag(A));
    iteration = 0;
    rnorm = norm(r);
    while rnorm/r0 > relTol && iteration < maxIteration
        x = x + Kinv * r;
        r = b-A*x;
        rnorm = norm(r);
        iteration = iteration + 1;
    end
end
```

Practice: Gauss-Seidel

```
function [x, iteration] = gaussSeidel(A, b, relTol, ...
                                      maxIteration)
    x = zeros(length(b), 1);
    r = b-A*x; r0 = norm(r);
    K = tril(A);
    iteration = 0;
    rnorm = norm(r);
    while rnorm/r0 > relTol && iteration < maxIteration
        dx = K\r;
        x = x + dx;
        r = b-A*x;
        rnorm = norm(r);
        iteration = iteration + 1;
    end
end
```

Krylov subspace methods

Krylov subspace

The common point of any Krylov subspace method is that, at the kth iterations, the approximate solution x_k belongs to Krylov subspace of dimensions k generated by A and b, *i.e*

$$x_k \in \mathcal{K}_k(A, b) = \text{span}\left(b, Ab, A^2b, \ldots, A^{k-2}b, A^{k-1}b\right).$$

which means that x_k is a linear combination of $A^m b$ $(m \in [0, k-1])$, *i.e*

$$x_k = \sum_{m=0}^{k-1} \alpha_m A^m b.$$

(Motivation)

A motivation for searching solution in $\mathcal{K}_k$ is the Cayley-Hamilton theorem.

Cayley-Hamilton theorem: Let $A \in M^{n\times n}(\mathbb{R})$ be a real square matrix and $p(\lambda) = \det(A - \lambda I)$ be its characteristic polynomial. A satisfies

$$p(A) = \sum_{k=0}^{n} c_k A^k = O \quad (\text{with } c_0 \propto \det(A)) .$$

Multiplying both sides by x, this theorem gives

$$c_0 x + \sum_{k=1}^{n} c_k A^k x = O$$

then using $Ax = b$

$$x = -\frac{1}{c_0} \sum_{k=1}^{n} c_k A^{k-1} b \quad \Rightarrow \quad x \in \mathcal{K}_n(A, b)$$

Conjugate Gradient (CG)

Requirements:

- Symmetric (S): $A^T = A$
- Positive definite (PD): $x^T A x > 0 \; \forall x \neq 0$

Goal:

Find $x_k \in \mathcal{K}_k(A, b)$ such that $\|e_k\|_A = \sqrt{e_k^T A e_k}$ is minimum where e_k is the relative error $x - x_k$.

Algorithm: From $x_0 = 0$, $r_0 = b$, $p_0 = r_0$, successively compute

$$x_k = x_{k-1} + \alpha_k p_{k-1} \qquad \text{with } \alpha_k = \frac{\|r_{k-1}\|^2}{\|p_{k-1}\|_A^2}$$

$$r_k \triangleq b - A x_k = r_{k-1} - \alpha_k A p_{k-1}$$

$$p_k = r_k + \beta_k p_{k-1} \qquad \text{with } \beta_k = \frac{\|r_k\|^2}{\|r_{k-1}\|^2}$$

Convergence: The error decreases as

$$\frac{\|e_k\|_A}{\|e_0\|} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^n$$

where κ is the condition number.

Practice

- Uncomment part on conjugate gradient
- Create `cg.m`
- Compare the behaviours for
  1. a dense SDP matrix A
  2. a sparse SDP matrix A
- In both cases, compute κ with the function `cond(A)` and explain the behaviour.

Practice: Conjugate Gradient

```
function [x, iteration] = cg(A, b, relTol, maxIteration)
    x = zeros(length(b), 1);
    r = b-A*x; rnormSqu = r'*r;
    r0Squ = rnormSqu;
    iteration = 0; p=r;
    while rnormSqu/r0Squ > relTol^2 ...
                 && iteration < maxIteration
        Ap = A * p;
        alpha = rnormSqu/(p' * Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        rnormSquOld = rnormSqu; rnormSqu = r'*r;
        beta = rnormSqu/rnormSquOld;
        p = r + beta * p;
        iteration = iteration + 1;
    end
end
```

Overall comparison

Practice

- Uncomment the good methods only (LU, `A\b`, CG)
- Compare direct and iterative methods for a SDP, diagonally dominant dense matrix
- Explain the results by looking at κ