Low-Rank Approximation

Lecture 5 – PDEs and matrix equation

Leonardo Robol, University of Pisa, Italy

Cagliari, 23–27 Sep 2019

Disclaimer

Our aim today is to try to use matrix equations for solving PDEs on a rectangular domain.

- The focus will be on the use of matrix equation, and not really on the differential problems.
- We will try to consider the simplest possible discretization.
- Most considerations work in a more general setting (e.g., we will mostly do finite differences, but there is little change if you like finite elements more).

Model problem

General idea: we consider a model problem, and we will enrich it with new features one step at a time. We are concerned with $2D$ problems, but to fix the notation we start with:

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + f(x) = 0, & x \in [-1, 1] \\ u(-1) = u(1) = 0 \end{cases}$$

- Model a diffusion process given a source $f(x, y)$.
- We consider a discretization of $[-1, 1]$ into $n + 2$ points.
- Can be discretized into finite differences by using:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} + \mathcal{O}(h^2).$$

Matrix discretization

The discrete differential operator is only well-defined in the inner-part of the interval:

$$\begin{bmatrix} \frac{\partial^2 u(x_1)}{\partial x^2} \\ \vdots \\ \frac{\partial^2 u(x_n)}{\partial x^2} \end{bmatrix} \approx \underbrace{\frac{1}{h^2} \begin{bmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{bmatrix}}_{A} \begin{bmatrix} u(x_0) \\ u(x_1) \\ \vdots \\ u(x_n) \\ u(x_{n+1}) \end{bmatrix}$$

- Since we know the value of $u(x_0)$ amd $u(x_{n+1})$, we can transform this into a square linear system by removing the first and last column of the above matrix.
- For the steady state, one has then to solve:

$$A\hat{u} = -\hat{f} - u(x_0)Ae_1 - u(x_{n+1})Ae_{n+2},$$

where $\hat{f}_j = f(x_j)$, for $j = 1, \ldots, n$.

See: `example_pde_1d.m`

Time-dependent equation

Instead of computing the steady-state, we might be interested in tracing the solution on a time interval $[0, T]$, given the PDE

$$\begin{cases} \frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} + f(x,t), & x \in [-1,1] \\ u(-1,0) = u(1,0) = 0 \end{cases}$$

- We may discretize in time with implicit Euler, to get unconditional stability, which yields:

$$\frac{u_{t+1} - u_t}{\Delta t} = A u_{t+1} + f_{t+1} + u(x_0) A e_1 + u(x_{n+1}) A e_{n+2}$$

- Re-arranging the terms:

$$(I - \Delta t A) u_{t+1} = u_t + f_{t+1} + u(x_0) A e_1 + u(x_{n+1}) A e_{n+2}$$

See: `example_pde_1d_time.m`

Going 2D

We can now replace $[-1,1]$ with $[-1,1]^2$. If we do so, we will need to replace the equation with:

$$\begin{cases} \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} + f(x,y) = 0. \\ u(x,y) \equiv 0 \text{ on } \partial[-1,1]^2. \end{cases}$$

- Clearly, $[-1,1]$ is chosen just to make computations easier.
- Now, if we discretize on an $(n+2) \times (n+2)$ grid, we have n^2 unknowns!
- If we put all the unknownn in a vector, we have:

$$\frac{\partial^2 u}{\partial x^2} \approx A \otimes I, \qquad \frac{\partial^2 u}{\partial y^2} \approx I \otimes A.$$

- $n = 512$ gives a linear system of size (about) 262000×262000!
- In principle we could repeat all the steps.

Matrix equations

If we arrange the unknown u in matrix form:

$$U = \begin{bmatrix} u(x_0, y_{n+1}) & \dots & u(x_{n+1}, y_{n+1}) \\ \vdots & & \vdots \\ u(x_0, y_0) & \dots & u(x_{n+1}, y_0) \end{bmatrix},$$

then the second derivative in x and y can be rephrased much more naturally as:

$$\frac{\partial^2}{\partial x^2} \approx U \mapsto UA, \qquad \frac{\partial^2}{\partial y^2} \approx U \mapsto AU.$$

Therefore, the linear system can be solved by solving:

$$AU + UA + F = 0.$$

Matrix equations

Note that:

- If $f(x, y)$ is smooth, then we expect its sampling F to be of low-numerical rank.
- Its representation can be retrieved using, for instance, ACA.
- Once F is of low-rank, we can exploit low-rank matrix equation solvers, since A and $-A$ have disjoint spectra (being A posdef). Notice that this is valid for any elliptic operator.

Matrix equations in practice

The `hm-toolbox` contains the following matrix equation solvers:

- `ek_lyap` based on the Extended Krylov subspaces, works for most cases with positive definite coefficients and, in general, with positive symmetric part.
- `rk_lyap` based on the general rational Krylov; needs the poles by the user.

They both have a similar syntax of MATLAB's `lyap`, but they require a low-rank RHS in factored form (UV^T). Example:

```
[Xu, Xv] = ek_lyap(A, B, U, V, maxit, tol, debug);
X = Xu * Xv';
```

solves $AX + XB + UV^T = 0$. The flags `maxit` gives the maximum number of iterations (can be `inf`), `tol` the required relative tolerance for the residual, and `debug` is a boolean variable.

Superfast Toeplitz solver

We can use (hierarchically) low-rank matrices to solve Toeplitz systems in $\mathcal{O}(n \log^2 n)$ time. Remember that the displacement equation

$$ZT - TZ = UV^T$$

gives little information on the low-rank structure of T. Therefore, we move our attention to $Z_1 T - TZ_{-1} = F$ where

$$Z_x := \begin{bmatrix} & & & x \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}$$

- Now both Z_1 and Z_{-1} are unitary and therefore normal;
- They have disjoint spectra (but not so much).
- F has only the first row and last column different from zero (i.e., it is rank 2).

Changing the displacement relation

We can modify the displacement relation to transform the Toeplitz matrix in another one that has a particular structure.

Let Ω_n denote the matrix of the Fourier transform, scaled to be unitary. Then, since Z_1 is circulant, $\Omega_n Z_1 \Omega_n^*$ is diagonal, and in particular it has the n-th roots of the unity on the diagonal. We call such matrix D_1.

In a similar fashion, if we define $D_0 = \mathrm{diag}(1, \omega_{2n}, \ldots, \omega_{2n}^{n-1})$, where ω_{2n} is $e^{\frac{i\pi}{n}}$, we have the relation

$$\Omega_n D_0 Z_{-1} D_0^* \Omega_n^* = D_{-1},$$

where $D_{-1} = \omega_{2n} D_1$.

Changing the displacement relation (continued)

Multiplying the displacement relation from the left by Ω_n and from the right by $D_0^*\Omega_n^*$ yields the new relation

$$D_1 C - C D_{-1} = G F^T$$

where G, F can be defined as $G = \Omega_n U$ and $F = \overline{\Omega_n} D_0^* V$, and $C = \Omega_n T D_0^* \Omega_n^*$. The previous relation tells us that the matrix C has a *Cauchy-like* structure; since the coefficients of the displacement equation are diagonal, we can explicitly write its entries as:

$$C_{ij} = \frac{G_i H_j^T}{\omega_{2n}^{2(i-1)} - \omega_{2n}^{(2j-1)}}$$

This Cauchy-like matrix is not low-rank: the two vectors are not well-separated; however, its off-diagonal blocks are! (Why)? Therefore, it has an HSS or HODLR structure.

Constructing the HODLR representation

If you remember on Monday, we discussed matrices with hierarchical low-rank structure.

Now, we know how to approximate the off-diagonal low-rank blocks: for instance, we can use ACA. This is already implemented in `hm-toolbox`:

```
C = hodlr('handle', @(i,j) cauchy_like(i,j));
```

where `cauchy_like` is a function that computes the element in position (i, j). If we have a fast-matrix product at our disposal, we can use more efficient constructors. Indeed, recall that

$$T = \Omega_n^* C \Omega_n D_0, \qquad C = \Omega_n T D_0^* \Omega_n^*.$$

We can use this relation both to recast linear systems with T into linear systems with C, and matrix vector products with C into matrix-vector products with T.

If you compute the complexity, you will get $\mathcal{O}(n \log^2 n)$ (this uses that the Cauchy-like has rank $\mathcal{O}(\log n)$).

Exercise session

You can try to:

- Solve some PDEs using matrix equations (see the assignment online);
- Construct the fast Toeplitz solver (full disclosure: it is also provided in the toolbox as the function `toeplitz_solve`).