Dirichlet feedback control of 2-d heat equation

Praveen Chandrashekar
TIFR Centre for Applicable Mathematics
Bangalore – 560065

1 Heat equation

The eigenfunctions of Δ on the unit square $\Omega = [0,1]\times[0,1]$ are of the form $\sin(n\pi x)\sin(m\pi y)$ and the corresponding eigenvectors are $-(n^2+m^2)\pi^2$ for $n, m = 1, 2, 3, \ldots$. The first few eigenvalues are given by

$$-2\pi^2, -5\pi^2, -5\pi^2, -8\pi^2, -10\pi^2, -10\pi^2, \ldots$$

Choose an $s > 2\pi^2$ and consider

$$\begin{aligned} \frac{\partial\theta}{\partial t} &= \Delta\theta + s\theta, \qquad \text{in } \Omega \\ \theta(x,0) &= 0, \qquad x \in \Omega \\ \theta(x,t) &= 0, \qquad x \in \partial\Omega \end{aligned}$$

This has a stationary solution which is just $\theta = 0$. To study the stability of this stationary solution to small perturbations, we perturb the initial condition and look at its evolution given by

$$\begin{aligned} \frac{\partial\theta}{\partial t} &= \Delta\theta + s\theta, \qquad \text{in } \Omega \\ \theta(x,0) &= \epsilon\theta_0(x), \qquad x \in \Omega \\ \theta(x,t) &= 0, \qquad x \in \partial\Omega \end{aligned}$$

where $\epsilon \ll 1$. The perturbations grow since (???)

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega |\theta|^2 \mathrm{d}x = \int_\Omega (-\|\nabla\theta\|^2 + s|\theta|^2)\mathrm{d}x > 0$$

We can introduce a Dirichlet control to stabilize this and consider

$$\begin{aligned} \frac{\partial\theta}{\partial t} &= \Delta\theta + s\theta, \qquad \text{in } \Omega \\ \theta(x,0) &= \epsilon\theta_0(x), \qquad x \in \Omega \\ \theta(x,t) &= u(x,t), \qquad x \in \partial\Omega \end{aligned}$$

We want to compute u in terms of θ by a feedback law.

2 Weak formulation

Multiply by a test function $\phi \in H^1(\Omega)$ and integrate by parts

$$\int_\Omega \frac{\partial \theta}{\partial t}\phi \mathrm{d}x = \int_\Omega [-\nabla\theta \cdot \nabla\phi + s\theta\phi]\mathrm{d}x + \int_{\partial\Omega} \frac{\partial\theta}{\partial n}\phi \mathrm{d}s$$

Introduce the flux as a new unknown $\sigma = \frac{\partial\theta}{\partial n}$ that we have to also determine as part of the solution and satisfy the boundary conditions in a weak manner. This leads to the problem: find $\theta \in H^1(\Omega)$, $\sigma \in L^2(\partial\Omega)$ such that

$$\int_\Omega \frac{\partial \theta}{\partial t}\phi \mathrm{d}x = \int_\Omega [-\nabla\theta \cdot \nabla\phi + s\theta\phi]\mathrm{d}x + \int_{\partial\Omega} \sigma\phi \mathrm{d}s, \qquad \forall \phi \in H^1(\Omega) \tag{1}$$

$$\int_{\partial\Omega} \theta\psi \mathrm{d}s = \int_{\partial\Omega} u\psi \mathrm{d}s, \qquad \forall \psi \in L^2(\partial\Omega) \tag{2}$$

3 Finite element method

Let $\mathcal{T}_h$ be a triangulation of Ω and we define the space of piecewise polynomials of degree k

$$V_h = \{\phi \in C^0(\Omega) : \phi|_K \in \mathbb{P}_k, \forall K \in \mathcal{T}_h\}$$

Let $\mathcal{E}_b$ denote the boundary edges of $\mathcal{T}_h$. We also define the space of piecewise polynomials on the boundary

$$W_h = \{\phi \in C^0(\partial\Omega) : \phi|_e \in \mathbb{P}_k, \forall e \in \mathcal{E}_b\}$$

We will use nodal Lagrange basis functions for V_h which we denote by $\{\phi_j, 0 \le j \le N-1\}$. The restriction of these basis function to the boundary gives the basis functions for W_h. Define $\mathcal{I}_b$ to be set of dofs supported on the boundary, $\mathcal{I}_f$ to be the other dofs and $\mathcal{I} = \mathcal{I}_b \cup \mathcal{I}_f = \{0, 1, 2, \ldots, N-1\}$. Then

$$W_h = \mathrm{span}\{\phi_j|_{\partial\Omega}, j \in \mathcal{I}_b\}$$

With this notation, we can write the finite element solution as

$$\theta_h = \sum_{j\in\mathcal{I}} \theta_j\phi_j, \qquad \sigma_h = \sum_{j\in\mathcal{I}_b} \sigma_j\phi_j$$

We also approximate the boundary condition u by u_h

$$u_h = \sum_{j\in\mathcal{I}_b} u_j\phi_j$$

Substituting these into the weak formulation leads to the following set of equations

$$\begin{aligned} M\frac{\mathrm{d}\theta}{\mathrm{d}t} &= A_1\theta + N\sigma \\ 0 &= N^\top\theta - M_b u \end{aligned}$$

where

$$M = (\phi_i, \phi_j), \qquad A_1 = -(\nabla\phi_i, \nabla\phi_j) + s(\phi_i, \phi_j), \qquad i,j \in \mathcal{I}$$
$$N = (\phi_i, \phi_j), \qquad i \in \mathcal{I}, \quad j \in \mathcal{I}_b$$
$$M_b = (\phi_i, \phi_j), \qquad i,j \in \mathcal{I}_b$$

Note that M_b is just the mass matrix of W_h. If we arrange the dofs such that the boundary dofs appear at the end, then N has the form

$$N = \begin{bmatrix} 0 & 0 \\ 0 & M_b \end{bmatrix}$$

We will now write this in state-space form. Define

$$E = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix}, \qquad A = \begin{bmatrix} A_1 & N \\ N^\top & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ -M_b \end{bmatrix}$$

so that

$$E\frac{\mathrm{d}x}{\mathrm{d}t} = Ax + Bu, \qquad x = \begin{bmatrix} \theta \\ \sigma \end{bmatrix}$$

If we add a shift $s > 2\pi^2$, then this system is unstable to small perturbations since (A, E) has some eigenvalues on the right side of the complex plane. The goal is to find a feedback law

$$u = -Kx$$

such that $(A - BK, E)$ is stable.

4 Computation of feedback law

For this we follow the steps as in Thevenet. The Fenics code `linear.py` computes the matrices M, A_1, N, M_b and saves them to file `linear.mat` which can then read by the Matlab code `gain.m` to compute eigenvalues and the feedback operator. Finally we run the Fenics code `heat.py` to solve the heat equation with or without control.

Suppose there are n_u unstable eigenvalues and let the eigenvalues be ordered as

$$\ldots < \lambda_{n_u+2} < \lambda_{n_u+1} < 0 < \lambda_{n_u} < \ldots \lambda_1$$

Compute the eigenvectors corresponding to the unstable eigenvalues

$$Ae_j = \lambda_j E e_j, \qquad j = 1, 2, \ldots, n_u$$

5 Solving the feedback stabilized problem

Using the feedback law, we get

$$E\frac{\mathrm{d}x}{\mathrm{d}t} = (A - BK)x$$

We can apply backward Euler scheme

$$E\frac{x^n - x^{n-1}}{\Delta t} = (A - BK)x^n$$

While E, A, B are sparse matrices, K can be full which makes it difficult to solve this problem. To avoid this complication, we can compute the feedback using the current solution, i.e., $u^n = -Kx^{n-1}$, which leads to

$$E\frac{x^n - x^{n-1}}{\Delta t} = Ax^n + Bu^n$$

or

$$\left[\frac{1}{\Delta t}E - A\right] x^n = \frac{1}{\Delta t}Ex^{n-1} + Bu^n$$

The matrix on the left is sparse so that the above equation can be solved explicitly. A more accurate estimate of the control can be obtained by extrapolation of previous two solutions

$$u^n = -K(2x^{n-1} - x^{n-2})$$

and use BDF2 scheme which is second order accurate in time

$$E\frac{\frac{3}{2}x^n - 2x^{n-1} + \frac{1}{2}x^{n-2}}{\Delta t} = Ax^n + Bu^n$$

In this procedure, we are simultaneously solving for θ, σ. We can avoid computation of σ as follows. We first compute the solution on the boundary

$$\theta_j^n = u_j^n, \qquad j \in \mathcal{I}_b$$

To determine the interior solution, we have to use test functions which vanish on the boundary. If we use the backward Euler scheme, then we have

$$\left[\frac{1}{\Delta t}M - A_1\right] \theta^n = \frac{1}{\Delta t}M\theta^{n-1} =: r^n$$

The matrix and right hand side vector are modified to satisfy the boundary condition. We can preserve the symmetry of the matrix which allows the use of conjugate gradients method to solve the matrix problem. If necessary, we can compute σ by a post-processing step by taking test functions supported on $\partial\Omega$.

6 Steps to run the code

You can set some parameters in the `param.py` file. Now generate the grid

```
$ python grid.py
```

We first solve the heat equation without control

```
$ python heat.py -time 1.0 -dt 0.01
```

Plot the energy as a function of time

```
$ python plot.py
```

You can see the plot in `energy.pdf` file. The energy should be increasing with time. Next, we will solve the problem with feedback control. First we generate state-space matrices, by running the Python code on terminal

```
$ python linear.py
```

This saves the state-space matrices into file `linear.mat` which can be read in Matlab. Now in Matlab, run

```
> linear
```

On terminal, run the Fenics code

```
$ python heat.py -with_control -time 10.0 -dt 0.01
```

Plot the energy as a function of time

```
$ python plot.py
```

You can see the plot in `energy.pdf` file which should now decrease with time.