Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Parabolické PDR

Budeme řešit rovnici tvaru:

$$\begin{aligned} \frac{\partial u(\vec{x},t)}{\partial t} + \mathcal{L}(u(\vec{x},t),\vec{x},t) &= f(u(\vec{x},t),\vec{x},t) \text{ na } \Omega \times (0,T\rangle \\ u(\vec{x},t) &= g(\vec{x},t) \text{ na } \partial\Omega \times \langle 0,T\rangle \\ u(\vec{x},0) &= u_{ini}(\vec{x}) \text{ na } \Omega \end{aligned}$$

kde

- $u(\vec{x},t) : \Omega \times \langle 0,T\rangle \to \mathbb{R}$ je neznámá
- $\mathcal{L}$ je diferenciální operátor
- Ω je oblast v $\mathbb{R}^n$
- $g(\vec{x},t)$ je okrajová podmínka
- $u_{ini}(\vec{x})$ je počáteční podmínka

Parabolické PDR

- proměnná t má význam vývoje funkce u v čase
- z podstaty naprosté většiny fyzikálních zákonů nemůže stav v čase t záviset na stavu v čase $t + \epsilon$
- časová závislost je tedy pouze zpětná do minulosti
- díky tomu nám stačí znát pouze jeden nebo několik předchozích časových stavů a nemusíme si k řešení pamatovat všechny najednou
- tím se pak liší časová a prostorová diskretizace

Rovnice vedení tepla v 1D

Budeme řešit tuto úlohu (s $\Omega \equiv \langle 0, 1 \rangle$):

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} - \frac{\partial^2 u(x,t)}{\partial x^2} &= f(x,t) \text{ na } \Omega \times (0, T\rangle \\ u(x,t) &= g(x,t) \text{ na } \partial\Omega \times \langle 0, T\rangle \\ u(x,0) &= u_{ini}(x) \text{ na } \Omega \end{aligned}$$

Metoda přímek

Zavedem opět síť uzlů

$$\omega_h = \{x_i = ih \mid i = 0, \dots N\},$$

kde $h = 1/N$ a provedeme diskretizaci v prostoru. Časovou závislost necháme spojitou.

Metoda přímek

Získáme tak soustavu ODR:

$$\begin{aligned}
\frac{du_i(t)}{dt} &= \frac{u_{i-1}(t) - 2u_i(t) + u_{i+1}(t)}{h^2} + f_i(t) \\
&\quad \text{pro } i,j = 1,\ldots N-1 \text{ a } t \in \times(0,T\rangle \\
u_i(t) &= g_i(t) \text{ pro } (i = \vee j = 0 \vee i = N \vee j = N) \text{ a } t \in \langle 0,T\rangle \\
u_i(0) &= u_{ini}(x_i) \text{ pro } i,j = 0,\ldots N
\end{aligned}$$

Na tuto soustavu pak lze aplikovat některou Rungovu-Kuttovu metodu.

Metoda přímek

Implementaci lze najít v souborech:

- `pde/HeatEquationProblem1D.h`
- `pde/HeatEquationProblem2D.h`
- `pde/heat-equation-1d.cpp`

Metoda přímek

```
1  class HeatEquationProblem1D : public ODEProblem
2  {
3     public:
4
5        HeatEquationProblem1D( int size );
6
7        int getDegreesOfFreedom();
8
9        void setParameters();
10
11       void setInitialCondition( double* u );
12
13       void getRightHandSide( const double& t,
14                              double* _u,
15                              double* fu );
16
17       bool writeSolution( const double& t,
18                           int step,
19                           const double* u );
20
21       ~HeatEquationProblem1D();
22
23    protected:
24
25       int size;
26
27       double h;
28 };
```

Metoda přímek

```
1  HeatEquationProblem1D::HeatEquationProblem1D( int size )
2  {
3     this->size = size;
4     this->h = 1.0 / size;
5  }
6
7  int HeatEquationProblem1D::getDegreesOfFreedom()
8  {
9     return this->size;
10 }
11
12 void HeatEquationProblem1D::setInitialCondition( double* u )
13 {
14    for( int i = 0; i < size; i++ )
15    {
16       double x = i * h;
17       u[ i ] = ( x > 0.4 && x < 0.6 ) ? 1.0 : 0.0;
18       //u[ i ] = rand() % 20 - 10;
19    }
20 }
```

Metoda přímek

```
1  void HeatEquationProblem1D::getRightHandSide(
2      const double& t,
3      double* u,
4      double* fu )
5  {
6     /***
7      * Zero Dirichlet boundary conditions
8      */
9     u[ 0 ] = 0.0;
10    u[ size - 1 ] = 0.0;
11    fu[ 0 ] = 0.0;
12    fu[ size -1 ] = 0.0;
13
14    /***
15     * Evaluate the Laplace operator
16     */
17    const double h_sqr = h * h;
18    for( int i = 1; i < size - 1; i++ )
19       fu[ i ] = ( u[ i - 1 ] - 2.0 * u[ i ] + u[ i + 1 ] )
20           / h_sqr;
21 }
```

Metoda přímek

```
1  #include <cstdlib>
2  #include "HeatEquationProblem1D.h"
3  #include "../ode/Euler.h"
4  #include "../ode/Merson.h"
5  #include "../ode/ode-solve.h"
6
7  using namespace std;
8
9  const double initialTime( 0.0 );
10 const double finalTime( 0.05 );
11 const double timeStep( 0.0001 );
12 const double integrationTimeStep( 1.0e-6 );
13 const int size( 100 );
14
15 int main( int argc, char** argv )
16 {
17    HeatEquationProblem1D problem( size );
18
19    Euler integrator;
20
21    double* u = new double[ problem.getDegreesOfFreedom() ];
22    problem.setInitialCondition( u );
23    problem.writeSolution( 0.0, 0, u );
24
25    if( ! solve( initialTime,
26             finalTime,
27             timeStep,
28             integrationTimeStep,
29             &problem,
30             &integrator,
31             u ) )
32    {
33       delete[] u;
34       return EXIT_FAILURE;
35    }
36    delete[] u;
37    return EXIT_SUCCESS;
38 }
```

Metoda přímek

Example 1

Proveďte řadu výpočtů se zmenšujícím se prostorovým krokem a stejným časovým krokem.

Metoda přímek

Domácí úkol: Implementujte metodu přímek pro rovnici vedení tepla ve 2D. Použijte nachystané soubory:

- `pde/HeatEquationProblem2D.h`
- `pde/HeatEquationProblem2D.cpp`
- `pde/heat-equation-2d.cpp`

Zvolte si vhodnou počáteční podmínku např. $sign(x^2 + y^2 - c^2)$ nebo náhodně generované hodnoty a sledujte, jak se řešení vyvíjí v čase. Napište report.

Metoda přímek

Domácí úkol: Pomocí metody `readPGM` třídy `Vector` v `Vector.h` načtěte jako počáteční podmínku obrázek `data/motyl.pgm`:

```
1  int width, height;
2  Vector v;
3  // readPGM ulozi do width a height rozmery obrazku
4  v.readPGM( "../data/motyl.pgm", width, height );
5  double* u = v.getData();
6  // provedte vypocet reseni rovnice vedeni tepla na u
7  ....
8  // zapiste vysledek do obrazku motyl.pgm
9  v.writePGM( "motyl.pgm", width, height );
```

Na obrázek aplikujte rovnici vedení tepla pro různé finální časy, tj. `finalTime`. Napište report.

Eulerova diskretizace

- kromě metody přímek lze také použít Eulerovu dopřednou nebo zpětnou časovou diskretizaci
- zvolíme si časový krok τ a budeme značit

$$u_i^k = u(ih, k\tau)$$

- derivaci podle t pak lze aproximovat takto

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u_i^{k+1} - u_i^k}{\tau}$$

Eulerova zpětná diskretizace

Zpětná Eulerova časová diskretizace vypadá takto:

$$\frac{\partial u(x,t)}{\partial t}\Big|_i^k \approx \frac{u_i^{k+1} - u_i^k}{\tau} = \mathcal{L}_i^k \approx -\frac{u_{i+1}^k - 2u_i^k + u_{i+1}^k}{h^2},$$

což lze přepsat jako

$$u_i^{k+1} = u_i^k - \frac{\tau}{h^2}\left(u_{i+1}^k - 2u_i^k + u_{i+1}^k\right).$$

Dostáváme tak stejnou metodu, jako když použijeme v kombinaci s metodou přímek Eulerovu metodu prvního řádu.

Eulerova dopředná diskretizace

Dopředná Eulerova časová diskretizace vypadá takto:

$$\frac{\partial u(x,t)}{\partial t}\Big|_i^k \approx \frac{u_i^{k+1} - u_i^k}{\tau} = \mathcal{L}_i^{k+1} \approx -\frac{u_{i+1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{h^2},$$

což lze přepsat jako

$$-\lambda u_{i-1}^{k+1} + (1+2\lambda)u_i^{k+1} - \lambda u_{i+1}^{k+1} = u_i^k,$$

kde $\lambda = \tau/h^2$ pro $i = 1, \ldots, N-1$.

Eulerova dopředná diskretizace

To lze přepsat jako soustavu lineárních rovnic:

$$\mathbb{A}\mathbf{u}^{k+1} = \mathbf{u}^{k},$$

kde

$$\mathbb{A} = \begin{pmatrix} 1 & & & & & & \\ -\lambda & 1+2\lambda & -\lambda & & & & \\ & -\lambda & 1+2\lambda & -\lambda & & & \\ & & -\lambda & 1+2\lambda & -\lambda & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -\lambda & 1+2\lambda & -\lambda \\ & & & & & & 1 \end{pmatrix}.$$

Eulerova dopředná diskretizace

- matice $\mathbb{A}$ je regulární a pro libovolné $\lambda > 0$ má předvládající diagonálu
- Jacobiho, Gauss-Seidelova a SOR metoda konvergují pro matice s převládající diagonálou
- pro zmenšující se h konverguje $\lambda \to 0$ a efekt převládající diagonály se zmenšuje, ale stále platí
- naopak, pro dané h mohu použít libovolně velké τ
- to je výhoda dopředné diskretizace - sice musím řešit linerání systém rovnic, ale **mohu volit mnohem větší časový krok τ a nevzniknou mi v řešení oscilace**

Diskretizace v čase

- zpětná Eulerova diskretizace
  - $u_i^{k+1} = u_i^k - \frac{\tau}{h^2}\left(u_{i+1}^k - 2u_i^k + u_{i+1}^k\right)$
  - vztah pro $k+1$ časovou hladinu je dán **explicitně**
  - **stabilita je podmíněna** vztahem $\tau < ch^2$
- dopředná Eulerova diskretizace
  - $-\lambda u_{i-1}^{k+1} + (1+2\lambda)u_i^{k+1} - \lambda u_{i+1}^{k+1} = u_i^k$
  - vztah pro $k+1$ časovou hladinu je dán **implicitně**
  - **stabilita je nepodmíněna**

Implicitní diskretizace rovnice vedení tepla v 1D je v souboru `pde/implicit-heat-equation-1d.cpp`.

Diskretizace v čase

Domácí úkol:

- implementujte implicitní řešič pro rovnici vedení tepla ve 2D do souboru `implicit-heat-equation-2d.cpp`
- proveďte stejné výpočty jako s pomocí metody přímek, ale s větším časovým krokem
- pokuste se najít optimální parametr ω pro SOR metodu
- porovnejte napočítané výsledky a CPU časy
- napište report

Peronova-Malikova úloha

Mějme obrázek zadaný pomocí odstínů šedi, tj. $u_{ini} : \langle 0,1\rangle \times \langle 0,1\rangle \to \langle 0,1\rangle$, kde

- 0 odpovídá černé barvě
- 1 odpovídá bílé barvě

Obrázek u_{ini} budeme filtrovat pomocí Peronovy-Malikovy rovnice:

$$\begin{aligned}
\frac{\partial u(x,y,t)}{\partial t} - \nabla \cdot \left(\frac{\nabla u(x,y,t)}{1 + \frac{\|\nabla u(x,y,t)\|}{K^2}} \right) &= 0 \\
&\quad \text{na } \Omega \times \langle 0, T\rangle, \\
u(t,x,y) &= u_{ini}(x,y) \text{ na } \partial\Omega \times (0,T), \\
u(0,x,y) &= u_{ini}(x,y) \text{ na } \Omega,
\end{aligned}$$

Peronova-Malikova úloha v 1D

Zavedeme značení:

$$p(u,x,t) = \frac{1}{1+\frac{\|\nabla u(x,y,t)\|}{K^2}},$$

což lze aproximovat takto

$$p(u(ih,k\tau),ih,k\tau) = p(u_{i-1}^k,u_i^k) \approx \frac{1}{1+\left\|\frac{u_i^k-u_{i-1}^k}{h}\right\|/K^2}.$$

Následně

$$\begin{aligned}\frac{\partial}{\partial x}\left(p(u,x,t)\frac{\partial u(x,t)}{\partial x}\right) &= \\ \frac{\left(p(u,x,t)\frac{\partial u(x,t)}{\partial x}\right)|_{i+1} - \left(p(u,x,t)\frac{\partial u(x,t)}{\partial x}\right)|_i}{h} &= \\ \frac{1}{h}\left(p(u_i^k,u_{i+1}^k)\frac{(u_{i+1}^k-u_i^k)}{h} - p(u_i^k,u_{i-1}^k)\frac{(u_i^k-u_{i-1}^k)}{h}\right).\end{aligned}$$

Peronova-Malikova úloha v 1D

Explicitní diskretizace pak vypadá takto:

$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{1}{h}\left(p(u_i^k, u_{i+1}^k)\frac{(u_{i+1}^k - u_i^k)}{h} - p(u_{i-1}^k, u_i^k)\frac{(u_i^k - u_{i-1}^k)}{h} \right).$$

Implicitní diskretizace takto:

$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{1}{h}\left(p(u_i^{k+1}, u_{i+1}^{k+1})\frac{(u_{i+1}^{k+1} - u_i^{k+1})}{h} - p(u_i^{k+1}, u_i^{k+1})\frac{(u_i^{k+1} - u_{i-1}^{k+1})}{h} \right),$$

což vede na řešení soustavy **nelineárních** algebraických rovnic. Proto zavedem i tzv. **semi-implicitní diskretizaci**, která vyčísluje nelineární členy s k-té časové řady a vede tak na soustavu lineárních rovnic:

$$\frac{u_i^{k+1} - u_i^k}{\tau} = \frac{1}{h}\left(p(u_i^{k}, u_{i+1}^{k})\frac{(u_{i+1}^{k+1} - u_i^{k+1})}{h} - p(u_i^{k}, u_i^{k})\frac{(u_i^{k+1} - u_{i-1}^{k+1})}{h} \right).$$

Peronova-Malikova úloha v 1D

Semi-implicitní diskretizaci lze přepsat takto:

$$\begin{aligned} u_i^{k+1} - \lambda\left(\alpha u_{i-1}^{k+1} - \beta u_i^{k+1} + \gamma u_{i+1}^{k+1}\right) &= u_i^k \\ -\lambda\alpha u_{i-1}^{k+1} + (1+\lambda\beta)\, u_i^{k+1} - \lambda\gamma u_{i+1}^{k+1} &\quad u_i^k, \end{aligned}$$

kde

$$\begin{aligned} \alpha &= p(u_{i-1}^k, u_i^k), \\ \beta &= p(u_{i-1}^k, u_i^k) + p(u_i^k, u_{i+1}^k), \\ \gamma &= p(u_i^k, u_{i+1}^k), \\ \lambda &= \tau/h^2. \end{aligned}$$

Tj.

$$\mathbb{A}\mathbf{u}^{k+1} = \mathbf{u}^k$$

$$\mathbb{A} = \begin{pmatrix} 1 & & & & & & \\ -\lambda\alpha & 1+\beta\lambda & -\gamma\lambda & & & & \\ & -\lambda\alpha & 1+\beta\lambda & -\gamma\lambda & & & \\ & & -\lambda\alpha & 1+\beta\lambda & -\gamma\lambda & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -\lambda\alpha & 1+\beta\lambda & -\gamma\lambda \\ & & & & & & 1 \end{pmatrix}.$$

Peronova-Malikova úloha v 1D

Explicitní diskretizace pomocí metody přímek je implementována v souborech:

- `PeronaMalikProblem1D.h`,
- `PeronaMalikProblem1D.cpp`,
- `perona-malik-1d.cpp`,

Semi-implicitní diskretizace je implementována v souboru:

- `semi-implicit-perona-malik-1d.cpp`.

Domácí úkol:
Implementujte ve 2D explicitní nebo semi-implicitní řešič Peronovy-Malikovy úlohy aalikutje na odtsranění šumu v obrazku `data/motyl-sum.pgm`.