SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Stavebná fakulta

Riešenie rovnice advekcie pomocou softvéru DUNE

Študentská vedecká konferencia

Bratislava 1.4.2009

Slovenská technická univerzita
v Bratislave
Stavebná fakulta

Študentská vedecká konferencia
konaná 1. apríla 2009
Sekcia: Matematicko-počítačové modelovanie

Riešenie rovnice advekcie pomocou softvéru DUNE.
Solution of advection equation using software DUNE.

Autor práce:	**Maroš Bohunčák**
Ročník/stupeň štúdia:	3. ročník/1. stupeň štúdia
Študijný program:	matematicko-počítačové modelovanie
Vedúci práce:	RNDr. Peter Frolkovič, PhD.
Katedra:	KMaDG

Anotácia:
Práca sa zaoberá návrhom a realizáciou numerickej metódy druhého rádu presnosti na riešenie rovnice advekcie pre dané rýchlostné vektorové pole. Dôraz je kladený na počítačovú realizáciu v modernom softvéri DUNE (Distributed and Unified Numerics Environment), ktorý na základe jednotného rozhrania pre použitie numerických metód na rôznych typoch výpočtových sietí umožňuje ich efektívne použitie od najjednoduchších ako sú rovnomerná, pevná, štruktúrovaná sieť až po komplexné, lokálne zjemnené, v čase premenlivé, neštruktúrované siete. Úlohou študentskej práce je pochopiť základy práce s týmto softvérom, realizovať v ňom novú numerické metódu a preveriť ju na rôznych príkladoch.

Annotation:
This work deals with a proposal and realization of a second order accurate numerical method for the solution of advection equation for a given velocity vector field. The emphasize is given on the implemenntation of such method in modern software tool DUNE (Distributed and Unified Numerics Environment) that offers a single interface for numerical methods based on different types of computational grids to work efficiently with them starting from the simplest uniform, fixed, structured grids and finishing with complex, locally adapted, changing in time, unstructured grids. The aim of this student work is to understand the principles of this software, to realize new numerical method in this environment and to check it on several examples.

1 Úvod

DUNE (*Distributed and Unified Numerics Environment*) je softvérová knižnica napísaná v programovacom jazyku C++ na numerické riešenie parciálnych diferenciálnych rovníc. Podporuje jednoduchú implementáciu metód ako Metóda konečných prvkov (MKP), Metóda konečných objemov (MKO), alebo Metóda konečných diferencií (MKD). **DUNE** je voľne dostupná pod licenciou GPL. Je možné aj použitie s iným komerčným softvérom. Knižnica **DUNE** je rozdelená do samostatných modulov. Pre aktuálnu verziu 1.1 (z 31. marca 2009) sú dostupné moduly:

- `dune-common` obsahuje základné triedy používané všetkými ostatnými **DUNE** modulmi.
- `dune-grid` je najrozvinutejší modul, ktorý je použitý v tejto práci. Definuje nekonformné, hierarchicky vnorené, paralelné mriežky a mriežky s rôznym typom elementov v priestore s ľubovoľným rozmerom. Podporuje grafický výstup napríklad vo formáte VTK.
- `dune-istl` - *Iterative Solver Template Library* poskytuje základné triedy riedkych matíc a vektorov a metódy na ich riešenie. Sú použité moderné programovacie techniky ako generické programovanie či statický polymorfizmus.
- `dune-grid-howto` - manuál [2] s ukážkovou implementáciou metódy prvého rádu, ktorú sme modifikovali v práci na metódu druhého rádu

2 Metóda konečných objemov

Budeme riešiť lineárnu hyperbolickú parciálnu diferenciálnu rovnicu, alebo tzv. rovnicu advekcie

$$\frac{\partial c}{\partial t} + \nabla \cdot (uc) = 0 \quad \text{na} \quad \Omega \times T \tag{1}$$

kde $\Omega \subset \mathbb{R}$ je výpočtová oblasť, $T = (0, t_{end})$ je časový interval, $c : \Omega \times T \to \mathbb{R}$ je neznáma funkcia a $u : \Omega \times T \to \mathbb{R}^d$ je dané rýchlostné pole. Divergencia rýchlostného poľa nech je nulová. Pre rovnicu máme definovanú počiatočnú podmienku

$$c(x, 0) = c_0(x) \quad x \in \Omega \tag{2}$$

a okrajovú podmienku

$$c(x,t) = b(x,t) \quad t > 0, x \in \Gamma_{in}(t) = \{y \in \partial\Omega \quad | \quad u(y,t) \cdot \nu(y) < 0\}. \tag{3}$$

$\nu(y)$ je jednotkový vektor vonkajšej normály v bode $y \in \partial\Omega$ a $\Gamma_{in}(t)$ je prítoková hranica v čase t.

Spojitú diferenciálnu rovnicu (1) aproximujeme diskrétnymi rovnicami pomocou metódy konečných objemov. Výpočtová mriežka (sieť) pozostáva z elementov ω_i a časový interval T je diskretizovaný na kroky $0 = t_0, t_1, \ldots, t_n, t_{n+1}, \ldots, t_N = t_{end}$. Rovnicu (1) prepíšeme do integrálneho tvaru, t.j. integrujeme cez každý element výpočtovej mriežky ω_i a časový interval (t_n, t_{n+1}):

$$\int_{\omega_i} \int_{t_n}^{t_{n+1}} \frac{\partial c}{\partial t} \, dt \, dx + \int_{\omega_i} \int_{t_n}^{t_{n+1}} \nabla \cdot (uc) \, dt \, dx = 0 \quad \forall i. \tag{4}$$

Po čiastočnej integrácii dostávame

$$\int_{\omega_i} c(x, t_{n+1}) \, dx - \int_{\omega_i} c(x, t_n) \, dx + \int_{t_n}^{t_{n+1}} \int_{\partial\omega_i} cu \cdot \nu \, ds \, dt = 0 \quad \forall i \tag{5}$$

Hranicu elementu $\partial\omega_i$ rozdelíme na časti γ_{ij}, čo je buď prienik s iným (susedným) elementom $\partial\omega_i \cap \partial\omega_j$, alebo prienik s hranicou oblasti $\partial\omega_i \cap \partial\Omega$.
Výpočet tokov vedie k nasledujúcej rovnici pre neznáme hodnoty elementov v čase t_{n+1}

$$C_i^{n+1}|\omega_i| - C_i^n|\omega_i| + \sum_{\gamma_{ij}} F_{ij}^{n+\frac{1}{2}} \, |\gamma_{ij}| \triangle t^n = 0 \quad \forall i, \tag{6}$$

kde $\triangle t^n = t_{n+1} - t_n$, $u_{ij}^{n+\frac{1}{2}}$ je rýchlosť na hranici elementu γ_{ij} v jej strednom bode $\mathbf{x}_{ij} \in \gamma_{ij}$ v čase $t_{n+\frac{1}{2}} = t_n + \frac{1}{2}\triangle t^n$, ν_{ij} je jednotkový vektor vonkajšej normály časti hranice γ_{ij} a funkcia toku je definovaná ako

$$F_{ij}^{n+\frac{1}{2}} = \begin{cases} b(\mathbf{x}_{ij},\, t_{n+\frac{1}{2}}) & \mathbf{x}_{ij} \in \Gamma_{in}(t_{n+\frac{1}{2}}) \\ C_{ji}^{n+\frac{1}{2}} \, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij} & u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij} < 0 \\ C_{ij}^{n+\frac{1}{2}} \, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij} & u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij} \geq 0 \end{cases} \tag{7}$$

Hodnota $b(\mathbf{x}_{ij})$ je daná okrajovou podmienkou na prítokovej časti hranice elementu γ_{ij}. Ak zvolíme $C_{ji}^{n+1/2} = b(\mathbf{x}_{ij})$ na $\mathbf{x}_{ij} \in \Gamma_{in}(t)$, môžeme tvar (7) prepísať na

$$F_{ij}^{n+\frac{1}{2}} = C_{ij}^{n+\frac{1}{2}} \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) - C_{ji}^{n+\frac{1}{2}} \max(0, -u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}). \tag{8}$$

2.1 Numerická metóda prvého rádu

Aproximujme $c(x,t)$ funkciou $C(x,t) = C_i^n$, $x \in \omega_i$, $t \in [t_n, t_{n+1})$, pričom C_i^n je hodnota na elemente ω_i v čase t_n. Pri metóde prvého rádu zvolíme $C_{ij}^{n+\frac{1}{2}} = C_i^n$ a funkcia toku má tvar

$$F_{ij}^{n+\frac{1}{2}} = C_i^n \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) - C_j^n \max(0, -u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}). \tag{9}$$

Z (6) po rozpísaní a vyjadrení C_i^{n+1} dostávame

$$C_i^{n+1} = C_i^n - C_i^n \triangle t^n \sum_{\gamma_{ij}} \frac{|\gamma_{ij}|}{|\omega_i|} \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) + \triangle t^n \sum_{\gamma_{ij}} C_j^n \frac{|\gamma_{ij}|}{|\omega_i|} \max(0, -u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij})\, \forall i. \tag{10}$$

Schéma je stabilná, ak je splnené

$$\forall i : 1 - \triangle t^n \sum_{\gamma_{ij}} \frac{|\gamma_{ij}|}{|\omega_i|} \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) \geq 0 \Leftrightarrow \triangle t^n \leq \min_i \left(\sum_{\gamma_{ij}} \frac{|\gamma_{ij}|}{|\omega_i|} \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) \right)^{-1} \tag{11}$$

Prepíšeme (10) na tvar

$$C_i^{n+1} = C_i^n - \triangle t^n \underbrace{\sum_{\gamma_{ij}} \frac{|\gamma_{ij}|}{|\omega_i|} (C_i^n \max(0, u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}) + C_j^n \max(0, -u_{ij}^{n+\frac{1}{2}} \cdot \nu_{ij}))}_{\delta_i} \quad \forall i \tag{12}$$

2.2 Numerická metóda druhého rádu

Pri metóde prvého rádu sme funkciu c aproximovali konštantnou funkciou C. Pri metóde druhého rádu [1] použijeme

$$C_{ij}^{n+\frac{1}{2}} = C^n(\mathbf{x}_{ij}) - \frac{\triangle t^n}{2} \nabla C_i^n \cdot u_i^{n+1/2}, \tag{13}$$

kde $C^n(\mathbf{x})$ je lineárna funkcia definovaná ako

$$C^n(\mathbf{x}) = C_i^n + \nabla C_i^n \cdot (\mathbf{x} - \mathbf{x}_i) \tag{14}$$

a $\mathbf{x}_{ij}$ sú súradnice stredu priesečníka dvoch susedných elementov (časť hranice, ktorou sa elementy dotýkajú).
Gradient pre vnútorný element mriežky ω_i počítame ako

$$\nabla C_{ij} = \left(\frac{C_{i+1j} - C_{i-1j}}{2h}, \frac{C_{ij+1} - C_{ij-1}}{2h} \right) \tag{15}$$

a pre hraničný element na (napríklad ľavom) okraji oblasti

$$\nabla C_{ij} = \left(\frac{C_{i+1j} - C_{ij}}{h}, \frac{C_{ij+1} - C_{ij-1}}{2h} \right) \tag{16}$$

2.3 Výpočet chyby

Na demonštráciu konvergencie oboch metód použijeme jednoduchý výpočet pre tzv. L1 chybu

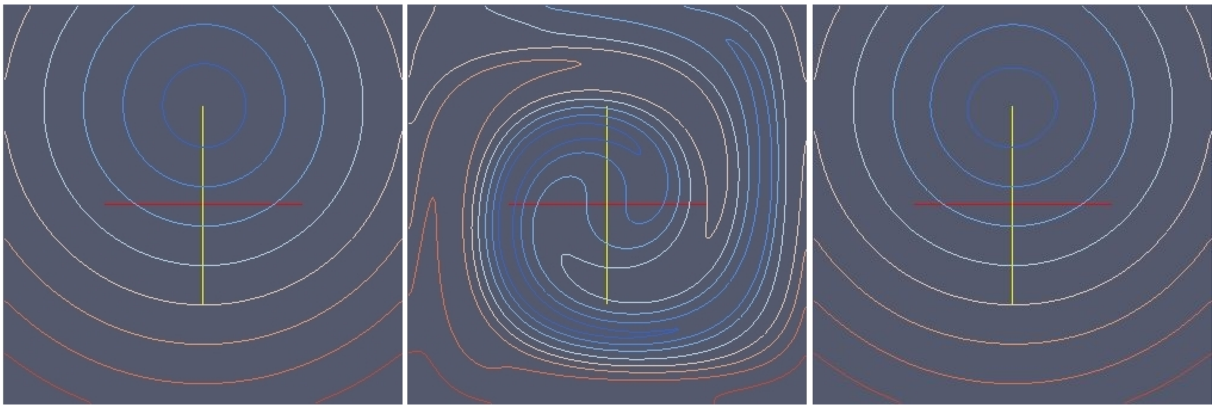
$$E = h^2 \sum_{\omega_{ij}} |c_{ij}^n - cp_{ij}^n| \tag{17}$$

kde cp je presné riešenie.

3 Implementácia v softvéri DUNE

Obe metódy predvedieme na takzvanom single vortex príklade, ktorý je často využívaný na testovanie numerických metód. Úlohu riešime na jednotkovom štvorci $[0, 1] \times [0, 1]$. Počiatočná podmienka je funkcia $\phi(0, x, y) = \sqrt{(x - 0.5)^2 + (y - 0.75)^2} - 0.15 = 0$. Časovo závislé rýchlostné pole je dané ako $\mathbf{V}(x, y, t) = cos(\frac{\pi t}{8})(u(x, y), v(x, y))$, (platí $\nabla \cdot \mathbf{V}(x, y, t) = 0$), pričom: $u(x, y) = -sin^2(\pi x)sin(\pi y)cos(\pi y)$, $v(x, y) = sin^2(\pi y)sin(\pi x)cos(\pi x)$.

Počiatočná funkcia, ktorá ma začiatku tvar kruhu, sa postupne v čase deformuje. V čase $t = 4$, keď je dosiahnutá maximálna deformácia, sa začína tvar funkcie vracať do pôvodného tvaru, ktorú dosiahne v čase $t = 8$ viď Obrázok 1.



Obr. 1: Kontúry numerického riešenia príkladu single vortex v časoch $t = 0$, $t = 4$ a $t = 8$.

3.1 Metóda prvého rádu

Ako prvé si zadefinujeme počiatočné podmienky, okrajové podmienky a rýchlostné pole.

```
// pociatocna podmienka c0
template<int dimworld, class ct>
double c0 (const Dune::FieldVector<ct,dimworld>& x)
{
  return sqrt(pow(x[0]-.5,2)+pow(x[1]-.75,2))-.15;
}

// casovo zavisle rychlostne pole
template<int dimworld, class ct>
Dune::FieldVector<double,dimworld> u (const Dune::FieldVector<ct,dimworld>& x, double t)
{
  Dune::FieldVector<double,dimworld> r;
```

```
  r[0] = -cos(pi*t/8.0)*pow(sin(pi*x[0]),2)*sin(pi*x[1])*cos(pi*x[1]);
  r[1] =  cos(pi*t/8.0)*pow(sin(pi*x[1]),2)*sin(pi*x[0])*cos(pi*x[0]);
  return r;
}

// okrajova podmienka na vtokovej casti hranice
template<int dimworld, class ct>
double b (const Dune::FieldVector<ct,dimworld>& x, double t)
{
  return c0(x);
}
```

Samotný výpočet (12) implementuje funkcia `evolve` s časovým krokom pre ktorý platí (11).

```
template<class G, class M, class V, class Z>
void evolve (const G& grid, const M& mapper, V& c, double t, double& dt, V& cp, Z& gr)
{
  // zistenie dimenzie mriezky
  const int dim = G::dimension;
  const int dimworld = G::dimensionworld;
  double dd = 0.0; double db = 0.0;

  // typ suradnic pouzitych v mriezke
  typedef typename G::ctype ct;

  // typ leaf iterator
  typedef typename G::template Codim<0>::LeafIterator LeafIterator;

  // typ intersection iterator
  typedef typename G::template Codim<0>::LeafIntersectionIterator IntersectionIterator;

  // typ entity pointer
  typedef typename G::template Codim<0>::EntityPointer EntityPointer;

  // pripocitavaci vektor
  V update(c.size());
  for (typename V::size_type i=0; i<c.size(); i++) update[i] = 0;

  // cyklus na vypocet pripocitavacieho vektora a
  for (LeafIterator it = grid.template leafbegin<0>(); it!=endit; ++it)
        {
          // typ geometrie
          Dune::GeometryType gt = it->type();

          // stred elementu v referencnom elemente
          const Dune::FieldVector<ct,dim>&
                local = Dune::ReferenceElements<ct,dim>::general(gt).position(0,0);

          // stred elementu v globalnych suradniciach
          Dune::FieldVector<ct,dimworld>
                global = it->geometry().global(local);

          // velkost elementu
          double volume = it->geometry().integrationElement(local)
                *Dune::ReferenceElements<ct,dim>::general(gt).volume();

          // index elementu
          int indexi = mapper.map(*it);

          // cyklus cez vsetky hrany elementu
          IntersectionIterator isend = it->ileafend();
          for (IntersectionIterator is = it->ileafbegin(); is!=isend; ++is)
                {
                  // typ geometrie hrany
                  Dune::GeometryType gtf = is.intersectionSelfLocal().type();

                  // stred hrany v referencnom elemente
                  const Dune::FieldVector<ct,dim-1>&
```

```
                        facelocal =
                    Dune::ReferenceElements<ct,dim-1>::general(gtf).position(0,0);

                  // vypocet normaloveho vektora
                  Dune::FieldVector<ct,dimworld> integrationOuterNormal
                        = is.integrationOuterNormal(facelocal);
                  integrationOuterNormal
                        *= Dune::ReferenceElements<ct,dim-1>::general(gtf).volume();

                  // stred hrany v globalnych suradniciach
                  Dune::FieldVector<ct,dimworld>
                        faceglobal = is.intersectionGlobal().global(facelocal);

                  // rychlost v strede hrany
                  Dune::FieldVector<double,dim> velocity = u(faceglobal,t);

                  // vypocet suciny a normaloveho vektora
                  double factor = velocity*integrationOuterNormal/volume;

                  // rozlisovanie typu suseda
                  if (is.neighbor())
                  {
                    // pristup na suseda
                    EntityPointer outside = is.outside();
                    int indexj = mapper.map(*outside);

                    // vypocet tokov
                    if ( it->level()>outside->level() ||
                        (it->level()==outside->level() && indexi<indexj) )
                    {
                       // sucin rychlosti a normaloveho vektora v susednom elemente
                       Dune::GeometryType nbgt = outside->type();
                       const Dune::FieldVector<ct,dim>&
                  nblocal = Dune::ReferenceElements<ct,dim>::general(nbgt).position(0,0);
                       double nbvolume = outside->geometry().integrationElement(nblocal)
                        *Dune::ReferenceElements<ct,dim>::general(nbgt).volume();
                       double nbfactor = velocity*integrationOuterNormal/nbvolume;

                        if (factor<0) // tok dovnutra
                        {
                          update[indexi] -= c[indexj] * factor;
                          update[indexj] += c[indexj] * nbfactor;
                        }

                        else // tok von
                        {
                          update[indexi] -= c[indexi] * factor;
                          update[indexj] += c[indexi] * nbfactor;
                        }
                     }
                   }

                // ak je vonkajsia hranica
                if (is.boundary())
                {
                  if (factor<0) // tok dovnutra, aplikacia okrajovej podmienky
                    update[indexi] -= b(faceglobal,t)*factor;

                  else // tok von
                  {
                    update[indexi] -= c[indexi] * factor;
                  }
                }
        } // koniec cyklu cez hranice elementu

    // presne riesenie
    cp[indexi] = b(global,t+dt);

  } // koniec cyklu cez mriezku
```

```
124
125   // nove riesenie
126   for (unsigned int i=0; i<c.size(); ++i)
127         c[i] += dt*update[i];
128
129   return;
130 }
```

Riadky 26-118 obsahujú cyklus cez všetky elementy mriežky, v ktorom sa počíta vektor δ_i. Ten je alokovaný na riadku 22, kde predpokladáme že `V` je objekt s kopírovacím konštruktorom a metódou `size`.

Výpočet tokov je implementovaný v riadkoch 75-116. Pomocou `IntersectionIterator` pristupujeme na časti hranice γ_{ij} elementu ω_i. Ak γ_{ij} je priesečník so susedným elementom ω_i metóda iterátora `neighbor()` vráti true (riadok 75), alebo γ_{ij} je priesečník s hranicou výpočtovej oblasti ak metóda `boundary()` vráti true (riadok 108).

A hlavný program:

```
template<class G>
void timeloop (const G& grid, double tend)
{
  // vytvorenie mappera
  Dune::LeafMultipleCodimMultipleGeomTypeMapper<G,P0Layout>
        mapper(grid);

  // vektor na numericke riesenie, presne riesenie a gradient
  std::vector<double> c(mapper.size());
  std::vector<double> cp(mapper.size());
  std::vector<grad> gr(mapper.size());

  // vypocet hodnot z pociatocnej podmienky
  initialize(grid,mapper,c,cp,gr);

  vtkout(grid,c,"concentration",0);

  // casove kroky
  double t=0,dt;
  int k=0;
  const double saveInterval = 0.1;
  double saveStep = 0.1;
  int counter = 0;

  while (t<tend)
        {
          // pocitadlo krokov
          ++k;

          // numericky vypocet
          evolve(grid,mapper,c,t,dt,cp);

          // casovy krok
          t += dt;

          // zapisovanie vysledkov
          if (t >= saveStep)
                {
                  // write data
                  vtkout(grid,c,"concentration",counter);

                  // zvysit pocitadlo a saveStep pre dalsi interval
                  saveStep += saveInterval;
                  ++counter;
                }

          // info o mriezke case, casovom kroku
```

```
48            std::cout << "s=" << grid.size(0) <<
49            " k=" << k << " t=" << t << " dt=" << dt << std::endl;
50          }
51    // chyba po poslednom casovom kroku
52    chyba(grid,mapper,c,cp);
53
54    // zapis vysledkov
55    vtkout(grid,c,"concentration",counter);
56  }
57
58  // hlavny program
59  int main (int argc , char ** argv)
60  {
61    Dune::MPIHelper::instance(argc,argv);
62
63    int I = 0, z = 0;
64    std::cout <<"Pocet elementov: "<< std::endl;
65    std::cin >>I;
66    std::cout <<"Stupen zjemnenia: "<< std::endl;
67    std::cin >>z;
68
69    // Dune vypise hlasku ak nastane vynimka
70    try {
71      using namespace Dune;
72
73      // vytvorenie siete
74       const int dim=2;
75       typedef Dune::SGrid<dim,dim> GridType;
76       Dune::FieldVector<int,dim> N(I);
77       Dune::FieldVector<GridType::ctype,dim> L(0.0);
78       Dune::FieldVector<GridType::ctype,dim> H(1.0);
79       GridType grid(N,L,H);
80       grid.globalRefine(z);
81
82      // vypocet do casu 8.0
83      timeloop(grid, 8.0);
84    }
85    catch (std::exception & e) {
86      std::cout << "STL ERROR: " << e.what() << std::endl;
87      return 1;
88    }
89    catch (Dune::Exception & e) {
90      std::cout << "DUNE ERROR: " << e.what() << std::endl;
91      return 1;
92    }
93    catch (...) {
94      std::cout << "Unknown ERROR" << std::endl;
95      return 1;
96    }
97
98    return 0;
99  }
```

Funkcia `timeloop` vytvorí mapper a alokuje vektor riešení, v ktorom jeden prvok je jedna hodnota riešenia na mriežke (hodnota vypočítaná na jednom elemente). Tento vektor sa inicializuje na riadku 14 kde sa vypočítajú riešenia z počiatočnej podmienky. Funkcia vtkout zapisuje vypočítané dáta do súboru vo formáte VTK, ktorý je možné použiť napr. v softvéri Paraview na vizualizáciu výsledkov. V cykle `while` je volaná funkcia `evolve`, ktorá realizuje samotný výpočet. Po poslednom časovom kroku je vypočítaná L1 chyba.

V hlavnom programe zadávame počet elementov časový krok a stupeň zjemnenia mriežky pričom jedno zjemnenie znamená rozdelenie každého elementu na polovicu v každom smere.

3.2 Metóda druhého rádu

Pri použití metódy druhého rádu upravíme vo funkcii `evolve` riadky 75-116 čo je implementácia rovnice (13):

```
if (is.neighbor())
 {
  // pristup na suseda
  EntityPointer outside = is.outside();
  int indexj = mapper.map(*outside);

  // vypocet tokov
  if ( it->level()>outside->level() ||
   (it->level()==outside->level() && indexi<indexj) )
    {
      // vypocet sucinu normaloveho vektora a rychlosti
      Dune::GeometryType nbgt = outside->type();
      const Dune::FieldVector<ct,dim>&
        nblocal = Dune::ReferenceElements<ct,dim>::general(nbgt).position(0,0);
      double nbvolume = outside->geometry().integrationElement(nblocal)
        *Dune::ReferenceElements<ct,dim>::general(nbgt).volume();
      double nbfactor = velocity*integrationOuterNormal/nbvolume;

      // globalne suradnice stredu suseda
      Dune::FieldVector<ct,dimworld>
         nbglobal = it->geometry().global(nblocal);

      // rychlost v strede suseda
         Dune::FieldVector<double,dim> velocitynbglobal = u(nbglobal,t+0.5*dt);

      if (factor<0) // tok dovnutra
        {
          dd = gr[indexj].h[0]*(faceglobal[0]-nbglobal[0]) +
            gr[indexj].h[1]*(faceglobal[1]-nbglobal[1]) - (dt/2.0) *
            (gr[indexj].h[0]*velocitynbglobal[0] +
            gr[indexj].h[1]*velocitynbglobal[1]);

          update[indexi] -= ( c[indexj] + dd ) * factor;
          update[indexj] += ( c[indexj] + dd ) * nbfactor;
        }
         else // tok von
        {
          db = gr[indexi].h[0]*(faceglobal[0]-global[0]) +
            gr[indexi].h[1]*(faceglobal[1]-global[1]) - (dt/2.0) *
            (gr[indexi].h[0]*velocityglobal[0] +
            gr[indexi].h[1]*velocityglobal[1]);

         update[indexi] -= ( c[indexi] + db ) * factor;
         update[indexj] += ( c[indexi] + db ) * nbfactor;
        }
    }
  }

// ak hranica oblasti
if (is.boundary())
  {
    if (factor<0) // aplikacia okrajovej podmienky
    update[indexi] -= b(faceglobal,t+0.5*dt)*factor;

    else // outflow
     {
       db = gr[indexi].h[0]*(faceglobal[0]-global[0]) +
         gr[indexi].h[1]*(faceglobal[1]-global[1]) - (dt/2.0) *
         (gr[indexi].h[0]*velocityglobal[0] +
         gr[indexi].h[1]*velocityglobal[1]);

    update[indexi] -= (c[indexi] + db)* factor;
   }
}
```

4 Záver a porovnanie výsledkov

V nasledujúcich tabuľkách uvádzame vypočítané L1 chyby po použití oboch metód pri N časových krokoch a deleniach mriežky 50, 100 a 150.

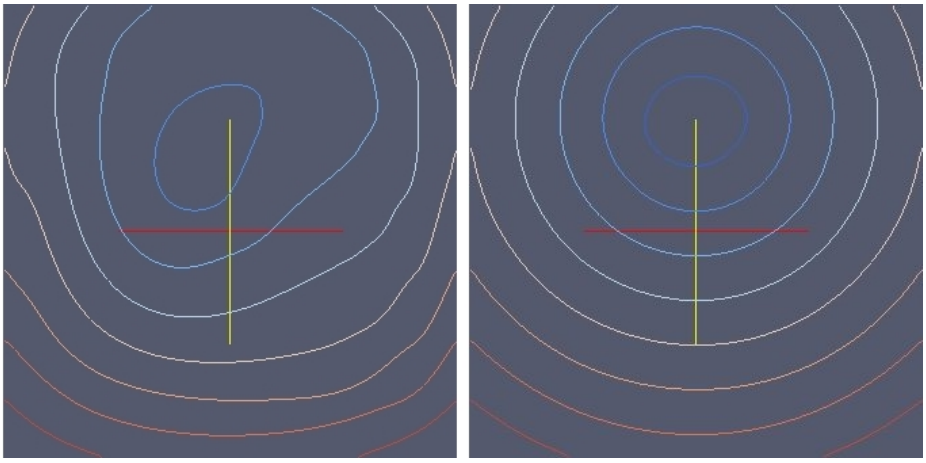
I	N	chyba
50	500	0.0555
100	1000	0.0370
150	1500	0.0280

Tabuľka 1: Chyby po použití metódy prvého rádu

I	N	chyba
50	500	0.00604
100	1000	0.00186
150	1500	0.00080

Tabuľka 2: Chyby po použití metódy druhého rádu

Vidíme, že chyba pri použití metódy prvého rádu konverguje s rádom 1 a metóda druhého rádu, ktorú sme odvodili, konverguje s rádom 2. Pre vizuálne porovnanie ešte uvádzame (Obrázku 2.) vykreslené riešenia v časoch $t = 8$ pri delení mriežky $I = 150$ po použití oboch metód.



Obr. 2: Kontúry numerického riešenia príkladu single vortex v časoch $t = 8$.

Literatúra

[1] Peter Frolkovič, Christian Wehner: *Flux-based level set method on rectangular grids and computation of first arrival time functions*, Computing and Visualization in Science, DOI 10.1007/s00791-008-0115-z, 2008

[2] Peter Bastian, Markus Blatt, Andreas Dedner, Christian Engwer, Robert Klöfkorn, Martin Nolte, Mario Ohlberger, Oliver Sander: *The Distributed and Unified Numerics Environment (DUNE) Grid Interface HOWTO*, Version 1.3svn, March 10, 2009