DISKRÉTNÍ MODEL FYDIK2D

DISCRETE MODEL FYDIK2D

Petr Frantík[1]

Abstract

The paper is focused on detailed description of implementation of discrete dynamical model based on the physical discretization used in the application FyDiK2D. This model is capable to effectively calculate large deformations, chaotic behavior, fracture of various materials and other demanding problems in engineering mechanics.

1 Úvod

Silně nelineární problémy spjaté se strukturální nestabilitou systému (viz např. [1]) patří bezesporu k náročným úlohám. Jedná se zejména o výpočet velkých deformací, přetváření a lomu materiálu, simulace kontaktu částí konstrukce, interakce s hmotou či tekutinami, apod. Ukazuje se, že tyto problémy lze efektivněji řešit pomocí simulací nelineárních dynamických systémů, které se s modelovanou úlohou shodují spíše kvalitativně. Tím se tento přístup výrazně liší od klasických metod zaměřených na kvantitativní výstižnost modelu.

Článek se věnuje detailnímu popisu implementace diskrétního dvourozměrného modelu vzniklého fyzikální diskretizací, tj. náhradou „spojitého“ materiálu diskrétními objekty majícími pouze nepřímý vztah k modelovanému problému. Aplikace modelu je velmi rozmanitá. Byl úspěšně použit pro analýzu stability a velkých deformací prutových konstrukcí [2,3], pro studium deterministického chaosu dynamicky zatížené konstrukce [4], plnění membrány tekutinou [5], pro výpočet lomu křehkých popř. kvazikřehkých materiálů [6].

Tato implementace je veřejně k dispozici ve dvou podobách: volně šířená Java aplikace FyDiK2D s grafickým uživatelským rozhraním [7] a Java balík FyDiK2D [8] vhodný pro integraci do vlastních programů, šířený pod licencí GNU GPL [9].

2 Základní objekty a funkční jednotky modelu

Model včetně objektů, ze kterých je složen, byl vytvořen tak, aby o něm existoval jasný a jednoduchý fyzikální obraz. Tato koncepce vychází z představy, že je možné tento model v určitém přiblížení reálně vyrobit a ověřit tak přímo jeho vlastnosti či chování. Teprve následně je o takovém modelu uvažováno jako o možné fyzikální diskretizaci spojitého problému. Výhodou tohoto přístupu je zejména jednoduchost a průhlednost výpočtu jeho stavu, čímž odpadá nutnost použití složitého matematického, potažmo programového, aparátu. V konečném důsledku tento přístup vede k jednoduchému a efektivnímu algoritmu, který lze dokonce – s výhodou vzhledem k dnešním trendům – snadno paralelizovat.

Na obr. 1 jsou v souladu s jejich fyzikální představou znázorněny dvě základní funkční jednotky: *normálová jednotka* a *ohybová jednotka*, složené ze tří základních

[1] Ing. Petr Frantík, Ph.D., Vysoké učení technické v Brně, Fakulta stavební, Ústav stavební mechaniky, Veveří 95, 602 00 Brno, e-mail: kitnarf@centrum.cz

objektů modelu. Normálová jednotka vznikla spojením dvou *hmotných bodů* pomocí *translační pružiny*. Ohybová jednotka vznikla spojením dvou translačních pružin pomocí *rotační pružiny*. Lze říci, že translační pružina dovoluje vázané normálové přetvoření celku a rotační pružina dovoluje vázané ohybové přetvoření celku.

normálová jednotka ohybová jednotka

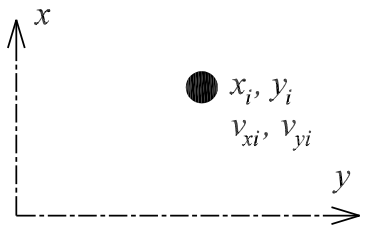
Obr. 1: Základní funkční jednotky modelu

Každý ze tří základních objektů lze z hlediska implementace v jazyce Java chápat jako objekt i ve smyslu objektově orientovaného programování [10], tj. má své vlastnosti, stav a rozhraní[2].

2.1 **Hmotný bod** (třída `MassPoint`)

Hmotný bod je klíčovým objektem modelu. Je vybrán jako jediný nositel hmotnosti a jeho stav je určen souřadnicemi x_i a y_i a složkami rychlosti v_{xi} a v_{yi}. Hmotný bod lze chápat rovněž jako kloub, jelikož zprostředkovává kloubové spojení translačních pružin a navíc tento termín vystihuje reálnou představu.

Množina souřadnic a rychlostí každého hmotného bodu tvoří většinu stavových proměnných modelu. V důsledku akumulace hmotnosti jsou body jedinými příjemci silových interakcí. Ostatní objekty slouží, značně zjednodušeně řečeno, pouze pro jejich zatěžování.



Obr. 2: Hmotný bod

Na obr. 2 je znázorněn hmotný bod s indexem i včetně použitého kartézského souřadného systému a svých stavových proměnných.

Každý hmotný bod má následující atributy: hmotnost m_i, počáteční podmínky $x_i(0) = x_{0i}$, $y_i(0) = y_{0i}$, $v_{xi}(0) = v_{x0i}$, $v_{yi}(0) = v_{y0i}$, okrajové podmínky (např. předepsaná hodnota stavových proměnných) a funkci viskózního útlumu. Funkce viskózního útlumu určuje velikost tlumící síly F_{di}:

$$\vec{F}_{di} = -c_i m_i \vec{v}_i (1 + c_{2i} |\vec{v}_i| + c_{3i} \vec{v}_i \vec{v}_i), \tag{1}$$

kde v_i je vektor rychlosti hmotného bodu a c_i, c_{2i} a c_{3i} jsou dané koeficienty.

Tlumící síla se ve výpočtu zavádí pomocí metody `addDampingForces()`, vypadající následovně:

```
protected void addDampingForces()
{
     rx-=c*mass*vx*(1+c2*Math.abs(vx)+c3*vx*vx);
     ry-=c*mass*vy*(1+c2*Math.abs(vy)+c3*vy*vy);
}
```

[2] Pojmem rozhraní jsou označovány metody sloužící pro zjištění a změnu stavu objektu.

kde `rx,ry` jsou složky výslednice R_i působící na hmotný bod a `mass` je hmotnost bodu m_i. Metoda `Math.abs()` počítá absolutní hodnotu čísla. Složky `rx,ry` jsou na počátku každého kroku inicializovány pomocí metody `initializeResultants()`:

```
protected void initializeResultants()
{
     rx=0;
     ry=0;
}
```

Zatěžování hmotného bodu je implementováno pomocí metody `addLoad()`, která zajišťuje sčítání složek sil působících na hmotný bod:

```
public void addLoad(double fx,double fy)
{
     rx+=fx;
     ry+=fy;
}
```

kde `fx,fy` jsou složky vektoru působící síly.

Uvedené metody doplňuje výpočet zrychlení hmotného bodu. Provádí se závěrem po stanovení celkové výslednice působících sil metodou `updateAcceleration()`:

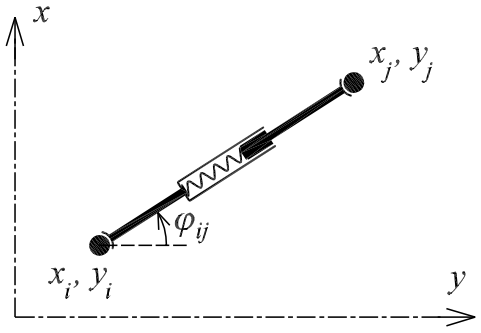
```
protected void updateAcceleration()
{
     ax=rx/mass;
     ay=ry/mass;
}
```

kde `ax,ay` jsou složky vektoru zrychlení hmotného bodu.

2.2 Translační pružina (třída `TranslationalSpring`)

Translační pružina zajišťuje spojení dvou hmotných bodů. Fyzikální představa translační pružiny je lehká neohebná teleskopická tyč s vnitřní pružinou, viz obr. 3. Na hmotný bod působí interakční silou F_{tij}, jejíž velikost je dána zvolenou funkcí a prodloužením pružiny Δl_{ij}. Směr síly je dán směrem translační pružiny.

Translační pružina má následující atributy: délku l_{ij}, počáteční úhel φ_{0ij}, reference na dva hmotné body a funkci interakční síly $F_{tij}(\Delta l_{ij})$.



Obr. 3: Translační pružina

Pro prodloužení pružiny Δl_{ij} platí:

$$\Delta l_{ij} = l_{aij} - l_{ij}, \quad l_{aij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}, \tag{2}$$

kde l_{aij} je aktuální délka pružiny.

Translační pružina má jednu stavovou proměnnou, kterou je její absolutní pootočení φ_{aij}. Důvodem proč se jedná o stavovou proměnnou je požadavek zaznamenání rotace větší než 2π respektive menší než nula. Tento požadavek souvisí s definicí rotační pružiny, viz dále. Ze souřadnic propojených hmotných bodů lze totiž získat pouze úhel φ_{sij} v intervalu $(0, 2\pi)$ z výrazu:

$$\sin\varphi_{sij} = \frac{y_j - y_i}{l_{aij}}. \tag{3}$$

Implementace metody `getActualLimitedAngle()` výpočtu úhlu φ_{sij} dle tohoto výrazu může vypadat v jazyce Java následovně:

```
public double getActualLimitedAngle()
{
     if(ux>=0)
     {
          if(uy>=0) return Math.asin(actualSinus);
          else return Math.asin(actualSinus)+2*Math.PI;
     }
     else return Math.PI-Math.asin(actualSinus);
}
```

kde `ux,uy` jsou složky vektoru translační pružiny v aktuálním stavu a `actualSinus` je aktuální sinus úhlu φ_{sij} respektive φ_{aij}. Implementace stanovení absolutního pootočení φ_{aij} je patrná v rámci metody `updateState()`, zajišťující výpočet aktuálního stavu translační pružiny:

```
protected void updateState(double time)
{
     //výpočet vektoru pružiny
     ux=massPoint2.getX()-massPoint1.getX();
     uy=massPoint2.getY()-massPoint1.getY();
     //
     //
     //výpočet aktuální délky
     //sqrt(x^2+y^2)
     actualLength=Math.hypot(ux,uy);
     //
     //výpočet funkcí úhlu
     actualSinus=uy/actualLength;
     actualCosinus=ux/actualLength;
     //
     //
     //výpočet aktuálního limitovaného úhlu
     double newFiLimited=getActualLimitedAngle();
     //
     //oprava na kumulovaný úhel včetně nastavení počátečních hodnot
     if(!isFi0Set)
     {
          //první voláni
          fi0=newFiLimited;
          isFi0Set=true;
          fi0Limited=fi0;
          //
          //nastavení aktuálních hodnot na počáteční hodnoty
          actualFi=fi0;
          actualFiLimited=fi0Limited;
     }
     else
     {
          //přírůstek oproti minulému stavu
          double dFiLimited=newFiLimited-actualFiLimited;
          //
          //oprava singularity
          if(dFiLimited>Math.PI) dFiLimited-=2*Math.PI;
```

```
            else if(dFiLimited<-Math.PI) dFiLimited+=2*Math.PI;
            //
            //nový stav
            actualFi+=dFiLimited;
            actualFiLimited=newFiLimited;
        }
        //
        //
        //výpočet prodloužení
        elongation=actualLength-length;
        //
        //výpočet aktuální síly
        if(isFunction())
        {
            getFunction().updateState(this);
            force=getFunction().getF(this);
        }
        else force=0;
        //
        //rozklad do složek
        nx=force*actualCosinus;
        ny=force*actualSinus;
    }
```

Metoda `updateState()` definovaná v objektu translační pružiny provádí výpočet aktuálního stavu v následujících krocích:

- Stanoví se složky vektoru translační pružiny `ux,uy`.
- Vypočte se aktuální délka pružiny speciální metodou z Java API `Math.hypot()`, viz [11], která zajišťuje bezpečný výpočet odmocniny součtu druhých mocnin.
- Určí se hodnoty goniometrických funkcí úhlu pružiny.
- Stanoví se limitovaný úhel φ_{sij} metodou `getActualLimitedAngle()` popsanou výše.
- Podle indikátoru `isFi0Set` nastavení počátečních hodnot úhlu pružiny se provede větvení. Buď se nastaví počáteční hodnoty φ_{0aij} a φ_{0sij}, nebo se vypočte absolutní pootočení φ_{aij}. Tento výpočet je založen na předpokladu, že přírůstek tohoto úhlu v jednom kroku je menší respektive větší než 2π respektive -2π. Jelikož hodnota limitovaného úhlu φ_{sij} může vlivem způsobu stanovení náhle „skočit“, lze díky této podmínce takový skok odhalit a změnu absolutního úhlu φ_{aij} provést korektně.
- Závěrem se stanoví prodloužení pružiny Δl_{ij} označené `elongation`, vnitřní síla `force` dle dané funkce $F_{tij}(\Delta l_{ij})$ a složky jejího vektoru `nx,ny`.

Zatížení hmotných bodů složkami `nx,ny` provádí metoda `loadMassPoints()`, která vypadá následovně:

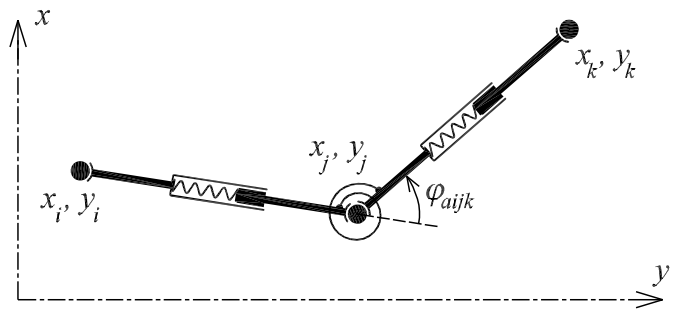
```
protected void loadMassPoints()
{
    massPoint1.addLoad(nx,ny);
    massPoint2.addLoad(-nx,-ny);
}
```

2.3 Rotační pružina (třída `RotationalSpring`)

Rotační pružina zajišťuje ohybové spojení dvou translačních pružin. Fyzikální představa rotační pružiny je lehký svinutý pásek kovu (pružina v hodinovém stroji) popř. určitá objímka kloubu, který spojuje translační pružiny, viz obr. 4. Na translační pružiny působí interakčním momentem M_{rijk}, jehož velikost je dána zvolenou funkcí a pootočením rotační pružiny $\Delta\varphi_{ijk}$. Moment M_{rijk} se rozkládá na dvě dvojice sil F_{rij}

a F_{rjk}, působící na hmotné body i, j respektive j, k. Tyto síly působí kolmo na translační pružiny.

Rotační pružina má následující atributy: úhel φ_{ijk}, reference na dvě translační pružiny a funkci interakčního momentu $M_{rijk}(\Delta\varphi_{ijk})$.



Obr. 4: Rotační pružina

Pro pootočení pružiny $\Delta\varphi_{ijk}$ platí:

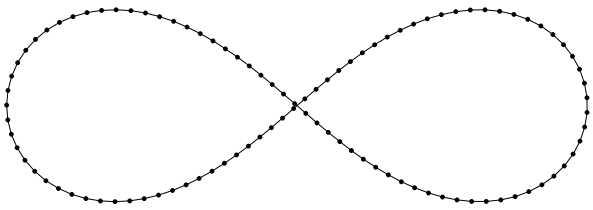
$$\Delta\varphi_{ijk} = \varphi_{aijk} - \varphi_{ijk}, \quad \varphi_{aijk} = \varphi_{ajk} - \varphi_{aij}, \tag{4}$$

kde φ_{aijk} je aktuální vzájemné pootočení translačních pružin, tj. rozdíl pootočení φ_{aij}, φ_{ajk}.

Rotační pružina nemá žádnou stavovou proměnnou. Její stav je jednoznačně dán parametry a stavem translačních pružin. Určení parametrů pružiny pro libovolně definované translační pružiny má ovšem při implementaci svá úskalí. Existují zde dva problémy související se stanovením úhlu φ_{ijk} nenapjaté pružiny.

První problém je způsoben variabilitou definice obou druhů pružin. Chceme-li namodelovat jednu ohybovou jednotku, je třeba definovat dvě translační pružiny spojující celkem tři hmotné body. Jelikož jeden hmotný bod je centrální a úhel každé translační pružiny závisí na pořadí hmotných bodů, máme celkem čtyři variace definice translačních pružin. Samotnou rotační pružinu lze pak definovat dvěma způsoby, což dohromady znamená osm variací definice. Každá variace dává jinou dvojici úhlů translačních pružin.

Druhý problém lze vysvětlit na cyklické struktuře, jakou je například prut stočený do kružnice, jehož konce jsou spojeny (obruč). Přímočarý model tohoto prvku vede ke vzniku „singularity“, která má za výsledek ustálení modelu do tvaru podobného lemniskátě (osmička), viz obr. 5.



Obr. 5: Deformace kružnice vlivem existence singularity při definici rotační pružiny

Pro odstranění obou problémů byla do implementace zavedena možnost volby korekce definice rotační pružiny. Jelikož je celý problém způsoben výpočtem úhlu translační pružiny φ_{aij} potažmo φ_{sij}, postačuje přičíst respektive odečíst násobky úhlu 2π. Implementace je patrná v rámci metody `updateState()`, tentokrát přítomná v rámci objektu rotační pružiny:

```
protected void updateState(double time)
{
     //získání úhlů translačních pružin
     fi1=translationalSpring1.getFi(massPoint1);
     fi2=translationalSpring2.getFi(massPoint2);
     //
     //výpočet vzájemného pootočení translačních pružin
     actualAngle=fi2-fi1;
     //
     //
     //kontrola počáteční hodnoty a korekce singularity
     if(!fiChecked)
     {
          //rozdíl aktuálního úhlu a úhlu pružiny
          double dFi=actualAngle-angle;
          //
          //výpočet násobku úhlu 2PI
          int foldFi=(int)(dFi/2/Math.PI);
          //
          //stanovení korekčního úhlu
          double angleCorrection=0;
          if(Math.abs(dFi)>Math.PI) angleCorrection=foldFi*2*Math.PI;
          //
          //korekce v případě volby correctAngle
          if(correctAngle) angle+=angleCorrection;
          fiChecked=true;
     }
     //
     //
     //výpočet pootočení pružiny
     rotation=actualAngle-angle;
     //
     //výpočet momentu
     if(isFunction())
     {
          getFunction().updateState(this);
          moment=getFunction().getF(this);
     }
     else moment=0;
     //
     //výpočet dvojic sil a jejich složek
     force1=moment/translationalSpring1.getActualLength();
     fx1=force1*translationalSpring1.getSinus(massPoint1);
     fy1=force1*translationalSpring1.getCosinus(massPoint1);
     //
     force2=moment/translationalSpring2.getActualLength();
     fx2=force2*translationalSpring2.getSinus(massPoint2);
     fy2=force2*translationalSpring2.getCosinus(massPoint2);
}
```

Metoda `updateState()` definovaná v objektu rotační pružiny provádí výpočet aktuálního stavu v následujících krocích:

- Stanoví se úhly translačních pružin `fi1,fi2`, odpovídající úhlům φ_{aij}, φ_{ajk}.
- Vypočte se aktuální úhel `actualAngle` mezi translačními pružinami odpovídající úhlu φ_{aijk}.
- Následuje inicializace rotační pružiny při prvním volání `updateState()` indikovaná pomocí `fiChecked`. V rámci tohoto kroku se provede oprava úhlu φ_{ijk} nenapjaté pružiny, pokud je nastaven indikátor `correctAngle`. Jinak se zadání ponechá v původním stavu.
- Stanoví se pootočení rotační pružiny $\Delta\varphi_{ijk}$ podle vztahu (4) označené `rotation`.
- Závěrem se stanoví velikost momentu M_{rijk} označeného `moment`, jemu odpovídající dvě dvojice sil F_{rij} a F_{rjk} označené `force1` a `force2` a složky jejich vektorů `fx1,fy1` a `fx2,fy2`.

Zatížení hmotných bodů složkami `fx1,fy1` a `fx2,fy2` provádí – analogicky jako u translační pružiny – metoda `loadMassPoints()`, která vypadá následovně:

```
protected void loadMassPoints()
{
    massPoint1.addLoad(fx1,-fy1);
    massPoint2.addLoad(-fx1-fx2,fy1+fy2);
    massPoint3.addLoad(fx2,-fy2);
}
```

3 Řešení dynamického systému

Diferenciální rovnice dynamického systému, uceleně definovaného třídou `Model`, vzniklého kompozicí uvedených objektů, lze zapsat ve tvaru:

$$\frac{dx_i}{dt} = v_{xi}, \quad \frac{dv_{xi}}{dt} = \frac{R_{xi}}{m_i}, \qquad \frac{dy_i}{dt} = v_{yi}, \quad \frac{dv_{yi}}{dt} = \frac{R_{yi}}{m_i}, \tag{5}$$

kde R_{xi}, R_{yi} jsou složky výslednice sil R_i působících na hmotný bod i. V aktuální verzi je tento systém řešen pomocí tří explicitních numerických metod vhodných pro disipativní dynamické systémy tohoto typu. Jedná se o metody: *Eulerovu*, *Modifikovanou Eulerovu* a metodu *Runge-Kutta*. Tyto metody mají parametr krok časové diskretizace h. Jejich použitím obdržíme stavy v časech $t = 0, h, 2h, 3h$, atd.

Popis výpočtu provedeme na nejsložitější z implementovaných metod, tj. na metodě Runge-Kutta. Připomeňme, že tato metoda provádí výpočet nového kroku podle následujícího předpisu, viz např. [1]:

$$\begin{aligned}
k_{1j} &= f_j\left(t, s_1(t), s_2(t), \ldots, s_n(t)\right), \\
k_{2j} &= f_j\left(t + h/2, s_1(t) + k_{11}h/2, s_2(t) + k_{12}h/2, \ldots, s_n(t) + k_{1n}h/2\right), \\
k_{3j} &= f_j\left(t + h/2, s_1(t) + k_{21}h/2, s_2(t) + k_{22}h/2, \ldots, s_n(t) + k_{2n}h/2\right), \\
k_{4j} &= f_j\left(t + h, s_1(t) + k_{31}h, s_2(t) + k_{32}h, \ldots, s_n(t) + k_{3n}h\right), \\
s_j(t+h) &= s_j(t) + h/6\left(k_{1j} + 2k_{2j} + 2k_{3j} + k_{4j}\right), \quad j = 1, 2, \ldots, n,
\end{aligned} \tag{6}$$

kde f_j je pravá strana zvolené diferenciální rovnice z výrazu (5), s_j je stavová proměnná s indexem j a dále n je počet stavových proměnných modelu respektive počet řešených diferenciálních rovnic, tj. čtyřnásobek počtu hmotných bodů (na každý bod připadají čtyři stavové proměnné).

Na nejvyšší úrovni – v rámci objektu řešiče (třída `Solver`) – je tato metoda implementována metodou `stepRungeKutta()`:

```
protected void stepRungeKutta(double time,double timeStep)
{
    double h2=timeStep/2;
    double h6=timeStep/6;
    //
    //uložení starého stavu
    saveOldStateVariables();
    //
    //získaní kolekce hmotných bodů
    ArrayList<MassPoint> mpCln=model.getMassPointCollection();
    //
    //
```

```
        //cyklus pro k1 (stavové proměnné pro k1 již nastaveny)
        //uložení k1
        for(MassPoint mp:mpCln) mp.saveK1();
        //
        //
        //cykly pro k2
        //nastavení předepsaných hodnot stavových proměnných a aktualizace stavu
        for(MassPoint mp:mpCln) mp.setStateVariablesForK2(h2);
        updateState(time+h2);
        //uložení k2
        for(MassPoint mp:mpCln) mp.saveK2();
        //
        //
        //cykly pro k3
        //nastavení předepsaných hodnot stavových proměnných a aktualizace stavu
        for(MassPoint mp:mpCln) mp.setStateVariablesForK3(h2);
        updateState(time+h2);
        //uložení k3
        for(MassPoint mp:mpCln) mp.saveK3();
        //
        //
        //cykly pro k4
        //nastavení předepsaných hodnot stavových proměnných a aktualizace stavu
        for(MassPoint mp:mpCln) mp.setStateVariablesForK4(timeStep);
        updateState(time+timeStep);
        //uložení k4
        for(MassPoint mp:mpCln) mp.saveK4();
        //
        //
        //výpočet nového stavu
        for(MassPoint mp:mpCln) mp.stepRungeKutta(h6);
        //
        //aktualizace nového stavu
        updateState(time+timeStep);
    }
```

K výpisu metody `stepRungeKutta()` poznamenejme, že metoda cyklu **`for`**`()` volá vpravo od ní uvedenou metodu – například `saveK1()` – hmotného bodu `mp` a to pro všechny hmotné body v kolekci `mpCln`.

Metoda `stepRungeKutta()` je členěna do následujících částí:

- Výpočet zlomků časového kroku h.
- Uschování aktuálního stavu $\boldsymbol{s}(t) = \{s_1(t), s_2(t), s_3(t), \ldots, s_n(t)\}$.
- Sestavení kolekce všech hmotných bodů `mpCln`.
- Postupný výpočet vektorů pravých stran $\boldsymbol{k}_1$, $\boldsymbol{k}_2$, $\boldsymbol{k}_3$, $\boldsymbol{k}_4$. Nejprve se přepočtou stavové proměnné dle předpisu (6) pomocí metod `setStateVariablesForK()`, v zápětí dojde k aktualizaci stavu pomocí metody `updateState()` uvedené dále a nakonec se uloží hodnoty $\boldsymbol{k}$ pomocí metod `saveK()`.
- Závěrem se provede výpočet nových hodnot stavových proměnných $\boldsymbol{s}(t + h)$ a poslední aktualizace stavu.

Zmíněná metoda `updateState()` zajišťující aktualizaci stavu všech objektů modelu (přítomná v rámci objektu řešiče) má následující tvar:

```
synchronized protected void updateState(double time)
{
    //inicializace
    for(MassPoint mp:model.getMassPointCollection()) mp.initializeResultants();
    for(MassPoint mp:model.getMassPointCollection()) mp.addDampingForces();
    //
    //aktualizace stavu
    for(ModelElement me:model.getModelElementCollection()) me.updateState(time);
    //
    //zatěžování
    for(ModelElement me:model.getModelElementCollection()) me.loadMassPoints();
```

```
        //
        //výpočet zrychlení
        for(MassPoint mp:model.getMassPointCollection()) mp.updateAcceleration();
    }
```

Provádí tedy v zásadě tři činnosti: Zajišťuje výpočet výslednice R_i působící na každý hmotný bod, aktualizuje stav všech objektů modelu a stanoví výsledné aktuální hodnoty zrychlení hmotných bodů.

Upřesněme, že výpočet výslednice R_i sestává z volání metod:

- `initializeResultants()` – nulování složek,
- `addDampingForces()`– přidání tlumících sil,
- `loadMassPoints()`– zatěžování připojenými objekty.

Vlastní simulace pak probíhá opakovaným voláním metody `stepRungeKutta()`.

4 Závěr

V článku byl detailně popsán diskrétní model FyDiK2D včetně implementací jeho klíčových objektů a metod v Jazyce Java. Z uvedeného je patrné že jeho hlavní předností je schopnost řešit velmi náročné úlohy pomocí základního aparátu (nejsložitější ucelenou matematickou operací byla *odmocnina* a funkce *arcus sinus*).

Poděkování

Výsledek byl získán za finančního přispění MŠMT, projekt 1M0579, v rámci činnosti výzkumného centra CIDEAS a s využitím výsledků při řešení projektu GA ČR 103/07/1276.

Literatura

[1] Macur, J. ÚVOD DO TEORIE DYNAMICKÝCH SYSTÉMŮ A JEJICH SIMULACE, skripta, nakladatelství PC-DIR, Brno 1995

[2] Frantík, P. STABILITY STUDY OF THE ELASTIC LOOP, 5th International PhD Symposium in Civil Engineering, Delft June 2004, Netherlands, 2004

[3] Frantík, P. SIMULATION OF THE STABILITY LOSS OF THE VON MISES TRUSS IN AN UNSYMMETRICAL STRESS STATE, journal ENGINEERING MECHANICS, Vol. 14, No. 3, 2007

[4] Frantík, P. POST-CRITICAL BEHAVIOUR OF BEAM LOADED BY FOLLOWER FORCE, CD proceedings of national conference ENGINEERING MECHANICS 2009, Svratka, 2008, 4 pages

[5] Frantík, P., Vořechovský M. DYNAMICKÝ MODEL FLEXIBILNÍ TRUBKY, CD sborník konference INŽENÝRSKÁ MECHANIKA 2007, Svratka, 2007, 8 stran

[6] Frantík, P., Keršner, Z., Veselý, V., Řoutil, L. FRACTALITY OF SIMULATED FRACTURE, KEY ENGINEERING MATERIALS, Vol. 409, 2009, pp 154-160

[7] Frantík, P. FYDIK2D APPLICATION: http://kitnarf.cz/fydik, 2007

[8] Frantík, P. FYDIK2D PACKAGE: http://kitnarf.cz/java/, GNU GPL licence, 2009

[9] Free Software Foundation, GNU GENERAL PUBLIC LICENSE, http://www.gnu.org/licenses/

[10] Wikipedia, OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ, http://cs.wikipedia.org/

[11] Sun Microsystems, JAVA 2 PLATFORM STANDARD EDITION, http://java.sun.com/javase/6/