Zpráva s popisem softwarového návrhu a specifikací rozhraní prototypového modulu XALM pro iterační řešení soustav nelineárních rovnic

XALM (EXternal module for Arc Length Method iteration) je knihovna sloužící pro iterační řešení soustav nelineárních rovnic pomocí Metody délky oblouku (ALM - Arc Length Method). Knihovna funguje jako modul, který může být připojen k libovolnému softwarovému nástroji založeném na analýze problémůpomocí metody konečných prvků (dále jen MKP řešič), který vyžaduje řešení soustavy nelineárních rovnic. ALM metoda byla vybrána pro svoji robustnost a možnost řízení přitěžování konstrukce buď prostřednictvím přírůstků zatížení nebo přírůstkem posunutí. Může být použita pro geometricky a materiálově nelineární konstrukce a velmi užitečná je pro úlohy s lokalizací deformace. Funkčnost knihovny byla ověřena na jednoduchých testech pro geometricky nelineární prutové konstrukce.

Knihovna je napsána v jazyce C++ a vydána pod otevřenou licencí GNU LGPLv2+. Zdrojové kódy, volně dostupné na domovské stránce http://mech.fsv.cvut.cz/xalm včetně dokumentace, využívají pouze standardní knihovny jazyka C++ a mohou být přeloženy na platformách Linux, Windows i Mac OS.

1 Strategie řešení nelineárních rovnic

Algoritmus deformační varianty MKP vede na přírůstkovou formu rovnic rovnováhy, které můžeme napsat např. ve tvaru (viz [1])

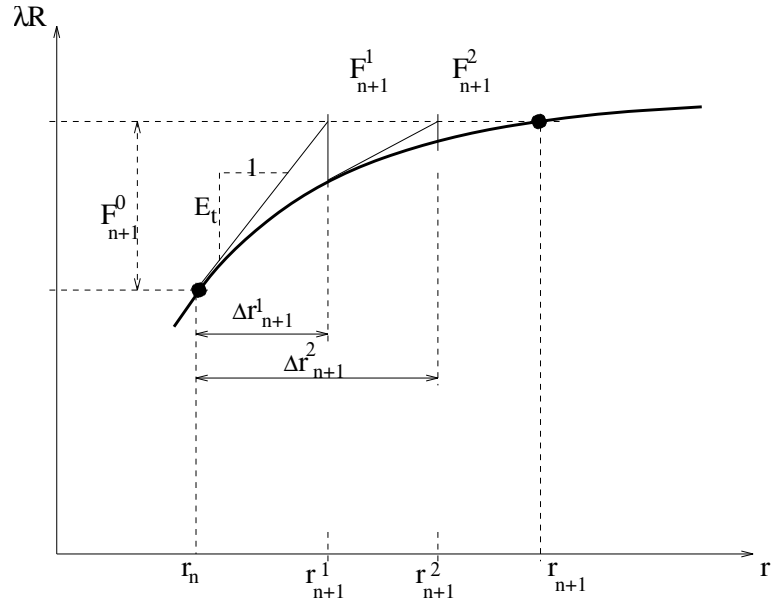
$$\boldsymbol{K}_t \Delta \boldsymbol{r} = \Delta \boldsymbol{R} + (\boldsymbol{R} - \boldsymbol{F}(\boldsymbol{r})). \tag{1}$$

Takovou soustavu nelineárních rovnic neumíme řešit přímo, řešení hledáme iteračně. Metoda ALM vychází z Newtonova-Raphsonovy iterační metody

1.1 Newtonova-Raphsonova metoda - NRM

Princip NRM spočívá v rozložení řešení na posloupnost kroků, ve kterých je postupně aplikován přírůstek zatížení $\Delta \boldsymbol{R}$. Pro každý přírůstek $\Delta \boldsymbol{R}$ se snažíme N-R iteračním postupem o splnění podmínek rovnováhy (1). Iterační postup je kontrolován normou nevyrovnaných sil nebo normou iteračních přírustků posunutí. Algoritmus metody je pro jeden stupeň volnosti znázorněn na obr.1 a shrnut v tab. 1. Podrobnější informace lze najít v mnoha pramenech, např. v [2]. Rychlost konvergence této metody je kvadratická. V této standardní variantě je tečná matice $\boldsymbol{K}_t$ sestavována a faktorizována v každém iteračním kroku, což v některých případech nemusí být nejefektivnější a vede k velké spotřebě strojového času. V modifikované NRM je algoritmus stejný jako v tab. (1), ale tečná matice tuhosti je aktualizována buď jen na začátku iterace, nebo po dosažení určitého počtu iteračních kroků. NRM může být použita ve dvou variantách a to v tzv. variantě řízené přírůstkem zatížení (“load control”) nebo ve variantě řízené přírůstkem posunutí (“displacement control”).[1] Bohužel, obě tyto varianty mají své nevýhody.

[1] Předepisují se vybrané posuny a síly se pak dopočítávají jako reakce.



Obrázek 1: Newtonova-Raphsonova metoda.

Pokud je použita kontrola přírůstkem zatížení, algoritmus není schopen překonat limitní bod na zatěžovací dráze. Limitní bod lze překonat pomocí kontroly přírůstkem posunutí, ta ale zase selhává při procházení dráhy následující po bodu zvratu. Poměrně robustní algoritmus lze založit na kombinaci řízení přírůstkem zatížení a posunutí, viz např. [2].

1.2 Metoda délky oblouku

Metodu délky oblouku (“Arc Length Method” - ALM) navrhli Riks a Wempner a vychází z Newtonova-Raphsonovy metody. Vyjděme z vektoru nevyrovnaných sil, který zapíšeme ve tvaru

$$\boldsymbol{g}(\lambda, \boldsymbol{r}) = \lambda \bar{\boldsymbol{R}} - \boldsymbol{F}(\boldsymbol{r}), \tag{2}$$

kde vektor skutečných vnitřních sil $\boldsymbol{F}$ je funkcí vektoru posunutí $\boldsymbol{r}$ a $\bar{\boldsymbol{R}}$ je vektor zatížení. Na rozdíl od klasických algoritmů je úroveň zatížení λ chápána jako další proměnná. Potřebujeme proto další skalární rovnici. Pro diferenciál délky oblouku platí

$$ds = \sqrt{d\boldsymbol{r}^T d\boldsymbol{r} + d\lambda^2 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}}\,. \tag{3}$$

Pokud tuto rovnici zapíšeme v přírustkovém tvaru, dostáváme hledanou skalární rovnici ve formě omezující podmínky

$$q = \Delta\boldsymbol{r}^T \Delta\boldsymbol{r} + \Delta\lambda^2 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}} - \Delta l^2 = 0. \tag{4}$$

V rovnici (3) se používá zobecněná metrika. Parametr ψ vyjadřuje poměr měřítek pro $\boldsymbol{r}$ a $\lambda\boldsymbol{R}$. Volbou $\psi = 0$ dosáhneme řízení iterace normou přírůstku vektoru posunutí (tato varianta bývá v literatuře nazývána cylindrickou ALM) nebo pokud položíme ψ dostatečně velké, potom řešení

Tabulka 1: Algoritmus NRM.

A.	Inicializace $i = 0$
A.1.	$\Delta \boldsymbol{r}^0_{n+1} = 0$
A.2.	$\boldsymbol{r}^0_{n+1} = \boldsymbol{r}_n$
A.3.	Sestavení a faktorizace $\boldsymbol{K}^0_{t,n+1}$
A.4.	$\boldsymbol{F}^0_{n+1} = \boldsymbol{F}_n$
B.	Pro $i = 1, 2, ...$ (iterace)
B.1.	Sestavení pravé strany $\Delta \boldsymbol{f}^i = \boldsymbol{R}_n + \Delta \boldsymbol{R} - \boldsymbol{F}^{i-1}_{n+1}$
B.2.	Výpočet $\delta \boldsymbol{r}^i = \left[\boldsymbol{K}^{i-1}_{t,n+1}\right]^{-1} \Delta \boldsymbol{f}^i$
B.3.	Aktualizace posunů: $\boldsymbol{r}^i_{n+1} = \boldsymbol{r}^{i-1}_{n+1} + \delta \boldsymbol{r}^i$ a $\Delta \boldsymbol{r}^i_{n+1} = \Delta \boldsymbol{r}^{i-1}_{n+1} + \delta \boldsymbol{r}^i$
B.4.	Výpočet skutečných vnitřních sil $\boldsymbol{F}^i_{n+1} = \boldsymbol{F}\left(\boldsymbol{r}^i_{n+1}\right)$
C.	Kontrola konvergence
C.1.	Je-li dosaženo požadované přesnosti, konec iterace
C.2.	Aktualizace a faktorizace matice tuhosti $\boldsymbol{K}^i_{t,n+1}$
C.3.	Jinak jdi na B.1.

je řízeno normou přírustku zatížení. Soustavu rovnic tvoří n-rovnic (2) a omezující podmínka (4). Klasický postup, využívající rozvojů veličin $\boldsymbol{g}(\lambda, \boldsymbol{r})$ a q do Taylorovy řady, vede na nesymetrickou matici linearizované soustavy. My se přidržíme varianty navržené Crisfieldem. Přírůstek vektoru posunutí zapišme ve tvaru (viz obr. 2)

$$\Delta \boldsymbol{r} = \Delta \boldsymbol{r}_0 + \delta \boldsymbol{r}. \tag{5}$$

Pro hledanou úroveň zatížení $\lambda = \lambda_0 + \delta\lambda$ lze pro iterační přírůstek vektoru zatížení psát

$$\begin{aligned} \delta \boldsymbol{r} &= \boldsymbol{K}_t^{-1} \boldsymbol{g}(\boldsymbol{r}_0, \lambda) = \boldsymbol{K}_t^{-1}(\lambda \bar{\boldsymbol{R}} - \boldsymbol{F}(\boldsymbol{r}_0)) = \boldsymbol{K}_t^{-1}(\boldsymbol{g}(\boldsymbol{r}_0, \lambda) + \delta\lambda \bar{\boldsymbol{R}}) \\ &= \boldsymbol{K}_t^{-1} \boldsymbol{g}(\boldsymbol{r}_0, \lambda) + \delta\lambda \boldsymbol{K}_t^{-1} \bar{\boldsymbol{R}} = \delta \bar{\boldsymbol{r}} + \delta\lambda \delta \bar{\boldsymbol{r}}_t. \end{aligned} \tag{6}$$

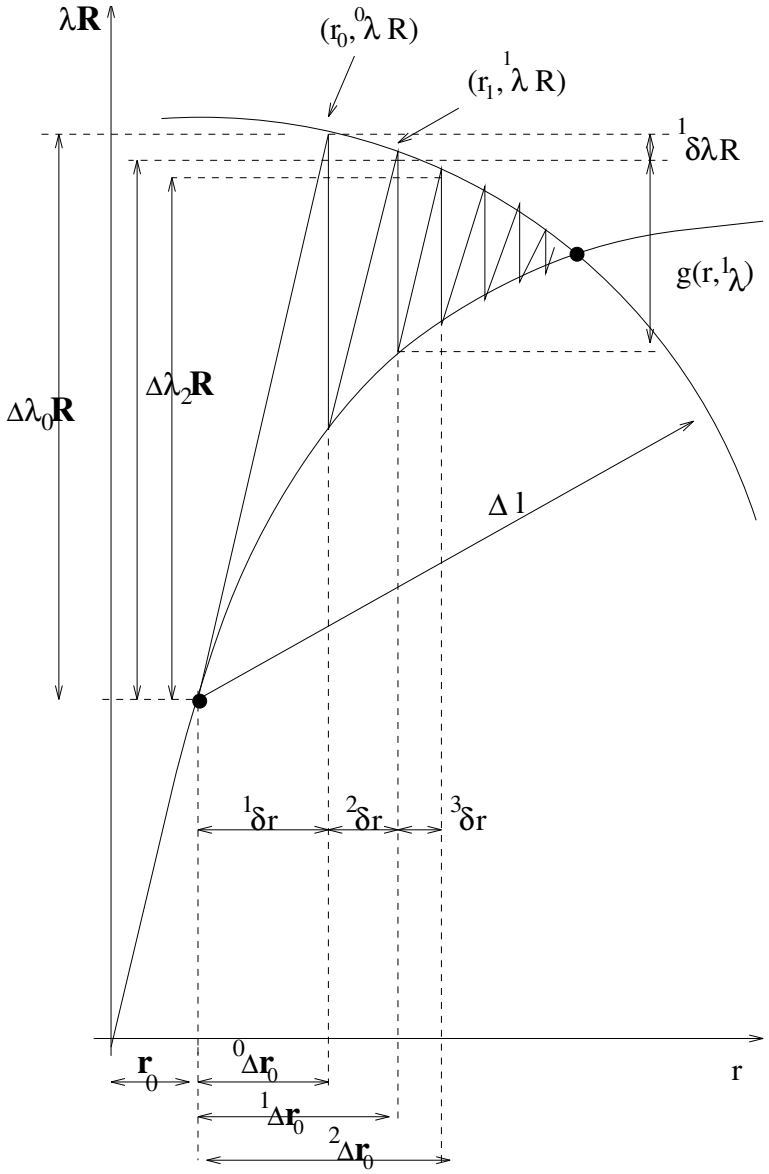
K určení neznámého parametru $\delta\lambda$ v rovnici (6) použijeme omezující podmínku (4). Dosazením rovnice (5) a užitím vztahu $\Delta\lambda = \Delta\lambda_0 + \delta\lambda$ postupně obdržíme

$$\begin{aligned} & \left(\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}} + \delta\lambda \delta \bar{\boldsymbol{r}}_t\right)^T \left(\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}} + \delta\lambda \delta \bar{\boldsymbol{r}}_t\right) \\ + \quad & \left(\Delta\lambda_0 + \delta\lambda\right)^2 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}} - \Delta l^2 = 0, \end{aligned} \tag{7}$$

$$\begin{aligned} & \delta\lambda^2 \left(\delta \bar{\boldsymbol{r}}_t{}^T \delta \bar{\boldsymbol{r}}_t + \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}} \right) + \\ & \delta\lambda \left(2 \left(\delta \bar{\boldsymbol{r}}_t^T \Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}}_t^T \delta \bar{\boldsymbol{r}} \right) + 2 \Delta\lambda_0 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}} \right) + \\ & \left(\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}}\right)^T \left(\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}}\right) - \Delta l^2 + \Delta\lambda_0^2 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}} = 0 \, . \end{aligned} \tag{8}$$

Tuto kvadratickou rovnici pro neznámou $\delta\lambda$ lze formálně zapsat ve tvaru

$$a_1 \delta\lambda^2 + a_2 \delta\lambda + a_3 = 0, \tag{9}$$



Obrázek 2: Metoda délky oblouku.

kde pro a_1, a_2 a a_3 je použito označení

$$a_1 = \delta\bar{\boldsymbol{r}}_t{}^T \delta\bar{\boldsymbol{r}}_t + \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}, \tag{10}$$

$$a_2 = 2\left(\delta\bar{\boldsymbol{r}}_t^T \Delta\boldsymbol{r}_0 + \delta\bar{\boldsymbol{r}}_t^T \delta\bar{\boldsymbol{r}}\right) + 2\Delta\lambda_0 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}, \tag{11}$$

$$a_3 = \left(\Delta\boldsymbol{r}_0 + \delta\bar{\boldsymbol{r}}\right)^T \left(\Delta\boldsymbol{r}_0 + \delta\bar{\boldsymbol{r}}\right) - \Delta l^2 + \Delta\lambda_0^2 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}. \tag{12}$$

Při řešení mohou nastat dva případy

- Oba kořeny jsou reálné. V tomto případě získáme dva přírůstky posunů $\Delta\boldsymbol{r}_1, \Delta\boldsymbol{r}_2$. Existují v podstatě dvě varianty pro výběr správného přírůstku a to:
  - Za správný se považuje ten, který svírá menší úhel s předcházejícím přírůstkem

Tabulka 2: Algoritmus ALM.

A.	Inicializace $i = 0$
A.1.	Vypočtěme $\delta \boldsymbol{r}_t^0$ z rovnice $\boldsymbol{K}_t^0 \delta \boldsymbol{r}_t^0 = \bar{\boldsymbol{R}}$
A.2.	Vypočteme λ_0 s využitím $\Delta\lambda_0 = \pm \dfrac{\Delta l}{\sqrt{\delta \boldsymbol{r}_t^T \delta \boldsymbol{r}_t + \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}}}$ Volba znaménka závisí na sledované větvi zatěžovací dráhy. Znaménko lze volit např. jako $sign(b)$, kde $b = \left(\delta \boldsymbol{r}_t^0\right)^T \bar{\boldsymbol{R}}$ je nenormovaný Berganův parametr tuhosti
A.3.	Výpočet $\Delta \boldsymbol{r}_0^0$ z rovnice $\boldsymbol{K}_t^0 \Delta \boldsymbol{r}_0 = \Delta\lambda_0 \bar{\boldsymbol{R}} = \lambda^0 \bar{\boldsymbol{R}} - \boldsymbol{F}^0$
B.	Pro i=1,2,.... (iterace)
B.1.	Výpočet $\delta \boldsymbol{r}_t^i$ z rovnice $\boldsymbol{K}_t^{i-1} \delta \boldsymbol{r}_t^i = \bar{\boldsymbol{R}}$
B.2.	Výpočet $\delta \bar{\boldsymbol{r}}$ z rovnice $\boldsymbol{K}_t^{i-1} \delta \bar{\boldsymbol{r}} = \lambda^{i-1} \bar{\boldsymbol{R}} - \boldsymbol{F}^{i-1}$
B.3.	Výpočet $\delta\lambda^i$, např. z rovnice $a_1 \delta\lambda^2 + a_2 \delta\lambda + a_3 = 0$
B.4.	$\delta \boldsymbol{r}^i = \delta \bar{\boldsymbol{r}} + \delta\lambda^i \delta \boldsymbol{r}_t^i$
B.5.	$\Delta \boldsymbol{r}_0^i = \Delta \boldsymbol{r}_0^{i-1} + \delta \boldsymbol{r}^i$
B.6.	$\lambda^i = \lambda^{i-1} + \delta\lambda^{i-1}$
C.	Kontrola konvergence
C.1.	Je-li dosaženo požadované přesnosti, konec iterace
C.2.	Jinak jdi na B.1.

$\Delta \boldsymbol{r}_0$. Pro oba možné vektory se spočítá cosinus tohoto úhlu, pro nějž platí

$$\begin{aligned} \cos\vartheta &= \frac{\Delta \boldsymbol{r}_0^T \Delta \boldsymbol{r}}{\Delta l^2} = \frac{\Delta \boldsymbol{r}_0^T (\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}})}{\Delta l^2} + \delta\lambda \frac{\Delta \boldsymbol{r}_0^T \delta \boldsymbol{r}_t}{\Delta l^2} \\ &= \frac{a_4 + a_5 \delta\lambda}{\Delta l^2}, \end{aligned} \tag{13}$$

kde $a_4 = \Delta \boldsymbol{r}_0^T (\Delta \boldsymbol{r}_0 + \delta \bar{\boldsymbol{r}})$ a $a_5 = \Delta \boldsymbol{r}_0^T \delta \boldsymbol{r}_t$. Vybereme ten kořen, kterému přísluší větší $\cos\vartheta$.

– Za lepší $\delta\lambda$ považujeme to, které se blíže přimyká lineárnímu řešení rovnice (9), které je

$$\delta\lambda_{lin} = -\frac{a_3}{a_2}. \tag{14}$$

- Oba kořeny jsou imaginární. V tomto případě je třeba iteraci nastartovat znovu s menší délkou kroku.

Výpočtu kvadratické rovnice a problémům s výběrem kořenů se lze vyhnout. Riks a Wempner vyšli z linearizované podmínky (4), kterou lze psát ve tvaru

$$\delta \boldsymbol{r}^T \Delta \boldsymbol{r}_0 + \delta\lambda (\Delta\lambda_0 \psi^2 \bar{\boldsymbol{R}}^T \bar{\boldsymbol{R}}) = 0, \tag{15}$$

která vede na lineární rovnici pro výpočet $\delta\lambda$. V této práci je použita kvadratická omezující podmínka (4). Celkově můžeme algoritmus metody shrnout do tabulky 2.

2 Popis softwarového návrhu

Modul XALM můžeme propojit s každým softwarovým balíkem založeným na řešení pomocí MKP, jehož algoritmus vede na rovnici 1. Může se jednat o analýzu konstrukcí namáhaných nejen mechanicky, ale i například teplotou.

Pro komunikaci mezi řešičem a modulem XALM je potřeba implementovat rozhraní, které se bude starat o potřebnou výměnu dat. Nejefektivnější způsob propojení je implementace rozhraní přímo do kódu MKP řešiče a přilinkování knihovny XALM. Pro ukázkovou implementaci byl vytvořen jednoúčelový kód pro analýzu jedoduchého vzpěradla na kterém je si ověřit fungování modulu a rozhraní bez nutnosti linkování modulu s konkrétním MKP řešičem. Tento kód je distribuován spolu se zdrojovými kódy modulu na stránkách projektu. Pro větší příklady bylo rozhraní a knihovna ověřeny ve spojení s konečněprvkovým balíkem OOFEM.

Celá výkonná část knihovny XALM, tj. provádění jednotlivých přitěžovacích kroků a hledání rovnováhy, se nachází v jedné třídě jménem XALM. Ta je doplněna souborem obecných funkcí, které poskytují základní matematickou a programovou funkcionalitu. Metody (členské funkce) i atributy (členské proměnné) třídy XALM se dělí do tří funkčních bloků, které se shodují s rozdělením podle typu přístupových práv. Jsou to metody a atributy veřejné, chráněné a soukromé uvedené klíčovými slovy public, protected a private. Datové rozhraní mezi MKP řešičem a modulem XALM je prakticky tvořeno metodami chráněnými (vstup dat) a veřejnými (výstup dat).

Pro každou implementaci modulu do MKP solveru je třeba vytvořit zděděnou třídu (v případě ukázkového kódu vzpěradla je to třída XALM_interface), která tvoří tak fakticky "rozhraní"mezi solverem a modulem.

Rozepsání jednotlivých kroků využití modulu:
1. ve zdeděné třídě jsou pomocí chráněných atributů nastaveny proměnné řídící výpočet;
2. v třídě XALM je zavolána veřejná funkce *solve*;
3. výsledky, tj. pole posunů v uzlech je získáno pomocí veřejné funkce *give_totalDisplacement*.

3 Dokumentace knihovny XALM a specifikace datového rozhraní

V této kapitole je uveden popis atributů a metod třídy XALM.

3.1 Chráněné atributy a metody

Chráněné atributy a metody slouží pro zadávání počátečních požadavků na výpočet a pro předávání potřebných dat mezi MKP programem a knihovnou XALM během vlastního iteračního výpočtu.

3.1.1 Chráněné atributy - volené

První skupina základních proměnných je volená *uživatelem* a řídí běh ALM metody. Tyto atributy se musí nastavit na začátku výpočtu ve funkci **`initialize`** ve zděděné třídě rozhraní.

`xalm_NR_ModeType` **`xalm_NR_Mode`** ... Proměnná určující strategii, kdy se bude počítat tečná matice tuhosti soustavy (převzato z Newton-Raphson metody iteračního výpočtu).

`int` **`xalm_MANRMSteps`** ... Počet kroků, po kterých se má znovu počítat matice, pouze pokud xalm_NR_Mode == xalm_accelNRM.

`long` **`nsteps`** ... Počet zatěžovacích kroků.

`double` **`minStepLength`** ... Minimální délka kroku.

`double` **`maxStepLength`** ... Maximální délka kroku.

`double` **`initialStepLength`** ... Počáteční délka kroku.

`int` **`nsMin`** ... Minimální počet kroků dorovnání nerovnováhy v jednom iteračním přitěžovacím kroku.

`int` **`nsReq`** ... Požadovaný počet kroků dorovnání nerovnováhy v jednom iteračním přitěžovacím kroku.

`int` **`nsMax`** ... Maximální počet kroků dorovnání nerovnováhy v jednom iteračním přitěžovacím kroku.

`double` **`Psi`** ... Parametr kontroly kroku. Pokud je rovno 0, tak se jedná o kontrolu přírůstkem posunutí, pokud je rovno nekonečnu, tak se jedná o kontrolu přírůstkem zatížení.

`int` **`verbose`** ... Pokud je větší než nule - budou vypisovány informace.

`double` **`rtolf`** ... Tolerance relativní chyby nevyrovnaných sil.

`double` **`rtold`** ... Tolerance relativní chyby nevyrovnaných posunů.

Proměnná **`xalm_NR_Mode`** může nabývat následujících hodnot:
`xalm_modifiedNRM` - Modifikovaná NR metoda (defaultní hodnota) - matice se počítá jen na začátku každého zatěžovacího kroku.
`xalm_fullNRM` - Plná NR metoda - matice se během zatěžovacího kroku počítá při každé iteraci.
`xalm_accelNRM` - Akcelerovaná NR metoda - matice se počítá jen na začátku každého n-tého zatěžovacího kroku, n si určí uživatel.

3.1.2 Chráněné atributy - nevolené

Druhá skupina základních proměnných se nevolí, ale závisí na zadání právě počítané úlohy. Tyto atributy se musí nastavit na začátku výpočtu ve funkci **`initialize`** ve zděděné třídě rozhraní.

`long` **`neq`** ... Počet rovnic v matici soustavy = počet neznámých.

`Dvctr` **`incrementalLoadVector`** ... Vektor přírůstkového zatížení, mění se se stupněm lambda.

`Dvctr` **`initialLoadVector`** ... Vektor počátečního zatížení, nemění se během výpočtu, je aplikován celý.

3.1.3 Chráněné metody

Metody pro přístup k aktuálním stavovým hodnotám ALM výpočtu.

`int` **`give_step`** `(void)`

Funkce vrací číslo aktuálního zatěžovacího kroku.

3.1.4 Chráněné metody - pure

Metody v třídě XALM deklarovány jako čistě virtuální (pure virtual) jsou implementačně závislé a jsou definovány až ve zděděné třídě.

`void` **`initialize`** `(void)`

Funkce nemá parametry a nic nevrací. Má za úkol inicializovat chráněné atributy.

`void` **`update_step`** `(void)`

Funkce není povinná. Ve funkci je možné po každém provedeném zatěžovacím kroku vykonat požadované úkony. Např. vypsat aktuání hodnoty, zastavit iteraci, pokud je dosaženo požadovaných výsledků, atd.

`void` **`update_stiffness_matrix`** `(const Dvctr` **`*X`**`)`

Funkce zajistí znovusestavení matice tuhosti soustavy pro daný vektor uzlových posunutí. Matice tuhosti soustavy se během výpočtu do modulu XALM nezasílá, proto i rozhodnutí zda a jak se bude matice během volání této funkce sestavovat (počáteční vs. tečná) zůstává na rozhodnutí MKP programu.

`X` ... Vektor celkových uzlových posunutí.

`void` **`update_internal_forces`** `(Dvctr` **`*F`**`, const Dvctr` **`*X`**`)`

Funkce vypočte vektor vnitřních sil pro daný vektor uzlových posunutí.

`F` ... Vektor uzlových vnitřních sil.

`X` ... Vektor celkových uzlových posunutí.

`void` **`lineq_solve`** `(Dvctr` **`*X`**`, const Dvctr` **`*R`**`)`

Funkce vypočte vektor uzlových posunutí pro daný vektor pravé strany.

`X` ... Vektor uzlových vnitřních sil.

`R` ... Vektor celkových uzlových posunutí.

3.2 Veřejné metody

Veřejné metody slouží pro spuštění výpočtu a pro předávání výsledků.

`XALM` `(void)`

Konstruktor třídy XALM je volaný bez parametrů.

`int` **`solve`** `(void)`

Hlavní a jediná výkonná funkce knihovny XALM. Zadané zatížení je aplikováno v požadovaném počtu kroků a výsledkem je vektor uzlových posunů.

`const double*` **`give_totalDisplacement`** `(void) const`

Funkce vrátí konstantní ukazatel na vektor celkových uzlových posunů.

`double` **`give_loadLevel`** `(void) const`

Funkce vrátí loadLevel - dosažený stupeň přírůstkového zatížení.

3.3 Soukromé atributy a metody

Soukromé atributy a metody nejsou přístupné pro používání z vnějšku třídy ani z třídy zděděné a slouží pro vlastní výpočet.

Reference

[1] Z. Bittnar, J. Šejnoha, Numerické metody mechaniky II, Vydavatelství ČVUT, 1992.

[2] K. Bathe, Finite element procedures in engineering analysis, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1982.