TECHNICKÁ UNIVERZITA V LIBERCI

Aplikace pro aproximaci vlastností dynamických systémů obrazovým přenosem

bakalářská práce

Fakulta: Strojní

Studijní obor: Výrobní systémy

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60(školní dílo) a § 35 (o nevýdělečném užití díla k vnitřní potřebě školy).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé práce a prohlašuji, že souhlasím s případným užitím mé práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury pod vedením vedoucího bakalářské práce.

Datum:

Podpis:

Poděkování

Touto cestou bych rád poděkoval vedoucímu bakalářské práce Ing. Michalu Moučkovi Ph.D. za ochotu a rady při řešení této práce.

Abstrakt

Cílem bakalářské práce je naprogramování aplikace pro aproximaci vlastností systému obrazovým přenosem z naměřených přechodových charakteristik. Úkol byl vypracován užitím programovacího jazyka C a aplikačního programovacího rozhraní Win32Api.

V teoretické části bakalářské práce jsou popsány numerické metody podílející se na chodu aplikace. Konkrétně pak metoda Runge–Kutta, Nelder-Mead, nejmenších čtverců a další s těmito metodami související informace. Zmíněn je také popis dynamických vlastností systému lineární diferenciální rovnicí, přenos systému a přechodová charakteristika.

V realizační části bakalářské práce jsou popsány algoritmy, založené na metodách Runge-Kutta a Nelder-Mead, psané v C . Představen je zde také chod aplikace.

Klíčová slova: programování, C, Runge-Kutta, Nelder-Mead, aplikace, aproximace funkce, optimalizace funkce, dynamické vlastnosti systému, obrazový přenos, přechodová charakteristika

Abstract

The purpose of this thesis is to program an application for approximation characteristics of the system with visual transmition from measured transitional characteristics. The task was worked out using the C programming language and Win32Api aplication programming interface.

The theoretical part of this work describes numerical methods which took part in the aplication. To be more specific there is text about Runge-Kutta, Nelder-Mead , least squares method and additional informations connected with these methods. There is also mention about the description of dynamical characteristics of the system with linear diferential equation, transmition of the system and transitional characteristic.

In the practial part of this thesis there is description of algorithms, based on Runge-Kutta and Nelder-Mead methods, written in C. There is also introduction to the functionality of an aplication.

Key words: programming, C, Runge-Kutta, Nelder-Mead, application, approximation of the function, optimalization of the function, dynamical characteristics of the system, visual transmition, transitional characteristic

Obsah

0 SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 9

1 ÚVOD 11

Teoretická část

2 POPIS DYNAMICKÝCH VLASTNOSTÍ SYSTÉU 13

2.1 Popis systému lineární diferenciální rovnicí 13

2.2 Přechodová charakteristika 13

2.3 Obrazový přenos systému 14

3 NUMERICKÉ METODY ŘEŠENÍ LINEÁRNÍCH DIFERENCIÁLNÍCH ROVNIC 16

3.1 Volba vhodné metody pro řešení LDR 16

3.2 Teorie numerického výpočtu ODR vyššího řádu 17

3.2.1 Počáteční úloha 17

3.2.2 Eulerova metoda 18

3.2.3 Geometrické vyjádření Eulerovy metody 18

3.2.4 Modifikace Eulerovy metody 19

3.2.5 Runge –Kutta 21

3.2.6 ODR vyššího řádu 22

3.2.7 Soustavy ODR 22

3.3 Dormand – Prince 23

3.3.1 Optimální krok 24

3.3.2 Konvergence metody 24

4 APROXIMACE FUNKCE 26

4.1 Volba vhodné metody pro aproximaci 26

4.2 Metoda nejmenších čtverců 26

4.2.1 Aproximace přímkou 26

5 OPTIMALIZACE FUNKCE 28

5.1 Volba vodné metody pro optimalizaci 28

5.2 Nelder –Mead 29

5.2.1 Parametry 29

5.2.2 Základní princip 29
5.3 Příklad iteračního cyklu NM 29
5.3.1 Úvodní body 30
5.3.2 Řazení 30
5.3.3 Bod odrazu 31
5.3.4 Bod prodloužení 32
5.3.5 Bod zkrácení 32
5.3.6 Zmenšení 33
Realizační část
6 REALIZACE RKDP 35
6.1 Zjednodušený vývojový diagram pro RKDP 35
6.2 Algoritmus pro RKDP vyššího řádu 35
6.3 Optimální krok 38
6.4 Úprava optimálního kroku 39
6.5 Podstatný čas 40
6.6 Shrnutí algoritmu RK 41
7 REALIZACE NM 42
7.1 Zjednodušený vývojový diagram pro NM 42
7.2 Výpočet funkční hodnoty 42
7.3 Iterační cyklus NM – realizace 43
8 UKÁZKA CHODU APLIKACE 49
8.1 Načtení dat ze souboru 49
8.2 Vstup 51
8.3 Výstup 52
9 ZÁVĚR 54
SEZNAM POUŽITÉ LITERATURY 55
SEZNAM OBRÁZKŮ 57
PŘÍLOHA 58

0 SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

$a_0,\ a_1,\ a_n,\ b_0,\ b_1,\ b_m$	- koeficienty rovnice
c_0, c_1	- koeficienty popisující přímku
d	- hrana polyedru
e_i	- chyba aproximace v *i*-tém kroku
$f_1,\ f_p,\ f_{p+1},$	- minimální, druhá největší a největší funkční hodnota
$G(s)$	- Obrazový přenos systému
h	- krok
h_{opt}	- doporučený optimální krok
$k_i,\ k_1,\ k_2,\ k_3,\ k_4,\ k_5,\ k_6,\ k_7$	- konstanty Runge-Kutta
K	- statické zesílení systému
$l_1,\ l_2$	- souřadnice bodů polyedru v matici
n	- počet proměnných(u Nelder-Mead)
R^n	- Euklidův *n*-rozměrný prostor
s	- koeficient pro úpravu optimálního kroku
S	- matice symbolizující souřadnice a body polyedru
T_1, T_2, T_n	- časové konstanty
$u(t)$	- vstupní veličina
$U(s)$	- Laplaceův obraz vstupní veličiny
v_i	- nové body po zmenšení polyedru
x_i	- souřadnice při *i*-tém kroku, bod
x_k	- koncový bod
$x_1,\ x_n,\ x_{n+1},$	- bod s minimální, druhou největší a nej. funkční hodotou
$x_L,\ x_T,\ x_H,$	- bod s minimální, druhou největší a nej. funkční hodotou
x_{n+2}	- bod odrazu polyedru

x_{n+3}	- těžiště polyedru
x_{n+4}	- bod prodloužení polyedru
x_{n+5V}	- bod vnějšího zkrácení polyedru
x_{n+5VV}	- bod vnitřního zkrácení polyedru
$y(t)$	- výstupní veličina
y_n, y_i	- výsledek *n*-tého kroku, výsledek *i*-tého kroku
y_{si}	- naměřená hodnota ze souboru v *i*-tém kroku
y_{n+1}	- výsledek *n + 1*-tého kroku
$Y(s)$	- Laplaceův obraz výstupní veličiny
z_{n+1}	- výsledek při *n+1*-tém kroku u Runge-Kutta 5. Řádu
$\alpha_i, \beta_{ij}, \omega_i$	- volitelné konstanty Eulerovy modifikované metody
γ	- parametr pro zkrácení polyedru
$\Delta_k,\ \Delta_{k+1}$	- polyedr(obrazec) při *k*-té, *k+1*-té iteraci
ε	- dovolená chyba při výpočtu optimálního kroku
η	- hodnota počáteční podmínky
ϑ	- parametr pro odraz polyedruSem zadejte rovnici.
ρ	- kvadratická odchylka
σ	- parametr pro zmenšení polyedru
χ	- parametr pro prodloužení polyedru
atd.	- a tak dále
LDR	– Lineární diferenciální rovnice
Obr.	- Obrázek, obrázku
ODR	- Obyčejná diferenciální rovnici
NM	- Nelder-Mead
RK	- Runge–Kutta
RKDP	– Runge–Kutta Dormand- Prince

1 ÚVOD

Cílem bakalářské práce bylo vytvořit aplikaci schopnu z naměřených hodnot po zadání určitých vstupních veličin do dynamické soustavy, zjistit parametry obrazového přenosu. Obrazový přenos bude aplikací zjištěn aproximováním naměřených dat, což znamená, že bude hledána funkce, která bude naměřeným hodnotám co nejblíže.

Naměřené hodnoty budou vloženy v požadovaném formátu do souboru, ze kterého budou načteny. Celá úloha byla složena ze tří částí. Tyto části měly provést následující: spočítat lineární diferenciální rovnici n-tého řádu, aproximovat naměřená data a najít minimum funkce. Všechny části se navzájem prolínaly a mým cílem bylo naprogramovat aplikaci, která by tuto úlohu s nalezením aproximace naměřených dat vyřešila.

Následující text, který popisuje postup řešení zadání, byl rozdělen do dvou částí. Část teoretickou a část realizační. V první části byla popsána teorie a v druhé, realizační části, jsou popsány některé úseky zdrojového kódu psané v programovacím jazyku C. Celý zdrojový kód má přes 1700 řádků a velmi se prolíná, proto bylo upřednostněno zdůraznění jen některých stěžejních částí.

V posledních kapitolách pak je představena funkčnost aplikace při aproximování dat ze vzorového souboru s naměřenými hodnotami, čímž je umožněno čtenáři si lépe aplikaci a její chod představit.

Teoretická část

2 POPIS DYNAMICKÝCH VLASTNOSTÍ SYSTÉU

Systém může být popsán zvnějšku nebo zevnitř. Rozebrán bude pouze popis vnější.

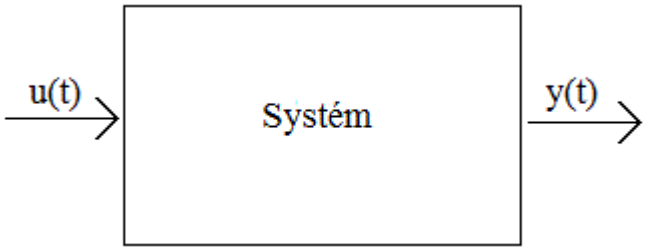
Vnější popis systému vyjadřuje dynamické vlastnosti reakcí systému mezi vstupem a výstupem. Dále bude předpokládán lineární systém. Systém může být popsán následujícími tvary: lineární diferenciální rovnicí, přechodovou funkcí a přechodovou charakteristikou, impulzovou funkcí a impulzovou charakteristikou, frekvenčním přenosem a frekvenční charakteristikou, polohou nul a pólů přenosu [10].

2.1 Popis systému lineární diferenciální rovnicí

Je uvažován systém s jedním vstupem a jedním výstupem znázorněný obdélníkovým blokem, ve kterém jsou soustředěny jeho dynamické vlastnosti (obr. 1). Chování takovéhoto spojitého systému je popsáno lineární diferenciální rovnicí s konstantními koeficienty jako v [10]:

$$a_n \cdot y^{(n)}(t) + \ ... + a_1 \cdot y'(t) + \ a_0 \ \cdot y(t) = \ b_k \cdot u^{(k)}(t) + \ ... + \ b_1 \cdot u'(t) + b_0 \cdot u(t). \qquad (1)$$

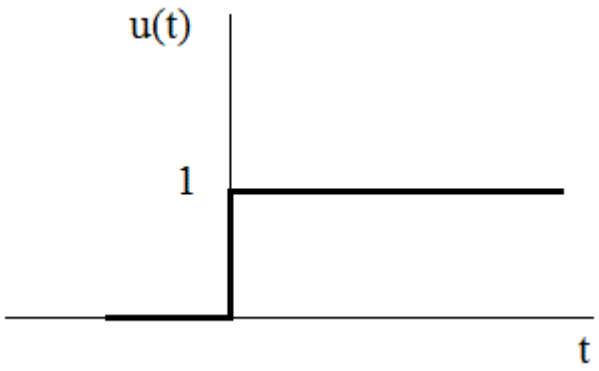
V rovnici (1) je $y(t)$ výstupní veličina a $u(t)$ je vstupní veličina. Tato lineární diferenciální rovnice musí splňovat podmínku fyzikální realizovatelnosti tím, že $k \leq n$. Řád diferenciální rovnice n určuje řád systému.



Obr. 1: Systém se vstupem a výstupem

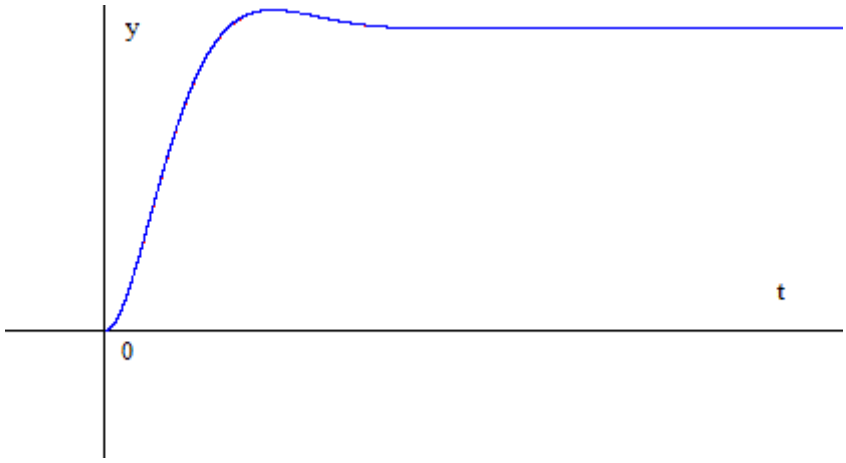
2.2 Přechodová charakteristika

Přechodová charakteristika je jedním z prostředků ke zjišťování dynamických vlastností soustavy. Jde o grafické znázornění přechodové funkce. Přechodová funkce je reakcí na jednotkový (Heavisideův) skok (obr. 2) při nulových počátečních podmínkách systému.



Obr. 2: Jednotkový skok

Souřadnice přechodové charakteristiky jsou zjišťovány odezvou $y(t)$ na vstupní skok $u(t)$o známé velikosti. Odezva na skok se dělí skokem, čímž vzniká přechodová charakteristika. Příklad přechodové charakteristiky je znázorněn na obr. 3.



Obr. 3: Ukázka přechodové charakteristiky

2.3 Obrazový přenos systému

Obrazový přenos(dále přenos) systému je poměr Laplaceova obrazu výstupní veličiny k Laplaceovu obrazu vstupní veličiny při nulových počátečních podmínkách systému a vstupního signálu. Laplaceova transformace je jednou ze základních integrálních transformací. Je používána k převodu, který zjednodušuje složité vztahy. Byla odvozena Pierrem Simonem de Laplacem. Použitím Laplaceovy transformace a splněním výše uvedených podmínek lze jako v [10] psát:

$$[a_n \cdot s^n + a_{n-1} \cdot s^{n-1} \ldots + a_1 \cdot s + a_0] \cdot Y(s) = [b_k \cdot s^k + \ldots + b_1 \cdot s + b_0] \cdot U(s). \quad (2)$$

Přenos systému má pak tvar jako v [10]:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_k \cdot s^k + \ \ldots + \ b_1 \cdot s + b_0}{a_n \cdot s^n + \cdots + a_1 \cdot s + \ a_0}. \tag{3}$$

Z důvodu toho, že je dosazován jednotkový skok, čili známá konstanta se tvar přenosu v našem případě zjednoduší na:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0}{a_n \cdot s^n + \cdots + a_1 \cdot s + \ a_0}. \tag{4}$$

Jinak může být zapsáno ve tvaru s časovými konstantami jako ve [11]:

$$G(s) = K \cdot \frac{1}{(T_1 \cdot s + 1) \cdot (T_2 \cdot s + 1) \ldots (T_n \cdot s + 1)}. \tag{5}$$

V rovnici (4) je konstanta zesílení systému $K = \frac{b_0}{a_0}$ a časové konstanty $T_1 = \frac{a_1}{a_0}$ $T_2 = \frac{a_2}{a_0}$ až $T_n = \frac{a_n}{a_0}$. V realizované aplikaci budou konstanty vypsány jako výsledek.

3 NUMERICKÉ METODY ŘEŠENÍ LINEÁRNÍCH DIFERENCIÁLNÍCH ROVNIC

Jak bylo zmíněno v úvodu, tak se úloha skládala ze tří částí. Tato kapitola se bude zabývat částí první, která se zabývá řešením lineárních diferenciálních rovnic (LDR) vyššího řádu. Vzhledem k tomu, že jde o počítačovou aplikaci tak zde bude pojednáno o numerických metodách výpočtu.

Numerické metody slouží k získávání přibližných výsledků zatížených chybou. Než se dostaneme k samotnému řešení, musíme reálnou fyzikální situaci převést na matematický model. Již samotná měření fyzikálních veličin jsou zatížena nepřesnostmi. Dále dochází ke zjednodušení úlohy matematické numerickou úlohou. Často se jedná o náhradu procesu nekonečného procesem konečným.

Chyby také vznikají zaokrouhlováním v průběhu výpočtu. Při velkém počtu početních operací se kumulují anebo naopak navzájem ruší.

Diferenciální rovnice je nástrojem právě k popisu reálných dějů. V této kapitole budeme seznámeni s tím jak tuto úlohu řešit.

3.1 Volba vhodné metody pro řešení LDR

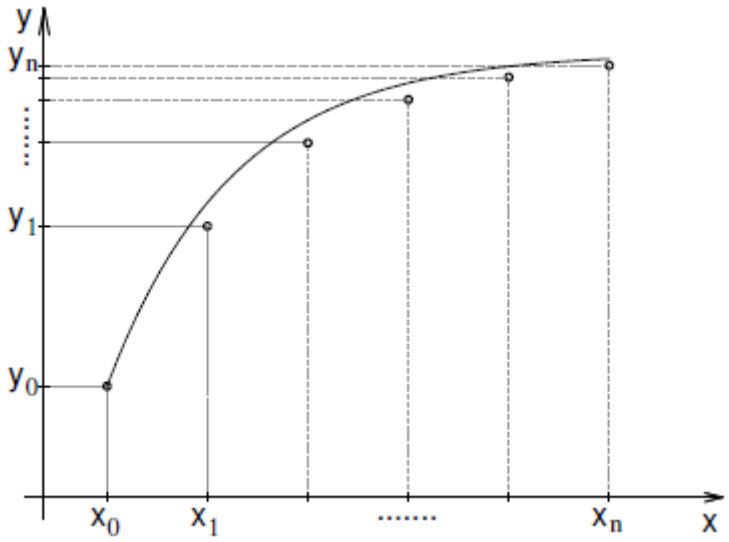
Zadané úvodní podmínky byly určující k tomu, jak se k problému přistupovalo. Úvodní podmínky jsou rozděleny do dvou skupin a to na počáteční a okrajové podmínky.

Pro počáteční podmínky je typické, že jsou známa $y(x)$ při určité počáteční hodnotě x a je požadováno najít $y(x_k)$ v určitém konečném bodě x_k anebo $y(x_i)$ na nějakém diskrétním seznamu bodů x_i pro $i = 1 \text{ až } n$. Tento typ úlohy odpovídá tomu co bylo v aplikaci řešeno. Podmínkami okrajovými se tedy nebudeme v textu zabývat.

Dále bylo potřeba zvolit vhodnou numerickou metodu. V knize [1] byly popsány následující tři: Runge–Kutta, Richardsonova extrapolace a Prediktor–korektor. Z těch byla vybrána metoda Runge–Kutta a to proto, že není složitá a přesto má dostačující přesnost $\leq 10^{-5}$. Obvykle není ze zde vyjmenovaných metod nejrychlejší, ale na úlohu, kterou budeme řešit její efektivita vystačí. Více v [1]. V další podkapitole bude úloha řešení LDR trošku zobecněna a to na obyčejné diferenciální rovnice (ODR).

3.2 Teorie numerického výpočtu ODR vyššího řádu

Společným znakem všech dále uvedených metod v této kapitole je to, že řešení není hledáno jako spojitá funkce definovaná na celém zkoumaném intervalu $< a, b >$, ale pouze v konečném počtu bodů $a = x_0 < x_1 < \cdots < x_n = b$. Tyto body se nazývají uzlové body nebo uzly sítě a množině $\{x_0, x_1, \ldots, x_n\}$ říkáme síť. Rozdíl $h_i = x_{i+1} - x_i$ se nazývá krok sítě v uzlu x_i . Přibližné hodnoty řešení v uzlových bodech budou značeny y_0, y_1 až y_n a hodnoty přesné budou značeny $y(x_0), y(x_1)$ až $y(x_n)$.



Obr. 4: Přesné a přibližné řešení [2]

3.2.1 Počáteční úloha

Na vzorovém příkladu si v následujících podkapitolách ukážeme jak z Eulerovy metody vznikla zvolená metoda Runge–Kutta(RK).

Příklad:

$$y' = f(x, y)\ ,\ y(x_0) = y_0. \tag{6}$$

Podmínky, které zajistí jednoznačnost řešení této počáteční úlohy jsou vyjádřeny následující větou z [2]: Je-li funkce $f(x, y)$ spojitá na obdelníku $R = \{(x, y; |x - x_0| \leq a, |y - y_0| \leq b\}$. Kde , $a > 0, b > 0$, pak existuje řešení počáteční úlohy na intervalu $< x_0 - \propto, x_0 + \propto >$, pro: $\propto = \min\left(a, \frac{b}{M}\right)$, $M = \max_R |f(x,y)|$. Je–li dále funkce $\frac{\partial f(x,y)}{\partial y}$ ohraničena na obdelníku R, pak je toto řešení jediné.

3.2.2 Eulerova metoda

Mějme dánu počáteční úlohu a pravidelnou síť $\{x_0\,,\ x_1, \dots\,,\ x_n\}$ s krokem h. Ve všech bodech sítě by podle rovnice (6) mělo platit:

$$y'(x_i) = f\big(x_i, y(x_i)\big). \tag{7}$$

Derivaci na levé straně této rovnice můžeme nahradit diferencí:

$$\frac{y(\mathrm{x_{i+1}}) - \mathrm{y(x_i)}}{h} \approx f\left(\mathrm{x_i}, y\big(\mathrm{x_i}\big)\right). \tag{8}$$

Bude-li nahrazeno $y(x_i)$ přibližnou hodnotou y_i, tak odtud může být vyjádřena přibližná hodnota $y(x_{i+1})$jako :

$$y(x_{i+1}) = y_i + h \cdot f\,(x_i, y_i). \tag{9}$$

Pomocí tohoto vzorce bude vypočteno přibližné řešení v dalším uzlovém bodě pomocí hodnoty v uzlu předchozím. Hodnota řešení v předchozím bodě x_0 ja známa z počáteční podmínky (6) a ta je rovna y_0

3.2.3 Geometrické vyjádření Eulerovy metody

Diferenciální rovnicí (6) je dáno tzv. směrové pole. V každém bodě $[x, y]$ roviny (x, y), kterým prochází některé řešení této rovnice, je hodnota $f(x, y)$ rovna směrnici tečny ke grafu tohoto řešení.

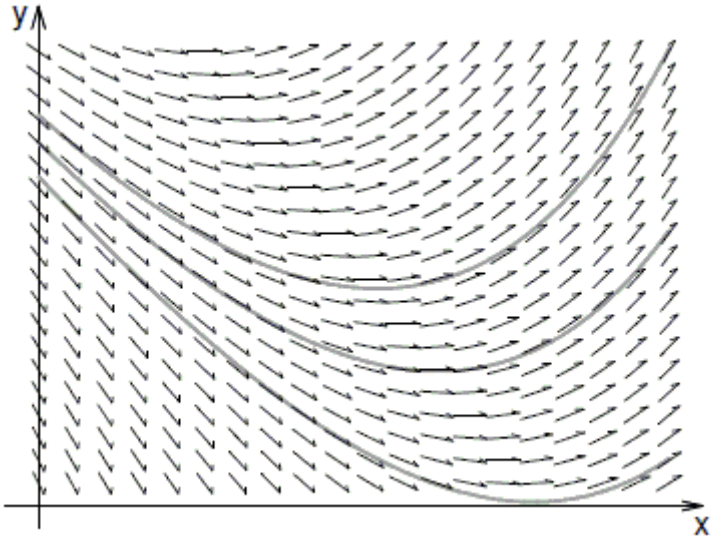
Směrové pole si můžeme představit tak, že v každém bodě roviny (x, y) stojí šipka, která říká, kterým směrem máme pokračovat, dostaneme-li se do tohoto bodu. Při řešení Eulerovou metodou vyjdeme z bodu $[x_0, y_0]$ směrem, který udává směrové pole v tomto bodě. To znamená, že půjdeme po přímce o rovnici:

$$y \;=\; y_0 + f(x_0, y_0) \cdot (x - x_0). \tag{10}$$

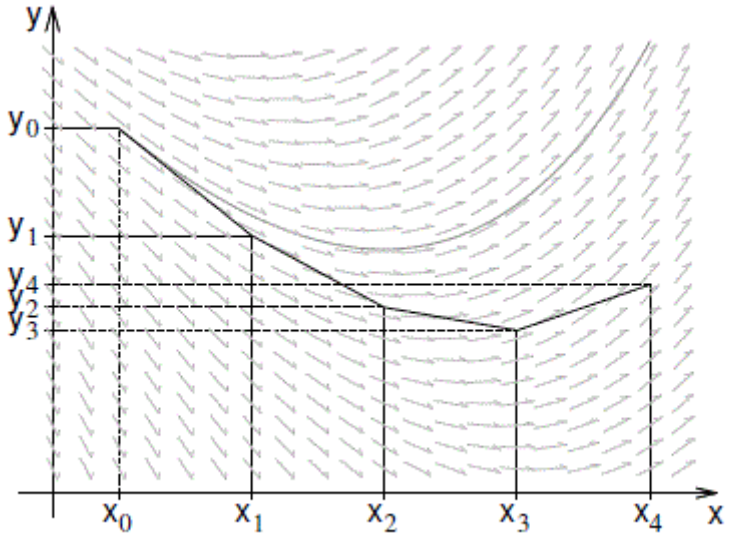
Dokud nebude dosaženo bodu s x-ovou souřadnicí x_1. Ypsilonová souřadnice tohoto bodu tedy je:

$$y_1 \;=\; y_0 + f(x_0, y_0) \cdot (x_1 - x_0) = y_0 + f(x_0, y_0) \cdot h. \tag{11}$$

Z takto získaného bodu $[x_1, y_1]$ můžeme pokračovat dále ve směru daném směrovým polem v tomto bodě.



Obr. 5: Směrové pole [2]



Obr. 6: Přibližné řešení diferenciální rovnice Eulerovou metodou [2]

3.2.4 Modifikace Eulerovy metody

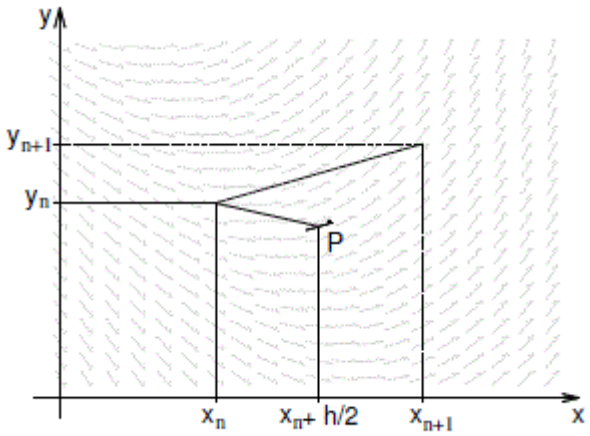
Z Eulerových modifikací bude snáze pochopitelné jak vznikají metody podle RK. Nejprve jsou vypočteny pomocné hodnoty k_1 a k_2 a pomocí nich pak určena přibližná hodnota řešení v dalším uzlovém bodě.

První modifikace jako v [2]:

$$k_1 = f(x_n, y_n), \tag{12}$$

$$k_2 = f\left(x_n + \frac{1}{2}\cdot h, y_n + \frac{1}{2}\cdot h \cdot k_1\right),$$

$$y_{n+1} = y_n + h \cdot k_2.$$



Obr. 7: První modifikace Eulerovy metody [2]

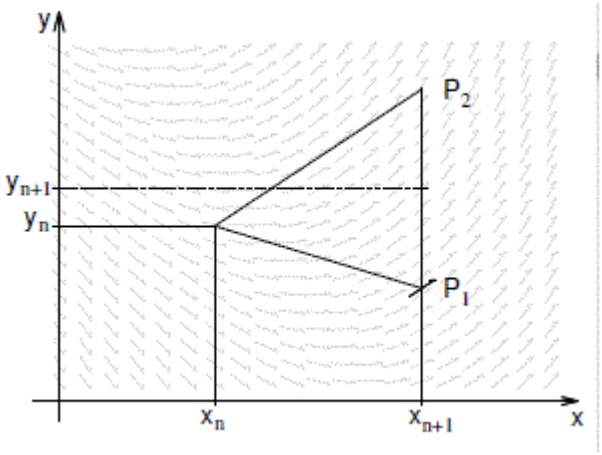
Nejprve je nalezen pomocný bod P, tak že z bodu $[x_n, y_n]$ vyrazíme po přímce se směrnicí $f(x_n, y_n)$, ale dojedeme jen do bodu $x_n + \frac{h}{2}$. Přibližné řešení v bodě x_{n+1} je získáno tím, že z bodu $[x_n, y_n]$ jdeme po přímce se směrnicí určenou směrovým polem v bodě P dokud není dosaženo na souřadnici x_{n+1}.

Druhá modifikace jako v [2]:

$$k_1 = f(x_n, y_n), \tag{13}$$

$$k_2 = f(x_n + h, y_n + h \cdot k_1),$$

$$y_{n+1} = y_n + \frac{1}{2} \cdot h \cdot (k_1 + k_2).$$



Obr. 8: Druhá modifikace Eulerovy metody [2]

Uděláme si dva pomocné body P_1 a P_2. Bod P_1 je dosažen krokem Eulerovy metody. Bod P_2 tím, že z bodu $[x_n, y_n]$ jdeme po přímce se směrnicí danou směrovým polem v bodě P_1 a to až do bodu o souřadnici x_{n+1}. Nový bod $[x_{n+1}, y_{n+1}]$, pak leží ve středu úsečkyP_1P_2.

3.2.5 Runge –Kutta

Modifikované Eulerovy metody jsou v podstatě jednoduchými příklady metod Runge–Kutta . V dalším textu bude název Runge-Kutta psán ve zkratce jako RK. Obecný tvar metody RK je jako v [2] následující:

$$y_{n+1} = y_n + h \cdot (\omega_1 \cdot k_1 + \cdots + \omega_s \cdot k_s). \tag{14}$$

V rovnici (14) se $k_1 = f(x_n, y_n)$ a $k_i = f(x_n + \alpha_i \cdot h, y_n + h \cdot \sum_{j=1}^{i-1} \beta_{ij} \cdot k_j)$ pro $i = 2, \ldots, s$. V téže rovnici jsou ω_i, α_i a β_{ij} volené konstanty, například pro první modifikaci Eulerovy metody (12) by bylo $\omega_1 = 0, \omega_2 = 1, \alpha_2 = \frac{1}{2}$ a $\beta_{21} = \frac{1}{2}$ a u druhé modifikace (13) $\omega_1 = \omega_2 = \frac{1}{2}, \alpha_2 = 1, \beta_{21} = 1$

Neznámější z modifikací je metoda RK 4. řádu jako v [2]:

$$y_{n+1} = y_n + \frac{1}{6} \cdot h \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4), \tag{15}$$

$$k_1 = f(x_n, y_n),$$

$$k_2 = f\left(x_n + \frac{1}{2} \cdot h, y_n + \frac{1}{2} \cdot h \cdot k_1\right),$$

$$k_3 = f\left(x_n + \frac{1}{2} \cdot h, y_n + \frac{1}{2} \cdot h \cdot k_2\right),$$

$$k_4 = f(x_n + h, y_n + h \cdot k_3).$$

3.2.6 ODR vyššího řádu

V této podkapitole bude vysvětleno jak vyřešit soustavu ODR *n*-tého řádu s počátečními podmínkami na následujícím příkladě jako v [2]:

$$y^{(n)} = f\left(x, y, y', \dots, y^{(n-1)}\right), \tag{16}$$

$$y(x_0) = y_0, y'(x_0) = y'_{0,\dots,} y^{(n-1)}(x_0) = y_0^{(n-1)}.$$

Ten může být převeden na soustavu diferenciálních rovnic prvního řádu. Bude-li označeno $y_1 = y$, $y_2 = y'$, …, $y_n = y^{(n-1)}$. Potom zřejmě bude platit, že $y_1' = y_2$, $y_2' = y_3$, $y_2' = y_3$ atd. Rovnici (16) při našem značení lze zapsat takto:

$$y_n' = f(x, y_1, y_2, \dots, y_n). \tag{17}$$

Rovnice (17) představuje soustavu n diferenciálních rovnic prvního řádu:

$$\begin{aligned} y_1' &= y_2, & y_1(x_0) &= y_0, \\ y_2' &= y_3, & y_2(x_0) &= y_0', \\ &\vdots & &\vdots \\ y_n' &= f_n(x, y_1, y_2, \dots, y_n), & y_n(x_0) &= y_0^{(n-1)}. \end{aligned} \tag{18}$$

Rovnicí (18) jsme se dopracovaly až k tomu jak řešit ODR vyššího řádu. Zjednodušeně jde o substituci derivací za proměnné, čímž z ODR *n*-tého řádu získáme soustavu ODR *n*-rovnic řádu prvního.

3.2.7 Soustavy ODR

Zde bude ukázáno jak řešit soustavy ODR s počátečními podmínkami. Bude zmíněno i to jak vše aplikovat na případ metody RK.

Soustava ODR prvního řádu s počátečními podmínkami bude jako v [2] rozepsána takto:

$$
\begin{aligned}
y_1' &= f_1(x, y_1, y_2, \dots, y_n), \qquad & y_1(x_0) &= \eta_1, \\
y_2' &= f_2(x, y_1, y_2, \dots, y_n), \qquad & y_2(x_0) &= \eta_2, \\
&\vdots & &\vdots \\
y_n' &= f_n(x, y_1, y_2, \dots, y_n), \qquad & y_n(x_0) &= \eta_n.
\end{aligned} \tag{19}
$$

Může být přepsáno vektorově jako:

$$
\boldsymbol{y} = \boldsymbol{f}(x, \boldsymbol{y}), \quad \boldsymbol{y}(x_0) = \boldsymbol{\eta}. \tag{20}
$$

V rovnici (17) se $\boldsymbol{y} = (y_1, \dots, y_n)^T$, $\boldsymbol{f} = (f_1, \dots, f_n)^T$ a $\boldsymbol{\eta} = (\eta_1, \dots, \eta_n)^T$.

Pro Eulerovu metodu tedy:

$$
\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \cdot \boldsymbol{f}(x_n, \boldsymbol{y}_n). \tag{21}
$$

V případě, že řešíme RK4 n-tého řádu, tak tato formule bude rozepsána stejně jako (20), (21). Konkrétně pak budou vektorově rozepsány konstanty k_1, k_2, k_3 a k_4 a bude více proměnných. Vše je podrobněji ukázáno v textu [2]. Stejně tak to bude moci být aplikováno i na v následujícím textu popisovanou metodu RK Dormand – Prince.

3.3 Dormand – Prince

V této podkapitole bude popsána metoda Dormand–Prince. Dále v textu jako RKDP. Ta je použita v aplikaci a jde o metodu založenou na RK 4.a 5. řádu. Ve volbě této konkrétní metody, která může být nalezena přímo v kódu aplikace, napomohl fakt, že je použita v současném MATLABU jako funkce *ode45* . Výpočet konstant vypadá následovně jako v [3]:

$$
k_1 = h \cdot f(x_n, y_n), \tag{22}
$$

$$
k_2 = h \cdot f\left(x_n + \frac{1}{5} \cdot h, y_n + \frac{1}{5} \cdot k_1\right),
$$

$$
k_3 = h \cdot f\left(x_n + \frac{3}{10} \cdot h, y_n + \frac{3}{40} \cdot k_1 + \frac{9}{40} \cdot k_2\right),
$$

$$
k_4 = h \cdot f\left(x_n + \frac{4}{5} \cdot h, y_n + \frac{44}{45} \cdot k_1 - \frac{56}{15} \cdot k_2 + \frac{32}{9} \cdot k_3\right),
$$

$$
k_5 = h \cdot f\left(x_n + \frac{8}{9} \cdot h, y_n + \frac{19372}{6561} \cdot k_1 - \frac{25360}{2187} \cdot k_2 + \frac{64448}{6561} \cdot k_3 - \frac{212}{729} \cdot k_4\right),
$$

$$k_6 = h \cdot f\left(x_n + h, y_n + \frac{9017}{6561} \cdot k_1 - \frac{355}{33} \cdot k_2 + \frac{46732}{5247} \cdot k_3 + \frac{49}{176} \cdot k_4 - \frac{5103}{18656} \cdot k_5\right),$$

$$k_7 = h \cdot f\left(x_n + h, y_n + \frac{35}{384} \cdot k_1 + \frac{500}{1113} \cdot k_3 + \frac{125}{192} \cdot k_4 - \frac{2187}{6784} \cdot k_5 + \frac{11}{84} \cdot k_6\right).$$

Následující krok vypočtený 4. Řádem RK, který se poté použije pro další cyklus jako výsledek je vypočten takto jako v [3]:

$$y_{n+1} = y_n + \frac{35}{384} \cdot k_1 + \frac{500}{1113} \cdot k_3 + \frac{125}{192} \cdot k_4 - \frac{2187}{6784} \cdot k_5 + \frac{11}{84} \cdot k_6. \tag{23}$$

Následující krok vypočtený 5. řádem RK se použije k výpočtu optimálního kroku a je vypočten jako v [4] takto:

$$z_{n+1} = y_n + \frac{5179}{57600} \cdot k_1 + \frac{7571}{16695} \cdot k_3 + \frac{393}{640} \cdot k_4 - \frac{92097}{339200} \cdot k_5 + \frac{187}{2100} \cdot k_6 + \frac{1}{40} \cdot k_7. \tag{24}$$

3.3.1 Optimální krok

Metoda může být organizována tak, aby sledovala vnitřní soulad. To dovoluje kontrolu numerických chyb, které jsou nevyhnutelné, automatickou změnou délky kroku. Je doporučeno k metodám přidávat možnost automatické kontroly kroku. Mnou použitý výpočet optimálního kroku je jako v [4] následující:

$$s = \left(\frac{\varepsilon \cdot h}{2 \cdot |z_{n+1} - y_{n+1}|}\right)^{\frac{1}{5}}, \tag{25}$$

$$h_{opt} = s \cdot h.$$

V rovnici (25) je ε dovolená chyba, h předchozí krok a h_{opt} je krok doporučený pro další posun.

3.3.2 Konvergence metody

Konvergence je základní vlastnost, která je u metody požadována. Čím menší krok tím blíže jsme přesnému řešení. Metoda by měla pro $h \to 0$ konvergovat k přesnému řešení.

Řekneme, že metoda je konvergentní, jestliže pro libovolnou počáteční úlohu (6) platí pro každé $x \in \langle a, b \rangle$ jako v [2] :

$$\lim_{\substack{n\to\infty \\ h\to 0}} y_n = y(x). \tag{26}$$

V rovnici (26) je $x = x_0 + n \cdot h$.

4 APROXIMACE FUNKCE

Hodnoty v souboru měly být proloženy nějakou funkcí. To, jak funkce bude nalezena napoví tato kapitola, která se zabývá aproximací funkce. Seznámíme se s tím jak nalézt funkci, která by pokud možno co nejlépe popisovala naměřená data. Začneme tedy volbou metody.

4.1 Volba vhodné metody pro aproximaci

Dvě hlediska, kterými se na problém dá nahlížet jsou aproximace funkce a interpolace funkce. Z důvodu toho, že se jedná o data naměřená, se budeme věnovat pouze tomu, co jsme v aplikaci použili a to aproximaci. Aproximace je nahrazení funkce f, funkcí jinou např. φ, která v nějakém vhodném smyslu napodobuje funkci f a snadno se přitom matematicky zpracovává či modeluje na počítači. Tuto φ funkci nazýváme aproximací funkce f. Z aproximačních metod byla použita metoda nejmenších čtverců a to proto, že je vhodná pro práci s naměřenými daty zatíženými chybou. To se o interpolačních metodách říci nedá.

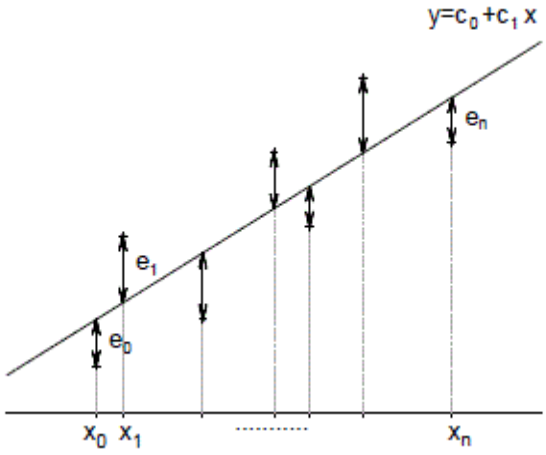
4.2 Metoda nejmenších čtverců

Zde bude popsána metoda nejmenších čtverců [2]. Vše podstatné o tom, proč jsme si vybrali právě tuto metodu, jsem již popsal v předchozím oddíle. Na nejjednodušším příkladě aproximace přímkou si ukážeme celý princip této metody.

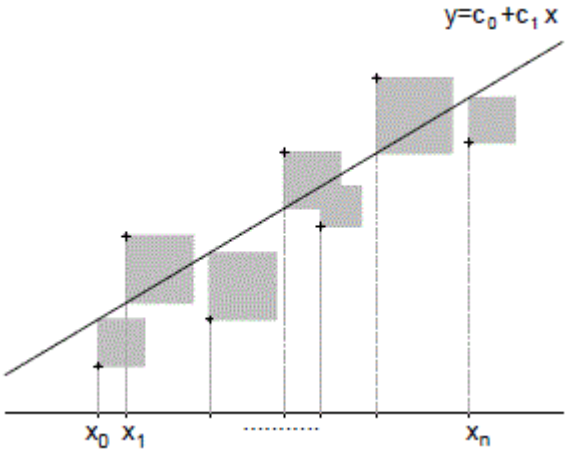
4.2.1 Aproximace přímkou

Jde o nejjednodušší případ, na kterém bude popsána základní myšlenka metody nejmenších čtverců. Jsou dány body $x_i \;\; i = 0, \dots, n$ a funkční hodnoty v těchto bodech y_i. Budeme hledat přímku o rovnici $y = \; c_0 + \; c_1 \cdot x$, která bude „co nejlépe“ procházet mezi body $[x_i, y_i], i = 0, \dots, n.$

Označme e_i chybu aproximace $e_i = y_i - \; y(x_i) = \; y_i - \; c_0 - \; c_1 \cdot x_i$. Kritérium pro nalezení „co nejlepšího“ procházení je, aby součet druhých mocnin (neboli čtverců) chyb v jednotlivých bodech byl minimální, tedy $\rho^2(c_0, c_1) = \; e_i^2 = \sum_{i=0}^{n} (y_i - c_0 - \; c_1 \cdot x_i)^2$.



Obr. 9: Odchylky [2]



Obr. 10: Hledáme přímku s minimálním součtem obsahů čtverců [2]

Pro náš případ bohatě vystačí podmínka jako v [2]:

$$\rho^2 = \sum_{i=0}^{n} \left(y_{si} - y(x_i)\right)^2, \tag{27}$$

kde ρ^2 je v aplikaci zvoleno , y_{si} je naměřená hodnota ze souboru a $y(x_i)$ vypočteme metodou RKDP. Rovnice (27) v podstatě představuje tvar funkce, který se bude minimalizovat. To jak se minimalizuje bude předvedeno v následující kapitole.

5 OPTIMALIZACE FUNKCE

Je hledáno minimum funkce:

$$f(x) = \sum_{i=0}^{n} \left(y_{si} - y(x_i)\right)^2. \tag{28}$$

V tomto tvaru pak $f(x)$ je funkční hodnota, y_{si} stejně jako v předchozí kapitole představuje naměřené hodnoty ze souboru a $y(x_i)$ se dá po dosazení chápat jako:

$$y(x_i) = \frac{b_0 - a_n \cdot y^{(n)} - a_{n-1} \cdot y^{(n-1)} - \ldots - a_1 \cdot y'}{a_0}. \tag{29}$$

Kde $y^{(n)}$ až y' jsou dopočítány metodou RKDP. To je tvar funkce, která bude minimalizována. Půjde o nalezení minimální hodnoty $f(x)$.

5.1 Volba vodné metody pro optimalizaci

Optimalizační úlohy se zabývají, jak jsme si již v úvodu naznačili, výběrem pro naši potřebu nejlepších řešení z dané množiny možných řešení. Matematicky můžeme optimalizační úlohu formulovat jako nalezení prvku $x^* \in M$ takového, že pro libovolný prvek $x \in M$ platí $f(x^*) \leq f(x) \quad \forall x \in M$, kde $f: M \mapsto R$ je minimalizovaná funkce a M je množina přípustných řešení. Jestliže přípustným řešením může být každý bod $x = (x_1, x_2, \ldots, x_n)^T$ n-rozměrného Euklidova prostoru R^n, tj. $M = R^n$ [12], hovoříme o nepodmíněné optimalizaci.Případy, kdy je množina nějak omezena, nazýváme optimalizací podmíněnou. Pro potřeby aplikace se jedná a nepodmíněnou optimalizaci.

Počet rozměrů je dán počtem neznámých proměnných. Úloha vyžaduje vyšší počet neznámých proměnných. Takové optimalizaci se říká vícerozměrná.

Některé metody vyžadují derivaci, avšak takové patří mezi ty komplexnější. Ve výběru metody jsem dal přednost spíše jednoduchosti.

Nakonec byla zvolena metoda Nelder – Mead, která nepotřebuje k řešení derivaci, jde o vícerozměrnou optimalizační metodu a je nepodmíněná, tudíž má vše co potřebujeme. Navíc je doporučena v knize [1], pro řešení optimalizačních problémů, kde je optimalizace na vedlejším místě, nejsou až takové požadavky na rychlost, je spolehlivá, jednoduchá a je i součástí MATLABU jako funkce $fminsearch$ pro hledání minima pro vícerozměrné případy.

5.2 Nelder –Mead

Nelder-Mead dále v textu pak ve zkratce jako NM. Jak jsme si již uvedli, metoda minimalizuje funkci o n -proměnných s tím, že používá pouze funkční hodnoty v bodech. NM v každém kroku tvoří geometrický obrazec o p dimenzích nenulového obejmu tvořeného $n+1$ body. Tento obrazec postupně mění množinu možných řešení směrem k minimu. Nakonec je množina tak malá, že je minimum nalezeno.

5.2.1 Parametry

K úspěšnému chodu algoritmu musí být napřed stanoveny čtyři parametry [5] ϑ pro odraz, σ pro zmenšení, γ pro zkrácení a χ pro prodloužení, které by měly splňovat následující podmínky: $\vartheta > 0$, $\chi > 1$, $\chi > \vartheta$, $0 < \gamma < 1$ a $0 < \sigma < 1$.

Standardně bývají pak zvoleny takto: $\vartheta = 1$, $\chi = 2$, $\gamma = \frac{1}{2}$ a $\sigma = \frac{1}{2}$. A takto bude zvoleno i pro aplikaci.

5.2.2 Základní princip

Na začátku k-té iterace $k \geq 0$ je obdržen obrazec Δ_k o $n+1$ bodech, každý z nich je součástí R^n. Iterace k začíná označením těchto bodů $x_1^{(k)}$ až $x_{n+1}^{(k)}$ takto $f_1^{(k)} \leq \ldots \leq f_n^{(k)} \leq f_{n+1}^{(k)}$, kde $f_i^{(k)}$ označuje $f\left(x_i^{(k)}\right)$. K-tá iterace vygeneruje $n+1$ bodů s tím, že obrazec $\Delta_{k+1} \neq \Delta_k$. Protože chceme funkci f minimalizovat, tak $x_1^{(k)}$ bereme jako nejlepší bod (s nejmenší funkční hodnotou), $x_{n+1}^{(k)}$ jako bod nejhorší(s nejvyšší funkční hodnotou) a $x_n^{(k)}$ jako druhý nejhorší bod. Stejně tak bereme $f_{n+1}^{(k)}$ jako nejhorší funkční hodnotu, $f_n^{(k)}$ jako druhou nejhorší a $f_1^{(k)}$ jako nejlepší. S těmito body pak pracujeme dále a mají vliv na to, jak se bude obrazec formovat. V každém iteračním cyklu přijímáme po určité operaci, které budou popsány dále v textu, bod, který nahrazuje nejhorší bod x_{n+1} anebo provedením zmenšení se změní všechny body až na x_1, které nadále zůstane součástí obrazce.[5]

5.3 Příklad iteračního cyklu NM

Na příkladě hledání extrému funkce $f(x_1, x_2)$[6], tj. $n = 2$ budou popsány jednotlivé úkony, které se při hledání nového obrazce používají. Napřed bude popsáno něco k úvodním bodům a pak přejdeme k řazení, odrazu, prodloužení, vnějšímu zkrácení, vnitřnímu zkrácení

a nakonec zmenšení obrazce. V následujícím textu použijeme trochu odlišné značení bodů, než v 5.2.2, ale princip zůstane stejný. Vše potřebné bude vysvětleno.

5.3.1 Úvodní body

Úvodní body jsou tvořeny vrcholy počátečního obrazce a lze je vyjádřit maticově takto:

$$S = [x_1, \dots, x_{n+1}] = \begin{pmatrix} 0 & l_1 & l_2 & l_2 & \dots & l_2 \\ 0 & l_2 & l_1 & l_2 & \dots & l_2 \\ 0 & l_2 & l_2 & l_1 & \cdots & l_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & l_2 \\ 0 & l_2 & l_2 & l_2 & l_2 & l_1 \end{pmatrix}. \tag{28}$$

V rovnici (28) pro l_1 a l_2 platí následující vztahy jako v [6]:

$$l_1 = \frac{d}{n \cdot \sqrt{2}} \cdot \left(\sqrt{n+1} + n - 1\right), \tag{29}$$

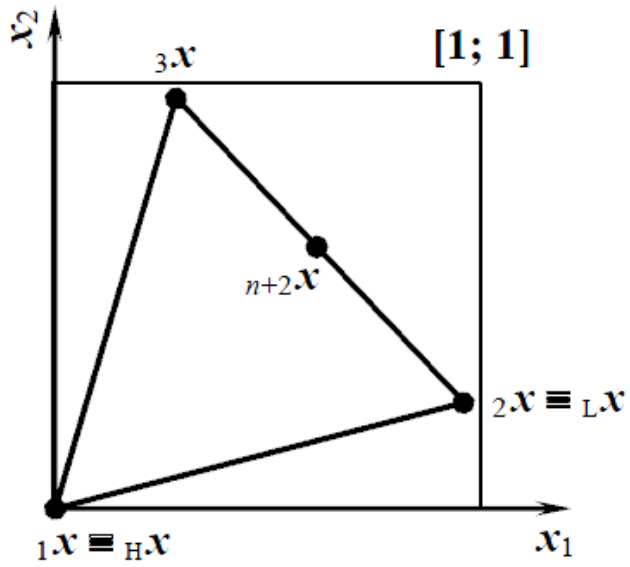
$$l_2 = \frac{d}{n \cdot \sqrt{2}} \cdot \left(\sqrt{n+1} - 1\right).$$

Pro hranu d a počet proměnných n můžeme dosadit následující $d = 1, \quad n = 2$ a výsledkem bude:

$$S = \begin{bmatrix} 0 & 0.965 & 0.259 \\ 0 & 0.259 & 0.965 \end{bmatrix}.$$

5.3.2 Řazení

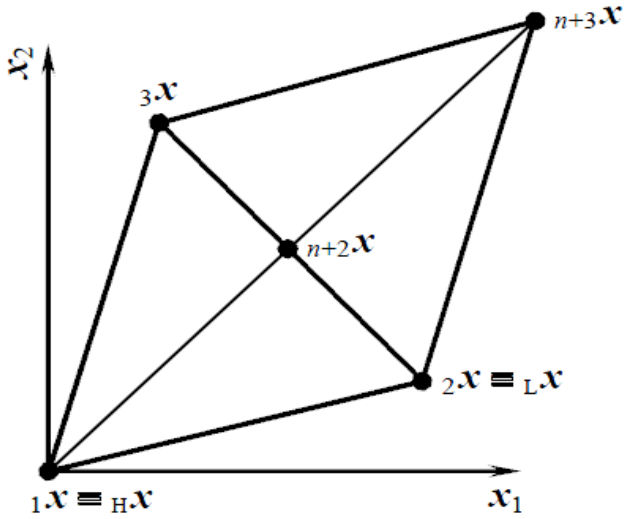
Napřed budou body seřazeny podle hodnoty funkce v daném bodě, pro ilustraci si nejlepší bod označíme x_L (minimální hodnota - minimalizace), nejhorší bod x_H (nejvyšší funkční hodnota) a druhý nejhorší bod x_T(druhá nejvyšší funkční hodnota). Jak již bylo popsáno výše jde o body, ve kterých je funkční hodnota $f(x) = \sum_{i=0}^{n} \left(y_i - y(x_i)\right)^2$ nejvyšší, druhá nejvyšší anebo nejnižší. Ostatní body nejsou podstatné.



Obr. 11: Základní body [6]

5.3.3 Bod odrazu

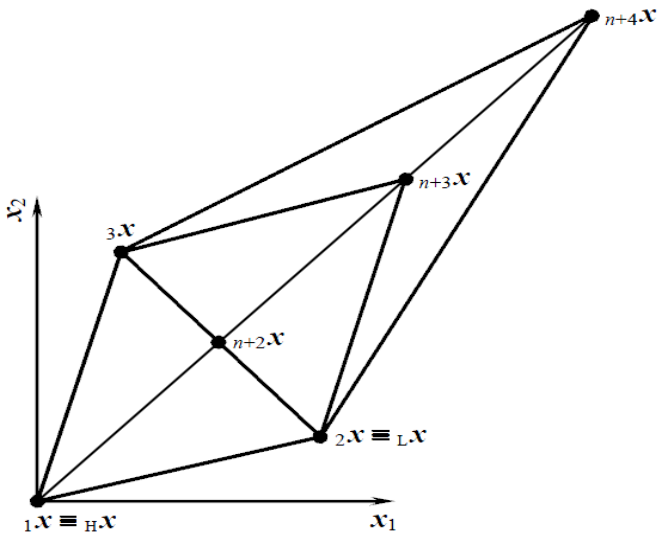
Bod odrazu bude označen x_{n+2}. Dále budeme potřebovat těžiště, které si označíme jako x_{n+3}. Oba body vypočítáme následovně jako v [5] : $x_{n+2} =,\frac{1}{n} \cdot \sum_1^n x_i$,(počítáme bez nejhoršího bodu x_H) a $x_{n+3} = x_{n+2} + \vartheta \cdot (x_{n+2} - x_H)$. Vypočítáme $f(x_{n+3})$ a porovnáme v podmínce $f(x_L) \leq f(x_{n+3}) \leq f(x_T)$. Pokud je splněna, tak bod x_{n+3} přijmeme a nahradíme jej za x_H (přijetí bodu bude vždy znamenat nahrazení za x_H, až na operaci zmešení, kde vše probíhá trochu jinak). Po přijetí ukončíme iteraci.



Obr. 12: Odraz [6]

5.3.4 Bod prodloužení

Spočítán bude pouze v případě jako v [5], že se $f(x_{n+3}) < f(x_L)$. Bod prodloužení bude označen x_{n+4}. A platí pro $x_{n+4} = x_{n+2} + \chi \cdot (x_{n+3} - x_{n+2})$. Vypočítáme $f(x_{n+4})$. Pokud je $f(x_{n+4}) < f(x_{n+3})$, tak přijmeme bod x_{n+4} a ukončíme iteraci a pokud je $f(x_{n+4}) \geq f(x_{n+3})$, tak přijmeme bod x_{n+3} a ukončíme iteraci.

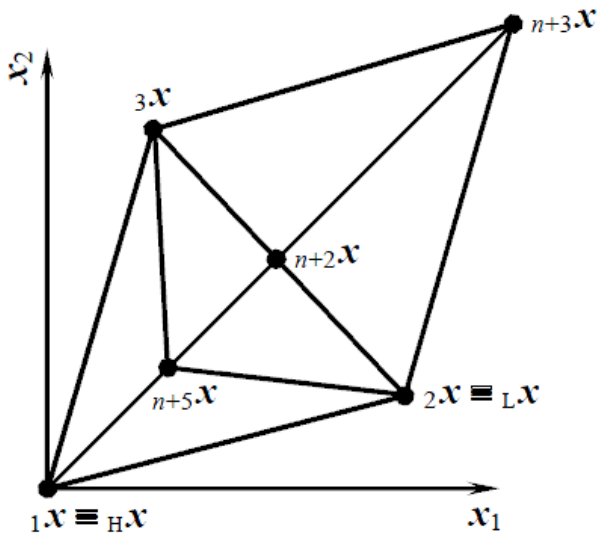


Obr. 13: Prodloužení [6]

5.3.5 Bod zkrácení

Zkrácení je provedeno jako v [5] , když $f(x_{n+3}) \geq f(x_T)$ a to mezi těžištěm x_{n+2} a lepším z bodů x_H(vnitřní zkrácení) a x_{n+3}(vnější zkrácení). Začneme vnějším, které nastane když je $f(x_T) \leq f(x_{n+3}) \leq f(x_H)$, bod vnějšího zkrácení bude značen x_{n+5V} a spočten bude následovně $x_{n+5V} = x_{n+2} + \gamma \cdot (x_{n+3} - x_{n+2})$. Vypočteme $f(x_{n+5V})$. Pokud je $f(x_{n+5V}) \leq f(x_{n+3})$, tak je přijat x_{n+5V} a ukončíme iteraci, jinak přejdeme ke kroku zmenšení.

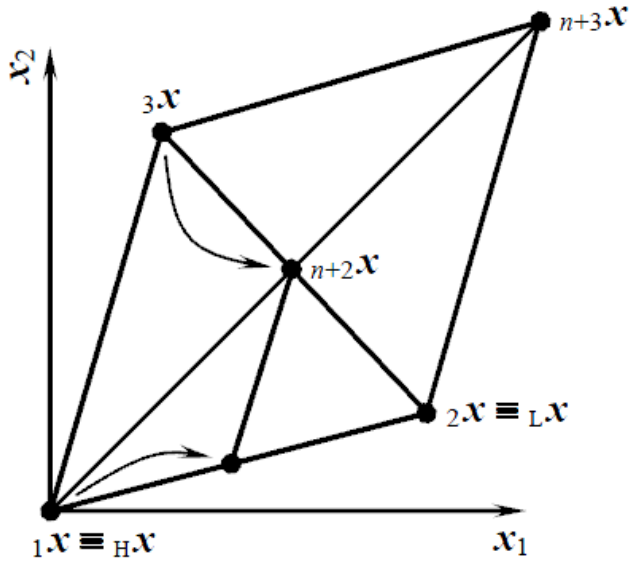
Vnitřní zkrácení je provedeno když je $f(x_{n+3}) \geq f(x_H)$. Vypočteme $x_{n+5VV} = x_{n+2} - \gamma \cdot (x_{n+2} - x_H)$. Je zjištěna hodnota $f(x_{n+5VV})$ a pokud je $f(x_{n+5VV}) < f(x_H)$, přijmeme x_{n+5VV} a ukončíme iteraci, v opačném případě přijde na řadu zmenšení



Obr. 14: Vnitřní zkrácení [6]

5.3.6 Zmenšení

Při zmenšení je vypočtena jako v [5] funkce f v n bodech $v_i = x_1 + \sigma \cdot (x_i - x_1)$, pro $i = 2$ až $n + 1$. Nové body obrazce pak jsou nejlepší bod x_1 s minimální funkční hodnotou(minimalizujeme) a body v_2 až v_{n+1}.



Obr. 15: Zmenšení [6]

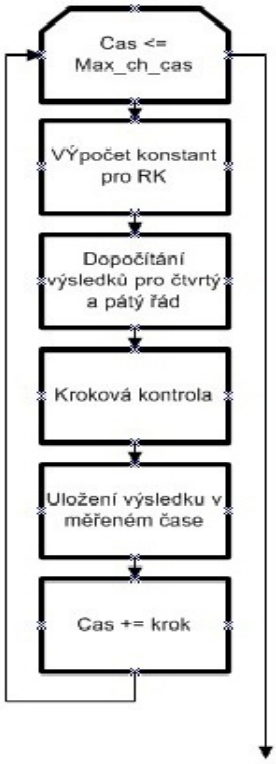
Realizační část

6 REALIZACE RKDP

V realizačních kapitolách záleží spíše na pochopení podstaty. Toho jak to pracuje. Vše podstatné je vysvětleno, ale z důvodu místa je opakující se kód zkracován a všemožně upravován tak, aby byl co nejpřehlednější. Součástí popisu je také zjednodušený vývojový diagram.

6.1 Zjednodušený vývojový diagram pro RKDP

Zde je na obr. 16 představen hodně zobecněný vývojový diagram pro RKDP, který nám dá ucelenější náhled na řešení problému. Podrobnosti se lze dočíst pročtením následujících částí,kde je zmíněn i zjednodušený zdrojový kód.



Obr. 16: Algoritmus RKDP

6.2 Algoritmus pro RKDP vyššího řádu

Zde si uděláme představu o algoritmu jako celku. Skládá se ze tří základních částí. Algoritmu inspirovaným metodou RKDP, který představuje první dvě části vývojového diagramu. Nalezení optimálního kroku a jeho úpravy. Nakonec pak vypíchnutí výsledku v podstatném čase a jeho uložení. Zde si popíšeme část první, kde se na cyklickém výpočtu řeší LDR libovolného řádu. Nyní tedy něco ke zdrojovému kódu.

Funkce() zajišťuje výpočet čitatele v rovnici:

$$y^{(n)} = \frac{b_0 \cdot u(t) - a_{n-1} \cdot y^{(n-1)} - \ldots - a_1 \cdot y' - a_0 \cdot y}{a_n}. \quad (30)$$

Rovnice (30) je upravená rovnice (1) pro skok $u(t) = 1$. To je rovnice pro kterou hledáme konstanty a_0 až a_n a b_0, ze kterých zjistíme zesílení $K = \frac{b_0}{a_0}$ a konstanty časové $T_1 = \frac{a_1}{a_0}$ až $T_n = \frac{a_n}{a_0}$. Jak již bylo zmíněno ve 2.3. Ke konstantám dospějeme postupnou minimalizací funkční hodnoty jako v 5. kapitole.

Proměnná ***cas*** je aktuální čas, po který se počítá a ***max_ch_cas*** hlídá až do kdy se má smyčka vyhodnocovat včetně hromadění double chyb. Je to ***cas***, ke kterému je přičtena chyba, která řeší nepřesnosti typu double. Chyba je definována konstantou CHYBA_KROK. ***Rad*** v sobě nese řád diferenciální rovnice, který zadal uživatel. ***Pom_idx*** ukládá hodnotu ***rad–1*** pro zrychlení běhu, aby nebylo nutné stále výpočet opakovat. Kr***ok*** udává aktuální velikost kroku. ***Vertex[][]*** ukazuje na bod v matici jako v rovnici (28), kde v prvním indexu je souřadnice bodu a ve druhém bod. ***Pom2*** až ***pom75*** jsou konstanty. ***Y[]*** je součástí vzorce RKDP (22) stejně tak jako ***k[][]***. V ***rk4_y[]*** a ***rk5_y[]*** se uloží výsledky RK 4. a RK 5. řádu. Nakonec je tu ***pom_y[]*** kde najdeme uložený výsledek, hodnotu v novém počátečním bodě, pro další cyklus

```
#include  <stdio.h>
#define  CHYBA_KROK 0.00001

while (cas <= max_ch_cas) {   /* podmínka cyklu  */

/* výpočet tvaru rovnice jako v (30) násobené krokem*/

     k[0][pom_idx] = krok * Funkce(vertex, y,
rad)/vertex[*rad + 1][z];

/* pro vyšší řád */

   for(i = *rad; i > 1; i--) {

      k[0][i - 2] = krok * y[i];
       y[i] = pom_y[i - 1] + (pom2 * k[0][i - 1]);

   }

y[1] = pom_y[0] + (pom2 * k[0][0]);
  /*
   * opakuje se s pozměněnými konstantami od k[0] až do k[6] jako (22)
   */
```

```
       k[5][pom_idx] = krok * Funkce(vertex, y, rad)/vertex[*rad
+ 1][z];

       for(i = *rad; i > 1; i--) {

          k[5][i - 2] = krok * y[i];
          y[i] = pom_y[i - 1] + (pom71 * k[0][i - 1] + pom72 *
k[2][i - 1] + pom73 * k[3][i - 1] -   pom74 * k[4][i - 1] +
pom75 * k[5][i - 1]);

      }

     y[1] = pom_y[0] + (pom71 * k[0][0] + pom72 * k[2][0] +
pom73 * k[3][0] - pom74 *  k[4][0] + pom75 * k[5][0]);
        k[6][pom_idx] = krok * Funkce(vertex, y,
rad)/vertex[*rad + 1][z];

      for (i = *rad; i > 1; i--) {

          k[6][i - 2] = krok * y[i];

      }
```

/* z konstant k[][] a předchozího bodu dopočítáno rk5_y a rk4_y jako (23) a (24), následně pro další počítání uložen výsledek rk4_y do pom_y

```
      /*

      for (i = 0; i < *rad; i++) {

          rk4_y[i] = pom_y[i] + (pom71 * k[0][i] + pom72 *
k[2][i] + pom73 * k[3][i] - pom74 *       k[4][i] + pom75 *
k[5][i]);
         rk5_y[i] = pom_y[i] + (pomRK51 * k[0][i] + pomRK52 *
k[2][i] + pomRK53 * k[3][i] -  pomRK54 * k[4][i] + pomRK55 *
k[5][i] + pomRK56 * k[6][i]);
          pom_y[i] = rk4_y[i];

       }
```

/*
* následují části: optimální krok(6.3), úprava optimálního kroku(6.4), podstatný krok(6.5)
*/

```
          cas += krok;
       }  /*začátek dalšího cyklu */
```

6.3 Optimální krok

Zde popíši jak byl nalezen optimální krok. Jde o pokračování 6.2, takže proměnné, konstanty a funkce, které byly popsány předtím nebudu znovu zmiňovat. Zprvu se spočítá rozdíl obou v předchozích části vypočtených hodnot RK 4. a 5. řádu. Nové proměnné jsou zde ***pom_optkrok***, který slouží pro zjištění kroku pro vyšší řády, a následně pak dojde k porovnání s ***optkrok***, což představuje optimální krok pro první řád. Vystupující optimální krok pak bude díky podmínce nejmenším krokem k z celé soustavy (***optkrok*** počítáme pro každou proměnnou zvlášť a pak najdeme minimum). Proměnná ***s*** slouží jako koeficient pro úpravu kroku.

```
#include <stdio.h>
#include <math.h>

#define CHYBA_RK 0.000001 /* povolená chyba jednoho kroku */

  /*
  *  Předcházející části: Algoritmus pro RK vyššího řádu (6.2)
  */


/* výpočet rozdílu pátého a čtvrtého řádu v daném kroku */

   rozdil = fabs(rk5_y[0] - rk4_y[0]);

/* vzorec pro výpočet chyby   uváděný v  teoretické části (25) */

   s = pow((CHYBA_RK * krok)/(2 * rozdil), poms );
  optkrok = s * krok;

  for (i = 1; i < *rad; i++) {

/* výpočet pro vyšší než první řád */

  rozdil = fabs(rk5_y[i] - rk4_y[i]);
    s = pow((CHYBA_RK * krok)/(2 * rozdil), poms );
    pom_optkrok[i] = s * krok;

    if (pom_optkrok[i] < optkrok) {

 /*nalezení nejmenšího optkroku ze všech proměnných   pouze pro případ vyššího řádu*/

      optkrok = pom_optkrok[i];

}

   }
```

```
 /* optkrok vystupuje k dalším úpravám */
/*
 *  následující části: úprava optimálního kroku (6.4), podstatný čas (6.5)
 */
```

6.4 Úprava optimálního kroku

V této části dojde k úpravě optimálního kroku tak, aby nový krok vyhovoval všem požadavkům výstupu. Především jde o to aby se vypočítané hodnoty daly vždy zaznamenat ve stejném čase jako jsou naměřené ve vstupním souboru. Proměnná ***minkrok*** v sobě nese minimum, které je možné u kroku nastavit. ***Pomkrok*** představuje minulý krok. ***Prvni_krok*** obsahuje základní rozestup jednotlivých měření v suboru. ***N*** slouží jako pomocná proměnná pro úpravu optimálního kroku a funkce ***Uprava_kroku()*** slouží ke konečné úpravě kroku, tak aby vyhovoval požadovanému času a nedošlo k přeskočení podstatných bodů, ve kterých probíhala měření.

```
#include <stdio.h>

      /*
       * Předcházející části: Algoritmus pro RK vyššího řádu (6.2), optimální krok (6.3)
       */

/*  nový krok nesmí být menší než minimální povolený krok */


   if (optkrok < *minkrok) {

     krok = *minkrok; /* pokud je menší tak se nový krok stane minimálním
povoleným i když je menší */
     pomkrok = krok;


  else {  /* pokud není menší než minimální provede se následující */

/* zaokrouhlení na násobek prvního kroku */

      while ( (optkrok * n) < *prvni_krok) {

/*spočítá se kolika násobek kroku překračuje první krok ze souboru */
         n++;

  }

/*odsekne se přebývající část a zpětně se podělí dříve nalezeným číslem */
```

```
      optkrok = optkrok * n + (*prvni_krok - optkrok * n);
       optkrok = optkrok / n;
```

/* pokud je číslo optkrok < než 0.6 * předchozí krok nebo větší než 1.9 * předchozí krok tak se provede následující funkce */

```
      if ((optkrok < pomkrok * 0.6) || (optkrok > pomkrok * 1.9))
{
```

/* upraví krok do konečné podoby */

```
          Uprava_kroku(&krok, prvni_krok, &optkrok, &cas,
&pomkrok);


           }

       n = 1;       /* n se znovu nastaví na 1 pro další cyklus */

       }
            /*
             Následující části: Podstatný čas (6.5)
            */
```

6.5 Podstatný čas

Zde budeme seznámeni s poslední částí algoritmu, kde se řeší to, jak uložit výsledek, který jsme získali v čase kdy proměnná ***cas*** odpovídá ***m-***násobku proměnné ***prvni_krok***. Ten představuje konstantní krok kterým se měřilo. Výsledek je uložen do ***mezi_y[][][]***, kde ***z*** je vstupním parametrem, který slouží mimo algoritmus RK k označení bodu pro který hodnoty počítáme. Prostřední index je nastaven na 1 protože tam se nachází řád výsledku, který potřebujeme k porovnání se souborem.

```
      #include <stdio.h>

      #define CHYBA_DBL 0.00000001 /* chyba typu double pro podmínku */

      /*
               Předcházející části: Algoritmus pro ODR vyššího řádu (6.2), optimální krok
(6.3), úprava optimálního kroku (6.4)
               */

       /*ukládám v čase, který je násobkem intervalu ze souboru */

       if ((cas <= ((*prvni_krok * m) + CHYBA_DBL)) && (cas >=
((*prvni_krok * m) - CHYBA_DBL))) {
```

```
        mezi_y[m][1][z] = pom_y[0];
            m++;

        }
```

6.6 Shrnutí algoritmu RK

Zde uvedu jak algoritmus spolupracuje s ostatními částmi. Vstupní parametry funkce jsou následující: double ***mezi_y[][][]***, double ***vertex[][]***, int ****mezpole***, double ****prvni_krok***, double ****max_cas***, int ****rad***, int *z*.

Mezi_y[][][] je trojrozměrné pole, které ukládá výsledky vypočtené uvnitř RK a ty pak slouží i mimo funkci. Prvním rozměrem je počet naměřených hodnot, druhým je řád výsledku a třetím je již dříve zmiňovaný bod, pro který hodnoty *y* počítáme. ***Vertex[][]*** představuje matici bodů se kterými pracujeme jako je v teoretické části znázorněno rovnicí (28). ****Mezpole*** udává počet naměřených hodnot ve vstupním souboru. ****Prvni_krok*** obsahuje rozestup mezi daty měřenými v souboru. ****Max_cas*** maximální čas, po který bude probíhat cyklus RK. ****Rad*** je uživatelův vstup značící řád, kterým budeme aproximovat a *z* označuje v trojrozměrném poli ***mezi_y[][][]*** bod, pro který zrovna počítáme. Výstup se zapisuje právě do tohoto trojrozměrného pole.

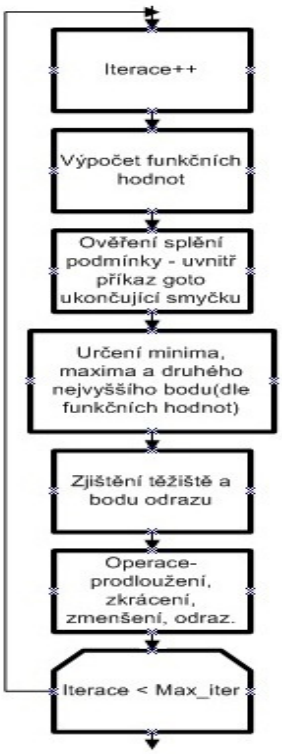
Ve zdrojovém kódu má funkce hlavičku ***RK45DP*** (double ***mezi_y, double **vertex, int *mezpole, double *prvni_krok, double *max_cas, int *rad, int z)

7 REALIZACE NM

V této části bude rozebrán algoritmus Nelder–Mead, jehož součástí je v teorii zmiňovaná metoda nejmenších čtverců (27) a také se v něm vyskytne výše zmiňovaný algoritmus RKDP, který nám vypočítá hodnoty y pro žádaný bod $\boldsymbol{i}$.

7.1 Zjednodušený vývojový diagram pro NM

Zde si ukážeme zjednodušený vývojový diagram pro Nelder – Mead(obr. 17). Na první pohled bije do očí umístění iterace na začátek cyklu. Důvodem je použití několika příkazů continue uvnitř algoritmu, takže pokud by některý z příkazů continue byl před iterací, pak by k iteraci nedošlo.



Obr. 17 Algoritmus NM

7.2 Výpočet funkční hodnoty

Výpočet funkční hodnoty v daném bodě je úvodním úkonem, který je v algoritmu proveden. Slouží k tomu, aby mohl být nalezen nejlepší, druhý nejhorší a nejhorší bod (již bylo popsáno v 5.2.2). Pomocí výše popsaného algoritmu RKDP vypočteme hodnotu y v požadovaném bodě i. Poté je přepočtena funkční hodnota metody nejmenších čtverců, což je

ve zkratce obsahem smyčky for. Snažíme se tedy o to, aby obsah čtverců odchylek byl minimální. To je to co ve skutečnosti minimalizujeme. Výsledná funkční hodnota v bodě se poté uloží do pole ***fce[]***.

```
     for (i = 0; i < nmrad + 1 ; i++) {

       RK45DP(mezi_y, vertex, mezpole,  prvni_krok, max_cas,
&rad, i);

       for (j = 0; j < *mezpole; j++) {

            nej_y[j] = Funkce3(vertex, mezi_y, rad, j, i)/
vertex[nmrad - 1][i];
             mezi_y[j][rad + 1][i] = nej_y[j];
             mezivysledek += pow (soub_y[j] - Funkce1(vertex,
mezi_y , rad, j, i) / vertex[1][i], 2);

          }

            fce[i] = mezivysledek;
            mezivysledek = 0;

       }
```

7.3 Iterační cyklus NM – realizace

Cyklus začíná jak bylo uvedeno v 5.3.2 vypočtením funkčních hodnot v jednotlivých bodech. Poté dochází k hledání nejlepšího, druhého nejhoršího a nejhoršího bodu. To se ukládá do polí o dvou proměnných ***fcemin[2], fce2max[2]*** a ***fcemax[2]***. Ve ***fcemin[0]*** se uloží funkční hodnota a ve ***fcemin[1]*** číslo sloupce matice ***vertex[][],*** ve kterém se bod s minimální funkční hodnotou nachází. Pro ostatní body je to provedeno analogicky. Podobně jsou řešeny i body těžiště ***bodtez[]*** , bod odrazu ***bododr[]*** , bod prodloužení ***bodprodl[]*** a bod zkrácení ***bodzkr[]***. Na pozicích o mezích 0 až ***nmrad*** – 1 ukládají souřadnice bodů a na pozici ***nmrad*** se uloží funkční hodnota v tomto bodě. Funkce ***Hodnota_v_novem_bode(…)*** zjišťuje právě onu funkční hodnotu a ukládá ji na již zmiňovanou pozici. Podle různých kritérií se pak provádějí operace, tak jak je uvedeno v teoretické části 5.3.x anebo v [5] . Pro upřesnění mez ***nmrad*** značí počet souřadnic a ***nmrad*** + 1 počet bodů. Ve ***vertexu[][]*** je vždy o jeden bod více než kolik je dimenzí jako v (26) . ***Iterace*** představuje aktuální iteraci a ****max_iter*** uživatelem zvolený maximální počet iterací.

/* hledání důležitých základních bodů */

```
        for (i = 0; i < (nmrad + 1); i++) {

          if (fce[i] <= fcemin[0]) {

            fcemin[0] = fce[i];
            fcemin[1] = i;

          }

         if (fce[i] >= fcemax[0]) {

            if (fcemax[0] != fce[i]) {

              fce2max[0] = fcemax[0];
              fce2max[1] = fcemax[1];

            }

            fcemax[0] = fce[i];
            fcemax[1] = i;

          }

         if (fce[i] > fce2max[0] && fce[i] < fcemax[0]) {

           fce2max[0] = fce[i];
           fce2max[1] = i;

         }

        }
```

/* těžiště všech vrcholů kromě xh(max)*/

```
         for (i = 0; i < nmrad ; i++) {

           suma[i] = 0;

           for (j = 0; j < (nmrad + 1); j++) {

             suma[i] += vertex[i][j]; /* sum(xi) */

           }

           bodtez[i] = pom2tez * (suma[i] -
vertex[i][(int)fcemax[1]]);

         }
```

```
          Hodnota_v_novem_bode(rad, bodtez, soub_y, mezpole,
prvni_krok, max_cas);


        /* změna polyedru */

        /* bod odrazu - podmínky */
          for (i = 0; i < nmrad ; i++) { /* bod n + 3(text) */

            bododr[i] = bodtez[i] + 1 * (bodtez[i] -
vertex[i][(int)fcemax[1]]);


          }

          Hodnota_v_novem_bode(rad, bododr, soub_y, mezpole,
prvni_krok, max_cas);


          if (fcemin[0] <= bododr[nmrad] && bododr[nmrad] <
fce2max[0]) {

         /* odraz */

            for (i = 0; i < (nmrad + 1); i++) {

              for (j = 0; j < nmrad; j++) {

                if (i == (int)fcemax[1]) {

                  vertex[j][i] = bododr[j];

                }

              }

            }

            continue;

          }

          if (bododr[nmrad] < fcemin[0]) {

         /* prodloužení */

            for (i = 0; i < nmrad; i++) {

              bodprodl[i] = bodtez[i] + 2 * (bododr[i] -
bodtez[i]);
```

```
            }

            Hodnota_v_novem_bode(rad, bodprodl, soub_y, mezpole,
prvni_krok, max_cas);

            if (bodprodl[nmrad] < bododr[nmrad]) {


              for (i = 0; i < nmrad; i++) {

                vertex[i][(int)fcemax[1]] = bodprodl[i];

              }

              continue;

            }

            else {
```

/* prodloužení - odraz */

```
              for (i = 0; i < nmrad; i++) {

                vertex[i][(int)fcemax[1]] = bododr[i];


              }

              continue;

            }

          }

          if (bododr[nmrad] >= fce2max[0]) {
```

/* zkrácení */

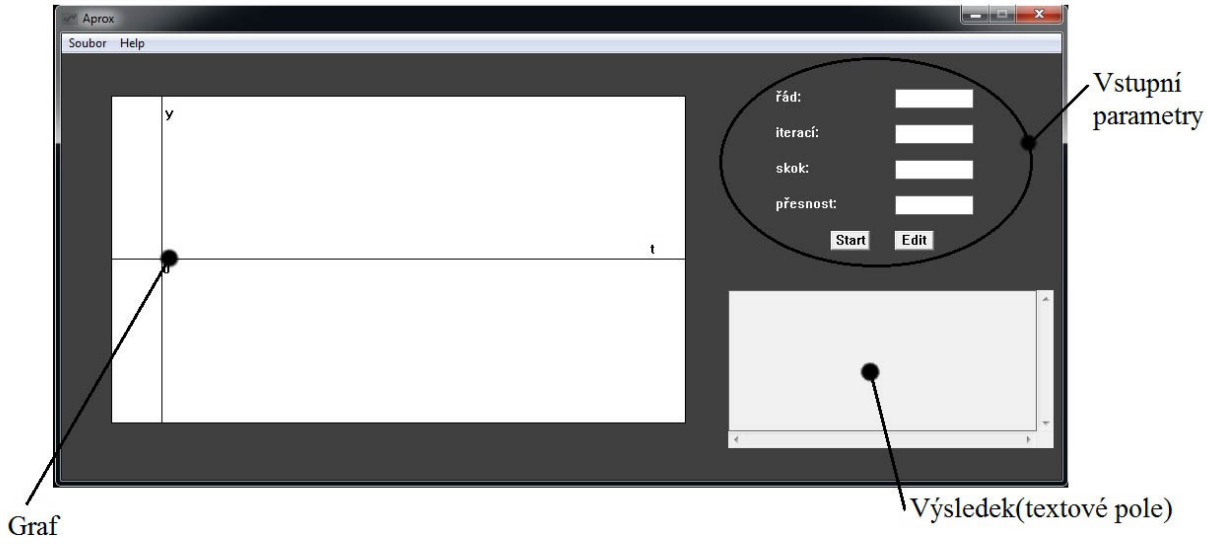
```
            if (fce2max[0] <= bododr[nmrad] && bododr[nmrad] <
fcemax[0]) {

              for (i = 0; i < nmrad; i++) {

                bodzkr[i] =  bodtez[i] + 0.5 * (bododr[i] -
bodtez[i]) ;


              }

              Hodnota_v_novem_bode(rad, bodzkr, soub_y, mezpole,
prvni_krok, max_cas);
```

```
            if (bodzkr[nmrad] <= bododr[nmrad]) {
          /* zkrácení vnější */
              for (i = 0; i < nmrad; i++) {
                  vertex[i][(int)fcemax[1]] = bodzkr[i]; /*
nahrazeni xh za n + 5 */
              }
              continue;
            }
          }
          if (bododr[nmrad] >= fcemax[0]) {
            for (i = 0; i < nmrad; i++) {
              bodzkr[i] = bodtez[i] - 0.5 * (bodtez[i] -
vertex[i][(int)fcemax[1]]);
            }
            Hodnota_v_novem_bode(rad, bodzkr, soub_y, mezpole,
prvni_krok, max_cas);
            if (bodzkr[nmrad] < fcemax[0]) {
          /* zkrácení vnitřní */
     for (i = 0; i < nmrad; i++) {
              vertex[i][(int)fcemax[1]] = bodzkr[i];
            }
            continue;
          }
        }
       }
      /* zmenšení - shrink */
      for (i = 0; i < (nmrad + 1); i++) {
```

```
            for(j = 0; j < nmrad; j++) {

              if (i != (int)fcemin[1]) {

                vertex[j][i] = vertex[j][(int)fcemin[1]] + 0.5 *
(vertex[j][i] - vertex[j][(int)fcemin[1]]);

              }

              else {

              }

            }

          }

        } while (iterace != *max_iter);
```

8 UKÁZKA CHODU APLIKACE

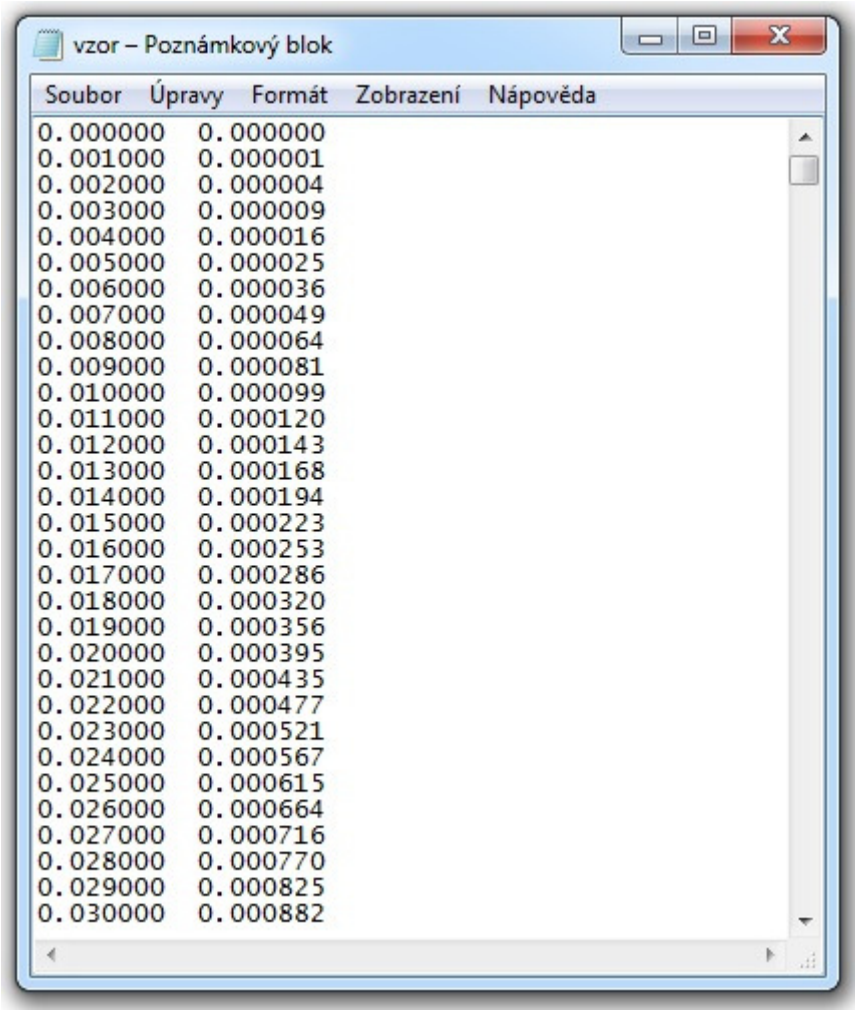
V této kapitole bude ukázáno na příkladě vstupního souboru vzor.dat jak by měla být vstupní data uspořádána. Aplikace bude představena již zde v úvodu obrázkem (obr. 18.). Grafické uživatelské rozhraní je psáno ve Win32 API[9]. Projdeme si postupně celý cyklus užití této aplikace.



Obr. 18: Náhled na aplikaci

8.1 Načtení dat ze souboru

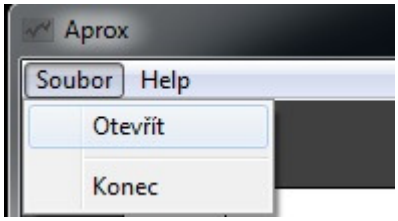
Vstup musí být zadán v požadovaném formátu tak, aby jej aplikace správně načetla. Mezi tyto požadavky patří koncovka .dat a uspořádání souboru, tak jak jej vidíme na obr. 19.



Obr. 19: Podoba vstupního souboru

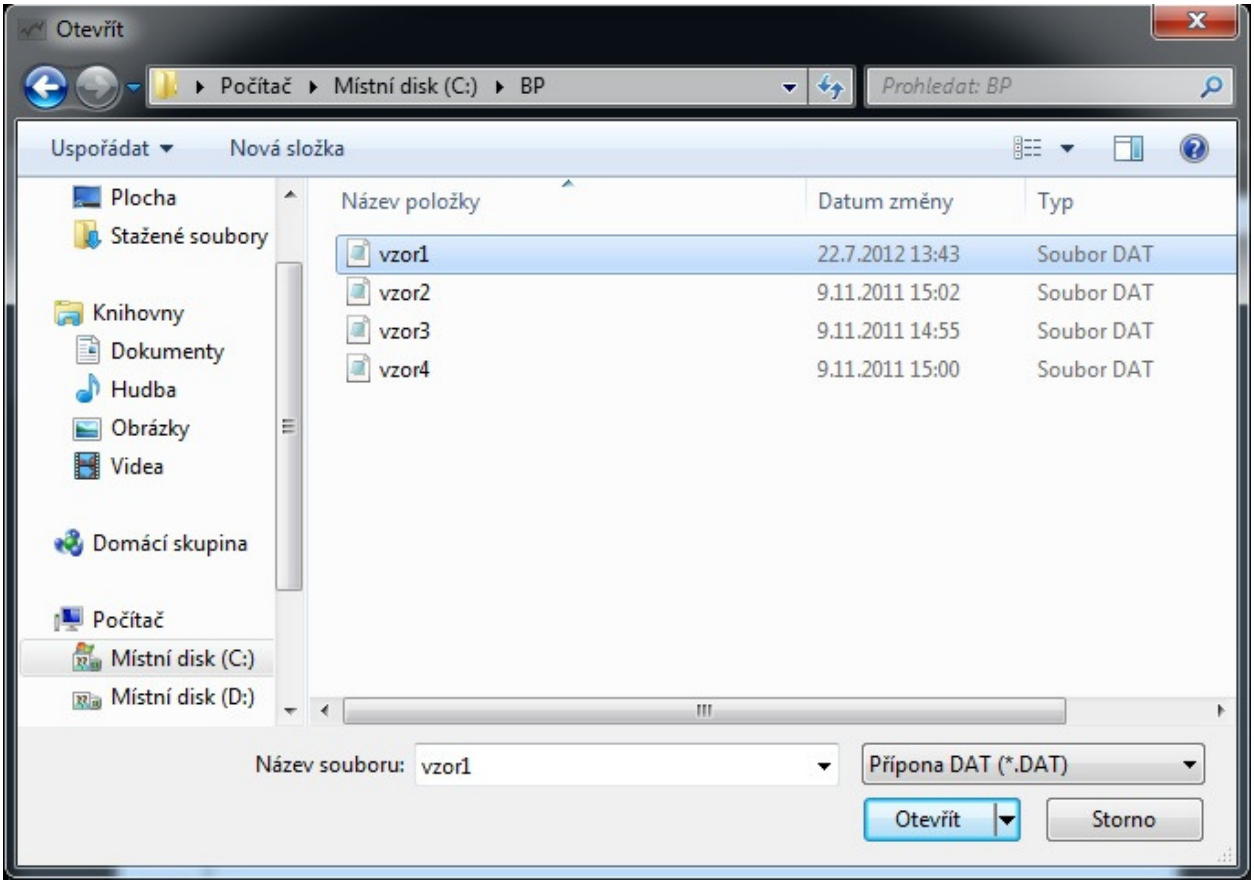
Soubor je rozdělen do dvou sloupců. První sloupec představuje čas, pro který se měřilo a druhý sloupec naměřené hodnoty y v čase t. Oba sloupce jsou odděleny libovolným počtem bílých znaků, což je důležité pro načtení obou dvou čísel zvlášť. Jako oddělovač desetinných míst může být použita jak tečka, tak i čárka.

Pro načtení souboru si v menu zvolíme možnost “Soubor” a v ní “Otevřít”.



Obr. 20: Otevřít

Poté se zobrazí okno jako na obr. 21, ve kterém máme možnost buďto nějaký .dat soubor otevřít a načíst anebo se vrátit zpět tlačítkem “storno”.



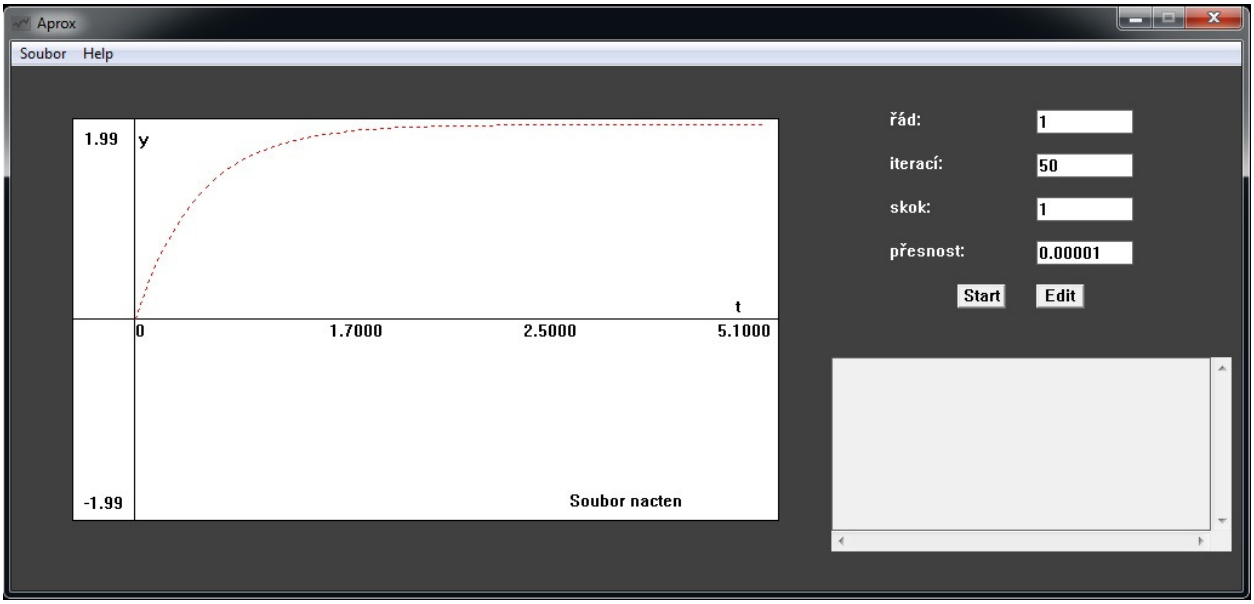
Obr. 21: Načíst

8.2 Vstup

Poté co již máme soubor načtený, můžeme zadat do editačních polí v pravé části aplikace naše parametry pro vstup. Mezi tyto parametry patří: maximální počet iterací, skok, řád a požadovaná přesnost.

Při nekorektním vstupu anebo při nenačtení souboru algoritmus aplikace nepůjde spustit. Vypíše se hlášení o chybě, které na případný nedostatek na vstupu upozorní. Po načtení souboru se zobrazí graf dat ze souboru. Podle něj je možné rozpoznat, zda-li dodaný vstupní soubor byl ve správném formátu.

Za běhu nám aplikace ukazuje (měřeno na iterace), v jakém bodě výpočtu se právě nachází. Na obr. 22. se například nacházíme ve chvíli těsně před spuštěním. Můžeme tam vidět přerušovanou křivku popisující data v souboru vzor1.dat.



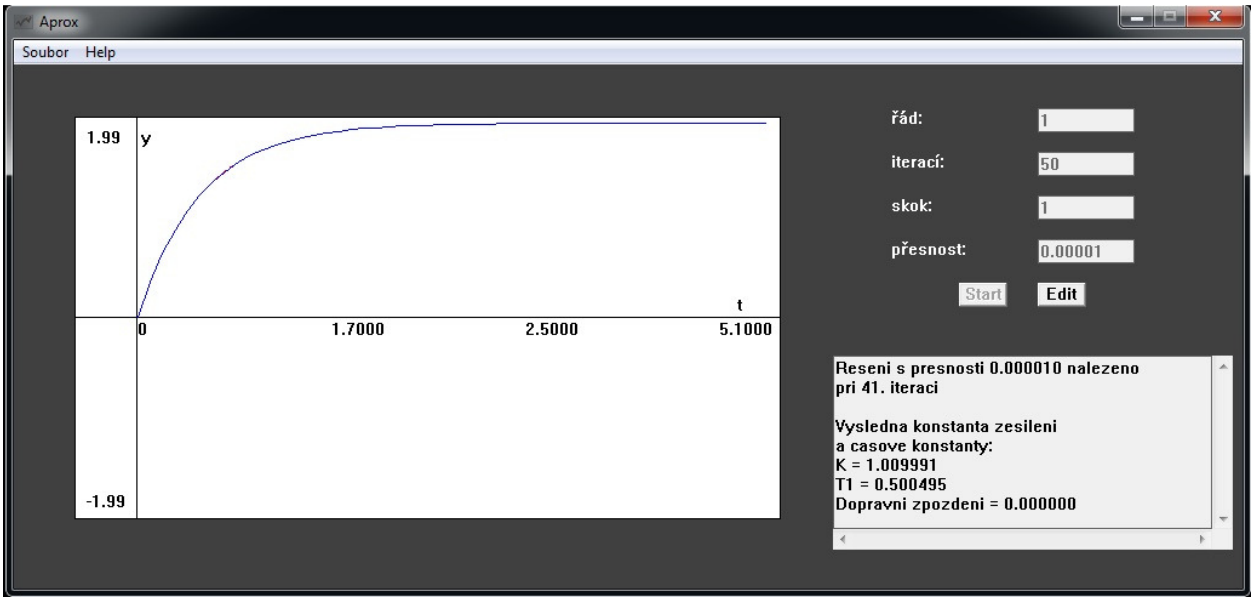
Obr. 22: Před spuštěním

8.3 Výstup

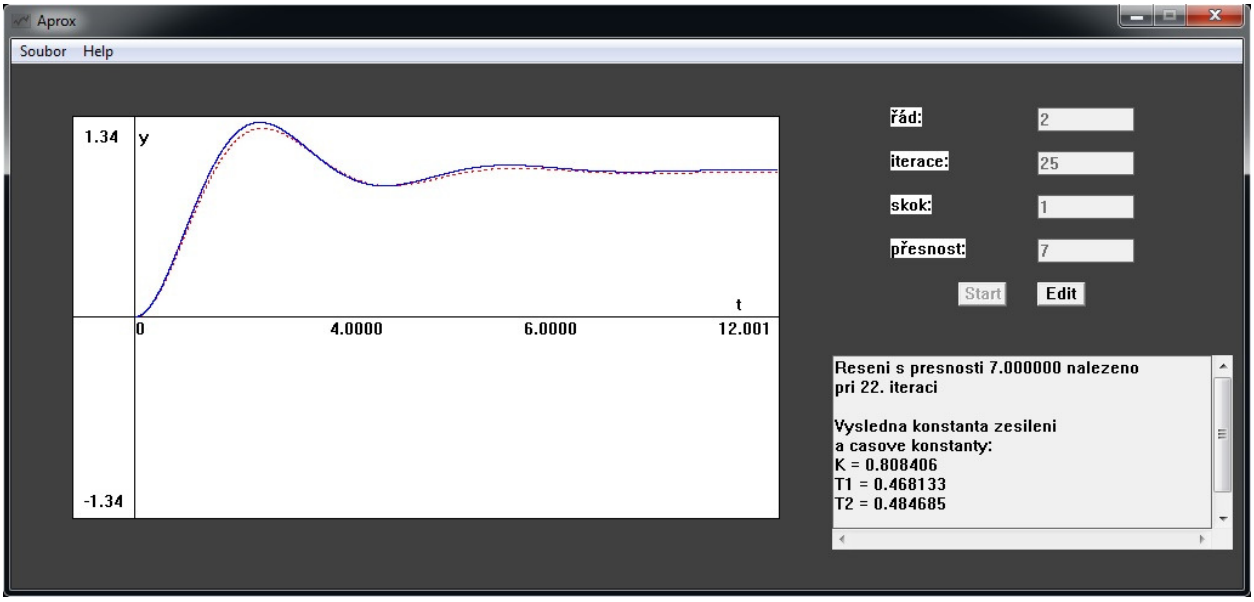
Výstup je zaznamenán v editačním poli vpravo dole, kde se vypíše to s jakou přesností jsme skončili, pokud se řešení v zadaném počtu iterací nenalezlo. Když je hledání úspěšné, tak výstupem je oznámení toho při kolikáté iteraci došlo k nálezu. Dále pak jsou vypsány hledané konstanty jak bylo zmíněno ve 2.3.

V obou případech je vykreslena křivka vypočtených hodnot v porovnání s křivkou hodnot načtených ze souboru, takže je možné sledovat vývoj toho jak se přibližujeme či vzdalujeme od řešení a přizpůsobit tomu námi zadávaný vstup. Na obr. 23 je ukázka výstupu, kde si můžeme všimnout toho, že se z přerušované vstupní křivky stala překrytím křivka nepřerušovaná. Řešení překreslilo hodnoty vstupního souboru. Na obr. 24 je vyobrazena křivka popisující data ze souboru vzor2.dat. V tomto případě byly nižší nároky na přesnost. Oba tyto soubory je možné nalézt na přiloženém kompaktním disku.

Výstup je zapsán také do tří textových souborů. V prvním jsou obsaženy hodnoty hledaných konstant, ve druhém body tvořící polyedr a jejich změna v průběhu výpočtu a ve třetím hodnoty $y(t)$ získané výpočtem RKDP z nalezených konstant. Data do souborů jsou uložena poté, co doběhne algoritmus. Všechny tyto soubory jsou uloženy do stejného adresáře jako soubor s naměřenými daty, ze kterého bylo čteno.



Obr. 23: Výstup-vzor1.dat



Obr. 24: Výstup-vzor2.dat, nižší přesnost

9 ZÁVĚR

V rámci bakalářské práce byla vytvořena aplikace, která má jako vstup textový soubor s naměřenými hodnotami. Pro něj pak hledá funkci, která se naměřeným hodnotám v bodech přibližuje alespoň natolik, aby byly splněny uživatelovy nároky na přesnost. Aplikace je ovladatelná z grafického uživatelského rozhraní.

Vstupní hodnoty udávají, jak se bude aplikace chovat během aproximace a je možné je zadávat editačními poli. Po vyplnění všech polí a úspěšném načtení souboru je možné aplikaci spustit.

Výstup je řešen zobrazením dvou funkcí v jednom grafu. Přibližnou funkcí tvořenou spojnicemi bodů dat načtených souborem a funkcí tvořenou spojnicemi bodů dat, která jsou výsledkem výpočtu algoritmu. Je možné tak sledovat přibližování se k řešení a podle toho lépe odhadnout podobu vstupu. Odhadu napomůže také fakt toho, že se v editačním poli při neúspěšném hledání zobrazí odchylka, kterou výpočet končil.

Při ukončení výpočtu se zobrazí v editačním poli výpis ze souboru, který obsahuje hodnoty nalezených konstant.

SEZNAM POUŽITÉ LITERATURY

[1] PRESS, W. H., S. A. TEUKOLSKY, W. T VETTERLING a B. P. FLANNERY. *Numerical Recipes in C*: *The Art of Scientific Computing*. New York: Cambridge, 1988. 2. ISBN 0-521-4108-5.

[2] FAJMON, B. a I RŮŽIČKOVÁ. *Matematika 3* [online]. Brno: UMAT FEKT, 2004 [cit. 2012-09-06]. Dostupné z: http://www.umel.feec.vutbr.cz/VIT/images/pdf/studijni_materialy/bc/Matematika3.pdf

[3] DORMAND, J. R. a P. J. PRINCE. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics* [online]. 1980, vol. 6, no 1. [cit. 2012-09-06]. Dostupné z: http://edu.chem.tue.nl/6KM06/Background%20reading%20material/DormandJR_1980.pdf

[4] KIMURA, T. On Dormand-Prince Method. In: [online]. 2009 [cit. 2012-09-09]. Dostupné z: http://toshi1224.info/physics/dopri.pdf

[5] LAGARIAS, J. C., J. A. REEDS, M. H. WRIGHT a P. E. WRIGHT. Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal of Optimization* [online]. 1998, vol.9, no. 1 [cit. 2012-09-06]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.120.6062&rep=rep1&type=pdf

[6] KUPKA, L. Simplexová metoda. In: [online]. [cit. 2012-09-09]. Dostupné z: http://www.fm.tul.cz/~libor.kupka/Simplexova_metoda.pdf

[7] HEROUT, P. *Učebnice jazyka C*. páté vydání. České Budějovice: Kopp, 2008. ISBN 978-80-7232- 351-7.

[8] HEROUT, P. *Učebnice jazyka C*: *2. díl*. třetí vydání. České Budějovice: Kopp, 2007. ISBN 978-80-7232-329-6.

[9] PETZOLD, Ch. *Programování ve Windows*: *Legendární publikace o programování Win32 API*. Praha: Computer Press, 1999. ISBN 80-7226-206-8.

[10] OLEHLA, M. a S. NĚMEČEK. *Základy aplikované kybernetiky*. 2. vyd. Liberec: Technická univerzita v Liberci, 2005. ISBN 80-7083-952-X.

[11] ŠVARC, I. *Základy Automatizace* [online]. Brno: CERM, 2002 [cit. 2012-09-06]. Dostupné z: http://web.tuke.sk/sjf-kaar/stranky/Predmetove_str/TK/material/Prednasky/Zaklady_Automatizace.pdf

[12] ČERMÁK, L. a R. HLAVIČKA. Numerické metody. In: [online]. 23. ledna 2006 [cit. 2012-10-09]. Dostupné z: http://mathonline.fme.vutbr.cz/UploadedFiles/246.pdf

SEZNAM OBRÁZKŮ

Obr. 1: Systém se vstupem a výstupem 13
Obr. 2: Jednotkový skok 14
Obr. 3: Ukázka přechodové charakteristiky 14
Obr. 4: Přesné a přibližné řešení 17
Obr. 5: Směrové pole 19
Obr. 6: Přibližné řešení diferenciální rovnice Eulerovou metodou 19
Obr. 7: První modifikace Eulerovy metody 20
Obr. 8: Druhá modifikace Eulerovy metody 21
Obr. 9: Odchylky 27
Obr. 10: Hledáme přímku s minimálním součtem obsahů čtverců 27
Obr. 11: Základní body 31
Obr. 12: Odraz 31
Obr. 13: Prodloužení 32
Obr. 14: Vnitřní zkrácení 33
Obr. 15: Zmenšení 33
Obr. 16: Algoritmus RKDP 35
Obr. 17 Algoritmus NM 42
Obr. 18: Náhled na aplikaci 49
Obr. 19: Podoba vstupního souboru 50
Obr. 20: Otevřít 50
Obr. 21: Načíst 51
Obr. 22: Před spuštěním 52
Obr. 23: Výstup-vzor1.dat 53
Obr. 24: Výstup-vzor2.dat, nižší přesnost 53

PŘÍLOHA

Jako příloha je vloženo CD, které obsahuje:

- bakalářskou práci v elektronické podobě v souboru BP.pdf
- spustitelnou aplikaci „Aprox.exe“
- zdrojové soubory „AproxFce.c“,“AproxWinMain.c“, „Resource.h“, „Aprox.rc“, „graf.ico“
- zkušební datové soubory s naměřenými daty „vzor1.dat“, „vzor2.dat“, „vzor3.dat“, „vzor4.dat“, „vzor5.dat“ a „vzor6.dat“
- instalaci knihovny „msvcr100d.dll“, nezbytné ke spuštění aplikace, v souboru „dffsetup-msvcr100d.exe“