JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

PEDAGOGICKÁ FAKULTA

Katedra matematiky

METODY APROXIMACE REÁLNÝCH KOŘENŮ UŽITÍM NÁSTROJŮ PROGRAMU MAPLE

DIPLOMOVÁ PRÁCE

Vypracovala: Jana Kotrejchová

Vedoucí práce: Mgr. Roman Hašek, Ph.D.

České Budějovice, duben 2007

Prohlášení:

Prohlašuji, že jsem na diplomové práci pracovala samostatně a že jsem uvedla veškerou literaturu, kterou jsem v této práci použila.

V Českých Budějovicích, dne 25.4.2007 ……..

ANOTACE

Diplomová práce nabízí čtenáři základní přehled nástrojů Maple 9.5 se zaměřením na metody aproximace reálných kořenů polynomiálních funkcí. Kromě vestavěných funkcí a Mapletů budou ukázány a použity vlastnoručně naprogramované Maplety. Práce může být použita jako vhodná pomůcka k pochopení problematiky, výuce, ale i následnému využití při řešení úloh.

ANNOTATION

The thesis offers the reader a general view of Maple 9.5 tools with orientation to methods of approximation in real roots of polynomial functions. Beside built-in functions and Maplets, autographically programmed Maplets will be shown and used. The work can be used as a suitable device for understanding problems, teaching and also as an application in problem solving exercises.

Poděkování:

Děkuji Mgr. Romanovi Haškovi, Ph.D., vedoucímu mé diplomové práce, za podnětné rady a připomínky, kterými mi pomohl při jejím vypracování.

Obsah

Úvod

Maple, ale i jiné matematické programy, v současné době pronikají do výuky. Jednak jako názorné pomůcky (například Cabri – představitel skupiny programů DGC – dynamic geometry system), ale i jako rozsáhlé nástroje pro řešení velkého množství matematických problémů. Mezi ně právě patří Maple (pro moji práci specielně Classic Worksheet Maple 9.5), ale i třeba Matlab. Maple je programem typu CAS (computer algebra system), který umožňuje symbolické a numerické výpočty, kreslení 2D a 3D grafů a programování.

Maple 9.5 jsem si vybrala z důvodu jeho přehlednosti, jednoduchosti, ale i jeho obrovskému potencionálu, který se navíc verzí 8 a následně 9.5 rozrostl o možnost programovaní vlastních aplikací – tzv. Mapletů.

Cílem této diplomové práce je předvést základní přehled nástrojů programu (jak vestavěných, tak vlastnoručně napsaných). Zaměřila jsem se na oblast lineární algebry, a to na metody aproximace reálných kořenů polynomiálních funkcí. Předvedené nástroje mohou být využity při řešení úloh i při výuce těchto metod.

Po úvodní části textu, kde se seznámíme se základní teorií polynomů, bude následovat příklad, na kterém budu všechny metody demonstrovat. Dále ve své práci uvedu tři metody. Všechny budou mít následující schéma: teoretická část, vizualizace metody – dynamické znázornění geometrického významu metod za pomoci Mapletů (v podstatě se bude jednat o ukázku prvních několika kroků) a část věnovaná příkladu. Ta bude opět členěna do čtyř částí – ověření předpokladů, grafické (geometrické) řešení, numerické přímé (tzv. blackbox – zajímají nás pouze výsledky a ne postup) a numerické postupné (zastupuje počítání „v ruce“). V kapitole 6 uvedu přehled dosažených výsledků a poslední kapitola bude věnovaná tvorbě Mapletů.

V příloze přikládám přehled nejdůležitějších funkcí pro práci s polynomy. Mým původním záměrem bylo uvést celý zdrojový kód Mapletů v další příloze. Ale musela jsem od toho ustoupit z důvodu velkého rozsahu, nepřehlednosti a navíc zbytečného opakování některých funkcí. Maplety totiž využívají funkce, které jsou společné pro všechny a pak ty, které jsou společné pro určitou metodu. Problém jsem tedy vyřešila následovně, procedury jsem rozdělila na ty, co jsou využívány všemi Maplety (příloha B) a ty, co jsou specifické pro některou z metod (příloha C – E). V těchto přílohách jsou uveřejněny i kódy Mapletů, jako jednotlivé podkapitoly. Poslední příloha obsahuje CD s diplomovou prací a všemi zdrojovými kódy a hotovými Maplety.

1 Teorie

Na začátek uvádím několik vět a definic, které jsou nezbytné pro další práci. Převzala jsem je z literatury [8] - [10] a internetových zdrojů [1] a [7]. Ty, které neuvádím, naleznete v této literatuře a zdrojích.

1.1 Základní věty a definice

Definice 1. Nechť n je přirozené číslo a nechť $a_0, a_1, ..., a_n$ jsou reálné, resp. komplexní čísla. Funkce $P(x)$, kterou lze definovat pro všechna (reálná, resp. komplexní) čísla x předpisem

$$P(x) = a_0 x^n + a_1 x^{n-1} + ... + a_n = \sum_{i=0}^{n} a_i x^{n-i} , \qquad (1)$$

se nazývá *mnohočlen* (v jedné proměnné x s reálnými, resp. komplexními koeficienty). Místo mnohočlen se také říká *polynom* nebo *celá racionální funkce*. Čísla $a_0, a_1, ..., a_n$ se nazývají *koeficienty mnohočlenu* $P(x)$. (Rektorys [9], s. 19)

Definice 2. *Stupněm mnohočlenu* $P(x)$ nazýváme nejvyšší mocninu proměnné x ve výrazu (1), u níž je nenulový koeficient. Je-li v (1) $a_0 \neq 0$, pak $P(x)$ je n-tého stupně.

(Rektorys [9], s. 20)

Definice 3. *Kořenem* (*nulovým bodem*) mnohočlenu $P(x) = \sum_{i=0}^{n} a_i x^{n-i}$ nazýváme takové (obecně komplexní) číslo α, pro něž $P(\alpha) = \sum_{i=0}^{n} a_i \alpha^{n-i} = 0$. (Rektorys [9], s. 21)

Věta 1 (základní věta algebry). Každý mnohočlen stupně $(n \geq 1)$ má aspoň jeden kořen. (Rektorys [9], s. 21)

Věta 2. Má-li mnohočlen $P(x)$ kořen α, pak $P(x)$ je dělitelný (tj. dělitelný beze zbytku) lineárním mnohočlenem $x-\alpha$, a naopak. [$x-\alpha$ je tzv. kořenový činitel mnohočlenu $P(x)$.] (Rektorys [9], s. 21)

1.2 Odhad počtu reálných kořenů

Věta 3 (Descartesova). Počet kladných kořenů rovnice $f(x) \equiv a_0 x^n + a_1 x^{n-1} + ... + a_n = 0$ je buď roven počtu znaménkových změn v posloupnosti $a_0, a_1, ..., a_n$, nebo je o sudý počet menší. (Rektorys [10], s. 605)

Poznámka 1. Počet znaménkových změn dostaneme tak, že v posloupnosti, kterou získáme z dané posloupnosti vynecháním všech nul, určíme počet dvojic po sobě jdoucích čísel, která jsou opačného znaménka. (Rektorys [10], s. 605)

Poznámka 2. Obdobný postup je pro zjištění počtu záporných kořenů. Použijeme pomocný polynom $R(x) = P(-x)$. Počet záporných kořenů polynomu $R(x) = b_0 x^n + ... + b_{n-1} x + b_n$ je buď roven počtu znaménkových změn v posloupnosti $b_0, b_1, ..., b_n$ jeho koeficientů nebo je o sudý počet menší.

Věta 4. Všechny reálné kořeny polynomu $P(x) = a_0 x^n + ... + a_{n-1} x + a_n$ leží v intervalu $\langle -1 - A; 1 + A \rangle$, kde $A = \max(|a_0|, |a_1|, ..., |a_n|)$. (internetový zdroj [1])

1.3 Separace kořenů

Separace kořenů je způsob jak najít intervaly, v nichž leží právě jeden kořen polynomu $P(x)$ (v některých větách budeme zaměňovat $f(x)$ a $P(x)$). Lze použít jakékoliv dostupné metody. Pro moji práci jsem zvolila právě tři, a to: metoda půlení

intervalu (bisekce), metoda tečen (Newtonova metoda) a jako poslední metoda tětiv (regula falsi). Jejich teoretický výklad bude uveden v každé kapitole, zabývající se tímto problémem.

Věta 5. Nechť $a < b$ a pro funkční hodnoty $f(a), f(b)$ polynomu s reálnými koeficienty $f(x)$ platí vztah $f(a)f(b) < 0$. Jestliže derivace $f'(x)$ nemění v intervale (a,b) znaménko, pak má rovnice $f(x) = 0$ v intervale (a,b) právě jeden reálný kořen.

(Mrkvička [8], s. 20)

Poznámka 3. V následujícím textu bude někdy zaměňováno slovo funkce se slovem polynom (mnohočlen). Nedochází k žádné újmě na obecnosti, neboť polynom patří do polynomiálních funkcí (též polynomické funkce), které jsou též označovány jako celé racionální funkce (viz definice 1.). Tedy při použití slova funkce, lze postup rozšířit na veškeré funkce. Jen je potřeba brát ohledy na definiční obory. U polynomů je definiční obor $D(f) = R$.

Poznámka 4. Všechny polynomické funkce, exponenciální funkce, sinus a kosinus a funkce absolutní hodnota jsou spojité v celém oboru reálných čísel.

(internetový zdroj [7])

Poznámka 5. Občas se setkáváme s následujícím dělením metod:

- startovací metody
    - metoda půlení intervalu (bisekce)
    - metoda prosté iterace
    - metoda tětiv (regula falsi)
- zpřesňující metody
    - Newtonova metoda (metoda tečen nebo metoda linearizace)
    - Metoda sečen (tzv. dvoukroková interpolační metoda)
    - Müllerova metoda (tzv. tříkroková interpolační metoda)
- speciální metody

Tímto výčet nekončí, například Rektorys [10], s. 607 – 617 uvádí další metody (Bernoulliova-Whittakerova metoda, Gräffova metoda a její modifikace, Bairstowova metoda, Obecná iterační metoda) .

2 Příklad

2.1 Zadání

Všechny metody budu demonstrovat na jednom jednoduchém příkladu. Čtenář tak může porovnat jednotlivé postupy a vybrat si tu metodu, která ho nejvíce zaujala. Případně si může vytvořit strategii na řešení dalších příkladů (například, že začne s jednou metodou a po nějaké době nasadí druhou). A také proto, aby se základní interval nemusel počítat několikrát.

Polynom, který jsem vybrala je: $P(x) = 4x^3 + 11x^2 + 5x - 2$. Úkolem je aproximovat všechny kladné reálné kořeny.

2.2 Určení výchozího intervalu

Abychom mohli použít některou z metod, potřebujeme vědět, kde přibližně hledaný kořen leží. Spokojíme se s intervalem o velikosti menší nebo rovno 1. Pro jednotlivé metody může dojít k jeho zmenšení.

Nejprve musíme určit posloupnost $a_0, a_1, ..., a_n$ z $P(x) = a_0 x^n + ... + a_{n-1} x + a_n$. V našem případě a_0, a_1, a_2, a_3 z polynomu $P(x) = 4x^3 + 11x^2 + 5x - 2$. To znamená: $a_0 = 4, a_1 = 11, a_2 = 5, a_3 = -2$. Vidíme, že je zde pouze jedna znaménková změna (viz. poznámka 1.). Z věty 3. vyvozujeme závěr: počet kladných reálných kořenů polynomu *P(x)* je jeden.

Pro úplnost uvádím způsob získání počtu záporných kořenů polynomu. Nejprve zavedeme pomocný polynom $R(x) = P(-x)$. Najdeme posloupnost $b_0, b_1, ..., b_n$ z polynomu $R(x) = b_0 x^n + ... + b_{n-1} x + b_n$ a určíme počet znaménkových změn. Tedy: $R(x) = 4(-x)^3 + 11(-x)^2 + 5(-x) - 2 = -4x^3 + 11x^2 - 5x - 2$ a dostáváme posloupnost

$b_0 = -4, b_1 = 11, b_2 = -5, b_3 = -2$. Vidíme, že dochází ke dvou znaménkovým změnám, tzn. podle poznámky 2. existují dva nebo žádný záporný reálný kořen.

Nyní určíme základní interval, dle věty 4., k tomu potřebujeme znát $A = \max(|a_0|, |a_1|, ..., |a_n|)$, tedy $A = \max(|4|, |11|, |5|, |-2|) = 11$. Interval je stanoven na $\langle -1 - A; 1 + A \rangle = \langle -1 - 11; 1 + 11 \rangle = \langle -12; 12 \rangle$.

Jelikož nás zajímají pouze kladné kořeny, interval zmenšíme na $\langle 0; 12 \rangle$.

K určení menšího intervalu se snažíme použít větu 5., k tomu je potřeba splnit její předpoklady.

Sestavíme tabulku funkčních hodnot:

x	P(x)	znaménko
0	-2	–
1	18	+
2	84	+
3	220	+
4	450	+
5	798	+
6	1288	+

x	P(x)	znaménko
7	1944	+
8	2790	+
9	3850	+
10	5148	+
11	6708	+
12	8554	+

Vypočítáme derivaci *P(x)*: $P'(x) = 12x^2 + 22x + 5$. V tomto jednoduchém příkladu vidíme, že $P'(x)$ na intervalu $\langle 0; 12 \rangle$ nemění znaménko (po dosazení kladného čísla bude $P'(x)$ větší než nula). Ve složitějších případech je nutné najít kořeny $P'(x)$. Z tabulky funkčních hodnot vidíme, že mezi 0 a 1 dochází ke znaménkové změně, neboli že $P(0)P(1) = -36 < 0$.

Všechny předpoklady pro $a = 0,\ b = 1$ byly splněny, proto můžeme stanovit závěr, že v $(0;1)$ leží právě jeden reálný kořen. Samozřejmě, že interval můžeme zvolit i jinak, např. $(0;4)$. Výhodnější ale je, aby byl co nejmenší.

Závěr. Pro svoji pozdější práci zvolíme uzavřený interval $\langle 0; 1 \rangle$.

Poznámka. Přidáním krajních bodů nedošlo k porušení žádných podmínek, protože víme, že v $\langle 0;12 \rangle$ leží právě jeden reálný kořen. A interval $\langle 0;1 \rangle$ je jeho částí a dále, že $(0;1)$ obsahuje výše zmíněný kořen.

2.3 Maple 9.5

V této podkapitole uvedu několik základních příkazů z Maple 9.5, které jsou nezbytné pro řešení rovnic (v našem případě je rovnicí myšlen výraz $P(x) = 0$). Předpokládám čtenářovu základní znalost syntaxe a příkazů (jako `plot`, `display` a další).

Pro zjednodušení další práce si polynom označím jako:

```
> pol:=4*x^3+11*x^2+5*x-2;
```

$$pol := 4x^3 + 11x^2 + 5x - 2$$

Prvním ze základních příkazů je `isolate`, který nám vyjádří proměnnou z rovnice. Nevýhodou je, že takto dostaneme pouze první kořen.

1. parametr – rovnice

2. parametr – proměnná, kterou chceme z rovnice vypočítat

```
> isolate(pol=0,x);
```

$$x = -1$$

Silnějším prostředkem je funkce `solve`. Hlavní jeho výhodou vůči příkazu `isolate` je možnost řešit soustavy rovnic o více neznámých. Příkaz `solve` také na rozdíl od `isolate` hledá i další kořeny u těch rovnic, které mají více řešení. Jeho další výhodou je schopnost řešit nerovnice, což `isolate` neumožňuje.

Parametry jsou stejné jako u `isolate`.

```
> solve(pol=0,x);
```

$$-1, -2, \frac{1}{4}$$

Dalším užitečným příkazem je `evalf`. Jeho úkolem je vyčíslit odmocniny, zlomky a podobně.

```
> evalf(%);
        -1., -2., 0.2500000000
```

Přidáním dalšího parametru, nastavíme počet zobrazovaných desetinných míst.

```
> evalf(Pi,15);
        3.14159265358979
```

Obdobnou funkcí je `evalb` – místo vyčíslení, nás informuje o pravdivosti výrazu v závorkách.

```
> evalb(-1*(-1)>0);
        true
> evalb(-1*(-1)<0);
        false
```

Posledním příkazem je `subs`. Dosazuje zvolenou hodnotu za proměnnou do rovnice. Vhodné je to například pro zjišťování funkčních hodnot.

Popis parametrů:

1. parametr – je ve tvaru proměnná = hodnota
2. parametr – výraz, do kterého chceme dosadit

```
> subs(x=0,pol);
         -2
```

Poznámka. Maple 9.5 nám přímo neumožňuje zadávat proměnné s indexy. Proto se využívá alternativního řešení a to, že x_1 je zastoupeno $x[1]$. V textu tedy budeme pracovat s označením x_1 a v Maple 9.5 $x[1]$ nebo *x_1*.

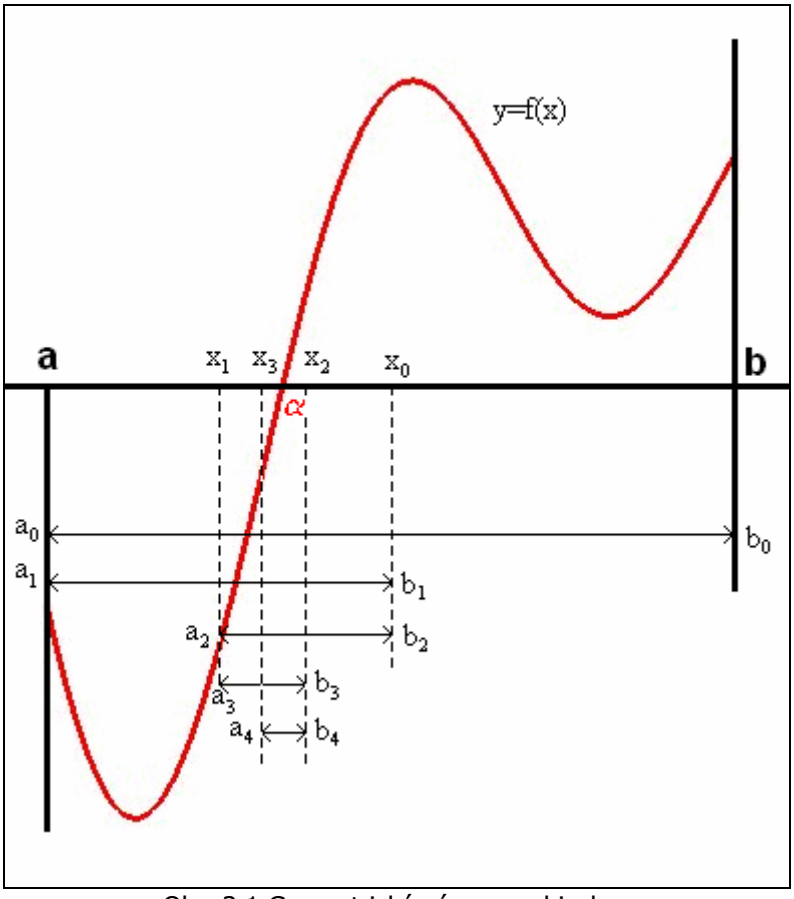
Poznámka. Další funkce pro práci s polynomy jsou v příloze A.

3 Metoda půlení intervalu (bisekce)

3.1 Teorie

Předpokládejme, že v intervalu $\langle a,b\rangle$ leží právě jeden kořen rovnice $f(x)=0$. Polohu kořene lze zpřesnit rozpůlením intervalu a zjištěním, ve které části intervalu kořen α leží. Zmenšený interval, v němž leží kořen, lze znovu rozpůlit a tak pokračovat dále. Střed posledního sestrojeného intervalu, nebo některý z jeho krajních bodů, lze považovat za aproximaci kořene α. Přesný postup je popsán následující větou.

(internetový zdroj [3])



Obr. 3.1 Geometrický význam – bisekce

Věta. Nechť funkce $f(x)$ je spojitá v intervalu $\langle a,b\rangle$ a platí $f(a)f(b)<0$. Vytvořme posloupnost intervalů $\langle a_k,b_k\rangle$ a posloupnost bodů x_k, $k=0,1,2,...$ takto:

1. Pro $k=0$ položíme $a_0=a,\ b_0=b$.
2. Je-li definován interval $\langle a_k,b_k\rangle$ vypočteme $x_k=\frac{a_k+b_k}{2}$.
3. Je-li $f(x)=0$, další interval nehledáme. Je-li $f(a_k)f(b_k)<0$, položíme $a_{k+1}=a_k$ a $b_{k+1}=x_k$ jinak položíme $a_{k+1}=x_k$ a $b_{k+1}=b_k$.

Jestliže má posloupnost $\{x_k\}$ konečný počet členů, je poslední člen kořenem α rovnice $f(x)=0$. Jestliže je posloupnost $\{x_k\}$ nekonečná, má limitu a platí $\max_{k\to\infty} x_k=\alpha$.

(internetový zdroj [3])

Odhad chyby (určuje přesnost našeho výpočtu)

Při odhadu chyby u metody půlení intervalu si uvědomíme, že vzhledem ke spojitosti funkce f a platnosti $f(a_k)f(b_k)<0$ leží kořen ξ vždy uvnitř intervalu $\langle a_k,b_k\rangle$. Aproximujeme-li kořen ξ hodnotou x_k plyne z právě uvedeného a věty:

$$|\xi-x_k|<\frac{b_k-a_k}{2}.$$ (internetový zdroj [3])

Poznámka. Anglický překlad: bisection method, halving method.

Poznámka. V grafech nebudu přeznačovat krajní body intervalů. Označení bude následující: původní interval $\langle a;b\rangle$ a každý nově vzniklý bod bude x_k, $k=0,1,2,...$. Docházelo by k nepřehlednosti grafů.

Poznámka. Na rozdíl od následujících metod zde neuvádím odstavec s názvem Bisekce – krok za krokem z důvodu, že obr. 3.1 je názorný a není třeba dalšího komentáře.

3.2 Vizualizace metody

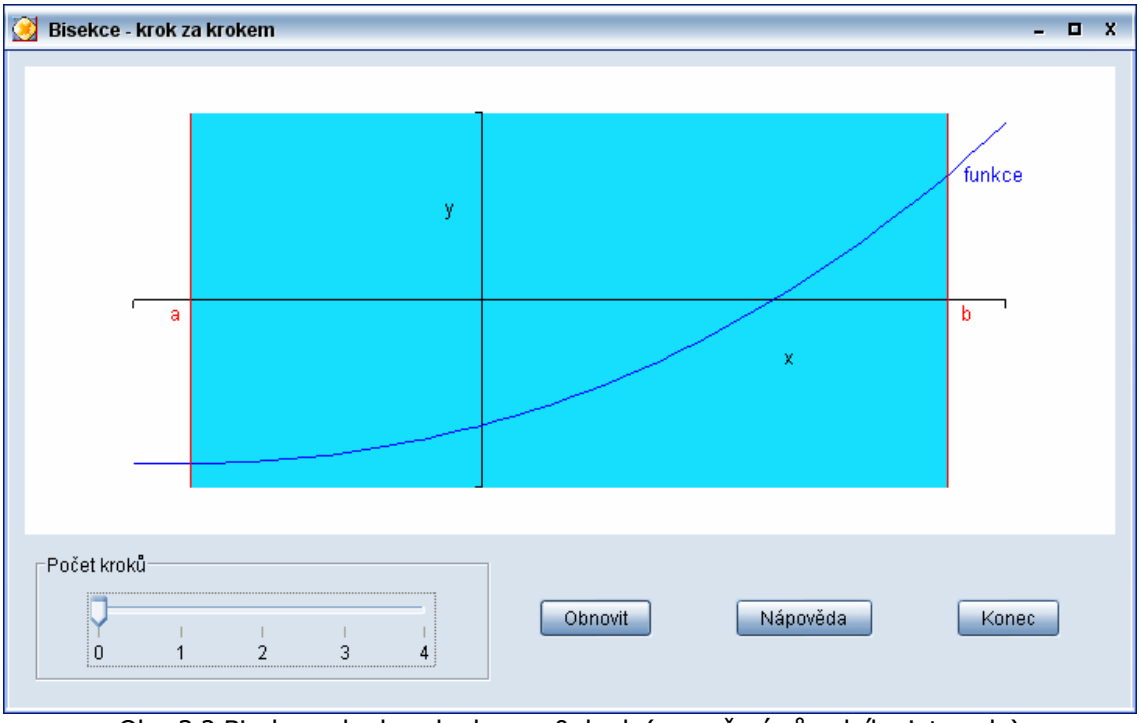
Abych objasnila princip této metody, vytvořila jsem Maplet „Bisekce – krok za krokem[1]“. Ten ukazuje jak se každým krokem blížíme ke kořeni (zmenšuje se interval).

Každý nově vzniklý interval je vyznačen odlišnou barvou a je o něco nižší než ten předchozí. Krok se určuje pomocí „posuvnítka“ v dolní části Mapletu – je v rozmezí od 0. (zadání) – 4. krok.

Nultý, první a třetí krok je znázorněn na obrázcích 3.2 až 3.4.

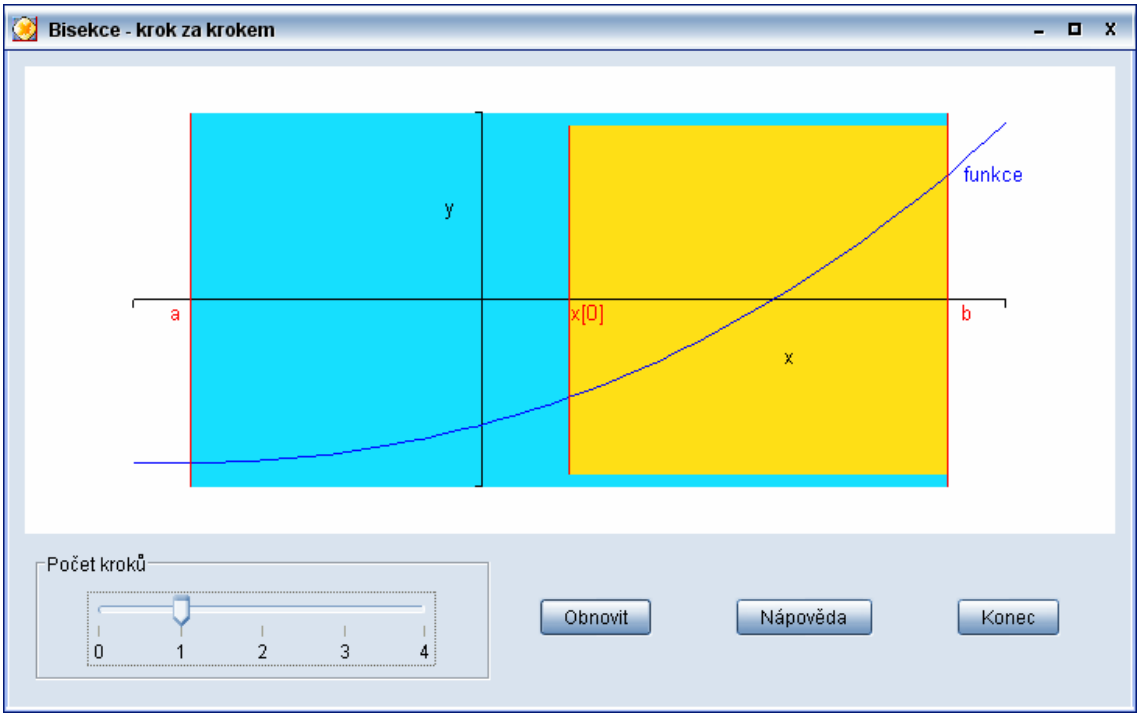
Maplet obsahuje 3 tlačítka:

- *Obnovit* – návrat do nultého kroku
- *Nápověda* – spustí okno se základním popisem Mapletu
- *Konec* – ukončí Maplet

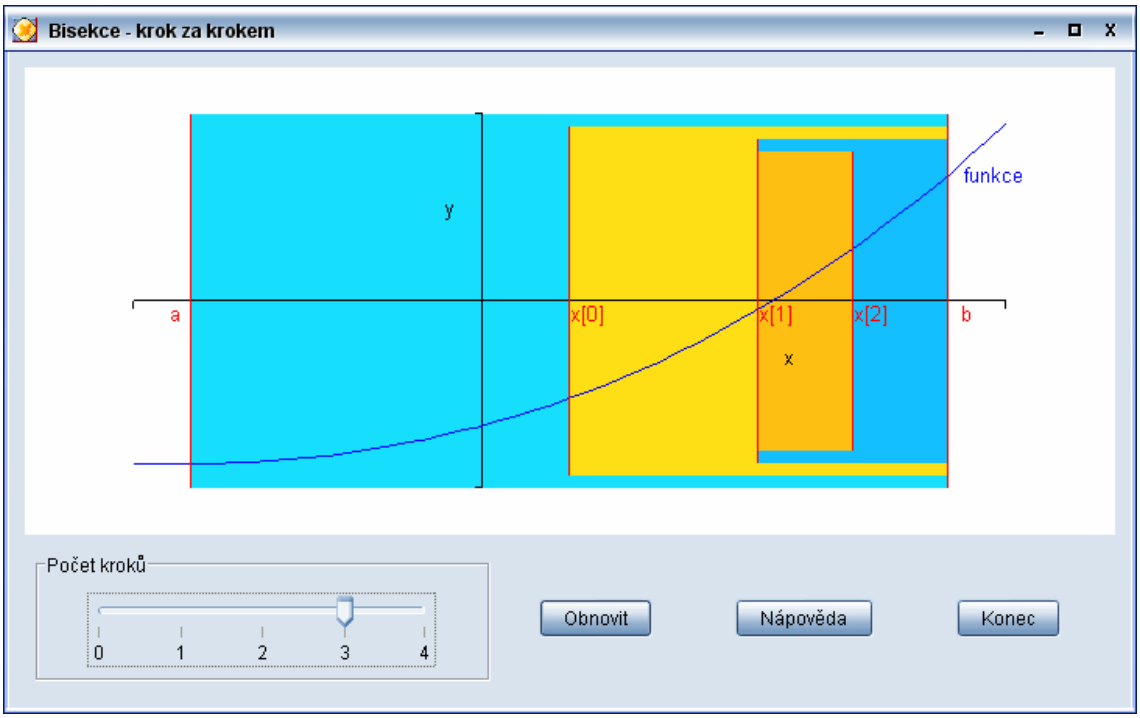


Obr. 3.2 Bisekce – krok za krokem – 0. krok (vyznačení původního intervalu)

[1] název souboru „3.2 Vizualizace\Bisekce – krok za krokem.maplet“
zdrojový kód je obsažen v: „3.2 Vizualizace\Bisekce – krok za krokem.mws“



Obr. 3.3 Bisekce – krok za krokem – 1. krok (první rozdělení intervalu)



Obr. 3.4 Bisekce – krok za krokem – 3. krok

Na obrázku 3.4 jsou vidět celkem 4 intervaly:

$\langle a;b\rangle$	původní interval	$\langle x_1;b\rangle$	po druhém půlení
$\langle x_0;b\rangle$	po prvním půlení	$\langle x_1;x_2\rangle$	po třetím půlení

3.3 Příklad

$$P(x) = 4x^3 + 11x^2 + 5x - 2$$

Dle kapitoly 2 budeme další výpočty stavět na intervalu $\langle 0;1 \rangle$, ve kterém leží právě jeden reálný kladný kořen ($a = 0$, $b = 1$).

3.3.1 Ověření splnění předpokladů metody

Nejprve musíme ověřit předpoklady. Označíme:

```
> pol:=4*x^3+11*x^2+5*x-2;
        pol := 4 x^3 + 11 x^2 + 5 x - 2
> a:=0;
  b:=1;
        a := 0
        b := 1
```

1) $f(a)f(b) < 0$

```
> f_a:=subs(x=a,pol);
        f_a := -2
> f_b:=subs(x=b,pol);
        f_b := 18
> evalb(f_a*f_b<0);
         true
```

2) $f(x)$ je spojitá v intervalu $\langle a;b \rangle$

$D(f) = R$, funkce je spojitá na celém oboru reálných čísel, tedy i na intervalu $\langle 0;1 \rangle$ (viz kapitola 1.3 poznámka 4.).

Závěr. Všechny předpoklady jsou splněny. Metodu můžeme použít.

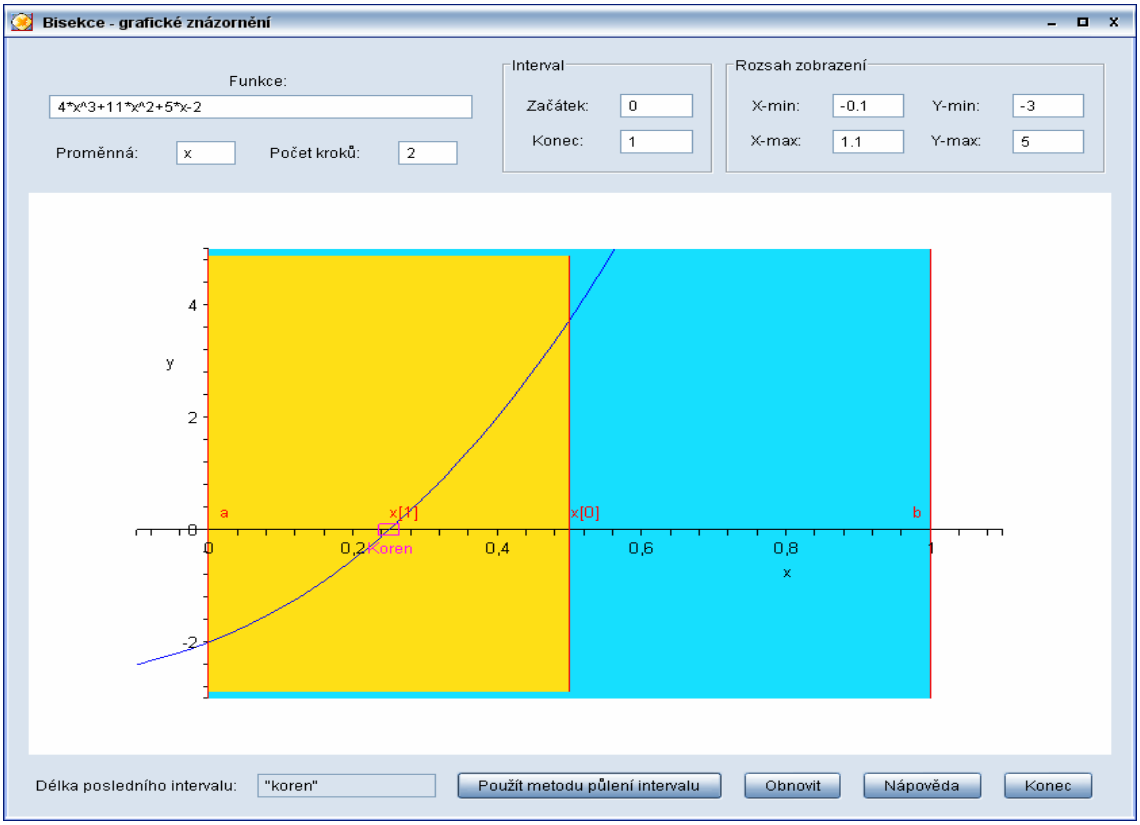
3.3.2 Grafické řešení

Maple 9.5 nám nenabízí žádné grafické řešení. Z tohoto důvodu jsem byla nucena naprogramovat Maplet „Bisekce – grafické znázornění[2]“. Je škoda, že tato metoda nebyla začleněna do Maple 9.5. Na druhou stranu mi to umožnilo zpracovat si celý matematický problém podle svého.

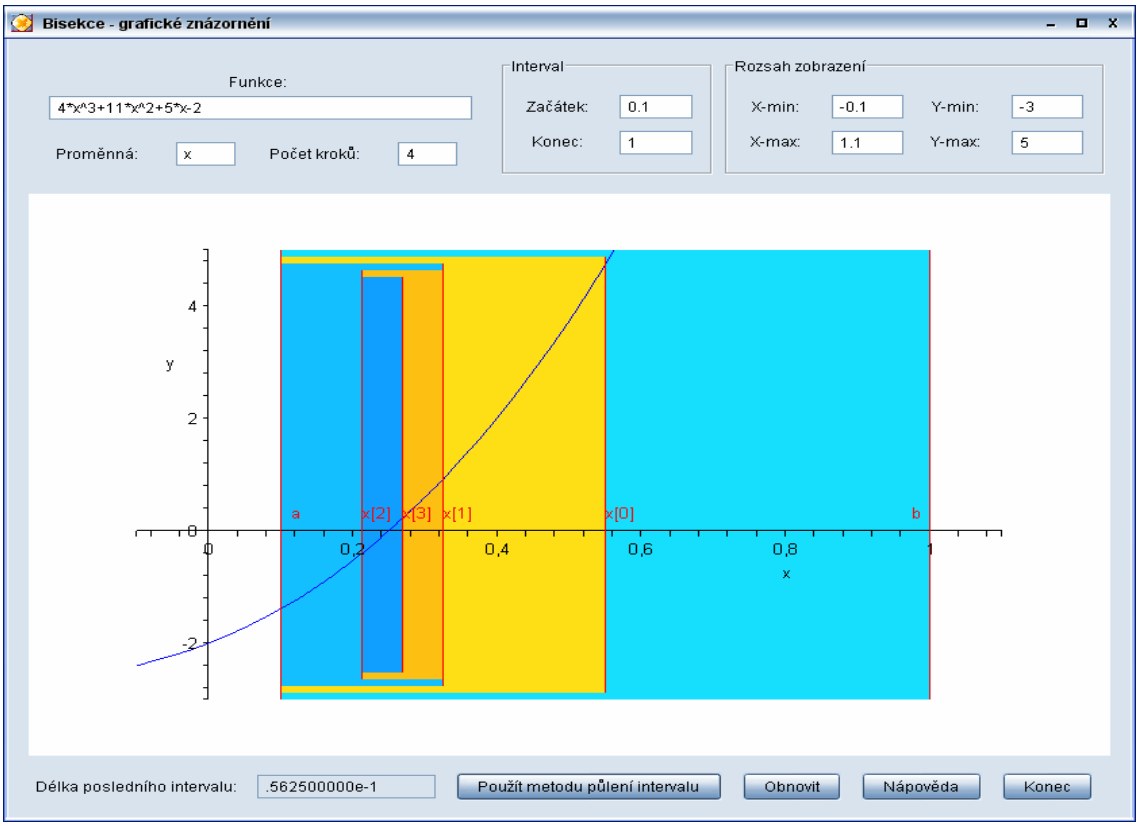
Popis jednotlivých částí Mapletu (znázorněn na obr. 3.5, 3.6):

- *Funkce*
- *Proměnná* – písmeno z abecedy, které je pro nás proměnnou (někdy je nutné použít jiné než *x*)
- *Počet kroků* – kolikrát bude použita metoda půlení intervalu (nultý krok znázorní původní interval) – jedná se o celé nezáporné číslo
- *Interval (Začátek, Konec)* – „startovní“ interval
- *X-min, X-max, Y-min, Y-max* – rozsah zobrazení (po každé změně je nezbytné graf překreslit pomocí tlačítka *Použít metodu půlení intervalu*)
- *Délka posledního intervalu* – vypisuje velikost posledně získaného intervalu. Pokud toto číslo vydělíme dvěma, dostaneme odhad maximální chyby. V případě nultého kroku je zobrazena velikost původního a pokud je nalezen kořen, zobrazí se text „koren“.
- Tlačítko *Použít metodu půlení intervalu* – zobrazí graf, tlačítko je nutné stisknout po každé změně údajů (funkce, proměnná, rozsah zobrazení apod.)
- *Obnovit* – nastaví původní hodnoty
- *Nápověda* – spustí okno s krátkým popisem Mapletu (obr. 3.7)
- *Konec* – ukončí Maplet

[2] název souboru „3.3 Priklad\Bisekce - graficke.maplet“
zdrojový kód je obsažen v: „3.3 Priklad\Bisekce - graficke.mws“

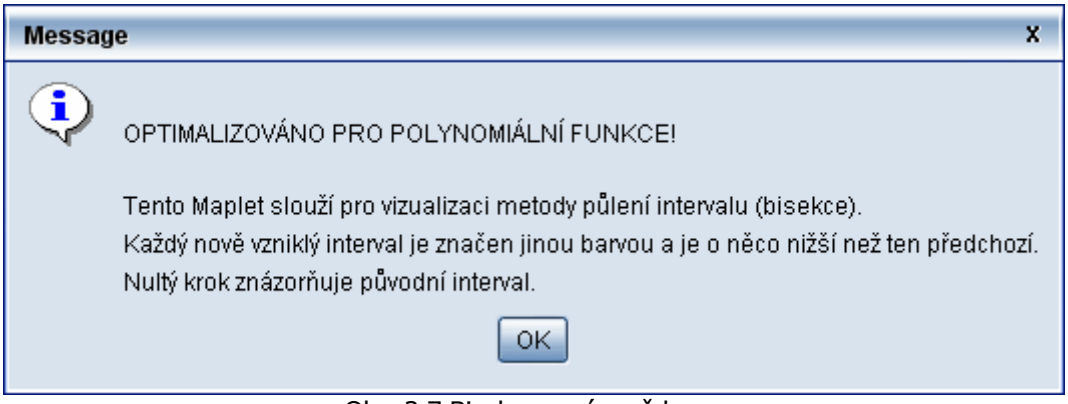


Obr. 3.5 Bisekce – grafické znázornění



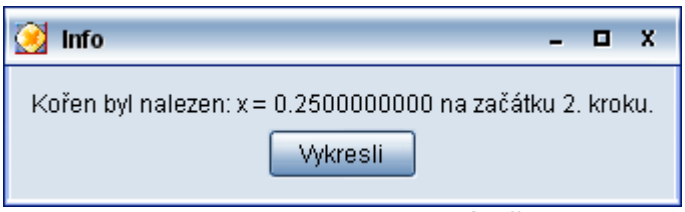
Obr. 3.6 Bisekce – grafické znázornění intervalu $\langle 0.1;1 \rangle$

Poznámka. Na obr. 3.5 a 3.6 je pěkně vidět, že efektivnost této metody velmi závisí na počáteční volbě intervalu. Přičemž velkou roli hraje zkušenost, náhoda, ale i štěstí.

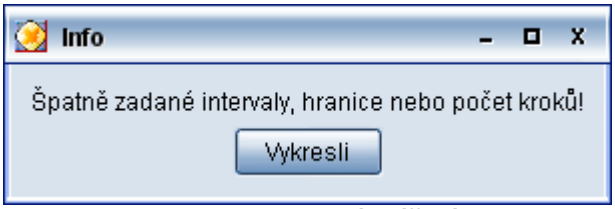


Obr. 3.7 Bisekce – nápověda

Maplet nás informuje nejen o chybném zadání intervalu, ale i o tom, že nejsou splněny předpoklady, nebo počet kroků není celé nezáporné číslo. Upozorní nás také, že jsme dosáhli, do zadaného počtu kroků, kořen. Viz následující obrázky.



Obr. 3.8 Informace o nalezení kořene



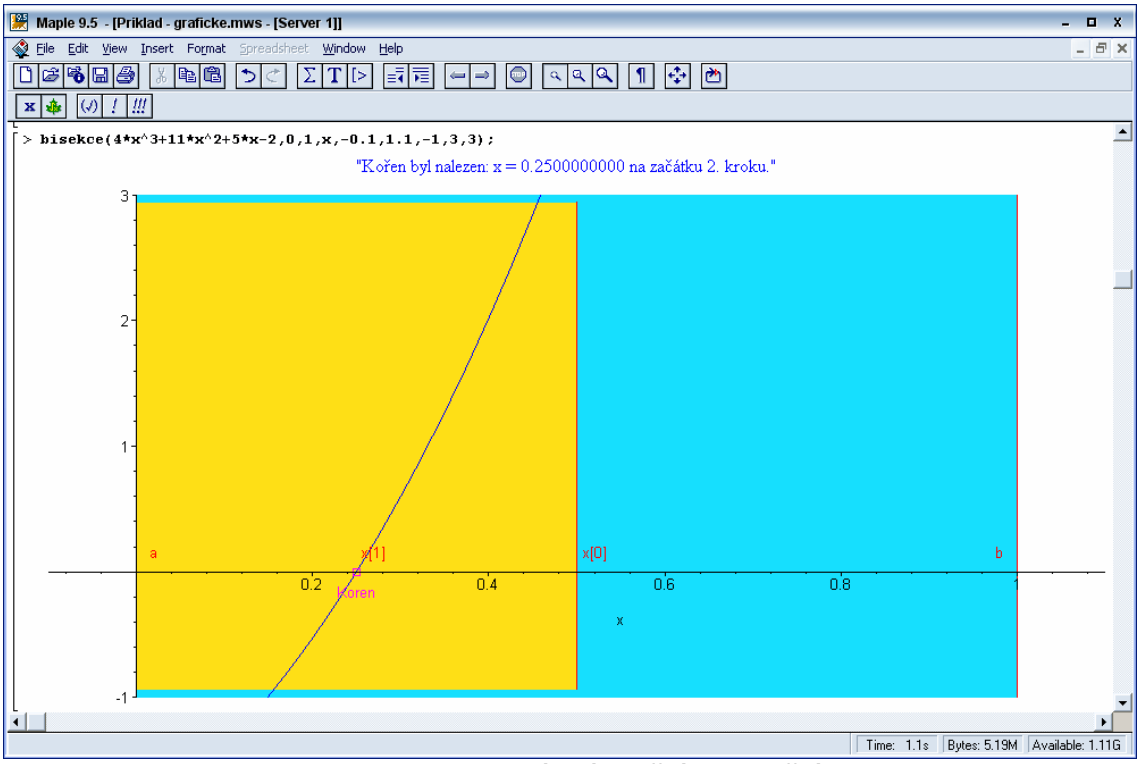
Obr. 3.9 Chybové hlášení

Toto grafické znázornění můžeme dostat i bez použití Mapletu. Stačí spustit soubor „3.3 Priklad\Priklad – graficke.mws“, provést „Execute the worksheet“ (zastoupeno ikonou ) a zadat příkaz `bisekce(Fce,a,b,prom,xmin,xmax,ymin,ymax,kroky)`. Výsledek je zobrazen na obr. 3.10.

Popis parametrů:

- `Fce` – funkce
- `a,b` – interval (`a` – začátek, `b` – konec)
- `prom` – písmeno z abecedy, které je pro nás proměnnou (někdy je nutné použít jiné než *x*)
- `xmin, xmax, ymin, ymax` – rozsah zobrazení
- `kroky` – počet kolikrát chceme použít metodu půlení intervalu (nultý krok znázorní původní interval)

```
> bisekce(4*x^3+11*x^2+5*x-2,0,1,x,-0.1,1.1,-1,3,3);
```

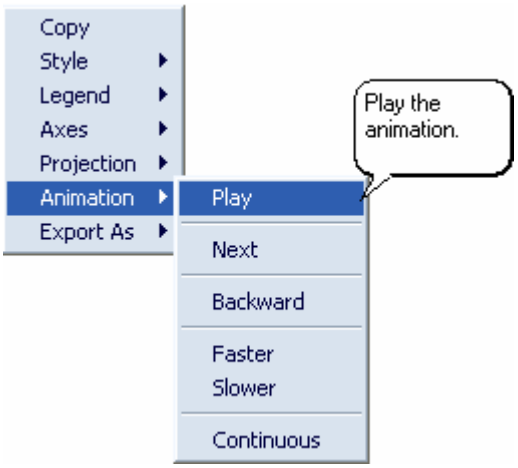


Obr. 3.10 Bisekce – grafické znázornění bez použití Mapletu

Posledním „kouzlem“, který můžeme provést, je použít funkci `animace(Fce,a,b,prom,xmin,xmax,ymin,ymax,kroky)`. Parametry mají stejný význam jako u předchozí procedury. Funkce je uložena v souboru „3.3 Priklad\Priklad – animace.mws“.

Po stisku klávesy Enter je zobrazen graf. Pokud na něj kliknete pravým tlačítkem, zvolíte nabídku *Animation* a *Play* (obr. 3.11), zobrazí se krátká animace,

postupně vykreslující zmenšující se interval, ve kterém se nachází kořen. Ovládání lze provádět i pomocí panelu nástrojů (obr. 3.12). Ten se zobrazí po té, co je graf vybrán myší.



Obr. 3.11 Animace – nabídka po stisknutí pravého tlačítka

Obr. 3.12 Animace – panel nástrojů

Poznámka. Výsledek jsem exportovala do „3.3 Priklad\Priklad – animace.gif“.

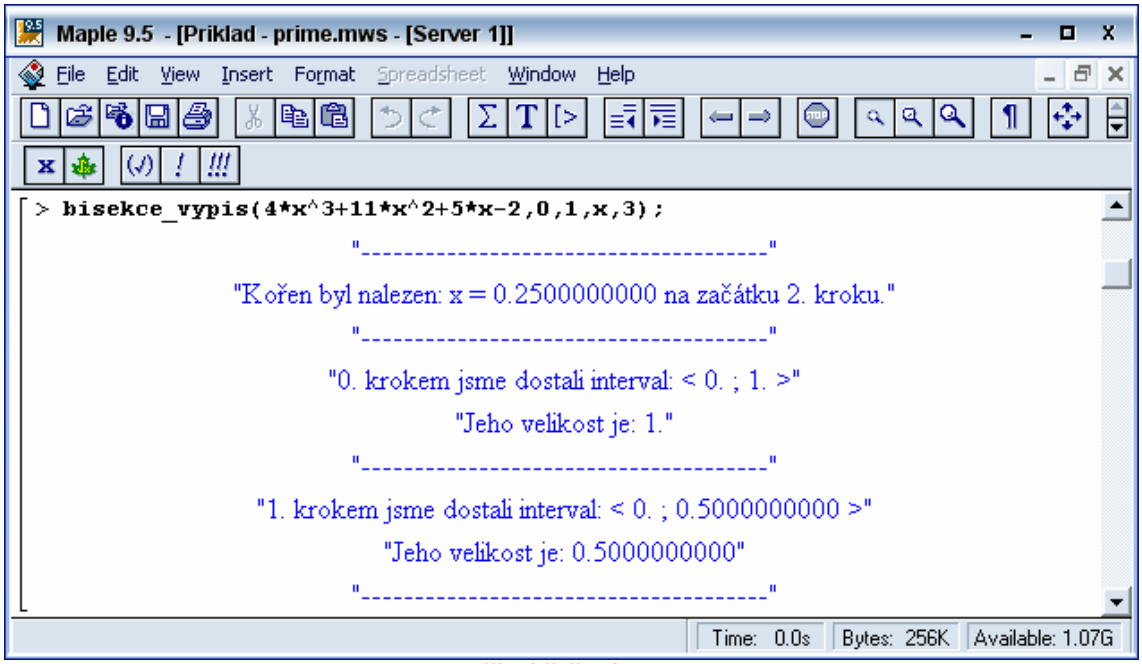
3.3.3 Numerické řešení (přímé)

Přímé řešení je někdy nazýváno „Blackboxem“ (v překladu: černou skříňkou). Protože nás nezajímá jak to funguje uvnitř, ale pouze výsledky.

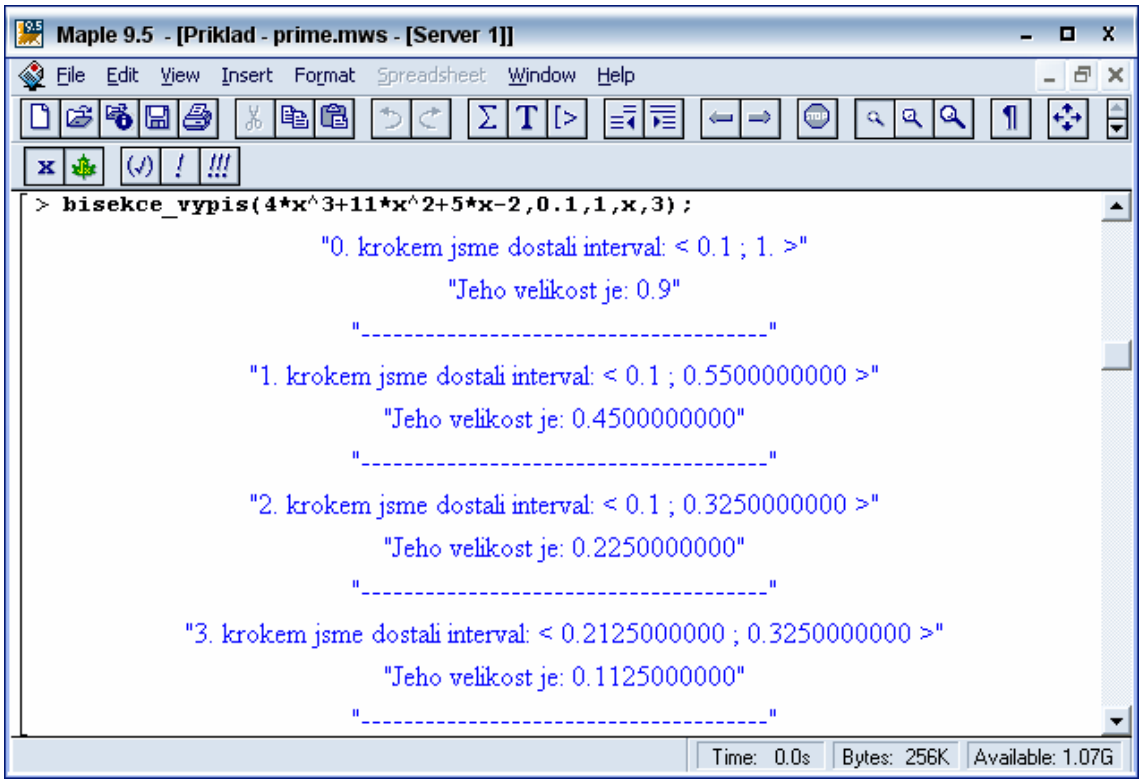
Po důkladném hledání v programu Maple 9.5 musím konstatovat, že jsem nenalezla přímé řešení. Proto jsem byla nucena takové řešení naprogramovat. Příkaz se nazývá `bisekce_vypis(Fce,a,b,prom,kroky)` v souboru „3.3 Priklad\Priklad – prime.mws“. Je potřeba vždy provést „Execute the worksheet“. Parametry mají stejný význam jako u předchozího příkazu. Jenom je jich o něco méně.

V případě, že byl zadán kořen jako jeden z krajních bodů intervalu, nebo se k němu po několika krocích dostaneme, je tato informace vypisována na hlavní obrazovku v textové podobě. Pod ní následuje výpis jednotlivých intervalů a jejich velikost (délka dělená dvěma nám tu symbolizuje maximální chybu). Při dosažení

kořene, není poslední interval vypisován. Jednotlivé kroky jsou od sebe odděleny vodorovnou čárou (obr. 3.13, 3.14).



Obr. 3.13 Bisekce – přímé řešení za pomoci bisekce_vypis



Obr. 3.14 Bisekce – přímé řešení na intervalu $\langle 0.1;1 \rangle$ za pomoci bisekce_vypis

3.3.4 Numerické řešení (postupné)

Postupné řešení zde symbolizuje klasické počítaní (tzv. „na papíře“) za pomoci výpočetní techniky. Předpoklady byly ověřeny na začátku kapitoly.

Poznámka. V tomto případě budeme dodržovat značení intervalů (dle věty). Je to z důvodu, že zde není použit žádný graf a tak nedochází k nepřehlednostem. A pro čtenáře to tak bude názornější a přehlednější. Bude mít lepší přehled o tom, o kterém intervalu se bavíme.

Pro kvalitní práci si nejdříve funkci a krajní body označíme:

```
> restart;
  with(linalg):
> pol:=4*x^3+11*x^2+5*x-2;
  a[0]:=0;
  b[0]:=1;
```

$pol := 4x^3 + 11x^2 + 5x - 2$

$a_0 := 0$

$b_0 := 1$

Zjistíme funkční hodnoty a_0, b_0:

```
> pol_a[0]:=subs(x=a[0],pol);
  pol_b[0]:=subs(x=b[0],pol);
```

$pol\_a_0 := -2$

$pol\_b_0 := 18$

```
> evalb(pol_a[0]*pol_b[0]<0);
```

true

Určíme velikost intervalu $\langle a_0; b_0 \rangle$:

```
> velikost[0]:=abs(b[0]-a[0]);
```

$velikost_0 := 1$

Vypočítáme nový bod x_0 a jeho funkční hodnotu:

```
> x[0]:=(a[0]+b[0])/2;
```

$x_0 := \frac{1}{2}$

```
> pol_x[0]:=subs(x=x[0],pol);
```

$pol\_x_0 := \frac{15}{4}$

Rozhodneme, která část intervalu splňuje předpoklady:

```
> evalb(pol_a[0]*pol_x[0]<0);
  evalb(pol_x[0]*pol_b[0]<0);
```

true

false

Dle předchozího zjištění, jsem rozhodla že nový interval je $\langle a_0; x_0 \rangle$. Přeznačíme ho na $\langle a_1; b_1 \rangle$:

```
> a[1]:=a[0];
  b[1]:=x[0];
```

$a_1 := 0$

$b_1 := \frac{1}{2}$

Pokud je úloha formulována stylem: hledání končí, když je kořen nalezen, nebo jestliže je odhad chyby nebo velikost intervalu menší než zvolené číslo. Z tohoto důvodu se nám vyplácí v každém kroku určovat velikost vzniklého intervalu.

```
> velikost[1]:=abs(b[1]-a[1]);
```

$velikost_1 := \frac{1}{2}$

Dostali jsme interval $\left\langle 0, \frac{1}{2} \right\rangle$ o velikosti $\frac{1}{2}$, ve kterém leží právě jeden kořen. Nyní vypočítáme x_1 a zjistíme jeho funkční hodnotu:

```
> x[1]:=(a[1]+b[1])/2;
```

$x_1 := \frac{1}{4}$

```
> pol_x[1]:=subs(x=x[1],pol);
```

$pol\_x_1 := 0$

Jelikož funkční hodnota bodu x_1 je nulová, dostali jsme kořen.

3.4 Procedury

Výpočty v této metodě nejsou moc náročné. Stačí pouze sečíst dvě čísla a vydělit dvěma. Může se ale stát, že budeme dosazovat do mnohem složitějšího výrazu (např. Newtonova metoda – viz. následující kapitola) a tento postup opakovat třeba padesátkrát. Zabralo by to spoustu času a námahy. Z tohoto důvodu zde uvádím několik postupů, které celý výpočet urychlí.

První pomůcka

Zadáme polynom pomocí tzv. funkčních operátorů.

```
> pol:=x->4*x^3+11*x^2+5*x-2;
```

$$pol := x \to 4x^3 + 11x^2 + 5x - 2$$

Nyní se `pol` bude chovat jako funkce. To znamená, že můžeme do ní dosazovat libovolné hodnoty. Nemusíme tedy používat metody `subs`. Například:

```
> pol(1);
```

18

Druhá pomůcka

Využití procedur.

Jsou to metody (lépe funkce), které si sami napíšeme. Určíme si v nich, co budou dělat a jaké budou mít návratové hodnoty (myšleno, jakou hodnotu, popř. hodnoty budou vracet).

Např. sestrojíme proceduru pro výpočet funkční hodnoty. Za prvé si musíme uvědomit, jaké vstupní parametry budeme používat. Budou následující:

1. parametr – funkce
2. parametr – bod, ve kterém chceme zjistit funkční hodnotu

Za druhé musíme uvést definici procedury:

```
> funkcni_hodnota := proc(f,bod)
  local r;
  r:=subs(x=bod,f);
  return r;
> end proc:
```

Podrobný popis (vysvětlení jednotlivých řádků kódu):

funkcni_hodnota := proc(f,bod)

název_procedury := proc(vstupní parametry)

local r;

seznam použitých pomocných proměnných, které budeme využívat v proceduře následuje blok operací, které má procedura provést

return r;

návratová hodnota – co bude procedura vracet

end proc:

ukončení procedury (Pro výpis použijeme středník. Pokud výpis z nějakého důvodu nechceme, použijeme dvojtečku.)

Ukázka volání procedury:

```
> funkcni_hodnota(4*x^3+11*x^2+5*x-2,1);
                              18
```

Stejným způsobem lze řešit rekurentní vzorec, rovnice tečny, derivace, … .

4 Metoda tečen (Newtonova metoda)

4.1 Teorie

Jestliže v rovnici $f(x)=0$ je f funkcí, která má na intervalu $\langle a,b\rangle$ spojitou druhou derivaci, $f(a)f(b)<0$, derivace f' a f'' jsou na intervalu $\langle a,b\rangle$ různé od nuly a $x_0 \in \langle a,b\rangle$ je takový bod, že $f(x_0)f''(x_0)>0$, pak první aproximací kořene rovnice $f(x)=0$ je bod $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$. Po výpočtu aproximace x_2 kořene lze metody tečen znovu použít pro nalezení lepší aproximace kořene. Přitom při každém dalším kroku se počet správných číslic aproximace kořene prakticky zdvojnásobí.

(Bartsch [2], s. 223 – 224, v citaci bylo pozměněno číslování a to proto, že nebudeme začínat jedničkou, ale nulou)

Poznámka. Abychom řešili rovnici $f(x)=0$, kde $f(x)$ je mnohočlen nebo obecněji funkce, která má derivaci v každém bodě nějakého intervalu (nebo komplexní oblasti), v němž hledáme kořeny, utvoříme při daném číslu x_0 posloupnost čísel $x_1, x_2, \ldots$ podle rekurentního vzorce

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Je-li číslo x_0 dosti blízké některému kořeni rovnice $f(x)=0$, konvergují čísla x_k k tomuto kořeni. Pro jednoduchý kořen je tato konvergence velmi rychlá (s každým krokem se počet správných číslic prakticky zdvojnásobí). (Rektorys [10], s. 612)

Poznámka. Odvození rekurentního vzorce.

Máme bod (bod dotyku tečny)

$$[x_0; f(x_0)]$$

a rovnici přímky (tečny)

$$y = kx + q\,, \tag{0}$$

kde *k* je její směrnice (derivace funkce v bodě dotyku, tzn. $k = f'(x_0)$).

Jelikož bod náleží tečně, můžeme ho společně s $k = f'(x_0)$ dosadit do (0), dostaneme

$$f(x_0) = f'(x_0)x_0 + q\,. \tag{1}$$

Z (1) vyjádříme *q*

$$q = f(x_0) - f'(x_0)x_0\,. \tag{2}$$

Směrnici $k = f'(x_0)$ a (2) dosadíme do (0)

$$y = f'(x_0)x + f(x_0) - f'(x_0)x_0$$

$$y = f'(x_0)(x - x_0) + f(x_0)\,. \tag{3}$$

Dostali jsme rovnici tečny. Nyní potřebujeme získat její průsečík s osou *x*. Jeho souřadnice si označíme jako $[x_1;0]$ a dosadíme do (3). Výsledkem je

$$0 = f'(x_0)(x_1 - x_0) + f(x_0)\,.$$

Pomocí úprav vyjádříme x_1

$$f'(x_0)x_1 = f'(x_0)x_0 - f(x_0)\,.$$

Dle předpokladů, je první derivace různá od nuly na intervalu. Proto

$$x_1 = \frac{f'(x_0)x_0 - f(x_0)}{f'(x_0)}$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}\,.$$

Pokud bychom postup opakovali pro $[x_1; f(x_1)]$, druhou aproximací kořene by bylo

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}\,.$$

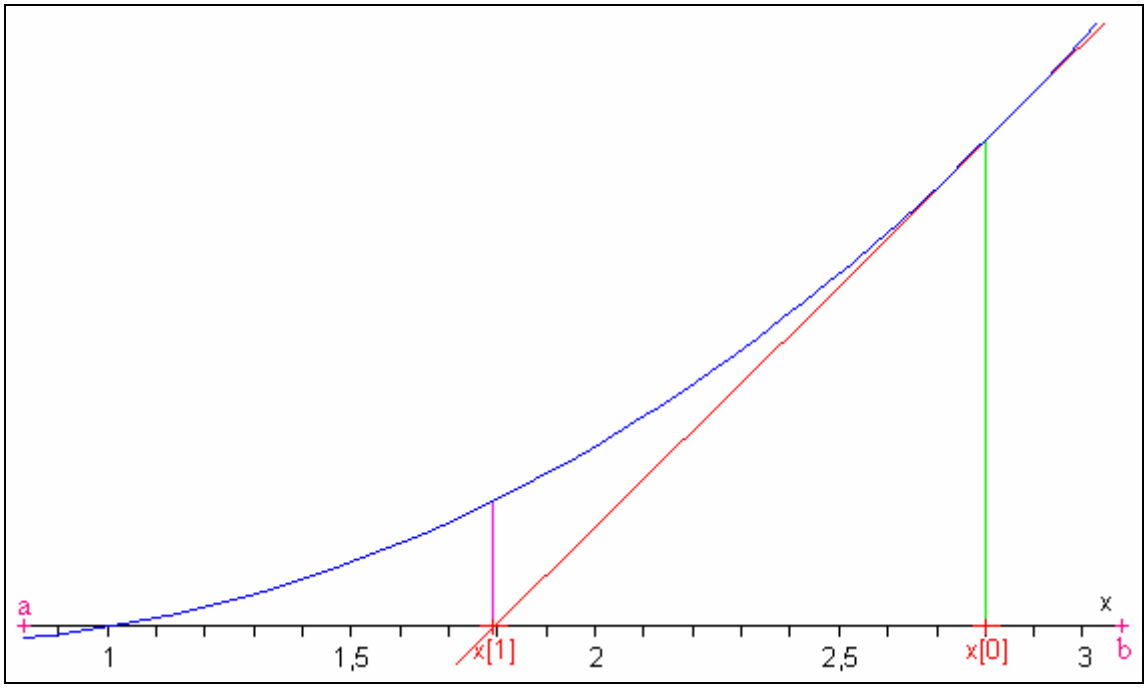
Je vidět, že k-tá aproximace bude vyjádřena

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,.$$

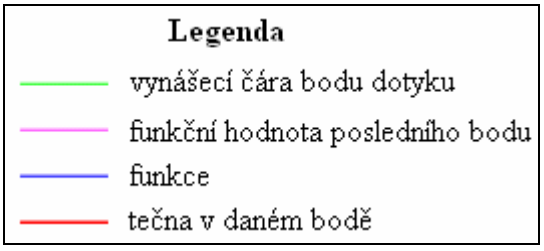
Geometrický význam metody tečen

Vedeme-li v bodě $[x_k, f(x_k)]$ tečnu ke křivce $y = f(x)$, je x_{k+1} první souřadnice průsečíku tečny s osou x. Jestliže v nějakém intervalu $\langle a,b \rangle$ mají derivace $f'(x)$ a $f''(x)$ stálé znaménko a platí-li $f(a)f(b) < 0$, pak Newtonova metoda s počátečním číslem $x_0 = a$, resp. $x_0 = b$ (podle toho, zda $f(a)$, resp. $f(b)$ má totéž znaménko jako $f''(x)$), konverguje k některému kořenu v $\langle a,b \rangle$. (Rektorys [10], s. 612)

Newtonova metoda krok za krokem



Obr. 4.1 Newtonova metoda – první tečna



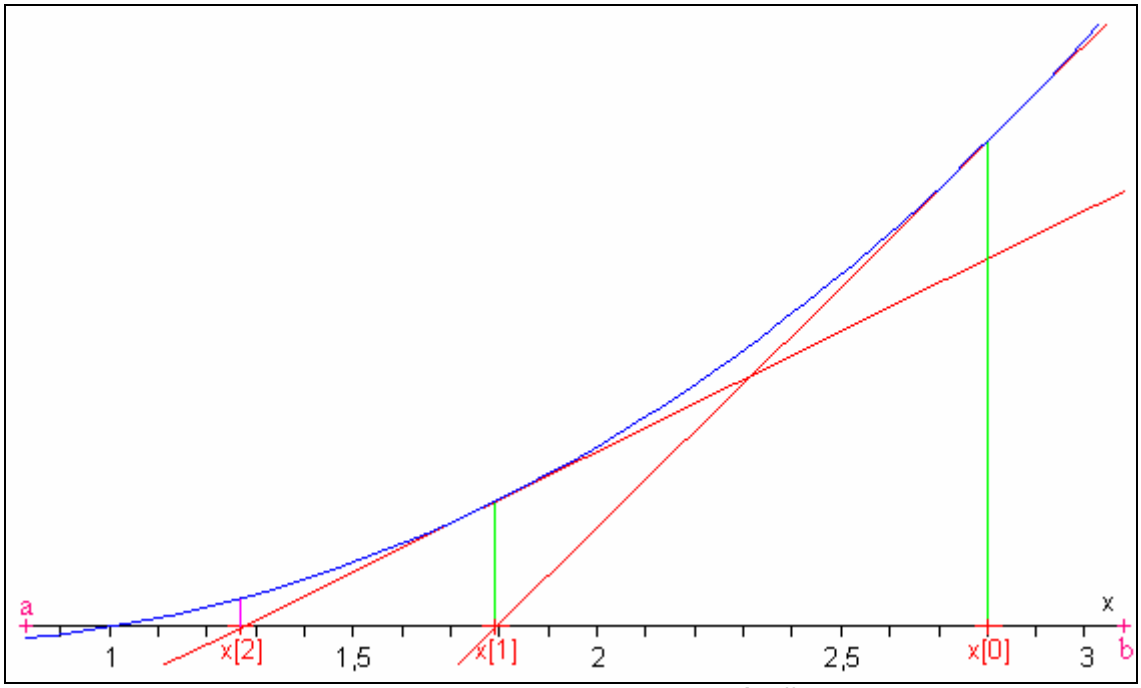
Obr. 4.2 Newtonova metoda – legenda

1. krok

V požadovaném bodě (na obr. 4.2 – pro nás *x[0]*, *x=2,8*, který patří do intervalu $\langle a;b \rangle$) vztyčíme kolmici a v průsečíku s funkcí vytvoříme tečnu k funkci. Najdeme bod, kde tečna protíná osu *x*.

2. krok

V nalezeném bodě opakujeme předchozí postup. Pokračujeme do té doby, dokud nedospějeme ke kořeni, anebo se k němu nepřiblížíme na stanovenou vzdálenost, či do požadované chyby.



Obr. 4.3 Newtonova metoda – druhá tečna

Odhad chyby I.

Označíme-li m nejmenší hodnotu funkce $|f'(x)|$ v intervalu $\langle a,b\rangle$, pak

$$|x_k - \alpha| \leq \frac{|f(x_k)|}{m}.$$

(přednášky optimalizačních metod doc. RNDr. Františka Mráze, CSc., odvození najdete v internetovém zdroji [6])

Poznámka. Tento odhad je univerzální. Je nezávislý na metodě, která byla použita.

Odhad chyby II.

Označíme-li m_1 nejmenší hodnotu $|f'(x)|$ a M_2 největší hodnotu $|f''(x)|$ v intervalu $\langle a,b\rangle$, potom pro odhad chyby platí:

$$|x_n - \alpha| \leq \frac{M_2}{2m_1}(x_n - x_{n-1})^2.$$

(přednášky optimalizačních metod doc. RNDr. Františka Mráze, CSc., odvození najdete v internetovém zdroji [6])

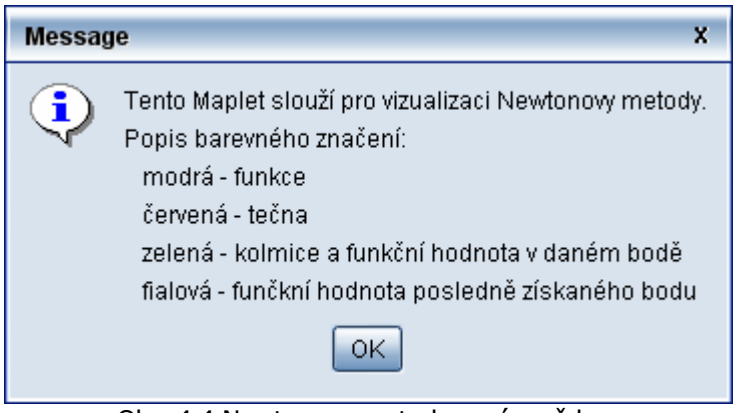
4.2 Vizualizace metody

Pro dobrou představu, jak funguje princip Newtonovy metody (geometrický význam), jsem vytvořila Maplet „Newtonova metoda – krok za krokem[3]“. Pomocí tlačítek s označením *0 – 4* můžete vidět jednotlivé kroky metody (nultý, první, třetí a čtvrtý krok je zobrazen na obr. 4.5 – 4.8).

Aplikace se ovládá pomocí pěti základních tlačítek s čísly *0 – 4*. Každé z nich reprezentuje jeden krok metody. Nultý krok zobrazuje počáteční bod. Dále jsou zde tři tlačítka:

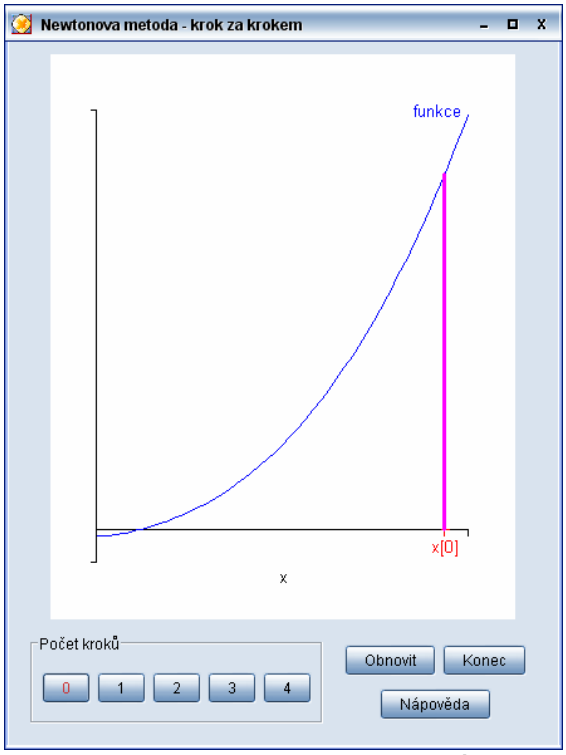
- *Obnovit* – návrat do nultého kroku
- *Konec* – ukončí Maplet
- *Nápověda* – viz obr. 4.4

Význam barevného značení: fialová úsečka symbolizuje funkční hodnotu posledně získaného bodu, v případě nultého kroku se jedná o funkční hodnotu počátečního bodu. Je nádherně vidět, jak se každým krokem blížíme ke kořeni (fialová úsečka se zmenšuje a na posledním obrázku (obr. 4.8) není téměř vidět). Zelené úsečky zastupují kolmice. Červené tečny nejsou značeny z důvodu možné nepřehlednosti.

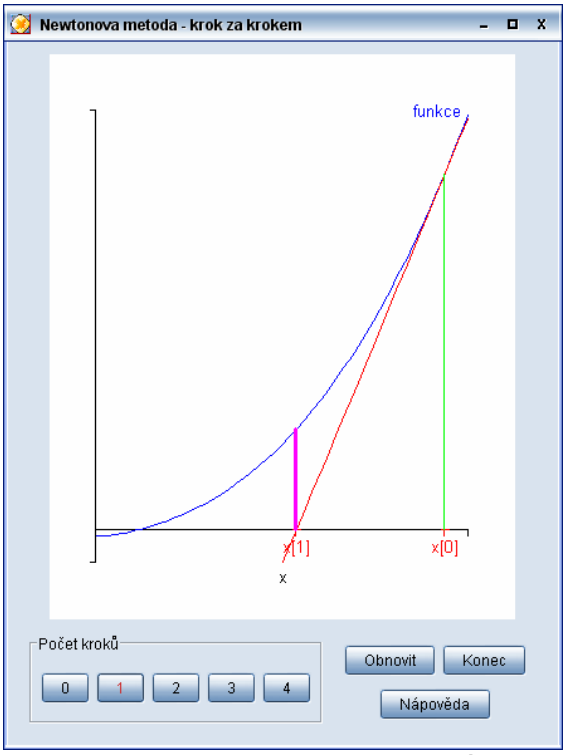


Obr. 4.4 Newtonova metoda – nápověda

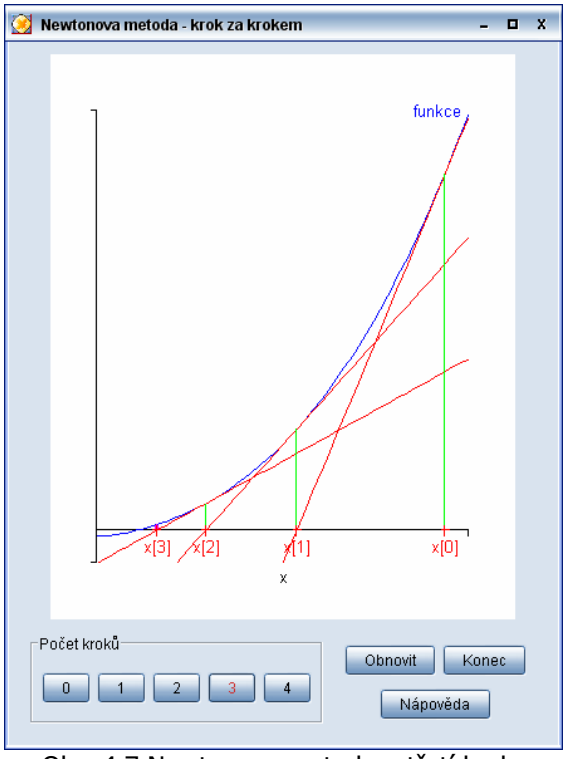
[3] název souboru „4.2 Vizualizace\Newtonova metoda – krok za krokem.maplet“
zdrojový kód je obsažen v: „4.2 Vizualizace\Newtonova metoda – krok za krokem.mws“



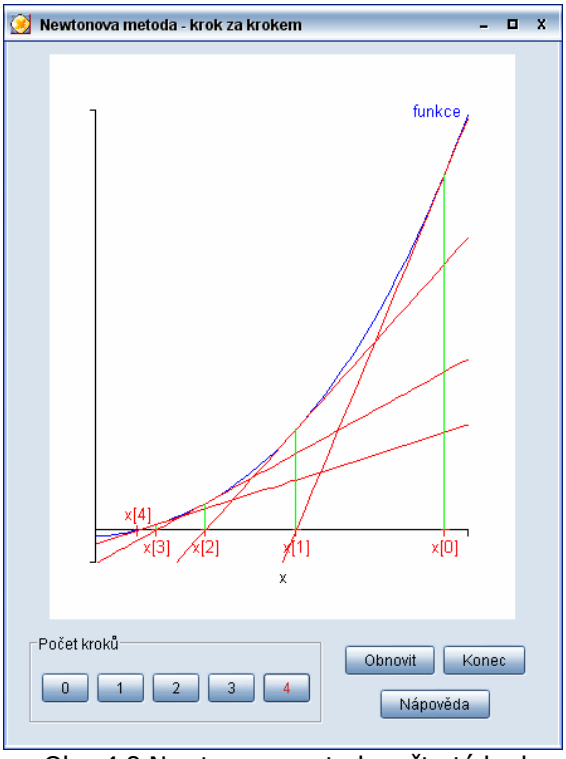
Obr. 4.5 Newtonova metoda – nultý krok



Obr. 4.6 Newtonova metoda – první krok



Obr. 4.7 Newtonova metoda – třetí krok



Obr. 4.8 Newtonova metoda – čtvrtý krok

4.3 Příklad

$$P(x) = 4x^3 + 11x^2 + 5x - 2$$

Dle kapitoly 2 budeme další výpočty stavět na intervalu $\langle 0;1 \rangle$, ve kterém leží právě jeden reálný kladný kořen. Za počáteční bod zvolím $x_0 = 1$ z intervalu $\langle 0;1 \rangle$ ($a = 0$, $b = 1$).

4.3.1 Ověření splnění předpokladů metody

Před použitím metody musíme ověřit předpoklady. Označení:

```
> pol:=4*x^3+11*x^2+5*x-2;
  x[0]:=1;
        pol := 4 x^3 + 11 x^2 + 5 x - 2
        x_0 := 1
> a:=0;
  b:=1;
        a := 0
        b := 1
```

1) $f''(x)$ je spojitá na intervalu

```
> d2_f:=diff(pol,x$2);
        d2_f := 24 x + 22
```

$f''(x)$ je spojitá (jedná se o přímku – polynom, viz kapitola 1.3 poznámka 4.).

2) $f(a)f(b) < 0$

```
> f_a:=subs(x=a,pol);
        f_a := -2
> f_b:=subs(x=b,pol);
        f_b := 18
> evalb(f_a*f_b<0);
        true
```

3) $f(x_0)f''(x_0) > 0$

```
> f[x0]:=subs(x=x[0],pol);
        f_x0 := 18
> d2_f_x0:=subs(x=x[0],d2_f);
        d2_f_x0 := 46
```

```
> evalb(f[x0]*d2_f_x0>0);
        true
```

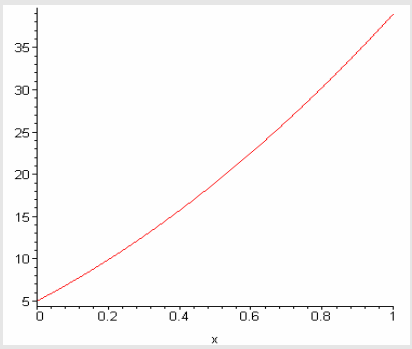
4) $x_0 = 1$ a platí, že $x_0 \in \langle 0;1 \rangle$

```
> is(x[0],RealRange(a,b));
        true
```

5) $f'(x) \neq 0$, $f''(x) \neq 0$

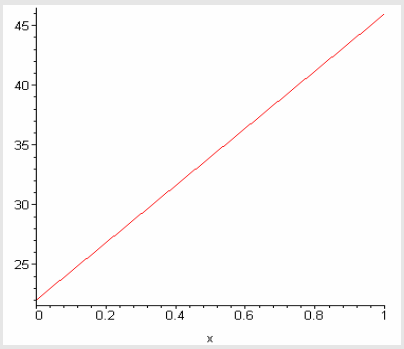
$f'(x) \neq 0$ neboli nemění znaménko na intervalu

```
> d1_f:=diff(pol,x);
        d1_f := 12 x^2 + 22 x + 5
> evalf(solve(d1_f,x));
        -0.2658125271, -1.567520806
> plot(d1_f,x=a..b);
```



$f''(x) \neq 0$ neboli nemění znaménko na intervalu

```
> d2_f:=diff(pol,x$2);
        d2_f := 24 x + 22
> evalf(solve(d2_f,x));
        -0.9166666667
> plot(d2_f,x=a..b);
```



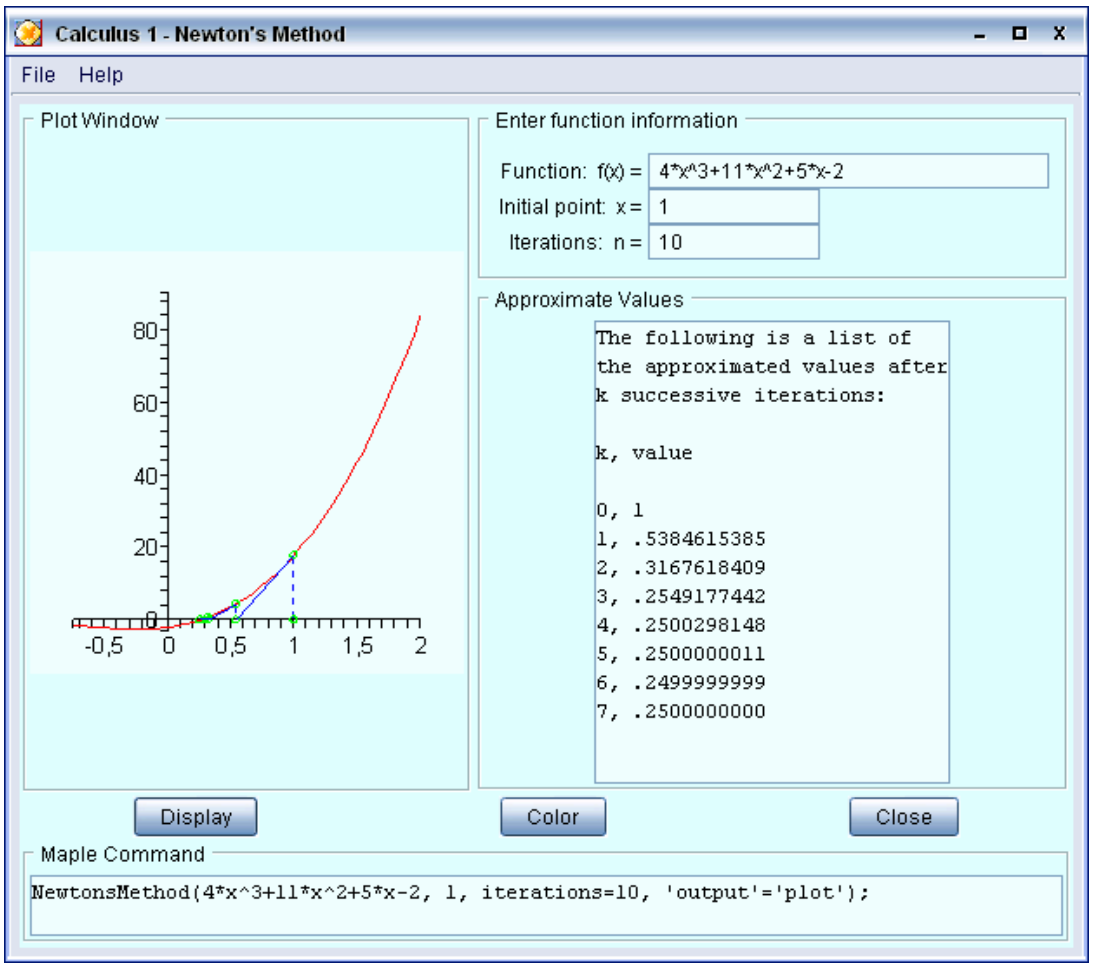
Závěr. Všechny předpoklady jsou splněny, metoda může být použita.

4.3.2 Grafické řešení

Matematický problém můžeme řešit pomocí grafického aparátu. Maple 9.5 nám to umožňuje více způsoby (postupy), nebo lze využít aplikací či funkcí, které jsem vytvořila. U každého jsou uvedeny výhody a nevýhody. Záleží pouze na uživateli (studentovi), který způsob si vybere, jaký postup mu bude bližší, nebo zda chce znát jednotlivé kroky, či pouze výsledek.

První způsob: Využití balíčku `Student` a metody `NewtonsMethodTutor`, která spouští tutoriál.

```
> with(Student[Calculus1]):
  NewtonsMethodTutor();
```



Obr. 4.9 Newton's Method

Popis jednotlivých částí (obr. 4.9):

- *Plot Window* – zobrazení průběhu Newtonovy metody
- *Enter function information* – zadání vstupních parametrů
  - *f(x)* – zadaná funkce
  - *x* – počáteční bod
  - *n* – počet kroků, které chceme znázornit
- *Approximate Values* – hodnoty postupné aproximace kořene ve tvaru: krok, hodnota
- *Display* – tlačítko pro vykreslení (po zadání všech hodnot)
- *Color* – barevné nastavení (funkce, tečen, bodů)
- *Close* – zavře tutoriál
- *Maple Command* – výpis příkazu, který lze použít bez použití Mapletu

Výhody:

- Kromě grafického znázornění se nám dostává i výpisu jednotlivých aproximací kořene.

Nevýhody:

- Pouze v anglickém jazyce.
- Nepřehlednost obrázku – někdy se může stát, že je obrázek zmatečný a to z několika důvodů. Například blízkost počátečního bodu u kořene, strmost monotonie, či jiných důvodů. To by nevadilo, kdyby bylo možné posunout hranice zobrazované části grafu (udělat výřez dle potřeby). To nám bohužel není umožněno.

Druhý způsob: Využití studentského balíčku a metody `NewtonsMethod`.

Popis jednotlivých parametrů `NewtonsMethod`

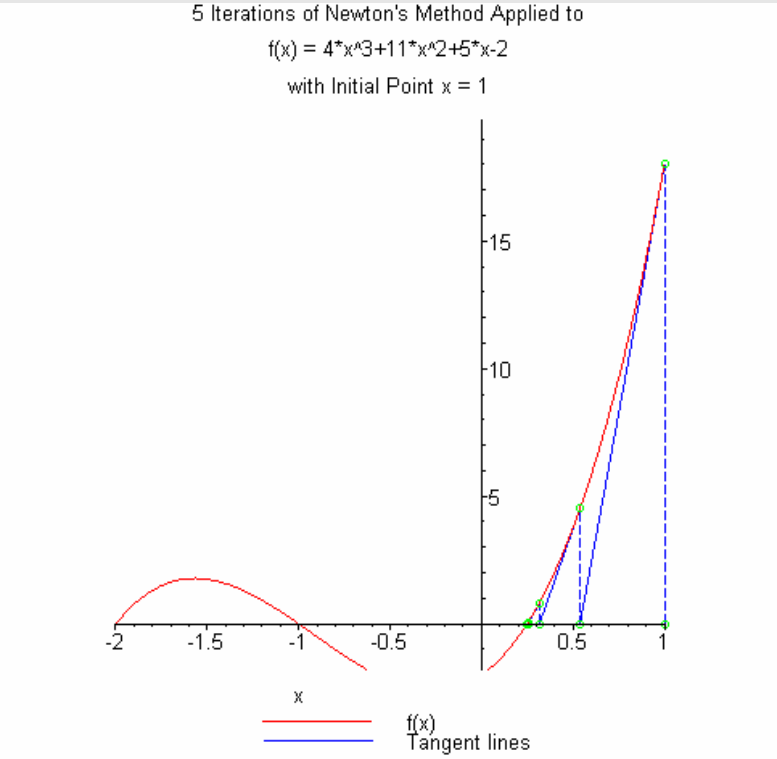
1. parametr – funkce
2. parametr – počáteční bod
3. parametr – *view = [xmin .. xmax, ymin .. ymax]* – rozsah znázornění
4. parametr – na výstup požadujeme graf

```
> with(Student[Calculus1]):
> pol:=4*x^3+11*x^2+5*x-2;x_0:=1;
```

$$pol := 4x^3 + 11x^2 + 5x - 2$$

$$x\_0 := 1$$

```
> NewtonsMethod(pol, x = x_0, view = [-2..1, -2..18], output = plot);
```



Výhody:

- Možnost znázornění jen určité části.

Třetí způsob: Postupné vykreslování tečen za pomoci studentského balíčku a metody `Tangent`. Tento způsob není pouze grafický, protože musíme provádět výpočty nových bodů.

Popis jednotlivých parametrů `Tangent`

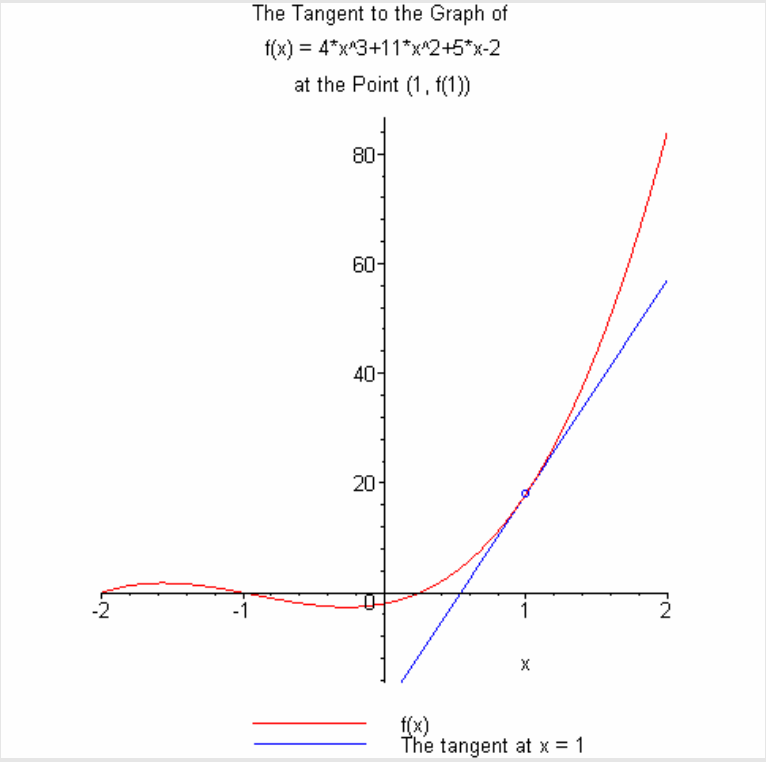
1. parametr – funkce
2. parametr – bod, ve kterém potřebujeme vykreslit tečnu
3. parametr – *view = [xmin .. xmax, ymin .. ymax]* – rozsah znázornění
4. parametr – na výstup požadujeme graf

Pokud použijeme jen první dva parametry, Maple 9.5 nám vypíše rovnici tečny v daném bodě (označila jsem si ji jako `tecna`).

```
> with(Student[Calculus1]):
tecna:=Tangent(pol, x = 1);
```

$$tecna := 39x - 21$$

```
> Tangent(pol, 1, view = [-2..2, DEFAULT], output = plot);
```



Nyní stačí zjistit průsečík tečny s osou *x* a metodu použít opakovaně.

```
> fsolve(tecna=0,x);
```

0.5384615385

Nevýhody:

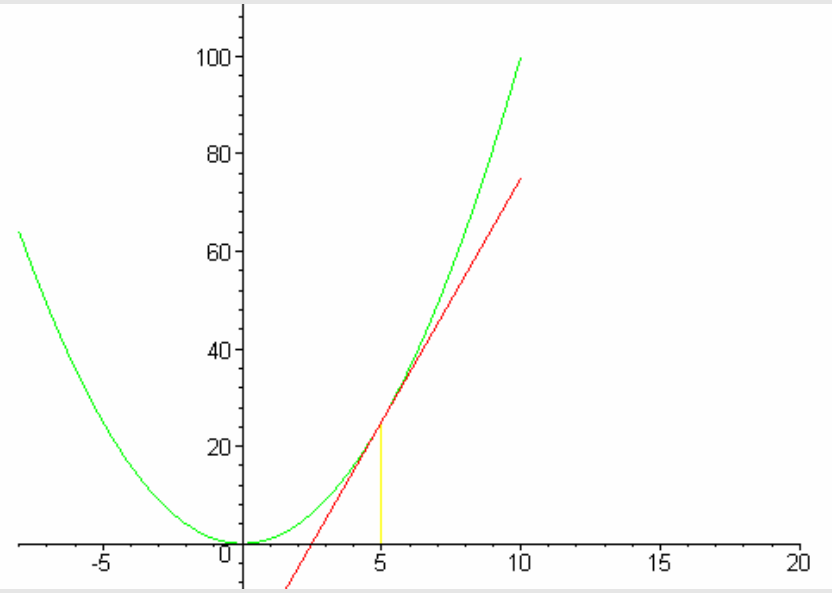
- Zdlouhavý způsob řešení a nutnost nalézt pro každý krok nový bod.

Výhody:

- Změna rozsahu zobrazení.

Poznámka. Ve studentském balíčku je uvedena metoda `showtangent`, která by měla vykreslit funkci a k ní tečnu v daném bodě. Problémem je, že graf je zobrazován pouze v rozmezí $x \in \langle -10;10 \rangle$ a pak se „čáry" prostě ztratí. Viz ukázka na funkci x^2.

```
> with(student):
showtangent(x^2,x=5,view=[-8..20,-10..150]);
```



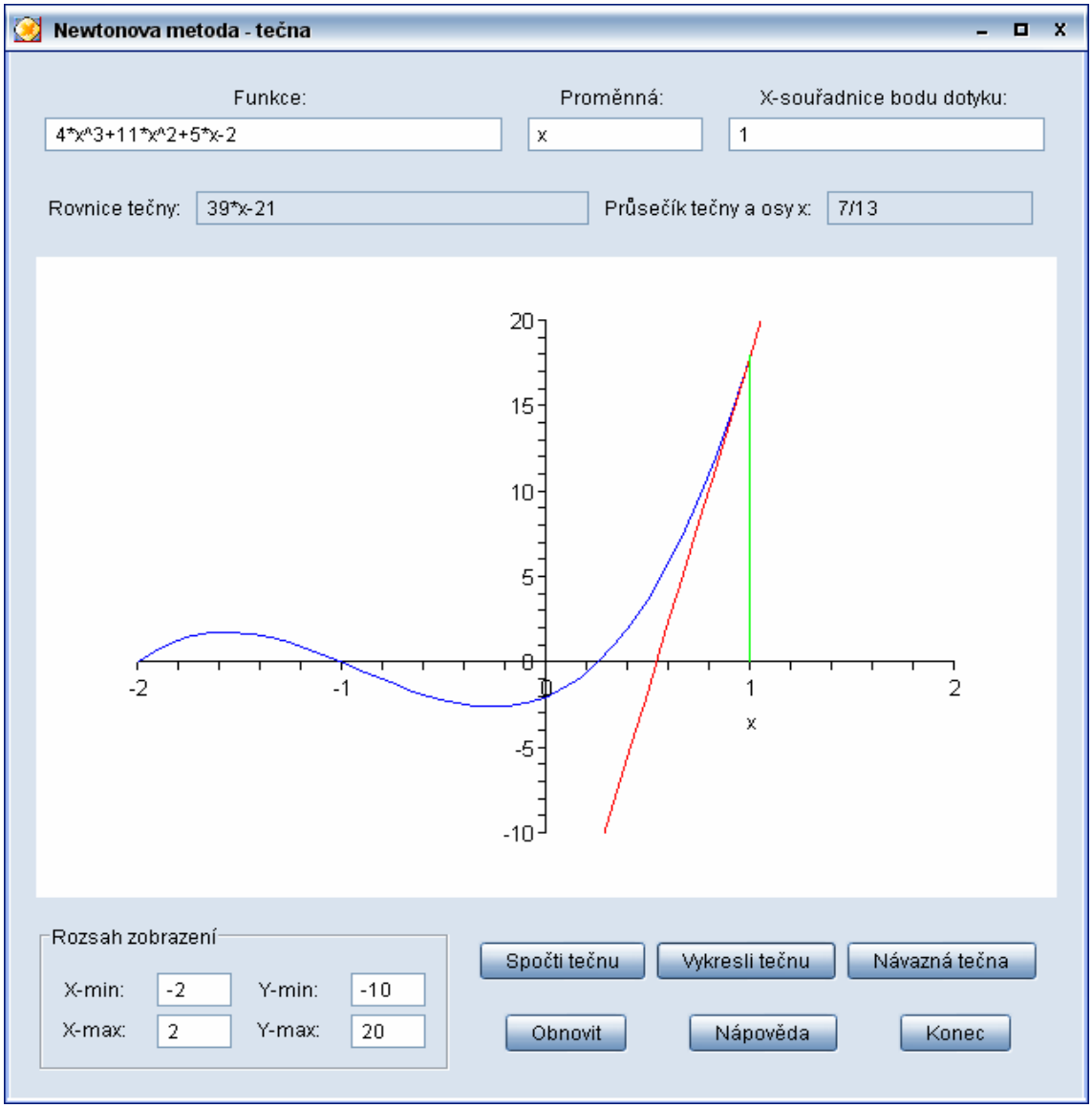
Čtvrtý způsob: Postupné vykreslování tečen za pomoci Mapletu „Newtonova metoda – tečna[4]", který jsem pro tento postup vytvořila. Provádění tohoto Mapletu je dvoukrokové. V prvním kroku je vypočítána a zobrazena rovnice tečny a průsečíku tečny s osou x (tlačítkem *Spočti tečnu*). Ve druhém je vykreslen graf funkce, tečny a vynášecí čáry bodu dotyku (tlačítkem *Vykresli*).

Popis jednotlivých částí (obr. 4.10):

- *Funkce*
- *Proměnná* – písmeno z abecedy, které je pro nás proměnnou (někdy je nutné použít jiné než x)

[4] název souboru „4.3 Priklad\Newtonova metoda - tecna.maplet"
zdrojový kód je obsažen v: „4.3 Priklad\Newtonova metoda - tecna.mws"

- *X-souřadnice bodu dotyku* – počáteční bod
- *Rovnice tečny* – po stisknutí tlačítka *Spočti tečnu* se zde objeví rovnice tečny
- *Průsečík tečny a osy x* – po stisknutí tlačítka *Spočti tečnu* se zde objeví souřadnice průsečíku tečny a osy *x*
- *X-min, X-max, Y-min, Y-max* – rozsah zobrazení (po každé změně je nezbytné graf překreslit pomocí tlačítka *Vykresli*)
- *Spočti tečnu* – vypočítá a zobrazí rovnici tečny na základě vstupních hodnot
- *Vykresli tečnu* – po vypočítání tečny zobrazí graf funkce a tečny
- *Návazná tečna* – dosadí získaný průsečík za nový bod dotyku
- Tlačítka *Obnovit*, *Nápověda* a *Konec* nechávám bez komentáře



Obr. 4.10 Tečna – ukázkový příklad použití Mapletu

Na základě vypsané hodnoty průsečíku tečny a osy *x* lze zobrazit další tečnu. Stačí změnit souřadnice bodu dotyku na získanou hodnotu a následného stisku tlačítek *Spočti tečnu* a *Vykresli* v tomto pořadí. Stejného výsledku dosáhneme použitím tlačítka *Návazná tečna*, která nahrazuje ruční přepisování hodnoty. Opět je nezbytné použít *Spočti tečnu* a *Vykresli*.

Aplikace zamezuje stisknutí tlačítek ve špatném pořadí. Graf není zobrazen do té doby, dokud není vypsaná rovnice tečny.

Výhody:

- Velmi vysoká nastavitelnost (rozsah zobrazení, volba proměnné).

Nevýhody:

- Zdlouhavost.
- Pouze pro polynomiální funkce.

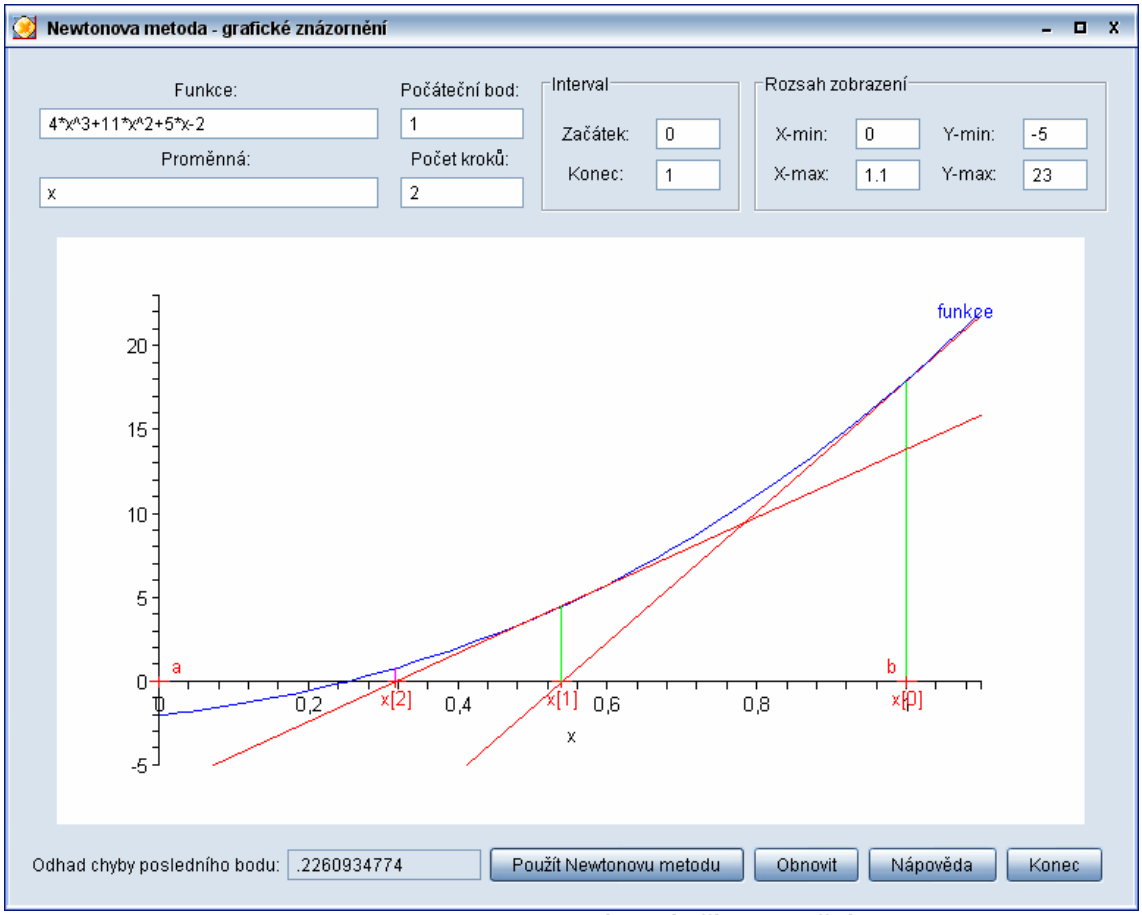
Pátý způsob: Využití mého Mapletu „Newtonova metoda – grafické znázornění[5]“.

Popis jednotlivých částí (obr. 4.11)

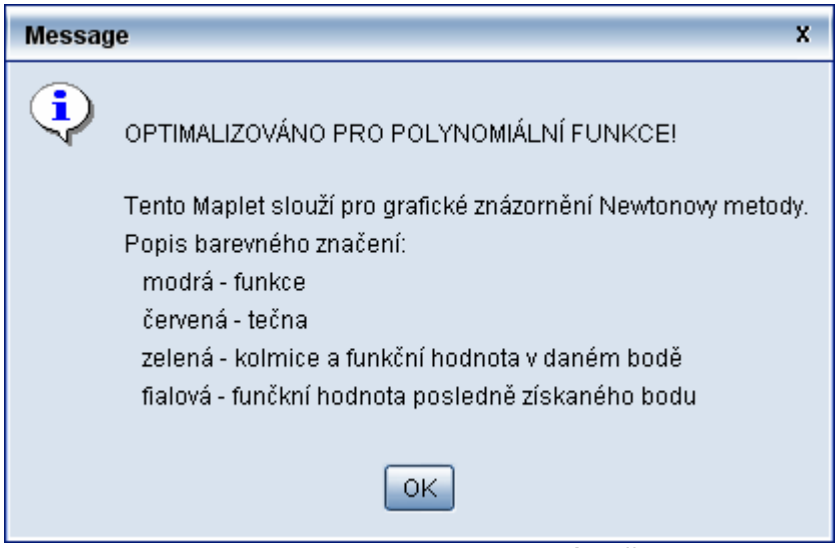
- *Funkce, Proměnná, X-min, X-max, Y-min, Y-max* – význam je stejný jako u předchozího postupu
- *Počáteční bod* – bod, ve kterém bude vytvořena první tečna
- *Počet kroků* – jedná se o celé nezáporné číslo, které udává kolik kroků Newtonovy metody chceme udělat
- *Interval* (dle předpokladů musí být počáteční bod z tohoto intervalu)
- *Použít Newtonovu metodu* – slouží pro vykreslení grafu, nutné ho použít po každé změně, kterou uděláme
- *Konec*, *Obnovit* – bez komentáře
- *Nápověda* – spustí okno s krátkým popisem Mapletu (obr. 4.12)
- *Odhad chyby posledního bodu* – zobrazuje odhad maximální chyby (jedná se o odhad II.) posledně získaného bodu a kořene. V případě nalezení kořene, je

[5] název souboru „4.3 Priklad\Newtonova metoda - graficke.maplet“
zdrojový kód je obsažen v: „4.3 Priklad\Newtonova metoda - graficke.mws“

chyba zobrazována. Je tak činěno pouze pro zajímavost, abychom viděli, kdy program akceptuje bod za kořen.

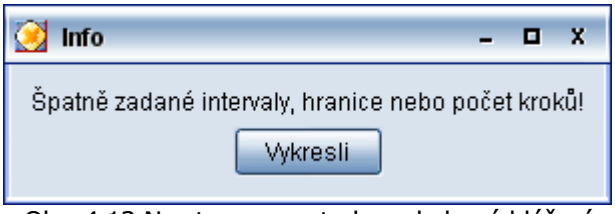


Obr. 4.11 Newtonova metoda – ukázkový příklad použití Mapletu

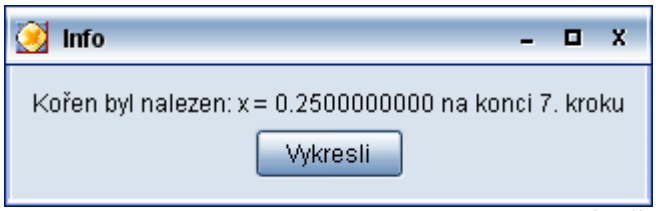


Obr. 4.12 Newtonova metoda – nápověda

Oproti tutoriálu jsem Maplet naprogramovala tak, aby nás informoval o tom, když zadáme neplatné vstupní hodnoty (tzn. zadáme bod, který nesplňuje předpoklady nebo neplatný interval, např. *X-min* ≥ *X-max* (obr. 4.13, po stisknutí tlačítka *Vykresli* se zobrazí prázdný graf) či počet kroků není celé nezáporné číslo, apod.). Maplet je optimalizován pro polynomiální funkce, v ostatních případech může dojít k jeho špatnému fungování. Dále nás informuje, zda se po několika krocích dostaneme ke kořeni (viz obr. 4.14). Samozřejmě myšleno do zadaného počtu kroků.



Obr. 4.13 Newtonova metoda – chybové hlášení



Obr. 4.14 Newtonova metoda – informace o nalezení kořene

Výhody:

- Velmi vysoká nastavitelnost (rozsah zobrazení, volba proměnné).
- Zobrazení odhadu chyby.

Nevýhody:

- Řešení je optimalizováno na polynomiální funkce. V ostatních případech není zaručena stoprocentní funkčnost, jak matematická, tak i programová.
- Chybí popis tečen. Je to z důvodu možné nepřehlednosti grafu.
- Když je odhad chyby menší než jedna, Maplet nezobrazuje nulu před desetinnou čárkou.
- V blízkosti kořene dochází k překrývání značení bodů. Zvlášť při zvoleném velkém počtu kroků.

4.3.3 Numerické řešení (přímé)

Maple 9.5 nám nabízí prostředek na přímé řešení pomocí Newtonovy metody a to `NewtonsMethod(funkce, iterations = pocet, prom = bod)`. Kde `pocet` udává kolik kroků Newtonovy metody bude použito a `bod` je počáteční bod.

```
> with(Student[Calculus1]):
> pol:=4*x^3+11*x^2+5*x-2;
> x_0:=1;
```

$pol := 4x^3 + 11x^2 + 5x - 2$

$x\_0 := 1$

```
> NewtonsMethod(pol, iterations=10, x = x_0);
```

0.2500000000

Pokud chceme znát hodnoty, které předcházejí výsledku musíme přidat další `output = sequence`.

```
> NewtonsMethod(pol, x = x_0, iterations=10, output = sequence);
```

1, 0.5384615385 , 0.3167618409 , 0.2549177442 , 0.2500298148 , 0.2500000011 , 0.2499999999 , 0.2500000000

Dostaneme sekvenci čísel, která jsou od sebe oddělená čárkou. První je počáteční bod a za ním následuje posloupnost aproximací kořene $x_1, x_2, \ldots$.

4.3.4 Numerické řešení (postupné)

Předpoklady nebudu ověřovat, byly ověřeny na začátku kapitoly.

První přístup: Budeme postupně dosazovat do rekurentního vzorce: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.

```
> pol:=4*x^3+11*x^2+5*x-2; x_0:=1;
```

$pol := 4x^3 + 11x^2 + 5x - 2$

$x\_0 := 1$

Nejprve vypočítáme $f(x_0)$ a pak $f'(x_0)$:

```
> f_x_0:=subs(x=x_0,pol);
```

$f\_x\_0 := 18$

```
> df_x_0:=subs(x=x_0,diff(pol,x));
```

$df\_x\_0 := 39$

Nyní stačí dosadit do rekurentního vzorce a vypočtenou hodnotu vyčíslit:

```
> x_1:=x_0-(f_x_0)/(df_x_0);
```

$$x\_1 := \frac{7}{13}$$

```
> x_1h:=evalf(x_1);
```

$$x\_1h := 0.5384615385$$

Výpočet hodnoty x_2 je obdobný:

```
> f_x_1:=subs(x=x_1,pol);
```

$$f\_x\_1 := \frac{9900}{2197}$$

```
> df_x_1:=subs(x=x_1,diff(pol,x));
```

$$df\_x\_1 := \frac{3435}{169}$$

```
> x_2:=x_1-(f_x_1)/(df_x_1);
```

$$x\_2 := \frac{943}{2977}$$

```
> x_2h:=evalf(x_2);
```

$$x\_2h := 0.3167618408$$

Přeskočíme pár kroků a dostáváme x_5:

```
> x_5:=x_4-(f_x_4)/(df_x_4);
```

x_5 := 505319204068440699338619020741892234883536033280166893581220897634998456801186553
25026639/
202127680733042275386401156726274886923984636193376878433660089835103004759511808617236145

```
> x_5h:=evalf(x_5);
```

$$x\_5h := 0.2500000011$$

Jak sami vidíte, zlomek x_5, je velmi složitý. Z tohoto důvodu je vhodné použití právě jednoho z matematických programů, neboť zde dochází k minimálnímu zaokrouhlování (minimální ztráta na přesnosti).

Postup opakujeme do té doby, dokud nejsme dostatečně blízko u kořene (odhad chyby). V našem případě stačí dojít k x_7:

```
> x_7h:=evalf(x_7);
```

$$x\_7h := 0.2500000000$$

Nyní ukážeme následující desetinná místa. Je vidět, že kořen je brán s přesností na 35 desetinných míst.

```
> x_7h1:=evalf(x_7,40);
```

$$x\_7h1 := 0.2500000000000000000000000000000028852$$

Pro srovnání zde uvádím sekvenci výsledků, které jsme touto cestou dostali:

```
> vysledek:=[x_0,x_1h,x_2h,x_3h,x_4h,x_5h,x_6h,x_7h];
      vysledek := [1, 0.5384615385, 0.3167618408, 0.2549177441, 0.2500298148, 0.2500000011,
      0.25000000000000000152, 0.2500000000]
```

a sekvenci výsledků, které byly vypočteny za pomoci `NewtonsMethod`.

```
> NewtonsMethod(pol, x = x_0, iterations=10, output = sequence);
      1,0.5384615385,0.3167618409,0.2549177442,0.2500298148, 0.2500000011, 0.2499999999,0.2500000000
```

Je vidět, že jsou téměř totožné na zobrazovaném počtu desetinných míst. Odlišnosti vidíme u třetího a čtvrtého čísla (poslední zobrazovaná číslice není stejná) a nejvíce rozdílností je u předposledního čísla. Ke zkreslení došlo zřejmě během počítání a to vinou zaokrouhlování.

Druhý přístup: Pokud si nepamatujeme rekurentní vzorec, tak nám stačí k výpočtu dalšího bodu znalost tečny. Tzn. $y = kx + q$, kde k je směrnice tečny (neboli derivace funkce v daném bodě). Upravíme na: $q = y - kx = f(x_0) - f'(x_0)x_0$. Po dosazení dostáváme: $y = kx + q = f'(x_0)x + f(x_0) - f'(x_0)x_0 = f'(x_0)(x - x_0) + f(x_0)$. Rovnice tečny je:

```
> rovnice_tecny_1:=df_x_0*(x-x_0)+f_x_0;
      rovnice_tecny_1 := 39 x - 21
```

Stejný výsledek dostaneme při použití metody `Tangent`.

```
> with(Student[Calculus1]):
  tecna:=Tangent(pol, x = x_0);
      tecna := 39 x - 21
```

Abychom dostali x_1, stačí spočítat průsečík tečny s osou x:

```
> x_1t:=solve(rovnice_tecny_1=0,x);
      x_1t := 7/13
> evalf(x_1t);
      0.5384615385
```

Odhad chyby I.

$|x_k - \alpha| \le \frac{|f(x_k)|}{m}$, m je nejmenší hodnota funkce $|f'(x)|$ v intervalu $\langle a,b\rangle$.

Nejmenší hodnotu získáme za použití nástroje `minimize`. Popis parametrů:

1. parametr – funkce
2. parametr – rozmezí intervalu, na kterém hledáme minimum

```
> m:=minimize(abs(diff(pol,x)),x=0..1);
      m := 5
```

Pokud nás zajímá přesná poloha minima, přidáme parametr `location`:

```
> minimize(abs(diff(pol,x)),x=0..1,location);
      5, {[{x = 0}, 5]}
```

Nyní vypočítáme odhad chyby, například pro x_5:

```
> f_x_5:=subs(x=x_5,pol):
> evalf(f_x_5);
      0.1244418568  10^-7
> odhad_chyby1:=evalf((abs(f_x_5))/m);
      odhad_chyby1  := 0.2488837137  10^-8
```

Závěr. $|x_5 - \alpha| < 0.2488837137 \cdot 10^{-8}$

Odhad chyby II.

$|x_n - \alpha| \le \frac{M_2}{2m_1}(x_n - x_{n-1})^2$, m_1 je nejmenší hodnota $|f'(x)|$ a M_2 největší hodnotu $|f''(x)|$ v intervalu $\langle a,b\rangle$.

m_1 bude shodná s m z minulého určení chyby. M_2 získáme za pomoci funkce `maximize` (parametry jsou totožné jako u `minimize`).

```
> m1:=minimize(abs(diff(pol,x)),x=0..1);
  M2:=maximize(abs(diff(diff(pol,x),x)),x=0..1);
      m1 := 5
      M2 := 46
```

Dopočítáme odhad chyby (stejně jako u předchozího odhadu pro x_5):

```
> odhad_chyby2:=evalf(M2/(2*m1)*(x_5-x_4)^2);
        odhad_chyby2    := 0.4088734206    10^-8
```

Závěr. $|x_5 - \alpha| < 0.4088734206 \cdot 10^{-8}$

Vidíme, že univerzální odhad chyby je přesnější. Ale ne vždy to platí. Zde uvádím rozdíl zjištěných odhadů:

```
> odhad_chyby2 - odhad_chyby1;
        0.1599897069    10^-8
```

Pro zajímavost nyní uvedu proceduru `vypocetChyby` na výpočet odhadu chyby II., kód vysvětlovat nebudu. Parametry jsou následující:

1. parametr – funkce

2., 3. parametr – poslední dva získané body

4., 5. parametr – vstupní interval

6. parametr – proměnná

```
> vypocetChyby := proc(f,bod1,bod2,a,b,prom)
> local m1,m2;
  m1:=minimize(abs(diff(f,prom)),prom=a..b);
  m2:=maximize(abs(diff(f,prom$2)),prom=a..b);
  if(m1=0) then
    return "nelze";
  else
    return evalf(m2/(2*m1)*(bod1-bod2)^2);
  end if;
> end proc:
```

Výsledek je následující

```
> vypocetChyby(pol,x_4,x_5,0,1,x);
        0.4088734206    10^-8
```

Pokud to porovnáme s předchozím získaným výsledkem (`odhad_chyby2`), zjistíme že jsou totožné. Neboli metoda nám vrací stejné odhady. Někdy se může stát, že při zaokrouhlování dojde k menším odlišnostem.

5 Metoda tětiv (regula falsi)

5.1 Teorie

V literatuře se sekáváme s dvěma různými interpretacemi metody. Uvedu obě. Ale pro pozdější výpočet budu využívat vždy té první, protože má silnější předpoklady.

5.1.1 Interpretace 1

(čerpáno z literatury Bartsch [2], str. 222)

Jestliže rovnice $f(x)=0$ má mezi hodnotami x_1 a x_2 právě jeden kořen (tj. jestliže platí $f(x_1)f(x_2)<0$ a funkce f je spojitá a ryze monotónní na intervalu $\langle x_1, x_2 \rangle$) a nemá v tomto intervalu inflexi, pak první aproximací kořene vzhledem k bodům x_1 a x_2 je

$$x_3 = x_1 - \frac{(x_2 - x_1)f(x_1)}{f(x_2) - f(x_1)}.$$

Další lepší aproximací kořene získáme obdobně v tom z intervalů $\langle x_1, x_3 \rangle$, $\langle x_3, x_2 \rangle$, kde platí uvedené předpoklady.

Poznámka. Odvození vzorce pro x_3.

Máme body

$$[x_1; f(x_1)],\ [x_2; f(x_2)]$$

a rovnici přímky procházející těmito body (tětiva)

$$y = kx + q. \qquad (0)$$

Jelikož oba dva body náleží přímce, dostáváme

$$f(x_1) = kx_1 + q\,, \tag{1}$$

$$f(x_2) = kx_2 + q\,. \tag{2}$$

Z (1) vyjádříme *q*

$$q = f(x_1) - kx_1 \tag{3}$$

a dosadíme do (2)

$$f(x_2) = kx_2 + f(x_1) - kx_1\,. \tag{4}$$

Z (4) vyjádříme *k*

$$f(x_2) - f(x_1) = kx_2 - kx_1$$

$$f(x_2) - f(x_1) = k(x_2 - x_1)\,.$$

Dle předpokladů této metody jsou x_2 a x_1 dva různé body (musí totiž mezi nimi ležet kořen), proto $x_2 - x_1 \neq 0$. Na základě toho dostáváme

$$k = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{5}$$

a (5) dosadíme zpět do (3)

$$q = f(x_1) - \frac{f(x_2) - f(x_1)}{x_2 - x_1} x_1\,. \tag{6}$$

Po dosazení (5), (6) do (0) dostáváme rovnici tětivy

$$y = \frac{f(x_2) - f(x_1)}{x_2 - x_1} x + f(x_1) - \frac{f(x_2) - f(x_1)}{x_2 - x_1} x_1\,. \tag{7}$$

Nyní potřebujeme získat průsečík tětivy (7) s osou *x*. Jeho souřadnice si označíme jako $[x_3;0]$ a dosadíme do (7). Výsledkem je

$$0 = \frac{f(x_2) - f(x_1)}{x_2 - x_1} x_3 + f(x_1) - \frac{f(x_2) - f(x_1)}{x_2 - x_1} x_1\,.$$

Pomocí úprav vyjádříme z poslední rovnice x_3

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} x_3 = \frac{f(x_2) - f(x_1)}{x_2 - x_1} x_1 - f(x_1)\,.$$

Dle předpokladů této metody jsou $f(x_2)$ a $f(x_1)$ dvě různé hodnoty (protože $f(x_1)f(x_2) < 0$), proto $f(x_2) - f(x_1) \neq 0$. Výsledkem tedy je

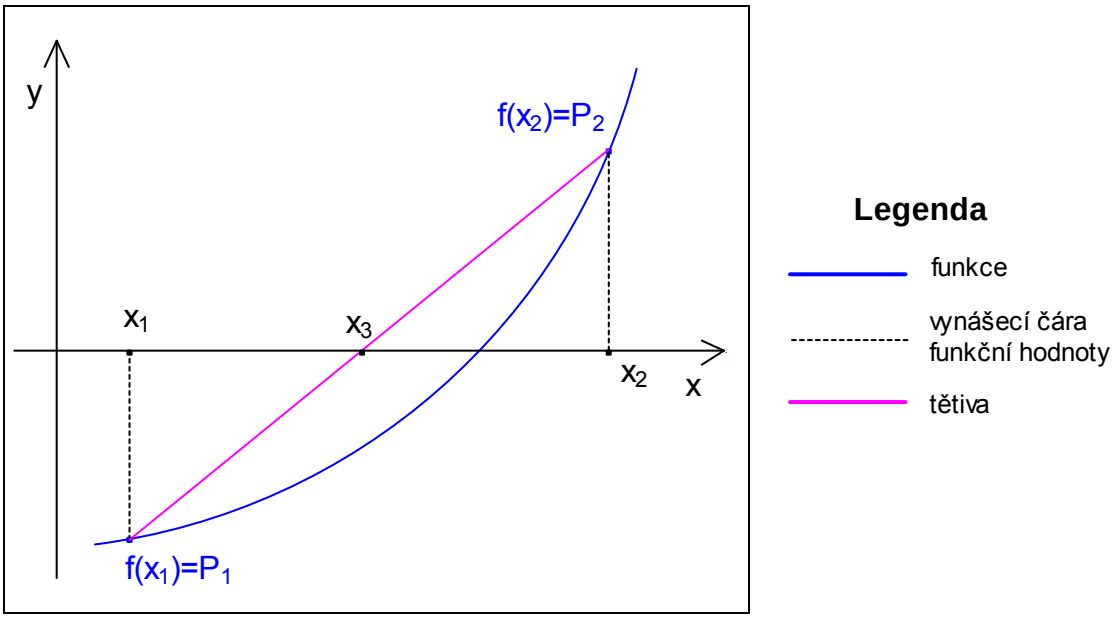
$$x_3 = x_1 - f(x_1)\frac{x_2 - x_1}{f(x_2) - f(x_1)}.$$

Tím jsme dostali výraz ze začátku kapitoly.

Geometrický význam metody tětiv

Křivka se nahradí tětivou vedenou body P_1 a P_2 (z toho plyne název *metoda tětiv*). Přitom aproximace x_3 kořene je první souřadnicí průsečíku osy x s přímkou P_1P_2.

Metoda tětiv krok za krokem



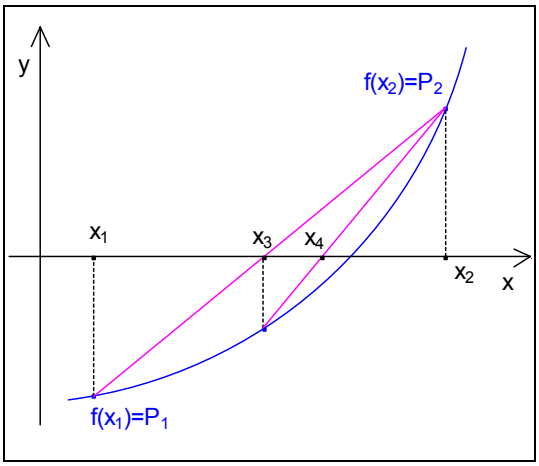
Obr. 5.1 Regula Falsi – první tětiva

1. krok

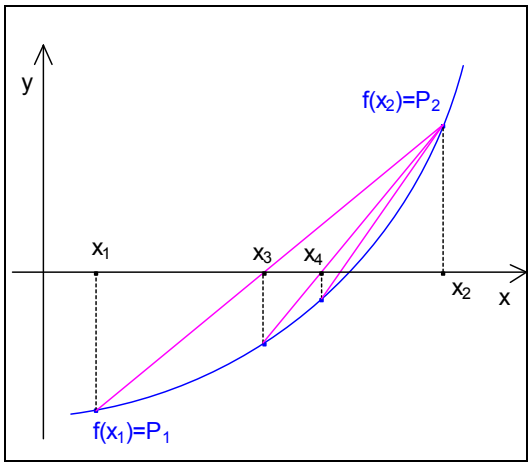
sestrojíme tětivu v krajních bodech intervalu. Tzn. v bodech $f(x_1) = P_1$ a $f(x_2) = P_2$. Najdeme bod, kde tětiva protíná osu x (což je x_3).

2. krok

Zjistíme, který ze dvou vzniklých intervalů splňuje předpoklady. V něm opakujeme předchozí postup. Pokračujeme do té doby, dokud nedospějeme ke kořeni, anebo se k němu nepřiblížíme na stanovenou vzdálenost (odchylku, chybu).



Obr. 5.2 Regula Falsi – druhá tětiva



Obr. 5.3 Regula Falsi – třetí tětiva

Odhad chyby

Poznámka. Budeme využívat univerzálního odhadu chyby z kapitoly 4.1. To znamená:

Označíme-li m nejmenší hodnotu funkce $|f'(x)|$ v intervalu $\langle a,b\rangle$, pak

$$|x_k - \alpha| \leq \frac{|f(x_k)|}{m}.$$

5.1.2 Interpretace 2

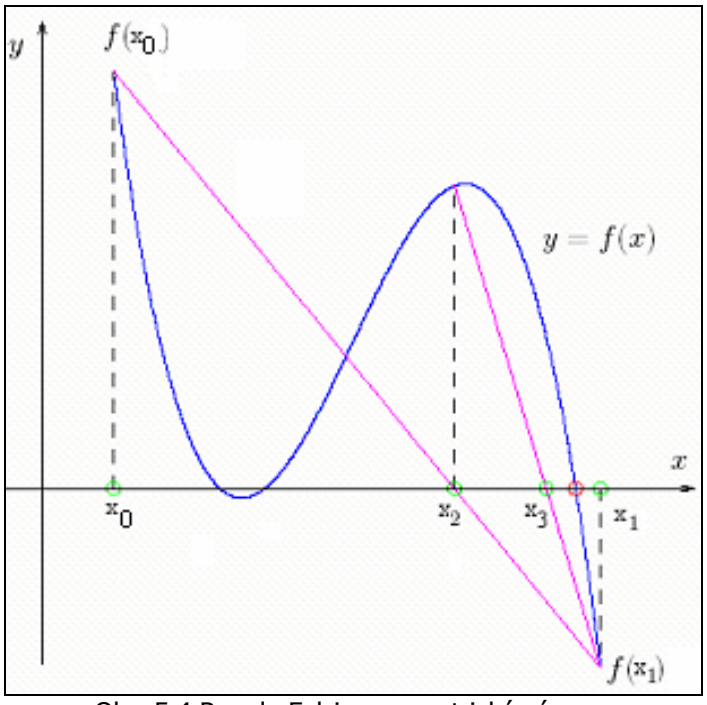
(čerpáno z literatury Rektorys [10], str. 613)

Touto metodou lze řešit rovnici $f(x)=0$ [$f(x)$ je reálná spojitá funkce v nějakém intervalu I], jestliže známe dvě čísla x_0 a x_1 [z tohoto intervalu], pro něž

$f(x_0)$ a $f(x_1)$ mají opačná znaménka, tj. $f(x_0)f(x_1) < 0$. Potom si totiž vypočteme postupně čísla $x_2, x_3, \ldots$ takto: Položíme $x_2 = \dfrac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$.

Je-li $f(x_2) = 0$, jsme hotovi. Je-li $f(x_2) \neq 0$, je buď $f(x_0)$, nebo $f(x_1)$ opačného znaménka než $f(x_2)$, takže s x_0 a x_2 nebo s x_1 a x_2 počítáme analogicky pomocí vzorce číslo x_3. Dále zase pomocí x_3 a jednoho z čísel x_2 a předtím vybraného čísla z x_0 a x_1 vypočteme x_4 atd. Posloupnost čísel $x_0, x_1, x_2, \ldots$ takto utvořená vždy konverguje, zpravidla však dosti pomalu.

Geometrický význam metody je následující



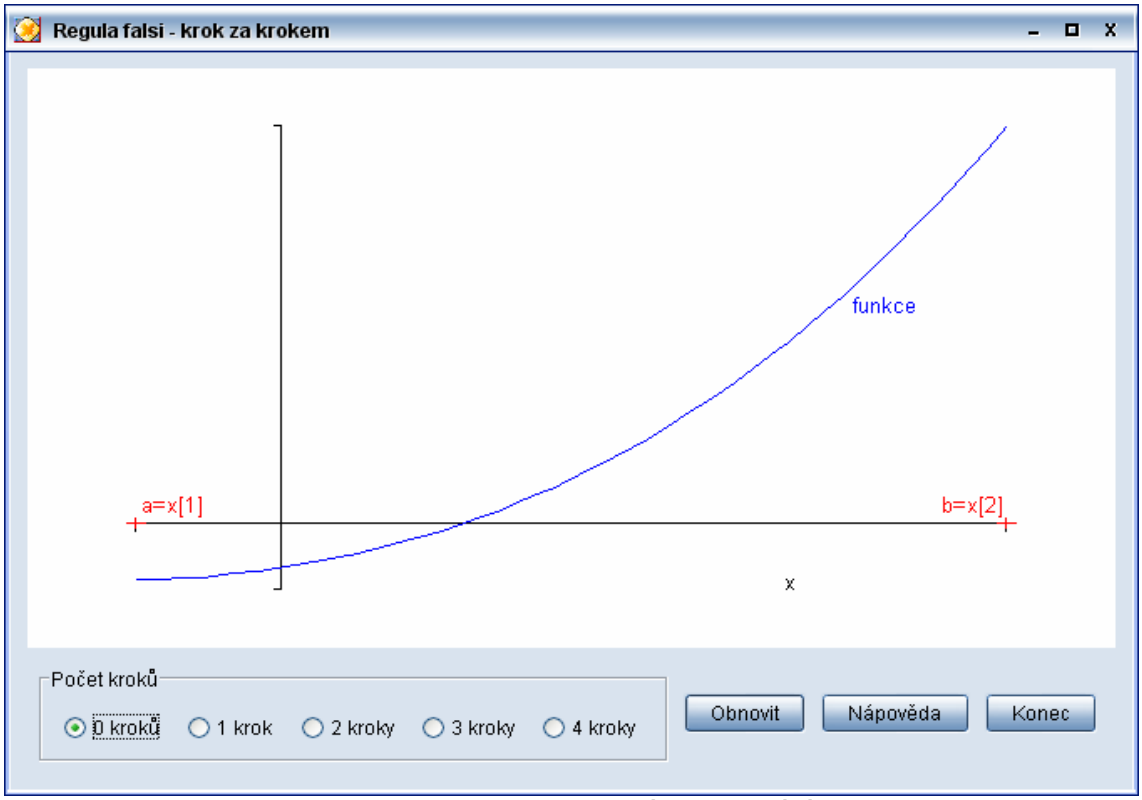
Obr. 5.4 Regula Falsi – geometrický význam

Poznámka. U této interpretace může být na daném intervalu i více kořenů (obr. 5.4).

5.2 Vizualizace metody

Stejně jako u předchozích metod, i zde, jsem vytvořila Maplet, který nám předvede princip této metody v dynamické podobě. Jmenuje se „Regula falsi – krok za krokem[6]“ (podrobný popis geometrického významu byl uveden v 5.1.1).

Použití Mapletu je intuitivní (pomocí radiobuttonů – „zaškrtávací puntíky“ s označením 0 – 4, zastupující jeden krok metody). Nultý, první a třetí krok je uveden na obr. 5.5 – 5.7. Na posledním obrázku je nádherně vidět, jak jsme se přiblížili ke kořeni.

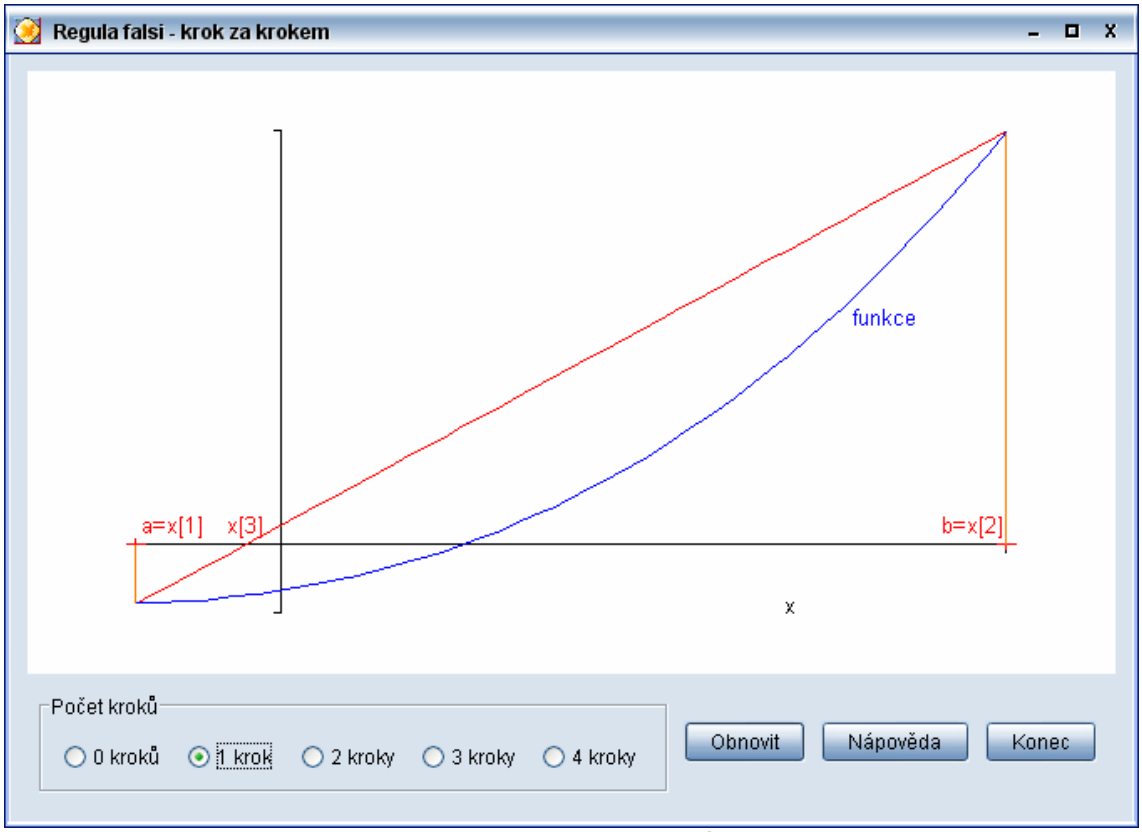


Obr. 5.5 Regula Falsi – nultý krok (zadání)

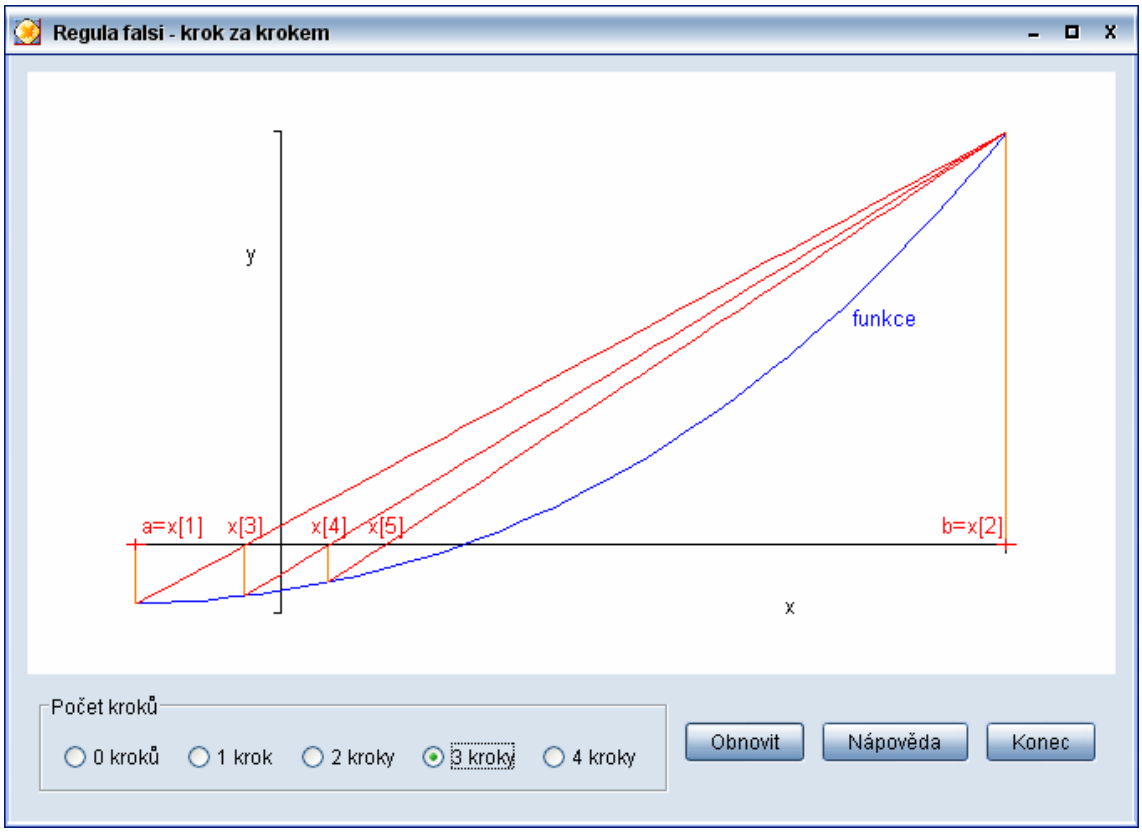
Funkce tlačítek:

- *Obnovit* – návrat do nultého kroku (zadání = původní interval)
- *Nápověda* – zobrazí krátký popis Mapletu
- *Konec* – ukončí Maplet

[6] název souboru „5.2 Vizualizace\Regula falsi – krok za krokem.maplet“
zdrojový kód je obsažen v: „5.2 Vizualizace\Regula falsi – krok za krokem.mws“



Obr. 5.6 Regula Falsi – první krok



Obr. 5.7 Regula Falsi – třetí krok

5.3 Příklad

$$P(x) = 4x^3 + 11x^2 + 5x - 2$$

Dle kapitoly 2 budeme další výpočty stavět na intervalu $\langle 0;1 \rangle$, ve kterém leží právě jeden reálný kladný kořen. Označím: $x_1 = 0,\ x_2 = 1$.

5.3.1 Ověření splnění předpokladů metody

Nejprve musíme ověřit předpoklady. Označíme si:

```
> pol:=4*x^3+11*x^2+5*x-2;
```

$$pol := 4x^3 + 11x^2 + 5x - 2$$

1) $f(x_1)f(x_2) < 0$

```
> pol_x[1]:=subs(x=0,pol);
  pol_x[2]:=subs(x=1,pol);
```

$$pol\_x_1 := -2$$

$$pol\_x_2 := 18$$

```
> evalb(pol_x[1]*pol_x[2]<0);
```

true

2) f je spojitá na $\langle x_1; x_2 \rangle = \langle 0;1 \rangle$

Funkce je spojitá na intervalu (viz kapitola 1.3 poznámka 4.).

3) ryze monotónní na intervalu $\langle x_1; x_2 \rangle = \langle 0;1 \rangle$

To znamená, že první derivace musí být na celém intervalu buď kladná nebo záporná (derivace nesmí být rovna nule a nesmí měnit znaménko).

```
> derivace:=diff(pol,x);
```

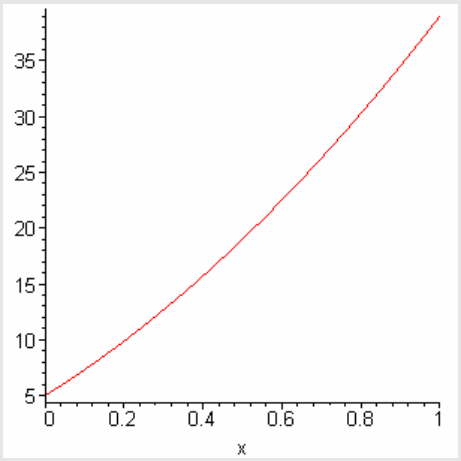
$$derivace := 12x^2 + 22x + 5$$

```
> evalf(solve(derivace=0));
```

-0.2658125271 , -1.567520806

Pro názornost zobrazím graf první derivace.

```
> plot(derivace,x=0..1);
```



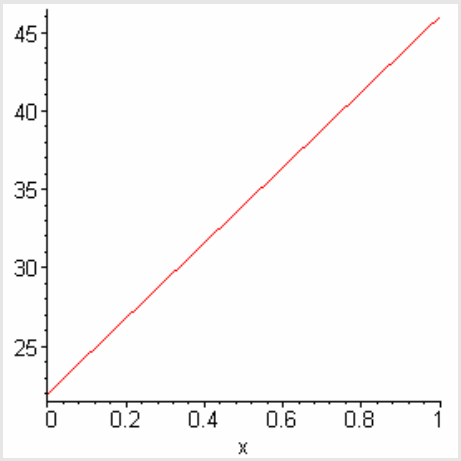
Vidíme, že na intervalu $\langle x_1; x_2 \rangle = \langle 0;1 \rangle$ je derivace kladná → funkce je ryze monotónní.

4) na intervalu $\langle x_1; x_2 \rangle = \langle 0;1 \rangle$ nemá inflexi

```
> derivace2:=diff(derivace,x);
```

$$derivace2 := 24x + 22$$

```
> plot(derivace2,x=0..1);
```




```
> prus:=solve(derivace2=0);
```

$$prus := \frac{-11}{12}$$

```
> is(prus,RealRange(0,1));
```

false

Závěr. Všechny předpoklady jsou splněny.

5.3.2 Grafické řešení

Maple 9.5 nám nenabízí žádné procedury pro grafické řešení problému. Proto jsem musela vytvořit příkaz `regulafalsi` i Maplet „Regula falsi – grafické znázornění[7]“.

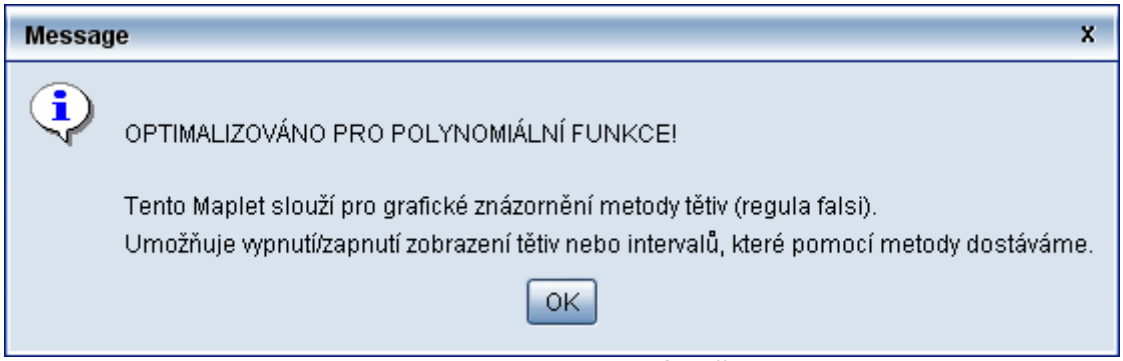
Popis jednotlivých částí Mapletu (znázorněn na obr. 5.9 – 5.12) je shodné jako u předchozích Mapletů („Bisekce – grafické znázornění“ nebo „Newtonova metoda – grafické znázornění“). Rozdíl je pouze v označení jednoho tlačítka a to *Použít metodu tětiv* (Po stisknutí je vykreslen graf. Tlačítko je nutné stisknout po každé změně – přenastavení intervalů, počtu kroků a podobně.). Navíc jsou tu dvě zaškrtávací políčka (*Zobrazit tětivy* a *Zobrazit intervaly*). Jejich význam je:

Zobrazit tětivy – zapne/vypne zobrazení tětiv – po přepnutí není nutné stisknout tlačítko *Použít metodu tětiv* – k překreslení dojde automaticky

Zobrazit intervaly – zapne/vypne zobrazení intervalů (i zde dojde k automatickému překreslení)

Na obr. 5.9 – 5.12 jsou ukázány všechny možné kombinace zaškrtnutí a odškrtnutí políček. A to následovně

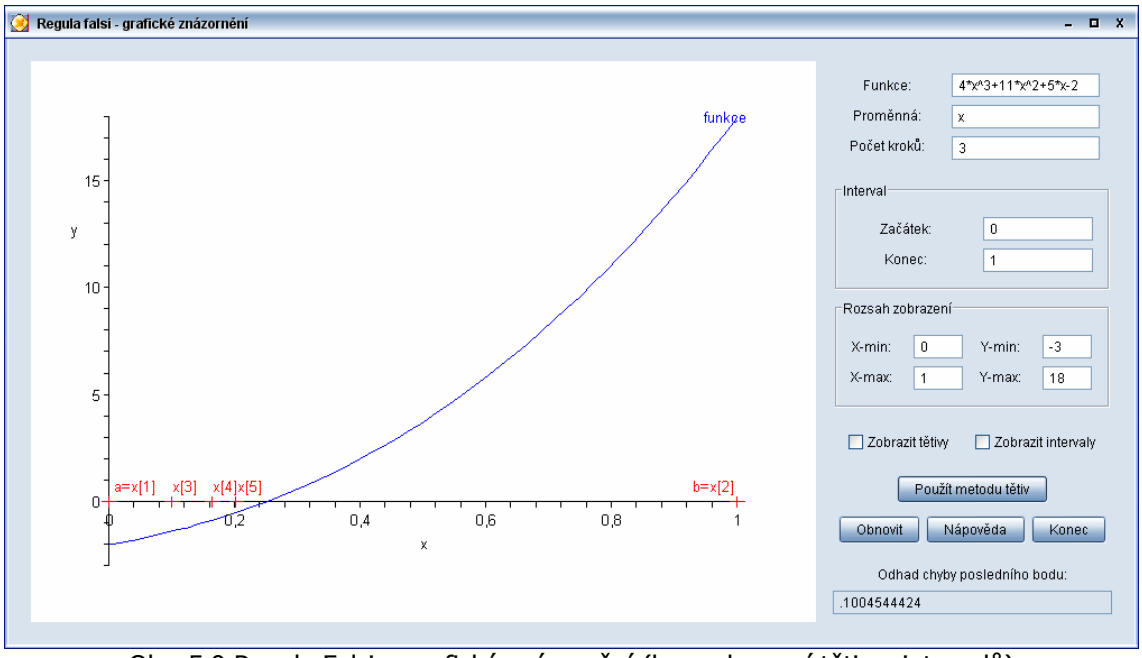
- obr. 5.9 – nejsou zobrazeny tětivy ani intervaly
- obr. 5.10 – zobrazeny pouze tětivy
- obr. 5.11 – zobrazeny pouze intervaly
- obr. 5.12 – zobrazeny jak tětivy, tak intervaly



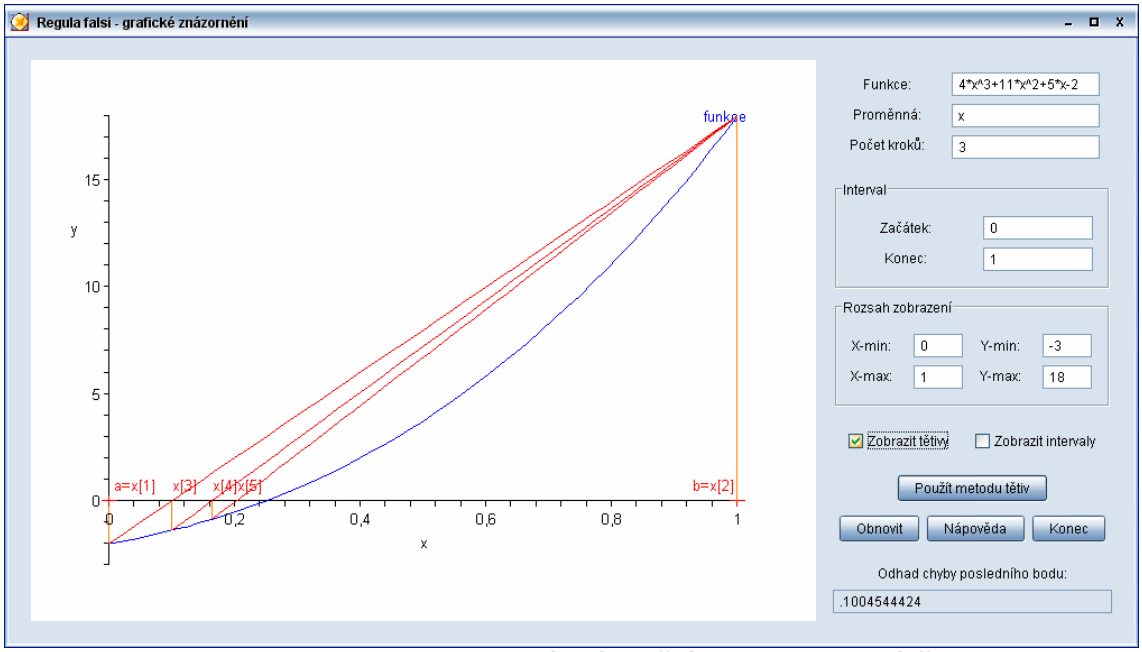
Obr. 5.8 Regula Falsi – nápověda

[7] název souboru „5.3 Priklad\Regula falsi - graficke.maplet“
zdrojový kód je obsažen v: „5.3 Priklad\Regula falsi - graficke.mws“

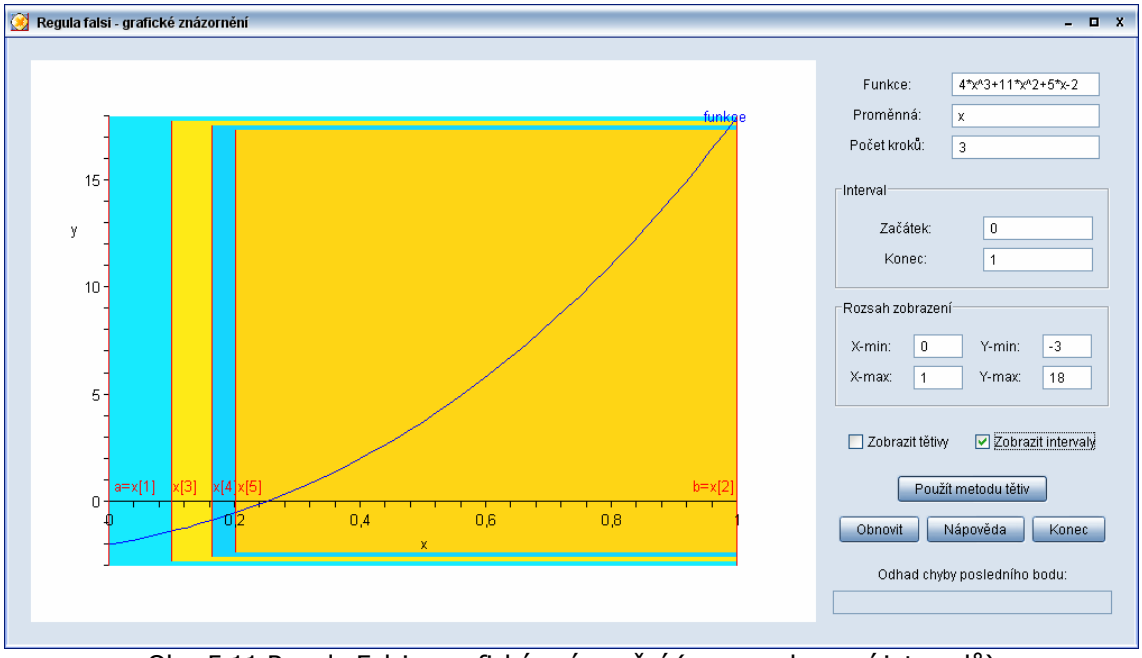
Stejně jako u Newtonovy metody, zobrazuji odhad maximální chyby posledně získaného bodu. Použila jsem univerzální odhad chyby. V případě nalezení kořene by měla být chyba nulová nebo velmi malá.



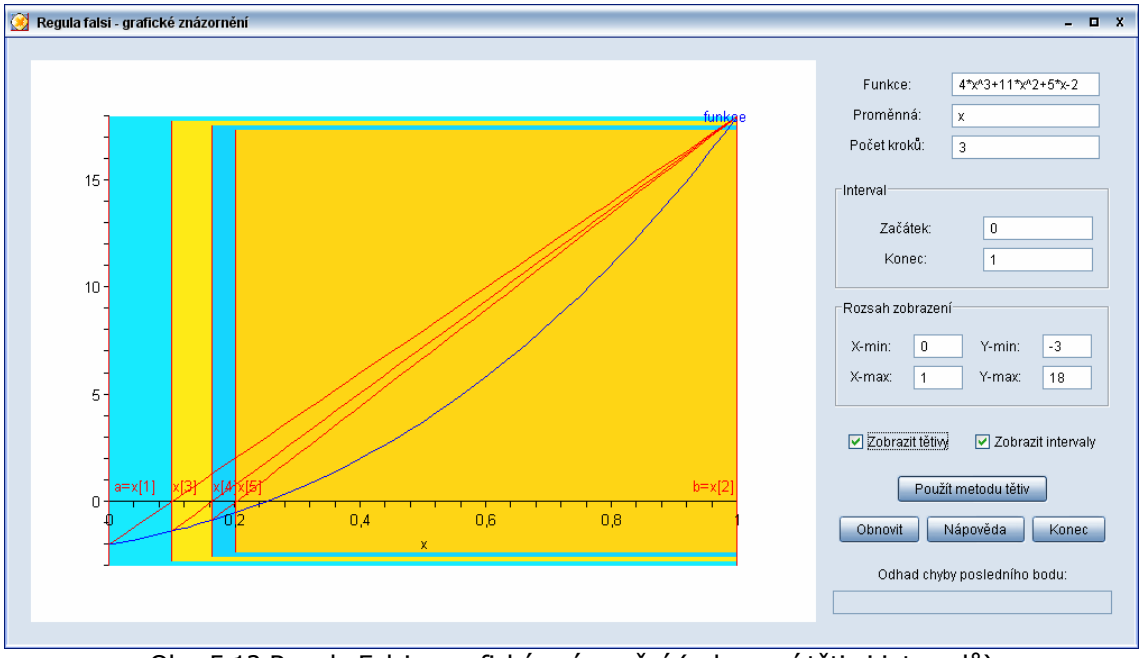
Obr. 5.9 Regula Falsi – grafické znázornění (bez zobrazení tětiv a intervalů)



Obr. 5.10 Regula Falsi – grafické znázornění (pouze zobrazení tětiv)



Obr. 5.11 Regula Falsi – grafické znázornění (pouze zobrazení intervalů)



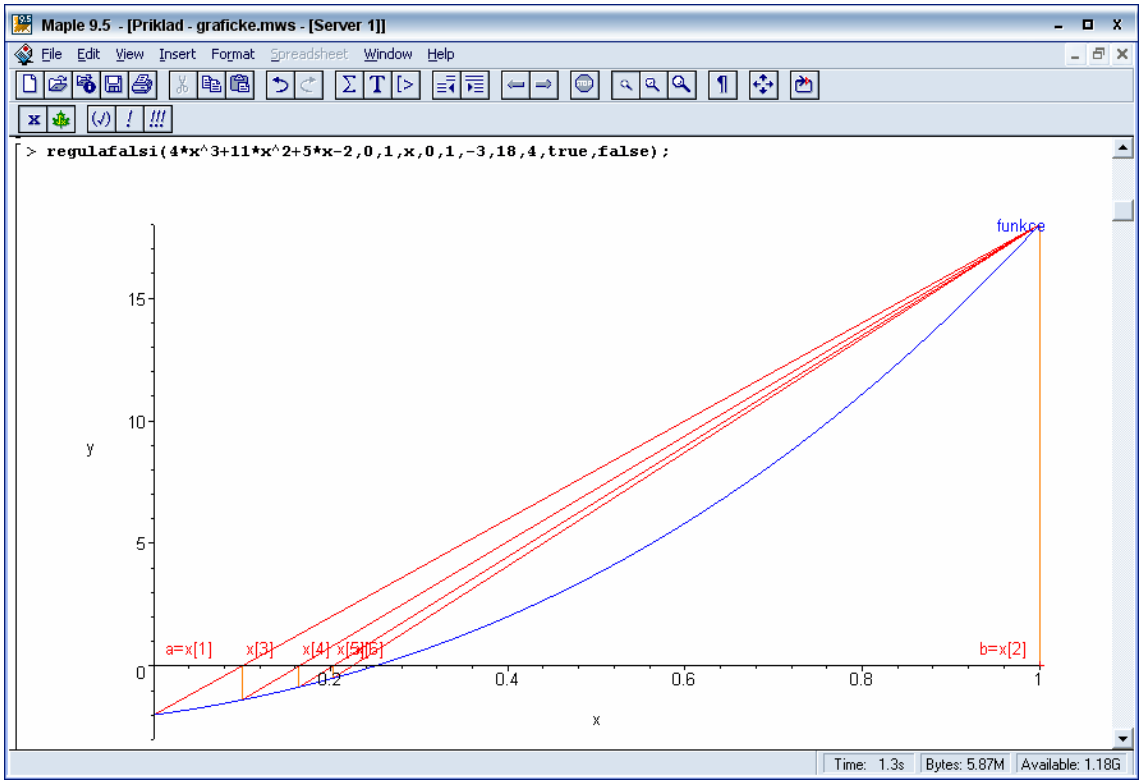
Obr. 5.12 Regula Falsi – grafické znázornění (zobrazení tětiv i intervalů)

Procedura je: `regulafalsi(Fce,a,b,prom,xmin,xmax,ymin,ymax, kroky,zda_tet,zda_int)`, najdeme ji v souboru „5.3 Priklad\Priklad – graficke.mws“, ale nejprve je nutno provést „Execute the worksheet“. Výsledek je zobrazen na obr. 5.13.

Popis parametrů:

- `Fce` – funkce
- `a,b` – rozsah intervalu
- `prom` – proměnná
- `xmin, xmax, ymin, ymax` – rozsah zobrazení
- `kroky` – celé nezáporné číslo určující počet kolikrát bude metoda použita
- `zda_tet` – true/false zapne/vypne zobrazení tětiv
- `zda_int` – true/false zapne/vypne zobrazení intervalů

```
> regulafalsi(4*x^3+11*x^2+5*x-2,0,1,x,0,1,-3,18,4,true,false);
```



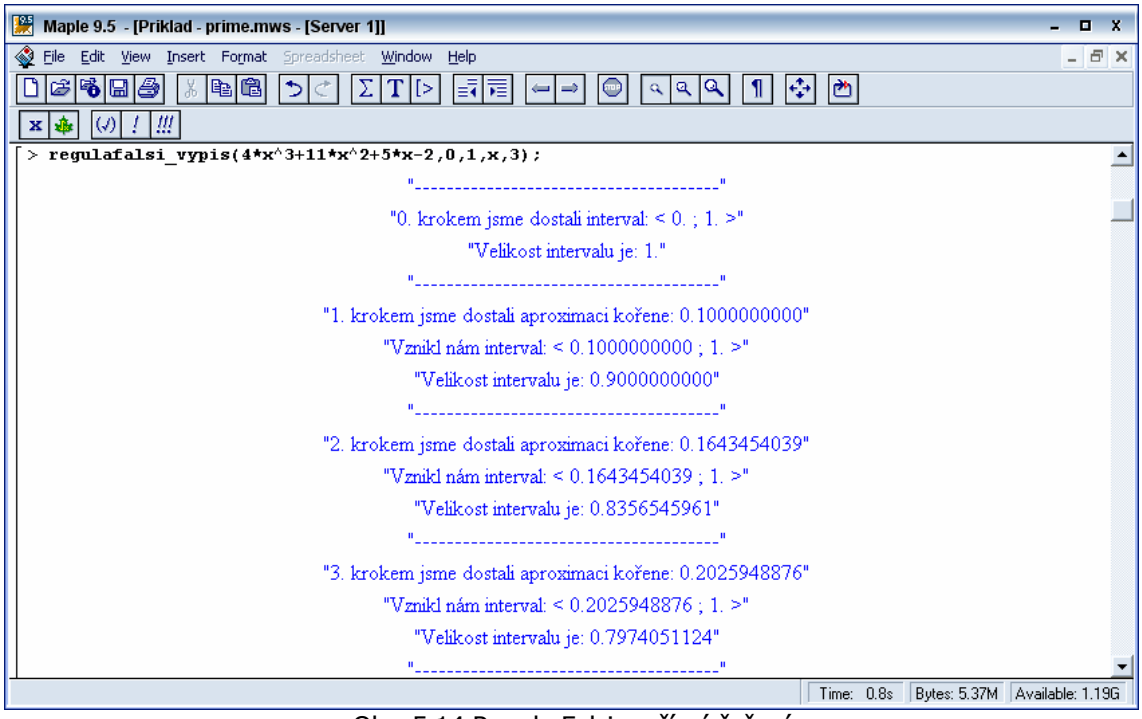
Obr. 5.13 Regula Falsi – procedura

5.3.3 Numerické řešení (přímé)

Stejně jako u grafického aparátu (grafického řešení) ani zde Maple 9.5 nenabízí žádný nástroj. V souboru „5.3 Priklad\Priklad – prime.mws“ je obsažena moje metoda

`regulafalsi_vypis(Fce,a,b,prom,kroky)`, která nám vypočítá postupné aproximace kořene. Parametry jsou obdobné jako u `regulafalsi`. Nejprve je nutné provést „Execute the worksheet“.

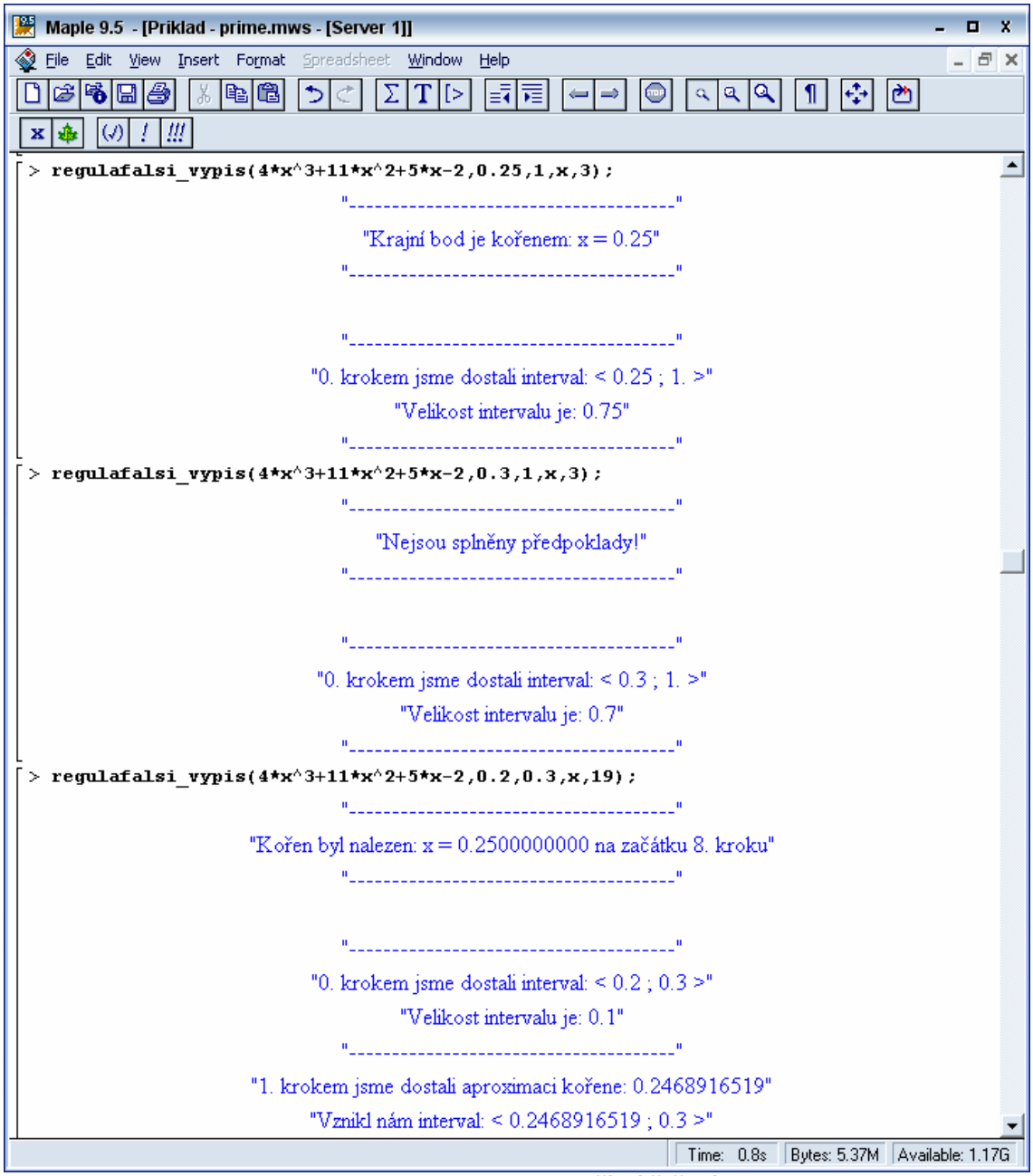
```
> regulafalsi_vypis(4*x^3+11*x^2+5*x-2,0,1,x,3);
```



Obr. 5.14 Regula Falsi – přímé řešení

Příkaz `regulafalsi_vypis` nás informuje nejen o tom, jaká aproximace kořene byla získána, ale i o krajních bodech intervalu a jeho velikosti (což je vidět na obr. 5.14). Dále o tom, zda jsou splněny předpoklady, jestli je jeden z krajních bodů kořenem, nebo dostaneme-li se ke kořeni do zadaného počtu kroků. Všechny zmíněné situace jsou ukázány v následující tabulce a na obr. 5.15:

```
> regulafalsi_vypis(4*x^3+11*x^2+5*x-2,0.25,1,x,3);
> regulafalsi_vypis(4*x^3+11*x^2+5*x-2,0.3,1,x,3);
> regulafalsi_vypis(4*x^3+11*x^2+5*x-2,0.2,0.3,x,19);
```



Obr. 5.15 Regula Falsi – přímé řešení

5.3.4 Numerické řešení (postupné)

Předpoklady pro první interval nebudu ověřovat (stalo se tak v 5.3.1). Jelikož se nově vzniklým bodem nedostaneme ven z intervalu (nemělo by se tak stát), musíme v dalších krocích ověřovat pouze to, že součin funkčních hodnot krajních bodů je menší než nula. Tím také budeme rozhodovat, která část intervalu nás po rozdělení bude zajímat (se kterou budeme dále pracovat).

Označení:

```
> restart;
  with(linalg):
> pol:=4*x^3+11*x^2+5*x-2;
  x[1]:=0;
  x[2]:=1;
```

$pol := 4x^3 + 11x^2 + 5x - 2$

$x_1 := 0$

$x_2 := 1$

Vypočteme funkční hodnoty:

```
> pol_x[1]:=subs(x=x[1],pol);
  pol_x[2]:=subs(x=x[2],pol);
```

$pol\_x_1 := -2$

$pol\_x_2 := 18$

Spočítáme x_3 dosazením do $x_3 = x_1 - \frac{(x_2 - x_1) f(x_1)}{f(x_2) - f(x_1)}$:

```
> x[3]:=x[1]-((x[2]-x[1])*pol_x[1])/(pol_x[2]-pol_x[1]);
```

$x_3 := \frac{1}{10}$

```
> evalf(x[3]);
```

0.1000000000

Potřebujeme znát funkční hodnotu x_3:

```
> pol_x[3]:=subs(x=x[3],pol);
```

$pol\_x_3 := \frac{-693}{500}$

```
> evalf(pol_x[3]);
```

-1.386000000

Na jejím základě rozhodneme, jaký bude nový interval (ostatní předpoklady není třeba ověřovat, platí pro celý interval):

```
> evalb(pol_x[1]*pol_x[3]<0);
  evalb(pol_x[3]*pol_x[2]<0);
```

false

true

Nový interval je $\langle x_3; x_2 \rangle$. Předpoklad o funkčních hodnotách byl ověřen, nyní můžeme spočítat hodnotu x_4 a její funkční hodnotu:

```
> x[4]:=x[3]-((x[2]-x[3])*pol_x[3])/(pol_x[2]-pol_x[3]);
```

$x_4 := \frac{59}{359}$

```
> evalf(x[4]);
        0.1643454039
> pol_x[4]:=subs(x=x[4],pol);
        pol_x[4] := -39948678/46268279
> evalf(pol_x[4]);
        -0.8634139601
```

Takto bychom pokračovali do nalezení kořene, nebo do určité délky intervalu, velikosti chyby, či do zvoleného počtu kroků. Záleží na zadání.

Odhad chyby

$|x_k - \alpha| \leq \frac{|f(x_k)|}{m}$, m je nejmenší hodnota funkce $|f'(x)|$ v intervalu $\langle a, b \rangle$.

Nejmenší hodnotu získáme za použití nástroje `minimize`.

Popis parametrů:

1. parametr – funkce
2. parametr – rozmezí intervalu, na kterém hledáme minimum

```
> m:=minimize(abs(diff(pol,x)),x=0..1);
        m := 5
```

Pokud nás zajímá přesná poloha minima, přidáme parametr `location`.

```
> minimize(abs(diff(pol,x)),x=0..1,location);
        5, {[{x = 0}, 5]}
```

Dopočítáme:

```
> odhad_chyby:=evalf((abs(pol_x[4]))/m);
        odhad_chyby := 0.1726827920
```

Závěr. $|x_4 - \alpha| < 0.1726827920$, jedná se o druhou aproximaci kořene.

Poznámka. Pro srovnání $x_{10} = 0.2478981244$ (osmá aproximace kořene) a odhad chyby je: $|x_{10} - \alpha| < 0.004716857400$

6 Přehled výsledků použitých metod

I když je cílem práce seznámit čtenáře s možnostmi Maple 9.5. Domnívám se, že by bylo chybou neuvést přehled dosažených výsledků použitých metod. Což udělám v této kapitole. Cílem je ukázat, že rychlost nalezení kořene velmi záleží na počáteční volbě intervalu popř. bodu. Velkou roli v tom hraje zkušenost, náhoda, ale i štěstí.

Z tohoto důvodu rozdělím kapitolu na dvě části. V první zvolím za počáteční interval $\langle 0;1 \rangle$ a bod 1, v druhé $\langle 0.1;1 \rangle$ a 0,9. Maximálně však 10 kroků.

6.1 Interval $\langle 0;1 \rangle$ a počáteční bod 1

1. Metoda bisekce

Počet kroků	**Interval**		**Délka intervalu**	**Odhad chyby**
0	0	1	1	0,5
1	0	0,5	0,5	0,25
2	Kořen nalezen: $x = 0{,}25$			

2. Newtonova metoda

Počet kroků	**Aproximace kořene**	**Odhad chyby**
0	1	
1	0,5384615385	0,9798816568
2	0,3167618409	0,2260934774
3	0,2549177442	0,01759358456
4	0,2500298148	$0{,}1099025231 \cdot 10^{-3}$
5	0,2500000011	$0{,}4088734206 \cdot 10^{-8}$
6	0,2499999999	$0{,}5628410309 \cdot 10^{-17}$
7	Kořen: $x = 0{,}25$	$0{,}1066509418 \cdot 10^{-34}$

3. Metody regula falsi

Počet kroků	Aproximace kořene	Interval		Délka intervalu	Odhad chyby
0	-	0	1	1	
1	0,1	0,1	1	0,9	0,2772
2	0,1643454039	0,1643454039	1	0,8356545961	0,1726827920
3	0,2025948876	0,2025948876	1	0,7974051124	0,1004544424
4	0,2242416575	0,2242416575	1	0,7757583425	0,0561121648
5	0,2361476051	0,2361476051	1	0,7638523949	0,0306327262
6	0,2425924545	0,2425924545	1	0,7574075455	0,0165136618
7	0,2460509157	0,2460509157	1	0,7539490843	0,0088418224
8	0,2478981244	0,2478981244	1	0,7521018756	0,0047168574
9	0,2488822675	0,2488822675	1	0,7511177325	0,002511401
10	0,2494058905	0,2494058905	1	0,7505941095	0,0013357584

Závěr. Vidíme, že u této volby intervalu a bodu byla nejúspěšnější metodou bisekce. Ale hned v následující části se přesvědčíme o tom, že to neplatí vždy, spíše naopak.

6.2 Interval $\langle 0{,}1;1\rangle$ a počáteční bod 0,9

1. Metoda bisekce

Počet kroků	Interval		Délka intervalu	Odhad chyby
0	0,1	1	0,9	0,45
1	0,1	0,55	0,45	0,225
2	0,1	0,325	0,225	0,1125
3	0,2125	0,325	0,1125	0,05625
4	0,2125	0,26875	0,05625	0,028125
5	0,240625	0,26875	0,028125	0,0140625
6	0,240625	0,2546875	0,0140625	0,00703125
7	0.24765625	0,2546875	0,00703125	0,003515625
8	0,24765625	0,251171875	0,003515625	0,001757813
9	0,2494140625	0,251171875	0,0017578125	0,000878906
10	0,2494140625	0,2502929687	0,0008789062	0,000439453

2. Newtonova metoda

Počet kroků	Aproximace kořene	Odhad chyby
0	0,9	
1	0,4849942063	0,7922571205
2	0,2974206818	0,1618456046
3	0,2525653144	0,00925521833
4	0,2500081494	$0{,}3007982705 \cdot 10^{-4}$
5	0,2500000001	$0{,}3054910162 \cdot 10^{-9}$
6	Kořen: $x = 0{,}25$	$0{,}46 \cdot 10^{-19}$

3. Metoda regula falsi

Počet kroků	Aproximace kořene	Interval		Délka intervalu	Odhad chyby
0	-	0,1	1	0,9	
1	0,1643454039	0,1643454039	1	0,8356545961	0,1179527268
2	0,2025948876	0,2025948876	1	0,7974051124	0,0686164224
3	0,2242416575	0,2242416575	1	0,7757583425	0,03832798142
4	0,2361476051	0,2361476051	1	0,7638523949	0,02092399331
5	0,2425924545	0,2425924545	1	0,7574075455	0,01127982363
6	0,2460509157	0,2460509157	1	0,7539490843	0,006039496175
7	0,2478981244	0,2478981244	1	0,7521018756	0,003221897131
8	0,2488822675	0,2488822675	1	0,7511177325	0,001715437842
9	0,2494058905	0,2494058905	1	0,7505941095	0,0009124032787
10	0,2496842906	0,2496842906	1	0,7503157094	0,0004850185792

V tomto případě nám změna nepřinesla žádné ovoce. Protože levý krajní bod intervalu $\langle 0{,}1;1 \rangle$ je první aproximací metody regula falsi na intervalu $\langle 0;1 \rangle$. Tím pádem jsou výsledky stejné (až na odhady), jen posunuté o jeden řádek.

Závěr. U této volby údajů vyhrála Newtonova metoda. Je patrné, že ne vždy se na poprvé trefíme do správné metody a ještě intervalu či bodu. Z tohoto důvodu je optimální strategií metody mezi sebou kombinovat. A to tak, že pomocí bisekce

stanovíme interval nevelké délky (popř. požadované délky) a na něj aplikujeme jednu z metod (regula falsi nebo Newtonovu metodu). Preferuji Newtonovu, protože je mnohem rychlejší. V našem případě se o tom můžeme přesvědčit pomocí odhadu chyb (čerpáno z tabulek v této kapitole).

– odhad chyby po třech krocích:

- Newtonova metoda: 0,00925521833
- regula falsi: 0,03832798142

– odhad chyby po pěti krocích:

- Newtonova metoda: $0{,}3054910162 \cdot 10^{-9}$
- regula falsi: 0,01127982363

Při použití více metod, nesmíme zapomenout vždy zkontrolovat předpoklady.

7 Pozadí tvorby Mapletů

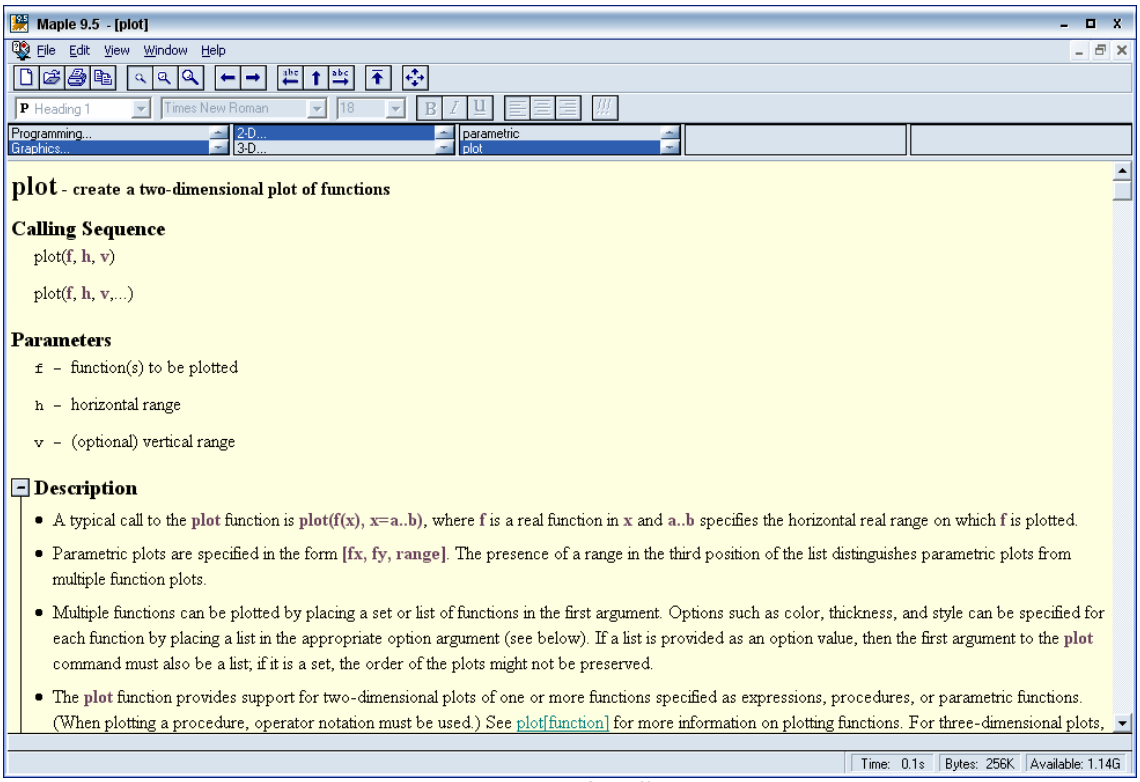
Možná Vás v průběhu čtení diplomové práce napadlo, jak se vlastně Maplety tvoří. Proto jsem pro Vás připravila tuto kapitolu. Uvedu zde nejen základy tvorby Mapletů, ale i vlastních knihoven. Zařadila jsem ji na konec, z toho důvodu, že námětem není tvorba Mapletů, ale využití různých nástrojů Maple 9.5.

Pokud neuvedu přesný popis některých z funkcí, odpověď naleznete v příloze A nebo v nápovědě. Jak používat nápovědu by měl čtenář znát, ale pro jistotu uvedu několik možností práce s Helpem.

Může nastat situace: znáte název příkazu, ale pouze si nepamatujete parametry, jejich pořadí nebo způsob použití. Vhodný je příkaz `?`. Následující příklad demonstruje získání informací o `plot`:

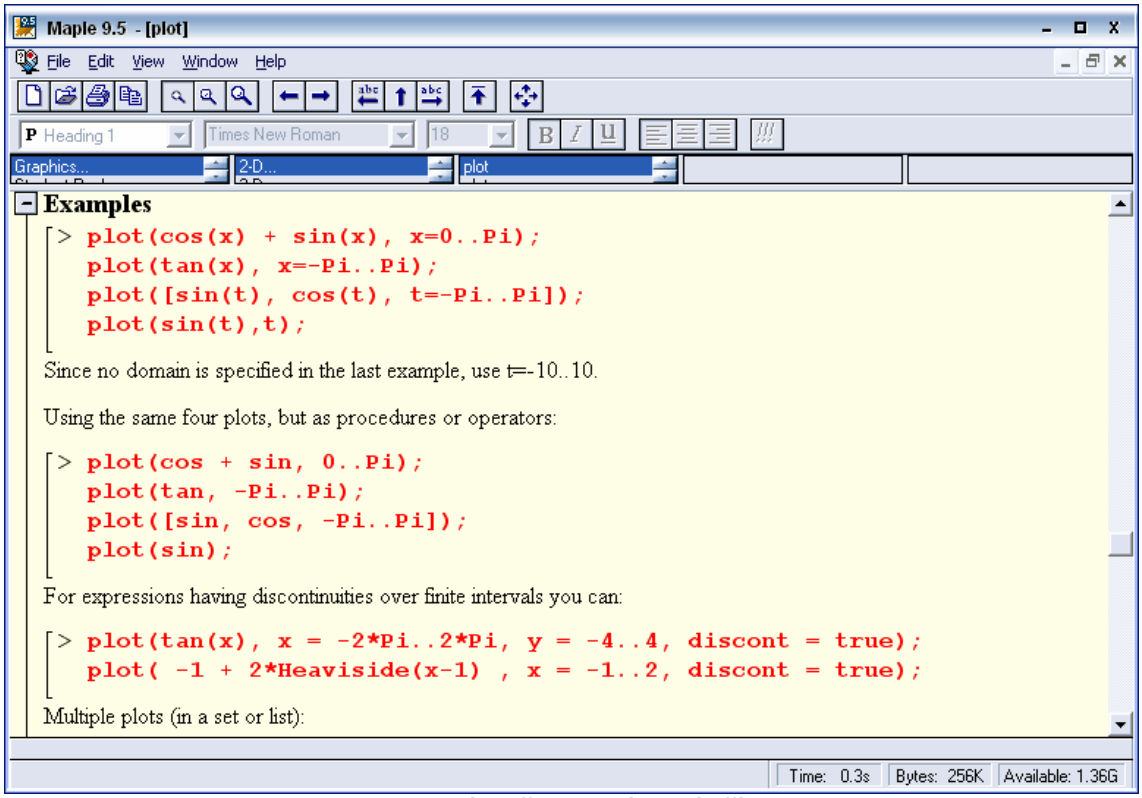
```
> ? plot
```

Otevře se nám okno s nápovědou.



Obr. 7.1 Nápověda

V dolní části jsou zobrazeny ukázkové příklady.



Obr. 7.2 Nápověda – ukázkové příklady

Další možností je napsat příkaz. Označit ho myší, nebo umístit kurzor kdekoliv ve slově a stisknout klávesu F1.

Dále můžeme použít menu *Help – Topic Search ...* (hledá téma obsahující klíčové slovo) nebo *Help – Full Text Search ...* (prohledává celé texty nápovědy).

7.1 Tvorba vlastních knihoven

Důležitým prvkem Maplu 9.5 je tvorba vlastních knihoven. Je to velmi nápomocné při použití funkcí v reálných situacích (při řešení příkladů apod.). Odpadne nám tím neustálé znovudefinování procedur. Jednoduše si ji (popř. více procedur) uložíme do knihovny, kterou můžeme kdykoliv použít. Bez nutnosti znovu vypisovat nebo kopírovat zdrojový kód funkce.

Nezbytnou součástí je základní znalost tvorby procedur. Což bylo popsáno v kapitole 3.4 Procedury.

Než začneme vytvářet knihovnu, musíme do pracovního listu uvést definice všech procedur, které má knihovna obsahovat. Nesmíme zapomenout na začátku vymazat vnitřní paměť. Tím odstraníme problém převzetí proměnných z jiných rozpracovaných problémů (listů). Vymazání provedeme příkazem `restart.`

Další bod, na který nesmíme zapomenout, je inicializovat balíčky funkcí, které využíváme. Pomocí `with(nazev_balicku).` Např.:

```
> with(plots):
```

Nyní zde uvedu přehled funkcí, které budeme využívat. Pak bude následovat aplikace příkazů a ukázkové použití.

- `libname` – bez parametrů – výpis aktuálně využívaných knihoven
- `libname := libname, "název_knihovny.lib"` – zavedení knihovny do paměti
- `march( 'create',"název_knihovny.lib")` – vytvoření vlastní knihovny
- `savelibname := "název_knihovny.lib"` – tímto určíme, do které knihovny se budou funkce ukládat
- `savelib( 'název_funkce')` – tímto uložíme zvolenou funkci do knihovny

Příkazy musí být řazeny do určité logické posloupnosti. Protože nelze například vytvořit knihovnu a ukládat do ní procedury, které ještě nebyly definovány. Celý postup může být rozdělen do pěti základních bloků:

1. blok – vymazání vnitřní paměti
2. blok – seznam použitých balíčků
3. blok – definice procedur
4. blok – tvorba knihovny, ukládání procedur, …
5. blok – zavedení knihovny do paměti

Nyní můžeme začít využívat knihovnu v aktuálním souboru (tzn. volat funkce, používat je), nebo v úplně novém, jak si ukážeme níže.

Konkrétní příklad[8]:

Na konkrétním příkladu ukážu jak vytvořit knihovnu a uložit do ní funkce. Nejprve je musíme definovat, budou celkem tři:

1. `funkcni_hodnota(f,bod,prom)` – vrací funkční hodnotu funkce `f` v bodě `bod` a `prom` určuje o jakou proměnnou jde
2. `der(f,bod,prom)` – vrací derivaci funkce `f` v bodě `bod` a `prom` určuje podle které proměnné se derivuje
3. `rovtec(f,bod,prom)` – vrací rovnici tečny funkce `f` v bodě `bod` a `prom` určuje proměnnou

Tyto procedury chceme uložit do knihovny `vlastni.lib`. Nyní se podíváme jak se to vše provede.

```
> # ------------- 1. blok -------------
> restart;

> # ------------- 2. blok -------------
> with(Maplets[Elements]):
  with(StringTools):

> # ------------- 3. blok -------------
> funkcni_hodnota := proc(f,bod,prom)
  return subs(prom=bod,f);
> end proc:

> der := proc(f,bod,prom)
  local deriv;
  deriv:=diff(f,prom);
  return subs(prom=bod, deriv);
> end proc:

> rovtec := proc(f,bod,prom)
  return der(f,bod,prom)*(prom-bod)+subs(prom=bod,f);
> end proc:

> # ------------- 4. blok -------------
> march('create',"vlastni.lib");
> savelibname := "vlastni.lib":
> savelib('funkcni_hodnota'):
  savelib('der'):
  savelib('rovtec'):

> # ------------- 5. blok -------------
> libname := libname, "vlastni.lib":
```

[8] zdrojový kód „7.1 Knihovna/knihovna.mws“, knihovna: „7.1 Knihovna/vlastni.lib“

Využití vytvořené knihovny[9]:

Otevřeme si nový pracovní list. V první části uvidíme, jak Maple 9.5 reaguje na volání procedur, které nezná. K nápravě stačí zavolat knihovnu `vlastni.lib` příkazem `libname := libname, "vlastni.lib"`. Pro jistotu doporučuji použít `libname` pro výpis inicializovaných knihoven.

```
> restart;
> # ------------- Bez zavolání knihovny -------------
> libname;
        "C:\Program Files\Maple 9.5/lib'
> funkcni_hodnota(x^2,5,x);
        funkcni_hodnota (x^2, 5, x)
> der(x^2,5,x);
        der (x^2, 5, x)
> rovtec(x^2,5,x);
        rovtec (x^2, 5, x)
>
>
> # ------------- Se zavoláním knihovny -------------
> libname := libname, "vlastni.lib":
> libname;
        "C:\Program Files\Maple 9.5/lib", "vlastni.lib"
> funkcni_hodnota(x^2,5,x);
        25
> der(x^2,5,x);
        10
> rovtec(x^2,5,x);
        10 x – 25
```

Kontrola vytvořené knihovny:

Pokud již máme vytvořenou knihovnu, je dobré zkontrolovat její obsah.

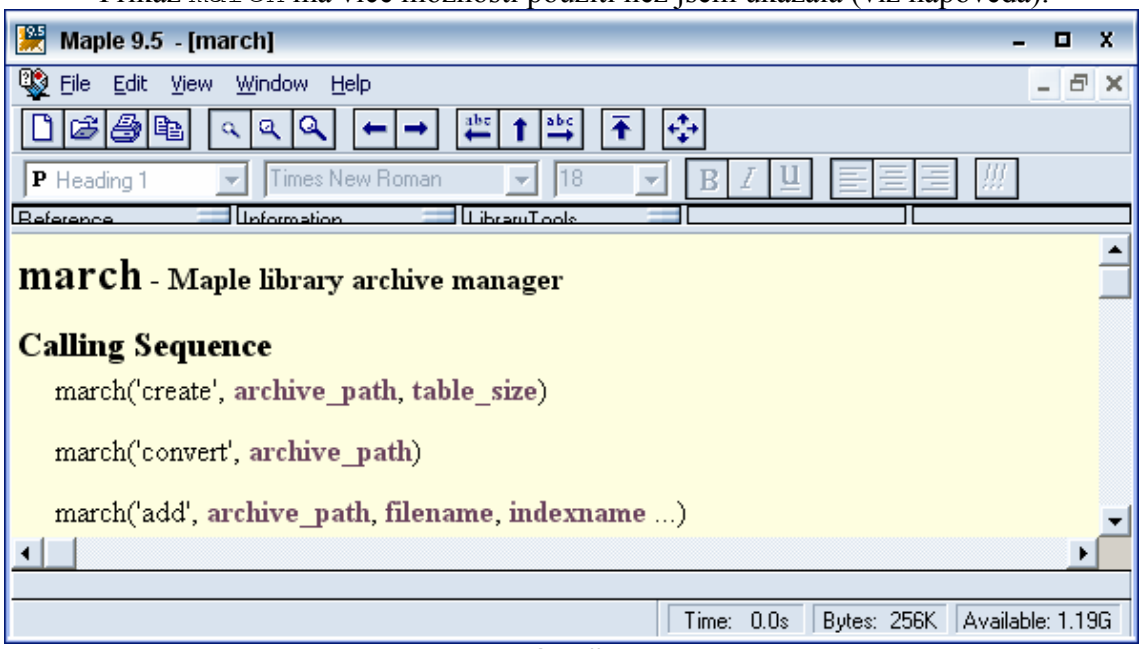
```
> march('list', "vlastni.lib");
        [["funkcni_hodnota.m" , [2007, 3, 9, 15, 23, 8], 1634, 76], ["der.m" , [2007, 3, 9, 15, 23, 8], 1710, 92],
                ["rovtec.m" , [2007, 3, 9, 15, 23, 8], 1802, 107], ["vypis.m" , [2007, 4, 16, 11, 40, 50], 1909, 106]]
```

A případně přebytečné či špatně vytvořené knihovny vymazat. Doporučuji provést opětovný výpis, pro kontrolu, zda smazání proběhlo úspěšně.

```
> march('delete',"vlastni.lib",'vypis');
> march('list', "vlastni.lib");
        [["funkcni_hodnota.m" , [2007, 3, 9, 15, 23, 8], 1634, 76], ["der.m" , [2007, 3, 9, 15, 23, 8], 1710, 92],
                ["rovtec.m" , [2007, 3, 9, 15, 23, 8], 1802, 107]]
```

[9] zdrojový kód „7.1 Knihovna/knihovna_pouziti.mws“

Příkaz `march` má více možností použití než jsem ukázala (viz nápověda).



Obr. 7.3 Nápověda – march

Poznámka na okraj. Pokud práci přenášíme z počítače na počítač, je potřeba věnovat více času umístění knihovny. Ta je uložena buď v aktuálním adresáři nebo v místě, kde je Maple 9.5 nainstalován, a to „C:\Program Files\Maple 9.5\“ nebo „C:\Program Files\Maple 9.5\LIB\“. A následně jejímu připojení k práci. Hlavně musíme správně uvést cesty ve všech souborech, kde byla použita. Když to neprovedeme, nemusí vše správně fungovat. Místo relativních cest, je lepší využít absolutních (hlavně v novém umístění). Což může někdy vyžadovat překompilovat celou práci nebo raději vygenerovat celou knihovnu znovu.

7.2 Tvorba Mapletů – základ

Již v průběhu čtení jste si mohli udělat představu o tom, jak Maplety fungují. Nyní Vás seznámím trochu s jejich tvorbou. Nejprve se však dozvíte základní informace.

Určitě jste se někdy setkali s javovskými aplety v internetových prohlížečích. V Maple 9.5 Maplety fungují na obdobném principu. Jsou to v podstatě interaktivní grafická uživatelská prostředí, která nám pomáhají snadněji přistupovat k maplovským

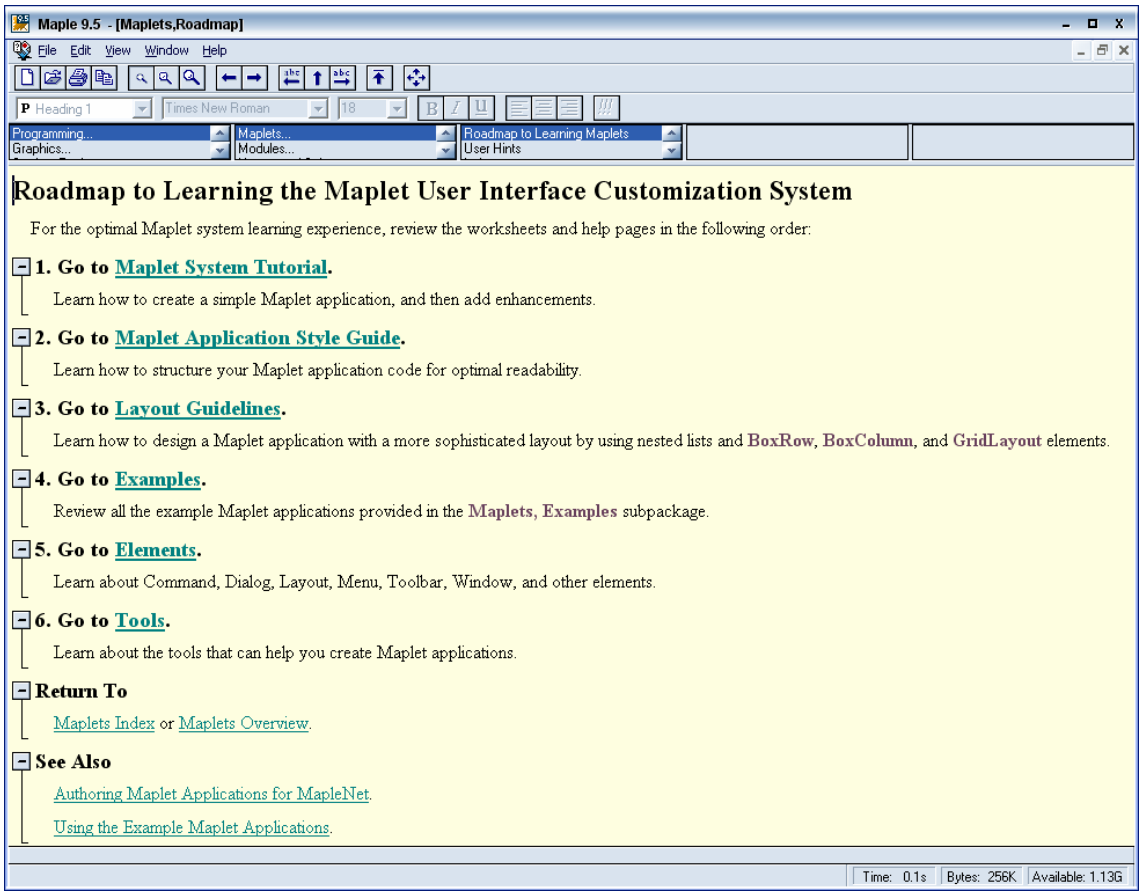
aplikacím. Můžeme využít nejen vstupů od uživatele, měnit zadání, vykreslovat grafy, ale provádět i mnohem složitější aplikace. Je to pouze na Vás, co vytvoříte.

Spuštění Mapletů může probíhat dvěma způsoby. A to buď přímo v Maple 9.5, nebo za pomoci MapleViewer, který je součástí instalace programu Maple 9.5. V prvním případě musíme mít zdrojový kód Mapletu a následně provést „Execute the worksheet“. V druhém stačí mít soubor s koncovkou *.maplet* a pouze ho spustit.

Možná Vás napadá otázka jak získat soubor s touto koncovkou. Pokud již máme napsaný zdrojový kód Mapletu, stačí použít menu *File – Save As* a zvolit *Uložit jako typ* Maplet.

Nyní se dostáváme k samotné tvorbě Mapletů. Nečekejte zde rozsáhlou publikaci, uvedu pouze nástin, jak začít. Důvod je, že touto problematikou se zabývá několik knih, spousta internetových stránek, ale i samotný Maple 9.5. Příkladem je dobře zpracovaná učebnice tzv. „Roadmap“ v Maple 9.5. Vyvoláme ji příkazem:

```
> ? roadmap;
```

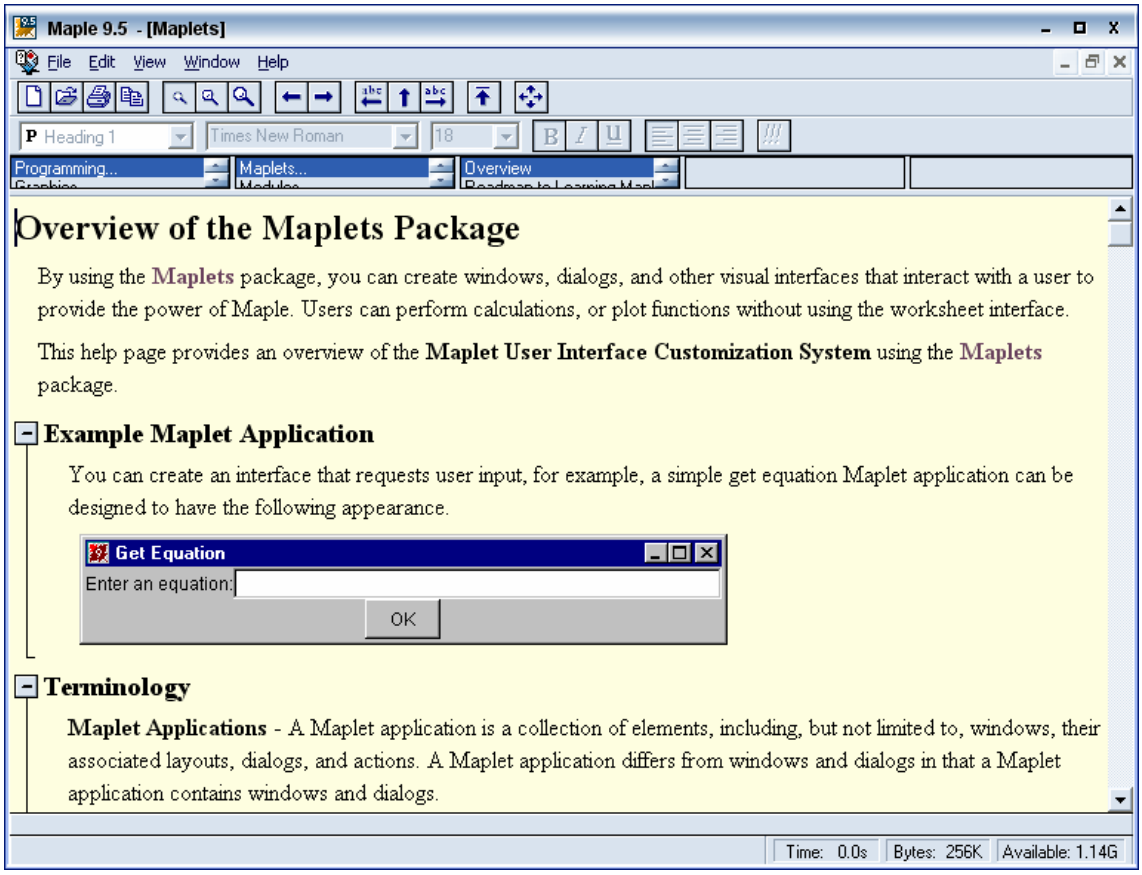


Obr. 7.4 Roadmap

Najdeme zde prakticky vše, co hledáme. Pro někoho je učebnice nepoužitelná, protože je v anglickém jazyce. Což může některým čtenářům způsobit nemalé problémy.

Další součástí Maple 9.5 je rozsáhlá dokumentace (opět v anglickém jazyce). Můžeme ji vyvolat příkazem:

```
> ? Maplets;
```



Obr. 7.5 Maplets

První krok před samotnou tvorbou Mapletů, je inicializovat balíček `Maplets`, konkrétně příkaz `Elements`:

```
> with(Maplets[Elements]):
```

Nyní je vše připraveno. Vytvoříme si první Maplet, který nám vypíše text „Toto je můj první Maplet!“. Kód je následující:

```
> maplet1:=Maplet([["Toto je můj první Maplet!"]]):
```

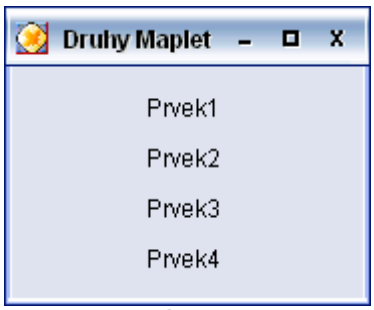
Označili jsme si ho jako „maplet1“. Teď už ho jenom spustíme:

```
> Maplets[Display](maplet1);
```

Obr. 7.6 První Maplet

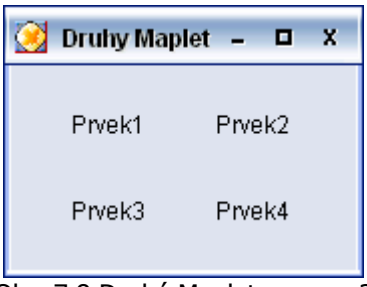
Pro tvorbu složitějších aplikací začneme využívat objektu `Window`. Lze u něho nastavit název (pomocí `'title'="Název Mapletu"`). Struktura je tvořena víceúrovňovým seznamem v hranatých závorkách. Použití je vidět v následujících dvou ukázkách:

```
> maplet2a:=Maplet(Window('title'="Druhy Maplet",
  [Prvek1,Prvek2,Prvek3,Prvek4])):
> Maplets[Display](maplet2a);
```



Obr. 7.7 Druhý Maplet – verze 1

```
> maplet2b:=Maplet(Window('title'="Druhy Maplet",
  [[Prvek1,Prvek2],[Prvek3,Prvek4]])):
> Maplets[Display](maplet2b);
```



Obr. 7.8 Druhý Maplet – verze 2

Používání je velmi intuitivní. Jen je třeba dbát na správné ukončování závorek. Myslím, že nemusím připomínat, že seznamy se mohou do sebe libovolně vnořovat.

7.2.1 Stavební prvky Mapletů

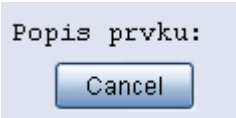
Součástí kvalitního uživatelského prostředí jsou různé stavební prvky. Mezi základní patří:

- **Button** – ovládací tlačítko


```
Button['nazev_tlacitka'] (“nazev“, vlastnosti)
```

Příklad: `Button['TL1']("Cancel", Shutdown()`

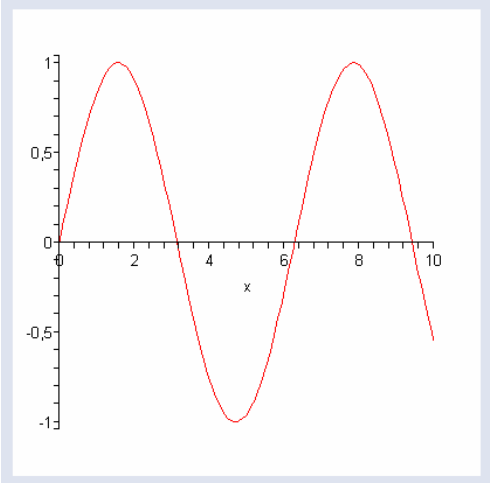
- **Label** – popis




```
Label(“text“, vlastnosti)
```

Příklad: `Label("Popis prvku: ", 'font' = Font("courier", 14))`

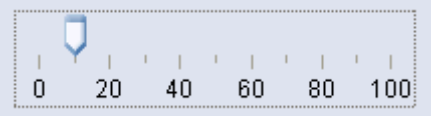
- **Plotter** – zobrazení 2D nebo 3D grafů




```
Plotter['nazev_grafu'](co se má zobrazit)
```

Příklad: `Plotter['PL1']( plot(sin(x), x = 0..10))`

- **Slider** – lidově nazýváno „posuvnítko“




```
Slider['nazev'](vlastnosti)
```

Příklad: `Slider['SL1'](0..100, 10, 'showticks', 'majorticks'=20, 'minorticks'=10, 'snapticks'=false)`

- **TextField** – jednoduchý řádek s textem (vhodné pro vstup i výstup)

`TextField['nazev'](vlastnosti)`

Příklad: `TextField['TF1']()`

Pokud potřebujeme využít delšího textu, je k dispozici prvek TextBox.

Prvky označujeme proto, abychom se mohli v průběhu práce na ně odvolávat, přenastavovat, nebo používat jejich hodnot a podobně. Nesmíme zapomenout na příkaz `Evaluate`, který je nezbytný pro spuštění externích příkazů.

Tímto jsem vyčerpala teoretický základ tvorby jednoduchých Mapletů. V následující části bude následovat několik ukázek.

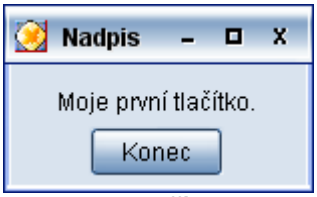
7.2.2 Příklady

Poznámka. Metodou pokus omyl můžete experimentovat a tím vytvářet jedinečné aplikace. Příklady jsou pouze motivační. Je to na Vaší fantazii co s nimi uděláte. Všechny jsou uvedeny v „7.2 Maplety/maplet.mws“.

Příklad 1. – První tlačítko

Cílem je nastavit titulek a vytvořit ukončovací tlačítko. Kód nebudu vysvětlovat.

```
> maplet3:=Maplet(Window("Nadpis",["Moje první tlačítko.",
  Button("Konec",Shutdown())])):
> Maplets[Display](maplet3);
```



Obr. 7.9 Příklad 1.

Příklad 2. – Derivace funkce

Hlavní úkolem je porozumět práci s objekty (TextField, Button a TextBox).

Vytvoříme dva TextFieldy, do kterých uživatel bude vepisovat funkci a proměnnou. Dále TextBox s výslednou derivací funkce a 3 tlačítka (*Derivovat* – zderivuje funkci podle proměnné, *Konec* – ukončí aplikaci a *Smaž* – vymaže vše).

```
> maplet4:=Maplet(Window("Derivace",[
  ["Funkce: ", TextField[TF1]()],
  ["Proměnná podle které budeme derivovat: ",TextField[TF2](3)],
  TextBox[TB1](editable=false, 3..40),
  [Button("Derivovat",Evaluate(TB1='diff(TF1,TF2)')),
  Button("Konec",Shutdown([TF1,TF2,TB1])),
  Button("Smaž",Action(SetOption(TF1= ""),
  SetOption(TF2= ""),SetOption(TB1= ""))]])):
> Maplets[Display](maplet4);
```

Vysvětlení kódu:

`TextField[TF1]()` – TextField pojmenovaný TF1

`TextField[TF2](3)` – TextField pojmenovaný TF2 o délce 3 znaků

`TextBox[TB1](editable=false, 3..40)` – TextBox se jménem TB1

`editable=false` – uživatel nemá právo do tohoto TextBoxu psát a měnit ho

`3..40` – 3 řádky a 40 sloupců

`Button("Derivovat",Evaluate(TB1='diff(TF1,TF2)')` – tlačítko Derivovat

`diff(TF1,TF2)` – zderivuje funkci (zastoupená TF1) podle proměnné (TF2)

`TB1='diff(TF1,TF2)'` – výsledek je uložen do TB1 (náš TextBox)

`Evaluate(TB1='diff(TF1,TF2)'))` – vypíše hodnotu

`Button("Konec",Shutdown([TF1,TF2,TB1]))` – tlačítko Konec

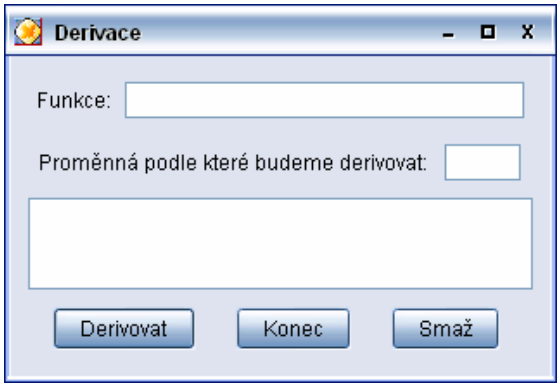
`Shutdown([TF1,TF2,TB1])` – ukončí a do pracovního listu vypíše hodnoty

`Button("Smaž",Action(SetOption(TF1=""),SetOption(TF2=""),`
`SetOption(TB1= ""))))` – tlačítko Smaž

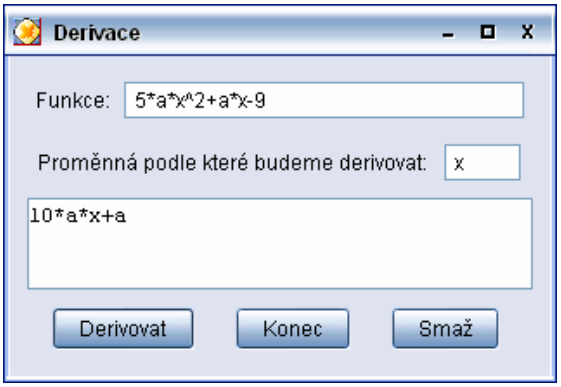
`SetOption(TF1="")` – nastaví hodnotu TF1 na prázdný znak neboli dojde k vymazání textu

`SetOption(TF2=""), SetOption(TB1= "")` – vymaže text v TF2

`Action(…)` – provede to, co je v závorce – změna nastavení (vymazání textu)



Obr. 7.10 Příklad 2. – po spuštění



Obr. 7.11 Příklad 2. – po provedení derivace

Příklad 3. – Graf funkce a derivace

V tomto příkladu se naučíme zobrazovat graf a měnit jeho hranice pomocí posuvnítka. Maplet bude dvoukrokový. Prvním krokem (pomocí tlačítka *Derivovat*) zderivujeme funkci a v druhém (do té doby nefunkčním tlačítkem *Vykresli*) zobrazíme graf funkce a její derivaci. Oproti minulému příkladu přibude Plotter, Slider a dvě tlačítka (*Vykresli*, *Nápověda*).

```
> maplet5 := Maplet( Window( "Derivace", [
 ["Funkce: ", TextField[TF1]()],
 ["Proměnná podle které budeme derivovat: ", TextField[TF2](3)],
 TextBox[TB1]( editable = false, 3..40 ),
 Plotter[PL1]( plot(undefined, x=-5..5) ),
 Slider[SL1]( 1..20, 5, Evaluate( PL1 = 'plot([TF1, TB1],
   TF2=-SL1..SL1)' ), 'showticks', 'majorticks'=5, 'minorticks'=1 ),
 [Button("Derivovat", Action(Evaluate(TB1 = 'diff(TF1, TF2)'),
   SetOption(B5=true, `option`=enabled))),
  Button("Konec", Shutdown([TF1, TF2, TB1])),
  Button("Smaž",
    Action(SetOption(TF1 = ""),
      SetOption(TF2 = ""),
      SetOption(TB1 = ""),
      SetOption(B5=false, `option`=enabled),
      Evaluate( PL1 = 'plot(undefined, x=-5..5)' ))),
  Button("Nápověda", RunDialog(MD1)),
  Button[B5]("Vykresli",
    Evaluate(PL1 = 'plot([TF1,TB1],TF2=-SL1..SL1)'),'not enabled')]]),
  MessageDialog[MD1]( "Maplet je dvoukrokový.
    V prvém kroku je spočtena derivace funkce.
    V druhém je vykreslen graf funkce a její derivace.
    Vhodné například pro analýzu monotónnosti.",
    'information' ) ):
> Maplets[Display](maplet5);
```

Vysvětlím jen ty složitější části kódu:

`Plotter[PL1]( plot(undefined, x=-5..5) )` – na počátku bude zobrazen prázdný graf v rozsahu -5 až 5

`Slider[SL1]( 1..20, 5, Evaluate( PL1 = 'plot([TF1, TB1], TF2=-SL1..SL1)' ), 'showticks', 'majorticks'=5, 'minorticks'=1 )` – posuvnítko ovlivňující rozsah zobrazení grafu

`1..20` – rozsah posuvnítka

`5` – implicitní pozice posuvnítka

`plot([TF1, TB1], TF2=-SL1..SL1)` – graf funkce (TF1) a derivace (TB1)

`TF2=-SL1..SL1` – graf bude zobrazen od opačné hodnoty posuvnítka do hodnoty posuvnítka

`Evaluate( PL1 = 'plot(…)')` – graf bude vykreslen v Plotteru (PL1)

Následující parametry určují vlastnosti posuvnítka (hlavně vzhledu):

`'showticks'` – zobrazí značky

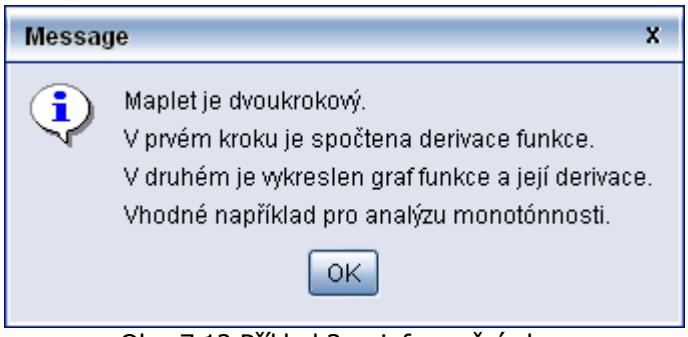
`'majorticks'=5` – každé páté číslo bude zobrazeno

`'minorticks'=1` – vzdálenost mezi značkami (neboli kam jde umístit „pačičku“ posuvnítka)

`Button("Nápověda", RunDialog(MD1))` – spustí okno (MD1) s nápovědou

`MessageDialog[MD1]( "Maplet je …",'information' )`

- první parametr udává text, který bude zobrazen v okně
- druhý říká o jaký typ se jedná, v našem případě o informační okno (obr. 7.12)



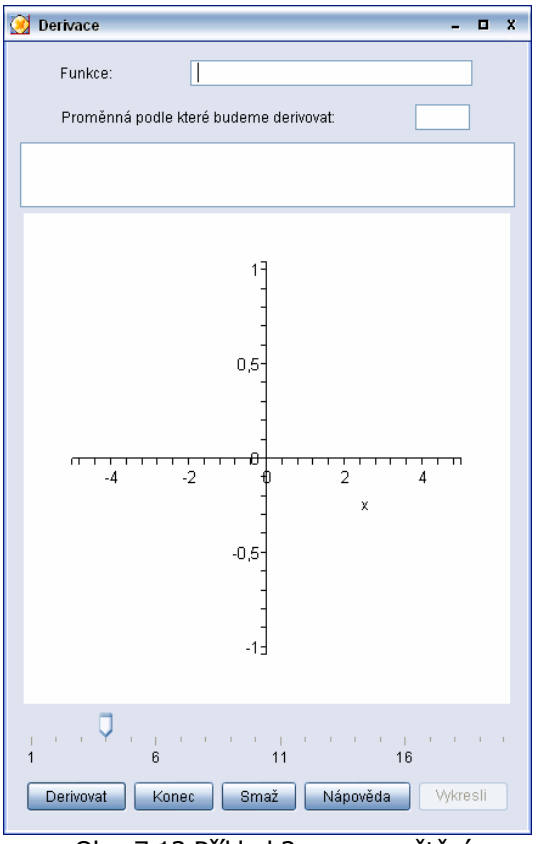
Obr. 7.12 Příklad 3. – informační okno

Na závěr vysvětlím způsob, jak probíhá vypnutí a zapnutí tlačítka *Vykresli*. Na obr. 7.13 je krásně vidět nefunkční tlačítko *Vykresli* a na následujícím obr. 7.14 je již funkční. Stačilo určit derivaci. Je to nezbytné, protože nelze zobrazovat graf derivace, pokud nebyla vypočítána.

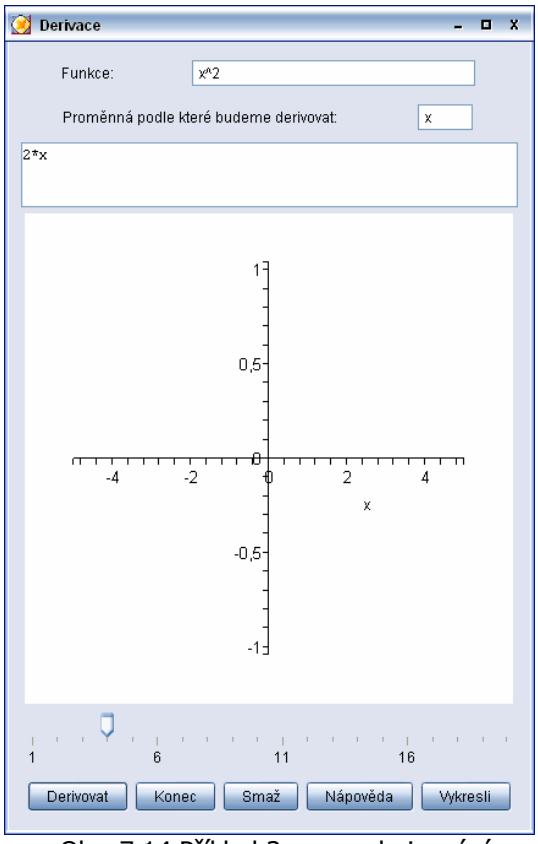
`Button[B5]( "Vykresli",…, 'not enabled')` – tlačítko jsem si označila jako B5 a na začátku nastavila jeho nefunkčnost (**`'not enabled'`**)

`Button("Derivovat", …, SetOption(B5=true, `option`=enabled))` – tlačítko *Derivovat* zapne tlačítko B5 (**`SetOption(B5=true,`option`=enabled)`**)

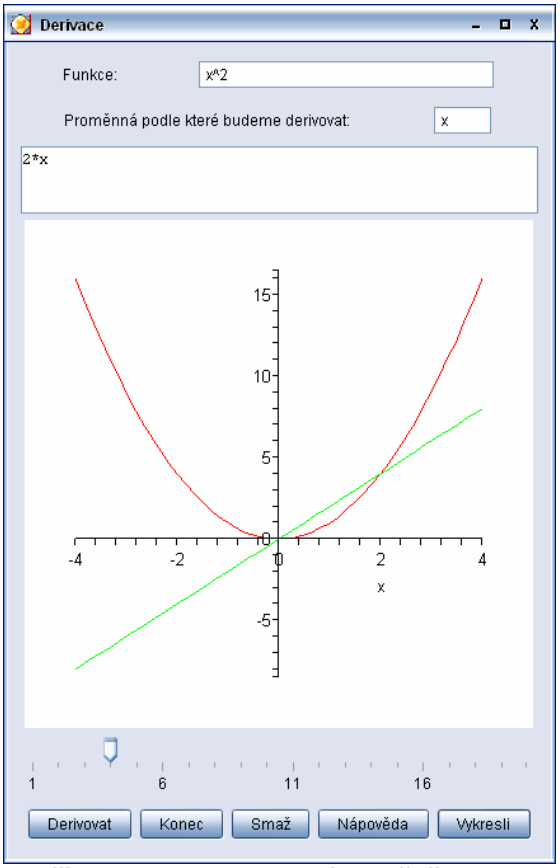
`Button("Smaž", …, SetOption(B5=false, `option`=enabled))` – po vymazání údajů je nezbytné zakázat tlačítko B5



Obr. 7.13 Příklad 3. – po spuštění



Obr. 7.14 Příklad 3. – po zderivování



Obr. 7.15 Příklad 3. – po vykreslení a změně hranic zobrazení

Předchozí tři příklady stačí k základnímu přehledu o tvorbě Mapletů. Tímto bych ukončila kapitolu a další práci nechala na fantazii čtenáře.

Závěr

Touto prací jsem se pokusila přiblížit některé nástroje programu Maple 9.5. Zároveň poukázat na to, že i když Maple 9.5 nedisponuje určitým nástrojem, lze tyto nedostatky napravit – naprogramovat si vlastní (procedury nebo Maplety). Proto jsem této problematice věnovala celou kapitolu.

Při psaní této práce jsem byla velmi potěšena, co všechno Maple 9.5 umí. Hlavně potenciálem, který Maplety nabízejí. Dále nově uvedeným studentským balíčkem `Student` (specielně `Student[Calculus1]`, který je dostupný od verze Maple 8), který v mnoha oblastech napravuje nedostatky předchozích verzí. Z mé oblasti je to například Newtonova metoda. Na druhou stranu mě překvapilo, že jsem nenašla určité funkce, které bych zde očekávala (například znázornění a použití metody tětiv).

Čtenář zde dostává ucelený přehled nástrojů. Jejich využití může být, jak doufám, při řešení úloh, ale i při výuce metod.

Seznam použité literatury

[1] **Adams, P., Smith, K., Výborný, R.**: *Introduction to MATHEMATICS WITH MAPLE*, World Scientific Publishing, 2004.

[2] **Bartsch, H. J.**: *Matematické vzorce*, Mladá Fronta, 1994.

[3] **Cornil, J. M., Testud, P**.: *An Introduction to Maple V*, Springer – Verlag Berlin Heidelberg, 2001.

[4] **Dont, M.**: *Numerické metody – cvičení*, ČVUT, 1990.

[5] **Hřebíček, J., Ráček, J., Pešl, J.**: *Úvod do Maplu 7*, Brno: Fakulta informatiky Masarykovy univerzity, 2002.

[6] **Hřebíček, J., Kohout, J.**: *Úvod do systému Maple*, Brno: Fakulta informatiky Masarykovy univerzity, 2004.

[7] **Char, B. W.**: *Maple 9 Learning Guide, Maplesoft*, Adivision of Waterloo Maple Inc., 2003.

[8] **Mrkvička, T.**: *Algebra V.*,http://www.pf.jcu.cz/stru/katedry/m/mrkvicka/, 2003.

[9] **Rektorys, K. a spolupracovníci**: *Přehled užité matematiky I.*, Praha: Prometheus, 2000.

[10] **Rektorys, K. a spolupracovníci**: *Přehled užité matematiky II.*, Praha: Prometheus, 2000.

[11] **Zörnig, P.**: *Numerické metody*, ČVUT, 1989.

Seznam použitých internetových zdrojů

[1] **Cifrikova matematika:** Seminární práce z algebry (13.11.2005)
URL: http://www.matematika.webz.cz/algebra/

[2] **Robert Mařík:** Mendel University of Agriculture and Forestry – Dpt. of Mathematics (13.11.2005)
URL: http://www.mendelu.cz/~marik/numerika.pdf

[3] **Ústav technické matematika ČVUT:** Základy algoritmizace a programování (13.11.2005)
URL: http://www.fsid.cvut.cz/cz/U201/ZAPG6.htm

[4] **Centrum aplikované matematika:** (28.02.2007)
URL: http://www.cam.zcu.cz/~danek/Students/2003_ZS/Materialy/nelinearni_rovnice.pdf

[5] **Úvod do numerických metod:** (28.02.2007)
URL: http://www.pef.zcu.cz/pef/kvd/cz/materialy/numet/_numet.html

[6] **ČVUT – Mirko Navara:** (12.04.2007)
URL: http://cmp.felk.cvut.cz/~navara/nm/koreny.pdf

[7] **WIKIPEDIE – Otevřená encyklopedie:** (12.04.2007)
URL: http://cs.wikipedia.org/wiki/Spojitost

Seznam příloh

PŘÍLOHY

PŘÍLOHA A

Přehled nejdůležitějších příkazů pro práci s polynomy

Přehled nejdůležitějších příkazů

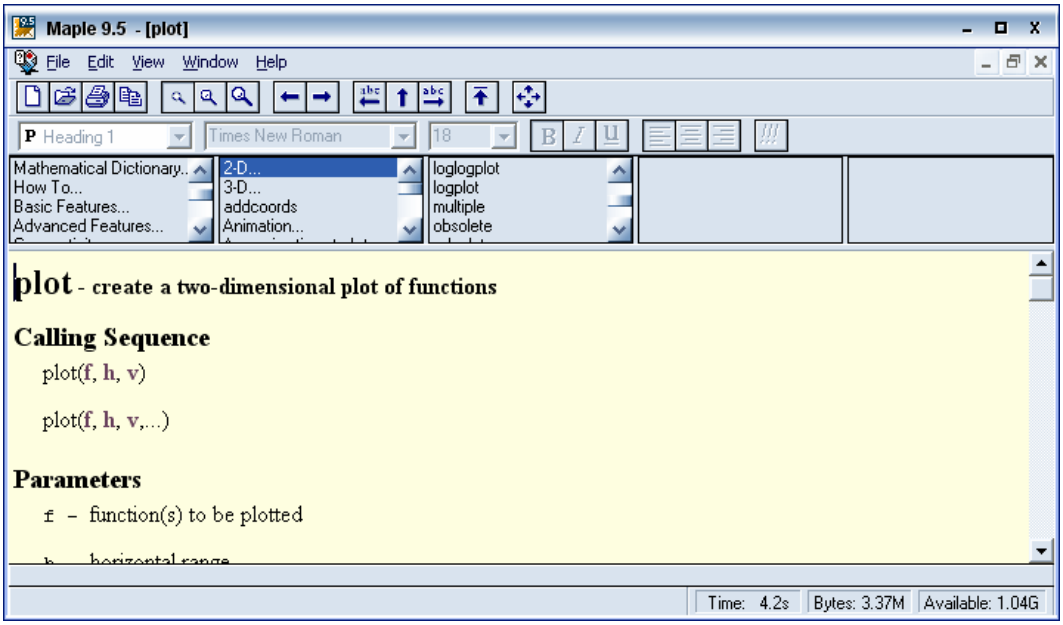
pro práci s polynomy

?	expand	minimize	sort
#	factor	plot	sqrt
^	infinity	RealRange	subs
diff	is	restart	with
evalb	isolate	simplify	
evalf	maximize	solve	

?

volání nápovědy

```
> ?plots
```



#

znak uvozující poznámku (text, který Maple 9.5 jakoby nevidí)

```
> # poznámka
```

^

mocnina, klávesová zkratka CTRL + 94

```
> 2^7;
```
128

diff

derivace výrazu

1. parametr - funkce

2. parametr – proměnná, podle které budeme derivovat

```
> diff(x^5+8*x^3+6,x);
```
$5x^4 + 24x^2$

derivace vyšších řádů

```
> diff(x^5+8*x^3+6,x$3);
```
$60x^2 + 48$

evalb

informuje nás o pravdivosti výrazu v závorkách

```
> evalb(-1*(-1)>0);
```
true

```
> evalb(-1*(-1)<0);
```
false

evalf

vyčíslení výrazu

```
> evalf(sqrt(4)+9.2);
```
11.2

```
> evalf(5487/599);
```
9.160267112

nastavení počet desetinných míst

```
> evalf(5487/6599,5);
```
0.83149

expand

rozvine výraz do řady součtů jeho podvýrazů

```
> expand((x+1)*(x+2));
```
$x^2 + 3x + 2$

```
> expand(sin(x+y));
```
$$\sin(x)\cos(y)+\cos(x)\sin(y)$$

factor

rozklad polynomů na kořenové činitele

```
> factor(6*x^2+18*x-24);
```
$$6(x+4)(x-1)$$

infinity

nekonečno

is

mnoho možností využití. Pro nás je důležité, že tato funkce dokáže posoudit, zda bod leží v zadaném intervalu

1. parametr – bod
2. parametr – interval (zadaný pomocí `RealRange`)

```
> is(2,RealRange(0,2));
```
true
```
> is(2.1,RealRange(0,2));
```
false

isolate

vyjádření proměnné z rovnice. Nevýhoda je, že dostaneme pouze první kořen.

1. parametr – rovnice
2. parametr – proměnná, kterou chceme z rovnice vypočítat

```
> isolate((x-1)^2-1=0,x);
```
$$x=2$$
```
> isolate((x-a)^2-1=0,x);
```
$$x=1+a$$

maximize

nalezne hodnotu maxima funkce (možnost určení omezujícího intervalu)

1. parametr – rovnice
2. parametr – proměnná

```
> maximize(-x*(x-1)+1,x);
```
$$\frac{5}{4}$$

určíme interval na kterém budeme hledat maximum

```
> maximize(-x*(x-1)+1,x=0.5..1);
                          1.25
```

přidáním parametru location, dostaneme informaci o x-ové souřadnici maxima

```
> maximize(-x*(x-1)+1,x=0.5..1,location);
                  1.25, {[{x = 0.5}, 1.25]}
```

minimize

nalezne hodnotu minima funkce (možnost určení omezujícího intervalu)

1. parametr – rovnice
2. parametr – proměnná

```
> minimize(x*(x-1)+1,x);
                          3/4
```

určíme interval, na kterém budeme hledat minimum

```
> minimize(x*(x-1)+1,x=0.5..1);
                          0.75
```

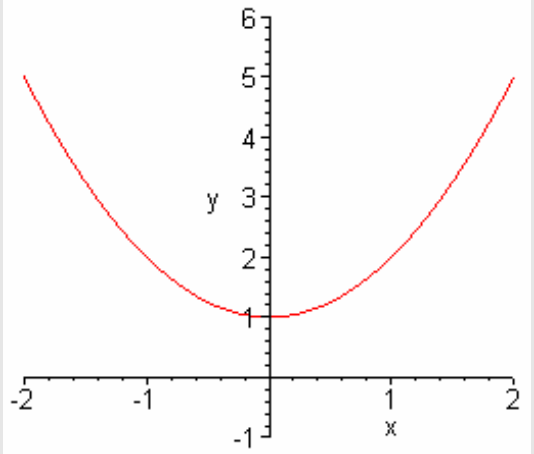
přidáním parametru location, dostaneme informaci o x-ové souřadnici minima

```
> minimize(x*(x-1)+1,x=0.5..1,location);
                  0.75, {[{x = 0.5}, 0.75]}
```

plot

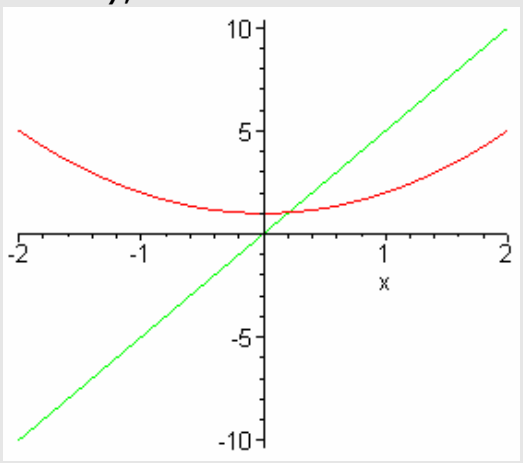
vykreslí graf

```
> plot(x^2+1,x=-2..2,y=-1..6);
```



popř. více grafů do „jednoho obrázku“

```
> plot({x^2+1,5*x},x=-2..2);
```



RealRange

označení pro reálný interval *(a; b)*

restart

vymaže vnitřní paměť. Od této doby si počítač nepamatuje žádnou proměnnou, která byla předtím použita.

simplify

zjednoduší výraz

```
> simplify(((a^4-b^4)/(a^2*b^2))/((1+b^2/a^2)*(1-2*a/b+a^2/(b^2))));
```

$$\frac{a+b}{-b+a}$$

solve

vyřeší rovnici

1. parametr – rovnice

2. parametr určuje, podle které proměnné se bude rovnice řešit

```
> solve((x-1)^2-1=0,x);
```

$$0, 2$$

```
> solve((x-a)^2-1=0,x);
```

$$1+a, a-1$$

soustavu rovnic

```
> solve({a+b=2,a-b=4});
```

$\{b = -1, a = 3\}$

i nerovnice

```
> solve(x^4-5*x^2+4<=0,x);
```

RealRange(-2, -1), RealRange(1, 2)

sort

upraví pořadí členů polynomu

```
> sort(x^15+x^5+x^17+x);
```

$x^{17} + x^{15} + x^5 + x$

sqrt

odmocnina

```
> sqrt(121);
```

11

subs

dosazuje zvolenou hodnotu za proměnnou do rovnice. Vhodné je to například pro zjišťování funkčních hodnot.

Popis parametrů:

1. parametr – je ve tvaru proměnná = hodnota
2. parametr – výraz, do kterého chceme dosadit

```
> subs(x=3,x^2+3);
```

12

with

načte balíček (knihovnu) funkcí, tyto funkce můžeme později využívat

```
> with(plots);
```

[*animate*, *animate3d*, *animatecurve*, *arrow*, *changecoords*, *complexplot*, *complexplot3d*, *conformal*, *conformal3d*, *contourplot*, *contourplot3d*, *coordplot*, *coordplot3d*, *cylinderplot*, *densityplot*, *display*, *display3d*, *fieldplot*, *fieldplot3d*, *gradplot*, *gradplot3d*, *graphplot3d*, *implicitplot*, *implicitplot3d*, *inequal*, *interactive*, *interactiveparams*, *listcontplot*, *listcontplot3d*, *listdensityplot*, *listplot*, *listplot3d*, *loglogplot*, *logplot*, *matrixplot*, *multiple*, *odeplot*, *pareto*, *plotcompare*, *pointplot*, *pointplot3d*, *polarplot*, *polygonplot*, *polygonplot3d*, *polyhedra_supported*, *polyhedraplot*, *replot*, *rootlocus*, *semilogplot*, *setoptions*, *setoptions3d*, *spacecurve*, *sparsematrixplot*, *sphereplot*, *surfdata*, *textplot*, *textplot3d*, *tubeplot*]

Poznámka. Ve své práci jsem využívala balíčku `Student`, především `Student[Calculus1]`, který je dostupný od verze Maple 8. Byl navržen k pochopení základních poznatků a zjednodušení výuky. Výpis všech funkcí, které jsou ve verzi Maple 9.5 je následující:

```
> with(Student[Calculus1]);
```

[AntiderivativePlot, AntiderivativeTutor, ApproximateInt, ApproximateIntTutor, ArcLength, ArcLengthTutor, Asymptotes, Clear, CriticalPoints, CurveAnalysisTutor, DerivativePlot, DerivativeTutor, DiffTutor, ExtremePoints, FunctionAverage, FunctionAverageTutor, FunctionChart, FunctionPlot, GetMessage, GetNumProblems, GetProblem, Hint, InflectionPoints, IntTutor, Integrand, InversePlot, InverseTutor, LimitTutor, MeanValueTheorem, MeanValueTheoremTutor, NewtonQuotient, NewtonsMethod, NewtonsMethodTutor, PointInterpolation, RiemannSum, RollesTheorem, Roots, Rule, Show, ShowIncomplete, ShowSteps, Summand, SurfaceOfRevolution, SurfaceOfRevolutionTutor, Tangent, TangentSecantTutor, TangentTutor, TaylorApproximation, TaylorApproximationTutor, Understand, Undo, VolumeOfRevolution, VolumeOfRevolutionTutor, WhatProblem]

PŘÍLOHA B

Společné procedury pro všechny metody

Společné procedury pro všechny metody

Poznámka. Procedury nejsou řazeny abecedně a to z důvodu, že záleží na pořadí ve kterém jsou použity. Protože některé na sebe navazují. To znamená, že v proceduře využíváme již zadefinované funkce.

funkcni_hodnota	prusecik	barvicky1
vypis	der	barvicky
nula	rovtec	IsKoren

Knihovny potřebné pro tyto procedury:

```
> with(Maplets[Elements]):
> with(plots):
> with(StringTools):
```

funkcni_hodnota

vrací funkční hodnotu funkce v daném bodě

1. parametr – funkce

2. parametr – bod

3. parametr – proměnná

```
> funkcni_hodnota := proc(f,bod,prom)
  return subs(prom=bod,f);
> end proc:
```

vypis

spustí okno Mapletu se zadaným textem

1. parametr – text

```
> vypis := proc(co)
> local slovo,maplet1;
> slovo:=co;
> maplet1 := Maplet( Window("Info",
 [co,
    Button("Vykresli",Shutdown(print(co)))
```

```
  ])):
> Maplets[Display](maplet1);
> end proc:
```

nula

desetinná čísla, které začínají nulou, Maplety vypisují až od desetinné tečky (nezobrazují nulu). Tato procedura ji tam přidává, když chybí.

1. parametr – číslo

```
> nula := proc(cislo)
if(cislo<1 and cislo>0) then
  return cat("0",convert(evalf(cislo),string));
else
  return convert(evalf(cislo),string);
end if;
end proc:
```

prusecik

vrací x-ovou souřadnici průsečíku přímky s osou *x*

1. parametr – přímka
2. parametr – proměnná

```
> prusecik := proc(primka,prom)
  fsolve(primka=0,prom);
> end proc:
```

der

vrací hodnotu první derivace funkce v daném bodě

1. parametr – funkce
2. parametr – bod
3. parametr – proměnná

```
> der := proc(f,bod,prom)
  local deriv;
  deriv:=diff(f,prom);
  return subs(prom=bod, deriv);
end proc:
```

rovtec

vrací rovnici tečny k funkci v daném bodě

1. parametr – funkce

2. parametr – bod (x-ová souřadnice)

3. parametr – proměnná

```
> rovtec := proc(f,bod,prom)
  return der(f,bod,prom)*(prom-bod)+subs(prom=bod,f);
end proc:
```

barvicky1

vrací pole barev (sudé barvy jsou do modrého zabarvení a liché jsou do žluta)

1. parametr – délka pole

```
> barvicky1 := proc(kolik)
# vypise pole barev
local u, barvy,pom,barvy1,barvy2,v;
pom:=1/(kolik+2);
for u from 0 by 1 to kolik do
   barvy1[u]:=COLOR(RGB, (kolik-u+1)*pom/10, (kolik-u+1)*pom,1);
   barvy2[u]:=COLOR(RGB, 1, (kolik-u+1)*pom, (kolik-u+1)*pom/10);
end do;
for v from 0 by 1 to kolik do
  if(v mod 2 = 0) then
    barvy[v]:=barvy1[v/2];
  else
    barvy[v]:=barvy2[(v-1)/2];
  end if;
end do;
return barvy;
end proc:
```

barvicky

vrací pole barev, kde se periodicky opakují

1. parametr – délka pole

2. parametr – perioda, po jakém počtu se budou barvy opakovat

```
> barvicky := proc(kolik,pocet)
# vypise pole barev, kde se periodicky opakuji v zavislosti na pocet
local u, barvy,pom;
pom:=barvicky1(pocet);
for u from 0 by 1 to kolik do
  if((u mod (2*pocet - 2))<pocet) then
    barvy[u]:=pom[u mod (2*pocet - 2)];
  else
    barvy[u]:=pom[pocet - 1 -(u mod (pocet - 1))];
  end if;
end do;
return barvy;
end proc:
```

IsKoren

vrací pravdivostní hodnotu, zda bod je kořenem funkce (true – ano, false – ne)

1. parametr – funkce
2. parametr – bod
3. parametr – proměnná

```
> IsKoren := proc(f,x0,prom)
# true - je korenem
# false - neni korenem
local zda, d1, d2;
zda:=simplify(subs(prom=x0,f));
d1:=eval(zda);
d2:=evalf(zda,50);
if(d1=0 or d2=0) then
  return true;
end if;
return false;
end proc:
```

PŘÍLOHA C

Bisekce

C.1 Společné procedury pro bisekci

Poznámka. Budeme využívat procedury z přílohy B.

predpoklady	reseni	bisekce_krok
korenShow	delkaIntervalu	bisekce_vypis
IntervalShow	bisekce	

Knihovny potřebné pro tyto procedury:

```
> with(Maplets[Elements]):
> with(plots):
> with(StringTools):
```

predpoklady

vrací pravdivostní hodnotu, zda jsou splněny předpoklady metody

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

```
> predpoklady :=proc(f,a,b,prom)
local k;
if(evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))<0) then
  k:=1;
elif(evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))=0)
then
  k:=0;
else
  k:=-1;
end if;
return k;
end proc:
```

korenShow

pomocná metoda pro vykreslení kořene

1. parametr – funkce

2., 3. parametr – interval

4. parametr – jak vysoko bude slovo kořen

5. parametr – proměnná

```
> korenShow := proc(f,a,b,bod,vyska,prom)
    cast[0] - vykresluje bod (kořen)
    cast[1] - vykresluje slovo kořen
> local cast;
cast[0]:=plot([[bod,0]], prom=a..b,
  style=point,symbol=box,color=magenta,symbolsize=12);
cast[1]:=textplot([bod,vyska,"Koren"],color=magenta);
return cast;
end proc:
```

IntervalShow

pomocná metoda pro zobrazení intervalu

1. parametr – funkce

2., 3. parametr – interval

4., 5. parametr – rozsah zobrazení na ose *y*

6. parametr – proměnná

7. parametr – barva, kterou bude interval vybarven

```
> intervalShow := proc(f,a,b,y1,y2,prom,vybarveni)
> return inequal( {prom>=a, prom<=b}, prom=a..b, y=y1..y2 ,
    optionsfeasible=(color=vybarveni),
    optionsclosed=(color=red, thickness=1));
> end proc:
```

reseni

rozepisuje pole hodnot ve stanoveném formátu

1. parametr – pole hodnot (formát [a1,a2,b1,b2,…], a1 je začátek prvního intervalu, a2 konec, b1, b2 označuje další interval, atd.)

2. parametr – počet dvojic

3. parametr – text (pomocná proměnná)

```
> reseni := proc(pole,num,zdaKoren)
local h,delka,text1,text2;
if(not(zdaKoren="")) then
  print("---------------------------------------");
  print(zdaKoren);
  print("---------------------------------------");
end if;

for h from 0 to 2*num by 2 do
  delka:=pole[h+1] - pole[h];
```

```
  text1:=cat(convert(h/2,string),
    ". krokem jsme dostali interval: < ",nula(pole[h])," ; ",
    nula(pole[h+1])," >");
  text2:=cat("Jeho velikost je: ",convert(nula(delka),string));
  print(text1);
  print(text2);
  print("---------------------------------------");
end do;
end proc:
```

delkaIntervalu

vrací délku intervalu získaného metodou bisekce v závislosti na 9. parametru

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

5., 6. parametr – rozsah zobrazení na ose *x*

7., 8. parametr – rozsah zobrazení na ose *y*

9. parametr – kolikátý interval nás zajímá

```
> delkaIntervalu := proc(f,a,b,prom,x1,x2,y1,y2,kolik)
> local novy,pl,u,chyba,a1,b1,intervaly;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  a1:=a;
  b1:=b;
  intervaly[0]:=a;
  intervaly[1]:=b;
  pl:=0;
  if(predpoklady(f,a,b,prom)=1) then
    for u from 0 by 1 to kolik do
      if(u>=1) then
        novy[u]:=(a1+b1)/2;
        if(funkcni_hodnota(f,novy[u],prom)=0) then
          return "koren";
          #break;
        else
          if(predpoklady(f,a1,novy[u],prom)=1) then
            b1:=novy[u];
          elif(predpoklady(f,novy[u],b1,prom)=1) then
            a1:=novy[u];
          end if;
        end if;
      end if;
      intervaly[2*u]:=a1;
      intervaly[2*u+1]:=b1;
      pl:=kolik;
    end do;
    #if(not(pl=0)) then
      return abs(intervaly[2*pl]-intervaly[2*pl+1]);
    #end if;
  elif(predpoklady(f,a,b,prom)=0) then
    return "koren";
```

```
  end if;
end if;
end proc:
```

bisekce

grafické znázornění – vykresluje graf znázorňujíci metodu půlení intervalu

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

5., 6. parametr – rozsah zobrazení na ose *x*

7., 8. parametr – rozsah zobrazení na ose *y*

9. parametr – kolikrát bude bisekce použita

```
> bisekce := proc(f,a,b,prom,x1,x2,y1,y2,kolik)
> local a1,b1,h,novy,u,barva,s,text,showSlovoKoren,showKoren,
  showSlovoFunkce,cele,barvy,interval,pl,krajni_body,pom,pom1,
  cojespolecne,pomocx,pomocy,vyska;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  barvy:=barvicky(kolik,6);
  showKoren:="";
  showSlovoKoren:=textplot([2,2,""]);
  pomocx:=(x2-x1)/60;
  pomocy:=(y2-y1)/100;
  vyska:=pomocy*4;
  a1:=a;
  b1:=b;
  pl:=0;
  barva:=color=[blue,red];
  interval[0]:=intervalShow(f,a,b,y1,y2,prom,barvy[0]);
  s:=plot(f,prom=x1..x2,y=y1..y2,barva,axes=normal);
```

Jeden z krajních bodů je kořen:

```
> if(predpoklady(f,a,b,prom)=0) then
  if(funkcni_hodnota(f,a,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string)," = ",
      convert(nula(a),string));
    cele:=korenShow(f,a,b,a,-1*vyska,prom);
  elif(funkcni_hodnota(f,b,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string)," = ",
      convert(nula(b),string));
    cele:=korenShow(f,a,b,b,-1*vyska,prom);
  end if;
  showKoren:=cele[0];
  showSlovoKoren:=cele[1];
  vypis(text);
  s:=plot(f,prom=a..b,y=y1..y2,barva,axes=normal);
```

Nejsou splněny předpoklady:

```
elif(predpoklady(f,a,b,prom)=-1) then
  vypis("Nejsou splněny předpoklady!");
```

Splněny předpoklady + není ani jeden z krajních bodů kořenem:

```
elif(predpoklady(f,a,b,prom)=1) then
  for u from 0 by 1 to kolik do
    barva:=color=[blue,red$u+2];
    if(u>=1) then
      novy[u]:=(a1+b1)/2;
      if(funkcni_hodnota(f,novy[u],prom)=0) then
        text:=cat("Kořen byl nalezen: ",convert(prom,string)," = ",
          convert(nula(novy[u]),string)," na začátku ",
          convert(u,string),". kroku.");
        vypis(text);
        showKoren:=plot([[novy[u],0]], prom=x1..x2, style=point,
          symbol=box,color=magenta,symbolsize=12);
        showSlovoKoren:=textplot([novy[u],-1*vyska,"Koren"],
          color=magenta);
        pl:=u-1;
        break;
      else
        if(predpoklady(f,a1,novy[u],prom)=1) then
          b1:=novy[u];
        elif(predpoklady(f,novy[u],b1,prom)=1) then
          a1:=novy[u];
        end if;
      end if;
    end if;
    interval[u]:=intervalShow(f,a1,b1,y1+pomocy*1.5*u,y2-pomocy*1.5*u,
      prom,barvy[u]);
    pl:=kolik;
  end do;
end if;

showSlovoFunkce:=textplot([x2-pomocx,funkcni_hodnota(f,x2,prom),
  "funkce"],color=blue);
krajni_body:=textplot([[a+pomocx,vyska,"a"],[b-pomocx,vyska,"b"]],
  color=red);
cojespolecne:=s,showSlovoFunkce,krajni_body;

pom1:=0;
for h from pl by -1 to 0 do
  pom[pom1]:=interval[h];
  pom1:=pom1+1;
end do;

if(showKoren="") then
  if(predpoklady(f,a,b,prom)=-1) then
    display({cojespolecne,interval[0]});
  else
    display({cojespolecne,
      textplot([[novy[w+1]+pomocx,vyska,x[w]]$ w=0..pl-1],color=red)},
      pom[z] $ z=0..kolik);
  end if;
else
  if(predpoklady(f,a,b,prom)=0) then
    display({cojespolecne,showSlovoKoren, showKoren},
      pom[0],view=[a..b,y1..y2]);
  else
    display({cojespolecne,showSlovoKoren, showKoren,
```

```
      textplot([[novy[w+1]+pomocx,vyska,x[w]]$ w=0..pl],color=red)},
      pom[z] $ z=0..pl);
 end if;
end if;
else
  vypis("Špatně zadané intervaly, hranice nebo počet kroků!");
  if(x1>=x2) then
    print(cat("Neplatí podmínka x1<x2  (",convert(nula(x1),string),
      " < ",convert(nula(x2),string),")"));
  end if;
  if(y1>=y2) then
     print(cat("Neplatí podmínka y1<y2  (",convert(nula(y1),string),
       " < ",convert(nula(y2),string),")"));
  end if;
  if(a>=b) then
     print(cat("Neplatí podmínka a<b  (",convert(nula(a),string),
       " < ",convert(nula(b),string),")"));
  end if;
  if(kolik<0) then
     print("Počet kroků musí být celé nezáporné číslo.");
  end if;
  if(not(is(kolik,integer))) then
    print("Počet kroků musí být celé nezáporné číslo.");
  end if;
end if;
end proc:
```

bisekce_krok

procedura pro vizualizaci metody – krok za krokem (nejsou zde ověřovány předpoklady, proto tato funkce lze použít jen na data uvedená v Mapletu)

1. parametr – funkce
2., 3. parametr – interval
4. parametr – proměnná
5., 6. parametr – rozsah zobrazení na ose *y*
7. parametr – kolikrát bude bisekce použita

```
> bisekce_krok := proc(f,a,b,prom,y1,y2,kolik)
> local a1,b1,novy,u,s,showSlovoFunkce,barvy,interval,pl,krajni_body,
  pom,pom1,barva,h;
barvy:=barvicky(kolik,6);
a1:=a;
b1:=b;
pl:=0;

barva:=color=[blue,red];
interval[0]:=intervalShow(f,a,b,y1,y2,prom,barvy[0]);
  for u from 0 by 1 to kolik do
    barva:=color=[blue,red$u+2];
    if(u>=1) then
      novy[u]:=(a1+b1)/2;
      if(predpoklady(f,a1,novy[u],prom)=1) then
```

```
        b1:=novy[u];
      elif(predpoklady(f,novy[u],b1,prom)=1) then
        a1:=novy[u];
      end if;
    end if;
    interval[u]:=intervalShow(f,a1,b1,y1+0.2*u,y2-0.2*u,prom,
      barvy[u]);
    pl:=kolik;
  end do;

s:=plot(f,prom=a-
0.05..b+0.05,y=y1..y2,barva,axes=normal,xtickmarks=0,ytickmarks=0);
showSlovoFunkce:=textplot([b+0.04,funkcni_hodnota(f,b,prom),
  "funkce"],color=blue);
krajni_body:=textplot([[a-0.015,-0.2,"a"],[b+0.015,0.2,"b"]],
  color=red);

pom1:=0;
for h from pl by -1 to 0 do
  pom[pom1]:=interval[h];
  pom1:=pom1+1;
end do;
display({s,showSlovoFunkce,krajni_body,textplot([[novy[w+1]+0.015,
  -0.2,x[w]]$ w=0..pl-1],color=red)},pom[z] $ z=0..kolik);
end proc:
```

bisekce_vypis

vypisuje řešení, získané za pomoci bisekce

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

5. parametr – počet kroků

```
> bisekce_vypis := proc(f,a,b,prom,kolik)
> local a1,b1,h,novy,u,text,intervaly,pl;
if(a<b and kolik>=0 and is(kolik,integer)) then

text:="";
a1:=a;
b1:=b;
pl:=0;
intervaly[0]:=a;
intervaly[1]:=b;
```

Jeden z krajních bodů je kořen:

```
> if(predpoklady(f,a,b,prom)=0) then
  if(funkcni_hodnota(f,a,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(a),string));
  elif(funkcni_hodnota(f,b,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(b),string));
```

```
  end if;
```

Nejsou splněny předpoklady:

```
elif(predpoklady(f,a,b,prom)=-1) then
  text:="Nejsou splněny předpoklady!";
```

Splněny předpoklady + není ani jeden z krajních bodů kořenem:

```
elif(predpoklady(f,a,b,prom)=1) then
  for u from 0 by 1 to kolik do
    if(u>=1) then
      novy[u]:=(a1+b1)/2;
      if(funkcni_hodnota(f,novy[u],prom)=0) then
        text:=cat("Kořen byl nalezen: ",convert(prom,string)," =",
          convert(nula(novy[u]),string)," na začátku",
          convert(u,string),". kroku.");
        pl:=u-1;
        break;
      else
        if(predpoklady(f,a1,novy[u],prom)=1) then
          b1:=novy[u];
        elif(predpoklady(f,novy[u],b1,prom)=1) then
          a1:=novy[u];
        end if;
      end if;
    end if;
    intervaly[2*u]:=a1;
    intervaly[2*u+1]:=b1;
    pl:=kolik;
  end do;
end if;
reseni(intervaly,pl,text);
else
  vypis("Špatně zadaný interval nebo počet kroků.");
  if(a>=b) then
    print(cat("Neplatí podmínka a<b  (",convert(nula(a),string),
      " < ",convert(nula(b),string),")"));
  end if;
  if(kolik<0) then
     print("Počet kroků musí být celé nezáporné číslo.");
  end if;
  if(not(is(kolik,integer))) then
    print("Počet kroků musí být celé nezáporné číslo.");
  end if;
end if;
end proc:
```

C.2 Maplet “Bisekce – krok za krokem”

Poznámka. Budeme využívat procedury z přílohy B a C.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_vizualizace := Maplet( Window( "Bisekce - krok za krokem",
> [
  Plotter[PL1](bisekce_krok(4*x^3+11*x^2+5*x-2,-0.25,0.4,
  x,-3,3,0),height=300,width=700 ),
  [
    BoxRow(caption="Počet kroků", border=true,
      Slider[SL1]( 0..4, 0,
        Evaluate(PL1='bisekce_krok(4*x^3+11*x^2+5*x-2,-0.25,0.4,
        x,-3,3,SL1)'),
        'majorticks'=1,showticks=true,'minorticks'=1, filled=true)
    ),
    Button("Obnovit",Action(
      Evaluate(PL1='bisekce_krok(4*x^3+11*x^2+5*x-2,-0.25,0.4,
      x,-3,3,0)'),
      SetOption(SL1=0))),
    Button("Nápověda",RunDialog(MD1)),
    Button("Konec",Shutdown())
  ]
]),
> MessageDialog[MD1]( "Tento Maplet slouží pro vizualizaci metody
půlení intervalu (bisekce).\nKaždý nově vzniklý interval je značen
tmavší barvou a je o něco nižší než ten předchozí.\nNultý krok
znázorňuje původní interval.",'information')):
> Maplets[Display]( maplet_vizualizace );
```

C.3 Maplet “Bisekce – grafické znázornění”

Poznámka. Budeme využívat procedury z přílohy B a C.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_graficke := Maplet( Window( "Bisekce - grafické znázornění",
[
  BoxRow(
    [BoxRow(["Funkce: ", TextField[Fce]("4*x^3+11*x^2+5*x-2")]),
     BoxRow("Proměnná: ", TextField[Prom]("x",3),"Počet kroků: ",
       TextField[Kroky]("2",3))
    ],
    BoxRow(caption="Interval",border=true,
      ["Začátek: ","Konec: "],
      [TextField[Inta]("0",3),TextField[Intb]("1",3)]
    ),
    BoxRow(caption="Rozsah zobrazení",border=true,
      ["X-min: ","X-max: "],
      [TextField[Xmin]("-0.1",3),TextField[Xmax]("1.1",3)],
      ["Y-min: ","Y-max: "],
      [TextField[Ymin]("-3",3),TextField[Ymax]("5",3)]
    )
  ),
  [Plotter[PL3]( plot(undefined, x=-100..100),width=800,height=500)],
  [
    "Délka posledního intervalu:",
    TextField[Delka]("",11,editable=false),
    Button[B2]("Použít metodu půlení intervalu",
      Action(
        Evaluate(PL3='bisekce(Fce,Inta,Intb,Prom,Xmin,Xmax,
          Ymin,Ymax,Kroky)'),
        Evaluate(Delka='delkaIntervalu(Fce,Inta,Intb,Prom,Xmin,Xmax,
          Ymin,Ymax,Kroky)')) ),
        Button("Obnovit",Action(
          SetOption(Fce="4*x^3+11*x^2+5*x-2"),
          SetOption(Prom="x"),
          SetOption(Kroky="0"),
          SetOption(Inta="0"),
          SetOption(Intb="1"),
          SetOption(Delka=""),
          Evaluate(PL3='plot(undefined, x=-100..100)'))),
        Button("Nápověda",RunDialog(MD1)),
        Button("Konec",Shutdown())
  ]
```

```
]),
> MessageDialog[MD1]( "\nOPTIMALIZOVÁNO PRO POLYNOMIÁLNÍ
FUNKCE!\n\nTento Maplet slouží pro vizualizaci metody půlení intervalu
(bisekce).\nKaždý nově vzniklý interval je značen tmavší barvou a je o
něco nižší než ten předchozí.\nNultý krok znázorňuje původní
interval.",'information')):
> Maplets[Display]( maplet_graficke );
```

PŘÍLOHA D

Newtonova metoda

D.1 Společné procedury pro Newtonovu metodu

Poznámka. Budeme využívat procedury z přílohy B.

pocet	predpoklady5	vypocetChyby
predpoklady1	predpoklady	odhadChyby
predpoklady2	IsRovn	newton
predpoklady3	tecnaShow	newton_krok
predpoklady4	korenShow	graftecna

Knihovny potřebné pro tyto procedury:

```
> with(Maplets[Elements]):
> with(plots):
> with(StringTools):
> with(student,showtangent):
```

pocet

vrací počet prvků v poli ukončené výjimkou (invalid subscript selector)

1. parametr – pole

```
> pocet :=proc(tr)
local j, result;
try
  for j from 1 by 1 to infinity do
    tr[j];
  end do;
catch "invalid subscript selector":
  result := j-1
end try;
return result;
> end proc:
```

predpoklady1

vrací pravdivostní hodnotu, zda je splněn první předpoklad $f(a)f(b) < 0$

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

```
> predpoklady1 :=proc(f,x0,a,b,prom)
# f(a)*f(b)<0
# true = je to v poradku
# 0 = jeden z krajnich bodu je korenem
# false = neplati podminka
> local k;
if(evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))<0) then
  k:=true;
elif (evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))=0)
then
  k:=0;
else
  k:=false;
end if;
return k;
> end proc:
```

predpoklady2

vrací pravdivostní hodnotu, jestli je splněn druhý předpoklad (zda není první derivace na intervalu rovna nule)

1. parametr – funkce

2. parametr – počáteční bod

3. parametr – začátek intervalu

4. parametr – konec intervalu

5. parametr – proměnná

```
> predpoklady2 :=proc(f,x0,a,b,prom)
# ověřuje zda na intervalu a,b není prvni derivace rovna nule
local der, reseni,j;
der:=diff(f,prom);
reseni:=[solve(der=0)];
for j from 1 by 1 to pocet(reseni) do
  if(is(reseni[j],RealRange(a,b))) then
    return false;
  end if;
end do;
return true;
> end proc:
```

predpoklady3

vrací pravdivostní hodnotu, jestli je splněn třetí předpoklad (ověřuje zda na intervalu není inflexe)

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

```
> predpoklady3 :=proc(f,x0,a,b,prom)
# ověřuje zda na intervalu a,b není inflexe
local der, reseni,j;
der:=diff(f,prom$2);
reseni:=[solve(der=0)];
for j from 1 by 1 to pocet(reseni) do
  if(is(reseni[j],RealRange(a,b))) then
    return false;
  end if;
end do;
return true;
> end proc:
```

predpoklady4

vrací pravdivostní hodnotu, jestli je splněn čtvrtý předpoklad (ověřuje zda počáteční bod leží ve zvoleném intervalu)

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

```
> predpoklady4 :=proc(f,x0,a,b,prom)
# x0 musi lezet v intervalu a,b
if(is(x0,RealRange(a,b))) then
  return true;
end if;
return false;
> end proc:
```

predpoklady5

vrací pravdivostní hodnotu, jestli je splněn pátý předpoklad $f(x_0)f''(x_0) > 0$

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

```
> predpoklady5 :=proc(f,x0,a,b,prom)
# f(x0) * f''(x0)>0
```

```
local k;
if(evalf(funkcni_hodnota(f,x0,prom)*
   funkcni_hodnota(diff(f,prom$2),x0,prom))>0) then
  return true;
end if;
return false;
> end proc:
```

predpoklady

vrací pravdivostní hodnotu – dává dohromady všech 5 předpokladů

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

```
> predpoklady :=proc(f,x0,a,b,prom)
# dává dohromady všech 5 podmínek
local pr_1, pr_2, pr_3, pr_4, pr_5;
pr_1:=predpoklady1(f,x0,a,b,prom);
pr_2:=predpoklady2(f,x0,a,b,prom);
pr_3:=predpoklady3(f,x0,a,b,prom);
pr_4:=predpoklady4(f,x0,a,b,prom);
pr_5:=predpoklady5(f,x0,a,b,prom);
if(pr_1=0) then
  return 0;
else
  return pr_1 and pr_2 and pr_3 and pr_4 and pr_5;
end if;
> end proc:
```

IsRovn

vrací pravdivostní hodnotu, zda tečna k funkci v daném bodě je rovnoběžná s osou *x*

1. parametr – funkce

2. parametr – bod

3. parametr – proměnná

```
> IsRovn := proc(f,x0,prom)
# true - je rovnobezna
# false - neni rovnobezna
local zda, d1, d2;
zda:=simplify(subs(prom=x0,diff(f,prom)));
d1:=eval(zda);
d2:=evalf(zda,50);
if(d1=0 or d2=0) then
  return true;
end if;
```

```
return false;
end proc:
```

tecnaShow

pomocná metoda pro zobrazení tečen

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5., 6. parametr – rozsah zobrazení na ose *x*

7. parametr – proměnná

```
> tecnaShow := proc(f,x0,a,b,x1,x2,prom)
local tecna,m,n;
> tecna:=rovtec(f,x0,prom);
m:=plot(tecna,prom,color=red);
n:=plot([[x0,prom,prom=0..evalf(funkcni_hodnota(f,x0,prom))]],
  prom,color=green);
display(m,n,view=[x1..x2,default]);
> end proc:
```

korenShow

pomocná metoda pro vykreslení kořenu

1. parametr – funkce

2., 3. parametr – interval

4. parametr – počáteční bod

5. parametr – jak vysoko bude slovo kořen

6. parametr – proměnná

```
> korenShow := proc(f,a,b,bod,vyska,prom)
    cast[0] - vykresluje bod (kořen)
    cast[1] - vykresluje slovo kořen
> local cast;
>  cast[0]:=plot([[bod,0]], prom,
style=point,symbol=box,color=magenta,symbolsize=12);
 cast[1]:=textplot([bod,vyska,"Koren"],color=magenta);
>  return cast;
> end proc:
```

vypocetChyby

vypočítá chybu II. na základě znalosti dvou aproximací kořene

1. parametr – funkce

2., 3. parametr – dvojice aproximací kořene

4., 5. parametr – interval

6. parametr – proměnná

```
> vypocetChyby := proc(f,bod1,bod2,a,b,prom)
> local m1,m2;
 m1:=minimize(abs(diff(f,prom)),prom=a..b);
 m2:=maximize(abs(diff(f,prom$2)),prom=a..b);
 if(m1=0) then
   return "nelze";
 else
   return evalf(m2/(2*m1)*(bod1-bod2)^2);
end if;
> end proc:
```

odhadChyby

vypočítá odhad chyby II.

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

6., 7. parametr – rozsah zobrazení na ose *x*

8., 9. parametr – rozsah zobrazení na ose *y*

10. parametr – počet kroků

```
> odhadChyby := proc(f,start,a,b,prom,x1,x2,y1,y2,kolik)
> local novy,novye,pl,u,chyba;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  novye[0]:=start;
  novy[0]:=start;
  pl:=0;
  if(predpoklady(f,start,a,b,prom)=true and
not(IsKoren(f,start,prom))) then
    for u from 0 by 1 to kolik do
      if(u>=1) then
        novye[u]:=novye[u-1]-evalf(funkcni_hodnota(f,novye[u-
1],prom))/(evalf(funkcni_hodnota(diff(f,x),novye[u-1],x)));
        novy[u]:=novy[u-1]-funkcni_hodnota(f,novy[u-
1],prom)/(funkcni_hodnota(diff(f,x),novy[u-1],x));
        if(IsKoren(f,novye[u],prom)) then
          return vypocetChyby(f,novy[u],novy[u-1],a,b,prom);
          break;
        end if;
      end if;
      pl:=kolik;
    end do;
    if(not(pl=0)) then
      return vypocetChyby(f,novy[pl],novy[pl-1],a,b,prom);
```

```
    end if;
  end if;
end if;
end proc:
```

newton

grafické znázornění – vykresluje graf znázorňující Newtonovu metodu

1. parametr – funkce

2. parametr – počáteční bod

3., 4. parametr – interval

5. parametr – proměnná

6., 7. parametr – rozsah zobrazení na ose *x*

8., 9. parametr – rozsah zobrazení na ose *y*

10. parametr – počet kroků

```
> newton := proc(f,start,a,b,prom,x1,x2,y1,y2,kolik)
> local u,novy,barva,s,text,showSlovoKoren,showKoren,showSlovoFunkce,
    cele,pl,cojespolecne,pomocx,pomocy,vyska,tecna,kraj,krajni_body,
    vynaseci,co,kl,krizky;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  novy[0]:=start;
  showKoren:="";
  showSlovoKoren:=textplot([2,2,""]);
  pomocx:=(x2-x1)/50;
  pomocy:=(y2-y1)/100;
  vyska:=pomocy*3.5;
  pl:=0;
  barva:=color=[blue,red];
  if(not(predpoklady(f,start,a,b,prom)=0)) then
    if(not(IsKoren(f,start,prom))) then
      tecna[0]:=tecnaShow(f,start,a,b,x1,x2,prom);
    end if;
  end if;

  s:=plot(f,prom=x1..x2,y=y1..y2,barva,axes=normal);
  showSlovoFunkce:=textplot([x2-pomocx,funkcni_hodnota(f,x2,prom),
    "funkce"],color=blue);
```

Jeden z krajních bodů je kořen:

```
> if(predpoklady(f,start,a,b,prom)=0) then
  pomocx:=(b-a)/100;
  if(IsKoren(f,a,prom)) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(a),string));
    cele:=korenShow(f,a,b,a,-1*vyska,prom);
  elif(IsKoren(f,b,prom)) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(b),string));
    cele:=korenShow(f,a,b,b,-1*vyska,prom);
  end if;
```

```
  showKoren:=cele[0];
  showSlovoKoren:=cele[1];
  vypis(text);
  s:=plot(f,prom=a..b,y=y1..y2,barva,axes=normal);
  showSlovoFunkce:=textplot([b-pomocx,funkcni_hodnota(f,b,prom),
    "funkce"],color=blue,view=[a..b,y1..y2]);
```

Nejsou splněny předpoklady:

```
elif(predpoklady(f,start,a,b,prom)=false) then
  vypis("Nejsou splněny předpoklady!");
```

Splněny předpoklady + není ani jeden z krajních bodů kořenem:

```
elif(predpoklady(f,start,a,b,prom)=true) then
  # startovní bod je kořenem
  if(IsKoren(f,start,prom)) then
    text:=cat("Počáteční bod je kořenem: ",convert(prom,string),
       " = ",convert(nula(start),string));
    cele:=korenShow(f,a,b,start,-1*vyska,prom);
    showKoren:=cele[0];
    showSlovoKoren:=cele[1];
    vypis(text);
    s:=plot(f,prom=a..b,y=y1..y2,barva,axes=normal);
    showSlovoFunkce:=textplot([b-pomocx,funkcni_hodnota(f,b,prom),
      "funkce"],color=blue);

  else
    for u from 0 by 1 to kolik do
      barva:=color=[blue,red$u+2];
      if(u>=1) then
        tecna[u]:=tecnaShow(f,novy[u-1],a,b,x1,x2,prom);
        novy[u]:=novy[u-1]-evalf(funkcni_hodnota(f,novy[u-1],prom))/
          (evalf(funkcni_hodnota(diff(f,x),novy[u-1],x)));
        if(IsKoren(f,novy[u],prom)) then
          text:=cat("Kořen byl nalezen: ",convert(prom,string),
            " = ",convert(nula(novy[u]),string)," na konci ",
            convert(u,string),". kroku");
          vypis(text);
          showKoren:=plot([[novy[u],0]], prom=x1..x2, style=point,
            symbol=box,color=magenta,symbolsize=12);
          showSlovoKoren:=textplot([novy[u],-1*vyska,"Koren"],
            color=magenta);
          pl:=u-1;
          break;
        end if;
            end if;
      pl:=kolik;
    end do;
  end if;
end if;

kraj:=textplot([2,2,""]);
if(predpoklady(f,start,a,b,prom)=0) then
  kraj:=plot([[a,0],[b,0]], prom=a..b, style=point,symbol=cross,
    color=red,symbolsize=12);
else
  if(a<x1 and b>x2) then
    kraj:=textplot([2,2,""]);
  elif(a<x1) then
```

```
    kraj:=plot([[b,0]], prom=x1..x2, style=point,symbol=cross,
      color=red,symbolsize=12);
 elif(b>x2) then
    kraj:=plot([[a,0]], prom=x1..x2,
style=point,symbol=cross,color=red,symbolsize=12);
 else
    kraj:=plot([[a,0],[b,0]], prom=x1..x2,style=point,symbol=cross,
      color=red,symbolsize=12);
  end if;
end if;

co:=0;
for kl from 0 by 1 to pl do
  if(novy[kl]>=x1 and novy[kl]<=x2) then
    krizky[co]:=novy[kl];
    co:=co+1;
  end if;
end do;


krajni_body:=textplot([[a+pomocx,vyska,"a"],[b-pomocx,vyska,"b"]],
  color=red);

cojespolecne:=s,showSlovoFunkce,kraj,krajni_body;
if(showKoren="") then
  if(predpoklady(f,start,a,b,prom)=false) then
    display({showSlovoFunkce,s});
  else
    vynaseci:=plot([novy[pl],prom,prom=0..evalf(subs(prom=novy[pl],f)+
      exp(-1000))],color=magenta);
    if(pl=0) then
      display({cojespolecne,textplot([[novy[0],-vyska,x[0]]],
        color=red),vynaseci});
    else
      if(co=0) then
         display({cojespolecne,textplot([[novy[w],
           -vyska,x[w]]$w=0..pl],color=red),tecna[k]$ k=0..pl,
           vynaseci});
      else
        display({cojespolecne,
          textplot([[novy[w],-vyska,x[w]]$ w=0..pl],color=red),
          plot([[krizky[q],0]$ q=0..pl], prom=x1..x2,
          style=point,symbol=cross,color=red,symbolsize=12),
          tecna[k]$ k=0..pl,vynaseci});
      end if;
    end if;
  end if;
else
  if(predpoklady(f,start,a,b,prom)=0) then

display({cojespolecne,showSlovoKoren,showKoren},view=[a..b,y1..y2]);
  else
 display({cojespolecne,showSlovoKoren,showKoren,
    textplot([[novy[w]+pomocx,-vyska,x[w]]$w=0..pl],color=red),
    tecna[k]$ k=0..pl});
  end if;
end if;

else
```

```
  vypis("Špatně zadané intervaly, hranice nebo počet kroků!");
  if(x1>=x2) then
    print(cat("Neplatí podmínka x1<x2  (",convert(nula(x1),string),
      " < ",convert(nula(x2),string),")"));
  end if;
  if(y1>=y2) then
     print(cat("Neplatí podmínka y1<y2  (",convert(nula(y1),string),
       " < ",convert(nula(y2),string),")"));
  end if;
  if(a>=b) then
     print(cat("Neplatí podmínka a<b  (",convert(nula(a),string),
       " < ",convert(nula(b),string),")"));
  end if;
  if(kolik<0) then
     print("Počet kroků musí být celé nezáporné číslo.");
  end if;
  if(not(is(kolik,integer))) then
    print("Počet kroků musí být celé nezáporné číslo.");
  end if;
end if;
> end proc:
```

newton_krok

procedura pro vizualizaci metody, tzv. krok za krokem (nejsou zde ověřovány předpoklady, proto tato metoda lze použít jen na data uvedená v Mapletu)

1. parametr – funkce
2. parametr – počáteční bod
3. parametr – proměnná
4. parametr – počet kroků

```
> newton_krok := proc(f,bod,prom,kolik)
> local prus,tecna,u,m,barva,s,d,krizky;
  prus[0]:=bod;
  for u from 0 by 1 to kolik do
    if(u=0) then
      m:=[f,[prus[0],prom,prom=0..evalf(subs(prom=prus[0],f))]];
      barva:=color=[blue,magenta], thickness=[1,2];
      d:=textplot([prus[0],-1.1,x[0]],color=red);
    else
      tecna[u]:=rovtec(f,prus[u-1],prom);
      prus[u]:=prusecik(tecna[u],prom);
      m:=[f,tecna[j] $ j = 1..u,[prus[k],prom,
        prom=0..evalf(subs(prom=prus[k],f))] $ k = 0..u];
      barva:=color=[blue,red$u,green$u,magenta],
        thickness=[1 $ 2*u+1 ,2];
      if(u=4) then
        d:=textplot([[prus[0],-1.1,x[0]],[prus[l],-1.1,x[l]]$l=1..u-1,
          [prus[4],1.1,x[4]]],color=red);
      else
        d:=textplot([[prus[0],-1.1,x[0]],[prus[l],-1.1,x[l]]$l= 1..u],
          color=red);
      end if;
```

```
    end if;

  end do;
  krizky:=plot([[prus[n],0]$ n=0..u], prom, style=point,symbol=CROSS,
    color=red,symbolsize=12);
  s:=plot(m,prom=0..2.2,-3..35,barva,axes=normal,
    xtickmarks=0,ytickmarks=0);
  display({s,krizky,textplot([2,33,"funkce"],color=blue),d});
end proc:
```

graftecna

grafické znázornění tečny k funkci v daném bodě

1. parametr – funkce

2. parametr – počáteční bod

3. parametr – proměnná

4., 5. parametr – rozsah zobrazení na ose *x*

6., 7. parametr – rozsah zobrazení na ose *y*

8. parametr – pomocná proměnná pro pozdější použití (v Mapletu)

```
> graftecna := proc(f,start,prom,xmin,xmax,ymin,ymax,co)
local m,tecna, cast,t,p,graf,prusecik;
if(xmin<xmax and ymin<ymax) then
   tecna:=rovtec(f,start,prom);
   if(IsKoren(f,start,prom)) then
     cast[0]:=plot([[start,0]], prom,style=point,symbol=box,
       color=magenta,symbolsize=12);
     cast[1]:=textplot([start,(ymax-ymin)/50,"Koren"],color=magenta);
     cast[2]:=plot([f,tecna],prom=xmin..xmax, ymin..ymax,
       color=[blue,red]);
     graf:={cast[0],cast[1],cast[2]};
   else
     graf:=plot([f,tecna,[start,prom,
        prom=0..evalf(subs(prom=start,f))]],
        prom=xmin..xmax, ymin..ymax,color=[blue,red,green]);
   end if;
   if(IsRovn(f,start,prom)) then
     prusecik:="neexistuje";
   else
     prusecik:=solve(tecna=0,prom);
   end if;

  if(co=0) then
    display(graf);
  elif(co=1) then
    return simplify(tecna);
  elif(co=2) then
    return prusecik;
  end if;
> else
  vypis("Rozsah zobrazení byl špatně zadán!");
```

```
  if(xmin>=xmax) then
    print(cat("Neplatí podmínka xmin<xmax  (",
      convert(nula(xmin),string), " < ",
      convert(nula(xmax),string),")"));
  end if;
  if(ymin>=ymax) then
    print(cat("Neplatí podmínka ymin<ymax  (",
      convert(nula(ymin),string)," < ",
      convert(nula(ymax),string),")"));
  end if;
end if;
> end proc:
```

D.2 Maplet "Newtonova metoda – krok za krokem"

Poznámka. Budeme využívat procedury z přílohy B a D.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_vizualizace := Maplet( Window( "Newtonova metoda - krok za
krokem",
> [
  Plotter[PL1](newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,0),
    height=450,width=370 ),
  [
    BoxRow(caption="Počet kroků",border=true,
      Button[krok0]("0",Action(Evaluate(
       PL1 = 'newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,0)'),
        SetOption('krok0'('foreground') = 'orange'),
        SetOption('krok1'('foreground') = 'black'),
        SetOption('krok2'('foreground') = 'black'),
        SetOption('krok3'('foreground') = 'black'),
        SetOption('krok4'('foreground') = 'black'))),
      Button[krok1]("1",Action(Evaluate(
       PL1 = 'newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,1)'),
        SetOption('krok0'('foreground') = 'black'),
        SetOption('krok1'('foreground') = 'orange'),
        SetOption('krok2'('foreground') = 'black'),
        SetOption('krok3'('foreground') = 'black'),
        SetOption('krok4'('foreground') = 'black'))),
      Button[krok2]("2",Action(Evaluate(
       PL1 = 'newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,2)'),
        SetOption('krok0'('foreground') = 'black'),
        SetOption('krok1'('foreground') = 'black'),
        SetOption('krok2'('foreground') = 'orange'),
        SetOption('krok3'('foreground') = 'black'),
        SetOption('krok4'('foreground') = 'black'))),
      Button[krok3]("3",Action(Evaluate(
       PL1 = 'newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,3)'),
        SetOption('krok0'('foreground') = 'black'),
        SetOption('krok1'('foreground') = 'black'),
        SetOption('krok2'('foreground') = 'black'),
        SetOption('krok3'('foreground') = 'orange'),
        SetOption('krok4'('foreground') = 'black'))),
      Button[krok4]("4",Action(Evaluate(
       PL1 = 'newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,4)'),
        SetOption('krok0'('foreground') = 'black'),
        SetOption('krok1'('foreground') = 'black'),
```

```
        SetOption('krok2'('foreground') = 'black'),
        SetOption('krok3'('foreground') = 'black'),
        SetOption('krok4'('foreground') = 'orange')))
    ),
    BoxColumn(
      [
        Button("Obnovit",Action(Evaluate(
         PL1='newton_krok((x+0.6)*(x-0.5)*(x+5),2.1,x,0)'),
          SetOption('krok0'('foreground') = 'orange'),
          SetOption('krok1'('foreground') = 'black'),
          SetOption('krok2'('foreground') = 'black'),
          SetOption('krok3'('foreground') = 'black'),
          SetOption('krok4'('foreground') = 'black'))),
        Button("Konec",Shutdown())
      ],
      Button("Nápověda",RunDialog(MD1)))
]]),
MessageDialog[MD1]( "Tento Maplet slouží pro vizualizaci Newtonovy
metody.\nPopis barevného značení:\n   modrá - funkce\n   červená -
tečna\n   zelená - kolmice a funkční hodnota v daném bodě\n   fialová
- funčkní hodnota posledně získaného bodu",'information')):
> Maplets[Display]( maplet_vizualizace );
```

D.3 Maplet "Newtonova metoda – grafické znázornění"

Poznámka. Budeme využívat procedury z přílohy B a D.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_graficke := Maplet( Window( "Newtonova metoda - grafické
znázornění",
[
  BoxRow(
    ["Funkce: ", TextField[Fce]("4*x^3+11*x^2+5*x-2"),
     "Proměnná: ", TextField[Prom]("x",3)],
    ["Počáteční bod: ", TextField[Start]("1",3),
     "Počet kroků: ",TextField[Kroky]("2",3)],
    BoxRow(caption="Interval",border=true,
      ["Začátek: ","Konec: "],
      [TextField[A]("0",3),TextField[B]("1",3)]
    ),
    BoxRow(caption="Rozsah zobrazení",border=true,
      ["X-min: ","X-max: "],
      [TextField[Xmin]("0",3),TextField[Xmax]("1.1",3)],
      ["Y-min: ","Y-max: "],
      [TextField[Ymin]("-5",3),TextField[Ymax]("23",3)]
    )
  ),
  [
  Plotter[GRAF]( plot(undefined, x=-100..100),width=700,height=400 )
  ],
  [
    "Odhad chyby posledního bodu:",
    TextField[Chyba]("",11,editable=false),
    Button[B2]("Použít Newtonovu metodu",Action(Evaluate(
      GRAF='newton(Fce,Start,A,B,Prom,Xmin,Xmax,Ymin,Ymax,Kroky)'),
      Evaluate(Chyba='odhadChyby(Fce,Start,A,B,Prom,Xmin,Xmax,
      Ymin,Ymax,Kroky)'))),

    Button("Obnovit",Action(
      SetOption(Fce="4*x^3+11*x^2+5*x-2"),
      SetOption(Prom="x"),
      SetOption(Kroky="0"),
      SetOption(Start="1"),
      SetOption(A="0"),
      SetOption(B="1"),
      SetOption(Chyba=""),
      Evaluate(GRAF='plot(undefined, x=-100..100)'))
```

```
    ),
    Button("Nápověda",RunDialog(MD1)),
    Button("Konec",Shutdown())
  ]
]),
> MessageDialog[MD1]( "\nOPTIMALIZOVÁNO PRO POLYNOMIÁLNÍ
FUNKCE!\n\nTento Maplet slouží pro grafické znázornění Newtonovy
metody.\nPopis barevného značení:\n   modrá - funkce\n   červená -
tečna\n   zelená - kolmice a funkční hodnota v daném bodě\n   fialová
- funčkní hodnota posledně získaného bodu\n\n",'information')):
> Maplets[Display]( maplet_graficke );
```

D.4 Maplet “Newtonova metoda – tečna”

Poznámka. Budeme využívat procedury z přílohy B a D.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_tecna := Maplet( Window( "Newtonova metoda - tečna",
[
  [
   ["Funkce: ", TextField[Fce]("4*x^3+11*x^2+5*x-2",onchange=Action(
     SetOption(B=false,`option`=enabled),
     SetOption(C=false,`option`=enabled)))],
   ["Proměnná: ", TextField[Prom]("x",3,onchange=Action(
     SetOption(B=false,`option`=enabled),
     SetOption(C=false,`option`=enabled)))],
   ["X-souřadnice bodu dotyku: ", TextField[Start]("1",3,
    onchange=Action(
     SetOption(B=false,`option`=enabled),
     SetOption(C=false,`option`=enabled)))]
  ],
  [
   BoxRow("Rovnice tečny:",TextField[Tecna]("",editable = false,20),
   "Průsečík tečny a osy x:",TextField[Prusn]("",editable = false,
   10))
  ],
  [Plotter[PL2]( plot(undefined, x=0..100),width=600,height=400 )],
  [
    BoxRow(caption="Rozsah zobrazení",border=true,
      ["X-min: ","X-max: "],
      [TextField[Xmin]("-2",3),TextField[Xmax]("2",3)],
      ["Y-min: ","Y-max: "],
      [TextField[Ymin]("-10",3),TextField[Ymax]("20",3)]
    ),
    BoxColumn(
      [Button("Spočti tečnu", Action(
        Evaluate(
          Tecna='graftecna(Fce,Start,Prom,Xmin,Xmax,Ymin,Ymax,1)'),
        Evaluate(
          Prusn='graftecna(Fce,Start,Prom,Xmin,Xmax,Ymin,Ymax,2)'),
        SetOption(B=true,`option`=enabled),
        SetOption(C=true,`option`=enabled))),
      Button[B]("Vykresli tečnu",Action(
        Evaluate(
          PL2='graftecna(Fce,Start,Prom,Xmin,Xmax,Ymin,Ymax,0)')),
          'not enabled' ),
```

```
      Button[C]("Návazná tečna",Action(
        SetOption('target' = 'Start', Argument('Prusn')),
        SetOption(B=false,`option`=enabled)),'not enabled')
      ],
      [
       Button("Obnovit",Action(
          SetOption(Fce="4*x^3+11*x^2+5*x-2"),
          SetOption(Prom="x"),
          SetOption(Start="1"),
          SetOption(Tecna=""),
          SetOption(Prusn=""),
          SetOption(B=false,`option`=enabled),
          SetOption(C=false,`option`=enabled),
          Evaluate(PL2='plot(undefined, x=0..100)'))),
       Button("Nápověda",RunDialog(MD1)),
       Button("Konec",Shutdown())
     ]
   )
  ]
]),
> MessageDialog[MD1]( "\nOPTIMALIZOVÁNO PRO POLYNOMIÁLNÍ
FUNKCE!\n\nTento Maplet slouží pro vykreslení tečny v daném
bodě.\nPopis barevného značení:\n   modrá - funkce\n   červená -
tečna\n   zelená - kolmice a funkční hodnota v daném bodě\n   fialová
- funčkní hodnota v posledně získaném bodu\nTlačítko\n   Návazná tečna
- slouží pro dosazení získaného průsečíku jako nového bodu
dotyku.\n",'information')):
> Maplets[Display]( maplet_tecna );
```

PŘÍLOHA E

Regula falsi

E.1 Společné procedury pro regula falsi

Poznámka. Budeme využívat procedury z přílohy B.

pocet	IntervalShow	odhadChyby
predpoklady1	tetivaShow	regulafalsi
predpoklady2	korenShow	regulafalsi_krok
predpoklady3	reseni	
predpoklady	vypocetChyby	

Knihovny potřebné pro tyto procedury:

```
> with(Maplets[Elements]):
> with(plots):
> with(StringTools):
> with(student,showtangent):
```

pocet

vrací počet prvků v poli ukončené výjimkou (invalid subscript selector)

1. parametr – pole

```
> pocet :=proc(tr)
local j, result;
try
  for j from 1 by 1 to infinity do
    tr[j];
  end do;
catch "invalid subscript selector":
  result := j-1
end try;
return result;
> end proc:
```

predpoklady1

vrací pravdivostní hodnotu, zda je splněn první předpoklad $f(a)f(b) < 0$

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

```
> predpoklady1 :=proc(f,a,b,prom)
# ověřuje zda f(a)*f(b)<0
# ano - vrací true, ne - vrací false, 0 - krajní bod je kořenem
> if(evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))<0) then
  return true;
elif (evalf(funkcni_hodnota(f,a,prom)*funkcni_hodnota(f,b,prom))=0)
then
  return 0;
else
  return false;
end if;
> end proc:
```

predpoklady2

vrací pravdivostní hodnotu, jestli je splněn druhý předpoklad (ryzí monotónnost na intervalu)

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

```
> predpoklady2 :=proc(f,a,b,prom)
# ověřuje ryzí monotónnost na intervalu a, b
local der, reseni_mensi, reseni_vetsi, interval, i,j;
> der:=diff(f,prom);
reseni_mensi:=[solve(der>=0)];
reseni_vetsi:=[solve(der<=0)];
interval:=RealRange(a,b);

for i from 1 by 1 to pocet(reseni_mensi) do
  if(is(interval,reseni_mensi[i])) then
    return true;
  end if;
end do;

for j from 1 by 1 to pocet(reseni_vetsi) do
  if(is(interval,reseni_vetsi[j])) then
    return true;
  end if;
end do;
return false;
> end proc:
```

predpoklady3

vrací pravdivostní hodnotu, jestli je splněn třetí předpoklad (ověřuje zda na intervalu není inflexe)

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

```
> predpoklady3 :=proc(f,a,b,prom)
# ověřuje zda na intervalu a,b není inflexe
local der, reseni,j;
der:=diff(f,prom$2);
reseni:=[solve(der=0)];
for j from 1 by 1 to pocet(reseni) do
  if(is(reseni[j],RealRange(a,b))) then
    return false;
  end if;
end do;
return true;
> end proc:
```

predpoklady

vrací pravdivostní hodnotu – dává dohromady všechny 3 předpoklady

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

```
> predpoklady :=proc(f,a,b,prom)
# dává dohromady všechny 3 podmínky
local pr_1, pr_2, pr_3;
pr_1:=predpoklady1(f,a,b,prom);
pr_2:=predpoklady2(f,a,b,prom);
pr_3:=predpoklady3(f,a,b,prom);
if(pr_1=0) then
  return 0;
else
  return pr_1 and pr_2 and pr_3;
end if;
> end proc:
```

IntervalShow

pomocná metoda pro zobrazení intervalu

1. parametr – funkce

2., 3. parametr – interval

4., 5. parametr – rozsah zobrazení na ose *y*

6. parametr – proměnná

7. parametr – barva, kterou bude interval vybarven

```
> intervalShow := proc(f,a,b,y1,y2,prom,vybarveni)
> return inequal( {prom>=a, prom<=b}, prom=a..b, y=y1..y2 ,
    optionsfeasible=(color=vybarveni),
    optionsclosed=(color=red, thickness=1));
> end proc:
```

tetivaShow

pomocná metoda pro zobrazení tětiv

1. parametr – funkce

2., 3. parametr – interval

4., 5. parametr – rozsah zobrazení na ose *x*

6. parametr – proměnná

```
> tetivaShow := proc(f,a,b,x1,x2,prom)
local k,q,m,n;
> k:=(funkcni_hodnota(f,a,prom)-funkcni_hodnota(f,b,prom))/(a-b);
 q:=funkcni_hodnota(f,b,prom)-k*b;
 m:=plot(k*prom+q,prom=a..b,color=red,view=[x1..x2,default]);
 n:=plot([[a,prom,prom=0..evalf(funkcni_hodnota(f,a,prom))],
   [b,prom,prom=0..evalf(funkcni_hodnota(f,b,prom))]],
   prom=x1..x2,color=coral);
 display(m,n);
> end proc:
```

korenShow

pomocná metoda pro vykreslení kořene

1. parametr – funkce

2., 3. parametr – interval

4. parametr – počáteční bod

5. parametr – jak vysoko bude slovo kořen

6. parametr – proměnná

```
> korenShow := proc(f,a,b,bod,vyska,prom)
    cast[0] - vykresluje bod (kořen)
    cast[1] - vykresluje slovo kořen
> local cast;
> cast[0]:=plot([[bod,0]], prom, style=point,symbol=box,
   color=magenta,symbolsize=12);
 cast[1]:=textplot([bod,vyska,"Koren"],color=magenta);
> return cast;
> end proc:
```

reseni

rozepisuje pole hodnot ve stanoveném formátu

1. parametr – pole hodnot (formát [a1,a2,b1,b2,…], a1 je začátek prvního intervalu, a2 konec, b1, b2 označuje další interval, atd.)
2. parametr – pole aproximací kořene
3. parametr – počet dvojic
4. parametr – text (pomocná proměnná)

```
> reseni := proc(pole,polekoren,num,zdaKoren)
local h,delka,text0,text1,text2;
if(not(zdaKoren="")) then
print("-------------------------------------");
print(zdaKoren);
print("-------------------------------------");
print(``)
end if;
for h from 0 to 2*num by 2 do
delka:=evalf(pole[h+1] - pole[h]);
if(h=0) then
  text0:="-------------------------------------";
  text1:=cat(convert(h/2,string),". krokem jsme dostali interval: < ",
    nula(pole[h])," ; ",nula(pole[h+1])," >");
else
  text0:=cat(convert(h/2,string),
    ". krokem jsme dostali aproximaci kořene: ",
    convert(nula(polekoren[h/2]),string));
  text1:=cat("Vznikl nám interval: < ",nula(pole[h]),
    " ; ",nula(pole[h+1])," >");
end if;
text2:=cat("Velikost intervalu je: ",convert(nula(delka),string));
print(text0);
print(text1);
print(text2);
print("-------------------------------------");
end do;
end proc:
```

vypocetChyby

vypočítá chybu I. na základě znalosti aproximace kořene

1. parametr – funkce
2. parametr – aproximace kořene
3., 4. parametr – interval
5. parametr – proměnná

```
> vypocetChyby := proc(f,bod,a,b,prom)
> local m;
 m:=minimize(abs(diff(f,prom)),prom=a..b);
```

```
 if(m=0) then
   return "nelze";
 else
   return evalf(abs(funkcni_hodnota(f,bod,prom))/m);
end if;
> end proc:
```

odhadChyby

vypočítá odhad chyby I.

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

5., 6. parametr – rozsah zobrazení na ose *x*

7., 8. parametr – rozsah zobrazení na ose *y*

9. parametr – počet kroků

```
> odhadChyby := proc(f,a,b,prom,x1,x2,y1,y2,kolik)
> local novy,pl,u,chyba,a1,b1;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  a1:=a;
  b1:=b;
  pl:=0;
  if(predpoklady(f,a,b,prom)=true) then
      for u from 0 by 1 to kolik do
    if(u>=1) then
      novy[u]:=a1-((b1-a1)*evalf(funkcni_hodnota(f,a1,prom)))/
        (evalf(funkcni_hodnota(f,b1,prom))-
        evalf(funkcni_hodnota(f,a1,prom)));
      if(IsKoren(f,novy[u],prom)) then
        return vypocetChyby(f,novy[u],a,b,prom);
        break;
      else
        if(predpoklady(f,a1,novy[u],prom)=true) then
          b1:=novy[u];
        elif(predpoklady(f,novy[u],b1,prom)=true) then
          a1:=novy[u];
        end if;
      end if;
    end if;
    pl:=kolik;
  end do;o;
    if(not(pl=0)) then
      return vypocetChyby(f,novy[pl],a,b,prom);
    end if;
  end if;
end if;
end proc:
```

regulafalsi

grafické znázornění – vykresluje graf znázorňující metodu regula falsi

1. parametr – funkce

2., 3. parametr – interval

4. parametr – proměnná

5., 6. parametr – rozsah zobrazení na ose *x*

7., 8. parametr – rozsah zobrazení na ose *y*

9. parametr – počet kroků

10. parametr – zda mají být zobrazeny tětivy (true/false)

11. parametr – zda mají být zobrazeny intervaly (true/false)

```
> regulafalsi := proc(f,a,b,prom,x1,x2,y1,y2,kolik,zda_tet,zda_int)
> local a1,b1,h,novy,u,barva,s,text,showSlovoKoren,showKoren,
    showSlovoFunkce,cele,barvy,interval,pl,krajni_body,pom,pom1,
    cojespolecne,pomocx,pomocy,vyska,tetiva,kraj,co,kl,krizky;
if(x1<x2 and y1<y2 and a<b and kolik>=0 and is(kolik,integer)) then
  barvy:=barvicky(kolik,10);
  showKoren:="";
  showSlovoKoren:=textplot([2,2,""]);
  pomocx:=(x2-x1)/50;
  pomocy:=(y2-y1)/100;
  vyska:=pomocy*3.5;
  a1:=a;
  b1:=b;
  pl:=0;
  barva:=color=[blue,red];
  if(not(predpoklady(f,a,b,prom)=0)) then
    tetiva[0]:=tetivaShow(f,a,b,x1,x2,prom);
  end if;

  interval[0]:=intervalShow(f,a,b,y1,y2,prom,barvy[0]);
  s:=plot(f,prom=x1..x2,y=y1..y2,barva,axes=normal);
  showSlovoFunkce:=textplot([x2-pomocx,funkcni_hodnota(f,x2,prom),
    "funkce"],color=blue);
```

Jeden z krajních bodů je kořen:

```
> if(predpoklady(f,a,b,prom)=0) then
  pomocx:=(b-a)/100;
  if(funkcni_hodnota(f,a,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(a),string));
    cele:=korenShow(f,a,b,a,-1*vyska,prom);
  elif(funkcni_hodnota(f,b,prom)=0) then
    text:=cat("Krajní bod je kořenem: ",convert(prom,string),
      " = ",convert(nula(b),string));
    cele:=korenShow(f,a,b,b,-1*vyska,prom);
  end if;
  showKoren:=cele[0];
  showSlovoKoren:=cele[1];
```

```
  vypis(text);
  s:=plot(f,prom=a..b,y=y1..y2,barva,axes=normal);
  showSlovoFunkce:=textplot([b-pomocx,funkcni_hodnota(f,b,prom),
    "funkce"],color=blue);
```

Nejsou splněny předpoklady:

```
elif(predpoklady(f,a,b,prom)=false) then
  vypis("Nejsou splněny předpoklady!");
```

Splněny předpoklady + není ani jeden z krajních bodů kořenem:

```
elif(predpoklady(f,a,b,prom)=true) then
  for u from 0 by 1 to kolik do
    barva:=color=[blue,red$u+2];
    if(u>=1) then
      novy[u]:=a1-((b1-a1)*evalf(funkcni_hodnota(f,a1,prom)))/
        (evalf(funkcni_hodnota(f,b1,prom))-
        evalf(funkcni_hodnota(f,a1,prom)));
      if(funkcni_hodnota(f,novy[u],prom)=0) then
        text:=cat("Kořen byl nalezen: ",convert(prom,string)," = ",
          convert(nula(novy[u]),string),
          " na začátku ",convert(u,string),". kroku");

        vypis(text);
        showKoren:=plot([[novy[u],0]], prom=x1..x2, style=point,
          symbol=box,color=magenta,symbolsize=12);
        showSlovoKoren:=textplot([novy[u],-1*vyska,"Kořen"],
          color=magenta);
        pl:=u-1;
        break;
      else
        if(predpoklady(f,a1,novy[u],prom)=true) then
          b1:=novy[u];
        elif(predpoklady(f,novy[u],b1,prom)=true) then
          a1:=novy[u];
        end if;
      end if;
    end if;
    interval[u]:=intervalShow(f,a1,b1,y1+pomocy*u,y2-pomocy*u,prom,
      barvy[u]);
    tetiva[u]:=tetivaShow(f,a1,b1,x1,x2,prom);
    pl:=kolik;
  end do;
end if;

kraj:=textplot([2,2,""]);

if(zda_int=false) then
  if(predpoklady(f,a,b,prom)=0) then
    kraj:=plot([[a,0],[b,0]], prom=a..b, style=point,symbol=cross,
      color=red,symbolsize=12);
  else
    if(a<x1 and b>x2) then
      kraj:=textplot([2,2,""]);
    elif(a<x1) then
      kraj:=plot([[b,0]], prom=x1..x2, style=point,symbol=cross,
        color=red,symbolsize=12);
    elif(b>x2) then
```

```
      kraj:=plot([[a,0]], prom=x1..x2, style=point,symbol=cross,
        color=red,symbolsize=12);
    else
      kraj:=plot([[a,0],[b,0]], prom=x1..x2, style=point,symbol=cross,
        color=red,symbolsize=12);
    end if;
  end if;
end if;

co:=0;
for kl from 1 by 1 to pl do
  if(novy[kl]>=x1 and novy[kl]<=x2) then
    krizky[co]:=novy[kl];
    co:=co+1;
  end if;
end do;

krajni_body:=textplot([[a+2*pomocx,vyska,"a=x[1]"],[b-
2*pomocx,vyska,"b=x[2]"]],color=red);
cojespolecne:=s,showSlovoFunkce,krajni_body;

pom1:=0;
for h from pl by -1 to 0 do
  pom[pom1]:=interval[h];
  pom1:=pom1+1;
end do;


if(zda_tet=false and zda_int=false) then
  if(showKoren="") then
    if(predpoklady(f,a,b,prom)=false) then
      display({cojespolecne,kraj});
    else
      if(pl=0) then
        display({cojespolecne,kraj,textplot([[novy[w+1]+pomocx,vyska,
          x[w+3]]$ w=0..pl-1],color=red)});
      else
        if(co=0) then
          display({cojespolecne,kraj,textplot([[novy[w+1]+pomocx,
            vyska,x[w+3]]$ w=0..pl-1],color=red)});
        else
          display({cojespolecne,kraj,textplot([[novy[w+1]+pomocx,
            vyska,x[w+3]]$ w=0..pl-1],color=red),
            plot([[krizky[t],0]$ t=0..co-1], prom=x1..x2, style=point,
            symbol=cross,color=red,symbolsize=12)});
        end if;
      end if;
    end if;
  else
    if(predpoklady(f,a,b,prom)=0) then
      display({cojespolecne,kraj,showSlovoKoren,showKoren});
    else
      display({cojespolecne,kraj,showSlovoKoren, showKoren,
        textplot([[novy[w+1]+pomocx,vyska,x[w+3]]$ w=0..pl],
        color=red)});
    end if;
  end if;

elif(zda_tet=false and zda_int=true) then
```

```
  if(showKoren="") then
    if(predpoklady(f,a,b,prom)=false) then
      display({cojespolecne,interval[0]});
    else
      display({cojespolecne,
        textplot([[novy[w+1]+pomocx,vyska,x[w+3]]$ w=0..pl-1],
        color=red)},pom[z] $ z=0..pl);
    end if;
  else
    if(predpoklady(f,a,b,prom)=0) then
      display({cojespolecne,showSlovoKoren, showKoren}, pom[0]);
    else
      display({cojespolecne,showSlovoKoren, showKoren,
        textplot([[novy[w+1]+pomocx,vyska,x[w+3]]$ w=0..pl],
        color=red)},pom[z] $ z=0..pl);
    end if;
  end if;

elif(zda_tet=true and zda_int=false) then
  if(showKoren="") then
    if(predpoklady(f,a,b,prom)=false) then
      display({cojespolecne,kraj});
    else
      if(pl=0) then
        display({cojespolecne,kraj,textplot([[novy[w+1]+pomocx,
          vyska,x[w+3]]$ w=0..pl-1],color=red)});
      else
        display({cojespolecne,kraj,textplot([[novy[w+1]+pomocx,
          vyska,x[w+3]]$ w=0..pl-1],color=red),tetiva[k]$ k=0..pl-1});
      end if;
    end if;
  else
    if(predpoklady(f,a,b,prom)=0) then
      display({cojespolecne,kraj,showSlovoKoren,showKoren});
    else
      display({cojespolecne,kraj,showSlovoKoren, showKoren,
        textplot([[novy[w+1]+pomocx,vyska,x[w+3]]$ w=0..pl],
        color=red),tetiva[k]$ k=0..pl});
    end if;
  end if;

elif(zda_tet=true and zda_int=true) then
  if(showKoren="") then
    if(predpoklady(f,a,b,prom)=false) then
      display({cojespolecne,interval[0]});
    else
      if(pl=0) then
        display({cojespolecne,textplot([[novy[w+1]+pomocx,
        vyska,x[w+3]]$ w=0..pl-1],color=red)},pom[z] $ z=0..pl);
      else
        display({cojespolecne,textplot([[novy[w+1]+pomocx,
        vyska,x[w+3]]$ w=0..pl-1],color=red),
        tetiva[k]$ k=0..pl-1},pom[z] $ z=0..pl);
      end if;
    end if;
  else
    if(predpoklady(f,a,b,prom)=0) then
      display({cojespolecne, showSlovoKoren, showKoren}, pom[0]);
    else
```

```
      display({cojespolecne,showSlovoKoren, showKoren,
        textplot([[novy[w+1]+pomocx,vyska,x[w+3]]$ w=0..pl],
       color=red),tetiva[k]$ k=0..pl},pom[z] $ z=0..pl);
    end if;
  end if;
end if;

else
  vypis("Špatně zadané intervaly, hranice nebo počet kroků!");
  if(x1>=x2) then
    print(cat("Neplatí podmínka x1<x2  (",convert(nula(x1),string),
      " < ",convert(nula(x2),string),")"));
  end if;
  if(y1>=y2) then
     print(cat("Neplatí podmínka y1<y2  (",convert(nula(y1),string),
       " < ",convert(nula(y2),string),")"));
  end if;
  if(a>=b) then
     print(cat("Neplatí podmínka a<b  (",convert(nula(a),string),
       " < ",convert(nula(b),string),")"));
  end if;
  if(kolik<0) then
     print("Počet kroků musí být celé nezáporné číslo.");
  end if;
  if(not(is(kolik,integer))) then
    print("Počet kroků musí být celé nezáporné číslo.");
  end if
end if;
> end proc:
```

regulafalsi_krok

procedura slouží pro vizualizaci metody, tzv. krok za krokem (nejsou zde ověřovány předpoklady, proto tato funkce lze použít jen na data uvedená v Mapletu)

1. parametr – funkce
2., 3. parametr – interval
4. parametr – proměnná
5., 6. parametr – rozsah zobrazení na ose *y*
7. parametr – počet kroků

```
> regulafalsi_krok := proc(f,a,b,prom,y1,y2,kolik)
> local a1,b1,h,novy,u,barva,s,showSlovoFunkce,pl,krajni_body,pom,pom1,
   pomocx,pomocy,vyska,tetiva,kraj;
  a1:=a;
  b1:=b;
  pl:=0;
  barva:=color=[blue,red];
  pomocx:=(b-a)/50;
  pomocy:=(y2-y1)/100;
  vyska:=pomocy*4;
```

```
  s:=plot(f,prom=a..b,y=y1..y2,barva,axes=normal,xtickmarks=0,
    ytickmarks=0);
  showSlovoFunkce:=textplot([b-7*pomocx,
    funkcni_hodnota(f,b-10*pomocx,prom),"funkce"],color=blue);

  for u from 0 by 1 to kolik do
    barva:=color=[blue,red$u+2];
    if(u>=1) then
      novy[u]:=a1-((b1-a1)*evalf(funkcni_hodnota(f,a1,prom)))/
        (evalf(funkcni_hodnota(f,b1,prom))-
        evalf(funkcni_hodnota(f,a1,prom)));
      if(predpoklady(f,a1,novy[u],prom)=true) then
        b1:=novy[u];
      elif(predpoklady(f,novy[u],b1,prom)=true) then
        a1:=novy[u];
      end if;
    end if;
    tetiva[u]:=tetivaShow(f,a1,b1,a,b,prom);
    pl:=kolik;
  end do;

kraj:=plot([[a,0],[b,0]], prom=a..b, style=point,symbol=CROSS,
  color=red,symbolsize=12);
krajni_body:=textplot([[a+2*pomocx,vyska,"a=x[1]"],
  [b-2*pomocx,vyska,"b=x[2]"]],color=red);

pom1:=0;
for h from pl by -1 to 0 do
  pom[pom1]:=interval[h];
  pom1:=pom1+1;
end do;

if(pl=0) then
  display({s,showSlovoFunkce,krajni_body,kraj});
else
  display({s,showSlovoFunkce,krajni_body,kraj,textplot([[novy[w+1],
     vyska,x[w+3]]$ w=0..pl-1],color=red),tetiva[k]$ k=0..pl-1});
end if;

end proc:
```

E.2 Maplet “Regula falsi – krok za krokem”

Poznámka. Budeme využívat procedury z přílohy B a E.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Pomocná procedura (bez popisu):

```
> vysledek:=proc(a,b)
local u;
for u from 1 by 1 to b do
  if(a[u]='true') then
    return(u-1);
    break;
 end if;
end do;
end proc:
```

Samotný Maplet:

```
> maplet_vizualizace := Maplet( Window( "Regula falsi - krok za
krokem",
> [
  Plotter[PL1]( regulafalsi_krok(4*x^3+11*x^2+5*x-2,-0.2,1,x,-3,18,0),
    height=360,width=650 ),

  [BoxRow(caption="Počet kroků",border=true,
    RadioButton['RB1']("0 kroků", 'value'=true, 'group'='BG1'),
    RadioButton['RB2']("1 krok", 'value'=false, 'group'='BG1'),
    RadioButton['RB3']("2 kroky", 'value'=false, 'group'='BG1'),
    RadioButton['RB4']("3 kroky", 'value'=false, 'group'='BG1'),
    RadioButton['RB5']("4 kroky", 'value'=false, 'group'='BG1')),

    Button("Obnovit",Action(
      Evaluate(
        PL1='regulafalsi_krok(4*x^3+11*x^2+5*x-2,-0.2,1,x,-3,18,0)'),
      SetOption('RB1'=true))),
    Button("Nápověda",RunDialog(MD1)),
    Button("Konec",Shutdown())
> ]
> ]),
```

```
> MessageDialog[MD1]( "Tento Maplet slouží pro vizualizaci metody tětiv
(regula falsi).\n0.kroků zobrazuje funkci a krajní body
intervalu.",'information'),
ButtonGroup['BG1'](onchange=(Evaluate(
   PL1='regulafalsi_krok(4*x^3+11*x^2+5*x-2,-0.2,1,x,
   -3,18,vysledek(['RB1','RB2','RB3','RB4','RB5'],5))') ))):
> Maplets[Display]( maplet_vizualizace );
```

E.3 Maplet “Regula falsi – grafické znázornění”

Poznámka. Budeme využívat procedury z přílohy B a E.1.

Knihovny potřebné pro tento Maplet:

```
> with(plots):
> with(Maplets[Elements]):
> with(StringTools):
```

Samotný Maplet:

```
> maplet_graficke := Maplet( Window( "Regula falsi - grafické
znázornění",[
> BoxRow(
[Plotter[PL3]( plot(undefined, x=-100..100),width=700,height=500 )],
[
  BoxRow('halign'='left',
  ["Funkce: ","Proměnná: ","Počet kroků: "
  ],
  [
    TextField[Fce]("4*x^3+11*x^2+5*x-2",10),
    TextField[Prom]("x",3),
    TextField[Kroky]("3",3)
  ]),

   BoxRow(caption="Interval",border=true,
      ["Začátek: ","Konec: "],
      [TextField[Inta]("0",3),TextField[Intb]("1",3)]
    ),
    BoxRow(caption="Rozsah zobrazení",border=true,
      ["X-min: ","X-max: "],
      [TextField[Xmin]("0",3),TextField[Xmax]("1",3)],
      ["Y-min: ","Y-max: "],
      [TextField[Ymin]("-3",3),TextField[Ymax]("18",3)]
    ),

  BoxRow(
    [CheckBox['zda_tet'](caption="Zobrazit tětivy",
      onchange=Evaluate(
        PL3='regulafalsi(Fce,Inta,Intb,Prom,Xmin,Xmax,Ymin,Ymax,Kroky,
        zda_tet,zda_int)'))
    ],
    [CheckBox['zda_int'](caption="Zobrazit intervaly",
      onchange=Evaluate(
        PL3='regulafalsi(Fce,Inta,Intb,Prom,Xmin,Xmax,Ymin,Ymax,Kroky,
        zda_tet,zda_int)'))
    ]),
    Button[B2]("Použít metodu tětiv",
      Action(
       Evaluate(
```

```
        PL3='regulafalsi(Fce,Inta,Intb,Prom,Xmin,Xmax,Ymin,Ymax,
        Kroky,zda_tet,zda_int)'),
      Evaluate(
        Chyba='odhadChyby(Fce,Inta,Intb,Prom,Xmin,Xmax,Ymin,Ymax,
        Kroky)'))),
  BoxRow(

    Button("Obnovit",Action(
      SetOption(Fce="4*x^3+11*x^2+5*x-2"),
      SetOption(Prom="x"),
      SetOption(Kroky="3"),
      SetOption(Inta="0"),
      SetOption(Intb="1"),
      SetOption(Chyba=""),
      Evaluate(PL3='plot(undefined, x=-100..100)'))),
    Button("Nápověda",RunDialog(MD1)),
    Button("Konec",Shutdown())
    ),
  HorizontalGlue(),
  "Odhad chyby posledního bodu:",
  TextField[Chyba]("",11,editable=false)
])
]),
> MessageDialog[MD1]( "\nOPTIMALIZOVÁNO PRO POLYNOMIÁLNÍ
FUNKCE!\n\nTento Maplet slouží pro grafické znázornění metody tětiv
(regula falsi).\nUmožňuje vypnutí/zapnutí zobrazení tětiv nebo
intervalů, které pomocí metody dostáváme.",'information')):
> Maplets[Display]( maplet_graficke );
```