13 Algoritmy pro numerické výpočty

Kreslíková, 16. 10. 2023

13.1 Úvod

V této kapitole si ukážeme skupinu obecně známých algoritmů pro numerické výpočty. Pro tuto skupinu algoritmů je typické, že řeší matematické problémy, které by bez počítače byly velice pracné a náchylné k omylům při ručních výpočtech. Zároveň je pro ně typické, že od začátku nepočítáme s přesným výsledkem, ale spokojíme se s určitou předem definovanou chybou. Tyto výpočty vedou na iterační algoritmy.

Seznámíme se zde s algoritmem zvaným Hornerovo schéma a jeho využitím při efektivním vyčíslování polynomů nebo vyčíslením hodnoty čísla v zadané číselné soustavě. Dále se seznámíme s metodou hledání kořene nelineární rovnice pomocí půlení intervalu a několika metodami výpočtu určitého integrálu zadané funkce.

13.2 Motivace

Často se uvádí, že numerické metody jsou současně vědou i uměním. Využívá se zde postupů, kdy při procesu řešení matematické úlohy zformulované na základě znalosti problému lze dojít k řešení úlohy s využitím pouze aritmetických a logických operací.

V praxi to znamená, že pomocí numerických metod lze vyřešit problémy, které nejdou řešit přímo, nebo by řešení bylo příliš složité, časově a ekonomicky náročné. Výsledky řešení jsou přibližné. Nepřesnost (tzv. chyba), která vzniká při numerickém řešení, je udávána pouze jako odhad chyby (mnohdy pesimistický), neboť přesné řešení není známo. Máme na mysli tzv. chybu metody.

Chyby vzniklé při formulaci matematického problému zanedbáním některých skutečností (použitím modelu), tvoří další skupinu chyb, se kterými je nutno někdy počítat. Tyto nepřesnosti jsou však často zanedbávány. Při výpočtech předpokládáme v současnosti výhradně použití počítačů. Často je k dispozici několik možných postupů výpočtu; to dává s ohledem na typ funkce možnost výběru vhodné metody. Každá metoda má výhody, ale i nevýhody. Dále jsou jednotlivé postupy stručně popsány a implementovány.

13.3 Výpočet polynomu

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

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

se nazývá mnohočlen (v jedné proměnné x s reálnými, resp. komplexními koeficienty). Místo mnohočlen se také říká ***polynom*** nebo celá racionální funkce. Čísla $a_0, a_1, \ldots, a_n$, se nazývají ***koeficienty polynomu*** $P(x)$. ***Stupněm polynomu*** $P(x)$ nazýváme nejvyšší mocninu proměnné x ve výrazu (1), u níž je nenulový koeficient. Je-li v (1) $a_n \neq 0$, pak P(x) je n-tého stupně.

Příklad: $P(x) = 3x^4 + 2x^3 - x^2 + x + 4$ --> n + n-1 + ... + 1 ==> $O(n^2)$

Výpočet hodnoty polynomu v bodě x

Prostá implementace funkce vyžaduje přímý výpočet pomocí funkce, která počítá x^N. Tento přístup potřebuje kvadratický čas $O(n^2)$. Méně prostoduchá implementace vyžaduje ukládání hodnot x^i v tabulce a pak je použije pro přímý výpočet. Tento přístup vyžaduje lineární prostor navíc.

Efektivní algoritmus snižující na minimum počet násobení je známý jako ***Hornerovo schéma***. Polynom stupně N tak může být vyhodnocen jen užitím *N-1* operací násobení a *N* operací sčítání.
Hornerův algoritmus je přímý, optimální, lineární algoritmus založený na využití závorek.

$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = (((\ldots.(a_n x + a_{n-1})x + \ldots + a_1)x + a_0$

$3x^4 + 2x^3 - x^2 + x + 4 = (((3x + 2)x - 1)x + 1)x + 4$
$P(2) = 66$

Hornerovo schéma:

koeficienty	3	2	-1	1	4
		6	16	30	62
v bodě 2	3	8	15	31	66

Příklad: Výpočet polynomu v zadaném bodě.
Následující funkce předpokládá, že polynom je v paměti počítače reprezentován strukturou, která obsahuje stupeň polynomu a pole koeficientů polynomu. Funkce vrací hodnotu polynomu v bodě daném parametrem x.

```
#include <stdio.h>
#define N 4
typedef struct poly
{
  int degree;          // stupeň polynomu
  double coef[N + 1]; // koeficienty polynomu
}Tpoly;

double evalHorner (Tpoly *polynom, double x)
{
  double sum = 0.0;
  for (int i = 0; i <= polynom->degree; i++)
    sum = sum * x + polynom->coef[i];
  return sum;
}
int main (void)
{
  Tpoly myPoly = { 4, {3.0, 2.0, -1.0, 1.0, 4.0} };
  double x = 2.0;  // inicializace
  printf ("Hodnota polynomu v bode %f je %f \n",x,evalHorner (&myPoly, x));
  return 0;
}
```

Důležitá poznámka: Na stejném principu funguje vyčíslení hodnoty čísla zapsaného v obecné číselné soustavě. V tomto případě fungují jednotlivé číslice vstupního čísla jako koeficienty polynomu a základ číselné soustavy jako bod, ve kterém se má polynom spočítat.

$$(2352)_6 = 2\times6^3 + 3\times6^2 + 5\times6^1 + 2\times6^0 = ((2\times6+3)\times6+5)\times6+2 = (572)_{10}$$

13.4 Řešení nelineárních rovnic

Hledání reálných kořenů rovnice *P(x) = 0*, kde *P(x)* je polynom:
$P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$

Hledáme reálné číslo *α* pro které platí *f(α) = 0*; *α* je kořen rovnice *f(x) = 0*. Hledání komplexních kořenů obecné rovnice *f(x)= 0* se vyskytuje poměrně zřídka. Numericky (jak známo) řešíme úlohy v případě, kdy nelze nalézt řešení analyticky, ale lze vymyslet efektivní algoritmus (dnes prakticky vždy implementovaný na počítači), pro řešení této úlohy. V případě nelineární rovnice, o který se zde zajímáme, musíme hledat metody, které vedou k přibližnému řešení. Při určování kořene rovnice je u některých metod požadováno, aby byl separován kořen rovnice (tj. aby byl určen interval, ve kterém leží jediný kořen). Separaci lze provést několika způsoby.

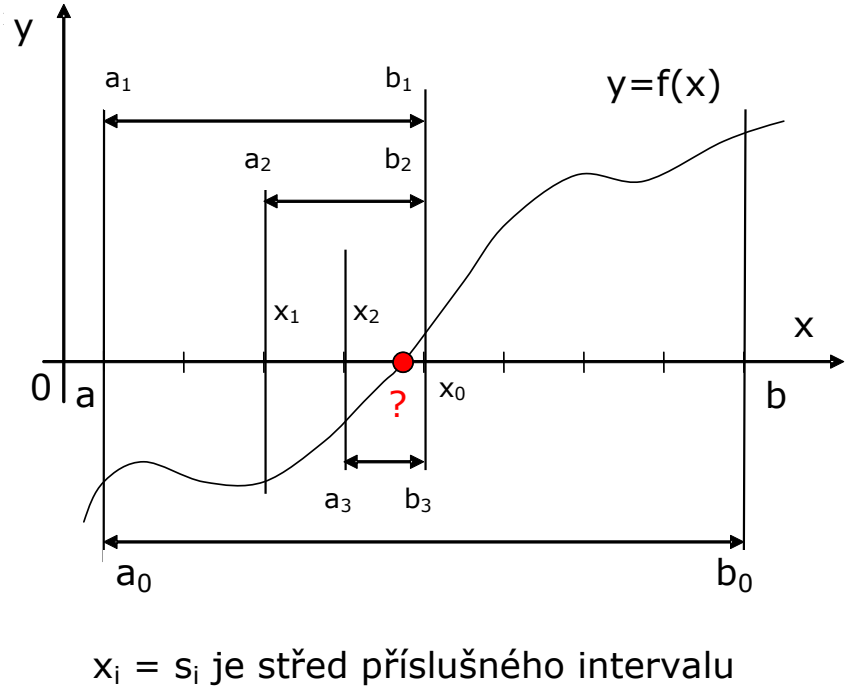
Můžeme např. vyšetřit průběh funkce a z funkce *f(x)* spočítat první a druhou derivaci. Z průběhů derivací lze zjistit, ve kterých intervalech je funkce rostoucí a klesající a vyšetřit pak lokální minima a maxima. Je důležité si uvědomit, že reálné kořeny rovnice jsou průsečíky grafu funkce a osy *x*. Zjištěné intervaly potom použijeme pro přibližný výpočet kořenů. Na počítači provádíme většinou separaci interakčně na základě znalosti grafického průběhu funkce *f(x)* nebo *P(x)* („grafická“ separace kořenů rovnice) neboť zjišťování průběhů derivací je mnohdy obtížné.
Více o této problematice se dozvíte např. na:
http://www.slu.cz/math/cz/knihovna/ucebni-texty/Numericke-metody/Numericke-metody.pdf
[on line, cit. 2018-11-19]

13.4.1 Metoda půlení intervalu (bisekce)

Jedná se o vždy konvergentní a dá se říci univerzální metodu, která se dá použít ve většině praktických případů. Aby bylo možno metodu užít, musí být splněny dvě podmínky. První je požadavek, aby funkce f byla spojitá pro $\forall x \in I_0 = \langle a_0, b_0 \rangle$.

Druhou podmínkou je, aby funkční hodnoty v krajních bodech zvoleného intervalu měly opačná znaménka tj. musí platit $f(a_0)\, f(b_0) < 0$. Pokud jsou obě podmínky splněny, pak tato metoda vždy konverguje.

Polohu kořene zjišťujeme rozpůlením intervalu $\langle a_i, b_i \rangle$ a zjištěním, ve které části kořen leží. Zmenšený interval, v němž leží kořen lze dále rozpůlit a tak zvyšovat přesnost výpočtu. Střed posledního sestrojeného intervalu lze považovat za aproximaci kořene řešené rovnice. Z tohoto postupu je patrné, že čím více kroků provedeme, tím přesnější dostaneme výsledek.



Obrázek 1.: Princip metody půlení intervalu.

Příklad: Funkce pro hledání kořene rovnice metodou půlení intervalu.

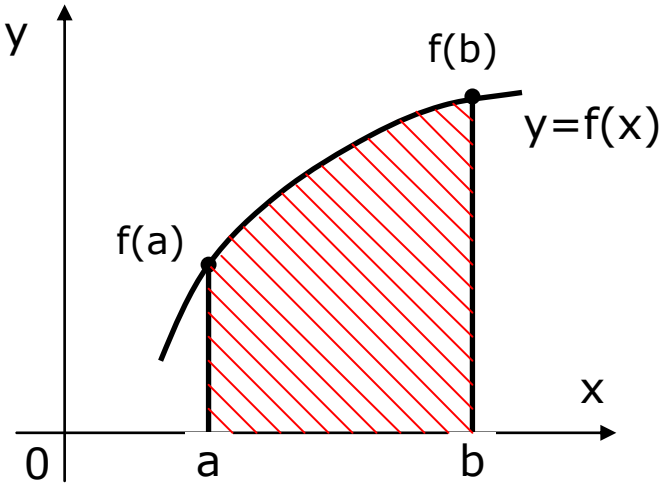
```
double root_equation (double a, double b,
    double eps,double (*evalFun) (double))
{
  double middle = (a + b) / 2;
  double fmid = evalFun(middle);
  while (fabs(fmid) > eps) {
    if (evalFun(a) * fmid < 0)
      b = middle;
    else
      a = middle;
    if (fabs (fmid) > eps) {
      middle = (a + b) / 2;
      fmid = evalFun(middle);
    }
  }
  return middle;
}
```

13.5 Numerický výpočet určitého integrálu

Jestliže je funkce *f(x)* spojitá v uzavřeném intervalu <*a,b*> a známe-li její primitivní funkci *F(x)*, můžeme vypočítat určitý integrál funkce *f(x)* v mezích od *a* do *b* pomocí vztahu:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Metody numerického integrování užíváme v takových případech, kdy je obtížné najít funkci *F(x),* nebo v případě, že funkce *f(x)* je dána tabulkou. Princip numerické integrace spočívá ve výpočtu určitého integrálu pomocí hodnot integrované funkce v řadě bodů.
Hodnota určitého integrálu v uzavřeném intervalu <*a,b*> představuje plochu křivočarého lichoběžníku omezeného zleva dolní mezí *a*, zprava horní mezí *b,* zdola osou *x* a shora funkcí *f(x).*

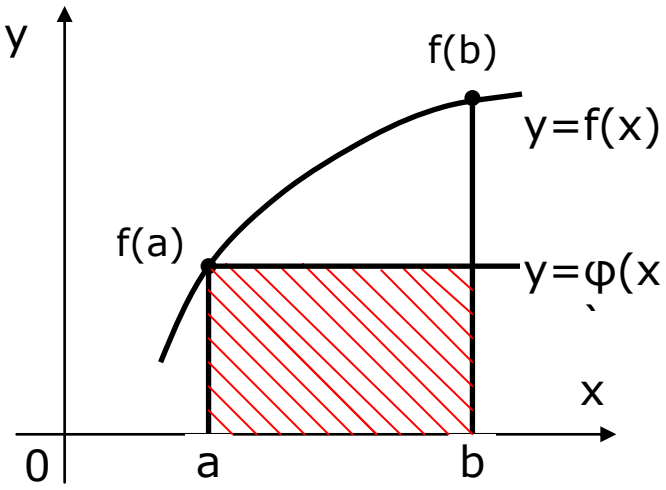


Obrázek 2.: Určitý integrál v uzavřeném intervalu <a,b>.

Vzorce pro výpočet se dají odvodit jednoduše na základě geometrické interpretace určitého integrálu (plocha pod křivkou v daném intervalu).

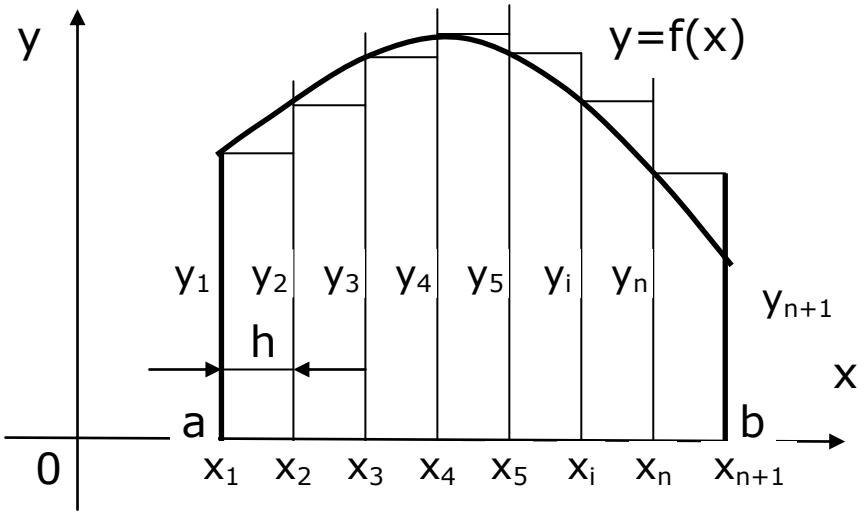
13.5.1 Obdélníková metoda (pravidlo)

Nejjednodušší numerickou integrací funkce *f(x)* je určení plochy pod aproximační funkcí *φ(x)* danou obdélníkem.



Obrázek 3.: Obdélníková metoda.

Rozdělíme danou plochu na obdélníky, kde *n* je počet dílčích intervalů a *n+1* značí počet souřadnic *x* v intervalu <*a,b*>.



Obrázek 4.: Princip obdélníkové metody.

Vzorec pro výpočet: $\int_a^b f(x)dx \approx \sum_{i=1}^{n} h_i y_i = h\sum_{i=1}^{n} y_i$,

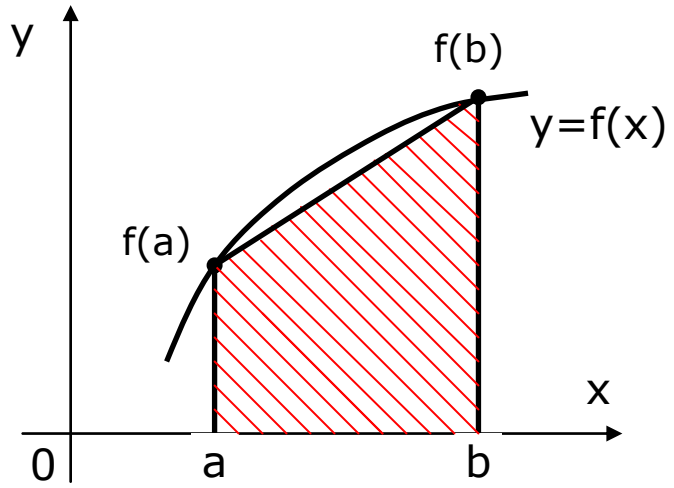
kde šířka dílčího intervalu $h = (b-a)/n$.

Příklad: Funkce pro výpočet integrálu obdélníkovou metodou.

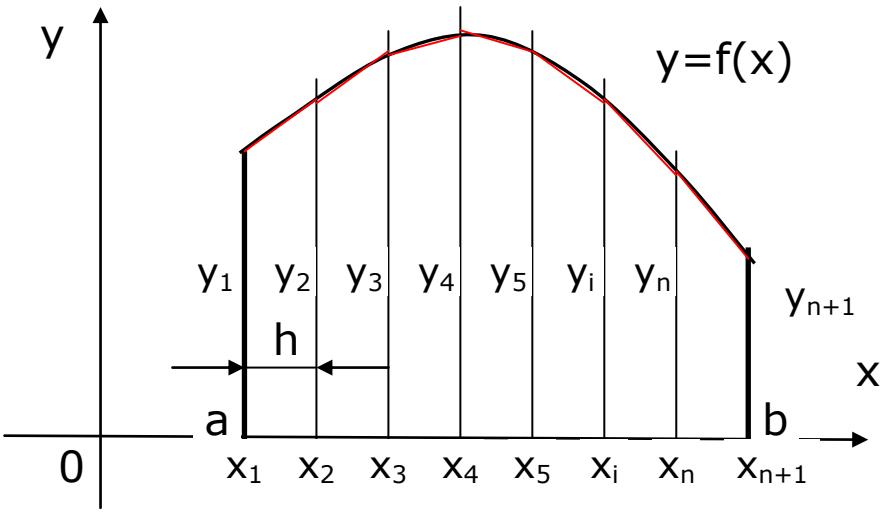
```
double integrate_rectangle (double a, double b, int n,
                            double (*evalFun)(double))
{
  double step, sum = 0.0;
  step = (b - a) / n;
  for (double x = a; x < b - (step/2); x += step)
    sum += evalFun (x);
  sum *= step;
  return sum;
}
```

13.5.2 Lichoběžníková metoda (pravidlo)

Přesnější integraci funkce *f(x)* získáme použitím lichoběžníků, kterými aproximujeme danou funkci.



Obrázek 5.: Lichoběžníková metoda.



Obrázek 6.: Princip lichoběžníkové metody.

Plochu omezenou lichoběžníkem určíme ze vzorce:

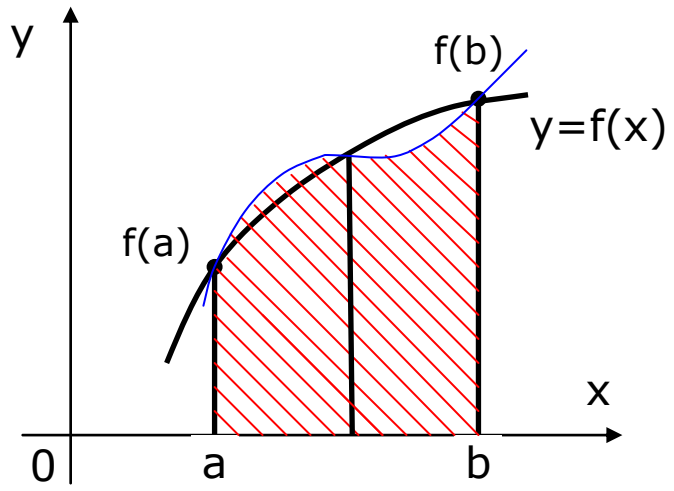
$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n}\frac{y_n + y_{n+1}}{2} = h(y_1/2 + y_{n+1}/2 + \sum_{i=2}^{n} y_i)$$

Příklad: Funkce pro výpočet integrálu lichoběžníkovou metodou.

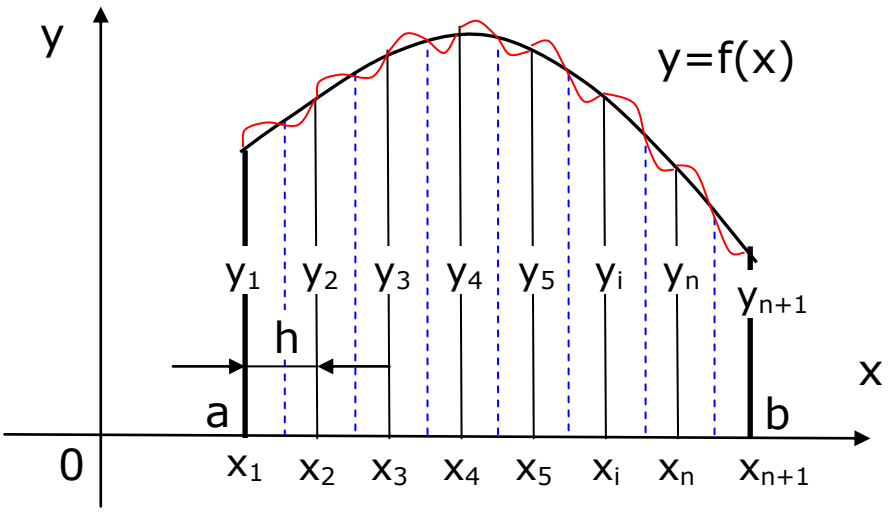
```
double integrate_trapezoid (double a, double b, int n,
                            double (*evalFun)(double))
{
  double step, sum = 0.0;
  step = (b - a) / n;
  for (double x = a + step; x < b - step; x += step)
    sum += evalFun (x);
  sum += (evalFun (a) + evalFun (b)) / 2;
  sum *= step;
  return sum;
}
```

13.5.3 Simpsonova metoda (pravidlo)

Při konstantní hodnotě šířky dílčích intervalů se velmi často kvůli přesnosti používá ***Simpsonova metoda***, při které je nezbytné dodržet podmínku, že počet dílčích intervalů musí být sudé číslo. Při této metodě se vždy tři sousední body na křivce *f(x)* aproximují vhodnou parabolou.



Obrázek 7.: Simpsonova metoda.



Obrázek 8.: Princip Simpsonovy metody.

Plochu omezenou dílčími parabolami určíme ze vzorce:

$$\int_a^b f(x)\ dx \approx \frac{h}{3}(y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \ldots + 2y_{n-1} + 4y_n + y_{n+1})$$

U Simpsonovy metody musíme dbát na správné přiřazení násobících součinitelů. U sudých indexů funkčních hodnot y_i je násobící koeficient roven *4* a u lichých indexů (kromě prvního a posledního) je násobící koeficient roven *2*! Pro určení sudé nebo liché hodnoty indexů funkčních hodnot použijeme v programu s výhodou logický operátor *(n & 1)* nebo dělení modulo.

Příklad: Funkce pro výpočet integrálu Simpsonovou metodou.

```
double integrate_simpson (double a, double b, int n,
                          double (*evalFun)(double))
{
  int c,tmp = 1;
  double step, sum = 0.0;
  step = (b - a) /(2 * n);
  for (double x = a + step; x < b - (step/2); x += step)
  {
    tmp++;
    c = (tmp&1)?2:4; // lichý/sudý bod?
    sum += c * evalFun (x);
  }
  sum += evalFun (a) + evalFun (b);
  sum *= step/3;
  return sum;
  }
```

Příklad: Použití funkcí pro numerický výpočet určitého integrálu.

$$\int (x^2+1)dx = \frac{x^3}{3} + x + C \qquad \int_a^b f(x)dx = F(b) - F(a)$$

$$\int_0^1 (x^2+1)dx = 4/3$$

Obrázek 9.: Průběh funkce $x^2 + 1$ v intervalu <0,1>.

```
double parabola (double argument)
{
  return argument * argument + 1;
}
int main (void)
{
  int n,lower_limit=0;
  int upper_limit=1;
  double integral_old, integral_new = 0.0;
  // zadání hodnoty n - počáteční počet dělení úseku
  // zadání faktoru násobení – jak se má zvyšovat
  // počet dílčích úseků
  // zadání hodnoty eps - přesnost výpočtu
  do{
    integral_old = integral_new;
    integral_new = integrate_rectangle (lower_limit,
                              upper_limit,n,parabola);
    n = n * factor;
  } while (fabs(integral_new - integral_old) > eps);
  // výpočet s hodnotou integrálu
  return 0;
}
```

Experimentální výsledky

n	obdélníková	lichoběžníková	Simpsonova
10	1.285000	1.335000	1.333333
100	1.328350	1.333350	1.333333
1000	1.332834	1.333334	1.333333
10000	1.333283	1.333333	1.333333
100000	1.333328	1.333333	1.333333
přesná	1.333333	1.333333	1.333333

13.6 Shrnutí

Tato kapitola je ryze praktická a seznamuje čtenáře s množinou běžně používaných algoritmů pro numerické výpočty.

Čtenář je nyní schopen popsat a používat algoritmus zvaný Hornerovo schéma. Umí pomocí něj vyčíslit polynom nebo hodnotu čísla v zadané číselné soustavě.

Čtenář je schopen definovat úlohu hledání kořene nelineární rovnice. Tento problém je schopen vyřešit metodou půlení intervalu. Tento algoritmus je schopen zapsat pomocí jazyka C s využitím datového typu ukazatel na funkci. Díky tomu je schopen algoritmus naprogramovat obecně pro libovolnou nelineární funkci.

Čtenář je schopen naprogramovat v jazyce C výpočet určitého integrálu pomocí obdélníkové, lichoběžníkové a Simpsonovy metody. Tyto implementace dokáže v jazyce C realizovat obecně pro libovolnou funkci, která se má integrovat. K tomu je schopen opět využít parametr typu ukazatel na funkci.

13.7 Úlohy k procvičení

1. V jazyce C napište funkci pro výpočet hodnoty polynomu pomocí Hornerova schématu.
2. V jazyce C napište funkci pro hledání kořene rovnice metodou půlení intervalu.
3. V jazyce C napište funkci pro výpočet integrálu obdélníkovou metodou.
4. V jazyce C napište funkci pro výpočet integrálu lichoběžníkovou metodou.
5. V jazyce C napište funkci pro výpočet integrálu Simpsonovou metodou.

13.8 Kontrolní otázky

1. Co je Hornerovo schéma? Vysvětlete princip implementace funkce pro výpočet hodnoty polynomu pomocí Hornerova schématu.
2. Vysvětlete algoritmus pro hledání kořene rovnice metodou půlení intervalu.
3. Vysvětlete algoritmus pro výpočet integrálu obdélníkovou metodou.
4. Vysvětlete algoritmus pro výpočet integrálu lichoběžníkovou metodou.
5. Vysvětlete algoritmus pro výpočet integrálu Simpsonovou metodou.