Dokumentace k projektu pro předměty IUS & IZP

Iterační výpočty

Projekt č. 2

Autor: Jan Kaláb (xkalab00@stud.fit.vutbr.cz)

Úvod

Úkolem bylo napsat v jazyce C program sloužící k výpočtům matematických funkcí druhé odmocniny, sinu, mocniny Eulerova čísla, přirozeného logaritmu a některých jejich kombinací (druhá odmocnina z mocniny Eulerova čísla, ...) pomocí základních matematických operací (sčítání, odečítání, násobení a dělení).

Dokument postupně rozebírá jednotlivé funkce a popisuje jejich implementaci. V poslední kapitole jsou informace o použití programu.

Druhá odmocnina

Druhá odmocnina je jedna ze základních matematických funkcí. Její definiční obor v množině reálných čísel a obor hodnot jsou všechna kladná reálná čísla včetně nuly.

Pro její výpočet existuje několik aproximačních metod, v programu je použita tzv. Babylonská. Tato metoda je odvozena z Newtonovy metody, avšak je starší.

$$x_0 \approx \sqrt{S}$$
$$x_{n+1} = \frac{1}{2}(x_n + \frac{S}{x_n})$$
$$\sqrt{S} = \lim x_n$$

Z výše uvedeného vzorce vyplývá celý algoritmus:

1 výsledek je zadané číslo
2 zpřesníme výsledek
  2.1 podělíme zadané číslo předchozím výsledkem
  2.2 přičteme předchozí výsledek
  2.3 podělíme dvěma
3 krok 2 opakujeme dokud nedosáhneme požadované přesnosti ε.

V praxi vypadá kód v jazyce C takto:

```
double druhaodmocnina (double neznama, double eps) {
  double y = neznama;
  if (y == 0) return 0;
  double last = 0;
  if (neznama > 0) {
    do {
      last = y;
      y = (neznama / last + last) / 2;
    } while (fabs (y – last) >= eps);
  } else {
    fprintf (stderr, "Odmocninu lze vypocitat jen z kladnych cisel!\n");
    return NAN;
  }
  return y;
}
```

Sinus

Sinus je periodicky se opakující (po 2π) goniometrická funkce. Definičním oborem jsou jí všechna reálná čísla, oborem hodnot pak interval <–1; 1>. Pro aproximaci její hodnoty lze dobře použít Taylorovu řadu:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Tu lze upravit do rekurentního vztahu:

$$y_n = \frac{-y_{n-1} * x^2}{k * k - 1}$$

y_n je aktuální člen člen, y_{n-1} je předchozí člen, x je hodnota pro kterou chceme vypočíst sinus, k se po každém kroku navýší o 2, takže $k * k - 1$ je rekurentní vztah pro faktoriál ob číslo. Takto počítáme dokud není člen menší nebo roven zadané přesnosti ε.
V jazyce C pak funkce sinus vypadá třeba takto:

```
double sinus (double neznama, double eps) {
  short neg = 0;
  if (neznama < 0) {
    neznama = fabs (neznama);
    neg++;
  }
  if (neznama > IZP_2PI) neznama = fmod (fabs (neznama), IZP_2PI);
  double neznama2 = pow (neznama, 2);
  double y = neznama;
  double prirustek = neznama;
  unsigned short k = 3;
  while (fabs (prirustek) >= eps) {
    prirustek = (-prirustek/(k * (k - 1))) * neznama2;
    k += 2;
    y += prirustek;
  }
  if (neg != 0) y *= -1;
  if (y > 1 || y < -1) {
    fprintf (stderr, "Vysledek je mimo rozsah <-1; 1>!\n");
    return NAN;
  }
  return y;
}
```

Za zmínku stojí řádek `if (neznama > IZP_2PI) neznama = fmod (fabs (neznama), IZP_2PI);` (`IZP_2PI` je konstanta pro 2π). Taylorův rozvoj totiž nejrychleji konverguje v intervalu 0–2π. Do toho intervalu se nejrychleji přesuneme tak, že zadané číslo podělíme 2π a použijeme zbytek po dělení.

Pokud umíme spočítat sinus, není problém ani kosinus. Stačí pouze k neznámé na začátku přičíst π / 2.

No a pokud umíme sinus i kosinus, zvládneme spočítat i tangens (sinus / kosinus) a kotangens (kosinus / sinus). Je ovšem třeba si uvědomit, že jejich definičním oborem nejsou všechna reálná čísla.

Mocnina Eulerova čísla

Mocnina Eulerova čísla se chová jako každá jiná exponenciální funkce, jejím základem je Eulerovo číslo (e ≈ 2,71828182845904523536...). Pro její výpočet je opět použita Taylorova řada:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Z ní lze opět dovodit rekurentní vztah:

$$y_n = \frac{y_{n-1} * x}{k}$$

y_n je aktuální člen, y_{n-1} je předchozí člen, x je hodnota pro kterou chceme vypočíst mocninu, k číslo kroku. Takto počítáme dokud není člen menší nebo roven zadané přesnosti ε.
V jazyce C lze zapsat takto:

```
double enax (double neznama, double eps) {
  short neg = 0;
  if (neznama < 0) {
    neznama = fabs (neznama);
    neg = 1;
  }
  double y = neznama + 1;
  double prirustek = neznama;
  unsigned short k = 2;
  while (fabs (prirustek) >= eps && y != INFINITY && prirustek != INFINITY) {
    prirustek *= (neznama / k);
    k++;
    y += prirustek;
  }
  if (y == INFINITY || prirustek == INFINITY) {
    fprintf (stderr, "Preteceni!\n");
    return NAN;
  }
  if (neg != 0) y = 1 / y;
  return y;
}
```

Problémem je prudký růst exponenciálních funkcí. Výše uvedená implementace dokáže počítat v intervalu ±709, pak dochází k přetečení výsledku z rozsahu double.

Přirozený logaritmus

Přirozený logaritmus je funkce logaritmu s Eulerovým číslem jako základem. Definičním oborem funkce je množina všech kladných reálných čísel včetně 0 (ln (0) = –nekonečno). Pro její výpočet existuje opět Taylorova řada:

$$\ln(x) = 2y\left(1 + \frac{y^2}{3} + \frac{y^4}{5} + \frac{y^6}{7} + \frac{y^8}{9} + \ldots\right)$$
$$y = \frac{x-1}{x+1}$$

Z ní se odvodí rekurentní vztah:

$$y_n = \frac{y_{n-1} * k * z^2}{k+2}$$

y_n je aktuální člen, y_{n-1} je předchozí člen, k je „číslo kroku“ (v každém kroku se navýší o 2), z je y z prvního vzorce. Takto počítáme dokud není člen menší nebo roven zadané přesnosti ε.
Zápis v jazyce C:

```
double prirozenylogaritmus (double neznama, double eps) {
  if (neznama == 0) return –INFINITY;
  if (neznama < 0){
   fprintf (stderr, "Prirozeny logaritmus lze pocitat jen z cisel vetsich nez 0!\n");
   return NAN;
  }
  int mocnina = 0;
  while (neznama >= 2) {
    neznama = neznama / 10;
    mocnina++;
  }
  while (neznama <= .1 && neznama > 0) {
    neznama = neznama * 10;
    mocnina––;
  }
  double z = (neznama – 1) / (neznama + 1);
  double zz = z * 2;
  double z2 = pow(z, 2);
  double y = zz;
  unsigned int k = 1;
  double prirustek = y;
  while (fabs (prirustek) >= eps && prirustek != INFINITY && y != INFINITY) {
    prirustek *= ((k * z2) / (k + 2));
    k += 2;
    y += prirustek;
  }
  if (mocnina != 0) y += (mocnina * log (10));
  return y;
}
```

Problémem tohoto Taylorova rozvoje je opět konvergence – nejrychleji konverguje kolem čísla 1. K tomu lze použít pravidlo o dělení logaritmů.

$$\log(123{,}456) = \log(1{,}23456 * 10^2) = \log(1{,}23456) + \log(10^2) = \log(1{,}23456) + 2 * \log(10)$$

K tomu slouží první dvě `while` smyčky, první pro velká čísla, druhá pro malá.

Složené funkce

Program také dokáže počítat složené funkce:

- Druhou odmocninu z mocniny Eulerova čísla
- Sinus z přirozeného logaritmu

Není třeba v tom hledat nějaké složitosti. Nejdříve vypočteme hodnotu vnitřní funkce se zadanou přesností, poté z výsledku vypočteme vnější funkci se stejnou přesností. Pokud jsou funkce naprogramovány dostatečně efektivně, je výpočet rychlý a přesný.

Použití programu

Program je psán v jazyce C podle standardu C99. Pro jeho přeložení použijte příkaz `gcc -std=c99 -Wall -W -lm -pedantic proj2.c -o proj2 -O2` (nemělo by se vypsat žádné varování).

Program se poté spouští příkazem `./proj2 -FUNKCE ε` (pro seznam funkcí a informace o ε si parametrem `-h` nebo spuštěním bez parametrů zobrazte nápovědu).

Program pak na standardním vstupu očekává zadání pro výpočet.

Použité zdroje

- http://cs.wikipedia.org/wiki/Taylorova_%C5%99ada#P.C5.99.C3.ADklady_Taylorova_rozvoje
- http://en.wikipedia.org/wiki/Natural_logarithm
- http://en.wikipedia.org/wiki/Trigonometric_function

Metriky kódu

1 soubor o 239 řádcích, po překladu dle dokumentace 9,8 kB·(10046·bytů), z toho 316 bytů jsou statická data.