30. 11. 2024

12. Iterační metody pro řešení soustav n lineárních rovnic o n neznámých

↳ pracuji se zadanou přesností (ε) ⇒ ZPŘESŇOVÁNÍ

Numerická metoda

= algoritmus popisující cestu k řešení numerické úlohy

- numerické matematiky (+; -; *; /)
  ↳ řešení problémy pro konkrétní číselné hodnoty
  ↳ obrovské množství výpočtů ⇒ přišlo s využitím počítačů

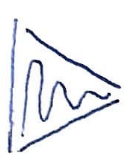
příklady:

• Výpočty

- interpolace – spojení

- aproximace – přímka co nejblíže všem naměřeným bodům

- derivace určitých integrálů



• Řešení soustav rovnic

- lineární
- nelineární
- obyčejných diferenciálních
- parciálních diferenciálních

Problém: PŘESNOST

- typ double uchovává jen cca 15 desetinných míst (dáno velikostí mantisy)

  double x = 1e16, y = x+1;

  x == y, protože jsme přesáhli limit double a číslo je tak velké, že přičtení +1 nemá vliv (přetečení typu)

- v počítači nelze obecně uložit všechna reálná čísla (Pi; E; 0,1)
- 0.1 je ve dvojkové soustavě periodické číslo !

  nekonečný cyklus: for(double x = 0.0; x != 10,0; x += 0.1) {}

  ↳ hodnota bude přeskočena <= lepší!

- každá hodnota v sobě obsahuje potenciální chybu
- * a / zvyšují celkovou chybu výpočtu
- Závěr: Numerické metody musí řešit (minimalizovat) i problémy s přesností výpočtu

Iterační výpočet

↳ REKURENTNÍ VZTAH

$$Y_{i+1} = F(Y_i, Y_{i-1}, Y_{i-2}, \dots, Y_{i-k})$$

$$Y_{i+1} = F(Y_i)$$

Y_i – rekurentní proměnná

F – je funkce, která určuje hodnoty $Y_i, \dots, Y_{i-k}$ další hodnotu Y_{i+1}

$k \geq 0$ je počet předchozích hodnot řešení

- pro výpočet další hodnoty potřebujeme $k+1$ předchozích hodnot
- musí existovat takové n, že Y_n je přibližnou hodnotou
  ↳ musíme umět rozpoznat přibližnou hodnotu

- Iterační výpočet funguje na principu ZPŘESŇOVÁNÍ
  ↳ končí po dosažení zadané přesnosti (ε)

Konvergence

= vlastnost posloupnosti (řady)

- prvky řady se musí blížit cílové hodnotě
- stabilní vývoj, který vede k výsledku

$|Y_i - Y_{i-1}|$ se od určitého členu i musí limitně blížit nule

↳ abs. z rozdílu prvků $\neq$ ~~$|Y_i| - |Y_{i-1}|$~~

- zmíněno u DDM (u ostatních neznámým)
  ↳ TESTOVAT !

Algoritmické schéma výpočtu

↳ obecné schéma výpočtu celé třídy podobných úloh

$$\boxed{\begin{array}{l} Y = y_0 \\ \text{while}(\neg B(Y)) \\ \quad Y = F(Y) \end{array}}$$

inicializace !

• Y – rekurentní proměnná

B – predikát (neboli podmínka dosažení ukončení hodnoty Y)

F – funkční symbol (funkce

- iteruji (konvergují) v cyklu dokud nepoznám výsledek

$$|Y_i - Y_{i-1}| < \varepsilon$$

↳ v ABS. hodnotě, co se od čeho odečítá (IMPLEMENTAČNÍ POZNÁMKA)

Stabilita (výpočtu)

↳ stabilní systém

⇒ malá změna vstupní hodnoty = malá změna výstupu

- opakem je např. HASHOVÁNÍ

- při malé chybě v iteračních metodách se nic neděje
  ↳ bude to jen trvat více iterací dosáhnutí dané přesnosti

- VELKÁ chyba hodnoty ⇒ DIVERGENCE

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

ABSOLUTNÍ ČLENY

Jacobiho a Gauss-Seidelova (GS) metody

11.11.2024

$$\left(\begin{array}{cc|c} 2.5 & 1.1 & 2.4 \\ 5.5 & -6.5 & 3.1 \end{array}\right) \sim \begin{array}{l} 2.5x_1 + 1.1x_2 = 2,4 \\ 5.5x_1 - 6.5x_2 = 3.1 \end{array}$$

DDM – ANO (nutné kontrolovat předtím, dostaneme je DDM)
↳ DDM → diag. musí být vždy větší ⇒ součet prvků na řádku (v ABS.) kromě ABSOLUTNÍCH členů < diagonální prvek

REKURENTNÍ VZOREC

$$x_1 = \frac{2.4 - 1.1x_2}{2.5} \qquad x_2 = \frac{3.1 - 5.5x_1}{-6.5}$$

OBECNĚ

$$x_n = \frac{b_n - \sum_{i=0,\ i\neq n}^{n-1} (a_{ni}\cdot x_i)}{a_{nn}}$$

Optimalizace

$$\begin{pmatrix} 2.5 & 1.1 & 2.4 \\ 5.5 & -6.5 & 3.1 \end{pmatrix} \begin{array}{l} /2.5 \\ /-6.5 \end{array} \text{ (dělení diag. prvkem)}$$

$$\begin{pmatrix} 1 & 0,44 & 0,96 \\ -0,85 & 1 & -0,48 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0,44 & 0,96 \\ -0,85 & 0 & -0,48 \end{pmatrix}$$

NAHRAZENÍ 1 za 0

každá rovnice teď počítá pouze se samostatnou hodnotou x (tedy bez závislosti na vlastním x_i)
↳ původně by každý x_i závisel na své vlastní hodnotě v každé iteraci

OBECNĚ

$$x_n = b_n - \sum_{i=0}^{n-1} (a_{ni}\cdot x_i)$$

osamostatněný DIAG. PRVEK

$$\begin{pmatrix} 0_{x_1} & 0,44 & 0,96 \\ -0,85 & 0_{x_2} & -0,48 \end{pmatrix} \qquad \begin{cases} \boxed{x_1} = 0,96 - 0,44\cdot x_2 \\ \boxed{x_2} = -0,48 + 0,85\cdot x_1 \end{cases}$$

ε (chyba) = 0,5
↳ přesnost

Jacobiho (2 sloupce výsledků: x prev, x new)

↳ při výpočtu nových hodnot používáme POUZE STARÉ HODNOTY Z PŘEDCHOZÍ ITERACE

– PARALIZACE!

Iterace	Previous	New
0	$x_0^{(0)}=0$, $x_1^{(0)}=0$	— —
1	$x_0^{(0)}=0$ $x_1^{(0)}=0$	$x_0^{(1)} = 0,96 - 0,44\cdot x_1^{(0)} = 0,96 - 0,44\cdot 0 = 0,96$ $x_1^{(1)} = -0,48 + 0,85\cdot x_0^{(0)} = -0,48 + 0,85\cdot 0 = -0,48$
2	$x_0^{(1)} = 0,96$ $x_1^{(1)} = -0,48$	$x_0^{(2)} = 0,96 - 0,44x_1^{(1)} = 0,96 - 0,44\cdot(-0,48) = 1,2$ $x_1^{(2)} = -0,48 + 0,85\cdot x_0^{(1)} = -0,48 + 0,85\cdot 0,96 = 0,33$
3	$x_0^{(2)} = 1,2$ $x_1^{(2)} = 0,33$	$x_0^{(3)} = 0,96 - 0,44x_1^{(2)} = 0,96 - 0,44\cdot 0,33 = 0,81$ $x_1^{(3)} = -0,48 + 0,85\cdot x_0^{(2)} = -0,48 + 0,85\cdot 1,2 = 0,51$

Vždy musím brát horší přesnost pro hodnocení

Iterace	ε (chyba, přesnost) TESTOVÁNÍ
1	$\lvert x_0^{(1)} - x_0^{(0)}\rvert = 0,96$ ✗ NE $\lvert x_1^{(1)} - x_1^{(0)}\rvert = 0,48$
2	$\lvert x_0^{(2)} - x_0^{(1)}\rvert = 0.2$ $\lvert x_1^{(2)} - x_1^{(1)}\rvert = 0,81$ ✗ NE
3	$\lvert x_0^{(3)} - x_0^{(2)}\rvert = 0,35 < 0,5$ $\lvert x_1^{(3)} - x_1^{(2)}\rvert = 0,18$

↳ DOSAŽENO PŘESNOSTI ⇒ KONEC

– snížení operací — Gauss-Seidelova (1 sloupec výsledků) stačí paměť

↳ používáme při každém výpočtu AKTUÁLNĚ SPOČÍTANÉ HODNOTY
↳ dosazujeme

Iterace 0
– výchozí hodnoty $x_1^{(0)} = 0$; $x_2^{(0)} = 0$

Iterace 1 ← implementační poznámka
– výpočet $x_1^{(1)}$: $x_1^{(1)} = 0,96 - 0,44\cdot x_2^{(0)} = 0,96 - 0,44\cdot 0 = 0,96$
– výpočet $x_2^{(1)}$: $x_2^{(1)} = -0,48 + 0,85\cdot x_1^{(1)} = -0,48 + 0,85\cdot 0,96 = 0,33$
CHYBA: $\max\left(\lvert x_1^{(1)} - x_1^{(0)}\rvert, \lvert x_2^{(1)} - x_2^{(0)}\rvert\right) = 0,96$ $0,96 > 0,5$

Iterace 2
– výpočet $x_1^{(2)}$: $x_1^{(2)} = 0,96 - 0,44\cdot x_2^{(1)} = 0,96 - 0,44\cdot 0,33 = 0,81$
– výpočet $x_2^{(2)}$: $x_2^{(2)} = -0,48 + 0,85\cdot x_1^{(2)} = -0,48 + 0,85\cdot 0,81 = 0,51$
CHYBA: $\max\left(\lvert x_1^{(2)} - x_1^{(1)}\rvert, \lvert x_2^{(2)} - x_2^{(1)}\rvert\right) = 0,1775 < 0,5$

DOSAŽENÍ PŘESNOSTI
KONEC

Iterace	x_1	x_2	CHYBA
1	$x_1^{(1)} = 0,96$	$x_2^{(1)} = 0,33$	0,96
2	$x_1^{(2)} = 0,81$	$x_2^{(2)} = 0,51$	0,1775

Algoritmus implementace Optimalizace

díky NULOVÁNÍ DIAGONÁLY jsme schopni dostat novou hodnotu x_n velmi jednoduše (klíčové pro tvorbu algoritmu při průchodu iterací)

TIP: Test konvergence jde provádět i na upravené matici soustavy. Poté jde zjednodušit i samotný test, když se sum absolutních hodnot koeficientů každého řádku porovnání přímo s hodnotou 1. Pozor! Pokud bychom tu porovnání prováděli hodnoty diagonálního prvku, když a_{ii}, porovnávali bychom s nulou, což je samozřejmě splněno.

```c
bool jeDDM(Tmatice *m) {
  float sum = 0.0;
  for (int r = 0; r < m->radku; ++r) {
    for (int s = 0; s < m->sloupcu - 1; ++s) {
      if (s != r) { // kromě diagonálního prvku
        sum += m->prvek[r][s];
      }
    }

    if (sum > m->prvek[r][r]) {
      return false;
    }
    sum = 0.0;
  }
  return true;
}
```

if (sum >= 1) { return false;}

```c
void upravaMatice(Tmatice *m) {
  for (int r = 0; r < m->radku; ++r) {
    for (int s = 0; s < m->sloupcu; ++s) {
      if (r != s) {
        m->prvek[r][s] /= m->prvek[r][r]; // vydělíme každý prvek diagonálním prvkem
      }
    }
    m->prvek[r][r] = 0.0;
  }
}
```

plní už funkci nulování diagonály

```c
void nulovaniDiagonaly(Tmatice *m) {
  for (int r = 0; r < m->radku; r++) {
    m->prvek[r][r] = 0.0;
  }
}
```

```c
void testJ(char *adresaSouboru, float eps) {
  printf("------ Jacobiho metoda ------\n");
  FILE* f = fopen(adresaSouboru, "r");
  if (f == NULL) {
    printf("\nChyba při otevírání souboru.\n");
    return;
  }

  Tmatice *m = maticeCtiZeSouboru(f);
  fclose(f);

  if (m == NULL) {
    printf("\nChyba při alokaci matice.\n");
    return;
  }

  Tmatice *x = maticeAlokuj(m->radku, 1);
  if (x == NULL) {
    printf("\nChyba při alokaci výsledkové matice.\n");
    return;
  }
  inicializujMatici(x, 0.0);

  maticeTiskni(m);

  if (!jeDDM(m)) {
    printf("\nMatice není DDM.\n");
    return;
  }

  upravaMatice(m);
  nulovaniDiagonaly(m);
  jacobiho(m, eps, x);

  tiskReseni(x);
  maticeUvolni(x);
  maticeUvolni(m);
}
```