Vaja 3

Učinkovito reševanje tridiagonalnih linearnih sistemov z Gaussovo eliminacijo

Neizčrpen vir linearnih sistemov enačb so sistemi, ki se pojavijo pri numeričnem reševanju navadnih ali parcialnih diferencialnih enačb oziroma robnih problemov. Če hočemo dobiti natančne približke za rešitve robnih problemov, imajo takšni sistemi pogosto preveč spremenljivk, da bi jih lahko učinkovito rešili z običajno Gaussovo eliminacijo oz. LU razcepom. Imajo pa takšni sistemi ponavadi zelo enostavno strukturo, ki jo lahko izkoristimo, da napišemo specializirano obliko algoritma za Gaussovo eliminacijo, ki časovno in prostorsko deluje bistveno učinkovitejše kot Gaussova eliminacija za splošne matrike, kot je implementirana recimo v *octave*-u.

Za zgled si bomo pogledali reševanje naslednjega preprostega robnega problema:

$$\begin{aligned} y''(x) &= f(x) && (1)\\ y(a) &= y_a && (2)\\ y(b) &= y_b && (3) \end{aligned}$$

kjer iščemo rešitev $y(x)$ na intervalu $x \in [a, b]$, $a < b$. Teoretično je zelo enostavno rešiti tak robni problem. Enačbo (1) integriramo in dobimo

$$y'(x) = \int f(x)dx + C$$

Integriramo še enkrat in dobimo splošno rešitev

$$y(x) = \int\int f(x)dxdx + Cx + D,$$

integracijski konstanti C in D pa izračunamo s pomočjo robnih pogojev (2) in (3). V resnici pa ni težko najti že elementarne funkcije $f(x)$, katere integral ne moremo analitično izračunati, recimo $f(x) = e^{-x^2}$ ali $f(x) = \sin(x)/x$, in si potem težko pomagamo s teoretično rešitvijo.

Radi bi imeli način, s katerim lahko poljubno natančno izračunamo rešitev $y(x)$ za poljubno funkcijo $f(x)$, tudi če ni elementarna funkcija, ali pa je za nas celo "črna škatla" (npr. lahko kličemo funkcijo, ki nam vrne vrednost $f(x)$ za poljuben $x \in [a, b]$, definicije te funkcije pa ne poznamo).

To lahko naredimo, tako da prevedemo reševanje diferencialne enačbe na reševanje linearnega sistema enačb z diskretizacijo definicijskega območja.

Razdelimo interval $[a, b]$ na $n + 1$ podintervalov enake dolžine. Krajišča teh podintervalov označimo z x_i, $i = 0, 1, \ldots, n, n + 1$, pri čemer definiramo $x_0 = a$ in $x_{n+1} = b$. Označimo še dolžino enega podintervala z

$$h = \frac{b - a}{n + 1} = x_{i+1} - x_i$$

Namesto, da iščemo vrednosti rešitve $y(x)$ za vse $x \in [a, b]$, bomo zadovoljni s tem, da bomo znali poiskati vrednosti rešitve samo v točkah x_i. To pomeni, da imamo n

neznank $y(x_i)$, ki jih označimo z

$$\begin{aligned} y_i &= y(x_i), \text{ za } i = 1, \ldots, n \\ y_0 &= y_a \\ y_{n+1} &= y_b \end{aligned}$$

Zadnji dve enakosti dobimo iz robnih pogojev (2) in (3). y_0 in y_{n+1} torej nista neznanki, ampak sta podatka.

Če zdaj napišemo enačbo (1) za točke x_i,

$$y''(x_i) = f(x_i),$$

vidimo, da ne bo veliko koristi od diskretizacije, če ne znamo vsaj približno izraziti odvoda $y''(x_i)$ s pomočjo neznank, ki jih imamo na voljo. Ampak če se spomnimo definicije odvoda, opazimo, da lahko odvod funkcije $y(x)$ aproksimiramo vsaj na tri načine:

$$y'(x_i) \doteq \frac{y_{i+1} - y_i}{h} \doteq \frac{y_i - y_{i-1}}{h} \doteq \frac{y_{i+1} - y_{i-1}}{2h}, \text{ za } i = 1, \ldots n$$

Vsaka od teh aproksimacij bo v splošnem dala malce drugačne vrednosti, ampak bo pa v vseh primerih napaka aproksimacije reda h^2 (zakaj?). Opazimo tudi, da lahko prvi dve aproksimaciji interpretiramo kot priblizka za odvod na sredini intervalov $[x_i, x_{i+1}]$ oz. $[x_{i-1}, x_i]$:

$$y'(x_i + \frac{h}{2}) \doteq \frac{y_{i+1} - y_i}{h} \text{ in } y'(x_i - \frac{h}{2}) \doteq \frac{y_i - y_{i-1}}{h},$$

kar pomaga pri aproksimaciji za 2. odvod,

$$y''(x_i) \doteq \frac{y'(x_i + \frac{h}{2}) - y'(x_i - \frac{h}{2})}{h} \doteq \frac{\frac{y_{i+1} - y_i}{h} - \frac{y_i - y_{i-1}}{h}}{h}$$

ali, če poenostavimo zadnji izraz,

$$y''(x_i) \doteq \frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1}) \tag{4}$$

Po (4) lahko zdaj enačbo (1) nadomestimo z linearnim sistemom n enačb

$$\frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1}) = f(x_i), \text{ za } i = 1, \ldots, n$$

z n neznankami $y_1, \ldots, y_n$. V prvi oziroma zadnji enačbi nastopata y_0 oz. y_{n+1}, ki ju lahko prestavimo na desno stran in potem lahko v matrični obliki ta sistem zapišemo kot

$$\begin{bmatrix} -2 & 1 & 0 & & \ldots & 0 \\ 1 & -2 & 1 & 0 & \ldots & \vdots \\ 0 & 1 & -2 & 1 & \ldots & \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ldots & 1 & -2 & 1 \\ 0 & & \ldots & 0 & 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} h^2 f(x_1) - y_a \\ h^2 f(x_2) \\ \vdots \\ \vdots \\ h^2 f(x_{n-1}) \\ h^2 f(x_n) - y_b \end{bmatrix} \tag{5}$$

Kar se tiče algoritma za reševanje sistemov enačb nas desna stran ne zanima, tam sta pač podatka y_a in y_b ter vrednosti $f(x_1), \ldots, f(x_n)$, ki jih lahko izračanumo s klicanjem funkcije f.

Zanima nas matrika na levi strani in vidimo, da ima ta *tridiagonalno* obliko: neničelni elementi ležijo samo na, nad in pod diagonalo.

Tudi pri diskretizaciji splošne navadne diferencialne enačbe 2. reda, ki ima obliko

$$p(x)y''(x) + q(x)y'(x) + r(x) = s(x)$$

za dane funkcije p, q, r in s (kakršno lahko dobite za domačo nalogo) se pojavi tridiagonalen sistem, le da pod, na in nad diagonalo niso nujno samo 1, -2 in 1.

Algoritem, ki ga bomo napisali, naj torej deluje za splošne tridiagonalne matrike dimenzije $n \times n$

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & & \ldots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \ldots & \vdots \\ 0 & a_{32} & a_{33} & a_{34} & \ldots & \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ldots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & \ldots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \tag{6}$$

Matriko, kakršno imamo v (5) lahko v *octave*-u sestavimo v eni vrstici (s pomočjo ukaza `diag`) in nato lahko sistem rešimo z vgrajeno implementacijo Gaussove eliminacije oz. LU razcepa (operator \). (Ta rešitev je tudi objavljena na učilnici).

Za manjše n (recimo $n < 1000$) je to še lahko smiselno, ker ne zahteva veliko časa ali prostora. Pri realnih problemih pa se hitro lahko zgodi, da je število neznank, ki jih imamo, lahko več milijonov. V tem primeru je popolnoma brezupno, če bi na takšen način reševali tak sistem. Že za zapis matrike v (5) ali (6) potrebujemo $n^2 \cdot$ `sizeof(double)` prostora, kar za recimo $n = 10^6$ že preseže spomin, ki ga imamo na razpolago (vsaj na 32-bitnih sistemih). Še huje pa je, kar se tiče časovne zahtevnosti, saj vemo, da LU razcep (in tudi razcep Choleskega) potrebuje reda n^3 operacij.

Očitno pa je, da za zapis vseh informacij, ki jih imamo v matriki (6), ne potrebujemo $n \times n$ matrike, saj je matrika "skoraj" prazna. Neničelnih elementov je samo $3n - 2$, torej bi vse podatke lahko spravili v $n \times 3$ matriko. Tudi potek Gaussove eliminacije za matriko (6) lahko precej optimiziramo, saj vidimo, da bi Gaussova eliminacija na polni matriki veliko večino časa množila, seštevala in odštevala same ničle.

Namesto matrike A iz (6) bomo torej raje uporabili naslednjo matriko

$$M = \begin{bmatrix} 0 & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} \\ \vdots & \vdots & \vdots \\ a_{i,i-1} & a_{i,i} & a_{i,i+1} \\ \vdots & \vdots & \vdots \\ a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,n-1} & a_{n,n} & 0 \end{bmatrix} \tag{7}$$

Poddiagonalo matrike A smo shranili v 1. stolpec, diagonalo v 2. stolpec in naddiagonalo v 3. stolpec matrike M.

Poglejmo si še, kaj moramo narediti na prvem koraku Gaussova eliminacije. Zapišimo prvi dve enačbi:

$$\begin{array}{rcrcrcl} a_{11}y_1 & + & a_{12}y_2 & & & = & b_1 \\ a_{21}y_1 & + & a_{22}y_2 & + & a_{23}y_3 & = & b_2 \end{array}$$

Da uničimo element a_{21}, moramo od 2. vrstice odšteti k-kratnik 1. vrtsice, kjer je

$$k = \frac{a_{21}}{a_{11}}$$

Popraviti moramo v resnici samo elementa a_{22} in b_2, saj je prva vrstica skoraj prazna. Torej

$$\begin{aligned} a'_{21} &= 0 \\ a'_{22} &= a_{22} - k \cdot a_{12} \\ b'_2 &= b_2 - k \cdot b_1 \end{aligned}$$

To je vse delo, ki ga moramo opraviti za uničenje celotnega prvega stolpca pod diagonalo, kar sicer za splošne matrike zahteva reda n^2 operacij.

Podobno imamo na i-tem koraku Gaussove eliminacije enačbi

$$\begin{array}{rcrcrcl} a_{i,i}y_i & + & a_{i,i+1}y_{i+1} & & & = & b_i \\ a_{i+1,i}y_1 & + & a_{i+1,i+1}y_{i+1} & + & a_{i+1,i+2}y_{i+2} & = & b_{i+1} \end{array}$$

saj smo element $a_{i-1,i}$ že uničili na prejšnjem koraku. Nove elemente potem izračunamo z

$$\begin{aligned} k &= \frac{a_{i+1,i}}{a_{i,i}} \\ a'_{i+1,i} &= 0 \\ a'_{i+1,i+1} &= a_{i+1,i+1} - k \cdot a_{i,i+1} \\ b'_{i+1} &= b_{i+1} - k \cdot b_i \end{aligned}$$

Sedaj moramo samo pazljivo prepisati dogajanje v matriki A iz (6) v dogajanje v matriki M iz (7). Napišimo kar kodo v *octave*-u:

```
for i=1:n-1
   k=M(i+1,1)/M(i,2);
   M(i+1,1)=0;
   M(i+1,2)=M(i+1,2)-k*M(i,3);
   b(i+1)=b(i+1)-k*b(i);
end
```

Skratka, namesto $O(n^3)$ operacij in $O(n^2)$ prostora tako potrebujemo samo $O(n)$ operacij in tudi $O(n)$ prostora.

Tudi obratno vstavljanje lahko izvedemo v $O(n)$ operacij (namesto običajnih $O(n^2)$), saj imajo vse enačbe, ki nam po Gaussovi eliminaciji ostanejo, kvečjemu dve neznanki. Zadnja enačba je

$$a_{n,n}y_n = b_n,$$

torej

$$y_n = b_n/a_{n,n}$$

Na i-tem koraku (kjer gre i od $n-1$ nazaj do 1) pa imamo enačbo

$$a_{i,i}y_i + a_{i,i+1}y_{i+1} = b_i$$

iz česar lahko izrazimo

$$y_i = (b_i - a_{i,i+1}y_{i+1})/a_{i,i}$$

Spet preostane samo še, da prevedemo elemente matrike A v ustrezne elemente matrike M:

```
y(n)=b(n)/M(n,2);
for i=n-1:-1:1
    y(i)=(b(i)-M(i,3)*y(i+1))/M(i,2);
end
```

Delujoči programi so objavljeni na učilnici, skupaj s funkcijo, ki primerja časovno zahtevnost vgrajene metode in naše metode za reševanje tridiagonalnega sistema, iz Jasno se vidi, da je časovna zahtevnost naše funkcije linearna, za vgrajeno funkcijo pa $O(n^3)$.

Za majhne n je vgrajena metoda hitrejša, ker je sprogramirana na nižjem nivoju kot funkcija, ki smo jo napisali, za malo večje n pa jo naša kmalu prehiti.