ADRIAN CHISĂLIŢĂ

ANA

Biblioteca de Analiză numerică – surse Fortran 90

Manual de utilizare

Universitatea Tehnică din Cluj-Napoca

Cluj-Napoca, 2014

Notă copyright

Versiune ANA (on-line): Decembrie 2014

Ediţie Manual de utilizare (on-line): Decembrie 2014

ANA se găseşte la următorul URL:

ftp.utcluj.ro/pub/users/chisalita/ANA/

(Manualul se găseşte în folderul ANA/_DOC/)

Drepturi de utilizare a bibliotecii ANA

1. În scop educaţional:

- Utilizarea este liberă.
- Utilizatorul are dreptul la o copie a Bibliotecii şi Manualului, pentru uz personal.
- Utilizatorul poate modifica rutinele după voie, în scop personal. Referirea la rutinele modificate se va face cu citarea sursei originale.

Prevederile de mai sus echivalează cu licenţa educaţională.

2. În scop comercial:

- Utilizarea în scopuri comerciale, sau încorporarea de rutine într-un produs comercial, este ilegală fără obţinerea unei licenţe comerciale.
- Reproducerea în orice formă (tipărită/electronică) a Bibliotecii sau Manualului, este interzisă.
- Alte prevederi ale Legii dreptului de autor sunt aplicabile.

Reproducerea unor părţi din ANA sau Manual, cu titlu de exemplu sau citare, în orice formă (tipărită/electronică), este permisă numai cu menţionarea copyright-ului.

Manualul de utilizare

Textul Manualului de utilizare se supune prevederilor Legii dreptului de autor.

Autorul garantează conţinutul bibliotecii ANA, numai pentru utilizarea în scop educaţional.

Autorul îşi declină responsabilitatea pentru orice consecinţe care ar decurge din utilizarea bibliotecii în alt scop.
Sursele din ANA sunt oferite ”aşa cum sunt”.
Codul sursă şi Manualul pot fi modificate de autor fără notificare.

Actualizare

Atât sursele din ANA, cât și Manualul, sunt actualizate permanent.
Pot exista surse mai recente, sau surse suplimentare, față de cele descrise în Manual.
Versiunile cele mai recente ale surselor se găsesc la URL-ul de mai sus.

Modificări în versiunea curentă

- Au fost create supra-foldere tematice, cu numele $ NUME. Ele conțin foldere ale metodelor numerice (sau ale unor problemelor de analiză numerică). Au fost păstrate numai ultimele versiuni ale proiectelor (ale fișierelor din proiecte).
- A fost creat folderul ANA-OLD care conține versiuni mai vechi ale proiectelor/surselor.

■

CUPRINS

INTRODUCERE – Biblioteca ANA

Metodele din ANA sunt bazate pe teoria şi algoritmii expuşi în manualul: Adrian Chisăliţă, "Numerical Analysis" (2002). În ceea ce urmează, citările unor capitole, paragrafe, se referă la această lucrare.

Notă:

- Unii algoritmi din ANA sunt mai evoluaţi decât cei expuşi în *"Numerical Analysis"*.
- Unele surse din ANA pot fi mai evoluate decât cele descrise in acest Manual. În acest caz, actualizările şi, în particular, structura fişierului de intrare, se descriu în comentariile din programul principal (Main-...).

■

ANA este organizată pe foldere care conţin rutinele necesare pentru utilizarea unei metode numerice, calculul unor mărimi (exemplu: numărul de condiţie), sau ilustrarea erorilor sau preciziei în calculul cu numere reprezentate în virgulă flotantă.

Folderele conţin fişierele sursă necesare pentru construcţia unui proiect. În general, folderul conţine:

1. Programul principal
2. Subrutina metodei
3. Subprogramul de tip `function/subroutine` pentru definirea funcţiei / funcţiilor cu care se lucrează (pentru ecuaţii neliniare)
4. Alte subrutine utilitare

Programul principal realizează, cel puţin, următoarele acţiuni:

- Citeşte datele de intrare
- Apelează subrutina metodei
- Tipăreşte rezultatele.

Unităţile de program 1 şi 3 se presupun scrise de utilizator. Pentru facilitate, ele sunt incluse în folder.

Unitatea 3 conţine cod pentru un exemplu de test. Pentru rădăcinile polinoamelor, sistemele de ecuaţii liniare şi pentru problema de valori proprii, în locul unităţii 3, în

folder este inclus un fişier cu datele pentru un exemplu de test. Utilizatorul va modifica codul din 3 sau datele din exemplul de test, pentru problema concretă cu care se lucrează.

De asemenea, utilizatorul poate modifica programul principal.

Utilitarele sunt rutine incluse în proiect, în special pentru tipărirea datelor de ieşire. Descrierea lor se dă în capitolul UTILITARE.

Nume foldere. Nume unităţi de program şi fişiere asociate

În general:

- Numele folderului este o mnemonică a metodei.
- Numele subrutinei metodei este tot o mnemonică a metodei.
  Exemplu: pentru metoda bisecţiei, numele folderului şi al subrutinei este BIS.
- Programul principal are numele MAIN- urmat de numele subrutinei. Exemplu: MAIN-BIS.
- Numele funcţiei este fun, f, g, etc.

În câteva cazuri, programul principal încorporează şi metoda; în acest caz, numele unităţii de program este o mnemonică a metodei (fără a mai fi precedat de MAIN-).

Nume foldere/Nume Fişiere Sursă

- Pentru metodele în dublă precizie, în general, numele folderului şi numele fişierelor sursă, sunt succedate de caracterele _D, _d.

  (În versiuni mai vechi, ar putea fi precedate de caracterele D, d).

- Fără sufix (sau prefix): În general: Metode şi surse în simplă precizie.

  Uneori, rutina încorporează mai multe niveluri de precizie.

Notă

- Când numele folderului sau al unui fișier-sursă conține o specificație de an, codul din acestea este în versiunea cea mai recentă.
  Exemplu (folder): Gauss 2010 și Gauss.
- Când numele folderului sau al unui fișier-sursă conține terminația -IVF, compilarea este presupusă pentru compilatorul Intel Visual Fortran.

■

Cod

În scrierea codului, s-a dat întâietate clarităţii programării metodei, uneori în dauna compacităţii codului. Datele de intrare se citesc fie de la terminal, fie dintr-un fişier de date. Aceeaşi observaţie pentru scrierea datelor de ieşire. Când se utilizează un fişier de date, în general, datele de ieşire se scriu în acest fişier, în continuarea datelor de intrare. Soluţia fişierului de date se poate adopta întotdeauna, prin modificarea codului.

- Pentru simplificare, în versiunile mai vechi de cod, s-a evitat utilizarea ferestrelor de dialog pentru deschiderea / selectarea / crearea unui fişier sau directoriu. În acest caz: fişierul de intrare se specifică cu cale completă; sau, se poate specifica numai cu nume, dacă fişierul se găseşte în folderul proiectului (în acelaşi folder cu sursele).
- În versiunile actuale de cod, operaţiile de mai sus se fac prin fereastra de dialog standard (în particular, selectarea fişierului de intrare).

Codurile prezentate în versiunea simplă precizie, pot fi convertite uşor la dublă precizie prin adăugarea declaraţiilor necesare. Limbajul este Fortran 90/95.

Unităţile de program conţin comentariile necesare pentru semnificaţia datelor de intrare, parametrilor de apel ai subrutinelor, etc. Pentru concretizarea datelor de intrare v. fişierul de test al metodei.

Observaţie

Testele de oprire a unei iteraţii sunt, în general, de forma: $|x_{n+1} - x_n| \le eps$ şi *număr de iteraţii* $\le$ *lnit*. Toleranţa *eps* şi numărul maxim de iteraţii *lnit* sunt date de intrare. Trebuie ca $eps \ge ULP(x_n)$, unde $ULP(x_n)$ este cel mai mic număr care adunat la x_n dă, prin rotunjire, un număr mai mare ca x_n – v. Cap. 1, 3.6. În caz contrar, cu excepţia cazului $x_{n+1} = x_n$, primul test nu poate fi satisfăcut şi iteraţia se opreşte prin depăşirea lui *lnit*. Uneori codul testează şi corectează toleranţa *eps* introdusă de utilizator – v. ca exemplu metoda bisecţiei. În caz contrar, la introducerea datei *eps*, utilizatorul se va ţine cont de condiţia de mai sus ■

Utilizare

Cu rutinele din folder se construieşte un proiect de tip "Console Application". Pentru simplificare şi portabilitate, rutinele nu conţin dialoguri şi nici ieşiri grafice. Utilizarea este presupusă pentru compilatorul Compaq Visual Fortran 6.6C, 2002 (și pentru mediul Developer Studio), dar rutinele pot fi utilizate şi pe o altă platformă. În particular, rutinele cu numele "nume-IVF" se referă la compilatorul Intel Visual Fortan 11.x, 2010 (și mediul Microsoft Visual Studio 2008).

Descrierea fişierelor din foldere şi a metodelor numerice, se dă în capitolele următoare. Pentru unele metode (în particular: Muller, problema de valori proprii, și Interpolare), prezentarea teoretică este mai desvoltată.

Pentru detalii suplimentare, ca și *pentru cele mai recente actualizări*, v. comentariile din surse. Unele comentarii pot fi în limba engleză.

Avertizare: *Exemple de test și Rezultate*

> Majoritatea rezultatelor numerice din exemplele de test sunt obținute prin rularea cu un procesor Pentium IV, și cu compilatorul CVF 6.6c – cu opțiunile de compilare implicite al mediului Developer Studio, dacă nu se specifică altfel. (Cu excepția rezultatelor indicate ca obtinute cu compilatorul IVF 11.1; procesorul este Intel Duo T2450).
>
> Utilizarea altor procesoare și altor compilatoare (sau opțiuni de compilare) ar putea face ca, în unele cazuri, *rezultatele numerice să nu se reproducă exact* – ci numai apropiat.

■

Fonturi

Text: Times New Roman

Nume foldere: Arial

Nume fişiere; nume rutine: Arial 11

Date de intrare; rezultate (din fişierul de ieşire); secvenţe de cod: `Courier New 11`

Caracterul ■ indică sfârşitul unei Propoziţii, Observaţii, Exemplu, etc.

CAPITOLUL I – REPREZENTAREA NUMERELOR ÎN CALCULATOR şi ERORI

Foldere tematice: **$ Recurrence; $ Representation; $ Errors; $ Ulp.**

Recurrence

Bessel

Calculul funcţiei Bessel: direct, şi prin recurenţă. Se pune în evidenţă, instabilitatea algoritmului bazat pe formula de recurenţă cu trei termeni. Implementarea Exemplului 4, Cap. 0, 2.2.

Fişiere:

Bessel.f90

Se calculează valorile funcţiilor Bessel de speţa I-a, pe argumentul $y = 1$, pentru $m = 1{,}10$, în două moduri:

(a) Direct (serie): $J_m(1) = \left(\frac{1}{2}\right)^m \sum_0^\infty \frac{\left(-\frac{1}{4}\right)^k}{k!(m+k)!}$

Seria se trunchiază în momentul când termenul curent este mai mic decât, sau egal cu 1E-15 (şi se însumează cel mult 100 termeni). Valorile obţinute se consideră valorile "exacte".

(b) Prin recurenţă: $J_{m+1}(1) = 2m\,J_m(1) - J_{m-1}(1)$

Se porneşte cu valorile $J_0(1)$ şi $J_1(1)$ calculate cu trei grade de precizie, exprimate prin numărul de cifre semnificative exacte: 15–16 (dublă-precizie), 10, şi 7 (simplă precizie). Calculul, cu aceste valori de pornire, se face în dublă precizie.

Rezultatele se scriu în fişierele bess-n.rez, unde $n = 7;10;16$.

Pentru calculul direct: numărul de termeni care se însumează efectiv este 9 … 5 (pentru $m = 0,1,\ldots,10$); acest număr se listează în fişierul de ieşire. Primul termen neglijat în serie este de ordinul $10^{-17}\ldots10^{-18}$; acest termen se listează la display.

Se remarcă divergenţa crescătoare cu *m*, a valorilor calculate prin recurenţă în raport cu valorile exacte – chiar pentru un număr mare de cifre semnificative (16), în J_0 şi

J_1. Singurele erori introduse în calculul (b) sunt erorile de rotunjire în valorile lui J_0 şi J_1.

Exemplul Muller-Kahan

Cf. Kahan [2006].

Se consideră funcția

$f(y,z) = 108 - (815 - 1500/z)/y$

Se definește șirul:

$x_0 = 4; \quad x_1 = 4.25$

$x_{n+1} = f(x_n, x_{n-1}), \quad n \geq 1$

Șirul converge către o limită *L*; se propune a se găsi *L*, prin calculul lui x_n pentru $n =$ "mare", de exemplu $n \sim 80$.

Calculul în dublă precizie (și în precizie extinsă) conduce la $L \approx 100$. Limita adevărată este $L = 5$ - v. Lucr. citată.

Exemplul arată instabilitatea formulei de recurență cu 3 termeni pentru calculul lui x_{n+1}.

Notă

Calculul în precizie cvadruplă dă rezultatul corect, anume:

$x_{81} =$ `4.99979690071464772103694582965119` ■

Representation

Represent_Int

Reprezentarea numerelor întregi, în format.

Tipuri: INTEGER(*n*): *n* = 1; 2; 4; 8;

Fişiere:

Main_Represent_Int-3.f90

Represent_Int-2.obj

width.f90

Exemplu

La intrarea: Tip: INTEGER(4); Intreg = 1082130432, programul are următoarea ieşire

```
I: 1082130432
bx:01000000 10000000 00000000 00000000
S|int:0|10000001000000000000000000000000
```

`I` = întregul introdus; `bx`: reprezentarea în format ca șir de biți (pentru claritate, octeții sunt separați cu un spațiu);

`S|int`: reprezentarea în format cu evidențierea bitului de semn (S).

Represent

Reprezentarea numerelor reale, în format.

Tipuri: REAL(*n*): *n* = 4; 8; [16];

Sub-foldere:

\CVF: compilare cu CVF; \IVF: compilare cu IVF;

Fişiere (\IVF):

Main_Represent.f90

Represent.obj

width.f90

Note:

- Tipul REAL(16) este suportat numai de IVF.
- Se va compila cu opțiunea: Enable IEEE Minus Zero Support: Yes.

▪

Valori *x*, admise la intrare (3 și 4 – numai cu IVF):

1) Numere finite (normale; denormale)

2) 0.0; -0.0

3) infinity; -infinity

4) nan

Exemplu

- Intrare: Tip: REAL(4); *x* = 1E38;
- Iesire:

```
   x:  9.9999997E+37
bx:01111110 10010110 01110110 10011001
  S:0
Exp:11111101
```

```
Sig:00101100111011010011001
 iSig:     1472153

(E_stocat = 253;  E =126)
(iSig: 1472153;  Sig = 1.175494;  Fr = 0.587747)
```

`x:` Realul introdus;

`bx:` Reprezentarea în format ca şir de biţi (octeţii sunt separaţi cu un spaţiu);

`S; Exp; Sig`: Bit de semn; Câmp exponent; Câmp semnificand;

`E_stocat; E:` Exponentul stocat (în câmp exponent); Exponentul reprezentării (în calculator);

`iSig; Sig; Fr:` Întregul stocat în cîmp semnificand; Semnificandul; Fracţia în model – returnată de funcţia intrinsecă fraction(x).

Exemplu-2 (IVF)

- Intrare: Tip: REAL(4); *x* = nan;
- Iesire:

```
x: NaN
bx:11111111 11000000 00000000 00000000
  S:1
Exp:11111111
Sig:10000000000000000000000

(E_stocat = 255;  E =128)
```

■

Intrinsec_2

Apelul funcţiilor intrinseci Fortran, pentru parametrii reprezentării.

Fişiere:

Intrinsec-2.f90

Se exemplifică apelul, pentru o dată `real(4)` şi `real(8)`, a următoarelor funcţii intrinseci:

Digits; Exponent; Fraction; Maxexponent; Minexponent; Nearest; Huge; Precision; Range; Spacing; Tiny; Epsilon.

Se testează `1 + epsilon(1.0) > 1`.

Pentru semnificaţia funcţiilor intrinseci – v.: “Compaq Visual Fortran Language Reference Manual”, 2001; “Intel Visual Fortran Compiler 11.x for Windows”, 2010.

Special_Values

Valori speciale.

Fişiere:

Special_Values.f90

Programul ilustrează calculul şi tipărirea următoarelor valori speciale – Cap.1, 3.5:

- Zero cu semn: -0.; $(-1)*0.$
- Infinit cu semn: $\pm 1./0.$
- NaN (Not a Number): 0./0; (1./0)∗0; (1./0) +(-1./0)
- Număr denormalizat: 10^{-45}
- Depăşire format (superioară): 10^{40}
- Depăşire format (inferioară): 10^{-46}

În Project Settings, se va seta opţiunea Enable IEEE Minus Zero Support, în tab-ul Fortran/Floating Point.

Errors

Folderul conţine următoarele sub-foldere, conţinând ilustrări de erori, în calculul cu numere reprezentate în virgulă flotantă.

Assoc_sum

Ne-asociativitatea sumei.

Fişiere:

Sum_1.f90, FP_precis.f90

Se consideră termenii

x1 = 1.25E20; x2 = 555.5,

şi sumele

s1_p = x1 + x2 – x1

s2_p = x1 – x1 + x2.

p notează precizia pentru reprezentarea termenilor (şi a sumelor parţiale), anume:

s: precizie simplă (24 biţi), *d*: precizie dublă (53 biţi), şi *e*: precizie extinsă (64 biţi).

Precizia de reprezentare de 64 biţi se setează cu subrutina fpprecis.

Rezultatele se scriu în fişierul sum.rez, anume:

```
s1_s = 0.000000
s1_d = 0.000000000000000
s1_e = 552.0000000000000000
```

Suma s2_p are valoarea corectă (pentru oricare *p*): `s2 = 555.50 … 0.`

Do

Calculul unei sume într-un ciclu DO, în formele:

sum1 = 0.; sum1 = sum1 +h; *n* termeni.

sum2 = j∗h; $j = 1, n$

Fişiere:

Do2003.f90

Programul listează sum1, sum2, şi eroarea relativă a lui sum1, pentru $n = 10^6$, cu pasul 20,000. Se observă că eroarea relativă creşte odată cu numărul termenilor.

Ecuatie_gr2

Pierderea de semnificaţie la rădăcinile unei ecuaţii de gradul 2.

Fişiere:

Ecuatie_gr2.f90

Se consideră ecuaţia $x^2 - 26x + 1 = 0$, cu rădăcinile $x_{1,2} = 13 \pm \sqrt{168}$. Rădăcinile se calculează în simplă şi dublă precizie; valorile în dublă precizie se consideră valorile exacte. În simplă precizie, rădăcina x_2 se calculează în două moduri: după formulă – caz în care apare pierdere de semnificaţie, şi sub forma $x_{2,m} = \frac{1}{13 + \sqrt{168}}$. Se listează erorile relative ale rădăcinilor; se constată că valoarea $x_{2,m}$ este exactă.

Integer2

Calculaţii cu integer(2).

Fişiere:

Integer2.f90

Se ştie că integer(2) se reprezintă pe 2 octeţi, iar plaja de reprezentare este $-32768 \le integer(2) \le 32767$ (Cap. 1, 2). Programul testează suma $32767 + i2$, unde $i2$ este integer(2); de exemplu, pentru $i2 = 3$, se obţine -32766. Explicaţie: termenii sumei se reprezintă ca mai jos (Cap 1, 2; vezi şi reprezentarea acestor numere cu utilitarul Bitview):

```
0|1111111 11111111 = 32767
0|0000000 00000011 = 3
```

Adunarea produce numărul

```
1|0000000 00000010 = -32766
```

Loss_signif

Pierderea de semnificaţie.

Fişiere:

Loss_signif.f90

Proiectul implementează Exemplul 2, Cap.2, 4.1. Se consideră calculul funcţiei:

$$f(x) = x(\sqrt{x+1} - \sqrt{x}),$$

pentru $x = 10^j$, j = 1, 2, …7, în simplă şi dublă precizie. Valorile $f(x)$ calculate în dublă precizie sunt considerate valorile 'exacte'. Se calculează în simplă precizie, valorile funcţiei $g(x)$ dată de

$$g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

g este transformata lui f, care evită calculul diferenţei $\sqrt{x+1} - \sqrt{x}$ care produce pierderea de semnificaţie. Analitic, $g(x) = f(x)$.

Se listează: valorile $f(x)$ în simplă precizie; valorile $f(x)$ în dublă precizie, rotunjite la 8 cifre semnificative; valorile $g(x)$ în simplă precizie. Se observă că:

- Valorile $f(x)$ în simplă precizie se alterează odată cu creşterea lui j, datorită pierderii de semnificaţie.
- Valorile $g(x)$ au 7 cifre semnificative corecte.

Rump_pol

"Polinomul" lui Rump [Rump S.E., 2010, 1988].

Fişiere:

main-rump.f90; FP_precis.f90;

Se consideră funcţia

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

şi evaluarea lui funcţiei, pentru $x = 77617$ şi $y = 33096$.

(Partea esenţială e constituită din polinomul format de primii 3 termeni şi, din acest motiv, exemplul poartă numele de "polinomul" lui Rump. Polinomul a fost considerat de Rump in 1983 şi reluat in lucrarea din 1988).

Programul listează valorile polinomului calculate cu 3 nivele de precizie (oferite de compilatorul CVF 6.6), anume:

```
24 biţi (precizie simplă):       -1.180592E+21
53 biţi (precizie dublă):        -1.180591620717411E+21
64 biţi (precizie extinsă):      5.764607523034234880E+17
Valoarea corectă:
-0.82739 60599 46821 36814 11650 95479 81629 19990 331
```

(obţinută cu precizie de 43 cifre zecimale de precizie; ≈ 144 biţi).

Notă

În precizie cvadruplă (compilatorul IVF 11.1), se obţine:

```
113 biţi (Precizie cvadrupla):   1.17260394005318
```

■

Pentru rezultate similare, obţinute cu compilatorul FORTE Developer 6 (Fortran 95, Sun) v. Loh & Walster (2002).

Explicaţie:

Cei patru termeni ai polinomului au valorile:

```
t1:    4.386057508463932E+29
t2:  -7.917111779274714E+36
t3:    7.917111340668962E+36
t4:     1.17260394005318
```

Astfel, nu se produce depăşire de format nici pentru simpla precizie. Dar termenii doi şi trei sunt egali cu aproximativ 2^{123}, şi astfel, pentru reprezentarea cu toate cifrele pentru evitarea pierderii de semnificaţie, este necesară o precizie de cel puţin 123 biţi. Ori, chiar precizia cvadruplă (`real(16)`) nu oferă decât 113 biţi. Astfel, pentru

evaluarea polinomului în forma dată, trebuie apelat la un pachet multi-precizie – cum este, de exemplu, MPFUN, v. "High-Precision Software Directory", 2010.
Explicaţia *analitică* este următoarea (Loh & Walster (2002)): *x* şi *y* daţi, satisfac relaţia $x^2 = 5.5y^2 + 1$. Cu aceasta, termenii de ordin superior lui 1 în *x* şi *y* se reduc, şi expresia polinomului devine

$$f_1(x, y) = -2 + \frac{x}{2y}.$$

Astfel, suma adevărată a primilor 3 termeni (exceptând fracţia) este -2.
Programul listează şi valoarea lui $f_1(x, y)$, calculată în precizie dublă.

Rump II

Polinomul nr. 2 al lui Rump [Rump S.E., 2010]
Fişiere:
main-rump II.f90; FP_precis.f90;

Exemplul este similar cu cel din paragraful anterior, dar cu o expresie mai simplă; a fost introdus de Rump in 2010.

$$f(a,b) = 21 \cdot b \cdot b - 2 \cdot a \cdot a + 55 \cdot b \cdot b \cdot b \cdot b - 10 \cdot a \cdot a \cdot b \cdot b + a/2 \cdot b$$

Se calculează în aceleaşi valori: *a* = 77617.; *b* = 33096.
Suma primilor 4 termeni este -2 (valoarea adevărată).
Programul listează, ca in ex. precedent, valorile polinomului calculate cu 3 nivele de precizie (compilatorul CVF 6.6), şi anume:

```
24 biţi (precizie simplă):       1.17260
53 biţi (precizie dublă):        1.17260396480560
64 biţi (precizie extinsă):      1.172603964805603027
```

Cu precizie cvadruplă (compilatorul IVF 11.1), se obţine:

```
-.827396059946821368141165
```

Valoarea corectă cu 43 cifre zecimale de precizie, este tot cea din paragraful Rump_Pol, întrucât analitic, polinomul se scrie

$$f_1(a,b) = -2 + \frac{a}{2b}$$

Note

- Rezultatul calculat cu precizie cvadruplă (113 biți, compilatorul IVF 11), are *toate cifrele semnificative corecte* (24);
- Valoarea corectă se obține cu MPFUN, cu minimum 22 cifre zecimale de precizie, anume: `-.827396059946821368141l;`

■

SSH

Problema ”Sea Surface Height”.

Fişiere:

Main_SSH-2.f90

ssh.dat

Un model de circulație a oceanului a fost desvoltat de He & Ding (2001). Variabila SSH stochează înălțimea medie a suprafeței mării, din simularea în model. Datele corespunzând simulării dintr-o zi, pentru o reţea de 120×64 = 7680 de puncte (120 longitudini şi 64 de latitudini), au fost stocate într-un tablou `ssh(120,64)`, cu precizia de 64 biţi. Datele sunt numere de ordinul de mărime 10^{10}…10^{14} şi de semne diferite, în timp ce rezultatul sumării e de ordinul lui 1.

Datele se citesc din fişierul de intrare ssh.dat. Ele se găsesc stocate cu 17 cifre zecimale.

Programul calculeaza suma elementelor din tabloul ssh, cu precizie dublă şi cvadruplă.

Notă

Pentru compilatorul CVF 6.x, instrucțiunile referitoare la tipul `real(16)` vor fi făcute comentarii (sau eliminate) ■

Ordonarea sumei după diferite criterii produce rezultatele de mai jos:

```
Real(8):
 sum_row =    7.39062500000000
 sum_col =    7.39062500000000

 sum_row_reverse =    6.93945312500000
 sum_col_reverse =    6.64746093750000
```

```
Sum on array (Real(8)):
  sum_plus =  2.651252030584858E+016
  sum_minus = -2.651252030584855E+016
  sum_plus +sum_minus =   24.0000000000000

Real(16):
 sum_row_16 =  0.35798583924770355224609375000000000
 sum_col_16 =  0.35798583924770355224609375000000000
```

Valoarea corectă este cea obţinută cu precizia cvadruplă (real(16)) – oferită de compilatorul IVF 11.1.

Notă

Rezultatul corect se poate obține și cu pachetul soft de precizie multiplă MPFUN (Bailey, 2010). He & Ding au utilizat metoda sumării auto-compensată (SCS).

■

Sum_Armonic

Suma seriei armonice – strategia de sumare.

Fişiere:

Sum-armonic.f90

Programul calculează suma

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n},$$

pentru $n = 10^i$, i = 2, 3, ..., 7, astfel:

a) Sum_Smallest: de la cel mai mic la cel mai mare – simplă precizie;

b) Sum_Largest: de la cel mai mare la cel mai mic – simplă precizie;

c) Sum_Double: de la mic la mare – dublă precizie, pentru

Sum_Double se consideră valoarea exactă. Se listează valorile sumelor şi erorile relative ale primelor două sume. Se observă, pentru $n \geq 5$, că eroarea relativă a sumei (a) este mai mică decât cea a sumei (b).

Proiectul conţine şi sursa pentru calculul sumei seriei armonice alternate.

În Project Settings, se va seta opţiunea "Enable Floating Point Consistency" ■

Sum_ErrFree

Algoritm de sumare fără erori.

Fişiere:

Main-ErrFree.f90; sub-Sum_2.f90; sub-Sum_N.f90; width.f90;

Proiectul implementează un algoritm de sumare precisă din Ogita T., Rump S.M., and Oishi S., "Accurate Sum and Dot Product" (2003). Acesta este definit de următorii doi algoritmi:

```
Algorithm 3.1. Error-free transformation of the sum of two
floating point numbers.
function [x, y] = TwoSum(a, b)
x = fl(a + b)
z = fl(x − a)
y = fl((a − (x − z)) + (b − z))
```

Algoritmul 3.1 transformă două numere *a, b* reprezentate FP, în alte două numere FP *x, y,* astfel că: $a+b=x+y$ şi $x=fl(a+b)$.

```
Algorithm 4.1. Cascaded summation.
function res = Sum2s(p)
π1 = p1; σ1 = 0;
for i = 2 : n
[πi, qi] = TwoSum(πi−1, pi)
σi = fl(σi−1 + qi)
res = fl(πn + σn)
```

Algoritmii sunt implementaţi prin rutinele `sum_2(a,b,x,y)` şi `sum_N(n, p, s_EF, cor)`. Cu aceştia, se calculează suma din proiectul Assoc_sum. Se obţine valoarea corectă.

Test_eps

Test pentru eroarea de rotunjire a unităţii.

Fişiere:

test_eps.f90

Se definesc (Cap.1, 3.8):

$machine\_EPS = 2.0^{-24}$; $ULP1 = 2.0^{-23}$; $EPS = EPSILON(1.0)$;

Se testează că:

1 + $ULP1$ >1.0

1 + *machine_EPS* = 1.0

1+ EPS >1.0

Ulp

Calculul lui $ULP(x)$, pentru o plajă de valori *x*.

Fişiere:

main-ULP2003.f90; ULP(x)-2003.f90;

Se defineşte – v. Cap.1, 3.8,

$ULP(x) = 2^{e-P}$,

unde: *e* este exponentul reprezentării lui *x* în modelul ştiinţific, returnat de funcţia `exponent(x)`; *P* este precizia reprezentării, în biţi ($P = 24$ sau $P = 53$).

Se calculează:

$x1 = x + ULP(x)$; $x2 = x + ULP(x)/2$

Se testează că:

$x1 > x$; $x2 = x$.

Pentru test, se recomandă:

- O plajă de valori $x = 1 \ldots 31$;
- O valoare mare (de ex. 1E6).

Programul listează parametrii reprezentării lui *x* (în modelul ştiinţific), anume fracţia şi exponentul, $ULP(x)$, şi verifică relaţiile de mai sus.

Folderul conţine sursele pentru versiunea în simplă precizie; folderul Ulp_d conţine sursele pentru versiunea în dublă precizie.

CAPITOLUL II – ECUAŢII NELINIARE

Folder tematic: **$ Nonlinear Equations**

Problema:	Foldere:
1. Ecuaţii de forma $f(x) = 0$:	**Bis; Secant; Newton;**
2. Ecuaţii de forma $x = g(x)$, metoda punctului fix:	**Fix; Stationar; Aitken;**
3. Sisteme de ecuaţii neliniare:	**Fix_Sys; Newton_Sys;**
4. Rădăcinile polinoamelor:	**C_pol; Pol; Pol_Direct; Pol_Complex; Laguerre; Muller.**

II-1 Ecuaţii de forma *f*(*x*) = 0

Pentru metodele bisecţiei, Newton şi secantei, funcţia de test este:

$f(x) = e^x - 3x^2$

Aceasta admite 3 rădăcini situate în intervalele [-1,0], [0,1], [3,4].

Aproximaţiile rădăcinilor, cerute de metodele numerice (respectiv, intervalele care conţin rădăcina), se determină prin metode algebrice.

Mai simplu, aproximaţiile rădăcinilor se pot găsi prin inspecţia graficului funcţiei.

Ca exemplu, graficul funcţiei $f(x) = e^x - 3x^2$ se dă mai jos.

Acesta este construit cu programul GRAPH, pe baza fişierului de ieşire din programul conţinut în folderul **Grafic**.

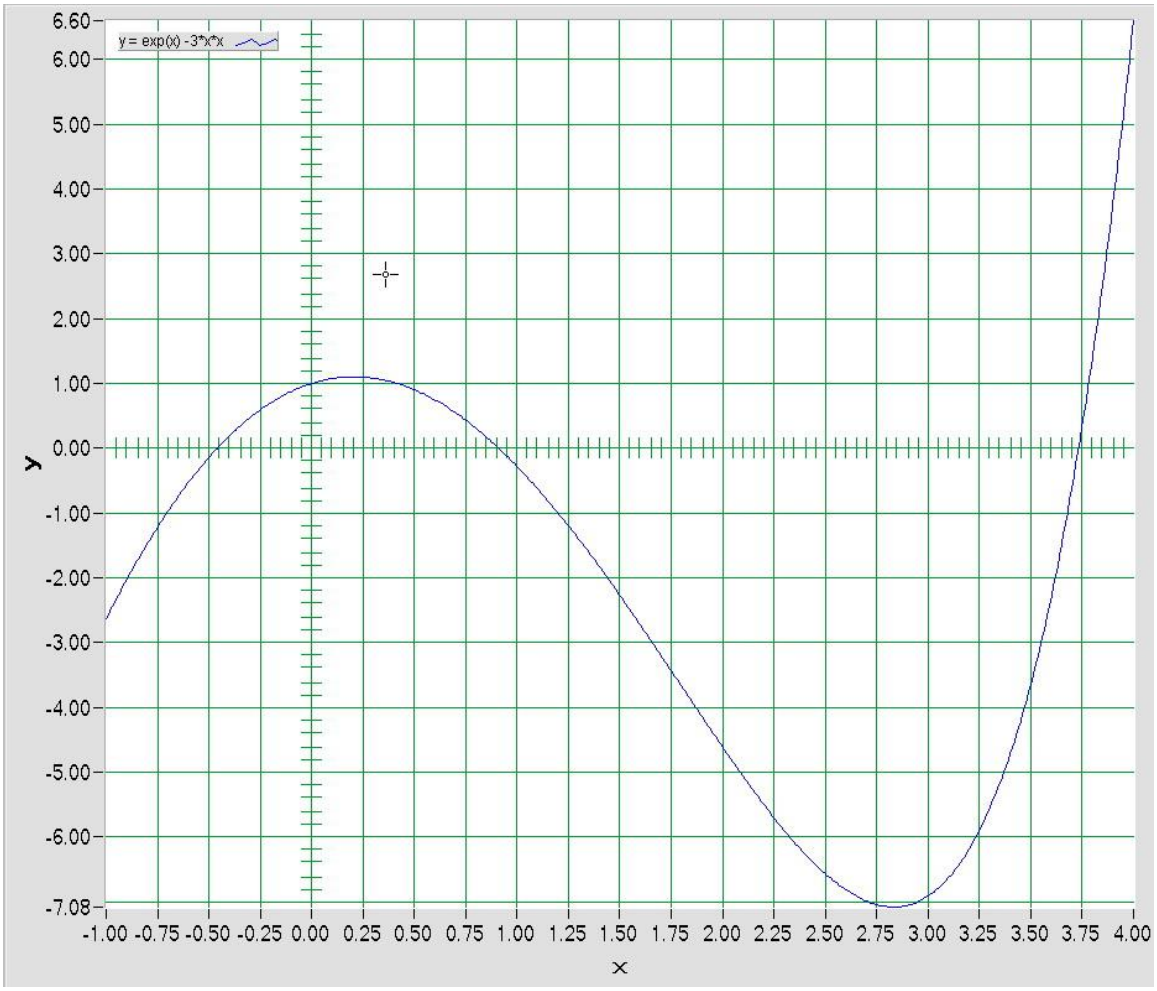


Figura 1 – Graficul funcţiei $f(x) = e^x - 3x^2$

Bis

Metoda bisecţiei.

Fişiere:

Bis2005.f90: subrutina metodei

Main_Bis2005.f90: programul principal

Fun.f90: subprogram de tip `function`, care defineşte funcţia *f*(*x*). Expresia funcţiei poate include un parametru *p*; dacă există, valoarea lui *p* este dată de intrare.

Function_expression.f90, width.f90: UTILITARE.

Metoda:

Funcţia *f* se presupune definită şi continuă pe intervalul $[a,b]$, luând valori de semne contrare în capetele intervalului, şi având o singură rădăcină în interval.
Se defineşte
$c = (a+b)/2$
Se ia ca nou interval $[a,b]$ acela dintre $[a,c]$ şi $[c,b]$ în care funcţia ia valori de semne contrare la capete; etc. Şirul punctelor *c* converge la rădăcină, liniar.

Subrutina metodei este:
bis(f, a, b, eps, iter, rad, kod, ktip)

Parametrii formali sunt:

f: Numele funcţei *f*(*x*). Este declarat cu atributul `external` in programul principal.
a, b: Intervalul care conţine rădăcina.
eps: Toleranţa.
iter: Numărul maxim de iteraţii *lnit* (intrare) / Numărul efectiv de iteraţii (ieşire)
rad: Rădăcina calculată.
kod: Cod de terminare a iteraţiei (utilizat intern):
0: s-a atins toleranţa *eps*
1: s-a depăşit *lnit*
-1: pentru intervalul specificat *f*(*a*)∗*f*(*b*) > 0
2: rădăcină în *a* sau *b*
ktip: Cod de tipărire iteraţii: 0 (Nu); ≠ 0 (Da)

Testul pentru atingerea toleranţei *eps* este
$|b-c| \le eps$,
unde $[a,b]$este intervalul curent. Metoda ajustează toleranţa *eps*: dacă $eps < ULP(c)$, se pune $eps = ULP(c)$.

Datele de intrare se introduc de la terminal.
Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator, şi conţin:

- expresia funcţiei *f* (scrisă de utilitarul Function_expression)

- [*p*, dacă există]
- datele de intrare *a*, *b*, *lnit*, *eps*
- modul de încheiere a iteraţiei (mesaj, conform codului *kod*)
- rădăcina calculată *rad*
- valoarea *f*(*rad*)
- numărul efectiv de iteraţii.

Datele de intrare pentru exemplul de test (3 rădăcini):

a, *b*: -1, 0; 0, 1; 3, 4

eps: 1E-6

lnit: 50; 20; 20

ktip: 0 sau 1

Rezultate:

```
Toleranţa eps atinsă.
Radacini / Număr de iteraţii
-0.4589624 20;   0.9100084  20;   3.733079   20;
```

Exerciţiu

Rulati, din nou, pentru una dintre rădăcini, cu toleranţa 1E-8 şi lnit = 30 ■

Secant

Metoda secantei.

Fişiere:

Secant2005.f90: subrutina metodei

Main_Secant2005.f90: programul principal

fun.f90: subprogram de tip `function`, care defineşte funcţia *f*(*x*).
Expresia funcţiei poate include un parametru *p*.

Period.f90: subrutină pentru perioada procesului staţionar

Function_expression.f90; width.f90: utilitare

Metoda:

Funcţie *f* continuă pe un interval în jurul rădăcinii α.

Formula de iterare este

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}; \quad n \geq 1; \quad x_0 \text{ şi } x_1 = \text{date.}$$

Aproximaţiile x_0 şi x_1 pot încadra sau nu rădăcina.

Dacă x_0 şi x_1 sunt suficient de apropiate de α (şi f, f' şi f'' sunt continue), şirul $x_n \to \alpha$, iar ordinul de convergenţă este $p \approx 1.618$.

Subrutina metodei este:

secant(f, x0, x1, eps, lnit, rad, kod)

Parametrii formali sunt:

f: numele funcţiei;

*x*0, *x*1: cele două aproximaţii iniţiale cerute de metoda secantei.

eps: Toleranţa

lnit: Numărul limită de iteraţii (intrare); numărul efectiv de iteraţii (ieşire)

rad: Rădăcina calculată

kod: Intrare: cod de tipărire a iteraţiilor: 0 (nu); ≠ 0 (da)

Ieşire: cod de încheiere a iteraţiei (intern)

0: toleranţa *eps* atinsă

1: depăşirea numărului maxim de iteraţii admis *lnit*

2: Proces staţionar

-1: divergenţă

-2: numitor nul în formula metodei.

Teste:

Testul pentru atingerea toleranţei *eps* este

$|x2 - x1| \leq eps$

Testul pentru divergenţa locală este

$|x2 - x1| \geq |x1 - x0|$,

unde *x*0, *x*1, *x*2, sunt trei iterate succesive. Acest test nu opreşte iteraţia.

Testul de numitor nul este

$|f(x1) - f(x0)| \leq eps\_f$,

unde toleranţa este aleasă ca $eps\_f = 2E-7$ (această valoare se poate modifica).

Datele de intrare se introduc de la terminal.
Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator, şi conţin:

- expresia funcţiei *f* (scrisă de utilitarul Function_expression)
- [*p*, dacă există]
- datele de intrare *x*0, *lnit*, *eps*
- diferenţa curentă *dif*, la încheierea iteraţiei
- modul de încheiere a iteraţiei (mesaj, conform codului *kod*)
- rădăcina calculată *rad*; valoarea *f*(*rad*)
- numărul efectiv de iteraţii.

În cazul tipăririi iteraţiilor ($ktip \neq 0$), dacă procesul manifestă divergenţă se scrie mesajul "divergenţă locală".

Datele de intrare pentru exemplul de test (3 rădăcini) se dau mai jos. Ca aproximaţii iniţiale se iau capetele intervalelor din metoda bisecţiei.
*x*0, *x*1: -1, 0; 0, 1; 3, 4;
eps: 1E-6
lnit: 10; 10; 10
ktip: 0 sau 1
Rezultate:

```
Toleranţa eps atinsă.
Rădăcini / Număr de iteraţii
-0.4589623 7;   0.9100076  6;   3.733079   7;
```

Observaţie

Ca aproximaţii iniţiale s-au luat capetele intervalelor considerate în metoda bisecţiei, şi aceeaşi toleranţa *eps* ca în metoda bisecţiei. Aceasta, pentru a compara rezultatele metodei secantei cu cele din metoda bisecţiei (anume: valoarea rădăcinii şi numărul de iteraţii) ■

Newton

Metoda Newton (o singură ecuaţie).

Fişiere:

Newton.f90: subrutina metodei

Main_Newton.f90: programul principal

sub.f90: subprogram care defineşte (şi returnează) valorile funcţiei *f*(*x*) şi derivatei $f'(x)$. Expresia funcţiei *f*(*x*) poate include un parametru *p*.

Period.f90: funcţie care returnează perioada procesului staţionar

Function_expression; Function_expression-2.f90; width.f90: utilitare.

Metoda:

Funcţie *f* continuă şi derivabilă, în vecinătatea unei rădăcini α.

Formula de iterare este

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}; \quad n \geq 0; \quad x_0 = \text{dat}$$

Dacă x_0 este suficient de aproape de rădăcina simplă α (şi f, f' şi f'' sunt continue), şirul x_n converge către α, cu ordinul de convergenţă 2.

Subrutina metodei este:

Newton(SUB, x0, eps, lnit, rad, kod, n_div)

Parametrii formali sunt:

SUB: Numele subrutinei care returnează valorile funcţiei şi derivatei

*x*0: Aproximaţia iniţială

eps: Toleranţa

lnit: Numărul limită de iteraţii (intrare); numărul efectiv de iteraţii (ieşire)

rad: Rădăcina calculată

kod: Intrare: cod de tipărire a iteraţiilor: 0 (nu); ≠ 0 (da)

Ieşire: cod de încheiere a iteraţiei (intern)

0: toleranţa *eps* atinsă

1: depăşirea numărului maxim de iteraţii admis *lnit*

2: Proces staţionar

-1: divergenţă

-2: derivata funcţiei in x0 este aproximativ nulă.

n_div: Numărulde divergenţe locale

Subrutina *SUB* este:

sub(f, f1, x)

f: Valoarea funcţiei pe *x*; Expresia funcţiei poate include un parametru *p*; dacă există, valoarea lui *p* este dată de intrare.

*f*1: Valoarea derivatei pe *x*

x: Aproximaţia curentă

Observaţie – Teste:

Se defineşte diferenţa curentă *dif*, prin:

$dif = x - x0$,

unde *x* este valoarea iteratei curente, şi *x*0 valoarea iteratei la pasul anterior.

- Testul pentru atingerea toleranţei *eps* este:

  $|dif| \le eps$

- Testul pentru divergenţă este

  $|dif| > |dif\_ant|$,

  unde *dif_ant* este diferenţa *dif* de la pasul anterior.

  Pentru analiza tipului de divergenţă, satisfacerea testului de divergenţă nu opreşte iteraţia, ci produce numai un mesaj în fişierul de ieşire.

- Testul pentru derivată nulă pe *x* este

  $|f1| \le 1E-10$

  unde $f1 = f'(x)$.

- Se mai testează întâlnirea unui proces staţionar.

■

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator, şi conţin:

- expresia funcţiei *f* (scrisă de utilitarul Function_expression)
- [*p*, dacă există]
- datele de intrare *x*0, *lnit*, *eps*
- diferenţa curentă *dif*, la încheierea iteraţiei
- [*n_div*, dacă există divergenţe locale]
- modul de încheiere a iteraţiei (mesaj, conform codului *kod*)
- rădăcina calculată *rad*; valoarea *f*(*rad*)
- numărul efectiv de iteraţii.

Datele de intrare pentru exemplul de test (3 rădăcini), sunt:

*x*0: -0.5; 0.5; 3.5;

eps: 1E-6

lnit: 10; 10; 10

ktip: 0 sau 1

Rezultate:

```
* Toleranta EPS s-a atins:
Rădăcini / Număr de iteraţii
-0.4589623 4;    0.9100076  5;    3.733079   5;
```

Observaţie

Ca aproximaţii iniţiale *x*0, s-au luat mijloacele intervalelor considerate în metodele bisecţiei şi secantei, şi aceeaşi toleranţă *eps*. S-a procedat astfel, pentru a compara rezultatele metodei Newton cu cele din metodele bisecţiei şi secantei (valoarea rădăcinii şi numărul de iteraţii).

Aproximaţii iniţiale mai bune pot fi obţinute de pe graficul funcţiei *f*. Cu o aproximaţie *x*0 mai apropiată de rădăcină, numărul de iteraţii necesar pentru atingerea toleranţei *eps* va fi mai mic ■

Proces staţionar în metoda Newton

Fie funcţia:

$f(x) = 7x - 5x\cos x + 3\sin x$

şi iterăm cu aproximaţia iniţială $x_0 = 5$. (În fapt, nu există rădăcini în vecinătatea lui x_0).

Iterarea în metoda Newton duce la divergenţă locală (întâlnită la pasul 3). Dacă nu oprim procesul iterativ, atunci, la iteraţia 19, apare rezultatul:

```
Iteratia 19: x(19) = x(17)

 ** Proces stationar de perioada = 2

    17           6.427927

    18           5.017017

    19           6.427927
```

Rezultatul se verifică în figura de mai jos, în care se reprezintă graficul funcţiei *f* şi al tangentelor la graficul lui *f*, în punctele de abscisă `5.017017` şi `6.427927`.

Procesul staţionar apare dacă iteraţia nu este oprită când se întâlneşte divergenţa – cum se face în mod obişnuit.

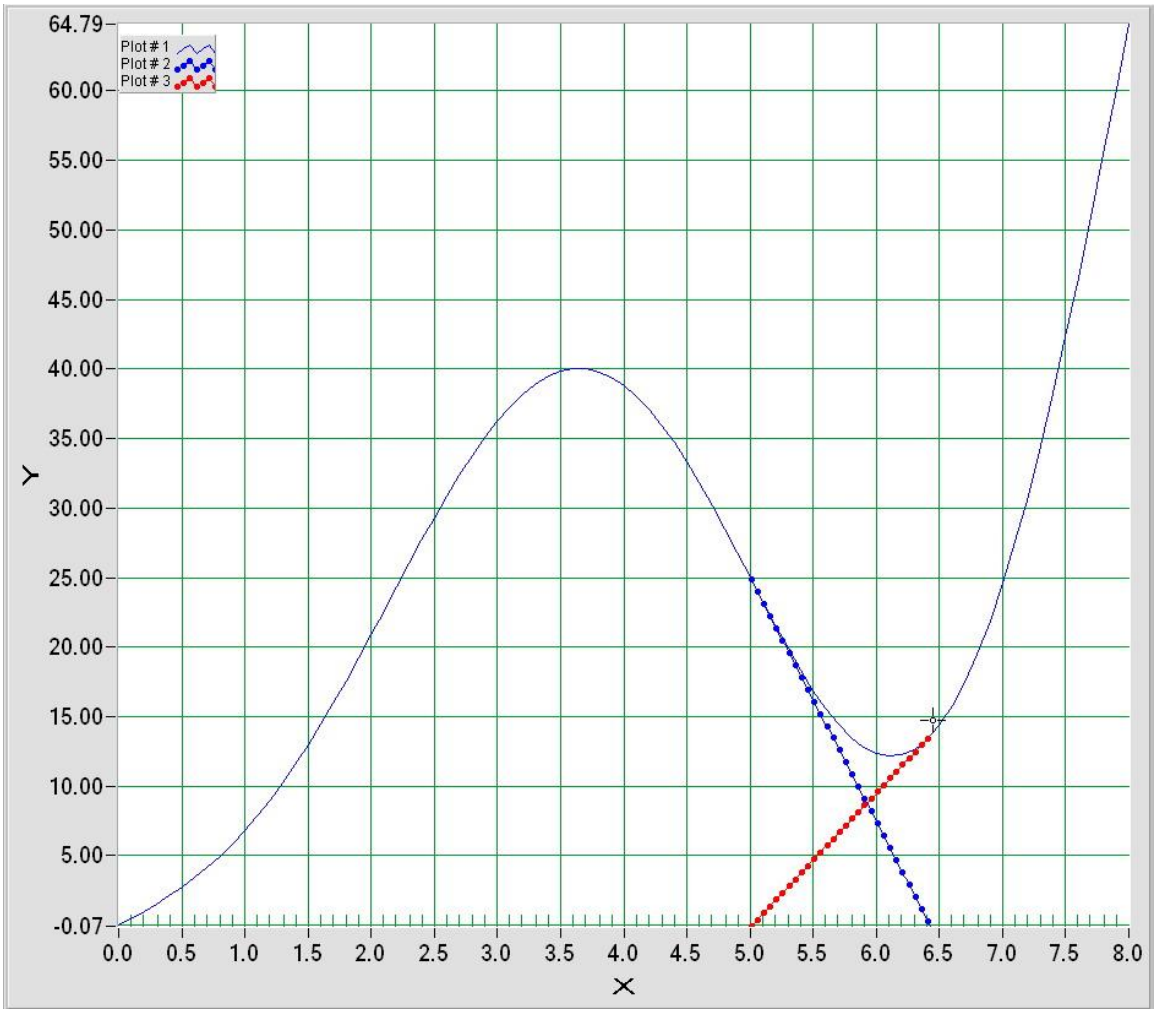


Figura 2 - Proces staţionar în metoda Newton

Exerciţiu:

Iteraţi cu: *x*0 = 6.12; *eps* = 1E-7 şi *nlim* = 1000. Rezultat: proces staţionar de perioadă 5 ■

Observaţie

Pentru acest exemplu, metoda secantei cu aproximaţiile iniţiale 4.9 şi 5.1 dă divergenţă locală în primii paşi, convergând apoi la rădăcina $x = 0$ ■

Newton-numeric

Structura de fişiere este aceeaşi.

Derivata se calculează numeric, prin aproximația:

$$\tilde{f}'(x) = \frac{f(x+h) - f(x)}{h},$$

unde h = ”mic”. Valori recomandate: $h = 10^{-2} \ldots 10^{-4}$.

Cu $f'(x) \approx \tilde{f}'(x)$, formula de iterare devine

$$x_{n+1} = x_n - \frac{f(x_n)}{\tilde{f}'(x_n)}; \quad n \geq 0$$

■

Newton_r.

Subrutina Newton_r tipăreşte, în plus, valorile $lambda = (x2 - x1)/(x1 - x0)$, pentru identificarea rădăcinilor multiple ale lui f.

Structura de fişiere e cea de la Newton, la care se adaugă:

polval.F90: funcţie care returnează perioada procesului staţionar
work_pol.f90: modul pentru date comune

Newton_r permite lucrul cu o funcție-polinom, definită prin grad și coeficienți.

Aceste date se citesc dintr-un fișier de intrare, cu structura:

- Titlu (text ≤ 80 caractere);
- Grad n;
- Coeficienții: $a_n, a_{n-1}, \ldots, a_0$.

Valoarea polinomului și a derivatei se calculează pe baza coeficienților, cu funcția polval. V. detalii în Pol_Direct.

Observație

Evident, polinomul și derivata sa se poate defini, și ca funcții, în subrutina SUB.

■

II-2 Ecuaţii de forma *x* = *g*(*x*). Metoda punctului fix.

Ecuaţia de test este $x = g(x)$, echivalentă cu $f(x) = 0$, unde $f(x) = e^x - 3x^2$.

$g(x)$ se determină prin prima procedură explicită de punct fix, anume

$g(x) = x - \Phi(x) f(x)$, unde $\Phi(x) = p$ = constant. Pentru valorile lui p – v. mai jos.

Fix

Metoda punctului fix (o singură ecuaţie).

Fişiere:

Fix2005.f90: subrutina metodei

Main_Fix2005.f90: programul principal

G.f90: subprogram de tip `function` care defineşte funcţia g(x)

Period.f90: subrutină pentru perioada procesului staţionar

Function_expression_g.f90; width.f90: utilitare

Metoda:

Funcţie *g*, continuă şi derivabilă pe o vecinătate $[a,b]$ a rădăcinii α.

Formula de iterare este

$x_{n+1} = g(x_n)$; $n \geq 0$; x_0 = dat

Dacă g' este continuă, $\lambda = \max_{[a,b]} |g'(x)| < 1$, şi $x_0 \in [a,b]$, şirul $x_n \to \alpha$, liniar.

Subrutina metodei este:

Fix(g, x, xtol, nlim, kod)

Parametrii formali sunt:

g: Numele funcţiei *g*(*x*), declarat external în programul principal. Expresia funcţiei poate include un parametru *p*.

x: Intrare: aproximaţia iniţială; Ieşire: soluţia calculată / ultima aproximaţie.

xtol: Toleranţa pentru diferenţa a două valori succesive ale lui *x*.

nlim: Numărul limită de iteraţii.

kod: Intrare: cod pentru tipărirea iteraţiilor: 0 (Nu); ≠ 0 (Da)

kod ≥ 2: tipăreşte şi valoarea $lambda = (x2 - x1)/(x1 - x0)$,

unde *x*0, *x*1 şi *x*2 sunt trei iterate succesive.

Ieşire: cod de încheiere a iteraţiei:

0: s-a obţinut toleranţa *xtol*.

1: depăşire a lui *nlim*.

2: proces staţionar.

-1: divergenţă.

Testul pentru atingerea toleranţei *xtol* este

$| x1 - x0 | \leq xtol$,

unde $x1$ este iterata curentă, şi $x0$ iterata anterioară.

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator,

şi conţin:

- expresia funcţiei *g* (scrisă de utilitarul Function_expression_g)
- [*p*, dacă există]
- datele de intrare $x0$, *nlim*, *xtol*
- modul de încheiere a iteraţiei (mesaj)
- indicele ultimei iteraţii, rădăcina calculată *x*, valoarea $g(x)$
- *diferenţa curentă* $= x - g(x)$

Nota

Sub-folderul **Dfix** conţine fişierele pentru metoda în dublă precizie ■

Funcţia *g*, pentru ecuaţia de test, este

$g(x) = x - p(e^x - 3x^2)$,

unde *p* este o constantă. Se iau valorile: $p = 0.01$; $p = 0.05$;

Datele de intrare pentru exemplul de test – rădăcina a treia, sunt:

$x0$: -0.5 0.5 3.7

xtol: 2.4E-7; *nlim*: 100;

Rezultate:

p = 0.01

```
      Iteratia         x                    g(x)
           47          3.733078             3.733078
** Diferenta curenta = 2.3841858E-07
```

$p = 0.05$

```
            5          3.733079              3.733079
** Diferenta curenta = 0.000000
```

Notă

A doua valoare pentru p este apropiată de $p\_optim \approx 0.055$; v. Exemplul din Cap. 3-II, 13.4.2 ■

Exemplu de proces staţionar de perioadă 2:

Se consideră funcţia

$$g(x) = \frac{1.78 - 0.2tg(x)}{1 + tg(x)},$$

şi se iterează cu $x0 = 0.8$, tol = 1E-7, $nlim$ = 1000. Se obţine rezultatul:

```
Iteratia 844: x(844) = x(842)

** Proces stationar de perioada = 2

Iteratia        x

   842          0.7877131

   843          0.7877082

   844          0.7877131

** Diferenta curenta =   4.8279762E-06
```

Explicaţie:

Derivata $g'(x)$, calculată în simplă precizie pe intervalul [0.7877080, 0.7877140], are valoarea – 0.9900053, care este foarte apropiată de valoarea -1. Astfel, este realizată condiţia pentru procesul staţionar de perioadă 2: $g'(x) \approx -1$.

■

Exerciţiu:

Iteraţi în dublă precizie, cu $x0 = 0.8$, tol = 1E-14, $nlim$ = 3000.

Stationar

Metoda punctului fix, proces staţionar: grafice.

Fişiere:

Stat2005.f90: programul principal

G.f90: funcţia *g*

Function_expression_g.f90: utilitar

Programul are ca ieşire fişierele pentru reprezentarea grafică a unui proces staţionar, anume:

Nume-iter.rez: iteratele;

Nume-g.rez, nume-bis.rez: funcţia *g* şi prima bisectoare.

Datele de intrare se introduc de la terminal. Acestea sunt:

- *x*0, *n*, *n*1: Aproximaţia iniţială; număr de iterate; număr de iterate omise;
- *Nume*: "Nume" – pentru construirea numelor fişierelor de ieşire.

Datele de ieşire constau din două linii-text, urmate de date numerice, şi anume:

- *Linie-text*-1: Expresia funcţiei *g*
- *Linie-text*-2: Tipul de date din fişier, anume: iterate; funcţia g; bisectoarea I-a.
- *x*, *y*: Perechi de date numerice, anume:
  - Nume-iter.rez: (k, x_k): Indice iterată – iterată
  - Nume-g.rez: $(x_k, g(x_k))$
  - Nume-bis.rez: (x_k, x_k)

  k ia valorile: $k = n1+1, n$.

Notă

Dacă programul de grafică nu are capabilitatea de a recunoaşte linii-text (şi a le ignora), se vor şterge cele două linii-text din fişier (primele două linii), înainte de încărcarea fişierului în programul de grafică.

În particular, programul GRAPH are această capabilitate ■

Exemplu:

Pentru exemplul de proces staţionar din metoda punctului fix, datele de intrare sunt:

0.8; 1000; 800;

Se omit 800 de iterate, pentru a reprezenta procesul staţionar (care începe la iterata 842), prin iteratele 801-1000.

Graficele iteratelor şi funcţiei *g* sunt date mai jos.

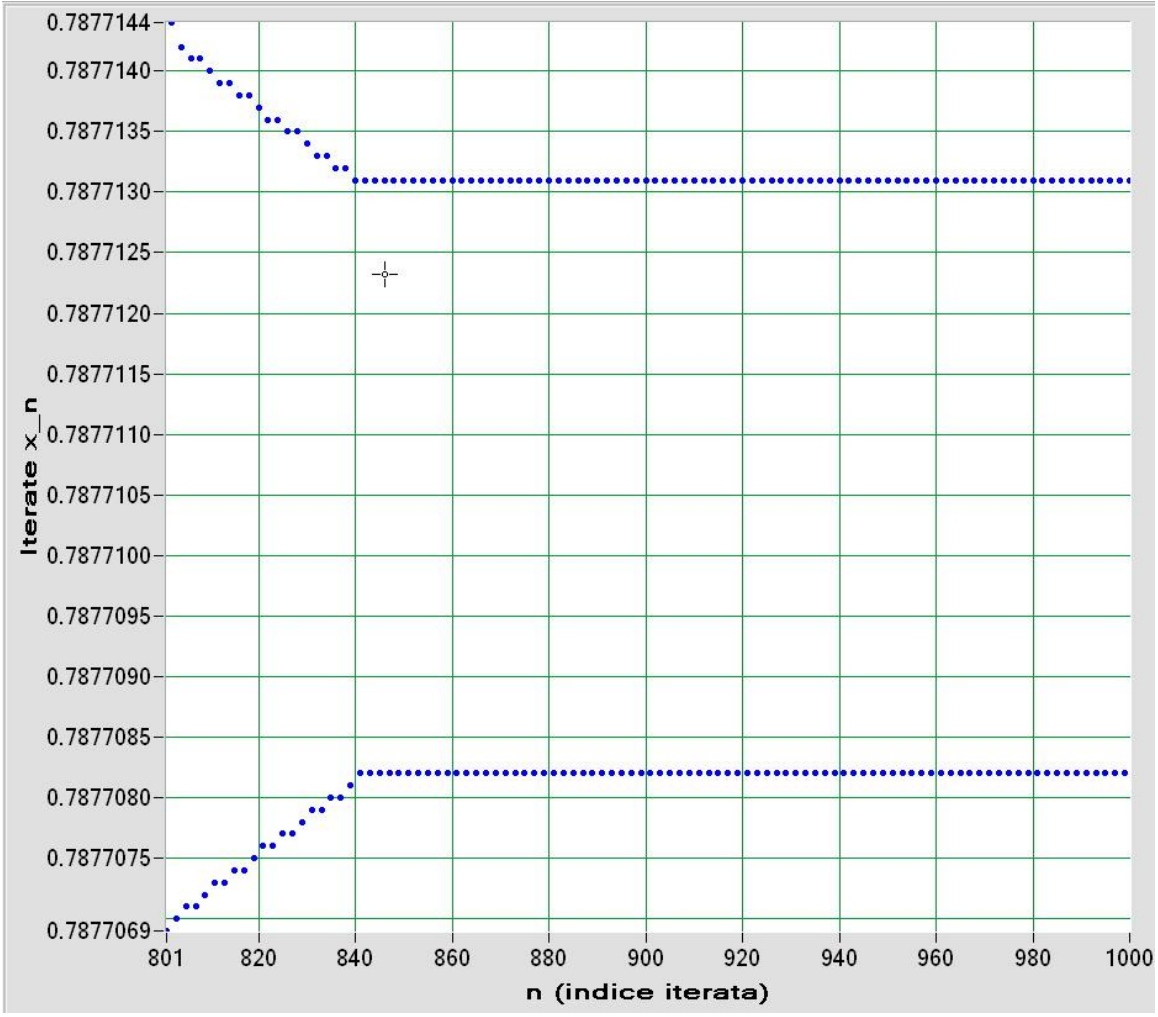


Figura 3 - Proces staţionar – Graficul iteratelor

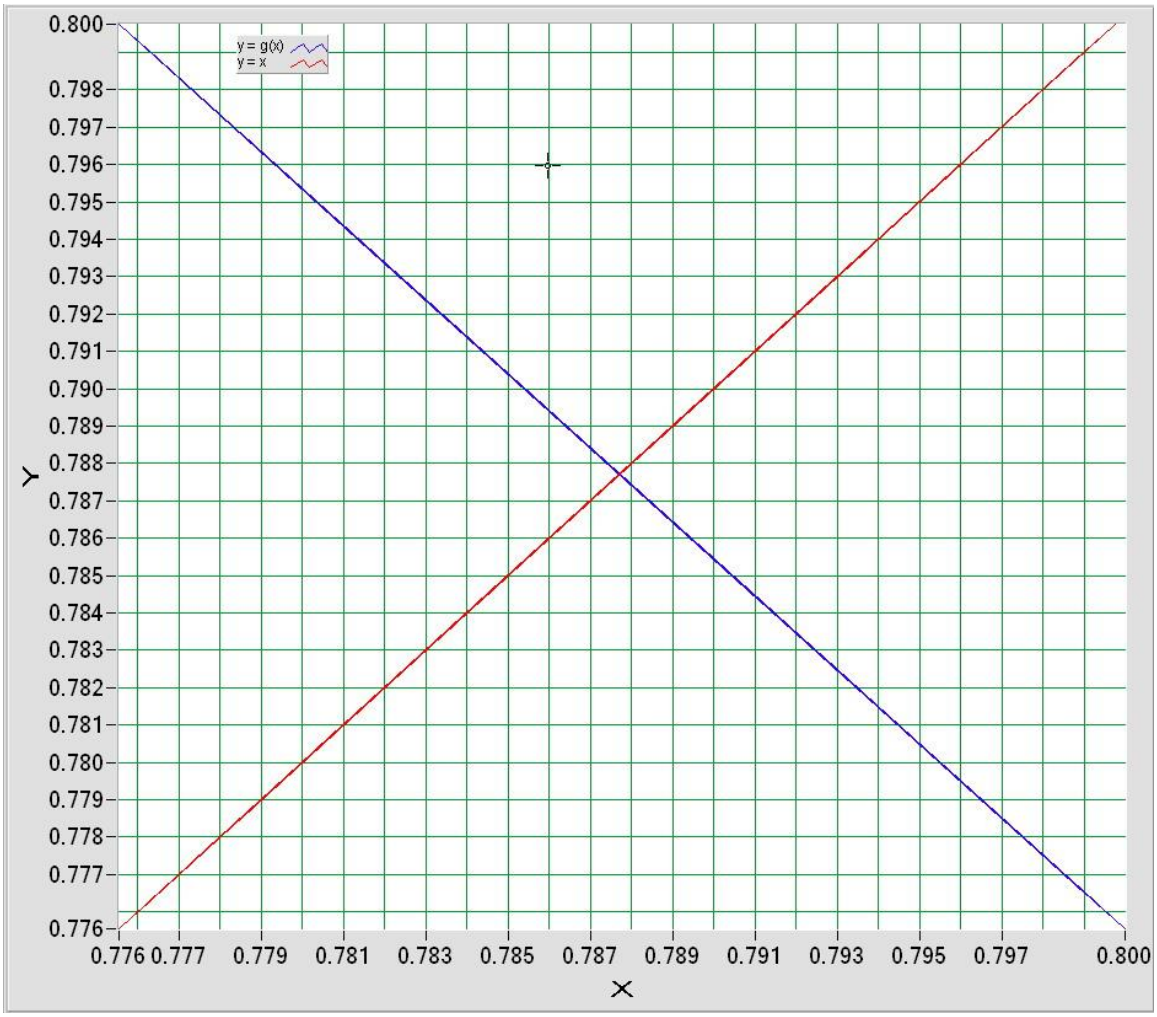


Figura 4 - Proces staţionar – Graficul funcţiei şi primei bisectoare

Aitken

Accelerarea Aitken.

Fişiere:

Aitken2005.f90: subrutina metodei

Main_Aitken2005.f90: programul principal

ga.f90: funcţia *g* pentru Aitken

Period.f90: perioada procesului staţionar

Function_expression_g.f90; width.f90: utilitare

Metoda:

Cu trei iterate succesive x_n, x_{n+1}, x_{n+2}, calculate ca în metoda punctului fix, se calculează

$$a_{n,\,n+2} = x_n - \frac{(x_{n+1} - x_n)^2}{(x_{n+2} - x_{n+1}) - (x_{n+1} - x_n)},$$

Valoarea $a_{n,\,n+2} \approx \alpha$ este o aproximaţie a rădăcinii (mai bună decât iterata următoare, $x_{n+3} = g(x_{n+2})$).

Procesul iterativ este următorul:

$x_0 = dat;\ x_1 = g(x_0);\ x_2 = g(x_1);$

$x_3 = a_{0,2};\ x_4 = g(x_3);\ x_5 = g(x_4);$

$x_6 = a_{3,5};$ …

Etc.

Metoda Aitken are cel puţin ordinul 2, într-o rădăcină simplă.

Subrutina metodei este:
aitken(g, x0, eps, lnit, rad, kod)

Parametrii formali au, în general, aceeaşi semnificaţie ca la metoda punctului fix.
Explicit:

g: Numele funcţiei *g*(*x*), declarat `external` în programul principal. Expresia funcţiei poate include un parametru *p*.

*x*0: Intrare: aproximaţia iniţială; Ieşire: soluţia calculată / ultima aproximaţie.

eps: Toleranţa – v. mai jos.

lnit: Numărul limită de iteraţii.

kod: Intrare: cod pentru tipărirea iteraţiilor: 0 (Nu); ≠ 0 (Da)

kod ≥ 2: tipăreşte şi valoarea $lambda = (x2 - x1)/(x1 - x0)$,

unde *x*0, *x*1 şi *x*2 sunt trei iterate succesive.

Ieşire: cod de încheiere a iteraţiei:

0: s-a obţinut toleranţa *eps*

1: depăşire a lui *lnit*

2: proces staţionar

-1: numitor nul în formula metodei

Formula metodei este codată astfel: iteratele curente sunt `x0,x1,x2`.

```
num =(x2-x1) -(x1-x0)
cor =(x1-x0)**2/num
a =x0 -cor
```

unde `num` este *numitorul*, iar `cor` este *corecţia* la *x*0.

Testele pentru atingerea toleranţei *eps* sunt:

1) testul obişnuit în metoda punctului fix:
   $|x2 - x1| \le eps$
2) testul pentru corecţie:
   $dif = a - x0; \quad |dif| \le eps$

Testul de numitor nul este *num* = 0.

Datele de intrare se introduc de la terminal. În programul principal, parametrul efectiv pentru toleranţă este numit *xtol*.

Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator,

şi conţin:

- expresia funcţiei *g* (scrisă de utilitarul Function_expression_g)
- [*p*, dacă există]
- datele de intrare *x*0, *lnit*, *xtol*
- modul de încheiere a iteraţiei (mesaj)
- indicele ultimei iteraţii, rădăcina calculată *rad*, valoarea *g*(*rad*)

Exemplu de test:

Se consideră aceeaşi funcţie *g*, ca la metoda punctului fix:

$g(x) = x - p(e^x - 3x^2)$,

unde *p* este o constantă.

Se caută rădăcina din intervalul [3, 4], cu: aproximaţia iniţială *x*0 = 3.7; *xtol* = 2.4E-7; *lnit* = 10. Se iterează pentru valori $p = 0.01, 0.02, \ldots, 0.09$, şi $p = 0.001$.

Rezultate – exemplificare pentru $p = 0.01$; $p = 0.001$:

```
p =  9.9999998E-03
Radacina nr. 1
Iteratia 3:  3.733080
 * x(3) -x(2) =-2.3841858E-07

* Toleranta intalnita
 Radacina =   3.733080    ; Numar iteratii = 3
 g(Rad)   =   3.733080

p =  1.0000000E-03
Radacina nr. 2
x2-x1; x1-x0:  2.1457672E-06  2.1457672E-06

** Numitor nul
 Radacina =   3.732974    ; Numar iteratii = 5
 g(Rad)   =   3.732976
```

Observaţie

Pentru comparaţia numărului de iteraţii din metoda Aitken, cu cele din metoda punctului fix:

Într-un pas, Aitken calculează două iterate ($x1$ şi $x2$, cu $x0$ dat); astfel, numărul de iterate calculate este 2 × *numărul iteraţiilor* afişat de Aitken.

Pentru $p = 0.01$: Aitken calculează 6 iterate, faţă de 47 în metoda punctului fix ■

Aitken\Aitken-G

Sub-folder, pentru metoda Aitken cu funcţia $G(x)$.

Fişiere:

Aitken_G2005-2 .f90; Main-Aitken_G2005.f90; ga.f90 9; Period.f90; Function_expression_g.f90; width.f90.

Semnificaţia fişierelor este aceeaşi ca în Aitken.

Metoda este:

$x_{n+1} = G(x_n)$,

unde

$$G(x) = x - \frac{(g(x)-x)^2}{g(g(x))-2g(x)+x}$$

Subrutina metodei este:
aitken_G(g, x0, eps, lnit, rad, kod)

Parametrii formali au aceeaşi semnificaţie ca la Aitken.
Metodei este codată astfel: iteratele curente sunt *a* şi *a*0.

```
num =g(g(a0)) -2*g(a0) +a0
cor =-(g(a0)-a0)**2/num
a =x0 +cor
```

Testele pentru toleranţă se pun astfel:

1) testul din metoda punctului fix:
   $|g(a0)-a0| \le eps$
2) testul pentru corecţie:
   $dif = a - a0;\ |dif| \le eps$.

Datele de intrare se introduc de la terminal. Parametrul efectiv pentru toleranţă este numit *xtol* (în programul principal).
Rezultatele se scriu în fişierul de ieşire nume.rez, unde nume este introdus de utilizator, şi conţin aceeaşi date ca rezultatele din Aitken.

Pentru exemplul de test din Aitken, se obţin rezultatele:

```
p =  9.9999998E-03
* XTOL intalnit
 Radacina =   3.733078    ; Numar iteratii = 4
 g(Rad)   =   3.733078

p =  1.0000000E-03
* XTOL intalnit
 Radacina =   3.733090    ; Numar iteratii = 17
 g(Rad)   =   3.733090 ■
```

II-3 Sisteme de ecuaţii neliniare

Se consideră sistemul de *n* ecuaţii cu *n* necunoscute

$$f_1(x_1, x_2, \ldots, x_n) = 0$$
$$\ldots\ldots$$
$$f_n(x_1, x_2, \ldots, x_n) = 0$$

Vectorial, sistemul se scrie

$$\mathbf{f}(\mathbf{x}) = \mathbf{0},$$

unde

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix}.$$

Se notează tot cu **x**, şirul argumentelor lui f_i, adică $x_1, \ldots, x_n$.

Pentru metoda punctului fix, sistemul dat se pune sub forma sistemului echivalent

$$\mathbf{x} = \mathbf{g}(\mathbf{x}).$$

Aceasta se realizează prin procedura explicită de punct fix $\mathbf{g}(\mathbf{x}) = \mathbf{x} - \mathbf{A}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})$,

unde **A**(**x**) este o matrice nesingulară.

Exemplu de test:

$$f_1(x, y) \equiv x^2 + y^2 - 5 = 0, \qquad f_2(x, y) \equiv y - e^x - 1 = 0.$$

Aproximaţiile iniţiale se iau:

$\mathbf{x}^{(0)} = (-2, 1)$, şi $\mathbf{x}^{(0)} = (0.5, 2)$

Acestea se determină din intersecţia graficelor celor două curbe.

Notă

Dificultatea majoră constă în găsirea aproximaţiilor iniţiale, în cazul $n \geq 3$ ■

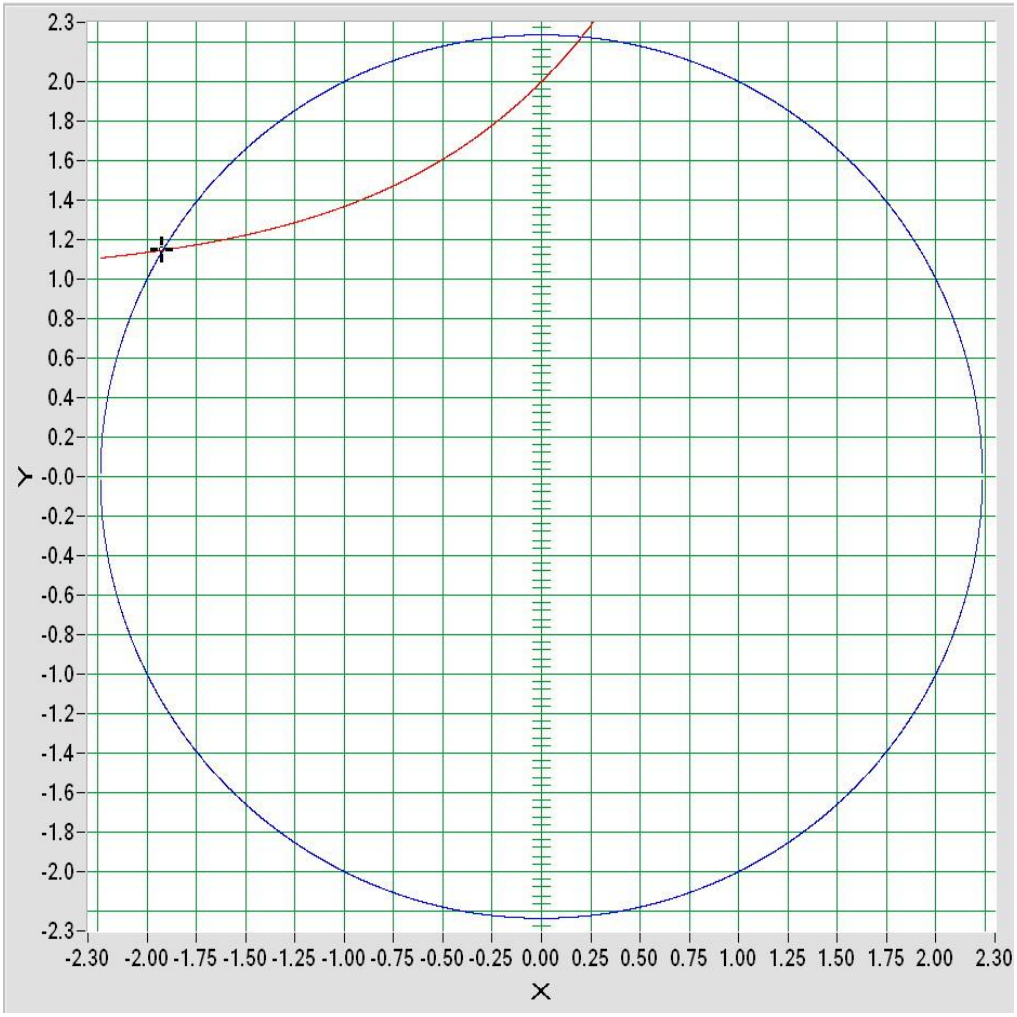


Figura 5 – Determinarea aproximaţiilor iniţiale

Fix_Sys

Metoda punctului fix – pentru sisteme de ecuaţii neliniare.

Fişiere:

Fix_Sys2005.f90: subrutina metodei

Main-Fix_Sys2005.f90: programul principal

fcns.f90: subrutina care returnează valorile funcţiilor $f_i(\mathbf{x})$

norm$.f90: norma-$\infty$ a unui vector `v(n)`

Elim.f90: subrutina pentru eliminarea Gauss

Function_expression_Sys.f90; width.f90: utilitare

Metoda:

Iterare cu matricea constantă $\mathbf{A} = \left[\mathbf{F}(\mathbf{x}^{(0)})\right]^{-1}$, cu actualizare după 3 paşi.

Iteraţia este definită de

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \mathbf{A} \cdot \mathbf{f}(\mathbf{x}^{(n)})$$

F(**x**) este jacobianul lui **f**. Derivatele parţiale ale funcţiilor $f_i(\mathbf{x})$ (elementele jacobianului) se calculează numeric, cu variaţia $delta = 0.001$.

Schema de iterare este următoarea. Se defineşte

$$\boldsymbol{\delta}^{(n+1)} = \mathbf{x}^{(n+1)} - \mathbf{x}^{(n)},$$

şi rezultă:

$$\mathbf{F}(\mathbf{x}^{(0)}) \cdot \boldsymbol{\delta}^{(n+1)} = -\mathbf{f}(\mathbf{x}^{(n)}) \qquad \text{(a)}$$

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \boldsymbol{\delta}^{(n+1)}$$

Ecuaţia (a) este un sistem liniar – în necunoscuta $\boldsymbol{\delta}^{(n+1)}$, care se rezolvă prin eliminare Gauss.

Testele de oprire a iteraţiei sunt:

$\| \boldsymbol{\delta}^{(n+1)} \| \le eps$

$n+1 \le lnit$

Metoda are un ordin de convergenţă $1 < p < 2$.

Subrutina metodei este

FIX_SYS(fcns, n, x, xtol, delta, lnit, f, kod),

Parametrii formali sunt:

n: ordinul sistemului (numărul de ecuaţii)

fcns: numele subrutinei care calculează valorile funcţiilor $f_i(\mathbf{x})$

x: tablou `x(n)` pentru valorile $x_1, \ldots, x_n$

xtol: toleranţa

delta: variaţia pentru calculul derivatelor parţiale

lnit: numărul limită de iteraţii

f: tablou `f(n)`, pentru valorile $f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})$

kod: cod de încheiere a iteraţiei (intern):

0: toleranţa atinsă

1: numărul limită de iteraţii atins

-1: Pivot < prag in eliminarea Gauss

Exemplul de test:

Se obţin rezultatele:

```
 Radacina nr.1
  Aproximatia initiala:  -2.000000       1.000000
  xtol: 1.000E-06; lnit: 10
 XTOL atins
   Solutia X:   -1.919684       1.146653
   F(X):       -2.7614632E-07 -2.9958017E-08
   Numar de iteratii = 5


  Radacina nr.2
  Aproximatia initiala:  0.5000000       2.000000
  xtol: 1.000E-06; lnit: 20
 XTOL atins
   Solutia X:   0.2043374       2.226712
   F(X):        7.2109735E-08 -4.6549125E-08
   Numar de iteratii = 17
```

■

Newton_Sys

Metoda Newton – pentru sisteme de ecuaţii neliniare.

Fişiere:

Newton_Sys.f90: subrutina metodei

Main-Newton_Sys.f90: programul principal

fcn.f90: subrutina care returnează valorile funcţiilor $f_i(\mathbf{x})$

jfcn.f90: subrutina care returnează jacobianul lui $\mathbf{f}(\mathbf{x})$

norm$.f90: norma-$\infty$ a unui vector `v(n)`

Elim.f90: subrutina de eliminare Gauss

Period_Sys.f90: funcţie care returnează perioada procesului staţionar

Function_expression_Sys.f90, width.f90: utilitare

Metoda:

Iteraţia este definită de

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \left[\mathbf{F}(\mathbf{x}^{(n)})\right]^{-1} \cdot \mathbf{f}(\mathbf{x}^{(n)}),$$

unde $\mathbf{F}(\mathbf{x})$ este jacobianul lui $\mathbf{f}$:

$$\mathbf{F}(\mathbf{x}) = \left[\frac{\partial f_i}{\partial x_j}\right]_{\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\mathbf{x}}$$

Elementele jacobianului se calculează analitic.

Schema practică de iterare este:

$$\mathbf{F}(\mathbf{x}^{(n)}) \cdot \boldsymbol{\delta}^{(n+1)} = -\mathbf{f}(\mathbf{x}^{(n)}) \qquad \text{(a)}$$

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \boldsymbol{\delta}^{(n+1)}$$

Corecţia $\boldsymbol{\delta}^{(n+1)}$ se calculează prin rezolvarea sistemului liniar (a).

Iteraţia se opreşte prin testele:

$\| \boldsymbol{\delta}^{(n+1)} \| \le eps$,

Număr de iteraţii $\le$ *lnit*

unde *eps* şi *lnit* sunt toleranţa, şi respectiv, numărul limită de iteraţii.

Pe lângă acestea, iteraţia se mai opreşte dacă Jacobianul este singular, sau dacă se întâlneşte un proces staţionar.

Metoda are ordinul de convergenţă $p = 2$.

Subrutina metodei este

Newton_Sys (fcn, jfcn, n, x, lnit, xtol, f, kod, n_div)

Parametrii formali sunt:

n: ordinul sistemului (numărul de ecuaţii)

fcn: numele subrutinei care calculează valorile funcţiilor $f_i(\mathbf{x})$

jfcn: numele subrutinei care calculează jacobianul lui **f**; elementele jacobianului sunt returnate în tabloul `jf(n,n)`.

X: tablou `x(n)` pentru valorile $x_1, \ldots, x_n$

xtol: toleranţa

lnit: numărul limită de iteraţii

f: tablou `f(n)`, pentru valorile $f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})$

kod: cod de încheiere a iteraţiei (intern):

0: toleranţa XTOL atinsă

1: numărul limită de iteraţii atins

2: Proces staţionar

-1: Divergentă

-2: Pivot < prag in eliminarea Gauss (Jacobian singular)

Exemplul de test:

Se obţin rezultatele:

```
Radacina nr.1
```

```
Aproximatia initiala:  -2.000000       1.000000
xtol: 1.000E-06; lnit: 10

 Norma corectiei:  5.9217999E-08
  XTOL atins
 Solutia X:   -1.919684       1.146653
 F(X):       -2.7614632E-07 -2.9958017E-08
 Numar de iteratii = 4

Radacina nr.2
Aproximatia initiala:  0.5000000       2.000000
xtol: 1.000E-06; lnit: 10

 Norma corectiei:  1.1825179E-08
  XTOL atins
 Solutia X:   0.2043374       2.226712
 F(X):        5.3840544E-08  8.2891738E-09
 Numar de iteratii = 5
```

■

II-4 Rădăcinile polinoamelor

C_pol

Calculul valorii polinomului, direct şi imbricat (schema Horner), pe un interval $[a,b]$, cu pasul h.

Fişiere:
C_pol2005.f90; sub_pol.f90

Metoda:
Exemplificăm pe un polinom de gradul 4. Generalizarea este imediată.
Forma dată:

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

Forma imbricată:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x)))$$

Calculul efectiv se face cum urmează:

1. Calculul prin ciclu:

a) Forma dată:

```
p =a(0)
zk =1.
do k =1,n
zk =zk*z
p =p +a(k) *zk
enddo
```

Observaţi că, pentru minimizarea numărului operaţiilor, puterile lui *x* se calculează succesiv prin $x^{k+1} = x^k \cdot x$.

b) Forma imbricată: Calculul pe forma imbricată face în următorul ciclu, care calculează succesiv expresiile din paranteze:

```
pi =a(n)
pid =dble(a(n))
do  k =n-1,0,-1
pi =a(k) +z*pi
pid =dble(a(k)) +dble(z)*pid
enddo
```

pi şi *pid* sunt valorile în simplă, respectiv dublă precizie, în forma imbricată. (`pid` este declarată în dublă precizie.)

2. Calculul prin funcţia *p*(*x*), în simplă precizie:

Se calculează valoarea polinomului, în forma dată şi cea imbricată. În acest calcul, membrul drept se calculează în format dublu extins (64 biţi, în coprocesor), şi rezultatul se rotunjeşte la precizia simplă.

sub_pol.f90: subrutină care calculează valoarea polinomului în modul 2 (simplă precizie).

C_pol.f90: programul principal, care calculează în modurile 1a, 1b.

Datele de intrare sunt:

- *n*: gradul polinomului
- $a_n, a_{n-1}, \ldots, a_0$: coeficienţii polinomului (în această ordine);
- *a, b, h*: intervalul $[a,b]$ şi pasul *h*

Acestea se citesc dintr-un fişier de date. Fişierul de date este utilizat şi ca fişier de ieşire pentru rezultatele *p, pi, pid*.

Exemplu de test:

$p(x) = x^3 - 3x^2 + 3x - 1 = (x-1)^3$

Fişierul de date şi rezultate:

```
3
1  -3  3  -1
0.9   1.2  .1

Calcul prin ciclu:
  x         p-Dat             p-Imbricat        p-Imbricat in
                                                Dubla precizie
  0.900    -9.9998713E-04    -9.9990633E-04    -1.0000007E-03
  1.000      0.000000          0.000000          0.000000
  1.100     1.0001659E-03     9.9998503E-04     1.0000007E-03
  1.200     8.0001354E-03     8.0000088E-03     8.0000057E-03

  x         Err(p_Dat)     Err(p_Imbricat)
  0.900    -1.359E-08    -9.438E-08
  1.000      0.00          0.00
  1.100    -1.652E-07     1.569E-08
  1.200    -1.297E-07    -3.040E-09

Calcul prin functie:
  x         p-Dat             p-Imbricat
  0.900    -1.0000007E-03     1.0000007E-03
  1.000      0.000000          0.000000
  1.100     1.0000007E-03    -1.0000007E-03
  1.200     8.0000060E-03    -8.0000060E-03
```

Se constată:

- La calculul prin ciclu: valoarea cea mai precisă este *pid* (cu excepţia valorii în $x = 0.9$).
- La calculul prin funcţie: nu există diferenţe între forma dată şi imbricată, şi valorile coincid cu valorile *pid*.

Pol

Metoda Newton pentru polinoame – Algoritmul cu reducerea gradului

Fişiere

Main-Pol.f90: programul principal

Pol2007.f90: subrutina metodei

Openfile.f90, GetFile.f90: subrutine pentru selectarea fişierului de intrare

Period.f90: funcţie pentru calculul perioadei procesului staţionar

width.f90: utilitar

Metoda

Se consideră polinomul

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Definim coeficienţii

$$b_n = a_n; \qquad b_k = a_k + \xi \cdot b_{k+1}, \qquad k = n-1, n-2, \ldots, 0$$

unde $b_0 = p(\xi)$. Definim polinomul cât, prin:

$$q(x;\xi) = b_1 + b_2 \cdot x + \ \ldots \ + b_n \cdot x^{n-1},$$

adică $p(x) = (x-\xi)\, q(x;\xi) + p(\xi)$. Avem:

$$p'(x) = q(x;x).$$

unde $q(x;x) = q(x;\xi)\big|_{\xi = x}$.

Astfel, formula de iterare Newton este

$$x_{n+1} = x_n - \frac{p(x_n)}{q(x_n;x_n)} \qquad (1)$$

Subrutina metodei: Calculează o rădăcină a lui $p(x)$.

Pol(n, a, x0, eps, itmax,rad, b, ier, n_div)

Parametri formali:

n: gradul polinomului

a: vectorul coeficienţilor polinomului: $a_0, a_1, \ldots, a_n$

*x*0: aproximaţia iniţială

eps: toleranţa pentru metoda Newton

itmax: Intrare: numărul maxim de iteraţii. Ieşire: numărul efectiv de iteraţii

rad: rădăcina / ultima aproximaţie

b: vectorul coeficienţilor lui $q(x)$: $b_1, b_2, \ldots, b_n$ (returnaţi pentru o noua rădăcină)

ier: cod de intrare/ieşire:

- Intrare: tipărirea iteraţiilor (0: da; ≠0: nu)
- Ieşire: cod de încheiere a iteraţiei:
  - 0: toleranţa *eps* atinsă
  - 1: numărul maxim de iteraţii atins
  - 2: Proces staţionar
  - -1: Divergenţă
  - -2: Numitor nul

n_div: numărulde divergenţe locale

Formula de iterare este codată astfel:

$x1 = x0 - dif$,

unde: *dif* este *corecţia* (sau *diferenţa*) curentă.

Observaţii

- Testul pentru toleranţă este:

  $|dif| \le eps$

- Testul pentru divergenţă locală este:

  $|dif| \ge |dif\_ant|$

  unde dif_ant este *dif* de la pasul anterior.

■

Programul principal

Este scris astfel ca să determine un număr specificat de rădăcini, prin reducerea succesivă a gradului (deflaţie), şi anume: după găsirea unei rădăcini, rădăcina următoare se caută ca rădăcină a lui $q(x)$, care are gradul cu o unitate mai mic decât precedentul polinom.

Rădăcinile sunt apoi, *purificate*, prin iterare în polinomul original.

Algoritmul cu reducerea gradului este mai puţin recomandat, întrucât: reducerea gradului poate fi un proces instabil; se cer, în plus, iteraţii de purificare a rădăcinilor. Alternativa este algoritmul fără reducerea gradului – v. folderul Pol_Direct.

Fişierul de intrare conţine următoarele date (scrise pe linii):

Linie_text: titlul problemei; max. 80 caractere;

n, nr, ktip: gradul polinomului, număr de rădăcini de calculat, cod de tipărire a iteraţiilor;

$a_n, a_{n-1}, \ldots, a_0$: coeficienţii polinomului: (în această ordine);

eps, itmax: toleranţa, numărul maxim de iteraţii;

*x*0(1:*nr*): aproximaţiile iniţiale, în număr de *nr*.

Datele de ieşire se scriu în continuarea datelor de intrare, şi conţin:

rad: Rădăcina calculată, sau ultima iterată dacă nu s-a atins toleranţa; Codul de încheiere a iteraţiei; Numărul de iteraţii; Valoarea *p*(*rad*).

Exemplu: Polinomul Laguerre de gradul 6.

Fişierul de intrare este:

```
Laguerre de ordinul 6
6  0  0
1 -36 450 -2400  5400  -4320  720
1e-6  40
0     1     3     5     9     15
```

Programul produce rezultatele:

```
Radacina  5.775151
 Iteratia 7: x(7) = x(5)
 ** Proces stationar de perioada = 2

 Radacina  9.837449
 Iteratia 5: x(5) = x(3)
 ** Proces stationar de perioada = 2
```

```
** Radacinile prin reducerea gradului:
 Rad                   Cod    Iteratii       p(Rad)
 0.2228466               0           5     0.0000000E+00
  1.188932               0           4    -1.8310547E-04
  2.992735               0           3     3.1127930E-03
  5.775151               2           7     1.8066406E-02
  9.837449               2           5     0.1802368
  15.98288               0           2     0.7625122

Radacina  9.837461
Iteratia 9: x(9) = x(1)
** Proces stationar de perioada = 8

Radacina  15.98288
Iteratia 6: x(6) = x(1)
** Proces stationar de perioada = 5

Radacina       Nr. de divergente locale
  5.775142               1
  9.837461               2
  15.98288               2


** Radacinile  polinomului:
 0.2228466               0           1     0.0000000E+00
  1.188932               0           1    -1.8310547E-04
  2.992736               0           3     6.1035156E-05
  5.775142               0           6    -6.1035156E-04
  9.837461               2           9    -0.1480713
  15.98288               2           6    -0.6418457

Cod   Semnificatie
  0   * Toleranta EPS atinsa
  2   ** Proces stationar
```

■

Pol_Direct

Metoda Newton pentru polinoame – fără reducerea gradului.

Metoda:

Metoda şi formula de iterare, sunt aceleaşi ca în POL. Deosebirea este că acum, programul principal calculează rădăcinile prin iterare *direct* în polinomul original (fără reducerea gradului). Acest procedeu se recomandă faţă de cel descris în POL.

Pol_Direct

Acest proiect implementează formula de iterare Newton pentru polinoame, în forma:

$$x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)} \qquad (2)$$

Deosebirea față de formula (1), este că acum, derivata se calculează direct, prin:

$p'(x) = na_n x^{n-1} + \ldots + 2a_2 x + a_1$.

Coeficienții polinomului și derivatei sunt stocați, respectiv, în tablourile `coef(0:n)` și `coef_1(0:n-1)`. Valoarea polinomului și a derivatei se calculează în formă imbricată (cu funcția polval), astfel:

```
b = polval(n, coef, x)
c = polval(n-1, coef_1, x)
```

Fișiere

Main-Pol_Direct.f90

Pol_Direct.f90

polval.F90

Openfile.f90, GetFile.f90

Period.f90

width.f90

Funcțiile fișierelor sunt aceleași ca la POL. Polval.f90 implementeazș funcția `polval`.

Subrutina metodei este:

POL_direct(n, a, x0, eps, itmax, rad, b, ier, n_div)

Parametri formali sunt aceeaşi ca la subrutina metodei POL, cu deosebirea că *b* este variabilă scalară, care returnează valoarea polinomului pe rădăcina calculată.

Fișierul de intrare este acelaşi cu cel de la POL.

Notă: Fişierul de intrare se selectează din fereastra de dialog standard.

Exemplu

```
Laguerre de ordinul 6
6  0  0
1     -36  450  -2400  5400  -4320  720
1e-6  40
0     1     3     5     9     15


 Radacina  9.837461
 Iteratia 14: x(14) = x(6)
 ** Proces stationar de perioada = 8


 Radacina  15.98288
 Iteratia 11: x(11) = x(6)
 ** Proces stationar de perioada = 5
```

```
 Radacina      Nr. de divergente locale
  5.775142               2
  9.837461               2
  15.98288               2


 ** Radacinile:
 Radacina                    Cod   Iteratii   p(Radacina)
 0.2228466              0           5     0.0000000E+00
  1.188932              0           4     2.4414063E-04
  2.992736              0           4    -6.1035156E-05
  5.775142              0          11    -6.1035156E-04
  9.837472              2          14    -0.1480713
  15.98287              2          11    -0.6418457

Cod   Semnificatie
  0   * Toleranta EPS atinsa
  2   ** Proces stationar
```

■

Pol_Direct-q

- Derivata se calculează prin câtul q: $p'(x_n) = q(x_n; x_n)$.

Formula de iterare este (1). Aceasta este potrivită mai degrabă, pentru versiunea cu reducerea gradului.

În rest, proiectul este similar cu cel din Pol_Direct.

Fişierul de intrare este acelaşi cu POL.

■

Pol_Complex

Metoda Newton pentru polinoame: coeficienţi reali şi rădăcini reale/complexe.

Fişiere

Main-Pol_Complex.f90: programul principal

Pol_Complext-deriv.f90: subrutina metodei

polval_c.F90: funcţie pentru calculul valorii complexe a polinomului (şi derivatei)

Openfile.f90, GetFile.f90

Period_Complex.f90: funcţie pentru calculul perioadei procesului staţionar

is_Real.f90: utilitar

width.f90: utilitar

Metoda şi formula de iterare sunt aceleaşi ca pentru POL_Direct.

Coeficienţii polinomului sunt reali, şi se pot calcula rădăcinile complexe. În plus, programul principal oferă posibilitatea inversării coeficienţilor polinomului – adică: se introduc coeficienţii a_k, şi se calculează rădăcinile polinomului cu coeficienţii $1/a_k$.

Datele de intrare şi ieşire sunt aceleaşi ca pentru fişierul de intrare din POL, cu deosebirea că, după coeficienţii polinomului se poate introduce o dată logică *opţională* pentru inversarea coeficienţilor:
Cod inversare coeficienti: Valoare: T (da), sau F (nu).

Aproximaţiile rădăcinilor trebuie să fie *complexe*.
Dezavantajul este că trebuie cunoscute aproximaţii complexe suficient de apropiate de rădăcini.

Exemplu de test:
Se consideră polinomul

$$p(x) = x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13068x - 5040,$$

care este desvoltarea lui $(x-1)\cdot(x-2)\cdot\ldots\cdot(x-7)$.

Coeficientul lui x^6 se modifică, din 28, în -28.002. Se calculează rădăcinile *polinomului perturbat*:

$$\tilde{p}(x) = x^7 - 28.002x^6 + 322x^5 + \ldots$$

Fişierul de intrare/ieşire este:

```
Polinomul: (x-1)* ... *(x-7) - perturbat
7  0  0
1     -28.002     322   -1960 6769  -13132      13068 -5040
1e-6  100
(1,0) (2,0)  (3,0) (4,0)  (5., 0.5) (5.,-0.5)  (7,0)
F


 Radacina  3.819557       0.000000
 Iteratia 7: x(7) = x(3)
 ** Proces stationar de perioada = 4

 Radacina  3.820045     -1.2611686E-44
 Iteratia 55: x(55) = x(43)
 ** Proces stationar de perioada = 12

 Radacina  3.820045      1.2611686E-44
 Iteratia 55: x(55) = x(43)
 ** Proces stationar de perioada = 12
```

```
 Radacina  7.233156       0.000000
 Iteratia 31: x(31) = x(18)
 ** Proces stationar de perioada = 13


 Radacina                        Nr. de divergente locale
(3.033329,0.0000000E+00)         12
(3.819557,0.0000000E+00)          2
(3.820045,-1.2611686E-44)        27
(3.820045,1.2611686E-44)         27
(7.233156,0.0000000E+00)         12


  ** Radacinile:
 Radacina                       Cod  Iteratii
 1 (1.000003,0.0000000E+00)     0    2
 2 (1.998937,0.0000000E+00)     0    3
 3 (3.033329,0.0000000E+00)     0    31
 4 (3.819557,0.0000000E+00)     2    7
 5 (3.820045,-1.2611686E-44)    2    55
 6 (3.820045,1.2611686E-44)     2    55
 7 (7.233156,0.0000000E+00)     2    31

p(Radacina)
 1   0.000000
 2   0.000000
 3   0.000000
 4 -1.3671875E-02
 5 (1.9531250E-02,-1.0733946E-42)
 6 (1.9531250E-02,1.0733946E-42)
 7  2.1972656E-02

 Cod   Semnificatie
   0   * Toleranta EPS atinsa
   2   ** Proces stationar
```

Notă

Polinomul perturbat are două rădăcini complexe. Cu aproximații mai bune pentru acestea, de exemplu:

```
(5.1, 0.51) (5.1,-0.51)
```

se obțin rădăcinile:

```
(5.458771,0.5410858)    (5.458771,-0.5410858)
```

■

Pol_Halley

Metoda Halley pentru polinoame.

Fișierul de intrare este același ca la Pol (sau Pol_Direct).

Alternativ, se poate folosi proiectul Halley_s.

Laguerre

Metoda Laguerre.

Pentru un polinom cu coeficienţi reali.

Metoda:

Formula de iterare este

$$z_{i+1} = z_i - \frac{np(z_i)}{p'(z_i) \pm \sqrt{H(z_i)}},$$

unde: *n* este gradul polinomului; $H(z_i) = (n-1)[(n-1)p'^2(z_i) - np(z_i)p''(z_i)]$, iar semnul din faţa radicalului se ia astfel ca numitorul să fie cel mai mare în modul (dacă z_i este real, semnul este semnul lui $p'(z_i)$. Se arată că pentru o rădăcină *simplă*, ordinul de convergenţă este 3 (pentru o rădăcină multiplă, ordinul este 1).

Fişiere:

comun.f90

GetFile.f90

is_Complex.f90

Main_Laguerre.f90

Openfile.f90

Period_Complex.f90

sub-Laguerre.f90

ULP(x).f90

width.f90

z_polval.f90

- Datele de intrare se citesc dintr-un fişier de date;
- Datele de ieşire se scriu în acest fişier după datele de intrare.

Fişierul de intrare conţine următoarele date – scrise pe linii separate:

1) *Titlul problemei*: text de max. 80 caractere
2) *n,n_rad, kod*: gradul polinomului; număr de rădăcini cerute; cod de tipărire a iteraţiilor (0/≠0);
3) $a_n, a_{n-1}, \ldots, a_0$: coeficienţii polinomului (în această ordine; reali.)
4) *tol, lnit*: toleranţa; numărul limită de iteraţii.
5) $x_1, x_2, \ldots, x_{n\_rad}$: aproximaţiile iniţiale – în număr de *n_rad*.

Opţional:

6) *lang*: limbă antet rezultate; text, primul caracter: r,R/alt char. = romana/engleza.

Fişierul de date se crează, de preferinţă, in sub-folderul ”Dat”.

Notă

Dacă există rădăcini complexe: Aproximaţii reale pot duce la rădăcinile complexe. V. exemplul pol7.dat.

Notă

Folderul **Laguerre-D** conţine versiunea în dublă precizie ■

Laguerre-IMSL

Metoda Laguerre - implementarea din IMSL.

Fişiere:

GetFile.f90

imsl_dll.lib

Main_Lag-3.f90

Openfile.f90

polval_c.F90

Subrutina metodei este apelată din Biblioteca IMSL, anume:

ZPLRC (n, coef, rad)

Parametrii de apel sunt:

n: gradul polinomului

coef: tablou (real); coeficienţii polinomului *a*(0:*n*)

rad: tablou (complex); rădăcinile calculate *rad*(*n*).

Metoda Laguerre – pentru un polinom cu coeficienţi reali – este implementată în IMSL prin subrutina ZPLRC, fiind combinată cu reducerea gradului, astfel că se determină toate zero-urile polinomului. Apelul lui ZPLRC nu cere aproximaţii iniţiale – v. IMSL Math/Libraries (1999).

(Subrutina ZPLRC returnează numai rădăcinile.)

■

Datele de intrare se citesc dintr-un fişier de date. Acesta conţine:

Titlul problemei: text de max. 80 caractere

1) *n*: gradul polinomului
2) $a_n, a_{n-1}, \ldots, a_0$: coeficienţii polinomului (în această ordine).

Urmatoarea dată este opțională:

3) *Inversare coeficienți* (logic): T/F

Datele de ieşire se scriu în continuarea datelor de intrare, şi sunt: Rădăcinile calculate și valorile polinomului pe rădăcini.

Exemplu: Fişierul de intrare/ieşire este:

```
Titlu: (x-1)* ... *(x-7) - perturbat:
7
1     -28.002     322   -1960 6769  -13132      13068 -5040


 Radacini:
           1 (7.233085,0.0000000E+00)
           2 (5.458664,0.5402883)
           3 (5.458664,-0.5402883)
           4 (3.819508,0.0000000E+00)
           5 (3.033140,0.0000000E+00)
           6 (1.998938,0.0000000E+00)
           7 (1.000003,0.0000000E+00)

 Valori polinom:
           1 (-0.3129883,0.0000000E+00)
           2 (-9.0820312E-02,-4.4250488E-02)
           3 (-9.0820312E-02,4.4250488E-02)
           4 (-2.0996094E-02,0.0000000E+00)
           5 (6.3476562E-03,0.0000000E+00)
           6 (-4.8828125E-04,0.0000000E+00)
           7 (0.0000000E+00,0.0000000E+00)
```

■

Muller

Metoda Muller.

Funcţie *f*, continuă şi cu derivate continue până la ordinul 3, pe o vecinătate a rădăcinii.

Metoda:

Metoda cere trei aproximaţii iniţiale x_0, x_1, x_2 ale rădăcinii. (Nu este necesar ca acestea să fie ordonate după mărime.)

Se construieşte polinomul de interpolare (de gradul 2) care trece prin punctele (x_i, f_i), unde $f_i = f(x_i)$; aproximaţia următore x_3, este una din rădăcinile lui $p(x)$ – v. mai jos.

Aproximaţii reale pot conduce la zero-uri complexe. În particular, aproximaţiile reale se pot lua forma: $x_1, x_1 \pm h$.

Polinomul de interpolare Newton pe nodurile x_2, x_1, x_0 (v. Folder: **Interpolation** INTERPOLARE), este:

$$p(x) = f_2 + f_{21}(x - x_2) + f_{210}(x - x_2)(x - x_1),$$

Coeficienţii sunt *diferenţele divizate* ale funcţiei *f* pe nodurile x_i, x_j, x_k, anume:

$$f_i = f(x_i); \qquad f_{ij} = f[x_i, x_j] = \frac{f_j - f_i}{x_j - x_i}; \qquad f_{ijk} = f[x_i, x_j, x_k] = \frac{f_{jk} - f_{ij}}{x_k - x_i},$$

Fie polinomul de interpolare desvoltat în $(x - x_2)$:

$$p(x) = a(x - x_2)^2 + b(x - x_2) + c$$

Coeficienţii *a*, *b*, *c*, se găsesc prin:

$$a = f_{210}; \qquad b = f_{21} + f_{02} - f_{10}; \qquad c = f_2$$

Formula de iterare este

$$x_3 = x_2 - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}$$

Ca aproximaţie următoare, se ia valoarea x_3 care este *cea mai apropiată de* x_2, adică rădăcina pentru care $|x_3 - x_2|$ este cel mai mic.

Pentru aceasta, semnul din faţa radicalului se ia astfel ca numitorul să fie cel mai mare, şi anume:

- Pentru rădăcini reale, se ia semnul lui *b* (dacă *b* = 0, orice semn);
- Pentru rădăcini complexe (inclusiv cazul în *b* este complex), semnul pentru care numitorul este cel mai mare în modul. Explicit: punem

  $b_{1,2} = b \pm \sqrt{b^2 - 4ac}$. Dacă $|b_1| > |b_2|$, atunci formula de iterare este:

  $x_3 = x_2 - 2c/b_1$; altfel, $x_3 = x_2 - 2c/b_2$.

După determinarea lui x_3:

- Dintre punctele x_i, $i = \overline{0,2}$, se exclude punctul cel mai îndepărtat de x_3 (cel pentru care $|x_3 - x_i|$ este cel mai mare).
- Se atribuie x_3 lui x_2. Celelalte două puncte devin x_0 şi x_1.
- Se procedează la o nouă iteraţie (cu noua secvenţa x_0, x_1, x_2).

Testul de oprire a iteraţiei este

$|x_3 - x_2| \le EPS$,

unde *EPS* este toleranţa; remarcaţi că $x_2 = x_3\_anterior$.

Convergenţa: metoda Muller are ordinul de convergenţă p = 1.84 – pentru o rădăcină simplă; v. Atkinson (1978). Pentru o rădăcină complexă multiplă, convergenţa poate fi mai înceată.

Fişiere:
Muller2005.f90: subrutina metodei
select.f90: subrutina de selecţie a valorilor x_0, x_1, x_2 (pentru iteraţia următoare).
Main_Muller2005.f90: programul principal
f.f90: funcţia $f(x)$
Function_expression.f90; width.f90: utilitare

Subrutina metodei:
muller(f, x0, x1, x2, eps, iter, rad, kod)

Parametrii formali:
f: Numele funcţiei *f*
*x*0, *x*1, *x*2: Aproximaţiile iniţiale
eps: Toleranţa
iter: Numărul limită de iteraţii (intrare); numărul efectiv de iteraţii (ieşire)
rad: Rădăcina calculată
kod: Intrare: cod de tipărire a iteraţiilor: 0 (nu); ≠ 0 (da)
Ieşire: cod de încheiere a iteraţiei (intern):
0: Toleranţa *eps* atinsă

1: *lnit* depăşit

2: Rădăcina în aproximaţiile iniţiale

-1: Radical din număr negativ

Subrutina muller apelează subrutina select (x0, x1, x2, x3).

Datele de intrare se introduc de la terminal, si constau din:

- *nume*: numele fişierului de ieşire (acesta primeşte extensia .rez)
- *Exist_p*: dată logică. Valori: T (true) sau F (false)
- [*p*: dacă *Exist_p* = T]
- *eps, lnit*: toleranţa, numărul limită de iteraţii
- *x*1, *h*: aproximaţia *x*1, pasul
- *kod*: cod de tipărire a iteraţiilor

Programul principal pune (înainte apelul subrutinei muller):

$x0 = x1 - h; \quad x2 = x1 + h$

Datele de ieşire se scriu în fişierul de ieşire nume.rez, şi sunt următoarele:

- expresia funcţiei *f* (scrisă de utilitarul Function_expression)
- [*p*, dacă există]
- aproximaţiile *x*0, *x*1, *x*2
- *lnit*, *eps*
- modul de încheiere a iteraţiei (mesaj, conform codului *kod*)
- rădăcina calculată *rad*; valoarea *f*(*rad*)
- numărul efectiv de iteraţii.

Exemplu de test:

$f(x) = e^x - px^2$,

cu $p = 3$; se caută rădăcina din vecinătatea valorii 0.8.

Se iau: $x0 = 0.8, \quad h = 0.2$ (Rezultă aproximaţiile iniţiale: 0.6; 0.8; 1.0)

eps = 1E-6, *lnit* 10; *ktip* = 0.

Fişier de ieşire:

```
Metoda MULLER
```

```
 f =exp(x) -p*x*x
 p:   3.000000

Radacina nr.1
 Aproximatii:  0.6000000      0.8000000       1.000000
 EPS: 1.000E-06; LNIT: 10

Iter    x0             x1             x2             * x3
 1      0.6000000      0.8000000      1.000000       0.9104018
 2      0.8000000      1.000000       0.9104018      0.9100071
 3      1.000000       0.9104018      0.9100071      0.9100076

 * Toleranta EPS atinsa
  Radacina =  0.9100076     ; f(Radacina) =  2.1429681E-08
  Numar de iteratii = 3
```

■

Graficul funcţiei şi polinomului de interpolare se dau mai jos.

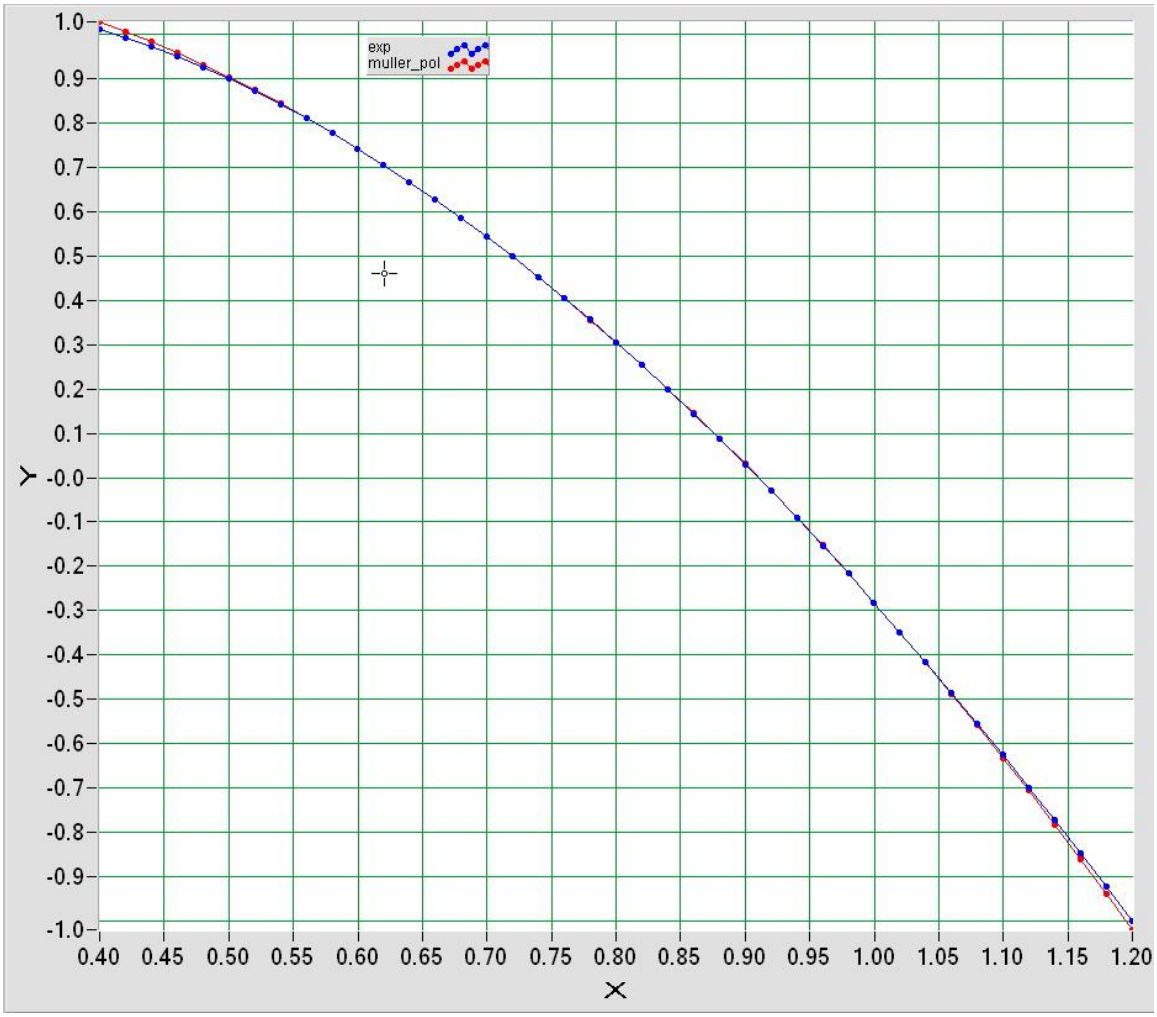


Figura 6 – Metoda Muller: graficul funcţiei şi polinomului de interpolare

Muller\Muller_Complex

Metoda Muller pentru rădăcinile unei funcţii complexe.

Fişiere:

Muller_Complex2005.f90; Main_Muller_Complex2005.f90; f_complex.f90; select_complex.f90; Function_expression.f90;; width.f90;

Subrutina metodei:

Muller_complex(f, h, z0, z1, z2, eps, iter, rad, kod)

Semnificaţiile parametrilor sunt aceleaşi ca la Muller, cu deosebirile:

- datele numerice şi funcţia *f* sunt declarate complexe

- programul este scris în dublă precizie

Datele de intrare se citesc dintr-un fişier de date (specificat de utilizator). Acesta are structura:

- Titlul problemei: text de max. 80 caractere
- *Exist_p*: dată logică. Valori: T (true) sau F (false)
- [*p*: dacă *Exist_p* = T]
- *nrad*: numărul rădăcinilor de calculat
- *z*1(1:*nrad*): *nrad* aproximaţii – reale sau complexe
- *h*: pasul. Valoare reală.
- *eps*, *lnit*: toleranţa, numărul limită de iteraţii
- *ktip*: cod de tipărire a iteraţiilor: 0 (da); ≠0 (nu)

Datele de ieşire se scriu după datele de intrare, şi conţin: rădăcina, valoarea funcţiei pe rădăcină, şi numărul de iteraţii (pentru *nrad* rădăcini).

Exemplu de test:

Pentru polinomul din exemplul de la Pol_Complex, se iterează cu $eps = 1E-10$; *lnit* = 20; aproximaţia reală $x_1 = 6$; $h = 0.1$. Fişierul de intrare/ieşire este următorul:

```
Wilkinson_7
F
1
6
0.1
1E-10 20
0

 Metoda MULLER (Complex)
 f =(z-1)*(z-2)*(z-3)*(z-4)*(z-5)*(z-6)*(z-7) -0.002*z**6

 EPS: 1.000E-10; LNIT: 20

#             Radacina                                  Iteratii

1    5.458675825432509       0.5401258000668572           9
```

```
        f = (-1.421085471520200E-014,3.197442310920451E-014)
2       5.458675825432509       0.5401258000668571          7
        f = (-2.842170943040401E-014,1.776356839400250E-014)
```

■

Rădăcina este găsită în 7 iteraţii. Cu aproximaţia $x_1 = 5$ se găseşte aceeaşi rădăcină în 9 iteraţii.

II-5 Metode de ordin 3 și mai înalt, pentru ecuații de forma $f(x) = 0$

Halley-s

Metodele Halley și ”super-Halley”. Metodele utilizează derivatele f' și f'', calculate într-un singur punct.

1) Metoda Halley:

Formula metodei:

$$x_{n+1} = x_n - h(x_n) \qquad (1)$$

Funcția de iterare este:

$$h(x) = \frac{f'(x)f(x)}{[f'(x)]^2 - \frac{1}{2}f''(x)f(x)}$$

Pentru algoritm se definesc:

$$u(x) = \frac{f(x)}{f'(x)}; \qquad w(x) = \frac{f''(x)}{f'(x)}; \qquad L(x) = u(x)w(x); \qquad (2)$$

Cu acestea, $h(x)$ se scrie:

$$h(x) = u(x)\frac{1}{1 - \frac{1}{2}L(x)} \qquad (1')$$

Metoda are ordinul 3 într-o rădăcină simplă.

2) Metoda Super-Halley:

$$x_{n+1} = x_n - s(x_n) \qquad (2)$$

$$s(x) = u(x)\left[1 + \frac{1}{2}\frac{L(x)}{1 - L(x)}\right] \qquad (2')$$

Metoda are ordinul 3 într-o rădăcină simplă. Totuși, pentru un polinom de gradul doi, metoda are ordinul 4.

Într-o rădăcină multiplă, metodele (1) și (2) au ordinul de convergență unu.

■

Fișiere:

Deriv_Pol.f90
Function_expression-4.f90
GetFile.f90
Halley-s.f90
Main_Halley-s.f90
Openfile.f90
Period.f90
polval.F90
sub.f90
width.f90
work.f90

■

Pentru datele de intrare v. mai jos, Date de intrare.

Jarratt

Metode Jarratt, de ordine 3,4, și 5. Formula de iterare utilizează derivata f', calculată în mai multe puncte; din acest motiv, metodele se zic metode multi-punct.

Formula metodelor:

$$x_{n+1} = x_n - j(x_n)$$

Funcția $j(x)$ se definește prin algoritmul de mai jos.

1) Ordinul 3

Se definesc:

$$\omega_1(x) = f'(x); \qquad \omega_2(x) = f'(x + \alpha u(x));$$

$$j(x) = \frac{f(x)}{a_1\omega_1(x) + a_2\omega_2(x)}$$

Parametrii α, a_1, a_2 au valorile:

a) Metoda #1 de ordin 3:

$$\alpha = -\frac{1}{2}; \quad a_1 = 0; \quad a_2 = 1; \qquad j(x) = \frac{f(x)}{f'(x - \frac{1}{2}u(x))}.$$

b) Metoda #2 de ordin 3:

$$\alpha = -\frac{2}{3}; \quad a_1 = \frac{1}{4}; \quad a_2 = \frac{3}{4}; \qquad j(x) = \frac{1}{4}\frac{f(x)}{f'(x) + 3f'(x - \frac{2}{3}u(x))}.$$

2) Ordinele 4 și 5

ω_1, ω_2 se definesc ca mai sus; în plus,

$$\omega_3(x) = f'[x + \beta u(x) + \gamma \frac{f(x)}{\omega_2(x)}]$$

$$j(x) = \frac{f(x)}{a_1\omega_1(x) + a_2\omega_2(x) + a_3\omega_3(x)}$$

Parametrii au valorile:

1) Ordinul 4:

$$\alpha = -\frac{1}{3}; \quad a_1 = \frac{1}{10}; \quad a_2 = \frac{1}{2}; \quad a_3 = \frac{2}{5}; \quad \beta = \frac{25}{24}; \quad \gamma = -\frac{15}{8};$$

2) Ordinul 5:

$$\alpha = -1; \quad a_1 = \frac{1}{6}; \quad a_2 = \frac{1}{6}; \quad a_3 = \frac{2}{3}; \quad \beta = -\frac{1}{8}; \quad \gamma = -\frac{3}{8};$$

Într-o rădăcină multiplă, metodele Jarratt (1) și (2) au ordinul de convergență unu.

■

Fișiere:

Deriv_Pol.f90

Function_expression-01.f90

gen_coef.f90

gen_Num.f90

GetFile.f90

Jarratt.f90

Main_Jarratt.f90

Openfile.f90

Period.f90

polval.F90

sub.f90
width.f90
work.f90

■

Date de intrare (Halley_s și Jarratt)

Ambele metode sunt implementate pentru o ecuație $f(x) = 0$; în particular, $f(x)$ poate fi un polinom.

- Dacă *f* este polinom: Programul cere un fișier de intrare. Structura lui este următoarea:
  - *Linie_text*: titlul problemei; max. 80 caractere;
  - *n*: gradul polinomului
  - $a_n, a_{n-1}, \ldots, a_0$: coeficienţii polinomului (în această ordine);
- Dacă *f* nu este polinom: *f* și derivatele sale (f' și f'' - pentru Halley_s; numai f' - pentru Jarratt) se definesc în subrutina `SUB`, și anume:

  Halley_s: $f = f(x)$; $f1 = f'(x)$; $f2 = f''(x)$;

  Jarratt: *f* și f' se definesc ca funcții_instrucțiune: $f0(x) = f(x)$; $f1(x) = f'(x)$.

  (În relațiile precedente: $f(x)$, $f'(x)$, și $f''(x)$ sunt expresiile funcțiilor respective.)

Toate datele necesare iterației se introduc de la tastatură.

■

Ordin 15

Ca exemplu de metodă de ordin foarte înalt, dăm o metodă care are ordinul 15 într-o *rădăcină simplă* (Soleymani & Sharifi, 2011).

Metoda are 4 pași; cere cinci evaluări de funcții/pas (patru evaluări *f* și una f'); și, are o aplicare simplă. Formulele metodei sunt:

$$y_n = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$z_n = y_n - \frac{f(x_n)}{f(x_n) - 2f(y_n)} \cdot \frac{f(y_n)}{f'(x_n)}$$

$$w_n = z_n - \frac{f(z_n) f[x_n, y_n]}{f[x_n, z_n] f[y_n, z_n]} \left(1 + \frac{f(z_n)}{f(x_n)}\right)$$

$$x_{n+1} = w_n - \frac{f(w_n)}{f[x_n, w_n] + (f[y_n, x_n, z_n] - f[y_n, x_n, w_n] - f[z_n, x_n, w_n])(x_n - w_n)}$$

În formulele anterioare, $f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$ și

$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$, sunt diferențele divizate ale funcției *f* pe nodurile respective.

Condiția de aplicare a formulelor precedente este ca, numitorii să nu fie nuli.

Pentru convergență, aproximația inițială x_0 trebuie să fi suficient de apropiată de rădăcină.

Notă

Pentru o comparație corectă a numărului de iterații, cu cel din metoda Newton (mai general, cu cel al unei metode de ordinul doi), numărul de iterații din metoda multi-pas se va înmulți cu 4 – deoarece cei patru pași/iterație sunt echivalenți cu 4 pași din metoda Newton.

■

Fișiere:

check_num.f90
Deriv_Pol.f90
div_dif_12.f90
f&f1.f90
fun.f90
Function_expression-01.f90
GetFile.f90
IsInfinity.f90
Main_15.f90
Openfile.f90
Ordin_15-2.f90

Pas-Ordin_15.f90
Period.f90
polval.F90
width.f90
work.f90

Date de intrare

- Dacă *f* este polinom: Programul cere un fișier de intrare. Structura lui este aceeași cu cea de la metoda Halley.
- Dacă *f* nu este polinom: *f* și derivata f' se definesc în fișierul `f&f1.f90`.

Restul datelor pentru iterație, se introduc de la tastatură.

■

CAPITOLUL III – SISTEME DE ECUAŢII LINIARE

Sistemul de ecuaţii liniare este

$\mathbf{Ax} = \mathbf{b}$,

unde **A** est matrice $n \times n$, iar **x** şi **b** vectori coloană cu n coordonate.

Rezolvarea se poate considera pentru mai mulţi termeni liberi **b**.

Folder tematic: **$ Linear Systems**

III-1 Metode directe

Gauss

Eliminarea Gauss – cu pivotare parţială.

Fişiere:

Elim.f90: subrutina metodei

Main-Elim.f90: programul principal

width.f90: utilitar

Fişier de date: Ex2.dat

Metoda:

Sistemul dat se transformă în sistemul echivalent $\mathbf{Ux} = \mathbf{g}$, cu matricea **U** superior triunghiulară, cum urmează.

Sistemul dat se notează

$\mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$.

La pasul curent k ($k = 1, 2, \ldots, n-1$):

Sistemul este

$\mathbf{A}^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$.

Se lucrează cu liniile $i = k, \ldots, n$, din $\mathbf{A}^{(k)}$ şi $\mathbf{b}^{(k)}$. (Liniile $1, \ldots, k-1$, procesate anterior, rămân neschimbate). Concret, se operează cu sub-matricea $\mathbf{A}^{(k)}(k:n, k:n)$ şi sub-coloana $\mathbf{b}^{(k)}(k:n)$ a termenilor liberi.

$$\mathbf{A}^{(k)} = \left[\begin{array}{ccccccccc} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1k}^{(1)} & . & & . & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2k}^{(2)} & . & & . & a_{2n}^{(2)} \\ . & 0 & \ddots & & . & & . & \\ . & . & \dots & a_{k-1,k-1}^{(k-1)} & . & & . & a_{k-1,n}^{(k-1)} \\ \hdashline 0 & . & \dots & 0 & a_{kk}^{(k)} & . & a_{kj}^{(k)} & . & a_{kn}^{(k)} \\ . & . & \dots & . & . & . & . & . & . \\ . & . & \dots & . & a_{ik}^{(k)} & . & a_{ij}^{(k)} & . & a_{in}^{(k)} \\ . & . & \dots & . & . & . & . & . & . \\ 0 & 0 & \dots & 0 & a_{nk}^{(k)} & . & . & . & a_{nn}^{(k)} \end{array}\right] \begin{array}{l} \times(-m_{ik}); \\ \downarrow \\ \oplus \hookleftarrow \end{array}$$

$$\mathbf{b}^{(k)} = \left[\begin{array}{c} b_1^{(1)} \\ b_2^{(2)} \\ \\ \hdashline b_{k-1,k-1}^{(k-1)} \\ \hdashline b_k^{(k)} \\ . \\ b_i^{(k)} \\ . \\ b_n^{(k)} \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \times(-m_{ik}) \\ \downarrow \\ \oplus \hookleftarrow \\ \\ \\ \end{array}$$

Presupunem $a_{kk}^{(k)} \neq 0$ – v. mai jos pivotare, şi definim multiplicatorii de la pasul *k*:

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}; \qquad i = k+1, \dots, n$$

Pentru $i = k+1, \dots, n$: linia *k* se înmulţeşte cu $-m_{ik}$, şi se adună la linia *i*. Rezultă elemente nule în coloana *k*, sub elementul diagonal $a_{kk}^{(k)}$. Linia *k* rămâne neschimbată. Noii coeficienţi şi termeni liberi vor fi:

$$\left.\begin{array}{l} a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} \cdot a_{kj}^{(k)}; \quad j = k+1, \dots, n \\ b_i^{(k+1)} = b_i^{(k)} - m_{ik} \cdot b_k^{(k)} \end{array}\right\} \quad i = k+1, \dots, n$$

La pasul *n* se notează, pentru convenienţă, $\mathbf{U} = \mathbf{A}^{(n)}$, $\mathbf{g} = \mathbf{b}^{(n)}$, şi sistemul devine

$\mathbf{Ux} = \mathbf{g}$.

Explicit:

$$\begin{bmatrix} u_{11} & . & . & . & . & u_{1n} \\ 0 & u_{22} & . & . & . & u_{2n} \\ & & . & . & . & . \\ 0 & & 0 & u_{kk} & . & u_{kn} \\ & & & & . & . \\ 0 & & & & 0 & u_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_k \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ . \\ g_k \\ . \\ g_n \end{bmatrix}$$

Acest sistem se rezolvă prin substituţie înapoi:

$$x_n = g_n / u_{nn}; \quad x_k = (g_k - \sum_{j=k+1}^{n} u_{kj} \cdot x_j) / u_{kk}; \quad k = n-1, n-2, \ldots, 1$$

■

Pivotarea parţială:

La fiecare pas *k*, se caută elementul de modul maxim din sub-coloana $a(k:n,k)$, numit pivot; fie acesta găsit în linia $I \geq k$.

Dacă pivotul este mai mare decât un prag şi $I > k$, atunci se schimbă liniile *k* şi *I* (în $\mathbf{A}^{(k)}$ şi în $\mathbf{b}^{(k)}$).

Dacă pivotul este mai mic decât pragul, eliminarea Gauss se opreşte, întrucât:

3) dacă pivotul este nul, matricea este singulară.

4) dacă pivotul este nenul dar mai mic decât pragul, matricea este "aproape" singulară.

■

Număr de operaţii: Pentru *n* = "mare", numărul de operaţii în eliminarea Gauss este:

$$NOP_{Guass} \approx \frac{n^3}{3}$$ ■

Subrutina metodei:

ELIM (n, nb, A, B, prag, kod, kodPrint)

Parametri formali:

n: numărul ecuaţiilor

nb: numărul de termeni liberi

A(*n*, *n*): matricea coeficienţilor

B(*n*, *nb*): matricea termenilor liberi

prag: prag pentru pivot ≈ 0.

kod : cod pentru eliminarea Gauss:

kod = 0: o.k.; *kod* ≠0: pivot ≈ 0 (pivot < prag)

kodPrint: cod de tipărire – v. mai jos.

ELIM face pivotare parţială. Dacă pivotul este mai mic decât pragul, se produce un mesaj de eroare. Valoarea *prag* este setată de programul principal la 1E-6 (aceasta se poate modifica).

Matricea **U** se stochează în triunghiul superior din *A*.

Soluţia este returnată în coloanele din *B*.

Datele de intrare se citesc dintr-un fişier de date, specificat de utilizator. Acesta are structura:

- *Titlu*: text de max. 80 caractere
- *n*, *nb*: numărul de ecuaţii; numărul de termeni liberi.
- *Matricea coeficienţilor* **A**: $n \times n$. Se scrie aşa cum e dată - pe linii şi coloane.
- *Termenii liberi* **b**: Se scriu pe linii: *nb* linii a *n* termeni liberi.
- *cod_ Inversare_Coeficienţi*, *cod_ Tipărire*:

  cod_Inversare_Coeficienţi: dacă este ≠ 0, atunci se rezolvă sistemul cu

  $$a(i, j) = 1./a(i, j).$$

  cod_ Tipărire: dacă este ≠ 0: Se tipăresc permutările şi pivoţii.
- *check_text*: text de max. 20 caractere. Primul caracter este semnificativ, anume:

  `C, c, P, p`: se face proba $\mathbf{Ax} = \mathbf{b}$ (concret, se tipăreşte vectorul **Ax** – **b**).

Observaţie

Dacă se cere proba, atunci programul principal salvează *A* şi *B* înainte de apelul lui ELIM ■

Datele de ieşire se scriu în continuarea datelor de intrare – v. mai jos

Exemplul-1: Matrice 3×3, 2 termeni liberi.

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 2 & 1 & 4\\ 3 & 4 & 1\end{bmatrix};\qquad \mathbf{b}=\left[\begin{array}{c|c}6 & 1\\ 7 & 1\\ 8 & 1\end{array}\right].$$

Fişierul de intrare/ieşire:

```
Test: solutia_1 =1, 1, 1
3 2
1 2 3
2 1 4
3 4 1

6     7     8
1     1     1
0     0
proba

 * Solutia pentru termenii liberi nr. 1
 x(1) =    1.000000
 x(2) =    1.000000
 x(3) =    1.000000

 * Solutia pentru termenii liberi nr. 2
 x(1) =   0.1490116E-07
 x(2) =   0.2000000
 x(3) =   0.2000000

* Proba Ax =b:

Solutia nr. 1: A*x -b
           1  0.0000000E+00
           2  0.0000000E+00
           3  0.0000000E+00

Solutia nr. 2: A*x -b
           1  0.0000000E+00
           2  0.0000000E+00
           3  0.0000000E+00
```

■

Exemplul-2: Matrice singulară

Fişierul de intrare/ieşire (Ex2.dat):

```
Matrice 4x4: singulara.
4 1
5     6     3     1
-1    0     -1    1
2     2     1     6
4     2     3     4

4*1
0     1           ! kodInv, kodPrint
Check

Permutare, Pivot:
1       5.000000
2<->4   -2.800000
3     -0.2857143
4      7.7486033E-07

 Linia 4: Pivot =  7.7486033E-07

* Pivot <1.00E-06 !
* Sistemul nu se poate rezolva.
```

■

Notă

Acest exemplu arată de ce s-a ales pragul egal cu 1E-6 ■

Gauss_AB

Eliminarea Gauss – lucrul cu matricea extinsă.

Fişiere:

Elim_AB.f90; Main-Elim_AB.f90; width.f90; GetFile.f90; Openfile.f90;

Subrutina metodei:

ELIM (AB, n, ncol, prag, kod, pivot)

Parametri formali:

AB(*n, ncol*): matricea coeficienţilor extinsă cu termenii liberi.

n: numărul ecuaţiilor.

ncol: *ncol* = *n* + *nb*, unde *nb* este numărul de termeni liberi (*ncol* este definit în programul principal).

prag: prag pentru pivot ≈ 0.

kod : cod pentru eliminarea Gauss:

kod = 0: o.k.; *kod* ≠0: pivot ≈ 0 (pivot < prag).

pivot: valoare pivot, când pivot < prag.

Metoda este aceeaşi ca la Gauss, cu deosebirea că acum, subrutina metodei lucrează cu matricea extinsă *AB*(*n, ncol*). Soluţiile sunt returnate in coloanele $n+1:nb$ ale matricii extinse.

Datele de intrare se citesc dintr-un fişier de date. Acesta are aceeaşi structură ca la Gauss, cu deosebirea că nu mai conţine codul de tipărire:

- *Titlu*: text de max. 80 caractere
- *n, nb*: nr. de ecuaţii; nr. de termeni liberi.
- *Matricea coeficienţilor* **A**: $n \times n$. Se scrie aşa cum e dată - pe linii şi coloane.
- *Termenii liberi* **b**: Se scriu pe linii: *nb* linii a *n* termeni liberi.
- *cod_ Inversare_Coeficienţi*:

  cod_Inversare_Coeficienţi: dacă este ≠ 0, atunci se rezolvă sistemul cu

  $$a(i, j) = 1./a(i, j).$$

- *check_text*: text de max. 20 caractere. Primul caracter este semnificativ, anume:

  `C, c, P, p`: se face proba $\mathbf{Ax} = \mathbf{b}$.

Observaţie

Dacă se cere proba, atunci programul principal salvează matricea *AB* înainte de apelul lui ELIM ■

Datele de ieşire se scriu în continuarea datelor de intrare şi conţin soluţia şi proba (dacă este comandată prin *check_text*).

LU

Rezolvarea prin descompunerea LU (calculul direct al factorilor L şi U) – metoda Doolittle. Cu pivotare parţială.

Fişiere:

Primele 5 fişiere constituie subrutinele metodei.

LUdecomp.f90: descompunerea LU

LUsolve.f90: substituţie înapoi

apvt-LU.f90: pivotare parţială

compute_a_ii.f90: calculul pivotului (înainte de pivotare)

swap_row.f90: schimbare de linii

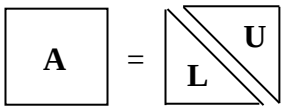
Main-LU.f90: programul principal

width.f90: utilitar

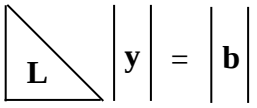
Metoda:

Metoda constă în paşii:

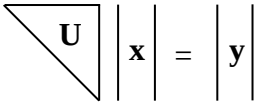
1. Factorizare $\mathbf{A} = \mathbf{LU}$**,** cu pivotare parţială.



2. Rezolvarea sistemului $\mathbf{Ly} = \mathbf{b}$, prin *substituţie înainte*; rezultă **y**.



3. Rezolvarea sistemului $\mathbf{Ux} = \mathbf{y}$, prin *substituţie înapoi*; rezultă **x**.



Formulele pentru calculul elementelor lui **L** şi **U** – cu l_{ii} ales arbitrar (nenul):

l_{ii} = ales arbitrar (nenul), $i = 1, 2, .. , n$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} \cdot u_{kj}}{l_{ii}}, \qquad j = i,\ i+1,\ \ldots,\ n \qquad \text{(a)}$$

Dacă $u_{ii} \neq 0,\ i = \overline{1, n-1}$, rezultă:

$$l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} \cdot u_{ki}}{u_{ii}}, \qquad j = i+1,\ \ldots,\ n \qquad \text{(b)}$$

Elementele lui **L** si **U** se determină în următoarea secvenţă, conform schemei de mai jos (se calculează elementele din linia *i* din **U,** şi coloana *i* din **L**).

l_{ii}	$\rightarrow$	$u_{i,i}$	$u_{i,i+1}$	$\ldots$	$u_{i,n}$
$\downarrow$					
$l_{i+1,i}$					
$\vdots$					
$l_{n,i}$					

În particular, la $i = n$, se calculează numai u_{nn} - din (a).

Metoda Doolittle pune $l_{ii} = 1$.

Numărul de operaţii este acelaşi ca în eliminarea Gauss ($\approx n^3/3$).

Pivotare:

Pivotarea trebuie făcută înainte de calculul elementelor lui **U** şi **L**. Adică, pivotul u_{ii} trebuie calculat şi testat *înainte* de aplicarea formulelor (a) şi (b).

Pentru metoda Doolittle, pivotul este dat de (cf. a):

$$u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik} \cdot u_{ki}$$

Aşa cum se arătă în Exemplu-2, trebuie pivotat şi dacă pivotul este foarte mic. Practic, se testează dacă pivotul este mai mic decât un *prag*.

Două strategii de pivotare parţială sunt disponibile, comandate printr-un cod: căutarea *pivotului maxim* nenul, sau căutarea *primului pivot* nenul. Pragul este ales ca $1E-6$. Prima strategie oferă, în general, o precizie mai bună, în satisfacerea verificării soluţiei, însă consumă mai mult timp de calcul.

■

Observaţie

- Dacă nu se pivotează, atunci avem **LU** = **A**.
- Dacă se pivotează, atunci avem $\mathbf{LU} = \mathbf{A}'$, unde $\mathbf{A}' = \mathbf{PA}$, iar **P** este o matrice de permutare de linii. Matricea $\mathbf{A}'$ se obţine din **A**, permutând liniile în aceeaşi ordine în care se permută la pivotare. Sistemul care se rezolvă este $\mathbf{A}'\mathbf{x} = \mathbf{b}'$, unde $\mathbf{b}'$ este $\mathbf{b}$ cu liniile permutate în ordinea de la pivotare

■

Calculul determinantului:

Determinantul matricii $\mathbf{A}' = \mathbf{LU}$ este

$$\det(\mathbf{A}') = u_{11}u_{22}\ldots u_{nn}$$

Determinantul matricii **A** va fi

$$\det(\mathbf{A}) = (-1)^{n\_l}\det(\mathbf{A}'),$$

unde *n_l* este numărul de schimbări de linii în cursul pivotării.

■

Subrutina de descompunere LU (Pasul 1) este:
LUdecomp(A, n, order, do_pvt, kodPivot, kod)
LUdecomp face:

- Stocare compactă: **L** şi **U** în **A**, în triunghiul inferior, respectiv superior. (Elementele diagonale din **L** fiind 1 nu se mai stochează.)
- Pivotare parţială - prin apel apvt-LU; aceasta apelează compute_a_ii.f90 şi swap_row.f90.
  Ordinea liniilor (după pivotare) e stocată în vectorul `order; order` se utilizează pentru ordonarea membrilor doi - la rezolvare (în LUsolve).

Parametri formali:

A: matricea sistemului

n: numărul de ecuaţii

order: vector întreg

do_pvt: dată logică; valori: `T/F`: pivotează sau nu.

kodPivot: cod pentru strategia de pivotare:

kodPivot ≠ 0: pivot maxim; *kodPivot* = 0: primul pivot nenul.

kod: cod de test pivot:

kod = 0: o.k.; *kod* <0: pivot <prag

Subrutina de rezolvare este:

LUsolve(LU, b, n, x, order)

Paşii 2 şi 3 de mai sus.

Elementele din `b` sunt re-aranjate conform schimbărilor de linii.

Soluţia este returnată in `x`.

Parametri:

LU: echivalentul `LU` a lui `A` (stocare compactă)

b: vectorul termenilor liberi

x: soluţia

order: vector întreg

Observaţie

În LU şi LUsolve, produsele scalare se calculează în dublă precizie ■

Subrutinele se apelează astfel (în programul principal):

```
call LUdecomp(A, n, order, do_pvt, kodPivot, kodLU)

! Solutia pentru "nb" termeni liberi:
do kb =1, nb
call LUsolve(A, b(1:n, kb), n, x1, order)
b(1:n, kb) =x1
enddo
```

Datele de intrare se citesc dintr-un fişier de date. Acesta conţine următoarele date (scrise pe linii):

- *Titlu*: text; max. 80 caractere
- *n, nb*: număr de ecuaţii; număr de termeni liberi.

- *Matricea coeficienţilor* **A**: $n \times n$ (Se scrie pe linii si coloane).
- *Termenii liberi* **b**: Se scriu pe linii: *nb* linii a *n* termeni liberi.
- *Cod_Tipărire, Cod_inversare_ coeficienţi*: 0 / ≠0; 0 / ≠0.
    - *Cod_Tipărire* = 0: Soluţia, în fişierul de date.

    Pentru *Cod_Tipărire* ≠ 0, se tipăresc în plus datele de mai jos.
    - *Cod_Tipărire* = 1: Soluţia, la display; Matricile **L** si **U,** în fişierul de date.
    - *Cod_Tipărire* ≥ 2: Vectorul `order`, la fiecare pivotare; Determinantul lui **A;** Dacă | *u*(*i*, *i*) | < *prag*: Se tipăreşte şi linia *i* a matricii **A** (la pasul respectiv).
    - *Cod_inversare_ coeficienţi* ≠ 0: $a(i, j) = 1./a(i, j)$
- *Cod pivotare, kod_pivot*:
    - *Cod_pivotare*: logic, valori `T/F`: Da / Nu.
    - *kodPivot*: ≠ 0: pivot maxim; = 0: primul pivot nenul.
- *Check_text*: max. 20 caractere; primul caracter este semnificativ:
  `C, c, P, p` / alt caracter: Verifică / Nu verifică - soluţia.
  `F, f`: Se verifică soluţia şi LU = PA (PA este A permutat).

Observaţie

Dacă se cere verificarea, atunci programul principal salvează `A` şi `b` înainte de apelul lui LUdecomp ■

Datele de ieşire se scriu în fişierul de date, după datele de intrare.

Exemplele 1 şi 2 de mai jos, se referă la sistemul:

$$\mathbf{A} = \begin{bmatrix} -4 & 1 & 1 & 1 \\ -6 & 6 & 4 & 4 \\ -1.5 & -1.5 & 1 & 1 \\ -6 & 6 & 4 & -4 \end{bmatrix}; \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Exemplu – 1 (Fişier de date u3.dat):

```
Matrice 4x4 cu u22 =0; u33=0. Fara pivotare.
```

```
4 1
-4          1           1     1
-6          6           4     4
-1.5        -1.5        1     1
-6          6           4     -4
4*1
3     0
F     0
Check

Pas 1 - ORDER: 1 2 3 4
  -4.000000

Pas 2 - ORDER: 2 3 4
   4.500000

Pas 3 - ORDER: 3 4
   1.666667

Pas 4 - ORDER: 4
  -8.000000

Numar de schimbari de linii = 0

Matricea L
   1.000000
   1.500000       1.000000
  0.3750000     -0.4166667       1.000000
   1.500000       1.000000       0.000000       1.000000

Matricea U
  -4.000000       1.000000       1.000000       1.000000
   4.500000       2.500000       2.500000
   1.666667       1.666667
  -8.000000

Determinant(A) =   240.0000
```

```
 * Solutia pentru termenii liberi nr. 1
 x(1) =  -0.2500000
 x(2) =  -0.2500000
 x(3) =   0.2500000
 x(4) =    0.000000

* Proba Ax =b:
               A*x -b
           1  0.0000000E+00
           2 -5.9604645E-08
           3  1.1920929E-07
           4 -5.9604645E-08
```

■

Exemplu – 2 (Fişier de date u3-p.dat):

```
Matrice 4x4 cu u22 =0; u33=0. Pivotare.
4 1
-4         1          1     1
-6         6          4     4
-1.5       -1.5       1     1
-6         6          4     -4
4*1
3     0
T     1
Check

Pas 1 - ORDER: 2 1 3 4
  -6.000000
pas 2 (test) - order: 4 3 1
 u(2,2) =    0.000000

Pas 2 - ORDER: 1 3 4
  -3.000000
pas 3 (test) - order: 4 3
 u(3,3) =    0.000000
```

```
Pas 3 - ORDER: 3 4
   1.666667

Pas 4 - ORDER: 4
  -8.000000

Numar de schimbari de linii = 1

Matricea L
   1.000000
  0.6666667       1.000000
  0.2500000       1.000000       1.000000
   1.000000       0.000000       0.000000       1.000000

Matricea U
  -6.000000       6.000000       4.000000       4.000000
  -3.000000      -1.666667      -1.666667
   1.666667       1.666667
  -8.000000

Determinant(A) =   240.0000

 * Solutia pentru termenii liberi nr. 1
 x(1) =  -0.2500000
 x(2) =  -0.2500000
 x(3) =   0.2500000
 x(4) =    0.000000

* Proba Ax =b:
               A*x -b
           1  0.0000000E+00
           2  0.0000000E+00
           3  0.0000000E+00
           4  0.0000000E+00
```

■

LU\Crout

Rezolvarea prin descompunerea LU – metoda Crout.

Fişiere:

LUdecomp-Crout.f90; LUsolve-Crout.f90; apvt-Crout.f90; compute_a_ii_Crout.f90; swap_row.f90; Main-Crout.f90; width.f90;

Semnificaţiile sunt aceleaşi ca ale fişierelor din metoda Doolittle.

Metoda:

Metoda este analogă cu Doolittle, cu deosebirea că acum se aleg arbitrari u_{ii}, iar pivotul este l_{ii}.

Metoda Crout pune $u_{ii} = 1$.

LU_Complex

Descompunere LU – matrice complexă, metoda Doolittle

Fişiere:

LUdecomp_z.f90; LUsolve_z.f90; apvt-LU_z.f90; compute_a_ii_z.f90; swap_row_z.f90; Main-LU_complex.f90; width.f90;

Semnificaţiile sunt aceleaşi ca ale fişierelor din metoda Doolittle.

Fişierul de date are aceeaşi structură ca fişierul de date din LU, cu particularităţile:

- Matricea **A**: elementele se introduc prin partea reală şi partea imaginară;
- Termenii liberi **b**: se introduc ca valori complexe, în forma (*x*, *y*). Parantezele sunt obligatorii.

Exemplu:

$$\mathbf{A} = \begin{bmatrix} 3-2i & 2+4i & -3i \\ 1+i & 2-6i & 1+2i \\ 4 & -5+i & 3-2i \end{bmatrix}; \qquad \mathbf{b} = \left[\begin{array}{c|c} 5-i & 10+5i \\ 4-3i & 6-7i \\ 2-i & -1+2i \end{array}\right].$$

Fişierul de date este (test-p.dat):

```
Test: solutia_1 =(1,0) (1,0) (1,0); solutia_2 =(1,-1) (2,1)
(0,3)
```

```
3 2
3 -2        2 4         0 -3
1 1         2 -6        1 2
4 0         -5 1        3 -2

(5,-1)      (4,-3)      (2,-1)
(10,5)      (6,-7)      (-1,2)
2     0
T     1
FProba

Pas 1 - ORDER: 3 2 1
 (4.000000,0.0000000E+00)

Pas 2 - ORDER: 2 1
 (3.500000,-5.000000)

Pas 3 - ORDER: 1
 (0.2046980,-0.4932886)

Numar de schimbari de linii = 1

Matricea L
   1.000000        0.000000
  0.2500000       0.2500000        1.000000        0.000000
  0.7500000      -0.5000000       0.3926175       0.7751678
1.000000        0.000000

Matricea U
   4.000000        0.000000       -5.000000        1.000000
3.000000       -2.000000
   3.500000       -5.000000      -0.2500000        1.750000
  0.2046980      -0.4932886

Determinant(A) =   7.000001        11.00000

 * Solutia pentru termenii liberi nr. 1
```

```
 x(1) =    1.000000     -0.1641187E-06
 x(2) =   0.9999999      0.9099531E-07
 x(3) =   0.9999998      0.2314062E-06

 * Solutia pentru termenii liberi nr. 2
 x(1) =    1.000000      -1.000000
 x(2) =    2.000000       1.000000
 x(3) =  -0.6782657E-06   3.000000

* Proba Ax =b:

Solutia nr. 1: A*x -b
           1 (-4.7683716E-07,2.3841858E-07)
           2 (-2.3841858E-07,2.3841858E-07)
           3 (-1.1920929E-07,0.0000000E+00)

Solutia nr. 2: A*x -b
           1 (0.0000000E+00,9.5367432E-07)
           2 (4.7683716E-07,-4.7683716E-07)
           3 (-1.1920929E-07,4.7683716E-07)

* Proba LU =PA: L*U
 (4.000000,0.0000000E+00) (-5.000000,1.000000) (3.000000,
 -2.000000)
 (1.000000,1.000000) (2.000000,-6.000000) (1.000000,2.000000)
 (3.000000,-2.000000) (2.000000,4.000000) (0.0000000E+00,
 -3.000000)
```

■

LU_Complex\Crout_Complex

Descompunere LU – matrice complexă, metoda Crout

Fişiere:

LU_Decomp_Crout_z.f90; LUsolve-Crout_z.f90; apvt-Crout_z.f90; compute_a_ii_Crout_z.f90; swap_row_z.f90; Main-Crout_Complex.f90; width.f90;

Semnificaţiile sunt aceleaşi ca în LU – metoda Crout.

Cholesky

Metoda Cholesky – cu matricea **L**.

Metoda:

Se aplică pentru un sistem cu *matrice simetrică şi pozitiv definită*. Pentru o astfel de matrice există o descompunere **LU**, în care **U** este transpusa lui **L**, adică $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. Metoda Cholesky nu necesită pivotare.

Paşii rezolvării sunt aceeaşi ca paşii LU, anume:

1) Factorizare: $\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2) Calculul lui **y**, prin substituţie înainte: $\mathbf{L}\mathbf{y} = \mathbf{b}$
3) Calculul soluţiei **x**, prin substituţie înapoi: $\mathbf{L}^T\mathbf{x} = \mathbf{y}$.

Pentru *n* = ”mare”, numărul de operaţii în metoda Cholesky este

$$NOP_{Cholesky} \approx \frac{n^3}{6}$$

Adică, $NOP_{Cholesk} \approx \frac{1}{2} NOP_{Gauss}$.

Fişiere:

Cholesky.f90; Forward.f90; Back.f90; Loca.f90; Main_Cholesky-2004.f90; width.f90

Subrutinele metodei sunt:

cholesky(a, kod): Descompunerea Cholesky (Pasul 1)

forward(l,b): Substituţie înainte (Pasul 2)

back(l,b): Substituţie înapoi (Pasul 3)

Ele se apelează astfel (în programul principal):

```
call Cholesky(a, kod)

do k =1,nb
call forward(a,b(:,k))
call back(a,b(:,k))
enddo
```

Parametrii formali sunt:

a: vector de dimensiune $n(n+1)/2$. Acesta stochează:

La intrare: matricea **A** (triunghiul inferior, pe linii);

La ieşire: matricea **L** (triunghiul inferior).

kod: cod de ieşire din descompunerea Cholesky:

0: ok; -1: Matricea nu este pozitiv definită.

l: vector de dimensiune $n(n+1)/2$. Acesta stochează matricea **L**.

b: vector de dimensiune *n*, care stochează la intrare termenii liberi, şi la ieşire soluţia. Anume:

Forward: **b**, şi **y**; Back: **y**, şi **x**.

La acestea, se adaugă o funcţie care calculează adresa elementului A(*i*, *j*) în vectorul de stocare `a`, anume Loca(i, j).

Main_Cholesky-2004.f90: programul principal

Observaţie

Programul principal stochează matricea **A** în vectorul `a` de dimensiune $n(n+1)/2$

■

width.f90: utilitar

Datele de intrare se citesc dintr-un fişier de date, care conţine:

Titlu: text, max. 80 caractere.

n, *nb*: *n* = numărul de ecuaţii; *nb* = numărul de termeni liberi.

a: triunghiul inferior din **A**.

b: termenii liberi; se scriu pe linii: *nb* linii a *n* termeni liberi.

kod: Cod de tipărire

check_text: text, max. 20 caractere. Primul caracter e semnificativ:

`C, c, P, p` : se verifica soluţia calculată.

Datele de ieşire se scriu în continuarea datelor de intrare.

Exemplu (Matricea **A**, din Jennings (1980), p. 106):

$$\mathbf{A}=\begin{bmatrix}16 & 4 & 8\\ 4 & 5 & -4\\ 8 & -4 & 22\end{bmatrix};\qquad \mathbf{b}=\left[\begin{array}{c:c:c}1 & 2 & 28\\ 1 & 2 & 5\\ 1 & 2 & 26\end{array}\right].$$

Fişier de date (2.dat):

```
Test pentru Cholesky
3     3
16
4     5
8     -4    22
1     1     1
2     2     2
28    5     26
1
Proba
```

```
 Matricea L
   4.000000
   1.000000       2.000000
   2.000000      -3.000000       3.000000

 * Solutia pentru termenii liberi nr. 1 - 3:
        1              2              3
 x1:  -0.1423611     -0.2847222       1.000000
 x2:   0.4583333      0.9166667       1.000000
 x3:   0.1805556      0.3611111       1.000000

* Proba Ax =b:

Solutia nr. 1: A*x -b
           1  0.0000000E+00
           2 -5.9604645E-08
           3  1.1920929E-07

Solutia nr. 2: A*x -b
           1  0.0000000E+00
           2 -1.1920929E-07
```

```
           3   2.3841858E-07

Solutia nr. 3: A*x -b
           1   0.0000000E+00
           2   0.0000000E+00
           3   0.0000000E+00
```

■

Cholesky_S

Metoda Cholesky – cu matricea **S**.

Metoda:

Metoda este aceeaşi cu cea expusă la Cholesky. Cu deosebirea că acum, matricea de referinţă este matricea $\mathbf{S} = \mathbf{L}^T$. Astfel, paşii sunt:

1. Factorizare: $\mathbf{A} = \mathbf{S}^T\mathbf{S}$
2. Calculul lui **y**, prin substituţie înainte: $\mathbf{S}^T\mathbf{y} = \mathbf{b}$
3. Calculul soluţiei **x**, prin substituţie înapoi: $\mathbf{S}\mathbf{x} = \mathbf{y}$.

Programul principal stochează acum, în vectorul *a*, triunghiul superior al lui **A** (pe coloane).

Datele de intrare şi ieşire sunt aceleaşi ca la Cholesky, cu deosebirea că matricea **A** se introduce prin elementele triunghiului superior.

Fişiere:

Cholesky_S.f90; Forward.f90; Back.f90; Loca.f90; Main_Cholesky_S.f90; width.f90;

Semnificaţiile sunt aceleaşi ca ale subrutinelor din Cholesky.

Exemplu:

Exemplul din Cholesky, pentru coloanele 1 şi 3 de termeni liberi.

Fişier de date: 1.dat

Cholesky_Band

Cholesky pentru matrici bandă (simetrice şi pozitiv definite).

Matrice bandă: elementele unei linii sunt alcătuite din:

- Elementul diagonal, un număr de *LIM*-1 elemente la stânga acestuia, şi *LIM*-1 elemente la dreapta.
- Celelalte elemente din linie sunt zero.

Numărul *LIM* se zice *semi-lăţimea* de bandă. *LIM* reprezintă numărul elementelor din semi-bandă, inclusiv elementul diagonal.

Observaţie

Între elementele din bandă, pot fi şi elemente nule, dar elementele situate în afara benzii sunt toate nule. În acest sens, *LIM* este semi-lăţimea de bandă *maximă* ■

De exemplu, o matrice 6×6, cu *LIM* = 3, are structura:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ & a_{22} & a_{23} & a_{24} & 0 & 0 \\ & & a_{33} & a_{34} & a_{35} & 0 \\ & & & a_{44} & a_{45} & a_{46} \\ & Simetric & & & a_{55} & a_{56} \\ & & & & & a_{66} \end{bmatrix}$$

Metoda:

Matricea bandă se presupune *simetrică şi pozitiv definită.*

Descompunerea $\mathbf{L}\mathbf{L}^T$ sau $\mathbf{S}^T\mathbf{S}$ poate fi făcută lucrând *exclusiv* în bandă. În ceea ce urmează se consideră descompunerea $\mathbf{A} = \mathbf{S}^T\mathbf{S}$, lucrând cu semi-banda superioară.

Paşii sunt cei de la Cholesky_S.

1) Descompunerea Cholesky:

Elementele active la un pas al descompunerii, sunt conţinute într-un triunghi cu laturile *LIM* – numit "triunghiul activ". În cursul procesului, triunghiul activ coboară cu câte o linie în bandă.

Metoda utilizează un vector de lucru Y, de dimensiune NY, unde se definesc:

$NY0 = LIM * (LIM + 1)/2$; $NY = NY0 + n * LIM$.

Acest vector este constituit din două părţi:

- Sub-vectorul Y1($1:NY0$), de dimensiune $LIM*(LIM+1)/2$: serveşte ca front de lucru, pentru descompunerea Cholesky; în acestea se generează elementele triunghiului activ la un pas (triunghi cu laturile *LIM*).
- Sub-vectorul YA($NY0+1:NY$), de dimensiune $n*LIM$: iniţial, stochează matricea **A**, pe linii. În cursul descompunerii Cholesky, stochează liniile procesate ale matricii A.

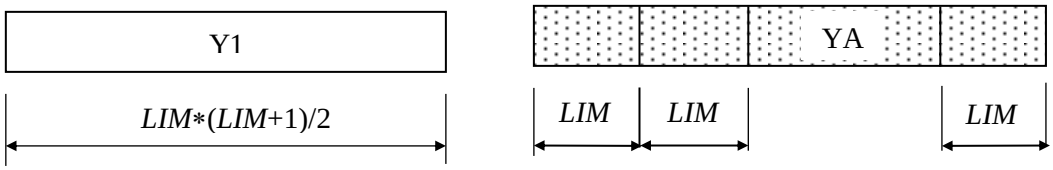


Figura 7 - Tablourile Y1 şi YA

2) Soluţia:

Aceasta se calculează prin substituţie înainte şi înapoi (paşii 2 şi 3).
Pentru alte consideraţii privind metoda (stocare, algoritm), v. Cap. 4, 5.6.

Fişiere:

AA.f90: funcţie care generează elementul (*i*, *j*) al matricii bandă *A*

Cholesky_Band.f90: descompunerea Cholesky în bandă

Loca.f90: funcţie care calculează adresa elementului *A*(*i*, *j*) în vectorul Y

Lof.f90: "Locul în front": adresa unui element din triunghiul activ, în vectorul Y1.

Main-Cholesky_Band_2005.f90: programul principal

Solve.f90: calculează soluţia

width.f90: utilitar

Fişier de date: b6-3.dat

Datele de intrare se citesc dintr-un fişier de date, care conţine:

Titlu: text, max. 80 caractere.

n, *LIM*, *nb*: *n* = numărul de ecuaţii; *LIM* = semi-lăţimea de bandă (inclusiv elementul diagonal); *nb* = numărul de termeni liberi.

A: semi-banda superioară din **A**, anume: se scriu *n* linii de *LIM* elemente; pentru ultimele linii se adaugă elemente nule (la dreapta), până la împlinirea numărului de *LIM* elemente.

b: termenii liberi: Se scriu pe linii; *nb* linii a *n* termeni liberi.

kod: cod de tipărire:

kod = 0: se tipăreşte soluţia;

kod ≠0: se tipăreşte şi matricea **S**;

kod ≥ 2: se tipăreşte şi proba $\mathbf{S}^T\mathbf{S} = \mathbf{A}$.

check_text: text, max. 20 caractere. Primul caracter este semnificativ:

`C, c, P, p`: se verifica soluţia.

Exemplu:

Se consideră sistemul definit de:

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & -1 & 0 & 0 & 0 \\ & 4 & -1 & -1 & 0 & 0 \\ & & 4 & -1 & -1 & 0 \\ & & & 4 & -1 & -1 \\ & \textit{Simetric} & & & 4 & -1 \\ & & & & & 4 \end{bmatrix}; \qquad \mathbf{b} = \left[\begin{array}{c:c} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{array}\right]$$

Fişier de date (b6-3.dat):

```
Test pentru Cholesky: matrice banda - semi-banda superioara
6     3     2
4     -1    -1
4     -1    -1
4     -1    -1
4     -1    -1
4     -1    0
4     0     0
6*1
2     1     0     0     1     2
1
Check
```

```
 Matricea A:
```

```
   4.000000      -1.000000      -1.000000
   4.000000      -1.000000      -1.000000
   4.000000      -1.000000      -1.000000
   4.000000      -1.000000      -1.000000
   4.000000      -1.000000
   4.000000


 Matricea S:
   2.000000     -0.5000000     -0.5000000
   1.936492     -0.6454972     -0.5163978
   1.825742     -0.7302967     -0.5477226
   1.788854     -0.7826238     -0.5590170
   1.757128     -0.8180962
   1.737302

 * Solutia pentru termenii liberi nr. 1 - 2:
        1              2
 x1:   0.9047619       1.000000
 x2:    1.190476      0.9999999
 x3:    1.428571      0.9999999
 x4:    1.428572       1.000000
 x5:    1.190476       1.000000
 x6:   0.9047620       1.000000

* Proba Ax =b:
Solutia nr. 1: A*x -b
           1  0.0000000E+00
           2 -2.3841858E-07
           3 -1.1920929E-07
           4  4.1723251E-07
           5  1.7881393E-07
           6  0.0000000E+00

Solutia nr. 2: A*x -b
           1  1.1920929E-07
           2 -1.7881393E-07
```

```
 3 -1.7881393E-07
 4  0.0000000E+00
 5 -5.9604645E-08
 6  4.7683716E-07
```

■

III-2 Analiza erorii şi condiţionare

Numar_Conditie

Calculul numărului de condiţie al unei matrici, după norma-∞ şi norma-1.

Metoda:

Se dă sistemul **Ax** = **b**. Considerăm o perturbaţie **r** a termenilor liberi, adică noi termeni liberi $\tilde{\mathbf{b}} = \mathbf{b} + \mathbf{r}$, unde $\| \mathbf{r} \| / \| \mathbf{b} \|$ este ”mic”. Sistemul devine $\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$.

Notăm perturbaţia soluţiei prin $\mathbf{e} = \tilde{\mathbf{x}} - \mathbf{x}$, şi avem

$$\frac{1}{Cond(\mathbf{A})} \le \frac{\| \mathbf{e} \| / \| \mathbf{x} \|}{\| \mathbf{r} \| / \| \mathbf{b} \|} \le Cond(\mathbf{A}),$$

în care:

Cond(**A**) = || **A** ||·|| **A**$^{-1}$ ||.

Numărul *Cond*(**A**) este numărul de condiţie al matricii **A**. Avem $Cond(\mathbf{A}) \ge 1$.

- Dacă *Cond*(**A**) ~ 1: **A** se zice *bine-condiţionată*;

  || **e** || / || **x** || este “mic”, adică este de ordinul lui $\| \mathbf{r} \| / \| \mathbf{b} \|$.

- Dacă *Cond*(**A**) >> 1: **A** se zice *rău-condiţionată*;

  || **e** || / || **x** || poate fi ”mare”, adică $\| \mathbf{e} \| / \| \mathbf{x} \| >> \| \mathbf{r} \| / \| \mathbf{b} \|$.

Numărul de condiţie depinde de norma considerată. În particular, se defineşte

$$Cond(\mathbf{A})_* = \rho(\mathbf{A})\rho(\mathbf{A}^{-1}),$$

în care: $\rho(\mathbf{A})$ este raza spectrală a matricii **A**, definită de $\rho(\mathbf{A}) = \max_i | \lambda_i(\mathbf{A}) |$ – unde $\lambda_i(\mathbf{A}), i = \overline{1,n}$ sunt valorile proprii ale matricii **A**. Avem, pentru orice normă, $Cond(\mathbf{A}) \ge Cond(\mathbf{A})_*$. Numărul $Cond(\mathbf{A})_*$ se calculează prin

$$Cond(\mathbf{A})_* = \frac{\max\limits_{\lambda \in \sigma(\mathbf{A})} |\lambda|}{\min\limits_{\lambda \in \sigma(\mathbf{A})} |\lambda|}.$$

Programul calculează numărul de condiţie pentru normele $\|.\|_\infty$ şi $\|.\|_1$, definite prin:

$$\| \mathbf{A} \|_\infty = \max_i \sum_j | a_{ij} |$$

Aceasta se zice *norma liniilor*, şi este maximul sumelor modulelor elementelor, pe linii.

$$\| \mathbf{A} \|_1 = \max_j \sum_i | a_{ij} |$$

Aceasta se zice *norma coloanelor*, şi este maximul sumelor modulelor elementelor, pe coloane.

Fişiere:
Main-Numar_Conditie.f90;Normlnf.f90; Norm1.f90; Elim2004.f90; width.f90;

Main-Numar_Conditie.f90: programul principal
Normlnf.f90, Norm1.f90: norma-∞ şi norma-1, ale unei matrici
Elim2004.f90: eliminarea Gauss; aceasta serveşte pentru calculul matricii inverse.
width.f90: utilitar

Subrutina de eliminare Gauss este ELIM2 (n, nb, A, B, prag, kod). Parametrii au aceleaşi semnificaţii ca la ELIM din Gauss (cu deosebirea că, între parametri, nu mai apare codul kodPrint).

Datele de intrare se citesc dintr-un fişier de date, cu structura de mai jos. Datele de ieşire se scriu în fişier, după datele de intrare.

Fişier de date (intrare/ieşire):
Titlu: text, max. 80 caractere
n: ordinul matricii **A**
Matricea **A**: $n \times n$; se scrie aşa cum e dată - pe linii şi coloane
Cod_Inversare_Coeficienţi, *Cod_Tipărire*: $0 / \neq 0$.

- *Cod_Inversare_Coeficienţi* $\neq 0$: se face $a(i, j) = 1./a(i, j)$
- *Cod_Tipărire* $\neq 0$: se tipăreşte şi $\mathbf{A}^{-1}$.

Cod_Norma: cod pentru normă. Valori:

0: norma-infinit (norma liniilor);

1: norma-1 (norma coloanelor)

2 (sau, alt număr decât 0 sau 1): norma-2: Nu este implementată; se ia implicit, norma-infinit.

Exemplu:

```
Test:
3
1 2 3
2 1 4
3 4 1
0    1
3    ! Cod norma

 Matricea inversa:
 -0.7500000      0.5000000      0.2500000
  0.5000000     -0.4000000      0.1000000
  0.2500000      0.1000000     -0.1500000

 Norma nedefinita: Se ia Norma-infinit (norma liniilor)

 Norm(A) =   8.000000
 Norm(A-invers) =   1.500000

 Numarul de conditie =   12.00000
```

Hilbert

Calculul inversei matricii Hilbert.

Metoda:

Matricea Hilbert de ordinul *n* este:

$$\mathbf{H}_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & & & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

Matricea Hilbert este cunoscută ca fiind rău-condiţionată, şi anume, cu atât mai rău-condiţionată cu cât ordinul *n* este mai mare.

Inversarea se face în modul obişnuit, anume: Coloanele $\mathbf{x}^{(i)}$ ale matricii inverse se calculează rezolvând sistemele liniare $\mathbf{H}_n\mathbf{x}^{(i)} = \mathbf{e}^{(i)}$, $i = \overline{1,n}$, unde $\mathbf{e}^{(i)}$ sunt coloanele matricii unitate de ordinul *n*.

Fişiere:

Main-Hilbert2004.f90: programul principal

Elim2004.f90: eliminarea Gauss

width.f90: utilitar

În principiu, pentru această problemă se pot utiliza oricare din metodele expuse anterior, pentru sisteme liniare generale (Gauss, LU).

Avantajul este că, programul principal generează matricea Hilbert de ordinul *n*, şi termenii liberi pentru ordinul *n* (coloanele matricii unitate).

(În loc de, specificarea acestor date în fişierul de intrare, cum ar fi necesar în metodele pentru o matrice generală.)

Rezolvarea se face prin eliminare Gauss, prin apelul subrutinei ELIM2. Parametrii formali sunt aceeaşi ca la subrutina ELIM din Gauss (cu deosebirea că între parametri nu mai figurează codul kodPrint).

Datele de intrare se introduc de la terminal şi constau în: ordinul *n*; o dată logică (`T/F`) pentru verificarea sau nu, a inversei calculate **X**.

Pentru verificare, se calculează produsul $\mathbf{H}_n\mathbf{X}$; ar trebui să avem proba $\mathbf{H}_n\mathbf{X} = \mathbf{I}_n$, unde $\mathbf{I}_n$ este matricea unitate de ordinul *n*.

Faptul că matricea Hilbert este rău-condiţionată, se poate constata din următoarele rezultate:

- Elementele inversei calculate sunt afectate de erori cu atât mai mari, cu cât ordinul este mai mare. (Erorile se apreciază în raport cu valorile exacte – care se cunosc analitic, sau în raport cu valorile calculate în dublă precizie);
- Inversa calculată verifică tot mai rău proba, odată cu creşterea lui *n*: adică, erorile $\mathbf{H}_n\mathbf{X}-\mathbf{I}_n$ sunt tot mai mari.

Exemplu:

Pentru $n = 5$, produsul $\mathbf{H}_n\mathbf{X}$ este:

```
 0.9999927       1.5778195E-04  -1.0029844E-03   1.2790319E-04
1.3136200E-04
-4.5019597E-06    1.000019       6.0700515E-04   5.3869281E-04
-1.0658416E-04
 7.5124717E-07   5.9086320E-05    1.000850      -2.8395746E-04
3.5124447E-04
 0.000000        0.000000        0.000000        1.000000
0.000000
 8.1506332E-06  -1.1893210E-04   2.6392794E-04  -7.2401995E-04
1.000145
```

■

III-3 Metode iterative

Jacobi

Metoda Jacobi – pentru sisteme liniare.

Exemplu numeric:

Fie sistemul:

$$8x_1 + x_2 - x_3 = 8$$

$$2x_1 + x_2 + 9x_3 = 12$$

$$x_1 - 7x_2 + 2x_3 = -4$$

Rezolvăm fiecare ecuaţie *i* = 1, 2, 3, în raport cu necunoscuta x_i, căutând ca aceasta să fie necunoscuta cu coeficientul cel mai mare din ecuaţie, eventual rearanjând ecuaţiile. Intervertind ecuaţiile 2 şi 3, avem:

$$x_1 = 1 - (1/8)x_2 + (1/8)x_3$$

$$x_2 = 4/7 + (1/7)x_1 + (2/7)x_3$$

$$x_3 = 12/9 - (2/9)x_1 - (1/9)x_2$$

Sau matriceal:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 4/7 \\ 12/9 \end{bmatrix} + \begin{bmatrix} 0 & -1/8 & 1/8 \\ 1/7 & 0 & 2/7 \\ -2/9 & -1/9 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (1')$$

care este de forma

$$\mathbf{x}^{(m+1)} = \mathbf{g} + \mathbf{M}\mathbf{x}^{(m)}$$

Se iterează, cu aproximaţia iniţială $\mathbf{x}^{(0)} = [1 \quad 4/7 \quad 12/9]^T$; testul de oprire a iteraţiei este $\| \mathbf{x}^{(m+1)} - \mathbf{x}^{(m)} \|_\infty \le 1E-6$. La iteraţia 12 rezultă soluţia $(1.0, 1.0, 1.0)$.

■

Metoda:

Fie sistemul dat $\mathbf{Ax} = \mathbf{b}$. Se rezolvă fiecare ecuaţie *i* în raport cu necunoscuta x_i.

Explicitând x_i se obţine:

$$x_i = (b_i - \sum_{j=1,\, j\neq i}^{n} a_{ij} x_j) / a_{ii}, \qquad i = 1, 2, \ldots, n$$

S-a presupus $a_{ii} \neq 0$. Iteraţia Jacobi va fi:

$$\begin{cases} \mathbf{x}^{(0)} = arbitrar \\ x_i^{(m+1)} = (b_i - \sum_{j=1,\, j\neq i}^{n} a_{ij} x_j^{(m)}) / a_{ii}, \quad i = 1, 2, ..., n; \quad m \ge 0 \end{cases}$$

Testul de oprire a iteraţiei este $\| \mathbf{x}^{(m+1)} - \mathbf{x}^{(m)} \|_\infty \le eps$.

Metoda Jacobi se mai zice metoda "iteraţiilor simultane", pentru că, coordonatele x_i ale soluţiei **x** se calculează independent unele de altele. (Pentru alt mod de calcul – v. metoda Gauss_Seidel, mai jos.)

Condiţie suficientă de convergenţă:

Matricea **A** este *diagonal dominantă*, adică avem:

$$|a_{ii}| > \sum_{j=1,\, j\neq i}^{n} |a_{ij}|\,, \quad i = \overline{1,n}$$

Pentru alte consideraţii privind metoda, v. Cap. 4-IV.

Fişiere:

Jacobi.f90: programul principal – care implementează metoda

Norm.f90: norma-∞ a unui vector

width.f90: utilitar

Datele de intrare se citesc dintr-un fişier de date. Acesta are structura:

- *Titlu*: linie-text, max. 80 caractere.
- *n*: ordinul matricii
- *Matricea* **A**: $n \times n$; se scrie pe linii şi coloane – aşa cum este dată.
- *Termenul liber* **b**: se scriu în linie, *n* valori b_i.
- *eps, lnit*: toleranţa, numărul limită de iteraţii.
- *kodPrint*: cod de tipărire iteraţii: 0 / ≠ 0: Da / Nu.
- *check_text*: text de max. 20 caractere. Primul caracter este semnificativ:
  `C, c, P, p`: se face proba – se tipăreşte vectorul **Ax** – **b**.

Datele de ieşire se scriu în fişier, după datele de intrare – v. mai jos.

Exemplul numeric de mai sus:

Fişierul de intrare/ieşire (Ex1.dat):

```
Exemplu pt. Jacobi
3
8     1     -1
1     -7    2
```

```
2     1     9
8     -4    12
1e-6  50
0
Proba

 Toleranta atinsa.
 Numar de iteratii: 12

 Solutia:
 1     1.000000
 2     1.000000
 3     1.000000

 Proba "Ax-b":
 1    9.5367432E-07
 2   -4.7683716E-07
 3    1.4305115E-06
```

■

Gauss_Seidel

Metoda Gauss-Seidel.

Se modifică metoda Jacobi, astfel că, la calculul coordonatei $x_i^{(m)}$ se utilizează valorile $x_1^{(m)}$, …, $x_{i-1}^{(m)}$ deja calculate, care în general, sunt aproximaţii mai bune ale soluţiei.

Exemplu:

Considerăm exemplul numeric din Jacobi. Ecuaţiile se înlocuiesc cu cele de mai jos, care ar trebui să ofere o convergenţă mai rapidă:

$$x_1^{(1)} = 1 - (1/8)x_2^{(0)} + (1/8)x_3^{(0)}$$

$$x_2^{(1)} = 4/7 + (1/7)x_1^{(1)} + (2/7)x_3^{(0)}$$

$$x_3^{(1)} = 12/9 - (2/9)x_1^{(1)} - (1/9)x_2^{(1)}$$

Într-adevăr, cu $\mathbf{x}^{(0)} = [0 \;\; 0 \;\; 0]^T$ şi aceeaşi toleranţă $1E-6$, soluţia $(1.0, 1.0, 1.0)$ se obţine la iteraţia 9. Rezidualul maxim este 0.0.

■

Metoda:

Formulele generale ale metodei Gauss-Seidel sunt:

$$\begin{cases} \mathbf{x}^{(0)} = arbitrar \\ x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)}) / a_{ii}, \quad i = 1,2,...,n; \quad m \geq 0 \end{cases}$$

Testul de oprire a iteraţiei este $\| \mathbf{x}^{(m+1)} - \mathbf{x}^{(m)} \|_\infty \leq eps$.

Metoda Gauss-Seidel se mai zice metoda "iteraţiilor succesive", pentru că la un pas $m+1$, $m \geq 0$, de îndată ce o coordonată $x_j^{(m+1)}$ a soluţiei este calculată, ea se utilizează în ecuaţiile pentru coordonatele următoare $x_i^{(m+1)}$, $i > j$.

Condiţii suficiente de convergenţă:

- matricea **A** este *diagonal dominantă*.
- matricea **A** este *simetrică şi pozitiv definită*.

Când ambele metode Jacobi şi Gauss-Seidel converg, metoda Gauss-Seidel converge mai rapid.

Pentru alte consideraţii privind convergenţa, v. Cap. 4-IV.

Fişiere:

Gauss_Seidel.f90: programul principal

norm.f90: norma-∞ a unui vector

width.f90: utilitar

Datele de intrare se citesc dintr-un fişier de date. Acesta are aceeaşi structură ca şi fişierul de date din Jacobi.

Exemplu (Ex1.dat):

Exemplul de la Jacobi. Rezultatele sunt, acum:

```
Toleranta atinsa.
Numar iteratii: 9
```

```
Solutia:
x(1) =    1.000000
x(2) =    1.000000
x(3) =    1.000000

Proba "Ax-b":
1     0.000000
2     0.000000
3     0.000000
```

■

Sor

Metoda relaxării (Succesive Over-Relaxation).

Metoda:

Reluăm formula metodei Gauss-Seidel:

$$x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)}) / a_{ii}, \quad i = 1, 2, ..., n; \quad m \geq 0$$

Formula se pune sub forma următoare, adunând şi scăzând $\mathbf{x}_i^{(m)}$ în membrul doi (observaţi că acum, în a doua sumă, indicele *j* ia valori de la *i* şi nu de la $i+1$):

$$x_i^{(m+1)} = x_i^{(m)} + (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(m)}) / a_{ii}, \quad i = \overline{1,n}$$

Termenul care se adună la $\mathbf{x}_i^{(m)}$ este diferenţa $(\mathbf{x}_i^{m+1} - \mathbf{x}_i^{m})$. Metoda SOR constă în înmulţirea acestei diferenţe cu un factor de accelerare (sau relaxare) $\omega > 1$. Întrucât $\omega > 1$, metoda se zice *supra-relaxare*. Alegerea lui ω se discută mai jos. Formula metodei SOR este deci

$$x_i^{(m+1)} = x_i^{(m)} + \omega(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(m)}) / a_{ii}, \quad i = \overline{1,n}$$

sau, explicitând $\mathbf{x}_i^{(m)}$ din a doua sumă:

$$x_i^{(m+1)} = \omega(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)}) / a_{ii} + (1-\omega) x_i^{(m)}, \quad i = \overline{1,n}$$

Se notează expresia din prima paranteză cu $\mathbf{z}_i^{(m+1)}$ (aceasta este coordonata *i* a iteratei $(m+1)$ din metoda Gauss-Seidel.) .

Formula de iterare este echivalentă cu următoarele ecuaţii considerate pentru $i = \overline{1,n}$:

$$z_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)}) / a_{ii},$$

$$x_i^{(m+1)} = \omega z_i^{(m+1)} + (1-\omega) x_i^{(m)}$$

În cele mai multe cazuri, valoarea optimă a lui ω satisface relaţia

$1 < \omega < 2.$

În practică, se poate alege ω, astfel: se utilizează valori ω de test, pe un număr limitat de iteraţii; se alege ca valoare optimă acel ω, pentru care convergenţa este cea mai rapidă.

Fişiere:
SOR.f90: programul principal
norm.f90: norma-∞ a unui vector
width.f90: utilitar

Datele de intrare se citesc dintr-un fişier de date. Acesta are aceeaşi structură ca şi fişierul de date din Jacobi, cu deosebirea: după datele *eps*, *lnit*, se introduce data:

- *omega*: factorul de accelerare ω

Exemplu:
Considerăm exemplul de mai jos. Pentru acesta, metoda Gauss-Seidel face 27 de iteraţii şi dă soluţia $\mathbf{x}^{(27)}$ = [0.9999999 0.9999999 0.9999999 0.9999999 0.9999999]^T. Alegem $\omega = 1.3$ (valoarea optimă). Se obţin rezultatele de mai jos.

Fişierul de intrare/ieşire:

```
Test pentru Gauss-Seidel & SOR. Solutia: 1,1,1,1,1.
5
4     -1    -1    0     0
-1    4     -1    -1    0
```

```
-1    -1    4     -1    -1
0     -1    -1    4     -1
0     0     -1    -1    4
2     1     0     1     2
1.E-7   50
1.3
0
P

 Toleranta atinsa.
 Numar iteratii: 16

 Solutia:
 x(1) =   1.000000
 x(2) =   1.000000
 x(3) =   1.000000
 x(4) =   1.000000
 x(5) =   1.000000

 Proba "Ax-b":
 1    0.000000
 2    0.000000
 3    0.000000
 4    0.000000
 5    0.000000
 ■
```

CAPITOLUL IV – VALORI ŞI VECTORI PROPRII

Folder tematic: **$ Eigen**

Foldere:

Power; Power_Complex; Inverse_Power_Shift; Inverse_Power_Complex;

Sim_Iter: metoda puterii şi variante ale acesteia.

Jacobi; QR: metodele Jacobi şi QR.

Problema de valori proprii (sumar)

Fie dată matricea $\mathbf{A}$ $n \times n$, reală sau complexă.

Dacă vectorul $\mathbf{x} \neq \mathbf{0}$ şi scalarul λ satisfac relaţia

$$\mathbf{Ax} = \lambda \mathbf{x}, \tag{1}$$

atunci: λ se numeşte *valoare proprie,* iar $\mathbf{x}$ *vectorul propriu* asociat lui λ.

Dacă $\mathbf{x}$ şi $\mathbf{y}$ sunt asociaţi cu λ, atunci şi $\alpha\mathbf{x} + \beta\mathbf{y}$, α, β scalari, este asociat lui λ.

Dacă $\mathbf{P}$ este o matrice $n \times n$ nesingulară, matricea

$$\mathbf{A}' = \mathbf{P}^{-1}\mathbf{AP}$$

se zice *similară* cu $\mathbf{A}$. $\mathbf{P}$ se zice matrice de transformare.

Matricile similare au aceleaşi valori proprii. Dacă $\mathbf{x}'$ sunt vectorii proprii ai lui $\mathbf{A}'$, vectorii proprii ai lui $\mathbf{A}$ se găsesc din relaţia

$$\mathbf{x} = \mathbf{Px}'.$$

Mai multe metode numerice transformă matricea $\mathbf{A}$ într-o matrice similară $\mathbf{A}'$, de o formă mai simplă, determinând valorile proprii şi vectorii proprii ai matricii $\mathbf{A}'$; apoi, se determină vectorii proprii ai lui $\mathbf{A}$, cu relaţia de mai sus.

Polinomul caracteristic:

Relaţia (1) este ecuaţia din care se găsesc λ şi $\mathbf{x}$. Aceasta se mai scrie

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \tag{2}$$

unde $\mathbf{I}$ este matricea unitate. Explicit,

$$\begin{bmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn}-\lambda \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (2')$$

Condiţia ca (2) să aibă soluţii netriviale **x**, este $\det(\mathbf{A}-\lambda\mathbf{I})=0$. Explicit,

$$p(\lambda)=\begin{vmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn}-\lambda \end{vmatrix}=0 \qquad (3)$$

$p(\lambda)$ se zice *polinomul caracteristic* al matricii **A**, şi $p(\lambda) = 0$ – ecuaţia caracteristică.

Fie determinantul desvoltat:

$$p(\lambda)=(-1)^n\lambda^n+c_1\lambda^{n-1}+\ldots+c_{n-1}\lambda+c_n$$

Proprietăţi ale coeficienţilor şi valorilor proprii:

$c_n=\det(\mathbf{A})$; $\qquad \lambda_1\lambda_2\ldots\lambda_n=\det(\mathbf{A})$.

$c_1=(-1)^{n-1}\sum_1^n a_{ii}$; $\qquad \lambda_1+\lambda_2+\ldots+\lambda_n=\sum_1^n a_{ii}$.

Notă: $Tr(\mathbf{A})=\sum_1^n a_{ii}$ se zice urma matricii **A**.

În general:

$c_i=(-1)^{n-i}\times$ Suma minorilor principali de ordin *i* ai matricii **A**.

Observaţii

1. Ordonarea valorilor proprii:

   Valorile proprii se ordonează în şirul $\lambda_1,\lambda_2,\ldots,\lambda_n$. În acest şir, o rădăcină multiplă de ordinul *r*, se repetă de *r* ori. Uzual, valorile proprii se indexează în ordinea descrescătoare a modulului, adică $|\lambda_1|\geq|\lambda_2|\geq\ldots\geq|\lambda_n|$. Valoarea proprie λ_1 se zice *dominantă*. Mulţimea valorilor proprii $\{\lambda_i, i=\overline{1,n}\}$ se zice *spectrul* matricii **A**.

2. Vectorii proprii asociaţi unei valori proprii:

   După determinarea valorilor proprii, vectorii proprii asociaţi cu λ_i se găsesc punând $\lambda = \lambda_i$ în (2'), şi rezolvând sistemul liniar şi omogen (2').

   - Dacă **A** este reală: În general, λ_i poate fi complex, şi atunci vectorul propriu $\mathbf{x}_i$ asociat cu λ_i este complex. Dacă **A** şi λ_i sunt reali, atunci vectorul propriu $\mathbf{x}_i$ este real.
   - Dacă **A** este complexă: În general, valorile proprii şi vectorii proprii sunt mărimi complexe. În particular, unele dintre acestea pot fi şi reale ■

   Subspaţiu propriu şi dimensiunea acestuia:

   Sistemul (2') este omogen, astfel că dacă **x**, **y** sunt soluţii, atunci $\alpha\mathbf{x} + \beta\mathbf{y}$ sunt soluţii. Adică, lui λ_i îi este asociat un subspaţiu liniar S_i de soluţii **x**. Se arată că:

   Dimensiunea subspaţiului S_i este *mai mică decât sau egală cu* ordinul de multiplicitate a rădăcinii λ_i ■

   Notând r_i = ordinul de multiplicitate a rădăcinii λ_i, şi p_i = dimensiunea subspaţiului S_i, avem: $p_i \le r_i$. Cu alte cuvinte, există cel mult r_i vectori liniar independenţi în S_i.

   Dacă $p_i < r_i$, valoarea λ_i se zice *defectivă*; în acest caz, şi matricea **A** se zice *defectivă*.

   (r_i se mai zice *multiplicitate algebrică*, iar p_i *multiplicitate geometrică*.)

3. Determinarea efectivă a sistemului propriu:

   Pentru calculaţia practică ($n > 3$), *nu se recomandă*:

   - *Rezolvarea directă* a ecuaţiei caracteristice. Aceasta, datorită faptului că problema calculului rădăcinilor unui polinom este foarte sensibilă la mici perturbaţii în coeficienţi; aceste perturbaţii apar din erorile de rotunjire.
   - *Calculul direct al vectorilor proprii*, din (2').

Metode numerice pentru găsirea valorilor proprii λ_i, şi a vectorilor proprii asociaţi $\mathbf{x}^{(i)}$, sunt prezentate în continuare.

■

Matrici hermitiene şi unitare

Operaţia $*$ aplicată unui vector sau unei matrici notează *transpus-conjugata*.

Dacă **x** este un vector, respectiv **A** o matrice, atunci:

$\mathbf{x}^* = \overline{\mathbf{x}}^T$; $\mathbf{A}^* = \overline{\mathbf{A}}^T$.

Explicit: $\mathbf{A} = [a_{ij}]$, $\mathbf{A}^* = [a_{ij}^*]$, avem: $a_{ij}^* = \overline{a}_{ji}$.

În expresiile anterioare, bara notează conjugata: $\overline{\mathbf{x}} = [\overline{x}_i]$; $\overline{\mathbf{A}} = [\overline{a}_{ij}]$.

Pentru un vector real, sau o matrice reală, operaţia $*$ revine la transpunere:

$\mathbf{x}^* = \mathbf{x}^T$; $\mathbf{A}^* = \mathbf{A}^T$.

În particular, pentru un scalar, operaţia revine la conjugare: $s^* = \overline{s}$.

Matricea **A** se zice *hermitiană*, dacă $\mathbf{A}^* = \mathbf{A}$.

Exemplu de matrice hermitiană:

$$\mathbf{A} = \begin{bmatrix} 1 & 1-7i & -i \\ 1+7i & 5 & 10-3i \\ i & 10+3i & -2 \end{bmatrix}$$

Remarcaţi că elementele diagonale sunt reale, iar elementele simetrice sunt conjugate.

O matrice reală este hermitiană, dacă este simetrică ($\mathbf{A}^T = \mathbf{A}$).

O matrice hermitiană se zice *pozitiv definită*, dacă $\forall \mathbf{x} \neq \mathbf{0} \quad \mathbf{x}^* \mathbf{A} \mathbf{x} > 0$ ($\mathbf{x}^* \mathbf{A} \mathbf{x}$ este real).

În particular, o matrice reală este pozitiv definită, dacă: $\forall \mathbf{x} \neq 0 \quad \mathbf{x}^T \mathbf{A} \mathbf{x} > 0$.

Matrici unitare:

O matrice se zice *unitară*, dacă $\mathbf{U}^* \mathbf{U} = \mathbf{I}$; echivalent, $\mathbf{U}^{-1} = \mathbf{U}^*$.

O matrice *reală* este unitară dacă $\mathbf{U}^T \mathbf{U} = \mathbf{I}$, sau $\mathbf{U}^{-1} = \mathbf{U}^T$.

Valorile proprii ale unei matrici unitare au modulul egal cu 1.

Proprietăţi ale matricilor hermitiene (în particular, reale şi simetrice):

P1. Dacă **A** este hermitiană, şi are valorile proprii $\{\lambda_i\}$ distincte sau nu, atunci:

(a) Există o matrice unitară **U**, astfel că $\mathbf{U}^*\mathbf{AU}$ este diagonală (se zice că **U** diagonalizează pe **A**), şi avem:

$$\mathbf{U}^*\mathbf{AU} = diag(\lambda_1, \ldots, \lambda_n).$$

(b) Există *n* vectorii proprii liniar independenţi care formează o bază ortonormată în $\mathbf{C}^n$ (aceştia sunt coloanele lui **U**);

(c) Valorile proprii sunt reale

■

În particular, pentru **A** reală şi simetrică:

(a-1) Există **U** unitară şi reală, astfel că $\mathbf{U}^T\mathbf{AU} = diag(\lambda_1, \ldots, \lambda_n)$.

(b-1) Există *n* vectorii proprii liniar independenţi; aceştia formează o bază ortonormată în $\mathbf{R}^n$ (coloanele lui **U**);

(c-1) Valorile proprii sunt reale, şi vectorii proprii sunt reali ■

Avem şi proprietatea:

P1'. Dacă **A** este hermitiană (reală şi simetrică) şi *pozitiv definită,* atunci valorile proprii ale lui **A** sunt reale şi *pozitive* ■

Produs scalar şi proprietăţi de ortogonalitate:

1. Spaţiu vectorial real V_n

Fie *x* un vector din V_n, şi **x** matricea coloană a coordonatelor sale într-o bază a lui V_n ($\mathbf{x} \in \mathbf{R}^n$).

Dacă matricea **A** reală, este *simetrică şi pozitiv definită*, se defineşte produsul scalar în raport cu matricea **A**, prin:

$$< x, y >_A = \mathbf{x}^T\mathbf{Ay} = \mathbf{y}^T\mathbf{Ax}$$

În particular, pentru $\mathbf{A} = \mathbf{I}$, acesta devine produsul scalar standard:

$< x, y > = \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$

Avem: $< y, x >_A =< x, y >_A$; $< y, x > =< x, y >$.

Vectorii *x* şi *y* se zic *ortogonali* (relativ la produsul scalar), dacă $< x, y >_A = 0$,

sau $< x, y > = 0$ ■

Astfel:

- Vectorii *x* şi *y* sunt *ortogonali relativ la matricea* **A** , dacă

  $\mathbf{x}^T \mathbf{A}\mathbf{y} = 0$, sau $\mathbf{y}^T \mathbf{A}\mathbf{x} = 0$.

- Vectorii sunt *ortogonali*, dacă $\mathbf{x}^T \mathbf{y} = 0$, sau $\mathbf{y}^T \mathbf{x} = 0$.

2. Spaţiu vectorial complex V_n

Fie *x* un vector din V_n, şi **x** matricea coloană a coordonatelor într-o bază a lui V_n ($\mathbf{x} \in \mathbf{C}^n$).

Dacă **A** este o matrice *hermitiană şi pozitiv definită*, se defineşte produsul scalar în raport cu matricea **A**, prin:

$< x, y >_A = \mathbf{x}^T \mathbf{A}\overline{\mathbf{y}} = \mathbf{y}^* \mathbf{A}^T \mathbf{x}$.

(Ultima egalitate rezultă din aceea că scalarul $s =< x, y >_A$ este egal cu transpusul său).

Produsul scalar definit de $\mathbf{A} = \mathbf{I}$ este:

$< x, y > = \mathbf{x}^T \overline{\mathbf{y}} = \mathbf{y}^* \mathbf{x}$

Avem: $< y, x >_A = \overline{< x, y >_A}$, şi $< y, x > = \overline{< x, y >}$.

Ortogonalitatea a doi vectori se defineşte ca înainte, prin condiţia $< x, y >_A = 0$,

sau $< x, y > = 0$ ■

Observaţie: Dacă avem $< x, y >_A = 0$, avem şi $< y, x >_A = 0$ ■

Astfel:

- Vectorii **x**, **y** sunt o*rtogonali în raport cu* **A**, dacă:

  $\mathbf{x}^T \mathbf{A}\overline{\mathbf{y}} = 0$, sau $\mathbf{y}^* \mathbf{A}^T \mathbf{x} = 0$.

- Vectorii **x**, **y** sunt o*rtogonali*,dacă:

  $\mathbf{x}^T\overline{\mathbf{y}} = 0$, sau $\mathbf{y}^*\mathbf{x} = 0$.

P2. Dacă **A** este hermitiană, atunci:

- Vectorii proprii asociaţi la două valori proprii distincte sunt *ortogonali*: dacă $\lambda_1 \neq \lambda_2$, atunci $\mathbf{x}_2^*\mathbf{x}_1 = 0$ şi $\mathbf{x}_1^*\mathbf{x}_2 = 0$.
- Avem şi: $\mathbf{x}_2^*\mathbf{A}\mathbf{x}_1 = 0$, $\mathbf{x}_1^*\mathbf{A}\mathbf{x}_2 = 0$.

  (Dacă **A** hermitiană este şi pozitiv definită, vectorii $\mathbf{x}_1, \mathbf{x}_2$ sunt *ortogonali relativ la matricea* $\mathbf{A}^T$.)

  ■

Dacă **A** este reală şi simetrică:

- Vectorii proprii asociaţi la două valori proprii *distincte* sunt *ortogonali*: $\mathbf{x}_2^T\mathbf{x}_1 = 0$, $\mathbf{x}_1^T\mathbf{x}_2 = 0$.
- Avem şi: $\mathbf{x}_2^T\mathbf{A}\mathbf{x}_1 = 0$, $\mathbf{x}_1^T\mathbf{A}\mathbf{x}_2 = 0$.

  (Dacă **A** reală şi simetrică, este şi pozitiv definită, vectorii $\mathbf{x}_1, \mathbf{x}_2$ sunt *ortogonali relativ la matricea* **A**)

  ■

Câtul Rayleigh

Fie **A** o matrice $n \times n$ complexă (sau reală). Fie $\mathbf{v} \in \mathbf{C}^n$ un vector arbitrar, şi definim

$$\rho(\mathbf{v}) = \frac{\mathbf{v}^*\mathbf{A}\mathbf{v}}{\mathbf{v}^*\mathbf{v}}$$

$\rho(\mathbf{v})$ se numeşte *câtul Rayleigh.*

Proprietate-1: Dacă **x** este vector propriu asociat cu λ, atunci $\rho(\mathbf{x}) = \lambda$ ■

Astfel, câtul Rayleigh poate fi utilizat pentru a găsi o aproximare a valorii proprii λ, dacă se cunoaşte o aproximare **v** a vectorului propriu **x**: $\mathbf{v} \approx \mathbf{x} \Rightarrow \lambda \approx \rho(\mathbf{v})$.

Proprietate-2: Dacă matricea este hermitiană, câtul Rayleigh este mărginit de valorile proprii extreme ■

Valorile proprii sunt reale; fie acestea $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, atunci, avem:

$\lambda_n \leq \rho(\mathbf{v}) \leq \lambda_1$, pentru orice $\mathbf{v} \in \mathbf{C}^n$.

■

Power

Metoda puterii – matrice reală, cu valori proprii reale.

Metoda:
Metoda puterii determină valoarea proprie dominantă şi vectorul propriu asociat, ale unei matrici **A**, reale sau complexe. Aplicată la inversa $\mathbf{A}^{-1}$ metoda determină valoarea proprie cea mai mică (în modul), şi vectorul propriu asociat cu aceasta. Se presupune că:

1. Există un set de *n* vectori proprii *liniar independenţi*.
2. Există *o singură* valoare proprie dominantă $\lambda_1 : |\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$

Metoda este următoarea:

Fie $\mathbf{w}^{(0)}$ un vector iniţial, ales arbitrar – cu singura condiţie ca să aibă o componentă în direcţia lui $\mathbf{x}^{(1)}$. Pentru a împlini această cerinţă, unele coduri generează un vector aleator. (În fapt, $\mathbf{w}^{(0)}$ este o aproximaţie a vectorului propriu $\mathbf{x}^{(1)}$; dacă o astfel de aproximaţie este cunoscută, atunci aceasta accelerează iteraţia de mai jos).

Desvoltăm $\mathbf{w}^{(0)}$ în baza vectorilor proprii:

$$\mathbf{w}^{(0)} = a_1\mathbf{x}^{(1)} + a_2\,\mathbf{x}^{(2)} + \ldots + a_n\mathbf{x}^{(n)} \qquad \text{(a)}$$

Presupunem că $a_1 \neq 0$, şi formăm şirul de vectori

$$\mathbf{w}^{(k+1)} = \mathbf{A}\mathbf{w}^{(k)} = \mathbf{A}^{k+1}\mathbf{w}^{(0)}, \quad k \geq 0$$

Avem:

$$\mathbf{w}^{(1)} = \mathbf{A}\mathbf{w}^{(0)} = \lambda_1 a_1\mathbf{x}^{(1)} + \lambda_2 a_2\,\mathbf{x}^{(2)} + \ldots + \lambda_n a_n\mathbf{x}^{(n)}$$
$$= \lambda_1\left[a_1\mathbf{x}^{(1)} + \left(\frac{\lambda_2}{\lambda_1}\right)a_2\,\mathbf{x}^{(2)} + \ldots + \left(\frac{\lambda_n}{\lambda_1}\right)a_n\mathbf{x}^{(n)}\right]$$

În general,

$$\mathbf{w}^{(k)} = \mathbf{A}^k \mathbf{w}^{(0)} = (\lambda_1)^k \left[a_1 \mathbf{x}^{(1)} + \left(\frac{\lambda_2}{\lambda_1} \right)^k a_2 \, \mathbf{x}^{(2)} + \ldots + \left(\frac{\lambda_n}{\lambda_1} \right)^k a_n \mathbf{x}^{(n)} \right] \qquad \text{(b)}$$

Cum $|\lambda_1| > |\lambda_i|$ pentru $i \geq 2$, urmează că rapoartele $(\lambda_i / \lambda_1)^k$ tind la 0 pentru $k \to \infty$. Astfel, pentru k crescător, vectorii $\mathbf{w}^{(k)}$ se aliniază din ce în ce mai mult la direcţia vectorului propriu $\mathbf{x}^{(1)}$. În consecinţă, pentru un *k* suficient de mare, avem $\mathbf{w}^{(k)} \approx (\lambda_1)^k a_1 \mathbf{x}^{(1)}$. Să considerăm de asemenea relaţia $\mathbf{w}^{(k+1)} \approx (\lambda_1)^{k+1} a_1 \mathbf{x}^{(1)}$. Luând orice coordonată non-zero a lui $\mathbf{w}^{(k+1)}$, $\mathbf{w}^{(k)}$, să zicem cea de-a *m*-a coordonată, obţinem

$$\lambda_1 \approx \frac{w_m^{(k+1)}}{w_m^{(k)}}$$

Dezavantajul formulei precedente este că, coordonatele nenule ale lui $\mathbf{w}^{(k)}$ devin fie foarte mici ($|\lambda_1| < 1$), fie foarte mari ($|\lambda_1| > 1$), odată cu creşterea lui *k*. Aceasta se evită prin normalizarea (sau scalarea) lui $\mathbf{w}^{(k)}$, la fiecare pas *k*. Normele utilizate sunt norma-∞ şi norma-2.

Astfel, algoritmul metodei este:

$\mathbf{w}^{(0)}$ = Vector iniţial

$\mathbf{z}^{(k)} = \dfrac{\mathbf{w}^{(k)}}{\| \mathbf{w}^{(k)} \|}$, pentru $k \geq 0$ … Normalizare

$\mathbf{w}^{(k+1)} = \mathbf{A}\mathbf{z}^{(k)}$, pentru $k \geq 0$ … Iteraţie

Teste de oprire a iteraţiei:

1. Test de coliniaritate a doi vectori succesivi:

   Vectorii $\mathbf{z}^{(k)}$ şi $\mathbf{z}^{(k+1)}$ sunt coliniari, dacă rapoartele $\rho_i = z_i^{(k+1)} / z_i^{(k)}$, $z_i^{(k)} \neq 0$ sunt egale (coordonatele nenule sunt proporţionale), sau $z_i^{(k+1)} = z_i^{(k)} = 0$. În calculaţia practică, punem condiţia $|\rho(i) - \rho(i_0)| \leq TOL$, dacă $|z_i^{(k)}| > TOL$; sau, să avem simultan, $|z_i^{(k)}| \leq TOL$ şi $|z_i^{(k+1)}| \leq TOL$. *TOL* este o toleranţă

specificată; i_0 este indicele unei coordonate fixate: este convenabil să luăm $i_0 = imax =$ indicele coordonatei de modul maxim din $\mathbf{z}^{(k+1)}$.

Se introduc atunci, factorii de coliniaritate prin vectorul $\mathbf{colin}(1:n)$, definit astfel:

$$colin(i) = \begin{cases} z^{(k+1)}(imax) / z^{(k)}(imax)) & \dots \text{daca } |z^{(k+1)}(i)| \le TOL \text{ si } |z^{(k)}(i)| \le TOL \\ z^{(k+1)}(i) / z^{(k)}(i)) & \dots \text{altfel} \end{cases}$$

$i = \overline{1,n}$

Testul este:

$$\| \mathbf{colin} - \mathbf{colin}(imax) \| \le TOL$$

Observaţie

Test pentru norma-2 (euclidiană):

Dacă, pentru normalizarea lui $\mathbf{w}^{(k)}$, se utilizează norma-2, vectorii $\mathbf{z}^{(k)}$ au norma-2 egală cu 1, şi testul de coliniaritate poate lua forma

$$\| \mathbf{z}^{(k+1)} - \mathbf{z}^{(k)} \|_2 \le TOL.$$

O problemă specială apare pentru **A** reală, dacă valoarea proprie dominantă este reală şi negativă, $\lambda_1 < 0$, şi anume: din

$\mathbf{w}^{(k)} = \mathbf{A}^{(k)}\mathbf{z}^{(0)} = (\lambda_1)^k [a_1\mathbf{x}^{(1)} + \mathbf{r}^{(k)}]$ şi $\mathbf{z}^{(k)} = \mathbf{w}^{(k)} / \| \mathbf{w}^{(k)} \|$, rezultă că vectorul $\mathbf{z}^{(k)}$ schimbă de semn de la pasul k la pasul $k+1$. (Valoarea proprie, dată de $\lambda_1^{(k+1)} = w_m^{(k+1)} / z_m^{(k)}$, nu este afectată.). Testul de coliniaritate trebuie să ţină cont de aceasta. Astfel, definim,

$$is = sign(1., \lambda_1^{(k+1)}), \qquad \mathbf{dz} = \mathbf{z}^{(k+1)} - is * \mathbf{z}^{(k)}$$

Testul corect este

$$\| \mathbf{dz} \|_2 \le TOL.$$

■

2. Test asupra lui λ:

Iteraţia se opreşte prin condiţia

$$|\lambda_1^{(k+1)} - \lambda_1^{(k)}| \le TOL$$

unde *TOL* este o toleranţă.

3. Test de satisfacere a relaţiei de definiţie:

$\| \mathbf{Az} - \lambda \mathbf{z} \| \leq TOL$

Practic, $\mathbf{z} = \mathbf{z}^{(k)}$, $\mathbf{Az}^{(k)} = \mathbf{w}^{(k+1)}$, şi $\lambda = \lambda^{(k+1)}$. Definind

$\mathbf{dif} = \mathbf{w}^{(k+1)} - \lambda^{(k+1)} \mathbf{z}^{(k)}$,

se pune testul

$\| \mathbf{dif} \| \leq TOL$

Observaţii:

- În cod, dacă valorile anterioare (*k*) sunt stocate în `z0` şi `lambda0`, iar valorile curente (*k*+1) în `z` şi `lambda`, definiţia lui `dif` devine

  ```
  dif = z - lambda*z0,
  ```

  luând `z` *înainte* de normalizare.

- Vectorul $\mathbf{r} = \mathbf{Az} - \lambda \mathbf{z}$ se zice *vectorul rezidual*. Astfel, testul se mai scrie $\| \mathbf{r} \| \leq TOL$.

Notă

Testul propriu pentru metodă este Testul 1, întrucât, în esenţă, metoda determină vectorul propriu nr. 1, şi apoi determină λ_1 din acesta. Cu Testul 1, se obţin vectori proprii mai precişi decât cu Testul 2 (la aceeaşi toleranţă) .

Testul 3 nu este specific metodei puterii, ci poate fi aplicat oricărei metode iterative pentru valori şi vectori proprii. Codurile din Lapack (Bai et al.(2000)), utilizează Testul 3, cu $TOL = \varepsilon_M \, | \lambda |$, unde ε_M este ε-maşină.

În metodele următoare, Testul 3 va fi utilizat numai la verificarea sistemului propriu. Excepţie face metoda puterii, unde, pentru studiu, se poate alege unul din Testele 1-3. (Alegerea testului se face printr-un cod.)

■

Convergenţa:

Convergenţa $\lambda_1^{(k)} \to \lambda_1$ este liniară, iar rata convergenţei este aproximativ $| \lambda_2 / \lambda_1 |$.

(Acest rezultat are loc în ipoteza metodei $|\lambda_1| > |\lambda_2|$, cu ipoteza suplimentară: pentru $k \geq k_1$, cantităţile $(\lambda_i / \lambda_2)^k$, $i \geq 3$, sunt neglijabile în raport cu $(\lambda_2 / \lambda_1)^k$.)

Fişiere:

Power-2.f90: subrutina metodei

Main-Power.f90: programul principal

Agen.f90: subrutina de generare a matricii `A(n,n)`.

Maxindex.f90: funcţie; indicele coordonatei de modul maxim din vectorul `v(n)`.

vNorm.f90: norma-∞ şi norma-2 a unui vector.

w0gen.f90: subrutina de generare a lui `w0`

work.f90: modul

width.f90: utilitar

Subrutina metodei este

Power(iter, kodTest, ikod, kodNorm, iter_flag, test_val)

Parametrii:

iter: Numărul maxim de iteraţii (intrare) / Numărul efectiv de iteraţii (ieşire)

kodTest: cod pentru testul de oprire a iteraţiei

ikod: cod de tipărire a iteraţiilor

kodNorm: codul normei

iter_flag: cod de încheiere a iteraţiei (intern).

test_val: valoarea de test.

Notă:

`TOL, lnit` şi `lambda` se transmit prin modulul work ■

Agen.f90: se scrie cod pentru generarea matricii `A(n,n)`.

w0gen.f90: se scrie cod pentru generarea vectorului de start `w0`.

Pentru a împlini cerinţa ca $\mathbf{w}^{(0)}$ să aibă o componentă în direcţia lui $\mathbf{x}^{(1)}$, unele coduri generează $\mathbf{w}^{(0)}$ ca vector aleator.

Datele de intrare se citesc dintr-un fişier de date (specificat). Datele de ieşire se scriu în continuarea datelor de intrare.

Datele de intrare sunt:

- *Titlu*: text de max. 80 caractere
- *n, kodA, kodw0*
- [*Matricea* **A**: dacă *kodA* = 0. Se scrie aşa cum e dată - pe linii.]
- [*Vectorul* **w0**: dacă *kodw0* = 0.]
- *TOL, lnit*: toleranţa, numărul limită d iteraţii.
- *kodTest, ikod*: cod test, cod tipărire iteraţii
- *kodNorm*: cod normă
- *Check_text*: cod de verificare.

Semnificaţia codurilor:

n: ordinul matricii

kodA: 0 / ≠ 0: citeşte **A** / generează **A** (cu subrutina `Agen`)

kodw0: 0 / ≠ 0: citeşte **w0** / generează **w0** (cu subrutina `w0gen`)

kodTest: ≤ 0: Test asupra lui λ (Testul 2);

1: Test de coliniaritate (Testul 1);

≥ 2: Test asupra lui $\| \mathbf{Az} - \lambda\mathbf{z} \|$ (Testul 3).

ikod: 0 / ≠0: Tipăreşte iteraţiile: Da / Nu.

kodNorm: ≤ 1: Norma-infinit; ≥ 2: Norma-2 (euclidiană)

Check_text: text de max. 10 caractere. Primul caracter este semnificativ, anume:

`C, c, P, p`: se face proba, adică se tipăreşte vectorul $\mathbf{Az} - \lambda\mathbf{z}$.

Exemplu:

```
Exemplu n = 3
```

```
3     0     0
1  2  3
2  3  4
3  4  5
3*1
1E-7   7
1     1
2
check

 Guess vector:
   1.000000       1.000000       1.000000

 Iteration      Current_vector
Current_lambda
      1  0.3713907      0.5570860      0.7427813       12.00000
      2  0.3859738      0.5596620      0.7333503       9.500000
      3  0.3850325      0.5595003      0.7339682       9.631579
      4  0.3850935      0.5595108      0.7339282       9.622951
      5  0.3850895      0.5595102      0.7339308       9.623509
      6  0.3850898      0.5595102      0.7339306       9.623473
      7  0.3850898      0.5595102      0.7339306       9.623475

 Iteration No. 7: Tolerance on "z" met.
 Last test_value is:   0.000000

 Lambda-1 (dominant eigenvalue):   9.623475

 Eigenvector #1:
 (1)  0.3850898
 (2)  0.5595102
 (3)  0.7339306

 Check: A*y -lambda*y
 -1.4137129E-08
  1.4959278E-07
  3.1332269E-07
```

```
Maximum difference (in modulus) = 3.133E-07
Coordinate # 3
```

■

Power_Complex

Metoda puterii – pentru o matrice complexă.

Metoda:

Este cea descrisă la Power.

Convergenţa:

Ipotezele sunt aceleaşi ca la Power. În particular, nu se obţine convergenţă, pentru o matrice reală la care $|\lambda_1| = |\lambda_2|$ – de exemplu, când λ_1, λ_2 sunt complex conjugate.

Fişiere:

Agen_complex.f90; Main-Power_complex.f90; Maxindex_z.f90;
Power_Complex&Eps.f90; vNorm.f90; width.f90; work_complex.f90;

Semnificaţia fişierelor şi rutinelor – aceleaşi ca la Power.

Subrutina metodei:

Power_Complex(iter, kodTest, ikod, kodNorm, iter_flag, test_val, test_val_min)

Semnificaţia parametrilor – ca la Power. Parametrul test_val_min returnează valoarea minimă de test, în cazul când nu este atinsă toleranţa *TOL*.

Datele de intrare – sunt datele la Power, cu deosebirile:

- **A** şi **w0**: sunt date complexe; pentru **A** şi **w0** reale, se introduc date reale.
- *ikod*: Pentru *ikod* = 2: se tipăresc şi valorile $eps1 = \lambda / |\lambda|$, şi $eps1 * z0$, unde *z*0 este *z_anterior*. (Aceste valori servesc la studiul convergenţei: avem $\mathbf{z}^{(k)} \approx eps1 \cdot \mathbf{z}^{(k-1)}$).

Exemplu:

```
Exemplu n = 3. Val. proprii: 15.38; -10.63; -0.75
3     0     0
```

```
(1,0) (1,-7)      (0,-1)
(1,7) (5,0)       (10,-3)
(0,1) (10,3)      (-2,0)
(1,0) (1,0) (1,0)
1.E-7  50
1     0
1
ch

 Iteration No. 44: Tolerance on "z" met.
 Current Test_value is:  1.4587842E-08

 Lambda-1 (dominant eigenvalue):   15.37663     -0.1718582E-06

 Eigenvector #1:
 (1)  0.1880912     -0.5012191
 (2)  0.9789736      0.2039873
 (3)  0.5570123      0.2972317

 Check: A*z -lambda*z
 (3.3876418E-08,-4.6645388E-07)
 (-6.6803612E-09,1.2591090E-07)
 (-8.2114104E-07,-1.4940717E-08)

 Maximum difference (in modulus) =-8.211E-07   -1.494E-08
 Coordinate # 3
```

■

Inverse_Power_Shift

Metoda puterii inverse cu translaţie (shift) – matrice reală, cu valori proprii reale.

Metoda:

Fie matricea **A**, cu valorile proprii λ_j, $j = \overline{1,n}$. Metoda găseşte valoarea proprie λ_i a lui **A**, cea mai apropiată de un număr dat *s*. Se consideră matricea

$$\mathbf{B} = \mathbf{A} - s\mathbf{I}$$

şi presupunem că **B** este nesingulară. Se verifică imediat că valorile proprii ale lui **B** sunt $\mu_j = \lambda_j - s$. Cantitatea *s* zice *deplasare* (shift).

Fie $\mu_i = \lambda_i - s$, valoarea proprie de modul minim a lui **B**, adică:

$0 < |\mu_i| < \varepsilon < |\mu_j|$, pentru $j \neq i$.

Atunci, metoda puterii aplicată lui $\mathbf{B}^{-1}$ produce valoarea proprie de modul maxim, adică $\nu_i = 1/\mu_i$. Avem $\mu_i = 1/\nu_i$, şi

$$\lambda_i = \frac{1}{\nu_i} + s .$$

Principala aplicaţie a metodei este de a găsi vectorul propriu, dacă este cunoscută o aproximaţie bună a valorii proprii, să zicem $\hat{\lambda}_i$. (Aceasta poate fi furnizată de o metodă în care se determină numai valorile proprii – nu şi vectorii proprii.)

Se aplică metoda puterii inverse, cu deplasarea $s = \hat{\lambda}_i$. Chiar dacă $\hat{\lambda}_i$ este apropiată de λ_i, matricea $\mathbf{B} = \mathbf{A} - \hat{\lambda}_i \mathbf{I}$ este încă nesingulară, şi se obţine o bună aproximaţie a vectorului propriu $\mathbf{x}^{(i)}$.

Metoda iteraţiei inverse cu deplasare, este una dintre cele mai precise metode pentru calculul vectorilor proprii.

Algoritmul practic, este următorul:
Iteraţia din metoda puterii, aplicată la matricea $\mathbf{B}^{-1}$, este $\mathbf{w}^{(k+1)} = \mathbf{B}^{-1}\mathbf{z}^{(k)}$. În loc de a inversa **B**, calculăm $\mathbf{w}^{(k+1)}$ din sistemul liniar $\mathbf{B}\mathbf{w}^{(k+1)} = \mathbf{z}^{(k)}$, prin descompunere **LU**. Factorizarea se face o singură dată, şi sistemul se rezolvă succesiv cu membrii drepţi $\mathbf{z}^{(k)}$, $k \geq 0$.

Notă: Pentru testare, Inverse_Power_Shift oferă şi varianta iterării directe cu $\mathbf{B}^{-1}$ ■

Fişiere:
LU_Shift.f90: subrutina metodei – algoritmul precedent.
Inverse_Shift.f90: subrutina metodei – iterare directă cu matricea $\mathbf{B}^{-1}$.
Main-InversePower_Shift.f90: programul principal

apvt-LU.f90; compute_a_ii.f90;; LUdecomp.f90; LUsolve.f90; swap_row.f90: semnificaţiile de la descompunerea LU.

Agen.f90; Maxindex.f90; w0gen.f90; width.f90; work.f90: aceleaşi semnificaţii ca la metoda puterii.
vNorm2.f90: norma euclidiană a unui vector `v(n)`.

Subrutinele metodei:

LU_Shift(iter, kodTest, kodPrint)

Inverse_Shift(iter, kodTest, kodPrint)

Parametrul *iter*: Numărul limită de iteraţii (intrare) / Numărul efectiv de iteraţii (ieşire). Pentru ceilalţi parametri, v. codurile de mai jos.

Datele de intrare se citesc dintr-un fişier de date (specificat). În acesta se scriu şi datele de ieşire.
Datele de intrare sunt:

- *Titlu*: text de max. 80 caractere
- *n, kodA, kodw0*
- [*Matricea* **A**: dacă *kodA* = 0. Se scrie aşa cum e dată - pe linii.]
- [*Vectorul* **w0**: dacă *kodw0* = 0]
- *ns*: Număr de aproximaţii *s,* ale valorilor proprii
- *Aproximaţiile s*: "*ns*" valori.
- *kod_met*: codul metodei
- *TOL, lnit*: toleranţa, numărul limită de iteraţii.
- *kodTest, kodPrint*: cod test, cod tipărire iteraţii şi vectori reziduali.
- *Check_text*: cod de verificare.

Semnificaţia codurilor:

n: ordinul matricii

kodA: 0 / ≠0: citeşte **A** / generează **A** (cu subrutina `Agen`)

kodw0: 0 / ≠0: citeşte **w0** / generează **w0** (cu subrutina `w0gen`)

kod_met: 0 / ≠0: iterare cu $\mathbf{B}^{-1}$ / descompunere LU.

kodTest: 0 / ≠0: Test asupra lui λ / Test de coliniaritate.

kodPrint: 0 / 1 / 2: Tipăreşte: Nimic / Vectorii reziduali $\mathbf{Az} - \lambda\mathbf{z}$ / Iteraţiile şi vectorii reziduali. Vectorii reziduali se tipăresc numai dacă este cerută proba – v. codul *Check_text*.

Check_text: text de max. 10 caractere. Primul caracter este semnificativ, anume:

`C, c, P, p`: Se face proba, adică se verifică $\mathbf{Az} - \lambda\mathbf{z} \approx \mathbf{0}$. Se scriu:

- coordonata de modul maxim din $\mathbf{Az} - \lambda\mathbf{z}$ (maximul pe toţi vectorii **z**);
- indicele vectorului, şi indicele coordonatei.

Exemplu:

Se consideră exemplul de la Power. Acesta este un caz special: matricea este singulară, şi deci există o valoare proprie egală cu 0. Luăm ca aproximaţii *s* valorile 9, -1, şi 0.01. Fişierul de intrare / ieşire este:

```
Exemplu n = 3
3     0     0
1  2  3
2  3  4
3  4  5

3*1.
1E-7   50
3
9     -1    0.1
1
1     0
ch

 Shift #1: test_val:  0.0000000E+00
 Iteration No. 7

 Eigenvalue:    9.623475
```

```
 Eigenvector:
 (1)   0.3850898
 (2)   0.5595102
 (3)   0.7339306


 Shift #2: test_val:  0.0000000E+00
 Iteration No. 8

 Eigenvalue: -0.6234753

 Eigenvector:
 (1)   0.8276709
 (2)   0.1424137
 (3) -0.5428436


 Shift #3: test_val:  0.0000000E+00
 Iteration No. 18

 Eigenvalue:  0.5871054E-07

 Eigenvector:
 (1) -0.4082482
 (2)  0.8164966
 (3) -0.4082483


 Check: A*z -lambda*z
 Maximum difference (in modulus) = 2.429E-07
 Eigenvector # 1;  Coordinate # 1.
```

Inverse_Power_Complex

Metoda puterii inverse cu translaţie (shift), pentru o matrice complexă.

În particular, o matrice reală care are (şi) valori complexe.

Metoda: cea de la Inverse_Power_Shift.

Fişiere:

LU_Shift_Complex.f90: subrutina metodei – algoritmul cu descompunerea **LU**.

Inverse_Shift_Complex.f90: subrutina metodei – iterare directă cu matricea $\mathbf{B}^{-1}$.

Main-InversePower_Complex.f90.f90: programul principal

Hermitian.f90: funcţie logică; adevărat – matrice hermitiană; fals – în caz contrar.

apvt-LU_z.f90; compute_a_ii_z.f90; LUdecomp_z.f90; LUsolve_z.f90;;
swap_row_z.f90: semnificaţiile de la descompunerea LU_Complex.

Agen.f90; Maxindex_z.f90; w0gen.f90; width.f90; work.f90: aceleaşi semnificaţii ca la metoda puterii.

vNorm2.f90: norma euclidiană a unui vector real, şi a unui vector complex, `v(n)`.

Subrutinele metodei:

LU_Shift_Complex(iter, kodTest, kodPrint, kodLU, test_val)

Inverse_Shift_Complex(iter, kodTest, kodPrint, kodLU, test_val)

Semnificaţiile parametrilor:

iter, kodTest, kodPrint: aceleaşi semnificaţii ca la Inverse_Power_Shift.

kodLU: cod descompunere LU; 0: o.k.; < 0: pivot < prag.

test_val: valoarea de test (comparată cu toleranţa); aceasta depinde de *kodTest*.

Fişierul de date:

Are aceeaşi structură ca fişierul de date de la Inverse_Power_Shift, cu particularitatea că: matricea **A**, vectorul **w0**, şi aproximaţiile *s* – sunt acum, valori complexe. Dacă **A** este reală, se pot introduce valori reale; pentru vectorul **w0** trebuie însă introduse valori complexe.

Notă: Aproximaţiile *s* trebuie să fie suficient de apropiate de valorile proprii.

Exemplu:

Exemplul de la Power_Complex. Matricea este hermitiană şi are valori proprii reale. În rezultate, partea complexă a valorilor proprii este de ordinul 10^{-7} …10^{-8}.

Fişierul de intrare / ieşire este:

```
Exemplu-2
3     0     0
(1,0) (1,-7)      (0,-1)
(1,7) (5,0) (10,-3)
(0,1) (10,3)      (-2,0)
3*(1,0)
3
(15.,0.)    (-10.,0.)   (-0.7,0.)
1e-7 10
1
1     0
ch

 Matrice hermitiana.

 Shift #1:
 Iteration No. 6

 Eigenvalue:   15.37663     -0.2939753E-07

 Eigenvector:
 (1)  0.1448912     -0.3861009
 (2)  0.7541264      0.1571365
 (3)  0.4290796      0.2289648

 Shift #2:
 Iteration No. 9

 Eigenvalue:  -10.63025      0.5430218E-07

 Eigenvector:
 (1) -0.2537639      0.1906199
 (2)  0.4263011      0.4526095
 (3) -0.3145403     -0.6432296

 Shift #3:
```

```
 Iteration No. 6

 Eigenvalue: -0.7463716      0.9294398E-08

 Eigenvector:
 (1)  0.6996646      0.4895611
 (2)  0.5348870E-01 -0.1309672
 (3)  0.3495674     -0.3585925

Check: A*z -lambda*z
 Maximum difference (in modulus) =-2.951E-07   -7.384E-07
 Eigenvector # 2;  Coordinate # 2.
```

■

Sim_Iter

Metoda iteraţiilor simultane – matrice hermitiană.

Metoda:

Aceasta este o extindere a metodei puterii pentru o matrice **A** *hermitiană* (în particular, reală şi simetrică). O astfel de matrice, are valori proprii reale. Presupunem, mai mult, că valorile proprii sunt de module distincte: $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$.

În loc de un vector de start $\mathbf{w}^{(0)}$, se utilizează o matrice de start , ale cărei coloane sunt vectorii de start: $\mathbf{W}^{(0)} = [\mathbf{w}^{(0,1)} \vdots \mathbf{w}^{(0,2)} \vdots \ldots \vdots \mathbf{w}^{(0,m)}]$.

Dacă matricea **A** este $n \times n$, $\mathbf{W}^{(0)}$ este $n \times m$, unde $m \le n$. Vectorii de start $\mathbf{w}^{(0,j)}$ trebuie să fie liniar independenţi. Metoda de bază rămâne înmulţirea la stânga a lui $\mathbf{W}^{(0)}$ cu matricea **A**, adică, $\mathbf{W}^{(k+1)} = \mathbf{A}\mathbf{W}^{(k)}, k \ge 0$. Înainte de fiecare etapă a iteraţiei, matricea curentă $\mathbf{W}$ este ortogonalizată prin procedeul Gram-Schmidt, astfel încât coloanele ei $\mathbf{w}^{(j)}$ devin vectori ortonormaţi (ortogonali, şi având norma euclidiană unitară). Astfel, vectorii $\mathbf{w}^{(j)}$ formează o bază ortonormată a sub-spaţiului *m*-dimensional al lui $\mathbf{R}^n$, sub-întins de vectorii iniţiali $\mathbf{w}^{(0,j)}$. În cursul iteraţiei, această bază se aliniază din ce în ce mai mult la baza vectorilor proprii ai lui **A**, *direcţia* $\mathbf{w}^{(j)} \rightarrow$ *direcţia* $\mathbf{x}^{(j)}$, $j \ge 1$. Valorile proprii se evaluează prin câtul Rayleigh. Iteraţia

se încheie când se atinge o toleranţă convenabilă, privitor la direcţiile a două baze succesive $\{\mathbf{w}^{(j)}\}$. Dacă matricea de start $\mathbf{W}^{(0)}$ are $m < n$ coloane, se obţin primii m vectori proprii. Pentru $m = n$, adică $\mathbf{W}^{(0)}$ este $n \times n$, se găsesc toţi vectorii proprii.

Algoritmul:

Se cere un număr $ne \leq n$ de vectori proprii. $\mathbf{W}^{(0)}$ şi $\mathbf{W}$ sunt matrici $n \times ne$.

1. Se defineşte matricea $\mathbf{W}^{(0)}$: dacă o aproximare iniţială a vectorilor proprii nu este cunoscută, se ia $\mathbf{W}^{(0)} = [\mathbf{e}^{(1)} \mid \mathbf{e}^{(2)} \mid \ldots \mid \mathbf{e}^{(ne)}]$, adică, $\mathbf{W}^{(0)}$ este formată din primele ne coloane ale matricii unitate $\mathbf{I}$. Pentru $ne = n$, $\mathbf{W}^{(0)} = \mathbf{I}$.
2. Se aplică Gram-Schmidt la $\mathbf{W}^{(0)}$ (cu excepţia cazului în care aceasta este deja ortonormată). Se atribuie $\mathbf{W} = \mathbf{W}^{(0)}$.
3. Iniţializare contor: iter = 0
4. Iteraţii: iter = iter +1;

   Atribuire: $\mathbf{W}^{(0)} = \mathbf{W}$ ($\mathbf{W}$ = matricea curentă; $\mathbf{W}^{(0)}$ = matricea anterioară.)
5. Se calculează $\mathbf{W} = \mathbf{A}\mathbf{W}^{(0)}$.
6. Se calculează λ_j, $j = \overline{1, ne}$, prin câtul Rayleigh:

   Fie $\mathbf{w}^{(j)}$ şi $\mathbf{w}^{(0,j)}$ a j-a coloană a lui $\mathbf{W}$ şi $\mathbf{W}^{(0)}$, respectiv; avem

   $\lambda_j = < \mathbf{w}^{(0,j)}, \mathbf{w}^{(j)} >$

   ($\mathbf{w}^{(j)} = \mathbf{A}\mathbf{w}^{(0,j)}$ – Pasul 5; şi $< \mathbf{w}^{(0,j)}, \mathbf{w}^{(0,j)} > = 1$ – Paşii 2 şi 4.)
7. Se aplică Gram-Schmidt la $\mathbf{W}$, astfel că $\mathbf{W}$ devine ortonormată.
8. Se verifică atingerea toleranţei *TOL*:

   Prin testul de coliniaritate – 1.3, 1: se defineşte **colin**(n) pentru fiecare vector $\mathbf{z} = \mathbf{W}(:, je)$, şi $test\_val = \max_{je=1,ne} \| \mathbf{colin} - \mathbf{colin}(imax) \|$.

   (Întrucât $\mathbf{z}$ sunt normalizaţi, testul se poate pune şi sub forma din 1, Observaţie.)

   - Dacă $\| test\_val \| \leq TOL$, ieşire din iteraţie.

- Altfel, GOTO 4.

Observaţii

- Se poate prescrie un număr limită de iteraţii *lnit*: atunci, se adaugă la Pasul 8 un test $iter \leq lnit$.
- Pasul 6 se poate realiza numai o singură dată, după ce s-a ieşit din iteraţie.

■

Observaţii

1. Valori proprii de acelaşi modul

Întrucât metoda iteraţiilor simultane este în esenţă metoda puterii, nu se obţine convergenţă pentru vectorii proprii, dacă există valori proprii de modul egal.

2. Matrici non-hermitiene

Dacă aplicăm algoritmul de mai sus unei matrici non-hermitiene, nu vom obţine convergenţă pentru vectorii proprii, cu excepţia vectorului propriu corespunzând valorii dominante λ_1 (pentru care, metoda revine la metoda puterii). Aceasta se întâmplă deoarece ipoteza esenţială a metodei este că vectorii proprii formează o bază ortogonală – şi procedeul Gram-Schmidt forţează ca, la fiecare pas *k*, vectorii $\mathbf{w}^{(k,j)}$ să fie ortogonali.

Totuşi, dacă valorile proprii sunt de module distincte, atunci vectorii proprii sunt liniar independenţi, şi se obţin aproximaţii foarte bune pentru valorile proprii. Vezi Exemplu-2. Explicaţia este că, procedeul Gram-Schmidt furnizează un set de vectori $\{\mathbf{w}^{(j)}\}$ liniar independenţi (ortonormaţi), iar valorile proprii sunt calculate cu câtul Rayleigh – care dă aproximaţii bune ale valorilor proprii chiar cu aproximaţii grosiere pentru vectorii proprii.

Valorile proprii găsite pot fi utilizate ulterior, în metoda puterii inverse cu translaţie, pentru a obţine vectorii proprii pentru valorile λ_i, $i \geq 2$.

■

Fişiere:

Sim_Iter.f90: subrutina metodei

Main-Sim_Iter.f90: programul principal

Gram_Schmidt_Complex.f90: ortogonalizare Gram-Schmidt

Herm&Sim.f90: 2 funcţii logice – pentru a recunoaşte o matrice hermitiană, respectiv complexă şi simetrică.

Maxindex_z.f90: funcţie; indicele coordonatei de modul maxim în vectorul complex `v(n)`.

Agen.f90: subrutina de generare a matricii `A(n,n)`.

W0gen.f90: subrutina de generare a matricii `W0(ne,n)`. Codul actual generează primele *ne* coloane ale matricii unitate.

vNorm.f90: norma unui vector complex `v(n)`

work.f90: modul

width.f90: utilitar

Subrutina metodei:

Sim_Iter(iter, kodTest, kodPrint, iter_flag, test_val)

Semnificaţiile parametrilor sunt aceleaşi ca la metoda puterii (kodPrint: ca şi ikod).

Datele de intrare se citesc dintr-un fişier de date (specificat). În acesta se scriu şi datele de ieşire. Datele de intrare sunt:

- *Titlu*: text de max. 80 caractere
- *n, ne, kodA, kodw0*
- [*Matricea* **A**: dacă *kodA* = 0. Se scrie aşa cum e dată - pe linii.]
- [*Vectorul* **w0**: dacă *kodw0* = 0]
- *TOL, lnit*: toleranţa, numărul limită de iteraţii.
- *kodTest, kodPrint*: cod test, cod tipărire iteraţii şi vectori reziduali.
- *Check_text*: cod de verificare.

Semnificaţia codurilor:

n: ordinul matricii **A**

ne: numărul de coloane ale matricii **W0** (numărul de vectori şi valori proprii de calculat)

kodA: ≤ 0 / $\neq 0$: citeşte **A** / generează **A** (cu subrutina `Agen`)

kodA < 0: matrice hermitiană – se introduce triunghiul superior (programul completează triunghiul inferior).

kodA = 0: matrice generală – se introduce întreaga matrice.

kodw0: 0 / ≠0: citeşte **w0** / generează **w0** (cu subrutina `w0gen`)

kodTest: ≤ 0 / ≥ 1: Test asupra lui λ / Test de coliniaritate.

kodPrint: 0 / 1 / 2: Tipăreşte: Nimic / Vectorii reziduali $\mathbf{Az} - \lambda\mathbf{z}$ / Iteraţiile şi vectorii reziduali. Vectorii reziduali se tipăresc numai dacă este cerută proba – v. codul *Check_text*.

Check_text: text de max. 10 caractere. Primul caracter este semnificativ, anume:

`C, c, P, p`: Se face proba, adică se verifică $\mathbf{Az} - \lambda\mathbf{z} \approx \mathbf{0}$. Se scriu:

- coordonata de modul maxim din $\mathbf{Az} - \lambda\mathbf{z}$ (maximul pe toţi vectorii **z**);
- indicele vectorului, şi indicele coordonatei.

Exemplu-1:

Matricea este hermitiană (se poate pune *kodA* < 0, şi introduce numai triunghiul superior). Ca variantă, s-a introdus întreaga matrice – luând *kodA* = 0.

```
Exemplu-1. Matrice hermitiana.
3     3     0     1
(1,0) (1,-7)      (0,-1)
(1,7) (5,0) (10,-3)
(0,1) (10,3)      (-2,0)

1e-7  50
1     0
ch

 Matrice hermitiana

 Iteration No. 46: Tolerance on "W" met.
 Test Value is: 3.542E-08

 Eigenvalues:
```

```
 1   15.37663     -0.2587173E-07
 2  -10.63025      0.2952792E-07
 3 -0.7463716     -0.9538131E-08

 Eigenvectors:
  1                              2
  0.4123923     -0.6783551E-08  -0.4174030E-01  0.3146264
  0.1178387      0.7612572       0.6216292      0.1285194E-01
 -0.6361333E-01  0.4821696      -0.6793670     -0.2261424

  3
  0.1378377      0.8427345
  0.1311176     -0.5311906E-01
  0.5007854     -0.5747215E-07


 Check: A*z -lambda*z
 Maximum difference (in modulus) =-7.843E-07  1.134E-08
 Eigenvector #1;  Coordinate # 1.
```

■

Exemplu-2:

Matrice non-hermitiană – ilustrarea Observaţiei 2, de mai sus.

Reproducem numai rezultatele pentru valorile proprii, şi verificarea pentru vectorul propriu nr. 1. Vectorii proprii 2-4 nu sunt determinaţi corect. Pentru a-i calcula, se va utiliza metoda iteraţiei inverse cu translaţie, luând ca aproximaţii valorile proprii calculate 2-4.

```
Exemplu (Westlake). Val. proprii: (1,5); (2,6); (3,7); (4,8)
4     0     0     1
(5.0,9.0), (5.0,5.0),  (-6.0,-6.0), (-7.0,-7.0)
(3.0,3.0), (6.0,10.0), (-5.0,-5.0), (-6.0,-6.0)
(2.0,2.0), (3.0,3.0),  (-1.0,3.0),  (-5.0,-5.0)
(1.0,1.0), (2.0,2.0),  (-3.0,-3.0), (0.0,4.0)

1e-6 200
```

```
1     1
ch

 Matrice generala

 Iteration No. 169: Tolerance on "W" met.
 Test Value is: 6.482E-07

 Eigenvalues:
 1   4.000000       8.000000
 2   2.999998       6.999999
 3   2.000001       6.000000
 4  0.9999999       5.000003

 Check: A*z -lambda*z

 Eigenvectors:
 1
 (3.9152843E-07,-6.0862516E-07)
 (2.4125961E-07,-9.3978620E-08)
 (3.2940940E-07,-2.9458778E-07)
 (1.3913922E-07,-1.3633637E-07)
```

■

Jacobi

Metoda Jacobi – matrice reală simetrică.

Metoda:

Metoda Jacobi este o metodă convenabilă pentru a găsi toate valorile proprii şi vectorii proprii ai unei matrici reale şi simetrice, de ordin moderat. Determinarea vectorilor proprii este opţională.

Fie **A** o matrice reală şi simetrică. Dacă **N** este o matrice nesingulară, atunci matricea $\overline{\mathbf{A}} = \mathbf{N}^{-1}\mathbf{A}\mathbf{N}$ este similară cu **A**, şi are aceleaşi valori proprii (bara nu notează acum conjugata). Vectorii proprii **x** ai lui **A**, sunt legaţi de vectorii proprii $\overline{\mathbf{x}}$ ai lui $\overline{\mathbf{A}}$, prin:

$\mathbf{x} = \mathbf{N}\bar{\mathbf{x}}$. (Vezi Problema de valori proprii – sumar, mai sus).

Să presupunem acum, că **N** este *unitară*, adică $\mathbf{N}^{-1} = \mathbf{N}^T$. Matricea $\overline{\mathbf{A}}$ devine

$$\overline{\mathbf{A}} = \mathbf{N}^T\mathbf{A}\mathbf{N}$$

Notaţi că $\overline{\mathbf{A}}$ este de asemenea simetrică.

Diagonalizarea lui **A**:

Să presupunem că **N** este aleasă astfel încât $\overline{\mathbf{A}}$ să devină *diagonală*:

$$\overline{\mathbf{A}} = \mathbf{N}^T\mathbf{A}\mathbf{N} = diag(\bar{a}_{ii})$$

Pentru matricea $\overline{\mathbf{A}}$, avem proprietăţile:

1) Valorile proprii ale lui $\overline{\mathbf{A}}$ sunt elementele diagonale: $\lambda_i = \bar{a}_{ii}$
2) Vectorii proprii ai lui $\overline{\mathbf{A}}$ sunt coloanele matricii unitate **I**: $\bar{\mathbf{x}}^{(i)} = \mathbf{e}^{(i)}$ (unde $e_j^{(i)} = \delta_{ij}$).

În consecinţă, rezultă pentru **A**:

- Valorile proprii ale lui **A** se găsesc pe diagonala matricii $\overline{\mathbf{A}}$.
- Vectorii proprii ai lui **A** sunt coloanele matricii **N**.

Diagonalizarea Jacobi:

Metoda Jacobi constă în transformări unitare (sau, ortogonale) aplicate succesiv lui **A**, până la obţinerea unei forme aproape diagonale. Anume, dacă $\mathbf{N}_i$ sunt matrici unitare, matricea **A** se transformă cum urmează:

$$\overline{\mathbf{A}}_1 = \mathbf{N}_1{}^T\mathbf{A}\mathbf{N}_1$$

$$\overline{\mathbf{A}}_2 = \mathbf{N}_2{}^T\overline{\mathbf{A}}_1\mathbf{N}_2 = (\mathbf{N}_2{}^T\mathbf{N}_1{}^T)\mathbf{A}(\mathbf{N}_1\mathbf{N}_2)$$

…

$$\overline{\mathbf{A}}_k = \mathbf{N}_k{}^T\overline{\mathbf{A}}_{k-1}\mathbf{N}_k = (\mathbf{N}_k{}^T\mathbf{N}_{k-1}{}^T\ldots\mathbf{N}_1{}^T)\mathbf{A}(\mathbf{N}_1\mathbf{N}_2\ldots\mathbf{N}_k)$$

Dacă matricea $\overline{\mathbf{A}}_k$ este *aproape* diagonală, adică, elementele non-diagonale sunt aproximativ zero, luăm

$$\overline{\mathbf{A}} \approx \overline{\mathbf{A}}_k = \mathbf{N}^T\mathbf{A}\mathbf{N}$$

unde

$$\mathbf{N} = \mathbf{N}_1\mathbf{N}_2 \ldots \mathbf{N}_k$$

Fiecare transformare $\mathbf{N}_i$ se alege astfel ca să elimine o pereche de elemente non-diagonale (să zicem a_{pq} şi a_{qp} – la pasul *i*). O astfel de matrice are structura:

$$\mathbf{N}_i = \begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \ddots & & & & & \\ & & & & \cos\alpha & & -\sin\alpha & & \ldots(p) \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & \sin\alpha & & \cos\alpha & & \ldots(q) \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix}$$
$$\qquad\qquad\qquad\qquad (p) \qquad\qquad (q)$$

Elementele nescrise sunt zero (cu excepţia diagonalei principale unde acestea sunt unu).

Transformările $\mathbf{N}_i$ se aplică succesiv, fiecare dintre ele eliminând elementul non-diagonal a_{pq}, de modul maxim. Principala proprietate a unei astfel de transformări este că produsul $\mathbf{N}_i^T\mathbf{A}\mathbf{N}_i$ modifică *numai* elementele lui **A** din liniile şi coloanele *p* şi *q*.

Unghiul α se alege astfel încât

$$\bar{a}_{pq} = 0$$

Noile elemente diagonale, sunt date de formulele:

$$r = \sqrt{4a_{pq}{}^2 + (a_{pp} - a_{qq})^2}\,; \qquad \sin\alpha = +\sqrt{\frac{1}{2} - \frac{a_{pp} - a_{qq}}{2r}}\,; \qquad \cos\alpha = \frac{a_{pq}}{r\sin\alpha}\,;$$

$$\bar{a}_{pp} = \tfrac{1}{2}(a_{pp} + a_{qq} + r)$$
$$\bar{a}_{qq} = \tfrac{1}{2}(a_{pp} + a_{qq} - r)$$

Pentru detalii, v. Cap. 5-II, 2.

Consideraţii de programare

Stocajul matricii:

Se lucrează cu triunghiul superior al lui **A**, astfel încât, după diagonalizarea lui **A**, elementul (1,1) conţine λ_1, etc. Triunghiul superior este stocat în vectorul `a(n*(n+1)/2)`, în ordinea coloanelor, adică:

$$\mathrm{a} = [a_{11} \mid a_{12} \quad a_{22} \mid a_{13} \quad a_{23} \quad a_{33} \mid \ldots \mid a_{1n} \quad a_{2n} \quad \ldots \quad a_{nn}]$$

Adresa elementului (*i, j*) este dată de următoarea funcţie:

$$Loca(i, j) = (j-1) * j/2 + i$$

În particular, elementul diagonal (*i, i*) are adresa $Loca(i,i) = (i+1) * i/2$. După ce s-a efectuat o transformare, matricea curentă $\overline{\mathbf{A}}$ este stocată în acelaşi vector `a`.

Strategia de eliminare:

Aceasta este *căutarea completă*, adică: se caută în toată matricea **A** (concret, în vectorul `a`), elementul non-diagonal de modul maxim. (Pentru alte strategii, v. Cap. 5-II, 2.)

Elementele non-diagonale se consideră zero, dacă sunt mai mici (în modul) decât o toleranţă *TOL*.

■

Fişiere:

Jacobi.f90: subrutina metodei

Main-Jacobi.f90: programul principal

Agen.f90: subrutina de generare a triunghiului superior al matricii `A(n,n)`; stocat pe coloane, în vectorul `ag(n*(n+1)/2)`.

Loca.f90: funcţie; adresa elementului `a(i,j)` în vectorul `ag`.

work.f90: modul

width.f90: utilitar

Subrutina metodei:

Jacobi(kodVP, kodI, iter, amax)

Parametri:

kodVP: v. mai jos

kodI: v. mai jos, *kodPrintJ*.

iter: Numărul limita de iteraţii (intrare) / Numărul efectiv de iteraţii (ieşire).

amax: Elementul non-diagonal de modul maxim, din **A**.

Fişier de date (intrare/ieşire):

- *Titlu*: text de max. 80 caractere
- *n, kodA, kodPrintA*
- [*Matricea* **A**: dacă *kodA* = 0. Se introduce triunghiul superior, pe linii.]
- *TOL, kodPrintJ*: toleranţa pentru elementele non-diagonale; cod de tipărire

  a iteraţiilor.
- *kodVP*: cod pentru calculul vectorilor proprii
- *Check_text*: cod de verificare.

Semnificaţia codurilor:

n: ordinul matricii **A**

kodA: 0 / ≠ 0: citeşte **A** / generează **A** (cu subrutina `Agen`).

kodA = 0: Se introduce triunghiul superior al matricii, scris pe linii. (Programul completează triunghiul inferior, pentru o matrice hermitiană.)

kodPrinA: 0 / ≠ 0: Nu / Da – Tipăreşte matricea **A** (triunghiul superior).

kodPrinA ≠ 0 se utilizează pentru a tipări o matrice generată.

kodPrinJ: 0 / ≠ 0: Nu / Da – Tipăreşte transformările succesive ale matricii **A**.

Check_text: Text de max. 10 caractere. Primul caracter este semnificativ, anume:

`C,c, P,p, F,f`: Se face proba, adică se verifică $\mathbf{Az} - \lambda\mathbf{z} \approx \mathbf{0}$. Se scriu:

- coordonata de modul maxim din $\mathbf{Az} - \lambda\mathbf{z}$ (maximul pe toţi vectorii **z**);
- indicele vectorului, şi indicele coordonatei.

Pentru `F` sau `f`: se scrie şi fiecare vector rezidual $\mathbf{Az} - \lambda\mathbf{z}$.

Notă:

Proba se face numai dacă se cere calculul vectorilor proprii (*kodVP* ≠ 0).

Exemplu:

```
Ex. Jennings - p. 260
4  0  0
1     -3    -2    1
      10    -3    6
            3     -2
                  1
1e-7  0
1
ch
```

```
 Iteration #18:
 Maximum off-diagonal:   0.000000

 Eigenvalues
 1)   14.32951
 2)   4.456960
 3) -0.3713751
 4)  -3.415091

 Eigenvectors' Matrix
  0.1219755       0.5795794      -0.4442457       0.5563881
 -0.8748031      -0.2025671       0.2595736E-02   0.4706312
  0.2856186      -0.6127338      -0.5624290       0.1803400
 -0.4274252       0.1705392      -0.5079207      -0.6839256


 Check: A*y -lambda*y

 Maximum difference (in modulus) =-7.133E-04
 Eigenvector #1;  Coordinate # 4.
```

■

Observaţie

Utilizând InversePower_Shift cu aproximaţiile iniţiale 14.3, 4.46, -0.371, -3.42 (valorile Jacobi cu 3 cifre semnificative corecte), $\mathbf{w}^{(0)} = (1,1,1,1)$, şi toleranţa 1E-7, se găsesc rezultatele:

- Aceleaşi valori proprii nr. 1 şi 3; valorile nr. 2 şi 4 diferă cu o unitate, în a 7-a cifră semnificativă.
- Verificarea este: `Maximum difference (in modulus) = 5.895E-07`

Se poate conchide că metoda Jacobi dă rezultate foarte bune pentru valorile proprii, şi rezultate mai puţin precise pentru vectorii proprii.

■

QR.

Algoritmul QR – matrice reală.

Metoda (sumar) .

QR este unul din cei mai utilizaţi algoritmi pentru a determina toate valorile proprii ale unei matrici.. În numele metodei, *Q* desemnează o matrice ortogonală, iar *R* o matrice superior-triunghiulară (*R* vine de la *right-triangular*).

Considerăm o *matrice reală* nesingulară **A**, simetrică sau nu.

Esenţa algoritmului *QR* rezidă în următoarea proprietate:

> Dacă **A** este factorizată în produsul $\mathbf{A} = \mathbf{QR}$ unde **Q** este nesingulară, atunci considerând produsul în ordine inversă, fie $\mathbf{A}' = \mathbf{RQ}$, această matrice are aceleaşi valori proprii ca şi **A**, fiind similară cu **A** ■
>
> Într-adevăr, avem $\mathbf{R} = \mathbf{Q}^{-1}\mathbf{A}$, şi $\mathbf{A}' = \mathbf{Q}^{-1}\mathbf{AQ}$.

Algoritmul *QR* realizează factorizări succesive ale şirului de matrici $\{\mathbf{A}_k\}$, unde $\mathbf{A}_1 = \mathbf{A}$, definite de:

$$\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1; \quad \mathbf{A}_2 = \mathbf{R}_1\mathbf{Q}_1;$$

$\mathbf{A}_2 = \mathbf{Q}_2\mathbf{R}_2$; $\mathbf{A}_3 = \mathbf{R}_2\mathbf{Q}_2$;

…

$\mathbf{A}_k = \mathbf{Q}_k\mathbf{R}_k$; $\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k$;

…

Matricile $\mathbf{A}_k$, şi $\mathbf{Q}_k, \mathbf{R}_k$, au următoarele proprietăţi:

1) Toate $\mathbf{A}_k$ au aceleaşi valori proprii ca şi **A**.
2) Dacă **A** este *simetrică*, sau *tridiagonală*, sau **A** are forma *Hessenberg superioară*, matricile $\mathbf{A}_k$ *vor păstra aceste forme*.
3) Fie $\overline{\mathbf{Q}}_k = \mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_k$, atunci (prin inducţie): $\mathbf{A}_{k+1} = \overline{\mathbf{Q}}_k{}^{-1}\mathbf{A}_1\overline{\mathbf{Q}}_k$.
4) Fie $\overline{\mathbf{R}}_k = \mathbf{R}_k \ldots \mathbf{R}_1$, atunci: $\overline{\mathbf{Q}}_k\overline{\mathbf{R}}_k = (\mathbf{A}_1)^k$

Algoritm:

Algoritmul parcurge următoarele etape:

I. Reducerea lui **A** la forma tridiagonală dacă **A** este simetrică, sau la forma Hessenberg dacă **A** este nesimetrică.

II. Reducerea lui **A** la forma triunghiulară, sau bloc- triunghiulară (v. mai jos).

III. Descompunerea **QR**.

Algoritmul *QR* propriu-zis, constă în Etapele II şi III.

a) Etapa I se realizează înainte de algoritmul *QR* pentru a aduce matricea **A** la o forma mai simplă, şi a reduce astfel efortul de calcul.
b) Etapa II se realizează prin premultiplicare cu matricea Householder $\mathbf{P}_k$, cum urmează:

$\mathbf{A}_1 = \mathbf{P}_1\mathbf{A}$

$\mathbf{A}_2 = \mathbf{P}_2\mathbf{A}_1 = \mathbf{P}_2\mathbf{P}_1\mathbf{A}$

…

$\mathbf{A}_{n-1} = \mathbf{P}_{n-1} \ldots \mathbf{P}_2\mathbf{P}_1\mathbf{A}$

Matricea $\mathbf{A}_{n-1}$ are forma triunghiulară şi este matricea **R**.

Într-adevăr: Să notăm $\mathbf{Q} = \mathbf{P}_1\mathbf{P}_2 \ldots \mathbf{P}_{n=1}$, avem $\mathbf{A}_{n-1} = \mathbf{Q}^T\mathbf{A}$. **Q** este o matrice ortogonală, deci avem $\mathbf{QA}_{n-1} = \mathbf{A}$. Cum **Q** este ortogonală şi $\mathbf{A}_{n-1}$ este superior triunghiulară, urmează că $\mathbf{R} = \mathbf{A}_{n-1}$.

■

Atunci, Etapele II şi III se realizează prin următorii paşi de iterare:

1) Pune $\mathbf{A}^{(1)} = \mathbf{A}$

- Triangularizează $\mathbf{A}^{(1)}$:

$$\mathbf{A}_1^{(1)} = \mathbf{P}_1^{(1)}\mathbf{A};\ \ \mathbf{A}_2^{(1)} = \mathbf{P}_2^{(1)}\mathbf{A}_1^{(1)};\ \ldots;\ \mathbf{A}_{n-1}^{(1)} = \mathbf{P}_{n-1}^{(1)}\mathbf{A}_{n-2}^{(1)}$$

- La sfârşitul pasului, avem:

$$\mathbf{R}^{(1)} = \mathbf{A}_{n-1}^{(1)};\qquad \mathbf{Q}^{(1)} = \mathbf{P}_1^{(1)}\mathbf{P}_2^{(1)}\ldots\mathbf{P}_{n-1}^{(1)}.$$

2) Calculează $\mathbf{A}^{(2)} = \mathbf{R}^{(1)}\mathbf{Q}^{(1)}$

- Triangularizează $\mathbf{A}^{(2)}$:

$$\mathbf{A}_1^{(2)} = \mathbf{P}_1^{(2)}\mathbf{A}^{(2)};\ \ \mathbf{A}_2^{(2)} = \mathbf{P}_2^{(2)}\mathbf{A}_1^{(2)};\ \ldots;\ \mathbf{A}_{n-1}^{(2)} = \mathbf{P}_{n-1}^{(2)}\mathbf{A}_{n-2}^{(2)}$$

- Pune:

$$\mathbf{R}^{(2)} = \mathbf{A}_{n-1}^{(2)};\qquad \mathbf{Q}^{(2)} = \mathbf{P}_1^{(2)}\mathbf{P}_2^{(2)}\ldots\mathbf{P}_{n-1}^{(2)}.$$

...

k) Calculează $\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)}\qquad (k \geq 2)$

- Triangularizează $\mathbf{A}^{(k)}$:

$$\mathbf{A}_1^{(k)} = \mathbf{P}_1^{(k)}\mathbf{A}^{(k)};\ \ \mathbf{A}_2^{(k)} = \mathbf{P}_2^{(k)}\mathbf{A}_1^{(k)};\ \ldots;\ \mathbf{A}_{n-1}^{(k)} = \mathbf{P}_{n-1}^{(k)}\mathbf{A}_{n-2}^{(k)}$$

- Pune:

$$\mathbf{R}^{(k)} = \mathbf{A}_{n-1}^{(k)};\qquad \mathbf{Q}^{(k)} = \mathbf{P}_1^{(k)}\mathbf{P}_2^{(k)}\ldots\mathbf{P}_{n-1}^{(k)}$$

...

■

Calculaţia continuă până când elementele triunghiului sub-diagonal în $\mathbf{A}_{n-1}^{(k)}$ sunt aproximativ zero. Atunci, valorile proprii sunt pe diagonala lui $\mathbf{A}_{n-1}^{(k)}$. Mai precis:

Dacă **A** are valori proprii *de module distincte*, avem:

a) Dacă **A** este simetrică, atunci toate elementele non-diagonale vor fi aproximativ zero. Reamintim că o matrice simetrică are numai valori proprii reale.

b) Dacă **A** este nesimetrică şi are numai valori proprii reale, calculaţia se termină când toate elementele sub-diagonale sunt aproximativ zero.

Dacă **A** (nesimetrică) are o pereche de valori proprii complexe (conjugate), pe diagonală rămân submatrici blocuri 2×2, ale căror valori proprii sunt perechea de valori proprii conjugate ale lui **A**.

Convergenţa:

- Dacă **A** este o matrice reală (simetrică sau nu) şi are *valori proprii de module distincte*, atunci matricile $\mathbf{A}^{(k)}$ produse de algoritmul *QR* converg spre o matrice superior triunghiulară, pe diagonala căreia se găsesc valorile proprii ale lui **A**.

- Dacă **A** are o valoare proprie λ de multiplicitate *m*, atunci $\mathbf{A}^{(k)}$ încă converg la matrice superior triunghiulară, cu excepţia unui bloc diagonal de ordinul *m*, ale cărui valori proprii au modulul egal cu $|\lambda|$.

Algoritmul QR cu deplasare:

Presupunem că **A** are valori proprii *de module distincte* $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| > 0$.

Se arătă că viteza de convergenţă la zero a elementelor sub-diagonale, şi cea de convergenţă la valorile proprii a elementelor diagonale, depind de rapoartele $(\lambda_{i+1}/\lambda_i)^k$, $1 \le i \le n-1$.

Dacă două valori proprii λ_i şi λ_{i+1} sunt apropiate una de alta, convergenţa va fi înceată. Atunci, se utilizează următoarea tehnică pentru a accelera convergenţa. Se

aplică o deplasare s_k la valorile proprii (acestea devin $\lambda_i - s_k$), la fiecare etapă *k*. Adică, punem

$\mathbf{A}_1 = \mathbf{A}$, şi

$\mathbf{A}^{(k)} - s_k\mathbf{I} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$,

$\mathbf{A}^{(k+1)} = \mathbf{R}^{(k)}\mathbf{Q}^{(k)} + s_k\mathbf{I}$

Există două strategii pentru a alege deplasarea s_k (s_k est o aproximaţie a lui λ_n):

1) $s_k = a_{nn}^{(k)}$

   Pentru o *matrice simetrică* (reală), această strategie asigură o convergenţă cubică, chiar în prezenţa unor valori proprii multiple (Wilkinson, (1965)).

2) Se calculează valorile proprii ale submatricii 2×2 cea mai de jos din $\mathbf{A}^{(k)}$:

   $$\begin{bmatrix} a_{n-1,n-1}^{(k)} & a_{n-1,n}^{(k)} \\ a_{n,n-1}^{(k)} & a_{nn}^{(k)} \end{bmatrix}$$

   Dacă valorile proprii sunt reale, se alege:

   s_k = valoarea proprie care este cea mai apropiată de $a_{nn}^{(k)}$.

   Dacă valorile proprii sunt complexe, se ia:

   s_k = partea reală a valorii proprii.

3) Experimentele numerice au condus la o a treia strategie, şi anume:

   $s_k = \min(a_{ii}^{(k)}), \quad i = \overline{1,n}$

   Pentru unele matrici simetrice, această strategie se dovedeşte cea mai bună, conducând la un număr mai mic de iteraţii, şi o eroare mai mică în $\mathbf{Ay} - \lambda\mathbf{y}$.

Algoritm şi cod

Programul calculează sistemul propriu (valori şi vectori proprii), în succesiunea următoare:

I. Transformarea preliminară a matricii **A** (Opţional).

II. Valorile proprii, prin reducerea matricii la forma bloc-triunghiulară.

III. Vectorii proprii (Opţional).

Se calculează, mai întâi, vectorii proprii ai matricii bloc-triunghiulare. Aceştia se aduc, prin transformarea de similaritate, la vectorii proprii ai lui **A**.

IV. Rafinarea sistemului propriu prin iteraţie inversă (Opţional; numai în cazul când s-a ales opţiunea de calcul a vectorilor proprii).

V. Verificarea sistemului propriu (Opţional):

Precizia sistemului propriu se verifică prin listarea erorii maxime (în modul), în $\mathbf{Ay} - \lambda\mathbf{y}$ (unde λ este valoarea proprie, iar **y** vectorul propriu asociat cu λ).

Datele se citesc dintr-un fişier de date. Rezultatele sunt scrise în acest fişier, în continuarea datelor de intrare.

Codul este scris în dublă precizie.

Detalii de implementare

1. *Introducerea matricii*

Matricea **A** se defineşte în unul din următoarele moduri:

- Citită din fişier
- Generată (de subrutina `Agen`)

Citirea se poate face, conform unui cod, astfel:

- Matrice simetrică: triunghiul superior
- Matrice generală: completă, pe linii şi coloane
- Matrice generală: elementele nenule (indice linie, indice coloană, element).

Nu este implementată citirea sau stocarea de matrici bandă.

2. *Etapa I – Transformări preliminare*

Conform unui cod, matricea dată se poate aduce la una din formele:

- Tridiagonală (superioară): numai pentru o matrice simetrică
- Hessenberg (superioară): matrice generală
- Forma dată: nu se face nici o transformare preliminară

Codul `auto` comandă forma tridiagonală pentru o matrice simetrică, şi forma Hessenberg – pentru o matrice generală.

3. *Etapele II, III – Iteraţia QR*

Iteraţia QR se face cu deplasare, şi este implementată în conformitate cu algoritmul de mai sus. Sunt implementate cele trei strategii, pentru alegerea deplasării s_k; strategia se alege printr-un cod.

Se prescriu:

- Un număr limită de iteraţii, *lnit*
- Toleranţa *eps* pentru forma bloc-triunghiulară
- Toleranţa *eps_lambda* pentru valorile proprii (v. mai jos)

Iteraţia QR este oprită dacă una din condiţiile 1-4 de mai jos este îndeplinită:

1. S-a atins *lnit*.
2. S-a obţinut forma bloc-triunghiulară.
3. S-a atins toleranţa *eps_lambda* pentru doua seturi succesive de valori proprii.

Forma bloc-triunghiulară se consideră obţinută, dacă următoarele două teste sunt satisfăcute:

$$|Max\_Sub\_Sub| \le eps\_sub \qquad \text{(a)}$$

$$|Max\_Left\_Diag| \le eps\_left\ , \qquad \text{(b)}$$

unde:

Max_Sub_Sub este elementul sub-sub-diagonal de modul maxim, iar *jmax* este linia pe care se atinge maximul (elementele sub-sub-diagonale din coloana *j* sunt $a(l, j), l = j+2, n$; maximul se caută pentru $j = 1, n-2$);

$eps\_sub = eps\,|\,a(jmax, jmax)\,|$

Max_Left_Diag este elementul stâng-diagonal de modul maxim, iar *lmax* este linia pe care se atinge acest maxim (elementul stâng-diagonal de pe linia *l* este $a(l-1, l)$);

$eps\_left = eps(|\,a(lmax-1, lmax-1) + |\,a(lmax, lmax)\,|)$.

Notă: Factorii lui *eps* sunt tipăriţi de program.

Observaţii:

- Testul (a) este îndeplinit (după cel puţin o iteraţie) dacă, iniţial, matricea este redusă la una din formele preliminarii (tridiagonală sau Hessenberg).
- Testul (b) se pune pentru elementul stâng-diagonal *a primei linii* a blocului 2×2. (În cazul unui un bloc de dimensiune 1 – pentru linia respectivă.)
- Codurile pentru metoda QR – v. Bai et al. (2000), Golub & Van Loan (1996) – pun testul

  $| a_{i,\,i-1} | \le tol(| a_{ii} | + | a_{i-1,\,i-1} |)$,

  unde *tol* este o toleranţă mai mare decât eroarea de rotunjire a unităţii. Elementele sub-diagonale $a_{i,\,i-1}$ care satisfac acest test sunt setate la zero.

4. Iteraţia se mai opreşte dacă elementul stâng-diagonal de modul maxim este *staţionar* (nu se mai reduce).

Testul pus este *Max_Left_Diag = Prev_ Max_Left_Diag*, unde membrul doi este *Max_Left_Diag* de la iteraţia precedentă. Testul se pune după ce *Max_Left_Diag* s-a redus deja în proporţia | *Max_Left_Diag* | ≤ *eps_left*∗10.

■

Balansare

Eroarea în calculul valorilor proprii ale matricii **A**, rezultată prin aplicarea metodei *QR*, este de ordinul $\varepsilon_M ||\mathbf{A}||_F$, unde ε_M este *ε-maşină* (Cap. 1, 3.8.2), iar $||\mathbf{A}||_F$ este norma Frobenius (Cap. 4, 3) – v. Chen & Demmel, Balancing Matrices – în Bai et al. (2000), Parlett & Reinsch (1969). Balansarea este o transformare preliminară a matricii **A**, prin similaritate cu matricea diagonală reală **D**, matricea de lucru devenind $\mathbf{A}_b = \mathbf{D}^{-1}\mathbf{A}\mathbf{D}$. Scopul este reducerea normei Frobenius a matricii date.

O matrice $n \times n$ se zice *balansată în norma-p* dacă, pentru fiecare $i = \overline{1,n}$, norma-*p* a liniei *i* este egală cu norma-*p* a coloanei *i*:

$$\| \mathbf{A}(i, 1:n) \|_p = \| \mathbf{A}(1:n, i) \|_p, \quad i = \overline{1,n}$$

Algoritmii reduc norma-1 sau norma-2 a matricii **A**. Ţinând cont de echivalenţa normelor, rezultă că va fi redusă şi norma Frobenius a matricii **A**.

Balansarea se poate face fără a introduce erori de rotunjire suplimentare, prin aceea că elementele matricii **D** se aleg ca puteri întregi ale bazei utilizate în reprezentarea numerelor în calculator (baza 2).

Principalul algoritm este descris în lucrarea de referinţă Parlett & Reinsch (1969). Înainte de balansare, prin permutări de linii şi coloane, matricea se aduce la forma:

$$\begin{bmatrix} \mathbf{T}_1 & \mathbf{X} & \mathbf{Y} \\ \mathbf{0} & \mathbf{Z} & \mathbf{W} \\ \mathbf{0} & \mathbf{0} & \mathbf{T}_2 \end{bmatrix},$$

unde $\mathbf{T}_i$ desemnează o matrice superior triunghiulară, iar **0** o matrice cu elemente nule. Algoritmul se aplică, în esenţă, matricii **Z** (dar transformarea de similaritate se aplică întregii matrici, astfel că se modifică şi matricile **X**, **Y**, **W**).

Algoritmul balansează matricea în norma-1, după un număr finit de paşi. Acest algoritm stă la baza subrutinei sgbal din LAPACK (2000). Alţi algoritmi sunt descrişi în Bai et al. (2000).

Exemple

Considerăm următoarea matrice (Chen & Demmel, Balancing Matrices – în Bai et al. (2000)):

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 10^{-4} \\ 1 & 1 & 10^{-2} \\ 10^{4} & 10^{2} & 1 \end{bmatrix}$$

1. Balansarea *directă*, cu $\mathbf{D} = diag(0.01 \quad 1 \quad 100)$, produce matricea balansată

$$\mathbf{A}_b = \mathbf{D}^{-1}\mathbf{A}\mathbf{D} = \begin{bmatrix} 1 & 0 & 1 \\ 10^{-2} & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Normele sunt: $\| \mathbf{A} \|_F = 1.00005 \times 10^4$ şi $\| \mathbf{A}_b \|_F = 2.65$. Pe de altă parte, normele liniilor şi coloanelor în $\mathbf{A}_b$ sunt aproximativ egale.

Metoda *QR* aplicată lui **A**, fără rafinare (cu *EPS* = 1E-8), cu şi fără balansare, dă următoarele rezultate:

- Valorile şi vectorii proprii coincid, cu cel puţin 10, şi respectiv 9 cifre semnificative.
- Erorile maxime (în modul) în $\mathbf{Ay} - \lambda\mathbf{y}$ sunt: 3.229E-11 şi 2.887E-14, respectiv. Se observă precizia mult mai mare a sistemului propriu calculat cu balansarea matricii **A**.

■

2. Balansarea cu subrutina sgbal (în simplă precizie), dă următoarele rezultate:

$$\mathbf{D} = diag(2.4414062\text{E-}04 \quad 1.5625000\text{E-}02 \quad 1.000000)$$

$$\mathbf{A}_b = \begin{bmatrix} 1.000000 & 0.000000 & 0.40960 \\ 1.5625000\times10^{-2} & 1.000000 & 0.6400000 \\ 2.441406 & 1.562500 & 1.000000 \end{bmatrix}$$

Este de remarcat că, prin acest procedeu, matricea este balansată *aproximativ* în norma-1. Normele liniilor şi coloanelor sunt de acelaşi ordin de mărime (însă, diferite):

```
1     1.409600    3.457031
2     1.655625    2.562500
3     5.003906    2.049600
```

Eroarea maximă în $\mathbf{Ay} - \lambda\mathbf{y}$ este 4.960E-13.

■

3. Subrutina Balance din programul *QR*, balansează matricea în norma-1 (fără a face permutări), cum urmează. Notăm: *norm_r* norma liniei *i*, şi *norm*1_*r* norma liniei *i* fără elementul diagonal, adică: $norm1\_r = \sum_{\substack{j=1,n \\ j\neq i}} |a(i,j)|$; analog, pentru coloana *i*, *norm_c* şi *norm*1_*c*. Algoritmul este următorul:

- Iniţializăm $\mathbf{D} = \mathbf{I}$, şi iterăm paşii $K = 1, 2, \ldots$
- La fiecare pas *K*, procesăm liniile (şi coloanele) în ordinea naturală, anume:

  Pentru $i = 1, 2, \ldots, n$:

- Calculăm elementele matricei $\mathbf{D}_1$, prin $d1 = \sqrt{norm1\_r / norm1\_c}$
- Se scalează linia şi coloana *i*, prin $\mathbf{A} = \mathbf{D}_1^{-1}\mathbf{A}\mathbf{D}_1$
- Punem $\mathbf{D} = \mathbf{D} * \mathbf{D}_1$

La $i = n$ rezultă matricea **D**, la pasul curent *K*, şi s-a calculat $\mathbf{A} = \mathbf{D}^{-1}\mathbf{A}\mathbf{D}$.

- Calculăm (pentru matricea curentă **A**):

$Ratio = \max_{i=1,n} | norm\_r - norm\_c | / norm\_r$

- Se pun următoarele condiţii de oprire a iteraţiei:

$Ratio \le EPS$; $Numar\_Iteratii \le lnit$;

(Se iau: $EPS = 1E-4$ şi $lnit = 100$).

Pentru matricea din Exemplu, la pasul 2, se obţine: $Ratio = 1.2064\text{E-}05$;

$\mathbf{D} = diag(\ 9.9750279\text{E-}05 \quad 1.0074407\text{E-}02 \quad 1.002465\ )$.

Normele liniilor şi coloanelor sunt:

```
1     2.004975    2.004951
2     2.004963    2.004963
3     3.000013    3.000037
```

Eroarea maximă (în modul) în $\mathbf{A}\mathbf{y} - \lambda\mathbf{y}$ este 4.291E-14.

■

Fişiere

Main-QR-2007-2.f90: programul principal

Agen.f90: subrutina de generare a matricii **A**

Openfile2.f90, GetFile.f90: Selectarea fişierului de intrare (API: `GetOpenFileName`)

work-2005.f90: modul (date globale)

Lib_QR.lib: Biblioteca statică; conţine fişiere modul obiect (pentru aceste fişiere nu se dă codul sursă).

Programul QR este scris în dublă precizie.

Programul principal mai apelează următoarele subrutine (şi o funcţie):

unit2(n, Q)

Iniţializează matricea reală Q(*n, n*) cu matricea unitate.

n: ordinul matricii Q

Q: matrice (*n, n*)

symm(n, a)

Funcţie logică; matrice A simetrică.

n: ordinul matricii A

a: tablou (*n, n*) pentru A.

PrintA(n, A)

Tipărirea matricii A(*n, n*)

Balance(n, a, diag_bal, 0), Direct_Balance(n, a, diag_bal)

Balansarea matricii A, şi balansarea directă a matricii A.

n: ordinul matricii A

a: tablou (*n, n*) pentru A.

diag_bal: elementele diagonale ale matricii de balansare D (Balance: ieşire; Direct_Balance: intrare).

Tridiagonal(kod), Hessenberg(ikod)

Reducerea al forma tridiagonală, respectiv Hessenberg.

ikod: cod de tipărire a transformărilor. Se ia 0.

QR_Z(iter, do_shift, kodShift, kodIter, BlockReduced, real_val)

Iteraţia QR.

iter: contor de iteraţii

do_shift: logic; cu sau fără shift.

kodShift: codul shift-ului. V. Semnificaţia codurilor.

kodIter: cod de tipărire a iteraţiilor.

BlockReduced: logic. Matrice redusă la forma bloc-triunghiulară (ieşire).

real_val: logic. Toate valorile proprii sunt reale (ieşire).

Vec_BlocTri(VEC, kodNorm), Vec_BlocTri_Z(VEC_Z, kodNorm).
Calculează vectorii proprii ai unei matrici bloc-triughiulare – valoare proprie reală, respectiv complexă.
VEC, *VEC_Z*: tablou (*n*, *n*) pentru vectorii proprii; real, respectiv complex.

Power_Refine(ivp), Power_Refine_Z(ivp)
Rafinarea vectorului şi valorii proprii nr. "ivp" – valoare proprie reală, respectiv complexă.

Normalize_r(VEC, kodNorm), Normalize_Z(VEC_Z, kodNorm)
Normalizarea vectorilor proprii – cazul real, respectiv complex.
VEC, *VEC_Z*: ca mai sus.
kodNorm: codul normei. V. mai jos: Semnificaţia codurilor

Check(real_val, full_print_check)
Subrutină pentru verificarea sistemului propriu.
real_val: logic; valori proprii reale.
full_print_check: logic; tipărirea la verificare, şi a vectorilor $\mathbf{Az} - \lambda\mathbf{z}$.

Fişierul de date

Acestea se găsește în sub-folderul QR\Dat. La lansarea în execuţie, programul deschide fereastra standard de selecţie a fişierului din acest folder. Un fişier nou de date, este preferabil să fie plasat în acest folder.

■

Structura

1) *Titlu*: text, max. 80 caractere (titlul problemei).
2) *n, kodA, kodPrintA*

   Datele următoare definesc matricea **A**. Se introduce una din variante, în funcţie de codul *kodA*:

   - *Matricea* **A** ... kodA = 0, 1 (matrice citită din fişierul de date)
   - *Job* ... kodA < 0 (matrice generată).

- *Fname_*2: ... kodA ≥ 2 (matricea se citeşte din fişierul Fname_2).

3) *Balance, kod_Balance*: cod pentru balansarea matricii; cod pentru modul de balansare.
   - [*diags*(*i*), *i* =1,*n* ... Dacă kod_Balance ≠ 0.]
4) *Form*: cod pentru transformări preliminare ale matricii.
5) *Eps, Eps_lambda, Lnit, kodIter*: toleranţe; număr limită de iteraţii, cod tipărire iteraţii.
6) *Shift, kodShift*: cod pentru shift ; cod strategie de alegere a shift-ului.
7) *kodVP*: cod pentru calculul vectorilor proprii.
8) [*Refine*: rafinarea sistemului propriu prin iteraţie inversă (dată logică)...Dacă *kodVP* ≠ 0]
   - [*is* (întreg): cod pentru mod rafinare .... Dacă Refine are valoarea "adevărat":]
   - [*kodNorm*: cod normă pentru vectorii proprii ... dacă kodVP ≠ 0]
   - *Check_text*: cod pentru verificarea sistemului propriu.

Semnificaţia codurilor

n: ordinul matricii

kodA: kodA ≥ 0: matrice citită; kodA < 0: matrice generată (de `Agen`).

Valori kodA şi datele pentru definirea matricii **A**:

kodA = 0: *triunghiul superior*, pe linii (matrice simetrică) – în fişierul de intrare;

kodA = 1: *întreaga matrice*, pe linii – în fişierul de intrare;

kodA ≥ 2: *Fname_2*: specificatorul fişierului care conţine elementele matricii.

Structura fişierului *Fname_2*:

- Prima linie: text arbitrar (titlul problemei)

Liniile următoare, conform codului:

kodA = 2: triunghiul superior (matrice simetrică) – pe linii;

kodA = 3: întreaga matrice – pe linii;

kodA ≥4: elementele nenule ale matricii, anume:

- $i, j, a(i, j)$.
- Sfârşitul de fişier este o dată nenumerică. Exemplu: `End`

kodA < 0: *job*: Dată text (1-20 caractere). Numele jobului definit în `Agen`.

Parametru pentru apel `Agen`.

kodPrintA: > 0: tipăreşte **A** (dată, şi redusă la forma bloc-triunghiulară);

0: tipăreşte **A** redusă la forma bloc-triunghiulară;

< 0: nu ţipari.

Balance: Dată text (1-20 caractere). Primul caracter este semnificativ, anume:

`B, b / Alt caracter`: Balansează / Nu balansa.

kod_Balance: 0: Balansare cu subrutina `Balance`;

≠ 0: Balansare directă, cu `diags` (subrutina `Direct_Balance`).

Dacă nu se balansează: se introduce orice întreg:

diags$(1:n)$: Elementele diagonale pentru balansare directă ($\mathbf{A} \to \mathbf{D}^{-1}\mathbf{AD}$)

Form: Dată text (1-20 caractere). Primul caracter este semnificativ:

- `A, a`: Auto, rezultând în:
  Matrice simetrică: forma Tridiagonală;
  Matrice nesimetrică: forma Hessenberg (superioară).
- `T, t`: forma Tridiagonală;
- `H, h`: forma Hessenberg;
- `Alte caractere`: matricea dată (Fără transformări preliminare).

Eps: Toleranţa la elementele non-diagonale

Eps_lambda: Toleranţa la două valori succesive λ

Lnit: Numărul limită de iteraţii

kodIter: 0 / ≠0: Da / Nu – tipăreşte iteraţiile.

Shift: text (1-20 caractere). Primul caracter este semnificativ:

`S, s / alt caracter`: Cu / Fără shift;

kodShift: 0: $s_k = a_{nn}^{(k)}$ (s_k este shiftul de la pasul *k*)

> 0: $s_k =$ Valoarea proprie a sub-matricii $\mathbf{A}^{(k)}(n-1:n, n-1:n)$, cea mai apropiată de $a_{nn}^{(k)}$.

< 0: $s_k =$ Elementul diagonal $a_{ii}^{(k)}$ minim. (optim pentru unele matrici simetrice).

kodVP: 0 / ≠0: Da / Nu – calculează vectorii proprii.

kodVP > 0: Scrie vectorii proprii;

kodVP < 0: Nu scrie vectorii proprii.

Refine: Dată logică. Valori:

`T / F`: Rafinează / Nu rafina – sistemul propriu prin iteraţie inversă (cu translaţie).

Notă: Dacă se rafinează: vectorii poprii sunt deja normalizaţi cu norma-2.

is: Defineşte shift-ul pentru iteraţia inversă. Se introduce numai pentru `Refine = T`.

Shiftul este definit prin $s = \lambda(i) - ds$, unde:

is > 0: *ds* = o unitate în cea de-a "is-a" cifră semnificativă a lui `lambda`.

Recomandare: *is* = 4; 5, 6.

is < 0: *ds* = `1D-4`

kodNorm: < 0: Nu normaliza.

0: Norma-infinit

1: Norma-1

2: Norma euclidiană (Norma-2)

3: Norma "inginerească": prima coordonată = 1.

Check_text: dată text (1-20 caractere). Primul caracter este semnificativ:

`C,c, P,p, F,f / Alt caracter`: Da / Nu - Verificarea sistemului propriu.

`F,f`: se tipăresc şi vectorii reziduali $\mathbf{Az} - \lambda\mathbf{z}$.

(*do_balance, do_Shift, do_check, full_print_check*: date logice; utilizare internă)

Semnificaţia unor date de ieşire

Row: Indicele liniei matricii reduse.

Bloc Flag = 1 / 2 : Indicele liniei “Row” în bloc (Prima linie / A Doua linie).

Bloc Flag = 0: Linia nu aparţine unui bloc.

(Intern, se utilizează tabloul `ibloc(1:n)`: Indicele liniei blocului; la ieşire, data *Bloc Flag* este variabila `i_flag`. V. programul principal.).

Dacă se balansează (`do_Balance = .true.`):

$$Ratio = \max_{i=1,n} \frac{|Row\_Norm - Column\_Norm|}{Row\_Norm}$$

Dacă ordinul matricii $n > 50$, programul afişează şi timpul de calcul.

■

Exemple

Exemplu-1: Fără rafinare.

```
Ex Rau Cond.
3     1     0
3.02 -1.05 2.53
4.33 0.56 -1.78
-0.83 -0.54 1.47
NoBal 0
Hess
1E-8  1E-7  50    0
shift 0
1
F
2
Proba

 Matrix A - Hessenberg form
   3.020000       1.507519       2.287091
  -4.408832       1.021202       1.866028
 -2.2204460E-16  0.6260283       1.008798

 * Reached block-triangular form (Iteration 4).
   Maximum Left-Diagonal (in modulus) is:  7.6697928E-12 (Row:
   lmax = 3).
   |a(lmax-1,lmax-1)| +|a(lmax,lmax)|:  1.25924
   Maximum sub_Sub-Diagonal (in modulus) is:  1.4049635E-21
   (Column 1).

   Processing block-triangular form:
   Iterating to reach tolerance on lambdas.

 * Met tolerance EPS_lambda on lambdas (Iteration 5).
   Actual test value is:  0.000
```

```
Real Eigenvalues: F

Iteration 5:
Reduced Matrix - Block-Triangular Form
  2.670576      -4.177637      -1.992024
  1.505434       2.381351      -2.346518

 4.0960151E-12  6.4844721E-12 -1.9272281E-03

Row  Block Flag
1    1
2    2
3    0


Eigenvalues
1   2.525963614058517       2.503646148445962
2   2.525963614058517      -2.503646148445962
3 -1.9272281170354718E-03

Eigenvectors
      1
 0.3676642025709480      0.4354636756397324
 0.7512290651054949     -0.1817280223639355
-0.1909470078347702      0.2033801346581520

      2
-0.4135378963777110     -0.3921632146355579
 0.2247448848453599     -0.7394997762727376
-0.2140525790598959      0.1789014599505183

      3
 0.2557213286279253E-01
 0.9337601708377145
 0.3569846066400766

Check: A*y - lambda*y
```

```
 Maximum absolute difference (A*y -lambda*y) = 3.695E-12
 Vector #  Coordinate(s) #
 3          3
```

■

Exemplu-2: cu rafinare.

Reluăm Exemplu-1, cu rafinare, cu *is* = 5.

```
Ex. Rau Cond. - cu rafinare.
3     1     0
3.02 -1.05 2.53
4.33 0.56 -1.78
-0.83 -0.54 1.47
NoBal 0
Hess
1E-8  1E-7  50    0
Shift      0
1
T
5
2
Proba

 Matrix A - Hessenberg form
   3.020000       1.507519       2.287091
  -4.408832       1.021202       1.866028
 -2.2204460E-16  0.6260283       1.008798

 * Reached block-triangular form (Iteration 4).
   Maximum Left-Diagonal (in modulus) is:  7.6697928E-12 (Row:
   lmax = 3).
   |a(lmax-1,lmax-1)| +|a(lmax,lmax)|:  1.25924
   Maximum sub_Sub-Diagonal (in modulus) is:  1.4049635E-21
   (Column 1).

   Processing block-triangular form:
   Iterating to reach tolerance on lambdas.
```

```
* Met tolerance EPS_lambda on lambdas (Iteration 5).
  Actual test value is:  0.000

Real Eigenvalues: F

Iteration 5:
Reduced Matrix - Block-Triangular Form
  2.670576      -4.177637      -1.992024
  1.505434       2.381351      -2.346518

 4.0960151E-12  6.4844721E-12 -1.9272281E-03

Row  Block Flag
1    1
2    2
3    0



* Refinement:
    Iterations      Test Value           Tolerance
1   2               2.9288392389E-16     4.441E-16
2   2               2.0378132692E-16     4.441E-16
3   5                0.000000000         2.168E-19

Eigenvalues
1   2.525963614055325       2.503646148447407
2   2.525963614055325      -2.503646148447407
3 -1.9272281106504925E-03

Eigenvectors
      1
-0.3676642303666432     -0.4354636521714133
-0.7512290535059487      0.1817280703162026
 0.1909469948529727     -0.2033801468451698

      2
 0.4135378713456177      0.3921632410317126
```

```
 -0.2247449320489531      0.7394997619273490
  0.2140525904780458     -0.1789014462875638

       3
 -0.2557213286362887E-01
 -0.9337601708378584
 -0.3569846066396395

 Check: A*y - lambda*y
 Maximum absolute difference (A*y -lambda*y) = 4.441E-16
 Vector #  Coordinate(s) #
 1          2
 2          2
```

■

Exemplu-3: cu balansare (şi fără rafinare).

Considerăm din nou, Exemplu-1, cu balansarea matricii – cu *kod_Balance* = 0. Se observă că, verificarea sistemului propriu este mai bună decât în Exemplu-1.

```
Ex Rau Cond. - cu balansare
3     1     0
3.02 -1.05 2.53
4.33 0.56 -1.78
-0.83 -0.54 1.47
Bal   0
Hess
1E-8  1E-7  50    0
shift 0
1
F
2
Proba

 Balance:  No. of steps =   5;  Ratio = 4.0314E-05

A-balanced
   3.020000      -1.844351       1.792431
   2.465094      0.5600000     -0.7179390
```

```
 -1.171537      -1.338832       1.470000

Matrix A - Hessenberg form
  3.020000       2.435185      0.8272331
 -2.729320       1.525048      0.6917772
 2.2204460E-16   1.312671      0.5049516

* Reached block-triangular form (Iteration 4).
  Maximum Left-Diagonal (in modulus) is:  1.2783143E-13 (Row:
  lmax = 3).
  |a(lmax-1,lmax-1)| +|a(lmax,lmax)|:  2.08140
  Maximum sub_Sub-Diagonal (in modulus) is: -2.4004666E-22
  (Column 1).

  Processing block-triangular form:
  Iterating to reach tolerance on lambdas.

* Met tolerance EPS_lambda on lambdas (Iteration 5).
  Actual test value is: 2.4973E-16

Real Eigenvalues: F

Iteration 5:
Reduced Matrix - Block-Triangular Form
  2.409872      -3.042890      -1.287376
  2.064393       2.642056      1.6150857E-02

 7.8669816E-14  1.0075681E-13 -1.9272281E-03

Row  Block Flag
1    1
2    2
3    0



Eigenvalues
1   2.525963614055325       2.503646148447388
```

```
2   2.525963614055325      -2.503646148447388
3 -1.9272281106496357E-03

Eigenvectors
      1
 0.4517818761876123      0.3474172059670668
 0.6954752629118252     -0.3371710857265424
-0.1433823750645967      0.2393025981785100

      2
-0.3679706023954490     -0.4352047959596641
 0.3045951628967318     -0.7103463726325355
-0.2324043571570585      0.1543128437576227

      3
 0.2557213286362881E-01
 0.9337601708378584
 0.3569846066396397

Check: A*y - lambda*y
Maximum absolute difference (A*y -lambda*y) = 6.362E-14
Vector #  Coordinate(s) #
1          2
```

■

Exemplu-4:

Considerăm matricea Eberlein de ordinul 16 (Westlake (1968)). Aceasta se defineşte prin:

$$\mathbf{C}=\begin{bmatrix} -2 & 2 & 2 & 2 \\ -3 & 3 & 2 & 2 \\ -2 & 0 & 4 & 2 \\ -1 & 0 & 0 & 5 \end{bmatrix}; \qquad \mathbf{B}=\begin{bmatrix} 5\mathbf{C} & -\mathbf{C} \\ 5\mathbf{C} & \mathbf{C} \end{bmatrix};$$

$$\mathbf{A}=\begin{bmatrix} \mathbf{B} & 2\mathbf{B} \\ 4\mathbf{B} & 3\mathbf{B} \end{bmatrix}.$$

Matricea **A** are următoarele valori proprii:

$60\pm 20i,\quad 45\pm 15i,\quad 30\pm 10i,\quad 15\pm 5i,\quad -12\pm 4i,\quad -9\pm 3i,\quad -6\pm 2i,\quad -3\pm i.$

Codul în Agen – pentru generarea matricii **A** în tabloul `Ag(n,n)`, este dat mai jos. (În datele de intrare, se pune $n = 16$).

```
...
ELSEIF(job =='ebe16') THEN
allocate(c(4,4), b(8,8))
c(1,1) =-2; c(1,2:4) =2;
c(2,1) =-3; c(2,2) =3; c(2,3:4) =2;
c(3,1) =-2; c(3,2) =0; c(3,3) =4; c(3,4) =2;
c(4,1) =-1; c(4,2:3) =0; c(4,4) =5

b(1:4,1:4) =5*c; b(1:4,5:8) =-c;
b(5:8,1:4) =5*c; b(5:8,5:8) =c;

ag(1:8,1:8) =b; ag(1:8,9:16) =2*b;
ag(9:16,1:8) =4*b; ag(9:16,9:16) =3*b;

deallocate(c, b)
ENDIF
```

■

Pentru programul fără rafinare, se introduc datele

```
Eberlein - ord. 16
16  -1     1
ebe16
NoBal 0
auto
0.01  0.01 100   0
Shift      0
1
F
2
ch
```

Forma bloc-triunghiulară este atinsă la iteraţia 21 (şi toleranţa la λ, la iteraţia 22).

Rezultatul verificării sistemului propriu este:

```
 Maximum absolute difference (A*y -lambda*y) = 5.572E-03
```

```
 Vector #  Coordinate(s) #
 5         15
```

■

Programul cu rafinare, cu *is* = 5, produce următoarele rezultate (pentru economie de spaţiu, vectorii proprii nu sunt listaţi):

```
Eigenvalues
 1   30.00000000000003       10.00000000000001
 2   30.00000000000003      -10.00000000000001
 3   15.00000000000001        4.999999999999977
 4   15.00000000000001       -4.999999999999977
 5   60.00000000000001       20.00000000000001
 6   60.00000000000001      -20.00000000000001
 7   44.99999999999998       15.00000000000002
 8   44.99999999999998      -15.00000000000002
 9  -2.999999999999998        1.000000000000001
 10  -2.999999999999998       -1.000000000000001
 11  -6.000000000000003        2.000000000000008
 12  -6.000000000000003       -2.000000000000008
 13  -12.00000000000000        4.000000000000002
 14  -12.00000000000000       -4.000000000000002
 15  -9.000000000000000        3.000000000000027
 16  -9.000000000000000       -3.000000000000027

 Check result:
 Maximum absolute difference (A*y -lambda*y) =9.769963E-15
 Vector #  Coordinate(s) #
 8          1
```

■

Observaţii:

Toleranţele *Eps* şi *Eps_lambda* s-au luat relativ mici (0.01), producând rezultate destul de imprecise în cazul fără rafinare: eroarea în verificarea sistemului propriu, este de ordinul *E*-3.

Acestea sunt însă, îmbunătăţite foarte mult, prin rafinare (eroare de ordinul *E*-15).

Toleranţe mai mici, nu îmbunătăţesc verificarea sistemului propriu. Exemplu: Toleranţe de 0.001 duc la o eroare de ordinul *E*-14 în verificare.

Acest exemplu dă o idee, despre dificultăţile care apar în calculul sistemului propriu al unei matrici de ordin relativ mare; aceste dificultăţi cresc odată cu ordinul matricii. Pentru o matrice de ordin mare, nu se vor alege toleranţe *Eps* şi *Eps_lambda* foarte mici.

■

Exemplu-5:

Considerăm matricea Wilkinson de ordinul 20 (Wilkinson (1963), Westlake (1968)):

$$\mathbf{A} = \begin{bmatrix} 1 & 20 & & & & & \\ & 2 & 20 & & & & \\ & & 3 & 20 & & & \\ & & & \ddots & & & \\ & & & & 19 & 20 & \\ \varepsilon & & & & & 20 \end{bmatrix}$$

Analitic, matricea $n \times n$, este definită prin $a(i, j) = 0$, cu excepţia elementelor:

$a(i,i) = i; \qquad i = \overline{1,n}$

$a(i,i+1) = n; \; i = \overline{1,n-1};$

$a(n,1) = \varepsilon$.

Exemplul Wilkinson ia $n = 20$.

Dacă perturbaţia $\varepsilon = 0$, matricea este superior triunghiulară, şi valorile proprii sunt $\lambda(i) = i$.

Pentru $\varepsilon = 1E-10$, cu metoda QR *fără rafinare* (cu *EPS* = 1E-6), se obţin următoarele rezultate:

a) *Fără balansare*: Valorile proprii rezultă toate reale. Ele reprezintă perturbaţii ale valorilor *i*, $i = \overline{20,1}$, anume (ordonate descrescător):

```
19.99999998462896, 19.00001298980951, 17.99999341364806,
..., 3.000002187998112, 1.999999854534825, 1.000000000000000
```

Eroarea în verificarea sistemului propriu este `1.232E-07`.

b) *Cu balansare*: Valorile proprii calculate conţin 14 valori complexe, şi anume (ordonate după *i* descrescător):

```
20   20.00424815126970
19   18.89075561522547
18   18.42511929989719
17   17.03466897322695       1.087736309816453
16   17.03466897322695      -1.087736309816453
15   15.10602239289448       1.948529250030004
14   15.10602239289448      -1.948529250030004
13   12.88192664561770       2.529181675892914
12   12.88192664561770      -2.529181675892914
11   10.49999998763635       2.733397362516592
10   10.49999998763635      -2.733397362516592
 9   8.118073427255256       2.529181726294424
 8   8.118073427255256      -2.529181726294424
 7   5.893977535006919       1.948529282139898
 6   5.893977535006919      -1.948529282139898
 5   3.965330675018651       1.087735665091806
 4   3.965330675018651      -1.087735665091806
 3   2.574881415008950
 2   2.109241835064037
 1   0.9957544102238627
```

Eroarea în verificarea sistemului propriu este `4.211E-06`.

(Cu balansare şi rafinare, eroarea devine `1.776E-15`.)

■

Explicaţia rezultatelor foarte diferite în cazurile (a) şi (b), este următoarea: polinomul caracteristic al matricii este $p(\lambda) = (\lambda - 1)(\lambda - 2)\ldots(\lambda - 20) - 20^{19}\varepsilon$, şi acesta este

foarte rău condiţionat; numerele de condiţie pentru rădăcini sunt cuprinse între 4.31×10^{7} şi 3.98×10^{12}. Odată cu polinomul, şi valorile proprii sunt rău condiţionate.

(Problema condiţionării valorilor proprii depăşeşte cadrul acestui manual. V. de exemplu, Golub & Van Loan (1996)).

Din acest exemplu, se poate conchide că simpla verificare a sistemului propriu (prin proba că $\| \mathbf{Az} - \lambda\mathbf{z} \|$ = "mică"), nu asigură corectitudinea sistemului propriu calculat – v. cazul (a).
Pe de altă parte, rezultă că balansarea matricii este necesară în acest caz.

În general:
Balansarea se recomandă pentru metoda *QR* aplicată la *matrici nesimetrice,* la care valorile proprii pot fi rău-condiţionate. Matricile simetrice au valori proprii bine-condiţionate.

■

QR-IMSL.

Valorile și vectorii proprii ai unei matrici A (formă: generală sau Hessenberg)
Metoda QR, implementată de IMSL 6.0.
Versiunea dublă precizie.

Subrutina metodei
devcrh(n, a, n, eval, evec, n): matrice Hessenberg
devcrg(n, a, n, eval, evec, n): matrice generală
V. detalii în documentația IMSL.

Fișiere
Main-QR_IMSL.f90: program principal
Agen.f90: subrutina de generare a matricii **A**
Gen_Eigen_Files.f90: subrutină pentru generarea fișierelor de valori și vectori proprii
OpenFile-new.F90: Selectarea fişierului de intrare (API: `GetOpenFileName`)

symm-2.f90: funcție (logică); verificare dacă matricea este simetrică

width.f90: utilitar

imsl_dll.lib: Bibliotecă IMSL (pentru subrutinele metodei)

Fisierul de intrare:

1) *Titlul problemei*

2) *n, kodA, kodPrintA*

3) *Tip matrice*

4) [*job* - pentru *kodA* <=0]

5) [*Matricea* A - pentru *kodA* = 0,1]

6) *kod_write_VP*

7) *Proba*

8) *Do_Gen_Files* (logic)

Semnificatia datelor

1) *Titlul*: text ≤ 80 caractere

2) *n*: ordinul matricii

kodA: cod de introducere a matricii A:

kodA < 0: matrice generată (de Agen)

kodA ≥ 0: matrice citită din fișiser:

kodA = 0,1: citită din fișierul de intrare:

- *kodA* = 0: triunghiul superior;
- *kodA* = 1: intreaga matrice

kodA ≥ 2: matrice citită din fișier specificat = "`Fname_2`":

kodA = 2: triunghiul superior

kodA = 3: intreaga matrice

kodA ≥ 4: elementele non-zero ale matricii: $i, j, a(i, j)$

Nota: Structura "`Fname_2`":

- Prima linie este o linie text (max. 80 caractere; ex.: titlul problemei)
- Elementele matricii - cf. *kodA*.

3) *Tip*: text, primul caracter este semnificativ

"H", "h": Hessenberg

"S", "s": Simetrică

alt caracter: Generală

Notă: Tipul servește numai pentru apelul subrutinei care rezolvă problema de valori proprii.

În programul actual din ANA (QR-IMSL), tipul ”simetrică” este rezolvat cu subrutina pentru tipul ”generală”. ■

4) *Job*: text ≤ 20 caractere: identifica secventa de generare din Agen
5) *Matricea* A:
kodA = 0: triunghiul superior, pe linii
kodA = 1: matricea A completă, pe linii
6*) kod_write_VP*:
> 0 / ≤ 0: Scrie / Nu scrie - vectorii proprii în fișierul de date (ca ieşire)
7) *Proba*: text, primul caracter este semnificativ
"C","c","P","p", "F", "f"/Alt caracter: Cu/Fara proba sistemului propriu;
"F" or "f": Tipărire diferențe (A*z -lambda*z), pentru fiecare pereche (valoare proprie-vector propriu)

c) *Do_Gen_Files*: T/F :
Genereaza sau nu: Fisierul de valori proprii şi Fisierul de vectori proprii.

Exemplu-1: matrice generală

```
Ex. IMSL (single): (2,+/-4) (1,0)
3     1     1
gen
8     -1    -5
-4    4     -2
18    -5    -7
0
Proba
T


 Eigenvalues:
 1     2.000000000000002       4.000000000000000
 2     2.000000000000002      -4.000000000000000
 3    0.9999999999999966

 Proba - Complex:
 A*z -lambda*z
 Check result:
 Maximum absolute difference (A*y -lambda*y) = 3.442E-15
 Eigenvector #3; Coordinate #2.
```

Exemplu-2: matrice simetrică

```
Exemplul IMSL
3     0     1
sim
7.0   -8.0  -8.0
            -16.0 -18.0
                   13.0
0
Proba

 * All Real

Matrix A (Symmetric):
   7.000000      -8.000000      -8.000000
  -8.000000      -16.00000      -18.00000
  -8.000000      -18.00000       13.00000

 * All Real
```

```
 Eigenvalues:
(1)  -27.90246106049339
(2)   22.68061222210748
(3)   9.221848838385910

 Proba - Real:
 A*x -lambda*x:

 Check result:
 Maximum absolute difference (A*y -lambda*y) = 1.421E-14
 Eigenvector #1; Coordinate #2.
```

Exemplu-3: matrice Hessenberg

```
Ex. IMSL Hessenberg
4     1     1
Hess
-1. -1. -1. -1.
1.    0.    0.    0.
0.    1.    0.    0.
0.    0.    1.    0.
0
Proba
F


 Eigenvalues:
 1    0.3090169943749477      0.9510565162951544
 2    0.3090169943749477     -0.9510565162951544
 3   -0.8090169943749477      0.5877852522924727
 4   -0.8090169943749477     -0.5877852522924727

 Proba - Complex:
 A*z -lambda*z
 Check result:
 Maximum absolute difference (A*y -lambda*y) = 8.327E-16
 Eigenvector #1; Coordinate #4.
```

■

General_R, General_R1

Problema generalizată – reducerea la problema standard.

Problema generalizată de valori proprii

Problema determinării valorilor şi vectorilor proprii ai unei matrici **A**, definită de $\mathbf{Ax} = \lambda\mathbf{x}$, sau de ecuaţia

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \qquad (1)$$

se va numi problema *standard*.

În Dinamica Structurilor apare următoarea problemă:

$$(\mathbf{K} - \lambda\mathbf{M})\mathbf{x} = \mathbf{0}, \qquad (2)$$

unde: $\lambda = \omega^2$, iar **K** şi **M** sunt matrici *simetrice şi pozitiv definite* (**K** şi **M** sunt respectiv, matricile de rigiditate şi de masă; ω este pulsaţia proprie).

Problema (2) se zice problema *generalizată* de valori proprii. Mai general considerând două matrici $n \times n$ **A** şi **B**, problema generalizată are forma $(\mathbf{A} - \lambda\mathbf{B})\mathbf{x} = \mathbf{0}$.

În continuare, considerăm problema (2).

Reducerea la problema standard

Problema (2) poate fi adusă la problema standard (1) pentru o *matrice simetrică şi pozitiv definită*, cum urmează.

1) Se face descompunerea Cholesky a lui **M**:

   $$\mathbf{M} = \mathbf{S}^T\mathbf{S}, \qquad (3)$$

   unde **S** este superior triunghiulară.. Atunci, (2) scrisă în forma $\mathbf{Kx} = \lambda\mathbf{Mx}$,

   devine $\mathbf{Kx} = \lambda\mathbf{S}^T\mathbf{Sx}$, şi se scrie din nou ca şi

   $$\mathbf{KS}^{-1}(\mathbf{Sx}) = \lambda\mathbf{S}^T(\mathbf{Sx}). \qquad (4)$$

2) Se notează

   $$\mathbf{y} = \mathbf{Sx} \qquad (5)$$

   şi ecuaţia (4) devine

   $$\mathbf{KS}^{-1}\mathbf{y} = \lambda\mathbf{S}^T\mathbf{y}.$$

   Premultiplicând cu $(\mathbf{S}^T)^{-1} = (\mathbf{S}^{-1})^T = \mathbf{S}^{-T}$, se obţine

   $$\mathbf{S}^{-T}\mathbf{KS}^{-1}\mathbf{y} = \lambda\mathbf{Iy}$$

3) Acum, definim

   $$\mathbf{R} = \mathbf{S}^{-T}\mathbf{KS}^{-1} \qquad (5)$$

   R este matrice *simetrică şi pozitiv definită*.

4) Problema (2) este pusă în forma standard pentru matricea **R**, şi anume,

   $$(\mathbf{R} - \lambda\mathbf{I})\mathbf{y} = \mathbf{0} \qquad (6)$$

Problemele (6) şi (2) au aceleaşi valori proprii.

5) După rezolvarea lui (6) pentru λ şi **y**, vectorii proprii ai problemei originale (2) sunt daţi, cf. (4), de

$$\mathbf{x} = \mathbf{S}^{-1}\mathbf{y} \tag{7}$$

■

Ţinând cont de faptul că **M** şi **K** sunt simetrice şi pozitiv definite, se verifică imediat că matricea **R** este simetrică şi pozitiv definită. Urmează că valorile proprii ale lui **R** sunt reale şi pozitive, aşa cum trebuie să avem conform cu $\lambda_i = \omega_i^2$.

Astfel, pentru a rezolva (6), orice metodă pentru problema de valori proprii ale unei matrici simetrice şi pozitiv definite poate fi aplicată lui **R**.

Observaţie

Cel mai simplu caz este acela în care matricea **M** este diagonală (model de mase concentrate), $\mathbf{M} = diag(m_i)$. Atunci, avem $\mathbf{S} = \mathbf{S}^T = diag(\sqrt{m_i})$, şi $\mathbf{S}^{-1} = \mathbf{S}^{-T} = diag(1/\sqrt{m_i})$. Astfel, elementele lui **R** şi **x** sunt date de**:**

$$r_{ij} = \frac{k_{ij}}{\sqrt{m_i}\sqrt{m_j}}, \quad i, j = \overline{1,n},$$

şi

$$x_i = \frac{y_i}{\sqrt{m_i}}, \quad i = \overline{1,n}.$$

■

Observaţie

Matricea **R** poate fi scalată, adică se pune $\mathbf{R} = \mathbf{R}' * factor$. Ecuaţia (6) devine

$(\mathbf{R}' - \lambda'\mathbf{I})\mathbf{y} = \mathbf{0}$, unde

$$\lambda' = \frac{\lambda}{factor}$$

Astfel, după determinarea valorilor proprii λ' avem $\lambda = \lambda' * factor$, iar pulsaţiile proprii se găsesc din:

$\omega = \sqrt{\lambda' * factor}$

■

Foldere:

General_R, General_R1

Ambele proiecte reduc problema generalizată (2) la problema standard. Deosebirea este că primul stochează şi procesează triunghiul superior al matricilor **M** şi **K**, iar al doilea, matricile complete `M(ngl,ngl),K(ngl,ngl).`

Ambele proiecte lucrează în dublă precizie.

Fişiere (General_R):

Back.f90: substituţie înapoi, pentru Cholesky.

Cholesky_S.f90: subrutina Cholesky.

loca.f90: funcţia de adresă din Cholesky.

Gen_M.f90, Gen_K.f90: subrutinele de generare a matricii **M**, respectiv **K**.

Gen_R.f90: subrutina de calcul a matricii **R**.

Main-General_R.f90: programul principal.

Write_R.f90: subrutină pentru scrierea matricii **R**.

work.f90: modul
GetFile.f90, Openfile.f90, width.f90: utilitare

Compact_mul: subrutină de înmulţire **S**∗**A**, unde: **S** este simetrică şi stocată în triunghiul superior, iar **A** este (n, n).

[Aceasta este numai în folderul General_R.]

Subrutina metodei este

Gen_R(kodPrint)

Parametru:

kodPrint: cod pentru tiparire

Datele de intrare se citesc dintr-un fişier de date (acesta se selectează din fereastra standard). Datele de ieşire se scriu în continuarea datelor de intrare.
Matricile **M** şi **K** se introduc conform codurilor *kod_genM* şi *kod_genK* (coduri de citire sau generare a matricilor):

- *kod_gen* ≤ 0: matrici generate (de subrutinele Gen_M.f90, Gen_K.f90);
- *kod_gen* ≥ 1: matrici citite din fişier, anume:
    - *kod_gen* = 1: fişierul de intrare;
    - *kod_gen* ≥2: fişier specificat (selectat).

Daca matricea este citită: se introduce şi "factor"; dacă este generată, nu.

În cazul matricilor citite dintr-un fişier specificat (*kod_gen* ≥2), fişierul va avea următoarea structură:

- Prima linie din fişier este o linie-text (ignorată la citire);
- Următoarele linii conţin elementele triunghului superior al matricii.

Datele de intrare (aceleaşi pentru General_R şi General_R1), sunt:

- *Titlu*: text de max. 80 caractere
- *ngl, kod_genM, kod_genK*

  Pentru *kod_genM* = 1, se introduce:

  - [*Matricea* **M** – triunghiul superior, pe linii.]

  Pentru *kod_genM* ≥ 1, se introduce:

  - [*m_factor*: factor de multiplicare pentru **M**; (**M** → *m_factor* ∗ **M**)

  Pentru *kod_genK* = 1, se introduc:

  - [*Matricea* **K** – triunghiul superior, pe linii.]

  Pentru *kod_genK* ≥ 1, se introduce:

  - [*k_factor*: factor de multiplicare pentru **K**; (**K** → *k_factor* ∗ **K**)

- *kodPrint*: cod tipărire

Semnificaţia codurilor:

ngl: ordinul matricii (numărul gradelor de libertate)

kod_gen: v. mai sus

kodPrint: 0 / ≠0: Nu se tipăresc / Se tipăresc: matricile **M**, **K**, şi **S**.

Observaţie

Matricea **R** este *scalată*: adică, în fişierul de ieşire, se tipăreşte matricea scalată $\mathbf{R}'$ şi factorul de scalare `scal`: avem $\mathbf{R} = \mathbf{R}' * scal$.

Factorul de scalare se calculează astfel:

```
norm1 = maxval(dabs(R))
scal = 2_8**(exponent(norm1)-1)
```

unde `norm1` este norma-1.

■

Exemplu:

$$\mathbf{M} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}; \qquad \mathbf{K} = \begin{bmatrix} 3 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

Fişierul de date:

```
Exemplu cu M non-diagonala; ngl =3
3     1     1
2     1     0
3     1
2
1.    ! m_factor

3     -1    0
2     -1
1
1.    ! k_factor
0
```

Rezultate:

```
 Matricea R = S^(-T)*K*S^(-1)
  1.500000000000000   -1.118033988749895   0.5590169943749473
 -1.118033988749895    1.500000000000000  -1.250000000000000
 0.5590169943749473   -1.250000000000000   1.500000000000000
```

```
 Matricea S^(-1):
  0.707106781186547   -0.316227766016838   0.158113883008419
  0.632455532033676   -0.316227766016838
  0.790569415042095
```

■

Retrieve_Eigen_from_R

Regăseşte valorile şi vectorii proprii ai problemei generalizate, după găsirea valorilor şi vectorilor propriiai matricii **R**.

Fişiere sursă:

Main-Retrieve_Eigen_from_R.f90: programul principal
work-2005.90: modul
GetFile.90, OpenFile.90: fişiere pentru fereastra de selecţie standard
width.90: utilitar

Fişiere de date (intrare):

Programul cere ca in folderul de lucru să existe urmatoarele 4 fişiere:

nume-S^-1.dat: fişierul care conţine matricea $\mathbf{S}^{-1}$ (triunghiul superior);
nume-Scal_R.dat: fişierul care conţine factorul de scalare.
nume-VAL.dat: fişierul care conţine valorile proprii λ ale lui **R**;
nume-VEC.dat: fişierul care conţine vectorii proprii **y** ai lui **R**;

Specificatorul de *fişier trebuie să aibă forma de mai sus*, în care, ”*nume*” este ales de utilizator (de regulă, numele exemplului).
În fiecare fişier, datele numerice sunt precedate de o line-text.

- Primele două fişiere sunt furnizate (ca fişiere de ieşire) de programul General_R.
- Ultimele două fişiere sunt furnizate (ca fişiere de ieşire) de programul de valori proprii QR sau Jacobi.

Date introduse de la terminal:

- Numărul gradelor de libertate (ordinul matricii R)
- Pulsaţii sau Frecvenţe: se introduce T, respectiv F.

Relaţia între pulsaţia ω şi frecvenţa *f* este: $f = \omega/(2\pi)$.

Fişiere de ieşire (rezultate):

nume-OMEGA.dat (pulsaţii), sau *nume-FRECV.dat* (frecvenţe);

nume-VEC_X.dat: vectorii proprii **x**, ai problemei generalizate.

Exemplu (v. exemplul de la General_R)

Fişiere de intrare:

Ex3-S^-1.dat:

```
Matricea S^(-1):
  0.707106781186547  -0.316227766016838    0.158113883008419
  0.632455532033676  -0.316227766016838
  0.790569415042095
```

Ex3-Scal_R.dat:

```
Factor de scalare pentru R:
   1.00000000000000
```

Următoarele fişiere sunt generate de Jacobi_D:

Ex3-VAL.dat:

```
Eigenvalues:
 1   3.478563691072702
 2  0.9454183848254143
 3  7.6017924101882794E-02
```

Ex3-VEC.dat:

```
Eigenvectors
       1
 -0.5223878791313503
  0.6464566721245862
```

```
-0.5560066361130130

      2
-0.7548454734848392
-0.4735615410061037E-01
 0.6541440345456881

      3
 0.3965697446473088
 0.7614626757958836
 0.5127445986996710
```

Folder: **Interpolation**

INTERPOLARE

Newton

Polinomul de interpolare Newton.

Metoda:

Problema

Fie funcţia *f*, cunoscută prin valorile ei pe $n+1$ puncte distincte x_i, ca în tabelul următor:

x	x_0	x_1	x_2	…	x_n
$f(x)$	f_0	f_1	f_2	…	f_n

Punctele x_i se numesc *noduri* şi s-a notat $f_i = f(x_i), i = \overline{0,n}$. Se cere să se găsească un polinom p_n de grad cel mult *n*, care să coincidă cu funcţia *f* pe nodurile x_i (sau, al cărui grafic să treacă prin punctele (x_i, f_i)), adică:

$$p_n(x_i) = f_i, \quad i = \overline{0,n}$$

Se zice că p_n este *polinomul de interpolare* al funcţiei *f*, pe nodurile date.

Se arată că:

Polinomul de interpolare (al funcţiei *f*, pe $n+1$ noduri) există şi este unic ■

Forma Newton a polinomului de interpolare

$$p_n(x) = c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0)\cdots(x - x_{n-1})$$

Definim polinoamele

$$q_0(x) = 1,\ q_1(x) = x - x_0,\ q_2(x) = (x - x_0)(x - x_1),\ \ldots,$$

sau, în general,

$$q_0(x) = 1; \quad q_k(x) = \prod_{i=0}^{k-1}(x - x_i), \quad k = \overline{1,n}$$

Polinomul de interpolare sub forma Newton se scrie atunci:

$$p_n(x) = \sum_{k=0}^{n} c_k q_k(x)$$

Coeficientul c_k se numeşte *diferenţa divizată de ordinul k* (a funcţiei *f*, pe nodul x_k) şi se notează cu

$$c_0 = f[x_0]; \quad c_k = f[x_0, x_1, \ldots, x_k],\ k \geq 1.$$

Cu aceasta, expresia polinomului de interpolare se scrie

$$p_n(x) = \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] q_k(x) \tag{1}$$

sau explicit:

$$\begin{aligned} p_n(x) = {} & f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ & + f[x_0, \ldots, x_n](x - x_0)\ldots(x - x_{n-1}) \end{aligned} \tag{1'}$$

Proprietăţi ale diferenţelor divizate

P1. Diferenţa divizată este o funcţie simetrică de argumentele sale ■

Adică, considerând o permutare $(i_0, \ldots, i_n)$ a numerelor (1, …, *n*), avem:

$$f[x_{i_0}, \ldots, x_{i_n}] = f[x_0, \ldots, x_n]$$

P2. Formula de recurenţă a diferenţelor divizate:

$$f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0} \tag{2}$$

Calculul practic al diferenţelor divizate

Notând nodurile cu $x_j, x_{j+1}, \ldots, x_{j+n}$, (2) devine:

$$f[x_j, \ldots, x_{j+n}] = \frac{f[x_{j+1}, \ldots, x_{j+n}] - f[x_j, \ldots, x_{j+n-1}]}{x_{j+n} - x_j}$$

şi $f[x_j] = f_j$. Utilizând notaţia simplificată

$$f_{j\ j+1\ldots\ j+n} = f[x_j, x_{j+1}, \ldots, x_{j+n}],$$

formula de recurenţă (2) se scrie:

$$f_{j\,j+1\ldots j+n} = \frac{f_{j+1\ldots j+n} - f_{j\ldots j+n-1}}{x_{j+n} - x_j} \qquad (3)$$

Astfel, obţinem diferenţele de ordinul 1, 2, 3, etc.:

1: $f_{01} = \frac{f_1 - f_0}{x_1 - x_0}$; $f_{12} = \frac{f_2 - f_1}{x_2 - x_1}$; $f_{23} = \frac{f_3 - f_2}{x_3 - x_2}$;

2: $f_{012} = \frac{f_{12} - f_{01}}{x_2 - x_0}$; $f_{123} = \frac{f_{23} - f_{12}}{x_3 - x_1}$;

3: $f_{0123} = \frac{f_{123} - f_{012}}{x_3 - x_0}$;

Etc.

Diferenţele divizate se pot aşeza în următorul tabel – exemplu pentru 4 noduri:

x_0	f_0	f_{01}	f_{012}	f_{0123}
x_1	f_1	f_{12}	f_{123}	
x_2	f_2	f_{23}		
x_3	f_3			

Primele două coloane sunt datele problemei (Coloana a doua reprezintă şi diferenţele de ordinul 0). Diferenţele de ordinul 1, 2, 3 sunt în coloanele 3, 4, 5. Tabloul are structura triunghiulară: datele nu permit calculul diferenţelor $f_{j\ldots j+n}$ cu $j+n=4$ (care implică nodul x_4).

Polinomul de interpolare Newton, pentru exemplul din tabel, este:

$$p_4(x) = f_0 + f_{01}(x-x_0) + f_{012}(x-x_0)(x-x_1) + f_{0123}(x-x_0)(x-x_1)(x-x_2) \qquad (4)$$

Coeficienţii polinomului se găsesc în prima linie din tabel (începând cu coloana 2).

Pentru calculul practic al coeficienţilor polinomului Newton, notăm coeficienţii ca în tabloul de mai jos, anume c_{jk} (unde *j* este indicele nodului, iar *k* indicele ordinului diferenţei):

x_0	c_{00}	c_{01}	c_{02}	c_{03}
x_1	c_{10}	c_{11}	c_{12}	
x_2	c_{20}	c_{21}		
x_3	c_{30}			

Corespondenţa este:

$c_{01} = f_{01}$; $c_{02} = f_{012}$; $c_{03} = f_{0123}$

$c_{11} = f_{12}$; $c_{12} = f_{123}$;

$c_{21} = f_{21}$;

Sau, în general,

$$c_{jk} = f_{j\,j+1\ldots j+k} \qquad (5)$$

Cu aceasta, (4) se scrie:

$$p_4(x) = c_{00} + c_{01}(x - x_0) + c_{02}(x - x_0)(x - x_1) + c_{03}(x - x_0)(x - x_1)(x - x_2)$$

Coeficienţii c_{jk} se calculează prin recurenţă, cu formula care derivă din (2) şi (5):

$$c_{jk} = \frac{c_{j+1,k-1} - c_{j,k-1}}{x_{j+k} - x_j} \qquad (6)$$

Fişiere:
Pol-newton.f90: programul principal
Fun.f90: funcţia *f*

Programul principal implementează calculul coeficienţilor c_{jk} conform (6), pentru funcţia definită în fun.90. În afară de aceasta, programul calculează erorile relative ale polinomului de interpolare pe nişte valori date z_i, adică $(f(z_i) - p(z_i))/f(z_i)$; dacă $|f(z_i)| \le 1E-6$, se listează eroarea (în loc de eroarea relativă).

Subprogramul `fun(x)` returnează valorile funcţiei *f* pe *x*. (Funcţia este numită `fun`, şi valorile ei sunt stocate, în programul principal, în tabloul `f(0:n)`.)

Datele de intrare se citesc dintr-un fişier de date, care conţine datele:

d) *Titlu*: text, max. 80 caractere
e) *n*1: numărul de noduri. ($n1 = n + 1$)
f) $x(0:n)$: nodurile x_i: *n*1 valori
g) *nz*: numărul de valori z_i
h) *z*(1:nz): valorile z_i: *nz* valori

.

Datele de ieşire constau în: valorile funcţiei *f* pe noduri; coeficienţii c_{jk}; valorile polinomului *p*(*z*); erorile $f(z_i) - p(z_i)$. (Dacă $|f(z_i)| \le 0.2$, în loc de eroarea relativă se listează eroarea.)

Notă

Programul are scopul de a lista valorile şi erorile polinomului de interpolare (în general, pe alte puncte decât nodurile).

Programul se poate modifica uşor, astfel ca să aibă ca intrare valorile funcţiei pe noduri, adică valorile `f(0:n)`.(În loc ca acestea să fie calculate de funcţia definită în fun.f90.)

■

Exemplu:

Considerăm funcţia $f(x) = e^x - 3x^2$, şi nodurile $-1, \quad 0, \quad 1, \quad 2, \quad 3, \quad 4$. Listăm valoarea şi eroarea polinomului, pe mijloacele intervalelor definite de noduri: $-0.5; \quad 0.5; \quad 1.5; \quad 2.5; \quad 3.5$.

Fişierul de intrare/ieşire este:

```
Functia f = exp(x)-3*x*x; 6 noduri.
6
-1  0   1    2    3    4
5
-0.5 0.5  1.5  2.5  3.5


 f(x):  -2.632121       1.000000     -0.2817182    -4.610944
-6.914463       6.598150


 Coeficienti c_ij:
  -2.632121       3.632121      -2.456919      0.3110553
0.1336201
  4.5919430E-02
```

```
   1.000000      -1.281718      -1.523754      0.8455356
0.3632172
 -0.2817182      -4.329226       1.012853       2.298404
  -4.610944      -2.303519       7.908066
  -6.914463       13.51261
   6.598150


      z             p(z)              Eroare relativa
     -0.500000      -8.368891E-02    -5.978E-02  (* eroarea)
      0.500000      0.8673121         3.495E-02
       1.50000      -2.242733         1.128E-02
       2.50000      -6.608934        -6.308E-03
       3.50000      -3.488262         4.025E-02
```

■

Graficele funcţiei şi polinomului de interpolare sunt date mai jos.

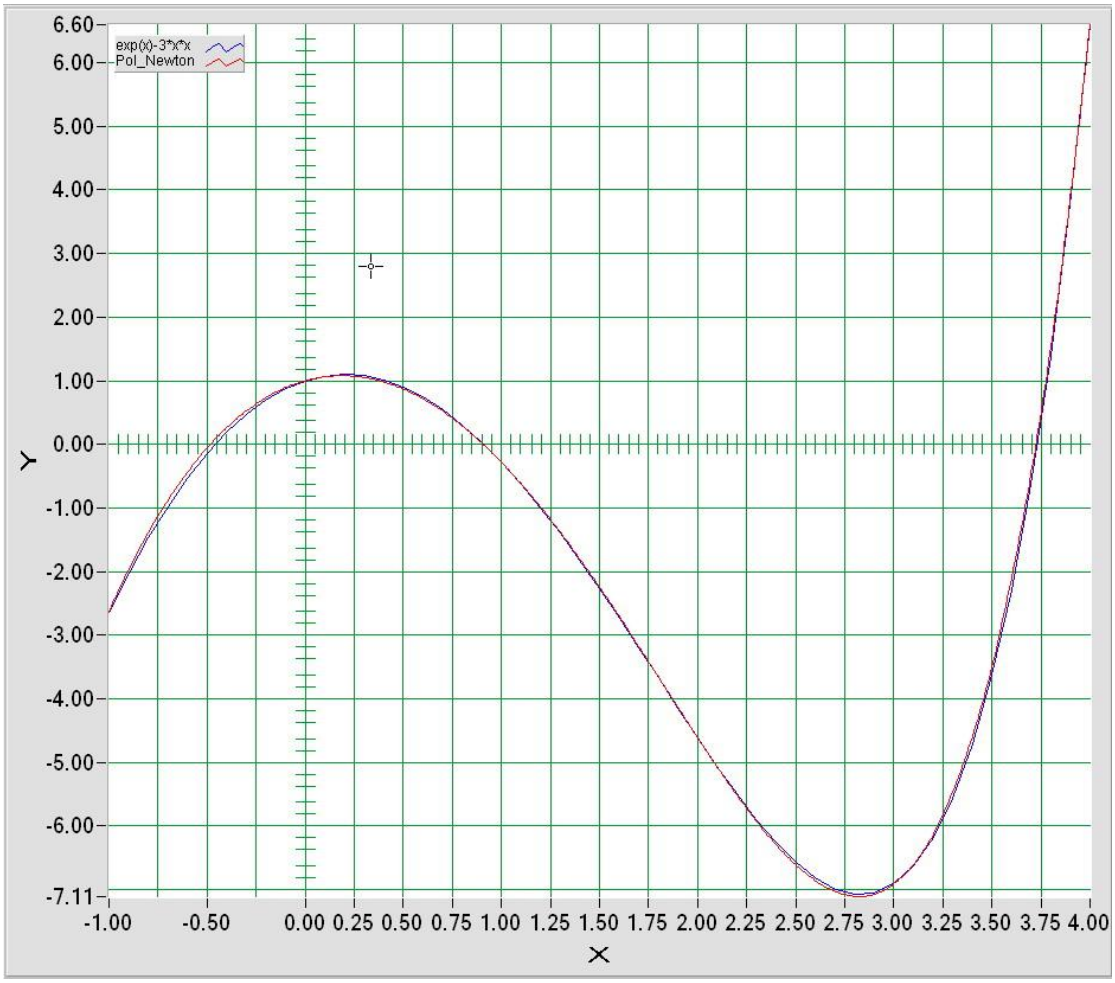


Figura 8 – Interpolare: graficul funcţiei şi polinomului de interpolare

UTILITARE

Folder tematic: $ **Utilities**

Function_expression şi width

Function_expression.f90; Function_expression_g.f90; Function_expression-2.f90; Function_expression_Sys.f90

Aceste fişiere conţin subrutine apelate de programul principal. Ele citesc şi returnează expresia funcţiei/funcţiilor cu care se lucrează; expresiile sunt citite (în cod Fortran) din fişierul sursă în care sunt definite funcţiile. Specificatorul fişierului sursă este definit în programul principal.
Programului principal scrie expresiile funcţiilor, în fişierul de ieşire şi la display.

Subrutinele sunt descrise mai jos.
Parametrii de apel sunt variabile caracter de lungime 132, sau, sau tablouri de astfel de variabile. (Lungimea 132 se poate modifica).

F_Expr(f_source, f_text)
La metode pentru ecuaţii de forma $f(x) = 0$ (Bis, Secant).
Parametri:
f_source: specificator de fişier sursă pentru funcţia *f*
f_text: expresia funcţiei *f*

F_Expr_g(g_source, g_text)
La metode pentru ecuaţii de forma $g(x) = 0$ (Fix, Aitken)
Parametri:
g_source: specificator de fişier sursă pentru funcţia *g*
g_text: expresia funcţiei g

F_Expr(f_source, f_text)
La metode pentru ecuaţii de forma $f(x) = 0$, care lucrează şi cu derivata f' (Newton).
Parametri:

f_source: specificator de fişier sursă pentru funcţia f şi f'

f_text: tablou de dimensiune 2. Expresiile funcţiilor f şi f'

F_Expr_Sys(f_source, n, f_text)

La metode pentru sisteme de ecuaţii neliniare (Fix_Sys, Newton_Sys).

Parametri:

f_source: specificator de fişier sursă

n: ordinul sistemului de ecuaţii neliniare

f_text: tablou de dimensiune n. Expresiile funcţiilor f_i sau g_i, $i = \overline{1,n}$

Notă

Instrucţiunea de definire a unei funcţii are forma `f = expresie` (respectiv, `f1 = expresie` pentru derivata lui f). Caracterul `f` (sau `f1`) poate fi precedat sau succedat, în linia instrucţiune, de spaţii sau tab-uri; acestea se omit la afişare.

Dacă utilitarul nu găseşte, în fişierul sursă specificat, o astfel de expresie a funcţiei, se afişează `f = ?` (respectiv, `f1 = ?`).

■

width.f90

Acest utilitar returnează numărul de cifre `iw` dintr-o variabilă întreagă. Serveşte la tipărirea valorilor întregi cu format în execuţie de tip `I<iw>`.

integer function width(n)

Parametri:

n: variabilă intreagă

Notă

În programul apelant, trebuie inclusă declaraţia `integer width` ■

Grafic

Calculul valorilor unei funcţii $f(x)$, pe un interval [a, b], cu pasul h.

Programul are ca fişier de ieşire valorile x, y, unde $x = a + jh$, $y = f(x)$

Fişierul de ieşire se va utiliza într-un program de grafică – pentru reprezentarea graficului lui *f*. V. de exemplu, programul GRAPH.
În pofida simplităţii lui, flexibilitatea oferită de program, îl face foarte util pentru scopul propus.

Fişiere:

grafic.f90: Subrutina de calcul a valorilor *f*(*x*)

Mgrafic.f90: Programul principal: singura lui funcţiune este de declara cu atributul external numele funcţiilor (definite în functie.f90), şi de a apela subrutina grafic pentru una (sau mai multe) dintre aceste funcţii.

functie.f90: defineşte funcţiile de lucru; expresia unei funcţii poate include un parametru *p*;

par.f90: modul pentru parametrul funcţiei.

Datele de intrare se introduc de la terminal şi constau în:

i) Există *p* = dată caracter: `Y` sau `y`, dacă funcţia include un parametru p; orice alt caracter, în caz contrar.
j) [*p*, dacă există parametru]
k) valorile *a*, *b*, *h*
l) nume fişier de ieşire (fără extensie: programul pune extensia `.dat`)

Observaţii

m) Graficul unei curbe, definită prin ecuaţia implicită $F(x, y) = 0$: se rezolvă în raport cu *y*, găsind $y = f(x)$. Dacă există două soluţii $y = f_1(x)$ şi $y = f_2(x)$, se vor construi fişierele pentru fiecare din funcţiile f_1 şi f_2. Se reprezintă apoi, ambele funcţii pe acelaşi grafic.

   Exemplu: $x^2 + y^2 = 1$; se obţine: $f_1(x) = \sqrt{1-x^2}$ şi $f_2(x) = -\sqrt{1-x^2}$.

n) Se pot construi, în aceeaşi rulare, fişierele de ieşire pentru mai multe funcţii:

   Pentru aceasta, în Mgrafic, se pun mai multe apeluri `call grafic(f1), ...,` call `grafic(fn)`, pentru funcţiile `f1, ..., fn`.

   Datele se introduc succesiv, pentru fiecare funcţie (în ordinea apelurilor) ■

Bibliografie

1. Chisăliţă A., “Numerical Analysis“, UTC-N, 2002.
2. Atkinson K. E., “An Introduction to Numerical Analysis”, John Wiley & Sons, N.Y., 1978. 2nd edition, 1989.
3. Bai Z., Demmel J., Dongarra J., Ruhe A., and van der Vorst H., “Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide“, SIAM, Philadelphia, 2000. http://www.cs.utk.edu/~dongarra/etemplates/book.html
4. Golub G. H. and Van Loan C. F., “Matrix Computations”, John Hopkins Univ. Press, Baltimore & London, 1996.
5. Goldberg D., “What Every Computer Scientist Should Know About Floating-Point Arithmetic”, ACM Computing Surveys, Vol 23, No 1, March 1991, http://www4.ncsu.edu/~gremaud/MA402/goldberg.pdf
6. Gutierrez J.M., Hernandez M.A., ” An acceleration of Newton's method: Super-Halley method”, Applied Mathematics and Computation 117 (2001) 223-239.
7. Jarratt P., Multipoint Iterative Methods for Solving Certain Equations, *The Computer Journal* (1966) 8(4): 398-400.
8. Jennings, A., “Matrix Computation for Engineers and Scientists”, J.Wiley & Sons, 1980.
9. Kahan W., "How Futile are Mindless Assessments of Roundoff in Floating-Point Computation ?", 2006, http://www.cs.berkeley.edu/~wkahan/Mindless.pdf
10. LAPACK, Version 3.0, 2000, http://www.netlib.org/lapack/ .
11. Loh E. and Walster G.W., “Rump’s Example Revisited“, Reliable Computing 8, 245–248, 2002, Kluwer Academic Publishers.
12. Neta B., Chun C., Melvin Scott M., ” A note on the modified super-Halley method”, Applied Mathematics and Computation 218 (2012) 9575–9577.
13. Ogita T., Rump S.M., and Oishi S., “Accurate Sum and Dot Product“, Technical report, Wased University, 2003.

14. Parlett B. N., and Reinsch C., “Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors”, Numer. Math. 13, 293-304, 1969.
15. Rump S. M., „Verification methods: rigorous results using floating-point arithmetic”, Acta Numerica 2010, Vol.19, Cambridge Univ. Press.
16. F. Soleymani F., Sharifi M., “On a Class of Fifteenth-Order Iterative Formulas for Simple Roots”, International Electronic Journal of Pure and Applied Mathematics, Volume 3 No. 3 2011, 245-252
17. Wilkinson J. H., “Convergence of LR, QR, and Related Algorithms”, Comput. J., 8, 1965, 77-84.
18. Westlake J. R., “A Handbook of Numerical Matrix Inversion and Solution of Linear Equations”, J. Wiley&Sons, N.Y., 1968.
19. “Fortran PowerStation Reference”, Microsoft Corporation, 1995.
20. “Fortran PowerStation Programmer’s Guide”, Microsoft Corporation, 1995.
21. “Compaq Visual Fortran Language Reference Manual”, 2001.
22. “Compaq Visual Fortran Programmer's Guide”, 2001.
23. “IMSL Mathematical and Statistical Libraries”, Compaq Visual Fortran 6.6, IMSL Help, 1999.
24. ”IMSL Fortran Numerical Library, Version 6.0”, VNI, 2008
25. “Intel Visual Fortran Compiler 11.x for Windows”, 2010 http://software.intel.com/en-us/articles/fortran-compilers/
26. “High-Precision Software Directory”, 2014, http://crd-legacy.lbl.gov/~dhbailey/mpdist/

Lista Figurilor

Index