

ADRIAN CHISĂLIȚĂ

ANA

Biblioteca de Analiză numerică – surse Fortran 90

Manual de utilizare

Universitatea Tehnică din Cluj-Napoca

Cluj-Napoca, 2013

Notă copyright

Versiune ANA (on-line): Februarie 2013

Ediție Manual de utilizare (on-line): Februarie 2013

ANA se găsește la următorul URL:

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

(Manualul se găsește în folderul ANA/_DOC/)

Copyright © Adrian Chisăliță 1991-2012

Drepturi de utilizare a bibliotecii ANA

1. În scop educațional:

- Utilizarea este liberă.
- Utilizatorul are dreptul la o copie a Bibliotecii și 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 licența educațională.

2. În scop comercial:

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

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

Manualul de utilizare

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

Autorul garantează conținutul bibliotecii ANA, numai pentru utilizarea în scop educațional.

Autorul își declină responsabilitatea pentru orice consecințe care ar decurge din utilizarea bibliotecii în alt scop.

Sursele din ANA sunt oferite ”așa cum sunt”.

Codul sursă și 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

Notă copyright.....	2
Actualizare	3
INTRODUCERE – Biblioteca ANA.....	8
CAPITOLUL I – REPREZENTAREA NUMERELOR ÎN CALCULATOR și ERORI	
.....	13
Recurrence	13
Bessel.....	13
Exemplul Muller-Kahan	14
Errors	14
Assoc_sum	14
Do	15
Ecuatie_gr2	15
Integer2	16
Intrinsec_2.....	16
Loss_signif	16
Rump_pol.....	17
Rump II	18
Special_Values	19
SSH.....	20
Sum_Armonic	21
Sum_ErrFree	22
Test_eps	22
Ulp	23
CAPITOLUL II – ECUAȚII NELINIARE	24
II-1 Ecuatii de forma $f(x) = 0$	24
Bis.....	25
Secant.....	27
Newton.....	30
II-2 Ecuatii de forma $x = g(x)$. Metoda punctului fix.	35
Fix.....	35
Stationar	38
Aitken	40
Aitken\Aitken-G	43

II-3 Sisteme de ecuații neliniare.....	45
Fix_Sys	46
Newton_Sys	49
II-4 Rădăcinile polinoamelor	51
C_pol.....	51
Pol.....	54
Pol_Direct	57
Pol_Direct-2011	58
Pol_Complex.....	59
Pol_Halley.....	61
Laguerre	61
Muller	64
Muller\Muller_Complex	68
II-5 Metode de ordin 3 și mai înalt, pentru ecuații de forma $f(x) = 0$	70
Halley-s.....	70
1) Metoda Halley:	70
2) Metoda Super-Halley:	70
Jarratt	71
1) Ordinul 3.....	71
2) Ordinele 4 și 5.....	72
Date de intrare (Halley_s și Jarratt):.....	73
Ordin 15	73
Date de intrare	75
CAPITOLUL III – SISTEME DE ECUAȚII LINIARE	76
III-1 Metode directe.....	76
Gauss.....	76
Gauss_AB	81
LU.....	83
LU\Crout	91
LU_Complex.....	91
LU_Complex\Crout_Complex	93
Cholesky	94
Cholesky_S	97

Cholesky_Band	97
III-2 Analiza erorii și condiționare	102
Numar_Conditie	102
Hilbert	104
III-3 Metode iterative	106
Jacobi	106
Gauss_Seidel	109
Sor.....	111
CAPITOLUL IV – VALORI ȘI VECTORI PROPRII	114
Problema de valori proprii (sumar)	114
Power	121
Power_Complex	128
Inverse_Power_Shift	129
Inverse_Power_Complex.....	133
Sim_Iter	136
Jacobi	142
QR.....	148
Metoda (sumar)	148
Algoritm și cod.....	152
Detalii de implementare	153
Balansare	155
Fișiere.....	158
Structura fișierului de date	161
Exemple	164
General_R, General_R1	175
Problema generalizată de valori proprii	176
Reducerea la problema standard.....	176
Retrieve_Eigen_from_R	181
Fișiere sursă:.....	181
Fișiere de date (intrare):	181
Fișiere de ieșire (rezultate):.....	182
INTERPOLARE	184
Newton.....	184
UTILITARE.....	191

Function_expression și width.....	191
Function_expression.f90; Function_expression_g.f90; Function_expression- 2.f90; Function_expression_Sys.f90	191
width.f90	192
Grafic	192
Bibliografie	194
Lista Figurilor	196
Index	197

INTRODUCERE – Biblioteca ANA

Metodele din ANA sunt bazate pe teoria și algoritmi expuși în manualul: Adrian Chisăliță, “Numerical Analysis” (2002). În ceea ce urmează, citările unor capitole, paragrafe, se referă la această lucrare.

Notă:

- Unii algoritmi din ANA sunt mai evoluți decât cei expuși în “*Numerical Analysis*”.
- Unele surse din ANA pot fi mai evaluate decât cele descrise în acest Manual. În acest caz, actualizările și, în particular, structura fișierului de intrare, se descriu în comentariile din programul principal (Main-...).

■

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

Folderele conțin fișierele sursă necesare pentru construcția unui proiect. În general, folderul conține:

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

Programul principal realizează, cel puțin, următoarele acțiuni:

- Citește datele de intrare
- Apelează subrutina metodei
- Tipărește rezultatele.

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

Unitatea 3 conține cod pentru un exemplu de test. Pentru rădăcinile polinoamelor, sistemele de ecuații liniare și pentru problema de valori proprii, în locul unității 3, în folder este inclus un fișier 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.

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

Nume foldere. Nume unități de program și fișiere asociate

În general:

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

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

Nume foldere/Nume Fișiere Sursă

- Pentru metodele în dublă precizie, în general, numele folderului și numele fișierelor 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 și 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 clarității programării metodei, uneori în dauna compacității codului. Datele de intrare se citesc fie de la terminal, fie dintr-un fișier de date. Aceeași observație pentru scrierea datelor de ieșire. Când se utilizează un fișier de date, în general, datele de ieșire se scriu în acest fișier, în continuarea datelor de intrare. Soluția fișierului 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 fișier sau directoriu. În acest caz: fișierul de intrare se specifică cu cale completă; sau, se poate specifica numai cu nume, dacă fișierul se găsește în folderul proiectului (în același folder cu sursele).
- În versiunile actuale de cod, operațiile de mai sus se fac prin fereastra de dialog standard (în particular, selectarea fișierului de intrare).

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

Unitățile de program conțin comentariile necesare pentru semnificația datelor de intrare, parametrilor de apel ai subrutinelor, etc. Pentru concretizarea datelor de intrare v. fișierul de test al metodei.

Observație

Testele de oprire a unei iterații sunt, în general, de forma: $|x_{n+1} - x_n| \leq eps$ și $num\ ar\ de\ itera\ ii \leq linit$. Toleranța eps și numărul maxim de iterații $linit$ sunt date de intrare. Trebuie ca $eps \geq ULP(x_n)$, 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 excepția cazului $x_{n+1} = x_n$, primul test nu poate fi satisfăcut și iterația se oprește prin depășirea lui $linit$. Uneori codul testează și corectează toleranța eps

introdusă de utilizator – v. ca exemplu [metoda biseției](#). În caz contrar, la introducerea datei *eps*, utilizatorul se va ține cont de condiția de mai sus ■

Utilizare

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

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

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 obținute 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 fișiere; nume rutine: Arial 11

Date de intrare; rezultate (din fișierul de ieșire); secvențe de cod: Courier New 11

Caracterul ■ indică sfârșitul unei Propoziții, Observații, Exemplu, etc.

CAPITOLUL I – REPREZENTAREA NUMERELOR ÎN CALCULATOR ȘI ERORI

Foldere tematice: \$ Recurrence ; \$ Errors; \$ Ulp.

Recurrence

Bessel

Calculul funcției Bessel: direct, și prin recurență. Se pune în evidență, instabilitatea algoritmului bazat pe formula de recurență cu trei termeni. Implementarea Exemplului 4, Cap. 0, 2.2.

Fișiere:

Bessel.f90

Se calculează valorile funcțiilor Bessel de speța I-a, pe argumentul $y = 1$, pentru $m = 1, 10$, în două moduri:

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

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

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

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

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

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

Se remarcă divergența crescătoare cu m , a valorilor calculate prin recurență în raport cu valorile exacte – chiar pentru un număr mare de cifre semnificative (16), în J_0 și J_1 . Singurele erori introduse în calculul (b) sunt erorile de rotunjire în valorile lui J_0 și 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 $n = \text{"mare"}$, 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:

$$x_{81} = 4.99979690071464772103694582965119 \quad \blacksquare$$

Errors

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

Assoc_sum

Ne-asociativitatea sumei.

Fișiere:

Sum_1.f90, FP_precis.f90

Se consideră termenii

$$x1 = 1.25E20; \quad x2 = 555.5,$$

și sumele

$$s1_p = x1 + x2 - x1$$

$$s2_p = x1 - x1 + x2.$$

p notează precizia pentru reprezentarea termenilor (și a sumelor parțiale), anume:

s : precizie simplă (24 biți), d : precizie dublă (53 biți), și e : precizie extinsă (64 biți).

Precizia de reprezentare de 64 biți se setează cu subrutina `fpprecis`.

Rezultatele se scriu în fișierul `sum.rez`, anume:

$$s1_s = 0.000000$$

$$s1_d = 0.0000000000000000$$

$$s1_e = 552.0000000000000000$$

Suma $s2_p$ are valoarea corectă (pentru oricare p): $s2 = 555.50 \dots 0$.

Do

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

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

`sum2 = j*h;` $j = 1, n$

Fișiere:

`Do2003.f90`

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

Ecuatie_gr2

Pierdere de semnificație la rădăcinile unei ecuații de gradul 2.

Fișiere:

`Ecuatie_gr2.f90`

Se consideră ecuația $x^2 - 26x + 1 = 0$, cu rădăcinile $x_{1,2} = 13 \pm \sqrt{168}$. Rădăcinile se calculează în simplă și 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 semnificație, și 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

Calculații cu integer(2).

Fișiere:

Integer2.f90

Se știe că integer(2) se reprezintă pe 2 octeți, iar plaja de reprezentare este $-32768 \leq \text{integer}(2) \leq 32767$ (Cap. 1, 2). Programul testează suma $32767 + i2$, unde $i2$ este integer(2); de exemplu, pentru $i2 = 3$, se obține -32766. Explicație: termenii sumei se reprezintă ca mai jos (Cap 1, 2; vezi și reprezentarea acestor numere cu utilitarul Bitview):

```
0 | 11111111 11111111 = 32767
```

```
0 | 00000000 00000011 = 3
```

Adunarea produce numărul

```
1 | 00000000 00000010 = -32766
```

Intrinsec_2

Apelul funcțiilor intrinseci Fortran, pentru parametrii reprezentării.

Fișiere:

Intrinsec-2.f90

Se exemplifică apelul, pentru o dată real(4) și real(8), a următoarelor funcții intrinseci:

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

Se testează $1 + \text{epsilon}(1.0) > 1$.

Pentru semnificația funcțiilor intrinseci – v. “Compaq Visual Fortran Language Reference Manual”, 2001.

Loss_signif

Pierderea de semnificație.

Fișiere:

Loss_signif.f90

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

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

pentru $x = 10^j$, $j = 1, 2, \dots, 7$, în simplă și dublă precizie. Valorile $f(x)$ calculate în dublă precizie sunt considerate valorile ‘exacte’. Se calculează în simplă precizie, valorile funcției $g(x)$ dată de

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

g este transformata lui f , care evită calculul diferenței $\sqrt{x+1} - \sqrt{x}$ care produce pierderea de semnificație. 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 creșterea lui j , datorită pierderii de semnificație.
- Valorile $g(x)$ au 7 cifre semnificative corecte.

Rump_pol

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

Fișiere:

main-rump.f90; FP_precis.f90;

Se consideră funcția

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

și evaluarea lui funcției, pentru $x = 77617$ și $y = 33096$.

(Partea esențială e constituită din polinomul format de primii 3 termeni și, din acest motiv, exemplul poartă numele de “polinomul” lui Rump. Polinomul a fost considerat de Rump în 1983 și reluat în lucrarea din 1988).

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

24 biți (precizie simplă):	-1.180592E+21
53 biți (precizie dublă):	-1.180591620717411E+21
64 biți (precizie extinsă):	5.764607523034234880E+17

Valoarea corectă:

-0.82739 60599 46821 36814 11650 95479 81629 19990 331

(obținută cu precizie de 43 cifre zecimale de precizie; ≈ 144 biți).

Notă

În precizie cvadruplă (compilerul IVF 11.1), se obține:

113 biți (Precizie cvadrupla): 1.17260394005318

■

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

Explicație:

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 depășire de format nici pentru simpla precizie. Dar termenii doi și trei sunt egali cu aproximativ 2^{123} , și astfel, pentru reprezentarea cu toate cifrele pentru evitarea pierderii de semnificație, este necesară o precizie de cel puțin 123 biți.

Ori, chiar precizia cvadruplă (`real(16)`) nu oferă decât 113 biți. 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.

Explicația *analitică* este următoarea (Loh & Walster (2002)): x și y dați, satisfac relația $x^2 = 5.5y^2 + 1$. Cu aceasta, termenii de ordin superior lui 1 în x și y se reduc, și expresia polinomului devine

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

Astfel, suma adevărată a primilor 3 termeni (exceptând fracția) este -2.

Programul listează și valoarea lui $f_1(x, y)$, calculată în precizie dublă.

Rump II

Polinomul nr. 2 al lui Rump [Rump S.E., 2010]

Fișiere:

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 în 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 aceleași valori: $a = 77617.$; $b = 33096.$

Suma primilor 4 termeni este -2 (valoarea adevărată).

Programul listează, ca în ex. precedent, valorile polinomului calculate cu 3 nivele de precizie (compilerul CVF 6.6), și anume:

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

Cu precizie cvadruplă (compilerul IVF 11.1), se obține:

-.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, compilerul IVF 11), are *toate cifrele semnificative corecte* (24);
- Valoarea corectă se obține cu MPFUN, cu minimum 22 cifre zecimale de precizie, anume: -.8273960599468213681411;

■

Special_Values

Valori speciale.

Fișiere:

Special_Values.f90

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

- Zero cu semn: -0.; (-1)*0.
- Infinit cu semn: ±1./0.
- NaN (Not a Number): 0./0; (1./0)*0; (1./0)+(-1./0)
- Număr denormalizat: 10^{-45}

- Depășire format (superioară): 10^{40}
- Depășire format (inferioară): 10^{-46}

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

SSH

Problema "Sea Surface Height".

Fișiere:

Main_SSH-2.f90

ssh.dat

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

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

Programul calculează suma elementelor din tabloul `ssh`, cu precizie dublă și 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 = 24.00000000000000
```

```
Real(16) :
  sum_row_16 = 0.357985839247703552246093750000000
  sum_col_16 = 0.357985839247703552246093750000000
```

Valoarea corectă este cea obținută 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.

Fișiere:

Sum-armonic.f90

Programul calculează suma

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

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

- Sum_Smallest: de la cel mai mic la cel mai mare – simplă precizie;
- Sum_Largest: de la cel mai mare la cel mai mic – simplă precizie;
- Sum_Double: de la mic la mare – dublă precizie, pentru

Sum_Double se consideră valoarea exactă. Se listează valorile sumelor și 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 conține și sursa pentru calculul sumei seriei armonice alternate.

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

Sum_ErrFree

Algoritm de sumare fără erori.

Fișiere:

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: $a + b = x + y$ și $x = fl(a + b)$.

Algorithm 4.1. *Cascaded summation.*

```
function res = Sum2s(p)
n1 = p1;  $\sigma_1 = 0$ ;
for i = 2 : n
[ $\pi_i, q_i$ ] = TwoSum( $\pi_{i-1}, p_i$ )
 $\sigma_i = fl(\sigma_{i-1} + q_i)$ 
res = fl( $\pi_n + \sigma_n$ )
```

Algoritmii sunt implementați prin rutinele `sum_2(a, b, x, y)` și `sum_N(n, p, s_EF, cor)`. Cu aceștia, se calculează suma din proiectul Assoc_sum. Se obține valoarea corectă.

Test_eps

Test pentru eroarea de rotunjire a unității.

Fișiere:

test_eps.f90

Se definesc (Cap.1, 3.8):

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

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 .

Fișiere:

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

Se definește – v. Cap.1, 3.8,

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

unde: e este exponentul reprezentării lui x în modelul științific, returnat de funcția `exponent(x)`; P este precizia reprezentării, în biți ($P = 24$ sau $P = 53$).

Se calculează:

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

Se testează că:

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

Pentru test, se recomandă:

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

Programul listează parametrii reprezentării lui x (în modelul științific), anume fracția și exponentul, $ULP(x)$, și verifică relațiile de mai sus.

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

CAPITOLUL II – ECUAȚII NELINIARE

Folder tematic: \$ Nonlinear Equations

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

II-1 Ecuatii de forma $f(x) = 0$

Pentru metodele biseției, Newton și secantei, funcția de test este:

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

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

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

Mai simplu, aproximațiile rădăcinilor se pot găsi prin inspecția graficului funcției.

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

Acesta este construit cu programul GRAPH, pe baza fișierului de ieșire din programul conținut în folderul **Grafic**.

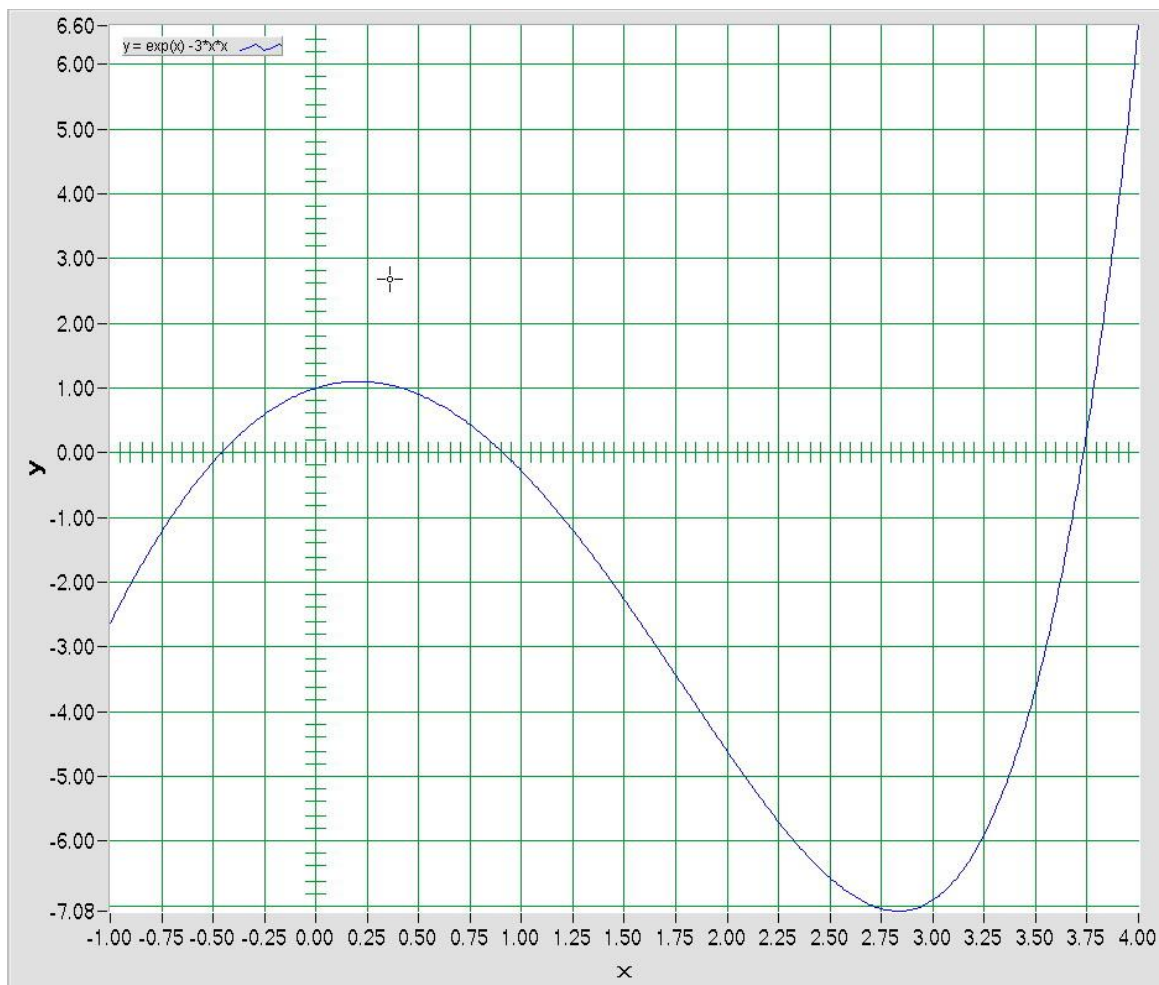


Figura 1 – Graficul funcției $f(x) = e^x - 3x^2$

Bis

Metoda biseecției.

Fișiere:

Bis2005.f90: subrutina metodei

Main_Bis2005.f90: programul principal

Fun.f90: subprogram de tip `function`, care definește funcția $f(x)$. Expresia funcției poate include un parametru p ; dacă există, valoarea lui p este dată de intrare.

Function_expression.f90, width.f90: UTILITARE.

Metoda:

Funcția f se presupune definită și continuă pe intervalul $[a, b]$, luând valori de semne contrare în capetele intervalului, și având o singură rădăcină în interval.

Se definește

$$c = (a + b) / 2$$

Se ia ca nou interval $[a, b]$ acela dintre $[a, c]$ și $[c, b]$ în care funcția ia valori de semne contrare la capete; etc. Șirul punctelor c converge la rădăcină, liniar.

Subrutina metodei este:

bis(f, a, b, eps, iter, rad, kod, ktip)

Parametrii formali sunt:

- f : Numele funcției $f(x)$. Este declarat cu atributul `external` în programul principal.
- a, b : Intervalul care conține rădăcina.
- eps : Toleranța.
- $iter$: Numărul maxim de iterații l_{nit} (intrare) / Numărul efectiv de iterații (ieșire)
- rad : Rădăcina calculată.
- kod : Cod de terminare a iterației (utilizat intern):
- 0: s-a atins toleranța eps
 - 1: s-a depășit l_{nit}
 - 1: pentru intervalul specificat $f(a)*f(b) > 0$
 - 2: rădăcină în a sau b
- $ktip$: Cod de tipărire iterații: 0 (Nu); $\neq 0$ (Da)

Testul pentru atingerea toleranței eps este

$$|b - c| \leq eps,$$

unde $[a, b]$ este intervalul curent. Metoda ajustează toleranța eps : dacă $eps < ULP(c)$, se pune $eps = ULP(c)$.

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fișierul de ieșire `nume.rez`, unde `nume` este introdus de utilizator, și conțin:

- expresia funcției f (scrisă de utilitarul `Function_expression`)
- [p , dacă există]
- datele de intrare $a, b, linit, eps$
- modul de încheiere a iterației (mesaj, conform codului `kod`)
- rădăcina calculată rad
- valoarea $f(rad)$
- numărul efectiv de iterații.

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

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

eps : 1E-6

$linit$: 50; 20; 20

$ktip$: 0 sau 1

Rezultate:

Toleranța eps atinsă.

Radacini / Număr de iterații

-0.4589624 20; 0.9100084 20; 3.733079 20;

Exercițiu

Rulați, din nou, pentru una dintre rădăcini, cu toleranța 1E-8 și $linit = 30$ ■

Secant

Metoda secantei.

Fișiere:

`Secant2005.f90`: subrutina metodei

`Main_Secant2005.f90`: programul principal

`fun.f90`: subprogram de tip `function`, care definește funcția $f(x)$.

Expresia funcției poate include un parametru p .

`Period.f90`: subrutină pentru perioada procesului staționar

`Function_expression.f90`; `width.f90`: utilitare

Metoda:

Funcție f continuă pe un interval în jurul rădăcinii α .

Formula de iterare este

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

Aproximațiile x_0 și x_1 pot încadra sau nu rădăcina.

Dacă x_0 și x_1 sunt suficient de apropiate de α (și f , f' și f'' sunt continue), șirul

$x_n \rightarrow \alpha$, iar ordinul de convergență este $p \approx 1.618$.

Subrutina metodei este:

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

Parametrii formali sunt:

f : numele funcției;

x_0, x_1 : cele două aproximații inițiale cerute de metoda secantei.

eps : Toleranța

$linit$: Numărul limită de iterații (intrare); numărul efectiv de iterații (ieșire)

rad : Rădăcina calculată

kod : Intrare: cod de tipărire a iterațiilor: 0 (nu); $\neq 0$ (da)

Ieșire: cod de încheiere a iterației (intern)

0: toleranța eps atinsă

1: depășirea numărului maxim de iterații admis $linit$

2: Proces staționar

-1: divergență

-2: numitor nul în formula metodei.

Teste:

Testul pentru atingerea toleranței eps este

$$|x_2 - x_1| \leq eps$$

Testul pentru divergența locală este

$$|x_2 - x_1| \geq |x_1 - x_0|,$$

unde x_0, x_1, x_2 , sunt trei iterații succesive. Acest test nu oprește iterația.

Testul de numitor nul este

$$|f(x_1) - f(x_0)| \leq \text{eps}_f,$$

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

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fișierul de ieșire `nume.rez`, unde `nume` este introdus de utilizator, și conțin:

- expresia funcției f (scrisă de utilitarul `Function_expression`)
- [p , dacă există]
- datele de intrare x_0 , l_{nit} , eps
- diferența curentă dif , la încheierea iterației
- modul de încheiere a iterației (mesaj, conform codului `kod`)
- rădăcina calculată rad ; valoarea $f(rad)$
- numărul efectiv de iterații.

În cazul tipăririi iterațiilor ($ktip \neq 0$), dacă procesul manifestă divergență se scrie mesajul “divergență locală”.

Datele de intrare pentru exemplul de test (3 rădăcini) se dau mai jos. Ca aproximații inițiale se iau capetele intervalelor din metoda biseției.

x_0, x_1 : -1, 0; 0, 1; 3, 4;

eps : 1E-6

l_{nit} : 10; 10; 10

$ktip$: 0 sau 1

Rezultate:

Toleranța eps atinsă.

Rădăcini / Număr de iterații

-0.4589623 7; 0.9100076 6; 3.733079 7;

Observație

Ca aproximații inițiale s-au luat capetele intervalelor considerate în metoda biseției, și aceeași toleranță eps ca în metoda biseției. Aceasta, pentru a compara rezultatele metodei secantei cu cele din metoda biseției (anume: valoarea rădăcinii și numărul de iterații) ■

Newton

Metoda Newton (o singură ecuație).

Fișiere:

Newton.f90: subrutina metodei

Main_Newton.f90: programul principal

sub.f90: subprogram care definește (și returnează) valorile funcției $f(x)$ și derivatei $f'(x)$. Expresia funcției $f(x)$ poate include un parametru p .

Period.f90: funcție care returnează perioada procesului staționar

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

Metoda:

Funcție f continuă și 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 (și f , f' și f'' sunt continue), șirul x_n converge către α , cu ordinul de convergență 2.

Subrutina metodei este:

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

Parametrii formali sunt:

SUB: Numele subrutinei care returnează valorile funcției și derivatei

x0: Aproximația inițială

eps: Toleranța

linit: Numărul limită de iterații (intrare); numărul efectiv de iterații (ieșire)

rad: Rădăcina calculată

kod: Intrare: cod de tipărire a iterațiilor: 0 (nu); $\neq 0$ (da)

Ieșire: cod de încheiere a iterației (intern)

0: toleranța *eps* atinsă

1: depășirea numărului maxim de iterații admis *linit*

2: Proces staționar

-1: divergență

-2: derivata funcției în x_0 este aproximativ nulă.

n_div : Numărul de divergențe locale

Subrutina *SUB* este:

sub(f, f1, x)

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

f1: Valoarea derivatei pe x

x: Aproximația curentă

Observație – Teste:

Se definește diferența curentă *dif*, prin:

$$dif = x - x_0,$$

unde x este valoarea iteratei curente, și x_0 valoarea iteratei la pasul anterior.

- Testul pentru atingerea toleranței *eps* este:

$$|dif| \leq eps$$

- Testul pentru divergență este

$$|dif| > |dif_ant|,$$

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

Pentru analiza tipului de divergență, satisfacerea testului de divergență nu oprește iterația, ci produce numai un mesaj în fișierul de ieșire.

- Testul pentru derivată nulă pe x este

$$|f1| \leq 1E-10$$

unde $f1 = f'(x)$.

- Se mai testează întâlnirea unui proces staționar.

■

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fișierul de ieșire `nume.rez`, unde `nume` este introdus de utilizator, și conțin:

- expresia funcției f (scrisă de utilitarul `Function_expression`)
- [p , dacă există]
- datele de intrare $x0$, $linit$, eps
- diferența curentă dif , la încheierea iterației
- [n_div , dacă există divergențe locale]
- modul de încheiere a iterației (mesaj, conform codului `kod`)
- rădăcina calculată rad ; valoarea $f(rad)$
- numărul efectiv de iterații.

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

`x0: -0.5; 0.5; 3.5;`

`eps: 1E-6`

`linit: 10; 10; 10`

`ktip: 0 sau 1`

Rezultate:

* Toleranta EPS s-a atins:

Rădăcini / Număr de iterații

`-0.4589623 4; 0.9100076 5; 3.733079 5;`

Observație

Ca aproximații inițiale $x0$, s-au luat mijloacele intervalelor considerate în metodele bisecției și secantei, și aceeași toleranță eps . S-a procedat astfel, pentru a compara rezultatele metodei Newton cu cele din metodele bisecției și secantei (valoarea rădăcinii și numărul de iterații).

Aproximații inițiale mai bune pot fi obținute de pe graficul funcției f . Cu o aproximație $x0$ mai apropiată de rădăcină, numărul de iterații necesar pentru atingerea toleranței eps va fi mai mic ■

Folderul Newton conține un sub-folder **Newton_r**. Structura de fișiere este aceeași.

Subrutina Newton (fișier Newton_r.f90) tipărește, în plus, valorile

$lambda = (x2 - x1)/(x1 - x0)$, pentru identificarea rădăcinilor multiple ale lui f ■

Observație: Proces staționar în metoda Newton

Fie funcția:

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

și iterăm cu aproximația inițială $x_0 = 5$. (În fapt, nu există rădăcini în vecinătatea lui x_0).

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

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

** Proces staționar 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 funcției f și al tangentelor la graficul lui f , în punctele de abscisă 5.017017 și 6.427927.

Procesul staționar apare dacă iterația nu este oprită când se întâlnește divergența – cum se face în mod obișnuit.

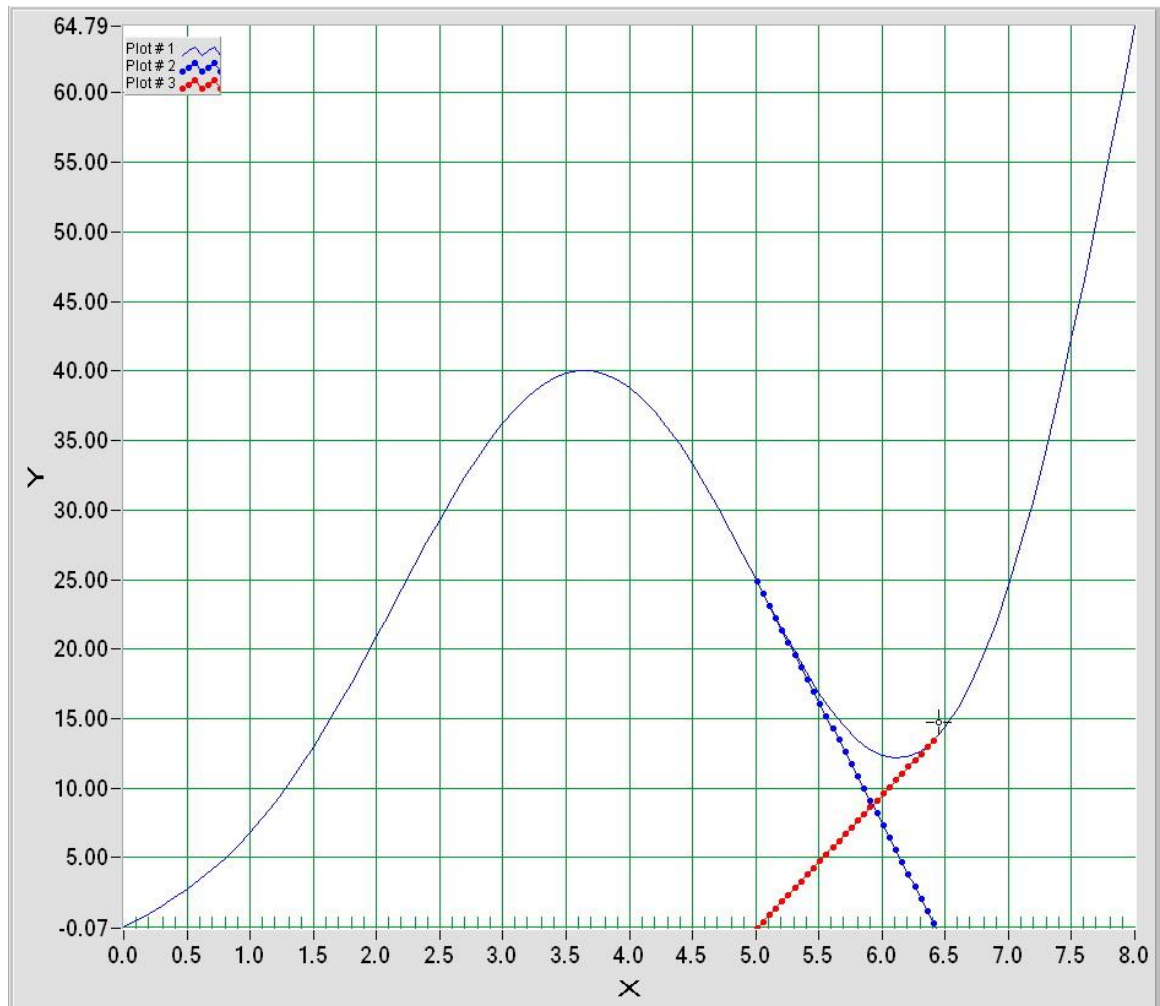


Figura 2 - Proces staționar în metoda Newton

Exercițiu:

Iterați cu: $x_0 = 6.12$; $eps = 1E-7$ și $nlim = 1000$. Rezultat: proces staționar de perioadă 5 ■

Observație

Pentru acest exemplu, metoda secantei cu aproximațiile inițiale 4.9 și 5.1 dă divergență locală în primii pași, convergând apoi la rădăcina $x = 0$ ■

II-2 Ecuații de forma $x = g(x)$. Metoda punctului fix.

Ecuația 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 $\Phi(x) = p = \text{constant}$. Pentru valorile lui $p - v$. [mai jos](#).

Fix

Metoda punctului fix (o singură ecuație).

Fișiere:

Fix2005.f90: subrutina metodei

Main_Fix2005.f90: programul principal

G.f90: subprogram de tip `function` care definește funcția $g(x)$

Period.f90: subrutină pentru perioada procesului staționar

Function_expression_g.f90; width.f90: utilitare

Metoda:

Funcție g , continuă și derivabilă pe o vecinătate $[a, b]$ a rădăcinii α .

Formula de iterare este

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

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

Subrutina metodei este:

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

Parametrii formali sunt:

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

x : Intrare: aproximația inițială; Ieșire: soluția calculată / ultima aproximație.

$xtol$: Toleranța pentru diferența a două valori succesive ale lui x .

$nlim$: Numărul limită de iterații.

kod : Intrare: cod pentru tipărirea iterațiilor: 0 (Nu); $\neq 0$ (Da)

$kod \geq 2$: tipărește și valoarea $lambda = (x_2 - x_1)/(x_1 - x_0)$,
unde x_0 , x_1 și x_2 sunt trei iterate succesive.

Ieșire: cod de încheiere a iterației:

0: s-a obținut toleranța $xtol$.

1: depășire a lui $nlim$.

2: proces staționar.

-1: divergență.

Testul pentru atingerea toleranței $xtol$ este

$$|x_1 - x_0| \leq xtol,$$

unde x_1 este iterata curentă, și x_0 iterata anterioară.

Datele de intrare se introduc de la terminal.

Rezultatele se scriu în fișierul de ieșire `nume.rez`, unde `nume` este introdus de utilizator,

și conțin:

- expresia funcției g (scrisă de utilitarul `Function_expression_g`)
- [p , dacă există]
- datele de intrare x_0 , $nlim$, $xtol$
- modul de încheiere a iterației (mesaj)
- indicele ultimei iterații, rădăcina calculată x , valoarea $g(x)$
- *diferența curentă* $= x - g(x)$

Nota

Sub-folderul **Dfix** conține fișierele pentru metoda în dublă precizie ■

Funcția g , pentru ecuația 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:

x_0 : -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 staționar de perioadă 2:

Se consideră funcția

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

și se iterează cu $x_0 = 0.8$, $tol = 1E-7$, $nlim = 1000$. Se obține rezultatul:

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

** Proces staționar de perioada = 2

Iteratia	x
842	0.7877131
843	0.7877082
844	0.7877131

** Diferenta curenta = 4.8279762E-06

Explicație:

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 condiția pentru procesul staționar de perioadă 2: $g'(x) \approx -1$.

■

Exercițiu:

Iterați în dublă precizie, cu $x_0 = 0.8$, $tol = 1E-14$, $nlim = 3000$.

Stationar

Metoda punctului fix, proces staționar: grafice.

Fișiere:

Stat2005.f90: programul principal

G.f90: funcția g

Function_expression_g.f90: utilitar

Programul are ca ieșire fișierele pentru reprezentarea grafică a unui proces staționar, anume:

Nume-iter.rez: iteratele;

Nume-g.rez, nume-bis.rez: funcția g și prima bisectoare.

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

- $x_0, n, n1$: Aproximația inițială; număr de iterate; număr de iterate omise;
- *Nume*: "Nume" – pentru construirea numelor fișierelor de ieșire.

Datele de ieșire constau din două linii-text, urmate de date numerice, și anume:

- *Linie-text-1*: Expresia funcției g
 - *Linie-text-2*: Tipul de date din fișier, anume: iterate; funcția 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 capacitatea de a recunoaște linii-text (și a le ignora), se vor șterge cele două linii-text din fișier (primele două linii), înainte de încărcarea fișierului în programul de grafică.

În particular, programul GRAPH are această capacitate ■

Exemplu:

Pentru [exemplul de proces staționar](#) din metoda punctului fix, datele de intrare sunt:

0.8; 1000; 800;

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

Graficele iteratelor și funcției g sunt date mai jos.

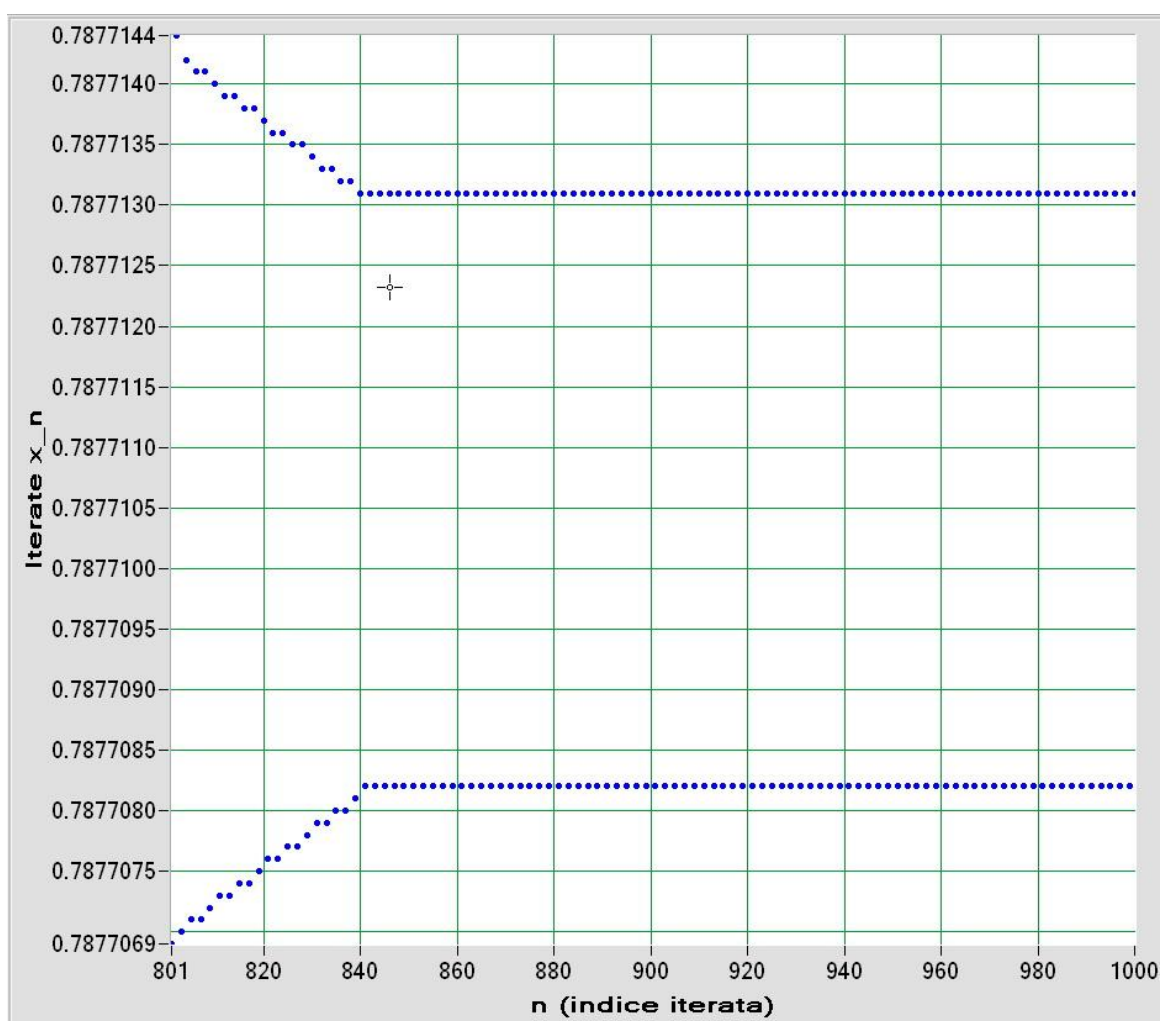


Figura 3 - Proces staționar – Graficul iteratelor

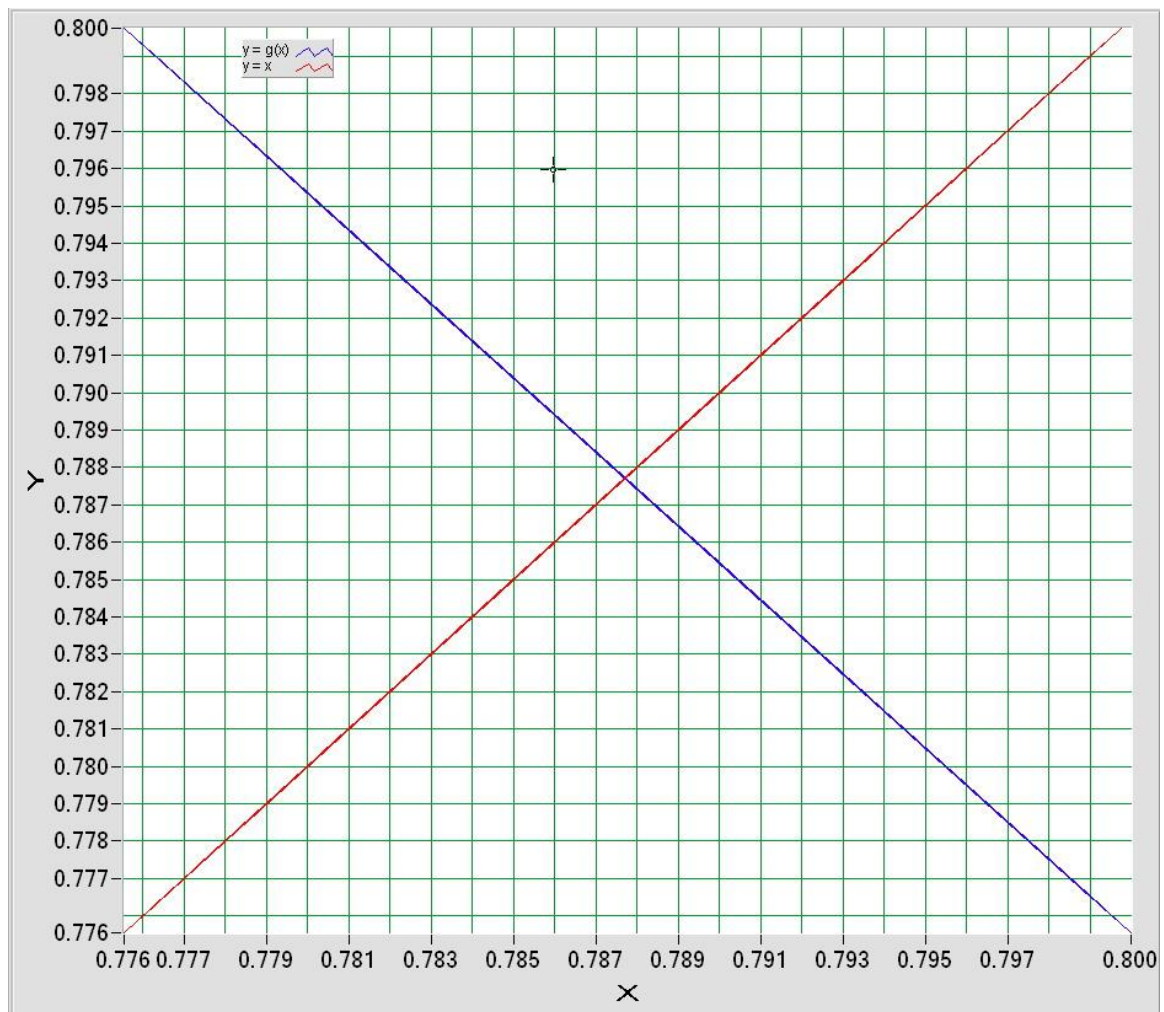


Figura 4 - Proces staționar – Graficul funcției și primei bisectoare

Aitken

Accelerarea Aitken.

Fișiere:

Aitken2005.f90: subrutina metodei

Main_Aitken2005.f90: programul principal

ga.f90: funcția g pentru Aitken

Period.f90: perioada procesului staționar

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 aproximație a rădăcinii (mai bună decât iterata următoare, $x_{n+3} = g(x_{n+2})$).

Procesul iterativ este următorul:

$$x_0 = \text{dat}; x_1 = g(x_0); x_2 = g(x_1);$$

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

$$x_6 = a_{3,5}; \dots$$

Etc.

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

Subrutina metodei este:

`aitken(g, x0, eps, linit, rad, kod)`

Parametrii formali au, în general, aceeași semnificație ca la metoda punctului fix.

Explicit:

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

x0: Intrare: aproximația inițială; Ieșire: soluția calculată / ultima aproximație.

eps: Toleranța – v. mai jos.

linit: Numărul limită de iterații.

kod: Intrare: cod pentru tipărirea iterațiilor: 0 (Nu); $\neq 0$ (Da)

$kod \geq 2$: tipărește și valoarea $lambda = (x_2 - x_1)/(x_1 - x_0)$,

unde x_0, x_1 și x_2 sunt trei iterate succesive.

Ieșire: cod de încheiere a iterației:

0: s-a obținut toleranța *eps*

1: depășire a lui *linit*

2: proces staționar

-1: numitor nul în formula metodei

Formula metodei este codată astfel: iteratele curente sunt x_0, x_1, x_2 .

$$\text{num} = (x_2 - x_1) - (x_1 - x_0)$$

$$\text{cor} = (x_1 - x_0)^2 / \text{num}$$

$$a = x_0 - \text{cor}$$

unde num este *numitorul*, iar cor este *corecția* la x_0 .

Testele pentru atingerea toleranței *eps* sunt:

1) testul obișnuit în metoda punctului fix:

$$|x_2 - x_1| \leq \text{eps}$$

2) testul pentru corecție:

$$\text{dif} = a - x_0; \quad |\text{dif}| \leq \text{eps}$$

Testul de numitor nul este $\text{num} = 0$.

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

Rezultatele se scriu în fișierul de ieșire *nume.rez*, unde *nume* este introdus de utilizator,

și conțin:

- expresia funcției g (scrisă de utilitarul *Function_expression_g*)
- [p , dacă există]
- datele de intrare x_0 , *lnit*, *xtol*
- modul de încheiere a iterației (mesaj)
- indicele ultimei iterații, rădăcina calculată *rad*, valoarea $g(\text{rad})$

Exemplu de test:

Se consideră aceeași funcție 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: aproximația inițială $x_0 = 3.7$; $\text{xtol} = 2.4\text{E-}7$; $\text{lnit} = 10$. Se iterează pentru valori $p = 0.01, 0.02, \dots, 0.09$, și $p = 0.001$.

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

```

Observație

Pentru comparația numărului de iterații din metoda Aitken, cu cele din metoda punctului fix:

Într-un pas, Aitken calculează două iterare (x_1 și x_2 , cu x_0 dat); astfel, numărul de iterare calculate este $2 \times$ numărul iterațiilor afișat de Aitken.

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

Aitken\Aitken-G

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

Fișiere:

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

Function_expression_g.f90; width.f90.

Semnificația fișierelor este aceeași 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, linit, rad, kod)

Parametrii formali au aceeași semnificație ca la [Aitken](#).

Metodei este codată astfel: iteratele curente sunt a și $a0$.

```
num =g(g(a0)) -2*g(a0) +a0
```

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

```
a =x0 +cor
```

Testele pentru toleranță se pun astfel:

1) testul din metoda punctului fix:

$$|g(a0) - a0| \leq eps$$

2) testul pentru corecție:

$$dif = a - a0; |dif| \leq eps.$$

Datele de intrare se introduc de la terminal. Parametrul efectiv pentru toleranță este numit *xtol* (în programul principal).

Rezultatele se scriu în fișierul de ieșire *nume.rez*, unde *nume* este introdus de utilizator, și conțin aceași date ca [rezultatele](#) din Aitken.

Pentru [exemplul de test](#) din Aitken, se obțin 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 ecuații neliniare

Se consideră sistemul de n ecuații cu n necunoscute

$$f_1(x_1, x_2, \dots, x_n) = 0$$

.....

$$f_n(x_1, x_2, \dots, x_n) = 0$$

Vectorial, sistemul se scrie

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

unde

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

Se notează tot cu \mathbf{x} , șirul argumentelor lui f_i , adică x_1, \dots, 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 $\mathbf{A}(\mathbf{x})$ este o matrice nesingulară.

Exemplu de test:

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

Aproximațiile inițiale se iau:

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

Acestea se determină din intersecția graficelor celor două curbe.

Notă

Dificultatea majoră constă în găsirea aproximațiilor inițiale, în cazul $n \geq 3$ ■

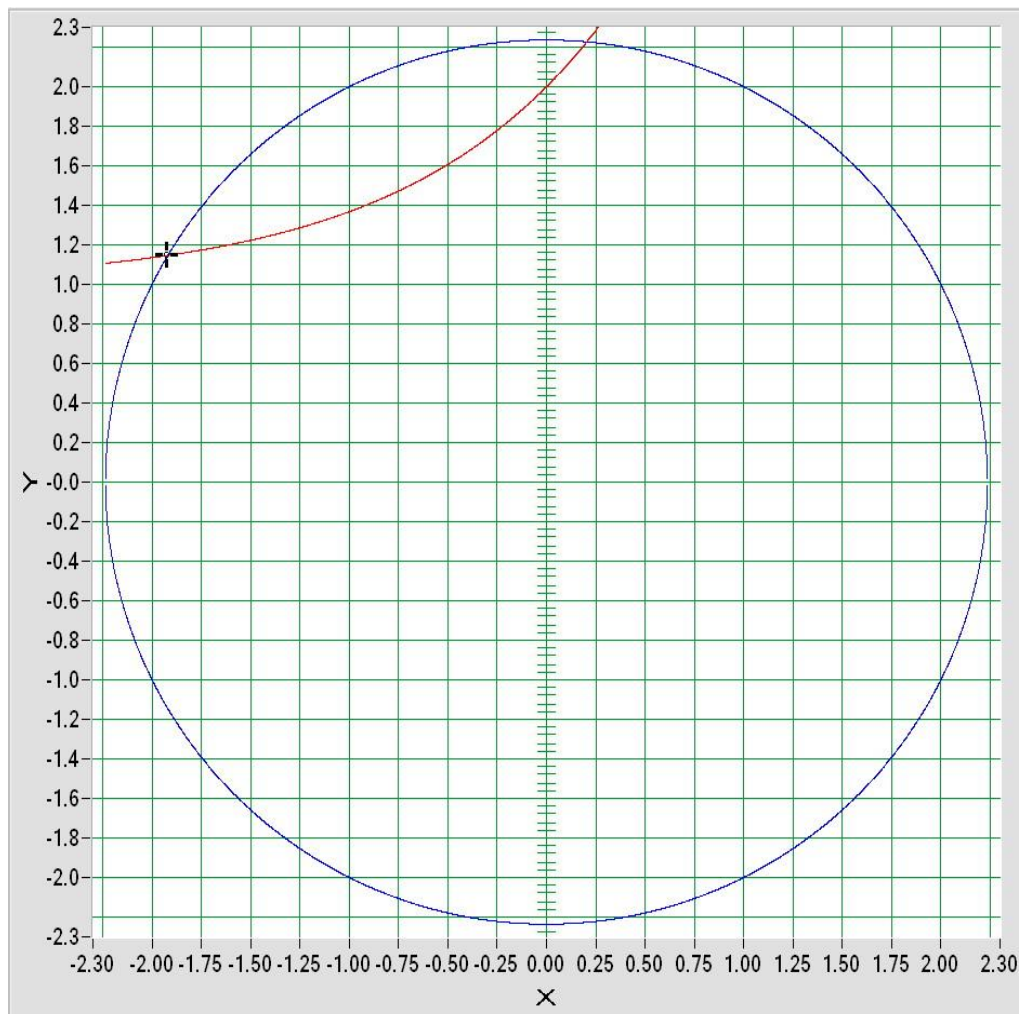


Figura 5 – Determinarea aproximațiilor inițiale

Fix_Sys

Metoda punctului fix – pentru sisteme de ecuații neliniare.

Fișiere:

Fix_Sys2005.f90: subrutina metodei

Main-Fix_Sys2005.f90: programul principal

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

norm\$.f90: norma- ∞ a unui vector \mathbf{v} (n)

Elim.f90: subrutina pentru eliminarea Gauss

Function_expression_Sys.f90; width.f90: utilitare

Metoda:

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

Iterația este definită de

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

$\mathbf{F}(\mathbf{x})$ este [jacobianul](#) lui \mathbf{f} . Derivatele parțiale ale funcțiilor $f_i(\mathbf{x})$ (elementele jacobianului) se calculează numeric, cu variația $\delta = 0.001$.

Schema de iterare este următoarea. Se definește

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

și rezultă:

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

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

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

Testele de oprire a iterației sunt:

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

$$n + 1 \leq \textit{lnit}$$

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

Subrutina metodei este

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

Parametrii formali sunt:

n : ordinul sistemului (numărul de ecuații)

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

x : tablou $\times (n)$ pentru valorile x_1, \dots, x_n

xtol: toleranța

delta: variația pentru calculul derivatelor parțiale

lnit: numărul limită de iterații

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

kod: cod de încheiere a iterației (intern):

0: toleranța atinsă

1: numărul limită de iterații atins

-1: Pivot < prag in eliminarea Gauss

Exemplul de test:

Se obțin 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 ecuații neliniare.

Fișiere:

Newton_Sys.f90: subrutina metodei

Main-Newton_Sys.f90: programul principal

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

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

norm\$.f90: norma- ∞ a unui vector \mathbf{v} (n)

Elim.f90: subrutina de eliminare Gauss

Period_Sys.f90: funcție care returnează perioada procesului staționar

Function_expression_Sys.f90, width.f90: utilitare

Metoda:

Iterația este definită de

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

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:

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

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

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

Iterația se oprește prin testele:

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

Număr de iterații $\leq l_{nit}$

unde *eps* și *lnit* sunt toleranța, și respectiv, numărul limită de iterații.

Pe lângă acestea, iterația se mai oprește dacă Jacobianul este singular, sau dacă se întâlnește un proces staționar.

Metoda are ordinul de convergență $p = 2$.

Subrutina metodei este

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

Parametrii formali sunt:

n: ordinul sistemului (numărul de ecuații)

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

jfcn: numele subrutinei care calculează jacobianul lui \mathbf{f} ; elementele jacobianului sunt returnate în tabloul $j f (n, n)$.

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

xtol: toleranța

linit: numărul limită de iterații

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

kod: cod de încheiere a iterației (intern):

0: toleranța XTOL atinsă

1: numărul limită de iterații atins

2: Proces staționar

-1: Divergentă

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

Exemplul de test:

Se obțin rezultatele:

Radacina nr.1

Aproximatia initiala: -2.000000 1.000000
 xtol: 1.000E-06; linit: 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 2.0000000
 xtol: 1.000E-06; linit: 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 și imbricat (schema Horner), pe un interval $[a,b]$, cu pasul h .

Fișiere:

C_pol2005.f90; sub_pol.f90

Metoda:

Exemplificăm pe un polinom de gradul 4. Generalizarea este imediată.

Forma dată:

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$$

Forma imbricată:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))$$

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
```

Observați că, pentru minimizarea numărului operațiilor, 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 și pid sunt valorile în simplă, respectiv dublă precizie, în forma imbricată. (pid este declarată în dublă precizie.)

2. Calculul prin funcția $p(x)$, în simplă precizie:

Se calculează valoarea polinomului, în forma dată și cea imbricată. În acest calcul, membrul drept se calculează în format dublu extins (64 biți, în coprocesor), și rezultatul se rotunjește 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}, \dots, a_0 : coeficienții polinomului (în această ordine);
- a, b, h : intervalul $[a, b]$ și pasul h

Acestea se citesc dintr-un fișier de date. Fișierul de date este utilizat și ca fișier de ieșire pentru rezultatele p, pi, pid .

Exemplu de test:

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

Fișierul de date și 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 excepția valorii în $x = 0.9$).
- La calculul prin funcție: nu există diferențe între forma dată și imbricată, și valorile coincid cu valorile *pid*.

Pol

Metoda Newton pentru polinoame – Algoritmul cu reducerea gradului

Fișiere:

Polall.f90: subrutina metodei

Main-Pol.f90: programul principal

Metoda:

Se consideră polinomul

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

Definim coeficienții

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

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

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

$$\text{adică } p(x) = (x - \xi)q(x; \xi) + p(\xi).$$

Formula de iterare este

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

$$\text{în care: } q(x_n; x_n) = q(x; \xi) \Big|_{x=x_n, \xi=x_n}.$$

Subrutina metodei:

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

Calculează o rădăcină a lui $p(x)$.

Parametri formali:

n : gradul polinomului

a : vectorul coeficienților polinomului: a_0, a_1, \dots, a_n

x_0 : aproximația inițială

eps : toleranța pentru metoda Newton

$itmax$: Intrare: numărul maxim de iterații. Ieșire: numărul efectiv de iterații

rad : rădăcina / ultima aproximație

b : vectorul coeficienților lui $q(x)$: b_1, b_2, \dots, b_n (returnați pentru o nouă rădăcină)

ier : cod de intrare/ieșire:

- Intrare: tipărirea iterațiilor (0: da; $\neq 0$: nu)

- Ieșire: cod de încheiere a iterației:

0: toleranța eps atinsă

1: numărul maxim de iterații atins

2: Proces staționar

-1: Divergență

-2: Numitor nul

n_div : numărul de divergențe locale

Formula de iterare este codată astfel:

$$x_1 = x_0 - dif$$

Observații

- Testul pentru toleranță este:

$$|dif| \leq eps$$

- Testul pentru divergență locală este:

$$|dif| \geq |dif_ant|$$

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 (deflație), și 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 puțin recomandat, întrucât: reducerea gradului poate fi un proces instabil; se cer, în plus, iterații de purificare a rădăcinilor. Alternativa este algoritmul fără reducerea gradului – v. folderul Pol_Direct.

Fișierul de intrare conține 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 iterațiilor;

a_n, a_{n-1}, \dots, a_0 : coeficienții polinomului: (în această ordine);

eps, itmax: toleranța, numărul maxim de iterații;

x0(1:nr): aproximațiile inițiale, în număr de *nr*.

Datele de ieșire se scriu în continuarea datelor de intrare, și conțin:

rad: Rădăcina calculată, sau ultima iterată dacă nu s-a atins toleranța; Codul de încheiere a iterației; Numărul de iterații; Valoarea $p(rad)$.

Exemplu: Polinomul Laguerre de gradul 6.

Fișierul de intrare/ieșire este:

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

** Radacinile prin deflatare:

Rad	Cod	Iteratii	p (Rad)
0.2228466	0	5	5.4396223E-06
1.188932	0	4	-4.6508652E-04
2.992738	0	3	-1.1693367E-03

5.775141	0	4	-1.5692167E-02
9.837474	0	4	4.5174375E-02
15.98287	0	2	-1.6622657E-02

** Radacinile polinomului:

0.2228466	0	1	5.4396223E-06
1.188932	0	1	-4.6508652E-04
2.992737	0	1	-1.1693367E-03
5.775149	1	40	-2.9657342E-02
9.837465	0	4	-5.3842347E-03
15.98287	0	1	-1.6622657E-02

■

Pol_Direct

Metoda Newton pentru polinoame – fără reducerea gradului.

Fișiere:

Pol_direct.f90: subrutina metodei

Main-Pol_direct.f90: programul principal

GetFile.f90, OpenFile.f90: subrutine pentru selectarea fișierului de intrare

Period.f90: funcție pentru calculul perioadei procesului staționar

width.f90: utilitar

Metoda:

Metoda și formula de iterare, sunt aceleași ca în [POL](#). Deosebirea este că acum, programul principal calculează rădăcinile prin iterare *direct* în polinomul original (fără reducerea gradului). Acest procedeu se recomandă față de cel descris în [POL](#).

Subrutina metodei:

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

Parametri formali sunt aceiași ca la [subrutina metodei POL](#), cu diferența că *b* este variabilă scalară, care returnează valoarea polinomului pe rădăcina calculată.

Datele de intrare și de ieșire, sunt aceleași ca pentru [fișierul de intrare din POL](#).

În plus, programul principal tipărește și semnificația codurilor de încheiere a iterației.

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

Exemplu: Polinomul Laguerre de gradul 6.

Fișierul de intrare/ieșire este:

Laguerre de ordinul 6

6 0 0

1 -36 450 -2400 5400 -4320 720

1e-6 40

0 1 3 6 9 15

** Radacinile:

Radacina	Cod	Iteratii	p(Radacina)
0.2228466	0	5	1.5187880E-05
1.188932	0	4	-4.1390103E-05
2.992736	0	3	-1.6699019E-05
5.775144	0	4	1.5937904E-04
9.837467	0	6	3.1938325E-03
15.98287	0	6	-1.2154851E-02

Cod Semnificatie

0 * Toleranta EPS atinsa

■

Pol_Direct-2011

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

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

Deosebirea față de [formula generală Newton \(1\)](#), este că acum, derivata se calculează

direct, prin: $p'(x) = na_n x^{n-1} + \dots + 2a_2 x + a_1$. 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.f90`),

astfel:

`b = polval(n, coef, x)`

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

Formula din [POL](#) este potrivită mai degrabă, pentru versiunea cu reducerea gradului. În rest, proiectul este similar cu cel din [Pol_Direct](#). Fișierul de intrare este același cu cel de la [Pol_Direct](#).

■

Pol_Complex

Metoda Newton pentru polinoame: coeficienți reali și rădăcini reale/complexe.

Fișiere:

Pol_complex.f90: subrutina metodei

Main-Pol_complex.f90: programul principal

GetFile.f90, OpenFile.f90: subrutine pentru selectarea fișierului de intrare

Period_Complex.f90: funcție pentru calculul perioadei procesului staționar

width.f90: utilitar

Metoda și formula de iterare sunt aceleași ca pentru [POL](#), iar algoritmul este cel din [POL_Direct](#).

Coeficienții polinomului sunt reali, și se pot calcula rădăcinile complexe. În plus, programul principal oferă posibilitatea inversării coeficienților polinomului – adică: se introduc coeficienții a_k , și se calculează rădăcinile polinomului cu coeficienții $1/a_k$.

Datele de intrare și ieșire sunt aceleași ca pentru [fișierul de intrare din POL](#), cu excepția că, după coeficienții polinomului se introduce o dată logică pentru inversarea coeficienților; Valoare: T (da), sau F (nu) – v. fișierul de mai jos.

Aproximațiile rădăcinilor trebuie să fie *complexe*.

Dezavantajul este că trebuie cunoscute aproximații complexe suficient de apropiate de rădăcini.

Exemplul de test:

Se consideră polinomul

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

care este dezvoltarea lui $(x-1) \cdot (x-2) \cdot \dots \cdot (x-7)$.

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

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

Fișierul de intrare/ieșire este:

```
Polinomul: (x-1)* ... *(x-7) - perturbat: -28.003
7 0 0
1 -28.002 322 -1960 6769 -13132 13068 -5040
F
1e-6 100
(1,0) (2,0) (3,0) (4,0) (5.1, 0.51) (5.1,-0.51) (7,0)
```

```
Radacina 3.819486 0.000000
Iteratia 42: x(42) = x(5)
** Proces stationar de perioada = 37
```

Radacina	Nr. de divergente locale
(1.998932,0.0000000E+00)	4
(3.819486,0.0000000E+00)	19
(5.458657,0.5402664)	9
(5.458657,-0.5402664)	9

** Radacinile:

Radacina	Cod	Iter	p(Radacina)
(1.000003 0.000000) 0	2	(1.0822721E-04	
0.000000)			
(1.998932 0.000000) 0	14	(-3.9895516E-05	
0.000000)			
(3.033130 0.000000) 0	3	(-4.4950219E-05	
0.000000)			
(3.819486 0.000000) 2	42	(-2.0952073E-03	
0.000000)			
(5.458657 0.5402664) 0	22	(1.0816487E-05 -	
4.6607955E-05)			
(5.458657 -0.5402664) 0	22	(1.0816487E-05	
4.6607955E-05)			

```
( 7.233083      0.000000      ) 0 7      (-3.4990243E-04
0.000000      )
```

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

■

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.

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 fața 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 convergență este 3 (pentru o rădăcină *multiplă*, ordinul este 1).

Fișiere:

Main_Lag.f90: programul principal; polval.f90 (subrutina compute_polval) – care
calculează valoarea polinomului (în simplă precizie)

Subrutina metodei este apelată din Biblioteca IMSL, anume:

ZPLRC (n, coef, rad)

Parametrii de apel sunt:

n : gradul polinomului

$coef$: tablou (real); coeficienții polinomului $a(0:n)$

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

Programul apelant trebuie să conțină instrucțiunea:

```
use numerical_libraries
```

Metoda Laguerre – pentru un polinom cu coeficienți 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 aproximații inițiale – v. IMSL Math/Libraries (1999).

(Subrutina ZPLRC returnează numai rădăcinile.)

■

Datele de intrare se citesc dintr-un fișier de date. Acesta conține:

Titlul problemei: text de max. 80 caractere

n: gradul polinomului

a_n, a_{n-1}, \dots, a_0 : coeficienții polinomului (în această ordine).

Inverse: dată logică; inversarea sau nu a coeficienților. Valori: T/F.

Distance: dată logică; calculul distanței rădăcinilor la un punct. Valori: T/F.

[*z*: complex; punctul z la care se calculează distanța.

Se introduce *numai dacă* *Distance* are valoarea T; în una din formele:

complex: (x, y) ; doi reali: x, y sau $x \ y]$

■

Notă:

Ultimele trei date servesc la a verifica dacă rădăcinile aparțin sau nu discului unitate cu centrul în punctul $z = (x, y)$; aceasta intervine la aprecierea stabilității metodelor numerice în mai mulți pași – pentru ecuații diferențiale ordinare.

Codul este scris astfel că:

- Dacă ambele date *Inverse* și *Distance* sunt false, ele se pot omite.
- Dacă data *Distance* este falsă, ea se poate omite.

■

Datele de ieșire se scriu în continuarea datelor de intrare, și conțin:

- Rădăcinile calculate și valorile polinomului pe rădăcini.

- Dacă *Distance* are valoarea T, se tipăresc și distanțele rădăcinilor la punctul z.

Exemplu:

Fișierul de intrare/ieșire 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 (2.1089716E-03,0.0000000E+00)

2 (-9.5487665E-04,-7.6583266E-05)

3 (-9.5487665E-04,7.6583266E-05)

4 (9.9300256E-04,0.0000000E+00)

5 (9.7102771E-04,0.0000000E+00)

6 (-2.0569569E-04,0.0000000E+00)

7 (1.0822721E-04,0.0000000E+00)

■

Notă

- Folderele *Laguerre* și *Laguerre_D* (dublă precizie): conțin sursele pentru versiunea cu utilizare IMSL.
- Folderul Laguerre Direct: conține sursele care implementează direct formula de iterare Laguerre. Pentru acest proiect, fișierul de intrare este descris în *Main_Laguerre_1.f90* - și e similar cu fișierul de la *Pol_Direct*.
În această versiune, se cer aproximații inițiale ale rădăcinilor.

■

Muller

Metoda Muller.

Funcție f , continuă și cu derivate continue până la ordinul 3, pe o vecinătate a rădăcinii.

Metoda:

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

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

Aproximații reale pot conduce la zero-uri complexe. În particular, aproximațiile 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 tematic:

Interpolation

INTERPOLARE), este:

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

Coefficienții sunt *diferențele divizate* ale funcției f pe nodurile x_i, x_j, x_k , anume:

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

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

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

Coefficienții a, b, c , se găsesc prin:

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

Formula de iterare este

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

Ca aproximație 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 fața radicalului se ia astfel ca numitorul să fie cel mai mare, și 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

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

$$x_3 = x_2 - 2c/b_1; \text{ altfel, } x_3 = x_2 - 2c/b_2.$$

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 și x_1 .
- Se procedează la o nouă iterație (cu noua secvența x_0, x_1, x_2).

Testul de oprire a iterației este

$$|x_3 - x_2| \leq EPS,$$

unde EPS este toleranța; remarcăți că $x_2 = x_3_{\text{ anterior}}$.

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

Fișiere:

Muller2005.f90: subrutina metodei

select.f90: subrutina de selecție a valorilor x_0, x_1, x_2 (pentru iterația următoare).

Main_Muller2005.f90: programul principal

f.f90: funcția $f(x)$

Function_expression.f90; width.f90: utilitare

Subrutina metodei:

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

Parametrii formali:

f: Numele funcției *f*

x0, x1, x2: Aproximațiile inițiale

eps: Toleranța

iter: Numărul limită de iterații (intrare); numărul efectiv de iterații (ieșire)

rad: Rădăcina calculată

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

Ieșire: cod de încheiere a iterației (intern):

0: Toleranța *eps* atinsă

1: *lnit* depășit

2: Rădăcina în aproximațiile inițiale

-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 fișierului de ieșire (acesta primește extensia .rez)
- *Exist_p*: dată logică. Valori: T (true) sau F (false)
- [*p*: dacă *Exist_p* = T]
- *eps, lmit*: toleranța, numărul limită de iterații
- *x1, h*: aproximația *x1*, pasul
- *kod*: cod de tipărire a iterațiilor

Programul principal pune (înainte apelul subrutinei muller):

$x_0 = x_1 - h; \quad x_2 = x_1 + h$

Datele de ieșire se scriu în fișierul de ieșire nume.rez, și sunt următoarele:

- expresia funcției *f* (scrisă de utilitarul Function_expression)
- [*p*, dacă există]

- aproximațiile x_0, x_1, x_2
- $lnit, eps$
- modul de încheiere a iterației (mesaj, conform codului *kod*)
- rădăcina calculată *rad*; valoarea $f(rad)$
- numărul efectiv de iterații.

Exemplu de test:

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

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

Se iau: $x_0 = 0.8$, $h = 0.2$ (Rezultă aproximațiile inițiale: 0.6; 0.8; 1.0)

$eps = 1E-6$, $lnit = 10$; $ktp = 0$.

Fișier de ieșire:

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 funcției și polinomului de interpolare se dau mai jos.

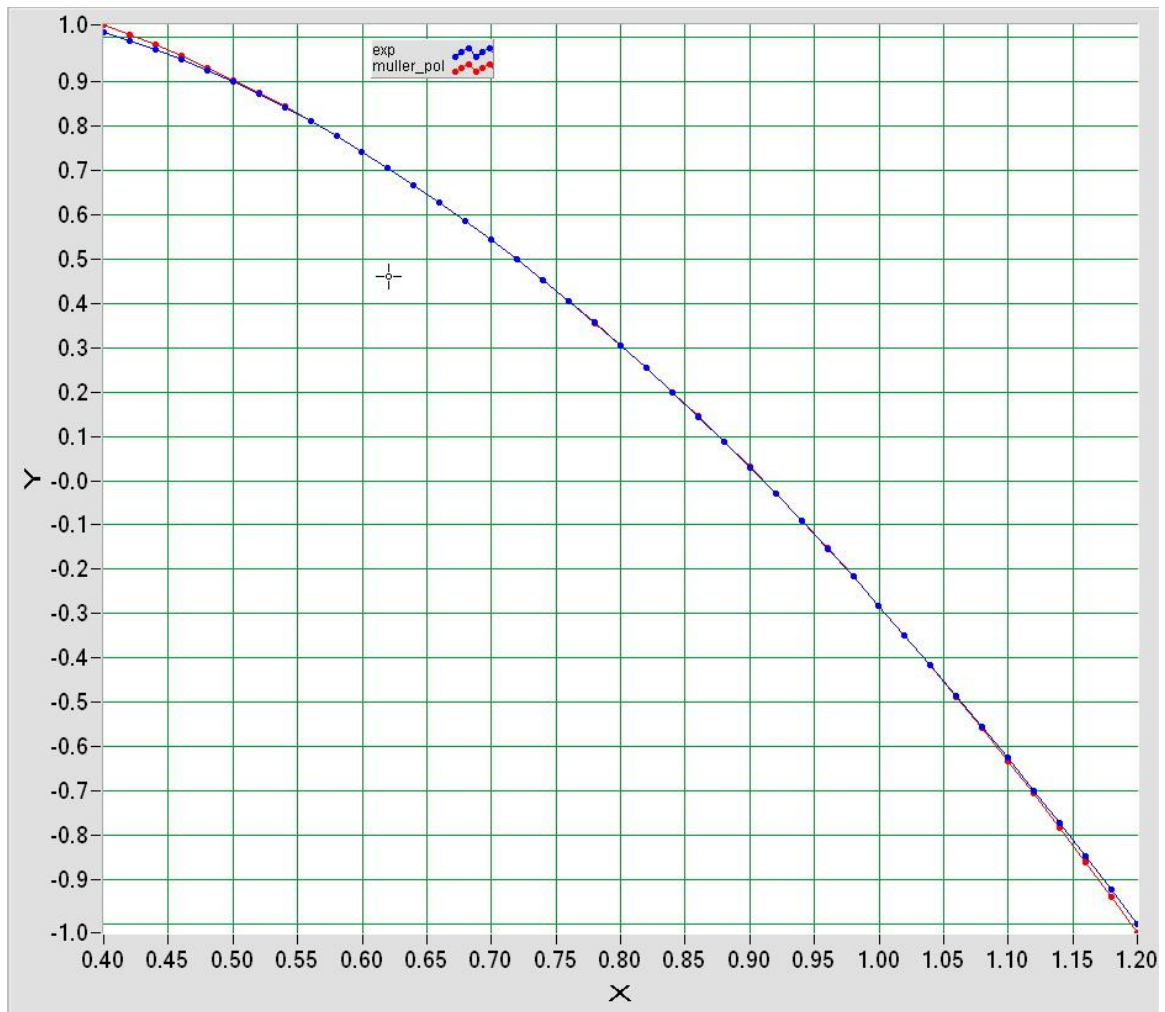


Figura 6 – Metoda Muller: graficul funcției și polinomului de interpolare

Muller\Muller_Complex

Metoda Muller pentru rădăcinile unei funcții complexe.

Fișiere:

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)

Semnificațiile parametrilor sunt aceleași ca la [Muller](#), cu deosebirile:

- datele numerice și funcția f sunt declarate complexe

- programul este scris în dublă precizie

Datele de intrare se citesc dintr-un fișier 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
- *z1(1:nrad)*: *nrad* aproximații – reale sau complexe
- *h*: pasul. Valoare reală.
- *eps*, *lnit*: toleranța, numărul limită de iterații
- *ktip*: cod de tipărire a iterațiilor: 0 (da); ≠0 (nu)

Datele de ieșire se scriu după datele de intrare, și conțin: rădăcina, valoarea funcției pe rădăcină, și numărul de iterații (pentru *nrad* rădăcini).

Exemplu de test:

Pentru polinomul din [exemplul de la Pol. Complex](#), se iterează cu $eps = 1E - 10$;

lnit = 20; aproximația reală $x_1 = 6$; $h = 0.1$. Fișierul de intrare/ieșire 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

```

$$\begin{array}{l}
 f = (-1.421085471520200\text{E}-014, 3.197442310920451\text{E}-014) \\
 2 \quad 5.458675825432509 \quad \quad \quad 0.5401258000668571 \quad \quad \quad 7 \\
 f = (-2.842170943040401\text{E}-014, 1.776356839400250\text{E}-014)
 \end{array}$$

■

Rădăcina este găsită în 7 iterații. Cu aproximația $x_1 = 5$ se găsește aceeași rădăcină în 9 iterații.

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:

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

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:

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

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

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

Metoda are ordinul 3 într-o rădăcină simplă.

2) Metoda Super-Halley:

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

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

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:

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

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

Parametrii α, a_1, a_2 au valorile:

a) Metoda #1 de ordin 3:

$$\alpha = -\frac{1}{2}; \quad a_1 = 0; \quad a_2 = 1;$$

$$j(x) = \frac{f(x)}{f'(x - \frac{1}{2}u(x))}.$$

b) Metoda #2 de ordin 3:

$$\alpha = -\frac{2}{3}; \quad a_1 = \frac{1}{4}; \quad a_2 = \frac{3}{4};$$

$$j(x) = \frac{1}{4} \frac{f(x)}{f'(x) + 3f'(x - \frac{2}{3}u(x))}.$$

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:

a) *Ordinul 4:*

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

b) *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}, \dots, a_0 : coeficienții 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 ECUAȚII LINIARE

Sistemul de ecuații liniare este

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

unde \mathbf{A} est matrice $n \times n$, iar \mathbf{x} și \mathbf{b} vectori coloană cu n coordonate.

Rezolvarea se poate considera pentru mai mulți termeni liberi \mathbf{b} .

Folder tematic: \$ Linear Systems

III-1 Metode directe

Gauss

Eliminarea Gauss – cu pivotare parțială.

Fișiere:

Elim.f90: subrutina metodei

Main-Elim.f90: programul principal

width.f90: utilitar

Fișier de date: Ex2.dat

Metoda:

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

Sistemul dat se notează

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

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

Sistemul este

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

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

$$\mathbf{A}^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1k}^{(1)} & \cdot & \cdot & \cdot & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2k}^{(2)} & \cdot & \cdot & \cdot & a_{2n}^{(2)} \\ \cdot & 0 & \ddots & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & a_{k-1,k-1}^{(k-1)} & \cdot & \cdot & \cdot & a_{k-1,n}^{(k-1)} \\ \hline 0 & \cdot & \dots & 0 & a_{kk}^{(k)} & \cdot & a_{kj}^{(k)} & \cdot & a_{kn}^{(k)} \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & a_{ik}^{(k)} & \cdot & a_{ij}^{(k)} & \cdot & a_{in}^{(k)} \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & 0 & a_{nk}^{(k)} & \cdot & \cdot & \cdot & a_{nn}^{(k)} \end{bmatrix} \times (-m_{ik});$$

$$\downarrow$$

$$\oplus \swarrow$$

$$\mathbf{b}^{(k)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ b_{k-1,k-1}^{(k-1)} \\ \hline b_k^{(k)} \\ \cdot \\ b_i^{(k)} \\ \cdot \\ b_n^{(k)} \end{bmatrix} \times (-m_{ik})$$

$$\downarrow$$

$$\oplus \swarrow$$

Presupunem $a_{kk}^{(k)} \neq 0$ – v. mai jos [pivotare](#), și defnim multiplicatorii de la pasul k :

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

Pentru $i = k+1, \dots, n$: linia k se înmulțește cu $-m_{ik}$, și se adună la linia i . Rezultă elemente nule în coloana k , sub elementul diagonal $a_{kk}^{(k)}$. Linia k rămâne neschimbată.

Noii coeficienți și termeni liberi vor fi:

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

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

$$\mathbf{U}\mathbf{x} = \mathbf{g}.$$

Explicit:

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

Acest sistem se rezolvă prin substituție înapoi:

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

■

Pivotarea parțială:

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 și $I > k$, atunci se schimbă liniile k și I (în $\mathbf{A}^{(k)}$ și în $\mathbf{b}^{(k)}$).

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

c) dacă pivotul este nul, matricea este singulară.

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

■

Număr de operații: Pentru $n = \text{"mare"}$, numărul de operații în eliminarea Gauss este:

$$NOP_{Guass} \approx \frac{n^3}{3} \quad \blacksquare$$

Subrutina metodei:

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

Parametri formali:

n : numărul ecuațiilor

nb : numărul de termeni liberi

$A(n, n)$: matricea coeficienților

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

prag: prag pentru pivot ≈ 0 .

kod : cod pentru eliminarea Gauss:

$kod = 0$: o.k.; $kod \neq 0$: pivot ≈ 0 (pivot < prag)

kodPrint: cod de tipărire – v. mai jos.

ELIM face [pivotare parțială](#). 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**.

Soluția este returnată în coloanele din **B**.

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

- *Titlu*: text de max. 80 caractere
- *n, nb*: numărul de ecuații; numărul de termeni liberi.
- *Matricea coeficienților A*: $n \times n$. Se scrie așa cum e dată - pe linii și coloane.
- *Termenii liberi b*: Se scriu pe linii: *nb* linii a *n* termeni liberi.
- *cod_Inversare_Coeficienți, cod_Tipărire*:

cod_Inversare_Coeficienți: dacă este $\neq 0$, atunci se rezolvă sistemul cu

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

cod_Tipărire: dacă este $\neq 0$: Se tipăresc permutările și pivoții.

- *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 tipărește vectorul $\mathbf{Ax} - \mathbf{b}$).

Observație

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

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

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

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 3 & 4 & 1 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 6 & 1 \\ 7 & 1 \\ 8 & 1 \end{bmatrix}.$$

Fișierul de intrare/ieșire:

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

Fișierul de intrare/ieșire (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.

Fișiere:

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 coeficienților extinsă cu termenii liberi.

n : numărul ecuațiilor.

$ncol$: $ncol = n + nb$, 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.; $kod \neq 0$: pivot ≈ 0 (pivot $<$ prag).

$pivot$: valoare pivot, când pivot $<$ prag.

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

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

- *Titlu*: text de max. 80 caractere
- n, nb : nr. de ecuații; nr. de termeni liberi.
- *Matricea coeficienților A*: $n \times n$. Se scrie așa cum e dată - pe linii și coloane.
- *Termenii liberi b*: Se scriu pe linii: nb linii a n termeni liberi.
- *cod_Inversare_Coeficienți*:

cod_Inversare_Coeficienți: dacă este $\neq 0$, 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 $\mathbf{Ax} = \mathbf{b}$.

Observație

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

Datele de ieșire se scriu în continuarea datelor de intrare și conțin soluția și proba (dacă este comandată prin *check_text*).

LU

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

Fișiere:

Primele 5 fișiere constituie subrutinele metodei.

LUdecomp.f90: descompunerea LU

LUsolve.f90: substituție înapoi

apvt-LU.f90: pivotare parțială

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 pașii:

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

$$\boxed{\mathbf{A}} = \boxed{\begin{array}{c|c} & \mathbf{U} \\ \hline \mathbf{L} & \end{array}}$$

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

$$\boxed{\begin{array}{c|c} & \\ \hline \mathbf{L} & \end{array}} \left| \begin{array}{c} \mathbf{y} \\ \end{array} \right| = \left| \begin{array}{c} \mathbf{b} \\ \end{array} \right|$$

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

$$\boxed{\begin{array}{c|c} \mathbf{U} & \\ \hline & \end{array}} \left| \begin{array}{c} \mathbf{x} \\ \end{array} \right| = \left| \begin{array}{c} \mathbf{y} \\ \end{array} \right|$$

Formulele pentru calculul elementelor lui \mathbf{L} și \mathbf{U} – cu l_{ii} ales arbitrar (nenul):

$$l_{ii} = \text{ales arbitrar (nenul)}, \quad i = 1, 2, \dots, n$$

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

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

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

Elementele lui \mathbf{L} și \mathbf{U} se determină în următoarea secvență, conform schemei de mai jos (se calculează elementele din linia i din \mathbf{U} , și coloana i din \mathbf{L}).

$$\begin{array}{c|cccc} l_{ii} & \rightarrow & u_{i,i} & u_{i,i+1} & \dots & u_{i,n} \\ \downarrow & & & & & \\ l_{i+1,i} & & & & & \\ \vdots & & & & & \\ l_{n,i} & & & & & \end{array}$$

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

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

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

Pivotare:

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

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

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

Așa cum se arată în Exemplu-2, trebuie pivotat și dacă pivotul este foarte mic. Practic, se testează dacă pivotul este mai mic decât un *prag*.

Două strategii de pivotare parțială 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 soluției, însă consumă mai mult timp de calcul.

■

Observație

- Dacă nu se pivotează, atunci avem $\mathbf{LU} = \mathbf{A}$.
- Dacă se pivotează, atunci avem $\mathbf{LU} = \mathbf{A}'$, unde $\mathbf{A}' = \mathbf{PA}$, iar \mathbf{P} este o matrice de permutare de linii. Matricea \mathbf{A}' se obține din \mathbf{A} , permutând liniile în aceeași 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

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

Determinantul matricii \mathbf{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: \mathbf{L} și \mathbf{U} în \mathbf{A} , în triunghiul inferior, respectiv superior. (Elementele diagonale din \mathbf{L} fiind 1 nu se mai stochează.)
- Pivotare parțială - prin apel apvt-LU; aceasta apelează compute_a_ii.f90 și 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 ecuații

order: vector întreg

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

kodPivot: cod pentru strategia de pivotare:

$kodPivot \neq 0$: 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)

Pașii [2 și 3](#) de mai sus.

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

Soluția este returnată în *x*.

Parametri:

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

b: vectorul termenilor liberi

x: soluția

order: vector întreg

Observație

În LU și 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 fișier de date. Acesta conține următoarele date

(scrise pe linii):

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

- *Matricea coeficienților A*: $n \times n$ (Se scrie pe linii și coloane).
- *Termenii liberi b*: Se scriu pe linii: nb linii a n termeni liberi.
- *Cod_Tipărire, Cod_inversare_coeficienți*: 0 / ≠0; 0 / ≠0.
 - *Cod_Tipărire = 0*: Soluția, în fișierul de date.
Pentru *Cod_Tipărire ≠ 0*, se tipăresc în plus datele de mai jos.
 - *Cod_Tipărire = 1*: Soluția, la display; Matricile **L** și **U**, în fișierul de date.
 - *Cod_Tipărire ≥ 2*: Vectorul `order`, la fiecare pivotare;
Determinantul lui **A**; Dacă $|u(i, i)| < prag$: Se tipărește și linia i a matricii **A** (la pasul respectiv).
 - *Cod_inversare_coeficienți ≠ 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 - soluția.
 - F, f: Se verifică soluția și $LU = PA$ (PA este A permutat).

Observație

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

Datele de ieșire se scriu în fișierul de date, după datele de intrare.

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

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

Exemplu – 1 (Fișier de date `u3.dat`):

Matrice 4×4 cu $u_{22} = 0$; $u_{33} = 0$. Fara pivotare.

88

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		
0.375000	-0.416667	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) = 0.0000000

* Proba Ax =b:

	A*x -b
1	0.0000000E+00
2	-5.9604645E-08
3	1.1920929E-07
4	-5.9604645E-08

■

Exemplu – 2 (Fișier 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) = 0.0000000

* 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.

Fișiere:

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

Semnificațiile sunt aceleași ca ale [fișierelor 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

Fișiere:

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;

Semnificațiile sunt aceleași ca ale [fișierelor din metoda Doolittle](#).

Fișierul de date are aceeași structură ca [fișierul de date din LU](#), cu particularitățile:

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

Exemplu:

$$\mathbf{A} = \begin{bmatrix} 3-2i & 2+4i & -3i \\ 1+i & 2-6i & 1+2i \\ 4 & -5+i & 3-2i \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 5-i & 10+5i \\ 4-3i & 6-7i \\ 2-i & -1+2i \end{bmatrix}.$$

Fișierul 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
0.250000      0.250000      1.000000      0.000000
0.750000      -0.500000      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      -0.250000      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_Decompcrout_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;

Semnificațiile sunt aceleași ca în LU – metoda Crout.

Cholesky

Metoda Cholesky – cu matricea \mathbf{L} .

Metoda:

Se aplică pentru un sistem cu *matrice simetrică și pozitiv definită*. Pentru o astfel de matrice există o descompunere \mathbf{LU} , în care \mathbf{U} este transpusa lui \mathbf{L} , adică

$\mathbf{A} = \mathbf{LL}^T$. Metoda Cholesky nu necesită pivotare.

Pașii rezolvării sunt aceiași ca [pașii LU](#), anume:

- 1) Factorizare: $\mathbf{A} = \mathbf{LL}^T$
- 2) Calculul lui \mathbf{y} , prin substituție înainte: $\mathbf{Ly} = \mathbf{b}$
- 3) Calculul soluției \mathbf{x} , prin substituție înapoi: $\mathbf{L}^T \mathbf{x} = \mathbf{y}$.

Pentru $n = \text{”mare”}$, numărul de operații în metoda Cholesky este

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

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

Fișiere:

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): Substituție înainte (Pasul 2)

back(l,b): Substituție 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 ieșire: matricea **L** (triunghiul inferior).

kod: cod de ieșire 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, și la ieșire soluția. Anume:

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

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

Main_Cholesky-2004.f90: programul principal

Observație

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

■

width.f90: utilitar

Datele de intrare se citesc dintr-un fișier de date, care conține:

Titlu: text, max. 80 caractere.

n, *nb*: *n* = numărul de ecuații; *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 soluția calculată.

Datele de ieșire se scriu în continuarea datelor de intrare.

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

$$\mathbf{A} = \begin{bmatrix} 16 & 4 & 8 \\ 4 & 5 & -4 \\ 8 & -4 & 22 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 1 & 2 & 28 \\ 1 & 2 & 5 \\ 1 & 2 & 26 \end{bmatrix}.$$

Fișier 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: $A \cdot x = b$

```
1 0.0000000E+00
2 0.0000000E+00
3 0.0000000E+00
```

■

Cholesky_S

Metoda Cholesky – cu matricea S .

Metoda:

Metoda este aceeași cu cea expusă la [Cholesky](#). Cu deosebirea că acum, matricea de referință este matricea $S = L^T$. Astfel, pașii sunt:

1. Factorizare: $A = S^T S$
2. Calculul lui y , prin substituție înainte: $S^T y = b$
3. Calculul soluției x , prin substituție înapoi: $Sx = y$.

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

Datele de intrare și ieșire sunt aceleași ca la [Cholesky](#), cu deosebirea că matricea A se introduce prin elementele triunghiului superior.

Fișiere:

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

Semnificațiile sunt aceleași ca ale [subrutinelor din Cholesky](#).

Exemplu:

[Exemplul din Cholesky](#), pentru coloanele 1 și 3 de termeni liberi.

Fișier de date: 1.dat

Cholesky_Band

Cholesky pentru matrici bandă (simetrice și pozitiv definite).

Matrice bandă: elementele unei linii sunt alcătuite din:

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

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

Observație

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

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

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

Metoda:

Matricea bandă se presupune *simetrică și pozitiv definită*.

Descompunerea \mathbf{LL}^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.

Pașii sunt cei de la [Cholesky S](#).

1) Descompunerea Cholesky:

Elementele active la un pas al descompunerii, sunt conținute 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:

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

Acest vector este constituit din două părți:

- Sub-vectorul $Y1(1:NY0)$, de dimensiune $LIM * (LIM + 1) / 2$: servește 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 $n * LIM$: inițial, stochează matricea A , pe linii. În cursul descompunerii Cholesky, stochează liniile procesate ale matricii A .

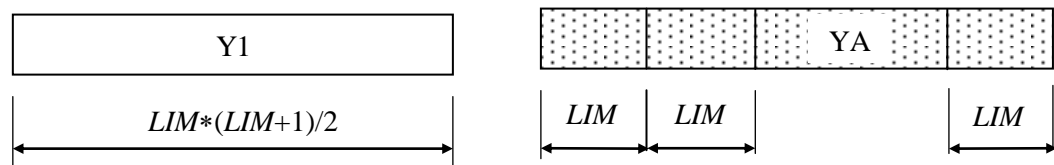


Figura 7 - Tablourile Y1 și YA

2) Soluția:

Aceasta se calculează prin substituție înainte și înapoi (pașii [2 și 3](#)).

Pentru alte considerații privind metoda (stocare, algoritm), v. Cap. 4, 5.6.

Fișiere:

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

Cholesky_Band.f90: descompunerea Cholesky în bandă

Loca.f90: funcție 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 soluția

width.f90: utilitar

Fișier de date: b6-3.dat

Datele de intrare se citesc dintr-un fișier de date, care conține:

Titlu: text, max. 80 caractere.

n, LIM, nb : n = numărul de ecuații; LIM = semi-lățimea 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 tipărește soluția;

kod ≠ 0: se tipărește și matricea **S**;

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

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

C, c, P, p: se verifică soluția.

Exemplu:

Se consideră sistemul definit de:

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & -1 & 0 & 0 & 0 \\ & 4 & -1 & -1 & 0 & 0 \\ & & 4 & -1 & -1 & 0 \\ & & & 4 & -1 & -1 \\ & \textit{Simetric} & & & 4 & -1 \\ & & & & & 4 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$$

Fișier 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 și condiționare

Numar_Conditie

Calculul numărului de condiție al unei matrici, după norma- ∞ și norma-1.

Metoda:

Se dă sistemul $\mathbf{Ax} = \mathbf{b}$. Considerăm o perturbație \mathbf{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 perturbația soluției prin $\mathbf{e} = \tilde{\mathbf{x}} - \mathbf{x}$, și avem

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

în care:

$$\text{Cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|.$$

Numărul $\text{Cond}(\mathbf{A})$ este numărul de condiție al matricii \mathbf{A} . Avem $\text{Cond}(\mathbf{A}) \geq 1$.

- Dacă $\text{Cond}(\mathbf{A}) \sim 1$: \mathbf{A} se zice *bine-condiționată*;

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

- Dacă $\text{Cond}(\mathbf{A}) \gg 1$: \mathbf{A} se zice *rău-condiționată*;

$\|\mathbf{e}\| / \|\mathbf{x}\|$ poate fi "mare", adică $\|\mathbf{e}\| / \|\mathbf{x}\| \gg \|\mathbf{r}\| / \|\mathbf{b}\|$.

Numărul de condiție depinde de norma considerată. În particular, se definește

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

în care: $\rho(\mathbf{A})$ este raza spectrală a matricii \mathbf{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 \mathbf{A} . Avem, pentru orice normă,

$\text{Cond}(\mathbf{A}) \geq \text{Cond}(\mathbf{A})_*$. Numărul $\text{Cond}(\mathbf{A})_*$ se calculează prin

$$\text{Cond}(\mathbf{A})_* = \frac{\max_{\lambda \in \sigma(\mathbf{A})} |\lambda|}{\min_{\lambda \in \sigma(\mathbf{A})} |\lambda|}.$$

Programul calculează numărul de condiție pentru normele $\|\cdot\|_\infty$ și $\|\cdot\|_1$, definite prin:

$$\|\mathbf{A}\|_\infty = \max_i \sum_j |a_{ij}|$$

Aceasta se zice *norma liniilor*, și este maximul sumelor modulelor elementelor, pe linii.

$$\|\mathbf{A}\|_1 = \max_j \sum_i |a_{ij}|$$

Aceasta se zice *norma coloanelor*, și este maximul sumelor modulelor elementelor, pe coloane.

Fișiere:

Main-Numar_Conditie.f90; NormInf.f90; Norm1.f90; Elim2004.f90; width.f90;

Main-Numar_Conditie.f90: programul principal

NormInf.f90, Norm1.f90: norma- ∞ și norma-1, ale unei matrici

Elim2004.f90: eliminarea Gauss; aceasta servește pentru calculul matricii inverse.

width.f90: utilitar

Subrutina de eliminare Gauss este ELIM2 (n, nb, A, B, prag, kod). Parametrii au aceleași semnificații ca la ELIM din Gauss (cu deosebirea că, între parametri, nu mai apare codul kodPrint).

Datele de intrare se citesc dintr-un fișier de date, cu structura de mai jos. Datele de ieșire se scriu în fișier, după datele de intrare.

Fișier de date (intrare/ieșire):

Titlu: text, max. 80 caractere

n: ordinul matricii \mathbf{A}

Matricea \mathbf{A} : $n \times n$; se scrie așa cum e dată - pe linii și coloane

Cod_Inversare_Coeficienți, Cod_Tipărire: 0 / \neq 0.

- *Cod_Inversare_Coeficienti* $\neq 0$: se face $a(i, j) = 1./a(i, j)$
- *Cod_Tiparire* $\neq 0$: se tipărește și \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:

$$\mathbf{H}_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \dots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \dots & \frac{1}{n+1} \\ \vdots & & & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \dots & \frac{1}{2n-1} \end{bmatrix}$$

Matricea Hilbert este cunoscută ca fiind rău-condiționată, și anume, cu atât mai rău-condiționată cu cât ordinul n este mai mare.

Inversarea se face în modul obișnuit, 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 .

Fișiere:

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 , și termenii liberi pentru ordinul n (coloanele matricii unitate).

(În loc de, specificarea acestor date în fișierul 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 aceiași ca la subrutina ELIM din Gauss (cu deosebirea că între parametri nu mai figurează codul kodPrint).

Datele de intrare se introduc de la terminal și constau în: ordinul n ; o dată logică (T/F) pentru verificarea sau nu, a inversei calculate \mathbf{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-condiționată, 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 creșterea 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 ecuație $i = 1, 2, 3$, în raport cu necunoscuta x_i , căutând ca aceasta să fie necunoscuta cu coeficientul cel mai mare din ecuație, eventual rearanjând ecuațiile. Intervertind ecuațiile 2 și 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:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 4/7 \\ 12/9 \end{bmatrix} + \begin{bmatrix} 0 & -1/8 & 1/8 \\ 1/7 & 0 & 2/7 \\ -2/9 & -1/9 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (1')$$

care este de forma

$$\mathbf{x}^{(m+1)} = \mathbf{g} + \mathbf{M}\mathbf{x}^{(m)}$$

Se iterează, cu aproximația inițială $\mathbf{x}^{(0)} = [1 \quad 4/7 \quad 12/9]^T$; testul de oprire a iterației este $\|\mathbf{x}^{(m+1)} - \mathbf{x}^{(m)}\|_\infty \leq 1E-6$. La iterația 12 rezultă soluția (1.0, 1.0, 1.0).

■

Metoda:

Fie sistemul dat $\mathbf{Ax} = \mathbf{b}$. Se rezolvă fiecare ecuație i în raport cu necunoscuta x_i .

Explicitând x_i se obține:

$$x_i = (b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j) / a_{ii}, \quad i = 1, 2, \dots, n$$

S-a presupus $a_{ii} \neq 0$. Iterația Jacobi va fi:

$$\begin{cases} \mathbf{x}^{(0)} = \text{arbitrar} \\ x_i^{(m+1)} = (b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j^{(m)}) / a_{ii}, \quad i = 1, 2, \dots, n; \quad m \geq 0 \end{cases}$$

Testul de oprire a iterației este $\|\mathbf{x}^{(m+1)} - \mathbf{x}^{(m)}\|_\infty \leq \text{eps}$.

Metoda Jacobi se mai zice metoda “iterațiilor simultane”, pentru că, coordonatele x_i ale soluției \mathbf{x} se calculează independent unele de altele. (Pentru alt mod de calcul – v. metoda Gauss_Seidel, mai jos.)

Condiție suficientă de convergență:

Matricea \mathbf{A} este *diagonal dominantă*, adică avem:

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|, \quad i = \overline{1, n}$$

Pentru alte considerații privind metoda, v. Cap. 4-IV.

Fișiere:

Jacobi.f90: programul principal – care implementează metoda

Norm.f90: norma- ∞ a unui vector

width.f90: utilitar

Datele de intrare se citesc dintr-un fișier de date. Acesta are structura:

- *Titlu*: linie-text, max. 80 caractere.
- *n*: ordinul matricii
- *Matricea A*: $n \times n$; se scrie pe linii și coloane – așa cum este dată.
- *Termenul liber b*: se scriu în linie, n valori b_i .
- *eps, lmit*: toleranța, numărul limită de iterații.
- *kodPrint*: cod de tipărire iterații: 0 / \neq 0: Da / Nu.
- *check_text*: text de max. 20 caractere. Primul caracter este semnificativ:
 C, c, P, p : se face proba – se tipărește vectorul $\mathbf{Ax} - \mathbf{b}$.

Datele de ieșire se scriu în fișier, după datele de intrare – v. mai jos.

Exemplul numeric de [mai sus](#):

Fișierul de intrare/ieșire (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 $x_1^{(m)}, \dots, x_{i-1}^{(m)}$ deja calculate, care în general, sunt aproximații mai bune ale soluției.

Exemplu:

Considerăm [exemplul numeric](#) din Jacobi. Ecuațiile se înlocuiesc cu cele de mai jos, care ar trebui să ofere o convergență 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 $\mathbf{x}^{(0)} = [0 \ 0 \ 0]^T$ și aceeași toleranță $1E-6$, soluția (1.0, 1.0, 1.0) se obține la iterația 9. Rezidualul maxim este 0.0.

■

Metoda:

Formulele generale ale metodei Gauss-Seidel sunt:

$$\begin{cases} \mathbf{x}^{(0)} = \text{arbitrar} \\ x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(m)}) / a_{ii}, \quad i = 1, 2, \dots, n; \quad m \geq 0 \end{cases}$$

Testul de oprire a iterației este $\|\mathbf{x}^{(m+1)} - \mathbf{x}^{(m)}\|_{\infty} \leq \text{eps}$.

Metoda Gauss-Seidel se mai zice metoda “iterațiilor succesive”, pentru că la un pas $m+1$, $m \geq 0$, de îndată ce o coordonată $x_j^{(m+1)}$ a soluției este calculată, ea se utilizează în ecuațiile pentru coordonatele următoare $x_i^{(m+1)}$, $i > j$.

Condiții suficiente de convergență:

- matricea \mathbf{A} este *diagonal dominantă*.
- matricea \mathbf{A} este *simetrică și pozitiv definită*.

Când ambele metode Jacobi și Gauss-Seidel converg, metoda Gauss-Seidel converge mai rapid.

Pentru alte considerații privind convergența, v. Cap. 4-IV.

Fișiere:

Gauss_Seidel.f90: programul principal

norm.f90: norma- ∞ a unui vector

width.f90: utilitar

Datele de intrare se citesc dintr-un fișier de date. Acesta are aceeași structură ca și [fișierul 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":

$$1 \quad 0.000000$$

$$2 \quad 0.000000$$

$$3 \quad 0.000000$$

■

Sor

Metoda relaxării (Successive Over-Relaxation).

Metoda:

Reluăm formula metodei [Gauss-Seidel](#):

$$x_i^{(m+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(m)}) / a_{ii}, \quad i = 1, 2, \dots, n; \quad m \geq 0$$

Formula se pune sub forma următoare, adunând și scăzând $\mathbf{x}_i^{(m)}$ în membrul doi (observați că acum, în a doua sumă, indicele j ia valori de la i și 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 diferența $(\mathbf{x}_i^{m+1} - \mathbf{x}_i^m)$. Metoda SOR constă în înmulțirea acestei diferențe 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 $z_i^{(m+1)}$ (aceasta este coordonata i a iteratei $(m+1)$ din metoda Gauss-Seidel.) .

Formula de iterare este echivalentă cu următoarele ecuații 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 relația

$$1 < \omega < 2.$$

În practică, se poate alege ω , astfel: se utilizează valori ω de test, pe un număr limitat de iterații; se alege ca valoare optimă acel ω , pentru care convergența este cea mai rapidă.

Fișiere:

SOR.f90: programul principal

norm.f90: norma- ∞ a unui vector

width.f90: utilitar

Datele de intrare se citesc dintr-un fișier de date. Acesta are aceeași structură ca și fișierul 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 iterații și dă soluția $\mathbf{x}^{(27)} = [0.9999999 \ 0.9999999 \ 0.9999999 \ 0.9999999 \ 0.9999999]^T$. Alegem $\omega = 1.3$ (valoarea optimă). Se obțin rezultatele de mai jos.

Fișierul de intrare/ieșire:

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 ȘI VECTORI PROPRII

Folder tematic: \$ Eigen

Foldere:

Power; Power_Complex; Inverse_Power_Shift; Inverse_Power_Complex;

Sim_Iter: metoda puterii și variante ale acesteia.

Jacobi; QR: metodele Jacobi și QR.

Problema de valori proprii (sumar)

Fie dată matricea \mathbf{A} $n \times n$, reală sau complexă.

Dacă vectorul $\mathbf{x} \neq \mathbf{0}$ și scalarul λ satisfac relația

$$\mathbf{Ax} = \lambda\mathbf{x}, \quad (1)$$

atunci: λ se numește *valoare proprie*, iar \mathbf{x} *vectorul propriu* asociat lui λ .

Dacă \mathbf{x} și \mathbf{y} sunt asociați cu λ , atunci și $\alpha\mathbf{x} + \beta\mathbf{y}$, 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 aceleași valori proprii. Dacă \mathbf{x}' sunt vectorii proprii ai lui \mathbf{A}' ,

vectorii proprii ai lui \mathbf{A} se găsesc din relația

$$\mathbf{x} = \mathbf{Px}'.$$

Mai multe metode numerice transformă matricea \mathbf{A} într-o matrice similară \mathbf{A}' , de o formă mai simplă, determinând valorile proprii și vectorii proprii ai matricii \mathbf{A}' ; apoi, se determină vectorii proprii ai lui \mathbf{A} , cu relația de mai sus.

Polinomul caracteristic:

Relația (1) este ecuația din care se găsesc λ și \mathbf{x} . Aceasta se mai scrie

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \quad (2)$$

unde \mathbf{I} este matricea unitate. Explicit,

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2')$$

Condiția ca (2) să aibă soluții netriviiale \mathbf{x} , este $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. Explicit,

$$p(\lambda) = \begin{vmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{vmatrix} = 0 \quad (3)$$

$p(\lambda)$ se zice *polinomul caracteristic* al matricii \mathbf{A} , și $p(\lambda) = 0$ – ecuația caracteristică.

Fie determinantul dezvoltat:

$$p(\lambda) = (-1)^n \lambda^n + c_1 \lambda^{n-1} + \dots + c_{n-1} \lambda + c_n$$

Proprietăți ale coeficienților și valorilor proprii:

$$c_n = \det(\mathbf{A}); \quad \lambda_1 \lambda_2 \dots \lambda_n = \det(\mathbf{A}).$$

$$c_1 = (-1)^{n-1} \sum_1^n a_{ii}; \quad \lambda_1 + \lambda_2 + \dots + \lambda_n = \sum_1^n a_{ii}.$$

Notă: $Tr(\mathbf{A}) = \sum_1^n a_{ii}$ se zice urma matricii \mathbf{A} .

În general:

$$c_i = (-1)^{n-i} \times \text{Suma minorilor principali de ordin } i \text{ ai matricii } \mathbf{A}.$$

Observații

1. Ordonarea valorilor proprii:

Valorile proprii se ordonează în șirul $\lambda_1, \lambda_2, \dots, \lambda_n$. În acest șir, o rădăcină multiplă de ordinul r , se repetă de r ori. Uzual, valorile proprii se indexează în ordinea descrescătoare a modului, adică $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$. Valoarea proprie λ_1 se zice *dominantă*. Mulțimea valorilor proprii $\{\lambda_i, i = \overline{1, n}\}$ se zice *spectrul* matricii \mathbf{A} .

2. Vectorii proprii asociați unei valori proprii:

După determinarea valorilor proprii, vectorii proprii asociați cu λ_i se găsesc punând $\lambda = \lambda_i$ în (2'), și rezolvând sistemul liniar și omogen (2').

- Dacă \mathbf{A} este reală: În general, λ_i poate fi complex, și atunci vectorul propriu \mathbf{x}_i asociat cu λ_i este complex. Dacă \mathbf{A} și λ_i sunt reali, atunci vectorul propriu \mathbf{x}_i este real.
- Dacă \mathbf{A} este complexă: În general, valorile proprii și vectorii proprii sunt mărimi complexe. În particular, unele dintre acestea pot fi și reale ■

Subspațiul propriu și dimensiunea acestuia:

Sistemul (2') este omogen, astfel că dacă \mathbf{x} , \mathbf{y} sunt soluții, atunci $\alpha\mathbf{x} + \beta\mathbf{y}$ sunt soluții. Adică, lui λ_i îi este asociat un subspațiu liniar S_i de soluții \mathbf{x} . Se arată că:

Dimensiunea subspațiului 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 , și $p_i =$ dimensiunea subspațiului S_i , avem: $p_i \leq r_i$. Cu alte cuvinte, există cel mult r_i vectori liniar independenți în S_i .

Dacă $p_i < r_i$, valoarea λ_i se zice *defectivă*; în acest caz, și matricea \mathbf{A} se zice *defectivă*.

(r_i se mai zice *multiplicitate algebrică*, iar p_i *multiplicitate geometrică*.)

3. Determinarea efectivă a sistemului propriu:

Pentru calculația practică ($n > 3$), *nu se recomandă*:

- *Rezolvarea directă* a ecuației caracteristice. Aceasta, datorită faptului că problema calculului rădăcinilor unui polinom este foarte sensibilă la mici perturbații în coeficienți; aceste perturbații apar din erorile de rotunjire.
- *Calculul direct al vectorilor proprii*, din (2').

Metode numerice pentru găsirea valorilor proprii λ_i , și a vectorilor proprii asociați $\mathbf{x}^{(i)}$, sunt prezentate în continuare.

■

Matrici hermitiene și unitare

Operația * aplicată unui vector sau unei matrici notează *transpus-conjugata*.

Dacă \mathbf{x} este un vector, respectiv \mathbf{A} o matrice, atunci:

$$\mathbf{x}^* = \bar{\mathbf{x}}^T; \quad \mathbf{A}^* = \overline{\mathbf{A}}^T.$$

Explicit: $\mathbf{A} = [a_{ij}]$, $\mathbf{A}^* = [a_{ij}^*]$, avem: $a_{ij}^* = \bar{a}_{ji}$.

În expresiile anterioare, bara notează conjugata: $\bar{\mathbf{x}} = [\bar{x}_i]$; $\overline{\mathbf{A}} = [\bar{a}_{ij}]$.

Pentru un vector real, sau o matrice reală, operația * revine la transpunere:

$$\mathbf{x}^* = \mathbf{x}^T; \quad \mathbf{A}^* = \mathbf{A}^T.$$

În particular, pentru un scalar, operația revine la conjugare: $s^* = \bar{s}$.

Matricea \mathbf{A} se zice *hermitiană*, dacă $\mathbf{A}^* = \mathbf{A}$.

Exemplu de matrice hermitiană:

$$\mathbf{A} = \begin{bmatrix} 1 & 1-7i & -i \\ 1+7i & 5 & 10-3i \\ i & 10+3i & -2 \end{bmatrix}$$

Remarcați 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: $\forall \mathbf{x} \neq \mathbf{0} \quad \mathbf{x}^T \mathbf{A} \mathbf{x} > 0$.

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.

Proprietăți ale matricilor hermitiene (în particular, reale și simetrice):

P1. Dacă \mathbf{A} este hermitiană, și are valorile proprii $\{\lambda_i\}$ distincte sau nu, atunci:

(a) Există o matrice unitară \mathbf{U} , astfel că $\mathbf{U}^* \mathbf{A} \mathbf{U}$ este diagonală (se zice că \mathbf{U} diagonalizează pe \mathbf{A}), și avem:

$$\mathbf{U}^* \mathbf{A} \mathbf{U} = \text{diag}(\lambda_1, \dots, \lambda_n).$$

(b) Există n vectorii proprii liniar independenți care formează o bază ortonormată în \mathbf{C}^n (aceștia sunt coloanele lui \mathbf{U});

(c) Valorile proprii sunt reale

■

În particular, pentru \mathbf{A} reală și simetrică:

(a-1) Există \mathbf{U} unitară și reală, astfel că $\mathbf{U}^T \mathbf{A} \mathbf{U} = \text{diag}(\lambda_1, \dots, \lambda_n)$.

(b-1) Există n vectorii proprii liniar independenți; aceștia formează o bază ortonormată în \mathbf{R}^n (coloanele lui \mathbf{U});

(c-1) Valorile proprii sunt reale, și vectorii proprii sunt reali ■

Avem și proprietatea:

P1'. Dacă \mathbf{A} este hermitiană (reală și simetrică) și *pozitiv definită*, atunci valorile proprii ale lui \mathbf{A} sunt reale și *pozitive* ■

Produs scalar și proprietăți de ortogonalitate:

1. Spațiu vectorial real V_n

Fie x un vector din V_n , și \mathbf{x} matricea coloană a coordonatelor sale într-o bază a lui

V_n ($\mathbf{x} \in \mathbf{R}^n$).

Dacă matricea \mathbf{A} reală, este *simetrică și pozitiv definită*, se definește produsul scalar în raport cu matricea \mathbf{A} , prin:

$$\langle x, y \rangle_{\mathbf{A}} = \mathbf{x}^T \mathbf{A} \mathbf{y} = \mathbf{y}^T \mathbf{A} \mathbf{x}$$

În particular, pentru $\mathbf{A} = \mathbf{I}$, acesta devine produsul scalar standard:

$$\langle x, y \rangle = \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$$

$$\text{Avem: } \langle y, x \rangle_A = \langle x, y \rangle_A; \quad \langle y, x \rangle = \langle x, y \rangle.$$

Vectorii x și y se zic *ortogonali* (relativ la produsul scalar), dacă $\langle x, y \rangle_A = 0$,

$$\text{sau } \langle x, y \rangle = 0 \quad \blacksquare$$

Astfel:

- Vectorii x și y sunt *ortogonali relativ la matricea \mathbf{A}* , dacă

$$\mathbf{x}^T \mathbf{A} \mathbf{y} = 0, \text{ sau } \mathbf{y}^T \mathbf{A} \mathbf{x} = 0.$$

- Vectorii sunt *ortogonali*, dacă $\mathbf{x}^T \mathbf{y} = 0$, sau $\mathbf{y}^T \mathbf{x} = 0$.

2. Spațiu vectorial complex V_n

Fie x un vector din V_n , și \mathbf{x} matricea coloană a coordonatelor într-o bază a lui V_n

$$(\mathbf{x} \in \mathbf{C}^n).$$

Dacă \mathbf{A} este o matrice *hermitiană și pozitiv definită*, se definește produsul scalar în raport cu matricea \mathbf{A} , prin:

$$\langle x, y \rangle_A = \mathbf{x}^T \mathbf{A} \bar{\mathbf{y}} = \mathbf{y}^* \mathbf{A}^T \mathbf{x}.$$

(Ultima egalitate rezultă din aceea că scalarul $s = \langle x, y \rangle_A$ este egal cu transpusul său).

Produsul scalar definit de $\mathbf{A} = \mathbf{I}$ este:

$$\langle x, y \rangle = \mathbf{x}^T \bar{\mathbf{y}} = \mathbf{y}^* \mathbf{x}$$

$$\text{Avem: } \langle y, x \rangle_A = \overline{\langle x, y \rangle_A}, \text{ și } \langle y, x \rangle = \overline{\langle x, y \rangle}.$$

Ortogonalitatea a doi vectori se definește ca înainte, prin condiția $\langle x, y \rangle_A = 0$,

$$\text{sau } \langle x, y \rangle = 0 \quad \blacksquare$$

Observație: Dacă avem $\langle x, y \rangle_A = 0$, avem și $\langle y, x \rangle_A = 0 \quad \blacksquare$

Astfel:

- Vectorii \mathbf{x} , \mathbf{y} sunt *ortogonali în raport cu \mathbf{A}* , dacă:

$$\mathbf{x}^T \mathbf{A} \bar{\mathbf{y}} = 0, \text{ sau } \mathbf{y}^* \mathbf{A}^T \mathbf{x} = 0.$$

- Vectorii \mathbf{x} , \mathbf{y} sunt *ortogonali*, dacă:

$$\mathbf{x}^T \bar{\mathbf{y}} = 0, \text{ sau } \mathbf{y}^* \mathbf{x} = 0.$$

P2. Dacă \mathbf{A} este hermitiană, atunci:

- Vectorii proprii asociați la două valori proprii distincte sunt *ortogonali*:
dacă $\lambda_1 \neq \lambda_2$, atunci $\mathbf{x}_2^* \mathbf{x}_1 = 0$ și $\mathbf{x}_1^* \mathbf{x}_2 = 0$.
- Avem și: $\mathbf{x}_2^* \mathbf{A} \mathbf{x}_1 = 0$, $\mathbf{x}_1^* \mathbf{A} \mathbf{x}_2 = 0$.

(Dacă \mathbf{A} hermitiană este și pozitiv definită, vectorii \mathbf{x}_1 , \mathbf{x}_2 sunt *ortogonali* relativ la matricea \mathbf{A}^T .)

■

Dacă \mathbf{A} este reală și simetrică:

- Vectorii proprii asociați la două valori proprii *distincte* sunt *ortogonali*:
 $\mathbf{x}_2^T \mathbf{x}_1 = 0$, $\mathbf{x}_1^T \mathbf{x}_2 = 0$.
- Avem și: $\mathbf{x}_2^T \mathbf{A} \mathbf{x}_1 = 0$, $\mathbf{x}_1^T \mathbf{A} \mathbf{x}_2 = 0$.

(Dacă \mathbf{A} reală și simetrică, este și pozitiv definită, vectorii \mathbf{x}_1 , \mathbf{x}_2 sunt *ortogonali* relativ la matricea \mathbf{A})

■

Câțul Rayleigh

Fie \mathbf{A} o matrice $n \times n$ complexă (sau reală). Fie $\mathbf{v} \in \mathbf{C}^n$ un vector arbitrar, și definim

$$\rho(\mathbf{v}) = \frac{\mathbf{v}^* \mathbf{A} \mathbf{v}}{\mathbf{v}^* \mathbf{v}}$$

$\rho(\mathbf{v})$ se numește *câțul Rayleigh*.

Proprietate-1: Dacă \mathbf{x} este vector propriu asociat cu λ , atunci $\rho(\mathbf{x}) = \lambda$ ■

Astfel, câțul Rayleigh poate fi utilizat pentru a găsi o aproximare a valorii proprii λ , dacă se cunoaște o aproximare \mathbf{v} a vectorului propriu \mathbf{x} : $\mathbf{v} \approx \mathbf{x} \Rightarrow \lambda \approx \rho(\mathbf{v})$.

Proprietate-2: Dacă matricea este hermitiană, câțul Rayleigh este mărginit de valorile proprii extreme ■

Valorile proprii sunt reale; fie acestea $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$, atunci, avem:

$$\lambda_n \leq \rho(\mathbf{v}) \leq \lambda_1, \text{ pentru orice } \mathbf{v} \in \mathbf{C}^n.$$

■

Power

Metoda puterii – matrice reală, cu valori proprii reale.

Metoda:

Metoda puterii determină valoarea proprie dominantă și vectorul propriu asociat, ale unei matrici \mathbf{A} , reale sau complexe. Aplicată la inversa \mathbf{A}^{-1} metoda determină valoarea proprie cea mai mică (în modul), și vectorul propriu asociat cu aceasta. Se presupune că:

1. Există un set de n vectori proprii *liniar independenți*.
2. Există o *singură* valoare proprie dominantă $\lambda_1 : |\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$

Metoda este următoarea:

Fie $\mathbf{w}^{(0)}$ un vector inițial, ales arbitrar – cu singura condiție ca să aibă o componentă în direcția lui $\mathbf{x}^{(1)}$. Pentru a împlini această cerință, unele coduri generează un vector aleator. (În fapt, $\mathbf{w}^{(0)}$ este o aproximație a vectorului propriu $\mathbf{x}^{(1)}$; dacă o astfel de aproximație este cunoscută, atunci aceasta accelerează iterația de mai jos).

Desvoltăm $\mathbf{w}^{(0)}$ în baza vectorilor proprii:

$$\mathbf{w}^{(0)} = a_1 \mathbf{x}^{(1)} + a_2 \mathbf{x}^{(2)} + \dots + a_n \mathbf{x}^{(n)} \quad (\text{a})$$

Presupunem că $a_1 \neq 0$, și formăm șirul de vectori

$$\mathbf{w}^{(k+1)} = \mathbf{A} \mathbf{w}^{(k)} = \mathbf{A}^{k+1} \mathbf{w}^{(0)}, \quad k \geq 0$$

Avem:

$$\begin{aligned} \mathbf{w}^{(1)} &= \mathbf{A} \mathbf{w}^{(0)} = \lambda_1 a_1 \mathbf{x}^{(1)} + \lambda_2 a_2 \mathbf{x}^{(2)} + \dots + \lambda_n a_n \mathbf{x}^{(n)} \\ &= \lambda_1 \left[a_1 \mathbf{x}^{(1)} + \left(\frac{\lambda_2}{\lambda_1} \right) a_2 \mathbf{x}^{(2)} + \dots + \left(\frac{\lambda_n}{\lambda_1} \right) a_n \mathbf{x}^{(n)} \right] \end{aligned}$$

În general,

$$\mathbf{w}^{(k)} = \mathbf{A}^k \mathbf{w}^{(0)} = (\lambda_1)^k \left[a_1 \mathbf{x}^{(1)} + \left(\frac{\lambda_2}{\lambda_1} \right)^k a_2 \mathbf{x}^{(2)} + \dots + \left(\frac{\lambda_n}{\lambda_1} \right)^k a_n \mathbf{x}^{(n)} \right] \quad (\text{b})$$

Cum $|\lambda_1| > |\lambda_i|$ pentru $i \geq 2$, urmează că rapoartele $(\lambda_i / \lambda_1)^k$ tind la 0 pentru $k \rightarrow \infty$. Astfel, pentru k crescător, vectorii $\mathbf{w}^{(k)}$ se aliniază din ce în ce mai mult la direcția vectorului propriu $\mathbf{x}^{(1)}$. În consecință, pentru un k suficient de mare, avem $\mathbf{w}^{(k)} \approx (\lambda_1)^k a_1 \mathbf{x}^{(1)}$. Să considerăm de asemenea relația $\mathbf{w}^{(k+1)} \approx (\lambda_1)^{k+1} a_1 \mathbf{x}^{(1)}$. Luând orice coordonată non-zero a lui $\mathbf{w}^{(k+1)}$, $\mathbf{w}^{(k)}$, să zicem cea de-a m -a coordonată, obținem

$$\lambda_1 \approx \frac{w_m^{(k+1)}}{w_m^{(k)}}$$

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 creșterea lui k . Aceasta se evită prin normalizarea (sau scalarea) lui $\mathbf{w}^{(k)}$, la fiecare pas k . Normele utilizate sunt norma- ∞ și norma-2.

Astfel, algoritmul metodei este:

$\mathbf{w}^{(0)}$ = Vector inițial

$$\mathbf{z}^{(k)} = \frac{\mathbf{w}^{(k)}}{\|\mathbf{w}^{(k)}\|}, \text{ pentru } k \geq 0 \dots \text{ Normalizare}$$

$$\mathbf{w}^{(k+1)} = \mathbf{A}\mathbf{z}^{(k)}, \text{ pentru } k \geq 0 \dots \text{ Iterație}$$

Teste de oprire a iterației:

1. Test de coliniaritate a doi vectori succesivi:

Vectorii $\mathbf{z}^{(k)}$ și $\mathbf{z}^{(k+1)}$ sunt coliniari, dacă rapoartele $\rho_i = z_i^{(k+1)} / z_i^{(k)}$, $z_i^{(k)} \neq 0$ sunt egale (coordonatele nenule sunt proporționale), sau $z_i^{(k+1)} = z_i^{(k)} = 0$. În calculația practică, punem condiția $|\rho(i) - \rho(i_0)| \leq TOL$, dacă $|z_i^{(k)}| > TOL$; sau, să avem simultan, $|z_i^{(k)}| \leq TOL$ și $|z_i^{(k+1)}| \leq TOL$. TOL este o toleranță

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 **colin**(1 : n), definit astfel:

$$colin(i) = \begin{cases} z^{(k+1)}(imax) / z^{(k)}(imax) & \dots \text{daca } |z^{(k+1)}(i)| \leq TOL \text{ si } |z^{(k)}(i)| \leq TOL \\ z^{(k+1)}(i) / z^{(k)}(i) & \dots \text{altfel} \end{cases}$$

$$i = \overline{1, n}$$

Testul este:

$$\| \mathbf{colin} - \mathbf{colin}(imax) \| \leq TOL$$

Observație

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, și testul de coliniaritate poate lua forma

$$\| \mathbf{z}^{(k+1)} - \mathbf{z}^{(k)} \|_2 \leq TOL.$$

O problemă specială apare pentru \mathbf{A} reală, dacă valoarea proprie dominantă este reală și negativă, $\lambda_1 < 0$, și anume: din

$$\mathbf{w}^{(k)} = \mathbf{A}^{(k)} \mathbf{z}^{(0)} = (\lambda_1)^k [a_1 \mathbf{x}^{(1)} + \mathbf{r}^{(k)}] \text{ și } \mathbf{z}^{(k)} = \mathbf{w}^{(k)} / \| \mathbf{w}^{(k)} \|, \text{ rezultă că}$$

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 țină cont de aceasta. Astfel, definim,

$$is = \text{sign}(1., \lambda_1^{(k+1)}), \quad \mathbf{dz} = \mathbf{z}^{(k+1)} - is * \mathbf{z}^{(k)}$$

Testul corect este

$$\| \mathbf{dz} \|_2 \leq TOL.$$

■

2. Test asupra lui λ :

Iterația se oprește prin condiția

$$| \lambda_1^{(k+1)} - \lambda_1^{(k)} | \leq TOL$$

unde TOL este o toleranță.

3. Test de satisfacere a relației de definiție:

$$\| \mathbf{Az} - \lambda \mathbf{z} \| \leq TOL$$

Practic, $\mathbf{z} = \mathbf{z}^{(k)}$, $\mathbf{Az}^{(k)} = \mathbf{w}^{(k+1)}$, și $\lambda = \lambda^{(k+1)}$. Definind

$$\mathbf{dif} = \mathbf{w}^{(k+1)} - \lambda^{(k+1)} \mathbf{z}^{(k)},$$

se pune testul

$$\| \mathbf{dif} \| \leq TOL$$

Observații:

- În cod, dacă valorile anterioare (k) sunt stocate în $z0$ și $lambda0$, iar valorile curente ($k+1$) în z și $lambda$, definiția 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 esență, metoda determină vectorul propriu nr. 1, și apoi determină λ_1 din acesta. Cu Testul 1, se obțin vectori proprii mai preciși decât cu Testul 2 (la aceeași toleranță).

Testul 3 nu este specific metodei puterii, ci poate fi aplicat oricărei metode iterative pentru valori și vectori proprii. Codurile din Lapack (Bai et al.(2000)), utilizează Testul 3, cu $TOL = \varepsilon_M |\lambda|$, unde ε_M este ε -mașină.

În metodele următoare, Testul 3 va fi utilizat numai la verificarea sistemului propriu. Excepție face metoda puterii, unde, pentru studiu, se poate alege unul din Testele 1-3. (Alegerea testului se face printr-un cod.)

■

Convergența:

Convergența $\lambda_1^{(k)} \rightarrow \lambda_1$ este liniară, iar rata convergenței este aproximativ $|\lambda_2 / \lambda_1|$.

(Acest rezultat are loc în ipoteza metodei $|\lambda_1| > |\lambda_2|$, cu ipoteza suplimentară: pentru $k \geq k_1$, cantitățile $(\lambda_i / \lambda_2)^k$, $i \geq 3$, sunt neglijabile în raport cu $(\lambda_2 / \lambda_1)^k$.)

Fișiere:

Power-2.f90: subrutina metodei

Main-Power.f90: programul principal

Agen.f90: subrutina de generare a matricii $A(n, n)$.

Maxindex.f90: funcție; indicele coordonatei de modul maxim din vectorul $v(n)$.

vNorm.f90: norma- ∞ și norma-2 a unui vector.

w0gen.f90: subrutina de generare a lui w_0

work.f90: modul

width.f90: utilitar

Subrutina metodei este

Power(iter, kodTest, ikod, kodNorm, iter_flag, test_val)

Parametrii:

iter: Numărul maxim de iterații (intrare) / Numărul efectiv de iterații (ieșire)

kodTest: cod pentru testul de oprire a iterației

ikod: cod de tipărire a iterațiilor

kodNorm: codul normei

iter_flag: cod de încheiere a iterației (intern).

test_val: valoarea de test.

Notă:

TOL, lmit și 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 w_0 .

Pentru a împlini cerința ca $\mathbf{w}^{(0)}$ să aibă o componentă în direcția lui $\mathbf{x}^{(1)}$, unele coduri generează $\mathbf{w}^{(0)}$ ca vector aleator.

Datele de intrare se citesc dintr-un fișier de date (specificat). Datele de ieșire 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 așa cum e dată - pe linii.]
- [*Vectorul w0*: dacă *kodw0* = 0.]
- *TOL*, *lnit*: toleranța, numărul limită d iterații.
- *kodTest*, *ikod*: cod test, cod tipărire iterații
- *kodNorm*: cod normă
- *Check_text*: cod de verificare.

Semnificația codurilor:

n: ordinul matricii

kodA: 0 / ≠ 0: citește **A** / generează **A** (cu subrutina *Agen*)

kodw0: 0 / ≠ 0: citește **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: Tipărește iterațiile: 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 tipărește 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_lambda	Current_vector
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](#).

Convergența:

Ipotezele sunt aceleași ca la [Power](#). În particular, nu se obține convergență, pentru o matrice reală la care $|\lambda_1| = |\lambda_2|$ – de exemplu, când λ_1, λ_2 sunt complex conjugate.

Fișiere:

Agen_complex.f90; Main-Power_complex.f90; Maxindex_z.f90;
 Power_Complex&Eps.f90; vNorm.f90; width.f90; work_complex.f90;

Semnificația fișierelor și rutinelor – aceleași ca la [Power](#).

Subrutina metodei:

Power_Complex(iter, kodTest, ikod, kodNorm, iter_flag, test_val, test_val_min)

Semnificația parametrilor – ca la [Power](#). Parametrul test_val_min returnează valoarea minimă de test, în cazul când nu este atinsă toleranța *TOL*.

Datele de intrare – sunt [datele la Power](#), cu deosebirile:

- **A** și **w0**: sunt date complexe; pentru **A** și **w0** reale, se introduc date reale.
- *ikod*: Pentru *ikod* = 2: se tipăresc și valorile $eps1 = \lambda / |\lambda|$, și $eps1 * z0$, unde $z0$ este $z_{anterior}$. (Aceste valori servesc la studiul convergenței: 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 translație (shift) – matrice reală, cu valori proprii reale.

Metoda:

Fie matricea \mathbf{A} , cu valorile proprii λ_j , $j = 1, n$. Metoda găsește valoarea proprie λ_i a lui \mathbf{A} , cea mai apropiată de un număr dat s . Se consideră matricea

$$\mathbf{B} = \mathbf{A} - s\mathbf{I}$$

și presupunem că \mathbf{B} este nesingulară. Se verifică imediat că valorile proprii ale lui \mathbf{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 \mathbf{B} , adică:

$$0 < |\mu_i| < \varepsilon < |\mu_j|, \text{ pentru } j \neq i.$$

Atunci, metoda puterii aplicată lui \mathbf{B}^{-1} produce valoarea proprie de modul maxim, adică $\nu_i = 1/\mu_i$. Avem $\mu_i = 1/\nu_i$, și

$$\lambda_i = \frac{1}{\nu_i} + s.$$

Principala aplicație a metodei este de a găsi vectorul propriu, dacă este cunoscută o aproximație bună a valorii proprii, să zicem $\hat{\lambda}_i$. (Aceasta poate fi furnizată de o metodă în care se determină numai valorile proprii – nu și 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, și se obține o bună aproximație a vectorului propriu $\mathbf{x}^{(i)}$.

Metoda iterației inverse cu deplasare, este una dintre cele mai precise metode pentru calculul vectorilor proprii.

Algoritmul practic, este următorul:

Iterația 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 \mathbf{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, și sistemul se rezolvă succesiv cu membrii dreپți $\mathbf{z}^{(k)}$, $k \geq 0$.

Notă: Pentru testare, Inverse_Power_Shift oferă și varianta iterării directe cu \mathbf{B}^{-1} ■

Fișiere:

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:
semnificațiile de la descompunerea LU.

Agen.f90; Maxindex.f90; w0gen.f90; width.f90; work.f90: aceleași semnificații 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 iterații (intrare) / Numărul efectiv de iterații (ieșire). Pentru ceilalți parametri, v. codurile de mai jos.

Datele de intrare se citesc dintr-un fișier de date (specificat). În acesta se scriu și datele de ieșire.

Datele de intrare sunt:

- *Titlu*: text de max. 80 caractere
- *n*, *kodA*, *kodw0*
- [*Matricea A*: dacă *kodA* = 0. Se scrie așa cum e dată - pe linii.]
- [*Vectorul w0*: dacă *kodw0* = 0]
- *ns*: Număr de aproximații *s*, ale valorilor proprii
- *Aproximațiile s*: “*ns*” valori.
- *kod_met*: codul metodei
- *TOL*, *lnit*: toleranța, numărul limită de iterații.
- *kodTest*, *kodPrint*: cod test, cod tipărire iterații și vectori reziduali.
- *Check_text*: cod de verificare.

Semnificația codurilor:

n: ordinul matricii

kodA: 0 / ≠0: citește **A** / generează **A** (cu subrutina *Agen*)

kodw0: 0 / ≠0: citește $\mathbf{w0}$ / generează $\mathbf{w0}$ (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: Tipărește: Nimic / Vectorii reziduali $\mathbf{Az} - \lambda\mathbf{z}$ / Iterațiile și 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 toți vectorii \mathbf{z});
- indicele vectorului, și indicele coordonatei.

Exemplu:

Se consideră [exemplul](#) de la Power. Acesta este un caz special: matricea este singulară, și deci există o valoare proprie egală cu 0. Luăm ca aproximații *s* valorile 9, -1, și 0.01. Fișierul de intrare / ieșire 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 translație (shift), pentru o matrice complexă.

În particular, o matrice reală care are (și) valori complexe.

Metoda: cea de la [Inverse Power Shift](#).

Fișiere:

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: funcție 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: semnificațiile de la descompunerea LU_Complex.

Agen.f90; Maxindex_z.f90; w0gen.f90; width.f90; work.f90: aceleași semnificații ca la metoda puterii.

vNorm2.f90: norma euclidiană a unui vector real, și 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)

Semnificațiile parametrilor:

iter, kodTest, kodPrint: aceleași semnificații ca la *Inverse_Power_Shift*.

kodLU: cod descompunere LU; 0: o.k.; < 0: pivot < prag.

test_val: valoarea de test (comparată cu toleranța); aceasta depinde de *kodTest*.

Fișierul de date:

Are aceeași structură ca [fișierul de date](#) de la *Inverse_Power_Shift*, cu particularitatea că: matricea **A**, vectorul **w0**, și aproximațiile *s* – sunt acum, valori complexe. Dacă **A** este reală, se pot introduce valori reale; pentru vectorul **w0** trebuie însă introduse valori complexe.

Notă: Aproximațiile *s* trebuie să fie suficient de apropiate de valorile proprii.

Exemplu:

[Exemplul](#) de la *Power_Complex*. Matricea este hermitiană și are valori proprii reale.

În rezultate, partea complexă a valorilor proprii este de ordinul $10^{-7} \dots 10^{-8}$.

Fișierul de intrare / ieșire 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 iterațiilor simultane – matrice hermitiană.

Metoda:

Aceasta este o extindere a metodei puterii pentru o matrice \mathbf{A} hermitiană (în particular, reală și simetrică). O astfel de matrice, are valori proprii reale.

Presupunem, mai mult, că valorile proprii sunt de module distincte:

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|.$$

În loc de un vector de start $\mathbf{w}^{(0)}$, se utilizează o matrice de start, ale cărei coloane sunt vectorii de start: $\mathbf{W}^{(0)} = [\mathbf{w}^{(0,1)} \mid \mathbf{w}^{(0,2)} \mid \dots \mid \mathbf{w}^{(0,m)}]$.

Dacă matricea \mathbf{A} este $n \times n$, $\mathbf{W}^{(0)}$ este $n \times m$, unde $m \leq n$. Vectorii de start $\mathbf{w}^{(0,j)}$ trebuie să fie liniar independenți. Metoda de bază rămâne înmulțirea la stânga a lui $\mathbf{W}^{(0)}$ cu matricea \mathbf{A} , adică, $\mathbf{W}^{(k+1)} = \mathbf{A}\mathbf{W}^{(k)}$, $k \geq 0$. Înainte de fiecare etapă a iterației, matricea curentă \mathbf{W} este ortogonalizată prin procedeul Gram-Schmidt, astfel încât coloanele ei $\mathbf{w}^{(j)}$ devin vectori ortonormați (ortogonali, și având norma euclidiană unitară). Astfel, vectorii $\mathbf{w}^{(j)}$ formează o bază ortonormată a sub-spațiului m -dimensional al lui \mathbf{R}^n , sub-întins de vectorii inițiali $\mathbf{w}^{(0,j)}$. În cursul iterației, această bază se aliniază din ce în ce mai mult la baza vectorilor proprii ai lui \mathbf{A} , direcția $\mathbf{w}^{(j)} \rightarrow$ direcția $\mathbf{x}^{(j)}$, $j \geq 1$. Valorile proprii se evaluează prin câtul Rayleigh. Iterația

se încheie când se atinge o toleranță convenabilă, privitor la direcțiile a două baze succesive $\{\mathbf{w}^{(j)}\}$. Dacă matricea de start $\mathbf{W}^{(0)}$ are $m < n$ coloane, se obțin primii m vectori proprii. Pentru $m = n$, adică $\mathbf{W}^{(0)}$ este $n \times n$, se găsesc toți vectorii proprii.

Algoritmul:

Se cere un număr $ne \leq n$ de vectori proprii. $\mathbf{W}^{(0)}$ și \mathbf{W} sunt matrici $n \times ne$.

1. Se definește matricea $\mathbf{W}^{(0)}$: dacă o aproximare inițială a vectorilor proprii nu este cunoscută, se ia $\mathbf{W}^{(0)} = [\mathbf{e}^{(1)} \mid \mathbf{e}^{(2)} \mid \dots \mid \mathbf{e}^{(ne)}]$, 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 excepția cazului în care aceasta este deja ortonormată). Se atribuie $\mathbf{W} = \mathbf{W}^{(0)}$.
 3. Inițializare contor: iter = 0
 4. Iterații: 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 $\lambda_j, j = \overline{1, ne}$, prin câțul Rayleigh:

Fie $\mathbf{w}^{(j)}$ și $\mathbf{w}^{(0,j)}$ a j -a coloană a lui \mathbf{W} și $\mathbf{W}^{(0)}$, respectiv; avem

$$\lambda_j = \langle \mathbf{w}^{(0,j)}, \mathbf{w}^{(j)} \rangle$$

($\mathbf{w}^{(j)} = \mathbf{A}\mathbf{w}^{(0,j)}$ – Pasul 5; și $\langle \mathbf{w}^{(0,j)}, \mathbf{w}^{(0,j)} \rangle = 1$ – Pașii 2 și 4.)

7. Se aplică Gram-Schmidt la \mathbf{W} , astfel că \mathbf{W} devine ortonormată.
8. Se verifică atingerea toleranței TOL :

Prin testul de coliniaritate – 1.3, 1: se definește $\mathbf{colin}(n)$ pentru fiecare vector

$$\mathbf{z} = \mathbf{W}(:, je), \text{ și } test_val = \max_{je=1,ne} \|\mathbf{colin} - \mathbf{colin}(imax)\|.$$

(Întrucât \mathbf{z} sunt normalizați, testul se poate pune și sub forma din 1, Observație.)

- Dacă $\|test_val\| \leq TOL$, ieșire din iterație.

- Altfel, GOTO 4.

Observații

- Se poate prescrie un număr limită de iterații *lnit*: atunci, se adaugă la Pasul 8 un test $iter \leq lmit$.
- Pasul 6 se poate realiza numai o singură dată, după ce s-a ieșit din iterație.

■

Observații

1. Valori proprii de același modul

Întrucât metoda iterațiilor simultane este în esență metoda puterii, nu se obține convergență 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 obține convergență pentru vectorii proprii, cu excepția vectorului propriu corespunzând valorii dominante λ_1 (pentru care, metoda revine la metoda puterii). Aceasta se întâmplă deoarece ipoteza esențială a metodei este că vectorii proprii formează o bază ortogonală – și procedeul Gram-Schmidt forțează ca, la fiecare pas k , vectorii $\mathbf{w}^{(k,j)}$ să fie ortogonali.

Totuși, dacă valorile proprii sunt de module distincte, atunci vectorii proprii sunt liniar independenți, și se obțin aproximații foarte bune pentru valorile proprii.

Vezi Exemplu-2. Explicația este că, procedeul Gram-Schmidt furnizează un set de vectori $\{\mathbf{w}^{(j)}\}$ liniar independenți (ortonormați), iar valorile proprii sunt calculate cu câțul Rayleigh – care dă aproximații bune ale valorilor proprii chiar cu aproximații grosiere pentru vectorii proprii.

Valorile proprii găsite pot fi utilizate ulterior, în metoda puterii inverse cu translație, pentru a obține vectorii proprii pentru valorile λ_i , $i \geq 2$.

■

Fișiere:

Sim_iter.f90: subrutina metodei

Main-Sim_iter.f90: programul principal

Gram_Schmidt_Complex.f90: ortogonalizare Gram-Schmidt

Herm&Sim.f90: 2 funcții logice – pentru a recunoaște o matrice hermitiană, respectiv complexă și simetrică.

Maxindex_z.f90: funcție; 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)

Semnificațiile parametrilor sunt aceleași ca la metoda puterii (kodPrint: ca și ikod).

Datele de intrare se citesc dintr-un fișier de date (specificat). În acesta se scriu și datele de ieșire. Datele de intrare sunt:

- *Titlu*: text de max. 80 caractere
- *n, ne, kodA, kodw0*
- [*Matricea A*: dacă *kodA* = 0. Se scrie așa cum e dată - pe linii.]
- [*Vectorul w0*: dacă *kodw0* = 0]
- *TOL, lnit*: toleranța, numărul limită de iterații.
- *kodTest, kodPrint*: cod test, cod tipărire iterații și vectori reziduali.
- *Check_text*: cod de verificare.

Semnificația codurilor:

n: ordinul matricii **A**

ne: numărul de coloane ale matricii **W0** (numărul de vectori și valori proprii de calculat)

kodA: $\leq 0 / \neq 0$: citește **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.

$kodw0$: $0 / \neq 0$: citește $w0$ / generează $w0$ (cu subrutina `w0gen`)

$kodTest$: $\leq 0 / \geq 1$: Test asupra lui λ / Test de coliniaritate.

$kodPrint$: $0 / 1 / 2$: Tipărește: Nimic / Vectorii reziduali $Az - \lambda z$ / Iterațiile și 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 $Az - \lambda z \approx 0$. Se scriu:

- coordonata de modul maxim din $Az - \lambda z$ (maximul pe toți vectorii z);
- indicele vectorului, și indicele coordonatei.

Exemplu-1:

Matricea este hermitiană (se poate pune $kodA < 0$, și 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 Observației 2, de mai sus.

Reproducem numai rezultatele pentru valorile proprii, și verificarea pentru vectorul propriu nr. 1. Vectorii proprii 2-4 nu sunt determinați corect. Pentru a-i calcula, se va utiliza metoda iterației inverse cu translație, luând ca aproximații 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 și vectorii proprii ai unei matrici reale și simetrice, de ordin moderat. Determinarea vectorilor proprii este opțională.

Fie \mathbf{A} o matrice reală și simetrică. Dacă \mathbf{N} este o matrice nesingulară, atunci matricea $\overline{\mathbf{A}} = \mathbf{N}^{-1}\mathbf{A}\mathbf{N}$ este similară cu \mathbf{A} , și are aceleași valori proprii (bara nu notează acum conjugata). Vectorii proprii \mathbf{x} ai lui \mathbf{A} , sunt legați 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 \mathbf{N} este *unitară*, adică $\mathbf{N}^{-1} = \mathbf{N}^T$. Matricea $\bar{\mathbf{A}}$ devine

$$\bar{\mathbf{A}} = \mathbf{N}^T \mathbf{A} \mathbf{N}$$

Notați că $\bar{\mathbf{A}}$ este de asemenea simetrică.

Diagonalizarea lui A:

Să presupunem că \mathbf{N} este aleasă astfel încât $\bar{\mathbf{A}}$ să devină *diagonală*:

$$\bar{\mathbf{A}} = \mathbf{N}^T \mathbf{A} \mathbf{N} = \text{diag}(\bar{a}_{ii})$$

Pentru matricea $\bar{\mathbf{A}}$, avem proprietățile:

- 1) Valorile proprii ale lui $\bar{\mathbf{A}}$ sunt elementele diagonale: $\lambda_i = \bar{a}_{ii}$.
- 2) Vectorii proprii ai lui $\bar{\mathbf{A}}$ sunt coloanele matricii unitate \mathbf{I} : $\bar{\mathbf{x}}^{(i)} = \mathbf{e}^{(i)}$ (unde $e_j^{(i)} = \delta_{ij}$).

În consecință, rezultă pentru \mathbf{A} :

- Valorile proprii ale lui \mathbf{A} se găsesc pe diagonala matricii $\bar{\mathbf{A}}$.
- Vectorii proprii ai lui \mathbf{A} sunt coloanele matricii \mathbf{N} .

Diagonalizarea Jacobi:

Metoda Jacobi constă în transformări unitare (sau, ortogonale) aplicate succesiv lui \mathbf{A} , până la obținerea unei forme aproape diagonale. Anume, dacă \mathbf{N}_i sunt matrici unitare, matricea \mathbf{A} se transformă cum urmează:

$$\bar{\mathbf{A}}_1 = \mathbf{N}_1^T \mathbf{A} \mathbf{N}_1$$

$$\bar{\mathbf{A}}_2 = \mathbf{N}_2^T \bar{\mathbf{A}}_1 \mathbf{N}_2 = (\mathbf{N}_2^T \mathbf{N}_1^T) \mathbf{A} (\mathbf{N}_1 \mathbf{N}_2)$$

...

$$\bar{\mathbf{A}}_k = \mathbf{N}_k^T \bar{\mathbf{A}}_{k-1} \mathbf{N}_k = (\mathbf{N}_k^T \mathbf{N}_{k-1}^T \dots \mathbf{N}_1^T) \mathbf{A} (\mathbf{N}_1 \mathbf{N}_2 \dots \mathbf{N}_k)$$

Dacă matricea $\bar{\mathbf{A}}_k$ este *aproape* diagonală, adică, elementele non-diagonale sunt aproximativ zero, luăm

$$\bar{\mathbf{A}} \approx \bar{\mathbf{A}}_k = \mathbf{N}^T \mathbf{A} \mathbf{N}$$

*Considerații de programare**Stocajul matricii:*

Se lucrează cu triunghiul superior al lui \mathbf{A} , astfel încât, după diagonalizarea lui \mathbf{A} , elementul (1,1) conține λ_1 , etc. Triunghiul superior este stocat în vectorul

a ($n * (n+1) / 2$), în ordinea coloanelor, adică:

$$a = [a_{11} \mid a_{12} \quad a_{22} \mid a_{13} \quad a_{23} \quad a_{33} \mid \dots \mid a_{1n} \quad a_{2n} \quad \dots \quad a_{nn}]$$

Adresa elementului (i, j) este dată de următoarea funcție:

$$Loca(i, j) = (j - 1) * j / 2 + i$$

În particular, elementul diagonal (i, i) are adresa $Loca(i, i) = (i + 1) * i / 2$. După ce s-a efectuat o transformare, matricea curentă $\bar{\mathbf{A}}$ este stocată în același vector a .

Strategia de eliminare:

Aceasta este *căutarea completă*, adică: se caută în toată matricea \mathbf{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 toleranță TOL .

■

Fișiere:

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: funcție; adresa elementului $a(i, j)$ în vectorul ag .

work.f90: modul

width.f90: utilitar

Subrutina metodei:

Jacobi(kodVP, kodl, iter, amax)

Parametri:

kodVP: v. mai jos

kodI: v. mai jos, *kodPrintJ*.

iter: Numărul limita de iterații (intrare) / Numărul efectiv de iterații (ieșire).

amax: Elementul non-diagonal de modul maxim, din \mathbf{A} .

Fișier de date (intrare/ieșire):

- *Titlu*: text de max. 80 caractere
- *n*, *kodA*, *kodPrintA*
- [*Matricea A*: dacă *kodA* = 0. Se introduce triunghiul superior, pe linii.]
- *TOL*, *kodPrintJ*: toleranța pentru elementele non-diagonale; cod de tipărire
a iterațiilor.
- *kodVP*: cod pentru calculul vectorilor proprii
- *Check_text*: cod de verificare.

Semnificația codurilor:

n: ordinul matricii \mathbf{A}

kodA: 0 / ≠ 0: citește \mathbf{A} / generează \mathbf{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 – Tipărește matricea \mathbf{A} (triunghiul superior).

kodPrinA ≠ 0 se utilizează pentru a tipări o matrice generată.

kodPrinJ: 0 / ≠ 0: Nu / Da – Tipărește transformările succesive ale matricii \mathbf{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 toți vectorii \mathbf{z});
 - indicele vectorului, și indicele coordonatei.

Pentru F sau f : se scrie și fiecare vector rezidual $Az - \lambda z$.

Notă:

Proba se face numai dacă se cere calculul vectorilor proprii ($kodVP \neq 0$).

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.

■

Observație

Utilizând [InversePower Shift](#) cu aproximațiile inițiale 14.3, 4.46, -0.371, -3.42 (valorile Jacobi cu 3 cifre semnificative corecte), $\mathbf{w}^{(0)} = (1, 1, 1, 1)$, și toleranța $1E-7$, se găsesc rezultatele:

- Aceleași valori proprii nr. 1 și 3; valorile nr. 2 și 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, și rezultate mai puțin precise pentru vectorii proprii.

■

QR

Algoritmul QR – matrice reală.

Metoda (sumar)

QR este unul din cei mai utilizați 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 \mathbf{A} , simetrică sau nu.

Esența algoritmului QR rezidă în următoarea proprietate:

Dacă \mathbf{A} este factorizată în produsul $\mathbf{A} = \mathbf{QR}$ unde \mathbf{Q} este nesingulară, atunci considerând produsul în ordine inversă, fie $\mathbf{A}' = \mathbf{RQ}$, această matrice are aceleași valori proprii ca și \mathbf{A} , fiind similară cu \mathbf{A} ■

Într-adevăr, avem $\mathbf{R} = \mathbf{Q}^{-1}\mathbf{A}$, și $\mathbf{A}' = \mathbf{Q}^{-1}\mathbf{A}\mathbf{Q}$.

Algoritmul QR realizează factorizări succesive ale șirului de matrici $\{\mathbf{A}_k\}$, unde

$\mathbf{A}_1 = \mathbf{A}$, definite de:

$$\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1; \quad \mathbf{A}_2 = \mathbf{R}_1\mathbf{Q}_1;$$

$$\mathbf{A}_2 = \mathbf{Q}_2 \mathbf{R}_2; \quad \mathbf{A}_3 = \mathbf{R}_2 \mathbf{Q}_2;$$

...

$$\mathbf{A}_k = \mathbf{Q}_k \mathbf{R}_k; \quad \mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k;$$

...

Matricile \mathbf{A}_k , și $\mathbf{Q}_k, \mathbf{R}_k$, au următoarele proprietăți:

- 1) Toate \mathbf{A}_k au aceleași valori proprii ca și \mathbf{A} .
- 2) Dacă \mathbf{A} este *simetrică*, sau *tridiagonală*, sau \mathbf{A} are forma *Hessenberg superioară*, matricile \mathbf{A}_k vor păstra aceste forme.
- 3) Fie $\overline{\mathbf{Q}}_k = \mathbf{Q}_1 \mathbf{Q}_2 \dots \mathbf{Q}_k$, atunci (prin inducție): $\mathbf{A}_{k+1} = \overline{\mathbf{Q}}_k^{-1} \mathbf{A}_1 \overline{\mathbf{Q}}_k$.
- 4) Fie $\overline{\mathbf{R}}_k = \mathbf{R}_k \dots \mathbf{R}_1$, atunci: $\overline{\mathbf{Q}}_k \overline{\mathbf{R}}_k = (\mathbf{A}_1)^k$

Algoritm:

Algoritmul parcurge următoarele etape:

- I. Reducerea lui \mathbf{A} la forma tridiagonală dacă \mathbf{A} este simetrică, sau la forma Hessenberg dacă \mathbf{A} este nesimetrică.
- II. Reducerea lui \mathbf{A} la forma triunghiulară, sau bloc- triunghiulară (v. mai jos).
- III. Descompunerea \mathbf{QR} .

Algoritmul QR propriu-zis, constă în Etapele II și III.

- a) Etapa I se realizează înainte de algoritmul QR pentru a aduce matricea \mathbf{A} la o forma mai simplă, și a reduce astfel efortul de calcul.
- b) Etapa II se realizează prin premultiplicare cu matricea Householder \mathbf{P}_k , cum urmează:

$$\mathbf{A}_1 = \mathbf{P}_1 \mathbf{A}$$

$$\mathbf{A}_2 = \mathbf{P}_2 \mathbf{A}_1 = \mathbf{P}_2 \mathbf{P}_1 \mathbf{A}$$

...

$$\mathbf{A}_{n-1} = \mathbf{P}_{n-1} \dots \mathbf{P}_2 \mathbf{P}_1 \mathbf{A}$$

Matricea \mathbf{A}_{n-1} are forma triunghiulară și este matricea \mathbf{R} .

Într-adevăr: Să notăm $\mathbf{Q} = \mathbf{P}_1 \mathbf{P}_2 \dots \mathbf{P}_{n-1}$, avem $\mathbf{A}_{n-1} = \mathbf{Q}^T \mathbf{A}$. \mathbf{Q} este o matrice ortogonală, deci avem $\mathbf{Q} \mathbf{A}_{n-1} = \mathbf{A}$. Cum \mathbf{Q} este ortogonală și \mathbf{A}_{n-1} este superior triunghiulară, urmează că $\mathbf{R} = \mathbf{A}_{n-1}$.

■

Atunci, Etapele II și III se realizează prin următorii pași de iterare:

1) Pune $\mathbf{A}^{(1)} = \mathbf{A}$

- Triangularizează $\mathbf{A}^{(1)}$:

$$\mathbf{A}_1^{(1)} = \mathbf{P}_1^{(1)} \mathbf{A}; \quad \mathbf{A}_2^{(1)} = \mathbf{P}_2^{(1)} \mathbf{A}_1^{(1)}; \quad \dots; \quad \mathbf{A}_{n-1}^{(1)} = \mathbf{P}_{n-1}^{(1)} \mathbf{A}_{n-2}^{(1)}$$

- La sfârșitul pasului, avem:

$$\mathbf{R}^{(1)} = \mathbf{A}_{n-1}^{(1)}; \quad \mathbf{Q}^{(1)} = \mathbf{P}_1^{(1)} \mathbf{P}_2^{(1)} \dots \mathbf{P}_{n-1}^{(1)}.$$

2) Calculează $\mathbf{A}^{(2)} = \mathbf{R}^{(1)} \mathbf{Q}^{(1)}$

- Triangularizează $\mathbf{A}^{(2)}$:

$$\mathbf{A}_1^{(2)} = \mathbf{P}_1^{(2)} \mathbf{A}^{(2)}; \quad \mathbf{A}_2^{(2)} = \mathbf{P}_2^{(2)} \mathbf{A}_1^{(2)}; \quad \dots; \quad \mathbf{A}_{n-1}^{(2)} = \mathbf{P}_{n-1}^{(2)} \mathbf{A}_{n-2}^{(2)}$$

- Pune:

$$\mathbf{R}^{(2)} = \mathbf{A}_{n-1}^{(2)}; \quad \mathbf{Q}^{(2)} = \mathbf{P}_1^{(2)} \mathbf{P}_2^{(2)} \dots \mathbf{P}_{n-1}^{(2)}.$$

...

k) Calculează $\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)} \mathbf{Q}^{(k-1)} \quad (k \geq 2)$

- Triangularizează $\mathbf{A}^{(k)}$:

$$\mathbf{A}_1^{(k)} = \mathbf{P}_1^{(k)} \mathbf{A}^{(k)}; \quad \mathbf{A}_2^{(k)} = \mathbf{P}_2^{(k)} \mathbf{A}_1^{(k)}; \quad \dots; \quad \mathbf{A}_{n-1}^{(k)} = \mathbf{P}_{n-1}^{(k)} \mathbf{A}_{n-2}^{(k)}$$

- Pune:

$$\mathbf{R}^{(k)} = \mathbf{A}_{n-1}^{(k)}; \quad \mathbf{Q}^{(k)} = \mathbf{P}_1^{(k)} \mathbf{P}_2^{(k)} \dots \mathbf{P}_{n-1}^{(k)}$$

...

■

Calculația 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 \mathbf{A} are valori proprii *de module distincte*, avem:

- a) Dacă \mathbf{A} este simetrică, atunci toate elementele non-diagonale vor fi aproximativ zero. Reamintim că o matrice simetrică are numai valori proprii reale.
- b) Dacă \mathbf{A} este nesimetrică și are numai valori proprii reale, calculația se termină când toate elementele sub-diagonale sunt aproximativ zero.

Dacă \mathbf{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 \mathbf{A} .

Convergența:

- Dacă \mathbf{A} este o matrice reală (simetrică sau nu) și 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 \mathbf{A} .
- Dacă \mathbf{A} are o valoare proprie λ de multiplicitate m , atunci $\mathbf{A}^{(k)}$ încă converg la matrice superior triunghiulară, cu excepția unui bloc diagonal de ordinul m , ale cărui valori proprii au modulul egal cu $|\lambda|$.

Algoritmul QR cu deplasare:

Presupunem că \mathbf{A} are valori proprii *de module distincte* $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| > 0$.

Se arată că viteza de convergență la zero a elementelor sub-diagonale, și cea de convergență la valorile proprii a elementelor diagonale, depind de rapoartele

$$(\lambda_{i+1} / \lambda_i)^k, \quad 1 \leq i \leq n-1.$$

Dacă două valori proprii λ_i și λ_{i+1} sunt apropiate una de alta, convergența va fi înceată. Atunci, se utilizează următoarea tehnică pentru a accelera convergența. Se

aplică o deplasare s_k la valorile proprii (acestea devin $\lambda_i - s_k$), la fiecare etapă k .

Adică, punem

$$\mathbf{A}_1 = \mathbf{A}, \text{ și}$$

$$\mathbf{A}^{(k)} - s_k \mathbf{I} = \mathbf{Q}^{(k)} \mathbf{R}^{(k)},$$

$$\mathbf{A}^{(k+1)} = \mathbf{R}^{(k)} \mathbf{Q}^{(k)} + s_k \mathbf{I}$$

Există două strategii pentru a alege deplasarea s_k (s_k este o aproximație a lui λ_n):

$$1) \quad s_k = a_{nn}^{(k)}$$

Pentru o *matrice simetrică* (reală), această strategie asigură o convergență cubică, chiar în prezența 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:

$$s_k = \text{valoarea proprie care este cea mai apropiată de } a_{nn}^{(k)}.$$

Dacă valorile proprii sunt complexe, se ia:

$$s_k = \text{partea reală a valorii proprii.}$$

3) Experimentele numerice au condus la o a treia strategie, și anume:

$$s_k = \min(a_{ii}^{(k)}), \quad i = \overline{1, n}$$

Pentru unele matrici simetrice, această strategie se dovedește cea mai bună, conducând la un număr mai mic de iterații, și o eroare mai mică în $\mathbf{A}\mathbf{y} - \lambda\mathbf{y}$.

Algoritm și cod

Programul calculează sistemul propriu (valori și vectori proprii), în succesiunea următoare:

I. Transformarea preliminară a matricii \mathbf{A} (Opțional).

II. Valorile proprii, prin reducerea matricii la forma bloc-triunghiulară.

III. Vectorii proprii (Opțional).

Se calculează, mai întâi, vectorii proprii ai matricii bloc-triunghiulare. Aceștia se aduc, prin transformarea de similaritate, la vectorii proprii ai lui \mathbf{A} .

IV. Rafinarea sistemului propriu prin iterație inversă (Opțional; numai în cazul când s-a ales opțiunea de calcul a vectorilor proprii).

V. Verificarea sistemului propriu (Opțional):

Precizia sistemului propriu se verifică prin listarea erorii maxime (în modul), în $\mathbf{A}\mathbf{y} - \lambda\mathbf{y}$ (unde λ este valoarea proprie, iar \mathbf{y} vectorul propriu asociat cu λ).

Datele se citesc dintr-un fișier de date. Rezultatele sunt scrise în acest fișier, în continuarea datelor de intrare.

Codul este scris în dublă precizie.

Detalii de implementare

1. Introducerea matricii

Matricea \mathbf{A} se definește în unul din următoarele moduri:

- Citită din fișier
- Generată (de subrutina *Agen*)

Citirea se poate face, conform unui cod, astfel:

- Matrice simetrică: triunghiul superior
- Matrice generală: completă, pe linii și 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, și forma Hessenberg – pentru o matrice generală.

3. Etapele II, III – Iterația QR

Iterația QR se face cu deplasare, și 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 iterații, *linit*
- Toleranța *eps* pentru forma bloc-triunghiulară
- Toleranța *eps_lambda* pentru valorile proprii (v. mai jos)

Iterația QR este oprită dacă una din condițiile 1-4 de mai jos este îndeplinită:

1. S-a atins *linit*.
2. S-a obținut forma bloc-triunghiulară.
3. S-a atins toleranța *eps_lambda* pentru doua seturi succesive de valori proprii.

Forma bloc-triunghiulară se consideră obținută, dacă următoarele două teste sunt satisfăcute:

$$|Max_Sub_Sub| \leq eps_sub \quad (a)$$

$$|Max_Left_Diag| \leq eps_left, \quad (b)$$

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 tipăriți de program.

Observații:

- Testul (a) este îndeplinit (după cel puțin o iterație) dacă, inițial, matricea este redusă la una din formele preliminare (tridiagonală sau Hessenberg).
- Testul (b) se pune pentru elementul stâng-diagonal a primei linii a blocului 2×2 . (În cazul unui bloc de dimensiune 1 – pentru linia respectivă.)
- Codurile pentru metoda QR – v. Bai et al. (2000), Golub & Van Loan (1996) – pun testul

$$|a_{i,i-1}| \leq tol(|a_{ii}| + |a_{i-1,i-1}|),$$

unde tol este o toleranță mai mare decât eroarea de rotunjire a unității.

Elementele sub-diagonale $a_{i,i-1}$ care satisfac acest test sunt setate la zero.

4. Iterația se mai oprește dacă elementul stâng-diagonal de modul maxim este staționar (nu se mai reduce).

Testul pus este $Max_Left_Diag = Prev_Max_Left_Diag$, unde membrul doi este Max_Left_Diag de la iterația precedentă. Testul se pune după ce Max_Left_Diag s-a redus deja în proporția $|Max_Left_Diag| \leq eps_left * 10$.

■

Balansare

Eroarea în calculul valorilor proprii ale matricii \mathbf{A} , rezultată prin aplicarea metodei QR , este de ordinul $\varepsilon_M \|\mathbf{A}\|_F$, unde ε_M este ε -mașină (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 \mathbf{A} , prin similaritate cu matricea diagonală reală \mathbf{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 \mathbf{A} . Ținând cont de echivalența normelor, rezultă că va fi redusă și norma Frobenius a matricii \mathbf{A} .

Balansarea se poate face fără a introduce erori de rotunjire suplimentare, prin aceea că elementele matricii \mathbf{D} se aleg ca puteri întregi ale bazei utilizate în reprezentarea numerelor în calculator (baza 2).

Principalul algoritm este descris în lucrarea de referință Parlett & Reinsch (1969). Înainte de balansare, prin permutări de linii și 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 $\mathbf{0}$ o matrice cu elemente nule. Algoritmul se aplică, în esență, matricii \mathbf{Z} (dar transformarea de similaritate se aplică întregii matrici, astfel că se modifică și matricile \mathbf{X} , \mathbf{Y} , \mathbf{W}).

Algoritmul balansează matricea în norma-1, după un număr finit de pași. Acest algoritm stă la baza subrutinei `sghal` din LAPACK (2000). Alți algoritmi sunt descriși 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 $\mathbf{D} = \text{diag}(0.01 \quad 1 \quad 100)$, 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: $\| \mathbf{A} \|_F = 1.00005 \times 10^4$ și $\| \mathbf{A}_b \|_F = 2.65$. Pe de altă parte, normele liniilor și coloanelor în \mathbf{A}_b sunt aproximativ egale.

Metoda *QR* aplicată lui \mathbf{A} , fără rafinare (cu $EPS = 1E-8$), cu și fără balansare, dă următoarele rezultate:

- Valorile și vectorii proprii coincid, cu cel puțin 10, și respectiv 9 cifre semnificative.
- Erorile maxime (în modul) în $\mathbf{A}\mathbf{y} - \lambda\mathbf{y}$ sunt: 3.229E-11 și 2.887E-14, respectiv. Se observă precizia mult mai mare a sistemului propriu calculat cu balansarea matricii \mathbf{A} .

■

2. Balansarea cu subrutina *sgbal* (în simplă precizie), dă următoarele rezultate:

$$\mathbf{D} = \text{diag}(2.4414062E-04 \quad 1.5625000E-02 \quad 1.0000000)$$

$$\mathbf{A}_b = \begin{bmatrix} 1.000000 & 0.000000 & 0.40960 \\ 1.5625000 \times 10^{-2} & 1.000000 & 0.6400000 \\ 2.441406 & 1.562500 & 1.000000 \end{bmatrix}$$

Este de remarcat că, prin acest procedeu, matricea este balansată *aproximativ* în norma-1. Normele liniilor și coloanelor sunt de același ordin de mărime (însă, diferite):

1	1.409600	3.457031
2	1.655625	2.562500
3	5.003906	2.049600

Eroarea maximă în $\mathbf{A}\mathbf{y} - \lambda\mathbf{y}$ 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 , și $norm1_r$ norma liniei i fără elementul diagonal, adică: $norm1_r = \sum_{\substack{j=1,n \\ j \neq i}} |a(i, j)|$; analog, pentru coloana i ,

$norm_c$ și $norm1_c$. Algoritmul este următorul:

- Inițializăm $\mathbf{D} = \mathbf{I}$, și iterăm pașii $K = 1, 2, \dots$

- La fiecare pas K , procesăm liniile (și coloanele) în ordinea naturală, anume:

Pentru $i = 1, 2, \dots, n$:

- Calculăm elementele matricei \mathbf{D}_1 , prin $d1 = \sqrt{\text{norm1}_r / \text{norm1}_c}$
- Se scalează linia și 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 \mathbf{D} , la pasul curent K , și s-a calculat $\mathbf{A} = \mathbf{D}^{-1} \mathbf{A} \mathbf{D}$.

- Calculăm (pentru matricea curentă \mathbf{A}):

$$\text{Ratio} = \max_{i=1,n} | \text{norm}_r - \text{norm}_c | / \text{norm}_r$$

- Se pun următoarele condiții de oprire a iterației:

$$\text{Ratio} \leq \text{EPS}; \quad \text{Numar_Iteratii} \leq \text{lnit};$$

(Se iau: $\text{EPS} = 1E - 4$ și $\text{lnit} = 100$).

Pentru matricea din Exemplu, la pasul 2, se obține: $\text{Ratio} = 1.2064E - 05$;

$$\mathbf{D} = \text{diag}(9.9750279E - 05 \quad 1.0074407E - 02 \quad 1.002465).$$

Norme liniilor și 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 $4.291E-14$.

■

Fișiere

e) Fișiere sursă:

Agen.f90: subrutina de generare a matricii \mathbf{A}

Main-QR-2005.f90: programul principal

work-2005.90: modul

f) Biblioteca statică:

QR_Lib.lib

Aceasta conține fișiere modul obiect (pentru aceste fișiere nu se dă codul sursă).

În proiect se vor include cele trei fișiere sursă și biblioteca statică.

g) Fișiere de date:

Acestea se găsesc în sub-folderul QR\Dat. La lansarea în execuție, programul deschide fereastra standard de selecție a fișierului din acest folder. Un fișier nou de date, este preferabil să fie plasat în acest folder.

Programul QR este scris în dublă precizie.

Programul principal apelează următoarele subrutine (și o funcție):

GetFile(fname, status)

Selectarea fișierului de intrare. Bazată pe API-ul GetOpenFileName.

fname: nume fișier

status: întreg

unit2(n, Q)

Inițializează matricea reală $Q(n, n)$ cu matricea unitate.

n: ordinul matricii Q

Q: matrice (n, n)

Agen(Ag, job)

Generarea matricii A

Ag: tablou (n, n), pentru matricea A generată.

Job: character (20); v. Semnificația codurilor.

symm(n, a)

Funcție logică; matrice A simetrică.

n: ordinul matricii A

a: tablou (n, n) pentru A.

PrintA(n, A)

Tipărirea matricii $A(n, n)$

Balance($n, a, \text{diag_bal}, 0$), Direct_Balance($n, a, \text{diag_bal}$)

Balansarea matricii A , și 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: ieșire;

Direct_Balance: intrare).

Tridiagonal(kod), Hessenberg(ikod)

Reducerea la forma tridiagonală, respectiv Hessenberg.

ikod : cod de tipărire a transformărilor. Se ia 0.

QR_Z($\text{iter}, \text{do_shift}, \text{kodShift}, \text{kodIter}, \text{BlockReduced}, \text{real_val}$)

Iterația QR.

iter : contor de iterații

do_shift : logic; cu sau fără shift.

kodShift : codul shift-ului. V. Semnificația codurilor.

kodIter : cod de tipărire a iterațiilor.

BlockReduced : logic. Matrice redusă la forma bloc-triunghiulară (ieșire).

real_val : logic. Toate valorile proprii sunt reale (ieșire).

Vec_BlocTri(VEC, kodNorm), Vec_BlocTri_Z(VEC_Z, kodNorm).

Calculează vectorii proprii ai unei matrici bloc-triunghiulare – 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 și 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: Semnificația 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, și a vectorilor $Az - \lambda z$.

Structura fișierului de date

h) *Titlu*: text, max. 80 caractere (titlul problemei).

i) *n*, *kodA*, *kodPrintA*

Datele următoare definesc matricea **A**. Se introduce una din variante, în funcție de codul *kodA*:

- *Matricea A* ... *kodA* = 0, 1 (matrice citită din fișierul de date)
- *Job* ... *kodA* < 0 (matrice generată).
- *Fname_2*: ... *kodA* ≥ 2 (matricea se citește din fișierul *Fname_2*).

j) *Balance*, *kod_Balance*: cod pentru balansarea matricii; cod pentru modul de balansare.

- [*diags*(*i*), *i* = 1, *n* ... Dacă *kod_Balance* ≠ 0.]

k) *Form*: cod pentru transformări preliminare ale matricii.

l) *Eps*, *Eps_lambda*, *Lnit*, *kodIter*: toleranțe; număr limită de iterații, cod tipărire iterații.

m) *Shift*, *kodShift*: cod pentru shift ; cod strategie de alegere a shift-ului.

n) *kodVP*: cod pentru calculul vectorilor proprii.

o) [*Refine*: rafinarea sistemului propriu prin iterație 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.

Semnificația codurilor:

n: ordinul matricii

kodA: *kodA* ≥ 0: matrice citită; *kodA* < 0: matrice generată (de *Agen*).

Valori *kodA* și datele pentru definirea matricii **A**:

$kodA = 0$: *triunghiul superior*, pe linii (matrice simetrică) – în fișierul de intrare;

$kodA = 1$: *întreaga matrice*, pe linii – în fișierul de intrare;

$kodA \geq 2$: *Fname_2*: specificatorul fișierului care conține elementele matricii.

Structura fișierului *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;

$kodA \geq 4$: elementele nenule ale matricii, anume:

- $i, j, a(i, j)$.
- Sfârșitul de fișier 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: tipărește **A** (dată, și redusă la forma bloc-triunghiulară);

0: tipărește **A** redusă la forma bloc-triunghiulară;

< 0: nu țipari.

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 ($\mathbf{A} \rightarrow \mathbf{D}^{-1}\mathbf{AD}$)

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: Toleranța la elementele non-diagonale

- Eps_lambda*: Toleranța la două valori succesive λ
- Lnit*: Numărul limită de iterații
- kodIter*: 0 / ≠0: Da / Nu – tipărește iterațiile.
- 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 iterație inversă (cu translație).
 Notă: Dacă se rafinează: vectorii proprii sunt deja normalizați cu norma-2.
- is*: Definește shift-ul pentru iterația 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 și vectorii reziduali $Az - \lambda z$.

(*do_balance, do_Shift, do_check, full_print_check*: date logice; utilizare internă)

Semnificația unor date de ieșire:

Row: Indicele liniei matricii reduse.

Bloc Flag = 1 / 2 : Indicele liniei "Row" în bloc (Prima linie / A Doua linie).

Bloc Flag = 0: Linia nu aparține unui bloc.

(Intern, se utilizează tabloul `ibloc(1:n)`: Indicele liniei blocului; la ieșire, 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 afișează și 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 (și 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 definește prin:

$$\mathbf{C} = \begin{bmatrix} -2 & 2 & 2 & 2 \\ -3 & 3 & 2 & 2 \\ -2 & 0 & 4 & 2 \\ -1 & 0 & 0 & 5 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} 5\mathbf{C} & -\mathbf{C} \\ 5\mathbf{C} & \mathbf{C} \end{bmatrix};$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & 2\mathbf{B} \\ 4\mathbf{B} & 3\mathbf{B} \end{bmatrix}.$$

Matricea \mathbf{A} are următoarele valori proprii:

$60 \pm 20i$, $45 \pm 15i$, $30 \pm 10i$, $15 \pm 5i$, $-12 \pm 4i$, $-9 \pm 3i$, $-6 \pm 2i$, $-3 \pm i$.

Codul în Agen – pentru generarea matricii \mathbf{A} în tabloul $ag(n, n)$, este dat mai jos. (În datele de intrare, se pune $n = 16$).

...

```
ELSEIF(job == 'ebel16') 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
ebel6
NoBal 0
auto
0.01  0.01 100  0
Shift    0
1
F
2
ch
```

Forma bloc-triunghiulară este atinsă la iterația 21 (și toleranța la λ , la iterația 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 spațiu, vectorii proprii nu sunt listați):

```
Eigenvalues
1   30.000000000000003    10.000000000000001
2   30.000000000000003   -10.000000000000001
3   15.000000000000001    4.999999999999977
4   15.000000000000001   -4.999999999999977
5   60.000000000000001    20.000000000000001
6   60.000000000000001   -20.000000000000001
7   44.999999999999998    15.000000000000002
8   44.999999999999998   -15.000000000000002
9   -2.999999999999998    1.000000000000001
10  -2.999999999999998   -1.000000000000001
11  -6.000000000000003     2.000000000000008
12  -6.000000000000003   -2.000000000000008
13 -12.000000000000000     4.000000000000002
```

14	-12.0000000000000000	-4.0000000000000002
15	-9.0000000000000000	3.0000000000000027
16	-9.0000000000000000	-3.0000000000000027

Check result:

Maximum absolute difference $(A*y - \lambda*y) = 9.769963E-15$

Vector #	Coordinate(s) #
8	1



Observații:

Toleranțele Eps și 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, îmbunătățite foarte mult, prin rafinare (eroare de ordinul $E-15$).

Toleranțe mai mici, nu îmbunătățesc verificarea sistemului propriu. Exemplu:

Toleranțe de 0.001 duc la o eroare de ordinul $E-14$ în verificare.

Acest exemplu dă o idee, despre dificultățile care apar în calculul sistemului propriu al unei matrici de ordin relativ mare; aceste dificultăți cresc odată cu ordinul matricii.

Pentru o matrice de ordin mare, nu se vor alege toleranțe Eps și 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 excepția elementelor:

$$a(i, i) = i; \quad i = \overline{1, n}$$

$$a(i, i+1) = n; \quad i = \overline{1, n-1};$$

$$a(n, 1) = \varepsilon.$$

Exemplul Wilkinson ia $n = 20$.

Dacă perturbația $\varepsilon = 0$, matricea este superior triunghiulară, și valorile proprii sunt $\lambda(i) = i$.

Pentru $\varepsilon = 1E-10$, cu metoda QR *fără rafinare* (cu $EPS = 1E-6$), se obțin următoarele rezultate:

- a) *Fără balansare*: Valorile proprii rezultă toate reale. Ele reprezintă perturbații ale valorilor i , $i = \overline{20,1}$, anume (ordonate descrescător):

19.99999998462896, 19.00001298980951, 17.99999341364806,
 ..., 3.000002187998112, 1.99999854534825, 1.0000000000000000

Eroarea în verificarea sistemului propriu este $1.232E-07$.

- b) *Cu balansare*: Valorile proprii calculate conțin 14 valori complexe, și 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 $4.211\text{E}-06$.

(Cu balansare și rafinare, eroarea devine $1.776\text{E}-15$.)

■

Explicația rezultatelor foarte diferite în cazurile (a) și (b), este următoarea: polinomul caracteristic al matricii este $p(\lambda) = (\lambda - 1)(\lambda - 2)\dots(\lambda - 20) - 20^{19}\varepsilon$, și acesta este foarte rău condiționat; numerele de condiție pentru rădăcini sunt cuprinse între 4.31×10^7 și 3.98×10^{12} . Odată cu polinomul, și valorile proprii sunt rău condiționate. (Problema condiționării valorilor proprii depășește 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 $\| \mathbf{Az} - \lambda \mathbf{z} \| = \text{“mică”}$), 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-condiționate. Matricile simetrice au valori proprii bine-condiționate.

■

General_R, General_R1

Problema generalizată – reducerea la problema standard.

Problema generalizată de valori proprii

Problema determinării valorilor și vectorilor proprii ai unei matrici \mathbf{A} , definită de $\mathbf{Ax} = \lambda\mathbf{x}$, sau de ecuația

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \quad (1)$$

se va numi problema *standard*.

În Dinamica Structurilor apare următoarea problemă:

$$(\mathbf{K} - \lambda\mathbf{M})\mathbf{x} = \mathbf{0}, \quad (2)$$

unde: $\lambda = \omega^2$, iar \mathbf{K} și \mathbf{M} sunt matrici *simetrice și pozitiv definite* (\mathbf{K} și \mathbf{M} sunt respectiv, matricile de rigiditate și de masă; ω este pulsația proprie).

Problema (2) se zice problema *generalizată* de valori proprii. Mai general considerând două matrici $n \times n$ \mathbf{A} și \mathbf{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 și pozitiv definită*, cum urmează.

1) Se face descompunerea Cholesky a lui \mathbf{M} :

$$\mathbf{M} = \mathbf{S}^T \mathbf{S}, \quad (3)$$

unde \mathbf{S} este superior triunghiulară.. Atunci, (2) scrisă în forma $\mathbf{Kx} = \lambda\mathbf{Mx}$,

devine $\mathbf{Kx} = \lambda\mathbf{S}^T \mathbf{Sx}$, și se scrie din nou ca și

$$\mathbf{KS}^{-1}(\mathbf{Sx}) = \lambda\mathbf{S}^T(\mathbf{Sx}). \quad (4)$$

2) Se notează

$$\mathbf{y} = \mathbf{Sx} \quad (5)$$

și ecuația (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 obține

$$\mathbf{S}^{-T} \mathbf{KS}^{-1} \mathbf{y} = \lambda \mathbf{I} \mathbf{y}$$

3) Acum, definim

$$\mathbf{R} = \mathbf{S}^{-T} \mathbf{K} \mathbf{S}^{-1} \quad (5)$$

\mathbf{R} este matrice *simetrică și pozitiv definită*.

4) Problema (2) este pusă în forma standard pentru matricea \mathbf{R} , și anume,

$$(\mathbf{R} - \lambda \mathbf{I}) \mathbf{y} = \mathbf{0} \quad (6)$$

Problemele (6) și (2) au aceleași valori proprii.

5) După rezolvarea lui (6) pentru λ și \mathbf{y} , vectorii proprii ai problemei originale (2) sunt dați, cf. (4), de

$$\mathbf{x} = \mathbf{S}^{-1} \mathbf{y} \quad (7)$$

■

Ținând cont de faptul că \mathbf{M} și \mathbf{K} sunt simetrice și pozitiv definite, se verifică imediat că matricea \mathbf{R} este simetrică și pozitiv definită. Urmează că valorile proprii ale lui \mathbf{R} sunt reale și pozitive, așa 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 și pozitiv definite poate fi aplicată lui \mathbf{R} .

Observație

Cel mai simplu caz este acela în care matricea \mathbf{M} este diagonală (model de mase concentrate), $\mathbf{M} = \text{diag}(m_i)$. Atunci, avem $\mathbf{S} = \mathbf{S}^T = \text{diag}(\sqrt{m_i})$, și

$\mathbf{S}^{-1} = \mathbf{S}^{-T} = \text{diag}(1/\sqrt{m_i})$. Astfel, elementele lui \mathbf{R} și \mathbf{x} sunt date de:

$$r_{ij} = \frac{k_{ij}}{\sqrt{m_i} \sqrt{m_j}}, \quad i, j = \overline{1, n},$$

și

$$x_i = \frac{y_i}{\sqrt{m_i}}, \quad i = \overline{1, n}.$$

■

Observație

Matricea \mathbf{R} poate fi scalată, adică se pune $\mathbf{R} = \mathbf{R}' * \text{factor}$. Ecuația (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 pulsațiile 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 și procesează triunghiul superior al matricilor \mathbf{M} și \mathbf{K} , iar al doilea, matricile complete $M(n \times n)$, $K(n \times n)$.

Ambele proiecte lucrează în dublă precizie.

Fișiere (General_R):

Back.f90: substituție înapoi, pentru Cholesky.

Cholesky_S.f90: subrutina Cholesky.

loca.f90: funcția de adresă din Cholesky.

Gen_M.f90, Gen_K.f90: subrutinele de generare a matricii \mathbf{M} , respectiv \mathbf{K} .

Gen_R.f90: subrutina de calcul a matricii \mathbf{R} .

Main-General_R.f90: programul principal.

Write_R.f90: subrutină pentru scrierea matricii \mathbf{R} .

work.f90: modul

GetFile.f90, Openfile.f90, width.f90: utilitare

Compact_mul: subrutină de înmulțire $\mathbf{S} * \mathbf{A}$, unde: \mathbf{S} este simetrică și stocată în

triunghiul superior, iar \mathbf{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 fișier de date (acesta se selectează din fereastra standard). Datele de ieșire se scriu în continuarea datelor de intrare.

Matricile **M** și **K** se introduc conform codurilor *kod_genM* și *kod_genK* (coduri de citire sau generare a matricilor):

- $kod_gen \leq 0$: matrici generate (de subrutinele Gen_M.f90, Gen_K.f90);
- $kod_gen \geq 1$: matrici citite din fișier, anume:
 - $kod_gen = 1$: fișierul de intrare;
 - $kod_gen \geq 2$: fișier specificat (selectat).

Daca matricea este citită: se introduce și "factor"; dacă este generată, nu.

În cazul matricilor citite dintr-un fișier specificat ($kod_gen \geq 2$), fișierul va avea următoarea structură:

- Prima linie din fișier este o linie-text (ignorată la citire);
- Următoarele linii conțin elementele triunghiului superior al matricii.

Datele de intrare (aceleași pentru General_R și 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 $kod_genM \geq 1$, se introduce:

- [*m_factor*: factor de multiplicare pentru **M**; ($\mathbf{M} \rightarrow m_factor * \mathbf{M}$)

Pentru $kod_genK = 1$, se introduc:

- [Matricea **K** – triunghiul superior, pe linii.]

Pentru $kod_genK \geq 1$, se introduce:

- [*k_factor*: factor de multiplicare pentru **K**; ($\mathbf{K} \rightarrow k_factor * \mathbf{K}$)
- *kodPrint*: cod tipărire

Semnificația codurilor:

ngl: ordinul matricii (numărul gradelor de libertate)

kod_gen: v. mai sus

kodPrint: 0 / ≠0: Nu se tipăresc / Se tipăresc: matricile **M**, **K**, și **S**.

Observație

Matricea **R** este *scalată*: adică, în fișierul de ieșire, se tipărește matricea scalată

R' și 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:

$$\mathbf{M} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}; \quad \mathbf{K} = \begin{bmatrix} 3 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

Fișierul 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 $R = S^{(-T)} * K * S^{(-1)}$

```
1.5000000000000000    -1.118033988749895    0.5590169943749473
-1.118033988749895    1.5000000000000000    -1.2500000000000000
0.5590169943749473    -1.2500000000000000    1.5000000000000000
```

Matricea $S^{(-1)}$:

```
0.707106781186547    -0.316227766016838    0.158113883008419
0.632455532033676    -0.316227766016838
0.790569415042095
```

■

Retrieve_Eigen_from_R

Regăsește valorile și vectorii proprii ai problemei generalizate, după găsirea valorilor și vectorilor proprii ai matricii \mathbf{R} .

Fișiere sursă:

Main-Retrieve_Eigen_from_R.f90: programul principal
work-2005.90: modul
GetFile.90, OpenFile.90: fișiere pentru fereastra de selecție standard
width.90: utilitar

Fișiere de date (intrare):

Programul cere ca în folderul de lucru să existe următoarele 4 fișiere:

nume-S^-1.dat: fișierul care conține matricea \mathbf{S}^{-1} (triunghiul superior);
nume-Scal_R.dat: fișierul care conține factorul de scalare.
nume-VAL.dat: fișierul care conține valorile proprii λ ale lui \mathbf{R} ;
nume-VEC.dat: fișierul care conține vectorii proprii \mathbf{y} ai lui \mathbf{R} ;

Specificatorul de *fișier trebuie să aibă forma de mai sus*, în care, ”*nume*” este ales de utilizator (de regulă, numele exemplului).

În fiecare fișier, datele numerice sunt precedate de o line-text.

- Primele două fișiere sunt furnizate (ca fișiere de ieșire) de programul `General_R`.
- Ultimele două fișiere sunt furnizate (ca fișiere de ieșire) de programul de valori proprii QR sau Jacobi.

Date introduse de la terminal:

- Numărul gradelor de libertate (ordinul matricii R)
- Pulsații sau Frecvențe: se introduce T, respectiv F.

Relația între pulsația ω și frecvența f este: $f = \omega/(2\pi)$.

Fișiere de ieșire (rezultate):

nume-OMEGA.dat (pulsații), sau *nume-FRECV.dat* (frecvențe);
nume-VEC_X.dat: vectorii proprii \mathbf{x} , ai problemei generalizate.

Exemplu (v. [exemplul de la General_R](#))

Fișiere 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:

```
1.000000000000000
```

Următoarele fișiere 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 tematic: Interpolation

INTERPOLARE

Newton

Polinomul de interpolare Newton.

Metoda:

Problema

Fie funcția f , cunoscută prin valorile ei pe $n + 1$ puncte distincte x_i , ca în tabelul următor:

x	x_0	x_1	x_2	\dots	x_n
$f(x)$	f_0	f_1	f_2	\dots	f_n

Punctele x_i se numesc *noduri* și 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 funcția 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 funcției f , pe nodurile date.

Se arată că:

Polinomul de interpolare (al funcției f , pe $n + 1$ noduri) există și este unic ■

Forma Newton a polinomului de interpolare

$$p_n(x) = c_0 + c_1(x - x_0) + \dots + c_n(x - x_0) \cdots (x - x_{n-1})$$

Definim polinoamele

$$q_0(x) = 1, \quad q_1(x) = x - x_0, \quad q_2(x) = (x - x_0)(x - x_1), \quad \dots,$$

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 numește *diferența divizată de ordinul k* (a funcției f , pe nodul x_k) și se notează cu

$$c_0 = f[x_0]; \quad c_k = f[x_0, x_1, \dots, x_k], \quad k \geq 1.$$

Cu aceasta, expresia polinomului de interpolare se scrie

$$p_n(x) = \sum_{k=0}^n f[x_0, x_1, \dots, x_k] q_k(x) \quad (1)$$

sau explicit:

$$p_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \dots + f[x_0, \dots, x_n](x - x_0) \dots (x - x_{n-1}) \quad (1')$$

Proprietăți ale diferențelor divizate

P1. Diferența divizată este o funcție simetrică de argumentele sale ■

Adică, considerând o permutare (i_0, \dots, i_n) a numerelor $(1, \dots, n)$, avem:

$$f[x_{i_0}, \dots, x_{i_n}] = f[x_0, \dots, x_n]$$

P2. Formula de recurență a diferențelor divizate:

$$f[x_0, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0} \quad (2)$$

Calculul practic al diferențelor divizate

Notând nodurile cu $x_j, x_{j+1}, \dots, x_{j+n}$, (2) devine:

$$f[x_j, \dots, x_{j+n}] = \frac{f[x_{j+1}, \dots, x_{j+n}] - f[x_j, \dots, x_{j+n-1}]}{x_{j+n} - x_j}$$

și $f[x_j] = f_j$. Utilizând notația simplificată

$$f_{j \ j+1 \dots \ j+n} = f[x_j, x_{j+1}, \dots, x_{j+n}],$$

formula de recurență (2) se scrie:

$$f_{j \ j+1 \dots \ j+n} = \frac{f_{j+1 \dots \ j+n} - f_{j \dots \ j+n-1}}{x_{j+n} - x_j} \quad (3)$$

Astfel, obținem diferențele de ordinul 1, 2, 3, etc.:

$$1: \quad f_{01} = \frac{f_1 - f_0}{x_1 - x_0}; \quad f_{12} = \frac{f_2 - f_1}{x_2 - x_1}; \quad f_{23} = \frac{f_3 - f_2}{x_3 - x_2};$$

$$2: \quad f_{012} = \frac{f_{12} - f_{01}}{x_2 - x_0}; \quad f_{123} = \frac{f_{23} - f_{12}}{x_3 - x_1};$$

$$3: \quad f_{0123} = \frac{f_{123} - f_{012}}{x_3 - x_0};$$

Etc.

Diferențele divizate se pot așeza 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 și diferențele de ordinul 0). Diferențele de ordinul 1, 2, 3 sunt în coloanele 3, 4, 5. Tabloul are structura triunghiulară: datele nu permit calculul diferențelor $f_{j \dots \ j+n}$ cu $j + n = 4$ (care implică nodul x_4).

Polinomul de interpolare Newton, pentru exemplul din tabel, este:

$$p_4(x) = f_0 + f_{01}(x - x_0) + f_{012}(x - x_0)(x - x_1) + f_{0123}(x - x_0)(x - x_1)(x - x_2) \quad (4)$$

Coeficienții polinomului se găsesc în prima linie din tabel (începând cu coloana 2).

Pentru calculul practic al coeficienților polinomului Newton, notăm coeficienții ca în tabloul de mai jos, anume c_{jk} (unde j este indicele nodului, iar k indicele ordinului diferenței):

$$\begin{array}{l|lll} x_0 & c_{00} & c_{01} & c_{02} & c_{03} \\ x_1 & c_{10} & c_{11} & c_{12} & \\ x_2 & c_{20} & c_{21} & & \\ x_3 & c_{30} & & & \end{array}$$

Correspondența este:

$$c_{01} = f_{01}; \quad c_{02} = f_{012}; \quad c_{03} = f_{0123}$$

$$c_{11} = f_{12}; \quad c_{12} = f_{123};$$

$$c_{21} = f_{21};$$

Sau, în general,

$$c_{jk} = f_{j j+1 \dots j+k} \quad (5)$$

Cu aceasta, (4) se scrie:

$$p_4(x) = c_{00} + c_{01}(x - x_0) + c_{02}(x - x_0)(x - x_1) + c_{03}(x - x_0)(x - x_1)(x - x_2)$$

Coeficienții c_{jk} se calculează prin recurență, cu formula care derivă din (2) și (5):

$$c_{jk} = \frac{c_{j+1, k-1} - c_{j, k-1}}{x_{j+k} - x_j} \quad (6)$$

Fișiere:

Pol-newton.f90: programul principal

Fun.f90: funcția f

Programul principal implementează calculul coeficienților c_{jk} conform (6), pentru funcția definită în fun.90. În afară de aceasta, programul calculează erorile relative ale polinomului de interpolare pe niște valori date z_i , adică $(f(z_i) - p(z_i))/f(z_i)$; dacă $|f(z_i)| \leq 1E-6$, se listează eroarea (în loc de eroarea relativă).

Subprogramul `fun(x)` returnează valorile funcției f pe x . (Funcția este numită `fun`, și valorile ei sunt stocate, în programul principal, în tabloul `f(0:n)`.)

Datele de intrare se citesc dintr-un fișier de date, care conține datele:

p) *Titlu*: text, max. 80 caractere

q) $n1$: numărul de noduri. ($n1 = n + 1$)

r) $x(0:n)$: nodurile x_i : $n1$ valori

s) nz : numărul de valori z_i

t) $z(1:nz)$: valorile z_i : nz valori

.

Datele de ieșire constau în: valorile funcției f pe noduri; coeficienții c_{jk} ; valorile polinomului $p(z)$; erorile $f(z_i) - p(z_i)$. (Dacă $|f(z_i)| \leq 0.2$, în loc de eroarea relativă se listează eroarea.)

Notă

Programul are scopul de a lista valorile și erorile polinomului de interpolare (în general, pe alte puncte decât nodurile).

Programul se poate modifica ușor, astfel ca să aibă ca intrare valorile funcției pe noduri, adică valorile $f(0:n)$. (În loc ca acestea să fie calculate de funcția definită în `fun.f90`.)

■

Exemplu:

Considerăm funcția $f(x) = e^x - 3x^2$, și nodurile $-1, 0, 1, 2, 3, 4$. Listăm valoarea și eroarea polinomului, pe mijloacele intervalelor definite de noduri: $-0.5; 0.5; 1.5; 2.5; 3.5$.

Fișierul de intrare/ieșire este:

```
Funcția 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
1.500000	-2.242733	1.128E-02
2.500000	-6.608934	-6.308E-03
3.500000	-3.488262	4.025E-02

■

Graficele funcției și polinomului de interpolare sunt date mai jos.

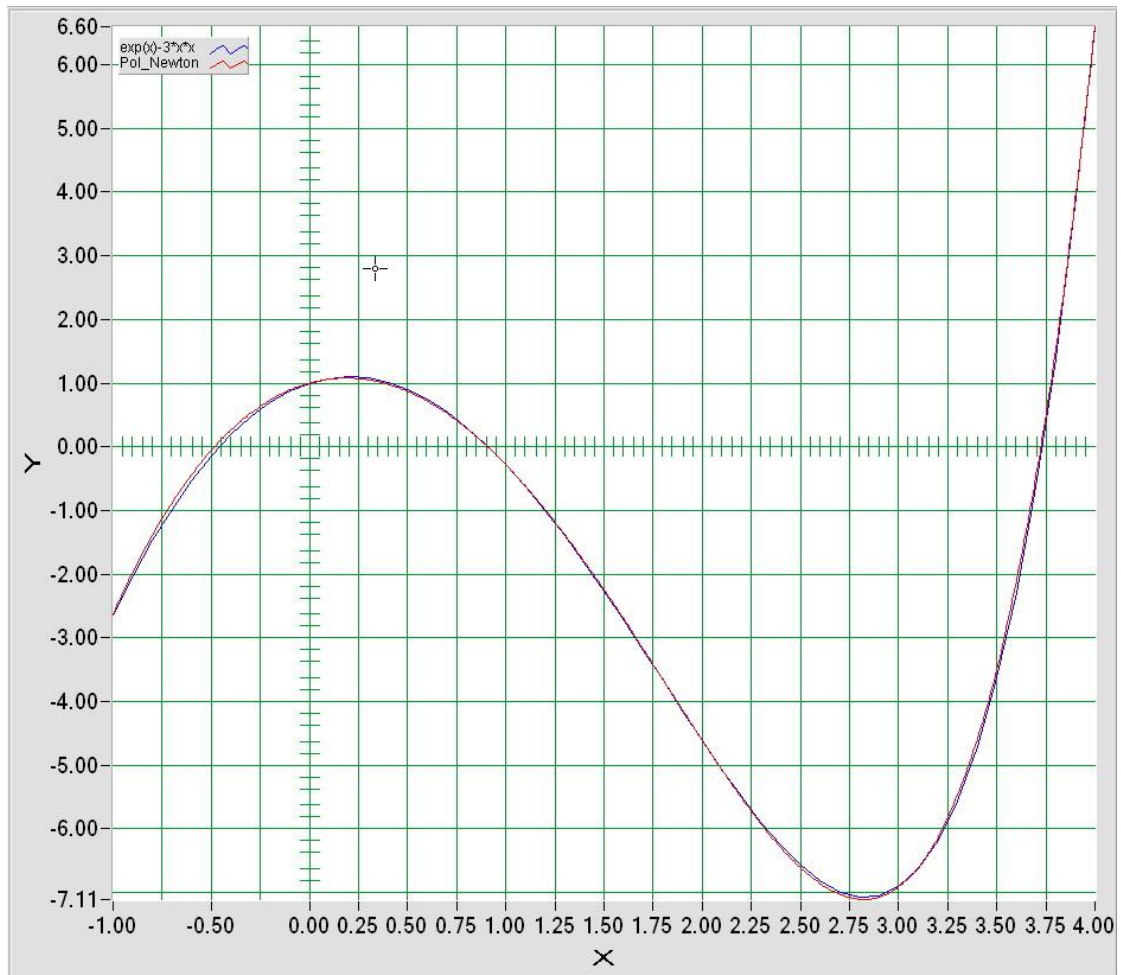


Figura 8 – Interpolare: graficul funcției și polinomului de interpolare

UTILITARE

Folder tematic: \$ Utilities

Function_expression și width

Function_expression.f90; Function_expression_g.f90;

Function_expression-2.f90; Function_expression_Sys.f90

Aceste fișiere conțin subrutine apelate de programul principal. Ele citesc și returnează expresia funcției/funțiilor cu care se lucrează; expresiile sunt citite (în cod Fortran) din fișierul sursă în care sunt definite funcțiile. Specificatorul fișierului sursă este definit în programul principal.

Programului principal scrie expresiile funcțiilor, în fișierul de ieșire și 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 ecuații de forma $f(x) = 0$ (Bis, Secant).

Parametri:

f_source: specificator de fișier sursă pentru funcția f

f_text: expresia funcției f

F_Expr_g(g_source, g_text)

La metode pentru ecuații de forma $g(x) = 0$ (Fix, Aitken)

Parametri:

g_source: specificator de fișier sursă pentru funcția g

g_text: expresia funcției g

F_Expr(f_source, f_text)

La metode pentru ecuații de forma $f(x) = 0$, care lucrează și cu derivata f' (Newton).

Parametri:

`f_source`: specificator de fișier sursă pentru funcția f și f'

`f_text`: tablou de dimensiune 2. Expresiile funcțiilor f și f'

`F_Expr_Sys(f_source, n, f_text)`

La metode pentru sisteme de ecuații neliniare (`Fix_Sys`, `Newton_Sys`).

Parametri:

`f_source`: specificator de fișier sursă

`n`: ordinul sistemului de ecuații neliniare

`f_text`: tablou de dimensiune n . Expresiile funcțiilor f_i sau g_i , $i = \overline{1, n}$

Notă

Instrucțiunea de definire a unei funcții are forma `f = expresie` (respectiv, `f1 = expresie` pentru derivata lui f). Caracterul `f` (sau `f1`) poate fi precedat sau succedat, în linia instrucțiune, de spații sau tab-uri; acestea se omit la afișare.

Dacă utilitarul nu găsește, în fișierul sursă specificat, o astfel de expresie a funcției, se afișează `f = ?` (respectiv, `f1 = ?`).

■

width.f90

Acest utilitar returnează numărul de cifre `iw` dintr-o variabilă întreagă. Servește la tipărirea valorilor întregi cu format în execuție de tip `I<iw>`.

`integer function width(n)`

Parametri:

`n`: variabilă întreagă

Notă

În programul apelant, trebuie inclusă declarația `integer width` ■

Grafic

Calculul valorilor unei funcții $f(x)$, pe un interval $[a, b]$, cu pasul h .

Programul are ca fișier de ieșire valorile x, y , unde $x = a + jh$, $y = f(x)$

Fișierul de ieșire se va utiliza într-un program de grafică – pentru reprezentarea graficului lui f . V. de exemplu, programul GRAPH.

În pofida simplității lui, flexibilitatea oferită de program, îl face foarte util pentru scopul propus.

Fișiere:

grafic.f90: Subrutina de calcul a valorilor $f(x)$

Mgrafic.f90: Programul principal: singura lui funcțiune este de declara cu atributul external numele funcțiilor (definite în funcție.f90), și de a apela subrutina grafic pentru una (sau mai multe) dintre aceste funcții.

funcție.f90: definește funcțiile de lucru; expresia unei funcții poate include un parametru p ;

par.f90: modul pentru parametrul funcției.

Datele de intrare se introduc de la terminal și constau în:

- u) Există $p =$ dată caracter: Υ sau \underline{y} , dacă funcția include un parametru p ; orice alt caracter, în caz contrar.
- v) [p , dacă există parametru]
- w) valorile a, b, h
- x) nume fișier de ieșire (fără extensie: programul pune extensia .dat)

Observații

- y) Graficul unei curbe, definită prin ecuația implicită $F(x, y) = 0$: se rezolvă în raport cu y , gășind $y = f(x)$. Dacă există două soluții $y = f_1(x)$ și $y = f_2(x)$, se vor construi fișierele pentru fiecare din funcțiile f_1 și f_2 . Se reprezintă apoi, ambele funcții pe același grafic.

Exemplu: $x^2 + y^2 = 1$; se obține: $f_1(x) = \sqrt{1-x^2}$ și $f_2(x) = -\sqrt{1-x^2}$.

- z) Se pot construi, în aceeași rulare, fișierele de ieșire pentru mai multe funcții:

Pentru aceasta, în Mgrafic, se pun mai multe apeluri `call grafic(f1), ..., call grafic(fn)`, pentru funcțiile f_1, \dots, f_n .

Datele se introduc succesiv, pentru fiecare funcție (în ordinea apelurilor) ■

Bibliografie

1. Chisăliță 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. “Intel Visual Fortran Compiler 11.x for Windows”, 2010
<http://software.intel.com/en-us/articles/fortran-compilers/>
25. “High-Precision Software Directory”, 2012,
<http://crd-legacy.lbl.gov/~dhbailey/mpdist/>

Lista Figurilor

Figura 1 – Graficul funcției $f(x) = e^x - 3x^2$	25
Figura 2 - Proces staționar în metoda Newton.....	34
Figura 3 - Proces staționar – Graficul iteratelor.....	39
Figura 4 - Proces staționar – Graficul funcției și primei bisectoare.....	40
Figura 5 – Determinarea aproximațiilor inițiale	46
Figura 6 – Metoda Muller: graficul funcției și polinomului de interpolare.....	68
Figura 7 - Tablourile Y1 și YA.....	99
Figura 8 – Interpolare: graficul funcției și polinomului de interpolare	190

Index

- Accelerarea Aitken, 40
- Aitken cu funcția $G(x)$, 43
- Algoritmul QR
 - algoritm și cod, 152
 - balansare, 155
 - implementare, 153
 - metoda, 148, 175, 181
- Bessel, 13
- Bisecția, 25
- Cholesky
 - cu matricea L, 94
 - cu matricea S, 97
 - matrici bandă, 97
- Descompunerea LU
 - calculul determinantului, 85
 - calculul determinantului, 85
 - Crout, 91, 93
 - Doolittle, 83, 91
 - matrice complexă, 91
- Eliminarea Gauss, 76, 81
- Exemplul Muller-Kahan, 14
- Funcția Bessel, 13
- Grafic, 193
- Graficul funcției, 25
- Interpolare
 - polinomul Newton, 184
- Laguerre, 61, 62
- Matricea Hilbert, 104
- Matrici hermitiene și unitare, 117
- Metoda iterațiilor simultane, 136
- Metoda Jacobi (valori proprii), 142
- Metoda Muller
 - rădăcini reale, 64
 - rădăcinile unei funcții complexe, 68
- Metoda Newton
 - o ecuație, 30
 - sisteme de ecuații, 49
- Metoda punctului fix
 - o ecuație, 35
 - sisteme de ecuații neliniare, 46, 49
- Metoda puterii
 - matrice complexă, 128
 - matrice reală, 121
 - teste de oprire a iterației, 122
- Metoda puterii puterii inverse cu
 - translație
 - matrice complexă, 133
 - matrice reală, 129
- Metode iterative
 - Gauss-Seidel, 109, 110
 - Jacobi, 106
 - SOR, 111
- Număr de condiție, 102
- Pierderea de semnificație, 15, 16
- Pivotarea parțială, 78
- Polinoame
 - calculul valorii, 51
 - POL (reducerea gradului), 54
 - Pol_Complex (rădăcini complexe), 59, 61
 - Pol_Direct (iterare în polinomul original), 57

- Polinomul lui Rump, 17
- Polinomul Rump # 2, 18
- Problema de valori proprii
 - definiții, 114
 - polinom caracteristic, 114
 - problema generalizată, 175, 181
 - subspațiu propriu, 116
- Proces staționar
 - în metoda Newton, 33, 34
 - în metoda punctului fix, 38
- Produs scalar
 - spațiu vectorial complex, 119
 - spațiu vectorial real, 118
- Recurență, 13
- Secanta, 27
- SSH, 20
- Sumare fără erori, 22
- ULP, 23
- Utilitare, 191
- Valori speciale, 19