Numeričke metode

Aleksandar Maksimović
IRB

Sadržaj

- Greške u numeričkom računanju
- Uvod u F90 (ili F77/C++)
- Sustavi linearnih jednadžbi
- Interpolacija i Ekstrapolacija
- Numerička integracija
- Rješavanje nelinearnih jednadžbi
- Obične diferencijalne jednadžbe

Greške u numeričkom računu

- Tipovi grešaka:
  - Greške zbog aproksimacija
    - Greške modela (zanemarivanje otpora zraka)
    - Greške diskretizacije (truncation error), zamjena integrala konačnom sumom, zamjena tangente sekantom i slično.
    - Greške u polaznim podacima
  - Greške zaokruživanja, (roundoff errors) konačna preciznost u računanju dovodi do zaokruživanja vrijednosti.
    - greške nastaju u računalima, jer ona koriste konačnu aritmetiku ili preciznije binarnu aritmetiku s pomičnom točkom, kod koje je unaprijed rezerviran određen broj binarnih mjesta za eksponent i za mantisu

Aritmetika s pomičnom točkom

Na kalkulatorima se često može izabrati tzv. "znanstvena notacija" brojeva, koja npr. broj –27.77 prikazuje kao $-2.777 \cdot 10^1$ pri čemu je – predznak broja (ili mantise), . je decimalna točka, 2.777 je mantisa, koja se još zove signifikantni ili razlomljeni dio broja, 10 je baza, a 1 je eksponent.

$$\begin{aligned} x &= -(2 \cdot 10^1 + 7 \cdot 10^0 + 7 \cdot 10^{-1} + 7 \cdot 10^{-2}) \\ &= -(2 \cdot 10^0 + 7 \cdot 10^{-1} + 7 \cdot 10^{-2} + 7 \cdot 10^{-3}) \cdot 10^1, \end{aligned}$$

binarna reprezentacija

$$\begin{aligned} y &= (11.1011)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= (1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5}) \cdot 2^1 \\ &= (1.11011)_2 \cdot 2^1. \end{aligned}$$

reprezentacija realnih brojeva

U računalu se realni brojevi reprezentiraju pomoću binarne reprezentacije u "znanstvenoj notaciji"

$$x = \pm m \times 2^e, \quad \text{gdje je} \quad 1 \leq m < 2.$$

Stoga je

$$m = (b_0.b_1b_2b_3\ldots b_{p-1})_2 \quad \text{pri čemu je} \quad b_0 = 1.$$

broj kao suma cijelog i razlomljenog dijela

$$\begin{aligned} x &= (a_k a_{k-1}\cdots a_1 a_0.b_1 b_2\cdots b_{l-1}b_l)_2 = x_c + x_r, \\ x_c &= (a_k a_{k-1}\cdots a_1 a_0)_2 \\ &= a_k\cdot 2^k + a_{k-1}\cdot 2^{k-1} + \cdots + a_1\cdot 2^1 + a_0\cdot 2^0, \\ x_r &= (.b_1 b_2\cdots b_{l-1}b_l)_2 \\ &= b_1\cdot 2^{-1} + b_2\cdot 2^{-2} + \cdots + b_{l-1}\cdot 2^{-(l-1)} + b_l\cdot 2^{-l}. \end{aligned}$$

pretvaranje realnog broja

algoritam za računanje binarnog prikaza realnog broja
u pseudokodu i dio algoritma implementiranog u fortranu

```
y := xc;
k := −1;
repeat
k := k + 1;
a(k) := mod(y, 2);
y := div(y, 2);
until y = 0;
```

```
z := xr;
l := 0;
repeat
l := l + 1;
zz := 2      z;
b(k) := int(zz);
z := zz − b(k);
until z = 0 or l > lmax
```

```
      aint = Int(a)
      adec = a-aint

Do i=1,50
      v(i)    = Mod(aint,two)
      nb(i-1) = Nint(v(i))
      aint    = 0.5D0*(aint-v(i))
      If(aint.le.0.00001) Go to 97
    End Do
97 Continue
```

pretvaranje realnog broja

pretvaranje broja 111.1 u binarni oblik

111	55	27	13	6	3	1	0
1	1	1	1	0	1	1	.

$111 = (110111)_2.$ krivo napisano (1101111)

0.1	0.2	0.4	0.8	1.6	1.2	0.4	0.8	1.6	1.2
.	0	0	0	1	1	0	0	1	1

$$0.1 = (0.0001100110011001100\ldots)_2 = (0.0\dot{0}01\dot{1})_2,$$

$$(111.1)_{10} = (1101111.0001100110011001100\ldots)_2 = (1101111.0\dot{0}01\dot{1})_2.$$

Primjeri grešaka

- Promašaj rakete patriot

U Zaljevskom ratu, 25. veljače 1991. godine, Patriot rakete iznad Dhahrana u Saudijskoj Arabiji nisu uspjele pronaći i oboriti iračku Scud raketu. Projektil je (pukim slučajem) pao na američku vojnu bazu usmrtivši 28 i ranivši stotinjak ljudi.

Izvještaj o katastrofi godinu dana poslije, rasvijetlio je okolnosti nesreće. U računalu koje je upravljalo Patriot raketama, vrijeme se brojilo u desetinkama sekunde proteklim od trenutka kad je računalo upaljeno. Kad desetinku prikažemo u binarnom prikazu, dobivamo

$$0.1_{10} = (0.0\dot{0}011\dot{1})_2.$$

Realne brojeve u tom računalu prikazivali su korištenjem nenormalizirane mantise duljine 23 bita. Spremanjem 0.1 u registar Patriot računala, napravljena je (apsolutna) greška približno jednaka $9.5 \cdot 10^{-8}$.

Primjeri grešaka

Zbog stalne opasnosti od napada Scud raketama, računalo je bilo u pogonu 100 *sati, što je* $100 \cdot 60 \cdot 60 \cdot 10$ *desetinki sekunde. Ukupna greška nastala greškom zaokruživanja je*

$$100 \cdot 60 \cdot 60 \cdot 10 \cdot 9.5 \cdot 10^{-8} = 0.34\,\mathrm{s}.$$

Ako je poznato da Scud putuje brzinom $1676\,\mathrm{m/s}$ *na predviđenoj visini susreta, onda su ga rakete Patriot pokušale naći više od pola kilometra daleko od njegovog stvarnog položaja.*

Tako je sistemski sat mjerio vrijeme u desetinkama sekunde, ali se vrijeme spremalo kao cijeli broj desetinki proteklih od trenutka kad je računalo upaljeno. Za sve ostale proračune, vrijeme se računa tako da se pomnoži broj desetinki n *s osnovnom jedinicom* $t_0 = 0.1\,\mathrm{s}$ *u kvazi-cjelobrojnoj aritmetici, a zatim pretvori u pravi 24-bitni floating point broj.*

Primjeri grešaka

- Eksplozija Ariane 5 str28

Raketa Ariane 5 lansirana 4. lipnja 1995. godine iz Kouroua (Francuska Gvajana) nosila je u putanju oko Zemlje komunikacijske satelite vrijedne oko 500 milijuna USD. Samo 37 sekundi nakon lansiranja izvršila je samouništenje.

Dva tjedna kasnije, stručnjaci su objasnili događaj. Kontrolna varijabla (koja je služila samo informacije radi) u programu vođenja rakete mjerila je horizontalnu brzinu rakete. Greška je nastupila kad je program pokušao pretvoriti preveliki 64-bitni realni broj u 16-bitni cijeli broj. Računalo je javilo grešku, što je izazvalo samouništenje. Zanimljivo je da je taj isti program bio korišten u prijašnjoj sporijoj verziji Ariane 4, pa do katastrofe nije došlo.

Primjeri grešaka

- Potonuće naftne platforme Sleipner

Naftna platforma Sleipner A potonula je prilikom sidrenja, 23. kolovoza 1991. u blizini Stavangera. Baza platforme su 24 betonske ćelije, od kojih su 4 produljene u šuplje stupove na kojima leži paluba. Prilikom uronjavanja baze došlo je do pucanja. Rušenje je izazvalo potres jačine 3.0 stupnja po Richterovoj ljestvici i štetu od 700 milijuna USD.

Greška je nastala u projektiranju, primjenom standardnog paketa programa, kad je upotrijebljena metoda konačnih elemenata s nedovoljnom točnošću (netko nije provjerio rezultate programa). Proračun je dao naprezanja 47% manja od stvarnih. Nakon detaljne analize s točnijim konačnim elementima, izračunato je da su ćelije morale popustiti na dubini od 62 metra. Stvarna dubina pucanja bila je 65 metara!

Izabran je pogrešan predsjednik

Možda je najbizarniji primjer da greška zaokruživanja može poremetiti izbore za predsjednika SAD. U američkom sustavu izbora predsjednika, svaka od saveznih država ima određen broj predstavnika (ljudi) u tijelu koje se zove Electoral College i koje formalno bira predsjednika. Broj predstavnika svake pojedine države u tom tijelu proporcionalan je broju stanovnika te države u odnosu na ukupan broj stanovnika. Pretpostavimo da u Electoral College-u ima a predstavnika, populacija SAD je p stanovnika, a država i ima p_i stanovnika. Broj predstavnika države i u Electoral College-u trebao bi biti

$$a_i = \frac{p_i}{p} \cdot a.$$

Ali, predstavnici su ljudi, pa bi a_i morao biti cijeli broj. Zbog toga se a_i mora zaokružiti na cijeli broj $\hat{a}_i$ po nekom pravilu. Naravno, na kraju mora biti $\sum_i \hat{a}_i = a$. Razumno i prirodno pravilo je:

- *$\hat{a}_i$ mora biti jedan od dva cijela broja koji su najbliži a_i (tzv. "uvjet kvote").*

Primjeri grešaka

- Izabran je pogrešan predsjednik

Naime, pravilno zaokruživanje (kao kod prikaza brojeva) je možda najpravednije, ali ne mora dati $\sum_i \hat{a}_i = a$*, pa se mora upotrijebiti slabije pravilo.*

Međutim, broj stanovnika p_i *se vremenom mijenja (a time i* p*). Isto tako, ukupni broj predstavnika* a *u tijelu se može promijeniti od jednih do drugih izbora. Zbog toga se dodaju još dva prirodna "politička" pravila:*

- *Ako se poveća ukupan broj predstavnika* a*, a svi ostali podaci se ne promijene,* $\hat{a}_i$ *ne smije opasti (tzv. "monotonost predstavničkog tijela").*
- *Ako je broj stanovnika države* p_i *porastao, a ostali podaci su nepromijenjeni,* $\hat{a}_i$ *ne smije opasti (tzv. "monotonost populacije").*

Svrha je jasna, jer ljudi vole uspoređivati prošle i nove "kvote". Nažalost, ne postoji metoda za određivanje broja predstavnika koja bi zadovoljavala sva tri kriterija.

Primjeri grešaka

- Izabran je pogrešan predsjednik

U američkoj povijesti zaista postoji slučaj da je izabran "pogrešan" predsjednik. Samuel J. Tilden izgubio je izbore 1876. godine u korist Rutherforda B. Hayesa, samo zbog načina dodjele elektorskih glasova u toj prilici. Da stvar bude još zanimljivija, ta metoda dodjele glasova nije bila ona koju je propisivao zakon iz tog vremena.

Kompajleri

- Linux
  - ovisno o distribuciji
    - gnu kompajleri za C,f77 kod starijih distribucija
    - dominis.phy.hr ima gcc i g++ kompajlere za c i c++, te f77 za fortran.
    - fedora core 4, gcc, g++, ver. 4.02. Fortran kompajler gfortran
- Intel (linux)
  - besplatni kompajleri, registracija je obavezna

Kompajleri

- odlična dokumentacija, laka instalacija
- ftp site: ftp://download.intel.com/software/products/compilers/downloads/
- Free software for linux open source developers: registracija http://www.intel.com/cd/software/products/asmo-na/eng/219771.htm
- Fortran kompajler je ifort, icc je kompajler za C i C++
- debuger je idb, a najnovija verzija 9.0.x ima i eclipse kao grafičko sučelje za programiranje.
- integrira se sa gnu kompajlerima

Kompajleri osnovne opcije

- Gnu i Intel kompajleri imaju iste osnovne opcije
  - gcc file.c napravi a.out program za izvršavanje
  - gcc -o exefile file.c, exefile je rezultat
  - gcc -c file.c, napravi object file file.o
  - gcc -o exefile file.o, link object file i napravi exefile
  - gcc -I /includefiles/ -I/josinclude -L/libraries -o exef file.c -lml, pri kompajliranju file.c uključi header fileove iz direktorija navedenih nakon -I i linka s libml.a datotekom.

Kompajleri osnovne opcije

- Gnu i Intel kompajleri imaju iste osnovne opcije
  - -c - compile
  - -o - konačni program
  - -L - mjesto gdje se nalaze library (.a na linuxu, .so dynamicki)
  - -I - mjesto dodatnih include fileova
  - -O[X] - optimizacija gdje je X=2,3 itd. ovisno o kompajleru
  - -E - preprocesiranje
  - -g - uključi informacije za debug u object file
  - -- help pomoć za gnu kompajlere, info program sadrži dodatne informacije o gnu kompajlerima i datotekama

Kompajleri osnovne opcije

- -help pomoć za Intel kompajlere, kompletan opis opcija i jezika u html i pdf formatu
- DEBUG
  - GNU, gdb, linijski editor
    - ddd, grafičko sučelje za gdb
    - eclipse kod novijih distribucija (IDE)
  - INTEL
    - idb, linijski editor, eclipse (IDE)

primjeri programa f77

- hello program: hello.f

```
      program fortranprint
c komentar
      write(*,*)  "Hello World!"
c prolazi i print *, "string"
      end program
```

- pretvara decimalni broj u binarni oblik: binary.f

EXE file: g77 -o hellof77 hello.f

primjeri programa f90

- hello program: hello.f90

```
program fortranprint
c komentar
      print *,  "Hello World!"
      end program
```

- površina trokuta: triangle.f90

EXE file: gfortran -o hellof90 hello.f90

ili neki drugi kompajler (ifort) ili f90 od NAG grupe.

primjeri programa C

- hello.c

```
#include <stdio.h>
int main()
{  printf("Hello World!\n");
return 0; }
```

EXE file: gcc -o helloC hello.c

primjeri programa C++

- helo.cc

```
#include <iostream>
using namespace std;

int main()
{ cout << "Hello, World\n";
return 0; }
```

EXE file: g++-3.3 -o helloCC hello.cc

make i makefile

- Makefile se može koristiti za specificiranje opcija kod kompajliranja
  - sve naredbe s komandne linije se mogu koristiti (shell)
  - ima jednostavnu strukturu
  - VAR= lista , $(VAR) - ekspandira listu

```
all: EXEfile
CC=gcc
SRC=file.c
```

```
file.o: file.c
        $(CC) -c file.c
EXEfile: file.o
        $(CC) -o file.EXE fileo.o -L/libs -lm
```

Top 10 algoritama

- Dongarra & Sullivan sastavili su listu najvažnijih algoritama

1. **1946:** The Monte Carlo method for modeling probabilistic phenomena.
2. **1947:** The Simplex method for linear optimization problems.
3. **1950:** The Krylov subspace iteration method for fast linear solvers and eigensolvers.
4. **1951:** The Householder matrix decomposition to express a matrix as a product of simpler matrices.
5. **1957:** The FORTRAN compiler that liberated scientists and engineers from programming in assembly.
6. **1959-1961:** The QR algorithm to compute many eigenvalues.
7. **1962:** The Quicksort algorithm to put things in numerical or alphabetical order fast.

Top 10 algoritama

8. **1965:** The Fast Fourier Transform to reduce operation count in Fourier series representation.
9. **1977:** The Integer relation detection algorithm, which is useful for bifurcations and in quantum field theory.
10. **1987:** The Fast multipole algorithm for N-body problems.

Fortran kratka povijest

- 1950- *IBM Mathematical FORmula TRANslation System*

(prvi jezik “višeg stupnja”)

- do 1963 bilo je 40 različitih prevodilaca (compilers)
- 1972 prvi standard “FORTRAN 66”
- 1980 izašao “FORTRAN 77”
- FORTRAN 90 sadrži FORTRAN 77 kao podskup
- FORTRAN 95 (High Performance Fortran (HPF))

F77 nedostaci/karakteristike

- Prvih pet stupaca rezervirani;
- 6 stupac rezerviran za oznaku nastavka prethodnog reda;
- Komentari se inicijaliziraju slovom u prvom stupcu;
- Samo velika slova;
- Imena varijabli mogu imati do 6 znakova.
- Komentari moraju biti u posebnim redovima;
- Nema dinamičko spremanje (dimenzije polja su fiksne)
- Korisnik ne može definirati svoj tip varijable
- Ne podržava rekurziju

F90 nove osobine

- dinamičko alociranje memorije
- moduli
- pointeri
- slobodni format
- izvedene strukture
- CASE naredba
- IMPLICIT NONE
- kompatibilan sa f77

F95 nove osobine

- naredba FORALL
- inicijalizacija izvedenih struktura
- bolja kompatibilnost sa IEEE aritmetikom
- A CPU_TIME subroutine
- A function NULL to nullify a pointer
- automatska dealokacija alociranog niza iz bloka programa
- incijalizacija pointera
- itd...

f90

Fortran 90 podržava slobodan format:

- 132 znaka po redu;
- Proširen skup znakova;
- `&' znak nastavka reda;
- `!' početak komentara;
- `;' razdvajanje naredbi;
- significant (blanks).

f90 imena

- A, aAa, INCOME, Num1, N12O5, under_score ! O.K.
- Slijedeće deklaracije su nevažeće zbog nevažećih imena:
  - INTEGER :: 1A ! Ne počinje sa slovom
  - INTEGER :: A_name_made_up_of_more_than_31_letters !
  - INTEGER :: Depth:0 ! Sadrži nevažeći znak ":"
  - INTEGER :: A-3 ! Interpretira se kao A minus 3
  - CHARACTER(LEN=12) :: user_name! O.K.
  - CHARACTER(LEN=12) :: username ! O.K.

f90 deklaracija varijabli

```
CHARACTER(LEN=10) :: name
CHARACTER :: sex
CHARACTER(LEN=32) :: str
CHARACTER(LEN=10) :: name, spol*1, str*32
CHARACTER(LEN=10), DIMENSION(10,10) :: tom(10)*2, harry(10,10,10)*20
CHARACTER, POINTER :: P2ch

INTEGER :: NumBabiesBorn = 0 ! Zbroj novorođenčadi
REAL :: TimeElapsed = 0.0
LOGICAL :: NHS !
REAL, PARAMETER :: pi = 3.141592
REAL :: radius = 3.5
REAL :: circum = 2 * pi * radius
INTEGER :: a(1:4) = (/1,2,3,4/)

COMPLEX :: val ! x + iy
```

f90 implicitna deklaracija

- Ako se varijabla upotrebljava bez deklaracije onda je tip određen prvim slovom
  - I, J, K, L, M i N su tipa integer;
  - Sva druga slova su tipa real.
- To je opasno jer krivo utipkavanje stvara novu varijablu koje korisnik nije svjestan .
- IMPLICIT NONE !isključuje implicitnu deklaraciju

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - A F90 Tutorial, 1993., (Z. Dodson, Univ. of New Mexico)
  - Harold W. Schranz (html):
    NUMERICAL METHODS: FORTRAN & MATHEMATICA
  - C Pozirkidis: C++ for Fortran Programmers, lecture notes
  - George W. Collins, II:
    Fundamental Numerical Methods and Data Analysis

Literatura

- Michael Metcalf: Fortran 90/95 Explained, Oxford University Press (2002), ISBN 0-19-850558-2
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- S. D. Conte, Carl de Boore: ELEMENTARY NUMERICAL ANALYSIS, McGraw-Hill Book Company, (1980).

Zadaci za praktikum

(a) Skompajlirajte i uvjerite se da radi barem jedan od hello programa iz teksta

   i. Promjenite poruku koju ispisuje na ekranu, npr. neka ispiše "Numeričke metode, Prezime Ime"

(b) Skompajlirajte i uvjerite se da radi program triangle

(c) Promjenite program triangle tako da računa površinu kruga

(d) Pošaljite mail na Aleksandar.Maksimovic@irb.hr koji za attachmente ima source ova dva programa i napišite koji bi programski jezik voljeli koristiti.

(e) Dodatni programi: prime (c,cc,f) da li je broj prost (prime) broj, Overflow.f dodavanje cijelog broja najvećem cjelom broju i IntegerDivide.cc primjer krivog pridruživanja (treba napisati output).

triangle.f

```
FUNCTION Area(x,y,z)
    REAL  x, y, z
    REAL theta, height
    theta = ACOS((x**2+y**2-z**2)/(2.0*x*y))
    height = x*SIN(theta)
    Area = 0.5*y*height
    END FUNCTION Area


    PROGRAM Triangle
    REAL  a, b, c
    PRINT *, 'Welcome, please enter the lengths of the 3 sides.'
    READ    *, a, b, c
    PRINT *, 'Triangle"s area: ', Area(a,b,c)
    END PROGRAM Triangle
```

triangle.f90

```
MODULE TriangleOperations
   IMPLICIT NONE
CONTAINS
  FUNCTION Area(x,y,z)
    REAL :: Area ! function type
    REAL, INTENT( IN ) :: x, y, z
    REAL :: theta, height
    theta = ACOS((x**2+y**2-z**2)/(2.0*x*y))
    height = x*SIN(theta); Area = 0.5*y*height
  END FUNCTION Area
END MODULE TriangleOperations
```

```
PROGRAM Triangle
use TriangleOperations

   IMPLICIT NONE
   REAL :: a, b, c
   PRINT *, 'Welcome, please enter the&
                &lengths of the 3 sides.'
   READ    *, a, b, c
   PRINT *, 'Triangle"s area: ', Area(a,b,c)
END PROGRAM Triangle
```

triangle.c

```
#include <stdio.h>
#include <math.h>
#define real float

real  Area(float x,float y,float z)
{
  real theta, height,povrsina ;
  theta = acos((x*x+y*y-z*z)/(2.0*x*y)) ;
  height = x*sin(theta);
  povrsina = 0.5*y*height ;
  return povrsina;
}
```

```
int main()
{  real  a, b, c;
  printf("Welcome, please enter the
lengths of the 3 sides.\n");
  scanf("%f,%f,%f" , &a, &b, &c);
  printf("Triangle"s area: %f",
Area(a,b,c));
  return 1;}
```

triangle.cc

```
#include <iostream>
#include <math.h>
using namespace std;
#define real float

int main()
{  real  a, b, c;
   cout << "Welcome, please enter the lengths of the 3 sides.\n" << endl;
   cin >> a >> b >> c;
   cout << "Triangle"s area: " << Area(a,b,c) << endl ;
   return 1; }

real  Area(float x,float y,float z)
{
  real theta, height,povrsina ;
  theta = acos((x*x+y*y-z*z)/(2.0*x*y)) ;
  height = x*sin(theta);
  povrsina = 0.5*y*height ;
  return povrsina; }
```

Uvod

- Podijeljeni smo u izboru jezika
  - 4 glasa C (u kombinaciji C++ i Mathematica)
  - 4 glasa F77
  - 1 glas neiskorišten
- Fortran je dugo vremena bio standard za matematičko programiranje
- Puno software-a napisan u F77 (npr. BLAS, LAPACK)
- C/C++ postaje dominantan jezik
- C i F77 mogu se povezati preko objektnog koda

Razlike f77 i C jezika

C jezik razlikuje mala i velika slova, npr. SUM i sum nije isto. Kod f77 u pitanju je ista funkcija. C ima samo funkcije. void funkcija odgovara podprogramima u f77. Kompajler f77 dodaje znak _ na ime funkcije.

C	*F77*
Nastavak je .c	Nastavak je .f ili .f77
Linija se terminira znakom ;	Nema posebnog znaka za kraj linije
Funkcije su definirane zagradama { ... }	Funkcije (podprogrami) završavaju END komandom
Slobodan format	Kod počinje u 7 koloni
Argumenti funkcija "by value" (kopiranje)	Argumenti funkcija "by reference"
Dinamičko alociranje memorije	Samo statičko alociranje
Indeksi polja počinju od 0	Indeksi polja počinju od 1
Matrice se spremaju "row major"	Matrice se spremaju "column major"
Komentar se nalazi između /* i */	Komentar počinje slovom c u prvoj koloni

tipovi varijabli

C	F77
int	integer
float	real
double	double precision
char	character

ekvivalentne naredbe kod F77 i C

Fortran 77	C

c ----------
c AUTHOR: L.M. Sophistis
c ----------
program main
. . . . . .
end
	/* ----------
	AUTHOR: L.M. Sophistis
	---------- */
	int main()
	{
	. . . . . .
	return 0;
	}

Dimension a(0:29),b(0:4)
Double precision a(0:9)
Double precision b(0:5,0:67)
Integer argos(100)
Integer WEDNESDAY
Parameter(WEDNESDAY=134)
Double precision a(0:39)
	float a[30], b[4];
	double a[10];
	double b[6][68];
	int argos[100];
	const int WEDNESDAY= 0;
	
	const int dim = 40;
	double a[dim];

Dimension b(0:2), A(0:1,0:1) b(0) = 0.1 b(1) = 0.3 A(0,0) = 0.1 A(0,1) = 0.2 A(1,0) = 0.3 A(1,1) = 0.4	b[3] = 0.1, 0.3 A[2][2] ={{0.1, 0.2},{0.3, 0.4}};
Stop	exit(1);

ekvivalentne naredbe/kontrole toka

Fortran 77	C

Do i=1,n
a = a+3
b = b+4
End Do
	for (i=1;i<=n;i++)
	{ a = a+3;
	 b+=4;
	}

Do i=1,n a = a+3 End Do	for (i=1;i<=n;i++) a = a+3; /* samo jedna linija */

Do i=j,k,m
. . . . . .
 	
End Do
	for (i=j;i<=k;i=i+m)
	{
	. . . . . .
	}

Do while (i.ne.0)
. . . . .
 	
End Do
	while (i!=0)
	{
	. . . . . .
	}

ekvivalentne naredbe/kontrole toka

Fortran 77	C

If(i.eq.1) then
. . . . . .
End If
	if(i == 1)
	{
	. . . . . .
	}

If(i.eq.1) then
x=3.0
Else If(i.eq.2)
x=4.0
Else
x = 5.0
End If
	if(i == 1)
	x=3.0;
	else if(i == 2)
	x=4.0;
	else
	x=5.0;

If(i.eq.1.and.j.eq.2) then
k = 3
End If
	if(i == 1 && j == 2)
	k = 3;

ekvivalentne matematičke funkcije

Fortran 77	C	Komentar
	#include <math.h>	C header file
y = sqrt(x)	y = sqrt(x)	; dolazi na kraju komande
y = exp(x)	y = exp(x)	
y = log(x)	y = log(x)	
y = log10(x)	y = log10(x)	
y = abs(x)	y = fabs(x)	
y = sin(x)	y = sin(x)	
y = cos(x)	y = cos(x)	
y = tan(x)	y = tan(x)	
y = asin(x)	y = asin(x)	
y = acos(x)	y = acos(x)	
y = atan(x)	y = atan(x)	
y = sinh(x)	y = sinh(x)	
y = cosh(x)	y = cosh(x)	
y = tanh(x)	y = tanh(x)	
z = x**y	z = pow(x,y)	
y = int(x)	y = ceil(x)	najmanji cijeli broj koji nije manji od x

Aritmetičke operacije i Operatori

.	Fortran	Primjer	C/C++	Primjer
zbroj	+	A=B+C	+	a=b+c;
razlika	-	A=B-C	-	a=b-c;
množenje	*	A=B*C	*	a=b*c;
dijeljenje	/	A=B/C	/	a=b/c;
zbroj	+	A=A+B	+=	a+=b
razlika	-	A=A-B	-=	a-=b
+1	+	A=A+1	++	++a,a++

		Primjer	C/C++	Primjer
		(A.EQ.B)...	==	if(a==b)...
		(A.NE.B)...	!=	if(a!=b)...
		(A.LT.B)...	<	if(a<b)...
		(A.GT.B)...	>	if(a>b)...
		(A.LE.B)...	<=	if(a<=b)...
		(A.GE.B)...	>=	if(a>=b)...
		(.NOT.A)...	!	if(!a)...
		(A.AND.B)...	&&	if(a&&b)...
logički OR	.OR.	IF(A.OR.B)...	||	if(a||b)...

Operatori

.	Fortran	Example	C/C++	Example
Bitwise AND	IAND	IAND(N,M)	&	n&m
Bitwise OR	IOR	IOR(N,M)	|	n|m
Bitwise Exclusive OR	IEOR	IEOR(N,M)	^	n^m
Bitwise 1's Complement	INOT	INOT(N)	~	~n
Bitwise Left Shift	ISHFT	ISHFT(N,M) (M > 0)	<<	n<<m
Bitwise Right Shift	ISHFT	ISHFT(N,M) (M < 0)	>>	n>>m

Pozivanje fortran funkcija iz C-a

```
#include <stdio.h>
void myfortranfn_ (int *, int *);

main(){

  int ia = 10, ib;
  myfortranfn_ (&ia, &ib);
  printf("\n fortran computed ib=%d \n", ib);
}
```

```
      subroutine myfortranfn(ia,ib)

      integer ia, ib

      ib = 100*ia

      return
      end
```

```
gcc -c mycdriver.c
f77 -c myfortranfn.f
```

```
f77 -o mycdriver mycdriver.o myfortranfn.o
gcc -o mycdriver mycdriver.o myfortranfn.o
```

program nm

- Neki kompajleri ne dodaju znak _ na kraj imena fortran funkcija. Kod f77 kompajlera -fno-underscoring opcija.
- Ovisno o kodu, link proces može biti kompliciran
- nm unix komanda list simbole iz objektnog koda ili datoteke (library).

```
> nm myfortranfn.o
   00000000 T myfortranfn_
```

```
> nm mycdriver.o
        00000000 T main
        U myfortranfn_
        U printf
```

polje kao argument

```
#include <stdio.h>
/* prototype for the fortran test function  */
void myffn_(int *, int *);
main() {
 /* data to be passed to Fortran */
 int in = 2;
 int *idata = (int*) calloc(in, sizeof(int));
 idata[0] = 1; idata[1] = 3;
 printf("C before call: idata[0] = %d \n", idata[0]);
 printf("C before call: idata[1] = %d \n", idata[1]);
 /* call myffn (a F77 function) note underscore */
 myffn_(&in, idata);
 /* printout entries of idata again*/
 printf("C after  call: idata[0] = %d \n", idata[0]);
 printf("C after  call: idata[1] = %d \n", idata[1]); }
```

mydriver1.c i myffn.f

```
c     SOURCE myffn.f
      subroutine myffn(in, idata)
c     indicate types of function arguments
      integer in
      integer idata(*)
c     print out idata entries
      write(*,*) 'f77 before change: idata(1) = ', idata(1)
      write(*,*) 'f77 before change: idata(2) = ', idata(2)
c     access and change idata contents (note indexing from 1)
      idata(1) = idata(1)+1
      idata(2) = idata(2)+3
c     print out idata entries
      write(*,*) 'f77 after  change: idata(1) = ', idata(1)
      write(*,*) 'f77 after  change: idata(2) = ', idata(2)
      return
      end
```

Pozivanje C funkcije iz F77

Fortran file:

```
      Program Callc
      INTEGER CR2
      N=10
      CALL CR1(N,M)
      WRITE(6,20) N,M
   20 FORMAT(' The square of',I3,' is',I4)
      K=CR2(N)
      WRITE(6,30) N,K
   30 FORMAT(' The cube of',I3,' is',I15)
      CALL EXIT
      END
```

C file:

```
void  cr1_(int *n, int *m)
{
   // Compute the square of n, return in m
   int k;
   k=*n;
   *m=k*k;
   return;
}
int  cr2_(int *n)
//  Compute the cube of n
{
   int m,k;
   k=*n;
   m=k*k*k;
   return m;
}
```

f77 -c myfortprog.f

gcc -c mycfn.c

f77 -o myfortprog myfortprog.o mycfn.o

Recept za miješanje

- Recept f77 & C
  - **nm** otkriva kojim simbolom je kompajler označio funkciju u objektnom kodu
  - f77 koristi pointere, argumenti su pointeri na varijable
  - Ako je argument u f77 podprogramu polje, trebamo pointer na prvi član u polju (a ne pointer na pointer)
  - ne koristimo _ (underscore) u f77 jer kompajler može dodati još
  - U C jeziku možemo dodati _ ako funkciju pozivamo iz f77

Makefile

Rastavljen je na dva dijela:

- Make.opts
- Makefile

1)Make.opts odabir kompajlera, linkera i datoteka

```
# specify which compilers to use for c, fortran and linking
CC      = gcc
FC = g77
LD      = gcc
LDF     = g77
# compiler flags to be used (set to compile with debugging on)
CFLAGS          =  -I$(HDRDIR) -g
LIBS    = -L. -L../lib  -llapack -lcblas -lblas -lg2c -lm
LIBSF  = -L. -L../lib  -llapack -lcblas -lblas
```

Makefile

2. Makefile

Naredba make kompajlira i linka sve

make cfortran napravi primjere koji se odnose na C i fortran (do ovog teksta)

make cblas napravi primjere za BLAS iz C jezika

make fblas napravi primjere za BLAS iz F77

```
# rule for .c files
.c.o:
        $(CC) $(CFLAGS) -c $<
all: cfortran cblas fblas
OBJ1 = mycdriver.o myfortranfn.o
fblas: blaslevel1 blaslevel2
mycdriver:  $(OBJ1)
        $(LD) -o mycdriver $(OBJ1) $(LDFLAGS) $(LIBS)
```

Komentar

Implicitno kompajliranje

Svi programi

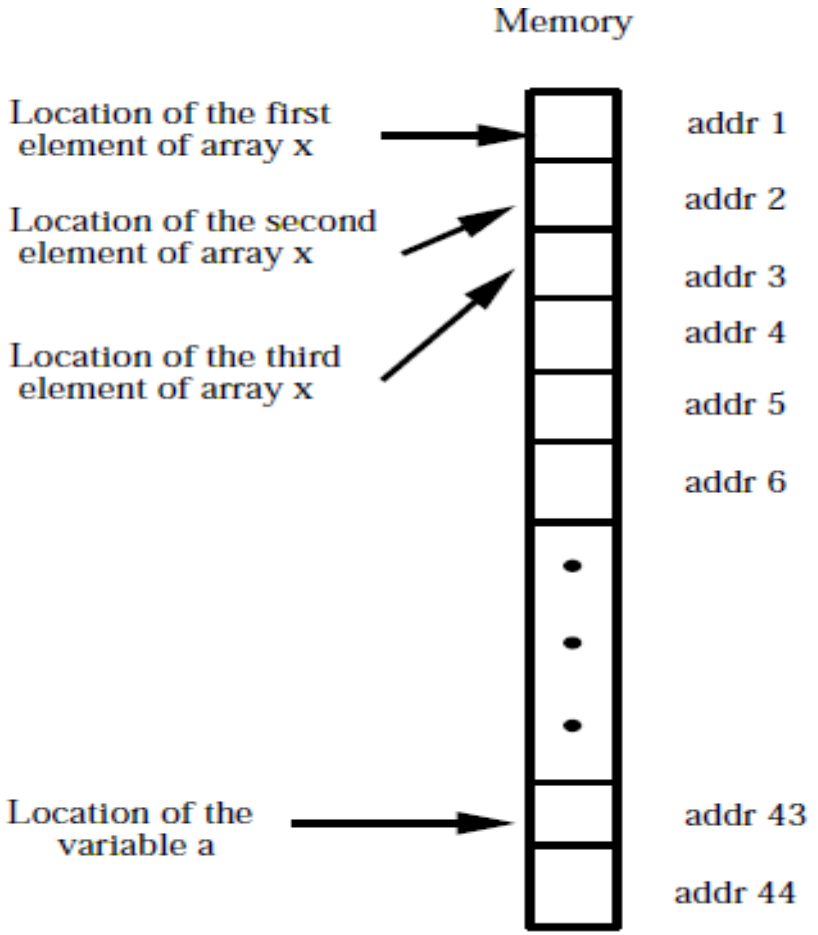
Objektni kod za 1. program

Dio programa

1. program

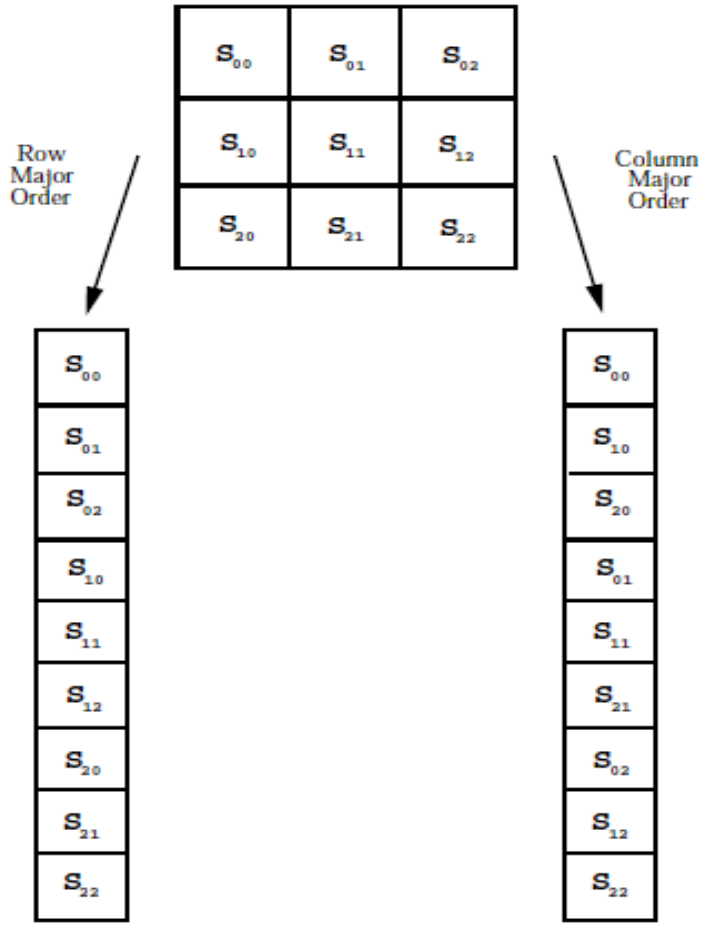
Memorija, matrice

- adresa u memoriji
  - addr2=addr1+addrset
    - addr mjesto gdje se nalazi memorija
    - addrset je offset

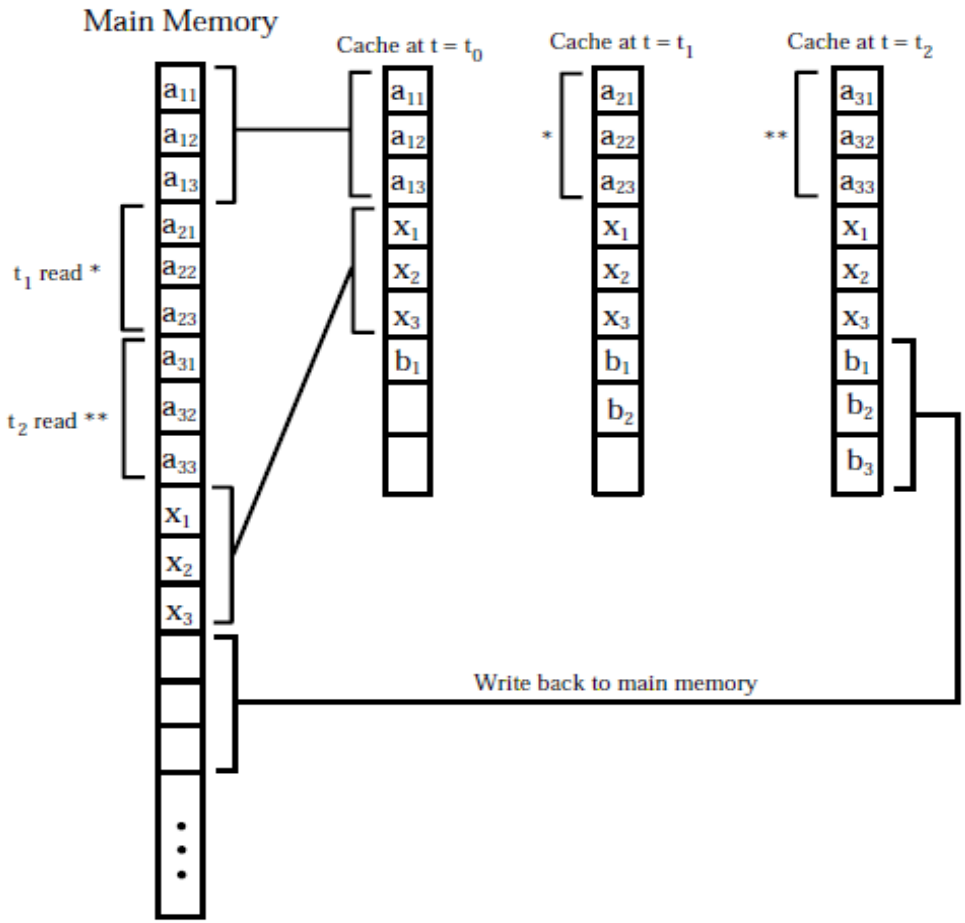


Matrice, memorija

- Row major i Column major order
  - veličina memorije ista
  - linearno uređenje je različito
  - Za simetrične matrice, isto uređenje
  - C koristi row major
  - Fortran koristi column major
- Glavna memorija CPU (spora)
  - Cache: Level 1 (L1) i Level 2 (L2)



- Ideja je da se cache memorija koristi što više kad se jednom napuni (cache hits, cache miss)
- Cache bloking maksimalno iskorištenje cache strukturom podataka i operacija npr. odabirom row major uređenja.
- brzina ovisi o algoritmu koji koristi raspored memorije u cache memoriji



Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - C Pozirkidis: C++ for Fortran Programmers, lecture notes
  - George W. Collins, II:
    Fundamental Numerical Methods and Data Analysis
  - Fortran and C/C++ Mixed Programming
  - Calling Fortran Subroutines from C/C++
  - USER NOTES ON FORTRAN PROGRAMMING (UNFP)

Zadaci za praktikum

(a) Napravite programe pomoću make naredbe: make cfortran

(b) Napišite program za množenje 2 vektora.

(c) Napišite program za množenje dviju matrica. Iskoristite programe matrix_mult i readAnddisplay(C/F) (.f,.c). Programi trebaju pročitati matrice iz tekst formata (npr. matrixVector.dat, pročita se dimenzija, a onda matrica), proširite format i dodajte oznaku za komentar npr. *, ili c kao prvi znak.

(d) Napišite program za množenje vektora i matrice.

(e) Pošaljite mail na Aleksandar.Maksimovic@irb.hr s odgovorima u attachmentu.

Sustavi linearnih jednažbi

- Sustav n jednadžbi i n nepoznanica

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2j}x_j + \cdots + a_{2n}x_n &= b_2 \\
\vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots & \\
a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n &= b_i \\
\vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots & \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nj}x_j + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

Matrica $A = [\, a_{ij} \,]_{i,j=1}^{n} \in \mathbb{R}^{n\times n}$ je **matrica sustava**, a njeni elementi su **koeficijenti sustava**. Vektor $b = [\, b_i \,]_{i=1}^{n} \in \mathbb{R}^{n}$ je **vektor desne strane** sustava. Treba odrediti **vektor nepoznanica** $x = [\, x_i \,]_{i=1}^{n} \in \mathbb{R}^{n}$ tako da vrijedi $Ax = b$.

Primjer 1

Interpolacijski polinom

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j$$

nepoznati su koeficijenti, određujemo ih poznatih vrijednosti y

$$p(x_i) = y_i,\ i = 0, \ldots, n,$$

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{n-1} x_0^{n-1} + a_n x_0^n &= y_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{n-1} x_1^{n-1} + a_n x_1^n &= y_1 \\
\vdots \quad \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \quad \vdots & \\
a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_{n-1} x_i^{n-1} + a_n x_i^n &= y_i \\
a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_{n-1} x_n^{n-1} + a_n x_n^n &= y_n.
\end{aligned}$$

Primjer 1

$$Va = y$$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{n-1} & x_1^n \\ & & \cdots & & \cdots & & \\ 1 & x_i & x_i^2 & x_i^3 & \cdots & x_i^{n-1} & x_i^n \\ & & \cdots & & \cdots & & \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^{n-1} & x_n^n \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}}_{y}.$$

Primjer 2

Promotrimo sljedeći rubni problem:

$$-\frac{d^2}{dx^2}u(x) = f(x), \quad 0 < x < 1,$$
$$u(0) = u(1) = 0.$$

$$h = \frac{1}{n+1}, \quad x_i = ih, \quad i = 0, \ldots, n+1.$$

$$u_i = u(x_i),\ i = 0, \ldots, n+1.$$

$$u_0 = u_{n+1} = 0$$

Primjer 2

$$u(x_i + h) = u(x_i) + u'(x_i)h + \frac{u''(x_i)}{2}h^2 + \frac{u'''(x_i)}{6}h^3 + \frac{u^{(4)}(x_i + \alpha_i)}{24}h^4$$

$$u(x_i - h) = u(x_i) - u'(x_i)h + \frac{u''(x_i)}{2}h^2 - \frac{u'''(x_i)}{6}h^3 + \frac{u^{(4)}(x_i + \zeta_i)}{24}h^4,$$

Kombinacijom ova dva Taylorova reda dobivamo drugu derivaciju

$$u_{i+1} + u_{i-1} = 2u_i + u''(x_i)h^2 + (u^{(4)}(x_i + \alpha_i) + u^{(4)}(x_i + \zeta_i))\frac{h^4}{24},$$

Druga derivacija metodom konačnih razlika

$$-u''(x_i) = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2}$$

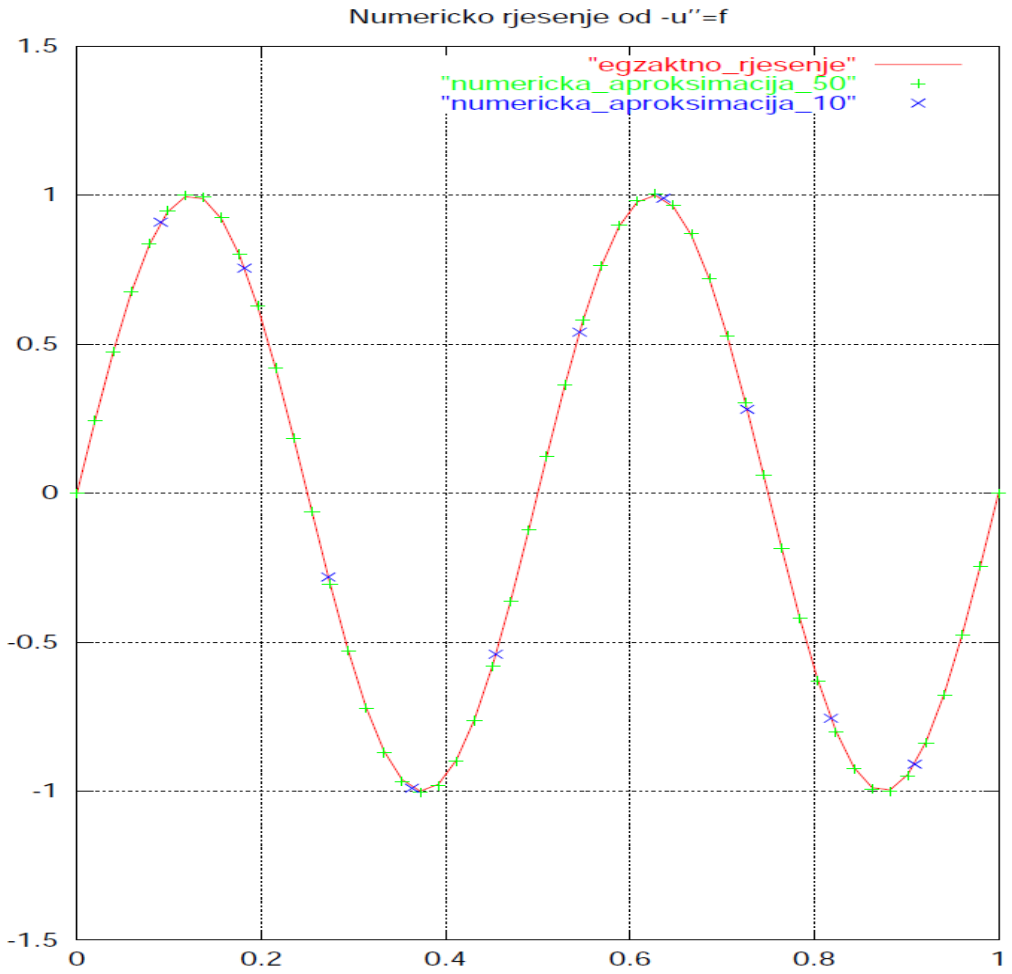
Primjer 2

$$\underbrace{\begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{bmatrix}}_{T_n} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{bmatrix}}_{u} = h^2 \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ f_n \end{bmatrix}}_{f}.$$

Primjer 2

$$f(x) = 16\pi^2 \sin 4\pi x$$

$$n = 10 \ i \ n = 50$$



Aleksandar Maksimović,
Institute Ruđer Bošković, Bijenička c. 54

Linearni sustav 2x2

$$\begin{aligned} 2x_1 - \ \ x_2 &= 1 \\ -x_1 + 2x_2 &= 1 \end{aligned}$$

$$x_1 = \frac{1}{2}(1 + x_2)$$

$$-\underbrace{\frac{1}{2}(1 + x_2)}_{x_1} + 2x_2 = 1, \quad \text{tj. } \frac{3}{2}x_2 = \frac{3}{2}, \quad \text{tj.} \quad x_2 = 1, \quad x_1 = 1$$

Metoda supstitucije

$$\begin{aligned} 5x_1 + \ x_2 + 4x_3 &= \ 19 \\ 10x_1 + 4x_2 + 7x_3 &= \ 39 \\ -15x_1 + 5x_2 - 9x_3 &= -32 \end{aligned} \quad \equiv \quad \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 10 & 4 & 7 \\ -15 & 5 & -9 \end{bmatrix}}_{A = [\, a_{ij} \,]_{i,j=1}^{3}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \underbrace{\begin{bmatrix} 19 \\ 39 \\ -32 \end{bmatrix}}_{b = [\, b_i \,]_{i=1}^{3}} .$$

Koristimo metodu supstitucija, odnosno eliminacija. Prvo iz prve jednadžbe izrazimo x_1 pomoću x_2 i x_3, te to uvrstimo u zadnje dvije jednadžbe, koje postaju dvije jednadžbe s dvije nepoznanice (x_2 i x_3). Dobivamo

$$x_1 = \frac{1}{5}\,(19 - x_2 - 4x_3),$$

Metoda supstitucije

pa druga jednadžba sada glasi

$$\frac{10}{5}(19 - x_2 - 4x_3) + 4x_2 + 7x_3 = 39,$$

tj.

$$-\frac{10}{5}(x_2 + 4x_3) + 4x_2 + 7x_3 = 39 + \left(-\frac{10}{5}19\right)$$

Dakle, efekt ove transformacije je ekvivalentno prikazan kao rezultat množenja prve jednadžbe s

$$-\frac{a_{21}}{a_{11}} = -\frac{10}{5} = -2$$

i zatim njenim dodavanjem (pribrajanjem) drugoj jednadžbi. Druga jednadžba sada glasi

$$2x_2 - x_3 = 1.$$

Metoda supstitucije

$$\underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 10 & 4 & 7 \\ -15 & 5 & -9 \end{bmatrix}}_{A} \mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{L^{(2,1)}} \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 10 & 4 & 7 \\ -15 & 5 & -9 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ -15 & 5 & -9 \end{bmatrix}}_{A^{(1)} = \left[a_{ij}^{(1)} \right]_{i,j=1}^{3}}$$

Nepoznanicu x_1 eliminiramo iz zadnje jednadžbe ako prvu pomnožimo s

$$\underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ -15 & 5 & -9 \end{bmatrix}}_{A^{(1)}} \mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}}_{L^{(3,1)}} \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ -15 & 5 & -9 \end{bmatrix}}_{A^{(1)}} = \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ 0 & 8 & 3 \end{bmatrix}}_{A^{(2)} = \left[a_{ij}^{(2)} \right]_{i,j=1}^{3}} .$$

Vektor desne strane je u ove dvije transformacije promijenjen u

Metoda supstitucije

Vektor desne strane je u ove dvije transformacije promijenjen u

$$\underbrace{\begin{bmatrix} 19 \\ 39 \\ -32 \end{bmatrix}}_{b} \mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{L^{(2,1)}} \begin{bmatrix} 19 \\ 39 \\ -32 \end{bmatrix} = \underbrace{\begin{bmatrix} 19 \\ 1 \\ -32 \end{bmatrix}}_{b^{(1)}}$$

$$\mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}}_{L^{(3,1)}} \begin{bmatrix} 19 \\ 1 \\ -32 \end{bmatrix} = \underbrace{\begin{bmatrix} 19 \\ 1 \\ 25 \end{bmatrix}}_{b^{(2)}}.$$

Novi, ekvivalentni, sustav je $A^{(2)}x = b^{(2)}$, *tj.*

Metoda supstitucije

Novi, ekvivalentni, sustav je $A^{(2)}x = b^{(2)}$, *tj.*

$$\begin{aligned} 5x_1 + \ x_2 + 4x_3 &= 19 \\ 2x_2 - \ x_3 &= \ 1 \\ 8x_2 - 3x_3 &= 25, \end{aligned}$$

$$-\frac{a_{32}^{(2)}}{a_{22}^{(2)}} = -4$$

$$\begin{aligned} 5x_1 + \ x_2 + 4x_3 &= 19 \\ 2x_2 - \ x_3 &= \ 1 \\ 7x_3 &= 21. \end{aligned}$$

Metoda supstitucije

$$\underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ 0 & 8 & 3 \end{bmatrix}}_{A^{(2)}} \mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix}}_{L^{(3,2)}} \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ 0 & 8 & 3 \end{bmatrix}}_{A^{(2)}} = \underbrace{\begin{bmatrix} 5 & 1 & 4 \\ 0 & 2 & -1 \\ 0 & 0 & 7 \end{bmatrix}}_{A^{(3)}}$$

transformaciju vektora desne strane

$$\underbrace{\begin{bmatrix} 19 \\ 1 \\ 25 \end{bmatrix}}_{b^{(2)}} \mapsto \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix}}_{L^{(3,2)}} \underbrace{\begin{bmatrix} 19 \\ 1 \\ 25 \end{bmatrix}}_{b^{(2)}} = \underbrace{\begin{bmatrix} 19 \\ 1 \\ 21 \end{bmatrix}}_{b^{(3)} = (b_i^{(3)})_{i=1}^3} .$$

Metoda supstitucije

1. *Iz treće jednadžbe je* $x_3 = \frac{21}{7} = 3.$
2. *Iz druge jednadžbe je* $x_2 = \frac{1}{2}(1 + x_3) = 2.$
3. *Iz prve jednadžbe je* $x_1 = \frac{1}{5}(19 - x_2 - 4x_3) = 1.$

Opći prikaz metode

$$\left[\begin{array}{cccc:c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{array}\right].$$

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, \ldots, n.$$

$$\left[\begin{array}{cccc:c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{array}\right].$$

Opći prikaz metode

$$m_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, \quad i = 3, \ldots, n,$$

$$\left[\begin{array}{cccc:c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ & & \ddots & \vdots & \vdots \\ & & & a_{nn}^{(n)} & b_n^{(n)} \end{array}\right].$$

Uz pretpostavku da je $a_{nn}^{(n)} \neq 0$, ovaj se linearni sustav lako rješava povratnom supstitucijom

pivotiranje

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}},$$

$$x_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j \right), \quad i = n-1, \ldots, 1.$$

Uobičajeno **parcijalno pivotiranje** kao pivotni element bira element koji je po apsolutnoj vrijednosti najveći u ostatku tog stupca — na glavnoj dijagonali ili ispod nje. Drugim riječima, ako je u k-tom koraku

$$|a_{rk}^{(k)}| = \max_{k \le i \le n} |a_{ik}^{(k)}|,$$

onda ćemo zamijeniti r-ti i k-ti redak i početi korak eliminacije elemenata k-tog stupca.

pivotiranje

Osim parcijalnog pivotiranja, može se provoditi i **potpuno pivotiranje**. U k-tom koraku, bira se maksimalni element u cijelom "ostatku" matrice $A^{(k)}$, a ne samo u k-tom stupcu. Ako je u k-tom koraku

$$|a_{rs}^{(k)}| = \max_{k \le i,j \le n} |a_{ij}^{(k)}|,$$

parcijalno pivotiranje - zamjena redaka

potpuno pivotiranje - zamjena redaka i stupaca

algoritam

Algoritam 5.2.1. (Gaussove eliminacije s parcijalnim pivotiranjem)

```
    {Trokutasta redukcija}
  for k := 1 to n − 1 do
    begin
      {Nađi maksimalni element u ostatku stupca}
    max_elt := 0.0;
    ind_max := k;
    for i := k to n do
      if abs(A[i, k]) > max_elt then
        begin
        max_elt := abs(A[i, k]);
```

algoritam

```
      ind_max := i;
      end;
if max_elt > 0.0 then
   begin
   if ind_max <> k then
        {Zamijeni k-ti i ind_max-t
      begin
      for j := k to n do
         begin
         temp := A[ind_max, j];
         A[ind_max, j] := A[k, j];
         A[k, j] := temp;
         end;
      temp := b[ind_max];
      b[ind_max] := b[k];
      b[k] := temp;
      end;
   for i := k + 1 to n do
      begin
      mult := A[i, k]/A[k, k];
      A[i, k] := 0.0;   {Ne treba, ne koristi se kasnije}
      for j := k + 1 to n do
         A[i, j] := A[i, j] − mult ∗ A[k, j];
      b[i] := b[i] − mult ∗ b[k];
      end;
   end
else
      {Matrica je singularna, stani s algoritmom}
   begin
   error := true;
   exit;
   end;
end:
```

algoritam

```
   {Povratna supstitucija, rješenje x ostavi u b}
b[n] := b[n]/A[n, n];
for i := n − 1 downto 1 do
   begin
   sum := b[i];
   for j := i + 1 to n do
      sum := sum − A[i, j] ∗ b[j];
   b[i] := sum/A[i, i];
   end;
error := false;
```

Zadatak 5.2.1. *Pokušajte samostalno napisati algoritam koji koristi potpuno pivotiranje. Posebnu pažnju obratite na efikasno pamćenje zamjena varijabli koje su posljedica zamjena stupaca. Može li se isti princip efikasno primijeniti i za pamćenje zamjena redaka, tako da se potpuno izbjegnu eksplicitne zamjene elemenata u matrici A i vektoru b?*

gaussj.c

```
#include <math.h>
#define NRANSI
#include "nrutil.h"
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}
void gaussj(float **a, int n, float **b, int m)
{
    int *indxc,*indxr,*ipiv;
    int i,icol,irow,j,k,l,ll;
    float big,dum,pivinv,temp;

    indxc=ivector(1,n);
    indxr=ivector(1,n);
    ipiv=ivector(1,n);
    for (j=1;j<=n;j++) ipiv[j]=0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if (ipiv[j] != 1)
                for (k=1;k<=n;k++) {
                    if (ipiv[k] == 0) {
                        if (fabs(a[j][k]) >= big) {
                            big=fabs(a[j][k]);
                            irow=j;
                            icol=k;
                        }
                    }
                }
        ++(ipiv[icol]);
        if (irow != icol) {
            for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
            for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
        }
        indxr[i]=irow;
        indxc[i]=icol;
        if (a[icol][icol] == 0.0) nrerror("gaussj: Singular Matrix");
        pivinv=1.0/a[icol][icol];
        a[icol][icol]=1.0;
        for (l=1;l<=n;l++) a[icol][l] *= pivinv;
        for (l=1;l<=m;l++) b[icol][l] *= pivinv;
        for (ll=1;ll<=n;ll++)
            if (ll != icol) {
                dum=a[ll][icol];
                a[ll][icol]=0.0;
                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
            }
    }
    for (l=n;l>=1;l--) {
        if (indxr[l] != indxc[l])
            for (k=1;k<=n;k++)
                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
    }
    free_ivector(ipiv,1,n);
    free_ivector(indxr,1,n);
    free_ivector(indxc,1,n);
}
#undef SWAP
#undef NRANSI
```

LU faktorizacija

$$A^{(3)} = L^{(3,2)} L^{(3,1)} L^{(2,1)} A.$$

Matrica $A^{(3)}$ je gornjetrokutasta, a produkt $L^{(3,2)} L^{(3,1)} L^{(2,1)}$ je donjetrokutasta matrica. Dakle, polaznu matricu A smo množenjem slijeva donjetrokutastom matricom načinili gornjetrokutastom. To možemo pročitati i ovako:

$$A = LA^{(3)}, \quad L = (L^{(2,1)})^{-1} (L^{(3,1)})^{-1} (L^{(3,2)})^{-1},$$

gdje je L donjetrokutasta matrica. Lako se provjerava da je

$$L = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{(L^{(2,1)})^{-1}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix}}_{(L^{(3,1)})^{-1}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{bmatrix}}_{(L^{(3,2)})^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 4 & 1 \end{bmatrix}.$$

LU faktorizacija

Matrica A je produkt gornjotrokutaste i donjotrokutaste matrice

$$A = LA^{(3)} \qquad U = A^{(3)} \qquad A = LU.$$

Imamo: A x =b, A=L U, L (U x) = b, U x=y,

L y = b

U x = y

Linearni sustav riješen je pomoću 3 koraka:

1. Matricu sustava A treba faktorizirati u obliku $A = LU$, gdje je L donjetrokutasta, a U gornjetrokutasta matrica.
2. Rješavanjem donjetrokutastog sustava $Ly = b$ treba odrediti vektor $y = L^{-1}b$.
3. Rješavanjem gornjetrokutastog sustava $Ux = y$ treba odrediti vektor $x = U^{-1}y = U^{-1}(L^{-1}b)$.

supstitucija unaprijed

$$\begin{bmatrix} \ell_{11} & 0 & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & \ell_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

linearni sustav s donjotrokutastom matricom

matrica je regularna $\ell_{ii} \neq 0$

algoritam:

$$x_1 = \frac{b_1}{\ell_{11}};$$

za $i = 2, \ldots, n$ {

$$x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) \Big/ \ell_{ii}; \}$$

$$x_1 = \frac{b_1}{\ell_{11}}$$

$$x_2 = \frac{1}{\ell_{22}} (b_2 - \ell_{21} x_1)$$

$$x_3 = \frac{1}{\ell_{33}} (b_3 - \ell_{31} x_1 - \ell_{32} x_2)$$

$$x_4 = \frac{1}{\ell_{44}} (b_4 - \ell_{41} x_1 - \ell_{42} x_2 - \ell_{43} x_3).$$

supstitucija unazad

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

linearni sustav s gornjotrokutastom matricom

algoritam

$$x_n = \frac{b_n}{u_{nn}};$$

za $i = n-1, \ldots, 1$ {

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \Big/ u_{ii}; \}$$

$$x_4 = \frac{b_4}{u_{44}}$$

$$x_3 = \frac{1}{u_{33}} (b_3 - u_{34} x_4)$$

$$x_2 = \frac{1}{u_{22}} (b_2 - u_{23} x_3 - u_{24} x_4)$$

$$x_1 = \frac{1}{u_{11}} (b_1 - u_{12} x_2 - u_{13} x_3 - u_{14} x_4).$$

gauss eliminacija

SUBROUTINE gaussj(a,n,np,b,m,mp)

Linear equation solution by Gauss-Jordan elimination, equation (2.1.1) above.

a(1:n,1:n) is an input matrix stored in an array of physical dimensions np by np.

b(1:n,1:m) is an input matrix containing the m right-hand side vectors, stored in an array of physical dimensions np by mp. On output, a(1:n,1:n) is replaced by its matrix inverse, and b(1:n,1:m) is replaced by the corresponding set of solution vectors.

6 argumenata

void gaussj(float **a, int n, float **b, int m)

4 argumenta, ne treba NP i MP dimenzije od matrice i vektora.

b argument kod C verzije je matrica.

NR primjer

- Fortran
  - Files: Xgaussj.for, gaussj.for, MATRX1.dat
    - Xgaussj.for je modificirani "kod" originala xgaussj.for, problem je bio s čitanjem podataka iz file-a. xgaussj.for nemodificirani kod.
    - f77 -o fgauss Xgaussj.for gaussj.for
- C
  - Files: xgaussj.c gaussj.c nrutils/nrutil.c nrutils/, MATRX1.dat
    - nrutils/ sadrži .h fileove, tj. nrutil.h i nr.h
    - nrutil/, tj. nrutil.c i nrutil.h dijelovi za alociranje matrice, vektora itd..
    - gcc -o xgauss xgaussj.c gaussj.c nrutils/nrutil.c -I nrutils/

F77 gauss

fortran primjer: tgauss.f

Kompajliranje: g77 -o tg1 tgauss.f gaussj.for lib1.f

```
 open (unit=8,file="linsys1.dat")

 call readmatrix(MA1,n1,m1,fp)
call readvektor(b,L1,fp)

call gaussj(TA1,n1,NP,x,1,MP)
```

```
print *, "Solution is: "
call displayvektor(x,L1)
write(*,*) 'Inverse of Matrix A : '
call displaymatrix(TA1,n1,m1)
call  multMatrixMatrix(TA1,MA1,XMM,n1,m1,n1,m1)
 write(*,*) 'Inverse of Matrix A * A: '
 call displaymatrix(XMM,n1,m1)
```

C gauss

C primjer: tgauss.c

Kompajliranje: gcc -g -o tg tgauss.c libnr.c gaussj.c nrutils/nrutil.c -I nrutils/

ako se nrutil.c i header fileovi nalaze u poddirektoriju nrutils

```
#include "nr.h"
#include "nrutil.h"
#include "libnr.h"
#define NP 20
---------------------------------
 float *b,*x;
  /*  float b[10],x[10]; */
  float **MA1,**TA1,**XMM,**X;
  X=matrix(1,NP,1,NP);
  b=vector(1,NP);
readmatrix(MA1,n1,m1,fp);
 readvektor(b,L1,fp);

gaussj(TA1,n1,X,1);
 printf("Solution is:\n");
          displaymatrix(X,n1,1);
printf("Inverse of  A : \n"); displaymatrix(TA1,n1,m1);
```

LU metoda

SUBROUTINE ludcmp(a,n,np,indx,d)

void ludcmp(float **a, int n, int *indx, float *d)

Given a matrix a[1..n][1..n], this routine replaces it by the LU decomposition of a rowwise permutation of itself. a and n are input. a is output; indx[1..n] is an output vector that records the row permutation effected by the partial pivoting; d is output as ±1 depending on whether the number of row interchanges was even or odd, respectively. This routine is used in combination with lubksb to solve linear equations or invert a matrix.

NR PRIMJER:

```
 gcc -o xludcmpC xludcmp.c  ludcmp.c nrutils/nrutil.c -I nrutils/
g77 -o xludcmpF xludcmp.for  ludcmp.for
gcc -o xlubsub xlubksb.c  lubksb.c ludcmp.c nrutils/nrutil.c -I nrutils/
 g77 -o xlubsubF xlubksb.for  lubksb.for ludcmp.for
```

SUBROUTINE lubksb(a,n,np,indx,b)

void lubksb(float **a, int n, int *indx, float b[])

Solves the set of n linear equations A X = B. Here a is input, not as the matrix A but rather as its LU decomposition, determined by the routine ludcmp. indx is input as the permutation vector returned by ludcmp. b(1:n) is input as the right-hand side vector B, and returns with the solution vector X. a, n, np, and indx are not modified by this routine and can be left in place for successive calls with different right-hand sides b. This routine takes into account the possibility that b will begin with many zero elements, so it is effcient for use in matrix inversion.

```
 g77 -g -o tludecmp tludecmp.for ludcmp.for lib1.f
gcc -g  -o cludecmp tludecmp.c ludcmp.c nrutils/nrutil.c libnr.c -I nrutils/
gcc -g  -o cludecmp tludecmp.c ludcmp.c nrutils/nrutil.c libnr.c -I nrutils/
g77 -o tlubksub tlubksub.for lubksb.for ludcmp.for lib1.f
gcc -g  -o clubksb tlubksub.c ludcmp.c lubksb.c nrutils/nrutil.c libnr.c -I nrutils/
```

Osobine metoda

- Kvadratični linearni sustavi, broj nepoznanica= jednadžbi
- Pivotiranje je neophodno za stabilnost algoritma
- Gaussova eliminacija
  - Slabosti
    - Zahtjeva desnu stranu (vektor b), koju treba mjenjati u algoritmu
    - Sporost, naročito kada nam ne treba inverzna matrica
  - Prednost, jednostavna
- LU faktorizacija, manji broj operacije od Gauss metode

Zadaci za praktikum

1. Izvršite programe napravljene od xgaussj i tgaussj rutina koje koriste gaussovu eliminaciju.
2. Napravite program koji pročita linearni sustav iz datoteke LIN.DAT i riješite sustav gaussovom metodom.
3. Izvršite programe dobivene iz xludcmp , xlubksb, tludecmp, tlubksb koji koriste LU faktorizaciju.
4. Napravite program koji pročita linearni sustav iz datoteke LIN.DAT i riješite sustav LU faktorizacijom.
5. Dobiveno riješenje pošaljite kao poruku u mailu (jedno riješenje), a source programa stavite kao attachment. Mail je Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran

Interpolacija, Ekstrapolacija

Znamo vrijednosti funkcije f(x) na skupu točaka $x_1, x_2, x_3, \ldots, x_N$ ali nemamo analitički izraz za f(x) koji bi nam omogućio da izračunamo vrijednost funkcije u bilo kojoj točki.

- Provući glatku krivulju kroz zadane točke
- Ako je željeni *x* između najmanjeg i najvećeg ➔ *Interpolacija*
- *Ako je x izvan toga područja (riskantno)* ➔ *Ekstrapolacija*

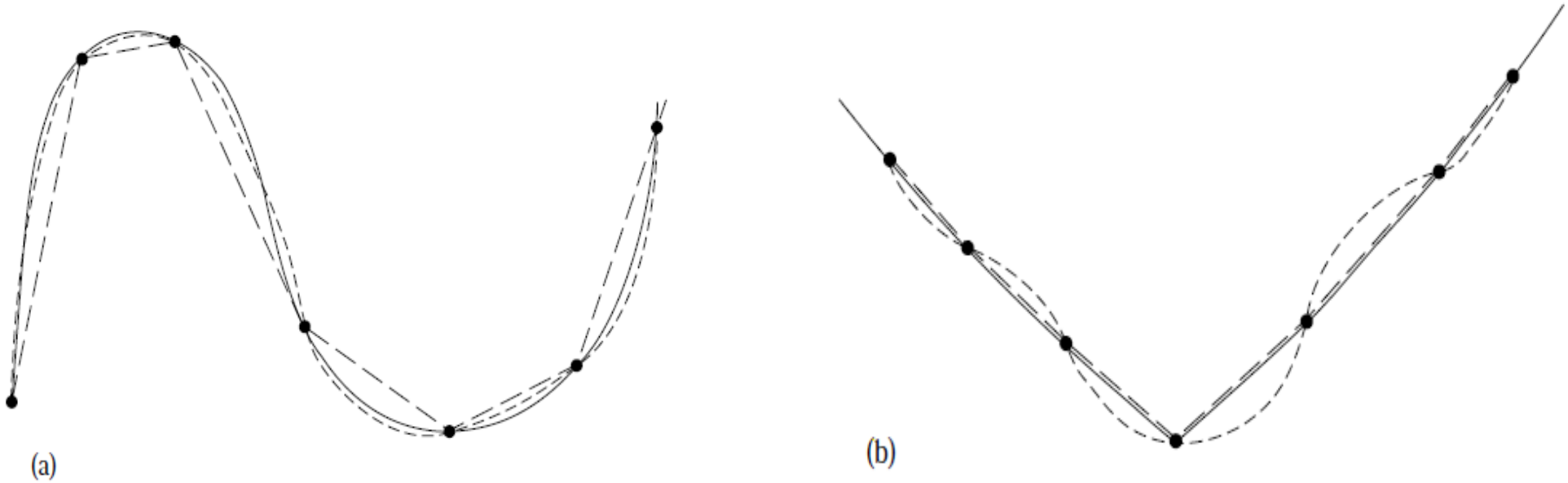
Koristimo slijedeće funkcije:

a) polinome

b) racionalne funkcije (kvocijenti polinoma)

c) trigonometrijske funkcije

Interpolacija



a) Glatka funkcija je bolje interpolirana s polinomom većeg reda

b) funkcija s oštrim kornerom je bolje aproksimirana s polinomom manjeg reda

Eksponencijalne i racionalne funkcije mogu biti loše aproksimirane polinomima većeg stupnja.

Interpolacija polinomom

Interpolacijski polinom

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j$$

nepoznati su koeficijenti, određujemo ih poznatih vrijednosti y

$$p(x_i) = y_i,\ i = 0, \ldots, n,$$

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{n-1} x_0^{n-1} + a_n x_0^n &= y_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{n-1} x_1^{n-1} + a_n x_1^n &= y_1 \\
\vdots \quad \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \quad \vdots & \\
a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_{n-1} x_i^{n-1} + a_n x_i^n &= y_i \\
a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_{n-1} x_n^{n-1} + a_n x_n^n &= y_n.
\end{aligned}$$

Interpolacija

$$Va = y$$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{n-1} & x_1^n \\ & & \cdots & & \cdots & & \\ 1 & x_i & x_i^2 & x_i^3 & \cdots & x_i^{n-1} & x_i^n \\ & & \cdots & & \cdots & & \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^{n-1} & x_n^n \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}}_{y}.$$

$$D_n = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix}.$$

Vandermonde determinanta

Vandermonde sustav je "loše" uvjetovan (ill-conditioned), pogreške zaokruživanja mogu biti velike.

Uvjetovanost sustava

broj uvjetovanosti matrice:

$$\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

- singularna matrica $\mathrm{cond}(A) = \infty$
- Linearni sustav je slabo uvjetovan (ill-conditioned) ako je $\mathrm{cond}(A)$ velik.
- Umjesto A^{-1} koeficijenti polinoma se traže alternativnim načinom
  - poglavlje 2.8 u Numerical Recipes

Interpolaciona krivulja prolazi točno kroz zadane točke.
Ako postoji pogreška kod točaka, izračunati koeficijenti su nepouzdani.
RAČUNANJE KOEFICIJENATA TREBA IZBJEGAVATI.

Podprogrami

SUBROUTINE polcoe(x,y,n,cof)

PARAMETER (NMAX=15) Largest anticipated value of n.

Given arrays x(1:n) and y(1:n) containing a tabulated function $y_i = f(x_i)$, this routine returns an array of coecients cof(1:n), such that $y_i = \sum_j cof_j x_i^{j-1}$

void polcoe(float x[], float y[], int n, float cof[])

Given arrays x[0..n] and y[0..n], returns cof[0..n]

SUBROUTINE polcof(xa,ya,n,cof)

C USES polint

Given arrays xa(1:n) and ya(1:n) of length n containing a tabulated function $ya_i = f(xa_i)$, this routine returns an array of coecients cof(1:n), also of length n, such that $ya_i = \sum_j cof_j xa_i^{j-1}$.

void polcof(float xa[], float ya[], int n, float cof[])

Given arrays xa[0..n] and ya[0..n], returns an array of coecients cof[0..n]

NR primjeri

NR primjeri:

C;

```
gcc -o polcoeC xpolcoe.c polcoe.c  nrutils/nrutil.c libnr.c  -lm -I nrutils/
gcc -o polcofC xpolcof.c polcof.c polint.c  nrutils/nrutil.c libnr.c  -lm -I nrutils/
```

F77:

```
f77 -o xpolcoeF  xpolcoe.for polcoe.for
f77 -o xpolcofF xpolcof.for polcof.for polint.for
```

Primjeri pokazuju interpolaciju sin(x) na intervalu od 0 do Pi, exp(x) na intervalu od 0 do 1

Lagrange polinom

Interpolirajući polinom stupnja *N-1* kroz *N* točaka ima Lagrangeov oblik:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2$$

$$+ \cdots + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

Upotrebljavanje Lagrangeove formule je "teško" programirati

Neville-ov algoritam (sličan njemu je i Aitkenov algoritam)

$$\begin{array}{lllll} x_1: & y_1 = P_1 & & & \\ & & P_{12} & & \\ x_2: & y_2 = P_2 & & P_{123} & \\ & & P_{23} & & P_{1234} \\ x_3: & y_3 = P_3 & & P_{234} & \\ & & P_{34} & & \\ x_4: & y_4 = P_4 & & & \end{array}$$

- Rekurzivan način punjenja (stupac po stupac)
- Bazira se na relaciji “kćerke” P i dva “roditelja”.

Joseph-Louis Lagrange (1736-1813)

Italian-French mathematician associated with many classic mathematics and physics – Lagrange multipliers in minimization of a function, Lagrange's interpolation formula, Lagrange's theorem in group theory and number theory, and the Lagrangian ($L=T-V$) in mechanics and Lagrange equations.

Određivanje polinoma

- Konstruiraju se vrijednosti u zadanim točkama (x_i,y_i), koje označimo P_1 ,P_2 , ...,P_N
- Tražimo polinom prvog stupnja koji prolazi kroz prve dvije točke
- Za vrijednosti P_1 i P_2 u $x=x_1$ i x_2, imamo $P_{12}(x) = \lambda(x)\,P_1+[1-\lambda(x)]\,P_2$

  Zahtjevamo $P_{12}(x_1) = P_1$ and $P_{12}(x_2) = P_2$, to je ispunjeno ako vrijedi

  $$\lambda(x_1) = 1,\ \lambda(x_2) = 0 \ \text{ili}\ \ \lambda(x) = (x-x_2)/(x_1-x_2)$$

- Slijedeći stupanj konstruiramo iz dobivene dvije vrijednosti

  $$P_{123}(x) = \lambda(x)\,P_{12}(\mathrm{x}) + [1-\lambda(x)]\,P_{23}(\mathrm{x})$$

  $P_{123}(x_2) = P_2$ ispunjeno za svaki $\lambda(x)$.

Interpolacija Lagrange

Zahtjevamo $P_{123}(x_1)=P_{12}(x_1)=P_1$ i $P_{123}(x_3)=P_{23}(x_3)=P_3$,

mora vrijediti $\lambda(x_1)=1$, $\lambda(x_3)=0$, odnosno $\lambda(x)=(x-x_3)/(x_1-x_3)$

Rekurzija:

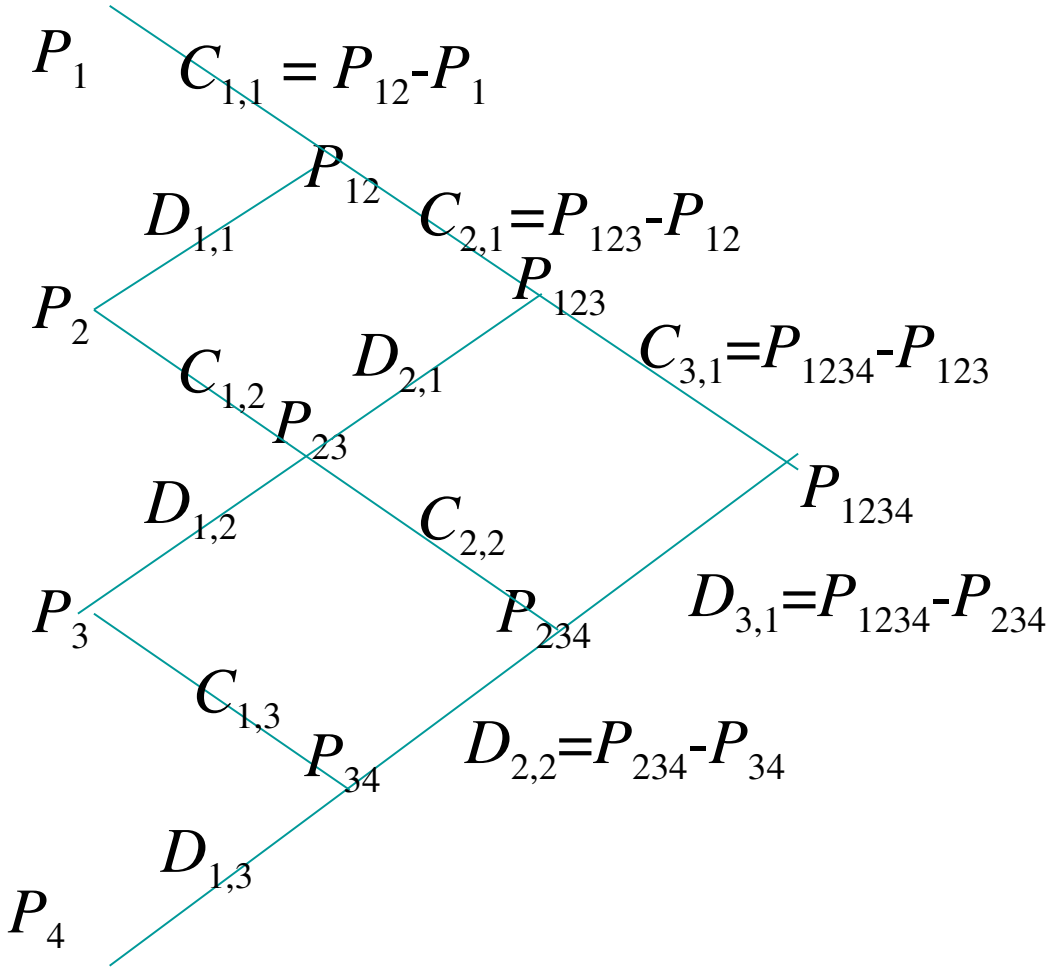
- Dvije interpolirane vrijednosti s m-točaka P, konstruirana iz točaka $i,i+1,i+2,\ldots,i+m-1$, i $i+1,i+2,\ldots,i+m$, određuju slijedeći nivo $m+1$ točaka interpolacije od i do $i+m$:

$$P_{i(i+1)(i+2)\ldots(i+m)}(x) = \lambda(x)\, P_{i(i+1)(i+2)\ldots(i+m-1)}(x) + [1-\lambda(x)]\, P_{(i+1)(i+2),\ldots(i+m)}(x)$$

$$\lambda(x) = (x-x_{i+m})/(x_i-x_{i+m}) \quad \text{ili raspisano}$$

$$P_{i(i+1)\ldots(i+m)} = \frac{(x-x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i-x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i-x_{i+m}}$$

Lagrange polinom



Interpolacija Lagrange

Prati se razlika između kćeri i roditelja, tj. definiraju se veličine:

$$C_{m,i} \equiv P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)}$$

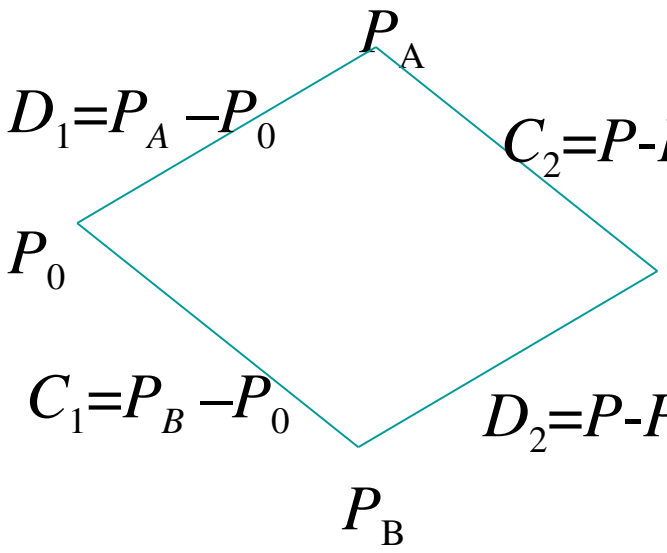
$$D_{m,i} \equiv P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}.$$

Na isti način kao kod popunjavanja tablice interpolacije možemo dobiti relacije

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

Izvod C,D koeficijenta



$$P = \lambda P_A + (1-\lambda)P_B$$

$$C_2 = P-P_A = P-P_0 +P_0 -P_A = P-P_0 -(P_A -P_0)$$

$$C_2 = \lambda P_A + (1-\lambda)P_B -P_0 -(P_A -P_0) =(1-\lambda)(C_1 -D_1)$$

Podprogrami: Lagrange interpolacija

SUBROUTINE polint(xa,ya,n,x,y,dy)

PARAMETER (NMAX=10) Largest anticipated value of n.

Given arrays xa and ya, each of length n, and given a value x, this routine returns a value y, and an error estimate dy. If P(x) is the polynomial of degree N − 1 such that $P(\mathtt{xa}_i) = \mathtt{ya}_i$; i = 1,...,n, then the returned value y = P(x).

void polint(float xa[], float ya[], int n, float x, float *y, float *dy)

Given arrays xa[1..n] and ya[1..n], and given a value x, this routine returns a value y, and an error estimate dy.

Podprogrami

NR primjer:
gcc -o polintC xpolint.c polint.c nrutils/nrutil.c -I nrutils/ -lm
interpolira sin(x) i exp(x) od Pi/n do Pi, tj od 1/n do 1.
f77 -o xpolintF xpolint.for polint.for

Interpolacija tablice:
gcc -o tpolintC tpolint.c polint.c nrutils/nrutil.c libnr.c -lm -I nrutils/
otvori Log10.dat, pozovi polint i spremi u Interp.dat, funkcija log(x) iz intervala 0.1 do 10

NAPOMENA: U polint.for je izmjenjen NMAX od 10 na 50, tj. prije je bila dopuštena interpolacija od samo 10 točaka. polint.c nema takvih problema zbog dinamičkog alociranja vektora.

Interpolacija racionalnim funkcijama

Neke funkcije se bolje aproksimiraju s racionalnim funkcijama, npr. kvocijenti polinoma.

Racionalne funkcije opisuju veći broj krivulja, manje osciliraju.

Racionalna funkcija koja prolazi kroz *m+1* točku je:

$$R_{i(i+1)\ldots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu}$$

P polinom stupnja μ

Q polinom stupnja ν

Imamo ν+μ+1 nepoznanica (*p*,*q*, i q_0 je proizvoljan)

mora vrijediti: $m + 1 = \mu + \nu + 1$

- U specifikaciji racionalne funkcije treba dati željeni stupanj brojnika i nazivnika
- Racionalne funkcije mogu modelirati funkcije s polom tj s nulama polinoma u brojniku kod izraza s lijeve strane
- Bulirsch i Stoer su razvili algoritam sličan Nevillev-om

Interpolacija racionalnim funkcijama

Konstruira se racionalna funkcija koja prolazi kroz zadani skup točaka
Konstruira se tablica racionalnih funkcija stupac po stupac kao kod Nevillovog algoritma. Rekurzija za racionalnu funkciju je:

$$R_{i(i+1)\ldots(i+m)} = R_{(i+1)\ldots(i+m)} + \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{\left(\frac{x-x_i}{x-x_{i+m}}\right)\left(1 - \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{R_{(i+1)\ldots(i+m)} - R_{(i+1)\ldots(i+m-1)}}\right) - 1}$$

Ponovo se rekurzija može napisati pomoću razlika "roditelja" i "kćeri", tj. preko koeficijenata C i D

$$C_{m,i} \equiv R_{i\ldots(i+m)} - R_{i\ldots(i+m-1)}$$

$$D_{m,i} \equiv R_{i\ldots(i+m)} - R_{(i+1)\ldots(i+m)}$$

podprogrami

SUBROUTINE ratint(xa,ya,n,x,y,dy)

PARAMETER (NMAX=10,TINY=1.e-25) Largest expected value of n, and a small number.

Given arrays xa and ya, each of length n, and given a value of x, this routine returns a value of y and an accuracy estimate dy. The value returned is that of the diagonal rational function, evaluated at x, which passes through the n points $(\mathtt{xa}_i, \mathtt{ya}_i)$, $i = 1...\mathtt{n}$.

void ratint(float xa[], float ya[], int n, float x, float *y, float *dy)

Given arrays xa[1..n] and ya[1..n], and given a value of x, this routine returns a value of y and an accuracy estimate dy.

NR primjer: interpolira x*exp(-x)/(SQR(x-1.0)+eps*eps); u intervalu 0.2 do 2

gcc -o ratintC ratint.c xratint.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o ratintF ratint.for xratint.for

Kubična splajn interpolacija

➢U situacijama gdje je važna kontinuiranost derivacija funkcija moramo koristiti "spline" funkcije !

Promatramo interval određen s dvije susjedne točke, linearna interpolacija daje

$$y = Ay_j + By_{j+1}$$

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j}$$

Specijalan slučaj interpolacije Lagrange

Splajn funkcija je suma kubičnih polinoma oblika

$$S(x) = \begin{cases} s_1(x) & \text{if} \quad x_1 \le x < x_2 \\ s_2(x) & \text{if} \quad x_2 \le x < x_3 \\ \vdots & \\ s_{n-1}(x) & \text{if} \quad x_{n-1} \le x < x_n \end{cases}$$

$$s_i(x) = a_i + b_i(x\text{-}x_i) + c_i(x\text{-}x_i)^2 + d_i(x\text{-}x_i)^3$$

Kubična splajn interpolacija

- Za N točaka (x_i,y_i), i=1,2,…,N, za svaki interval i do i+1, napravi kubični polinom tako da vrijedi $s_i(x_i)=y_i$ and $s_i(x_{i+1})=y_{i+1}$.
- 1. i 2. derivacija moraju biti kontinuirane u intervalu, tj., $s_i^{(n)}(x_{i+1}) = s^{(n)}_{i+1}(x_{i+1})$, n = 1 and 2.
- Rubni uvjet $s''(x_{1 \text{ ili } N})=0$, or $s''(x_{1 \text{ ili } N}) =$ konstanta.

Kubični polinom koji zadovoljava uvjete ima oblik:

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1}$$

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2$$

Polinom ispunjava cilj kubične splajn interpolacije: 1.) glatka interpolacija u prvoj derivaciji, 2.) kontinuirana u drugoj derivaciji, na intervalu i na granici.

Kubična splajn interpolacija

Prva derivacija je

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1}$$

druga derivacija

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1}$$

Druga derivacija se dobiva iz jednakosti prve derivacije u točki $x=x_j$ iz dva susjedna intervala, nakon uređivanja:

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}$$

Imamo N-2 linearne jednadžbe i N nepoznanica (druge derivacije).

1. **Prirodni kubični splajn**, druge derivacije na rubu su jednake nula
2. Vrijednosti druge derivacije na rubu se računaju iz zadanih prvih derivacija

Podprogrami

SUBROUTINE spline(x,y,n,yp1,ypn,y2)

Given arrays x(1:n) and y(1:n) containing a tabulated function, i.e., yi = f(xi), with $x_1 < x_2 < \ldots < x_N$, and given values yp1 and ypn for the first derivative of the interpolating function at points 1 and n, respectively, this routine returns an array y2(1:n) of length n which contains the second derivatives of the interpolating function at the tabulated points xi. If yp1 and/or ypn are equal to 1×10^{30} or larger, the routine is signaled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary. Parameter: NMAX is the largest anticipated value of n.

void spline(float x[], float y[], int n, float yp1, float ypn, float y2[])

SUBROUTINE splint(xa,ya,y2a,n,x,y)

Given the arrays xa(1:n) and ya(1:n) of length n, which tabulate a function (with the xai 's in order), and given the array y2a(1:n), which is the output from spline above, and given a value of x, this routine returns a cubic-spline interpolated value y.

void splint(float xa[], float ya[], float y2a[], int n, float x, float *y)

primjeri

NR primjer:

gcc -o splintC spline.c splint.c xsplint.c nrutils/nrutil.c -I nrutils/ -lm

isto kao kod xpolint.c, samo su vektori duljine 10, i potrebne su derivacije u 1 i zadnjoj tocki.

```
spline(xa,ya,NP,yp1,ypn,y2);
splint(xa,ya,y2,NP,x,&y);
```

F77:

f77 -o splintF xsplint.for splint.for spline.for

C:

gcc -g -o tsplint tsplint.c splint.c spline.c polint.c nrutils/nrutil.c libnr.c -lm -I nrutils/

open Log10.dat, call polint and save in Interpsp.dat, funkciju log(x) iz 0.1 i 10

F77:

f77 -o tsplintF tsplint.for splint.for spline.for lib1.f

Zadaci za praktikum

1. Izvršite programe xpolcoe, xpolcof koji određuju koeficijente polinoma
2. Izvršite programe xpolint i tpolint koji koriste Lagrange interpolaciju
3. Izvršite program xratint, interpolaciju racionalnim funkcijama
4. Izvršite programe xsplint, tsplint za kubičnu splajn interpolaciju
5. Notebook PlotTable.nb za program Mathematica koristite kako bi usporedili krivulje dobivene splajn i Lagrange interpolacijom za funkciju log(x). Log10.dat sadrži tablicu s 40 točaka ove funkcije.
6. Notebook PolCoeff.nb ilustrira polinom interpolacije 4 stupnja. Provjerite da li povećanje stupnja polinoma daje bolju ekstrapolaciju.
7. Napravite Lagrange i splajn interpolaciju na Runge.dat podacima. Nacrtajte krivulje. (1 ili 3 .nb u attachment) Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- George Em Karniadakis and Robert M. Kirby II: Parallel Scientific Computing in C++ and MPI, Cambridge University Press.

/home/maks/numerickemetode/v5/PlotTable.nb

File Edit Cell Format Input Kernel Find Window Help

100% Input << Go Back

```
Directory[]
```

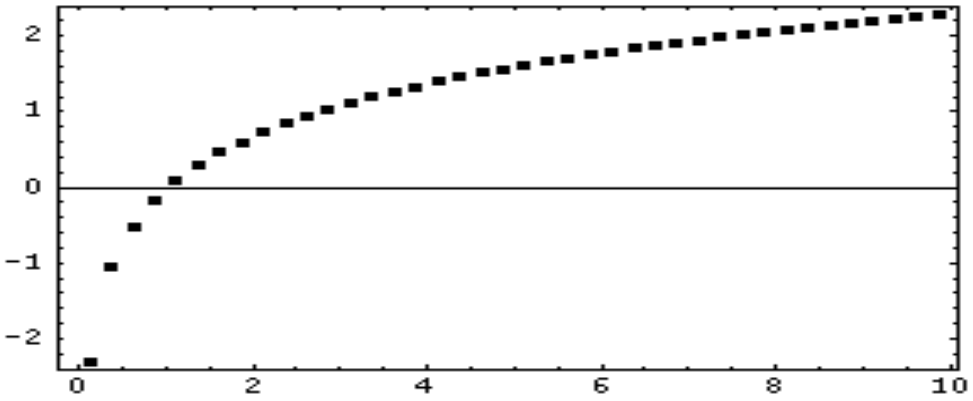
```
/home/maks
```

```
SetDirectory["/home/maks/numerickemetode/v5"]
```

```
/home/maks/numerickemetode/v5
```

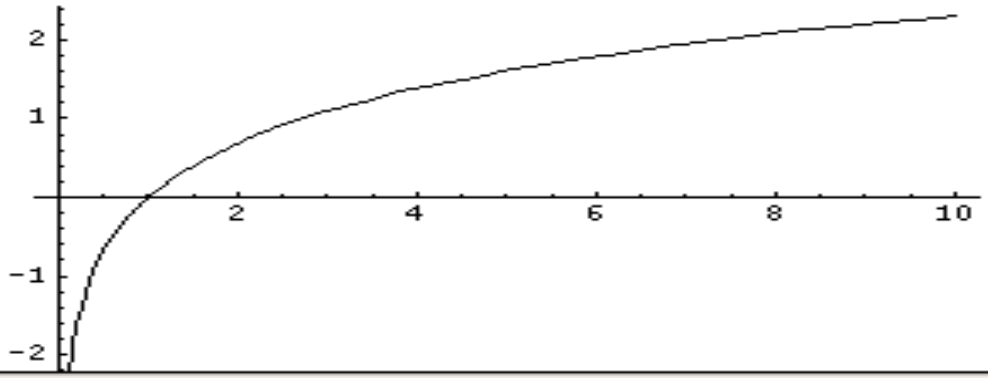
```
xyinterp = Import["InterpspF.dat", "Table"];
```

```
g1 = lPlot[xyinterp]
```



- Graphics -

```
g2 = Plot[Log[x], {x, 0.1, 10}]
```

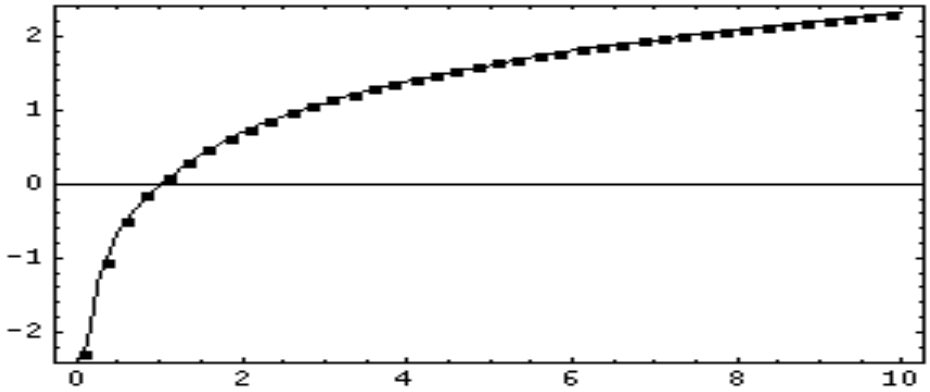


/home/maks/numerickemetode/v5/PlotTable.nb

File Edit Cell Format Input Kernel Find Window Help

100% Input << Go Back

```
Show[g1, g2]
```



- Graphics -

```
lPlot[matr_] :=
  Module[{g1}, g1 = ListPlot[matr, DisplayFunction → Identity, Frame -> True];
   Show[g1, Prolog → AbsolutePointSize[5], DisplayFunction -> $DisplayFunction]];
```

$$f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1]. \tag{3.6}$$

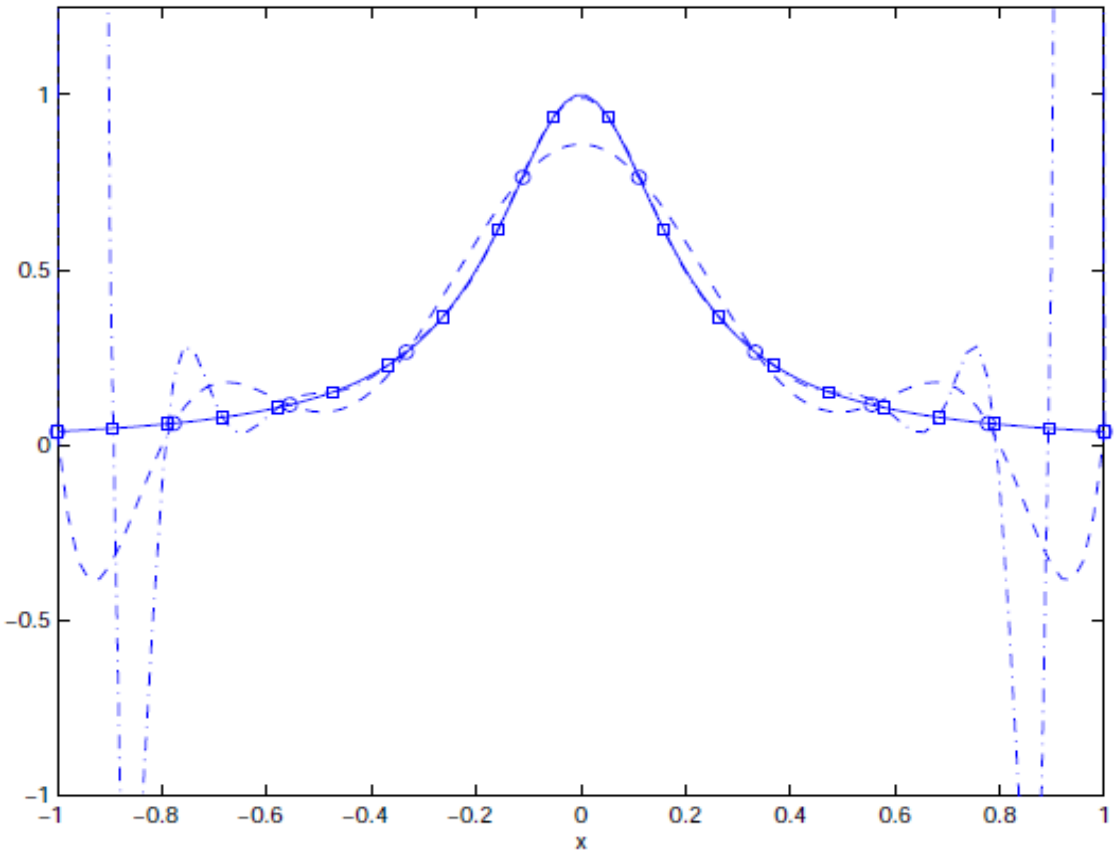


Figure 3.3: Plot of the Runge function (equation (3.4); solid line) and approximations using 10 equidistant points (dashed line) and 20 equidistant points (dashed-dot line).

Numerička integracija

Želimo izračunati integral funkcije f na intervalu $[a, b]$,

$$I(f) = \int_a^b f(x)\,dx.$$

Newton-Liebnitz formula. Ne znamo *I(f)*, preostaje numerika.

$$I(f) = \int_a^b f(x)\,dx = F(b) - F(a).$$

Identičan problem, tražimo vrijednost *I=y(b)* za diferencijalnu jednadžbu s rubnim uvjetom:

$$\frac{dy}{dx} = f(x) \qquad y(a) = 0$$

Rubni uvjet

Numerička integracija

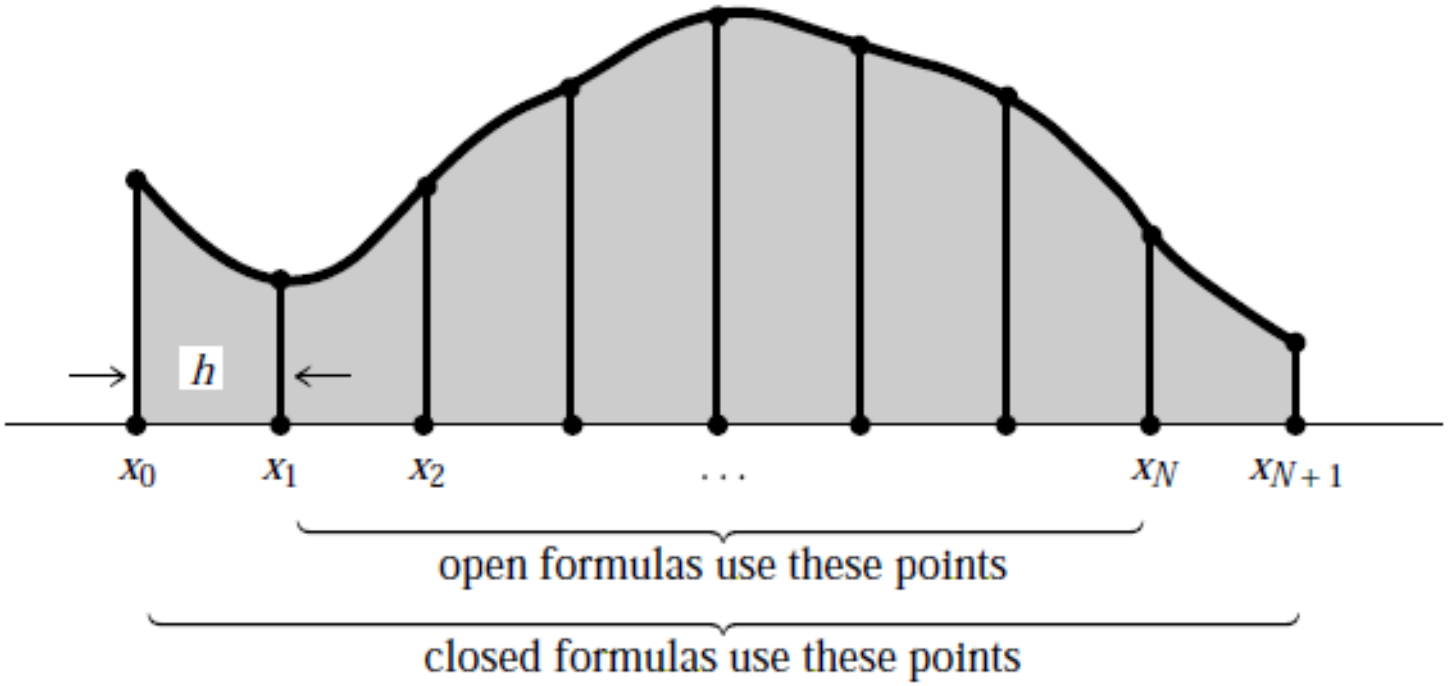
Opća integracijska formula ima oblik

$$I(f) = I_m(f) + E_m(f),$$

pri čemu je $m+1$ broj korištenih točaka, $I_m(f)$ pripadna aproksimacija integrala, a $E_m(f)$ pritom napravljena greška. Ovakve formule za približnu integraciju funkcija jedne varijable (tj. na jednodimenzionalnoj domeni) često se zovu i **kvadraturne** formule, zbog interpretacije integrala kao površine ispod krivulje.

$$I_m(f) = \sum_{k=0}^{m} w_k^{(m)} f(x_k^{(m)}),$$

Numerička integracija



$$x_i = x_0 + ih \qquad i = 0, 1, \ldots, N+1$$

h konstantan korak

$$f(x_i) \equiv f_i$$

Newton Cotes

Newton–Cotesove formule zatvorenog tipa imaju ekvidistantne čvorove, s tim da je prvi čvor u točki $x_0 := a$, a posljednji u $x_m := b$. Preciznije, za zatvorenu (to se često ispušta) Newton–Cotesovu formulu s $(m+1)$-nom točkom čvorovi su

$$x_k^{(m)} = x_0 + kh_m, \quad k = 0, \ldots, m, \quad h_m = \frac{b-a}{m}.$$

Drugim riječima, osnovni je oblik Newton–Cotesovih formula

$$\int_a^b f(x)\,dx \approx I_m(f) = \sum_{k=0}^{m} w_k^{(m)} f(x_0 + kh_m).$$

Trapezna formula

Izvedimo najjednostavniju (zatvorenu) Newton–Cotesovu formulu za $m = 1$.

Za $m = 1$, aproksimacija integrala (9.2.1) ima oblik

$$I_1(f) = w_0^{(1)} f(x_0) + w_1^{(1)} f(x_0 + h_1),$$

$$h := h_1 = \frac{b-a}{1} = b - a,$$

moramo pronaći težine w_0 i w_1, tako da integracijska formula egzaktno integrira polinome što višeg stupnja na intervalu $[a, b]$, tj. da za polinome f što višeg stupnja bude

$$\int_a^b f(x)\,dx = I_1(f) = w_0 f(a) + w_1 f(b).$$

Trapezna formula

Za $f(x) = 1 = x^0$ dobivamo

$$b - a = \int_a^b x^0 \, dx = w_0 \cdot 1 + w_1 \cdot 1.$$

Za $f(x) = x$ izlazi

$$\frac{b^2 - a^2}{2} = \int_a^b x \, dx = w_0 \cdot a + w_1 \cdot b.$$

Trapezna formula

Sada imamo dvije jednadžbe s dvije nepoznanice

$$w_0 + w_1 = b - a$$
$$aw_0 + bw_1 = \frac{b^2 - a^2}{2}.$$

Pomnožimo li prvu jednadžbu s $-a$ i dodamo drugoj, dobivamo

$$(b-a)w_1 = \frac{b^2 - a^2}{2} - a(b-a) = \frac{b^2 - 2ab + a^2}{2} = \frac{(b-a)^2}{2}.$$

$$w_1 = \frac{1}{2}(b-a) = \frac{h}{2}. \qquad w_0 = b - a - w_1 = \frac{1}{2}(b-a) = \frac{h}{2},$$

Trapezna formula

Vidimo da je integracijska formula $I_1(f)$ dobivena iz egzaktnosti na svim polinomima stupnja manjeg ili jednakog 1, i glasi

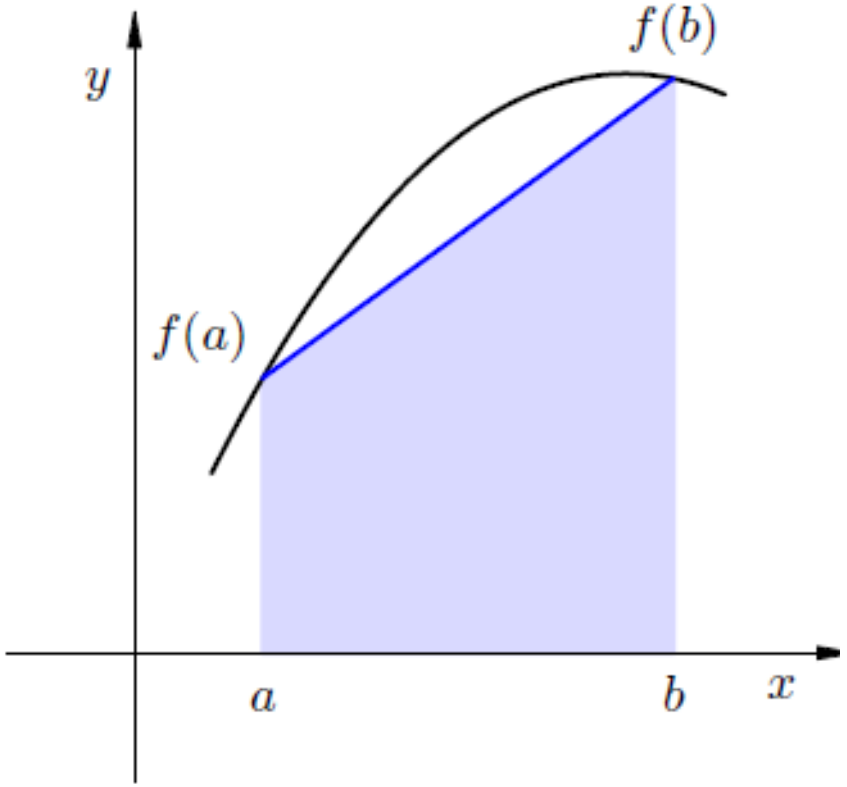
$$\int_a^b f(x)\,dx \approx \frac{h}{2}\,(f(a)+f(b)).$$

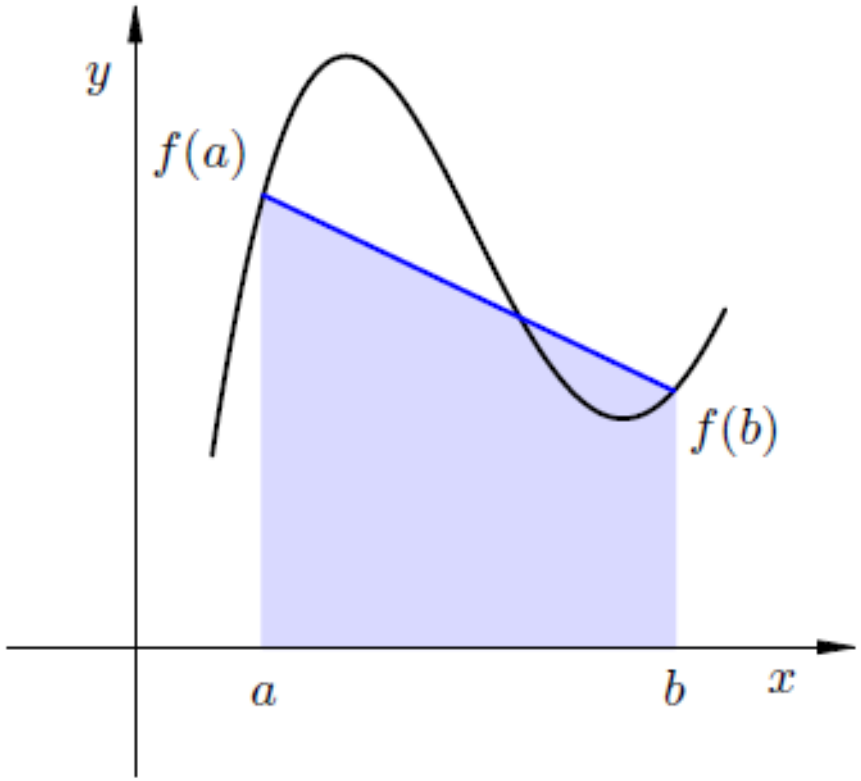
Ta formula zove se trapezna formula. Odakle joj ime? Napišemo li je na malo drugačiji način, kao

$$\int_a^b f(x)\,dx \approx \frac{f(a)+f(b)}{2}\,(b-a),$$

odmah ćemo vidjeti da je $(f(a)+f(b))/2$ srednjica, a $b-a$ visina trapeza sa slike.

Trapezna formula





Desna slika prikazuje funkciju koju loše aproksimira pravac.

Simpsonova formula

Izvedimo sljedeću (zatvorenu) Newton–Cotesovu formulu za $m = 2$, poznatu pod imenom Simpsonova formula.

Za $m = 2$, aproksimacija integrala (9.2.1) ima oblik

$$I_2(f) = w_0^{(2)} f(x_0) + w_1^{(2)} f(x_0 + h_2) + w_2^{(2)} f(x_0 + 2h_2),$$

$$h := h_2 = \frac{b-a}{2}.$$

$$I_2(f) = w_0 f(a) + w_1 f\left(\frac{a+b}{2}\right) + w_2 f(b).$$

Za $f(x) = 1$ dobivamo

$$b - a = \int_a^b x^0 \, dx = w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1.$$

Simpsonova formula

Za $f(x) = x$ izlazi

$$\frac{b^2 - a^2}{2} = \int_a^b x\,dx = w_0 \cdot a + w_1 \frac{a+b}{2} + w_2 \cdot b.$$

Konačno, za $f(x) = x^2$ dobivamo

$$\frac{b^3 - a^3}{3} = \int_a^b x^2\,dx = w_0 \cdot a^2 + w_1 \frac{(a+b)^2}{4} + w_2 \cdot b^2.$$

Simpsonova formula

Sada imamo linearni sustav s tri jednadžbe i tri nepoznanice

$$w_0 + w_1 + w_2 = b - a$$
$$aw_0 + \frac{a+b}{2} w_1 + bw_2 = \frac{b^2 - a^2}{2}$$
$$a^2 w_0 + \frac{(a+b)^2}{4} w_1 + b^2 w_2 = \frac{b^3 - a^3}{3}.$$

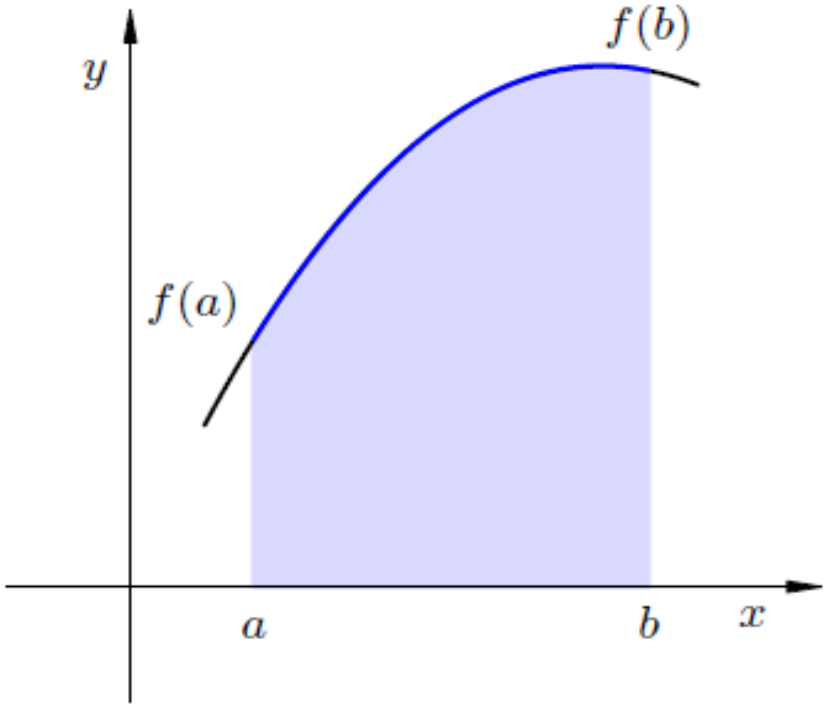
Rješavanjem ovog sustava, dobivamo

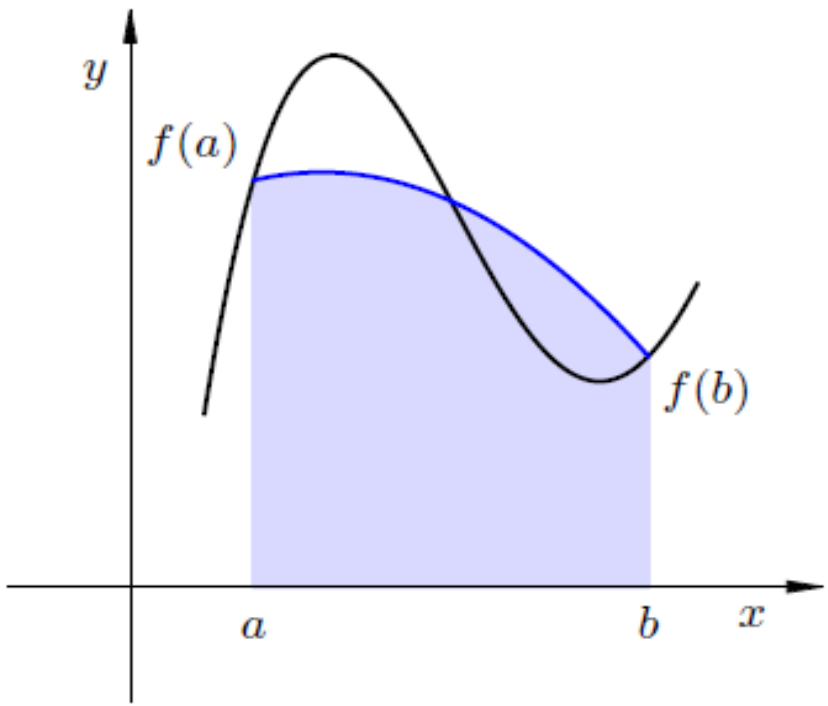
$$w_0 = w_2 = \frac{h}{3} = \frac{b-a}{6}, \quad w_1 = \frac{4h}{3} = \frac{4(b-a)}{6}.$$

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

Simpsonova formula

Ako pogledamo kako ona funkcionira na funkcijama koje smo već integrirali trapeznom formulom, vidjet ćemo da joj je greška bitno manja. Posebno, na prvom primjeru, kvadratni interpolacijski polinom tako dobro aproksimira funkciju f, da se one na grafu ne razlikuju.





Formula srednje točke (midpoint)

Ako u Newton–Cotesovim formulama ne interpoliramo (pa onda niti ne integriramo) jednu ili obje rubne točke, dobili smo otvorene Newton–Cotesove formule. Ako definiramo $x_{-1} := a$, $x_{m+1} := b$ i

$$h_m = \frac{b-a}{m+2},$$

onda otvorene Newton–Cotesove formule imaju oblik

$$\int_a^b f(x)\,dx \approx I_m(f) = \sum_{k=0}^{m} w_k^{(m)} f(x_0 + kh_m). \qquad (9.2.6)$$

Vjerojatno najkorištenija i najpoznatija otvorena Newton–Cotesova formula je ona najjednostavnija za $m = 0$, poznata pod imenom "midpoint formula" (formula srednje točke).

Formula srednje točke (midpoint)

Dakle za bismo odredili midpoint formulu, moramo naći koeficijent $w_0 := w_0^{(0)}$ takav da je

$$\int_a^b f(x)\,dx = w_0 f\left(\frac{a+b}{2}\right)$$

egzaktna na vektorskom prostoru polinoma što višeg stupnja.

Za $f(x) = 1$, imamo

$$b - a = \int_a^b 1\,dx = w_0,$$

odakle odmah slijedi da je

$$\int_a^b f(x)\,dx = (b-a) f\left(\frac{a+b}{2}\right).$$

Produljene formule

Općenito, produljenu trapeznu formulu dobivamo tako da cijeli interval $[a, b]$ podijelimo na n podintervala oblika $[x_{k-1}, x_k]$, za $k = 1, \ldots, n$, s tim da je

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b,$$

i na svakom od njih upotrijebimo "običnu" trapeznu formulu. Znamo da je tada

$$\int_a^b f(x)\,dx = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)\,dx,$$

Aproksimacija produljenom trapeznom formulom je

$$\int_a^b f(x)\,dx = h\left(\frac{1}{2}f_0 + f_1 + \cdots + f_{n-1} + \frac{1}{2}f_n\right) + E_n^T(f),$$

Gaussove integracijske formule

Za razliku od Newton–Côtesovih formula, **Gaussove integracijske formule** su oblika

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i),$$

u kojima točke integracije x_i nisu unaprijed poznate, nego se izračunaju tako da greška takve formule bude najmanja. Motivirani praktičnim razlozima, promatrat ćemo malo općenitije integracijske formule oblika

$$\int_a^b w(x)\,f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i),$$

gdje je w **težinska funkcija**, pozitivna na otvorenom intervalu (a, b). Koeficijente w_i zovemo **težinski koeficijenti** ili, skraćeno, **težine** integracijske formule. specijalni slučaj u kojem je $w \equiv 1$ čine formule koje se zovu **Gauss–Legendreove**

Gaussove integracijske formule

Bitno je znati da se za neke težinske funkcije na određenim intervalima, čvorovi i težine standardno tabeliraju u priručnicima. To su

težinska funkcija w	interval	formula Gauss–
1	$[-1,1]$	Legendre
$\frac{1}{\sqrt{1-x^2}}$	$[-1,1]$	Čebišev
$\sqrt{1-x^2}$	$[-1,1]$	Čebišev 2. vrste
e^{-x}	$[0,\infty)$	Laguerre
e^{-x^2}	$(-\infty,\infty)$	Hermite

Gaussove integracijske formule

Glavni rezultat je sljedeći: ako zahtijevamo da formula integrira egzaktno polinome što je moguće većeg stupnja, onda su točke integracije x_i nultočke polinoma koji su ortogonalni na intervalu (a, b) obzirom na težinsku funkciju w, a težine w_i mogu se eksplicitno izračunati po formuli

$$w_i = \int_a^b w(x)\, \ell_i(x)\, dx, \quad i = 1, \ldots, n.$$

Pritom je ℓ_i poseban polinom Lagrangeove baze

$$\int_a^b w(x)\, f(x)\, dx \approx \sum_{i=1}^{n} w_i f(x_i), \longrightarrow \int_a^b g(x)dx \approx \sum_{j=1}^{N} v_j g(x_j)$$

$$g(x) \equiv W(x) f(x) \text{ and } v_j \equiv w_j / W(x_j)$$

Gauss integracija

Budući da su apscise i težine ortogonalnih polinoma tabelirane (Gauss Legendre, *w(x)=1*) možemo napisati podprogram tipa qgaus bez znanja o detaljima teorije.

Kao što znamo, Legendreov polinom stupnja n definiran je **Rodriguesovom formulom**

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n.$$

Tako definirani polinomi čine **ortogonalnu bazu** u prostoru polinoma stupnja n, tj. oni su linearno nezavisni i ortogonalni obzirom na skalarni produkt

$$\langle P, Q \rangle := \int_{-1}^{1} P(x)\, Q(x)\, dx. \qquad \int_{-1}^{1} P_m(x)\, P_n(x)\, dx = 0, \quad \text{za} \quad m \neq n.$$

Ortogonalni polinomi

$$p_{-1}(x) \equiv 0$$

$$p_0(x) \equiv 1$$

$$p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x) \qquad j =$$

$$a_j = \frac{\langle xp_j | p_j \rangle}{\langle p_j | p_j \rangle} \qquad j = 0, 1, \ldots$$

$$b_j = \frac{\langle p_j | p_j \rangle}{\langle p_{j-1} | p_{j-1} \rangle} \qquad j = 1, 2, \ldots$$

Točke integracije u Gaussovim formulama su nultočke polinoma $p_N(x)$ kod integracije u N točaka. Kad je apscisa poznata, težinske funkcije se mogu odrediti iz linearnog sustava (nije najbolja metoda)

$$\begin{bmatrix} p_0(x_1) & \ldots & p_0(x_N) \\ p_1(x_1) & \ldots & p_1(x_N) \\ \vdots & & \vdots \\ p_{N-1}(x_1) & \ldots & p_{N-1}(x_N) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \int_a^b W(x)p_0(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

$$w_j = \frac{\langle p_{N-1} | p_{N-1} \rangle}{p_{N-1}(x_j)p'_N(x_j)}$$

$$w_j = \frac{2}{(1 - x_j^2)[P'_N(x_j)]^2}$$

Gauss Legendre težinske funkcije

podprogrami

SUBROUTINE trapzd(func,a,b,s,n)

This routine computes the nth stage of refinement of an extended trapezoidal rule. func is input as the name of the function to be integrated between limits a and b, also input. When called with n=1, the routine returns as s the crudest estimate of $\int_a^b f(x)dx$. Subsequent calls with n=2,3,... (in that sequential order) will improve the accuracy of s by adding 2^{n-2} additional interior points. s should not be modied between sequential calls.

float trapzd(float (*func)(float), float a, float b, int n)

SUBROUTINE qtrap(func,a,b,s)

C USES trapzd

Returns as s the integral of the function func from a to b. The parameters EPS can be set to the desired fractional accuracy and JMAX so that 2 to the power JMAX-1 is the maximum allowed number of steps. Integration is performed by the trapezoidal rule.

float qtrap(float (*func)(float), float a, float b)

podprogrami

SUBROUTINE qsimp(func,a,b,s)

C USES trapzd

Returns as s the integral of the function func from a to b. The parameters EPS can be set to the desired fractional accuracy and JMAX so that 2 to the power JMAX-1 is the maximum allowed number of steps. Integration is performed by Simpson's rule.

float qsimp(float (*func)(float), float a, float b)

podprogrami

SUBROUTINE qgaus(func,a,b,ss)

Returns as ss the integral of the function func between a and b, by ten-point Gauss-Legendre integration: the function is evaluated exactly ten times at interior points in the range of integration.

float qgaus(float (*func)(float), float a, float b)

SUBROUTINE gauleg(x1,x2,x,w,n)

Given the lower and upper limits of integration x1 and x2, and given n, this routine returns arrays x(1:n) and w(1:n) of length n, containing the abscissas and weights of the Gauss-Legendre n-point quadrature formula.

void gauleg(float x1, float x2, float x[], float w[], int n),

NR primjeri

Trapezna i simpson formula za funkciju: $x^2\ (x^2-2)\ sin(x)$ na intervalu [0,pi/2]

gcc -o xqtrap xqtrap.c qtrap.c trapzd.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o xqtrap xqtrap.for trapzd.for qtrap.for

gcc -o xqsimp xqsimp.c qsimp.c trapzd.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o xqsimp xqsimp.for qsimp.for trapzd.for

Midpoint formula za funkciju: $1/\sqrt{x}$, na intervalu [0,1]

gcc -o xmidpnt xmidpnt.c midpnt.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o xmidpnt xmidpnt.for midpnt.for

Gauss-Legendre integracija za funkciju $x\ exp(-x)$ na intervalu [0,5]

gcc -o xqgaus xqgaus.c qgaus.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o xqgaus xqgaus.for qgaus.for

izračuna 10 integrala od [0,0.5 i], i=1..10

gcc -o xgauleg xgauleg.c gauleg.c nrutils/nrutil.c -I nrutils/

f77 -o xgauleg xgauleg.for gauleg.for

izračuna $w(i)$ za interval [0,1], i integral za interval [0,10]

Nepravi integrali

Uglavnom se radi promjena varijabli kako bi se uklonio singularitet, odnosno beskonačna granica se preslikava u konačnu.
Slijedeća relacija se može koristiti za $b \to \infty$ i a pozitivno, ili b negativno i $a \to -\infty$

$$\int_a^b f(x)dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt \qquad ab > 0$$

Za midpoint formulu jednadžba je implementirana u rutini midinf.
Primjer: Integral od funkcije *exp(x)* od [0,∞], rastavljen je na integracije od [0,1] i [1,∞].

gcc -g -o tmidinf tmidinf.c midinf.c nrutils/nrutil.c midpnt.c -I nrutils/ -lm
f77 -o tmidinf tmidinf.for midinf.for midpnt.for

Nepravi integrali

SUBROUTINE midinf(funk,aa,bb,s,n)

This routine is an exact replacement for midpnt, i.e., returns as s the nth stage of refinement of the integral of funk from aa to bb, except that the function is evaluated at evenly spaced points in 1/x rather than in x. This allows the upper limit bb to be as large and positive as the computer allows, or the lower limit aa to be as large and negative, but not both.

aa and bb must have the same sign.

float midinf(float (*funk)(float), float aa, float bb, int n)

Zadaci za praktikum

1. integral od 0 do beskonačno za funkciju f(x)=(2x+4) exp(-x)
2. integral od 2 do 5 za funkciju f(x)= 1/[x(2x+3)]

2a. Koliko iznosi čvor integracije i težina u Gaussovoj formuli za n=1 i n=10?

2b. Usporedite rezultat s nekom drugom metodom koju smo obradili

3. Iskoristite program Mathematica i notebooka "Integrali.nb" kako bi utvrdili točnu vrijednost integrala.

4. Pošaljite mail na Aleksandar.Maksimovic@irb.hr, koji će imati rezultate integrala pod 1, 2a i 2b, te procjenu pogreške za korištene metode dobivene usporedbom s rezultatima iz Mathematice.

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)

Iterativne metode

Računanje nultočaka nelinearnih funkcija jedan je od najčešćih zadataka primijenjene matematike. Općenito, neka je zadana funkcija

$$f : I \to \mathbb{R},$$

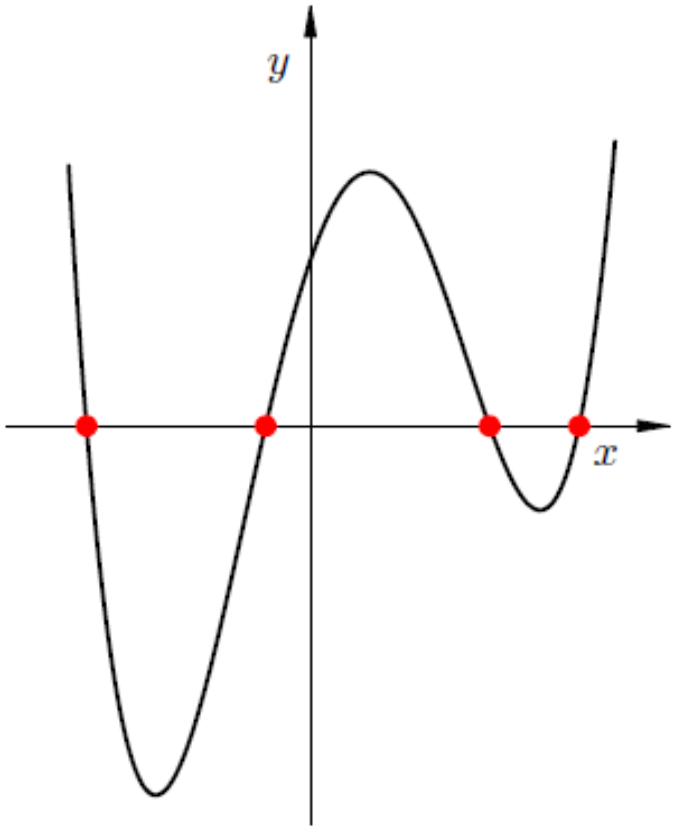
gdje je I neki interval. Tražimo sve one $x \in I$ za koje je

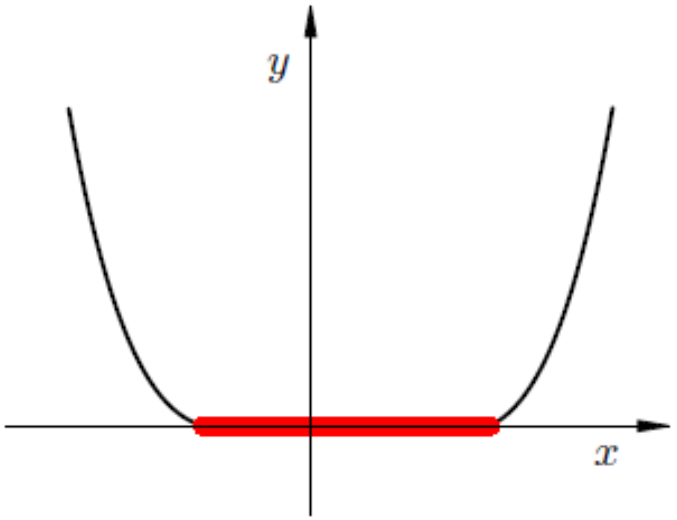
$$f(x) = 0.$$

Takve točke x zovu se rješenja, korijeni pripadne jednadžbe ili nultočke funkcije f.

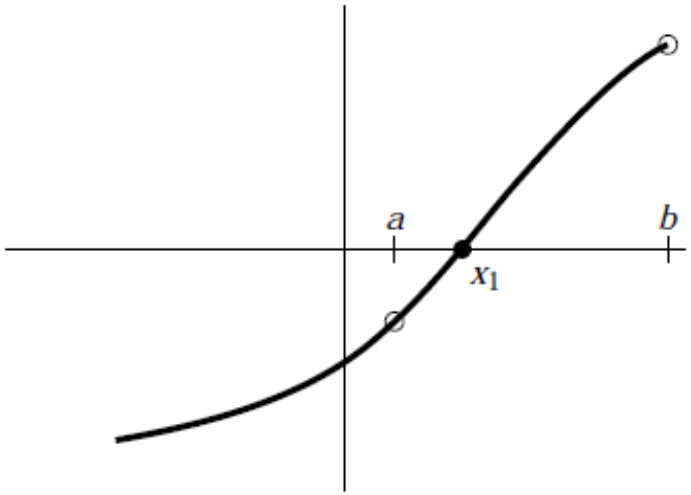
U pravilu, pretpostavljamo da je f **neprekidna** na I i da su joj nultočke izolirane. U protivnom postojao bi problem konvergencije.

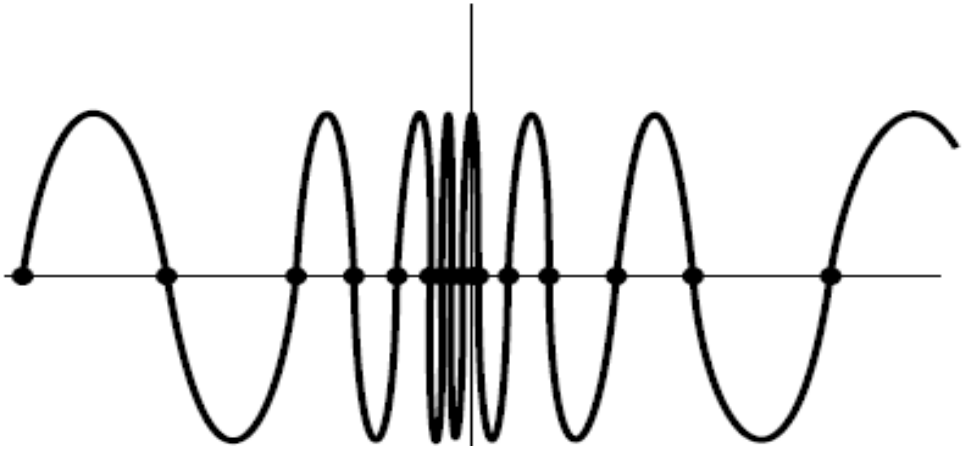
Iterativne metode

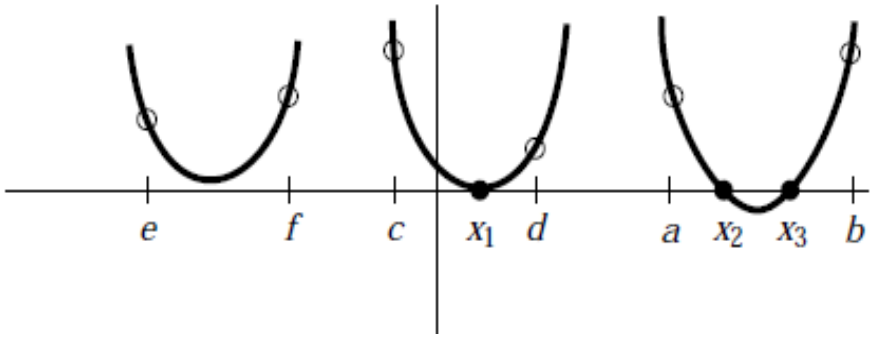


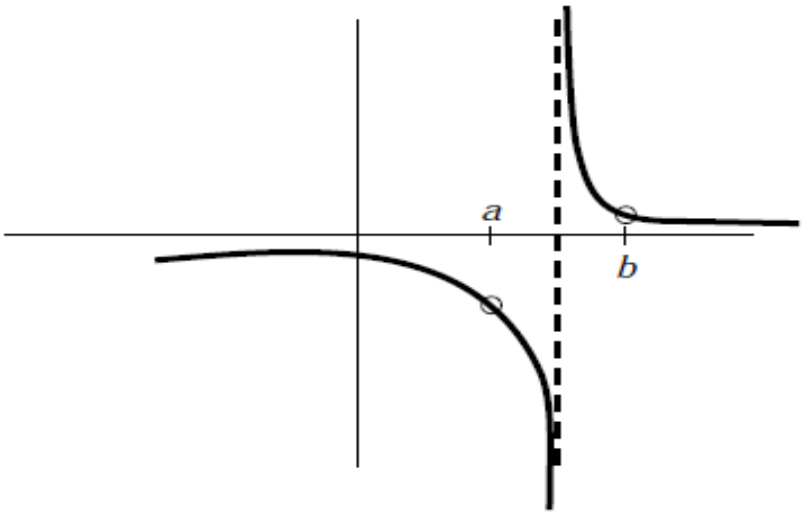


Primjeri funkcija









Iterativne metode

Traženje nultočki na zadanu točnost sastoji se od dvije faze.

1. Izolacije jedne ili više nultočki, tj. nalaženje intervala I unutar kojeg se nalazi bar jedna nultočka. Ovo je teži dio posla i obavlja se na temelju analize toka funkcije.
2. Iterativno nalaženje nultočke na traženu točnost.

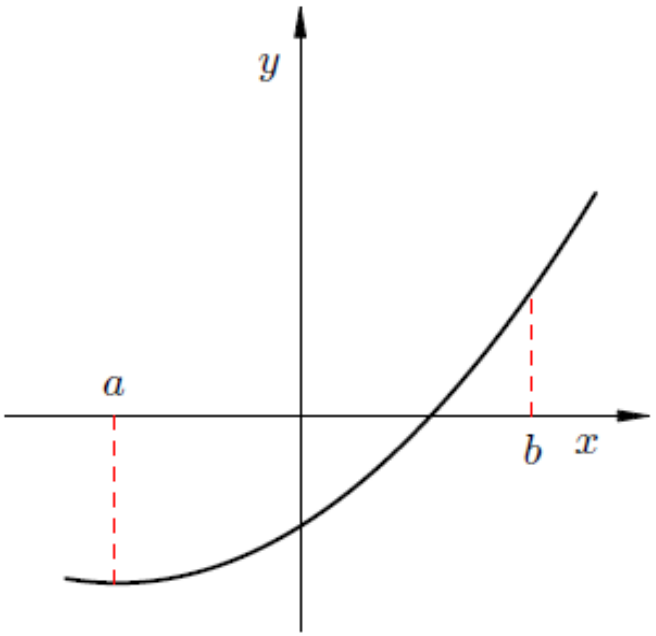
Postoji mnogo metoda za nalaženje nultočaka nelinearnih funkcija na zadanu točnost. One se bitno razlikuju po tome hoće li uvijek konvergirati, tj. imamo li sigurnu konvergenciju ili ne i po brzini konvergencije. Uobičajen je slučaj da brze metode nemaju sigurnu konvergenciju, dok je sporije metode imaju.

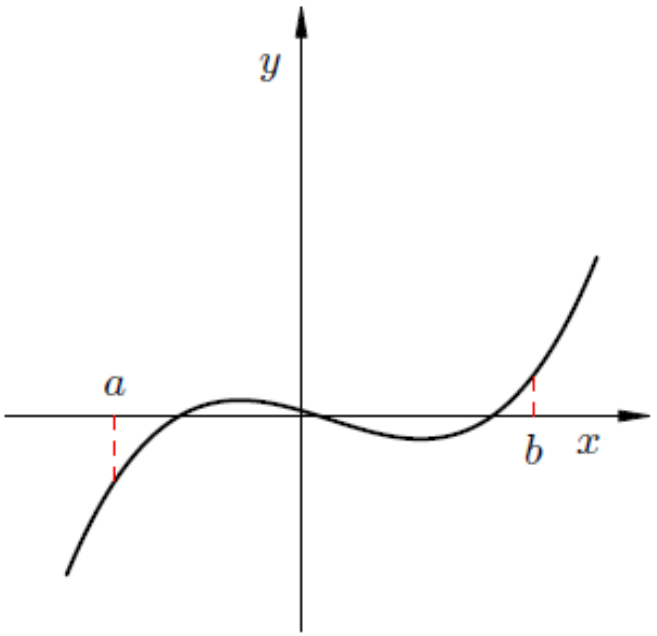
metoda bisekcije

Osnovna pretpostavka za primjenu algoritma raspolavljanja je **neprekidnost** funkcije f na intervalu $[a, b]$ i uvjet

$$f(a) \cdot f(b) < 0.$$

Prethodna relacija znači da funkcija f ima na intervalu $[a, b]$ **barem jednu** nultočku.

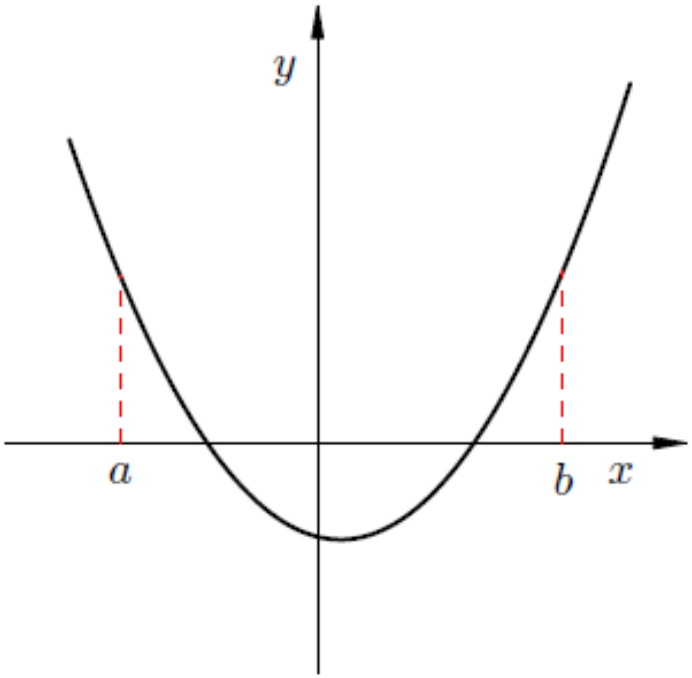


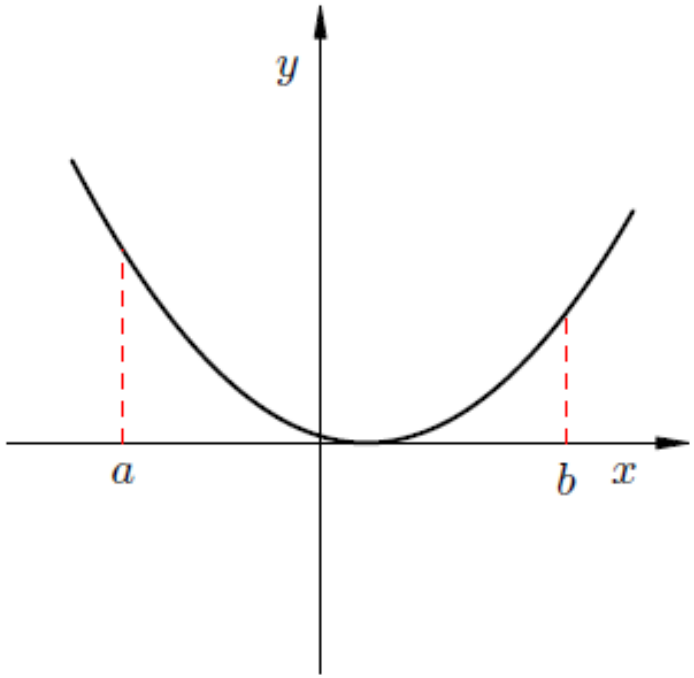


metoda bisekcije

S druge strane, ako je

$$f(a) \cdot f(b) > 0,$$





metoda bisekcije

Algoritam raspolavljanja je vrlo jednostavan. Označimo s α pravu nultočku funkcije, a zatim s $a_0 := a$, $b_0 := b$ i x_0 polovište $[a_0, b_0]$, tj.

$$x_0 = \frac{a_0 + b_0}{2}.$$

Neka je $n \geq 1$. U n-tom koraku algoritma konstruiramo interval $[a_n, b_n]$ kojemu je duljina polovina duljine prethodnog intervala, ali tako da je nultočka ostala unutar intervala $[a_n, b_n]$.

Konstrukcija intervala $[a_n, b_n]$ sastoji se u raspolavljanju intervala $[a_{n-1}, b_{n-1}]$ točkom x_{n-1} i to tako da je

$$a_n = x_{n-1},\ b_n = b_{n-1} \quad \text{ako je} \quad f(a_{n-1}) \cdot f(x_{n-1}) > 0,$$
$$a_n = a_{n-1},\ b_n = x_{n-1} \quad \text{ako je} \quad f(a_{n-1}) \cdot f(x_{n-1}) < 0.$$

Postupak zaustavljamo kad je $|\alpha - x_n| \leq \varepsilon.$

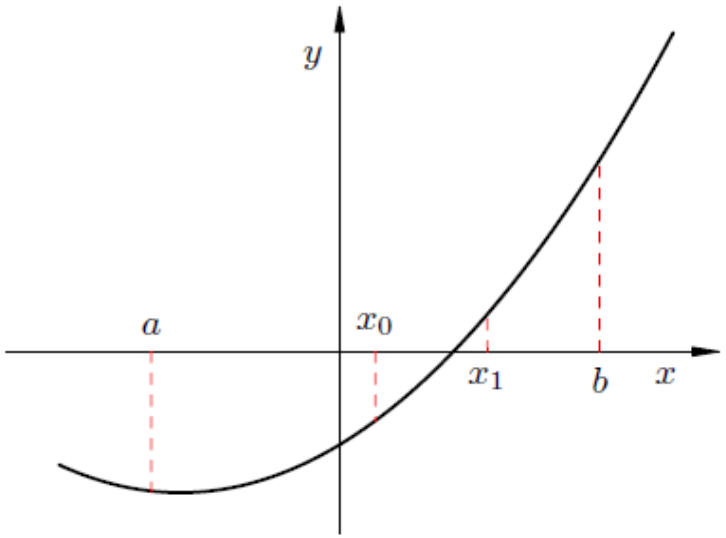
metoda bisekcije

budući da je x_n polovište intervala $[a_n, b_n]$, a $\alpha \in [a_n, b_n]$, onda je

$$|\alpha - x_n| \leq b_n - x_n,$$

pa je dovoljno postaviti zahtjev

$$b_n - x_n \leq \varepsilon.$$



metoda bisekcije

Algoritam 8.2.1 (Metoda raspolavljanja)

$x := (a + b)/2;$
while $b - x > \varepsilon$ **do**
  begin;
  if $f(x) * f(b) < 0.0$ **then**
    $a := x$
  else
    $b := x;$
  $x := (a + b)/2;$
  end;
{ Na kraju je $x \approx \alpha$. }

metoda bisekcije

Iz konstrukcije metode lako se izvodi pogreška n-te aproksimacije nultočke. Vrijedi

$$|\alpha - x_n| \leq b_n - x_n = \frac{1}{2}(b_n - a_n) = \frac{1}{2^2}(b_{n-1} - a_{n-1}) = \cdots = \frac{1}{2^{n+1}}(b-a).$$

$$\frac{1}{2^{n+1}}(b-a) \leq \varepsilon. \qquad \text{a zatim logaritmiranjem nejednakosti}$$

$$n \geq \frac{\log(b-a) - \log\varepsilon}{\log 2} - 1, \quad n \in \mathbb{N}_0.$$

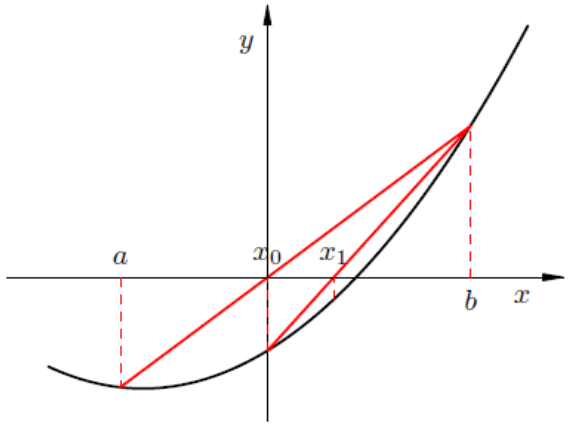
Opaska: U NR koristi se n-1 aproksimacija, tj. $n = \log_2 \frac{\epsilon_0}{\epsilon}$

Regula falsi

Aproksimirajmo funkciju f pravcem koji prolazi točkama $(a, f(a))$, $(b, f(b))$. Njegova je jednadžba

$$y - f(b) = \frac{f(a) - f(b)}{a - b}(x - b), \quad \text{odnosno} \quad y - f(a) = \frac{f(b) - f(a)}{b - a}(x - a).$$

Nultočku α funkcije f možemo aproksimirati nultočkom tog pravca, točkom x_0. Nakon toga, pomaknemo ili točku a ili točku b u x_0, ali tako da nultočka α ostane unutar novodobivenog intervala. Postupak ponavljamo sve dok ne postignemo željenu točnost.



Regula falsi

Točka x_0 dobiva se jednostavno iz jednadžbe pravca, pa je

$$x_0 = b - f(b)\,\frac{b-a}{f(b)-f(a)} = a - f(a)\,\frac{a-b}{f(a)-f(b)},$$

$$x_0 = b - \frac{f(b)}{f[a,b]} = a - \frac{f(a)}{f[a,b]}, \qquad f[a,b] = \frac{f(b)-f(a)}{b-a},$$

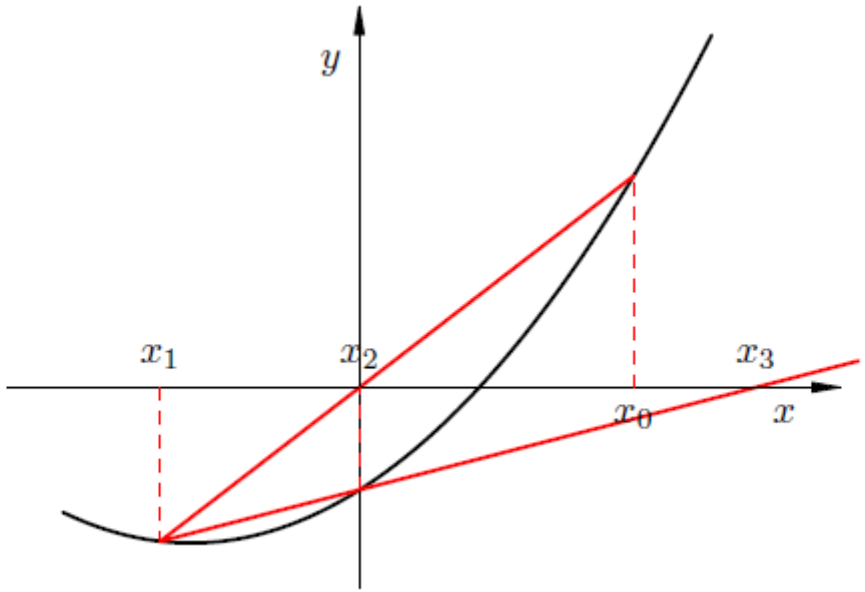
Metoda sekante

Ako graf funkcije f aproksimiramo sekantom, slično kao kod *regule falsi*, samo ne zahtijevamo da nultočka funkcije f ostane "zatvorena" unutar posljednje dvije iteracije, dobili smo metodu sekante. Time smo izgubili svojstvo sigurne konvergencije, ali se nadamo da će metoda, kad konvergira konvergirati brže nego *regula falsi*.

Počinjemo s dvije početne točke x_0 i x_1 i povlačimo sekantu kroz $(x_0, f(x_0))$, $(x_1, f(x_1))$. Ta sekanta siječe os x u točki x_2. Postupak nastavljamo povlačenjem sekante kroz posljednje dvije točke $(x_1, f(x_1))$ i $(x_2, f(x_2))$. Formule za metodu sekante dobivaju se iteriranjem početne formule za *regulu falsi*, tako da dobivamo

$$x_{n+1} = x_n - f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Metoda sekante



Primijetite da je treće iteracija izašla izvan početnog intervala, pa metoda sekante ne mora konvergirati. Jednako tako, da smo "prirodno" numerirali prve dvije točke, tako da je $x_0 < x_1$, imali bismo konvergenciju prema rješenju.

Metoda sekante

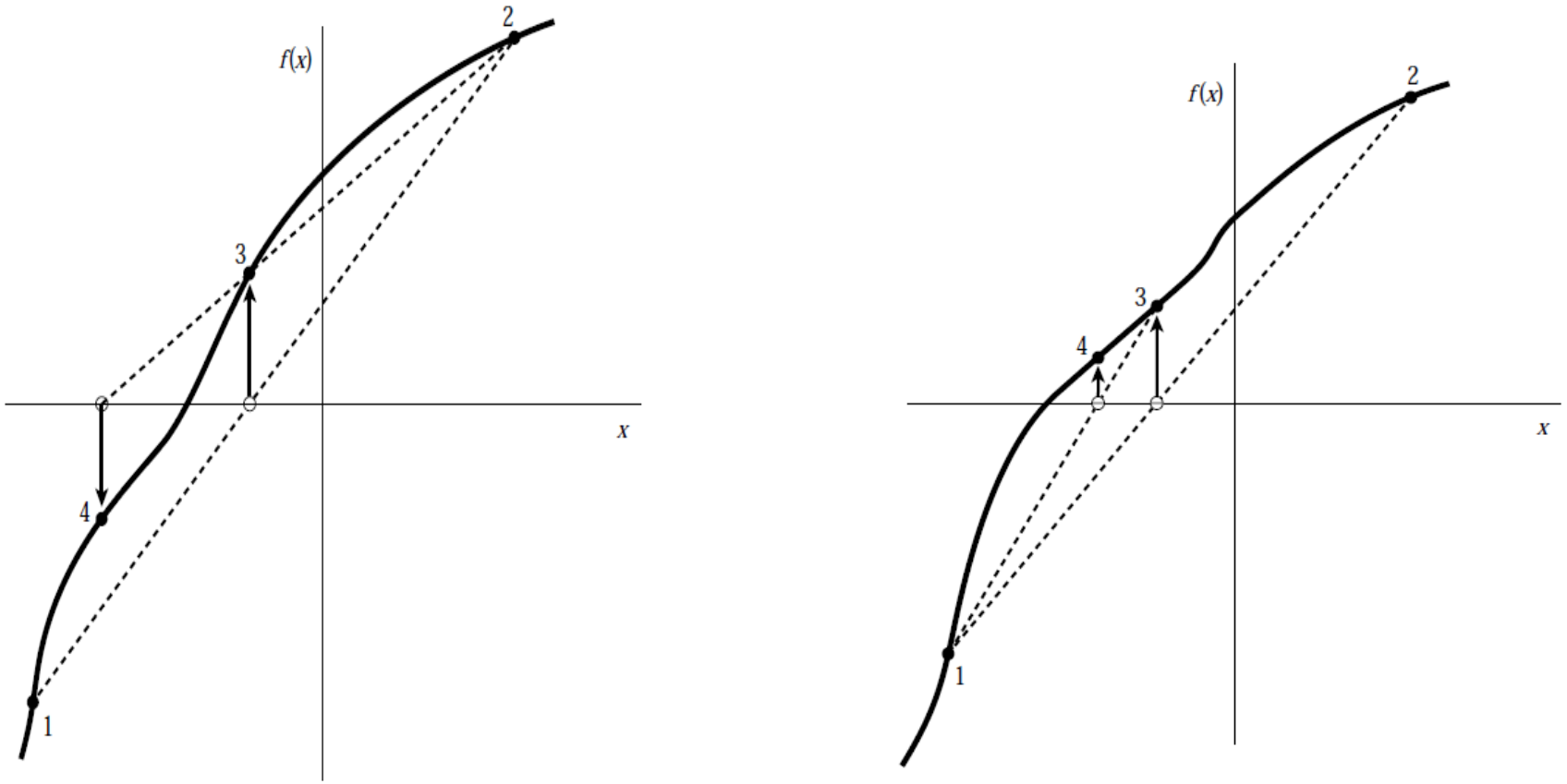
Kod metode sekante postoji nekoliko problema. Prvi je da može divergirati ako početne aproksimacije nisu dobro odabrane. Drugi problem se može javiti zbog kraćenja u brojniku i (posebno) nazivniku kvocijenta

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})},$$

kad $x_n \to \alpha$. Osim toga, budući da iteracije ne "zatvaraju" nultočku s obje strane nije lako reći kad treba zaustaviti iterativni proces.

Konačno, primijetimo da je za svaku iteraciju metode sekante potrebno samo jednom izvrednjavati funkciju f i to u točki x_n, jer $f(x_{n-1})$ čuvamo od prethodne iteracije.

regula falsi i sekanta

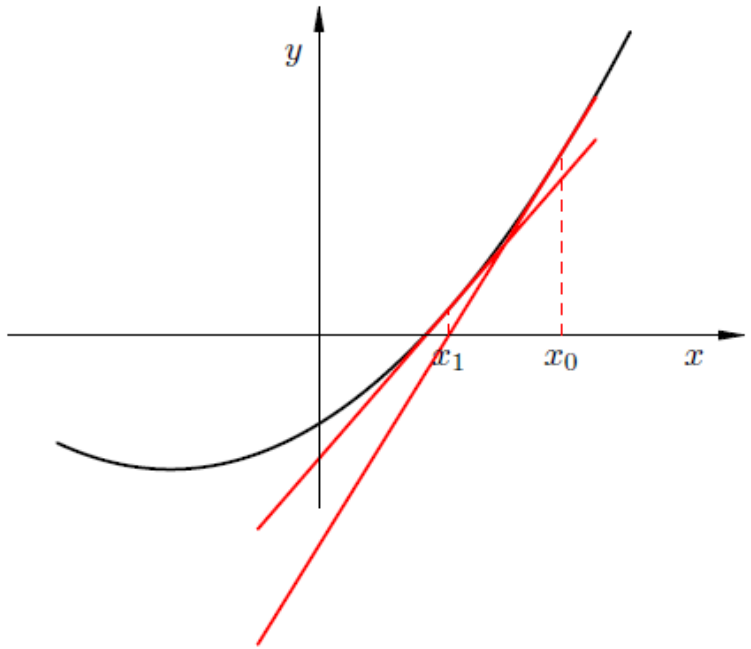


Metoda sekante i regula falsi metoda

Metoda tangente

Ako graf funkcije f umjesto sekantom, aproksimiramo tangentom, dobili smo metodu tangente ili Newtonovu metodu. Slično kao i kod sekante, time smo izgubili svojstvo sigurne konvergencije, ali se nadamo da će metoda brzo konvergirati.

Pretpostavimo da je zadana početna točka x_0. Ideja metode je povući tangentu u točki $(x_0, f(x_0))$ i definirati novu aproksimaciju x_0 u točki gdje ona siječe os x.



Metoda tangente

Geometrijski izvod je jednostavan. U točki x_n napiše se jednadžba tangente i pogleda se gdje siječe os x. Jednadžba tangente je

$$y - f(x_n) = f'(x_n)(x - x_n),$$

odakle izlazi da je nova aproksimacija $x_{n+1} := x$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Primijetite da je prethodna formula usko vezana uz metodu sekante, jer je

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Hibridna Brent-Dekker metoda

Brent–Dekkerova metoda smišljena je kao metoda koja će imati sigurnu konvergenciju, a nadamo se da će konvergirati brže nego metoda sekante, u najboljem slučaju kvadratično. Ona **ne zahtijeva** računanje derivacija, pa ako joj je red konvergencije u prosjeku bolji od sekante, možemo očekivati da će metoda po brzini biti slična Newtonovoj, ali će imati sigurnu konvergenciju.

Metoda se sasoji od tri dijela, koje grubo možemo opisati kao inverznu kvadratnu interpolaciju, metodu sekante i metodu bisekcije. Algoritam počinje metodom sekante koja generira treću točku. Ako se prema nekim kriterijima ta točka prihvaća kao dobra, možemo nastaviti raditi s kvadratnom interpolacijom kroz posljednje tri točke, ali inverznom (uloga x i y zamijenjena) i time dobivamo četvrtu točku.

Ako je treća točka odbačena kao loša, radi se jedan korak metode bisekcije. Drugim riječima, metoda se "vrti" između svoja tri sastavna dijela, a mi se nadamo da će rijetko koristiti bisekciju.

podprogrami

SUBROUTINE zbrac(func,x1,x2,succes)

Given a function func and an initial guessed range x1 to x2, the routine expands the range geometrically until a root is bracketed by the returned values x1 and x2 (in which case succes returns as .true.) or until the range becomes unacceptably large (in which case succes returns as .false.).

int zbrac(float (*func)(float), float *x1, float *x2)

root is bracketed by the returned values x1 and x2 (in which case zbrac returns 1) or until the range becomes unacceptably large (in which case zbrac returns 0).

SUBROUTINE zbrak(fx,x1,x2,n,xb1,xb2,nb)

Given a function fx dened on the interval from x1-x2 subdivide the interval into n equally spaced segments, and search for zero crossings of the function. nb is input as the maximum number of roots sought, and is reset to the number of bracketing pairs xb1(1:nb), xb2(1:nb) that are found.

void zbrak(float (*fx)(float), float x1, float x2, int n, float xb1[], float xb2[], int *nb)

podprogrami

FUNCTION rtbis(func,x1,x2,xacc)

PARAMETER (JMAX=40) Maximum allowed number of bisections.

Using bisection, find the root of a function func known to lie between x1 and x2. The root, returned as rtbis, will be refined until its accuracy is ±xacc.

float rtbis(float (*func)(float), float x1, float x2, float xacc)

FUNCTION rtflsp(func,x1,x2,xacc)

Using the false position method, find the root of a function func known to lie between x1 and x2. The root, returned as rtflsp, is refined until its accuracy is ±xacc.

float rtflsp(float (*func)(float), float x1, float x2, float xacc)

FUNCTION rtsec(func,x1,x2,xacc)

PARAMETER (MAXIT=30) Maximum allowed number of iterations.

Using the secant method, find the root of a function func thought to lie between x1 and x2. The root, returned as rtsec, is refined until its accuracy is ±xacc

float rtsec(float (*func)(float), float x1, float x2, float xacc)

podprogrami

FUNCTION zbrent(func,x1,x2,tol)

Using Brent's method, find the root of a function func known to lie between x1 and x2.

The root, returned as zbrent, will be refined until its accuracy is tol.

Parameters: Maximum allowed number of iterations, and machine floating-point precision.

float zbrent(float (*func)(float), float x1, float x2, float tol)

FUNCTION rtnewt(funcd,x1,x2,xacc)

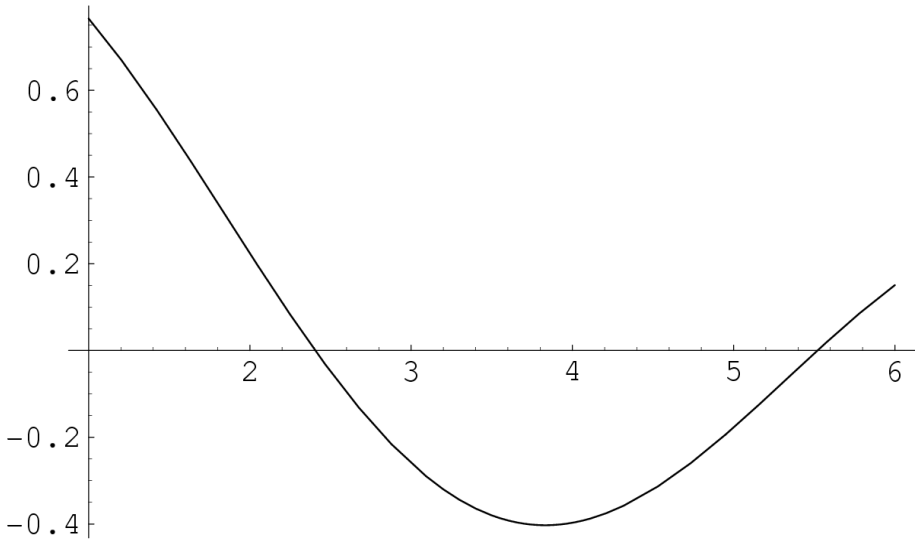
PARAMETER (JMAX=20) Set to maximum number of iterations.

Using the Newton-Raphson method, find the root of a function known to lie in the interval [x1; x2]. The root rtnewt will be refined until its accuracy is known within xacc. funcd is a user-supplied subroutine that returns both the function value and the first derivative of the function at the point x.

float rtnewt(void (*funcd)(float, float *, float *), float x1, float x2, float xacc)

NR primjeri

Besselova funkcija 0-tog reda prve vrste $J_n(x)$




```
for (i=1;i<=10;i++) {
                    x1=i;
                    x2=x1+1.0;
                    succes=zbrac(fx,&x1,&x2);
          .....
Bracketing values:      Function values:
   X1       X2          BESSJ0(X1)  BESSJ0(X2)
  1.00     3.60         0.765198   -0.391769
  2.00     3.00         0.223891   -0.260052
```

```
f77 -o Fzbrac xzbrac.for zbrac.for bessj0.for
 gcc -o Czbrac xzbrac.c zbrac.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Fzbrak xzbrak.for zbrak.for bessj0.for
gcc -o Czbrak xzbrak.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
```

NR primjeri

```
gcc -o Czbrac xzbrac.c zbrac.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Fzbrak xzbrak.for zbrak.for bessj0.for
```

```
xb1=vector(1,NBMAX);
        xb2=vector(1,NBMAX);
        zbrak(fx,X1,X2,N,xb1,xb2,&nb);
```

	lower	upper	F(lower)	F(upper)
Root 1	1.9800	2.4700	0.2354	-0.0334
Root 2	5.4100	5.9000	-0.0378	0.1220

```
gcc -o Crtbis xrtbis.c rtbis.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/-lm
f77 -o Frtbis xrtbis.for  rtbis.for zbrak.for bessj0.for
```

```
xb1=vector(1,NBMAX);
xb2=vector(1,NBMAX);
zbrak(fx,X1,X2,N,xb1,xb2,&nb);
for (i=1;i<=nb;i++) {
  xacc=(1.0e-6)*(xb1[i]+xb2[i])/2.0;
   root=rtbis(fx,xb1[i],xb2[i],xacc);
```

	x	F(x)
Root 1	2.404827	-0.5610E-06
Root 2	5.520077	-0.4607E-06

Analogni primjer za metodu sekante i false position

```
f77 -o Frtsec xrtsec.for  rtsec.for zbrak.for bessj0.for
gcc -o Crtsec xrtsec.c rtsec.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Frtflsp xrtflsp.for  rtflsp.for zbrak.for bessj0.for
gcc -o Crtflsp xrtflsp.c rtflsp.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
```

NR primjeri

f77 -o Fzbrent xzbrent.for zbrent.for zbrak.for bessj0.for

gcc -o Czbrent xzbrent.c zbrent.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm

./Czbrent

Roots of bessj0:

		x	f(x)
root	1	2.404826	-0.000000
root	2	5.520078	0.000000

```
xb1=vector(1,NBMAX);
        xb2=vector(1,NBMAX);
        zbrak(fx,X1,X2,N,xb1,xb2,&nb);
        for (i=1;i<=nb;i++) {
                tol=(1.0e-6)*(xb1[i]+xb2[i])/2.0;
                root=zbrent(fx,xb1[i],xb2[i],tol);
```

f77 -o Frtnewt xrtnewt.for rtnewt.for zbrak.for bessj0.for bessj1.for

gcc -o Crtnewt xrtnewt.c rtnewt.c zbrak.c bessj0.c bessj1.c nrutils/nrutil.c -I nrutils/ -lm

Isti princip za metodu Newtona:

Roots of bessj0:

		x	f(x)
root	1	2.404825	0.000000
root	2	5.520078	0.000000
root	3	8.653728	0.000000

Primjer

Treći korijen broja 1.5 može se prikazati pomoću jedandžbe x^3-1.5=0, odnosno nalaženje nultočke funkcije f(x)=x^3-1.5.

Metoda bisekcije:

f77 -o Fbis trtbis.for rtbis.for zbrak.for

gcc -o Cbis trtbis.c rtbis.c zbrak.c nrutils/nrutil.c -I nrutils/ -lm

Newtonova metoda:

f77 -o Fnewt trtnewt.for rtnewt.for zbrak.for

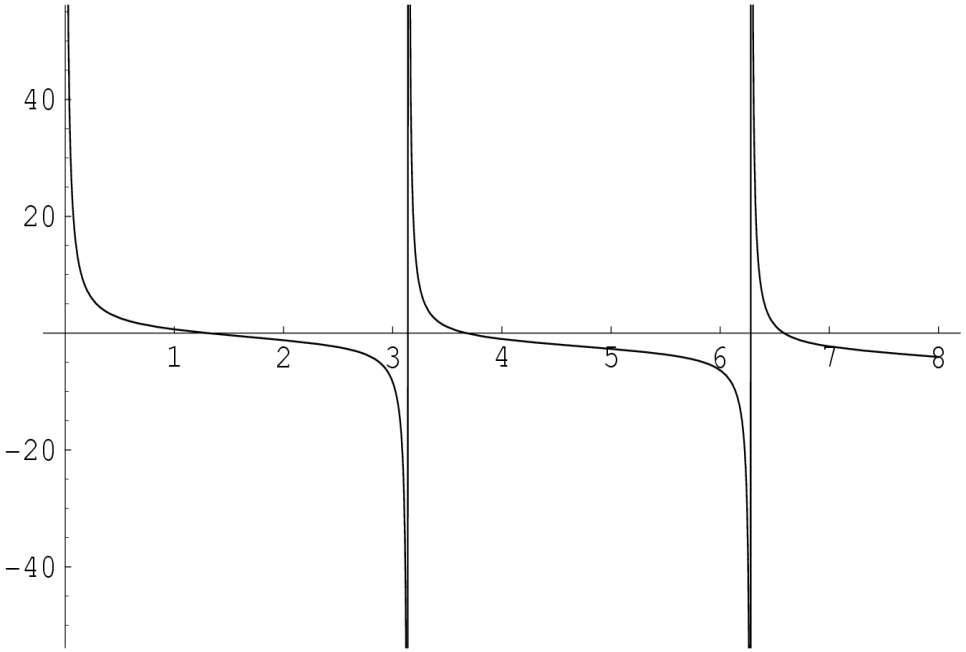
gcc -o Cnewt trtnewt.c rtnewt.c zbrak.c nrutils/nrutil.c -I nrutils/ -lm

Obratite pažnju na definiciju prve derivacije kod ove metode.

Zadatak za praktikum

Prilikom transporta neutrona u štapu dolazi do transcendentalne jednadžbe. O korijenima jednadžbe ovise kritične duljine, za štap duljine L jednadžba je

$$\cot(\ell x) = \frac{x^2 - 1}{2x}.$$



Graf prikazuje funkciju za vrijednost L=1.

Odredi prve dvije najmanje pozitivne nultočke.

Koristite graf kako bi odredili interval za traženje nultočke.

cot=1/tan, csc=1/sin, f' - prva derivacija

f'(x)=-csc(x)^2+(x^2-1)/(2*x^2)-1

Pošaljite rezultat i source na mail Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- George Em Karniadakis and Robert M. Kirby II: Parallel Scientific Computing in C++ and MPI, Cambridge University Press.

ODJ

Opisat ćemo nekoliko najčešćih numeričkih metoda za rješavanje običnih diferencijalnih jednadžbi (skraćeno ODJ) oblika

$$y'(x) = f(x, y(x)), \quad y \in (a, b),$$

uz zadani početni uvjet $y(a) = y_0$ ili uz zadani rubni uvjet $r(y(a), y(b)) = 0$, gdje je r neka zadana funkcija.

Sustav običnih diferencijalnih jednadžbi je općenitiji problem:

$$\begin{aligned} y_1' &= f_1(x, y_1, \ldots, y_n), \\ y_2' &= f_2(x, y_1, \ldots, y_n), \\ &\vdots \\ y_n' &= f_n(x, y_1, \ldots, y_n). \end{aligned}$$

$$\mathbf{y} = [y_1, \ldots, y_n]^T \quad \text{i} \quad \mathbf{f} = [f_1, \ldots, f_n]^T$$

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)),$$

ODJ

Diferencijalne jednadžbe višeg reda

$$y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)})$$

supstitucijama

$$y_1 = y, \quad y_2 = y', \quad \ldots, \quad y_n = y^{(n-1)}$$

svodimo na sustav jednadžbi prvog reda:

$$\begin{aligned} y_1' &= y' = y_2, \\ y_2' &= y'' = y_3, \\ &\vdots \\ y_n' &= y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}) = f(x, y_1, y_2, y_3, \ldots, y_n), \end{aligned}$$

Eulerova metoda

Eulerova metoda je zasigurno najjednostavnija metoda za rješavanje inicijalnog problema za ODJ oblika

$$y' = f(x, y), \quad y(a) = y_0.$$

Metoda se zasniva na ideji da se y' u gornjoj jednadžbi zamijeni s podijeljenom razlikom

$$y'(x) = \frac{y(x+h) - y(x)}{h} + \mathcal{O}(h),$$

Zanemarivanjem kvadratnog člana u gornjem razvoju dobivamo aproksimaciju

$$y(x+h) \approx y(x) + hf(x, y(x)).$$

$h = \dfrac{b-a}{n}, \quad x_i = a + ih,$

$i = 0, \ldots, n.$

prvo aproksimiramo rješenje u točki $x_1 = a + h$

$$y(x_1) \approx y_1 = y_0 + hf(x_0, y_0).$$

Eulerova metoda

$x_2 = x_1 + h$:

$$y_2 = y_1 + hf(x_1, y_1),$$

Opisani postupak nazivamo Eulerova metoda, i možemo ga kraće zapisati rekurzijom

$$y_{i+1} = y_i + hf(x_i, y_i), \quad i = 1, \ldots, n,$$

RK metode

Koristeći sličnu ideju kao u Eulerovoj metodi, diferencijalnu jednadžbu

$$y' = f(x, y), \quad y(a) = y_0$$

na intervalu $[a, b]$, možemo rješavati tako da podijelimo interval $[a, b]$ na n jednakih podintervala, označivši

$$h = \frac{b-a}{n}, \quad x_i = a + ih, \quad i = 0, \ldots, n.$$

Sada y_{i+1}, aproksimaciju rješenja u točki x_{i+1}, računamo iz y_i korištenjem aproksimacije oblika

$$y(x+h) \approx y(x) + h\Phi(x, y(x), h; f),$$

te dobivamo rekurziju:

jednokoračne metode

$$y_{i+1} = y_i + h\Phi(x_i, y_i, h; f), \quad i = 0, \ldots, n-1.$$

RK metode

Funkciju Φ nazivamo **funkcija prirasta**, a različit izbor te funkcije definira različite metode. Uočimo da je funkcija f iz diferencijalne jednadžbe (10.3.1) parametar od Φ (tj. Φ zavisi o f). Tako je npr. u Eulerovoj metodi

$$\Phi(x, y, h; f) = f(x, y).$$

O odabiru funkcije Φ ovisi i točnost metode.

$$\Delta(x; h) = \frac{y(x+h) - y(x)}{h},$$

nazivamo **lokalna pogreška diskretizacije**.

Runge–Kuttine metode.

$$\Phi(x, y, h) = \sum_{j=1}^{r} \omega_j k_j(x, y, h),$$

RK metode

a k_j su zadani s

$$k_j(x, y, h) = f\left(x + c_j h, y + h \sum_{l=1}^{r} a_{jl} k_l(x, y, h, f)\right), \quad j = 1, \ldots, r.$$

Broj r zovemo broj stadija Runge–Kuttine (RK) metode, i on označava koliko puta moramo računati funkciju f u svakom koraku.

Različit izbor koeficijenata ω_j, c_j i a_{jl} definira različite metode.

k_i nalazi na lijevoj i na desnoj strani jednadžbe, tj. zadan je implicitno govorimo o **implicitnoj** Runge–Kuttinoj metodi.

$a_{jl} = 0$ za $l \geq j$. Tada k_j možemo izračunati preko $k_1, \ldots, k_{j-1}$,

RK metode nazivamo **eksplicitnima**.

RK metode

Primjer odabira koeficijenata prikazat ćemo na RK metodi s dva stadija:

$$\Phi(x,y,h) = \omega_1 k_1(x,y,h) + \omega_2 k_2(x,y,h),$$
$$k_1(x,y,h) = f(x,y),$$
$$k_2(x,y,h) = f(x+ah, y+ahk_1).$$

Razvojem k_2 u Taylorov red po varijabli h dobivamo

$$k_2(x,y,h) = f + h(f_x a + f_y a f) + \frac{h^2}{2}(f_{xx}a^2 + 2f_{xy}a^2 f + f_{yy}a^2 f^2) + \mathcal{O}(h^3),$$

$$y(x+h) = y(x) + hf + \frac{h^2}{2}(f_x + f_y f) + \frac{h^3}{6}[f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_y(f_x + f_y f)] + \mathcal{O}(h^4).$$

Ovdje smo iskoristili da je $y(x)$ rješenja diferencijalne jednadžbe:

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

RK metode

te pravila za deriviranje

$$y''(x) = f_x + f_y f,$$
$$y'''(x) = f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_y(f_x + f_y f).$$

Sada je lokalna pogreška diskretizacije jednaka

$$\frac{y(x+h)-y(x)}{h} - \Phi(x, y(x), h) = \frac{y(x+h)-y(x)}{h} - (\omega_1 k_1(x, y, h) + \omega_2 k_2(x, y, h))$$
$$= (1 - \omega_1 - \omega_2) f + h(f_x + f_y f)\left(\frac{1}{2} - \omega_2 a\right)$$
$$+ h^2\left[(f_{xx} + 2f_{xy}f + f_{yy}f^2) \cdot \left(\frac{1}{6} - \frac{\omega_2 a^2}{2}\right) + \frac{1}{6} f_y (f_x + f_y f)\right]$$
$$+ \mathcal{O}(h^3).$$

RK metode

$$1 - \omega_1 - \omega_2 = 0.$$ 1. red

$$\frac{1}{2} - \omega_2 a = 0$$ 2. red

$$\omega_2 = t \neq 0, \quad \omega_1 = 1 - t, \quad a = \frac{1}{2t}.$$

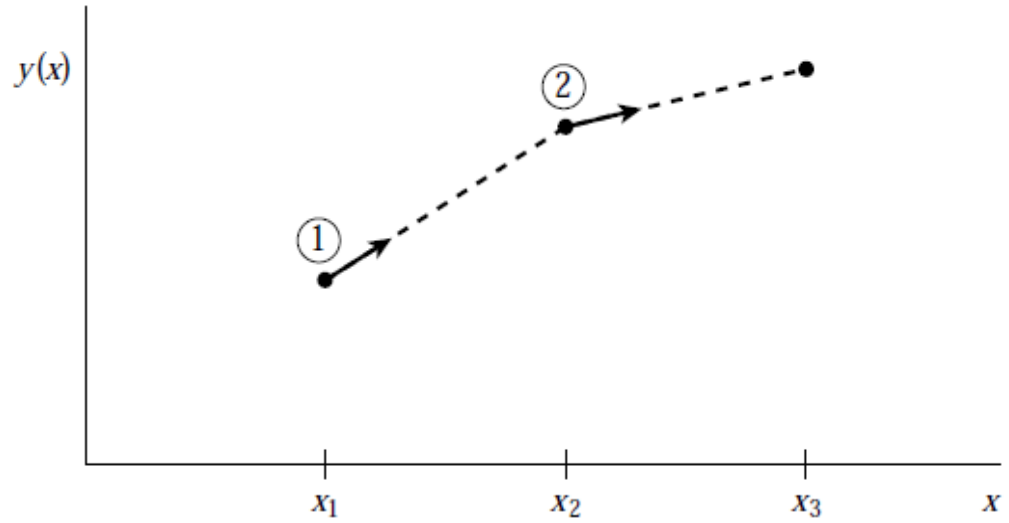
Za $t = 1/2$ dobivamo Heunovu metodu:

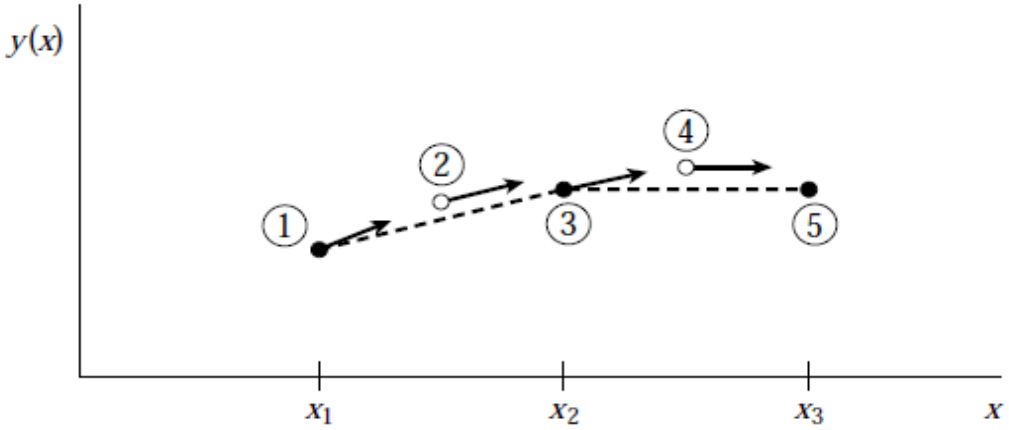
$$\Phi = \frac{1}{2}(k_1 + k_2),$$
$$k_1 = f(x, y),$$
$$k_2 = f(x + h, y + hk_1),$$

$t = 1$ dobiva modificirana Eulerova metoda:

$$\Phi = f\left(x + \frac{h}{2}, y + \frac{h}{2} f(x, y)\right).$$

RK metode





Eulerova metoda

Runge Kutta metoda drugog reda ili modificirana Eulerova metoda ili midpoint metoda.
Pogreška prvog reda eliminirana je derivacijama na početku i sredini svakog koraka.

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf\left(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1\right)$$

$$y_{n+1} = y_n + k_2 + O(h^3)$$

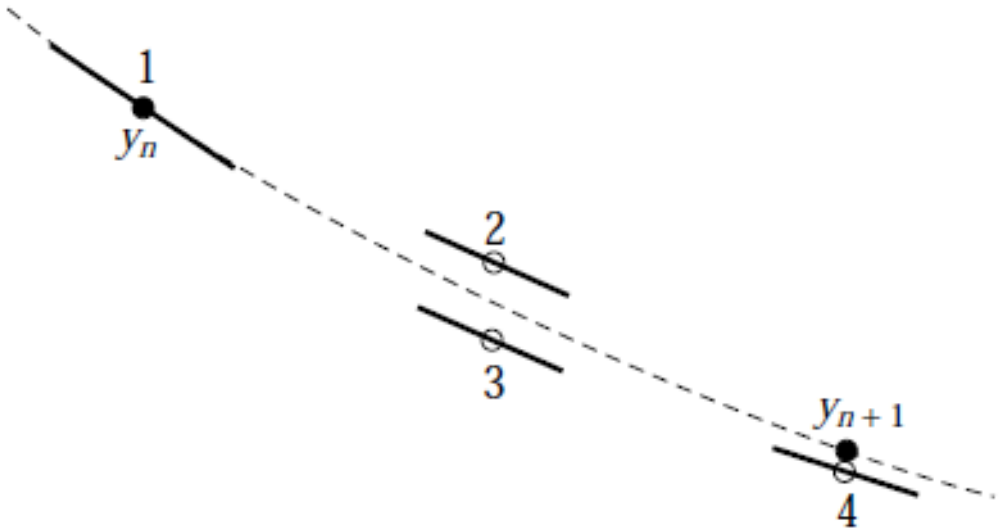
Aleksandar Maksimović,
Institute Ruđer Bošković, Bijenička c. 54

RK 4

Evo nekoliko primjera RK-4 metoda. Najpopularnija je "klasična" Runge–Kutta metoda, koja se u literaturi najčešće naziva Runge–Kutta ili RK-4 metoda (iako je to samo jedna u nizu Runge–Kutta metoda):

$$\Phi = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$
$$k_1 = f(x, y),$$
$$k_2 = f\left(x + \frac{h}{2}, y + \frac{h}{2}k_1\right),$$
$$k_3 = f\left(x + \frac{h}{2}, y + \frac{h}{2}k_2\right),$$
$$k_4 = f\left(x + h, y + hk_3\right).$$

RK 4



U svakom koraku 4 puta izračunavamo derivaciju. Pomoću informacija dobivenih derivacijama izračunava se konačna vrijednost funkcije (prikazana punim krugom na slici).

RK 3/8

Spomenimo još i 3/8-sku metodu:

$$\Phi = \frac{1}{8}(k_1 + 3k_2 + 3k_3 + k_4),$$
$$k_1 = f(x, y),$$
$$k_2 = f\left(x + \frac{h}{3}, y + \frac{h}{3}k_1\right),$$
$$k_3 = f\left(x + \frac{2}{3}h, y - \frac{h}{3}k_1 + hk_2\right),$$
$$k_4 = f(x + h, y + h(k_1 - k_2 + k_3))$$

primjer

Svedite na sistem diferencijalnih jednadžbi prvog reda i riješite RK–2 metodom s korakom $h = 0.1$ *diferencijalnu jednadžbu*

$$y'' + 2y' + 3x = 5, \quad y(0) = 1, \quad y'(0) = 2$$

u točki $x = 0.1$.

Označimo s $z = y'$. *Deriviranjem i uvrštavanjem u polaznu jednadžbu dobivamo sistem diferencijalnih jednadžbi*

$$y' = z$$
$$z' = 5 - 2z - 3x$$

uz početne uvjete $y(0) = 1$, $z(0) = 2$. *Rješenje zadatka dobivamo odmah u prvom koraku*

$$\begin{bmatrix} y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ z_0 \end{bmatrix} + \frac{1}{2}\left(\begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} + \begin{bmatrix} k_{21} \\ k_{22} \end{bmatrix} \right).$$

primjer

Uočimo da je

$$\begin{bmatrix} y' \\ z' \end{bmatrix} = \begin{bmatrix} z \\ 5 - 2z - 3x \end{bmatrix}, \quad \begin{bmatrix} y(0) \\ z(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Odatle, po formuli za RK–2 slijedi

$$\begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} = 0.1 \begin{bmatrix} 2 \\ 5 - 2 \cdot 2 - 3 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}.$$

Jednako tako, imamo

$$\begin{bmatrix} y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} = \begin{bmatrix} 1.2 \\ 2.1 \end{bmatrix},$$

odakle izračunavamo k_2

primjer

$$\begin{bmatrix} k_{21} \\ k_{22} \end{bmatrix} = 0.1 \begin{bmatrix} 2.1 \\ 5 - 2 \cdot 2.1 - 3 \cdot 0.1 \end{bmatrix} = \begin{bmatrix} 0.21 \\ 0.05 \end{bmatrix}.$$

Sve zajedno daje

$$\begin{bmatrix} y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \frac{1}{2} \left(\begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.21 \\ 0.05 \end{bmatrix} \right) = \begin{bmatrix} 1.205 \\ 2.075 \end{bmatrix},$$

što znači $y'(0.1) \approx 1.205$ *i* $z(0.1) = y'(0.1) \approx 2.075.$

Adaptivne metode

Iako smo u prošlom potpoglavlju pretpostavili da je korak integracije h konstantan tijekom cijelog postupka rješavanja diferencijalne jednadžbe, očito je da se h može mijenjati u svakom koraku integracije, pa jednokoračnu metodu možemo pisati u obliku:

$$y_{i+1} = y_i + h_i \Phi(x_i, y_i, h_i).$$

Prvo ćemo pokazati kako se određuje duljina koraka h_i tako da bude postignuta neka unaprijed zadana točnost ε.

Neka su s Φ i $\bar{\Phi}$ zadane dvije metode reda p i $p+1$. Tada računamo aproksimacije

$$y_{i+1} = y_i + h_i \Phi(x_i, y_i, h_i),$$
$$\bar{y}_{i+1} = y_i + h_i \bar{\Phi}(x_i, y_i, h_i).$$

Iz (10.3.4) slijedi da je:

$$y(x_i + h_i) = y(x_i) + h_i \Phi(x_i, y(x_i), h_i) + C(x_i) h_i^{p+1} + \mathcal{O}(h_i^{p+2}),$$

RK Fehlberg metode

$$y(x_i + h_i) = y(x_i) + h_i \bar{\Phi}(x_i, y(x_i), h_i) + \mathcal{O}(h_i^{p+2}).$$

Cilj je da pogreška u i-tom koraku bude manja od ε. Stoga ćemo pretpostaviti da je aproksimacija y_i za $y(x_i)$ točna, tj. $y_i = y(x_i)$. Sada oduzimanjem gornje dvije jednadžbe slijedi

$$h_i[\Phi(x_i, y_i, h_i) - \bar{\Phi}(x_i, y_i, h_i)] = C(x_i)h_i^{p+1} + \mathcal{O}(h_i^{p+2}).$$

Iz prve dvije jednakosti oduzimanjem slijedi

$$h_i[\Phi(x_i, y_i, h_i) - \bar{\Phi}(x_i, y_i, h_i)] = \bar{y}_{i+1} - y_{i+1},$$

te uvrštavanjem u (10.3.17) dobivamo

$$y_{i+1} - \bar{y}_{i+1} = C(x_i)h_i^{p+1} + \mathcal{O}(h_i^{p+2}).$$

RK Fehlberg metode

RK 5-tog reda

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + a_2 h, y_n + b_{21} k_1)$$

$$\dots$$

$$k_6 = hf(x_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5)$$

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

RK 4-tog reda $y^*_{n+1} = y_n + c^*_1 k_1 + c^*_2 k_2 + c^*_3 k_3 + c^*_4 k_4 + c^*_5 k_5 + c^*_6 k_6 + O(h^5)$

pogreška $\Delta \equiv y_{n+1} - y^*_{n+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i$

odnos koraka i pogreške, indeks 0 označava željenu pogrešku $h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.2}$

RK Fehlberg metode

Cash-Karp Parameters for Embedded Runga-Kutta Method								
i	a_i	b_{ij}					c_i	c_i^*
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$
$j =$		1	2	3	4	5		

podprogrami

Algoritmi: rk4, rkck, mmid, stoerm

biranje koraka: rkqs, bsstep, stiff

glavni programi: rkdumb i odeint

void rk4(float y[], float dydx[], int n, float x, float h, float yout[], void (*derivs)(float, float [], float []))

Given values for the variables y[1..n] and their derivatives dydx[1..n] known at x, use the fourth-order Runge-Kutta method to advance the solution over an interval h and return the incremented variables as yout[1..n], which need not be a distinct array from y. The user supplies the routine derivs(x,y,dydx), which returns derivatives dydx at x.

```
SUBROUTINE rk4(y,dydx,n,x,h,yout,derivs)
INTEGER n,NMAX
REAL h,x,dydx(n),y(n),yout(n)
EXTERNAL derivs
PARAMETER (NMAX=50) Set to the maximum number of functions.
```

podprogrami

void rkdumb(float vstart[], int nvar, float x1, float x2, int nstep, void (*derivs)(float, float [], float []))
Starting from initial values vstart[1..nvar] known at x1 use fourth-order Runge-Kutta
to advance nstep equal increments to x2. The user-supplied routine derivs(x,v,dvdx)
evaluates derivatives. Results are stored in the global variables y[1..nvar][1..nstep+1]
and xx[1..nstep+1].

```
SUBROUTINE rkdumb(vstart,nvar,x1,x2,nstep,derivs)
INTEGER nstep,nvar,NMAX,NSTPMX
PARAMETER (NMAX=50,NSTPMX=200) Maximum number of functions and maximum number of
values to be stored.
REAL x1,x2,vstart(nvar),xx(NSTPMX),y(NMAX,NSTPMX)
EXTERNAL derivs
COMMON /path/ xx,y Storage of results.
C USES rk4
```

podprogrami

void rkqs(float y[], float dydx[], int n, float *x, float htry, float eps, float yscal[], float *hdid, float *hnext, void (*derivs)(float, float [], float []))

Fifth-order Runge-Kutta step with monitoring of local truncation error to ensure accuracy and adjust stepsize. Input are the dependent variable vector y[1..n] and its derivative dydx[1..n] at the starting value of the independent variable x. Also input are the stepsize to be attempted htry, the required accuracy eps, and the vector yscal[1..n] against which the error is scaled. On output, y and x are replaced by their new values, hdid is the stepsize that was actually accomplished, and hnext is the estimated next stepsize. derivs is the user-supplied routine that computes the right-hand side derivatives.

```
SUBROUTINE rkqs(y,dydx,n,x,htry,eps,yscal,hdid,hnext,derivs)
INTEGER n,NMAX
REAL eps,hdid,hnext,htry,x,dydx(n),y(n),yscal(n)
EXTERNAL derivs
PARAMETER (NMAX=50) Maximum number of equations.
C USES derivs,rkck
```

podprogrami

```
extern int kmax,kount;
extern float *xp,**yp,dxsav;
```

User storage for intermediate results. Preset kmax and dxsav in the calling program. If kmax != 0 results are stored at approximate intervals dxsav in the arrays xp[1..kount], yp[1..nvar] [1..kount], where kount is output by odeint. Dening declarations for these variables, with memory allocations xp[1..kmax] and yp[1..nvar][1..kmax] for the arrays, should be in the calling program.

```
void odeint(float ystart[], int nvar, float x1, float x2, float eps, float h1,  float hmin, int *nok, int *nbad,
        void (*derivs)(float, float [], float []),  void (*rkqs)(float [], float [], int, float *, float, float, float [],
        float *, float *, void (*)(float, float [], float [])))
```

Runge-Kutta driver with adaptive stepsize control. Integrate starting values ystart[1..nvar] from x1 to x2 with accuracy eps, storing intermediate results in global variables. h1 should be set as a guessed first stepsize, hmin as the minimum allowed stepsize (can be zero). On output nok and nbad are the number of good and bad (but retried and fixed) steps taken, and ystart is replaced by values at the end of the integration interval. derivs is the user-supplied routine for calculating the right-hand side derivative, while rkqs is the name of the stepper routine to be used.

podprogrami

```
SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,rkqs)
INTEGER nbad,nok,nvar,KMAXX,MAXSTP,NMAX
REAL eps,h1,hmin,x1,x2,ystart(nvar),TINY
EXTERNAL derivs,rkqs
PARAMETER (MAXSTP=10000,NMAX=50,KMAXX=200,TINY=1.e-30)
```

Runge-Kutta driver with adaptive stepsize control. Integrate the starting values ystart(1:nvar) from x1 to x2 with accuracy eps, storing intermediate results in the common block /path/. h1 should be set as a guessed first stepsize, hmin as the minimum allowed stepsize (can be zero). On output nok and nbad are the number of good and bad (but retried and fixed) steps taken, and ystart is replaced by values at the end of the integration interval. derivs is the user-supplied subroutine for calculating the right-hand side derivative, while rkqs is the name of the stepper routine to be used. /path/ contains its own information about how often an intermediate value is to be stored.

```
COMMON /path/ kmax,kount,dxsav,xp,yp
```

User storage for intermediate results. Preset dxsav and kmax.

NR primjeri

gcc -o Cxrk4 xrk4.c rk4.c -I nrutils/ nrutils/nrutil.c bessj0.c bessj.c bessj1.c -lm

f77 -o Frk4 xrk4.for rk4.for bessj.for bes

$$x' = -y$$

$$y' = x - \frac{1}{x} y$$

$$z' = y - \frac{2}{x} z$$

$$w' = z - \frac{3}{x} w$$

```
void derivs(float x,float y[],float dydx[])
{       dydx[1] = -y[2];
        dydx[2]=y[1]-(1.0/x)*y[2];
        dydx[3]=y[2]-(2.0/x)*y[3];
        dydx[4]=y[3]-(3.0/x)*y[4];
}
```

```
SUBROUTINE derivs(x,y,dydx)
    REAL x,y(*),dydx(*)
    dydx(1)=-y(2)
    dydx(2)=y(1)-(1.0/x)*y(2)
    dydx(3)=y(2)-(2.0/x)*y(3)
    dydx(4)=y(3)-(3.0/x)*y(4)
    return
    END
```

Bessel Function: J0 J1 J3 J4

For a step size of: 0.20

RK4:	0.671133	0.498290	0.159351	0.032869
Actual:	0.671133	0.498289	0.159349	0.032874

For a step size of: 0.40

RK4:	0.566879	0.541971	0.207395	0.050358
Actual:	0.566855	0.541948	0.207356	0.050498

NR primjeri

f77 -o Frkdumb xrkdumb.for rkdumb.for rk4.for bessj.for bessj0.for bessj1.for

gcc -o Crkdumb xrkdumb.c rkdumb.c rk4.c -I nrutils/ nrutils/nrutil.c bessj0.c bessj.c bessj1.c -lm

Source F77:

```
COMMON /path/ x,y
    EXTERNAL derivs
    x1=1.0
    vstart(1)=bessj0(x1)
    vstart(2)=bessj1(x1)
    vstart(3)=bessj(2,x1)
    vstart(4)=bessj(3,x1)
    x2=20.0
    call rkdumb(vstart,NVAR,x1,x2,NSTEP,derivs)
```

Source C:

```
float x1=1.0,x2=20.0,*vstart;
        vstart=vector(1,NVAR);
        xx=vector(1,NSTEP+1);
        y=matrix(1,NVAR,1,NSTEP+1);
        vstart[1]=bessj0(x1);
        vstart[2]=bessj1(x1);
        vstart[3]=bessj(2,x1);
        vstart[4]=bessj(3,x1);
        rkdumb(vstart,NVAR,x1,x2,NSTEP,derivs);
```

primjer

Lorenzov atraktor:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = rx - y - xz$$
$$\dot{z} = xy - bz.$$

$$\sigma = 10,\ b = \tfrac{8}{3},\ r = 28.$$

gcc -o lorenz lorenz.c rkdumb.c rk4.c nrutils/nrutil.c -I nrutils/

```
void derivs(float x, float y[], float dydx[]) {
    dydx[1] = -sigma*y[1] + sigma*y[2] ;
    dydx[2] = -y[1]*y[3] + r*y[1] - y[2] ;
    dydx[3] = y[1]*y[2] - b*y[3] ;
}
```

y[1]=x, y[2]=y,y[3]=z.

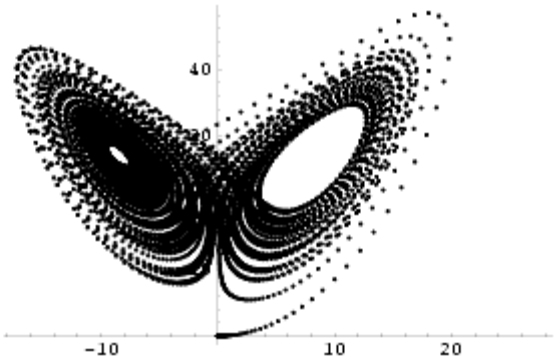
f77 -g -o Florenz lorenz.for rk4.for

rkdumb podprogram nalazi se u lorenz.for

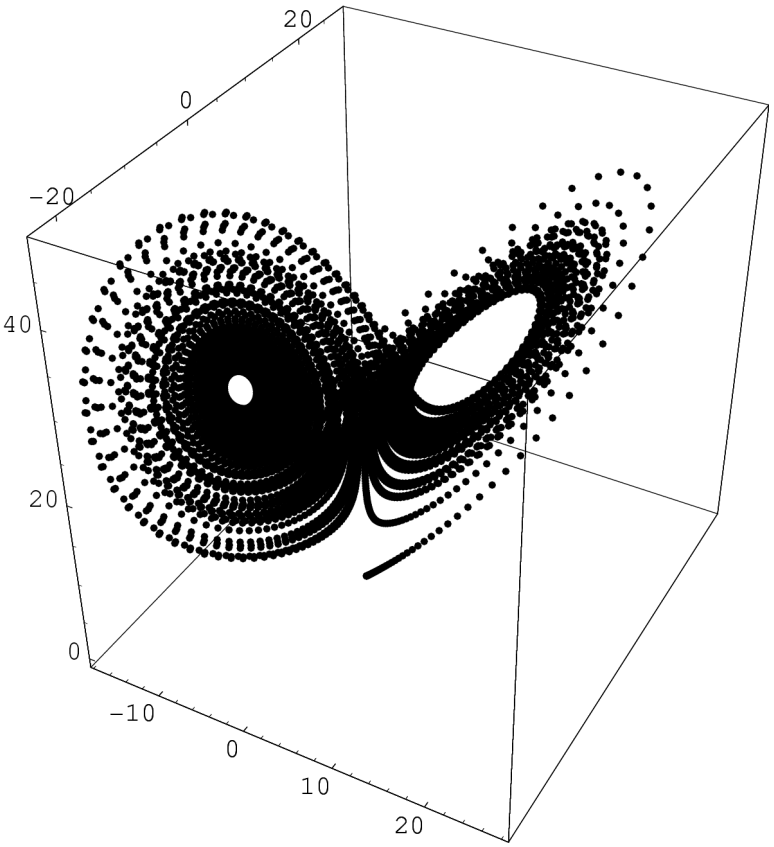
- zbog velike duljine niza 6000 promjenjeni su parametri u podprogramu
- maksimalna duljina niza 200 promjenjena je na 6000, a NMAX 50->3.

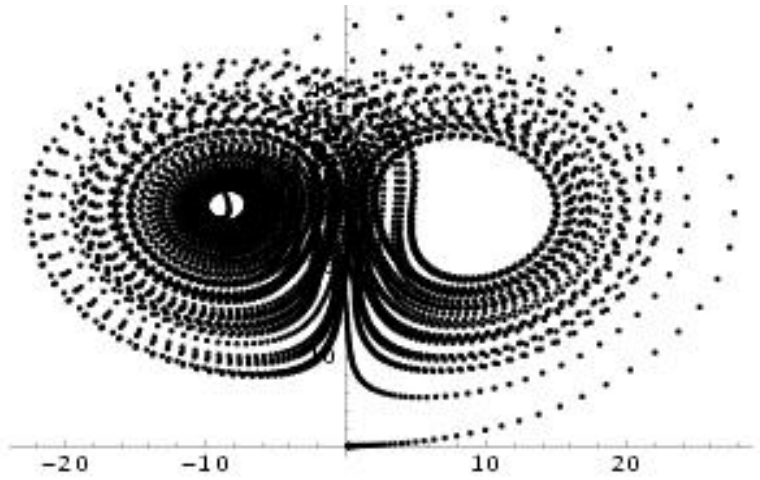
```
SUBROUTINE rkdumb(vstart,nvar,x1,x2,nstep,derivs)
  INTEGER nstep,nvar,NMAX,NSTPMX
  PARAMETER (NMAX=3,NSTPMX=6000)
```

primjer



x-y ravnina





y-z ravnina

primjer

van der Pol jednadžba

$$y_1'' - \mu(1 - y_1^2)\, y_1' + y_1 = 0$$

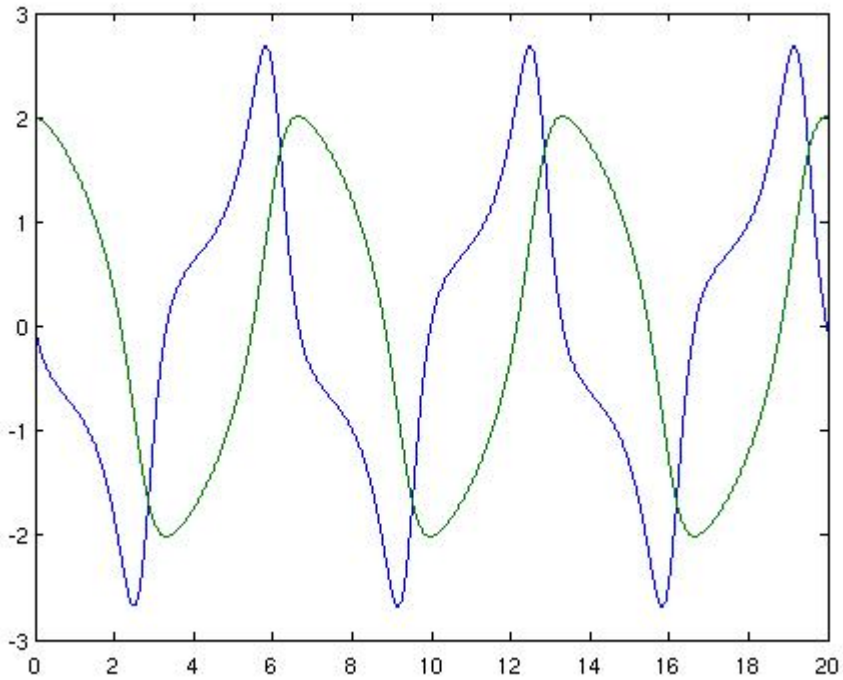
Pretvori u sustav prvog reda

$$y'_1 = y_2$$

$$y'_2 = \mu(1 - y_1^2) y_2 - y_1$$

Početni uvjet y(1)=2, y(2)=0

parametar μ=1



gcc -o Cvdpol vdpol.c rkdumb.c rk4.c nrutils/nrutil.c -I nrutils/

f77 -o Fvdpol vdpol.for rk4.for

RKDUMB

fiksan korak

gcc -o CvdpolOde vdpol1.c odeint.c rkqs.c rkck.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o Fvdpol1 vdpol1.for odeint.for rkqs.for rkck.for

ODEINT varijabilan

Krute (stiff) diferencijalne jednadžbe

Najvažnija upotreba višekoračnih metoda je rješavanje (sistema) krutih diferencijalnih jednadžbi. Najpoznatija metoda poznata je kao Gearova metoda (po Williamu C. Gear-u) i sastoji se od Adams prediktora i korektora varijabilnog reda i varijabilnog koraka.

Za diferencijalnu jednadžbu reći ćemo da je kruta, ako mala perturbacija početnih uvjeta dovede do velike perturbacije u rješenju problema.

Zadana je diferencijalna jednadžba

$$y' = 10(y - x) - 9, \quad y(0) = 1.$$

Opće rješenje ove diferencijalne jednadžbe je

$$y(x) = ce^{10x} + x + 1.$$

Partikularno rješenje za ovaj početni uvjet je $y = x + 1.$

Krute (stiff) diferencijalne jednadžbe

Ako malo perturbiramo početni uvjet na $y(0) = 1 + \varepsilon$, *onda je partikularno rješenje te jednadžbe*

$$y = \varepsilon e^{10x} + x + 1.$$

2. Primjer
van der Pol jednadžba

Početni uvjeti y(1)=2, y(2)=0
parametar μ=1000
granice intervala t=[0,3000]

gcc -g -o Cvdpol1stiff vdpol1stiff.c odeint1.c rkqs.c rkck.c nrutils/nrutil.c -I nrutils/ -lm
promjenjena je vrijednost parametra MAXSTP 10000 na 1500000, file odeint1.c

Zadatak za praktikum

Neka je θ(t) kut njihala s vertikalom u vremenu t kao na slici. Po;etni uvjeti su θ(0)= i θ'(0)=0. Pozicija njihala je određena je diferencijalnom jednadžbom

$$\theta''(t) - \frac{g}{l}\sin(\theta(t)) = 0,$$

gdje je g=9.8 *m/sec*^{2} akceleracija zbog gravitacije i *l*=0.5 *m* je duljina njihala.

1. Napiši problem kao sustav diferencijalnih jednadžbi prvog reda
2. Napiši funkciju derivs koja predstavlja diferencijalnu jednadžbu njihala
3. Riješi problem u vremenu t=0 do 5 za 400 koraka. Koristi podprograme rkdumb ili odeint. Procijeni **period** njihala.
4. Pošaljite rezultat i source na mail Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- George Em Karniadakis and Robert M. Kirby II: Parallel Scientific Computing in C++ and MPI, Cambridge University Press.