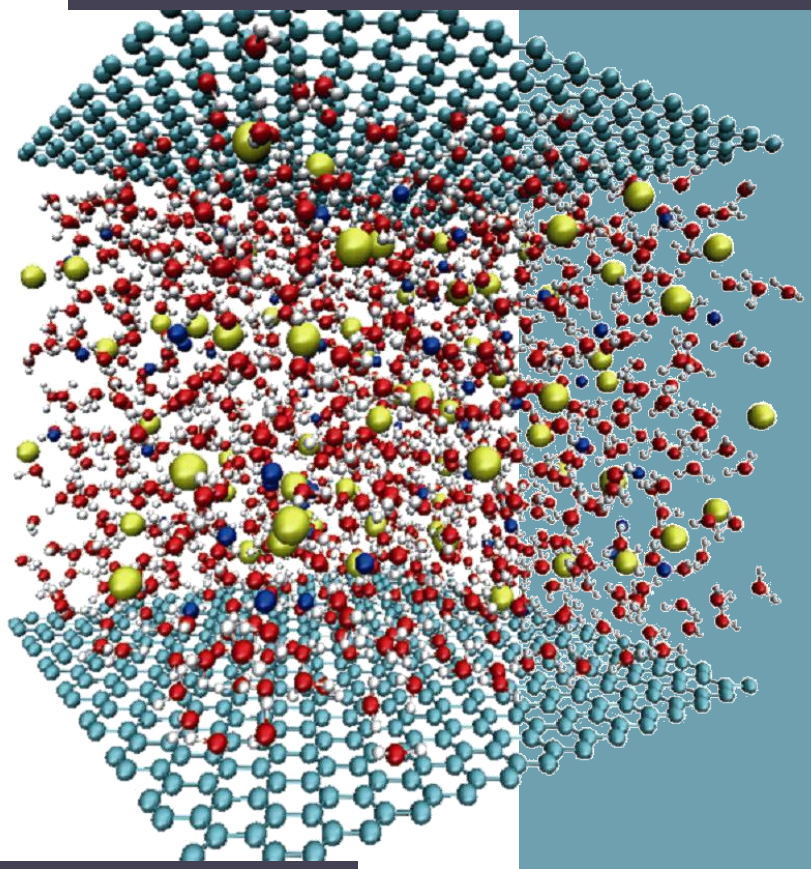
Природно-математички факултет
Институт за физика

Вовед во Комплутерска Физика



Наце Стојанов
Благоја Вељаноски

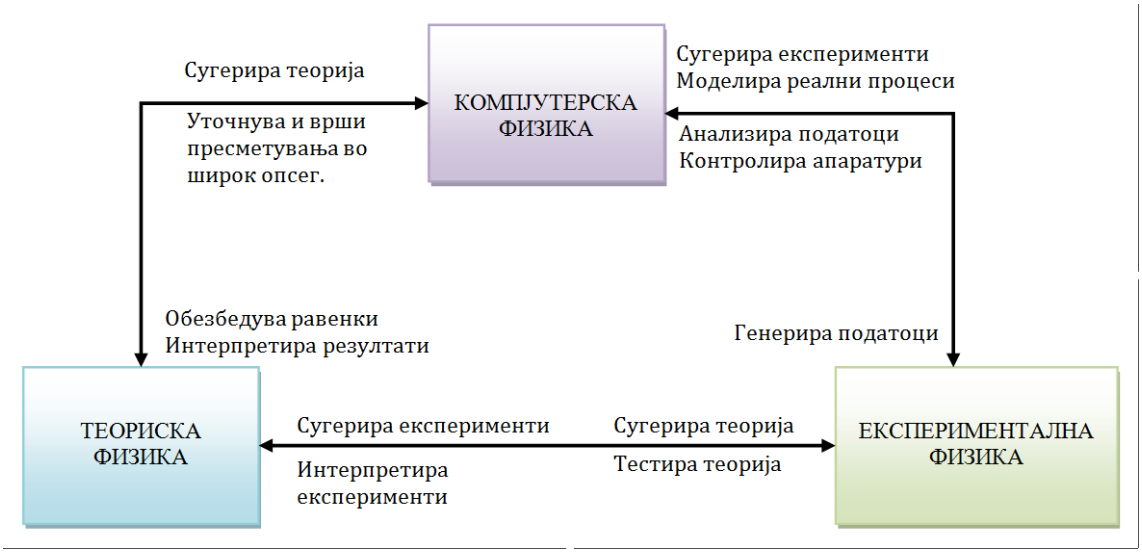
Скопје, 2013

СОДРЖИНА

1. ВОВЕД

Очигледно е дека компјутерите имаат огромно влијание врз развојот на нашето општество а со тоа го промениле и начинот на кој размислуваме за некои работи меѓу кои се и науката и образованието. Кога е во прашање физиката, развојот на компјутерската техника предизвикал појва на нова научна дисциплина - компјутерска физика. Вообичаено, териската физика се занимава со развој на теории со кои се моделира физичката реалност додека експерименталната физика ја проверува валидноста на тие теории во лабораториски услови. Улогата на компјутерската физика е да направи посебен „микс“ на традиционалните гранки на физиката и да понуди нешто ново а тоа е нумерички експеримент или компјутерска симулација.

Како што теориската физика побарува солидно познавање на математичката анализа а експерименталната физика „добро снаоѓање“ во лабораториски услови, компјутерската физика е синтеза на: физика, нумеричка анализа и компјутерско програмирање. Како и да е, овие три гранки (приоди) на физиката се во длабока корелација која графички може да се престави како на слика 1.1.



Сл. 1.1. Корелација помеѓу гранките на физиката.

Во книгите по компјутерска физика вообичаено се обработуваат голем број на методи што се користат за приближно (нумеричко) решавање на аналитички нерешливи проблеми во физиката или сродните научни дисциплини. Методите за приближно пресметување може да се поделат во две главни групи:

- **Директни методи** се засновани на алгоритми кои имаат делумна или целосна примена на аналитички методи, како на пример Гаусовата елиминација за разрешување на системи од линеарни равенки;

- **Итеративни методи** се засновани на алгоритми со сукцесивни повторувања или итерации на дадени постапки.

Бидејќи зборуваме за методи за приближно решавање, може да се очекува дека при нивната практична инплементација ќе наидеме на разни фактори што влијаат врз точноста но и брзината на конвергенција кон решението. Во текстот што следи ќе спомнеме некои од факторите како и нивното влијание во компјутерските симулации.

1.1. Итерација и рекурзија

Поимот *итерација* е еден од најчесто користените поими во компјутерската физика со кој се опишува сукцесивното повторување на дадена постапка со која се подобрува точноста на некој резултат, на пример повторувањето на изразот

$$x_{i+1} = f(x_i). \tag{1.1}$$

За секоја вредност на $i = 1, 2 \ldots \mathrm{n}$, всушност имаме една итерација во постапката на подобрување или конвергенција на изразот, додека самиот израз во целина е познат како *рекурзивна* или *рекурентна формула*. Овие формули се најчесто застапени во методите за приближно решавање, но треба да спомнеме дека тие не секогаш конвергираат кон некое решение, па во тие случаи треба да се модифицираат или во крајна мера да не се применуваат.

1.2. Концепти во нумеричката анализа

Во нумеричката анализа најчесто имаме работа со два вида на броеви: *цели или integer* броеви како што се 0, 1, 2... итн., и *реални или real* броеви како што е $\pi = 3.14159$. При користење на овие броеви треба да се имаат предвид некои концепти кои што суштински влијаат врз методите за приближно решавање а тоа се:

- Значајни цифри,
- Нумерички грешки,
- Представување на броеви.

1.2.1. Значајни цифри

Како значајни цифри на даден приближен број вообичаено се земаат сите цифри што се различни од нула, на пример, бројот 23 има две значајни цифри додека бројот 56,942 има пет значајни цифри. Но, ситуацијата се усложнува кога предвид се зема и нулата. Така, бројот 0,0034 има само две значајни цифри додека бројот 7,910 има четири значајни цифри. За разлика од нив, бројот 5200 може да има две три или четири значајни цифри што зависи од тоа дали има и каде се наоѓа децималната запирка.

Во аритметичките операции, вообочаено, бројот на значајни цифри кај броевите не е секогаш ист, затоа по извршената операција бројот на значајни цифри во финалниот

број ќе зависи од тоа дали оперираме со бездимензионални броеви или пак станува збор за исти или различни физички величини.

Ако работиме со бездимензионални броеви и треба да ја извршиме операцијата $z = x - y$ при што $y = 49$ а $x = 23.058$, ќе добиеме $z \approx 26$ односно финалниот број ќе има само две значни цифри, односно, онолку значајни цифри колку што има бројот со најмалку значајни цифри ($y = 49$).

Кога треба да извршиме артметичка операција со бројните вредности на две исти физички величини, на пример ако $z = x\,y$, каде $x = 0.0045\,\mathrm{kg}^{-1}$ и $y = 34.89\,\mathrm{kg}$, конечно ќе добиеме $z \approx 0{,}16$ што како број има само две значајни цифри. Во случаите кога треба да извршиме аритметичка операција со бројните вредности на две различни величини, на пример $A = f \cdot r$, каде $f = 201.804\,\mathrm{N}$ а $r = 0.67\,\mathrm{m}$, тогаш конечната величина $A \approx 135.21\,\mathrm{J}$ ќе има само две децимални места, значи наместо значи цифри имаме број на децимални места.

1.2.2. Нумерички грешки

При нумеричките пресметки може да се појавуваат разни видови на грешки пришто на некои може да влијаеме а на други не. Затоа, треба да се биде внимателен се со цел имање целосна контрола врз акумацијата на грешките во конечниот резултат посебно на оние што се резултат на несовршеноста на методите за приближно пресметување.

Можните извори на грешки при нумеричките пресметувања може да се поделата во неколку групи:

- *Грешки во влезните податоци*. Тие настануваат при мерењата на физичките величини и определување на нивните бројни вредности. Познато е дека при мерењата се прават субјективни и објектвни грешки кои што лесно може да се лоцираат а со тоа и да се контролира нивното влијание.
- *Грешки на заокружување*. Овие грешки се случуваат при компјутерските пресметувања затоа што тие се засновани на бинарна алгебра со подвижна точка според која на секоја операција и се доделени конечен број на бинарни места. На пример, имаме компјутерот кој што може да пресметува со максимални n_{max} значајни цифри. Да претпоставиме дека во некоја програма треба да се помножат два броја од кои секој има по n_{max} цифри, тогаш производот на двата броја ќе биде прикажан само со максималните n_{max} значајни цифри, што значи неточно. Истотака, ако на некој број *x* треба да му додадиме многу мал број *y* тогаш во собирокот $z = x + y$, заради конечниот број на значајни цифри, може да се случи некои од цифрите на малиот број *y* да не бидат земени предвид што повторно доведува до нумеричка грешка на заокружување. Овие грешки се познати како акумулациони така да посебно доаѓаат до израз при долготрајните пресметувања.
- *Грешки на отфрлање*. Овие грешки се јавуваат кога бесконечниот ред со кој се прикажува некоја функција ќе се „заокружи“ или апроксимира со конечен број членови. Истотака, овие грешки се појавуваат и во случаите кога за при-

ближните пресметувања се користат конечни разлики. Таков е случајот со изводот на функциите и неговата апроксимација со количник од конечни разлики, $f'(x) \approx \Delta f / \Delta x$.

- *Грешки на моделот*. Овој вид на грешки се речиси неизбежни затоа што многу често нумеричките модели се „идеализација“ или упростување на реалните физички проблеми. Тоа е неопходно за да се дојде до некои почетно решение а потоа истото ќе се уточнува со подобрување на нумеричкиот модел. Грешките на моделот директно зависат од степенот на познавање на проблематиката што поединецот сака да ја реши со приближни методи.

- *Субјективни грешки и компјутерски грешки*. При нумеричките пресметувања лесно може да се случи погрешно да се внесат некои податоци или да се направи и грешка во програмирањето на некој алгоритам. Заради човечка грешка, погрешно напишаниот DO цуклус во FORTRAN програма довел до загуба на вселенското летало Venus. Имено, наместо DO 1,3 било напишано е DO 1.3. Ако зборуваме за хардверските или компјутерски грешки, познат е примерот од 1994 година кога конструкторска грешка во процесорот INTEL Pentium довел до погрешни резултати при делење на определи групи на броеви. Затоа, со цел намалување на влијанието на овие грешки добро позната препорака е повеќекратно проверување на излезните резултати и ако е можно нивно споредување со некои слични пресметувања.

Имајќи ги предвид можните извори на грешки, како и апроксимациите што се користат за моделирање во компјутерската физика, заради оцена и контрола на грешките се воведини поимите за апсолутна и релативна грешка. Апсолутната грешка се дефинира со изразот

$$\Delta x = \left| \bar{x} - x \right|, \tag{1.2}$$

каде што $\bar{x}$ е средна вредност или точна вредност на некоја величина, а x е приближна вредност на истата величина. Релативната грешка се дефинира како

$$\varepsilon = \left| \frac{\Delta x}{\bar{x}} \right| = \left| \frac{\bar{x} - x}{\bar{x}} \right|. \tag{1.3}$$

Многу е важно да се има предвид дека секоја од овие грешки не е погодна за оценка на грешките при нумеричките пресметувања. Еве еден пример за тоа. Да претпоставиме дека во некоја компјутерска програма величината x треба да биде пресмета со апсолутна грешка $\Delta x = 0.0001$ и е добиена точна вредност $\bar{x} = 53,0020$ со пет значајни цифри. Приближната вредност до која е дојдено во оваа програма изнесува $x = 53,0020 \pm 0.0001$ и има, истотака, пет значни цифри. Но во случај ако $\bar{x} = 0.0001$, тогаш приближната вредност е $x = 0.0001 \pm 0.0001$ и нема ниту една значајна цифра. Ова ни кажува дека во овој случај апсолутната грешка не е добра за оценка на точноста.

Од друга страна, примената на релативната грешка не доведува до следниов заклучок. Нека релативната грешка при пресметувањето биде $\varepsilon = 0.0001$, а точната вредност $\bar{x} = 53,0020$. Во тој случај апсолутната грешка според (1.3) изнесува $\varepsilon \bar{x} = 0,0053$

што значи дека приближната вредност сигурно ќе има пет значни цифри. Истотака, ако релативната грешка остане иста а $\bar{x} = 0.0001$, тогаш апсолутната грешка изнесува $\varepsilon \bar{x} = 10^{-8}$, што значи дека приближната вредност повторно ќе има пет значни цифри. Од овде може да заклучиме дека при нумеричките пресметувања релативната грешка обезбедува приближната вредност да има ист број на значни цифри како и точната вредност.

Во контекст на погоре кажаното, со поимот *точност* се прикажува грешката (апсолутна или релативна) со која се апроксимира некоја величина, а *прицизност* е точноста со која се извршуваат математичките операции во дадена програма или нумерички метод.

1.2.3. Представување на броеви и компјутерска аритметика

Аритмртичките операции кои се изведуваат со компјутерите се разликуваат од оние што ние ги изведуваме секојдневно затоа што не се изведуваат во исти бројни системи. Кога треба да направиме некоја пресметка вообичаено се користиме со декадниот систем на броеви, додека компјутерите го користат бинарниот систем во кој има само два броја 0 и 1. Во тој контекст, може да се кажи дека луѓето и компјутерите на операцијата $5-2=3$ гледаат речиси слично, но не и исто, додека при коренувањето $(\sqrt{2})^2 = 2$ постои целосна разлика. Имено, во математичката анализа $\sqrt{2}$ е единствен позитивен реален број кој кога ќе се помножи со самиот себе ни го дава бројот 2. Но, во компјутерската аритметика тоа не е така. Секој број се претставува со конечен број на битови или бинарни места, така да при компјутерското множење на бројот $\sqrt{2}$ со самиот себе сигурно не се добива точно бројот 2, туку приближно со некоја точност.

Оваа неточност е резултат на грешките на заокружување или round-off errors што се случуваат во компјутерите затоа што реалните броеви се прикажуваат со конечен број на децимални места и подвижна точка (floating point) која произлегува од научниот формат на прикажување на броевите.

Според оваа нотација, бројот $x = -31.41$ во декаден систем со подвижна точка може да се прикажи како $-3.141 \cdot 10^1$ при што мантисата на бројот е $m = 3.141$а карактеристиката или експонентот е $c = 1$, базата на приказот е $b = 10$ а знакот на бројот е (-). Или накратко

$$x = \pm m \cdot 10^c . \tag{1.4}$$

Во бинарен систем со 32 битна презентација на броевите или single precison, бројот x се прикажува како

$$x = 1\ 1000001\ 11111011\ 01000111\ 10101110, \tag{1.5}$$

каде знакот на бројот (-) се означува со 1, мантисата на бројот е

$$m = 11111011\ 01000111\ 10101110, \tag{1.6}$$

а степенот е

$$c = 1000001. \tag{1.7}$$

Било кој број, но во база $b = 2$, може да се прикаже со формулата

$$x = (-1)^s \, m \, 2^c , \tag{1.8}$$

каде $s = 0$ или 1, m е мантиса а c е експонент. Бројот $x = 31.41$ во бинарен систем може да се прикажи како $x = (-1)^0 \times 1.963125 \times 2^4 = 1 \times 1.963125 \times 16 = 31.41$.

Од 32 бинарни места при single precison презентација на реален број со подвижна точка, едно бинарно место или 1 бит се доделува на знакот, седум бита се доделуваат на степенот а 24 бита се резервирани за мантисата со што се обезбедуваат 6-7 точни децимални места. Најмалиот број кој може да се пресмета со 32 битната презентација е $\approx$1.0E-44 а најголемиот е ≈ 1.0 E+38. Доколку сакаме да ги опфатиме и поголеми и помали броеви од наведените, неопходно е да примениме 64 битна презентација на броевите или double precision. Во тој случај, едно бинарно место или 1 бит се доделува на знакот, 11 бита се доделуваат на степенот а 52 бита се за мантисата со што се обезбедуваат 15-16 точни децимални места. Најмалиот број кој може да се пресмета со 64 битната презентација е $\approx$1.0E-322 а најголемиот е $\approx$1.0E+308.

Кога нумеричките пресметки го надминуваат интервалот на прецизност со кои се декларирани променливите можни се два случаи. Ако пресметуваниот број е помал од најмалата можна вредност компајлерот најчесто му доделува вредност нула и ни јавува грешка *underflow*, додека ако пресметуваната вредност е поголема од најголемата дозволена вредност се јавува грешка *overflow* и пресметувањето се стопира.

1.3. Програмски јазик

Изборот на програмскиот јазик за нумеричките методи во компјутерската физика може да произлези од одговорот на овие две прашања: Кој е најдобар програмски јазик и кој е најсоодветен програмски јазик за компјутерската физика? Доколку на ова прашање одговори професионален програмер тогаш одогворот би бил дека се зависи од тоа што треба да се направи во дадена симулација, што значи дека може/треба да се употребат и повеќе програмски јазици. Но, физичарите најчесто не се професионални програмери, па на изборот на програмскиот јазик гледаат чисто прагматично, во стилот да биде брзо и едноставно. Ако го имаме ова во предвид, без никаква „носталгија“ за можностите, речиси единствен избор за секој физичар што сака сериозно да се занимава со компјутерската физика е програмскиот јазик фортран. Всушност и неговото оригинално име FORTRAN е акроним од FORmula TRANsalation што во слободен превод значи преведувч на формули.

При изборот на програмскиот јазик треба да се има предвид дека и денес, најголемиот дел од нумеричките библиотеки што се користат во компјутерската физика, но и во сродните области како што се хемијата и инженерството, се напишани во некаква верзија на фортран. Покрај позитивните страни, од програмерска гледна точка, чесно е да спомени дека фортранот има и негативности како што е честата употреба на GOTO скоковите но и сложените процедури за графички приказ на податоците. Како и да е, ако земиме се во предвид, слободно може да кажиме дека фортранот е јазик на природните науки!

Во последната декада, на пазарот за програмски софтвер се појавија многу CAS (Computer Algebra System) производи како што се Mathematica и MATLAB. Тие се одли-

куваат со голема флексибилност во програмирањето што значи дека овозможуваат и т.н. *симболичко програмирање* што е извонредно ефикасно и неверојатно големи можности за професионално но „лесно“ графичко прикажување на датотеките. Заради тоа, се почесто, во компјутерската физика се прави и природен „микс“ на можностите на фортранот со овие апликативни програмски пакети.

Пример 1.1

Во програмата dvojna_preciznost.for се покажува разликата во прецизноста на нумречките пресметки кога променливата F е декларирана со обична прицизност (single precision) а Y со двојна прецизност (double precision).

```
PROGRAM DVOJNA_PRECIZNOST
IMPLICIT REAL*8(A-H,P-Z)
CHARACTER NAME1*30,AA*1

PRINT *,'Program: DVOJNA_PRECIZNOST '
PRINT *,'Se prikazuva razlikata pomegju realnite '
PRINT *,'promenlivi I onie so dvojna preciznost. '
PRINT *,'F-realna,Y-so dvojna preciznost '

WRITE(*,*)' '
WRITE(*,*)'Odberete vid na izlez: '
WRITE(*,*)'1.Ekran '
WRITE(*,*)'2.Text file '
WRITE(*,*)'Vnesi 1 ili 2 '
READ(*,*)FLAG
WRITE(*,*)' '

IF ( FLAG .EQ. 2 ) THEN
   WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid- '
   WRITE(*,*) 'drive:name.ext'
   WRITE(*,*) 'patot da bide staven vo navodnici'
   WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
   READ (*,*) NAME1
   WRITE(*,*) ' '
   OUP = 3
   OPEN(UNIT=OUP,FILE=NAME1)
ELSE
   OUP = 6
END IF
F=1.
Y=1.D0

DO I=1,1000000
   F=F+0.000001
   Y=Y+0.000001D0
END DO
WRITE(OUP,*)'STO E POBLISKU DO BROJOT 2?'
WRITE(OUP,*)'F=',F
WRITE(OUP,*)'Y=',Y

CLOSE(UNIT=5)
CLOSE(UNIT=OUP)
IF(OUP.NE.OUP) CLOSE(UNIT=6)
STOP
END
```

Излезот на програмата е:

```
STO E POBLISKU DO BROJOT 2?
F = 1.953674
Y = 1.99999999991773
```

1.4. Задачи

1. Напиши програма во фортран со која ќе се покажи како влијае машинската прецизност врз збирот на два броја дефинирани со рекурзивната релација $y_i = 1 + x_i/2$, во случаите кога:

а) Ако броевите се дефинирани со обична прецизност тогаш бројот на итерации да биде 150,

б) Ако броевите се дефинирани со двојна прецизност тогаш бројот на итерации да биде 330?

Почетните врдности се $y_0 = x_0 = 1$.

2. Напиши програма во фортран со која ќе се определат границите на overflow и underflow за рекурзивните релации $x_{i+1} = x_i/2$ и $y_{i+1} = 2 \cdot y_i$ во случаите кога:

а) Ако променливите се дефинирани со обична прецизност,

б) Ако променливите се дефинирани со двојна прецизност?

Почетните врдности се $y_0 = x_0 = 1$ а бројот на итерации е $N = 1024$.

3. Напиши програма во фортран со која ќе се „табелира“ функцијата

$$r(\theta) = e^{\cos(\theta)} - 2\cos(4\theta) + \sin^5(\theta/12),$$

за интервалот $0 \le \theta \le 24\pi$. Датотеката да се прикажи графички со програмата GnuPlot или Mathematica. Потоа, со користење на трансформацијата

$$x = r\cos\left(\theta + \frac{\pi}{2}\right) \qquad x = r\sin\left(\theta + \frac{\pi}{2}\right),$$

да се генерира нова датотека и истата повторно да се нацрта? Каква форма има заротираната функција?

2. ПРИБЛИЖНО РЕШАВАЊЕ РАВЕНКИ

Многу често при пресметување имаме работа со доста комплицирани алгебарски или трансцедентни равенки, како на пример равенката за зонска структура на енергетскиот спектар на трдите тела, за кои не постојат аналитички начини за наоѓање точни решенија, а и ако постојат тие се практично непогодни за примена. Исто така, во некои случаи равенките содржат коефициенти што се познати само приближно, па затоа има смисла да се бараат само нивни приближни решенија. За многу практични цели доволно е да се знаат само приближните решенија односно корени на равенките со однапред зададена точност.

Да претпоставиме дека имаме равенка од видот

$$f(x)=0, \tag{2.1}$$

каде функцијата $f(x)$ е дефинирана и континуирана во некој интервал $a<x<b$. Секоја вредност ξ за која функцијата $f(x)$ е нула, или

$$f(\xi)=0,$$

се вика **корен** или **решение** на равенката (2.1) или нула на функцијата $f(x)$.

Задачата за изнаоѓање на приближно решение на равенката (2.1) обично содржи два чекора:

- Изолирање или локализирање на корените на функцијата со што треба да се најдат доволни услови за (2.1) да има решение и потоа да се одредат најмалите интервали што содржат само еден корен на функцијата;
- Откако е најдено некое приближно решение на равенката (2.1) да се изврши подобрување или уточнување со однапред зададена точност.

Постапките за приближно решавање равенки најчесто се некои итеративни методи. Постапката се состои во тоа што некој процес се повторува или „се итерира“ (на латински iteratio значи повторување). Имено, ако е утврдено дека постои корен ξ на равенката $f(x)=0$, и ако е извршена негова изолација, тогаш се почнува со некое приближување (апроксимација) x_0 како „влезна“ информација, па се применува некое правило кое ќе произведе ново приближување x_1 кон ξ. Понатаму, добиената апроксимација x_1 се користи како влезна за добивање на ново приближување x_2 кон ξ користејќи го истото правило, итн. На тој начин се добива низа од приближувања $x_1, x_2, \ldots, x_n$ чија граница конвергира кон коренот ξ на равенката (2.1).

Процесот на приближување или апроксимација на коренот завршува кога две последователни апроксимации се „доволно блиски“ за практично да можеме да ги сметаме за еднакви. Притоа, последното приближување x_n кон ξ го земаме за корен на равенката (2.1).

2.1. Метод на преполовување (бисекција)

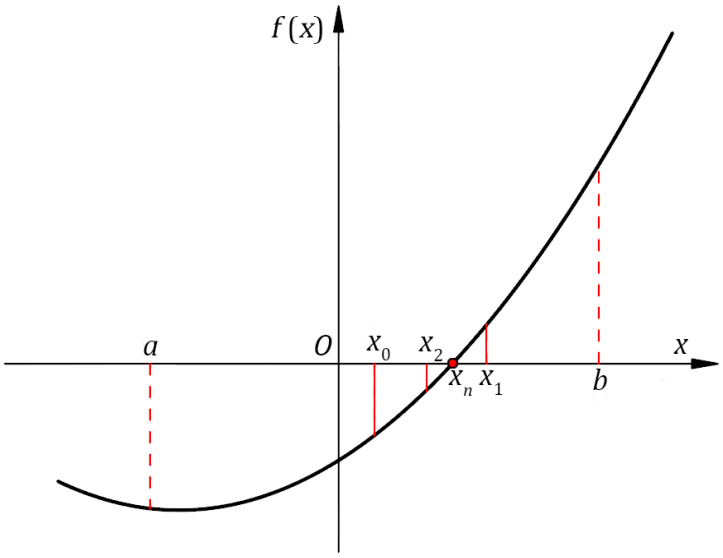
Методот на преполовување е наједноставен и погоден за решавање на сите видови алгебарски равенки. Тој е сигурен но доста бавен метод, односно има бавна конвергенција кон точното решение на равенката. Да земеме една непрекината функција $f(x)$ која има корен x_n во интервалот $x \in [a,b]$, слика 1. Во тој случај, сигурно важи релацијата

$$f(a) \cdot f(b) < 0 . \tag{2.2}$$

Целта на овој метод е да се намали интервалот во која се наоѓа решението на равенката, со тоа што во секоја наредна итерација се проверува дали важи

$$f(a_n) f(b_n) < 0 . \tag{2.3}$$

Постапката завршува после n итерации кога е задоволено $|a_n - b_n| < \varepsilon$, при што ε е бараната прецизност на пресметката.



Сл. 1. Графички приказ на методот на преполовување или бисекција. Решението на равенката се наоѓа во точката x_n.

Програма 2.1:

Во програмата prepolovuvanje.for, дадена подолу во текстот, се бара решение на равенката $f(x) = \cos(x) - x = 0$. Решението $x_n = 0.73908$ се наоѓа со точност $\mathrm{TOL} = 10^{-6}$ после $n = 22$ итерации.

```
PROGRAM PREPOLOVUVANJE
IMPLICIT REAL*8 (A-H,P-Z)
EXTERNAL FX
CHARACTER NAME1*30,AA*1

PRINT*, 'Program: PREPOLOVUVANJE'
PRINT*, 'Koren na funkcija od edna promenliva f(x)=0'
PRINT*, 'so metod na prepolovuvanje'
WRITE(*,*) ' '
WRITE(*,*) 'Odberete vid na izlez: '
WRITE(*,*) '1. Ekran '
```

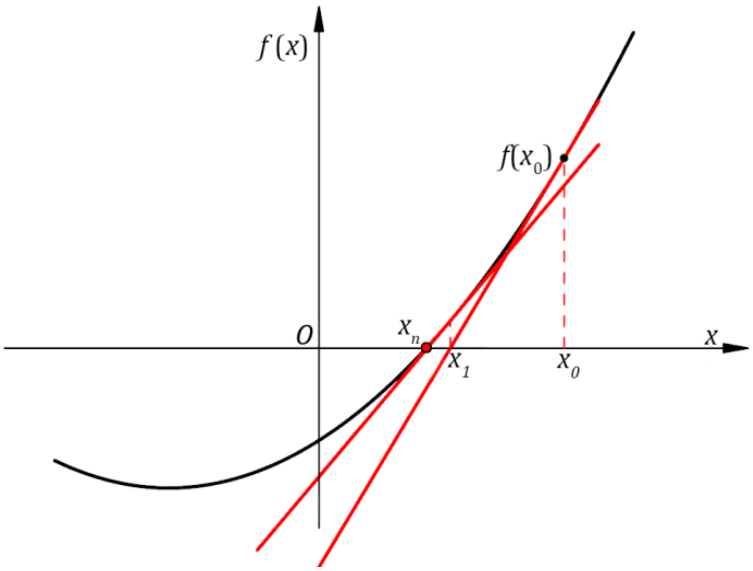
```
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ (*,*) FLAG
      WRITE(*,*) ' '
      IF ( FLAG .EQ. 2 ) THEN
         WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
         WRITE(*,*) 'drive:name.ext '
         WRITE(*,*) 'patot da bide staven vo navodnici '
         WRITE(*,*) 'na primer:  "C:/DATA.TXT" '
         READ (*,*) NAME1
         WRITE(*,*) ' '
         OUP = 3
         OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
         OUP = 6
      END IF

      XL=0.D0
      XR=3.D0
      ITER=1
      CALL BISECT(XL,XR,ITER,XMID,FX)
      WRITE(OUP,*)'KOREN E NAJDEN VO X=',XMID
      WRITE(OUP,'(A,I4,A)')'POSLE:',ITER,'ITERACII
      CLOSE(UNIT=5)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END

      REAL*8 FUNCTION FX(X)
C     DEFINIRANJE NA FUNKCIJATA CII KORENI SE BARAAT
C     F(X)=COS(X)-X
      REAL*8 X
      FX=COS(X)-X
      RETURN
      END
      SUBROUTINE BISECT(XL,XR,IT,XMID,F)
C     PROCEDURA ZA NAOGJANJE KOREN NA FUNKCIJA SO EDNA
C     PROMENLIVA SO METOD NA PREPOLOVUVANJE
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(TOL=1.D-06)
      FL=F(XL)
      FR=F(XR)
  100 XMID=(XL+XR)/2.D0
      FMID=F(XMID)
      IF((FL*FMID).GT.0.D0)THEN
         XL=XMID
         FL=FMID
      ELSE
         XR=XMID
         FR=FMID
      END IF
      ERROR=DABS((XR-XL)/XMID)
      IF(ERROR.GT.TOL)THEN
         IT=IT+1
         GOTO 100
      END IF
      RETURN
      END
```

Излезот на програмата е:

```
KOREN E NAJDEN VO X=  0.739084482192993
POSLE:  22 ITERACII
```

2.2. Метод на Њутн-Рафсон

Ова е еден од поточните методи за наоѓање корен на функција кој е познат и како метод на тангенти. За да видиме како функционира овој метод да претпоставиме дека имаме континуирана функција $f(x)$ која има корен интервалот $[a,b]$, слика 2.



Сл. 2. Графички приказ на методот на Методот на Њутн-Рафсон. Решението на равенката се наоѓа во точката x_n.

Конвергенцијата кон решението според методот на Њутн-Рафсон се изведува според рекурентната формула

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{2.4}$$

која побарува да го знаеме и изводот на функцијата $f'(x_n)$, што во дадени ситуации ја прави неприменлива. Ова приближување кон коренот геометриски се прикажува со пресечните точки на тангентите со оската Ox. Неприменливоста на методот на Њутн-Рафсон е во случаите кога $f'(x) \cong 0$, што значи дека тангентата е речиси паралелна со x оската, при што рекурентната формула (2.4) не оддалечува од точното решение наместо да конвергира кон него. Од оваа ситуација нема излез, затоа се препорачува претходно функцијата да се нацрта и поточно да се лоцира почетното решени се со цел формулата (2.4) да не нè доведе во областа каде $f'(x) \cong 0$. Доколку оваа незгодна ситуација се избегни, тогаш методот на Њутн-Рафсон е неверојатно брз и за неколку итерации не доведува до точното решение.

***Програма* 2.2:**

Програмата njutn_rafson.for ни овозможува наоѓање решение на равенката $f(x) = \cos(x) - x = 0$ со точност $\mathrm{TOL} = 10^{-6}$. Само после $n = 4$ итерации се доаѓа до решението $x_n = 0.739085$.

```
PROGRAM NJUTN_RAFSON
```

```
      IMPLICIT REAL*8 (A-H,P-Z)
      EXTERNAL FX,DFX
      CHARACTER NAME1*30,AA*1
      PRINT *,'Program: NJUTN_RAFSON '
      PRINT *,'Koren na funkcija od edna promenliva f(x)=0 '
      PRINT *,'so metod na Njutn-Rafson '
      WRITE(*,*)' '
      WRITE(*,*)'Odberete vid na izlez: '
      WRITE(*,*)'1.Ekran '
      WRITE(*,*)'2.Text file '
      WRITE(*,*)'Vnesi 1 ili 2 '
      WRITE(*,*)FLAG
      WRITE(*,*)' '
      IF ( FLAG .EQ. 2 ) THEN
         WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid- '
         WRITE(*,*) 'drive:name.ext'
         WRITE(*,*) 'patot da bide staven vo navodnici'
         WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
         READ (*,*) NAME1
         WRITE(*,*) ' '
         OUP = 3
         OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
         OUP = 6
      END IF
      X=1.D0
      ITER=1
      CALL NEWTON(X,ITER,FX,DFX)
      WRITE(OUP,*)'KOREN E NAJDEN VO X=,X
      WRITE(OUP,'(A,I4,A)')' POSLE:',ITER,'ITERACII'

      CLOSE(UNIT=5)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END
      REAL*8 FUNCTION FX(X)
C     DEFINIRANJE NA FUNKCIJATA CII KORENI SE BARAAT
C     F(X)=COS(X)-X
      REAL*8 X
      FX=X*COS(X)-2*LOG(X)
      RETURN
      END
      REAL*8 FUNCTION DFX(X)
C     IZVODOT NA FUNKCIJATA F(X) IZNESUVA
C     F'(X)=-SIN(X)-1
      REAL*8 X
      DFX=COS(X)-X*SIN(X)-2/X
      RETURN
      END

      SUBROUTINE NEWTON(X,IT,F,FP)
C     PROCEDURA ZA NAOGANJE KOREN NA FUNKCIJA SO EDNA
C     PROMENLIVA SO METOD NA NEWTON-RAPHSON
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(TOL=1.D-06)
  100 DELTA=-F(X)/DFX(X)
      X=X+DELTA
      ERROR=DABS(DELTA/X)
      IF(ERROR.GT.TOL)THEN
         IT=IT+1
         GOTO 100
      END IF
      RETURN
      END
```

Излезот на програмата е:

```
KOREN E NAJDEN VO X=  0.739085133215161
POSLE:   4 ITERACII
```

2.3. Метод на тетиви

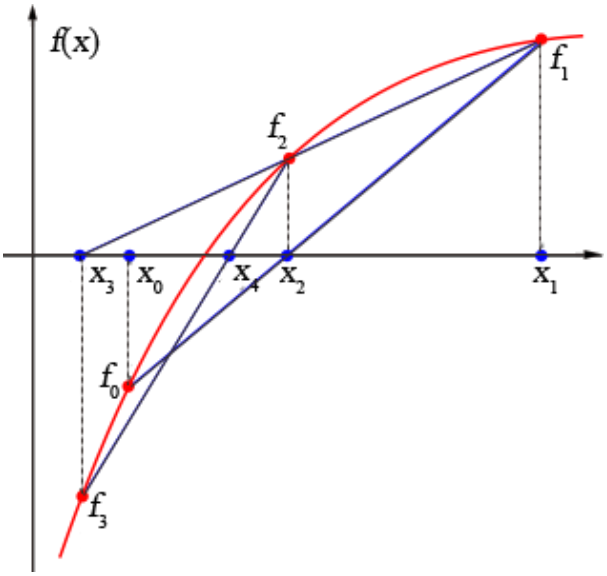
Методот на Њутн-Рафсон е многу ефикасен, но има слабост во тоа што пресметувањето на изводот $f'(x)$, во принцип, побарува поголем број пресметувања отколку на самата функција $f(x)$. Истотака, понекогаш, не е можно да се знае изводот на функцијата, што го прави практично неупотреблив. Со цел надминување на овој проблем, предложен е нов метод за инзаоѓање корен на функција со тоа што аналитичкиот извод се заменува со конечни разлики од прв ред, или

$$f'(x_n) = \frac{f(x) - f(x_n)}{x - x_n}, \tag{2.5}$$

со што од (2.4) добиваме

$$x_{n+1} = x_n - f(x_n)\frac{x - x_n}{f(x) - f(x_n)}. \tag{2.6}$$

Гледано геометриски, како е прикажано на слика 2.3, постапката започнува со задавање на две точки, x_0 и x_1, а првата апроксимација x_2 се добива како пресечна точка на тетивата меѓу дочките $(x_0, f(x_0))$ и $(x_1, f(x_1))$. Следната апроксимација на решението е точката x_3 која е апциса на тетивата меѓу точките $(x_1, f(x_1))$ и $(x_2, f(x_2))$ и т.н..



Сл. 3. Графички приказ на методот на Методот на тетиви. Прикажани се првите три итерации со што решението се апроксимира со x_4.

2.5. Задачи

1. Користејќи го методот на бисекција да се лоцира можното решение на равенката $f(x) = x^3 + 4x^2 - 10 = 0$ со точност 10^{-2} а потоа решението да се уточни со методот на тангенти до точност 10^{-8}. Да се користат променливи со двојна прецизност.

2. Да се анализира решението на равенката $f(x)=x-2\sin(x)=0$, добиено со метод на Њутн-Рафсон, кога почетното решение е $x_0=1{,}1$ и $x_0=1.5$?

3. Функцијата

$$f(x)=\ln(x^2+1)-e^{0.4x}\cos(\pi x),$$

има бесконечен број на решенија. Користејќи најсоодветен од погоре спомнатите методи:

а) Да се определи негативен корен на оваа функција со точност 10^{-6}.

б) Да се определат четири најмали но позитивни корени на оваа функција со точност 10^{-6}.

2.6. Проекти

1. Користејќи го методот на теиви да се реши равенката $\cos(x)-x=0$, со точност 10^{-6} и решението да се спореди со методот на Њутн-Рафсон?

2. Еден модел со кој се опишува промената на популацијата во една заедница, кога како константен фактор се зема и степенот на имиграција ν се опишува со равенката

$$N(t)=N_0e^{\lambda t}+\frac{\nu}{\lambda}(e^{\lambda t}-1),$$

каде λ е прирастот на популацијата со новороденчиња. Ако земиме дека за една година $t=1$, од $N_0=1000000$ со имиграција $\nu=435000$, популацијата се зголемила на $N(1)=1564000$, да се определи бројот на новороденчиња λ со точност од 10^{-4}. Да се прошири моделот со тоа што предвид ќе биде земена и емиграцијата на популацијата?

3. МЕТОДИ ЗА РЕШАВАЊЕ ЛИНЕАРНИ СИСТЕМИ

При опишување на многу физички процеси се јавува потреба од линеаризација, односно апроксимирање на нелинеарен систем равенки со систем од линеарни равенки, запишан во матрична форма, кој што релативно поедноставно се решава. Оттука, матричните операции за решавање на системи линеарни равенки при постапките за наоѓање на сопствени вредности или решавање на парцијални диференцијални равенки се многу често застапени. Гледано генерално, методите за решавање на систем линеарни равенки може да се поделат во две групи:

- *Директни или „точни“ методи* се оние што користат аналитички приод за пресметување корени на систем равенки. Такви се на пример, Крамеровото правило, Гаусовиот метод на елиминација или LU факторизацијата.
- *Приближни или итеративни методи* се оние што овозможуваат приближно добивање корени на систем равенки со помош на конвергентни итеративни процеси како што се методот на Гаус-Зајдел или методот на релаксација.

Примената на компјутерите за изведување на матрични операции е многу корисна затоа што алгоритмите за матрични операции најчесто побаруваат повторување на голем број едноставни опеарации што најлесно и најбрзо може да се направат со компјутер. Како што се зголемувала моќта на копмјутерите, така и матриците полесно се применувале за разрешување на физички проблеми со голем број степени на слобода, но со тоа доаѓало до акумулација на грешките на заокружување (round-off errors). Со цел надминување на овој проблем, напишани се голем број на професионални рутини што ги минимизираат овие грешки, па затоа се вели дека кога се во прашање матриците препорачливо е да се користат само „проверени“ рутини а не да се пишуваат сопствени.

3.1. Метод на Гаусова елиминација

Со овој метод се решава систем од *n* линеарни алгебарски равенки со *n* непознати, кој во матрична форма може да се запише како $Ax_i = b_i$, за $i = 1,2,\ldots n$. Матрицата A ги содржи коефициентите a_{ij} на системот и може да се запише како

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \tag{3.1}$$

b_i се слободни членови на системот зададени како вектор колона

$$b_i = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \tag{3.2}$$

а x_i се решенијата на системот, истотака, зададени со вектор колона

$$x_i = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}. \tag{3.3}$$

Основната идеја на методот на Гаусова елиминација е матрицата A со која се претставува системот да се трансформира во горна тригонална матрица од видот

$$A^{u,t} = \begin{bmatrix} a_{11}^{u,t} & a_{12}^{u,t} & a_{13}^{u,t} & \dots & a_{1n}^{u,t} \\ 0 & a_{22}^{u,t} & a_{23}^{u,t} & \ddots & \vdots \\ 0 & \ddots & a_{33}^{u,t} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & a_{n-1,n}^{u,t} \\ 0 & \dots & \dots & 0 & a_{nn}^{u,t} \end{bmatrix}. \tag{3.4}$$

За таа цел, тргнуваме од првата равенка, $a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$, и ја решаваме по променливата x_1 во однос на останатите променливи x_2, x_3, … x_n, а потоа вршиме замена на x_1 во останатите n-1 равенки. Оваа процедура, позната како *елиминација нанапред*, ја повторуваме се додека не дојдиме до последната равенка во која како непозната се јавува само x_n. Од последната равенка директно се определува вредноста на решението x_n како

$$x_n = \frac{b_n}{a_{nn}^u}. \tag{3.5}$$

Потоа, тоа решение се става во претпоследната равенка од каде се определува решението x_{n-1} и така наназад се додека не дојдиме до првата равенка од каде ќе го определиме и решението x_1. Очигледно е дека оваа постапка се нарекува решавање со *замена наназад* а општото решение, за $i = n-1, n-2, \dots, 2, 1$, може да се запише како

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}^u x_j}{a_{ii}^u}. \tag{3.6}$$

При користење на овој метод може да се случи некој од коефициентите на главната дијагонала да е нула или има многу мала вредност. Затоа, неопходно е да се направи замена на редовите како би се избегнало делењето со нула. Постапката на замена е оправдана затоа што се применуваат две својства на матриците кои кажуваат дека при промена на коефициентите не се менуваат решенијата на системот ако:

- Било кој ред на матрицата (или равенка) се помножи со константа, односно се скалира,

➲ Редовите или колоните на матрицата (равенките) ги замената местата.

Замената на редовите или колоните, со цел избегнување на делење со нула или мал број е позната како постапка за „пивотирање“ или барање елемент со најголема „тежина“ или вредност. Во случај на Гаусовата елиминација, пивот елементи се елементите од главната дијагонала a_{ii}^{ut} при секоја итерација посебно во постапката за добивање на матрицата A^{ut}.

Пример 3.1.

Како функционира алгоритмот на елиминација на Гаус на систем од линерани алгебарски равенки со и без пивот елемент е прикажано во програмата gaus_elination.for. Кога системот од равенки, даден подолу:

$$\begin{aligned} x_1 + x_2 + 0 + 3x_4 &= 4 \\ 2x_1 + x_2 - x_3 + x_4 &= 1 \\ 3x_1 - x_2 - x_3 + 2x_4 &= -3 \\ -x_1 + 2x_2 + 3x_3 - x_4 &= 4 \end{aligned}, \qquad (3.7)$$

нема пивот елемент, како влез за програмата се користи датотеката system_1.dat, а како излез ги добиваме решенијата на системот: x_1=-1, x_2=2, x_3=0 и x_4=1. Кога системот равенки го решаваме со пивот елемент, каков што е следниот:

$$\begin{aligned} x_1 - x_2 + 2x_3 - x_4 &= -8 \\ 2x_1 - 2x_2 + 3x_3 - 3x_4 &= -20 \\ x_1 + x_2 + x_3 + 0x_4 &= -2 \\ x_1 - x_2 + 4x_3 + 3x_4 &= 4 \end{aligned}, \qquad (3.8)$$

како влез за програмата се користи датотеката system_2.dat, а како излез ги добиваме решенијата на системот: x_1=-7, x_2=3, x_3=2 и x_4=2. Во овој случај, после првата трансформација на системот како пивот елемент во втората равенка се јавува елементот $a_{22}^{(2)} = 0$ па процедурата не може да продолжи. Затоа, ќе направиме замена на редовите се додека не дојдиме до ред за кој $a_{22}^{(2)} \neq 0$ ..

```
      PROGRAM GAUSS_ELIMINATION
      IMPLICIT REAL*8(A-H,P-Z)
      DIMENSION A(10,11), X(10)
      CHARACTER NAME1*30

C     VLEZNATA DATOTEKA SYSTEM_1.DAT NEMA PIVOT ELEMENT
C     VLEZNATA DATOTEKA SYSTEM_2.DAT IMA PIVOT ELEMENT A(2,2)=0
      OPEN(UNIT=4,FILE='SYSTEM_1.DAT',ACCESS='SEQUENTIAL')

      WRITE(*,*) ' '
      WRITE(*,*) 'Metod na Gausova eliminacija so pivot element'
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) FLAG
      WRITE(*,*) ' '

      IF(FLAG .EQ. 2.)THEN
      WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
      WRITE(*,*) 'drive:name.ext'
      WRITE(*,*) 'patot da bide staven vo navodnici'
      WRITE(*,*) 'na primer:"C:/DATA.TXT" '
```

```
      WRITE(*,*) 'direktoriumot kade se zapisuva treba da postoi!'
      READ (*,*) NAME1
      WRITE(*,*) ' '
        OUP = 5
      OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
      OUP = 6
      ENDIF

C     ICHG BROJ NA IZMENI NA ELEMENTITE NA MATRICATA
      ICHG = 0

      READ(4,*)N
      M=N+1

      READ(4,*)((A(I,J),J=1,M),I=1,N)
      WRITE(OUP,3)
      WRITE(OUP,4) ((A(I,J),J=1,M),I=1,N)
C     POCETOK NA PROCES NA ELIMNIACIJA NANPRED
      NN = N-1
      DO 10 I=1,NN
      IP = I
C     PREBARUVANJE ZA PIVOT ELEMENT
   1  0  0  IF(ABS(A(IP,I)).GE.1.0E-20.OR.IP.GT.N) GOTO 200
      IP = IP+1
      GOTO 100
   2  0  0  IF(IP.EQ.N+1)THEN
C     SISTEMOT NEMA EDINSTVENO RESENIE
      WRITE(OUP,5)
      GOTO 400
      ENDIF
C     ZAMENA NA REDOVI DOKOLKU SE NAJDI PIVOT ELEMENT
      IF(IP.NE.I) THEN
      DO 20 JJ=1,M
      C = A(I,JJ)
      A(I,JJ) = A(IP,JJ)
   2  0  A(IP,JJ) = C
      ICHG = ICHG+1
      ENDIF
C     PRODOLZUVANJE NA PROCESOT NA ELIMINACIJA
      JJ = I+1
      DO 30 J=JJ,N
      XM = A(J,I)/A(I,I)
      DO 40 K=JJ,M
   4  0  A(J,K) = A(J,K)-XM*A(I,K)
   3  0  A(J,I) = 0
   1  0  CONTINUE
      IF(ABS(A(N,N)).LT.1.0E-20) THEN
C     SISTEMOT NEMA EDINSTVENO RESENIE
      WRITE(OUP,5)
      GOTO 400
      ENDIF

C     POCETOK NA ZAMENA (RESAVANJE) NANAZAD
      X(N) = A(N,N+1)/A(N,N)
      L = N-1
      DO 15 K=1,L
      I = L-K+1
      JJ = I+1
      SUM = 0.0
      DO 16 KK=JJ,N
   1  6  SUM = SUM-A(I,KK)*X(KK)
   1  5  X(I) = (A(I,N+1)+SUM)/A(I,I)
      WRITE(OUP,6)((A(I,J),J=1,M),I=1,N)
```

```
C           ISPISUVANJE NA RESENIJATA NA SISTEMOT
            DO 50 I=1,N
            WRITE(OUP,7)I,X(I)
   5  0     CONTINUE
C           ISPISUVANJE NA BROJOT NA ZAMENA NA REDOVI
            WRITE(OUP,8)ICHG
   4  0  0  CLOSE(UNIT=5)
            CLOSE(UNIT=OUP)
            IF(OUP.NE.6) CLOSE(UNIT=6)

   3        FORMAT(1X,'ORIGINALEN SISTEM:')
   4        FORMAT(5(1X,E15.8))
   5        FORMAT(1X,'SISTEMOT NEMA EDINSTVENO RESENIE')
   6        FORMAT(/,1X,'REDUCIRAN SISTEM:',/,(5(1X,E14.8)))
   7        FORMAT(/,'X',I1,'=',E15.8)
   8        FORMAT(/,1X,'VKUPNO ZAMENI NA REDOVI:',3X,I2)
            END
```

Дел од излезот на програмата за систем без пивот елемент е:

```
X1=-0.10000000E+01
X2= 0.20000000E+01
X3= 0.00000000E+00
X4= 0.10000000E+01

VKUPNO ZAMENI NA REDOVI:    0
```

Дел од излезот на програмата за систем со пивот елемент е:

```
X1=-0.70000000E+01
X2= 0.30000000E+01
X3= 0.20000000E+01
X4= 0.20000000E+01

VKUPNO ZAMENI NA REDOVI:    1
```

3.2. Итеративни методи на Јакоби и Гаус-Зајдел

Многу често во научните и техничките проблеми матрицата A со која се прикажува даден систем е ретка матрица или претежно дијагонална, што значи дека примената на директните методи за решавање на линеарни системи значително го троши компјутерското време во „празен ὸд“. Затоа, во такви случаи се препорачува примена на итеративни методи. Вообичаено, кај итеративните методи се започнува со некое почетно решение $x_j^{(0)}$, за j=1,…n, а потоа, почетните решенија се користат за подобрување на истите во секоја итерација, се додека не се достигне посакуваната точност. Во тој случај, општиот облик на решенијата за секоја $k-$та итерација е познат како итеративен метод на Јакоби и се задава со изразот

$$x_i^{(k)} = \frac{b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}, \qquad (3.9)$$

за $i = 1,2,\ldots n$.

Со цел подобрување на Јакобиевата шема, може да направиме определени подобрувања на изразот (3.9) на следниов начин. Бидејќи за пресметување на $x_i^{(k)}$ се користат компоненти од $x_j^{(k-1)}$, за секое $j>1$, во $k-$та итерација, решенијата $x_j^{(k)}$ за $j=1,\ldots i-1$, веќе се уточнети во однос на истите решенија $x_j^{(k-1)}$ од претходната итерација. Затоа, поопправдано е во секоја $k-$та итерација, за $j=1,\ldots i-1$, да се искористат уточнетите решенија $x_j^{(k)}$ наместо $x_j^{(k-1)}$. Со ова, изразот (3.9) може да се презапише како

$$x_i^{(k)}=\frac{b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k-1)}}{a_{ii}}. \tag{3.10}$$

со што се дефинира итеративната шема (метод) на Гаус-Зајдел. За да може да се примени овој метод, претходно системот од линеарни равенки да се трансформира во облик во кој сите дијагонални коефициенти ќе бидат различни од нула и тие коефициенти да бидат најголеми, односно доминантни, т.е. да биде исполнет условот

$$\left|a_{ii}\right|\geq\sum_{j=1,j\neq i}^{n}\left|a_{ij}\right|, \tag{3.11}$$

при што знакот > треба да биде исполнет барем за една равенка или ред во матрицата. Брзината на конвергенција зависи од доминантноста на дијагоналните коефициенти, така што со зголемување на нивната доминантност, односно бројна вредност, се намалува зависноста од бројот на итерации како и од вредноста на почетните решенија.

Пример 3.2.

Решавањето на линеарен ситем од равенки со метод на Гаус-Зајдел е прикажано во програмата gaus_zajdel.for. Како пример е земен следниот систем:

$$\begin{aligned}12x_1+3x_2-5x_3&=1\\ x_1+5x_2+3x_3&=28,\\ 3x_1+7x_2+13x_3&=76\end{aligned} \tag{3.12}$$

што во програмата се внесува преку датотеката system_3.dat, а за излез ги добиваме решенијата на системот: x_1=1, x_2=3 и x_3=4.

```
      PROGRAM GAUS_ZAJDEL

      IMPLICIT REAL*8(A-H,P-Z)
      DIMENSION COEF(5,6), D, DX(5), X(5,6), XN(5), XNP(5)
      CHARACTER NAME1*30,AA*1

C     VLEZNATA DATOTEKA SYSTEM_3.DAT ZA SISTEM OD TRI RAVENKI
C     VLEZNATA DATOTEKA SYSTEM_4.DAT ZA SISTEM OD PET RAVENKI

      OPEN(UNIT=4,FILE='SYSTEM_3.DAT',ACCESS='SEQUENTIAL')

      WRITE(*,*) ' '
      WRITE(*,*) 'Resavanje na sistem linearni ravenki'
      WRITE(*,*) 'so metod na Gaus-Zajdel'

      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) FLAG
```

```
      WRITE(*,*) ' '

      IF ( FLAG .EQ. 2 ) THEN
         WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
         WRITE(*,*) 'drive:name.ext'
         WRITE(*,*) 'patot da bide staven vo navodnici'
         WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
         READ (*,*) NAME1
         WRITE(*,*) ' '
         OUP = 3
         OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
         OUP = 6
      ENDIF
C     NO E BROJ NA RAVENKI A NV E BROJ NA PROMENLIVI
      READ(4,*) NO
      DO I=1,NO
         XN(I) = 0.D0
         XNP(I) = 0.D0
      ENDDO
      READ(4,*) NV
      WRITE(OUP,901)
C     KOEFICIENTITE SE VNESUVAAT VO MATRICATA X(I,J)
C     RESENIJATA NA RAVENKITE SE VO POSLEDNATA KOLONA
      DO I=1,NO
         READ(4,*)(X(I,J),J=1,NO+1)
C        D SE KOEFICIENTITE OD GLAVNATA DIJAGONALA NA X(I,J)
C        STO VLEGUVAAT VO BROITELITE NA FORMULITE ZA
C        PRESMETUVANJE NA PROMENLIVITE
         D = X(I,I)
         DO J=1,NO+1
            COEF(I,J) = X(I,J)/D
         ENDDO

C     ZARADI PRILAGODUVANJE KON PRESMETKITE, KOEFICIENTITE
C     OD GLAVNATA DIJAGONALA NA MATRICATA X(I,J) SE IZDNACUVAAT
C     SO NULA, A OSTANATITE SE PREZAPISUVAAT KAKO DA SE PODELENI
C     SO D
         COEF(I,I) = 0.D0
         WRITE(OUP,900)(X(I,J),J=1,NV+1)
      END DO
      WRITE(OUP,902)
      DO I=1,NO
         WRITE(OUP,900)(COEF(I,J),J=1,NV+1)
      ENDDO
      WRITE(OUP,903)

      TOL = 1.E-4
      ITERATE = 0
    1 ITER = 0
C     ITERATE E BROJAC KOJ GI SLEDI ITERACIITE VO PRESMETKITE
      ITERATE = ITERATE+1

C     SLEDI RESAVANJE NA SISTEMOT RAVENKI SPORED FORMULATA NA
C     NA GAUSS-SEIDEL
      DO I=1,NO
         XN(I) = COEF(I,NV+1)
         DO J=1,NV
            XN(I) = XN(I) - COEF(I,J)*XN(J)
         ENDDO
      ENDDO

C     DX E VEKTOR KOJ NI JA DAVA KONVERGENCIJATA NA RESENIJATA
      DO I=1,NO
         DX(I) = XNP(I) - XN(I)
         XNP(I) = XN(I)

C     NA POCETOKOT NA SEKOJ CIKLUS ITER E EDNAKVO NA 0.
C     AKO PROMENATA E POGOLEMA OD NEKOJA MINIMALNA VREDNOST
C     TOGAS ITER E EDNAKVO NA 1. SE DODEKA ITER E EDNAKVO
C     NA 1 ZNACI DEKA CIKLUSOT NA ITERACII TREBA DA PRODOLZI
         IF (DABS(DX(I)).GT. TOL) ITER=1
C     SE OBNOVUVA VREDNOSTA NA RESENIJATA
```

```
            XN(I) = XNP(I)
            IF (ITERATE.GE.50)THEN
               WRITE(OUP,905)
               GOTO 60
            ENDIF
         ENDDO
         WRITE(OUP,904)ITERATE,(XN(I),I=1,NV),(DX(I),I=1,NV)
         IF (ITER.GT.0) GO TO 15
         CLOSE(UNIT=5)
         CLOSE(UNIT=OUP)
         IF(OUP.NE.OUP) CLOSE(UNIT=6)
    6  0 STOP

 9  0  0 FORMAT(5X,4(F10.4,5X))
 9  0  1 FORMAT('KOEFICIENTI NA RAVENKITE SO'
     *  ' NIVNITE RESENIJA (SLOBODNI CLENOVI):',/)
 9  0  2 FORMAT(//,'KOEFICIENTI NA PREZAPISANITE RAVENKI'
     *  ' ZA PRESMETUVANJE NA PROMENLIVITE:',/)
 9  0  3 FORMAT(//,'PRESMETANI REZULTATI POSLE SEKOJA ITERACIJA:'
     *  //,'ITER',7X,'X(1)',8X,'X(2)',9X,'X(3)',7X,'DX(1)',7X,
     *  'DX(2)',7X,'DX(3)',/)
 9  0  4 FORMAT(I3,4X,6(F10.5,2X))
 9  0  5 FORMAT(//,'NEKOE RESENIE NE KONVERGIRA, KRAJ!',/)
      END
```

Дел од излезот на програмата за систем со пивот елемент е:

```
PRESMETANI REZULTATI POSLE SEKOJA ITERACIJA:

ITER       X(1)        X(2)        X(3)       DX(1)       DX(2)       DX(3)

  1       0.08333     5.58333     2.82051    -0.08333    -5.58333    -2.82051
  2      -0.13729     3.93515     3.75891     0.22062     1.64818    -0.93840
  3       0.66576     3.21150     3.96325    -0.80304     0.72365    -0.20434
  4       0.93181     3.03569     3.99652    -0.26605     0.17581    -0.03327
  5       0.98963     3.00416     4.00015    -0.05782     0.03153    -0.00363
  6       0.99902     3.00010     4.00017    -0.00940     0.00406    -0.00002
  7       1.00004     2.99989     4.00005    -0.00102     0.00021     0.00012
  8       1.00005     2.99996     4.00001     0.00000    -0.00007     0.00004
```

3.3. Тридијагонални матрици

Тридијагоналните матрици се специјален тип на„зонски“ матрици затоа што различни од нула се само елементите a_{ij} по главната дијагонала и дијагоналните под и над главната дијагонала, што во општ случај може да се запиши како

$$T = \begin{bmatrix} t_{11} & t_{12} & 0 & \dots & 0 \\ t_{21} & t_{22} & t_{23} & \ddots & \vdots \\ 0 & t_{32} & t_{33} & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & t_{n-1,n} \\ 0 & \dots & 0 & t_{n,n-1} & a_{nn} \end{bmatrix}. \qquad (3.13)$$

За разрешување на системите од линеарни равенки од типот $T x_i = b_i$, што се прикажуваат со вакви матрици може да се искористат алгоритмите за факторизација кои во овој случај се упростуваат затоа што многу елементи t_{ij} за $i \neq j$ се еднакви на нула. Како пример, ќе го разгледаме методот на Крут со кој матрицата (3.13) се факторизира на L и U матрици како:

$$L=\begin{bmatrix} l_{11} & 0 & \dots & \dots & 0 \\ l_{21} & l_{22} & \ddots & \ddots & \vdots \\ 0 & l_{32} & l_{33} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & l_{n,n-1} & l_{nn} \end{bmatrix} \quad U=\begin{bmatrix} 1 & u_{12} & \dots & \dots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix} \tag{3.14}$$

Ако се има предвид дека матрицата A има само $(3n-2)$ елементи различни од нула, тогаш L матрицата ќе има $(2n-1)$ елементри различни од нула а U матрицата има само $(n-1)$ елементри различни од нула. Ако ги помножиме L и U матриците, и производот го споредиме со матрицата A, ќе добиеме

$$t_{11}=l_{11}, \tag{3.15}$$

а за останатите $i=2,3\dots,n$ имаме

$$t_{i,i-1}=l_{i,i-1}, \tag{3.16}$$

и

$$t_{ii}=l_{i,i-1}u_{i-1,i}+l_{ii}, \tag{3.17}$$

а истотака, за $i=1,2\dots,n-1$ може да ја напишиме релацијата

$$t_{i,i+1}=l_{ii}u_{i,i+1}. \tag{3.18}$$

Решението на линеарниот систем кој во матрична форма се запишува со тридијагонална матрица може да го најдиме со методите на Гаусова елиминација, така што матричната форма на системот, сега се запишува во облик

$$T\,x_i=LU\,x_i=b_i. \tag{3.19}$$

Ако земиме предвид дека $U\,x_i=z_i$, решенијата на линеарниот ситем произлегуваат од матричната равенка

$$L\,z_i=b_i, \tag{3.20}$$

за $i=1,2,3\dots,n$.

3.3.1. Сопствени вредности на тридијагонални матрици

Вакви матрици најчесто се јавуваат при моделирање на физичките процеси. Сопствените вредности λ_i на тридиагоналните матрици всушност се корени на карактеристичниот полином од *n*-ти ред

$$p_n(\lambda)\equiv\det\left(T-\lambda I\right)=0. \tag{3.21}$$

Заради поедноставно разгледување, ќе се задржиме на матрици T кои се реални и симетрични $t_{ij}=t_{ji}$, така што сопствените вредности се секогаш реални а сопствените вектори можат да бидат ортонормирани.

Полиноманата равенка (3.21) може да се генерира со рекурентната релација

$$p_i(\lambda)=(a_i-\lambda)\,p_{i-1}(\lambda)-b_{i-1}^2 p_{i-2}(\lambda), \tag{3.22}$$

каде што $a_i=t_{ii}$ а $b_i=t_{ii+1}=t_{i+1i}$. Со ова, решенијата на тридијагоналната матрица се сцедуваат на примена на приближни методи за наоѓање корен на полиномни равенки од *n*-ти ред.

Пример 3.3.

Како се преименува методот на LU факторизација за решавање на линеарен симетричен тридијагонален систем ќе покажиме со програмата tridag.for. Проширената форма на *T* матрицата на системот што го разгледуваме е:

$$T=\begin{cases} 2x_1-x_2+0+0=1 \\ -x_1+2x_2-x_3+0=0 \\ 0-x_2+2x_3-x_4=0 \\ 0+0-x_3+2x_4=1 \end{cases}, \tag{3.23}$$

и истата во програмата се внесува преку датотеката sistem1.dat и тоа во посебен облик зададен со

$$T^{(m)}=\begin{cases} 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & \\ -1 & -1 & -1 & \\ 1 & 0 & 0 & 1 \end{cases}, \tag{3.24}$$

каде што во првиот ред се коефициентите од главната дијагонала, во вториот и третиот ред се коефциентите од двете поддијагонали и во четвртиот ред се слободните коефициенти. Како излез, програмата ни ги дава решенијата на системот: x_1=1, x_2=1, x_3=1 и x_4=1.

Кодот на програмата е прикажан во текстот што следи.

```
      PROGRAM GAUSS_ELIMINATION
      IMPLICIT REAL*8(A-H,P-Z)
      DIMENSION A(10), B(10),C(10),BB(10),Z(10),X(10)
      CHARACTER NAME1*30
      OPEN(UNIT=4,FILE='SISTEM1.DAT',ACCESS='SEQUENTIAL')
      INP=4
      READ(4,*) N
      IF (N .GT. 0) THEN
      NN=N-1
C     A(I,I) SE ZACUVANI KAKO A(I), 1<=I<= N
      READ(INP,*) (A(I),I=1,N)
C     DOLNATA SUBDIJAGONALA A(I,I-1) SE ZACUVUVA KAKO
C     B(I), 2<=I<=N
      READ(INP,*) (B(I),I=2,N)
C     GORNATA SUBDIJAGONALA A(I,I+1) SE ZACUVUVA KAKO
C     C(I), 1<=I<=N-1
      READ(INP,*) (C(I),I=1,NN)
      READ(INP,*) (BB(I),I=1,N)
      OK = .TRUE.
      CLOSE(UNIT=INP)
      ENDIF
      WRITE(*,*) ' '
      WRITE(*,*) 'LU faktorizacija na tridijagonalen sistem'
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
```

```
      WRITE(*,*) ' '
      IF ( NFLAG .EQ. 2 ) THEN
      WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
      WRITE(*,*) 'drive:name.ext'
      WRITE(*,*) 'patot da bide staven vo navodnici'
      WRITE(*,*) 'na primer:"C:/DATA.TXT" '
      WRITE(*,*) 'direktoriumot kade se zapisuva treba da postoi!'
      READ (*,*) NAME1
      WRITE(*,*) ' '
      OUP = 5
      OPEN(UNIT=OUP,FILE=NAME1,STATUS='NEW')
      ELSE
      OUP = 6
      ENDIF
      WRITE(OUP,*) 'LU FAKTORIZACIJA ZA TRIDIJAGONALNI SISTEMI'
C     ELEMENTITE NA U SE ZAMENETI SO C A ELEMENTITE NA L SE ZAMENETI SO A
      C(1)=C(1)/A(1)
      Z(1)=BB(1)/A(1)
      DO 10 I=2,NN
C     I-TI RED NA L
      A(I)=A(I)-B(I)*C(I-1)
C     (I+1)-VA KOLONA NA U
      C(I)=C(I)/A(I)
      Z(I)=(BB(I)-B(I)*Z(I-1))/A(I)
   1  0  CONTINUE
C     N-TI RED NA L
      A(N)=A(N)-B(N)*C(N-1)
      Z(N)=(BB(N)-B(N)*Z(N-1))/A(N)
C     SE RESAVA RAVENKATA U*X=Z
      X(N)=Z(N)
      DO 30 II=1,NN
      I=NN-II+1
   3  0  X(I)=Z(I)-C(I)*X(I+1)
      WRITE(OUP,8)
      WRITE(OUP,9) (X(I),I=1,N)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.6) CLOSE(UNIT=6)
      STOP
   8  FORMAT(1X,'RESENIJATA SE:')
   9  FORMAT(1X,4(2X,F15.8))
      END
```

Излезот на програмата е:

```
 LU FAKTORIZACIJA ZA TRIDIJAGONALNI SISTEMI
 RESENIJATA SE:
 X(1)             X(2)             X(3)             X(4)
 1.00000000       1.00000000       1.00000000       1.00000000
```

3.4. Задачи

1. Користејќи го итеративниот метод на Гаус-Зајдел да се реши системот од линеарни равенки зададен со:

$$\begin{aligned} 4x_1 - x_2 + 0 + x_4 + 0 &= 100 \\ -x_1 + 4x_2 - x_3 + 0 + x_5 &= 100 \\ 0 - x_2 + 4x_3 - x_4 + 0 &= 100 \\ x_1 + 0 - x_3 + 4x_4 - x_5 &= 100 \\ 0 + x_2 + 0 - x_4 + 4x_5 &= 100 \end{aligned}.$$

со точност TOL=0.00001. Решенијата на системот се x_1=25.0000, x_2=35.7143, x_3=42.8571, x_4=35.7143 и x_5=25.0000.

2. Во дадено електрично коло, примената на Кирховите закони ни го дава следниот систем од шест линерани равенки со шест непознати:

$$\begin{array}{rcrcrcrcrcrcl}
11\upsilon_1 & - & 5\upsilon_2 & + & 0 & + & 0 & + & 0 & - & \upsilon_6 & = & 500 \\
-20\upsilon_1 & + & 41\upsilon_2 & - & 15\upsilon_3 & + & 0 & - & 6\upsilon_5 & + & 0 & = & 0 \\
0 & - & 3\upsilon_2 & + & 7\upsilon_3 & - & 4\upsilon_4 & + & 0 & + & 0 & = & 0 \\
0 & + & 0 & - & \upsilon_3 & + & 2\upsilon_4 & - & \upsilon_5 & + & 0 & = & 0 \\
0 & - & 3\upsilon_2 & + & 0 & - & 10\upsilon_4 & + & 28\upsilon_5 & - & 15\upsilon_6 & = & 0 \\
-2\upsilon_1 & + & 0 & + & 0 & + & 0 & - & 15\upsilon_5 & + & 47\upsilon_6 & = & 0
\end{array}.$$

Да се реши овој систем со користење на методите на Гаусова елиминација и Гаус-Зајдел. Кој метод е посоодветен за решавање на овој линеарен систем?

3. Користејќи го методот на Гаусова елиминација да се реши системот од линеарни равенки зададен со:

$$\begin{array}{rcrcrcrcl}
0.7321x_1 & + & 0.4135x_2 & + & 0.3126x_3 & + & 0.5163x_4 & = & 0.8132 \\
0.2317x_1 & + & 0.6123x_2 & + & 0.4137x_3 & + & 0.6696x_4 & = & 0.4753 \\
0.4283x_1 & + & 0.8176x_2 & + & 0.4257x_3 & + & 0.8312x_4 & = & 0.2167 \\
0.8653x_1 & + & 0.2165x_2 & + & 0.8265x_3 & + & 0.7123x_4 & = & 0.5154
\end{array}.$$

Да се споредат приближните решенија со точните: x_1=8,973596, x_2=70.07471, x_3=64,51428, x_4=-106,3324, во случаите кога промнливите се декларирани како реални променливи и променливи со двојна прецизност.

4. ИНТЕРПОЛАЦИЈА И ПОЛИНОМНА АПРОКСИМАЦИЈА

Во компјутерската и експерименталната физика постојано се среќаваме со резултати што се приближно пресметани или измерени, односно во процедурите на мерење или пресметување се среќаваме со соодветни грешки и неопределености. Затоа, за физичарите е многу важно да знаат да донесат заклучоци врз основа на група податоци што се фитувани со дадена крива и истите да ги воопштат и донесат нови заклучоци.

Проблемот на фитување на дадена функција $f(x_i)$, или табела на податоци, се сведува на определување функцијата $F(x_i)$ која припаѓа на позната класа на функции со која се апроксимира $f(x_i)$, за $i=0,1,...n$, при што јазлите $x_0, x_1 ... x_n$ и вредностите на $f(x_i)$ во тие точки се познати, т.е.

$$f(x_0)=y_0, \quad f(x_1)=y_1, \ldots, f(x_n)=y_n. \tag{4.1}$$

Воопшто гледано, низ точките x_i можат да се повлечат безброј разни криви, па со тоа задачата не е еднозначна. За да биде решението еднозначно, класата на функции за апроксимација во која се бара решение се дели на *линеарни* и *нелинеарни* апроксимации. За линеарните апроксимации најчесто се користат алгебарските и тригонометриските полиноми, а за нелинеарните апроксимации се користат експоненцијалните и рационалните функции. За определување на апроксимациските функции се користат два приода:

1. Интерполација,

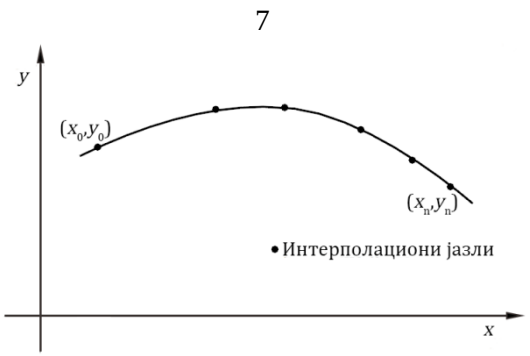
2. Апроксимација.

Кога е во прашање интерполацијата, потребно е функцијата $f(x_i)$ и апроксимираната функција $F(x_i)$ да се совпаѓаат целосно во сите интерплациони јазли, т.е.

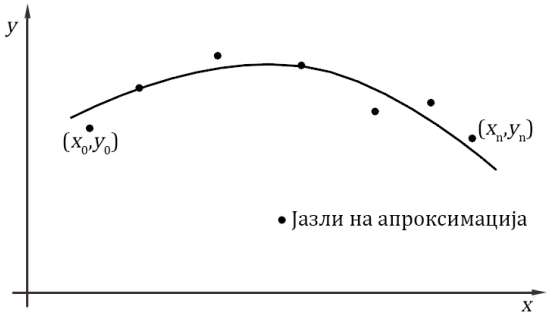
$$F(x_0)=y_0, \quad F(x_1)=y_1, \ldots F(x_n)=y_n, \tag{4.2}$$

што геометриски значи дека кривата $y=F(x)$ треба да минува низ сите точки (интерполациони јазли) $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$ како што е прикажано на слика 4.1. Ова ни кажува дека интерполацијата ни е потребна кога сакаме да донесиме некои „локални“ заклучоци од група дискретни податоци за областите измеѓу интерплолационите јазли.

Од друга страна, апроксимацијата на функцијата не побарува задолжително совпаѓање на $f(x_i)$ и $F(x_i)$ во сите јазли, па затоа, геометриски, апроксимираната функција $F(x_i)$ може да помини само низ некои од зададена точки x_i, слика 4.2. Затоа се вели дека апроксимацијата ни го определува општото или глобалното поведение на податоците.



Сл. 4.1.



Сл. 4.2.

4.1. Лагранжова интерполација

Основната задача на интерполацијата е даена функција $f(x)$ да се апроксимира со друга функција $g(x)$ така што

$$f(x) = g(x) + O(x), \tag{4.3}$$

каде што $O(x)$ е грешката на апроксимацијата или грешка на отфрлање (truncation error). Заради (4.1) може да напишиме за $j = 0,1,...n$ важи

$$f(x_j) \approx g_n(x_j), \tag{4.4}$$

каде $g_n(x_j)$ е полином од $n-$ти степен. Ако избериме само две соседни точки, на пример x_0 и x_1 и $n=1$, тогаш низ тие двете точки може еднозначно да повлечиме дамо една права (отсечка) која ќе ни ја дади *линеарната интерполација* на податоците зададена со

$$g_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1), \tag{4.5}$$

или во поконцизна форма

$$f(x) \approx g_1(x) = \sum_{j=i}^{i+1} f(x_j) p_{1j}(x), \tag{4.6}$$

каде коефициентите на интерполација се

$$p_{1j} = \frac{x - x_k}{x_j - x_k} \tag{4.7}$$

за $k \neq j$. Ако сакаме оваа постапка да ја воопштиме за крива од $n-$ти ред која поминува низ $n+1$ точки, интерполационите коефициенти се задаваат со изразот

$$p_{nj} = \prod_{\substack{k \neq j \\ j=0}}^{n} \frac{x - x_k}{x_j - x_k}, \tag{4.8}$$

со што за интерполациониот полином го добиваме изразот

$$g_n(x) = \sum_{j=0}^{n} f(x_j)\, p_{nj}(x)\,. \tag{4.9}$$

Овој полином е познат како *Лагранжов интерполационен полином* а постапката за негово определување како Лагранжова интерполација.

Пример 4.1.

Во програмата lagranz_quadratic.for се пресметува Лагранжовиот интерполационен полином $g_3(x_i)$ со кој се апроксимира вредноста на $\sin(44.4444°)$ за податоците што се дадени во табелата подолу

x_i	$f(x_i)$
40º	0.642788
45º	0.707107
50º	0.766044

а потоа пресметаната вредност да се спореди со точната $\sin(44.4444°) = 0.70021679$.

```
      PROGRAM LAGRANGE_INTERPOLATION
C     PROGRAM ZA INTERPOLACIJA SO SO KVADRATEN LAGRANGE-OV POLINOM
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER*4 MAXN,I,N
      PARAMETER(MAXN = 1000)
      DIMENSION X(5),Y(5),XP(1),XI(MAXN),YI1(MAXN),YI2(MAXN),YI3(MAXN)
      OPEN(11,FILE='XY.txt',STATUS='UNKNOWN')
      OPEN(12,file='XIYI.txt',STATUS='UNKNOWN')

C     VNESI GI JAZLITE (TOCKITE) ZA INTERPOLACIJA
      WRITE(*,*)'VNESI GI PODATOCITE:'
      DO I=1,3
      WRITE(*,*)'X(',I,') = '
      READ(*,*) X(I)
      WRITE(*,*)'Y(',I,') = '
      READ(*,*) Y(I)
      ENDDO

C     VNESI JA VREDNOSTA NA XI ZA KOJA SE BARA YI
      WRITE(*,*)'VNESI GO XP ZA KOE SE PRESMETUVA Y(XP)'
      WRITE(*,*)'XP = '
      READ(*,*) XP(1)

      YP=FINTRP2(XP(1),X,Y)
      WRITE(*,'(2(2X,F10.7))')XP(1),YP
C     OBLAST NA INTERPOLACIJA OD XMIN DO XMAX
      WRITE(*,*) 'VNESI JA MINIMALNATA VREDNOST NA X: '
      READ(*,*) XMIN
      WRITE(*,*) 'VNESI JA MAKSIMALNATA VREDNOST NA X: '
      READ(*,*) XMAX

C     OPREDELYVANJE NA YI ZA DADENI VREDNOSTI NA XI SO
C     FUNKCIJATA FINTRP
      NPLOT = 100!BROJ NA TOCKI ZA INTERPLOCIJA
      DO I=1,NPLOT
      XI(I) = XMIN + (XMAX-XMIN)*(I-1)/(NPLOT-1)
      YI2(I) = FINTRP2(XI(I),X,Y)
      ENDDO

      DO I=1,3
      WRITE(11,*) X(I),Y(I)
      ENDDO

      DO I=1,NPLOT
      WRITE(12,'(2(2X,F10.7))') XI(I),YI2(I)
      ENDDO
      END
```

```
      REAL*8 FUNCTION FINTRP2(XI,X,Y)
C     FUNKCIJA ZA INTERPOLACIJA POMEGJU PODATOCI SO
C     KORISTENJE NA KVADRATEN LAGRANGE-OV POLINOM
      REAL*8 XI,X(3),Y(3),YI2

      AL1=(XI-X(2))*(XI-X(3))/((X(1)-X(2))*(X(1)-X(3)))
      AL2=(XI-X(1))*(XI-X(3))/((X(2)-X(1))*(X(2)-X(3)))
      AL3=(XI-X(1))*(XI-X(2))/((X(3)-X(1))*(X(3)-X(2)))
      YI2 =AL1*Y(1)+AL2*Y(2)+AL3*Y(3)
      FINTRP2 = YI2
      RETURN
      END
```

```
  X(I)          F(X(I))
  . . .         . . .
  44.0404040    0.6951803
  44.1414141    0.6964451
  44.2424242    0.6977076
  44.3434343    0.6989680
  44.4444444    0.7002262
  44.5454545    0.7014822
  44.6464646    0.7027360
  44.7474747    0.7039876
  44.8484848    0.7052370
  44.9494949    0.7064842
  45.0505051    0.7077292
  45.1515152    0.7089720
```

Еден дел од излезната датотека xiyi.txt, во кој ќе наоѓа и апроксимираната вредност $\sin(44.444444°) = 0.7002262$, е прикажана погоре. Може да се забележи дека совпаѓањето на пресметаната и точната вредност е со ред на големина $\approx 10^{-5}$, што зависи од бројот на децимали со кој се изразува аргументот на функцијата.

4.2. Њутнов интерполационен полином

Полином кој поминува низ сите интерполациони јазли (x_i, y_i), што се поставени еквидистантно во интервалот $[x_0, x_1, \ldots, x_n]$, е и *првиот Њутновиот интерполационен полином* или *Њутнов интерполационен полином за диференцирање напред* кој што може да се запиши како:

$$P_n(x) = y_0 + \binom{t}{1}\Delta y_0 + \binom{t}{2}\Delta^2 y_0 + \cdots + \binom{t}{n}\Delta^n y_0, \tag{4.10}$$

каде параметарот *t*, познат и како интерполациона променлива, се задава со изразот

$$t = \frac{x - x_0}{h}, \qquad \text{или} \qquad x = x_0 + t\,h, \tag{4.11}$$

при што $h = x_{i+1} - x_i$ а конечните разлики од *n*-ти ред се дефинираат со:

$$\Delta^n y_0 = \Delta^{n-1} y_1 - \Delta^{n-1} y_0 = \sum_{i=0}^{n} (-1)^{i+n} \binom{n}{i} y_i \,. \tag{4.12}$$

Изразот (4.10) се користи за интерполација на вредности кои се наоѓаат на почетокот или средината на интервалот $x \in [x_0, x_n]$. Во случаите кога треба да се интерполираат вредности што се наоѓаат во близина на x_n треба да се користи друга

формула позната како *втор интерполационен полином на Њутн* или *Њутнов интерполационен полином за диференцирање наназад* кој се задава со:

$$P_n(x) = y_0 + \binom{t^+}{1}\Delta y_0 + \binom{t^+}{2}\Delta^2 y_0 + \cdots + \binom{t^+}{n}\Delta^n y_0, \tag{4.13}$$

при што интерполационата променлива t^+ се дефинира со изразот:

$$\binom{t^+}{i} = \frac{t(t+1)(t+2)\cdots(t+[i-1])}{i!}, \tag{4.14}$$

а останатите променливе се исти како и претходните во (4.10).

Пример 4.2.

Во програмата njutnova_interpolacija.for се пресметува првиот Њутнов интерполационен полином за податоците што се дадени во табелата подолу

x_i	$f(x_i)$
1.0	0.7651977
1.3	0.6200860
1.6	0.4554022
1.9	0.2818186
2.2	0.1103623

а потоа се пресметува $P_4(1.5)$ според формулата:

$$\begin{aligned} P_4(1.5) &= 0.7651977 - 0.4837057(x-1.0) - 0.1087339(x-1.0)(x-1.3) + \\ &+0.0658784(x-1.0)(x-1.3)(x-1.6) + \\ &+0.0018251(x-1.0)(x-1.3)(x-1.6)(x-1.9) = 0.5118200. \end{aligned} \tag{4.15}$$

```
      PROGRAM NJUTNOVA_INTERPOLACIJA
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION X(25),Q(25,25)
      CHARACTER NAME1*30

      OPEN(UNIT=4,FILE='DATA.DAT',ACCESS='SEQUENTIAL')
      WRITE(*,*) 'Njutnov interpolacionen polinom'
      WRITE(*,*) 'Odberi vid na izlez na podatoci: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Tekstualna datoteka '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      WRITE(*,*) ' '
      READ(*,*) FLAG
      IF ( FLAG .EQ. 2 ) THEN
      WRITE(*,*)'Vnesi go imeto na datotekata kako - '
      WRITE(*,*)'drive:ime.ext'
      WRITE(*,*)'zadilzitelno staveni vo navodnici'
      WRITE(*,*)'na primer:''C:/OUTPUT.DAt'' '
      WRITE(*,*)' '
      READ(*,*) NAME1
      OUP = 3
      OPEN(UNIT=OUP,FILE=NAME1,STATUS='NEW')
      ELSE
      OUP = 6
      ENDIF

C     SE VCITUVA BROJOT NA INTERPOLACIONI JAZLI
      READ(4,*)N
C     SE VCITUVAAT VREDNOSTITE XI I YI NA JAZLITE
      DO 50 I = 1, N
```

```
      READ(4,*) X(I) , Q(I,1)
   5  0  CONTINUE
      CLOSE(UNIT=4

C     SE ISPISUVAAT VREDNOSTITE NA INTERPOLACIONITE JAZLI
      WRITE(OUP,3)
      WRITE(OUP,6)
      WRITE(OUP,4)  (X(I),Q(I,1),I=1,N)

C     SE PRESMETUVAAT KOEFICIENTITE NA INTERPOLACIONIOT POLINOM
      DO 11 I=2,N
      DO 21 J=2,I
   2  1  Q(I,J)=(Q(I,J-1)-Q(I-1,J-1))/(X(I)-X(I-J+1))
   1  1  CONTINUE

      WRITE(OUP,5)
      DO 60 I=1,N
      WRITE(OUP,7) Q(I,I)
   6  0  CONTINUE

      CALL NIP(N,Q,X,VAL)
      WRITE(OUP,8)
      WRITE(OUP,7) VAL

   4  0  0  CLOSE(UNIT=5)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.6) CLOSE(UNIT=6)
      STOP

   3  FORMAT(/,1X,'NJUTNOV INTERPOLACIONEN POLINOM')
   6  FORMAT(/,1X,'VLEZNI PODATOCI')
   4  FORMAT(1X,E15.8,1X,E15.8)
   5  FORMAT(/,1X,'NJUTNOVI KOEFICIENTI Q(0,0),...,Q(N,N)')
   7  FORMAT(1X,E15.8)
   8  FORMAT(/,1X,'INTERPOLIRANATA VREDNOST PN(X0) E:')
      END

      SUBROUTINE NIP(NX,A,X,S)
C     PROCEDURA ZA PRESMETUVANJE NA INTERPOLIRANA
C     VREDNOST VO DADENA TOCKA X0=1.5
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION A(25,25),X(25)
      S=0.D0
      X0=1.5D0
      DO 70 I = 2, NX-1
      AP=1.D0
      DO 80 J=1,I-1
      AP=AP*(X0-X(J))
   8  0  CONTINUE
      S=S+A(I,I)*AP
   7  0  CONTINUE
      S=S+A(1,1)
      RETURN
      END
```

Излезот на програмата за Њутнов интерполационен полином:

```
 NJUTNOV INTERPOLACIONEN POLINOM

 VLEZNI PODATOCI
  0.10000000E+01  0.76519770E+00
  0.13000000E+01  0.62008600E+00
  0.16000000E+01  0.45540220E+00
  0.19000000E+01  0.28181860E+00
  0.22000000E+01  0.11036230E+00

 NJUTNOVI KOEFICIENTI Q(0,0),...,Q(N,N)
  0.76519770E+00
 -0.48370567E+00
 -0.10873389E+00
  0.65878395E-01
  0.18251029E-02
```

```
INTERPOLIRANATA VREDNOST PN(X0) E:
 0.51181269E+00
```

4.3. Задачи

1. Да се дополни програмата njutnova_interpolacija.for со излез кој ќе ги прикажува конечните (диференцни) разлики од *n*-ти ред?

2. Користејќи го првиот Њутнов интерполационен полином и податоците во табелата да се пресмета вредноста $f(0.05)$ и $f(0.65)$?

x_i	$f(x_i)$
0.0	1.00000
0.2	1.22140
0.4	1.49182
0.6	1.82212
0.8	2.22554

Да се направи програма во која ќе се примени втората Њутнова интерполациона формула?

3. Да се воопшти програмата од примерот 4.1 за да се пресмета $f(1.5)$ за податоците дадени во табелата

x_i	$f(x_i)$
1.0	0.7651977
1.3	0.6200860
1.6	0.4554022
1.9	0.2818186
2.2	0.1103623

5. ПРИБЛИЖНО ДИФЕРЕНЦИРАЊЕ

Изводот на функцијата $f(x)$ што е зададена табеларно може да се добие со диференцирање на интерполациониот полином со кој се апроксимира приближно во даден интервал $x \in [x_0, x_n]$. Како што покажавме во претходното поглавје, полином кој поминува низ сите интерполациони јазли (x_i, y_i) е *првиот Њутнов интерполационен полином* кој што може да се запиши како:

$$P_{\mathrm{n}}(x) = \sum_{i=1}^{n} Q_{i,i} \prod_{j=1}^{i-1} (x_0 - x_j). \tag{5.1}$$

Оваа формула може да се искористи за пресметување извод на функцијата $f(x)$, според изразот:

$$P_{\mathrm{n}}'(x) = \sum_{i=1}^{n} Q_{i,i} \sum_{i=1}^{k} \left[\frac{\mathrm{d}f_i(x)}{\mathrm{d}x} \prod_{j \neq i} f_j(x) \right], \tag{5.2}$$

при што се користи:

$$\frac{\mathrm{d}}{\mathrm{d}x} \left[\prod_{i=1}^{k} f_i(x) \right] = \sum_{i=1}^{k} \left[\frac{\mathrm{d}f_i(x)}{\mathrm{d}x} \prod_{j \neq i} f_j(x) \right]. \tag{5.3}$$

Овој начин на пресметување на изводот може да биде неточен доколку функцијата $f(x)$ има екстреми поради кои и мали отстапување во апроксимацијата на функцијата $|f(x) - P_n(x)| < \varepsilon$, може да предизвикаат големи разлики во вредностите на изводите во дадени интерполациони јазли $|f'(x) - P_n'(x)|$ ε. Затоа, наместо интерполациони полиноми се препорачува користење на *формули за приближно диференцирање*.

***Пример* 5.1.**

Во програмата priblizno_diferenciranje.for, прикажана подолу, се пресметува првиот извод на функција зададена табеларно со користење на првиот Њутнов интерполационен полином. Табеларните вредности на функцијата се дадени во табелата

x_i	$f(x_i)$
1.8	10.889365
1.9	12.703199
2.0	14.778112
2.1	17.148957
2.2	19.855030

Вредноста на првиот извод за $x = 2.0$, е $P_4'(2.0) = 22.167168$.

```
PROGRAM PRIBLIZNO_DIFERENCIRANJE
IMPLICIT REAL*8 (A-H,P-Z)
DIMENSION X(20),Q(20,20)
```

```
      CHARACTER NAME1*30
      OPEN(UNIT=4,FILE='IZVOD.DAT',ACCESS='SEQUENTIAL')

      WRITE(*,*) 'Prv izovd na funkcijata f(x) so'
      WRITE(*,*) 'Njutnov interpolacionen polinom'
      WRITE(*,*) 'Odberi vid na izlez na podatoci: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Tekstualna datoteka '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      WRITE(*,*) ' '
      READ(*,*) FLAG
      IF ( FLAG .EQ. 2 ) THEN
      WRITE(*,*) 'Vnesi go imeto na datotekata kako - '
      WRITE(*,*) 'drive:ime.ext'
      WRITE(*,*) 'zadilzitelno staveni vo navodnici'
      WRITE(*,*) 'na primer:''C:/OUTPUT.DAt'' '
      WRITE(*,*) ' '
      READ(*,*) NAME1
      OUP = 3
      OPEN(UNIT=OUP,FILE=NAME1,STATUS='NEW')
      ELSE
      OUP = 6
      ENDIF

      WRITE(*,*)'Vnesete ja vrednosta na x kade se bara izvodot:'
      READ(*,*)A
C     SE VCITUVA BROJOT NA INTERPOLACIONI JAZLI
      READ(4,*)N
C     SE VCITUVAAT VREDNOSTITE XI I YI NA JAZLITE
      DO 50 I = 1, N
      READ(4,*) X(I) , Q(I,1)
   5  0 CONTINUE
      CLOSE(UNIT=4)
C     SE ISPISUVAAT VREDNOSTITE NA INTERPOLACIONITE JAZLI
      WRITE(OUP,3)
      WRITE(OUP,6)
      WRITE(OUP,4) (X(I),Q(I,1),I=1,N)

C     SE PRESMETUVAAT KOEFICIENTITE NA INTERPOLACIONIOT POLINOM Q
      DO 10 J=2,N
      DO 10 I=1,N-J+1
      Q(I,J)=(Q(I+1,J-1)-Q(I,J-1))/(X(I+J-1)-X(I))
   1  0 CONTINUE
      V=Q(1,2)

C     SE PRESMETUVAAT PRVIOT IZVOD PN'(X0)
      DO 20 I=3,N
      S=0
      DO 30 J=1,I-1
      P=1
      DO 40 K=1,I-1
      IF(K.NE.J) P=P*(A-X(K))
   4  0 CONTINUE
      S=S+P
   3  0 CONTINUE
      V=V+S*Q(1,I)
   2  0 CONTINUE
      WRITE(OUP,8)A,V
      WRITE(OUP,9)A

   3  FORMAT(/,1X,'NJUTNOV INTERPOLACIONEN POLINOM')
   6  FORMAT(/,1X,'VLEZNI PODATOCI')
   4  FORMAT(1X,E15.8,1X,E15.8)
   5  FORMAT(/,1X,'NJUTNOVI KOEFICIENTI Q(0,0),...,Q(N,N)')
   7  FORMAT(1X,E15.8)
   8  FORMAT(/,1X,'VREDNOSTA NA IZVODOT VO X= ',F12.8,' E :',F12.8)
   9  FORMAT(/,1X,'TOCNATA VREDNOSTA VO X=',F12.8,' E : 22.167168')
      END


      Излезот на програмата за приближно диференцирање е:
```

```
NJUTNOV INTERPOLACIONEN POLINOM

 VLEZNI PODATOCI
  0.18000000E+01  0.10889365E+02
  0.19000000E+01  0.12703199E+02
  0.20000000E+01  0.14778112E+02
  0.21000000E+01  0.17148957E+02
  0.22000000E+01  0.19855030E+02

 VREDNOSTA NA IZVODOT VO X=   2.00000000 E : 22.16699917

 TOCNATA VREDNOSTA VO X=  2.00000000 E : 22.167168
```

5.1. Задачи

1. Користејќи ја програмата за приближно диференцирање и податоците во табелите подолу да се пресметаат изводите во интерполационите јазли каде може да се примени првиот Њутнов интерполационен полином ?

x_i	$f(x_i)$	$f'(x_i)$
1.1	1.949477	
1.2	2.199796	
1.3	2.439189	
1.4	2.670324	

x_i	$f(x_i)$	$f'(x_i)$
2.1	-1.709847	
2.2	-1.373823	
2.3	-1.119214	
2.4	-0.9160143	
2.5	-0.7470223	
2.6	-0.6015966	

Да се објаснат добиените резултати?

6. ПРИБЛИЖНО ИНТЕГРИРАЊЕ

Ако некоја функција $f(x)$ е континуирана на интервалот $[a,b]$ и ако е позната нејзината примитивна функција која може да се изрази со помош на елементарни функции, тогаш може да се пресмета определениот интеграл на таа функција според Њутн-Лајбницовата формула

$$I(f)=\int_a^b f(x)dx=F(b)-F(a). \tag{6.1}$$

Меѓутоа, во многу случаи примената на горната формула не е можно, или пак постои примитивна функција $F(x)$ што е премногу комплицирана или непогодна за пресметување. Исто така, при решавање на многу реални практични проблеми, подинтегралната функција не е дадена со аналитички израз туку табеларно, така што формулата (6.1) не може ни да се примени. Значи, многу често пресметувањето на определен интеграл според формулата Њутн-Лајбниц е многу тешко или практично невозможно затоа интегралот треба да се пресмета приближно, што може да се прикажи како

$$I(f)=I_n(f)+E_n(f), \tag{6.2}$$

каде $I_n(f)$ е приближната вредност на интегралот а $E_n(f)$ е грешката при пресметката. Приближната вредност може да се пресмета со изразот

$$I_n(f)=\sum_{k=0}^{n}\omega_k^{(n)}f(x_k^{(n)}), \tag{6.3}$$

каде што n е цел број, $x_k^{(n)}$ се јазли на интеграција а $\omega_k^{(n)}$ се тежински функции што зависат од начинот на апроксимација на $f(x)$.

Суштината на приодот при нумеричкото интегрирање се состои во тоа, што подинтегралната функција $f(x)$ се заменува со некоја друга, поедноставна функција $\varphi(x)$ која има исти вредности како $f(x)$ во точките x_k, доволно е блиска до $f(x)$ и што е најважно, може лесно да се интегрира. Изборот на функцијата $\varphi(x)$ зависи од природата на задачата, но најчесто таа се избира да биде полином. Точноста, пак на резултатот за приближно интегрирање ќе зависи од тоа колку добро е апроксимирана подинтегралната функција со избраниот полином. Затоа, пред да се почне нумеричкото пресметување на еден интеграл, пожелно е да се утврди природата на подинтегралната функција и да се нацрта график, па потоа да почне пресметувањето по некоја од многуте формули за приближно интегрирање.

Ако имаме полиномна апроксимација на $f(x)$, приближната вредност на интегралот може да се прикажи како

$$I(f)\approx\int_a^b P_n(x)\mathrm{d}x=h\int_{t(a)}^{t(b)}P_n(t)\mathrm{d}t\,, \tag{6.4}$$

каде $x=x_0+th$ а h е чекор на интеграција. Ако избериме $x=a$, добиваме

$$I_n(f)=\int_a^b P_n(x)\mathrm{d}x=h\int_0^{(b-a)/h}P_n(x_0+th)\mathrm{d}t\,. \tag{6.5}$$

Во зависност од степенот на полиномот, $n=0,1,2\ldots$ се добиваат различни формули за приближно интегрирање познати како *формули на Њутон-Котес*.

6.1. Формула на трапез

Нека функцијата $f(x)$ е непрекината на интервалот $[a,b]$ и нека тој интервал е поделен на n еднакви подинтервали со должина $h=(b-a)/n$, при што

$$a=x_0<x_1<\cdots<x_n=b,\quad x_i=x_0+ih,\quad f(x_i)=y_i\,.$$

Ако за апроксимирање на оваа функција ја употребиме Њутновата интерполациона формула, ќе имаме $f(x)\approx P_n(x_0+th)$, односно

$$P_n(x_0+th)=y_0+t\Delta y_0+\frac{t(t-1)}{2!}\Delta^2y_0+\ldots+\frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0\,. \tag{6.6}$$

Со замена на (6.6) во (6.5) добиваме

$$\int_a^b f(x)\mathrm{d}x\approx\int_{x_0}^{x_0+nh}P_n(x_0+th)\mathrm{d}x=h\int_0^n\left(y_0+t\Delta y_0+\frac{t(t-1)}{2!}\Delta^2y_0+\ldots\right)\mathrm{d}t$$

или

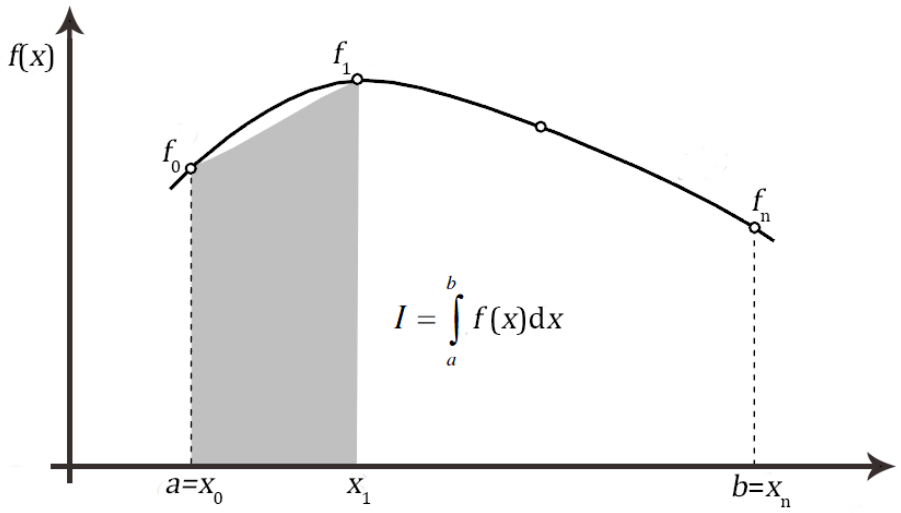
$$\int_a^b f(x)\mathrm{d}x\approx h\left[ny_0+\frac{n^2}{2}\Delta y_0+\left(\frac{n^3}{3}-\frac{n^2}{2}\right)\frac{\Delta^2y_0}{2!}+\left(\frac{n^4}{3}-n^3+n^2\right)\frac{\Delta^3y_0}{3!}+\ldots\right] \tag{6.7}$$

Од оваа формула можат да се добијат разни приближни формули за пресметување определени интеграли во зависност од тоа на колку поделоци е поделен интервалот $[a,b]$. Така, ако во (6.7) замениме $n=1$, и се задржиме до конечните разлики од прв степен, се добива формулата

$$\int_{x_0}^{x_0+h}f(x)\mathrm{d}x\approx h\left[y_0+\frac{1}{2}\Delta y_0\right]=h\frac{y_0+y_1}{2}\,, \tag{6.8}$$

Каде $h=b-a,\quad y_0=f(x_0),\quad y_1=f(x_1)$.

Геометриски, оваа формула значи дека плоштината на криволинискиот трапез е приближно еднаква со плоштината на обичен трапез, па затоа оваа формула се нарекува **правило на трапези**, слика 6.1.



Сл. 6.1. Графички приказ на пресметување на интеграл со формула на трапез.

Ако интервалот на интеграција $[a<x_i<b]$ се подели на n подинтервали

$$[x_0, x_0+h],[x_0+h, x_0+2h],\ldots,[x_0+(n-1)h, x_0+nh],$$

при што $x_0=a$ а $h=(b-a)/n$ и на секој од нив се примени формулата (6.8), се добива изразот

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f(x)dx = h\left[\frac{y_0+y_n}{2}+y_1+y_2+\ldots+y_{n-1}\right], \tag{6.9}$$

познат како *продолжена формула (правило) на трапез*, која може да се презапиши и како

$$I \approx \frac{b-a}{2n}\left[f_0+f_n+2\sum_{i=1}^{n-1}f_i\right]. \tag{6.10}$$

Пример 6.1.

Во програмата formula_na_trapez.for, прикажана подолу, се пресметува приближната вредност на интегралот

$$\int_0^{\pi/2} x^2(x^2-2)\sin(x)\mathrm{d}x = -0.4791589. \tag{6.11}$$

```
PROGRAM FORMULA_NA_TRAPEZ

IMPLICIT REAL*8 (A-H,P-Z)
PARAMETER(PIO2=1.5707963)
EXTERNAL FUNC
CHARACTER NAME1*30

WRITE(*,*)'Integriranje na funkcija so metod na trapez'
WRITE(*,*) 'Odberete vid na izlez: '
WRITE(*,*) '1. Ekran '
WRITE(*,*) '2. Text file '
WRITE(*,*) 'Vnesi 1 ili 2 '
READ(*,*) FLAG
```

```
      WRITE(*,*) ' '

      IF ( FLAG .EQ. 2 ) THEN
         WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
         WRITE(*,*) 'drive:name.ext'
         WRITE(*,*) 'patot da bide staven vo navodnici'
         WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
         READ (*,*) NAME1
         WRITE(*,*) ' '
         OUP = 3
         OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
         OUP = 6
      END IF

      A=0.0
      B=PIO2
      JMAX=1500
      EXACT=FINT(B)-FINT(A)

      WRITE(OUP,3)
      WRITE(OUP,4)EXACT

      DO J=4,JMAX
      CALL TRAPZD(FUNC,A,B,S,J)
      IF(MOD(J,50).EQ.0.D0)THEN
      WRITE(OUP,5)S,J
      ENDIF
      END DO

      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP

    3 FORMAT(/,1X,'Integriranje na funkcija so metod na trapez')
    4 FORMAT(/,1X,'Tocna vrednost na integralot =',E14.8)
    5 FORMAT(/,1X,'Priblizna vrednost na integralot e ',E14.8,
     *' posle ',I5,' iteracii')

      END

      REAL FUNCTION FUNC(X)
C     FUNKCIJA STO SE INTEGRIRA
      REAL X
      FUNC=(X**2)*(X**2-2.0)*SIN(X)
      END

      REAL FUNCTION FINT(X)
C     INTEGRAL NA FUNKCIJATA
      REAL X
      FINT=4.0*X*((X**2)-7.0)*SIN(X)-((X**4)-14.0*(X**2)+28.0)*COS(X)
      END

      SUBROUTINE TRAPZD(FUNC,A,B,S,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FUNC

      DEL=(B-A)/N
      S0=0.5D0*DEL*(FUNC(A)+FUNC(B))
      X=A+DEL
      SUM=0.D0
      DO J=1,N-1
         SUM=SUM+FUNC(X)
         X=X+DEL
      END DO
      S=S0+DEL*SUM
      RETURN
      END
```

Дел од излезот на програмата за приближно интегрирање:

```
Integriranje na funkcija so metod na trapez

Tocna vrednost na integralot =-.47915884E+00

Priblizna vrednost na integralot e -.47840056E+00 posle    50 iteracii

Priblizna vrednost na integralot e -.47896927E+00 posle   100 iteracii
...
...
...
Priblizna vrednost na integralot e -.47915787E+00 posle  1400 iteracii

Priblizna vrednost na integralot e -.47915794E+00 posle  1450 iteracii

Priblizna vrednost na integralot e -.47915800E+00 posle  1500 iteracii
```

Може да се забележи дека после N=1500 итерации точната и пресметаната вредност на интегралот се совпаѓаат на шест децимали. Понатамошното зголемување на бројот на итерации не е оправдано затоа што е очигледно дека се троши големо компјутерско време. Тоа ни кажува дека оваа формула за приближно интегрирање во суштина има бавна конвергенција и не се препорачува за користење.

6.2. Фромула на Симпсон

Формулата на Симпсон 1/3 или *правило на параболи*, како уште се вика, се добива кога во формулата (6.7) ќе замениме $n=2$ и ќе се задржиме до конечните разлики од втор степен,

$$\begin{aligned}\int_{x_0}^{x_0+2h} f(x)\mathrm{d}x &\approx h\left[2y_0+2\Delta y_0+\frac{2}{3}\frac{\Delta^2 y_0}{2}\right]= \\ &= h\left[2y_0+2y_1-2y_0+\frac{1}{3}\left(y_2-2y_1+y_0\right)\right],\end{aligned} \tag{6.12}$$

или

$$\int_{x_0}^{x_0+2h} f\left(x\right)\mathrm{d}x \approx \frac{h}{3}\left(y_0+4y_1+y_2\right), \tag{6.13}$$

каде $h=(b-a)/n=(b-a)/2$. За да ја добиеме проширената форма на формулата на Симпсон 1/3, треба интервалот на интеграција $[a,b]$ да го поделиме на $2n$ подинтервали, со што добиваме

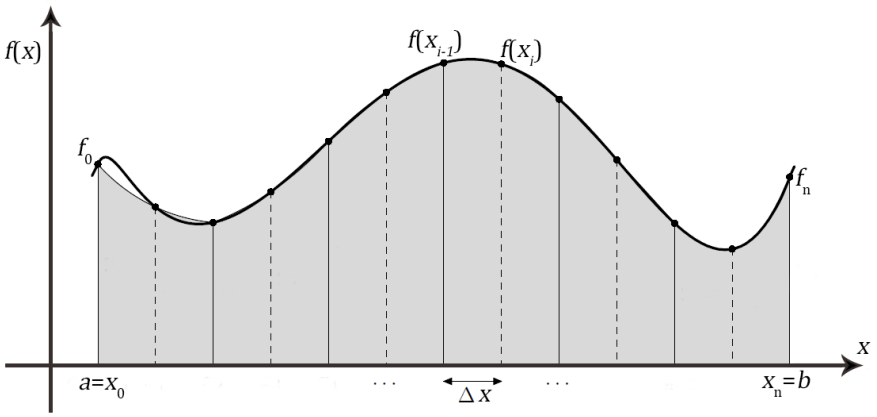
$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &\approx \sum_{i=1}^{n/2}\int_{x_{2i-2}}^{x_{2i}} f(x)\mathrm{d}x = \\ &= \frac{h}{3}\left[\left(y_0+4y_1+y_2\right)+\left(y_2+4y_3+y_4\right)\ldots+\left(y_{2n-2}+4y_{n-1}+y_n\right)\right],\end{aligned} \tag{6.14}$$

што во концизна форма може да се запиши како

$$I_n=\frac{b-a}{6n}\left[f_0+f_{2n}+4\sum_{i=0}^{n-1}f_{2i+1}+2\sum_{i=1}^{n-1}f_{2i}\right]. \tag{6.15}$$

при што предвид е земено дека $h=(b-a)/2n$.

Со тоа, функцијата $f(x)$ во три соседни еквидистантни точки од интервалот на интеграција се апроксимираат со плоним од втор степен (парабола). Графички, интегрирањето со формулата на Симпсон 1/3 може да се прикажи како на слика 6.2.



Сл. 6.2. Графички приказ на пресметување на интеграл со формула на Симпсон.

За сите досега изнесени формули за приближно интегрирање постојат и формули за оценка на погрешноста со која е пресметан бараниот интеграл, но тие се доста комплицирани и во многу случаи практично неприменливи. Поради тоа се применува следново практично правило. Чекорот h се избира со груба проценка, а потоа, откако ќе се добие резултатот, бројот n се удвојува, т.е. чекорот се преполовува. Ако новиот резултат се совпаѓа со претходниот во децималите кои ги задржуваме и од кои можеме да ја процениме бараната погрешност, тогаш пресметувањето завршува. Во спротивно, оваа постапка се повторува.

Пример 6.2.

Во програмата formula_na_simpson.for, прикажана подолу, се пресметува приближната вредност на интегралот (6.11) прикажан претходно.

```
PROGRAM FORMULA_NA_TRAPEZ

IMPLICIT REAL*8 (A-H,P-Z)
PARAMETER(PIO2=1.5707963)
EXTERNAL FUNC, FINT
CHARACTER NAME1*30

WRITE(*,*)'Integriranje na funkcija so metod na Simpson'
WRITE(*,*) 'Odberete vid na izlez: '
WRITE(*,*) '1. Ekran '
WRITE(*,*) '2. Text file '
WRITE(*,*) 'Vnesi 1 ili 2 '
READ(*,*) FLAG
WRITE(*,*) ' '

IF ( FLAG .EQ. 2 ) THEN
   WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
   WRITE(*,*) 'drive:name.ext'
   WRITE(*,*) 'patot da bide staven vo navodnici'
   WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
   READ (*,*) NAME1
   WRITE(*,*) ' '
```

```
      OUP = 3
      OPEN(UNIT=OUP,FILE=NAME1)
   ELSE
      OUP = 6
   END IF

   A=0.0
   B=PIO2
   JMAX=70
   EXACT=FINT(B)-FINT(A)

   WRITE(OUP,3)
   WRITE(OUP,4)EXACT

   DO J=4,JMAX,2
   CALL SIMP(FUNC,A,B,S,J)
   IF(MOD(J,2).EQ.0.D0)THEN
   WRITE(OUP,5)S,J
   ENDIF
   END DO

   CLOSE(UNIT=OUP)
   IF(OUP.NE.OUP) CLOSE(UNIT=6)
   STOP

 3 FORMAT(/,1X,'Integriranje na funkcija so metod na trapez')
 4 FORMAT(/,1X,'Tocna vrednost na integralot =',E14.8)
 5 FORMAT(/,1X,'Priblizna vrednost na integralot e ',E14.8,
  * ' posle ',I5,' iteracii')

   END

   REAL FUNCTION FUNC(X)
C  FUNKCIJA STO SE INTEGRIRA
   REAL X
   FUNC=(X**2)*(X**2-2.0)*SIN(X)
   END

   REAL FUNCTION FINT(X)
C  INTEGRAL NA FUNKCIJATA
   REAL X
   FINT=4.0*X*((X**2)-7.0)*SIN(X)-((X**4)-14.0*(X**2)+28.0)*COS(X)
   END

   SUBROUTINE TRAPZD(FUNC,A,B,S,N)
   IMPLICIT REAL*8 (A-H,O-Z)
   EXTERNAL FUNC

C  CEKOR NA INTEGRACIJA
   DEL=(B-A)/N
   XI0=FUNC(A)+FUNC(B)
C  SUMIRANJE NA F(X(2*I-1))
   XI1 = 0.0D0
C  SUMIRANJE NA F(X(2*I))
   XI2 = 0.0D0

   MM=N-1
   DO 20 I=1,MM
   X=A+I*H
   IF(I.EQ.2*(I/2))THEN
   XI2=XI2+FUNC(X)
   ELSE
   XI1=XI1+FUNC(X)
   END IF
 2 0 CONTINUE
   XI=XI0+2.D0*XI2+4.D0*XI1
   S=XI*H/3.D0
   RETURN
   END
```

Дел од излезот на програмата за приближно интегрирање:

```
Integriranje na funkcija so formula na Simpson

Tocna vrednost na integralot =-.47915884E+00

Priblizna vrednost na integralot e -.47496676E+00 posle     4 iteracii

Priblizna vrednost na integralot e -.47847573E+00 posle     6 iteracii

Priblizna vrednost na integralot e -.47895791E+00 posle     8 iteracii

Priblizna vrednost na integralot e -.47907937E+00 posle    10 iteracii
...
...
Priblizna vrednost na integralot e -.47915880E+00 posle    66 iteracii

Priblizna vrednost na integralot e -.47915881E+00 posle    68 iteracii

Priblizna vrednost na integralot e -.47915881E+00 posle    70 iteracii
```

Излезните податоци добиени со примена на формулата на Симпсон ни покажуваат дека оваа формула конвергира многу побргу во однос на формулата на трапези, затоа точната вредност на интегралот се достигнува за само 70-на итерации.

6.3. Гаусова квадратурна формула

Во формулите за приближно интегрирање што ги спомнавме претходно, јазлите за интеграција се поставени еквидинстантно. Во општ случај, тие не мора да се еквидистанти, туку по потреба може да се определат. Ако ја земиме предвид формулата (6.3), може да заклучиме дека за пресметување на вредноста на интегралот треба да определиме n јазли на интеграција и n тежински функции. Задачата за нивно определување, а со тоа и интегрирање е позната како квадратурна формула на Гаус, која може да се запише како

$$\int_a^b \omega(x) f(x) dx \approx \sum_{i=1}^{n} \omega_i f(x_i), \tag{6.16}$$

каде $\omega(x)$ е тежинска функција, x_i се јазли на интеграција а ω_i се тежински коефициенти кои зависат и од тежинската функција. Оваа формула е точна за сите полиноми функции $f(x_i)$ до степен $2n+1$. Во табелата што следи дадени се некои од најчесто применуваните Гаусови квадратурни формули.

Тежинска функција	Интервал на интеграција	Квадратурна формула
1	[-1,1]	Гаус-Лежандр
$\frac{1}{\sqrt{1-x^2}}$	[-1,1]	Гаус-Чебишев
e^{-x}	[0,∞)	Гаус-Лагер
e^{-x^2}	(-∞,∞)	Гаус-Ермит

Главната придобивка од ова е дека ако формулата (6.16) точно интегрира полиномни функции од што е можно поголем степен, тогаш јазлите на интеграција x_i се нули на полиноми што се ортогонални на интервалот $[a,b]$. Притоа, тежинските коефициенти се поврзани со тежинските функции преку формулата

$$\omega_i = \int_a^b \omega(x) L_i(x) \mathrm{d}x, \qquad i = 1,2,\ldots,n . \tag{6.17}$$

каде $L_i(x)$ е Лагранжов интерполационен полином за кој важи $L_i(x_j) = \delta_{ij}$. За тежинските функции важи следнава формула

$$\int_a^b x^k q(x) \omega(x) \mathrm{d}x = 0, \qquad k = 0,1,\ldots n, \tag{6.18}$$

каде $q(x)$ е полином од степен $n+1$, при што ако јазлите на интеграција x_i се нули на $q(x)$ тогаш важи квадратурната формула (6.16). Ако за пример земиме $\omega(x) = 1$, а интервалот на интеграција е $[-1 \le x \le 1]$, станува збор за Гаус-Лежандрова квадратурна формула која гласи

$$\int_{-1}^{1} f(x) dx \approx \sum_{i=1}^{n} \omega_i f(x_i) . \tag{6.19}$$

Некои од поважните својства на Лежандровите полиноми што се важни за Гаусовата квадратурна формула се спомнати во текстот што следи. Како прво, Лежандровите полиноми од $n-$ти степен се дефинираат со формулата на Родригез

$$P_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} (x^2 - 1)^n . \tag{6.20}$$

Карактеристично за нив е дека тие образуваат ортогонална база во полиномен простор од $n-$ти степен за која важи

$$\int_{-1}^{1} P_m(x) P_n(x) \mathrm{d}x = 0, \qquad m \ne n \tag{6.21}$$

а нормата им се пресметува како

$$\left\| P_n \right\|^2 = \int_{-1}^{1} [P_n(x)]^2 \mathrm{d}x = \frac{2}{2n+1} . \tag{6.22}$$

Исто така, во согласност со формулата (6.18), Лежандровите полиноми се ортогонални со сите степени функции x^k од понизок степен, $k = 1,2,\ldots n-1$, при што важи

$$\int_{-1}^{1} x^k P_n(x) \mathrm{d}x = \begin{cases} 0 & k = 0,1,2,\ldots n-1 \\ \dfrac{2^{n+1} (n!)^2}{(2n+1)!} & k = n \end{cases} . \tag{6.23}$$

Тежинските коефициенти на Гаус-Лежандровата квадратурна формула (6.19) се пресметуваат со изразот

$$\omega_i = \int_{-1}^{1} L_i(x)\mathrm{d}x = \frac{2(1-x_i^2)}{[nP_{n-1}(x_i)]^2} = \frac{2}{(1-x_i^2)[P_n'(x_i)]^2}. \tag{6.24}$$

За практична употреба на Гаусовата квадратурна формула се препорачува тежинските функции како и јазлите за интеграција да не се пресметуваат туку да се користат готови табели, како на пример табела 6.1.

Број на јазли	Тежински коефициенти	Нули на Лежандровите полиноми од n-ти степен
2	$\omega_1 = 1.000000000$ $\omega_2 = 1.000000000$	$x_1 = -0.577350269$ $x_2 = 0.577350269$
3	$\omega_1 = 0.555555556$ $\omega_2 = 0.888888889$ $\omega_3 = 0.555555556$	$x_1 = -0.774596669$ $x_2 = 0.000000000$ $x_3 = 0.774596669$
4	$\omega_1 = 0.347854845$ $\omega_2 = 0.652145155$ $\omega_3 = 0.652145155$ $\omega_4 = 0.347854845$	$x_1 = -0.861136312$ $x_2 = -0.339981044$ $x_3 = 0.339981044$ $x_4 = 0.861136312$
5	$\omega_1 = 0.236926885$ $\omega_2 = 0.478628670$ $\omega_3 = 0.568888889$ $\omega_4 = 0.478628670$ $\omega_5 = 0.236926885$	$x_1 = -0.906179846$ $x_2 = -0.538469310$ $x_3 = 0.000000000$ $x_4 = 0.538469310$ $x_5 = 0.906179846$
6	$\omega_1 = 0.171324492$ $\omega_2 = 0.360761573$ $\omega_3 = 0.467913935$ $\omega_4 = 0.467913935$ $\omega_5 = 0.360761573$ $\omega_6 = 0.171324492$	$x_1 = -0.932469514$ $x_2 = -0.661209386$ $x_3 = -0.238619186$ $x_4 = 0.238619186$ $x_5 = 0.661209386$ $x_6 = 0.932469514$

Пример 6.3.

Како се применува Гаус-Лежандровата квадратурна формула ќе покажиме со следниов пример во кој треба да се пресмета интегралот

$$\int_a^b \sqrt{1+x}\,dx, \tag{6.25}$$

каде $a = 0$, а $b \in [0.5,5]$. Веднаш може да забележиме дека во нашиот пример границите на интеграција не се симетрични, па затоа неопходно е да се направи замена

$$x = \frac{b-a}{2}y + \frac{a+b}{2}, \tag{6.26}$$

со што интегралот, во општа форма, може да се запиши како

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{a+b}{2}+\frac{b-a}{2}y\right)dy \approx \sum_{i=1}^{n}\omega_i f\left(\frac{a+b}{2}+\frac{b-a}{2}y_i\right), \qquad (6.27)$$

каде јазлите за интеграција y_i и соодветните тежински коефициенти ω_i се дадени во табела 6.1. Со програмата gausova_integracija.for, се пресметува приближната вредност на интегралот (6.25) во случаите кога $a=0$, и $b\in[0.5,5]$.

```
      PROGRAM METOD_NA_GAUS

      IMPLICIT REAL*8(A-N,P-Z)
      PARAMETER(X1=0.0,X2=5.0,NVAL=10)
      EXTERNAL FUNC
      CHARACTER NAME1*30

      WRITE(*,*)'Priblizno integriranje so metod na Gaus'
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF ( NFLAG .EQ. 2 ) THEN
         WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
         WRITE(*,*) 'drive:name.ext'
         WRITE(*,*) 'patot da bide staven vo navodnici'
         WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
         READ (*,*) NAME1
         WRITE(*,*) ' '
         OUP = 3
         OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
         OUP = 6
      END IF

      DX=(X2-X1)/NVAL
      WRITE(OUP,'(3X,A,5X,A,6X,A,7X,A/)') 'X',' PRIBLIZNA',
     * ' TOCNA',' RAZLIKA'
      DO I=1,NVAL
         X=X1+I*DX
         TOCNA=(2.D0/3.D0)*(1.D0+X)**(3.D0/2.D0)-
     *    (2.D0/3.D0
         CALL QGAUS(FUNC,X1,X,SS)
         WRITE(OUP,'(1X,F5.2,3(2X,F12.9))') X,SS,TOCNA,TOCNA-SS
      END DO

      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END

C     DEFINIRANJE NA FUNKCIJATA ZA INGRIRANJE
      REAL*8 FUNCTION FUNC(X)
      REAL*8 X
      FUNC=DSQRT(1.D0+X)
      END

C     POTPROGRAMA ZA PRIMENA NA GAUSOVA KVADRATURNA FORMULA

      SUBROUTINE QGAUS(FUNC,A,B,SS)
      IMPLICIT REAL*8(A-H,P-Z
      EXTERNAL FUNC
      DIMENSION XR,W(5),X(5)
      SAVE W,X

C     TEZINSKI KOEFICIENTI
      DATA W/.2955242247,.2692667193,.2190863625,.1494513491,
     * .0666713443/
C     JAZLI NA INTEGRACIJA
      DATA X/.1488743389,.4333953941,.6794095682,.8650633666,
```

```
* .9739065285/

XM=0.5*(B+A)
XR=0.5*(B-A)
SS=0
DO J=1,5
   DX=XR*X(J)
   SS=SS+W(J)*(FUNC(XM+DX)+FUNC(XM-DX))
END DO
SS=XR*SS
RETURN
END


Излезот на програмата за приближно интегрирање со Гаус-Лежандрова
квадратура:

   X      PRIBLIZNA       TOCNA        RAZLIKA

  0.50   0.558078213   0.558078205  -0.000000008
  1.00   1.218951435   1.218951416  -0.000000018
  1.50   1.968564747   1.968564717  -0.000000030
  2.00   2.797434991   2.797434948  -0.000000043
  2.50   3.698600342   3.698600285  -0.000000057
  3.00   4.666666739   4.666666667  -0.000000073
  3.50   5.697294453   5.697294364  -0.000000089
  4.00   6.786893365   6.786893258  -0.000000107
  4.50   7.932429019   7.932428893  -0.000000126
  5.00   9.131292450   9.131292304  -0.000000146
```

Може да се забележи дека со „само“ пет јазли на интеграција, $n=5$, Гаус-Лежандровата квадратурна формула достигнува прецизност од најмалку 10^{-6}!

6.4. Неправи интеграли

При примена на формулите за нумеричка интеграција треба да се внимава на обликот на подинтегралната функција и на фактот дали има сингуларности. На пример ако имаме интеграл со голема горна граница, $b\gg 1$, најдобро е да извршиме соодветна смена на променливите. Така, ако треба приближно да го пресметаме интегралот

$$I=\int_1^b x^{-2}g(x)\mathrm{d}x\,, \tag{6.28}$$

со константно $g(x)$ за големи x, тогаш сумата ќе конвергира многу бавно ако b е големо и тоа ќе бара многу компјутерско верме. Меѓутоа, ако извршиме смена $t=1/x$ ќе имаме интеграл

$$I=\int_{1/b}^1 g\left(\frac{1}{t}\right)\mathrm{d}t\,,$$

за чие пресметување ќе треба многу пократко време.

Ако претпоставиме дека имаме неправ интеграл, во кој областа на интегрирање се простира до бесконечност, како на пример

$$I=\int_0^\infty x^2\mathrm{e}^{-x}\mathrm{d}x\,, \tag{6.29}$$

лесно може погрешно да заклучиме да интегралот дивергира, $I \to \infty$, што се разбира дека во овој случај не е точно бидејќи $I = 2$. Ако сакаме да примениме некој од методите за приблилжно интегрирање со еквидистантни јазли на интеграција, доаѓаме до практичен проблем за неможност на дефинирање на горната граница, $b \to \infty$. Затоа, препорачливо е областа на интегрирање да ја поделиме на два дела со што добиваме $I = I_1 + I_2$, или

$$\int_0^\infty f(x)\mathrm{d}x = \int_0^a f(x)dx + \int_a^\infty f(x)dx . \tag{6.30}$$

Со ова, првиот интеграл се сведува на определен интеграл во конечни граници кој може да се реши со некој од постоечките методи, а вториот интеграл може да се реши со смената што ја направивме со интегралот (6.28). Во праксата, честопати се применува поедноставен приод со кој горната граница се заменува со некој конечно голем број n_1 и се забележува вредноста на интегралот $I(n_1)$. Потоа се зголемува вредноста на горната граница $n_2 > n_1$ и се споредува вредноста на интегралот $I(n_2)$ со претходната. Ако, вредноста на интегралот се менува во помали граници од посакуваната прецизност, тогаш добиената вредност $I(n_1)$ може да се земи за конечна вредност.

Ако во интегралот има сингуларност, треба да се внимава дали сингуларноста е на некоја од границите на интеграција или некаде између. Ако сингуларноста е на граница на интегрирање, како на пример

$$\int_0^1 \frac{1}{\sqrt{1-x^2}} g\left(x\right)\mathrm{d}x , \tag{6.31},$$

таа може лесно да биде отстранета со смената $t = \sqrt{1-x}$ со која се добива интеграл од облик

$$2\int_0^1 \frac{1}{\sqrt{2-t^2}} g\left(1-t^2\right)\mathrm{d}t ,$$

кој се пресметува без тешкотии.

6.5. Задачи

1. Со формулите на трапез и Симпсон да се пресмета вредноста на интегралот

$$\int_0^\pi \sin(x)dx ,$$

за $n = 8,16,32,\ldots 1024$ јазли на интеграција, а потоа да се спореди прецизноста на методите?

2. Користејќи ги формулите на трапез и Симпсон да се пресмета за колку јазли, $n_1 = ?$, вредноста на интегралот

$$\int_{-1}^1 \sqrt{1-x^2}\mathrm{d}x ,$$

достигнува точност 10^{-3}. Потоа, ако ја направиме смената $x=\cos(x)$ да се определи со колкав број на јазли се постигнива истата точност?

3. Користејќи произволен метод да се пресмета приближната вредност на интегралот

$$\int_0^2 \frac{dx}{\sqrt{x}(1+x)},$$

користејќи замена на променливите или избегнување на сингуларноста. Да се спореди точноста на овие два приода?

4. Користејќи ја Гаус-Лежандровата квадратуна формула со пет јазли, $n=5$, да се пресмета вредноста на интегралот

$$\int_{-1}^{1} x^m \mathrm{d}x,$$

за $m=0,1,\ldots 9$. Да се коментираат добиените резултати?

5. Користејќи ја Гаус-Лежандровата квадратурна формула, за $n=4$, да се пресмета интегралот

$$\int_0^1 e^{-x^2} \mathrm{d}x.$$

Потоа, да се пресмета истиот интеграл со некој друг метод и да се потенцира што треба да се направи за да добиеме точност од 6 значни цифри во овој случај?

6.6. Проекти

Да се изведи Гаус-Лагеровата квадратурна формула? Потоа, со нејзина помош да се пресмета вредноста на еден од интегралите што се среќаваат при изучувањето на зрачењето на црно тело

$$\int_0^\infty \frac{x^3}{e^x-1} \mathrm{d}x.$$

Да се пресмета неговата вредност со користење на Гаус-Лагеровата квадратурна формула за $n=2,4,6,8$ јазли на интерација?

Корисни линкови:

http://www.efunda.com/math/num_integration/num_int_gauss.cfm

http://keisan.casio.com/

7. ПРИБЛИЖНО РЕШАВАЊЕ ОБИЧНИ ДИФЕРЕНЦИЈАЛНИ РАВЕНКИ

Многу од законите во физиката најчесто се изразени во облик на диференцијални равенки, па оттука произлегува и фактот што нумеричкото решавање на диференцијалните равенки е една од најчестите и најважните задачи при моделирање на физичките процеси. Овде ќе ги разгледаме методите за приближно решавање на обични диференцијални равенки од облик

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f\left(x, y\right) \tag{7.1}$$

каде x е независно променлива, а y е зависно променлива. Методите за решавање на ваква диференцијална равенка од прв ред лесно можат да се уопштат за систем диференцијални равенки од прв ред. Диференцијанлинте равенки од повисок ред можат да се сведат на равенки од прв ред со воведување на помошни функции. Така, основната равенка на динамиката за еднодимензионално движење на честица со маса m под дејство на сила $F(t)$ се опишува со диференцијална равенка од втор ред

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = F\left(t\right). \tag{7.2}$$

Воведувајќи го импулсот на честичката

$$p(t) = m\frac{\mathrm{d}x}{\mathrm{d}t},$$

равенката (7.2) се сведува на систем од две диференцијални равенки од прв ред

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{p}{m} \qquad \frac{\mathrm{d}p}{\mathrm{d}t} = F\left(t\right). \tag{7.3}$$

кои се од облик (7.1).

Во зависност од тоа какви гранични услови имаме, разликуваме два вида на диференцијални равенки:

1. *Диференцијални равенки со почетни вредности* се оние кај кои граничните услови се дадени само за една вредност на независно променливата. Пример за тоа се диференцијанлите равенки од облик (7.1) чие решение $y(x)$ се добива со задавање на почетна вредност $y_0 = y(x_0)$. Постапката за изнаоѓање на решенијата, во овој случај, е позната како решавање на *проблем на Коши* или *проблем на почетни вредности*.

2. *Диференцијални равенки со гранични вредности* се оние кај кои граничните услови се дадени за две или повеќе вредности на независно променливата, на пример:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + P(x,y)\frac{\mathrm{d}y}{\mathrm{d}t} + Q(x,y)y(t) = F(y,t)$$
$$y(x_1) = y_1 \qquad y(x_2) = y_2.$$

Постапката за решавање на овие задачи е позната како *проблем на гранични услови*.

Во ова поглавје ќе се запознаеме со нумеричките методи за решавање на проблемот на почетни вредности, што како задача најчесто се среќава при изучување на кинематиката и динамиката на тела во движење.

7.1. Метод на Ојлер

Методот на Ојлер е еден од наједноставните за нумеричките решавање на проблемот на почетни вредности а како идеја се користи како основна за развој на други нумерички методи. Постојат неколку методи на Ојлер, од кои најзастапени се следниве два: *експлицитен* и *имплицитен*.

7.1.1. Експлицитен метод на Ојлер

За да видиме во што е суштината на овој метод, да ја запишиме диференцијалната равенка (7.1) во точката $x_i = x_0 + ih$, т.е.

$$y'(x_i) = f(x_i, y_i), \tag{7.4}$$

со почетен услов $y_0 = y(x_0)$. Основната идеја се состои во тоа изводот да се замени со количник на првите разлики, т.е.

$$y' = \frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{\Delta y}{\Delta x}, \tag{7.5}$$

со што равенката (7.4) може да се запише како

$$\frac{y_{i+1} - y_i}{h} \approx f(x_i, y_i), \tag{7.6}$$

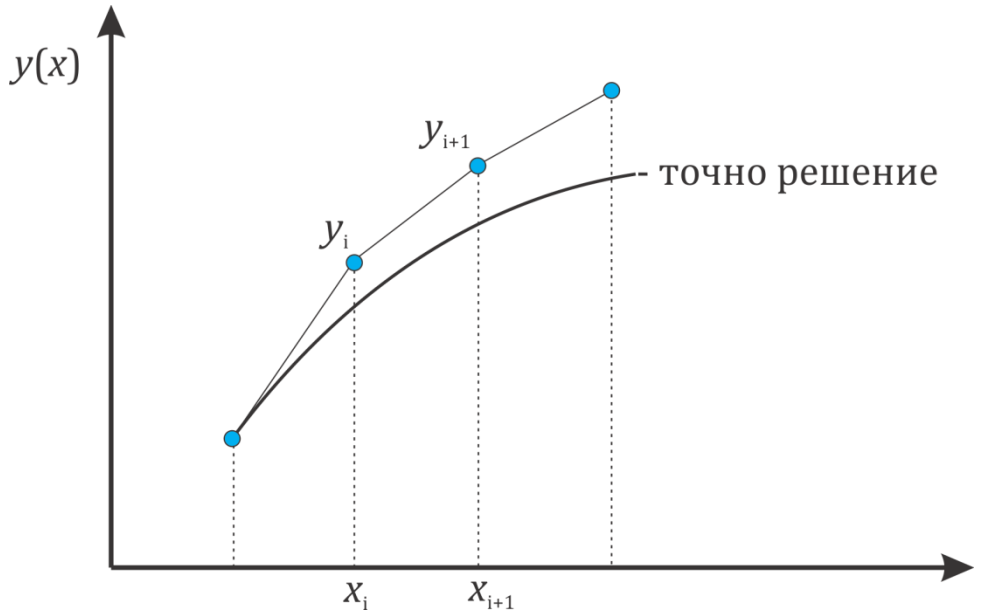
каде h е чекорот, односно $h = x_{i+1} - x_i$ а $y_i = y_i(x_i)$, $y_{i+1} = y_{i+1}(x_i + h)$. За да ја оцениме грешката при оваа апроксимација ќе напишиме

$$\frac{y_{i+1} - y_i}{h} + O(h) = f(x_i, y_i) \tag{7.7}$$

каде што со $O(h)$ е претставен остаточниот член пропорционален на h. Од (7.7) ја добиваме рекурзивната релација за пресметување на y_{i+1} која гласи,

$$y_{i+1} = y_i + hf(x_i, y_i) + O(h^2). \tag{7.8}$$

Ова формула има „локална грешка“ (т.е. грешка што се прави при еден чекор од y_i до y_{i+1}) пропорционална на h^2, односно $O(h^2)$ а „глобалната грешка” направена при пресметување на y во интервалот $[x_0, x_n]$ со n чекори изнесува $nO\left(h^2\right) \approx O\left(h\right)$ т.е. таа е линеарна со чекорот h. Геометриски, методот на Ојлер покажува дека интегралната крива $y = F(x)$ е заменета со искршена линија која почнува од заедничка точка (x_0, y_0) со кривата $y = F(x)$, а секое „ребро“ од таа искршена линија е паралелно со тангентата во левата крајна точка од соодветниот интервал, слика 7.1.



Сл. 7.1. Графички приказ на апроксимациите на експлицитниот метод на Ојлер.

Како што може да се види, методот на Ојлер е лесно сфатлив и многу едноставен за програмирање, но тој дава груби резултати и се смата за *нумерички условно стабилен*. А за еден алгоритам велиме дека е нумерички стабилен кога грешките на заокружување што се прават при изведување на големиот број операции не влијаат на крајниот резултат.

7.1.2. Имплицитен метод на Ојлер

Формулата за имплицитна шема на Ојлер се добива ако во равенка (7.6) изводот го пресметаме наназад,

$$\frac{y_i - y_{i-1}}{h} \approx f\left(x_i, y_i\right), \tag{7.9}$$

каде чекорот $h = x_i - x_{i-1}$. Од тука, ја добиваме рекурзивната релација за пресметување на y_i која гласи,

$$y_i = y_{i-1} + hf\left(x_i, y_i\right),$$

што може да се презапише и како

$$y_{i+1} = y_i + hf\left(x_{i+1}, y_{i+1}\right) + O(h^2). \tag{7.10}$$

И оваа формула има „локална грешка“ пропорционална на h^2, односно $O(h^2)$ поради што и глобалната грешка изнесува $\approx O(h)$. Сега, би рекле зошто ни е оваа шема ако има иста прецизност како и експлицитната? Во прашање е стабилноста на методот, т.е. имплицитната шема е значително постабилна во однос на експлицитната што значи конвергира кон точното решение за „речиси“ произволни вредности на чекорот на интегрирање h. Но, треба да се кажи и дека определувањето на y_{i+1} со имплицитната шема побарува задолжително и нумеричко решавање на нелинеарна равенка, што значително го зголемува побарувањето на компјутерски ресурси, односно, го зголемува потребното компјутерско време.

7.2. Кеплеров проблем

Да разгледаме еден конкретен пример на примена на експлицитниот метод на Ојлер, а тоа е Кеплеровиот проблем. Тој се состои во определување на траекториите под дејство на централни сили од облик

$$\vec{F} = \frac{k}{r^2}\vec{r}_0, \tag{7.11}$$

при што k може да биде како позитивно така и негативно. Таква сила е на пример Њутновата сила на општа гравитација

$$\vec{F} = -\gamma\frac{mM}{r^2}\vec{r}_0. \tag{7.12}$$

Гравитационата сила со која телото со маса M го привлекува телото со маса m ја разложуваме на компоненти во xy-рамнината (бидејќи секое централно движење е рамнинско). Значи

$$\begin{aligned}\vec{F} &= F_x\vec{i} + F_y\vec{j} = -\gamma\frac{mM}{r^2}\vec{r}_0 = \\ &= -\gamma\frac{mM}{r^2}\left(\frac{x}{r}\vec{i} + \frac{y}{r}\vec{j}\right) = -\gamma\frac{mM}{r^3}x\vec{i} + -\gamma\frac{mM}{r^3}y\vec{j},\end{aligned} \tag{7.13}$$

па компонентите на силата се

$$F_x = -\gamma\frac{mM}{r^3}x \qquad F_y = -\gamma\frac{mM}{r^3}y, \tag{7.14}$$

од каде равенките на движење ќе бидат

$$\begin{aligned}m\frac{\mathrm{d}v_x}{\mathrm{d}t} &= -\gamma\frac{mM}{r^3}x \\ m\frac{\mathrm{d}v_y}{\mathrm{d}t} &= -\gamma\frac{mM}{r^3}y,\end{aligned} \tag{7.15}$$

каде $r = (x^2 + y^2)^{1/2}$, а заради поедноставување понатаму ќе земеме дека $m = 1$ и $\gamma M = 1$. Значи, се поставува проблем на одредување траекториите кога се дадени равенките на движење

$$\frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{x}{\left(x^2+y^2\right)^{3/2}}$$
$$\frac{\mathrm{d}v_y}{\mathrm{d}t} = -\frac{y}{\left(x^2+y^2\right)^{3/2}}. \tag{7.16}$$

Според методот на Ојлер ќе имаме

$$v_x\left(t+\Delta t\right) = v_x\left(t\right) - \frac{x(t)}{r^3(t)}\Delta t$$
$$v_y\left(t+\Delta t\right) = v_y\left(t\right) - \frac{y(t)}{r^3(t)}\Delta t. \tag{7.17}$$

За да се одреди траекторијата ќе ги земеме предвид и релаците

$$v_x = \frac{\mathrm{d}x}{\mathrm{d}t} \qquad v_y = \frac{dy}{dt}, \tag{7.18}$$

односно според методот на Ојлер

$$x\left(t+\Delta t\right) = x\left(t\right) + v_x\left(t\right)\Delta t$$
$$y\left(t+\Delta t\right) = y\left(t\right) + v_y\left(t\right)\Delta t, \tag{7.19}$$

Значи, знаејќи ја почетната положба и брзина на честицата што се движи под дејство на гравитационата сила, од горните равенки можеме *приближно* да ја одредиме положбата и брзината во наредниот момент Δt, а потоа ова положба и брзина ќе служат како почетни за одредување на положбата и брзината во следниот момент итн. Така, одредувајќи ја положбата во голем број моменти и нанесувајќи ги вредностите за x и y координатите на секој од овие моменти ќе ја добиеме приближната траекторија на честицата.

Прецизноста на пресметувањето на положбата на честицата со методот на Ојлер зависи линеарно од избраниот чекор Δt, значи ако земеме два пати помал чекор, односно временски интервал, ќе добиеме два пати помала грешка, а со тоа и нумерички определената траекторија ќе биде поблиска до вистинската.

Но сето ова има свое ограничување, бидејќи земањето на премногу кратко Δt го успорува пресметувањето, а исто така може да доведе до нумерички нестабилности.

Затоа почесто се применуваат модификаци на методот на Ојлер со кои се тој се подобрува, односно за дадено Δt да се добијат почетни резултати во однос на основниот метод. Така во гореопишениот метод, најголема погрешка произлегува од тоа што положбата на честицата во следниоит момент ја одредуваме со помош на брзината во претходниот момент која општо не е еднаква на брзината во конечната (следната) положба. Таа погрешка можеме прилично да ја смалиме ако брзината ја пресметуваме во средината на на интервалот на време меѓу две соседни положби на честици $\left(\Delta t/2, 3\Delta t/2, \ldots, (n-1/2)\Delta t\right)$. Притоа положбата и силата ги пресметуваме за целобројни вредности на Δt. Така, за првиот чекор ќе имаме

$$v_x\left(\frac{1}{2}\right)=v_x(0)+F_x(0)\frac{\Delta t}{2}$$
$$x(1)=x(0)+v_x\left(\frac{1}{2}\right)\Delta t, \tag{7.20}$$

за вториот чекор

$$v_x\left(\frac{3}{2}\right)=v_x\left(\frac{1}{2}\right)+F_x(1)\Delta t$$
$$x(2)=x(1)+v_x\left(\frac{3}{2}\right)\Delta t, \tag{7.21}$$

за $n+1$–от чекор имаме

$$v_x\left(n+\frac{1}{2}\right)=v_x\left(n-\frac{1}{2}\right)+F_x(n)\Delta t$$
$$x(n+1)=x(n)+v_x\left(n+\frac{1}{2}\right)\Delta t. \tag{7.22}$$

Овие релаци ја чинат основата на програмата ojler_pikard.for со која се одредени разни облици на траектории за честица што се движи под дејство на централна сила. Како што е познато Кеплеровиот проблем може и аналитички да се реши, т.е. да се добие егзактен израз за траекторијата кој во поларни координати има форма

$$r=\frac{p}{1-\varepsilon\cos\varphi}, \tag{7.23}$$

каде

$$p=-\frac{L^2}{mk}\qquad \varepsilon=\sqrt{1-\frac{2L^2E}{mk^2}},$$

а $L=mvr$ е момент на импулс на честица со маса m и E е вкупна енергија на честицата. Равенката (7.23) претставува конусен пресек со фокус во полот на координатен систем (значи во центарот на силата) и тоа елипса ако $\varepsilon<1$, хипербола ако $\varepsilon>1$, а парабола ако $\varepsilon=1$.

Според тоа под влијание на централна сила која опаѓа со квадратот на растојанието, честицата опишува траекторија со облик на конусен пресек со фокус во изворот на силата и тоа *елипса* ако $E<0$, *хипербола* ако $E>0$ и *круг* ако $E=0$, што претставува всушност формулација на првиот Кеплеров закон.

Ако е $k<0$ што одговара на привлечната сила, потенцијалната енергија $U=k/r$ ќе биде негативна, па вкупната енергија $E=mv^2/2+k/r$ може да биде и позитивна, и негативна и нула што зависи од соодносот на кинетичката и потенцијалната енергија.

Значи ако $k<0$ траекторијата може да биде и елипса, и хипербола и парабола. Специјално, кога $E=mk^2/(2L^2)$ траекторијата е круг. Меѓутоа ако $k>0$, што одговара на одбивна сила, потенцијалната енергија е позитивна па вкупната енергија е секогаш позитивна. Затоа, за случајот $k>0$ треаекторијата на честицата може да биде само

хипербола, како на пример во случајот на расејување на α–честици во поле на атомски јадра (Редерфордов експеримент).

За движењето на планетите околу Сонцето („Кеплерово“ движење) важи $k<0$ поради привлечната гравитациона сила, а вкупната енергија е исто така негативна. Значи орбитите на планетите ќе бидат елипси во чиј фокус се наоѓа Сонцето. Од константноста пак, на секторската брзина (втор Кеплеров закон) се изведува третиот Кеплеров закон, кој гласи

$$\frac{T^2}{a^3}=const\,, \tag{7.24}$$

т.е. односот на квадратот на периодот на обиколување и кубот на големата полуоска на елипсата е константен. Бидејќи реалното прикажување на „Кеплеровото“ движење со демонстрационен обид е скоро неможно, а астрономските набљудувања се долготрајни, компјутерското прикажување на разни облици на траектории при ова движење секако е пожелно и корисно. Таков нумерички пристап кон Кеплеровото движење е оправден и од историски причини, бидејќи најспектакуларните проверки на законот на гравитација се добиени со нумерички пресметувања на движењата на планетите (откритието на Нептун и Плутон).

Исто така, егзактно решавање на Кеплеровото движење (можно само со заемодејство на два тела) е повеќе примерено за курсеви по теориска механика, додека пак реалните движења (на пример на сателити, ракети итн.) се решаваат нумерички.

Во програмата подолу даден е кодот во фортран со кој нумерички се пресметуваат конусните пресеци или траекториите на тело што се движи во гравитрационо поле, со методите на Ојлер и Пикадр.

```
      PROGRAM OJLER_PIKARD
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION T(1000),X(1000),Y(1000),VX(1000),VY(1000)
      PARAMETER (N = 101,IN= 2)
      CHARACTER NAME1*30,NAME2*30

      WRITE(*,*)'Program: OJLER I PIKARD'
      WRITE(*,*)'Dvizenje na telo vo dve dimenzii pod dejstvo'
      WRITE(*,*)'na gravitasiona sila.'
      WRITE(*,*)'Primeneti se metodite na Ojler i'
      WRITE(*,*)'podobreniot Ojlerov metod - metod na Pikard'
      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF ( NFLAG .EQ. 2 ) THEN
           WRITE(*,*) 'Za sekoj metod vnesi ime na fajl vo vid - '
           WRITE(*,*) 'drive:name.ext'
           WRITE(*,*) 'patot da bide staven vo navodnici'
           WRITE(*,*) 'primer1:   "C:/DATA1.TXT" '
*          WRITE(*,*) 'primer2:   "C:/DATA2.TXT" '
           READ (*,*) NAME1
           READ (*,*) NAME2
           WRITE(*,*) ' '
           OUP1 = 3
           OUP2 = 4
           OPEN(UNIT=OUP1,FILE=NAME1)
```

```
            OPEN(UNIT=OUP2,FILE=NAME2)
          ELSE
            OUP1 = 6
            OUP2 = 6
        END IF

C       SE ZADAVAAT POCETNITE USLOVI
C       X, VX I Y OSTANUVAAT ISTI.
C       VY = 0.5 - ELIPSA
C       VY = 1.0 - KRUG
C       VY = SQRT(2) - PARABOLA
C       VY = 2.0 - HIPERBOLA

        PI =4.0D0*DATAN(1.0D0)
        DT =2.0D0*PI/100
        T(1) = 0.0D0
        X(1) = 1.0D0
        VX(1)= 0.0D0
        Y(1) = 0.0D0
        VY(1)= 0.5D0

C       METOD NA OJLER
        DO I = 1, N-1
           T(I+1) = REAL(I)*DT
           R  = DSQRT(X(I)*X(I)+Y(I)*Y(I))
           R3 = R**3.0
           X(I+1)  = X(I)  + VX(I)*DT
           VX(I+1) = VX(I) - X(I)*DT/R3
           Y(I+1)  = Y(I)  + VY(I)*DT
           VY(I+1) = VY(I) - Y(I)*DT/R3
        END DO
        WRITE(OUP1,'(17X,A)') '#METOD NA OJLER'
        WRITE(OUP1,*) ' '
        WRITE(OUP1,'(8X,A,10X,A,10X,A,10X,A,10X,A)')'#T(I)','X(I)',
     * 'VX(I)','Y(I)','VY(I)'
        WRITE(OUP1,*) ' '
        WRITE(OUP1,'(5(2X(F12.5)))') (T(I),X(I),VX(I),Y(I),VY(I),
     * I=1,N,IN)

C       METOD NA PIKARD
        DO I = 1, N-1
           T(I+1) = REAL(I)*DT
           R  = DSQRT(X(I)*X(I)+Y(I)*Y(I))
           R3 = R**3.0
           X(I+1)  = X(I)  + VX(I)*DT
           VX(I+1) = VX(I) - X(I)*DT/R3
           Y(I+1)  = Y(I)  + VY(I)*DT
           VY(I+1) = VY(I) - Y(I)*DT/R3
C       KOREKCIJA NA POZICIJATA I BRZINATA
           X(I+1)  = X(I)  + (VX(I)+VX(I+1))*DT/2.
           VX(I+1) = VX(I) - (X(I) +X(I+1))*DT/2.
           Y(I+1)  = Y(I)  + (VY(I)+VY(I+1))*DT/2.
           VY(I+1) = VY(I) - (Y(I) +Y(I+1))*DT/2.
        END DO

        WRITE(OUP2,*) ' '
        WRITE(OUP2,'(17X,A)') '#METOD NA PIKARD'
        WRITE(OUP2,*) ' '
        WRITE(OUP2,'(8X,A,10X,A,10X,A,10X,A,10X,A)') '#T(I)','X(I)',
     * 'VX(I)','Y(I)','VY(I)'
        WRITE(OUP2,*) ' '
        WRITE(OUP2,'(5(2X(F12.5)))') (T(I),X(I),VX(I),Y(I),VY(I),
     * I=1,N,IN)

        CLOSE(UNIT=OUP1)
        CLOSE(UNIT=OUP2)
        IF(OUP1.NE.OUP1.OR.OUP2.NE.OUP2) CLOSE(UNIT=6)
        STOP
        END
```

Дел од излезот на програмата за решавање на Кеплеров проблем со метод на Ојлер и Пикард:

```
METOD NA OJLER

  T(I)          X(I)          VX(I)         Y(I)          VY(I)

0.00000       1.00000       0.00000       0.00000       0.50000
0.31416       0.96048      -0.31633       0.15583       0.47976
0.62832       0.81885      -0.66351       0.29811       0.39648
0.94248       0.55867      -1.10261       0.40081       0.18090
1.25664       0.13840      -1.73176       0.39239      -0.53885
1.57080      -0.39138      -0.67445      -0.06244      -2.64926
1.88496      -0.43872       0.21417      -0.83540      -2.17605
2.19911      -0.35629       0.30480      -1.48612      -1.95398
2.51327      -0.25742       0.32457      -2.08549      -1.85166
2.82743      -0.15456       0.33027      -2.65915      -1.79310
3.14159      -0.05055       0.33184      -3.21732      -1.75508
3.45575       0.05373       0.33194      -3.76512      -1.72833
3.76991       0.15797       0.33148      -4.30546      -1.70844
4.08407       0.26202       0.33081      -4.84015      -1.69306
4.39823       0.36586       0.33008      -5.37043      -1.68078
4.71239       0.46946       0.32935      -5.89716      -1.67076
5.02655       0.57284       0.32865      -6.42096      -1.66240
5.34071       0.67601       0.32800      -6.94230      -1.65534
5.65487       0.77897       0.32739      -7.46156      -1.64928
5.96903       0.88175       0.32683      -7.97901      -1.64402
6.28319       0.98436       0.32631      -8.49490      -1.63941


METOD NA PIKARD

  T(I)          X(I)          VX(I)         Y(I)          VY(I)

0.00000       1.00000       0.00000       0.00000       0.50000
0.31416       0.95078      -0.30889       0.15458       0.47552
0.62832       0.80642      -0.58720       0.29346       0.40455
0.94248       0.57857      -0.80664       0.40079       0.29450
1.25664       0.28793      -0.94397       0.45955       0.15790
1.57080      -0.02823      -0.98472       0.45295       0.01285
1.88496      -0.31893      -0.92920       0.39684      -0.12130
2.19911      -0.57209      -0.78827       0.32281      -0.23475
2.51327      -0.77703      -0.57486       0.22947      -0.32203
2.82743      -0.91201      -0.30754       0.11754      -0.37692
3.14159      -0.96067      -0.01108      -0.00492      -0.39474
3.45575      -0.91560       0.28602      -0.12665      -0.37391
3.76991      -0.77857       0.55436      -0.23526      -0.31657
4.08407      -0.55955       0.76644      -0.31772      -0.22887
4.39823      -0.27522       0.89893      -0.35674      -0.12151
4.71239       0.03638       0.93609      -0.31842      -0.01302
5.02655       0.28575       0.88361      -0.22849       0.07242
5.34071       0.50156       0.75964      -0.16924       0.13443
5.65487       0.69102       0.57124      -0.11376       0.17902
5.96903       0.82412       0.33153      -0.05160       0.20515
6.28319       0.88126       0.06157       0.01453       0.21101
```

7.3. Метод на Рунге-Кута

Користејќи ја идејата на Ојлер, прирастот на решението $y(x)$ на диференцијалните равнки од облик (7.1) во општа форма може да се запиши како

$$y_{i+1} = y_i + h\,\Phi(y, x, h), \tag{7.25}$$

со што останува да се определи функцијата на прираст $\Phi(y, x, h)$ во согласност со даден нумеричкиот метод, Како што веќе покажавме, во експлицитниот метод на Ојлер имаме $\Phi(y, x, h) = f(x, y)$.

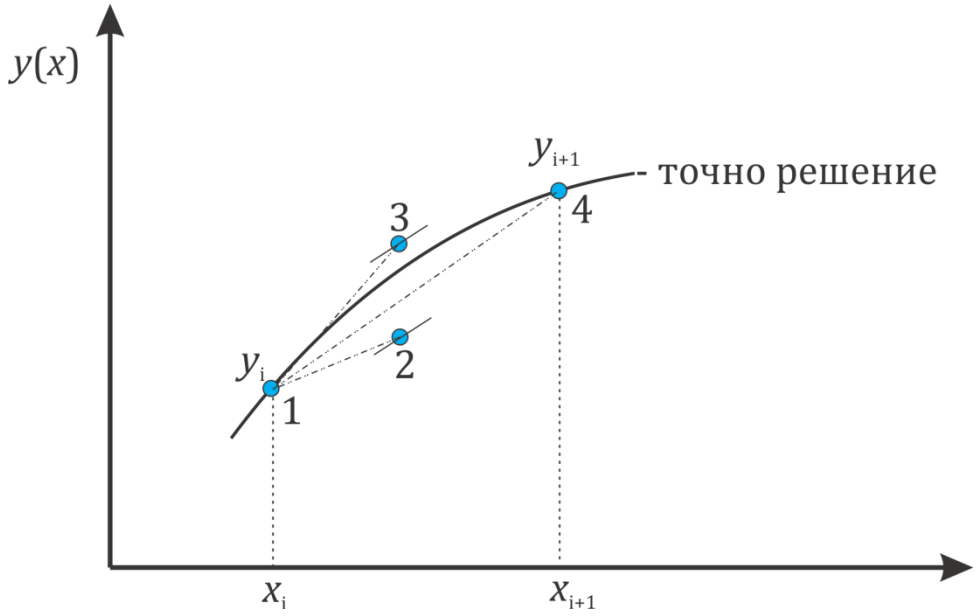
За нумеричко решавање диференцијални равенки со поголема прецизност постојат многу други методи од кој секој има своја мана но и предност. Овде ќе го изложиме еден од најприменливите познат како *методот на Рунге-Кута од четврти ред* познат и како *типичен метод на Рунге-Кута*. Според овој метод, на функцијата на прираст се пресметува со изразот $\Phi(y,x,h)=\left(k_1+2k_2+2k_3+k_4\right)/6$, со што прирастот на решението се врши според формулата

$$y_{i+1}=y_i+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right), \tag{7.26}$$

каде што

$$\begin{aligned} k_1&=hf(x_i,y_i)\\ k_2&=hf(x_i+\frac{h}{2},y_i+\frac{1}{2}k_1)\\ k_3&=hf(x_i+\frac{h}{2},y_i+\frac{1}{2}k_2)\\ k_4&=hf(x_i+h,y_i+k_3). \end{aligned} \tag{7.27}$$

Геометрискиот приказ на апроксимацијата што ја прави методот на Рунге-Кута е прикажана на слика 7.2.



Сл. 7.2. Графички приказ на апроксимациите на метод на Рунге-Кута од 4-ти ред. Функцијата на прираст во овој случај се усреднува во четири точки со различни тежински коефициенти.

Со програмата kepler.for, нумерички се пресметуваат сите облици на траекториите на движење на тело во гравитационо поле (Кеплеров проблем) во зависност од вредноста на коефициентот k и вкупната енергија E.

```
PROGRAM KEPLER
IMPLICIT REAL*8 (A-H,P-Z)
DIMENSION Y(10)
```

```
      CHARACTER NAME1*30
      EXTERNAL DERIV

      WRITE(*,*)'Program: KEPLER'
      WRITE(*,*)'Resavanje na diferencijalna ravenka'
      WRITE(*,*)'n: broj na ravenki od prv red'
      WRITE(*,*)'ns: broj na cekori, h: cekor'
      WRITE(*,*)'Y(1): pocetna pozicija x, Y(2): pocetna brzina vx'
      WRITE(*,*)'Y(3): pocetna pozicija y, Y(4): pocetna brzina vy'
      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF ( NFLAG .EQ. 2 ) THEN
           WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
           WRITE(*,*) 'drive:name.ext'
           WRITE(*,*) 'patot da bide staven vo navodnici'
           WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
           READ (*,*) NAME1
           WRITE(*,*) ' '
           OUP = 3
           OPEN(UNIT=OUP,FILE=NAME1)
        ELSE
           OUP = 6
      END IF
C     N GO DEFINIRA BROJOT NA DIFERENCIJALNI RAVENKI
      N=4
C     NS E BROJ NA JAZLI ZA INTEGRACIJA
      NS=300
      PI = 4.0D0*DATAN(1.0D0)
C     H E CEKOR NA INTEGRACIJA
      H =2.0D0*PI/NS

C     DEFINIRANJE NA POCETNITE USLOVI
      Y(1)=1.0D0
      Y(2)=0.0D0
      Y(3)=0.0D0
      Y(4)=0.5D0

      WRITE(OUP,900) '#T','X','VX','Y','VY'
      WRITE(OUP,*) ' '
      WRITE (OUP,'(5(2X,E12.5))')0.D0,Y(1),Y(2),Y(3),Y(4)
  9 0 0 FORMAT (10X,A,12X,A,10X,A,10X,A,10X,A)

C     CIKLUS SO KOJ VO SEKOJ JAZOL SE PRIOMENUVA METODOT RK4
      DO J = 1, NS
         T=REAL(J,8)*H
         CALL RK4(T, Y, H, N)
         WRITE (OUP,'(5(2X,E12.5))')T,Y(1),Y(2),Y(3),Y(4)
      END DO

      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END
C     POTPROGRAMA ZA METOD NA RUNKGE-KUTA OD 4-TI RED
      SUBROUTINE RK4(T, Y, TSTEP, N)
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION AK1(5),AK2(5),AK3(5),AK4(5),Y(10),TEMP1(5),TEMP2(5),
     * TEMP3(5)
      EXTERNAL DERIV
      H=TSTEP/2.0D0
      DO I = 1,N
         AK1(I) = TSTEP * DERIV(T, Y, I)
         TEMP1(I) = Y(I) + 0.5D0*AK1(I)
      END DO
      DO I = 1,N
         AK2(I) = TSTEP * DERIV(T+H, TEMP1, I)
         TEMP2(I) = Y(I) + 0.5D0*AK2(I)
```

```
      END DO
      DO I = 1,N
         AK3(I) = TSTEP * DERIV(T+H, TEMP2, I)
         TEMP3(I) = Y(I) + AK3(I)
      END DO
      DO I = 1,N
         AK4(I) = TSTEP * DERIV(T+TSTEP, TEMP3, I)
         Y(I) = Y(I) + (AK1(I) + (2.D0*(AK2(I) + AK3(I))) + AK4(I))
     *  /6.0D0
      END DO
      RETURN
      END
C     FUNKCIJA SO KOJA SE DEFINIRAAT DIFERENCIJALNITE RAVENKI
      REAL*8 FUNCTION DERIV(T,TEMP,I)
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION TEMP(10)
      R=DSQRT(TEMP(1)*TEMP(1)+TEMP(3)*TEMP(3))
      R3=R**3.0D0
      IF (I .EQ. 1) DERIV=TEMP(2)
      IF (I .EQ. 2) DERIV=-TEMP(1)/R3
      IF (I .EQ. 3) DERIV=TEMP(4)
      IF (I .EQ. 4) DERIV=-TEMP(3)/R3
      RETURN
      END
```

Дел од излезот на програмата за решавање на Кеплеров проблем со метод на Рунге-Кута од 4-ти ред:

T	X	VX	Y	VY
0.00000E+00	0.10000E+01	0.00000E+00	0.00000E+00	0.50000E+00
0.20944E-01	0.99978E+00	-0.20946E-01	0.10471E-01	0.49989E+00
0.23038E+00	0.99912E+00	-0.41903E-01	0.20938E-01	0.49956E+00
0.43982E+00	0.99803E+00	-0.62884E-01	0.31395E-01	0.49901E+00
0.64926E+00	0.99649E+00	-0.83899E-01	0.41839E-01	0.49824E+00
0.85870E+00	0.99451E+00	-0.10496E+00	0.52264E-01	0.49724E+00
0.10681E+01	0.99209E+00	-0.12608E+00	0.62666E-01	0.49602E+00
0.12776E+01	0.98923E+00	-0.14727E+00	0.73040E-01	0.49457E+00
0.14870E+01	0.98592E+00	-0.16854E+00	0.83381E-01	0.49289E+00
0.16965E+01	0.98217E+00	-0.18991E+00	0.93684E-01	0.49096E+00
0.19059E+01	0.97797E+00	-0.21138E+00	0.10394E+00	0.48880E+00
...				
0.40003E+01	0.91060E+00	-0.43515E+00	0.20298E+00	0.45209E+00
0.42097E+01	0.90124E+00	-0.45878E+00	0.21240E+00	0.44667E+00
0.44192E+01	0.89138E+00	-0.48271E+00	0.22169E+00	0.44087E+00
0.46286E+01	0.88102E+00	-0.50696E+00	0.23086E+00	0.43468E+00
0.48381E+01	0.87014E+00	-0.53156E+00	0.23990E+00	0.42807E+00
0.50475E+01	0.85875E+00	-0.55654E+00	0.24879E+00	0.42101E+00
0.52569E+01	0.84683E+00	-0.58191E+00	0.25753E+00	0.41347E+00
0.54664E+01	0.83437E+00	-0.60770E+00	0.26611E+00	0.40544E+00
0.56758E+01	0.82137E+00	-0.63395E+00	0.27451E+00	0.39687E+00
0.58853E+01	0.80781E+00	-0.66068E+00	0.28273E+00	0.38772E+00
0.60947E+01	0.79369E+00	-0.68793E+00	0.29074E+00	0.37796E+00

После ова, преостанува со помош на Excel или Gnuplot да се нацртаат конусните пресеци во случаите кога се применуваат методите на Ојлер, Пикард и Рунге-Кута. Што заклучуваш од обликот на траекториите?

7.4. Кеплеров проблем во присутство на сила на триење

Методот на Рунге-Кута ќе го применуваме на разрешување на Кеплеров проблем, но за еден пореален случај кога имаме присутство на сила на триење која зависи линеарно од брзината. За овој случај, равенката на движење ќе биде

$$m\frac{\mathrm{d}\vec{v}}{\mathrm{d}t}=\lambda\vec{v}+\frac{k}{r^2}\vec{r}_0 \tag{7.28}$$

каде λ е константа на триење. Со оглед дека и за овој случај движењето е рамнинско, оваа векторска равенка ја разложуваме на две скаларни равенки

$$\begin{aligned} m\frac{\mathrm{d}v_x}{\mathrm{d}t}&=\lambda v_x+\frac{k\,x}{\left(x^2+y^2\right)^{3/2}}\\ m\frac{\mathrm{d}v_y}{\mathrm{d}t}&=\lambda v_y+\frac{k\,y}{\left(x^2+y^2\right)^{3/2}} \end{aligned} \tag{7.29}$$

Имајќи ги предвид и равенките

$$\frac{\mathrm{d}x}{\mathrm{d}t}=v_x \qquad \frac{\mathrm{d}y}{\mathrm{d}t}=v_y \tag{7.30}$$

ќе можеме да ги добиеме брзините и положбите во секој момент на време, а со тоа и траекториите на честицата за разни почетни положби и брзини.

Со програмата trienje.for овозможено е добивање на траекториите за разни вредности на константата, коефициентот на триење, почетната брзина и положба.

```
      PROGRAM KEPLER
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION Y(10)
      CHARACTER NAME1*30
      EXTERNAL DERIV

      WRITE(*,*)'Program: KEPLER'
      WRITE(*,*)'Resavanje na diferencijalna ravenka'
      WRITE(*,*)'n: broj na ravenki od prv red'
      WRITE(*,*)'ns: broj na cekori, h: cekor'
      WRITE(*,*)'Y(1): pocetna pozicija x, Y(2): pocetna brzina vx'
      WRITE(*,*)'Y(3): pocetna pozicija y, Y(4): pocetna brzina vy'
      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF ( NFLAG .EQ. 2 ) THEN
           WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
           WRITE(*,*) 'drive:name.ext'
           WRITE(*,*) 'patot da bide staven vo navodnici'
           WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
           READ (*,*) NAME1
           WRITE(*,*) ' '
           OUP = 3
           OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
           OUP = 6
      END IF
C     N E BROJ NA RAVENKI OD PRV RED
      N=4
C     NS SE BROJ NA CEKORI NA INTEGRACIJA
      NS=200
C     H E VREMENSKI CEKOR
      H =0.1D0

C     POCETNI USLOVI
      Y(1)=-10.0D0
```

```
      Y(2)=  1.0D0
      Y(3)=  1.0D0
      Y(4)=  0.0D0

      WRITE(OUP,900) '#T','X','VX','Y','VY'
      WRITE(OUP,*) ' '
      WRITE (OUP,'(5(2X,E12.5))')0.D0,Y(1),Y(2),Y(3),Y(4)
  900 FORMAT (10X,A,12X,A,10X,A,10X,A,10X,A)

      DO J = 1, NS
         T=REAL(J,8)*H
         CALL RK4(T, Y, H, N)
         WRITE (OUP,'(5(2X,E12.5))')T,Y(1),Y(2),Y(3),Y(4)
      END DO

      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END
C     POTPROGRAMA ZA RUNGE-KUTA OD 4-TI RED
      SUBROUTINE RK4(T, Y, TSTEP, N)
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION AK1(5),AK2(5),AK3(5),AK4(5),Y(10),TEMP1(5),TEMP2(5),
     * TEMP3(5)
      EXTERNAL DERIV
      H=TSTEP/2.0D0
      DO I = 1,N
         AK1(I) = TSTEP * DERIV(T, Y, I)
         TEMP1(I) = Y(I) + 0.5D0*AK1(I)
      END DO
      DO I = 1,N
         AK2(I) = TSTEP * DERIV(T+H, TEMP1, I)
         TEMP2(I) = Y(I) + 0.5D0*AK2(I)
      END DO
      DO I = 1,N
         AK3(I) = TSTEP * DERIV(T+H, TEMP2, I)
         TEMP3(I) = Y(I) + AK3(I)
      END DO
      DO I = 1,N
         AK4(I) = TSTEP * DERIV(T+TSTEP, TEMP3, I)
         Y(I) = Y(I) + (AK1(I) + (2.D0*(AK2(I) + AK3(I))) + AK4(I))
     *   /6.0D0
      END DO
      RETURN
      END

C     FUNKCIJA SO KOJA SE DEFINIRAAT DIFERENCIJALNITE RAVENKI
      REAL*8 FUNCTION DERIV(T,TEMP,I)
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION TEMP(10)
      R=DSQRT(TEMP(1)*TEMP(1)+TEMP(3)*TEMP(3))
      R3=R**3.0D0
      AK=20.D0
      ALAMBDA=1.D0
      IF (I .EQ. 1) DERIV=TEMP(2)
      IF (I .EQ. 2) DERIV=ALAMBDA*TEMP(2)+AK*TEMP(1)/R3
      IF (I .EQ. 3) DERIV=TEMP(4)
      IF (I .EQ. 4) DERIV=ALAMBDA*TEMP(4)+AK*TEMP(3)/R3
      RETURN
      END
```

Дел од излезот на програмата за решавање на Кеплеров проблем со сили на триење со метод на Рунге-Кута од 4-ти ред:

```
          T           X           VX          Y           VY
   0.00000E+00 -0.10000E+02  0.10000E+01  0.10000E+01  0.00000E+00
   0.10000E+00 -0.98959E+01  0.10842E+01  0.10001E+01  0.21038E-02
   0.20000E+00 -0.97829E+01  0.11769E+01  0.10004E+01  0.44990E-02
   0.30000E+00 -0.96602E+01  0.12788E+01  0.10010E+01  0.72261E-02
   0.40000E+00 -0.95268E+01  0.13908E+01  0.10019E+01  0.10332E-01
   0.50000E+00 -0.93816E+01  0.15139E+01  0.10031E+01  0.13871E-01
   0.60000E+00 -0.92236E+01  0.16492E+01  0.10047E+01  0.17906E-01
   0.70000E+00 -0.90514E+01  0.17980E+01  0.10067E+01  0.22510E-01
```

```
 0.80000E+00  -0.88635E+01   0.19613E+01   0.10092E+01   0.27770E-01
 0.90000E+00  -0.86585E+01   0.21408E+01   0.10123E+01   0.33787E-01
 0.10000E+01  -0.84348E+01   0.23377E+01   0.10160E+01   0.40684E-01
 0.11000E+01  -0.81904E+01   0.25539E+01   0.10204E+01   0.48605E-01
 ...
 0.40000E+01   0.27102E+02   0.36696E+02   0.10800E+02   0.12609E+02
 0.41000E+01   0.30962E+02   0.40558E+02   0.12126E+02   0.13936E+02
 0.42000E+01   0.35227E+02   0.44825E+02   0.13592E+02   0.15402E+02
 0.43000E+01   0.39942E+02   0.49540E+02   0.15212E+02   0.17023E+02
 0.44000E+01   0.45152E+02   0.54752E+02   0.17002E+02   0.18813E+02
 0.45000E+01   0.50910E+02   0.60511E+02   0.18981E+02   0.20792E+02
 0.46000E+01   0.57274E+02   0.66875E+02   0.21167E+02   0.22979E+02
 0.47000E+01   0.64307E+02   0.73909E+02   0.23584E+02   0.25396E+02
 0.48000E+01   0.72081E+02   0.81682E+02   0.26255E+02   0.28067E+02
 0.49000E+01   0.80671E+02   0.90273E+02   0.29207E+02   0.31019E+02
 0.50000E+01   0.90165E+02   0.99768E+02   0.32469E+02   0.34281E+02
```

Како задача останува да се нацртаат сите решенија што се добиваат со програмата trienje.for за различни вредности на почетните услови. Добиените резултати да се споредат со појавата на расејување на позитивно наелектризирани честици од атомски јадра?

7.5. Задачи

1. Со користење на методот на Рунге-Кута, да се реши диференцијалната равенка

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + 2\lambda\omega_0 \frac{\mathrm{d}y}{\mathrm{d}t} + \omega_0^2 y = 0,$$

со почетни услови $y(0)=0$ и $y'(0)=1$, сопствена фреквенција $\omega_0 = 2$ и коефициент на пригушување $\lambda \in [0,0.8]$. Добиените решенија да се нацртаат а потоа да се коментира нивното поведени во зависност од почетните услови и параметарите *r* и ω_0 ?

2. Со користење на методите на Пикард и Рунге-Кута, да се реши диференцијалната равенка

$$\frac{dy}{dt} = \lambda(y - t^3) + 3t^2,$$

со почетни услови $y(0) = 0, \pm 10^{-3}, \pm 10^{-6}, \pm 10^{-9}$, и $\lambda = \pm 3, \pm 27$. Добиените решенија да се нацртаат а потоа да се коментира нивното поведени во зависност од почетните услови и параметарот λ?

7.6. Проекти

1. Со користење на методот на Рунге-Кута, да се реши диференцијалната равенка

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{\lambda}{mR^2}\frac{\mathrm{d}\theta}{\mathrm{d}t} + \frac{g}{R}\sin\theta = \frac{A}{mR^2}\cos(kt),$$

со почетни услови $\theta(0)=0$ и $\theta'(0)=1$, а останатите параметри се: константа на триење $\lambda = 0.5$, фреквенција на надворешна сила $k = 0.666$, гравитационо забрзување $g = 1$, амплитуда на надворешна сила $A = 1$, должина на нишало $R = 1$ и маса на нишало $m = 1$

. Добиеното решение да се анимира а потоа да се коментира зависноста на решенијата од почетните услови и параметарите што влегуваат во диференцијалната равенка ?

корисен линк:

http://www.myphysicslab.com/pendulum2.html

2. Да се направи симулација за движење на n тела во правоаголен билијар со дадена димензија.

а) Телата се одбиваат еластично од зидовите на билјарот и димензиите на телата да не се земат предвид. Заемодејството меѓу телата да се занемари.

б) Заемодејството меѓу телата да биде парно, на пример потенцијал на Ленард-Џонс, и димензиите на телата да се земат предвид?

в) Да се определи зависноста на средното растојание меѓу телата во зависност од енергијата на телата, димензиите на билијарот и бројот на телата?

г) Да се определи зависноста на бројот на судари меѓу телата и меѓу телата и зидовите на билијарот во зависност од енергијата на телата, димензиите на билијарот и бројот на телата?

8. ГРАНИЧНИ УСЛОВИ И СОПСТВЕНИ ВРЕДНОСТИ

За физиката од посебна важност се проблемите што побаруваат решавање на обични диференцијални равенки со вредности на физичките величини што се дадени на границите на некоја област. Ваков проблем се среќава при решавањето на Пуасоновата рвенка

$$\nabla^2 \Phi = -4\pi\rho, \tag{8.1}$$

со дадена распределба на електричниот полнеж $\rho(r)$ и познати гранични вредености на електростастскиот потенцијал или пак стационарната Шредингерова равенка за даден потенцијал и соодветни гранични услови

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} + \frac{2m}{\hbar^2}\left[E - V(x)\right]\psi(x) = 0. \tag{8.2}$$

Сепак, во општ случај, проблемот на гранични врдности се задава со равенки од обликот

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} = f(\psi', \psi, x), \tag{8.3}$$

што значи дека во равенката покрај функцијата $\psi(x)$ се среќава и нејзиниот прв и втор извод $\psi'(x)$ и $\psi''(x)$, соодветно. За еднодимензионалните проблеми има четири можни гранични услови

$$\begin{aligned} &\psi(0) = \psi_0 \quad && \psi(1) = \psi_1 \\ &\psi(0) = \psi_0 \quad && \psi'(1) = \zeta_1 \\ &\psi'(0) = \zeta_0 \quad && \psi(1) = \psi_1 \\ &\psi'(0) = \zeta_0 \quad && \psi'(1) = \zeta_1 \end{aligned}, \tag{8.4}$$

а најчесто во нумеричките методи е застапен првиот. Тоа значи дека ако имаме друг вид на гранични услови потребна е соодветно пролагодување на нумериката.

Проблемот на гранични вредности, во принцип, е потежок за решавање во однос на проблемот на почетни вредности затоа што покрај методот за решавање на обична диференцијална равенка треба да се примени и метод за изнаоѓање корен на функција. Но, има и дополнителен проблем. Типичните гранични проблеми, како што е стационарната Шредингерова равенка, покрај постојаните променливи, зависи и од дополнителен праметар познат како сопствена вредност λ

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} = f(\psi, x, \lambda), \tag{8.5}$$

при што треба да се има предвид дека сопствената врдност λ може да има само одредени вредности што зависат од граничните услови, но исто така тие вердности треба да имаат и физичко значење што соодветствува на проблемот кој се решава.

8.1. Проблем на Штурм-Лиувил

Кога зборуваме за проблемот на гранични вредности, равенката на Штурм-Лиувил означува една многу важна група на проблеми во физиката кои во општ облик може да се запишат како

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[p(x)\frac{\mathrm{d}y}{\mathrm{d}x}\right]+q(x)y(x)=s(x), \tag{8.6}$$

каде $p(x)$ и $q(x)$ се коефициент функции. Најчесто, во практичните проблеми имаме $s(x)=0$, а $q(x)=r(x)+\lambda w(x)$ при што λ е сопствената вредност (8.6) а $r(x)$ и $w(x)$ се предефинирани коефициент функции. Со цел на напишиме алгоритам за приближно пресметување на оваа равенка, најпрво ќе ги искористиме изразите за прв и втор извод во три точки

$$\Delta_1=\frac{y_{i+1}-2y_i+y_{i-1}}{2h}=y_i'+\frac{h^2}{6}y_i^{(3)}+O(h^4), \tag{8.7}$$

и

$$\Delta_2=\frac{y_{i+1}-2y_i+y_{i-1}}{h^2}=y_i''+\frac{h^2}{12}y_i^{(4)}+O(h^4). \tag{8.8}$$

Сега, изразот (8.7) ќе го помножиме со p' а (8.8) со p, па изразите ќе ги собериме

$$\Delta_1 p_i'+\Delta_2 p_i=\left(y_i' p_i\right)'+\frac{h^2}{12}\left(2y_i^{(3)}p_i'+y_i^{(4)}p_i\right)+O(h^4). \tag{8.9}$$

Од (8.6) следи $(y_i' p_i)'=s_i-q_i y_i$, па со замена во (8.8) добиваме

$$h(y_{i+1}-y_{i-1})p_i'+2(y_{i+1}-2y_i+y_{i-1})p_i=2h^2(s_i-q_i y_i),$$

каде од десната страна на (8.9) предвид е земен само првиот собирок. Со решавање на горната равенка по y_{i+1} добиваме

$$(hp_i'+2p_i)y_{i+1}-(4p_i-2h^2q_i)y_i+(2p_i-hp_i')y_{i-1}=2h^2s_i. \tag{8.10}$$

Ако оваа рекурзивна релација ја презапишиме во облик погоден за нумерички итерации, ќе имаме

$$c_{i+1}y_{i+1}+c_i y_i+c_{i-1}y_{i-1}=d_i, \tag{8.11}$$

каде коефициентите се дефинини со изразите

$$\begin{aligned} c_{i+1}&=2p_i+hp_i' \qquad c_i=-4p_i+2h^2q_i \\ c_{i-1}&=2p_i-hp_i' \qquad d_i=2h^2s_i. \end{aligned} \tag{8.12}$$

Прецизноста на овој алгоритам е до $O(h^4)$, што најчесто не е доволна, па затоа треба да се направат определени модификации. Ќе започниме со двојно диференцирање на (8.6), со што добиваме

$$p\,y^{(4)}+3p'y^{(3)}+3p''y''+q\,y''+2q'y'+p^{(3)}y'+q''y=s'', \tag{8.13}$$

додека од првото диференцирање го добиваме изразот

$$y^{(3)}=\frac{1}{p}(s'-p''y'-2p'y''-q'\,y-q\,y'). \tag{8.14}$$

Со замена на двата горни изрази во (8.9) добиваме

$$c_{i+1}y_{i+1}+c_iy_i+c_{i-1}y_{i-1}=d_i, \tag{8.15}$$

каде коефициентите се дефинирани со изразите

$$\begin{aligned}
c_{i+1} &= 24p_i+12hp_i'+2h^2q_i+6h^2p_i''-4h^2p_i'^2/p_i \\
&\quad +h^3p_i^{(3)}+2h^3q_i'-h^3p_i'q_i/p_i-h^3p_i'p_i''/p_i \\
c_i &= 48p_i-20h^2q_i-8h^2p_i'^2/p_i+12h^2p_i''+2h^4p_i'q_i'/p_i-2h^4q_i'' \\
c_{i-1} &= 24p_i-12hp_i'+2h^2q_i+6h^2p_i''-4h^2p_i'^2/p_i \\
&\quad -h^3p_i^{(3)}-2h^3q_i'+h^3p_i'q_i/p_i+h^3p_i'p_i''/p_i \\
d_i &= 48h^2s_i\,h^4s_i''-2h^4p_i's_i'/p_i.
\end{aligned} \tag{8.16}$$

За да не се намали прецизноста на алгоритмот од $O(h^6)$, при пресметување на коефициентите треба да се промени формулата во три точки за прв и втор извод на $p(x)$ и $q(x)$.

8.2. Алгоритам на Нумеров

За физиката од посебно значење се равенките од облик (8.6) во кои $p(x)=1$ додека $q(x)=k^2$, со што добиваме

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2}=-k^2y(x)+s(x)=y''. \tag{8.17}$$

Ако оваа равенка ја диференцираме двапати, и за вторите изводи на десната страна ја примениме формулата во три точки, имаме

$$\frac{\mathrm{d}^4y}{\mathrm{d}x^4}=y^{(4)}=\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left[-k^2y(x)+s(x)\right], \tag{8.18}$$

или

$$y^{(4)}=-\frac{(k^2y)_{i+1}-2(k^2y)_i+(k^2y)_{i-1}}{h^2}+\frac{s_{i+1}-2s_i+s_{i-1}}{h^2}+O(h^2).$$

Со замена на оваа равенка во (8.8) добиваме

$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2}=-\frac{h^2(k_i^2-s_i)}{h^2}+$$
$$+\frac{h^2}{12}\left(\frac{s_{i+1}-2s_i+s_{i-1}}{h^2}-\frac{k_{i+1}^2y_{i+1}-2k_i^2y_i+k_{i-1}^2y_{i-1}}{h^2}\right)+O(h^6). \tag{8.19}$$

Кога овој израз ќе го решиме по y_{i+1} може да запишиме уште една рекурзивна релација во истиот облик како и (8.15), со тоа што коефициентите ќе се дефеинираат со изразите

$$c_{i+1}=1+\frac{h^2}{12}k_{i+1}^2 \qquad c_i=-2\left(1-\frac{5h^2}{12}k_i^2\right)$$
$$c_{i-1}=1+\frac{h^2}{12}k_{i-1}^2 \qquad d_i=\frac{h^2}{12}(S_{i+1}+10S_i+S_{i-1}). \tag{8.20}$$

Лесно се забележува дека формулата на Нумеров е со локална грешка $O(h^6)$, значи е за еден ред поточна од онаа на Рунге-Кута. Исто така, формулата на Нумеров е поефикасна бидејќи вредностите k^2 и S ги пресметува само во точките на решетката за интеграција.

Пример 8.1.

Како пример за решавање на проблем со гранични вредности со формулата на Нумеров ќе го разгледаме решавањето на Поасоновата равенка (8.1) чија распределба на густина на електричество е

$$\rho(r)=\frac{1}{8\pi}e^{-r}. \tag{8.21}$$

Егзактното решение на овој проблем е

$$\varphi(r)=1-\frac{1}{2}(r+2)e^{-r}, \tag{8.22}$$

од каде следи $\Phi=\varphi r^{-1}$. Анализата на решението ни дава дека за големо r потенцијалот е од облика $\Phi\to r^{-1}$, што е во согласност со физичното поведение на потенцијал од единечен полнеж.

Ова значи дека равенката (8.1) ќе ја интегрираме на напред почнувајќи од нула, при што е неопходно преуредување на (8.17) со $k^2=0$, од каде следи

$$\varphi_{i+1}-2\varphi_i+\varphi_{i-1}=\frac{h^2}{12}\left(S_{i+1}+10S_i+S_{i-1}\right), \tag{8.23}$$

а изворната функција е

$$S(r)=-4\pi r\rho=-\frac{1}{2}re^{-r}. \tag{8.24}$$

Меѓутоа, за да може да интегрираме според формулата на Нумеров, мора да ја знаеме $\varphi_0=0$ и вредноста на $\varphi_1=\varphi(h)$ што е исто со $d\varphi/dr=\Phi(0)$. Секако дека ова не е добро бидејќи φ_1 е дел од функцијата која треба да ја најдеме. Сепак, во конкретниов

случај, без дополнителна дискусија за оправданоста, бидејќи го знаеме аналитичкото решение може да напишиме дека $\varphi_1 = \varphi_{x=h}$.

Со програмата numerov.for, дадена подолу во текстот, спроведена е интеграција на Пуасоновата равенка чија распределба на електричниот полнеж е зададена со (8.21). Од нумеричките решенија може да се види дека при големи r грешката на пресметките се зголемува. Знаејќи го аналитичкото решение, може да кажиме дека причината за ова е што при големи радиуси доминира нефизичкото решение $\Phi \approx const.$ за разлика од пригушувањето на физички коректното поведение $\Phi \approx r^{-1}$. Надминувањето на овој преблем повторно е поврзан со познавањето на точното решение, што значи дека како пример ќе важи само во овој случај, па затоа нема да биде дополнително анализиран.

```
      PROGRAM NUMEROV

      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION PHI(0:300)
      CHARACTER NAME1*30
      EXTERNAL S,T

      WRITE(*,*)'Program: NUMEROV'
      WRITE(*,*)'Primena na semata na Numerov za resavanje'
      WRITE(*,*)'difernecijalni ravenki od vtor red:'
      WRITE(*,*)'y''(x) + k^2(x) y(x) = s(x)'
      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF ( NFLAG .EQ. 2 ) THEN
           WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
           WRITE(*,*) 'drive:name.ext'
           WRITE(*,*) 'patot da bide staven vo navodnici'
           WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
           READ (*,*) NAME1
           WRITE(*,*) ' '
           OUP = 3
            OPEN(UNIT=OUP,FILE=NAME1)
        ELSE
           OUP = 6
      END IF

      H=0.1D0
      NSTEP=INT(20.D0/H)
      CONST=H**2/12.D0
      SM=0.D0
      SZ=S(H)
      PHI(0)=0.D0
      PHI(1)=T(H)
      WRITE(OUP,900) 'R','T(R)','PHI(IR+1)','DIFFERENCE'
      WRITE(OUP,*) ' '
      DO IR=1,NSTEP-1
         R=(IR+1)*H
         SP=S(R)
         PHI(IR+1.D0)=2.D0*PHI(IR)-PHI(IR-1)+
     *   CONST*(SP+10.D0*SZ+SM)
      SM=SZ
      SZ=SP
      WRITE(OUP,'(4(2X,F12.8))')R,T(R),PHI(IR+1),T(R)-PHI(IR+1)
      END DO

  900 FORMAT (9X,A,11X,A,7X,A,5X,A)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END
```

```
C       DEFINIRANJE NA "IZVORNATA" FUNKCIJATA
        REAL*8 FUNCTION S(X)
        REAL*8 X
        S=-X*EXP(-X)/2.D0
        RETURN
C       END

C       DEFINIRANJE NA TOCNATA FUNKCIJATA
        REAL*8 FUNCTION T(X)
        REAL*8 X
        T=1.D0-(X+2.D0)*EXP(-X)/2.D0
        RETURN
        END
```

Дел од излезот на програмата numerov.for:

R	T(R)	PHI(IR+1)	DIFFERENCE
0.20000000	0.09939617	0.09939618	-0.00000001
0.30000000	0.14805905	0.14805907	-0.00000002
0.40000000	0.19561594	0.19561599	-0.00000004
0.50000000	0.24183668	0.24183674	-0.00000007
0.60000000	0.28654487	0.28654497	-0.00000009
0.70000000	0.32960984	0.32960997	-0.00000013
0.80000000	0.37093945	0.37093961	-0.00000016
0.90000000	0.41047399	0.41047420	-0.00000020
1.00000000	0.44818084	0.44818108	-0.00000024
1.10000000	0.48404982	0.48405011	-0.00000029
...			
19.00000000	0.99999994	1.00001063	-0.00001069
19.10000000	0.99999995	1.00001070	-0.00001075
19.20000000	0.99999995	1.00001076	-0.00001081
19.30000000	0.99999996	1.00001082	-0.00001087
19.40000000	0.99999996	1.00001089	-0.00001093
19.50000000	0.99999996	1.00001095	-0.00001098
19.60000000	0.99999997	1.00001101	-0.00001104
19.70000000	0.99999997	1.00001107	-0.00001110
19.80000000	0.99999997	1.00001113	-0.00001116
19.90000000	0.99999998	1.00001119	-0.00001122
20.00000000	0.99999998	1.00001125	-0.00001128

Да се потсетиме дека во овој пример нестабилностите што може да се јават заради претпоставената вредност PHI(1)=? при интеграцијата нанапред се надминуваат со тоа што се заменува точната вредност на решението PHI(1)=T(H). Како алтернативен приод во случаите кога не се знае точното решение, погодно е да се претпостави малата промена на решението во вид $\varphi_1 = \varphi_0 + \delta$, што како вежба е корисно да се проба.

8.3. Метод на гаѓање

Погоден метод за нумеричко решавање на проблем на сопствени вредности е итеративниот. За таа цел, избираме пробна сопствена вредност k_0 и генерираме решение со интегрирање на диференцијалната равенка како проблем на почетната вредност за кој $\varphi_0 = 0$. Ако добиеното решение не го задоволува граничниот услов $\varphi_n \approx 0$, ја менуваме пробната сопствена вредност $k^{(0)} \to k^{(1)}$ и повторно интегрираме, повторувајќи го процесот додека не најдеме сопствена вредност за која граничните услови ќе бидат задоволени со однапред зададена толеранција $\varphi_n = 0 \pm \delta$. Оваа постапка е позната како *метод на гаѓање* и во себе имплементира два нумерички метода, еден за решавење на проблем на почетни вредности и друг за приближно наоѓање корен на функција,

$f(k)=\varphi^{(k)}(1)-\varphi_{x=1}=0$. Имајќи го предвид ова, во пракса најчесто се користат комбинации на методите на Рунге-Кута и Нумеров со методите на бисекција и секанти.

Пример 8.2.

Можеби најчесто споменувам пример за определување на сопствени вредности во физиката е оној на брановата равенка, поточно вибрирањето на жица прицврстена на краевите. Соодветната равенка со која се опишува овој проблем на сопствени вредности со соодветни граничните услови е

$$\frac{\mathrm{d}^2\varphi}{\mathrm{d}x^2}=-k^2\varphi, \tag{8.25}$$

каде граничните услови се $\varphi_{x=0}=\varphi_{x=1}=0$, што значи дека должината на жицата е $l=1$ па затоа важи $0<x<1$. Со φ се означува трансверзалното поместување на жицата, k е бранов број додека $\lambda=k^2$ се сопствените вредности. Оваа равенка е равенка за сопствени вредности затоа што решенијата кои ги задоволуваат граничните услови егзистираат само за посеби вредности на k_n кои треба да се определат.

Нормираните сопствени функции и сопствени вредности на овој проблем се многу добро познати аналитички

$$\varphi_n(x)=\sqrt{2}\sin\left(k_n x\right) \tag{8.26}$$

и

$$\lambda_n=k_n^2=(n\pi)^2 \tag{8.27}$$

каде *n* е позитивен цел број.

Во програмата sopstveni.for се наоѓа најмалата сопствена вредност за проблемот на трансверзални осцилации на прицврстена жица (8.25).

```
PROGRAM SOPSTVENI_VREDNOSTI

IMPLICIT REAL*8 (A-H,P-Z)
PARAMETER (N=101)
DIMENSION Q(N),S(N),U(N)
CHARACTER NAME1*30

WRITE(*,*)'Program: SOPSTVENI_VREDNOSTI'
WRITE(*,*)'Iznaogjanje na sopstveni vrednosti'
WRITE(*,*)'na ravenkata: u" = -k**2*u'
WRITE(*,*)'so pocetni uslovi: u(0)=u(1)=0'
WRITE(*,*) ' '
WRITE(*,*) 'Odberete vid na izlez: '
WRITE(*,*) '1. Ekran '
WRITE(*,*) '2. Text file '
WRITE(*,*) 'Vnesi 1 ili 2 '
READ(*,*) NFLAG
WRITE(*,*) ' '

IF ( NFLAG .EQ. 2 ) THEN
WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
WRITE(*,*) 'drive:name.ext'
WRITE(*,*) 'patot da bide staven vo navodnici'
WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
READ (*,*) NAME1
WRITE(*,*) ' '
OUP = 3
OPEN(UNIT=OUP,FILE=NAME1)
ELSE
```

```
      OUP = 6
      END IF

C     CEKOR NA INTEGRACIJA
      H = 1.0D0/(N-1)
C     INTERVAL NA BARANJE SOPSTVENA VREDNOST (AK,BK)
      AK = 2.0D0
      BK = 4.0D0
C     POCETNO RESENIE ZA SOPSTVENA VREDNOST K
      DK = 1.0D0
C     TOCNOST NA METODOT NA DIHOTOMIJA E DL
      DL = 1.0E-06
      ISTEP = 0
C     POCETNI VREDNOSTI NA SOPSTVENATA FUNKCIJA
      U(1) = 0.0D0
      U(2) = 0.01D0
      EK = AK

      DO I = 1,N
         S(I) = 0.0D0
         Q(I) = EK*EK
      END DO

      CALL NMRV(N,H,Q,S,U)
      F0 = U(N)

      WRITE (OUP,*)'NAJMALATA SOPSTVENA VRENDOST E VO KOLONATA EK:'
      WRITE (OUP,'(1X,A,8X,A,14X,A,14X,A)') 'ISTEP','EK','DK','F1'

C     METOD NA PREPOLOVUVANJE ZA NAOGJANJE NA K
      DO WHILE (DABS(DK).GT.DL)
         EK = (AK+BK)/2.0D0
         DO I = 1,N
            Q(I) = EK*EK
         END DO
         CALL NMRV (N,H,Q,S,U)
         F1 = U(N)
         IF ((F0*F1).LT.0D0) THEN
              BK = EK
              DK = BK-AK
         ELSE
              AK = EK
              DK = BK-AK
              F0 = F1
         END IF
      WRITE (OUP,999) ISTEP,EK,DK,F1
      ISTEP = ISTEP+1
      END DO
      WRITE (OUP,*) ' '
C     WRITE (OUP,999) ISTEP,EK,DK,F1

      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
  999 FORMAT (I4,3F16.8)
      END

C     ALGORITAM NA NUMEROV
      SUBROUTINE NMRV (NS,HS,QS,SS,US)
      IMPLICIT REAL*8 (A-H,P-Z)
      PARAMETER (N=101)
      DIMENSION QS(N),SS(N),US(N)

      G = HS*HS/12.0D0
      DO I = 2, NS-1
         C0 = 1.0D0+G*((QS(I-1)-QS(I+1))/2.0D0+QS(I))
         C1 = 2.0D0-G*(QS(I+1)+QS(I-1)+8.0D0*QS(I))
         C2 = 1.0D0+G*((QS(I+1)-QS(I-1))/2.0D0+QS(I))
         D = G*(SS(I+1)+SS(I-1)+10.0D0*SS(I))
         UTMP = C1*US(I)-C0*US(I-1)+D
         US(I+1) = UTMP/C2
      END DO
      RETURN
      END
```

```
Излезот на програмата sopstveni.for е:

 NAJMALATA SOPSTVENA VRENDOST E VO KOLONATA EK:
 ISTEP        EK              DK              F1
   0      3.00000000      1.00000000      0.04704706
   1      3.50000000      0.50000000     -0.10024425
   2      3.25000000      0.25000000     -0.03329667
   3      3.12500000      0.12500000      0.00531027
   4      3.18750000      0.06250000     -0.01439969
   5      3.15625000      0.03125000     -0.00464452
   6      3.14062500      0.01562500      0.00030816
   7      3.14843750      0.00781250     -0.00217439
   8      3.14453125      0.00390625     -0.00093466
   9      3.14257812      0.00195313     -0.00031364
  10      3.14160156      0.00097656     -0.00000284
  11      3.14111328      0.00048828      0.00015264
  12      3.14135742      0.00024414      0.00007489
  13      3.14147949      0.00012207      0.00003603
  14      3.14154053      0.00006104      0.00001659
  15      3.14157104      0.00003052      0.00000688
  16      3.14158630      0.00001526      0.00000202
  17      3.14159393      0.00000763     -0.00000041
  18      3.14159012      0.00000381      0.00000081
  19      3.14159203      0.00000191      0.00000020
  20      3.14159298      0.00000095     -0.00000011
```

8.4. Задачи

1. Да се модифицира програмата sopstveni.for со цел наоѓање на сопствената вредност $k_2 = 2\pi$?

2. Да се определат сопствените врдености на диференцијалната равенка

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = -\frac{\pi^2}{4}(u+1),$$

со гранични услови $u(0)=0$ и $u(1)=1$?

3. Со користење на методите на секанти и Нумеров (или Рунге-Кута), да се определи сопствената вредност $l=1$ за Лежандровата диференцијална равенка

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[(1-x^2)\frac{\mathrm{d}u}{\mathrm{d}x}\right] + l(l+1)u = 0,$$

со почетни услови $u0=0$ и $u1=h$, и точност на изнаоѓање сопствена вредност од 10^{-6}? Да се има предвид дека за оваа равенка $x\in[-1,1]$. За $l=1$ да се нацрта полиномот $P_1(x)$?

4. Стационарниот проблем на пренос на топлина во една димензија низ шипка (прачка) се состои од дифузија на толината низ прачката и размена на топлина со околината со конвекција. Овој проблем може да се опиши со следната обична диференцијална равенка

$$\frac{\mathrm{d}^2 T}{\mathrm{d}x^2} = \alpha^2 (T - T_{\mathrm{ok}}),$$

каде граничните услови се зададени со $T(0)=0\ ℃$ и $T(L)=100\ ℃$. Останатите параметри се: должина на шипката $L=0.01\,\mathrm{m}$, коефициент $\alpha=400\ \mathrm{m}^{-1}$ и температура на околината $T_{ок}=0\ ℃$.

8.5. Проекти

1. Да се реши еднодимензионалната Шредингерова равенка

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2}+\frac{2m}{\hbar^2}[E-V(x)]\psi(x)=0,$$

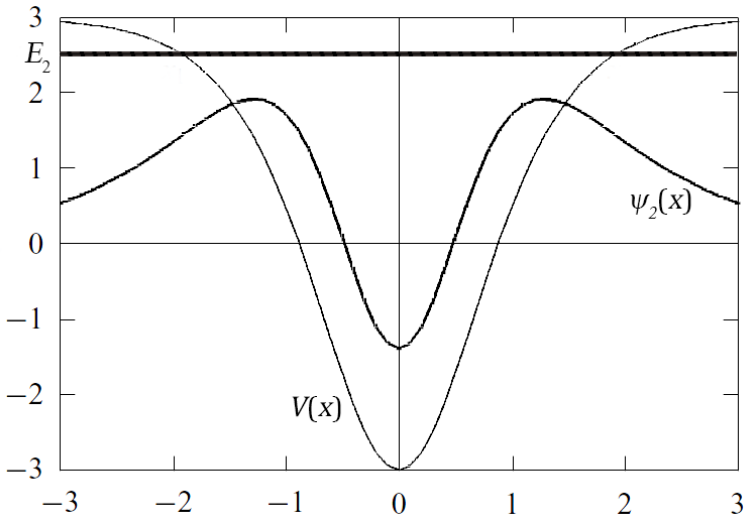
за која потенцијалот е зададен со изразот

$$V(x)=\frac{\hbar^2}{2m}\alpha^2\lambda(\lambda-1)\left[\frac{1}{2}-\frac{1}{\cosh^2(\alpha x)}\right].$$

За овој потенцијал равенката може да се реши аналитички со што за енергетските нивои се добива решението

$$E_n=\frac{\hbar^2}{2m}\alpha^2\left[\frac{\lambda(\lambda-1)}{2}-(\lambda-1-n)^2\right].$$

при што $n=0,1,2,\ldots$. Решението да се бара во областа $x\in[-10,10]$ со $\mathrm{N}=501$ јазли на интеграција, а заради поедностаување зимаме $\hbar=m=1$.



Сл. 8.1. Графички приказ на решенијата на Шредингеровата равенка.

На сликата 8.1. се прикажани решенијата кога $n=2$, $\lambda=4$ и $\alpha=1$

.

9. *ЕЛИПТИЧНИ ПАРЦИЈАЛНИ ДИФЕРЕНЦИЈАЛНИ РАВЕНКИ*

Во претходните поглавја аборувавме за нумерички методи за решавање на проблемите на почетни и гранични услови кај обичните диференцијални равенки. Некои од овие методи може да се воопштат и да се применат за решавање на диференцијални равенки со повеќе променливи какви што се парцијалните диференцијални равенки (ПДР). Многу физички проблеми од областа на квантната физика, електродинамиката, хидродинамиката итн. се задаваат со парцијални диференцијални равенки од втор ред што се поделени во три класи: *елиптични*, *параболични и хиперболични*.

Грубо земено, елиптичните парцијални диференцијални равенки вклучуваат изводи од втор ред со ист знак за секоја од независните просторни променливи. Во оваа класа спаѓаат Поасоновата равенка за електростатски потенцијал Φ, која при константна вредност на диелектричната константа ε гласи

$$\nabla^2 \Phi = -\frac{\rho}{\varepsilon}, \tag{9.1}$$

и временски независната Шредингерова равенка

$$\nabla^2 \psi + \frac{2m}{\hbar^2}\left[E - V(x)\right]\psi(x) = 0. \tag{9.2}$$

Во ова поглавје, за линеарните елиптични парцијални диференцијални равенки најпрво ќе го разгледаме проблемот на гранични вредности во две димензии, со тоа што равенката (9.1) ќе ја презапишиме во облик

$$-\left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right]\varphi = S(x, y), \tag{9.3}$$

при што φ е електростатски потенцијал и S, генерално, е густина на електричеството. Покрај равенката, потребни се и гранични услови од типот на Диришле со кои φ ќе биде зададена на затворена крива во (x, y) рамнината.

Проблемите на сопствени вредности ќе прикажиме со равенката (9.2), презапишана за две димензии

$$-\left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right]\varphi + V(x, y)\varphi = \varepsilon\varphi, \tag{9.4}$$

при што φ е брановата функција, V е потенцијал, а ε е сопствена вредност. Исто така, неопходно е да се зададат и Диришлетовите гранични услови.

9.1. Дискретизација на равенката

Прв чекор во решавање на парцијалните диференцијални равенки е да се доведат во форма погодна за нумерички пресметувања. За да го направиме тоа, потребно е направиме дискретизација на континуалните променливи какви што се просторните координати и времето. За таа цел, ќе ги примениме формулите за конечни разлики, со кои првиот и вториот извод ќе ги прикажиме со формулите за конечни разлики во две и три точки. Така на пример, првиот извод на функцијата $\varphi(r,t)$ со формулите во две и три точки може да го прикажиме како

$$\frac{\partial\varphi(r,t)}{\partial t}=\frac{\varphi(r,t_{k+1})-\varphi(r,t_k)}{\tau}, \tag{9.5}$$

и

$$\frac{\partial\varphi(r,t)}{\partial t}=\frac{\varphi(r,t_{k+1})-\varphi(r,t_{k-1})}{2\tau}. \tag{9.6}$$

За вториот извод, вообичаено се користи формулата во три точки

$$\frac{\partial^2\varphi(r,t)}{\partial t^2}=\frac{\varphi(r,t_{k+1})-2\varphi(r,t_k)+\varphi(r,t_{k-1})}{\tau^2}, \tag{9.7}$$

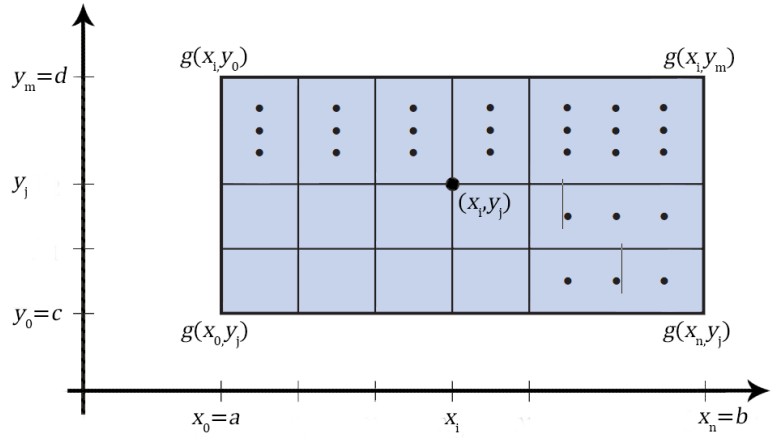
каде со $\tau=t_{k+1}-t_k$ е означен времнскиот чекор на интеграција. На сличен начин вршиме дискретизација и на просторните координати, а ќе прикажиме како се прави тоа за x координатата,

$$\frac{\partial\varphi(r,t)}{\partial x}=\frac{\partial\varphi(x_{k+1},y,z,t)-\varphi(x_{k-1},y,x,t)}{2h_x}, \tag{9.8}$$

и

$$\frac{\partial^2\varphi(r,t)}{\partial x^2}=\frac{\partial\varphi(x_{k+1},y,z,t)-2\varphi(x_k,y,x,t)+\varphi(x_{k-1},y,x,t)}{h_x{}^2}, \tag{9.9}$$

каде $h_x=x_{k+1}-x_k$ е чекор на интеграција по координатата x.



Сл. 9.1. Графички приказ на умрежување на областа на интеграција.

За да може да се применат овие формули во дискретизација на равенките, најпрво ќе ја дефинираме решетка со точки која ќе ја препокрива областа на интерес во (x,y) рамнината. Ќе земеме константата на решетката по x оската да биде h_x со што еквидистантните јазли на решетката се дефинираат со $x_i = a + ih_x$ за $i = 0,1,2\ldots n$. Соодветно за y координатата имаме $y_i = b + ih_y$, каде константата н решетката по y оската е h_y. Со ова умрежување, областа на интеграција ќе биде препокриена со $(n+1)\times(m+1)$ решеткини јазли, слика 9.1.

Ако понатаму дефинираме дека $\varphi_{ij} = \varphi(x_i, y_j)$, $S_{ij} = S(x_i, y_j)$ а константите на решетката го означиме како $h_x = h$ и $h_y = k$, и применувајќи ја диференцната формула во три точки за вториот извод за x и y координатите, равенката (9.3) може да ја апроксимираме како

$$-\left[\frac{\varphi_{i+1,j}+\varphi_{i-1,j}-2\varphi_{ij}}{h^2}+\frac{\varphi_{i,j+1}+\varphi_{i,j-1}-2\varphi_{ij}}{k^2}\right]=S_{ij}, \tag{9.10}$$

или во подруга форма

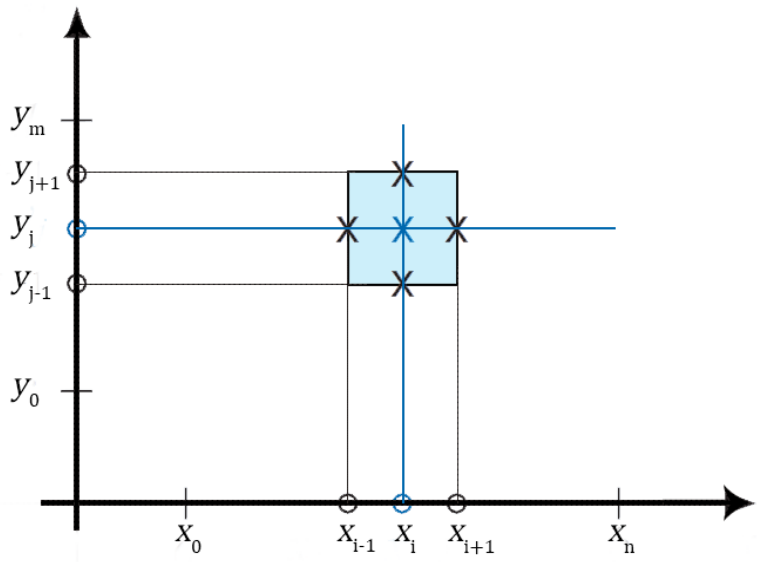
$$2\left[1+\left(\frac{h}{k}\right)^2\right]\varphi_{i,j}-(\varphi_{i+1,j}+\varphi_{i-1,j})-\left(\frac{h}{k}\right)^2(\varphi_{i,j+1}+\varphi_{i,j-1})=h^2S_{ij}, \tag{9.11}$$

каде граничните услови на Диришле се

$$\begin{aligned}\varphi_{0,j} &= g(x_0, y_j) \qquad \varphi_{n,j} = g(x_n, y_j)\\ \varphi_{i,0} &= g(x_i, y_0) \qquad \varphi_{i,n} = g(x_i, y_n),\end{aligned} \tag{9.12}$$

при што $i = 1,2\ldots(n-1)$ и $j = 1,2\ldots(m-1)$.

Формулата (9.11) овозможува $\varphi_{i,j}$ да се апроксимира во пет точки во форма на крст, што е прикажано на слика 9.2.



Сл. 9.2. Графички приказ на умрежување на областа на интеграција.

9.2. Метод на релаксација

Со овој метод се добива општа шема за итерација со која се обновува веќе добиеното решение со рекурентната формула (9.11), со цел да се забрза конвергенција кон точното решение. Овој приод е познат како метод на релаксација и се покажал како извонредно погоден за нумеричко разрешување на елиптичните ПДР.

Најпрво, во случај на една променлива, равенката (9.11) ќе ја запишиме како

$$-\frac{\varphi_{i+1}+\varphi_{i-1}-2\varphi_i}{h^2}=S_i\,, \tag{9.13}$$

или во решена форма по φ_i

$$\varphi_i=\frac{1}{2}(h^2S_i+\varphi_{i+1}+\varphi_{i-1})\,. \tag{9.14}$$

Методот на релаксација ни дава за право подобрената вредност на решението $\varphi_i^{(k)}$ да ја запишиме како

$$\varphi_i^{(k)}=\varphi_i^{(k-1)}+\omega r_i\,, \tag{9.15}$$

каде коефициентот r_i се дефинира како

$$r_i=\varphi_i-\varphi_i^{(k-1)}\,. \tag{9.16}$$

Од тука следи

$$\varphi_i^{(k)}=(1-\omega)\varphi_i^{(k-1)}+\omega\varphi_i\,, \tag{9.17}$$

или рекурентната формула што ќе ја користиме во понатамошните пресметки за добивање на подобреното решение

$$\varphi_i^{(k)}=(1-\omega)\varphi_i^{(k-1)}+\frac{\omega}{2}(h^2S_i+\varphi_{i+1}+\varphi_{i-1})\,, \tag{9.18}$$

каде ω е параметар на релаксација со кој се регулира брзината на конвергенција. Кога имаме $0<\omega<1$, станува збор за метод на подрелаксација, а кога $1<\omega$ имаме метод на надрелаксација. Во случаите кога системот од равенки се решава со итеративниот метод на Гаус-Зајдел и $\omega>1$, всушност го користиме сукцесивниот метод на надрелаксација или скратено СОР (Successive Over Relaxation - SOR).

Пример **9.1.**

Како практично се применува диференцната формула (9.18) ќе покажиме со примерот на деформација на еластична шипка. Станува збор за хомогена еластична шипка со фолжина L која е прицврстена на двата краја. На определено растојание од потпорните точки ја оптоваруваме со еден тег, па како резултат на тоа прачката се деформира по целата должина. Определувањето на поместувањето на шипката од рамнотежната положба се определува со елиштична парцијална диференцијална равенка која има облик

$$\frac{\partial^2\varphi}{\partial x^2}=\frac{f(x)}{J\,I}, \tag{9.19}$$

каде J е Јунгов модул на еластичност, $I=d^3a/3$ е инерцијален момент каде d и a се дебелина и ширина на шипката, соодветно. $f(x)$ е функција со која се опишува распределбата на силата по должина на шипката. Бидејќи двата краја се фиксни, граничните услови се $\varphi(0)=\varphi(L)=0$. Јазлите на интеграција се дефинираат со $x_i=a+ih$ при што $i=0,1\ldots n$, а чекорот на интеграција е $h=L/n$. Распределбата на силата по должина на прачката е земено да има ограничена Гаусова форма што се дефинира со функцијата

$$f(x)=\begin{cases}-f_0[e^{-\left(\frac{x-L/2}{x_0}\right)^2}-e^{-1}]-\rho g & |x-L/2|\le x_0\\ -\rho g & \text{на други места,}\end{cases} \tag{9.20}$$

каде $\rho=3.0\,\mathrm{kg/m}$ е линеарна густина на шипката, $g=9.81\,\mathrm{m/s^2}$ е Земјиното забрзување, $f_0=200\,\mathrm{N/m}$ е константа на линеарно оптоварување, $x_0=0.25\,\mathrm{m}$, должината на шипката е $L=3.0\,\mathrm{m}$, ширината и е $a=0.2\,\mathrm{m}$ а дебелината $d=0.03\,\mathrm{m}$ додека Јунговиот модул на еластичност е $J=1.0\times10^9\,\mathrm{N/m^2}$.

Програмата relax.for што го разрешува овој проблем користи упростена форма на диференцната равенка (9.11) за една променлива, x. На тој начин добиваме систем од линеарни равенки кој може да се претстави со тридијагонална матрица а решението на системот, во овој случај, ќе го направиме со итеративниот метод на Гаус-Зајдел, што го изучвме во примерот 3.2.

```
      PROGRAMA RELAX
      IMPLICIT REAL*8 (A-H,P-Z)
      PARAMETER (NI=10000)
      DIMENSION W(NI),X(NI),S(NI),XR(NI)
      CHARACTER NAME1*30,AA*1
      LOGICAL OK

            WRITE(*,*) 'Method na centralni konecni razliki'
            WRITE(*,*) 'za Elipticni Ravenki.'
            WRITE(*,*) ' '
            WRITE(*,*) 'Vnesi go krajnite tocki na X-oskata [A,B]'
            WRITE(*,*) 'odvoeni so blanko (eden prazen karakter.'
            WRITE(*,*) 'na primer: 0.0 3.0'
            WRITE(*,*) ' '
            READ(*,*) A, B
 1  1         OK = .FALSE.
 1  2         IF (OK) GOTO 13
            WRITE(*,*) 'Vnesi broj na intervali na X-oskata n'
            WRITE(*,*) 'na primer : 100'
            WRITE(*,*) '2 < n < 500.'
            WRITE(*,*) ' '
            READ(*,*) N
                  IF (N.LE.2) THEN
                    WRITE(*,*) 'Mora da e cel broj pogolem od 2.'
                  ELSE
                    OK = .TRUE.
                  ENDIF
               GOTO 12
 1  3         OK = .FALSE.
 1  4         IF (OK) GOTO 15
                  WRITE(*,*) 'Vnesi ja tolerancijata (1.E-6)'
                  WRITE(*,*) ' '
                  READ(*,*) TOL
                  IF (TOL.LE.0.0) THEN
                     WRITE(*,*) 'Tolerancijata treba da bide pozitivna '
                  ELSE
```

```
                  OK = .TRUE.
                ENDIF
              GOTO 14
 1  5        OK = .FALSE.
 1  6        IF(OK) GOTO 17
                WRITE(*,*) 'Vnesi go maksimalniot broj na iteracii.'
           WRITE(*,*) 'na primer: 4000'
                WRITE(*,*) ' '
                READ(*,*) LBOUND
                IF(LBOUND.LE.0) THEN
                   WRITE(*,*) 'Mora da bide pozitiven.'
                ELSE
                   OK = .TRUE.
                ENDIF
                  GOTO 16
 1  7          OK = .FALSE.
 1  8          IF (OK) GOTO 19
                WRITE(*,*) 'Vnesi parametar na relaksacija p:'
           WRITE(*,*) 'treba 0 < p < 2 '
                WRITE(*,*) ' '
                READ(*,*) P
                IF (P.LE.0.0) THEN
                   WRITE(*,*) 'treba p > 0! '
                ELSE
                   OK = .TRUE.
                ENDIF
              GOTO 18
 1  9     CONTINUE
       IF(.NOT.OK) GOTO 700
            WRITE(*,*) ' '
            WRITE(*,*)'Vnesi kade da bide prikazan izlezotna programata: '
            WRITE(*,*) '1. Ekran '
            WRITE(*,*) '2. Tekstualna datoteka '
            WRITE(*,*) 'Vnesi 1 ili 2 '
            WRITE(*,*) ' '
            READ(*,*) NFLAG
            IF ( NFLAG .EQ. 2 ) THEN
               WRITE(*,*) 'Napisi go imeto na datotekata kako - '
               WRITE(*,*) 'drive:ime.ext'
               WRITE(*,*) 'Imeto zadolzitelno da bide vo navodnici'
               WRITE(*,*) 'na primer:"c:/ime.txt"'
               WRITE(*,*) ' '
               READ(*,*) NAME1
               OUP = 3
               OPEN(UNIT=OUP,FILE=NAME1,STATUS='UNKNOWN')
            ELSE
               OUP = 6
            ENDIF
            WRITE(OUP,*) '#RAVENKA NA POISSON SO METOD NA RELAKSACIJA (SOR)'
            WRITE(OUP,*) '#A= ',A,'  B= ',B
            WRITE(OUP,*) '#TOL= ',TOL,'  N= ',N
            WRITE(OUP,*) '#MAX. ITERACII = ',LBOUND

            XL  = B-A !DOLZINA NA KLUPATA
            H   =(B-A)/N!CEKOR NA INTEGRACIJA
            H2  = H*H
            Y0  = 1.0E09*0.03D0**3*0.20D0/3.0D0!Y*I
            X0  = 0.25D0
            RHO = 3.0D0!GUSTINA NA KLUPATA
            G   = 9.8D0         !ZEMJINO ZABRZUVANJE
            F0  = 200.0D0!KONSTANTA
            E0  = 1.0D0/DEXP(1.0D0)

            NN=N-1
            NNN=N-2

C           IZVORNA FUNKCIJA
            DO 500 I = 1,NN
              XD   =  H*I
              S(I) = -RHO*G
              IF (DABS(XD-XL/2.0D0).LT.X0) THEN
                S(I) = S(I)-F0*(DEXP(-((XD-XL/2.D0)/X0)**2)-E0)
              ELSE
              ENDIF
```

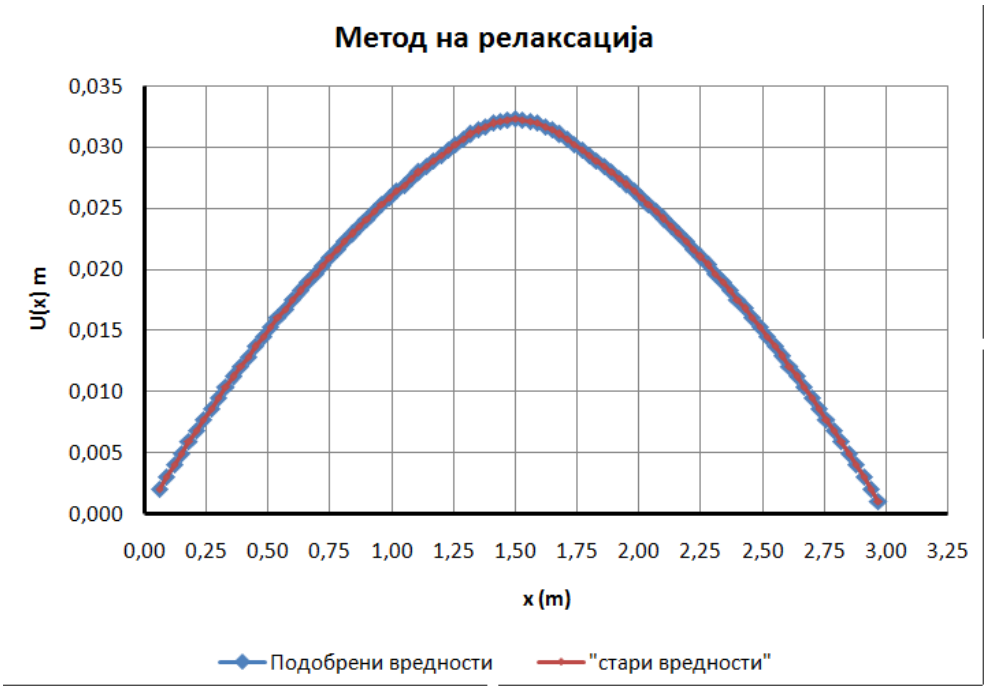
```
                        S(I) = S(I)/Y0
   5  0  0              CONTINUE

C                       GRANICNI USLOVI
                        S0=0.D0
                        SN=0.D0

C                       A,B KE SE KORISTAT ZA X(0),X(N)
C                       PRI PRAVENJE NA MREZA

                        DO 10 I=1,NN
   1  0                      X(I)=A+I*H
C
                         DO 30 I=1,NN
   3  0                      W(I)=0.D0

C                       SE KORISTI V ZA LAMBDA A VV ZA MU
                        VV=2.D0
                        L=1

C                       Z E NOVA VREDNOST NA W(I,J).
C                       ZA PRESMETUVANJE NA GRESKATA E SE KORISTI E NAMESTO NORM
   1  0  0              IF (L.GT.LBOUND) GOTO 200

C                       METOD NA GAUSS-ZAJDEL
                                  Z=(-H2*S(1)+S0+W(2))/VV
                                  E=DABS(W(1)-Z)
                                  W(1)=Z
C
                                  DO 40 I=2,NNN
                                       Z=(-H2*S(I)+W(I-1)+W(I+1))/VV
                                       IF(DABS(W(I)-Z).GT.E) E=DABS(W(I)-Z)
   4  0                       W(I)=Z
C
                                  Z=(-H2*S(NN)+SN+W(NNN))/VV
                                  IF(DABS(W(NN)-Z).GT.E) E=DABS(W(NN)-Z)
                                  W(NN)=Z
C
                                  IF(E.LE.TOL) THEN
                                       WRITE(OUP,2) L
                                       GOTO 400
                                  END IF
C                       KRAJ NA GAUSS-ZAJDEL
                             L=L+1
                             GOTO 100
   4  0  0              DO  600 I = 2,NNN
                                CALL RLXN(W,S,N,P,H,XR)
   6  0  0              CONTINUE
                        WRITE(OUP,3)
                        WRITE(OUP,4)  (I,H*I,W(I),XR(I),(W(I)-XR(I)),I=2,NN)
                        GOTO 700
   2  0  0              WRITE(OUP,1) LBOUND
   7  0  0              CLOSE(UNIT=5)
                        CLOSE(UNIT=OUP)
                        IF(OUP.NE.6) CLOSE(UNIT=6)
                        STOP
   1                    FORMAT('PROGRAMATA PREKINA DA RABOTI POSLE ITERACII:',I4)
   2                    FORMAT('#BROJ NA ITERACII :',I4)
   3                    FORMAT('#  I        X(I)                W(I)
     *                            XR(I)                DELW(I)')
   4                    FORMAT(I4,1X,F10.5,1X,F20.10,1X,F20.10,1X,F20.10)
                        END

C                       POTPROGRAMA ZA METOD NA RELAKSACIJA ZA
C                       EDNODIMENZIONALNA ELIPTICNA PD RAVENKA.
C
                        SUBROUTINE RLXN (FN,S,N,P,H,FNR)
                        IMPLICIT REAL*8 (A-H,P-Z)
                        PARAMETER (NR = 10000)
                        DIMENSION FN(NR),S(NR),FNR(NR)

                          H2 = H*H
                          Q  = 1.0D0-P
                          DO 100 I = 2, N-1
```

```
      FNR(I) = Q*FN(I)+P*(FN(I+1)+FN(I-1)-H2*S(I))/2.0D0
1 0 0 CONTINUE
      RETURN
      END
```

Дел од излезот на програмата relax.for:

```
...
#BROJ NA ITERACII :3489
#  I       X(I)              W(I)              XR(I)             DELW(I)
   2    0.06000         0.0020075552      0.0020076178      -0.0000000626
   3    0.09000         0.0029894113      0.0029894946      -0.0000000833
   4    0.12000         0.0039566956      0.0039567995      -0.0000001040
   5    0.15000         0.0049094398      0.0049095643      -0.0000001245
   6    0.18000         0.0058476755      0.0058478203      -0.0000001448
   7    0.21000         0.0067714340      0.0067715990      -0.0000001650
   8    0.24000         0.0076807463      0.0076809314      -0.0000001850
   9    0.27000         0.0085756434      0.0085758482      -0.0000002048
  10    0.30000         0.0094561556      0.0094563800      -0.0000002244
  ...
  85    2.55000         0.0136598660      0.0136601379      -0.0000002719
  86    2.58000         0.0128502725      0.0128505260      -0.0000002535
  87    2.61000         0.0120263690      0.0120266038      -0.0000002348
  88    2.64000         0.0111881267      0.0111883427      -0.0000002160
  89    2.67000         0.0103355167      0.0103357136      -0.0000001969
  90    2.70000         0.0094685096      0.0094686873      -0.0000001777
  91    2.73000         0.0085870759      0.0085872342      -0.0000001583
  92    2.76000         0.0076911858      0.0076913246      -0.0000001388
  93    2.79000         0.0067808092      0.0067809284      -0.0000001192
  94    2.82000         0.0058559160      0.0058560154      -0.0000000994
  95    2.85000         0.0049164757      0.0049165553      -0.0000000796
  96    2.88000         0.0039624580      0.0039625177      -0.0000000598
  97    2.91000         0.0029938322      0.0029938720      -0.0000000399
  98    2.94000         0.0020105677      0.0020105876      -0.0000000199
  99    2.97000         0.0010126338      0.0010126338       0.0000000000
```

Програмата relax.for создава датотека чие име и пат го дефинираме во самата програма, на пример c:/relax.txt. Графичкиот приказ на оваа датотека, кога параметарот на релаксација е $\omega = 1.3$ прикажан е на слика 9.3.



Сл. 9.3. Графички приказ на резултатите добиени со метод на релаксација.

Како дополнителна задача, корисно е да се види како се менува брзината на конвергенција во зависност од параметарот на релаксација.

9.3. Задачи

1. Да се реши приближно елиптичната парцијална диференцијална равенка

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2}=-12x^2,$$

со гранични услови $u(0)=0$ и $u(1)=0$? Приближното решение да се спореди со точното $u_{exe}=x(1-x^3)$. Да се испита влијанието на параметаропт на релаксација врз брзината на конвергенција?

2. Да се реши приближно елиптичната парцијална диференцијална равенка

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2}=6x,$$

со гранични услови $u(-1)=-6$ и $u(1)=6$? Приближното решение да се спореди со точното $u_{exe}=6x^3$. Да се испита влијанието на параметаропт на релаксација врз брзината на конвергенција?

9.4. Проекти

1. Да се реши дводимензионалната Лапласова равенка

$$\frac{\partial^2u}{\partial x^2}+\frac{\partial^2u}{\partial y^2}=0,$$

со која се прикажува распределбата на топлината на дводимензионална метална плоча со ширина $a=0.5\mathrm{m}$ и должина $b=0.5\mathrm{m}$? Граничните услови се:

$$u(x,y)=0 \qquad u(x,0)=0$$
$$u(x,0.5)=200x \qquad u(0.5,y)=200y.$$

2. Да се реши дводимензионалната Поасонова равенка

$$\frac{\partial^2u}{\partial x^2}+\frac{\partial^2u}{\partial y^2}=x\,e^y,$$

за која $0<x<2$ и $0<y<1$, а граничните услови се:

$$u(0,y)=0 \qquad u(2,y)=2e^y \qquad 0\le y\le 1$$
$$u(x,0)=x \qquad u(x,1)=e\,x \qquad 0\le x\le 2.$$

10. ПАРАБОЛИЧНИ ПАРЦИЈАЛНИ ДИФЕРЕНЦИЈАЛНИ РАВЕНКИ

Параболичните парцијални диференцијални равенки вклучуваат изводи од втор ред со ист знак за секоја од независните просторни променливи и еден извод од прв ред по времето. Во оваа класа спаѓаат временски зависната Шредингерова равенка

$$i\hbar\frac{\partial\Psi}{\partial t}(r,t)=\left[-\frac{\hbar^2}{2m}\nabla^2+V(r,t)\right]\Psi(r,t), \tag{10.1}$$

каде $\Psi(r,t)$ е бранова функција чија временска еволуција се определува и равенката за топлоспроводливост или дифузија

$$\frac{\partial\varphi}{\partial t}(x,t)=\alpha^2\frac{\partial^2\varphi}{\partial x^2}(x,t), \tag{10.2}$$

каде $t>0$ и $0<x<l$ и соодветни гранични услови.

10.1. Експлицитна шема

И во овој случја, прв чекор во нумеричкото решавање на параболичните парцијалните диференцијални равенки е да се доведат во форма погодна за нумерички пресметувања. Заради тоа, ќе направиме дискретизација на континуалните променливи или поточно просторните координати и времето со што првиот и вториот извод ќе ги прикажиме со формулите за конечни разлики во две и три точки (9.5) и (9.7), како што беше покажано во претходното поглавје.

Според ова, константата на решетката по x оската h ги дефинира еквидистантните јазли како $x_i=a+ih$, за $i=0,1,\ldots m$, а константата на решетката по y оската k ги дефинира еквидистантните времемски интервали $y_j=b+jk$, за $j=0,1,\ldots$.

Имајќи го предвид ова, диференцната форма на (10.2) за точката (јазол) (x_i,t_j) за секое $i=1,2,\ldots m-1$ и $j=1,2,\ldots$, може да се запише како

$$\frac{\varphi_{i,j+1}-\varphi_{i,j}}{k}-\alpha^2\frac{\varphi_{i+1,j}+\varphi_{i-1,j}-2\varphi_{ij}}{h^2}=0, \tag{10.3}$$

или ако ја решиме по $\varphi_{i,j+1}$ имаме

$$\varphi_{i,j+1}=\left(1-\frac{2\alpha^2k}{h^2}\right)\varphi_{i,j}+\alpha^2\frac{k}{h^2}(\varphi_{i+1,j}+\varphi_{i-1,j}), \tag{10.4}$$

каде граничните услови се:

$$\varphi_{0,j}=g(x_0,y_j)=0 \qquad \varphi_{n,j}=g(x_n,y_j)=0$$
$$\varphi_{i,0}=f(x_i), \tag{10.5}$$

при што $i=1,2\ldots(m-1)$ и $j=1,2\ldots.$ Ова ни дава за право да заклучиме дека за $j=1$, со замена на граничните вредности $\varphi_{i,0}=f(x_i)$ во (10.4) може да се определи временската еволуција $\varphi_{i,1}$ за секое $i=1,2,\ldots(m-1)$ на што треба да се додадат и граничните вредности $\varphi_{0,1}=\varphi_{0,m}=0$. Со повторување на оваа постапка за останатите вредности на $j=2,\ldots$ може да се определи временската еволуција на $\varphi(x,t)$.

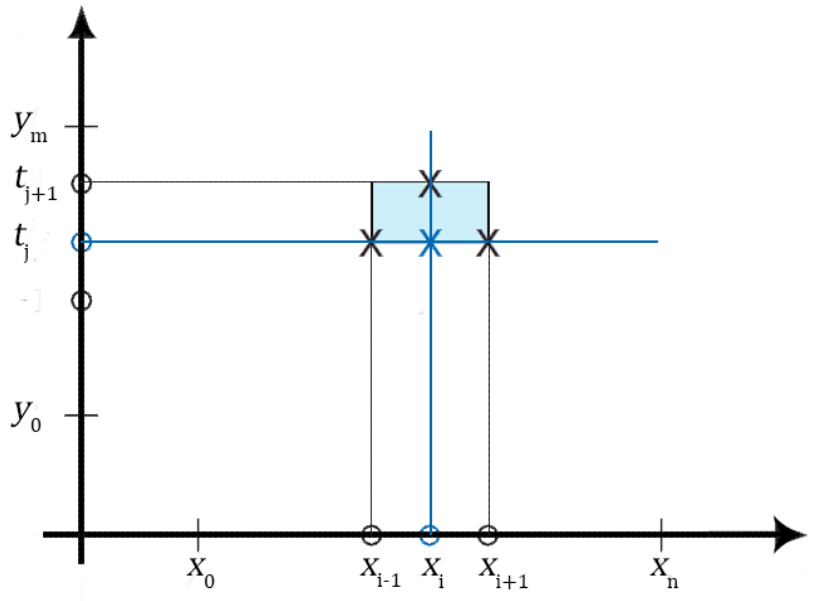
Оваа експицитна шема покажува дека системот равенки (10.4) може да се претстави со помош на тридијагонална матрица

$$W=\begin{bmatrix}(1-2\lambda) & \lambda & 0 & \cdots & 0\\ \lambda & (1-2\lambda) & \lambda & \ddots & \vdots\\ 0 & \ddots & \ddots & \ddots & 0\\ \vdots & \ddots & \ddots & \ddots & \lambda\\ 0 & \cdots & 0 & \lambda & (1-2\lambda)\end{bmatrix}, \tag{10.6}$$

како

$$\varphi^{(j)}=W\varphi^{(j-1)}, \tag{10.7}$$

за $j=1,2,\ldots.$ Оваа шема е позната и како *метод на конечни разлики напред* што графички може да се прикажи како на слика 10.1.



Сл. 10.1. Графички приказ на умрежување на експлицитната шема.

Имајќи ги предвид формулите со кои изводите се пресметуваат со конечни разлики во две и три точки, точноста на овој метод е со ред на големина $O(k+h^2)$, но има проблем со стабилноста или е *условно стабилен*. Тоа значи дека за да се добие точното решение потребно е да биде задоволен некој услов. Не навлегувајќи во деталите

на изведувањето, условот за стабилноста на експлицитната шема се сведува на тоа да сопствените вредности $\mu_i = 1 - 4\lambda(\sin(i\pi/2m))^2$ на (10.6) да го задоволуваат неравенството $\max|\mu_i| \le 1$, за $1 \le i \le m-1$, што дополнително може да се упрости до обликот

$$0 \le \lambda \le \frac{1}{2}. \tag{10.8}$$

Имајќи предвид дека $\lambda = \alpha^2(k/h^2)$, значи дека стабилноста на експлицитна шема зависи од вредностите на параметрите на ППДР. Во случајот кога $\alpha = 1$, $h = 0.1$ и $k = 0.0005$ условот (10.8) е задоволен, $\lambda = 0.05$. Но ако временскиот чекор го зголемиме на $k = 0.01$, условот за стабилност е нарушен, $\lambda = 1$. Условната стабилност на експлицитната шема најдобро се гледа на примерот што следи.

Пример 10.1.

Како се применува диференцната формула (10.4), или експлицитниот метод за приближно решавање параболична парцијална диференцијална равенка (ППДР), ќе покажиме со примерот на равенката (10.2), во случај кога $\alpha = 1$. Почетните вредности и граничните услови се дефинираат согласно (10.5) како

$$\begin{aligned} &\varphi_{0,j} = 0, \quad \varphi_{n,j} = 0, \quad t > 0, \\ &\varphi_{i,0} = \sin(\pi x) \quad 0 \le x \le 1, \end{aligned} \tag{10.9}$$

а точното решение на равенката со кое ќе ги споредуваме нумеричките резултати е $u(x,t) = e^{-\pi^2 t}\sin(\pi x)$.

Програмата explicit.for, за приближно решавање на параболичната ПДР ја користи експлицитна шема, при што предвид се земаат следниве врдности на променливите: конечното време е $t = 0.5$, просторниот чекор е $h = 0.1$, временскиот чекор на еволуција е $k = 0.001$ а параметарот $\lambda = \alpha^2(k/h^2) = 0.25$.

```
      PROGRAM  PARAB_EXPLICIT_METHOD
      IMPLICIT REAL*8 (A-H,P-Z)
      DIMENSION X(0:100), U(0:100,2)
      CHARACTER NAME1*30,AA*1
      LOGICAL OK

      F(XZ)=DSIN(PI*XZ)
      PI=4.D0*ATAN(1.0D0)

      WRITE(6,*) 'Metod na konecni razliki napred'
      WRITE(6,*) 'r-ka za toplosprovodlivost.'
      WRITE(6,*) 'dali funkcijata F da bide sozdadena? '
      WRITE(6,*) 'vnesi Y ili N '
      WRITE(6,*) ' '
      READ(5,*)  AA
      IF(( AA .EQ. 'Y' ) .OR. ( AA .EQ. 'y' )) THEN
      OK = .FALSE.
      WRITE(6,*) 'Levata krajna tocka na X-oskata e 0.'
1 9   IF (OK) GOTO 11
        WRITE(6,*) 'Vnesi ja desnata krajna tocka na X-oskata.'
        WRITE(6,*) 'na primer: 1.0.'
        WRITE(6,*) ' '
        READ(5,*) FX
        IF (FX.lE.0.0) THEN
           WRITE(6,*) 'Mora da e pozitiven broj!'
        ELSE
           OK = .TRUE.
```

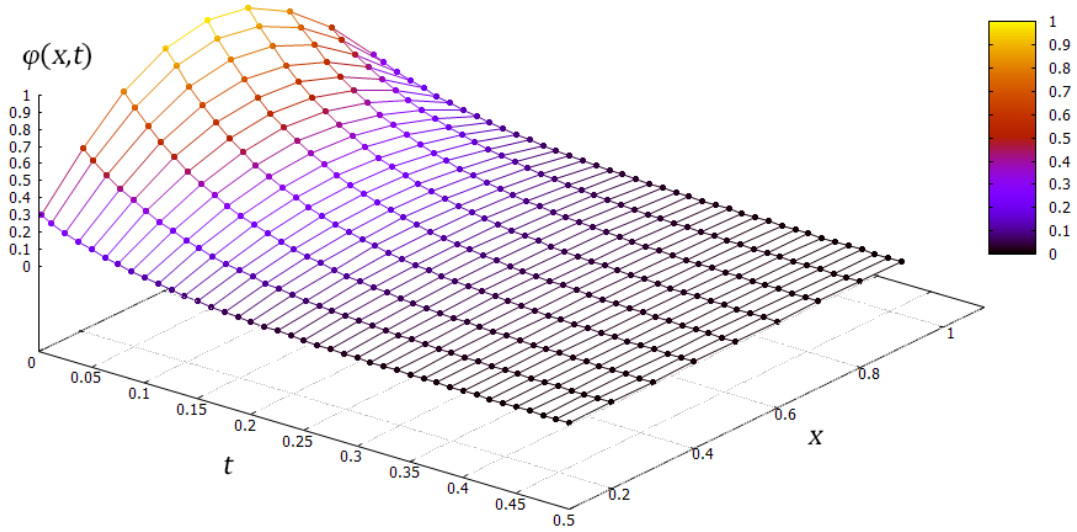
```
      ENDIF
      GOTO 19
1  1  OK = .FALSE.
1  2  IF (OK) GOTO 13
         WRITE(6,*) 'Vnesi go maksimalnoto vreme za iteracii'
         WRITE(6,*) 'Tmax.'
         WRITE(6,*) 'na primer: 0.5.'
         WRITE(6,*) ' '
         READ(5,*) FT
         IF ( FT .LE. 0.0 ) THEN
             WRITE(6,*) 'Mora da e pozitiven broj!'
         ELSE
           OK = .TRUE.
         ENDIF
      GOTO 12
1  3  WRITE(6,*) 'Vnesi ja konstantata Alfa.'
      WRITE(6,*) 'na primer: 1.0.'
      WRITE(6,*) ' '
      READ(5,*) ALPHA
1  5     OK = .FALSE.
1  4  IF (OK) GOTO 16
      WRITE(6,*) 'Vnesi gi M (broj an intervali) '
      WRITE(6,*) 'na X-oska i N (broj na vremenski intervali)'
      WRITE(6,*) 'M mora da bide 3 ili pogolemo.'
      WRITE(6,*) 'odvoeni so blanko. '
      WRITE(6,*) 'na primer: 10 1000.'
      READ(5,*) M,N
      IF ((M.LE.2).OR.(N.LE.0)) THEN
         WRITE(6,*) 'Vrednostite ne se vo predvidenite intervali.'
      ELSE
         OK = .TRUE.
      ENDIF
      GOTO 14
1  6  CONTINUE
      ELSE
      WRITE(6,*) 'Programata ke zavrsi taka sto funkcijata F '
      WRITE(6,*) 'ke bide sozdadena.'
      OK = .FALSE.
      ENDIF
      IF(.NOT.OK) GOTO 400
      WRITE(6,*) 'Odberi vid na izlez: '
      WRITE(6,*) '1. Ekran '
      WRITE(6,*) '2. Datoteka '
      WRITE(6,*) 'Enter 1 or 2 '
      WRITE(6,*) ' '
      READ(5,*) NFLAG
      IF ( NFLAG .EQ. 2 ) THEN
      WRITE(6,*) 'Vnesi go imeto na datotekata kako - '
      WRITE(6,*) 'drive:name.ext'
      WRITE(6,*) 'taka sto patot ke sodrzi navodnici'
      WRITE(6,*) 'na primer: "c:/OUTPUT.DTA"'
      WRITE(6,*) ' '
      READ(5,*) NAME1
      OUP = 5
      OPEN(UNIT=OUP,FILE=NAME1,STATUS='UNKNOWN')
      ELSE
      OUP = 6
      ENDIF
      WRITE(OUP,*)'      #METOD NA KONECNI RAZLIKI NAPRED ZA PPDR'
      WRITE(OUP,3)

      H=FX/M
      H2=H*H
      DT=FT/N

      CF=ALPHA*DT/H2

C     ZA  T=0 (I=1) SITE TOCKI IMAAT POCETNI VREDNOSTI
      DO 10  I=1,M-1
      X(I)=I*H
      U(I,1) = F(X(I))
1  0  CONTINUE

C     OSVEN KRAJNITE TOCKI SE NULA
```

```
                DO 20  I=1,2
                U(0,I)=0.0
                U(M,I)=0.0
   2  0         CONTINUE

C               ZAPOCNUVA RESAVANJETO PO VREMETO
                DO 100   K=1,N

C               CIKLUS PO PROSTRNA KOORDINATA,KRAJNITE TOCKI SE FIKSNI
                DO 30  I=1,M-1
                U(I,2)=(1.D0-2.D0*CF)*U(I,1)+CF*(U(I+1,1)+U(I-1,1))
   3  0         CONTINUE

C               TEMPERATURATA SE OPREDELUVA NA SEKOI 1000 VREMESNKI CEKORI
                IF((MOD(K,100).EQ.0).OR.(K.EQ.1))THEN
                DO 40  I=0,M
                WRITE(OUP,2) K*DT,X(I),U(I,2)
   4  0         CONTINUE
                WRITE (OUP,2)
                ENDIF

C               NOVITE VREDNOSTI SE ZAMENUVAAT ZA STARI
                DO 50  I=1,M-1
                U(I,1) =  U(I,2)
   5  0         CONTINUE

   1  0  0      CONTINUE

   4  0  0      CLOSE(UNIT=5)
                CLOSE(UNIT=OUP)
                IF(OUP.NE.6) CLOSE(UNIT=6)
                STOP

   1            FORMAT(1X,'KRAJNOTO VREME E',1X,E15.8)
   2            FORMAT(1X,3(1X,F15.10))
   3            FORMAT(9X,'#T(J)',12X,'X(I)',12X,'W(I)')

                END
```

Дел од излезот на програмата relax.for:

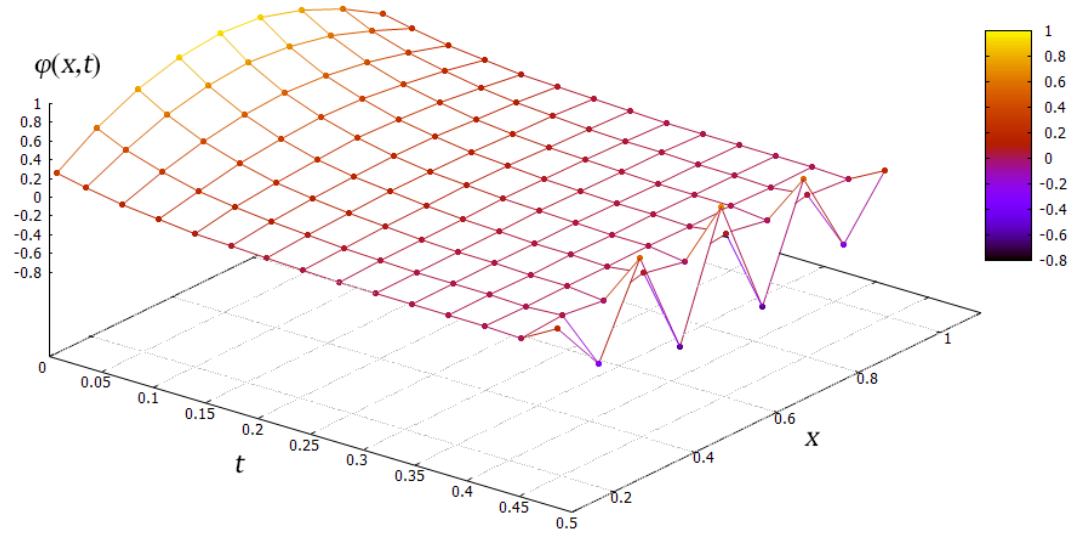
```
       #METOD NA KONECNI RAZLIKI NAPRED ZA PPDR
         #T(J)             X(I)             W(I)
      0.50000E-03     0.00000E+00     0.00000E+00
      0.50000E-03     0.10000E+00     0.30750E+00
      0.50000E-03     0.20000E+00     0.58491E+00
      0.50000E-03     0.30000E+00     0.80506E+00
      0.50000E-03     0.40000E+00     0.94640E+00
      0.50000E-03     0.50000E+00     0.99511E+00
      0.50000E-03     0.60000E+00     0.94640E+00
      0.50000E-03     0.70000E+00     0.80506E+00
      0.50000E-03     0.80000E+00     0.58491E+00
      0.50000E-03     0.90000E+00     0.30750E+00
      0.50000E-03     0.00000E+00     0.00000E+00

      ...

      0.50000E+00     0.00000E+00     0.00000E+00
      0.50000E+00     0.10000E+00     0.22865E-02
      0.50000E+00     0.20000E+00     0.43492E-02
      0.50000E+00     0.30000E+00     0.59862E-02
      0.50000E+00     0.40000E+00     0.70372E-02
      0.50000E+00     0.50000E+00     0.73993E-02
      0.50000E+00     0.60000E+00     0.70372E-02
      0.50000E+00     0.70000E+00     0.59862E-02
      0.50000E+00     0.80000E+00     0.43492E-02
      0.50000E+00     0.90000E+00     0.22865E-02
      0.50000E+00     0.00000E+00     0.00000E+00
```

За подобра нагледност на добиените резултати, направен е график, слика 10.2, на кој се прикажани сите нумерички решенија $\varphi(x_i,t_j)\equiv\varphi_{ij}$ за $0\le x\le 1$ и $0\le t\le 0.5$ во услови кога методот е стабилен, односно кога е задоволен условот (10.8).



Сл. 10.2. Графички приказ на нумеричките решенија φ_{ij} кога експлицитната шема е стабилна.

Веќе спомнавме дека стабилноста на експлицитната шема се нарушува ако, на пример, го зголемиме временскиот чекор $k=0.0069$, а останатите параметри останат не променети. Во тој случај имаме $\lambda=0.69$, што значи дека условот (10.8) не е задоволен. Решенијата φ_{ij} за овој случај се прикажани на слика 10.3 каде може да се забележи дека отстапување од физичкото поведение имаме после $t\approx 0.4$ а може да се случи и порано со соодветен избор на параметрите *t*, *k* и *h*.



Сл. 10.3. Графички приказ на нумеричките решенија φ_{ij} кога експлицитната шема е не стабилна.

10.2. Имплицитна шема

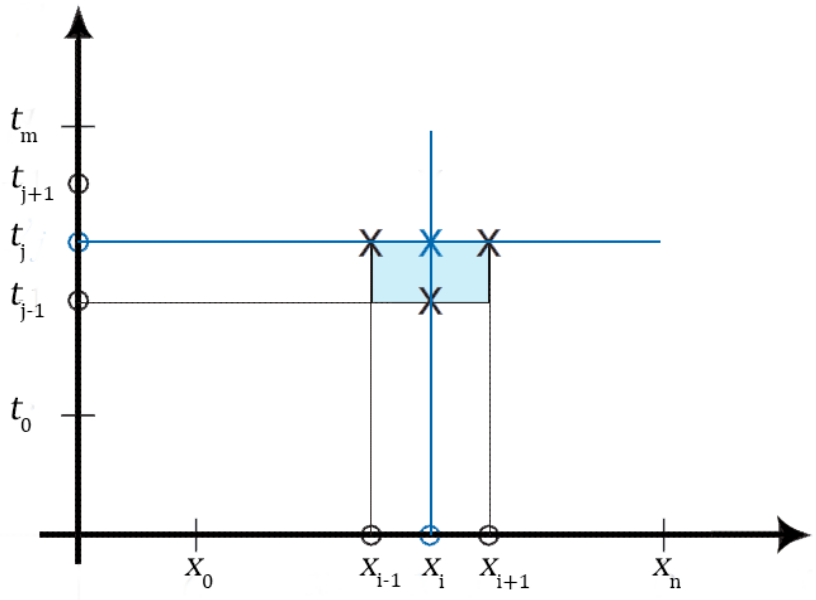
За да добиеме метод што е безусловно стабилен за нумеричко решавање на ППДР, ќе ја искористиме формулата за диференцирање назад со што диференцната форма на (10.2) ќе ја запишиеме како

$$\frac{\varphi_{i,j}-\varphi_{i,j-1}}{k}-\alpha^2\frac{\varphi_{i+1,j}+\varphi_{i-1,j}-2\varphi_{ij}}{h^2}=0\,, \tag{10.10}$$

за секое $i=1,2,\ldots m-1$ и $j=1,2,\ldots$ Ако повторно ја воведиме ознаката $\lambda=\alpha^2(k/h^2)$, горниот израз може да го презапишиме како

$$(1+2\lambda)\varphi_{i,j}-\lambda\varphi_{i+1,j}-\lambda\varphi_{i-1,j}=\varphi_{i,j-1}\,, \tag{10.11}$$

за кој важат истите гранични услови дефинирани со (10.5). Апроксимацијата на временската еволуција во дадена точка што ја дава *методот на конечни разлики назад* графички може да се прикажи како на слика 10.4.



Сл. 10.4. Графички приказ на умрежувањето на имплицитната шема.

Оваа шема овозможува системот равенки (10.11) да се претстави со помош на тридијагонална матрица во облик

$$W\varphi^{(j)}=\varphi^{(j-1)}\,, \tag{10.12}$$

за $j=1,2,\ldots.$, каде матрицата W е дефинирана како

$$W=\begin{bmatrix} (1+2\lambda) & -\lambda & 0 & \cdots & 0 \\ -\lambda & (1+2\lambda) & -\lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\lambda \\ 0 & \cdots & 0 & -\lambda & (1+2\lambda) \end{bmatrix}. \qquad (10.13)$$

Формулата (10.12) е позната како *имплицитна шема* за нумеричко решавање на ППДР. Таа има иста точност како и експлицитната шема, $O(k+h^2)$ но е безусловно стабилна. Имено, бидејќи $\lambda>0$, матрицата W е тридијагонална па за решавање на системот равенки може да се искористи некој директен метод или пак итеративниот SOR метод.

Пример 10.2.

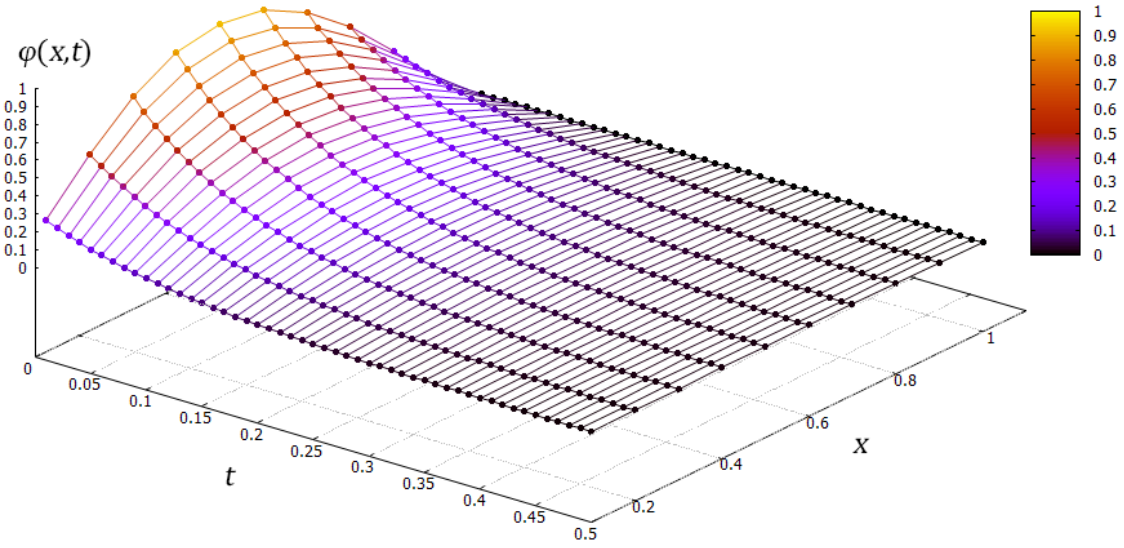
За да покажиме како делува имплицитната шема за приближно решавање на параболични ПДР, повторно ќе го искористиме примерот 10.1. Некои параметри ќе имаат иста вредност, $t=0.5$ и $h=0.1$, додека за врменскиот чекор ќе земиме $k=0.01$. Приближните решенијата φ_{ij} се добиени со програмата implicit.for прикажана подолу.

```
      PROGRAM PARAB_IMPLICIT_METHOD
      IMPLICIT REAL*8 (A-H,P-Z)
      PARAMETER (NX=10)
      DIMENSION W(0:NX),XL(NX-1),XU(NX-1),Z(NX-1)
      CHARACTER NAME1*30,AA*1
      LOGICAL OK

      F(XZ)=DSIN(PI*XZ)
      PI=4.D0*ATAN(1.0D0)

      WRITE(6,*) 'Metod na konecni razliki nazad'
      WRITE(6,*) 'r-ka za toplosprovodlivost.'
      WRITE(6,*) 'dali funkcijata F da bide sozdadena? '
      WRITE(6,*) 'vnesi Y ili N '
      WRITE(6,*) ' '
      READ(5,*)  AA
      IF(( AA .EQ. 'Y' ) .OR. ( AA .EQ. 'y' )) THEN
      OK = .FALSE.
      WRITE(6,*) 'Levata krajna tocka na X-oskata e 0.'
1  9  IF (OK) GOTO 11
      WRITE(6,*) 'Vnesi ja desnata krajna tocka na X-oskata.'
      WRITE(6,*) 'na primer: 1.0.'
      WRITE(6,*) ' '
      READ(5,*) FX
      IF (FX.lE.0.0) THEN
      WRITE(6,*) 'Mora da e pozitiven broj!'
      ELSE
      OK = .TRUE.
      ENDIF
      GOTO 19
1  1  OK = .FALSE.
1  2  IF (OK) GOTO 13
      WRITE(6,*) 'Vnesi go maksimalnoto vreme za iteracii'
      WRITE(6,*) 'Tmax.'
      WRITE(6,*) 'na primer: 0.5.'
      WRITE(6,*) ' '
      READ(5,*) FT
      IF ( FT .LE. 0.0 ) THEN
      WRITE(6,*) 'Mora da e pozitiven broj!'
      ELSE
      OK = .TRUE.
      ENDIF
      GOTO 12
1  3  WRITE(6,*) 'Vnesi ja konstantata Alfa.'
```

```
      WRITE(6,*) 'na primer: 1.0.'
      WRITE(6,*) ' '
      READ(5,*) ALPHA
1  5  OK = .FALSE.
1  4  IF (OK) GOTO 16
      WRITE(6,*) 'Vnesi gi M (broj an intervali) '
      WRITE(6,*) 'na X-oska i N (broj na vremenski intervali)'
      WRITE(6,*) 'M mora da bide 3 ili pogolemo.'
      WRITE(6,*) 'odvoeni so blanko. '
      WRITE(6,*) 'na primer: 10 50.'
      READ(5,*) M,N
      IF ((M.LE.2).OR.(N.LE.0)) THEN
      WRITE(6,*) 'Vrednostite ne se vo predvidenite intervali.'
      ELSE
      OK = .TRUE.
      ENDIF
      GOTO 14
1  6  CONTINUE
      ELSE
      WRITE(6,*) 'Programata ke zavrsi taka sto funkcijata F '
      WRITE(6,*) 'ke bide sozdadena.'
      OK = .FALSE.
      ENDIF
      IF(.NOT.OK) GOTO 400
      WRITE(6,*) 'Odberi vid na izlez: '
      WRITE(6,*) '1. Ekran '
      WRITE(6,*) '2. Datoteka '
      WRITE(6,*) 'Enter 1 or 2 '
      WRITE(6,*) ' '
      READ(5,*) NFLAG
      IF ( NFLAG .EQ. 2 ) THEN
      WRITE(6,*) 'Vnesi go imeto na datotekata kako - '
      WRITE(6,*) 'drive:name.ext'
      WRITE(6,*) 'taka sto patot ke sodrzi navodnici'
      WRITE(6,*) 'na primer: "c:/OUTPUT.DTA"'
      WRITE(6,*) ' '
      READ(5,*) NAME1
      OUP = 5
      OPEN(UNIT=OUP,FILE=NAME1,STATUS='UNKNOWN')
      ELSE
      OUP = 6
      ENDIF
      WRITE(OUP,*)'      #METOD NA KONECNI RAZLIKI NAZAD ZA PPDR'
      WRITE(OUP,3)

      MM=M-1
      MMM=MM-1
      NN=N-1
C
      H=FX/M
C     SE KORISTI TK NAMESTO K
      TK=FT/N
C     SE KORISTI VV ZA LAMBDA
      VV=ALPHA*ALPHA*TK/(H*H)
C
C     POCETNI VRDNOSTI
      W(0)=0.D0
      W(M)=0.D0
      DO 10 I=1,MM
1  0  W(I)=F(I*H)
C     SE RESAVA TRIDIJAGONALEN SISTEM OD RAVENKI
C     SE KORISTI XL FOR L, XU U
      XL(1)=1+2*VV
      XU(1)=-VV/XL(1)
C
      DO 20 I=2,MMM
      XL(I)=1+2*VV+VV*XU(I-1)
2  0  XU(I)=-VV/XL(I)
C
      XL(MM)=1+2*VV+VV*XU(MMM)
C
      DO 30 J=1,N
C     TEKOVNO VREME T(J)
      T=J*TK
```

```
      Z(1)=W(1)/XL(1)
C
      DO 40 I=2,MM
   4  0  Z(I)=(W(I)+VV*Z(I-1))/XL(I)
C
      W(MM)=Z(MM)
C
      DO 50 II=1,MMM
      I=MMM-II+1
   5  0  W(I)=Z(I) -XU(I)*W(I+1)
C
      DO 60 I=0,M
      X=I*H
   6  0  WRITE(OUP,2) T,X,W(I)
      WRITE(OUP,2)

   3  0  CONTINUE
C
   4  0  0  CLOSE(UNIT=5)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.6) CLOSE(UNIT=6)
      STOP
   1  FORMAT(1X,'KRAJNOTO VREME E',1X,E15.8)
   2  FORMAT(1X,3(1X,F15.10))
   3  FORMAT(9X,'#T(J)',12X,'X(I)',12X,'W(I)')

      END
```

Ако го погледнеме графичкиот приказ на приближните решенијата φ_{ij} прикажани на слика 10.5 лесно ќе забележиме дека сите решенија имаат физичко поведение во целиот интервал на временската еволуција $0 \le t \le 0.5$, иако вредноста на временскиот чекор е значително голема $k = 0.01$. Ова укажува на фактот дека имплицитната шема е безусловно стабилна а со тоа и покорисна во приближното решавање на параболични ПДР.



Сл. 10.5. Графички приказ на нумеричките решенија φ_{ij} добиени со имплицитната шема.

10.3. Временски зависна Шредингерова равенка

Кога сакаме да ја опишиме просторната и временската локализираност на една квантна честица, на пример електрон затворен во потенцијална јама, се користиме со временски зависна Шредингерова равенка, која во една димензија може да се запише како

$$i\hbar\frac{\partial\psi(x,t)}{\partial t}=H\psi(x,t), \tag{10.14}$$

каде Хамилтоновиот оператор H се дефинира како

$$H=-\frac{1}{2m}\frac{\partial^2}{\partial x^2}+V(x). \tag{10.15}$$

Во овој случај m е маса на честицата а $V(x)$ е потенцијал во кој се движи честицата. Заради поедноставување на проблемот, ќе претпоставиме дека $\hbar=1$ и $2m=1$.

Од квантната механика знаеме дека брановата функција како решение на временски зависната Шредингерова равенка всушност е комплексна функција што значи дека при нумеричкото разрешување треба да се користиме со комплексни броеви што без проблем може да се користат и во програмскиот јазик Фортран. Сепак, во оваа прилика, ќе примениме друг приод и брановата функција ќе ја разложиме на реален и имагинарен дел:

$$\psi(x,t)=R(x,t)+i\,I(x,t). \tag{10.16}$$

Со замена на (10.16) во (10.14) добиваме систем од две симултани параболични ПДР:

$$\begin{aligned}\frac{\partial R(x,t)}{\partial t}&=-\frac{\partial^2 I(x,t)}{\partial x^2}+V(x)I(x,t)\\ \frac{\partial I(x,t)}{\partial t}&=\frac{\partial^2 R(x,t)}{\partial x^2}-V(x)R(x,t).\end{aligned} \tag{10.17}$$

За нивно разрешување ќе искористиме модифицирана експлицитна шема која е поефикасна затоа што ја избегнува инверзијата на големите матрици што е карактериситчна за имплицитните шеми за нумеричка интеграција на ППДР.

За разлика од приближното решавање на „обичните“ ППДР, кај временски зависната Шредингерова равенка треба да се задоволат неколку услови. Како прво, шемата да конвергира и да е безусловно стабилна, веројатноста да се запазува конечно долго време за што добра препорака е реланиот и имагинарниот дел од брановата функција да се пресметаат со мало временско поместување. Со тоа, реланиот дел $R(x,t)$ се пресметува во временските интервали $0,\Delta t,\ldots$, а имагинарниот дел во $\Delta t/2, 3\Delta t/2,\ldots$.

Со користење на развој во ред на Тејлор на реалниот и имагинарниот дел на брановата функција, ја добиваме диференцната форма на системот (10.17), која гласи:

$$\begin{aligned}R_i^{n+1}&=R_i^n-2\left\{\alpha\left[I_{i+1}^n+I_{i-1}^n\right]-2\left[\alpha+V_i\Delta t\right]I_i^n\right\}\\ I_i^{n+1}&=I_i^n+2\left\{\alpha\left[R_{i+1}^n+R_{i-1}^n\right]-2\left[\alpha+V_i\Delta t\right]R_i^n\right\},\end{aligned} \tag{10.18}$$

каде $\alpha = \Delta t/2(\Delta x^2)$. Со цел да се постигне поголема прецизност при пресметување на веројатноста, густината на веројатност се пресметува со изразот

$$\rho(t) = \begin{cases} R^2(t) + I(t+\frac{\Delta t}{2})I(t-\frac{\Delta t}{2}) & t = 0, \Delta t, \ldots \\ I^2(t) + R(t+\frac{\Delta t}{2})R(t-\frac{\Delta t}{2}) & t = \frac{\Delta t}{2}, 3\frac{\Delta t}{2}, \ldots \end{cases}, \tag{10.19}$$

што се сведува на вообичаената вредност $\rho(t) = R^2(t) + I^2(t)$ кога $\Delta t \to 0$.

Како почетна бранова функцица која ќе ја опишува состојбата на електронот во почетниот момент $t = 0$ земаме Гаусиан центриран во положбата $x = 5$ помножен со рамен бран:

$$\psi(x,0) = \exp\left[-\frac{1}{2}\left(\frac{x-5.0}{\sigma_0}\right)^2\right]\exp\left(i\,k_0\,x\right), \tag{10.20}$$

каде σ_0 е почетната ширина на брановиот пакет а k_0 почетната вредност на брановиот број. Бидејќи разгледуваме движење на квантана честица во бесконечно висока правоаголна јама, потенцијалот се задава со изразот

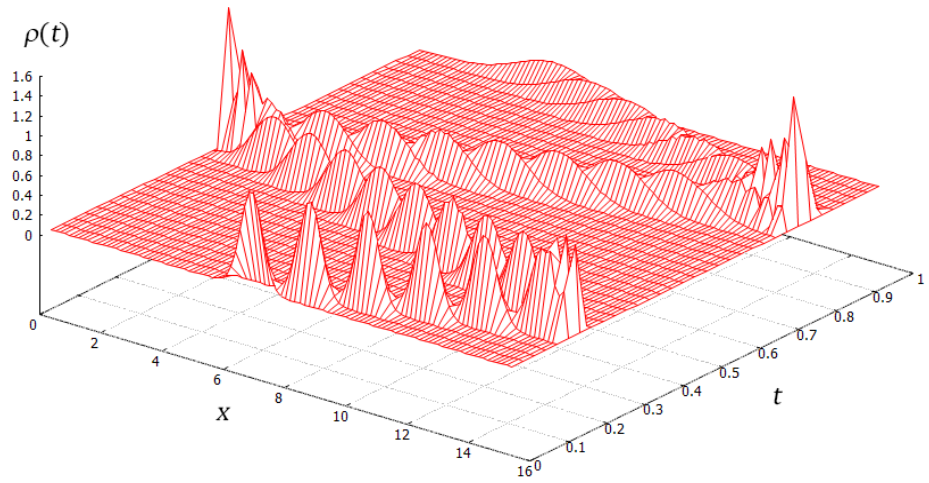
$$V(x) = \begin{cases} \infty & x > 15 \\ 0 & 0 \le x \le 15 \\ \infty & x < 0 \end{cases}.$$

Програмата schr_sqw.for со која се пресметува приближно просторно-временската еволуција на брановата функција е прикажана подолу во текстот.

```
      PROGRAM SCHR_SQW

      IMPLICIT NONE

      REAL*8 PSR(751,2),PSI(751,2),P2(751)
      REAL*8 DX,K0,DT,X,PI
      INTEGER I,N,MAX

      COMPLEX   EXC,ZI

      COMMON /VALUES/DX,DT

      OPEN(9,FILE='SQWELL.DAT',STATUS='UNKNOWN')

      MAX= 750
      PI = 3.14159265358979323846D0
      ZI = DCMPLX(0.0D0,1.D0)
      DX = 0.02D0
      K0 = 17.D0*PI
      DT  = DX*DX

C     POCETNI USLOVI

      X=0.0D0
      DO 10 I=1,MAX+1
      EXC=EXP(ZI*K0*X)

C     REALEN DEL OD BRANOVIOT PAKET ZA T=0
      PSR(I,1)=REAL(EXC*EXP(-0.5D0*(2.0D0*(X-5.0D0))**2.0))
C     IMAGINAREN DEL OD BRANOVIOT PAKET ZA T=0
      PSI(I,1)=IMAG(EXC*EXP(-0.5D0*(2.0D0*(X-5.0D0))**2.0))
      X=X+DX
   10 CONTINUE
```

```
                DO 40 N=1,6000

C               REALEN DEL OD BRANOVIOT PAKET I VEROJATNOSTA
                DO 50 I=2,MAX
                PSR(I,2)= PSR(I,1)-DT*(PSI(I+1,1)+PSI(I-1,1)
               *-2.0*PSI(I,1))/(2.0*DX*DX)
                P2(I)=PSR(I,1)*PSR(I,2)+PSI(I,1)*PSI(I,1)
   5  0         CONTINUE

C               IMAGINAREN DEL OD BRANOVIOT PAKER
                DO 60 I=2,MAX
                PSI(I,2)=PSI(I,1)+DT*(PSR(I+1,2)+PSR(I-1,2)
               *-2.D0*PSR(I,2))/(2.0D0*DX*DX)
   6  0         CONTINUE

C               VEROJATNOSTA KE SE VPISUVA SAMO ZA OPREDELENO VREME
                IF(MOD(N,300).EQ.0D0) THEN
                DO 80  I=1,MAX+1,15
                WRITE(9,11)P2(I)
   8  0         CONTINUE
                WRITE(9,11)
                ENDIF

C               NOVITE ITERACII GI PRAVIME STARI
                DO 70  I=1,MAX+1
                PSI(I,1) =  PSI(I,2)
                PSR(I,1) =  PSR(I,2)
   7  0         CONTINUE
   4  0         CONTINUE
   1  1         FORMAT (F12.6)
                CLOSE(9)
                STOP 'PODATOCITE SE ZACUVANI VO SQWELL.DAT'

                END
```

Вредностите на параметрите што се внесуваат во програмата се: ширина на брановиот пакет $\sigma_0 = 0.5$, просторен чекор $\Delta x = 0.02$, временски чекор $\Delta t = \Delta x^2/2$ и $k_0 = 17\pi$. Еволуцијата на брановиот пакет за определени временски интервали е прикажана на слика 10.6.



Сл. 10.6. Просторно-временска еволуција на бранов пакет (кванта честица) во бесконечно висока јама.

10.3. Задачи

1. Да се реши приближно параболичната парцијална диференцијална равенка

$$\frac{\partial u}{\partial t}=\frac{1}{\pi^2}\frac{\partial^2 u}{\partial x^2},$$

со гранични услови $u(0,t)=u(1,t)=0$ за $t>0$, и $u(x,0)=\cos[\pi(x-1/2)]$ за $0\le x\le 1.$ При пресметките да се земи $h=0.1$ и $k=0.04$. Приближното решение да се спореди со точното $u_{exe}(x,t)=e^{-t}\cos[\pi(x-1/2)]$ за $t=0.4$?

2. Разгледуваме хомогена прачка со должина $l=1\text{m}$ во која распределбата на температурата по нејзината должина се опишува со параболичната парцијална диференцијална равенка

$$\frac{\partial u}{\partial t}=0.02\frac{\partial^2 u}{\partial x^2}.$$

Граничните услови се $u(0,t)=0$ и $u(1,t)=100$ за $t>0$, и $u(x,0)=0$ за $0\le x\le 1.$ Со користење на имплицитниот метод да се најде приближното решение за распределбата на температурата и истото да се спореди со точното решение

$$u_{exe}(x,t)=100x+\frac{200}{\pi}\sum_{n=1}^{\infty}\frac{(-1)^n}{n}e^{-0.02n^2\pi^2 t}\sin(n\pi x).$$

10.4. Проекти

1. Да се реши нумерички еднодимензионална нестационарна равенка на Шредингер

$$i\hbar\frac{\partial\Psi}{\partial t}(x,t)=\left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+V(x,t)\right]\Psi(x,t),$$

во случаите кога потенцијалот се задава со изразите:

$$a)\quad V(x)=\begin{cases}0 & x>a/2\\ \pm V_0 & |x|<a/2\\ 0 & x<-a/2\end{cases}\quad\cdot$$

$$b)\quad V(x)=\frac{1}{2}k\,x^2\quad -\infty<x<\infty$$

11. МОНТЕ КАРЛО МЕТОДИ

Методот Монте Карло е еден од најголемите достигнувања во нумеричката анализа за приближно пресметување на повеќедимензионални интеграли и интегрални равенки. Методот започнал да се развива во 50-те години од минатиот век речиси заедно со развојот на компјутерската техника. Овој метод наоѓа посебна примена во областа на статистичката физика и квантната механика, но и во други области каде се решаваат проблеми со голем број степени на слобода како што е физиката на кондензирана материја.

Што е предноста на овој метод во однос на другите „стандардни“ техники ќе покажиме преку неговата примена во пресметување на повеќедимензионални интеграли. Да претпоставиме дека треба да пресметаме некои својства на релативно „мал“ атом на магнезиум кој има само 12 електрони. Ако интеграцијата за секој атом има по три координати, тогаш проблемот за целиот атом се сведува на $12\times 3=36$-кратна интеграција. Ако користиме 64 точки за секоја интеграција, вкупно ќе бидат потребни $N=n^d=64^{36}\approx 10^{65}$ итерации за да ја добиеме вредноста на интегралот. Ако претпоставиме дека имаме на располагање суперкомпјутер кој може да пресмета 10^6 операции во една секунда, тогаш за добивање на приближната вредност на интегралот ќе бидат потребни околу $10^{59}\,\mathrm{s}$, што е многу кратно повеќе од староста на Вселената која се проценува на $10^{17}\,\mathrm{s}$!

11.1. Метод на средна вредност

Наједноставниот метод за интегрирање со Монте Карло метод е заснован на теоремата за средна вредност на големи броеви која може да се запише како:

$$I=\int_a^b f(x)dx=(b-a)\langle f\rangle, \tag{11.1}$$

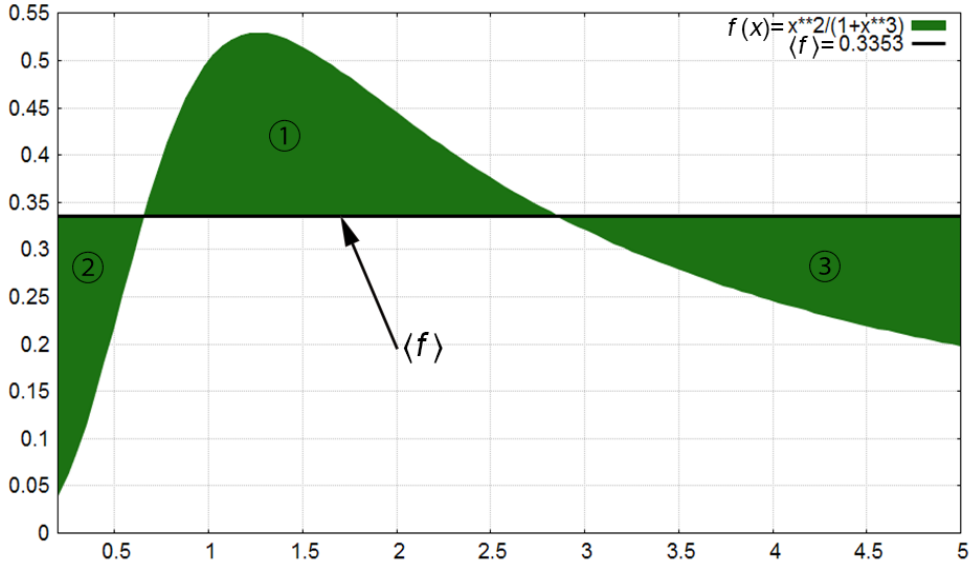
што ни кажува дека вредноста на интегралот во интервалот од a до b е еднаква на должината на интервалот $(b-a)$ помножена со средната вредност на функцијата во тој интервал $\langle f\rangle$, слика 11.1. Значи, за оваа техника важно е што поточно да се пресмета средната вредност, а за та цел може да се искористи добро позната формула од теоријата на веројатност

$$\langle f\rangle=\frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{11.2}$$

каде x_i се случајно генерирани броеви во интервалот $[a,b]$. Со ова, според Монте Карло методот вредноста на интегралот (11.1) може да се пресмета со изразот

$$I=\int_a^b f(x)dx=(b-a)\langle f\rangle=\frac{(b-a)}{N}\sum_{i=1}^{N}f(x_i)\,, \tag{11.3}$$

што наликува на интегрирање со метод на трапези, но апцисите x_i се случајни избрани со постојана тежина $\omega_i=(b-a)/N$.



Сл. 11.1. Функцијата $f(x)$ е прикажана во границите a=0.2 до b=5.0. Средната вредност на функцијата во тој интервал е $\langle f\rangle=0.3353$ и е означена со црна линија. Таа е повлечена така да површината ① над линијата е еднаква на збирот на површините ②+③.

Од претходниот израз јасно се забележува дека точноста на интегралот во оваа техника директно зависи од бројот N на избрани јазли на интеграција x_i, и во принцип ќе тежи кон точната вредност кога $N\to\infty$. Сепак ова е аналитички заклучок кој во нумеричката анализа се сведува на конечен број.

Од статистичката анализа на податоци е познато дека неопределеноста на вредноста на интегралот (11.3) после „земени“ N примероци на $f(x_i)$ се определува со стандардната девијација σ_I како:

$$\sigma_I\approx\frac{\sigma_f}{\sqrt{N}}=\frac{1}{\sqrt{N}}\sqrt{\frac{1}{N}\sum_{i=1}^{N}f_i^2-\left(\frac{1}{N}\sum_{i=1}^{N}f_i\right)^2}\,, \tag{11.4}$$

каде σ_f е стандардната девијација на функцијата кога секој примерок има иста вредност за реализација. Забележуваме дека грешката на методот се намалува како $N^{-1/2}$ што е значително полошо дури и од методот на трапези кој има грешка $\varepsilon\sim N^{2/d}$ каде d е димензијата на проблемот. Сепак, со зголемување на димензионалноста на проблемот, веќе за $d\geq 4$ методот Монте Карло е поточен! Во пракса, затоа што за споредба се земаат други класични методи на интеграција, методот Монте Карло се употребува кога $d\geq 8$.

Пример 11.1.

Со методот на средна вредност да се пресмета интегралот

$$\int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} \approx 0.78540 .$$

При пресметките да се земат $N = 100, 1000$ и 10000 точки (апциси x_i). Да се анализира точноста на методот во зависност од бројот на апциси? Како изборот на генераторот на случајни броеви влијае врз точноста на методот?

Програмата со која се пресметува интегралот е дадена во текстот подолу.

```
      PROGRAM MONTE_KARLO_1

      IMPLICIT REAL*8 (A-H,P-Z)

      DATA ISEED/987654321/     ! SEED ZA GENERATOROT NA SLUCAJNI BROEVI
      DATA EXACT/.78540/        ! TOCNA VREDNOST NA INTEGRALOT

      CHARACTER NAME1*30,AA*1
      INTEGER OUP,NFLAG

      WRITE(*,*)'Presmetuvanje integral so metodt:'
      WRITE(*,*)'Monte Karlo so sredna vrednost'
      WRITE(*,*) ' '
      WRITE(*,*) 'Odberete vid na izlez: '
      WRITE(*,*) '1. Ekran '
      WRITE(*,*) '2. Text file '
      WRITE(*,*) 'Vnesi 1 ili 2 '
      READ(*,*) NFLAG
      WRITE(*,*) ' '

      IF (NFLAG .EQ. 2 ) THEN
      WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
      WRITE(*,*) 'drive:name.ext'
      WRITE(*,*) 'patot da bide staven vo navodnici'
      WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
      READ (*,*) NAME1
      WRITE(*,*) ' '
      OUP = 3
      OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
      OUP = 6
      ENDIF
      IF (OUP.EQ.6) THEN
      GO TO 10
      END IF

1  0  WRITE(*,*)'VNESI GO BROJOT NA TOCKI N= (0 ZA STOP)'
      READ(*,*) N
      IF (N.EQ.0) STOP
      WRITE(OUP,'(3X,A,4X,A,7X,A,8X,A,8X,A)')'N','NUM. VREDNOST',
     * 'TOCNA','SIGMA','RAZLIKA'
      FV=0.D0
      M=10
      DO I=1,M
      FAVE=0.D0
      F2AVE=0.D0
      SUMF=0.D0
      SUMF2=0.D0
      DO IX=1,N
      CALL RANDOM_NUMBER(X)
      FX=FUNC(X)
      SUMF=SUMF+FX
      SUMF2=SUMF2+FX**2
      END DO
      FAVE=SUMF/N
      F2AVE=SUMF2/N
      SIGMA=DSQRT((F2AVE-FAVE**2)/N)
      WRITE(OUP,900)N,FAVE,EXACT,SIGMA,EXACT-FAVE
      FV=FV+FAVE
      END DO
      FV=FV/M
```

```
      WRITE(OUP,920)'POSLE',M,'POVTORUVANJA'
      WRITE(OUP,*)'SREDNA VREDNOST','   RAZLIKA'
      WRITE(OUP,*),' '
      WRITE(OUP,910)FV,EXACT-FAVE
      WRITE(OUP,*),' '
  9 0 0 FORMAT(I5,F12.5,5X,3(1X(F12.5)))
  9 1 0 FORMAT(F12.5,2X,F12.5)
  9 2 0 FORMAT(1X,A,I4,2X,A)
      GO TO 10
      CLOSE(UNIT=3)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END

C     DEFINIRANJE NA FUNKCIJA ZA INTEGRIRANJE
      REAL*8 FUNCTION FUNC(X)
      IMPLICIT REAL*8 (A-H,P-Z)
      FUNC=1.D0/(1.D0+X**2)
      RETURN
      END

C     DOPLONITELNI GENERATORI NA SLUCAJNI BROEVI
      REAL*8 FUNCTION URAND(XN)
      INTEGER XN
      INTEGER A, M, C
      DATA A/2147437301/
      DATA M/80000000/
      DATA C/453816693/
      XN=A*XN+C
      IF(XN.LT.0.D0) XN=XN+M
      URAND=XN/2.0**31
      RETURN
      END

      REAL*8 FUNCTION RAND(IX)
      INTEGER A,P,IX,B15,B16,XHI,XALO,LEFTLO,FHI,K
      DATA A/16807/,B15/32768/,B16/65536/,P/2147483647/
      XHI=IX/B16
      XALO=(IX-XHI*B16)*A
      LEFTLO=XALO/B16
      FHI=XHI*A+LEFTLO
      K=FHI/B15
      IX=(((XALO-LEFTLO*B16)-P)+(FHI-K*B15)*B16)+K
      IF(IX.LT.0.D0) IX=IX+P
      RAND=DFLOAT(IX)*4.656612875D-10
      RETURN
      END
```

```
   N    NUM. VREDNOST       TOCNA        SIGMA        RAZLIKA
 100      0.77021           0.78540      0.01626      0.01519
 100      0.76331           0.78540      0.01659      0.02209
 100      0.78543           0.78540      0.01524     -0.00003
 100      0.77817           0.78540      0.01666      0.00723
 100      0.78127           0.78540      0.01706      0.00413
 100      0.76482           0.78540      0.01508      0.02058
 100      0.76628           0.78540      0.01589      0.01912
 100      0.79593           0.78540      0.01613     -0.01053
 100      0.77424           0.78540      0.01698      0.01116
 100      0.78157           0.78540      0.01646      0.00383
POSLE  10  POVTORUVANJA
SREDNA VREDNOST   RAZLIKA
  0.77612          0.00383
```

11.2. Метод на важност на земање примерок

Овој метод на Монте Карло интеграција се применува во случаите кога имаме голема дисперзија а причина за тоа се варијациите или промените на функцијата $f(x)$ во интегравалот на интегрирање. Доколку се примени претходниот метод за интегрирање на

ваква „осцилирачка“ функција, тогаш голем број од случајните примероци на апциси x_i сигурно ќе бидат надвор од областа на варирање на функцијата, па затоа и вредноста на интегралот ќе биде пресметана со мала точност. Излезот од оваа ситуација е во изборот на доволен број точки токму во „варирачките“ области, што значи дека изборот на случајните точки x_i не треба да биде со иста веројатност, туку со определена распределба што ќе предизвика дисперзијата на интегралот да биде што помала.

За таа цел, интегралот (11.1) ќе го презапишеме во друга форма

$$I=\int_a^b f(x)\mathrm{d}x=\int_a^b \frac{f(x)}{\omega(x)}\,\omega(\mathrm{x})\mathrm{d}x\,, \tag{11.5}$$

каде $\omega(x)$ е *тежинска функција* или *функција на распределба на веројатноста* што се избира во зависност од функцијата што ја интегрираме. На овој начин, интегралот се пресметува со изразот

$$I=\left\langle\frac{f}{\omega}\right\rangle\approx\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{\omega(x_i)}, \tag{11.6}$$

и доколку $\omega(x)$ е избрана правилно, тогаш дисперзијата ќе биде мала величина што значи поголема точност на методот. Важно својство на тежинската функција е дека таа е ненегативна функција $\omega(x)>0$ и е нормира на единица

$$\int_a^b \omega(x)\mathrm{d}x=1\,. \tag{11.7}$$

11.2.1. Смена на променливи со инверзна трансформација

Да разгледаме таква смена на променливите со што интегралот (11.5) ќе премине во нова форма

$$I=\int_a^b f(x)\mathrm{d}x=\int_0^1 \frac{f[x(z)]}{\omega[x(z)]}\omega(x)\mathrm{d}x=\int_0^1 \frac{f[x(z)]}{\omega[x(z)]}\mathrm{d}z(x)\,, \tag{11.8}$$

така што униформната распредлеба по новата променлива z во интервалот $[0,1]$ ќе доведи до зголемена веројатност на избор на $x(z)$ во областите на варирање на $f(x)$, што во принцип значи поголема точност во пресметувањето на интегралот.

За новата променлива z важи $\mathrm{d}z=\omega(x)\mathrm{d}x$, со што се определува веројатноста x да биде во интервалот $(x,x+\mathrm{d}x)$, од каде следи

$$z(x)=\int_{-\infty}^{x}\omega(x')dx'\,. \tag{11.9}$$

Од теоријата на веројатност произлегува дека $z(x)$ е *функција на распределба* или *кумулативна функција на распределба* додека $\omega(x)$ е *густина на веројатноста*. Од претходно кажаното, за $z(x)$ ќе важи селдново својство

$$z(-\infty)=0,\qquad z(\infty)=1\,. \tag{11.10}$$

Во принцип, од (11.9) може да ја добиеме зависноста $z(x)$, но за да го пресметаме интегралот (11.8) неопходно е да ја знаеме и инверзната зависност $x(z)$, за што треба да извршиме инверзија на интегралниот израз (11.9). За таа цел, ќе воведиме нова променлива y која има униформна распределба $u(y)=1$ во интервалот $0\le y\le 1$, а со релацијата $u(y)\mathrm{d}y=\omega(x)\mathrm{d}x$, или

$$\omega(x)=u(y)\left|\frac{\mathrm{d}y}{\mathrm{d}x}\right|, \tag{11.11}$$

се обезбедува променливата x да има распредлба со веројатност $\omega(x)$ во интервалот $a\le x\le b$. Значи, најважно за овој метод е да се определи инверзната зависност

$$x=z^{-1}(y). \tag{11.12}$$

Во текстот што следи, ќе покажиме како практично се пресметува оваа величина во зависност од обликот на $\omega(x)$.

11.2.2. Постојана густина на веројатност

Како пример, ќе го разгледаме случајот кога функцијата на густина на веројатност има константна вредност зададена со

$$\omega(x)=\begin{cases}\dfrac{1}{b-a} & a\le x\le b\\ 0 & \text{на другите места}\end{cases}. \tag{11.13}$$

Од (11.9) произлегува дека

$$z(x)=\int_a^x\omega(x')\mathrm{d}x'=\int_a^x\frac{dx'}{b-a}=\frac{x-a}{b-a} \tag{11.14}$$

по што добиваме дека

$$x=a+(b-a)z, \tag{11.15}$$

или

$$z^{-1}(y)=a+(b-a)y. \tag{11.16}$$

Со ова се овозможува ако променливата $y\in[0,1]$ има униформна распредлба, тогаш и променливата $x(y)$, истотака, да има униформна распредлеба во интервалот $x\in[a,b]$.

Пример 11.2.

Со методот на важност на земање примероци да се пресмета интегралот

$$\int_0^1 f(x)\,\mathrm{d}x=\int_0^1\left(\frac{1}{\sqrt[3]{x}}+\frac{x}{10}\right)\mathrm{d}x\approx 1.55,$$

ако тежинската функција е $\omega(x)=2/(3x^{1/3})$. При пресметките да се земат $N=100$ и 10000 точки (апциси x_i).

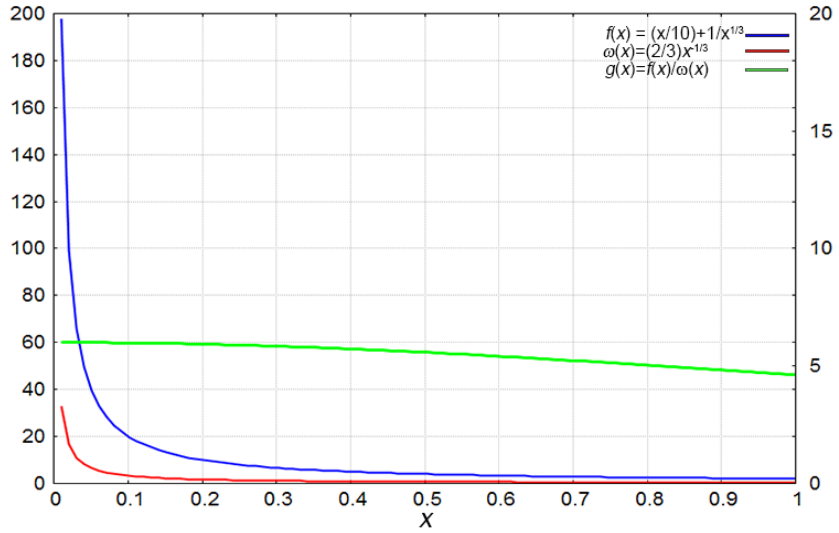
За да можеме нумерички да го пресметаме овој интеграл, најпрво ќе направиме неколку аналитички пресметувања. Ќе започнеме со определување на функцијата на распределба $z(x)$

$$z(x)=\int_0^1 \omega(x)\mathrm{d}x=\frac{2}{3}\int_0^1 x^{-1/3}\mathrm{d}x=x^{2/3},$$

а потоа , може да ја определиме инверзната зависност $x(z)$ како

$$x=z^{3/2}.$$

Понатаму, алгоритмот е речиси идентичен со претходниот. Започнуваме со генерирање на случаен број во интервалот $z\in[0,1]$ а потоа со инверзната функција се одредува $x(z)$. Функцијата $f(x)$ како и тежинска функција $\omega(x)=2/(3x^{1/3})$ ги користиме за да го пресметаме интегралот според формулата (11.8). На сликата 11.2 се прикажани функциите што се користат во овој метод.



Сл. 11.2. Графички приказ на функциите што се користат во методот на важност на земање примероци.

Програмата со која се пресметува интегралот е дадена во текстот подолу.

```
PROGRAM MONTE_KARLO_2

IMPLICIT REAL*8 (A-H,P-Z)

DATA ISEED/987654321/      ! SEED ZA GENERATOROT NA SLUCAJNI BROEVI
DATA EXACT/.78540/         ! TOCNA VREDNOST NA INTEGRALOT

CHARACTER NAME1*30,AA*1
INTEGER OUP,NFLAG

WRITE(*,*)'Presmetuvanje integral so metodt:'
WRITE(*,*)'Monte Karlo so sredna vrednost'
WRITE(*,*) ' '
WRITE(*,*) 'Odberete vid na izlez: '
WRITE(*,*) '1. Ekran '
WRITE(*,*) '2. Text file '
WRITE(*,*) 'Vnesi 1 ili 2 '
READ(*,*) NFLAG
```

```
      WRITE(*,*) ' '

      IF (NFLAG .EQ. 2 ) THEN
      WRITE(*,*) 'Vnesi ime na fajl vo sledniov vid - '
      WRITE(*,*) 'drive:name.ext'
      WRITE(*,*) 'patot da bide staven vo navodnici'
      WRITE(*,*) 'na primer:   "C:/DATA.TXT" '
      READ (*,*) NAME1
      WRITE(*,*) ' '
      OUP = 3
      OPEN(UNIT=OUP,FILE=NAME1)
      ELSE
      OUP = 6
      ENDIF
      IF (OUP.EQ.6) THEN
      GO TO 10
      END IF
1  0  WRITE(*,*)'VNESI GO BROJOT NA TOCKI N= (0 ZA STOP)'
      READ(*,*) N
      IF (N.EQ.0) STOP
      WRITE(OUP,'(3X,A,4X,A,7X,A,8X,A,8X,A)')'N','NUM. VREDNOST',
     * 'TOCNA','SIGMA','RAZLIKA'
      FV=0.D0
      M=10
      DO I=1,M
      FAVE=0.D0
      F2AVE=0.D0
      SUMF=0.D0
      SUMF2=0.D0
      DO IX=1,N
      CALL RANDOM_NUMBER(Y)
      X=XZ(Y)
      FX=FUNC(X)/WEIGHT(X)
      SUMF=SUMF+FX
      SUMF2=SUMF2+FX**2
      END DO
      FAVE=SUMF/N
      F2AVE=SUMF2/N
      SIGMA=DSQRT((F2AVE-FAVE**2)/N)
      WRITE(OUP,900)N,FAVE,EXACT,SIGMA,EXACT-FAVE
      FV=FV+FAVE
      END DO
      FV=FV/M
      WRITE(OUP,920)'POSLE',M,'POVTORUVANJA'
      WRITE(OUP,*)'SREDNA VREDNOST','   RAZLIKA'
      WRITE(OUP,*),' '
      WRITE(OUP,910)FV,EXACT-FAVE
      WRITE(OUP,*),' '
9  0  0  FORMAT(I5,F12.5,5X,3(1X(F12.5)))
9  1  0  FORMAT(F12.5,2X,F12.5)
9  2  0  FORMAT(1X,A,I4,2X,A)
      GO TO 10
      CLOSE(UNIT=3)
      CLOSE(UNIT=OUP)
      IF(OUP.NE.OUP) CLOSE(UNIT=6)
      STOP
      END
C     DEFINIRANJE NA FUNKCIJA ZA INTEGRIRANJE
      REAL*8 FUNCTION FUNC(X)
      IMPLICIT REAL*8 (A-H,P-Z)
      ST=1.D0/3.D0
      F1=X/10.D0
      F2=1.D0/X**ST
      FUNC=F1+F2
      RETURN
      END
C     DEFINIRANJE NA TEZINSKA FUNKCIJA
      REAL*8 FUNCTION WEIGHT(X)
      IMPLICIT REAL*8 (A-H,P-Z)
      ST=1.D0/3.D0
      F1=2.D0/3.D0
      F2=1.D0/X**ST
      WEIGHT=F1*F2
      RETURN
```

```
      END
C     DEFINIRANJE NA INVERZNA FUNKCIJA
      REAL*8 FUNCTION XZ(X)
      IMPLICIT REAL*8 (A-H,P-Z)
      ST=3.D0/2.D0
      XZ=X**ST
      RETURN
      END

C     DOPLONITELNI GENERATORI NA SLUCAJNI BROEVI
      REAL*8 FUNCTION URAND(XN)
      INTEGER XN
      INTEGER A, M, C
      DATA A/2147437301/
      DATA M/80000000/
      DATA C/453816693/
      XN=A*XN+C
      IF(XN.LT.0.D0) XN=XN+M
      URAND=XN/2.0**31
      RETURN
      END

      REAL*8 FUNCTION RAND(IX)
      INTEGER A,P,IX,B15,B16,XHI,XALO,LEFTLO,FHI,K
      DATA A/16807/,B15/32768/,B16/65536/,P/2147483647/
      XHI=IX/B16
      XALO=(IX-XHI*B16)*A
      LEFTLO=XALO/B16
      FHI=XHI*A+LEFTLO
      K=FHI/B15
      IX=(((XALO-LEFTLO*B16)-P)+(FHI-K*B15)*B16)+K
      IF(IX.LT.0.D0) IX=IX+P
      RAND=DFLOAT(IX)*4.656612875D-10
      RETURN
      END
```

```
  N    NUM. VREDNOST        TOCNA        SIGMA        RAZLIKA
 1000     1.55269           1.55000      0.00144     -0.00269
 1000     1.55026           1.55000      0.00142     -0.00026
 1000     1.54891           1.55000      0.00144      0.00109
 1000     1.55133           1.55000      0.00142     -0.00133
 1000     1.55000           1.55000      0.00139      0.00000
 1000     1.54953           1.55000      0.00141      0.00047
 1000     1.54691           1.55000      0.00138      0.00309
 1000     1.54885           1.55000      0.00144      0.00115
 1000     1.55026           1.55000      0.00138     -0.00026
 1000     1.54894           1.55000      0.00142      0.00106

 POSLE  10  POVTORUVANJA
 SREDNA VREDNOST   RAZLIKA
     1.54977       0.00106
```

Од сликата 11.2 се гледа дека функцијата $g(x)=f(x)/\omega(x)$ е скоро константна во границите на интегрирање па затоа и дисперзијата и е мала, $\sigma=0.001$, што значи зголемена точност на методот. Колку за споредба, ако овој интеграл го пресметаме со методот на средна вредност, дисперзијата е $\sigma_I \approx 0.03$, што е за повеќе од еден ред на големина помала точност.

Сепак, треба да спомниме дека методот на важност на земање примероци иако е подобар од методот на средна вредност, функционира добро само за ограничен број класи на функции, од проста прчина што за нив може аналитички да се пресмета инверзната функција. Затоа, неопходно е да се воведат нови методи кој ќе дадат речиси универзални решенија што не зависат од изборот на функцијата за интеграција.

11.3. Метод на Метрополис

За да појасниме за каков метод станува збор, најпрво ќе го разгледаме следниов физички проблем. Фермогнетните материјали имаат постојано магнетно поле затоа што се содржат домени во кои спиновите на електроните се ориентирани во еден правец. Ако аплицираме надворешно магнетно поле, тогаш домените се подредуваат во насока на надворешното поле, така што може да се појават многу јаки внатрешни магнетни полиња. Сепак, и во такви услови, ако ја зголемуваме температурата континуирано, во даден момент ќе се случи фазен премин што ќе предизвика рапидно намалување на магнетизацијата. Токму за компјутерско симулирање на овие премини, како и термодинамиката на магнетните системи, ни треба алгоритмот на Метрополис.

Со овој метод треба да се разрешат повеќедимензионални интеграли што произлегуваат од следниов израз

$$E_{S_1 S_2} = -J\sum_{i=1}^{N-1} s_i s_{i+1} - B\mu\sum_{i=1}^{N} s_i \,, \tag{11.17}$$

кој ни ја дава енергијата на заемодејство помеѓу спиновите на еднодимензионален Изингов модел во надворешно магнетно поле *B*. Не навлегувајќи во физиката на проблемот, повеќедимензионалноста на интегралите во овој случај произлегува од методите што треба да се применат за нумеричко решавање на сумите. За N честици, имаме 2^N различни конфигурации на спиновите s_i, што значи дека, на пример, за скромни $N = 20$ честици ќе имаме повеќе од 10^6-та кратна интеграција! Очигледно е дека ваков нумерички проблем може да се реши само со Монте Карло методи.

Овој алгоритам во суштина е еден општ метод за генерирање на случајни броеви со „произволна“ густина на веројатност, што значи дека може да се примени на реалните физички проблеми без разлика дали можеме аналитички да најдиме соодветна тежинска функција.

Да разгледаме ситем со *N* честици кој во 3*N* димензионален простор се опишува со радиус вектор $R = (r_1, r_2, \ldots, r_N)$ и физичка величина што зависи од овој вектор (или конфигурација) $G(R)$.

Ако претпоставиме дека $G(R)$ што се опишува со канонскиот ансамбл, тогаш според формализмот на статистичката физика нејзината средна вредност се пресметува како

$$\langle G \rangle = \int G(R)\omega(R)\mathrm{d}R \,, \tag{11.18}$$

при што функцијата на густина на веројатност е

$$\omega(R) = \frac{e^{-U(R)/k_B T}}{\int e^{-U(R)/k_B T}\,\mathrm{d}R} \,, \tag{11.19}$$

каде $U(R)$ е потенцијална енергија на системот што зависи од конфигурацијата на системот зададена со R, k_B е Боцманова константа а T е апсолутна температура.

За да го пресметаме ефикасно повеќедимензионалниот интеграл (11.18) според методот на Метрополис, земањето примероци треба да се разгледува како Марков процес, што најпросто кажано значи дека веројатноста за секој нов случајно избран

примерок зависи само од веројатноста за реализација на неговиот претходник. Во состојба на термодинамичка рамнотежа, вредностите на функцијата на густина на веројатност во различни точки се задаваат со релацијата

$$\omega(R)T(R\to R')=\omega(R')T(R'\to R), \tag{11.20}$$

каде $T(R\to R')$ е степен на премин од состојба што се карактеризира со R во состојба определена со R'. Преминот од конфигурациска состојба R во R' се прифаќа ако е задоволено

$$\frac{T(R\to R')}{T(R'\to R)}=\frac{\omega(R)}{\omega(R')}\geq w_i, \tag{11.21}$$

каде w_i е униформно избран број во интервалот $[0,1]$, во спротивно останува старата конфигурација. Оваа постапка е позната и како состојба на случаен шетач чија позиција во просторот за секоја итерација $i=1,\ldots n$ зависи од R_i. Почетната позиција на шетачот е R_0, а преминувањето во нова позиција, на пример R_1 за зависи од вредноста што ја дава релацијата (11.21). Ако оваа постапка се повтори n пати, ќе ја добие состојбата на ситемот определена со R_n. Таа состојба всушност е резултат на n неуниформно избрани броеви со распределба $\omega(R)$.

11.3. Пример

Со алгоритмот на Метрополис да се симулира рамнотежната конфигурација при дадена температура на еднодимензионална решетка од магнетни диполи (спинови). Поведението на овој систем да се опише со Изинговиот модел ?

Заради поедноставување, ќе разгледаме Изингов модел за магнетни системи без надворешно магнетно поле што се опишува со хамилтонијанот

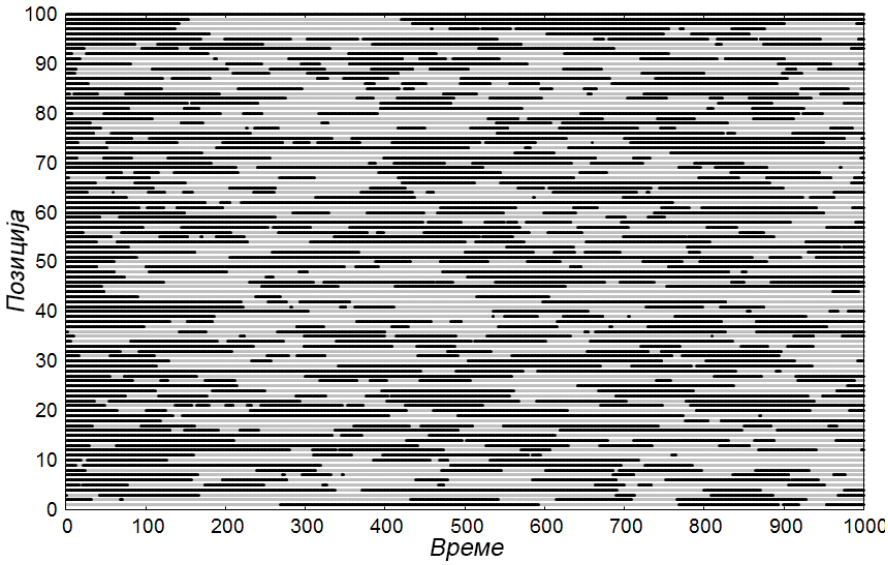
$$H_0=-J\sum_{i=1}^{N-1}s_i s_{i+1}\,.$$

Со овој модел вообичаено се опишуваат фазните премини на магнетните системи како што се феромагнетиците и антиферомагнетиците. При тоа, предвид се зема само заемодејството на два соседни спина, $(s_i s_{i+1})$, а енергијата на заемодејство се задава со величината J. Во нашит случај, $J=\pm 1$, а вредноста (насоката) на спиновите може да биде само ± 1.

Алгоритмот на Метрополис, со кој ќе го симулираме фазниот премин кај еднодимензионалната магнетна решетка ќе го спроведиме на следниов начин. Најпрво, започнуваме со една „ладна“ конфигурација на решетката во која сите спинови имаат вредност $+1$ при дадена нормализирана температура $T/k_B=100$. Таа конфигурација е дадена со првата колона долж y оската на слика 11.3.

Започнуваме да го „затоплуваме“ системот на тој начин што случајно избираме еден дипол и ја менуваме неговата насока. Ја пресметуваме новата вредност на енергијата E_i на системот, како и релативната веројатност за реализација на новата конфигурација со изразот $\Delta\omega=\exp(-\Delta E/k_B T)$, каде $\Delta E=E_i-E_0$. Потоа, генерираме еден случаен број во интервалот $w_i\in[0,1]$ и ако $\Delta\omega<w_i$, во согласност со (11.21), ја отфрламе новата

конфигурација, а во спротивно ја прифаќаме и преминуваме на избор на нов случаен дипол од решетката.



Сл. 11.3. „Еволуција“ на конфигурација на спиновите на еднодимензионална магнетна решетка од диполи за $T/k_B = 100$ и $J = 1$ со тек на времето.

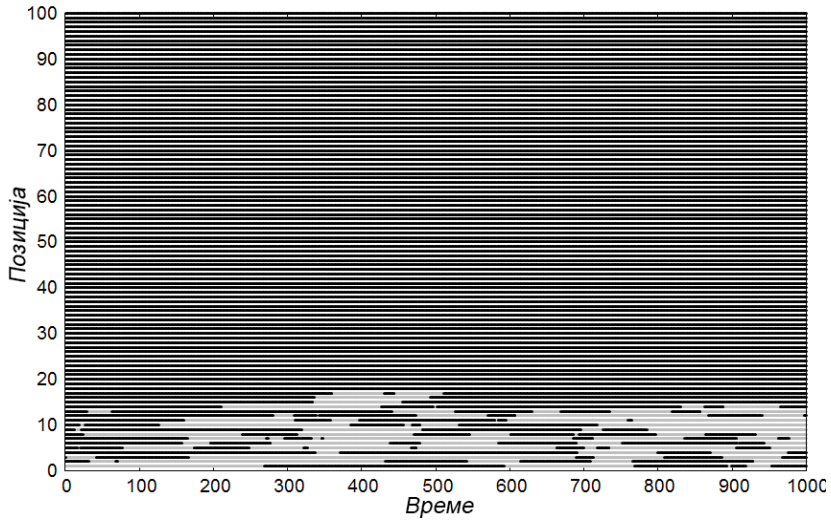
Во принцип, после доволно долго време, или доволоно голем број на случајно избрани диполи, ќе се достигне рамнотежната конфигурација на системот, но во симулацијата тоа не може да се достигне целосно затоа што нумеричките грешки се акумилираат и внесуваат голема несигурнст во пресметките.

Програмата со која е генерирана сликата 11.3. е дадена во текстот подолу.

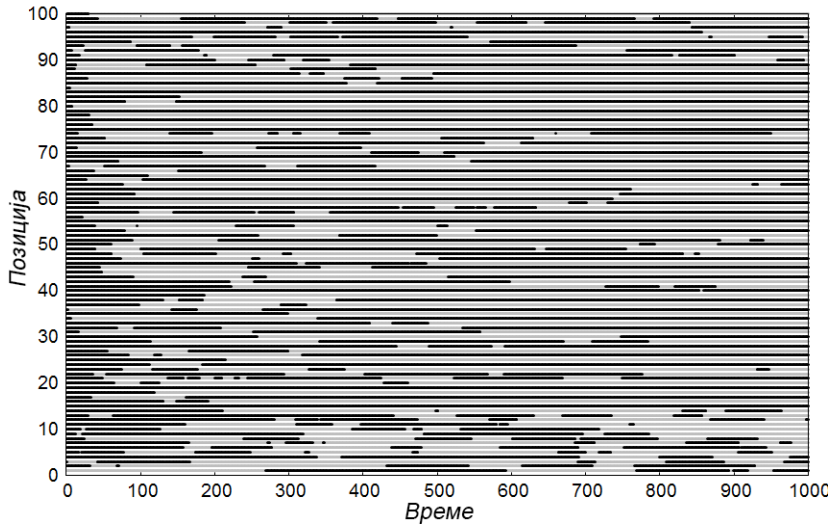
```
      PROGRAM  ISING
      IMPLICIT REAL*8 (A-H,P-Z)
      PARAMETER(MAX=100)
      DIMENSION ISPINS(MAX)

C     TEMPERATURA I ENERGUJA NA IZMENA
      PARAMETER(AKT=100.D0, EJ=1.D0)

      OPEN(8,FILE='SPIN-UP.DAT', STATUS='UNKNOWN')
      OPEN(9,FILE='SPIN-DO.DAT', STATUS='UNKNOWN')

C     UNIFORMNA KONFIGURACIJA NA SPINOVI S=+1
      DO 10  I=1,MAX
      ISPINS(I) =  1
   1  0
      CONTINUE

C     CEKOR NIZ VREMETO
      DO 20  IT=1, 1000
C     ENERGIJA NA SISTEMOT
      EOLD=ENERGY(ISPINS, EJ, MAX)
C     IZBIRAME EDEN SPIN OD NIZATA
      CALL RANDOM_NUMBER(X)
      IELEMENT=X*MAX+1
C     JA MENUVAME NASOKATA NA SPINOT
      ISPINS(IELEMENT)=ISPINS(IELEMENT)*(-1)
C     JA PRESMETUVAME NOVATA ENERGIJA
      ENEW=ENERGY(ISPINS, EJ, MAX)
C     JA OTFRLAME SMENATA AKO E ISPOLNET USLOVOT DADEN PODOLU
      IF((ENEW.GT.EOLD).AND.(DEXP((-ENEW+EOLD)/AKT).LT.X)) THEN
      ISPINS(IELEMENT)=ISPINS(IELEMENT)*(-1)
```

```
                ENDIF
C               JA ZACUVUVAME KONFUGURACIJATA OD SPINOVI
                DO 30  I=1,MAX
                IF (ISPINS(I).EQ.1) THEN
                WRITE(8,200) IT, I
                ENDIF
                IF (ISPINS(I).EQ.(-1)) THEN
                WRITE(9,200) IT, I
                ENDIF
   3 0          CONTINUE
                WRITE(8,200)
                WRITE(9,200)
   2 0          CONTINUE
                CLOSE(8)
                CLOSE(9)
   2 0 0        FORMAT(2I6)

                STOP
                END

C               FUNCTION CALCULATES  ENERGY OF THE SYSTEM
                REAL*8 FUNCTION ENERGY(IEN, EJ, MAX)
                IMPLICIT REAL*8 (A-H,P-Z)
                DIMENSION IEN(MAX)
                ENERGY=0.D0
                DO 22  I=1,(MAX-1)
                DELE=-EJ*IEN(I)*IEN(I+1)
                ENERGY=ENERGY+DELE
   2 2          CONTINUE
                RETURN
                END
```

Една од карактеристикте на овој систем е што при ниски температури, на пример $T/k_B = 2$ и $J = 1$, сите диполи се подредени во насока „горе“ и ситемот е во фаза на феромагнетик, слика 11.4. При иста температура, но изменска енергија $J = -1$ диполите се поставени наизменично и системот е во фаза на антиферомагнетик, слика 11.5.



Сл. 11.4. Почетната случајна конфигурација на магнетните диполи при $T/k_B = 2$ и $J = 1$ со тек на времето преминува во состојба на феромагнетик.



Сл. 11.5. Почетната случајна конфигурација на магнетните диполи при $T/k_B = 2$ и $J = -1$ со тек на времето преминува во состојба на антиферомагнетик.

11.3. Задачи

1. Да се модифицира програмата monte karlo 1.for така да може да се пресмета определен интеграл во произволни конечни граници?

2. Да се анализира точноста на методот од примерот 11.2 во зависност од бројот на апциси и изборот на тежинската функција? Да се споредат стандардните девијации кога тежинската функција е $\omega(x) = 1$ и $\omega(x) = 2/(3x^{1/3})$?

3. Користејќи го методот Монте Карло како генератор на униформно генерирани случајни броеви да се определи вредноста на бројот π со точност до три значни цифри. Да се испита влијанието на генераторот на слулчајни броеви врз точноста на алгоритмот?

11.4. Проекти

1. Користејќи го методот на важност на земање примероци да се пресмета тројниот интеграл

$$I = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\left(x^2+y^2+z^2\right)}\mathrm{d}z\,\mathrm{d}y\,\mathrm{d}z\,,$$

чија точна вредност е $I = \pi^{3/2}$.

2. Да се модифицира програмата од примерот 11.3 за да се определат внатрешната енергија $U(T)$, специфичната топлина $C(T)$ и магнетизацијата $M(T)$ за едно-димензионална решетка од магнетни диполи со и без надворешно магнетно поле?

12. ХАОТИЧНИ СИСТЕМИ И ФРАКТАЛНИ ОБЛИЦИ

12.1. Линеарни и нелинеарни системи

Последниве децении сме сведоци на експлозивниот развој на една област од физиката која се нарекува теорија на хаос. Еден од највжните услови за вака брзиот развој на оваа област е примената на компјутерите во пресметувањата кои бараат примена на голем број итеративни постапки. Со помош на компјутерите научниците можат поефикасно да ги проучуваат сложените природни феномени во чија основа е нелинеарноста. Досегашниот развој на физиката и останатите природни науки се базирал на проучување на линераните системи. Тоа се системи кои произведуваат ефекти пропорционални на причините за промена, односно пропорционални на силите. Линеарните системи се обичен збир на нивните составни делови. За разлика од нив, нелинеарните системи се нешто повеќе од обичен збир за нивните составни делови. Ефектот на нелинеарните системи како целина не е ист со ефектот на нивните составни делови. Нелинеарноста е резултат на непостоење на едноставна пропорционалност во делувањето на силите. Сите физички системи се, всушност нелинеарни, но тие можат да се однесуваат како линеарни кога се блиску до воспоставување на рамнотежа. Поведението на нелинеарните системи е непредвидливо и секогаш различно. Кога ќе излезат од рамнотежа, тие спонтано преминуваат или во нови, посложени и многу организирани состојби или стануваат хаотични.

Едноставни, затворени линеарни системи како што обично досега се разгледувале претставуваат екстремна идеализација. Научниците нив ги издвојувале и проучувале поради тоа што имале на располагање само доста едноставни теориски и експериментални методи со кои не можеле да ги регистрираат и анализираат сложените феномени во чија основа е нелинеарноста. Но, со текот на времето откако се развиле соодветните математички теории а особено откако е овозможена најширока примена на компјутерите, научниците се веќе во можност ефективно да ги проучуваат сложените нелинеарни системи.

12.2. Регуларни и случајни движења

Како што е добро познато од класичната механика, диференцијалните равенки на движење потполно и еднозначно ги одредуваат положбите и брзините на сите честици на даден динамички систем во секој момент на време ако се познати почетните положби и брзини на честиците и силите кои делуваат на нив. Ова е всушност основата на познатиот механички принцип на каузалноста на која се заснова механичкиот, лапласовски детерминизам. Притоа, премолчено се претпоставува дека во еден идеализиран експеримент секогаш може истовремено и апсолутно прецизно да ги одредиме положбата

на секоја честита од системот. Оваа претпоставка ќе се покаже од суштинска важност при засновување на теоријата на хаос преку отфрлање на таа претпоставка што имплицира можност за бесконечно точно набљудување. Движењата кои го задоволуваат принципот на каузалност, поточно движењата кои се потполно одредени се нарекуваат **регуларни** или **детерминистички движења**. Такви се на пример движењето на математичкото нишало, движењето на планетите околу Сонцето и многу други појави. Но постојат и движења кај кои е присутна случајноста, односно движењата чиј тек е наполно непредвидлив. Типичен пример за такви движења е брауновото движење кое се објаснува со молекуларно – кинетичката теорија. Брауновото движење се состои во непредвидливо движење на честица со мали димензии под влијание на ударите на молекулите од течноста во која е поставена честицата. Молекулите на течноста се во постојано движење и тие удираат на честицата, но со оглед на нивниот огромен број не може воопшто да се предвиди насоката на движење на честицата изложена на тие удари. Во природата постојат многу појави во кои има движења од ваков карактер и тие движења поради нивната непредвидливост, односно случајност се наречени **случајни движења.**

12.3. Конзервативни и дисипативни системи

Според познатиот Хамилтонов формализам од теориска механика, движењето на еден динамички систем со *n* степени на слобода може да се претстави со Хамилтоновите равенки

$$\dot{p}_i = -\frac{\partial H}{\partial q_i} \qquad \dot{q}_i = \frac{\partial H}{\partial p_i} \qquad (i = 1,2,...n) \tag{12.1}$$

каде p_i и q_i се обопштени имплуси и обопштени координати, а H е Хамилтонова функција

$$H(q_i, p_i, t) = \sum_{i=1}^{n} p_i \dot{q}_i - L \tag{12.2}$$

при што $L = L(q_i, \dot{q}_i, t)$ е Лагранжова функција.

Решението на Хамилтоновите равенки (12.1) содржат 2*n* константи кои одговараат на почетните вредности на координатите и импулсите. Тие решенија еднозначно ја определуваат временската еволуција на системот, која може да се претстави како движење на точката во 2*n*-димензионален *фазен простор* на системот. Множеството на сите почетни услови сочинуваат фазен волумен на даден систем. Доколку во текот на времето фазниот волумен контрахира, тогаш системот е **дисипативен**, а доколку фазниот волумен е константен, ситемот е **конзервативен.**

Во случај кога Хамилтоновата функција не зависи експлицитно од времето, таа претставува еден интеграл на движење, кој обично се поклопува законот за запазување на енергијата. Така, во случај на конзервативен ситем со два степени на слобода ќе имаме

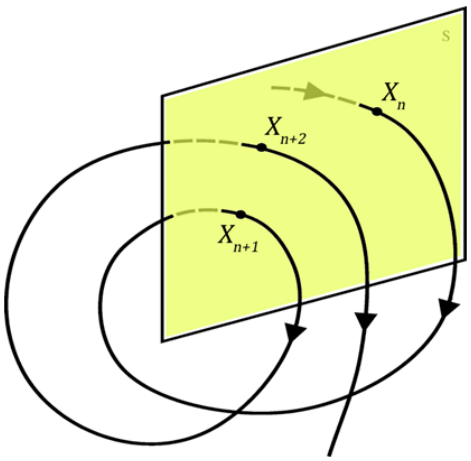
$$H(p_1, p_2, q_1, q_2) = E \tag{12.3}$$

каде E е енергија на ситемот. Со оваа всушност траекторијата е ограничена да лежи на некоја тридимензионална енергетска површина поставена во четиридимензионален фазен простор.

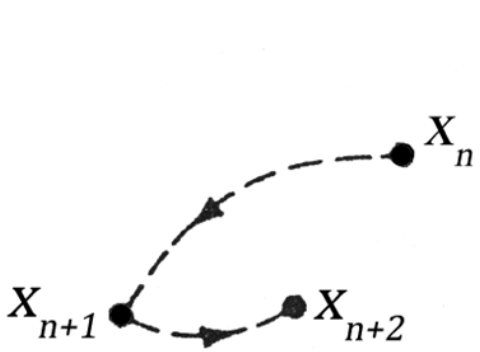
Постои една важна класа на Хамилтонови функции со два степени на слобода, а тоа се т.н. *интеграбилни* Хамилтонови функции за кои, освен енергијата постои и втора константа на движење со што траекторијата се ограничува на дводимензионална површина во фазниот простор. Познати се два вида на системи со такво својство, а тоа се системите со сепарабилен и системите со централен потенцијал. Ова додатно ограничување на траекторијата кај интеграбилните системи овозможува равенките на движење да бидат разрешени со редуцирање на проблем на наоѓање на некој интеграл како кај еднодимензионалните проблеми. Сите досега познати аналитички разрешливи проблеми од класичната механика се од овој тип на интеграбилни системи.

12.4. Пресек на Поанкаре

Еден од најзначјните методи за анализа на поведението на динамичките системи е т.н. метод на пресек на Поанкаре. Така за системи со два степени фазниот простор е четиридимензионален. Во тој фазен простор избираме некоја дводимензионална површина S и разгледуваме последователни пресеци (прободи) на некоја траекторија со таа површина (сл. 12.1а). Множеството на пресечни точки, при што се земаат предвид само влезните точи на траекторијата од една страна на површината, се вика **пресек на Поанкаре**. Пресликувањето кое се води од една точка на пресекот до друга долж траекторијата се вика **пресликување на Поанкаре** (сл 12.1б). Тоа ја заменува временски континуираната еволуција со дискретно пресликување.



Сл.12.1а. Пресек на Поанкаре.



Сл.12.1б. Пресликување на Поанкаре.

Површината на пресекот на Поанкаре може да се избере на следниов начин. Најпрво да уочиме дека траекторијата лежи на тридимензионална енергетска површина $H_0 = H(p_1, q_1, p_2, q_2)$ во четиридимензионалниот фазен простор (сл. 12.2 а). Ова равенство ја определува некоја од променливите, на пример p_2 како функција од трите останати

$$p_2 = p_2(p_1, q_1, q_2). \tag{12.4}$$

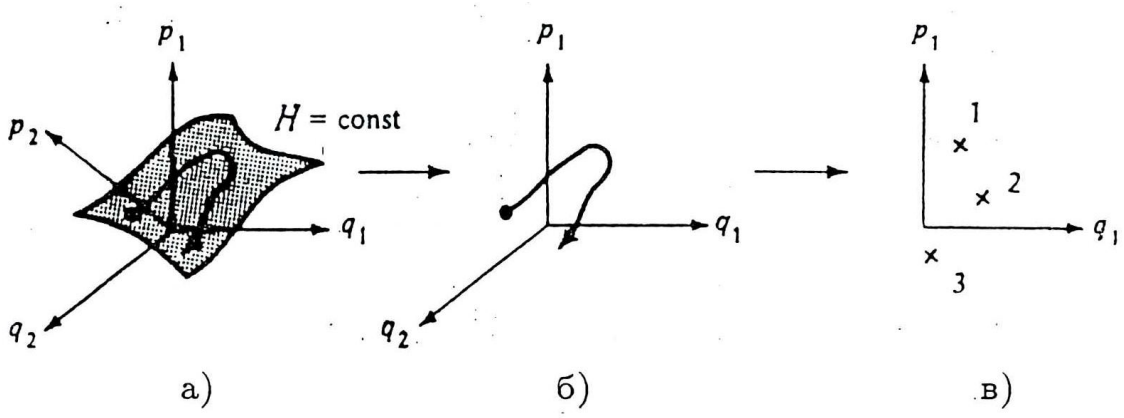
Да ја разгледаме проекцијата на траекторијата на тридимензионалниот волумен (p_1, q_1, q_2) како на слика 12.2 б. Ако е движењето ограничено, тогаш траекторијата секогаш ќе пресекува определена рамнина q_2 = *const.* внатре во тој волумен. Таа рамнина, зададена со кординатата q_1 и коњугираниот имплус p_1, погодно е да се избере како површина на пресекот на Поанкаре. Во општ случај пресечните точки на траекторијата со површината S ќе бидат произволно распределени во некоја ограничена област на S. Во случај на постоење на дополнителен (освен H_0) интеграл на движење

$$I(p_1, p_2, q_1, q_2) = const. \tag{12.5}$$

тогаш од (12.4) и (12.5) следува дека

$$p_1 = p_1(q_1, q_2) \tag{12.6}$$

Значи последователните пресечни точки треба да лежат на некоја инваријантна крива, определена со (12.6) за $q_2 = const.$ (сл. 12.2 в). На тој начин, постоењето на интеграл на движење е можно да се определи од анализата на пресечните точки на траекторијата со површината S, односно од изгледот на пресекот на Поанкаре може да се заклучи за карактерот на движењето кое го генерира. Така на пример, ако пресекот се состои од една единствена точка, може да се заклучи дека движењето се одвива по затворена крива, односно дека е периодично.



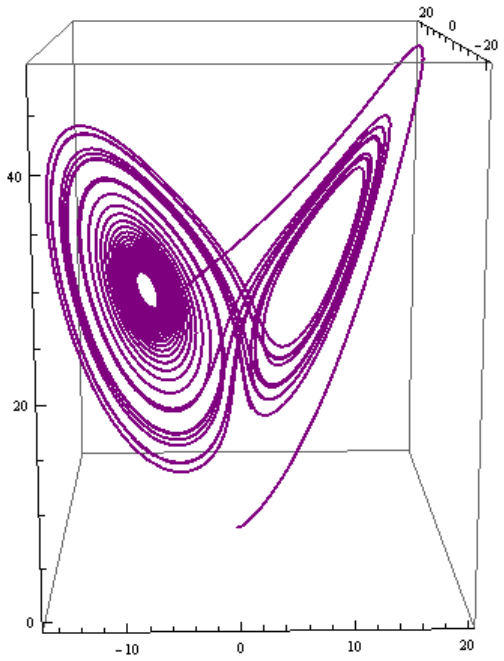
Сл.12.2.

12.5. Чудни атрактори и хаотични движења

Во 1963 година Едвард Лоренц, со цел подобро да се разберат процесите во атмосферата, го испитувал моделот на термална конвекција даден со системот равенки

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= -xz + rx - y \\ \dot{z} &= xy - bz \end{aligned} \tag{12.7}$$

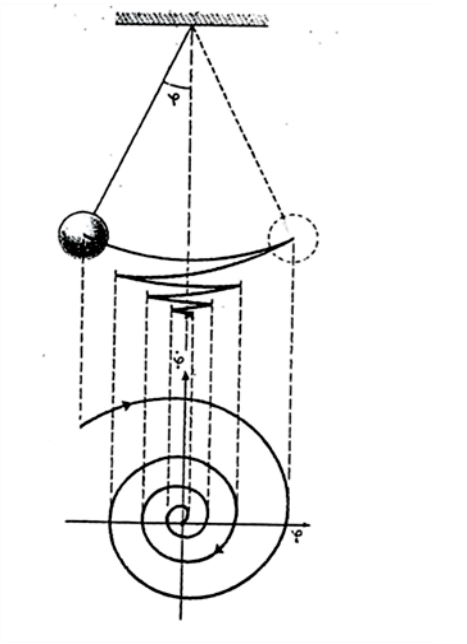
каде σ, r, b се контролни параметри. За вредности на параметрите σ = 10, r = 28, b = 8/3, Лоренц нумерички утврдил дека траекториите во фазниот простор конвергираат кон многу чуден објект прикажан на слика 12.3. Ова бил прв пример на т.н **чуден атрактор**, поим кој го вовеле Риел и Такенс во 1971 година при опишување на дисипативни динамички системи. По ова следува еден период на бурен развој на една нова дисциплина наречена теорија на хаос. Многу од новите идеи на хаосот се појавиле независно едни од други и практично истовремено во различни области, така што методите на теоријата на хаосот навлегле во многу современи научни области. Напредокот на оваа дисциплина го овозможиле развојот на новите апстрактни математички идеи и добро осмислените експерименти. Но, она што е пресудно и најмногу влијаело за вака брзиот напредок, како што веќе напоменавме, е широката примена на компјутерите во проучување на овие феномени.



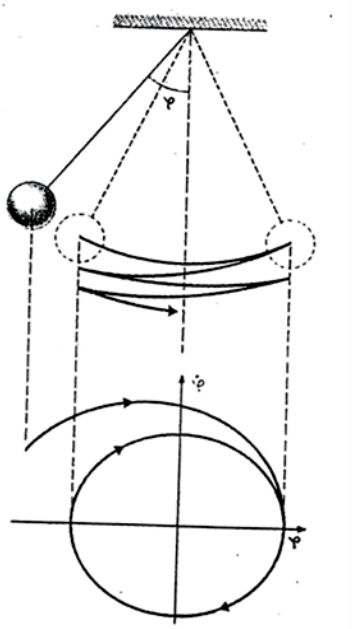
Сл.12.3. Чуден атрактор на Лоренц**.**

Да разгледаме осцилации на реално нишало, т.е. нишало во средина со триење (сл. 12.4). Фазната траекторија за осцилациите на ова нишало е во форма на спирала, бидејќи амплитудата на осцилациите опаѓа со текот на времето, се додека потполно не се анулира поради делувањето на силата на триењето. Независно од почетните услови, нишалото се стреми кон една точка со минимум потенцијална енергија.

Во фазната рамнина изгледа како таа точка да ги привлекува сите траектории на нишалото, односно сите траектории за различни почетни услови се стремат кон една единствена неподвижна точка.



Сл. 12.4. Фазна траекторија на нишало со сила на триење.



Сл. 12.5. Фазна траекторија на нишало без сила на триење.

Ако замислиме осцилации на идеализирано нишало (сл. 12.5) кај кое не постои губиток на енергија (или пак, тој губиток се компензира од некој надворешен извор) фазната траекторија ќе има облик на елипса (за мали амплитуди на осцилирање). Ако пак на потпорната точка на нишалото делуваме со некоја хармониска сила, траекторијата на нишалото во тридимензионалниот простор ќе има облик на соленоид кој ја обвиткува површината на еден торус. Доколку односот на двете фреквенции на овој биперiодичен систем е ирационален број, траекторијата после извесно време ќе го прекрие торусот. Сите овие состојби (неподвижната точка, граничниот циклус и торусот) се таканаречени атрактори. *Атрактор* или *привлекувач* е таква состојба, односно множество на точки кон кое се стремат сите траектории на даден систем.

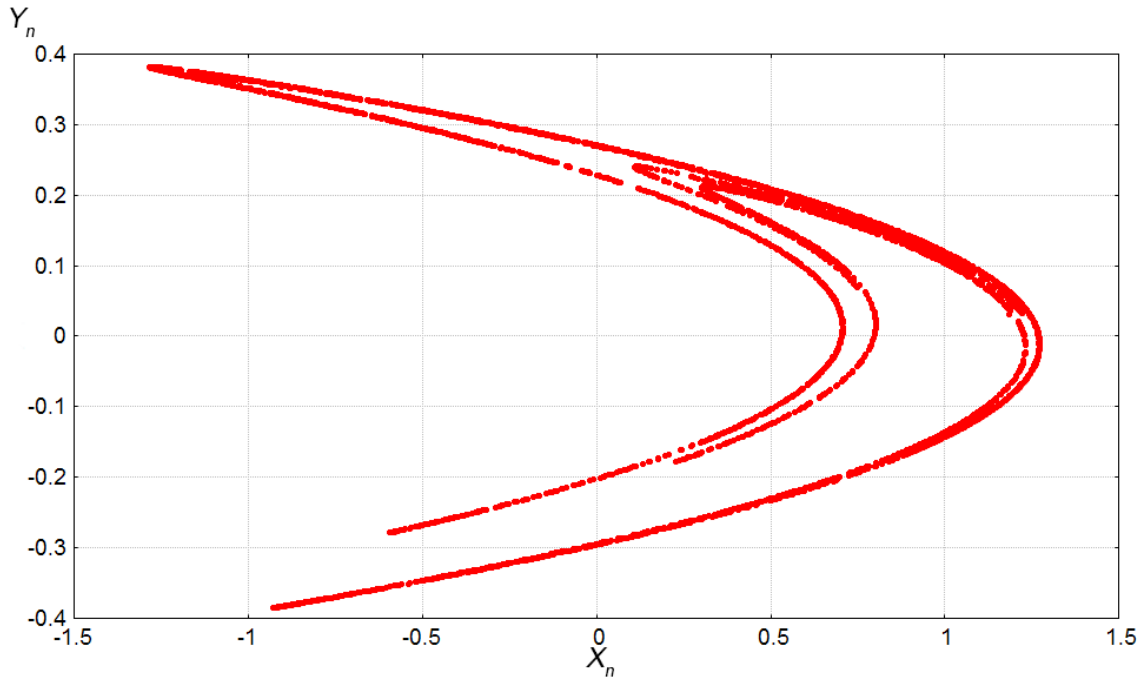
Но, за разлика од овие атрактори кои имаат релативно едноставно геометриска структура, видовме дека постојат атрактори кои се многу сложени и изгледаат доста чудно, па затоа се наречени чудни атрактори. Еден таков атрактор е веќе споменатиот атрактор на Лоренц.

Друг таков чуден атрактор е познатиот атрактор на Енон, што се добива кај пресликувањето на Енон

$$\begin{aligned} x_{n+1} &= 1 - ax_n^2 + y_n \\ y_{n+1} &= bx_n \end{aligned} \tag{12.8}$$

За чудните атрактори е карактеристично што траекториите на било кои две точки што припаѓаат на атракторите се сосема различни, односно тие дивергираат. Според тоа, кога движењето се одвива по чуден атрактор, а поради неможноста почетните услови да се познаваат бесконечно точно, не е можно да се предвиди понатамошниот негов тек.

Такво движење се нарекува **хаотично движење**, па поради тоа чудните атрактори се нарекуваат **хаотични атрактори**.



Сл.12.6. Чуден атрактор на Енон.

Кај хаотичното движење се јавува ефект на чувствителност на почетните услови, така наречен *пеперутка ефект*. Имајќи го предвид искажувањето на Лоренц дека кога имаме непредвидливо или хаотично движење може да се случи едно мало придвижување на крилјата на пеперутката на еден крај на Земјата да предизвика, после некое време, ураган на сосема друг крај на Земјата. Хаотичното движење изгледа случајно или непредвидливо иако е сепак сосема детерминирано. Феноменот на хаос како поим има и дијалектички карактер па затоа се вели дека хаосот е „детерминистичка случајност“ или „уреден неред“.

Чувствителноста на почетните услови кај хаотичните движења може да се проучува преку пресметување на раздвојувањето на две блиски почетни точки во фазниот простор. Тоа раздвојување може да се менува експоненцијално со времето со експонент λ, познат како експонент на Љапунов. Ако $\lambda < 0$, тоа значи дека траекториите на две блиски точки ќе се придвжуваат и ќе тежат кон некој атрактор. Доколку $\lambda > 0$, тогаш раздвојувањето ќе расте екпоненцијално и ќе постои голема чувствителност на почетните услови и во поведението на системот ќе се јави хаос.

Со програмата henon.for, дадена подолу во текстот, се пресметуваат сукцесивните итерации од пресликувањето на Енон. Обликот на атракторот на Енон за вредности на параматрите a = 1,4 и b = 0,3 е даден на сликата 12.6. Може да се забележи дека какви и да се изберат почетни услови, траекторијата на пресликувањето на Енон доведува до ист чуден облик, односно чуден атрактор.

```
      PROGRAM HENON

C     ******************************************
C     PRESLIKUVANJE NA ENON
C     ******************************************

      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N=5000)
      DIMENSION X(0:N), Y(0:N)

C     POCETNI VREDNOSTI
      X(0) = .1D0
      Y(0) = .2D0
      A = 1.4D0
      B = 0.3D0

      OPEN(6, FILE='HENON.DAT', STATUS='UNKNOWN')

C     CIKLUS ZA VREDNOSTITE NA X(I) I Y(I)

      DO 30 I = 1 , N
         X(I) = Y(I-1) + 1.D0 - A*X(I-1)*X(I-1)
         Y(I) = B * X(I-1)
         WRITE(6,50) X(I),Y(I)
   30 CONTINUE

   50 FORMAT(2F10.6)
      CLOSE(6)

      STOP
      END
```

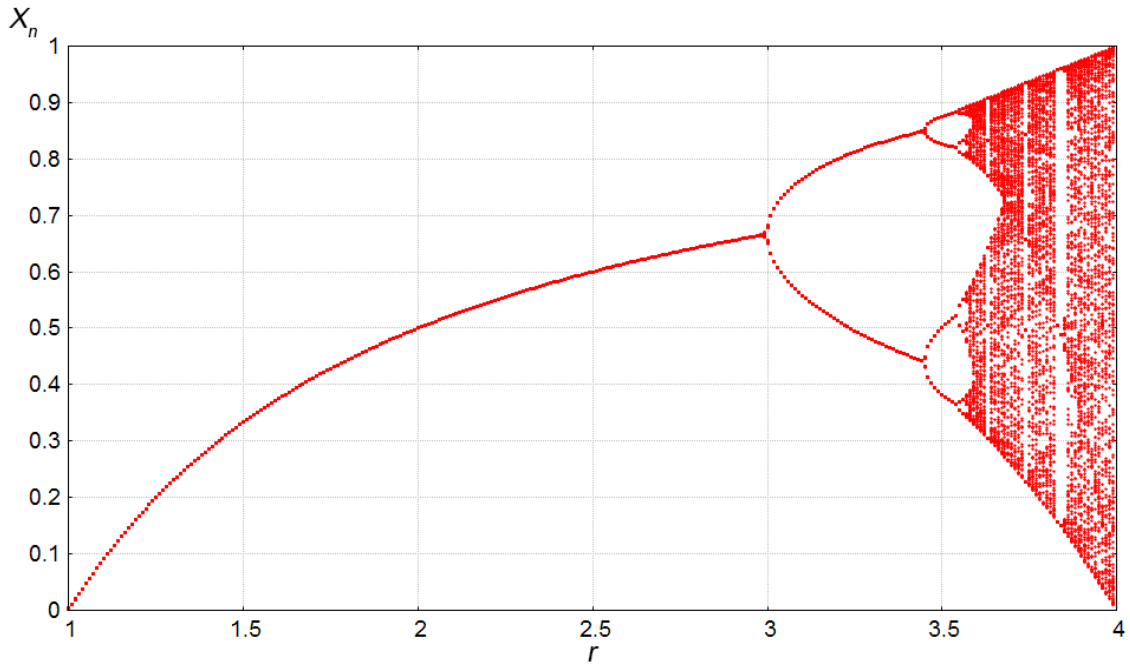
12.6. Логистичко пресликување и бифуркации

Пресликувањето

$$x_{n+1} = r x_n (1 + x_n) \tag{12.9}$$

познато како **логистичко пресликување** е едно од наједноставните еднодимензионални пресликувања кое најпрво се јавило во популационата динамика како едноставен модел за опишување на популациониот раст на дадена генерација животински видови при што x_n е популацијата после n генерации. Меѓутоа, значајот на ова пресликување не е толку во популационата динамика, колку во можноста со него да се илустрираат многу својства на хаотичните феномени. Со детално проучување на ова пресликување покажано е дека поведението на итерационите членови во зависност од вредноста на параметарот r е многу комплексно и води кон хаос. Со тоа е потврдено искажувањето на Роберт Меј кои вели дека „на луѓето им било многу подобро кога би сфатиле дека едноставни модели воопшто не мора да водат кон едноставно поведение“.

На слика 12.7 прикажан е т.н **бифуркациониот дијаграм** на логистичкото пресликување добиен со програмата bifur.for. Процедурата за добивање на дијаграмот е доста едноставна. Најпрво параметарската оска, односно оската на која е преставен параметарот r се дели на низа блиски точки. Во секоја точка се итерира равенката (12.9) многу пати, земајќи за почетна вредност на x_0, произволна вредност помеѓу 0 и 1. Притоа првите стотина вредности не се земаат, а останатите се запишуваат во датотека која потоа се визуализира со некоја програма, на пример GnuPlot.



Сл.12.7. Дијаграм на логистичко пресликување.

Може да се забележи дека за $r>3$ атрактор на ова пресликување е една единствена точка. За вредности на параметарот $3<r<3,488$ атракторот се состои од две точки коишто наизменично се повторуваат, односно велиме дека се појавил цилкус со период 2. За $r>3,488$ атракторот е низа од 4 точки, потоа добиваме циклус од 8 точки и сè така во бесконечност до $r_\infty \approx 3,5699$ ќе имаме сукцесивно удвојување на периодите кое што се нарекува **бифуркација.**

Во оваа област, т.е. $3<r<r_\infty$ експонентот на Љапунов секаде е негативен, освен во точките на бифуркација, каде е нула. За $r>r_\infty$ експонентот на Љапунов е главно позитивен, и со тоа имаме хаотично дижење. Внатре во овој хаотичен интервал на многу места се отвараат периодични „прозори“ во кои експонентот на Љапунов е негативен. Најдено е нумерички, а од страна на Фајгенбаум во 1978 година покажано е аналитички дека ако r_n е точката на *n*-та бифуркација, тогаш за големи *n* имаме

$$\delta = \lim_{n\to\infty} \frac{r_n - r_{n-1}}{r_{n+1} - r_n} \tag{12.10}$$

каде $\delta = 4,6692016091029$ е наречена константа на Фајгенбаум која покажува за колку пати е потребна помала промена на параметарот r , за да настане следната бифуркација. Оваа константа е универзална за сите пресликувања со квадратен максимум и е првата откриена константа која квантитативно ги карактеризира феномените на хаосот.

Бифуркациониот дијаграм во $x-r$ рамнината, прикажан на слика 12.7 е добиен со програмата BIFUR.for.

```
      PROGRAM BIFURCATION
C     ***********************************************************
```

```
C               BIFURKACIONA MAPA NA LOGISTICKO PRESLIKUVANJE  M*Y*(1-Y)
C               *****************************************************

                IMPLICIT REAL*8 (A-H,O-Z)

                R_MIN = 1.0D0
                R_MAX = 4.0D0
                STEP  = 0.01D0

                OPEN(6, FILE='BIFUR.DAT', STATUS='UNKNOWN')

C               CIKLUS ZA VREDNOSTITE NA M
                DO 10 R=R_MIN, (R_MAX-STEP), STEP
                Y = 0.5D0

C               SE CEKA DODEKA PREMINOT OD 200 TOCKI NE POMINE
                DO 20 IX=0, 200
                   Y = R * Y * (1.D0 - Y)
   2  0         CONTINUE

C               POTOA SE ZAPISUVAAT SLEDNITE 200 TOCKI
                DO 30 IX=201, 401
                   Y = R * Y * (1.D0  - Y)
                   WRITE(6,50) R,Y
   3  0         CONTINUE

   1  0         CONTINUE

   5  0         FORMAT(F5.3,F10.6)
                CLOSE(6)

                STOP
                END
```

12.7. Фрактали

Во секојдневниот живот имаме интуитивна претстава за тоа што е димензија: знаеме дека линијата има една димензија, рамнината две димензии, а просторот во кој живееме е тридимензионален. Исто така во физиката се зборува за четиридимензионален простор–време, а во математиката за разни повеќедимензионални математички простори. Сите овие претстави за димензија ја имаат во основа претпоставката за целобројност на димензијата.

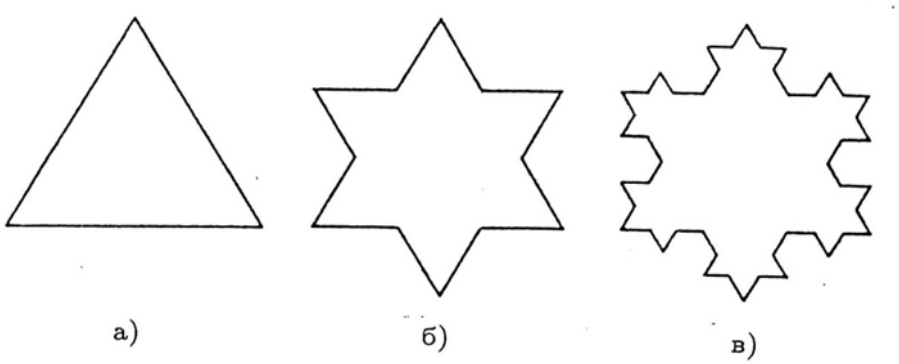
Се поставува прашање дали има геометриски предмети, облици чија димензија не е целобројна, на пример дали има предмети со димензија 1/3. Се покажало дека такви предмети постојат. Најпрво да ја воведеме дефиницијата на димензија која мора да го земаме предвид основното својство на геометриските облици, т.е. инваријантноста на димензијата во однос на промената на размерот, односно големината на геометриските облици. Така една мала коцка мора да има иста димензија како и една голема коцка. Да замислиме сега еден произволен геометриски предмет сместен во n–димензионален Евклидски простор. Нека за прекривање на тој предмет ни се потреби $M(\varepsilon)$ мали n–димензионални сфери со дијаметар ε. Бројот на сферите потребни за потполно прекривање на предметот зависи од нивните линеарни димензии: доколку ε е помало, потребен е поголем број. Доколку тој број $M(\varepsilon)$ расте како $1/\varepsilon^D$, со намалување на ε, тогаш т.н. **фрактална димензија** според Хаусдорф ке биде

$$D = \lim_{\varepsilon \to 0} \frac{\ln M(\varepsilon)}{\ln\left(\frac{1}{\varepsilon}\right)} \tag{12.11}$$

За мали ε следува дека

$$M(\varepsilon) \sim \varepsilon^{-D} \tag{12.12}$$

Најпрво да видиме дали оваа дефиниција се сложува со нашата претстава за целобројни димензии. Да разгледаме права линија со должина на пример 1 поделена на 10 еднакви делови. Тогаш $M = (1/10)^{-1} = 10$ односно $M(\varepsilon) \sim \varepsilon^{-1}$, што значи дека $D = 1$. За квадрат со единична површина $M = (1/10)^{-2} = 100$, односно $M(\varepsilon) \sim \varepsilon^{-2}$ т.е. $D = 2$.



Сл.12.8. Графички приказ на нецелобројна димензија.

Да разгледаме сега еден облик со нецелобројна, односно дробна димензија. На сл.12.8а претставен е рамностран триаголник чија страна земаме дека е 1. Секоја страна ја делиме на три еднакви дела при што го отфрламе средниот дел и го заменуваме со два други дела со должина 1/3. Така ја добиваме фигурата на сл. 8.8б. Истата постапка ја повторуваме и за секоја страна од вака добиената фигура со што сега ја добиваме сл. 12.8в. Од начинот на конструкција на оваа крива позната како *Кохова крива („снегулка")* се гледа дека со намалувањето на страната за 1/3 бројот на страните на триаголникот се зголемува за 4, па според тоа фракталната димензија на оваа крива ќе биде

$$D = \frac{\ln 4}{\ln 3} \approx 1{,}262 \tag{12.13}$$

односно Коховата крива не е ни еднодимензионална, ни дводимензионлна, туку со нецелобројна, дробна димензија. Исто така таа е самболична, односно инваријантна на промената на размерот. Така, поради природата на конструкцијата, било која страна на триаголникот од 0 до 1 ќе изгледа исто како и нејзин дел од 0 до 1/3 доколку овој се зголеми трипати. Такви објекти кои се самослични и имаат нецелобројна димензија се нарекуваат **фрактали** според латинскиот збор *fractus* што значи раздробен. Овој термин го вовел математичарот Беноа Манделброт кој во 1983 година ја напишал книгата „Фрактална геометрија на природата" во која детално ја проучува геометријата на фракталите. Оттогаш фракталите станале научна хит тема, особено со брзиот развој на

компјутерите кои се како создадени за проучување на вакви фрактални објекти кои бараат примена на голем број итеративни операции.

Фракталите, односно фракталната димензија се тесно поврзани со феномените на хаос. Се покажало дека фракталните димензии на чудните атрактори се едни од основните квантитативни карактеристики кои ги разликуваат хаотичните движења од регуларните или случајни дижења. Така атракторот фиксна точка има димензија 0, граничниот циклус–димензија 1, а квазипериодичниот торус – димензија 2. Чудните, хаотични атрактори по правило, имаат комплицирана фина структура која локално не личи ни на еден Евклидски простор. Нивната димензија, значи, не може да биде целобројна и тие претставуваат типичен пример на фрактали.

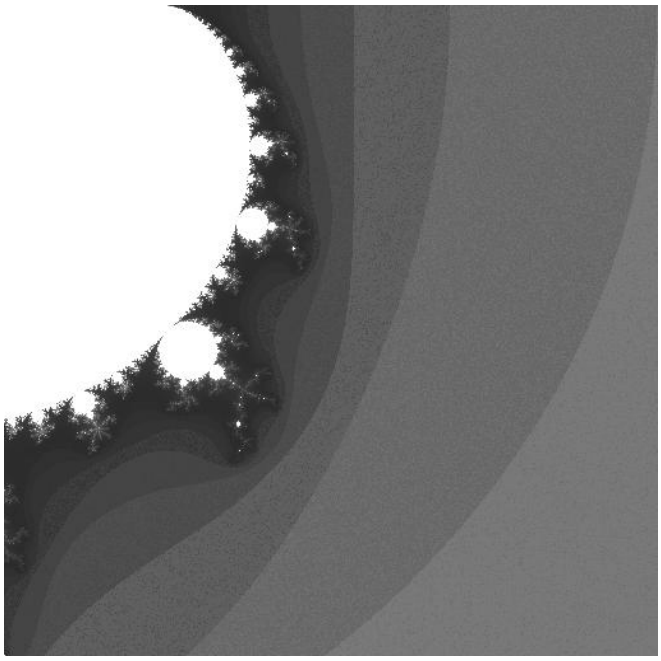
Фракталната димензија за прочуеното Манделбротово множество, слика 12.9, може да се покажи со многу едноставна итеративна постапка

$$z_{n+1} = z_n^2 + c \tag{12.14}$$

каде $z = x + iy$ и $c = a + ib$ се комплексни броеви. Значи, реалните и имагинарните делови се добиваа според

$$\begin{aligned} x_{n+1} &= x_n^2 - y_n^2 + a \\ y_{n+1} &= 2x_n y_n + b \end{aligned} \tag{12.15}$$

Почетната вредност за z се избира за нула, т.е $x_0 = y_0 = 0$. Во секоја итерација се пресметува апсолутната вредност $\left|z^2\right| = a^2 + b^2$ и се проверува дали ја надминува вредноста 4.



Сл.12.9. Множество на Манделброт прикажано во нијанси на сиво.

Покажано е дека кога $\left|z^2\right|$ ја надминува вредноста 4 тогаш тоа оди во бескрајност. Бројот на итерации потребни $\left|z^2\right|$ да ја надмине вредноста 4 ја одредува

бојата на пикселот за дадени вредности на *c* во определена рамка. Наспроти едноставната итеративна постапка, сликата што ќе се добие изобилува со множество бои и форми за кои е покажано дека имаат фрактална структура.

12.8. Примена

Теоријата на хаос наоѓа широка примена во небесната механика, во хидромеханика при објаснувањето на проблемот на турбуленција, во електротехниката кај појавите на нелинеарни струи, во физиката на високи енергии кај акцелераторите, во физиката на плазмата, во квантната механика при проучување на феноменот на постоење на квантен хаос, во телекомуникациите при проучување на начините на најефективно пренесување на информациите, во хемиската динамика, во метеорологијата при објаснување на атмосферските феномени, во економијата при објаснување на движењата на цените, во демографијата при објаснување на масовните движења, во медицината при проучување на работата на срцето, брановите во мозокот и разни други физиолошки процеси, во биологијата, во екологијата итн. Благодарејќи на универзалноста на своите законитости и сеопфатноста на појавите кои ги разгледува, а користејќи се со развојот на компјутерската техника, теоријата на хаосот станува еден од темелите на современата наука кој брзо се развива. Хаосот е насекаде околу нас, само треба да се разбере и потоа искористи.

12.9. Проекти

1. Да се напише програма во фортран со која ќе се реплицира чудниот атрактор на Лоренц прикажан на слика 12.3?

2. Да се напише програма во фортран со која ќе се реплицира множеството на Манделброт прикажано на слика 12.9: