Записки за упражненията по Числени методи на линейната алгебра 2016/2017 година

Тихомир Иванов

Съдържание

Глава 1

Въведение в числените методи

1.1 Какво представляват числените методи?

Най-общо казано, числените методи са техники, чрез които математически задачи се представят във вид, в който могат да бъдат решени с помощта на аритметични операции. Въпреки че има много видове числени методи, те имат обща характеристика – изискват голям брой аритметични пресмятания. Ето защо тяхното прилагане става посредством имплементирането им в компютърни програми.

Обикновено числените методи включват **апроксимация** (т.е. приближение) на оригиналната математическа задача. Ето защо можем да ги разглеждаме като техники за **приближеното решаване** на дадена математическа задача посредством аритметични операции.

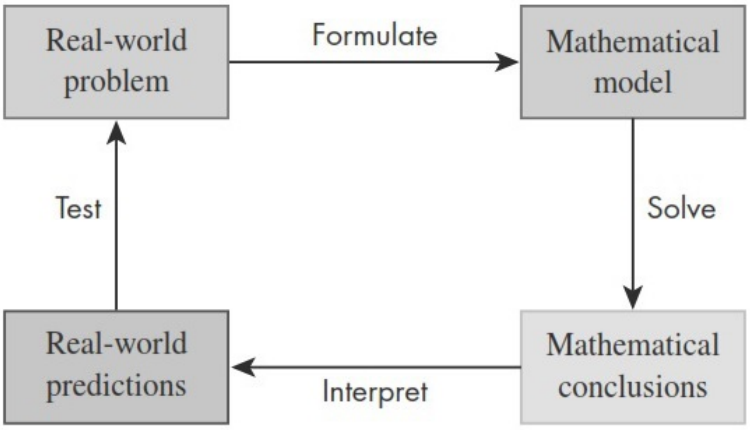
Преди да преминем към разглеждането на конкретни числени методи, нека разгледаме въпроса защо изобщо е необходимо тяхното използване. Математиката е езикът, на който се описват процесите от света около нас. За да изучим даден реален процес или да решим дадена практическа задача, ние трябва да определим кои са основните характеристики, които ги описват – това са някакви величини, които дават информация за съответния процес (например време, скорост, температура, сила, бързодействие на алгоритъм, компресия на данни и др.). Величините се измерват в дадени мерни единици, т.е. им се съпоставят някаква числени стойности. Изучавайки даден процес, ние искаме да изучим зависимостите между величините, които го описват, като за целта създаваме и изследваме математически модел на процеса.

Най-общо казано, **математически модел** е описание на някакъв реален процес или реална задача на езика на математиката. Често това става чрез функция или уравнение (или система от уравнения), обикновено диференциално, свързващо величините, описващи процеса. Математическият модел обаче може да представлява и друг математически обект. Например, изследвайки една компютърна мрежа, може да се наложи решаването на задача от теория на графите.

Целта на математическото моделиране е да се опише даденият процес и по-добре да се разберат механизмите, които го обуславят, както и, евентуално, да се направят компютърни симулации и/или предвиждания за бъдещото му поведение.

Често в литературата се дава следната схема, описваща методологията на

математическото моделиране:



Да коментираме накратко етапите, описани в нея.

1. Имайки някаква реална задача, първото, което трябва да направим, е да **формулираме математически модел**, който да я описва. За целта трябва да определим основните величини, които характеризират процеса (от гледна точка на математиката – променливи и параметри), и да съставим математическата задача, която ги свързва (например диференциално уравнение, оптимизационна задача и др.). Важно е да се има предвид, че **всеки математически модел е една абстракция, идеализация на реалния процес**. В него трябва да има баланс – от една страна, моделът трябва достатъчно подробно да описва процеса, така че резултатите от него да бъдат полезни, но, от друга страна, трябва да е достатъчно прост, за да позволява математическо изследване. Всеки модел се базира на някакви допускания (абстракции), които позволяват опростяването на реалната ситуация. При създаването на математически модел използваме физически закони, обуславящи процеса, и математически техники, за да получим уравнения (или други обекти), свързващи променливите. В ситуации, когато не са известни физически закони, които да ни ръководят, може да е необходимо да се съберат данни от експерименти, на базата на които да се състави математическият модел.

2. Имайки предвид, че математическият модел на един процес представлява математическа задача, **вторият етап е да решим тази задача** и да получим математически заключения. **В настоящия курс ние ще разгледаме именно техники, които ще можем да използваме в този етап**. Важно е да се отбележи, че практическите задачи водят твърде често до математически задачи, които не могат да бъдат решени със стандартните аналитични техники. Както знаем, дори просто изглеждащи алгебрични уравнения като полиномиалните уравнения от пета и по-висока степен в общия случай не могат да бъдат решени точно. Същото се отнася за повечето определени интеграли и др. Въпреки това обаче съществуват техники за тяхното **приближено решаване** и именно с такива ще се запознаем в курса по Числени методи.

3. След като сме решили (в някакъв смисъл) математическата задача, **следва да интерпретираме резултатите от гледна точка на реалния процес**.

4. Да обърнем внимание, че резултатите за реалния процес, които получихме, са следствие на математическия модел, а не на самия процес. От друга страна, казахме, че математическият модел е една абстракция на реалния процес, т.е. може и да не го описва достатъчно добре. Затова е необходимо да направим **проверка дали тези резултати съответстват на реалността**. Ако това е така, можем да считаме, че моделът ни е удачен. В противен случай се връщаме в началото и трябва да модифицираме модела така, че той да отразява действителността по-добре. С други думи, математическото моделиране е един **итеративен процес**.

Да формулираме няколко причини за изучаването на числени методи:

- Числените методи са много мощни средства за решаването на реални задачи. С тяхна помощ е възможно решаването на големи системи уравнения, справянето с нелинейности и сложни геометрии, които са присъщи за задачите от практиката и към които често е невъзможно да се подходи аналитично.

- Често в практиката се налага използването на готови софтуерни продукти, чието действие се базира на дадени числени методи. Интелигентното използване на тези продукти изисква познаването на основната теория, обуславяща съответните числени методи.

- Невинаги готовите софтуерни продукти са достатъчни за решаването на дадена практическа задача. В тези случаи познаването на основната теория в областта на числените методи ни позволява проектирането и направата на собствени програми.

1.2 Грешка. Източници на грешка. Представяне на числата в компютъра.

Както отбелязахме, повечето числени методи включват някаква апроксимация. Ето защо разбирането на идеята за грешка е от много голяма важност за ефективното им използване. Нека първо дадем следните дефиниции:

Дефиниция 1. ***Абсолютна грешка*** *наричаме разликата между точната и приближената стойност при дадена апроксимация:*

$$\varepsilon_a := exact\ value - approximation.$$

Дефиниция 2. ***Относителна грешка*** *дефинираме по следния начин:*

$$\varepsilon_r := \frac{exact\ value - approximation}{exact\ value} = \frac{\varepsilon_a}{exact\ value}.$$

Основните източници на грешка при решаването на една практическа задача са следните:

- Математическият модел – както казахме, математическият модел сам по себе си е една апроксимация на реалността, с други думи самото му съставяне въвежда грешка по отношение на реалния процес.

- Грешка от числения метод – обикновено числените методи се базират на някаква апроксимация, т.е. въвеждат някаква грешка. Тъй като ние на практика не знаем точното решение на съответната математическа задача, обикновено е невъзможно да намерим каква е грешката при въпросната апроксимация. От друга страна, за да разберем дали даден числен метод е приложим, или не, ние трябва да знаем с каква точност той ще реши съответната задача. Затова се налага да се правят оценки на грешката, например да се намери някаква стойност, която тя със сигурност не надминава, или да се определи нейният порядък.

- Грешки от закръгляване – те са свързани с начина, по който числата се представят в компютъра. Ще се спрем по-подробно на този вид грешка в настоящия параграф.

- Грешки от входните данни – математическите модели обикновено зависят от някакви параметри, които се определят чрез провеждането на експерименти, правенето на измервания. Дори и най-съвършената техника позволява измерване с определена точност, т.е. стойностите на измерените величини, с които работим, също носят определена грешка.

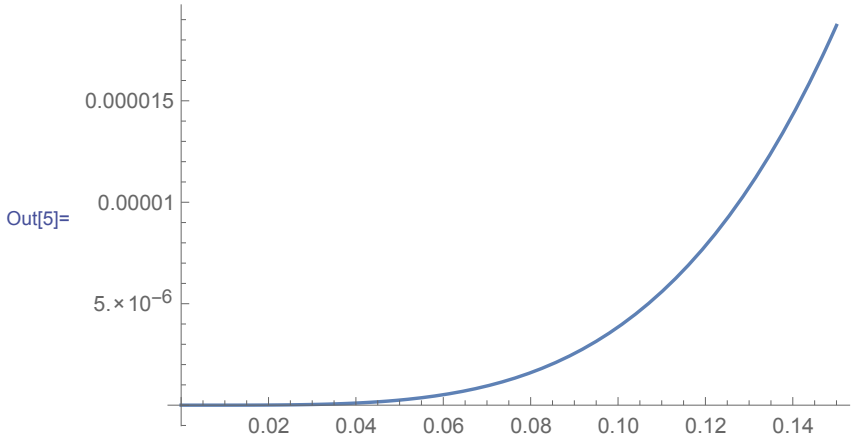
Задача 1. Да се намерят абсолютната и относителната грешка, които се получават, като приближим стойността на e^x с полинома на Тейлър от трета степен $1 + x + x^2/2 + x^3/6$ за $x = 0.1$. Да се построи графика на относителната грешка за $x \in [0, 0.15]$.

Решение. Ще използваме системата Wolfram Mathematica.

```
In[1]:= absErr[x_] := E^x - (1 + x + x^2 / 2 + x^3 / 6)
        relErr[x_] := absErr[x] / E^x
        absErr[0.1]
        relErr[0.1]
Out[3]= 4.25141 × 10^-6
Out[4]= 3.84683 × 10^-6
```

За графиката получаваме

```
In[5]:= Plot[relErr[x], {x, 0, 0.15}]
```



Грешката расте с нарастването на аргумента, което е естествено, имайки предвид, че експоненциалната функция расте значително по-бързо от полинома. Все пак в разглеждания интервал тя не надминава 0.002%. □

Сега ще се спрем на грешката от закръгляване. Причината за нея, както казахме, е начинът, по който числата се представят в компютъра. По-точно, ще се занимаем с т.нар. числа с плаваща точка (floating-point). При този подход числото се представя чрез дробна част, наречена **мантиса**, и цяло число, което се нарича **експонента** или характеристика по следния начин:

$$m.b^e,$$

където m е мантисата, b е основата на бройната система, в която работим (в компютъра $b = 2$), e – експонентата. Например $156.78 = 0.15678 \times 10^3$ е представянето във вид на число с плаваща точка на числото 156.78 в десетична бройна система. Да обърнем внимание, че обикновено дробната част се нормализира, така че първият знак след десетичната точка да бъде различен от нула.

Предимството на числата с плаваща точка е, че те позволяват представянето както на дроби, така и на много големи числа. От друга страна обаче, се появява т.нар. грешка от закръгляване, тъй като мантисата може да съдържа само краен брой значещи цифри. В компютъра, с t-битова дума могат да се представят най-много 2^t различни реални числа. Очевидно има безброй много числа, които не могат да бъдат представени точно. За тяхното представяне се използва най-близкото число, което може да се представи точно. По този начин въвеждаме грешка от закръгляване. Нещо повече, тъй като има максимално (по абсолютна стойност) число, то при опит да запишем число, което има по-голяма стойност, получаваме т.нар грешка "overflow". Освен това, по аналогична причина, не можем да представяме много малки по абсолютна стойност числа (т.е. близки до нулата). Опитът за записването на такова число води до грешка "underflow". Нека отбележим, че някои компютри заместват "underflow" с нула.

За да илюстрираме ефектите от грешките от закръгляване, нека разгледаме един хипотетичен компютър, който използва десетична бройна система и представя числата с плаваща точка чрез 1-цифрена експонента със знак и 3-цифрена мантиса.

Най-малкото положително число, което можем да представим в този компютър, е 0.100×10^{-9}, а следващото по големина число е 0.101×10^{-9}. Всяко друго

число между тези две трябва да бъде апроксимирано. Това ни дава максимална грешка от закръгляване 0.5×10^{-12}.

Най-голямото число, което можем да представим, е 0.999×10^9, докато следващото по-малко число е 0.998×10^9, което дава максимална грешка 0.5×10^6.

Вижда се, че грешката съществено зависи от големината на числата, които апроксимираме. Затова е по-смислено да говорим за относителната вместо за абсолютната грешка. Може да се покаже, че тя е под 5×10^{-3}, т.е. под 0.5%. Да разгледаме следния пример, който ще ни покаже защо относителната грешка е по-добрия показател за точността на приближението. Ясно е, че абсолютна грешка, равна на 1, при число от порядъка на 10^8 е, по принцип, много по-пренебрежима, отколкото грешка от 0.001 при число от порядъка на 10^{-2}. Относителните грешки в този случай са съответно 10^{-8} и 0.1.

Изобщо, при работа в система числа с плаваща точка с p значещи цифри, може да се покаже, че за относителната грешка ε_r е в сила

$$|\varepsilon|_r \leq \mathbf{0.5 \times 10^{-p}} =: \boldsymbol{\varepsilon}.$$

При работа с числа с двойна точност (double), имаме $p \approx 16$, а с единична (float) – $p \approx 7$. Като резултат от грешките от закръгляване, дори фундаменталните асоциативни и дистрибутивни закони на алгебрата може и да не са в сила при числени пресмятания. Да разгледаме следните примери:

- Асоциативност на събирането

  $$a + (b + c) = (a + b) + c.$$

  Нека $a = 0.456 \times 10^{-2}$, $b = 0.123 \times 10^0$, $c = -0.128 \times 10^0$. Тогава

  $$(a + b) + c = 0.128 \times 10^0 - 0.128 \times 10^0 = 0,$$
  $$a + (b + c) = 0.456 \times 10^{-2} - 0.500 \times 10^{-2} = -0.440 \times 10^{-3}.$$

  Очевидно първият резултат не е верен и причината за това е **събирането на голямо с малко число**. Можем да разгледаме и още по-показателен пример за този проблем – ако съберем 0.100×10^0 с 0.100×10^{-3}, резултатът е 0.100×10^0, т.е. все едно не сме извършили събирането!

- Асоциативност на умножението

  $$a \times (b \times c) = (a \times b) \times c.$$

  При стойности $a = 10^{-6}$, $b = 10^{-6}$, $c = 10^8$ лявата страна на асоциативния закон дава верен резултат. При използване на дясната страна обаче, при изчисленията ще се получи "underflow". Виждаме, че дори при работата с числа, които могат да бъдат представени точно, не сме застраховани от наличието на тази грешка. С други думи **за това как ще протече изпълнението на един алгоритъм сериозно влияние може да има редът, в който се изпълняват операциите**.

Горните примери ни показват, че всяка аритметична операция, която извършваме, би могла да въведе грешка. Вместо да работим с точната стойност

на $a \odot b$, където $\odot$ е някоя от операциите събиране, изваждане, умножение, деление, работим с

$$fl(a \odot b) = (a \odot b)(1 + \delta)$$

и δ е съответната относителна грешка, която е ограничена от $|\delta| < \epsilon$. Както казахме, числените методи се базират на голям брой аритметични операции, така че това е нещо, което не можем да пренебрегнем при тяхното използване.

За да илюстрираме ефекта на грешките от закръгляване, нека разгледаме следния пример.

Задача 2. Даден е алгебричният полином

$$p(x) = (x-2)^9 = x^9 - 18x^8 + x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512.$$

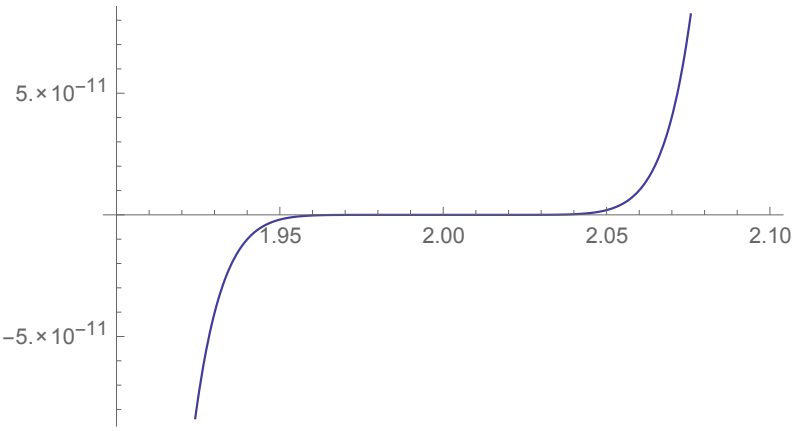
Да се построи неговата графика, като за пресмятане на стойностите му в точката x се използва

а) $p(x) = (x-2)^9$

б) $p(x) = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$.

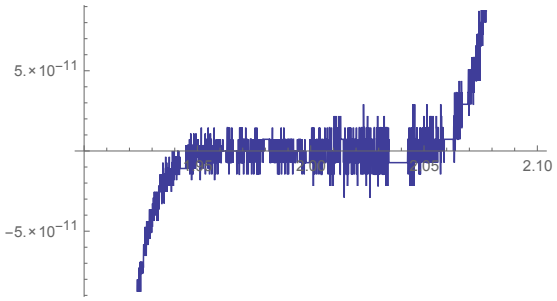
Решение. Решението на а) прилагаме по-долу.

```
f[x_]:=(x-2)^9
Plot[f[x],{x,1.9,2.1}]
```



За б) имаме.

```
p[x_]:=-512+2304 x-4608 x^2+5376 x^3-4032 x^4+2016 x^5-672 x^6+144 x^7-18 x^8+x^9
Plot[p[x],{x,1.9,2.1}]
```



Очевидно във втория случай резултатът е по-лош. Причината е в големия брой аритметични операции, които извършваме при него. Грешките от закръгляване водят до появилия се "шум". □

Вземайки предвид казаното дотук, **числените методи, които използваме трябва да са такива, че грешките от закръгляване да не водят до драстично изменение на резултата. Такива методи се наричат устойчиви.**

Глава 2

Директни методи за решаване на системи линейни алгебрични уравнения

Основна задача на числените етоди на линейната алгебра е решаването на системи линейни уравнения

$$A\overline{x} = \overline{b}. \tag{2.1}$$

Съществуват както директни методи (ще разгледаме модификации на метода на Гаус), така и итерационни методи (тръгвайки от начално приближение $\overline{x}_0$, построяваме по някакво правило редицата от последователни приближения $\overline{x}_0$, $\overline{x}_1$,...,$\overline{x}_n$,..., която да клони към точното решение на системата (2.1)). Когато изучаваме един числен метод, следва да се спрем на следните въпроси:

- описание и имплементация на алгоритъма;
- реализуемост – в кои случаи методът приключва успешно работа и връща работа;
- устойчивост – какъв е ефектът на грешките от закръгляване; казваме, че един метод е неустойчив, ако малки грешки от входните данни (от закръгляване) водят до големи грешки в крайния резултат;
- "скорост" на алгоритъма – можем да разглеждаме този въпрос като брой операции и като скорост на сходимост при итерационните методи;
- оценка на грешката – ако нямаме оценка за точността на резултата, който получаваме, то той би бил абсолютно неизползваем.

2.1 Метод на Гаус и негови модификации

2.1.1 Метод на Гаус

Методът на Гаус е класическият метод за решаване на системи линейни уравнения. При него оригиналната система се свежда до такава с горна триъгълна матрица (прав ход на алгоритъма). Получената система вече може да бъде лесно решена, като от последното уравнение намерим последното неизвестно,

след това от предпоследното уравнение – предпоследното неизвестно и т.н. Ще илюстрираме метода със следния пример.

Задача 3. Да се реши системата

$$\begin{array}{rcrcrcr} 4x_1 & - & 2x_2 & + & x_3 & = & 11 \\ -2x_1 & + & 4x_2 & - & 2x_3 & = & -16 \\ x_1 & - & 2x_2 & + & 4x_3 & = & 17 \end{array}$$

Решение. Разширената матрица на системата има вида

$$[A|b] = \left[\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ -2 & 4 & -2 & -16 \\ 1 & -2 & 4 & 17 \end{array}\right].$$

- Прав ход на алгоритъма:

  1. На първата стъпка ($i = 1$) към втория ред прибавяме първия, умножен по $1/2$, а към третия прибавяме първия, умножен по $-1/4$. Получаваме матрицата

  $$[A^{(1)}|b^{(1)}] = \left[\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ 0 & -3 & -3/2 & -10.5 \\ 0 & -3/2 & 15/4 & 14.25 \end{array}\right].$$

  2. На втората стъпка ($i = 2$) към третия ред прибавяме втория, умножен по $1/2$. Получаваме окончателно горната триъгълна матрица

  $$[A^{(2)}|b^{(2)}] = \left[\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ 0 & 3 & -3/2 & -10.5 \\ 0 & 0 & 3 & 9 \end{array}\right].$$

- Обратен ход на алгоритъма:

  1. $i = 3$: $x_3 = 3$.
  2. $i = 2$: $x_2 = \frac{-10.5+1.5\times 3}{3} = -2$.
  3. $i = 1$: $x_1 = \frac{11-1\times 3+2\times(-2)}{4} = 1$.

□

Да запишем алгоритъма в общ вид.

1. **Прав ход на алгоритъма** (получаваме горна триъгълна матрица):

   За $i = \overline{1, n-1}$ (на i-тата стъпка от алгоритъма правим всички елементи от i-тия стълб под диагоналния равни на 0)

   За $j = \overline{i+1, n}$ от j-тия ред вадим i-тия, умножен по $l_j^{(i)} = \frac{a_{ji}^{(i-1)}}{a_{ii}^{(i-1)}}$:

   $$a_{jk}^{(i)} = a_{jk}^{(i-1)} - l_j^{(i)} a_{ik}^{(i-1)},\ k = \overline{i, n}, \tag{2.2}$$
   $$b_j^{(i)} = b_j^{(i-1)} - l_j^{(i)} b_i^{(i-1)}. \tag{2.3}$$

2. **Обратен ход на алгоритъма** (намираме неизвестните):

   За $i = n, n-1, \ldots, 1$ намираме x_i по формулата

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}} \tag{2.4}$$

Прилагаме една примерна имплементация на метода на Гаус на Wolfram Mathematica.

```
In[14]:= (*Input data*)
A = {{4, -2, 1}, {-2, 4, -2}, {1, -2, 4}};
b = {11, -16, 17};
n = Length[A];
(*Forward elimination*)
For[i = 1, i ≤ n, i++,
 For[j = i + 1, j ≤ n, j++,
  l = A[[j, i]] / A[[i, i]];
  A[[j, i]] = 0;
  For[k = i + 1, k ≤ n, k++,
   A[[j, k]] = A[[j, k]] - l * A[[i, k]];
  ];
  b[[j]] = b[[j]] - l * b[[i]];
 ]
]
(*Backward substitution*)
x = Table[0, {i, n}] ; (*We initilize a list,
in which we shall keep the values of the unknowns*)
For[i = n, i ≥ 1, i--,
 x[[i]] = (b[[i]] - Sum[A[[i, j]] x[[j]], {j, i + 1, n}]) / A[[i, i]]

]
x
```

Методът на Гаус обаче има един много сериозен недостатък. Нека разгледаме следната матрица.

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

На първата стъпка алгоритъмът ни казва от втория ред да извадим първия, умножен по a_{21}/a_{11}, но $a_{11} = 0$, което означава, че алгоритъмът не може да продължи работа. Изобщо казано, на k-тата стъпка имаме операцията деление на a_{kk}. Следователно **методът на Гаус е реализуем, само когато всички водещи елементи са различни от 0.**

По-сериозният проблем обаче се вижда от следния пример. Да разгледаме системата, чиято разширена матрица е

$$[A|b] = \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 1 & 0 \end{array}\right].$$

На първата стъпка от алгоритъма получаваме матрицата

$$[A^{(1)}|b^{(1)}] = \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 1-10^{20} & -10^{20} \end{array}\right].$$

Нека работим в система от числа с плаваща точка с 16-цифрена мантиса. С други думи, можем да запазим в паметта 16 значещи цифри. За да запишем числото $1-10^{20}$ обаче ще са ни необходими 20 значещи цифри. С други думи, в паметта това число ще се закръгли към най-близкото число, което може да бъде представено в разглежданата система от числа с плаваща точка. Нека приемем, че това е -10^{20}. Тогава, вместо матрицата $A^{(1)}$, в паметта ще се запази матрицата

$$float([A^{(1)}|b^{(1)}]) = \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & -10^{20} & -10^{20} \end{array}\right].$$

Единствената грешка от закръгляване е в елемента a_{22}, като относителната грешка е от порядъка на 10^{-20}. Да видим обаче до какво води тази изключително малка грешка. Лесно се вижда, че решението на системата с разширена матрица $[\overline{A}^{(1)}|b^{(1)}]$ е $[0,1]^T$. Оригиналната система обаче има решение, което е приблизително равно на $[-1,1]^T$! Виждаме, че съвсем малка грешка от закръгляване при метода на Гаус може да доведе до съвършено различен резултат.

Методът на Гаус е неустойчив – малки грешки от закръгляване могат да доведат до много големи грешки в крайния резултат.

Важно е да се подчертае каква е причината за появилата се неустойчивост – тя се крие в много малкия по абсолютна стойност водещ елемент a_{11}. Делейки на него, получихме много голям по абсолютна стойност елемент a_{22}, което доведе до невъзможността да запишем полученото число в система с 16-цифрена мантиса. И така, да обобщим проблемите при метода на Гаус:

- Ако някой от водещите елементи е 0, алгоритъмът не може да продължи работа;
- Ако някой от водещите елементи е малък по абсолютна стойност, това може да доведе до неустойчивост – малки грешки от закръгляване водят до големи грешки в резултата.

2.1.2 Метод на Гаус с частичен избор на главния елемент

Методът на Гаус с частичен избор на главния елемент има за цел да реши двата проблема, които посочихме в края на предходния параграф. Тъй като причината за тези проблеми е в малка абсолютна стойност на водещия елемент, на k-тата стъпка за водещ елемент избираме най-големия по абсолютна стойност елемент от k-тия стълб на матрицата. За целта на практика разменяме k-тия ред и реда, съдържащ въпросния елемент. Това не променя нищо при решаването на системата, просто пренарежда уравненията.

Да разгледаме имплементацията на метода:

In[1]:=
```
(*Input data*)
A = {{4, -2, 1}, {-2, 4, -2}, {1, -2, 4}};
b = {11, -16, 17};
n = Length[A];
(*Forward elimination*)
For[i = 1, i ≤ n, i++,
 (*Find the index of the row with maximal element*)
 maxIndex = i;
 For[j = i + 1, j ≤ n, j++,
  If[Abs[A[[j, i]]] > Abs[A[[maxIndex, i]]],
   maxIndex = j]
 ];
 (*Change the rows with indices i and maxIndex*)
 For[j = i, j ≤ n, j++,
  temp = A[[i, j]];
  A[[i, j]] = A[[maxIndex, j]];
  A[[maxIndex, j]] = temp;
  temp = b[[i]];
  b[[i]] = b[[maxIndex]];
  b[[maxIndex]] = temp;
 ];
 (*Eliminate the elements under the main diagonal in the i-th column*)
 For[j = i + 1, j ≤ n, j++,
  l = A[[j, i]] / A[[i, i]];
  A[[j, i]] = 0;
  For[k = i + 1, k ≤ n, k++,
   A[[j, k]] = A[[j, k]] - l * A[[i, k]];
  ];
  b[[j]] = b[[j]] - l * b[[i]];
 ]
]
(*Backward substitution*)
x = Table[0, {i, n}] ;(*We create a list,
in which we shall keep the values of the unknowns*)
For[i = n, i ≥ 1, i--,
 x[[i]] = (b[[i]] - Sum[A[[i, j]] x[[j]], {j, i + 1, n}]) / A[[i, i]]

]
x
```

Могат да се намерят академични примери, за които методът на Гаус с частичен избор на главния елемент е неустойчив. Да разгледаме следния пример.

Нека имаме матрицата

$$A = \begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

По метода на Гаус с частичен избор на главния елемент (да отбележим, че в случая смени на редове не са необходими) се получават последователно матриците

$$A = \begin{bmatrix} 1 & & & & 1 \\ 0 & 1 & & & 2 \\ 0 & -1 & 1 & & 2 \\ 0 & -1 & -1 & 1 & 2 \\ 0 & -1 & -1 & -1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & & & & 1 \\ 0 & 1 & & & 2 \\ 0 & 0 & 1 & & 4 \\ 0 & 0 & -1 & 1 & 4 \\ 0 & 0 & -1 & -1 & 4 \end{bmatrix}$$

$$\longrightarrow \cdots \longrightarrow \begin{bmatrix} 1 & & & & 1 \\ 0 & 1 & & & 2 \\ 0 & 0 & 1 & & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{bmatrix}.$$

Ако вземем аналогична на матрицата A, но с по-висока размерност, нека бъде $n \times n$, е ясно, че горната триъгълна матрица, която ще получим по метода на Гаус, ще съдържа елемента 2^{n-1}, което за достатъчно голямо n, аналогично на примера от предишния параграф, ще доведе до съществени грешки в резултата.

Въпреки широката практическа употреба на метода на Гаус с частичен избор на главния елемент обаче не е известен пример, идващ от практиката, за който методът да е неустойчив. Могат да се направят и някои вероятностни съображения, които показват, че вероятността методът да е неустойчив за произволна зададена матрица е нищожна. С други думи, **за всички практически цели методът на Гаус с частичен избор на главния елемент може да се разглежда като "устойчив"**.

Забележка. Тук няма да се спираме на метода на Гаус с (пълен) избор на главния елемент, тъй като неговото програмно реализиране е по-сложно, а подобряването на устойчивостта в сравнение с метода на Гаус с частичен избор на главния елемент е много малко. Затова на практика се използва най-вече методът на Гаус с частичен избор на главния елемент.

2.1.3 Метод на Гаус–Жордан

Разликата между метода на Гаус–Жордан и класическия метод на Гаус е, че на k-тата стъпка при метода на Гаус-Жордан k-тият ред се изважда от всички останали редове, а не само от редовете след k-тия. По този начин се получава диагонална матрица и системата може да се реши непосредствено. Методът на Гаус–Жордан също може да бъде приложен с частичен или пълен избор на главния елемент. Имплементацията оставяме за самостоятелна работа.

2.1.4 Сложност на метода на Гаус

За да оценим сложността на метода на Гаус, трябва да преброим колко операции се извършват при решаването на дадена система. Първо, нека видим колко са операциите в правия ход на алгоритъма.

Алгоритъмът привежда оригиналната матрица до горна триъгълна за $n-1$ стъпки. Тогава броят операции е

$$N = \sum_{i=1}^{n-1} (\text{брой операции на } i\text{-тата стъпка}).$$

На i-тата стъпка вадим i-тия ред от всички следващи. Тоест операциите на i-тата стъпка ще получим, като съберем операциите при изваждането на i-тия от $i+1$-вия, $i+2$-ия и т.н.

$$N = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\text{брой операции при изваждането на } i\text{-тия ред от } j\text{-тия}).$$

Окончателно получаваме

$$\begin{aligned} N &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (1 + \sum_{k=i}^{n} 2) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (2n - 2i + 1) \\ &= \sum_{i=1}^{n-1} (2n - 2i + 1)(n - i - 1) \\ &= \sum_{i=1}^{n-1} (2n^2 - 2ni - 2n - 2ni + 2i^2 + 2i + n - i - 1). \end{aligned}$$

Оценявайки сложността на един алгоритъм, ние се интересуваме от членовете от най-висок ред. В случая това са оцветените в червено членове. За пресмятането на горната сума ще използваме известните формули

$$\begin{aligned} \sum_{i=1}^{n-1} i &= \frac{n(n+1)}{2} = \frac{n^2}{2} + O(n), \\ \sum_{i=1}^{n-1} i^2 &= \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + O(n^2). \end{aligned}$$

Тогава

$$N = 2n^3 - 2n\frac{n^2}{2} - 2n\frac{n^2}{2} + 2\frac{n^3}{3} + O(n^2) = \frac{2n^3}{3} + O(n^2).$$

Окончателно получихме, че методът на Гаус има сложност $\frac{2}{3}n^3 + O(n^2)$. Това означава, че **методът на Гаус не е приложим за системи с много голяма размерност**. Ако искаме да решим система с 10^9 уравнения например, трябва да направим от порядъка на 10^{27} операции, което е твърде много дори за съвременните изчислителни машини. В тези случаи се използват итеративни методи,

с които ще се запознаем по-късно в курса. **За системи с по-малка размерност обаче методът на Гаус (с частичен избор на главния елемент) е най-широко използваният метод**, тъй като със сигурност дава решението за практически всяка система (методът е точен, т.е. няма грешка от апроксимация, а само от закръгляванията при работа в компютърна аритметика).

Забележка. Лесно се вижда, че при метода на Гаус с частичен избор на главния елемент изборът на главен елемент на всяка стъпка и размяната на редовете добавя $O(n^2)$ операции, т.е. неговата сложност е също $2/3n^3 + O(n^2)$.

2.2 LU декомпозиция

В редица случаи е удобно **дадена матрица да се разложи на произведение от матрици** по определен начин. Например диагонализирането на една матрица, т.е. представянето ѝ във вида

$$A = T\Lambda T^{-1},$$

където Λ е диагоналната матрица от собствените стойности на матрицата A, а стълбовете на T са съответстващите им собствени вектори ни дава удобен начин за повдигането на матрица на дадена степен. Имаме

$$A^n = \underbrace{(T\Lambda\cancel{T^{-1}})(\cancel{T}\Lambda\cancel{T^{-1}})(\cancel{T}\Lambda\cancel{T^{-1}})\dots(\cancel{T}\Lambda T^{-1})}_{n \text{ пъти}} = T\Lambda^n T^{-1}.$$

Последното може да бъде лесно пресметнато, тъй като, за да повдигнем диагоналната матрица Λ на n-та степен трябва просто да повдигнем диагоналните елементи на съответната степен и не е необходимо да се правят n на брой матрични умножения, какъвто щеше да бъде случаят, ако директно бяхме повдигнали A^n.

Сега ще се спрем на една друга важна декомпозиция и ще коментираме случаите, когато тя е полезна. Ясно е, че ако една линейна система има триъгълна матрица, то нейното решаване е непосредствено – от всяко уравнение намираме едно неизвестно. Тогава, ако можем дадена матрица A да представим във вида

$$A = LU,$$

където L и U са съответно долна триъгълна и горна триъгълна матрици, то решаването на системата $Ax = b$ е еквивалентно на решаването на системата $LUx = b$ и се свежда до последователното решаване на системите $Ly = b$ и $Ux = y$, които имат триъгълни матрици.

Преди да покажем как може да се намери искането разлагане на матрицата A, нека припомним някои факти от линейната алгебра:

1. Умножение с матрица отляво (умножение по редове).

   Да разгледаме следното матрично равенство:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \cdots B_1 \cdots \\ \cdots B_2 \cdots \\ \cdots B_3 \cdots \end{bmatrix} = \begin{bmatrix} \cdots C_1 \cdots \\ \cdots C_2 \cdots \\ \cdots C_3 \cdots \end{bmatrix}.$$

В сила е следната зависимост:

$$C_i = a_{i1}B_1 + a_{i2}B_2 + a_{i3}B_3,\ i = \overline{1,3}.$$

С други думи, ако умножим матрицата B отляво с A, то i-тият ред на резултата се получава като линейна комбинация на редовете на B. Коефициентите в тази линейна комбинация са елементите от i-тия ред на матрицата A.

За простота разгледахме случая на матрици 3×3. Обобщението за произволни матрици е тривиално.

2. Умножение отдясно (умножение по стълбове)

   Нека сега да видим какъв е ефектът на това да умножим една матрица отдясно с друга:

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ B_1 & B_2 & B_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ C_1 & C_2 & C_3 \\ \vdots & \vdots & \vdots \end{bmatrix}.$$

   В този случай за стълбовете на резултата е в сила

$$C_i = a_{1i}B_1 + a_{2i}B_2 + a_{3i}B_3,\ i = \overline{1,3}.$$

С други думи, ако умножим матрицата B отдясно с A, то i-тият стълб на резултата се получава като линейна комбинация на стълбовете на B. Коефициентите в тази линейна комбинация са елементите от i-тия стълб на матрицата A.

2.2.1 Алгоритъм

И така, вече сме готови да видим как може да се намери исканото разлагане. Ще го илюстрираме с пример. Да разгледаме матрицата

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix}.$$

Извършваме правия ход на метода на Гаус и получаваме горна триъгълна матрица:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} =: U.$$

С други думи, методът на Гаус **трансформира** матрицата A до горната триъгълна матрица U. Но това е една линейна трансформация, следователно тя може да се представи чрез умножнението на A с дадена матрица отляво, т.е. $L^{-1}A = U$, където L^{-1} е матрицата на въпросната трансформация. Тогава е в сила $A = LU$. Въпросът е как да намерим матрицата L (която, както ще видим, е долна триъгълна матрица, т.е. ни дава исканото разлагане).

- На първата стъпка от метода на Гаус матрицата A се преобразува до $A^{(1)}$ и нека матрицата на това преобразование е L_1^{-1} ($L_1^{-1}A = A^{(1)}$). Първият ред на матрицата $A^{(1)}$ е равен на първия ред на матрицата A:

$$A_1^{(1)} = 1 \times A_1 + 0 \times A_2 + 0 \times A_3. \tag{2.5}$$

Вторият ред на $A^{(1)}$ се получава, като от втория ред на A извадим първия, умножен по 1, т.е.

$$A_2^{(1)} = -1 \times A_1 + 1 \times A_2 + 0 \times A_3. \tag{2.6}$$

Аналогично

$$A_3^{(1)} = -1 \times A_1 + 0 \times A_2 + 1 \times A_3. \tag{2.7}$$

Вземайки предвид (2.5), (2.6), (2.7) и това, което казахме за умножение с матрица отляво, е ясно, че $A^{(1)} = L_1^{-1}A$, където

$$L_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

- Аналогично $U = L_2^{-1}A^{(1)}$, където

$$L_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}.$$

Тогава $(L_2^{-1}L_1^{-1})A = U$ и следователно $A = L_1L_2U$.

- Следващият въпрос е как да намерим обратните матрици на L_1^{-1} и L_2^{-1}. Ако разглеждаме матрицата L_1^{-1} като трансформация, можем лесно да отговорим на този въпрос. Това е трансформация, която приложена върху дадена матрица X вади първия ѝ ред от втория. Обратната трансформация тогава трябва да прибавя първия ред на X към втория. Аналогично обратната трансформация трябва да прибавя първия ред на X към третия:

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

С други думи, всички елементи под главния диагонал сменят знаците си. Аналогично

$$L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

- Лесно се вижда, че

$$L := L_1 L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

И така, приведеният пример илюстрира факта, че за да разложим матрицата A във вида $A = LU$, трябва да извършим правия ход на метода на Гаус и така получаваме матрицата U.

Матрицата L е долна триъгълна матрица, чиито елементи a_{ij} под главния диагонал са коефициентите, с които умножаваме j-тия ред, за да го извадим от i-тия.

Да разгледаме още една задача, с която да затвърдим казаното.

Задача 4. Да се намери LU декомпозицията на матрицата

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}$$

Решение. Правият ход на алгоритъма ни дава

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 8 & 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix} =: U.$$

На първата стъпка от втория ред вадим първия, умножен по 2, а от третия вадим първия, умножен по -1. На втората стъпка от третия ред вадим втория, умножен по -1. Тогава матрицата L има вида

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}.$$

С други думи,

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}.$$

□

Можем лесно да модифицираме програмата, която предложихме за метода на Гаус, като в една нова матрица L, която в началото е единичната матрица и на всяка стъпка запазваме коефциентите, с които i-тият ред се умножава, за да се извади от следващите. Прилагаме примерна имплементация на Mathematica.

In[8]:=
```
LU[A_] :=
 (U = A;
  n = Length[A];
  L = Table[0, {n}, {n}];
  For[i = 1, i ≤ n, i++,
   L[[i, i]] = 1
  ];
  For[i = 1, i ≤ n, i++,
   For[j = i + 1, j ≤ n, j++,
    l = U[[j, i]] / U[[i, i]];
    L[[j, i]] = l;
    U[[j, i]] = 0;
    For[k = i + 1, k ≤ n, k++,
     U[[j, k]] = U[[j, k]] - l * U[[i, k]];
    ]
   ]
  ];
  {L, U}
 )
```

Както казахме, имайки LU декомпозицията на дадена матрица, можем лесно да решим системата $LUx = b$ на две стъпки.

Първо, решаваме системата $Ly = b$, която има долна триъгълна матрица. i-тото уравнение на тази система има вида

$$l_{i1}y_1 + \cdots + l_{ii}y_i = b_i,$$

откъдето получаваме

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij}y_j}{l_{ii}}, \quad i = 1, \ldots, n.$$

След това решаваме системата $Ux = y$, която има горна триъгълна матрица. Имаме

$$u_{ii}x_i + \cdots + u_{in}x_n = y_i$$

и следователно

$$x_i = \frac{y_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}}, \quad i = n, \ldots, 1.$$

Прилагаме примерна имплементация на обратния ход на алгоритъма.

```
In[1]:= LUSolve[L_, U_, b_] := (
  n = Length[b];
  y = Table[0, {i, n}] ;
  For[i = 1, i ≤ n, i++,
   y[[i]] = (b[[i]] - Sum[L[[i, j]] y[[j]], {j, 1, i - 1}])/L[[i, i]]
  ];
  x = Table[0, {i, n}] ;
  For[i = n, i ≥ 1, i--,
   x[[i]] = (y[[i]] - Sum[U[[i, j]] x[[j]], {j, i + 1, n}])/U[[i, i]]
  ];
  x
 )
```

2.2.2 Кога е полезно използването на LU-декомпозицията?

Както казахме, ако сме разложили матрицата A във вида $A = LU$, системата $Ax = b$ може да се реши със сложност $O(n^2)$. Самото разлагане на матрицата обаче става по метода на Гаус. С други думи, ако решим системата директно по метода на Гаус, ще направим дори по-малко операции, защото иначе след метода на Гаус (за да разложим A) ще трябва да решим две системи с триъгълна матрица.

Предимството на това разлагане обаче идва, ако трябва да решим много системи с една и съща матрица. Ако искаме да решим n системи по метода на Гаус, това означава да направим $n \times O(n^3) = O(n^4)$ операции, докато, ако първо разложим матрицата (за $O(n^3)$ операции) и после решим n-те системи за $n \times O(n^2) = O(n^3)$ операции, в крайна сметка ще сме направили $O(n^3)$ операции.

Един много важен случай, в който това се налага, е намирането на обратна матрица.

2.2.3 Намиране на обратна матрица

Нека е дадена матрицата A. Нейната обратна е такава матрица, че

$$A \begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Тогава стълбовете на матрицата A^{-1} са решенията на системите

$$A\overline{x}_i = \overline{e}_i,\ i = \overline{1, n}$$

където e_i е i-тият единичен вектор и за решаването им е удобно матрицата A да бъде разложена във вида $A = LU$. Нека разгледаме примерна функция, намираща обратната матрица на дадена несингулярна квадратна матрица.

```
In[2]:= InverseMatrix[A_] := (
          {L, U} = LU[A];
          n = Length[A];
          AInverse = Table[0, {n}, {n}];
          For[k = 1, k ≤ n, k++,
           b = Table[If[l == k, 1, 0], {l, 1, n}];
           AInverse[[k]] = LUSolve[L, U, b];
          ];
          Transpose[AInverse]
         )
```

2.3 Числени методи за системи със специална структура

Често на практика матриците, които се получават при решаването на дадена реална задача, не са произволни, а имат някаква специална структура, например:

- симетрични матрици – $A = A^T$;
- тридиагонални матрици – матрици, които имат ненулеви елементи само по главния диагонал и диагоналите под и над него;
- разредени матрици – матрици с много нулеви елементи.

Когато матрицата на дадена система има специална структура, тя може да се използва, за да получим решението на системата по-лесно (с по-малка времева сложност).

Ще дадем някои важни приложения, които водят до такива матрици.

Метод на най-малките квадрати

Един от най-често използваните на практика числени методи е методът на най-малките квадрати. Нека са дадени точките $(x_1, y_1), \ldots, (x_s, y_s)$. Търсим полином от степен, ненадминаваща n, който да е възможно "най-близо" до точките. Можем да формулираме задачата и по следния начин. Искаме вектора от грешките, които се получават във всяка от дадените точки, да е възможно "най-малък", т.е. да има най-малка Евклидова дължина. Векторът на грешките има вида

$$\overline{r} := \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_s \end{bmatrix} = \begin{bmatrix} p(x_1) \\ p(x_2) \\ \vdots \\ p(x_s) \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{bmatrix}.$$

Нека запишем полинома $p(x)$ във вида

$$p(x) = (1, x, \ldots, x^n)(a_0, a_1, \ldots, a_n)^T,$$

където $a_0, a_1, \ldots, a_n$ са коефициентите на полинома. Тогава получаваме

$$\overline{r} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_s & \cdots & x_s^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{bmatrix} =: X\overline{a} - \overline{y}. \tag{2.8}$$

Може да се покаже, че въпросът за минимизирането на $\|X\overline{a} - \overline{y}\|_2$ е еквивалентен на този за решаването на системата

$$X^T X\overline{a} = X^T\overline{y}. \tag{2.9}$$

Да обърнем внимание, че системата има симетрична и положително определена матрица (от линейната алгебра знаем, че $X^T X$ има именно такава структура).

Числено решаване на обикновени диференциални уравнения от втори ред.

Много физически процеси се описват с диференциални уравнения от втори ред:

$$pu''(x) + qu'(x) + ru(x) = f(x), \quad x \in (0,1). \tag{2.10}$$

Единият основен подход е следният. Разделяме интервал на подинтервали, въвеждайки мрежата от точки (на разстояние h една от друга):

$$\omega_h := \{x_i = ih, i = \overline{0,n}, n = 1/h\}.$$

Ще апроксимираме диференциалното уравнение ('refDE) в i-тата точка, като използваме следните формули за числено диференциране:

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + O(h^2),$$
$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2).$$

Нека означим с y_i апроксимацията на решението в точката x_i. Тогава диференциалното уравнение се апроксимира с алгебричното уравнение

$$p \cdot \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + q\frac{y_{i+1} - y_{i-1}}{2h} + ry_i = f(x_i).$$

Написвайки по едно уравнение като горното за всяка точка, получаваме система от линейни уравнения, която очевидно има тридиагонална матрица (т.е. в i-тото уравнение участват само y_{i-1}, y_i, y_{i+1}.

Да обърнем внимание още, че системата има толкова уравнения, колкото са възлите, с които сме разделили интервала. За да получим висока точност, може да се наложи интервала (или, изобщо, областта, в която се решава задачата и която може да бъде двумерна или тримерна) да се раздели на много подинтервали, т.е. **се налага решаването на система с много висока размерност**.

2.3.1 Метод на Холецки за разлагане на симетрични и положително определени матрици

В случая, когато матрицата A е симетрична и положително определена, тя може да бъде разложена във вида

$$A = LL^T,$$

където L е долна триъгълна матрица, т.е. има вида

$$L = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix}.$$

За това разлагане последователно се попълват първият, вторият и т.н. стълб, започвайки от диагоналния елемент:

$$\begin{aligned} for\ k = & \overline{1,n}: \\ & l_{kk} = \sqrt{a_{kk} - l_{k1}^2 - \cdots - l_{k,k-1}^2}, \\ & l_{jk} = \frac{a_{kj} - \sum_{i=1}^{k-1} l_{ki} l_{ji}}{l_{kk}}, \quad j = \overline{k+1,n}. \end{aligned}$$

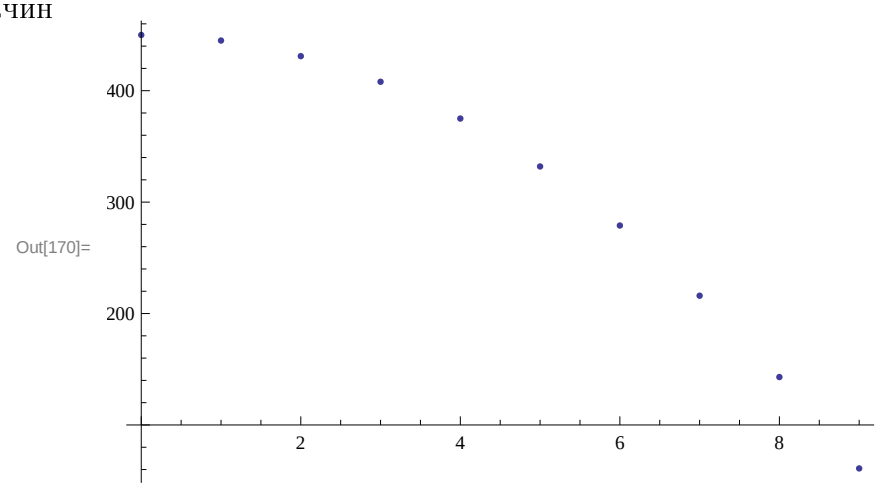
За извеждането на горните формули виж учебника. Ще покажем имплементацията на метода на базата на една конкретна задача.

Задача 5. Тяло е пуснато от височина $450m$. Неговата височина е измервана през интервали от 1 сек. Данните са систематизирани в следната таблица:

t, sec	0	1	2	3	4	5	6	7	8	9
h, m	450	445	431	408	375	332	279	216	143	61

Да се намери подходяща функция, описваща процеса.

Решение. Нека първо да изобразим точките. Графиката изглежда по следния начин



Точките изглежда да лежат върху парабола. Затова ще търсим функцията, моделираща процеса, във вида $f(x) = a_0 + a_1 x + a_2 x^2$. За да определим по метода на най-малките квадрати коефициентите на полинома, вземайки предвид (2.8) и (2.9), трябва да решим системата

$$X^T X \overline{x} = X^T \overline{y},$$

където

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 6 & 36 \\ 1 & 7 & 49 \\ 1 & 8 & 64 \\ 1 & 9 & 81 \end{bmatrix}, \ \overline{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \ \overline{y} = \begin{bmatrix} 450 \\ 445 \\ 431 \\ 408 \\ 375 \\ 332 \\ 279 \\ 216 \\ 143 \\ 61 \end{bmatrix}.$$

Нека означим $A := X^T X, \overline{b} := X^T \overline{y}$. Тогава системата, която трябва да решим, е $A\overline{x} = \overline{b}$. Първо, ще разложим матрицата A, която е симетрична и положително определена по метода на Холецки:

```
(*Input data*)
x = Range[0, 9];
X = Table[{1, x[[i]], x[[i]]^2}, {i, 1, 10}];
y = {{450, 445, 431, 408, 375, 332, 279, 216, 143, 61}} // Transpose;
A = Transpose[X].X;
b = Transpose[X].y;
n = Length[A];
(*Choletsky decomposition of B*)
L = Table[0, {i, n}, {j, n}];
For[k = 1, k ≤ n, k++, (*Iterate over the columns*)
 (*Find the diagonal element*)
 L[[k, k]] = Sqrt[A[[k, k]] - Sum[L[[k, p]]^2, {p, 1, k - 1}]];
 For[j = k + 1, j ≤ n, j++, (*Find all the elements below the diagonal*)
  L[[j, k]] = (A[[k, j]] - Sum[L[[k, i]] L[[j, i]], {i, 1, k - 1}])/L[[k, k]]
 ]
]
```

В резултат от изпълнението на горния код получаваме, че матрицата L има вида

$$L = \begin{bmatrix} \sqrt{10} & 0 & 0 \\ 9\sqrt{\frac{5}{2}} & \sqrt{\frac{165}{2}} & 0 \\ 57\sqrt{\frac{5}{2}} & 9\sqrt{\frac{165}{2}} & 4\sqrt{33} \end{bmatrix}.$$

Остава да решим системата $LL^T\overline{a} = \overline{b}$ на две стъпки, както обяснихме в параграфа, посветен на LU-декомпозицията.

1. Намираме $\overline{z} := L^T\overline{a}$, като решим системата $L\overline{z} = \overline{b}$. Тя има вида

$$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

и следователно от i-тото уравнение

$$l_{i1}z_1 + l_{i2}z_2 + \cdots + l_{ii}z_i = b_i$$

изразяваме

$$z_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij}z_j}{l_{ii}}.$$

Последователно намираме $x_1, x_2, \ldots, x_n$:

```
z = Table[0, {i, n}];
For[i = 1, i ≤ n, i++,
 z[[i]] = (b[[i]] - Sum[L[[i, j]] z[[j]], {j, 1, i - 1}])/L[[i, i]] // N
]
```

2. Решаваме системата $L^T\overline{a} = \overline{z}$ и намираме неизвестните $\overline{a}$ по аналогичната на първата стъпка формула

$$a_i = \frac{z_i - \sum_{j=i+1}^{n} l_{ij}^T a_j}{l_{ii}^T},$$

като започваме от a_n:

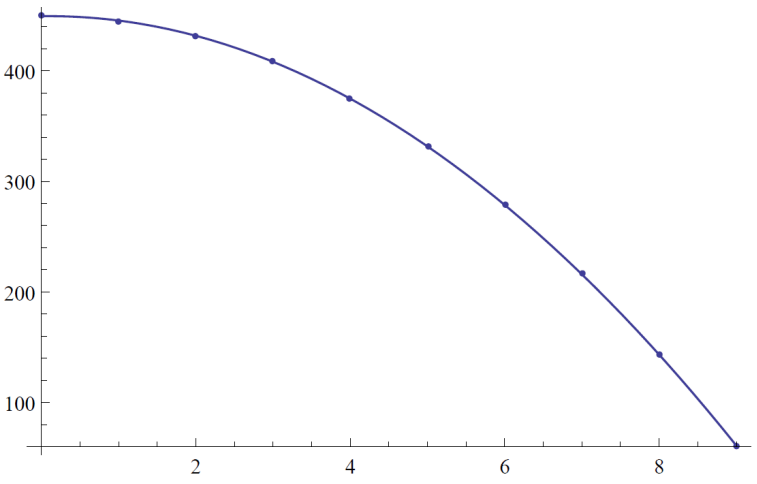
```
a = Table[0, {i, n}];
Ltr = Transpose[L];
For[i = n, i ≥ 1, i--,
 a[[i]] = (z[[i]] - Sum[Ltr[[i, j]] a[[j]], {j, i + 1, n}])/Ltr[[i, i]] // N
]
```

Окончателно получихме $\overline{a} = (449.364, 0.962121, -4.90152)^T$.

Нека построим графиките на полинома $P(x) = (1, x, x^2).\overline{a}$ и дадените точки в една координатна система, за да визуализираме резултата.

```
P[x_] := a[[1, 1]] + Sum[a[[i, 1]] x^(i-1), {i, 2, n}]
plot1 = Plot[P[x], {x, 0, 9}];
plot2 = ListPlot[Table[{x[[i]], y[[i, 1]]}, {i, 1, Length[x]}]];
Show[plot1, plot2]
```

Out[138]=



□

2.3.2 Метод на дясната прогонка за решаване на системи с тридиагонална матрица

Важен клас системи са тези с тридиагонална матрица, т.е. тези, които имат ненулеви елементи само по главния диагонал, под него и над него:

$$\begin{bmatrix} * & * & & & & \\ * & * & * & & & \\ & * & * & * & & \\ & \ldots & \ldots & \ldots & \ldots & \\ & & & * & * & * \\ & & & & * & * \end{bmatrix}.$$

Нека запишем в общ вид системата, зададена с тридиагонална матрица, по следния начин:

$$\begin{array}{rcrcrcll} & - & C_1x_1 & + & B_1x_2 & = & F_1, & \\ A_ix_{i-1} & - & C_ix_i & + & B_ix_{i+1} & = & F_i, & i = 2, \ldots, n-1 \\ A_nx_{n-1} & - & C_nx_n & & & = & F_n. & \end{array}$$

Тогава от първото уравнение можем да изразим първото неизвестно чрез второто:

$$x_1 = \frac{B_1}{C_1}x_2 - \frac{F_1}{C_1} =: \alpha_2x_2 + \beta_2.$$

Така, ако сме изразили $x_{i-1} = \alpha_ix_i + \beta_i$, можем да го заместим в i-тото уравнение и оттам да изразим x_i чрез x_{i+1}:

$$x_i = \alpha_{i+1}x_{i+1} + \beta_{i+1}. \tag{2.11}$$

Коефициентите α_i и β_i се определят от формулите

$$\alpha_{i+1} = \frac{B_i}{C_i - A_i\alpha_i},\ \beta_{i+1} = \frac{A_i\beta_i - F_i}{C_i - A_i\alpha_i},\ i = 1, \ldots, n-1.$$

Накрая, заместваики $x_{n-1} = \alpha_nx_n + \beta_n$ в последното уравнение, получаваме

$$x_n = \frac{A_n\beta_n - F_n}{C_n - A_n\alpha_n}.$$

Тогава от (2.11) можем да изразим последователно $x_{n-1}, x_{n-2}, \ldots, x_1$.

Задача 6. Като се използва методът на дясната прогонка, да се реши системата

$$\begin{array}{rcrcrcrcl} 3x_1 & - & x_2 & & & & & = & 3, \\ -x_1 & + & 3x_2 & - & x_3 & & & = & 0, \\ & & -x_2 & + & 3x_3 & - & x_4 & = & 0, \\ & & & & -x_3 & + & 3x_4 & = & 0. \end{array}$$

Решение. От първото уравнение изразяваме

$$x_1 = \frac{1}{3}x_2 + 1.$$

Заместваики горното във второто уравнение, получаваме

$$-\frac{1}{3}x_2 - 1 + 3x_2 - x_3 = 0 \Longrightarrow x_2 = \frac{3}{8}x_3 + \frac{3}{8}.$$

□

Продължавайки аналогично, получаваме

$$-\frac{3}{8}x_3 - \frac{3}{8} + 3x_3 - x_4 = 0 \Longrightarrow x_3 = \frac{8}{21}x_4 + \frac{1}{7}$$

и окончателно

$$-\frac{8}{21}x_4 - \frac{1}{7} + 3x_4 = 20 \Longrightarrow x_4 = \frac{423}{55}.$$

Сега, връщайки се назад, имаме

$$x_3 = \frac{8}{21}x_4 + \frac{1}{7} = \frac{169}{55},\ x_2 = \frac{3}{8}x_3 + \frac{3}{8} = \frac{84}{55},\ x_1 = \frac{1}{3}x_2 + 1 = \frac{83}{55}.$$

Прилагаме примерна имплементация на метода:

```
In[41]:= A = {{3, -1, 0, 0}, {-1, 3, -1, 0}, {0, -1, 3, -1}, {0, 0, -1, 3}} // N;
         b = {3, 0, 0, 20} // N;
         (*Find the coefficients αi, βi*)
         n = Length[A];
         α = Table[0, {i, n}];
         β = Table[0, {i, n}];
         x = Table[0, {i, n}];
         α[[2]] = A[[1, 2]] / (-A[[1, 1]]); (*C1=-A[[1,1]]*)
         β[[2]] = -b[[1]] / (-A[[1, 1]]);
         For[i = 2, i < n, i++,
          α[[i + 1]] = A[[i, i + 1]] / (-A[[i, i]] - A[[i, i - 1]] α[[i]]); (*Ci=-A[[i,i]]*)
          β[[i + 1]] = (A[[i, i - 1]] β[[i]] - b[[i]]) / (-A[[i, i]] - A[[i, i - 1]] α[[i]]);
         ]
         x[[n]] = (A[[n, n - 1]] β[[n]] - b[[n]]) / (-A[[n, n]] - A[[n, n - 1]] α[[n]]);
         For[i = n - 1, i ≥ 1, i--,
          x[[i]] = α[[i + 1]] x[[i + 1]] + β[[i + 1]]
         ]
         x
Out[51]= {1.50909, 1.52727, 3.07273, 7.69091}
```

2.4 "Case Study". Изследване на стационарното състояние на физически[1] процеси.

Много физически процеси, изменящи се във времето, се описват с диференциални уравнения от вида

$$\frac{du}{dt} = f(t, u(t)). \tag{2.12}$$

[1] "Физически" в смисъл на реални, съществуващи, а не задължително идващи от физиката.

Това, разбира се, е в сила, когато функцията u не зависи от точката в пространството, а е функция само на времето t. В противен случай моделът ще бъде частно диференциално уравнение. Да обърнем внимание, че (2.12) може да бъде и система от диференциални уравнения, ако $u : \mathbb{R} \to \mathbb{R}^n$ е векторна функция.

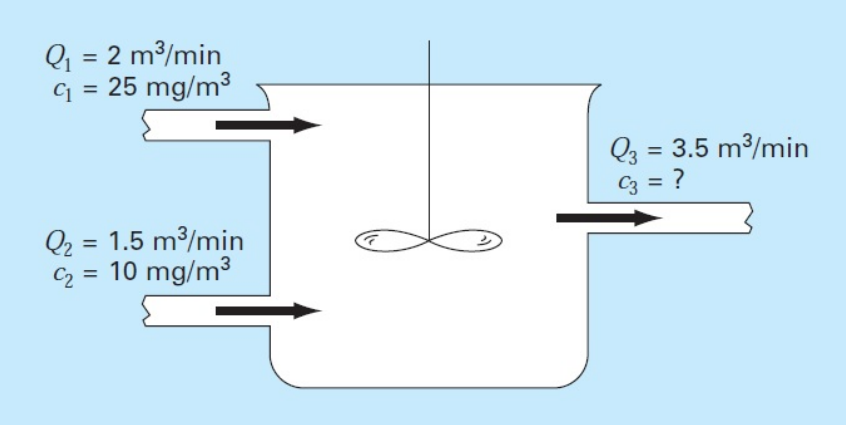
Стационарното състояние на една система е състояние, което не се изменя във времето, т.е. състояние, за което $du/dt = 0$. Тогава (2.12) се свежда до система алгебрични уравнения от вида

$$f(t, u) = 0.$$

Ще разгледаме две примерни системи в стационарно състояние.

Анализ на стационарното състояние на система реактори

Нека разгледаме следния реактор:

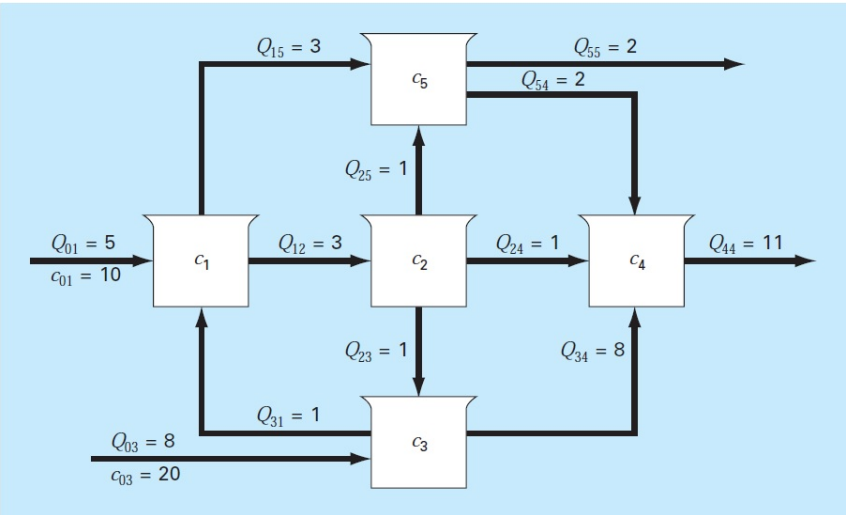


За да бъде той в стационарно състояние, е необходимо входът в реактора точно да се компенсира от изхода. С други думи, трябва да е изпълнено

$$Q_1 c_1 + Q_2 c_2 = Q_3 c_3,$$
$$50 + 15 = 3.5 c_3,$$

т.е. стационарната концентрация е решение на едно алгебрично уравнение.

Нека сега разгледаме една многокомпонентна система:



Ясно е, че **когато системата е многокомпонентна, ще получим система линейни алгебрични уравнения**, тъй като в този случай за всеки един реактор в системата ще "отговаря" по едно уравнение Тя има вида (i-тото уравнение отговаря на баланса на масите в i-тия реактор, $i = 1, \ldots, 5$)

$$\begin{aligned}
&Q_{01}c_{01} + Q_{31}c_3 = (Q_{12} + Q_{15})c_1,\\
&Q_{12}c_1 = (Q_{25} + Q_{24} + Q_{23})c_2,\\
&Q_{03}c_{03} + Q_{23}c_2 = (Q_{31} + Q_{34})c_3,\\
&Q_{24}c_2 + Q_{34}c_3 + Q_{54}c_5 = Q_{44}c_4,\\
&Q_{15}c_1 + Q_{25}c_2 = Q_{54}c_5 + Q_{55}c_5.
\end{aligned}$$

Замествайки данните от фигурата, получаваме

$$\begin{aligned}
&6c_1 - c_3 = 50,\\
&-3c_1 + 3c_2 = 0,\\
&-c_2 + 9c_3 = 160,\\
&-c_2 - 8c_3 + 11c_4 - 2c_5 = 0,\\
&-3c_1 - c_2 + 4c_5 = 0.
\end{aligned}$$

Записана във векторно-матрична форма, системата има вида $A\overline{c} = \overline{b}$, където

$$A = \begin{bmatrix} 6 & 0 & -1 & 0 & 0\\ -3 & 3 & 0 & 0 & 0\\ 0 & -1 & 9 & 0 & 0\\ 0 & -1 & -8 & 11 & -2\\ -3 & -1 & 0 & 0 & 4 \end{bmatrix}, \overline{c} = \begin{bmatrix} c_1\\ c_2\\ c_3\\ c_4\\ c_5 \end{bmatrix}, \overline{b} = \begin{bmatrix} 50\\ 0\\ 160\\ 0\\ 0 \end{bmatrix}.$$

Ще използваме стандартния подход за решаването на система с не много голяма размерност, чиято матрица е в общ вид, а именно метода на Гаус с частичен избор на главния елемент.

Като входни данни подаваме

```
A = {{6, 0, -1, 0, 0}, {-3, 3, 0, 0, 0},
     {0, -1, 9, 0, 0}, {0, -1, -8, 11, -2}, {-3, -1, 0, 0, 4}} // N;
b = {50, 0, 160, 0, 0} // N;
```

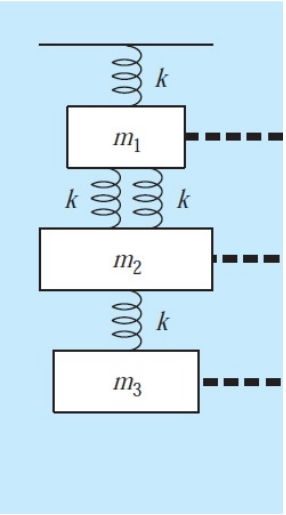
Резулатът от изпълнението на програмата за метода на Гаус с частичен избор на главния елемент е

```
Out[7]= {11.5094, 11.5094, 19.0566, 16.9983, 11.5094}
```

С други думи, концентрациите на биореакторите, за които системата е в покой, са приблизително $c_1 = 11.5$, $c_2 = 11.5$, $c_3 = 19.1$, $c_4 = 17$, $c_5 = 11.5$.

Стационарно състояние на система от пружини и маси.

Системи от пружини и маси играят важна роля като идеализация в редица приложения в механиката и инженерното дело. Нека разгледаме следната примерна система:



Нека телата имат маси съответно $m_1 = 2kg$, $m_2 = 3kg$, $m_3 = 2.5kg$, а еластичните сили в пружините се описват със закона на Хук $F = kx$, където $k = 10kg/s^2$, а x е дължината на пружината. Нека в покой означим разстоянията на трите тела до основата, на която са закачени с x_1, x_2, x_3.

За да бъде системата в покой, върху всяко от телата силите, действащи във вертикално направление, трябва да се уравновесяват. За първото тяло имаме една пружина, действаща на тялото нагоре със сила $F_U = kx_1$. Надолу действат силата на тежестта $G = m_1 g$ и две пружини със сили $F_D = k(x_2 - x_1)$. Тогава за първото тяло трябва да е изпълнено

$$kx_1 = m_1 g + 2k(x_2 - x_1).$$

Аналогично, съставяйки уравнения и за другите две тела, получаваме системата

$$\begin{aligned} 3kx_1 - 2kx_2 &= m_1 g, \\ -2kx_1 + 3kx_2 - kx_3 &= m_2 g, \\ -kx_2 + kx_3 &= m_3 g. \end{aligned}$$

Замествайки в системата със съответните стойности на величините и записвайки системата във векторно-матрична форма, получаваме окончателно $A\overline{x} = \overline{b}$, където

$$A = \begin{bmatrix} 30 & -20 & \\ -20 & 30 & -10 \\ & -10 & 10 \end{bmatrix}, \overline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \overline{b} = \begin{bmatrix} 19.6 \\ 29.4 \\ 24.5 \end{bmatrix}.$$

Системата има матрица, която е симетрична и положително определена. **Симетрията в матрицата е естествено отражение на симетрията във физическия процес** – еластичните сили, които действат на първото тяло надолу, действат на второто в обратна посока. Много процеси се характеризират с някаква симетрия, която по аналогичен начин води до симетрия в разглежданите математически задачи, в частност линейни алгебрични системи.

Ще използваме симетрията и ще решим задачата, като първо разложим матрицата по метода на Холецки.

```
(*Input data*)
A = {{30, -20, 0}, {-20, 30, -10}, {0, -10, 10}} // N;
b = {19.6, 29.4, 24.5} // N;
n = Length[A];
(*Choletsky decomposition of A*)
L = Table[0, {i, n}, {j, n}];
For[k = 1, k ≤ n, k++, (*Iterate over the columns*)
 (*Find the diagonal element*)
 L[[k, k]] = √(A[[k, k]] - Sum[L[[k, p]]^2, {p, 1, k - 1}]) ;
 For[j = k + 1, j ≤ n, j++, (*Find all the elements below the diagonal*)
  L[[j, k]] = (A[[k, j]] - Sum[L[[k, i]] L[[j, i]], {i, 1, k - 1}])/L[[k, k]]
 ]
]
z = Table[0, {i, n}];
For[i = 1, i ≤ n, i++,
 z[[i]] = (b[[i]] - Sum[L[[i, j]] z[[j]], {j, 1, i - 1}])/L[[i, i]]
]
a = Table[0, {i, n}];
Ltr = Transpose[L];
For[i = n, i ≥ 1, i--,
 a[[i]] = (z[[i]] - Sum[Ltr[[i, j]] a[[j]], {j, i + 1, n}])/Ltr[[i, i]]
]
a
```

Резултатът от изпълнението на горната програма е следният.

```
Out[32]= {7.35, 10.045, 12.495}
```

Дотук трябва да знаете

- Как се представят числата в компютъра и как това води до въвеждането на грешка при числените пресмятания?
- Къде в практиката възникват линейни алгебрични системи?
  - При изследването на стационарното състояние на редица инженерни и приложни задачи.
  - При решаването на важни математически задачи, например апроксимации по метода на най-малките квадрати, интерполация и др.
  - При числеността решаване на диференциални уравнения, т.е. при математическото изследване на нестационарни и/или нехомогенни процеси.

  - При много практически задачи възникващите системи имат специална структура.

- Директни методи и кога те са приложими.
  - Методът на Гаус с частичен избор на главния елемент е стандартният избор на числен метод за решаване на системи с не много голяма размерност, чиито матрици нямат специална структура.
    * Защо методът на Гаус се използва с избор на главния елемент, а не в класическия си вид?
    * Защо методът на Гаус не е приложим за системи с голяма размерност?
  - LU-декомпозиция и нейните приложения (за решаване на много системи с еднакви десни страни, в частност за намиране на обратни матрици).
  - Използване на специалната структура на една матрица за конструиране на по-ефективни методи. Решаване на системи със симетрична и положително определена матрица и с тридиагонална матрица.

Глава 3

Итерационни методи за решаване на системи линейни алгебрични уравнения

Основен проблем на директните методи, които разгледахме за решаване на системата

$$A\overline{x} = \overline{b},$$

е тяхното бързодействие. За големи системи уравнения тяхното прилагане е нецелесъобразно, а често пъти – и невъзможно. На практика, от друга страна, възникват системи с десетки хиляди и дори милиони уравнения. За системи с голяма размерност обикновено се използват т.нар. итерационни методи. Идеята при тях е следната. Започвайки с дадено начално приближение $\overline{x}_0$, построяваме редицата $\overline{x}_0, \overline{x}_1, \overline{x}_2, \ldots$ от последователни приближения, която да бъде сходяща към решението на системата. Тогава очевидно е важен въпросът за **сходимостта на съответните числени методи**.

3.1 Метод на простата итерация

Нека е дадена системата

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \multicolumn{9}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}$$

Ако от първото уравнение изразим x_1, от второто – x_2 и т.н. получаваме еквивалентната система

$$x_i = -\sum_{j=1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \quad i = \overline{1, n}. \tag{3.1}$$

Ако имаме дадено приближение, заместваики с него в дясната страна на (3.1) и използвайки съответните равенства, ще получим нови стойности на $x_1, \ldots, x_n$.

И така, формулите за построяване на редицата от последователни приближения имат вида

$$x_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}}, \quad i = \overline{1,n}. \tag{3.2}$$

Задача 7. Да се приведе системата

$$\begin{array}{rcrcrcl} 4x_1 & - & x_2 & + & x_3 & = & 12, \\ -x_1 & + & 4x_2 & - & 2x_3 & = & -1, \\ x_1 & - & 2x_2 & + & 4x_3 & = & 5 \end{array} \tag{3.3}$$

във вид, удобен за прилагане на метода на простата итерация. Да се направят две итерации, започвайки с начално приближение $\overline{x}_0 = (0,0,0)^T$.

Решение. От първото уравнение изразяваме x_1 чрез другите две неизвестни, от второто – x_2, а от третото – x_3. Получаваме еквивалентната система

$$\begin{array}{rcrcrcl} x_1 & = & \frac{1}{4}x_2 & - & \frac{1}{4}x_3 & + & 3, \\ x_2 & = & \frac{1}{4}x_1 & + & \frac{1}{2}x_3 & - & \frac{1}{4}, \\ x_3 & = & -\frac{1}{4}x_1 & + & \frac{1}{2}x_2 & + & \frac{5}{4}. \end{array}$$

Тогава итерационният процес се построява по формулите

$$\begin{array}{rcrcrcl} x_1^{(k+1)} & = & \frac{1}{4}x_2^{(k)} & - & \frac{1}{4}x_3^{(k)} & + & 3, \\ x_2^{(k+1)} & = & \frac{1}{4}x_1^{(k)} & + & \frac{1}{2}x_3^{(k)} & - & \frac{1}{4}, \\ x_3^{(k+1)} & = & -\frac{1}{4}x_1^{(k)} & + & \frac{1}{2}x_2^{(k)} & + & \frac{5}{4}. \end{array} \tag{3.4}$$

Тогава, започвайки от началното приближение $\overline{x}_0 = (0,0,0)^T$, получаваме

$$\begin{aligned} x_1^{(1)} &= \frac{1}{4} \times 0 - \frac{1}{4} \times 0 + 3 = 3, \\ x_2^{(1)} &= \frac{1}{4} \times 0 + \frac{1}{2} \times 0 - \frac{1}{4} = -\frac{1}{4}, \\ x_3^{(1)} &= -\frac{1}{4} \times 0 + \frac{1}{2} \times 0 + \frac{5}{4} = \frac{5}{4}. \end{aligned}$$

За второто приближение $\overline{x}_2$ получаваме

$$\begin{aligned} x_1^{(2)} &= \frac{1}{4} \times \left(-\frac{1}{4}\right) - \frac{1}{4} \times \frac{5}{4} + 3 = \frac{21}{8}, \\ x_2^{(2)} &= \frac{1}{4} \times 3 + \frac{1}{2} \times \frac{5}{4} - \frac{1}{4} = \frac{9}{8}, \\ x_3^{(2)} &= -\frac{1}{4} \times 3 + \frac{1}{2} \times \left(-\frac{1}{4}\right) + \frac{5}{4} = \frac{3}{8}. \end{aligned}$$

□

Разбира се, възниква въпросът за сходимостта на така построения итерационен процес. Отговор на този въпрос дава следното твърдение.

Твърдение 1. *Итерационният процес*

$$\overline{x}_{k+1} = B\overline{x}_k + \overline{d} \tag{3.5}$$

е сходящ за произволно начално приближение тогава и само тогава, когато всички собствени стойности на матрицата B са по модул по малки от 1.

Задача 8. Да се изследва сходимостта на метода на простата итерация за системата (3.3).

Решение. Първо, да запишем итерационния процес във вида (3.5). Използвайки (3.4), получаваме

$$\overline{x}_{k+1} = \underbrace{\begin{bmatrix} 0 & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{2} \\ -\frac{1}{4} & \frac{1}{2} & 0 \end{bmatrix}}_{B} \overline{x}_k + \underbrace{\begin{bmatrix} 3 \\ -\frac{1}{4} \\ \frac{5}{4} \end{bmatrix}}_{\overline{d}}$$

Собствените стойности на матрицата B са решенията на уравнението

$$\det(B - \lambda I) = \begin{vmatrix} -\lambda & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & -\lambda & \frac{1}{2} \\ -\frac{1}{4} & \frac{1}{2} & -\lambda \end{vmatrix} = 0,$$

т.е. са $1/2$ и $(-1\pm\sqrt{3})/4$ и са по модул по-малки от 1. Следователно за произволно начално приближение методът на простата итерация е сходящ за дадената система. □

Задача 9. Да се намерят стойностите на реалния параметър a, за които методът на простата итерация е сходящ при всяко начално приближение за системата

$$\begin{aligned} ax + y &= 1, \\ x + y + z &= 1, \\ y + az &= 1. \end{aligned}$$

Решение. Ясно е, че при $a = 0$ системата е неопределена. Нека $a \neq 0$. Отново ще запишем итерационния процес във вида (3.5). Имаме

$$\begin{aligned} x^{(k+1)} &= -\tfrac{1}{a}y^{(k)} + \tfrac{1}{a}, \\ y^{(k+1)} &= -x^{(k)} - z^{(k)} + 1, \\ z^{(k+1)} &= -\tfrac{1}{a}y^{(k)} + \tfrac{1}{a}, \end{aligned}$$

т.е. матрицата B има вида

$$B = \begin{bmatrix} 0 & -\frac{1}{a} & 0 \\ -1 & 0 & -1 \\ 0 & -\frac{1}{a} & 0 \end{bmatrix}.$$

Нейният характеристичен полином е

$$\det(B - \lambda I) = \begin{vmatrix} -\lambda & -\frac{1}{a} & 0 \\ -1 & -\lambda & -1 \\ 0 & -\frac{1}{a} & -\lambda \end{vmatrix} = -\lambda\left(\lambda^2 - \frac{2}{a}\right).$$

Следователно собствените стойности на B са $\lambda_1 = 0$, и $\lambda_{2,3} = \pm\sqrt{\frac{2}{a}}$.

- Първи случай: $a > 0$. Ясно е, че в този случай условието за сходимост е изпълнено т.с.т.к $a > 2$.
- Втори случай: $a < 0$. Имаме $|\pm\sqrt{2/a}| = |i\sqrt{2/-a}| < 1 \iff 2/-a < 1$, т.е. $a < -2$.

Окончателно получихме, че методът е сходящ при произволно начално приближение, точно когато $a \in (-\infty, -2) \cup (2, +\infty)$. □

Теоретично, точното решение "се достига" при $n \to \infty$. Разбира се, на практика ние не можем да направим безброй много итерации. Възниква необходимостта от **критерий, по който да спираме разглеждания итерационен процес**, когато сме достигнали решение, което е достатъчно близо (в някакъв смисъл) до точното решение.

Говорейки за "близо", ясно е, че този критерий трябва да е свързан с понятията норма и разстояние. Да припомним някои често използвани норми и породените от тях разстояния. Нека $x \in \mathbb{R}^n$. Тогава равномерната (максимум) норма и породеното от нея разстояние се дефинират с

$$\|\overline{x}\|_\infty = \max_{i=\overline{1,n}} |x_i|, \quad \rho(\overline{x}, \overline{y}) = \|\overline{x} - \overline{y}\|_\infty = \max_{i=\overline{1,n}} |x_i - y_i|.$$

Евклидовата норма и евклидовото разстояние са съответно

$$\|\overline{x}\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}, \quad \rho(\overline{x}, \overline{y}) = \|\overline{x} - \overline{y}\|_2 = \left\{ \sum_{i=1}^{n} (x_i - y_i)^2 \right\}^{1/2}.$$

Тогава можем да спираме итерационния процес, когато две последователни приближения са достатъчно близо едно до друго в смисъла на някое разстояние (обикновено се използва относително разстояние), т.е. когато

$$\varepsilon_r := \frac{\|\overline{x}^{(k+1)} - \overline{x}^{(k)}\|}{\|\overline{x}^{(k)}\|} < \varepsilon_{max},$$

където ε_{max} е отнапред зададена желана точност. Ако правенето на нови итерации не променя съществено резултата, можем да считаме, че сме стигнали с достатъчно добра точност до точното решение.

Вече сме готови да разгледаме имплементацията на метода в Mathematica:

```
A = {{4, -1, 1}, {-1, 4, -2}, {1, -2, 4}} // N;
b = {12, -1, 5} // N;
n = Length[A];
ϵ = 0.001;
maxIter = 100;
iter = 1;
xOld = Table[0, {i, n}];
xNew = Table[0, {i, n}];
For[i = 1, i ≤ n, i++, (*Compute the first approximation x1*)
  xNew[[i]] = 1/A[[i, i]] (-Sum[A[[i, j]] xOld[[j]], {j, 1, i - 1}] -
      Sum[A[[i, j]] xOld[[j]], {j, i + 1, n}]) + b[[i]]/A[[i, i]]
 ];
(*Compute the successive approximations
 until the desired accuracy is reached or
 the maximum number of iterations is reached*)
While[Norm[xNew - xOld] / Norm[xNew] ≥ ϵ && iter < maxIter,
 xOld = xNew;
 For[i = 1, i ≤ n, i++,
  xNew[[i]] = 1/A[[i, i]] (-Sum[A[[i, j]] xOld[[j]], {j, 1, i - 1}] -
      Sum[A[[i, j]] xOld[[j]], {j, i + 1, n}]) + b[[i]]/A[[i, i]]
 ];
 iter++
]
xNew
iter
```

Резултатът от изпълнението на горния код е

```
Out[11]= {3.00045, 0.999379, 1.00062}

Out[12]= 19
```

Желаната точност е постигната за 19 итерации. Намереното решение с точност до четвъртия знак след десетичната запетая е $\overline{x} = (3.0005, 0.9994, 1.0006)^T$. Ако сравним с точното решение, което е $\overline{\xi} = (3, 1, 1)^T$, виждаме, че получената точност действително отговаря на зададената (0.1%).

Задача 10. Да се изследва сходимостта на метода на простата итерация за системата

$$\begin{aligned} x_1 &- 5x_2 = -4, \\ 7x_1 &- x_2 = 6. \end{aligned}$$

на базата на Твърдение 3.5. Да се направят първите 20 итерации на метода на простата итерация върху тази система при начално приближение $\overline{x}_0 = (0, 0)^T$.

Решение. Проверката на условието оставяме за самостоятелна работа. Може да се покаже, че то не е изпълнено, т.е. не за всяко начално приближение методът

ще бъде сходящ. Обърнете внимание, че в този случай твърдението не ни дава отговор на въпроса дали съществуват начални приближения, за които методът е сходящ, и кои са те.

За началното приближение $\overline{x}_0 = (0,0)^T$, използвайки вече реализираната програма със съответните входни данни, получаваме

```
Out[59]= {-2.75855 × 10^15, -2.75855 × 10^15}
```

Резултатът показва, че методът е разходящ.

□

3.2 Метод на Зайдел

Методът на Зайдел е модификация на метода на простата итерация. При него на $k+1$-вата итерация приближението се намира по формулите

$$x_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}}, \quad i = \overline{1,n}. \tag{3.6}$$

С други думи, се използват и всички вече намерени на $k+1$-вата стъпка координати на вектора $\overline{x}_{k+1}$, т.е. "най-добрата" в дадения момент информация за съответните координати.

Имплементацията в Mathematica е следната:

```
A = {{4, -1, 1}, {-1, 4, -2}, {1, -2, 4}} // N;
b = {12, -1, 5} // N;
n = Length[A];
ϵ = 0.001;
maxIter = 100;
iter = 1;
xOld = Table[0, {i, n}];
xNew = Table[0, {i, n}];
For[i = 1, i ≤ n, i++,
  xNew[[i]] = 1/A[[i, i]] (-Sum[A[[i, j]] xOld[[j]], {j, 1, i - 1}] -
      Sum[A[[i, j]] xOld[[j]], {j, i + 1, n}]) + b[[i]]/A[[i, i]]
 ];
While[Norm[xNew - xOld] / Norm[xNew] ≥ ϵ && iter < maxIter,
 xOld = xNew;
 For[i = 1, i ≤ n, i++,
  xNew[[i]] = 1/A[[i, i]] (-Sum[A[[i, j]] xNew[[j]], {j, 1, i - 1}] -
      Sum[A[[i, j]] xOld[[j]], {j, i + 1, n}]) + b[[i]]/A[[i, i]]
 ];
 iter++
]
xNew
iter
```

Резултатът от изпълнението на горния код е следният:

```
Out[35]= {3.00015, 1.00017, 1.00005}

Out[36]= 7
```

Желаната точност е постигната за 7 итерации, което е по-бързо от метода на простата итерация.

Глава 4

Някои практически въпроси, свързани с прилагането на числените методи за решаване на линейни алгебрични системи

4.1 Сравнение на бързодействието на методите

Нека разгледаме системата $Ax = b$, където матрицата A е от ред n и има вида

$$A = \begin{bmatrix} 2.2 & 1 & & & & & \\ 1 & 2.2 & 1 & & & & \\ & 1 & 2.2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 2.2 & 1 & \\ & & & & 1 & 2.2 & \end{bmatrix},$$

а дясната страна е $b = (1, 1, \ldots, 1)^T$.

Матрицата A е избрана така, че да могат да бъдат приложени всички изучени методи, с цел да се направи сравнение на тяхното бързодействие за системи с различна размерност n. Времето в секунди, за което всеки от тях решава системата, е дадено в таблицата по-долу. Всички числени експерименти са направени в системата Wolfram Mathematica 8 на персонален компютър с процесор Intel Core i5-6200U, 2.4GHz.

n	Гаус (с част. избор) $2/3n^3 + O(n^2)$	Холецки $1/3n^3 + O(n^2)$	Прогонка $O(n)$	Якоби $O(n^2)$	Зайдел $O(n^2)$
50	0.14	0.062	<0.01	0.125 (51 ит.)	0.047 (17 ит.)
100	1.078	0.328	<0.01	0.766 (79 ит.)	0.14 (15 ит.)
200	8.422	2.328	<0.01	2.937 (80 ит.)	0.516 (14 ит.)
400	66.297	18.172	<0.01	11.985 (80 ит.)	1.734 (12 ит.)
800	613.594	147.25	0.015	47.238 (81 ит.)	6.234(11 ит.)
1600	над 1 час	1379.73	0.03	190.766 (81 ит.)	23.078 (10 ит.)

Както може да се очаква, методът на дясната прогонка е чувствително по-бърз от останалите, което го прави приложим за системи с много голяма размерност, имащи подходяща структура.

Останалите методи обаче имат сравнимо бързодействие за системи с малка размерност. За $n = 50, 100, 200$ методът на Холецки е дори по-бърз от метода на Якоби, който има сложност от по-нисък ред.

За системи с малка размерност директните методи са за предпочитане. Те имат сравнимо бързодействие, но са значително по-надеждни. **За системи с голяма размерност обаче виждаме, че итерационните методи имат съществено преимущество.** Това може да се обясни с факта, че броят итерации не се променя много при увеличаване размерността на системата.

4.2 Число на обусловеност. Априорни и апостериорни оценки на грешката.

Както видяхме, неустойчивостта на даден метод може да доведе до съществени грешки в крайния резултат. Причината за това е, че алгоритъмът "увеличава" грешките от закръгляване. Използването на друг, устойчив, метод ще доведе до получаването на верен резултат.

Съществува обаче и друга причина, поради която резултатите от числено решаване на дадена линейна система могат да бъдат лоши. Това е т.нар. обусловеност на задачата. Това понятие е свързано с грешката, до която грешката във входните данни може да доведе в резултата.

Ще изведемм оценка на грешката при решаването на една линейна алгебрична система, ако променим малко входните данни. Нека x е решението на оригиналната система, а $\tilde{x}$ е решението на системата с променени входни данни, т.е.

$$\begin{aligned}(A + \Delta A)\tilde{x} &= b,\\ Ax &= b.\end{aligned}$$

Вадейки второто уравнение от първото, за грешката получаваме

$$\begin{aligned}A(\tilde{x} - x) &= -\Delta A\tilde{x},\\ \tilde{x} - x &= -A^{-1}\Delta A\tilde{x}.\end{aligned}$$

Тогава за абсолютната грешка, измерена в дадена норма, е в сила

$$\varepsilon_a := \|\tilde{x} - x\| = \|A^{-1}A\tilde{x}\| \leq \|A^{-1}\|\|A\|\|\tilde{x}\|.$$

За да изразим относителната грешка, делим двете страни на $\|\tilde{x}\|$ и получаваме

$$\varepsilon_r := \frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq \underbrace{\|A^{-1}\|\|A\|}_{cond(A)} \frac{\|\Delta A\|}{\|A\|}. \tag{4.1}$$

И така, виждаме от горната оценка, че дори незначителна промяна във входните данни, $\|\Delta A\|/\|A\|$, може да доведе до голяма грешка в резултата, ако $cond(A)$ е голямо. Числото $cond(A)$ се нарича число на обусловеност за матрицата A.

Да обърнем внимание, че обусловеността е свойство на решаваната задача, а не на метода, с който я решаваме.

Да пресметнем числото на обусловеност за матрицата на Хилберт от ред 20:

```
H = HilbertMatrix[20];
N[Norm[Inverse[H], 2] Norm[H, 2]]
```

Out[10]= 2.45216×10^{28}

Вземайки предвид последния резултат и оценката (4.1), можем да очакваме, че работейки в компютърна аритметика, решаването на система с матрица на Хилберт, ще бъде лошо обусловена задача и получените грешки ще са съществени. За да илюстрираме този факт, нека първо решим системата $Hx = b$ за $b = (1, 1, \ldots, 1)^T$ символно, използвайки вградената функция LinearSolve в Mathematica:

```
In[18]:= b = Table[1, {20}];
         LinearSolve[H, b]
```

```
Out[19]= {-20, 7980, -790 020, 34 321 980, -823 727 520, 12 355 912 800,
  -124 932 007 200, 894 921 112 800, -4 698 335 842 200, 18 503 322 637 800,
  -55 509 967 913 400, 127 994 058 246 600, -227 544 992 438 400,
  311 023 037 001 600, -323 717 854 838 400, 251 780 553 763 200,
  -141 626 561 491 800, 54 396 360 988 200, -12 759 640 231 800, 1 378 465 288 200}
```

Да обърнем внимание, че тъй като не използваме числа с плаваща точка във входните данни, а работим символно, Mathematica намира точното решение на системата.

Нека сега решим същата система, като единствено по главния диагонал на матрицата H добави 10^{-15}, което е сравнимо с машинната грешка:

```
In[22]:= H1 = H + DiagonalMatrix[Table[10^-15, {20}]];
         LinearSolve[H1, b] // N
```

```
Out[23]= {4.18461, -428.767, 9520.28, -66 286.5, -19 407.5, 2.19682 × 10^6,
  -1.01854 × 10^7, 1.79997 × 10^7, -4.97211 × 10^6, -1.87141 × 10^7,
  3.77987 × 10^6, 2.21275 × 10^7, 7.80635 × 10^6, -1.97735 × 10^7, -2.34981 × 10^7,
  5.52176 × 10^6, 3.27829 × 10^7, 1.19906 × 10^7, -4.52997 × 10^7, 1.83143 × 10^7}
```

Отново сме работили символно, т.е. Mathematica е върнала точното решение на променената система. Виждаме, че решението няма нищо общо с решението на системата $Hx = b$. От друга страна, ние сме променили входните данни със стойности, сравними с машинната грешка, т.е. такива грешки са неизбежни заради закръгляването. Тогава можем да очакваме, че числено решаване на системата $Hx = b$ няма да бъде добро. Действително:

```
In[13]:= LinearSolve[N[H], Table[1, {20}]]
```

```
LinearSolve::luc : Result for LinearSolve of badly conditioned matrix {≪1≫} may contain significant numerical error
```

```
Out[13]= {-9.92855, 1073.33, -21 305.6, -18 047.6, 3.8623 × 10^6,
  -4.18194 × 10^7, 2.09821 × 10^8, -5.51336 × 10^8, 6.27157 × 10^8, 3.06172 × 10^8,
  -1.50994 × 10^9, 6.47086 × 10^8, 1.62751 × 10^9, -8.38695 × 10^8, -3.11475 × 10^9,
  4.47977 × 10^9, -1.8183 × 10^9, -4.77868 × 10^8, 5.73674 × 10^8, -1.22314 × 10^8}
```

Работейки числено, Mathematica връща съвършено различен резултат, като извежда съобщение, че системата е лошо обусловена.

Ако задачата е лошо обусловена, от нито един числен метод не може да очакваме добър резултат. Численият метод работи със закръглените входни данни и дори да реши абсолютно точно тази задача, резултатът можем да очакваме да е съществено различен от решението на оригиналната задача (т.е. без да са закръглени входните данни). Ето защо, единствената възможност е задачата да се запише в еквивалентна форма, в която ще бъде по-добре

обусловена.

Глава 5

Числени методи за намиране на собствени стойности и собствени вектори на матрица

Дефиниция 3. *Казваме, че числото λ е собствена стойност на матрицата A, ако съществува вектор $\overline{x}$, за който $A\overline{x} = \lambda\overline{x}$, т.е. изображението на вектора $\overline{x}$ е вектор, колинеарен на дадения.*

Твърдение 2. *Собствените стойности на матрицата A са решенията на уравнението* $\det(A - \lambda I) = 0$, *където I е единичната матрица.*

Доказателство. От $\overline{x} = \lambda\overline{x}$ следва, че хомогенната система $(A - \lambda I)\overline{x} = 0$ има поне едно ненулево решение, $\overline{x}$, и следователно матрицата $A - \lambda I$ е особена, т.е. има нулева детерминанта. □

Собствените стойности са съществена характеристика на линейния оператор, задаващ се със съответната матрица.

5.1 Метод на Данилевски

Методът на Данилевски е директен метод за намиране на собствените стойности на дадена матрица.

Основна идея, на която се базират директните методи, е да приведем оригиналната матрица в такава, за която задачата (в случая за намиране на собствените стойности) е лесно решима. При това трябва да извършваме такива преобразования, че задачата за оригиналната матрица и тази, получена след преобразованията, да имат едни и същи решения.

Ако една матрица е в нормална форма на Фробениус, т.е.

$$A = \begin{bmatrix} p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix},$$

то нейният характеристичен полином може лесно да бъде намерен, като развием $\det(A - \lambda I)$ по първия ред, и собствените стойности на A са решенията на уравнението

$$p(x) \equiv \lambda^n - p_1\lambda^{n-1} - \cdots - p_n = 0.$$

Ще илюстрираме метода със следния пример.

Задача 11. Да се намерят собствените стойности на матрицата

$$A = \begin{bmatrix} -1 & 3 & -2 \\ 1 & 1 & -1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Решение. На първата стъпка ще приведем последния ред във вид, съответстващ на нормалната форма на Фробениус. За тази цел втория стълб ще прибавим, умножен с -1, към първия. Това преобразование съответства на умножението отдясно на матрицата A с матрицата

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

За да бъде това преобразование на подобие, трябва да умножим отляво с нейната обратна

$$M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Така, след първата стъпка получаваме

$$A^{(1)} = M_1^{-1}AM_1 = \begin{bmatrix} -4 & 3 & -2 \\ -4 & 4 & -3 \\ 0 & 1 & 0 \end{bmatrix}.$$

На втората стъпка остава да приведем и втория ред на матрицата в подходящия вид. За тази цел прибавяме първия стълб към втория, прибавяме първия стълб, умножен с -3/4 към третия и делим първия стълб на -4. Това преобразование можем да извършим, като умножим матрицата $A^{(1)}$ отдясно с

$$M_2 = \begin{bmatrix} -1/4 & 1 & -3/4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

За да направим преобразование на подобие, отляво умножаваме с обратната матрица

$$M_2^{-1} = \begin{bmatrix} -4 & 4 & -3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Окончателно получаваме матрицата

$$A^{(2)} = M_2^{-1}A^{(1)}M_2 = \begin{bmatrix} 0 & 1 & -4 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

която е подобна на A, т.е. има същите собствени стойности като нея, и е в нормална форма на Фробениус. Нейният характеристичен полином е $\lambda^3 - \lambda + 4$ и следователно собствените стойности на A са 1.79632 и $0.898161 \pm 1.19167i$. $\square$

Привеждаме примерна имплементация на метода на Данилевски.

In[1]:=
```
A = {{-1, 3, -2}, {1, 1, -1}, {1, 1, 0}};
n = Length[A];
For[i = n, i ≥ 2, i--,
 M = IdentityMatrix[n];
 For[j = 1, j ≤ i - 2, j++,
  M[[i - 1, j]] = -A[[i, j]] / A[[i, i - 1]];
 ];
 For[j = i, j ≤ n, j++,
  M[[i - 1, j]] = -A[[i, j]] / A[[i, i - 1]];
 ];
 M[[i - 1, i - 1]] = 1 / A[[i, i - 1]];
 A = Inverse[M].A.M
]
p[λ_] = λ^n;
For[i = 1, i ≤ n, i++,
 p[λ_] = p[λ] - A[[1, i]] λ^(n-i)
]
p[λ]
```