Introduction to numerical methods with Matlab procedures

Jacek Ronda

Graeme John Oliver

2010

Contents

List of Figures

Chapter 1

Introduction

This textbook is based on several well recognized textbooks and our lectures and other materials used in presenting the course on 'Numerical methods in engineering' in the Department of Material Engineering and Industrial Information Technology at the AGH University of Technology in Krakow as well as for postgraduate courses in the Department of Mechanical Engineering of the Cape Peninsula University of technology in Cape Town, South Africa. The chapters cover the following topics:

1. Representation of numbers and words in computers,
2. How to start numerical programming with Matlab,
3. Roots and zeros,
4. Interpolation and extrapolation,
5. Least squares,
6. Solution of systems of equation system,
7. Numerical integration and quadrature,
8. Random numbers,
9. Fourier analysis and other analytical methods of approximation,
10. Eigen-problems, eigenvalues and singular values,
11. Ordinary differential equations,
12. Simple partial differential equations,

13. Working with vectors, matrices and Cartesian Tensors for continuum mechanics

14. Introduction to programming the Finite Element Method (FEM) for stress analysis

15. Introduction to programming computational fluid dynamics (CFD) problems

The reader should follow the following steps to successfully develop computer programs or MATLAB scripts for numerical methods:

- understand each numerical recipe step by step,
- know the limitations and advantages of each numerical method,
- be aware of the unavoidable presence of errors stemming from computer operations and approximation methods,
- test newly developed solvers by comparison with solutions of benchmark problems which would adequately cover any problems that may arise.

Within numerical computations, there are two principal sources of error. One source of error is due to the approximations within the numerical method itself, e.g. truncated Taylor series are used as a basis of finite difference approximations for derivatives. Such errors are called *truncation errors*. The second source of error is due to the fact that a computer has limited precision, i.e. it can store only a finite number of decimal places. These errors are called *roundoff errors*. Truncation error is present not only in the approximation of derivatives, e.g. truncation error arises also in numerical integration when each increment of area under a graph is calculated using a polynomial approximation to the true function. Then the truncation error is related to the order of such polynomial. In this case, using n-th order approximating polynomial leads to an error in the integral, related to the size of an increment $\triangle x = h$, of the order $O(h)^{n+2}$, and is called the local error. The summation of the integral over a domain $a \leq x \leq b$ results also in the summation of all fractal errors and gives a global error of the order $O(h)^{n+1}$. The truncation error decreases along with the reduction of increment h. In the case of the finite difference approximation this is in agrement with the fundamental theorem of calculus, i.e. the Cauchy definition of a derivative.

Roundoff errors stem from the storage capacity of computers and the fact that a finite number of digits can be used to express a number in a computer.

This means that the computer value assigned to numbers without a finite digit representation, e.g. number

$$\pi = 3.14159265358979323846264338327950288419716939937 51$$

with e.g. 50 digits, will be rounded or truncated due to the limited precision of the machine. Therefore, taking π with even more digits, e.g.

$$\pi = 3.14159265358979323846264338327950288419716939937 51$$
$$05820974944592307816406286208998628034825342117067$$

with 100 digits, will lead to the same value being assigned in the computer memory. It is also important to know that there is a limit as to how large and how small a number can be stored or processed in a computer. The small values resulting from roundoff are usually insignificant. But sometimes, for example, when two nearly identical numbers are subtracted, it can lead to a relatively large roundoff error which depends on the number of significant digits retained in the calculation. It means that approximations to derivatives could be improved by reducing h to some optimal level subject where the truncation error is also reduced. However, any further decrease of h would make the approximation worse because the roundoff error would become large and would dominate the solution for very small values of h. In the case of logical operations, the roundoff error could be a deteriorating factor when a logical operation depends on establishing equality between two values which are evaluated in the presence of truncation or roundoff errors.

Finally, in applied science, where measured data is often used in calculations, the so called measurement errors appear, which could overlap with the above mentioned errors reducing the accuracy of numerical methods used and/or lead to difficulties in the interpretation of the results obtained because of the increased uncertainty.

We assume that the reader is familiar with the elementary ideas of calculus, vector operations and matrices, ordinary differential equations and has some computer programming experience although we will briefly introduce some of these topics again in the text.

Matlab procedures are included in each chapter of this book.

Chapter 2

Representation of information in computers

There are two types of data stored in a computer memory: numeric, and alphanumeric. The numeric data can be of the integer or real type and alphanumeric is text or character based. On the other hand all information available in a memory can be split into two categories: data, and instructions to perform certain operations. An individual digit is called a bit and all the numerical data is stored in the form of groups of binary digits. Bits are grouped into collections of 8 bits called bytes. The computer memory is measured in terms of bytes. Bytes are grouped further into four or more bytes and they form a single computer word. Currently the most frequently used computer word size is 32 bits, i.e. 4×8 bits but systems of 64 bits, i.e. 8×8 have been used in high end computing applications for some time because of the amount of memory that needs to be addressed and personal computers (PCs) are also reaching this limit as 32 bit computers can only effectively utilize up to 4 gigabytes, i.e. 2^{32} bytes of Random Access Memory (RAM). Operating system overheads may also reduced this value further. A larger word size increases the number of digits that can be stored and therefore reduces round off error also. Alternatively the representation of numbers can be coded to use 2 or 4 words to represent numbers leading to the so called double or quadruple precision in certain programming languages which can reduce round off error as well as represent larger or smaller numbers than with single precision.

There are several methods for representing numeric and character data. Numbers, characters and instructions are represented in a computer by a specific patterns of bits. A given pattern, subject to its interpretation, represents different objects. For example, the pattern of bits: [**1101**] can be

interpreted in three ways:

- it is the decimal number 13,
- it is the representation of the letter E,
- it is the arithmetic instruction ADD.

Therefore the method, by which we utilize bit representations, must be specified exactly because bits themselves are meaningless.

2.1 Elementary approach to number representation and errors

Notes on the computer representation of numbers should be introduced by saying that the most popular number base is motivated by the construction of the human body, i.e. ten fingers in total on human hands. Similarly designers of computer systems with on-off switches,i.e. 2 states, where numbers are stored using either the presence or absence of voltage, used 2 as the appropriate base.

Generally, for different base systems unique one-digit symbols are used for the representation of a number. For example for a base **n**, **n** unique symbols are specified. The most four popular base systems used in computer science are: decimal, binary, octal, and hexadecimal. Specific features of those systems are listed in Table(2.1).

Table (2.1)

System's name	Base	Symbol	"Decimals"
decimal	$base_{10}$	none or D	0,1,2,3,4,5,6,7,8,9
binary	$base_2$	B	0,1
octal	$base_8$	Q or O	0,1,2,3,4,5,6,7
hexadecimal	$base_{16}$	H	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

A simple algorithm for converting decimals to binary can be written as follows:

1. Draw some blanks to hold the bits (0's and 1's) about as many as would be necessary. Under each blank, write the powers of 2 that go with the positions:

   ... ...
   32 16 8 4 2 1

2. Answer the question: What is the biggest power of 2 that is smaller than or equal to the converted number?
3. Put a 1 in the position of that power of 2.
4. Subtract the power of 2 just found from the processed number.
5. Repeat steps 2, 3 and 4 until the processed number is 0.
6. Put 0's in all the positions that do not have 1's.

Example 2.0

♣*Convert decimal* 11_{10} *to binary*

- Step 1

  ... ...
  8 4 2 1

- Step 2 The largest power of 2 less then (or equal to) 11 is 8.
- Step 3 Put a 1 in the 8 position

  1
  ... ...
  8 4 2 1

- Step 4 Subtract $11 - 8 = 3$
- Repeat
- Step 2 The largest power of 2 less than (or equal) 3 is 2.
- Step 3 Put a 1 in the 2 position.

  1 1
  ... ...
  8 4 2 1

- Step 4 Subtract $3 - 2 = 1$
- Repeat
- Step 2 The largest power of 2 less than or equal to 1 is 1.
- Step 3 Put a 1 in the 1 position.

  1 1 1
  ... ...
  8 4 2 1

- Step 4 Subtract $1 - 1 = 0$.
- Step 5 Put a 0 in all empty positions

1	0	1	1
...	...	...	...
8	4	2	1

So decimal 11 is 1011 in binary ♠

In general, a number *abcde* in base **n** can be written symbolically as $(abcde)_n$ and is represented in decimals by

$$(a \times n^4) + (b \times n^3) + (c \times n^2) + (d \times n^1) + (e \times n^0) \tag{2.1}$$

The procedure for **converting any integer** $\mathcal{N}$ expressed in the base 10 to the number represented in the base **n** is as follows:

- divide $\mathcal{N}$ by **n**,
- take the remainder as a base **n** digit. The remainder will form the rightmost digit of the whole number in base **n** representation.
- divide the quotient by **n**,
- the new remainder is taken as a base **n** digit the left of the last digit in the result,
- repeat steps 3 and 4 until a quotient of zero is obtained,
- the string of successive remainders at this point represents the base **n** equivalent of the given number.

Example 2.1

♣ *The binary representation of* 10_{10} *is*

Base	Decimal number		Remainder
2	10	5	0
2	5	2	1
2	2	1	0
2	1	0	1

The group remainders written in reverse order is 1010.♠

Example 2.2

♣ *The base 10 expansion for the number 123456 is*

$$\begin{aligned} 123456 &= 6 + 50 + 400 + 3000 + 20000 + 100000 \\ &= 6 \times 10^0 + 5 \times 10^1 + 4 \times 10^2 + 3 \times 10^3 + 2 \times 10^4 + 1 \times 10^5 \end{aligned}$$

♠

The routine for **a conversion of a real number decimal fraction** $\mathcal{D}_\mathcal{F}$ to base **n** is the following:

1. Multiply $\mathcal{D}_\mathcal{F}$ by **n**,
2. Take the integral part of the product as a base **n** digit in the result,
3. Multiply the fractional part of the product by **n**,
4. Take the integral part of the new product as a base **n** digit to the right of the last digit in the result,
5. Repeat step 3 and 4 and then
   - either a zero value is obtained for the fractional part,
   - or the required number of significant digits is obtained.

Example 2.3

♣ *Convert decimal fraction* 0.625_{10} *to the binary base*

Fractional part	× New base	Decimal+remainder
0.625	×2	=**1**.250
0.250	×2	=**0**.500
0.500	×2	=**1**.000

where the integral part of the products is highlighted in bold. The corresponding binary representation is obtained by grouping the bold numbers and placing a zero in the front of the decimal place in the binary number, i.e. $\mathbf{0.101}_2$.♠

Example 2.4

♣ *Find the binary representation for the decimal fraction* 0.525_{10} *up to six digits*

Fractional part	× New base	Decimal+remainder
0.525	×2	**1**.050
0.050	×2	**0**.100
0.100	×2	**0**.200
0.200	×2	**0**.400
0.400	×2	**0**.800
0.800	×2	**1**.600

The binary representation for the number is thus $\mathbf{0.100001}_2$.♠

Decimal fraction numbers can be rational or irrational and they can be written as

$$(i_n i_{n-1} i_{n-2} ... i_2 i_1 i_0 . d_1 d_2 d_3 ... d_{n-2} d_{n-1} d_n ...)_{10} = \sum_{k=0}^{n} i_k \times 10^k + \sum_{k=1}^{\infty} 0.d_k \times 10^{-k} \tag{2.2}$$

Following that, a general formula for converting a number in a system with the base β to the decimal system is expressed by

$$(i_n i_{n-1} ... i_2 i_1 i_0 . d_1 d_2 d_3 ... d_{n-1} d_n ...)_{\beta} = [\sum_{k=0}^{n} i_k \times \beta^k + \sum_{k=1}^{\infty} 0.d_k \times \beta^{-k}]_{10} \tag{2.3}$$

Example 2.5

♣ *Convert the number* $(5432.2345)_8$ *to the decimal system.*

Start by converting the integer part and then work out the fractional part of the given number.

$$\begin{aligned} (5432)_8 &= 2 \times 8^0 + 3 \times 8^1 + 4 \times 8^2 + 5 \times 8^3 \\ &= 2 + 8(3 + 8(4 + 8 \times 5)) \\ &= 2842; \end{aligned} \tag{2.4}$$

$$\begin{aligned} (0.2345)_8 &= 2 \times 8^{-1} + 3 \times 8^{-2} + 4 \times 8^{-3} + 5 \times 8^{-4} \\ &= 8^{-4}(5 + 8(4 + 8(3 + 8 \times 2))) \\ &= \frac{1253}{4096} \\ &= 0.3059 \end{aligned} \tag{2.5}$$

The decimal number is 2842.3059 ♠

Any system with a base α can be converted to any system with a base β with the following procedure:

1. For the case where $\alpha < \beta$
   - Express the number $\mathcal{N}_\alpha$ in nested form for appropriate powers of α,
   - Replace each digit by the corresponding number in base β,
   - Carry out calculations in the base β,
2. For the case where $\alpha > \beta$
   - Divide the number representation by β,
   - Take the next remainder as the next digit in the base β representation,
   - Repeat the procedure on the quotient.

The above can be illustrated by applying an intermediate base, i.e. binary base, to convert a digital number to the octal base number. It can be done quite easy using the binary-octal Table(2.2) where groups of three binary digits can be translated to octal.

Table (2.2)

$\alpha = 2$	000	001	010	011	100	101	110	111		
$\beta = 8$	0	1	2	3	4	5	6	7		
$\alpha = 2$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
$\gamma = 10$	0	1	2	3	4	5	6	7	8	9

The following example illustrates the procedure.

Example 2.6

♣ *Express a number in three bases*

$$\begin{aligned}(425.451)_{10} &= [(110)(101)(001).(011)(100)(111)]_2 \\ &= (651.347)_8\end{aligned}$$

♠

2.2 Advanced approach to computer representation of numbers

2.2.1 Computer representation of numbers

The class of real numbers can be split into two subclasses: rational numbers and irrational numbers.

Number class	Description
Rational numbers	real numbers which consist of a ratio of two integers
Irrational numbers	can not be written as the ratio of two integers, eg. $\sqrt{2}$, π, e.

Rational numbers are of two types: integers and non-integral fractions.

Signed integers: negative or positive (plus or minus) can be represented in three ways:

Characteristic feature	Description
Signed magnitude (SM)	leading bit represents a sign and the remaining bits represent the corresponding unsigned number
One's complement representation (1'sCR)	negative representation is obtained by **flipping** all bits of the unsigned number
Two's complement representation (2'sCR)	negative representation is obtained by flipping all bits and adding one

$\diamondsuit$ **Definition** : **Flipping** a bit means the replacement of a bit by its "opposite" value, i.e. flipping **0** produces **1** and vice versa. $\diamondsuit$

All three methods have two common features:

- the leftmost bit is reserved to represent the sign of a number,
- **0** indicates a positive number and **1** indicates a negative number.

Signed modulus (SM) representation

In this method the sign and modulus, i.e. absolute value, of the integer are represented separately. The integer value is negative when the leftmost bit is **1** and it is positive when the bit is **0**. A negative (positive) integer is represented by **1** (**0**) in the leftmost bit and the absolute value in the remaining bits. Integers that can be stored in SM ranges from -2^{n-1} to $+2^{n-1}$.

Example 2.7

♣ *Integers* $+(30)_{10}$ *and* $-(30)_{10}$ *can be represented on an 8-bit computer as follows:*

- $+(30)_{10} = 00011110$,
- $-(30)_{10} = 10011110$,

♠

♠

The procedure used for converting the decimal base to SM using N bits is the following:

- Ignore the minus sign,if there is one, and convert the base 10 value to binary,
- When the binary representation has less than $(N-1)$ bits, it can be filled to $(N-1)$ bits with zeros.
- If the value is positive, set the bit b_{N-1} to zero alternatively if it is negative, set it to one.

The inverse process of converting SM to decimal follows:

- Ignore the sign bit and consider only the last (rightmost) $(N-1)$ bits of the number and convert them to the decimal base. It will produce a non-negative value.
- Add a negative sign to the base ten value if the sign bit is 1.

Example 2.7

♣ *Represent the number* $(3)_{10}$ *in binary base using SM procedure and four bits.*

Analytically, the value $(3)_{10}$ in the base 2 is equivalent to $(11)_2$ but in SM it must be represented by four bits and therefore, two additional zeros must be added in front of the value that produces

$$(3)_{10} = (11)_2 = (0011)_{SM}$$

♠

Example 2.8

♣ *Represent the number* $-(3)_{10}$ *in binary base using SM procedure and four bits.*

It can be represented by three bits 011 and since -3 is negative a sign bit 1 should be added and finally 1011 will be produced. The same operation can be done by converting the representation from three to four bits and flipping the sign bit. ♠

Example 2.8

♣ *Represent the number* 15_{10} *in SM by a four bit system*

Expressing 15_{10} in base 2 gives $(1111)_2$. Converting this back to base 10 gives -7. We can see that the number 15_{10} represented by four bits is larger than the largest possible representable number using four bits in SM. Therefore, 15_{10} cannot be represented by using four bits and the minimum necessary bits, in this case, is five.♠

The last problem is a common occurrence because, for a finite number of bits, only a finite number of values can be represented and therefore there will be values (infinitely many, mathematically speaking) beyond the maximum value that cannot be represented by a particular number of bits.

Therefore we can ask the following question: How many positive and negative values can be expressed by N bits? It is well known that N bits produce 2^N various bit-strings. Half of them are positive, i.e. $\frac{2^N}{2} = 2^{N-1}$, and the other half 2^{N-1} is negative. Let us consider first the positive values. The most significant bit for them is 0, and this means that only $(N-1)$ bits can used to represent positive values. Therefore we may rephrase the above question: What is the maximum value that can be expressed by $(N-1)$ bits? The answer is $(2^{N-1}-1)$. The smallest negative value is $-(2^{N-1}-1)$ where the sign bit is 1.

There are some problems with using SM values:

- One of them relates to the problem with the two representations of zero: a "positive" zero, and a "negative" zero. The first zero is represented by a string of bits with N zeros. Another representation is by a string starting with 1 followed by $(N-1)$. This creates a problem with the comparison or summation of numbers that are performed by operations built into the hardware.

- Only (2^N-1) values can be represented in 2^N in SM because of the presence of two zeros.

- Another problem is with the summation (addition) of numbers. It would be convenient to add signed integers like unsigned integers, but this does not work with the SM system, as is illustrated by the example: *Add* -1 *and* -1 *using four SM bits. Adding* 1001 *and* 1001 *results in*

0010, *i.e.* $+2$ *which is obviously incorrect as the most significant bit is ignored because of the shortage of four bit range. In five bits it can be expressed by 10010. Another method is to add everything and ignore the sign bit, evaluate the result and keep the same sign bit as before. Thus, adding* -1 *to* -1 *results in* 2*, and preserving the sign bit the value* -2 *is obtained. It works fine when adding two positive or two negative numbers. The problem appears when adding* -1 *to* 1*. It requires two separate encodings, in the hardware, of integer arithmetic for the addition of unsigned and signed integers. unsigned integers.*

Fortunately, negating is easy for SM but it should be stressed that representing a negative value and negating a value are two different instructions. To negate a value $\mathbf{v}$ means to perform an operation, i.e. take some value $\mathbf{v}$ and compute $-\mathbf{v}$. The prime property of negation is $-(-\mathbf{v}) = \mathbf{v}$, i.e. double negation of a value results in the original value. In the case of the SM representation of a number with N bits, the negation can be denoted as $\mathbf{b}_{N-1} = \hat{\mathbf{b}}_{N-1}$, where $\hat{}$ means the logical negation, i.e. (NOT). Negation in SM is easy by flipping the most significant bit.

One's complement representation

The one's complement representation (1'sCR)is another approach to the representation of computer numbers where the negation of a number is produced by complementing each bit of this number. The sign bit is selected as before for SM, i.e. 0 for positive numbers and 1 for negatives. **Complementing** a single bit is equivalent to subtracting it from 1, e.g. $\hat{0} = 1 \Rightarrow 1 - 0 = 1$ and $\hat{1} = 0 \Rightarrow 1 - 1 = 0$. Similarly, complementing each bit of an n-bit number is equivalent to subtracting that number from $2^N - 1$, e.g. negating the five bit number 01101 is based on subtracting it from $11111 = 11111_2 \equiv 2^N - 1 = 2^5 - 1 = 31_{10}$ that results in 10010 as can be seen below

	1	1	1	1	1
-	1	1	1	0	1
	1	0	0	1	0

The understanding of 1'sCR requires the understanding of what flipping a bit means (as for the SM representation). Briefly, flipping all bits of N bit number means to obtain $\mathbf{b}_i = \hat{\mathbf{b}}_i$ for $0 \leq i \leq N-1$, where $\hat{\mathbf{b}}_i$ means flipping bit i. Sometimes, the operation "flip all bits" is marked by the operator "$\sim$" as used in the computer language C. Negating a bit is the same as flipping a bit which is equivalent to the logical negation of that bit and it is also the same as subtracting the bit's value from 1. For the addition of 1'sCR

numbers we first perform unsigned addition, including the sign bit, and we then add the sign bit to the answer. This is illustrated by the following two examples.

Example 2.9 ♣

		0	1	1	1		(+7)
+		1	0	1	1	+	(- 4)
	1	0	0	1	0		
		0	0	1	0		
+					1		
		0	0	1	1		(+3)

♠

Example 2.10 ♣

		0	0	1	1		(+3)
+		0	0	1	0	+	(+2)
	0	0	1	0	1		
		0	1	0	1		
+					0		
		0	1	0	1		(+5)

♠

The addition of 1'sCR numbers is simpler and more regular than the addition of SM numbers.

The algorithm for converting decimal numbers to 1'sCR code using N bits can be written as follows:

- ignore the minus sign and express a value in the decimal base with N bits, unsigned,
- with the minus sign, flip all bits of the N bit number from step (1), otherwise, step (1) has the correct answer.

The reverse process is carried out with the following steps:

- when the most significant bit is 1, flip all bits, but when the most significant bit is 0, go to the next step,
- convert the result to base 10 (the value should be non-negative),
- add a negative sign when the sign bit of 1'sCR representation was 1.

The above algorithm works only for some numbers and there are some numbers that cannot be represented in 1'sCR. Each positive number can be transferred to a unique negative number by flipping just the sign bit, as in SM, or flipping all bits but unfortunately, 0 is the exception to this rule.

Half of 1'sCR representation has a 1 as the most significant bit and the other half has a 0. This is exactly the same as for the SM representation where $2^N - 1$ numbers, including positive zero, are positive and $2^N - 1$ numbers, including negative zero, are negative. In 1'sCR representation two zeros exist and it makes the hardware more complicated. Adding values in 1'sCR works better than in SM. The negation operation in 1'sCR requires flipping of all bits.

Both SM and 1'sCR have the same range of values as both of them have the same minimum and maximum values as well as two representations of zero. Both representations can not use the hardware for unsigned integer addition to carry out addition. There are many similarities, but most users would find 1'sCR more difficult to understand than SM, because at the first glance it seems that it requires more work for flipping all bits instead of flipping just one sign bit.

Two's complement representation

The two's complement (2'sCR) system has a single zero and the addition of two numbers expressed in this system can be done by using the same hardware as for the addition of unsigned binary numbers (UB). The 2'sCR number is represented by a N bit string, i.e. $\mathbf{B} = \mathbf{b}_{N-1}\mathbf{b}_{N-2}...\mathbf{b}_1\mathbf{b}_0$ where $\mathbf{b}_{N-1}$ is the most significant bit, which we can think of as the sign bit. A number is negative when the leftmost bit, known as the most significant bit (MSB), is 0 and a number is nonnegative when the MSB is 1.

Negating a number in 2'sCR requires flipping the bits and adding 1 that can be noted by $-\mathbf{B} = \sim \mathbf{B} + 1$, with $\sim \mathbf{B} = \hat{\mathbf{b}}_{N-1}\hat{\mathbf{b}}_{N-2}...\hat{\mathbf{b}}_1\hat{\mathbf{b}}_0$ where $\sim \mathbf{B}$ is $\mathbf{B}$ with all bits flipped and $\hat{\mathbf{b}}_i$ means $\mathbf{b}_i$-th bit flipped. Both numbers: $\sim \mathbf{B}$ and 1 are treated as unsigned binary numbers when carrying out $\sim \mathbf{B} + 1$. When a bit string $\sim \mathbf{B}$ is all composed of all ones, the result is $(N+1)$-bit number consisting of 1 followed by N zeros.

We illustrate this with examples of transformations from the binary to digital:

Transformation	Binary	Digital
four bit unsigned number	1101_2	13_{10}
positive number in 2'sCR	01101	$+13_{10}$
negative number in 1'sCR	10010	-13_{10}
negative number in 2'sCR	10011	-13_{10}

A frequently used confusing phrase is: "take the two's complement of a number" this usually means: "negate some number that is already in two's complement format". The algorithm for addition of 2'sCR numbers is as follows:

- conduct unsigned addition on A and B including their MSB/sign bits,
- ignore any carry over.

This is illustrated by the following example.

Example 2.11 ♣

Find $0111 + 1100$ or alternatively find $(+7) + (-4)$.

First add 0111 and 1100 as unsigned numbers.

		0	1	1	1
+		1	1	0	0
	1	0	0	1	1

♠

and next discard the carry over of 1. The correct answer is 0011, i.e. +3.

The question is: why does it work? The answer is rather simple:

- for N-bit numbers in 2'sCR, the negation of B is $2^N - B$,
- $A - B = A + (-B) = A + (2^N - B) = (A - B) + 2^N$,
- if $A \geq B$ then $(A - B)$ is a positive number and 2^N represents a carry out of unity,
- discarding this out is equivalent to subtracting 2^N, that leads to the desired result $(A - B)$,
- if $A < B$ then $(A - B)$ is a negative number that leads to $2^N - (A - B)$ that is the desired result, i.e. $-(A - B)$.

The algorithm for converting a number from the decimal base to N-bites number 2'sCR can be written as follows:

- if a number is non-negative, convert it to unsigned binary (UB),
- when the number B is negative, convert $-B$ to UB.

The reverse process, i.e. the algorithm for converting 2'sCR number to the decimal base consists three steps:

- when the most significant bit is 1, flip all bits and add 1 but when it is 0 then go to the next step,
- convert the result of step 1 to the decimal base. The value should be non-negative.
- add the negative sign if the sign bit was originally 1.

Alternate ways for negation numbers in 2'sCR

There are three other methods than flipping bits and adding the unity and producing the same result. They are described in Table (2.3)

Table(2.3)

Method	Algorithm
1-st method	• find the rightmost 1, i.e. the least significant 1, • that means, find i such that $\mathbf{b}_i = 1$ and $\mathbf{b}_j = 1$ for $j < i$ • flip all bits left of this, i.e. flip bits $\mathbf{b}_k$ for $k > i$.
2-nd method	◇ negate number B expressed in N bits ◇ carry on unsigned binary subtraction $-B = 1(0^N) - (B)$, where $1(0^N)$ means 1 followed by N zeros ◇ subtract from 1 followed by N zeros as if both numbers were unsigned,
3-rd method	○ subtract 1 and then flip bits

Uniqueness and other properties of 2'sCR representation

Negating 0 in 2'sCR results back to 0 that confirms the single representation for 0. It results from the set of operations:

- represent 0 in 4-bits system $\Rightarrow$ 0000,
- flip all bits $\Rightarrow$ 1111,
- add 1 $\Rightarrow$ 10000,

- through out the most significant bit $\Rightarrow$ 0000.

Now consider the uniqueness problem: can one find another number that has the same uniqueness property? We can see that the negation of 1000 results is also 1000. It follows from the following operations:

- start with 1000 and flip all bits $\Rightarrow$ 0111,
- add 1 $\Rightarrow$ 1000.

However this is not true for all four-bit 2'sCR numbers. We can this by trying out another number, eg. 0100. The negation of 0100 in 2'sCR is 1100 which is not the same.

Another question is how to identify what value is represented by 1000? Clearly it is a negative value because the sign bit is 1. One idea is to flip all bits and add 1 at first and next convert it to the digital base and add a negative sign. To this purpose the number $1(0^{N-1})$, where $N = 4$, can be treated as unsigned integer. The representation of 1000 in 2'sCR corresponds to digital 8. Adding a negative sign results in -8.

Consider now the addition problem of two 2'sCR numbers: $X + Y$. It works without problems, when both numbers are non-negative or one is negative and another is positive. But how it works when both of them are negative? Then numbers X and Y can be correspondingly written as $1(0^N) - -X$ and $1(0^N) - -Y$, and adding them results in $1(0^{N+1}) - (-X + (-Y))$, which is written for $(N+1)$-bits rather than for N-bits representation. The key point in understanding how it works is to realize that adding two negative values results in a negative value. Therefore, when $-X + (-Y)$ is not too large it can be written as $1(0^N) - (-X + (-Y))$ that is equivalent to writing the value for N-bits in 2'sCR. The representation $1(0^{N+1}) - (-X + (-Y))$ for $(N+1)$-bits in 2'sCR is equivalent to the representation for N-bits in 2'sCR subject that $(-X + (-Y))$ is not too large.

Number of values for N-bits in 2'sCR

The important subject is o determine how many values can be represented in 2'sCR using N-bits and assuming that N is fixed. Evaluating that number we should consider that half the representation has 1 as the most significant bit (MSB) and the other half has 0 in that place. Fortunately, there are no negative zeros. The smallest negative number is -1 and the largest magnitude negative number is -2^{N-1}. The range of negative numbers is from -1

to -2^{N-1}. There are also 2^{N-1} number representations with 0 as MSB. However, one of those representations is zero and therefore, there are $2^{N-1}-1$ starting with +1. That means that there is one more negative value than positive values.

Comparison of the signed number systems

Presented above three representations of signed numbers: SM, 1'sCR, and 2'sCR, are compared by using eg. 4-bits numbers in Table (2.4).

Table (2.4)

Decimal	SM	1'sCR	2'sCR
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

It can be seen that:

- positive numbers are the same in all three representations,
- SM i 1'sCR have two ways for representing zero that makes computing more complicated,
- 2'sCR has asymmetric ranges because there is one more negative numbers and -8 but not $+8$.

Table(2.5)

	unsigned	SM	1'sCR	2'sCR
smallest	$0000 \Rightarrow 0$	$1111 \Rightarrow -7$	$1000 \Rightarrow -7$	$1000 \Rightarrow -8$
largest	$1111 \Rightarrow 15$	$0111 \Rightarrow +7$	$0111 \Rightarrow +7$	$0111 \Rightarrow +7$

Table(2.6)

	unsigned	SM	1'CR	2'sCR
smallest	0	$-(2^{N-1}-1)$	$-(2^{N-1}-1)$	-2^{N-1}
largest	2^N-1	$+(2^{N-1}-1)$	$+(2^{N-1}-1)$	$+(2^{N-1}-1)$

The representation 2'sCR is preferred because of one zero presence and simplicity of the addition algorithm. Ranges of signed number systems are compared in Tables (2.5) and (2.6) at first for a 4-bit number and next for N-bit number.

Non-integral real numbers

Positive binary fractions

The representation of unsigned binary fractions proceeds in the same way as decimal fractions.

Example 2.12 ♣

Express the decimal fraction 0.625_{10} as the binary fraction.

$$\begin{aligned} 0.625_{10} &= 1 \times \tfrac{1}{2} + 0 \times \tfrac{1}{4} + 1 \times \tfrac{1}{8} \\ &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 0.101_2 \end{aligned}$$

♠

Similarly as in decimals, each place to the right of the decimal point represents a negative power of the base number 2. If the binary representation consists of N-bits to the right of a decimal, the precision of the number is 2^{-N}. The largest number that can be represented by such a representation is $1-2^{-N}$ but the smallest number is 2^{-N}. For a fraction with 15-bits representation this gives a range of approximately from 0.99997 to $3.05E-5$.

Signed binary fractions

In signed binary fractions the first single digit to the left of the decimal point represents the number $-1 = -2^0$. The rest of the representation of the fraction is like for signed integers. This leftmost bit represents a sign bit just as with 2'sCR. When this bit is set, the number is negative, otherwise the number is positive. The largest positive number that can be represented is

still $1 - 2^{-N}$ but the largest negative number is -1. The resolution is still $1 - 2^{-N}$.

Signed binary fractions can be extended to include all numbers by representing the number to the left of the decimal point as a 2'sCR, and the number to the right of the decimal point as a positive fraction. That can be verified in the example.

Example 2.12

♣ *Convert decimal* -6.625 *to binary.*

$(-6.625)_{10} = (-7 + 0.375)_1 0 = 1001.011_2$

♠

2.3 IEEE floating point representation

IEEE floating point standard (FPS) was developed in [9] and was applied by Intel and Motorola as the ANSI/IEEE Std 754-1985.

Floating point numbers are represented by numbers which do have decimal points and digits: zero or non-zero, to the right of decimal point. A floating point number is represented in the computer by

$$\mathcal{N}_{fln} = \mathbf{sign} \times \mathbf{fraction} \times (\mathbf{base}^{\mathbf{exponent-bias}}) \tag{2.6}$$

where

- **sign** can be + or −,
- fraction range is $0 \leq \mathbf{fraction} < 0$,
- **base** is usually 2 or 10,
- **exponent** and **bias** are integers.

The **fraction** is referred also as mantissa, magnitude or significand. The **base** is referred to as the radix. The **exponent** is sometimes referred to as the characteristic. The fraction is normalized in such way that the leading bit is a one. Following these nomenclature variations, Eq. (2.6) can be also written as

$$\begin{aligned} \mathcal{N}_{fln} &= \mathbf{sign} \times \mathbf{mantissa} \times (\mathbf{radix}^{\mathbf{exponent-bias}}) \\ \mathcal{N}_{fln} &= \mathbf{sign} \times \mathbf{significand} \times (\mathbf{radix}^{\mathbf{exponent-bias}}) \end{aligned} \tag{2.7}$$

This standard sorts out three important requirements for machines:

- consistency of floating point number representation across the machine,
- correctness of number rounding,
- consistency and stability of exceptional situations treatment, eg. overflow problems (division by zero) etc.

Three precision standards are defined in IEEE FPS: single, double, and quad as can be seen in Table (2.7).

Table (2.7)

Precision standard type	Length of bit word	length of bit-fields
Single	32-bits	sign (1) | exponent (8) | significand (23)
Double	64-bits	sign (1) | exponent (11) | significand (52)
Quadruple	127-bit	sign (1) | exponent (15) | significand (113)

The exponent is biased [[13]], i.e. the stored exponent value is offset from the actual value by the exponent bias. Biasing is done because exponents have to be signed values and represent both small and very large values. The exponent is biased before being stored. The bias is subtracted to retrieve the actual exponent, as can be seen in Table (2.8).

Table (2.8)

Precision standard	Exponent range	Biased by adding	To get a value in range	Values of special meaning
Single	$-126 \cdots + 127$	127	$1 \ldots 254$	0 and 255
Double	$-1022 \cdots + 1023$	1023	$1 \ldots 2046$	0 and 2047
Quadruple	$-16382 \cdots + 16383$	16383	$1 \ldots 32766$	0 and 32767

Two definitions can be formulated for all precision standards:

- **Precision**: The number of bits in the significand, including the hidden bit, is called the precision of the floating point system and is denoted by p (see Table (2.9)).
- **Machine epsilon**: The gap between the number 1 and the next larger floating point number is called the machine epsilon of the floating point system and is denoted by ϵ (see Table (2.9)).

Table (2.9)

Precision standard	Precision p	Machine precision ϵ
Single	24	$2^{-23} \approx 1.2 \times 10^{-7}$
Double	53	$2^{-52} \approx 2.2 \times 10^{-16}$
Quad	114	$2^{-113} \approx$???

2.3.1 Single precision

In the single precision representation the significand stores 23 bits in the order: $b_1 b_2 \dots b_{23}$. The bit b_0, which is always 1, is not stored following the idea called the hidden bit normalization. Without b_0, the bit-string is the fractional part of the significand field and is also refereed to as the fractional field.

IEEE single precision number is called a floating point number when it can be stored exactly as follows from Table (2.10).

Table (2.10)

$\pm$	$a_1 a_2 a_3 \dots a_8$	$b_1 b_2 b_3 \dots b_{22} b_{23}$

If exponent		$a_1 \dots a_8$ is	Then value is		
$(00000000)_2$	=	$(0)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{-126}
$(00000001)_2$	=	$(1)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{-126}
$(00000010)_2$	=	$(2)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{-125}
$(00000011)_2$	=	$(3)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{-124}
	$\vdots$			$\vdots$	
$(01111111)_2$	=	$(127)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{0}
$(10000000)_2$	=	$(128)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{1}
	$\vdots$			$\vdots$	
$(11111100)_2$	=	$(252)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{125}
$(11111101)_2$	=	$(253)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{126}
$(11111110)_2$	=	$(254)_{10}$	$\pm(0.b_1 \dots b_{23})_2$	$\times$	2^{127}
$(11111111)_2$	=	$(255)_{10}$	$\pm\infty$ if $b_1, \dots, b_{23}$ = 0 NaN othervise		

The symbol NaN means: Not a Number. The first and the last lines refer to special numbers. The first line shows zero which requires a zero bit-string both for the exponent field and for the fraction, as can be seen below:

0	00000000	00000000000000000000

We should remember that, in the first line, the initial unstored bit is 0.

The last line is the special pattern for $\pm\infty$ or NaN and shows the exponent bit-string that contains all ones. Other numbers refer to the normalized numbers.

We have also so called subnormal numbers. Such numbers have zero exponent and nonzero fraction. Using subnormal numbers we can represent numbers smaller than the smallest normalized positive number, eg. $2^{-127} = (0.1)_2 \times 2^{-126}$ which is expressed by

0	00000000	10000000000000000000

The smallest number that can be stored as the single precision floating point number is $2^{-149} = (0.000\ldots01)_2 \times 2^{-126}$

0	00000000	00000000000000000001

Subnormal numbers cannot be normalized because the normalization is resulting in exponents which do not fit. They are also less accurate because they have less room for nonzero bits in the fraction, eg. $\frac{1}{10} \times 2^{-133} = (0.11001100)_2 \times 2^{-136}$ that is represented by

0	00000000	00000000011001100011001

2.3.2 Double precision

The name comes from the fact that the double precision number uses twice more bits as the single precision floating point number. The extra bits increase also the magnitude that can be represented by floating point format. The double precision floating point number is encoded in a 64 bits or 8 bytes by using 64-bit double word. The double precision number is shown in Table (2.11).

Table (2.11)

$\pm$	$a_1a_2a_3\ldots a_{11}$	$b_1b_2b_3\ldots b_{51}b_{52}$

If exponent		$a_1 \dots a_{11}$ is	Then value is		
$(000\dots000)_2$	=	$(0)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{-1022}
$(000\dots001)_2$	=	$(1)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{-1022}
$(000\dots010)_2$	=	$(2)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{-1021}
$(000\dots011)_2$	=	$(3)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{-1020}
	$\vdots$			$\vdots$	
$(011\dots111)_2$	=	$(1023)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{0}
$(100\dots000)_2$	=	$(1024)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{1}
	$\vdots$			$\vdots$	
$(111\dots100)_2$	=	$(2044)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{1021}
$(111\dots101)_2$	=	$(2045)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{1022}
$(111\dots110)_2$	=	$(2046)_{10}$	$\pm(0.b_1\dots b_{52})_2$	$\times$	2^{1023}
$(111\dots111)_2$	=	$(2047)_{10}$	$\pm\infty$ if $b_1, \dots, b_{52}$ = 0 NaN othervise		

2.4 Floating-point operations

A complying computer system, following the IEEE standard, must support the following floating-point operations:

Operation	Description
Addition	Algebraic addition
Subtraction	Algebraic subtraction
Multiplication	Algebraic multiplication
Division	Algebraic division
Comparison	There are four possible relations between any two floating-point values: less than,equal, greater than, and unordered. The unordered relation occurs when one or both of the operands is a NaN (Not a Number).
Square root	The square root operation never overflows or underflows
Conversion	The following conversions must be supported by a conforming implementation, if the implementation supports single-precision, double precision, and quad-precision formats: • Single precision to double precision • Single-precision to quad-precision • Double-precision to single-precision • Double-precision to quad-precision
	• Quad-precision to single-precision • Quad-precision to double-precision • Floating-point to integer • Integer to floating point • Binary floating-point to decimal • Decimal to binary floating-point
Round to nearest integral value	Rounds an argument to the nearest integral value (in floating-point format) based on the current rounding mode.
Remainder	The remainder operation takes two arguments, x and y and is defined as $x - y \times n$, where n is the integer nearest the exact value $\frac{x}{y}$

2.4.1 Rounded arithmetic

This section presents only some headlines of rounded mathematics for floating-point numbers. For more advanced study the following papers are recommended [18] [25], [33]. Floating point numbers (FPN) form a finite set and include:

- ± 0,
- normalized and subnormal numbers,

- $\pm\infty$.

One should notice that NaN is not included. A real number $r \in \mathcal{R}$ is either in the normalized range if $N_{min} \leq |r| \leq N_{max}$, or is in the subnormal range if $S_{min} \leq |r| \leq S_{max}$, where

N_{min}	the minimum positive normalized FPN,
N_{max}	the maximum positive normalized FPN,
S_{min}	the minimum positive subnormal FPN,
S_{max}	the maximum positive subnormal FPN,

The IEEE standard defines correctly rounded value of r by a function $round(r)$ and if r is FPN then $round(r) = r$. For other numbers $round(r)$ depends on the rounding mode:

Rounding mode	Value of $round(r)$
Round down	$round(r) = r_-$
Round up	$round(r) = r_+$
Round towards zero	$round(r)$ is either r_- or r_+ whichever is between zero and r
Round to nearest	$round(r)$ is either r_- or r_+ whichever is nearer to r

where r_- is the closest FPN less than r and r_+ is the closest FPN greater than r.

Modes: "Round down" and "Round up" for a positive r have the same effect because r_- is between zero and r. The mode: "Round towards zero" requires truncating the binary expression, i.e. discarding bits. The mode: "Round" without qualification usually means: "Round to nearest".

Absolute and relative rounding errors

The absolute rounding error (ARE) associated with r reads as

$$ARE = |round(r) - r|$$

and depends on a rounding mode, and for all modes it fulfills the condition

$$|round(r) - r| < |r_+ - r_-|. \tag{2.8}$$

The last relation can be derived considering the single precision FPN and expressed in terms of the machine precision ϵ. To this end, we assume r from the normalized range, i.e. $N_{min} \leq r \leq N_{max}$, so that $r = (b_0 b_1 b_2 \dots b_{23} b_{24} b_{25} \dots)_2 \times$

$2^E, b_0 = 1$. Taking representations r_- and r_+

$$\begin{aligned} r_- &= (b_0 b_1 b_2 \dots b_{23})_2 \times 2^E \\ r_+ &= r_- + (0.000 \dots 001)_2 \times 2^E \end{aligned}$$

and subsequently substituting them to [2.8], leads to the required condition

$$|round(r) - r| < 2^{-23} \times 2^E$$

that can be generalized by the relationship

$$|round(r) - r| < \epsilon \times 2^E. \tag{2.9}$$

For the mode: "Round to nearest", the ARE is no more than half the gap between r_- and r_+ and is reads as

$$ARE \leq 2^{-(m+1)} \times 2^E. \tag{2.10}$$

The relative rounding error (RRE) for $r \neq 0$ is the absolute error divided by the magnitude of the number which is approximated

$$RRE \equiv \delta = \frac{|round(r) - r|}{|r|} \tag{2.11}$$

It is known that $|round(r) - r| \leq 2^{-m} \times 2^E$ and from the binary expansion $r = (1.b_1 b_2 b_3 \dots b_{m-1} b_m b_{m+1} \dots)_2 \times 2^E$ the following inequality can be obtained $|r \geq 2^E|$. Hence the RRE is

$$RRE = \frac{|round(r) - r|}{|r|} \leq \frac{2^{-m} \times 2^E}{2^E} = 2^{-m} \tag{2.12}$$

The RRE for the mode "Round to nearest" is half of this.

The inequality $|r| \geq 2^E$ becomes an equality when all the bits $b_1 b_2 b_3 \dots b_m$ are 0. When all the bits are 1, the absolute value of r is very nearly twice as large, and ARE and RRE are almost halved.

2.4.2 Arithmetic of floating-point numbers

Arithmetic operations on floating-point numbers consists of addition, subtraction, multiplication and division.

Addition

The example on floating-point numbers addition is given in binary representations. Adding two numbers: $r_1 = m \times 2^E$ and $r_2 = p \times 2^F$, can be done in four steps:

- Shift one significand to express both numbers with the same exponent. The exponent for both FLNs is evaluated as $G = max(E, F)$.
- Add significands,
- Normalize the result if necessary,
- Round the result - if necessary.

Example (2.13)♣

Following [14] add two FPNs: r_1 and r_2, given in binary representation:

$$\begin{array}{lllll} r_1 = 0.25 & = 0 & 01111101 & 00000000000000000000000 \\ r_2 = 100 & = 0 & 10000101 & 10010000000000000000000 \end{array}$$

- Step 1: Align radix points by shifting the mantissa of 0.25 right by 8 places, i.e. shifting 01111101 by 10000101

$$\begin{array}{ll} + & 10000101 \\ - & 01111101 \\ \hline & 00001000 \end{array}$$

to get:

$$0.25 \quad = 0 \quad 10000101 \quad 00000001000000000000000$$

- Step 2: Add but do not forget the hidden bit for 100

$$\begin{array}{llll} (0.25) & 0 & 10000101 & 0.00000001000000000000000 \\ (100) & 0 & 10000101 & 1.10010000000000000000000 \\ \hline & 0 & 10000101 & 1.10010001000000000000000 \end{array}$$

- Step 3: Normalize the result, i.e. get the hidden bit to be 1. It is not necessary as 1 is already in the place. So, the final result is

0	10000101	1.10010001000000000000000

♠

Subtraction

The subtraction can also be done in four steps:

- Shift one significand to express both numbers with the same exponent. The exponent for both FLNs is evaluated as $G = max(E, F)$.
- Subtract significands,
- Normalize the result if necessary,
- Round the result - if necessary.

The following example illustrates some difficulty that can appear when subtracting two correctly rounded FPNs.

Example (2.14)♣

Subtract two positive FLNs: $r_1 = (1.0)_2 \times 2^0$ and $r_2 = (1.111\ldots111)_2 \times 2^{-1}$. After aligning significands, we get

	0	00000001	1.00000000000000000000000	(r_1)
-	0	00000001	0.11111111111111111111111|1	(r_2)
	0	00000001	0.00000000000000000000000|1	

♠

The final result is the FPN, $(1.0)_2 \times 2^{-24}$, obtained with carrying out the subtraction by using an extra bit, so called a guard bit. It highlights the problem, that FLNs with 24-bit significands does not guarantee correctly rounded subtraction when using the mode: "Round to nearest". Machines with correctly implemented arithmetics achieve correctly rounding results for all rounding modes using two guard bits and the sticky bit, in addition. The sticky bit is used to flag the above mentioned rounding problem.

Multiplication and division

Multiplication of two FLNs: $r_1 = m \times 2^E$, and $r_2 = p \times 2^F$, can be done in three steps:

- Multiply the significands $(m \times p)$,
- Add the exponents $(E + F)$,
- Normalize and correctly round the result.

Division of two FPNs requires taking the quotient of the significands and the difference of exponents.

Example (2.15)♣

Following [14], multiply two FLNs expressed by binary representations. To make the multiplication less tedious, take mantissas represented by four bits only.

$$\begin{array}{rccc} & 0 & 10000100 & 0100 \\ \times & 1 & 00111100 & 1100 \\ \hline \end{array}$$

- Step1: Multiply mantissas with hidden bits

$$\begin{array}{cccccccccc} & & & & & 1. & 0 & 1 & 0 & 0 \\ & & & & \times & 1. & 1 & 1 & 0 & 0 \\ \hline & & & & & 0 & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 & 0 & 0 & \\ & & & 1 & 0 & 1 & 0 & 0 & & \\ & & 1 & 0 & 1 & 0 & 0 & & & \\ & 1 & 0 & 1 & 0 & 0 & & & & \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array} \tag{2.13}$$

  that finally becomes

$$1 \quad 0. \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0$$

- Add true exponents avoiding twice adding of the bias

$$\begin{array}{cccccccccl} & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \\ - & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\ \hline & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \text{true exponent is 5} \end{array} \tag{2.14}$$

$$\begin{array}{cccccccccl} & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\ - & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & \\ \hline & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & \text{true exponent is -67} \end{array} \tag{2.15}$$

- Evaluate the true exponent $5 + (-67) = -62$,
- Re-bias the exponent $-62 + 127 = 65$, which has the binary representation: 01000001,
- Finally write the result as

1	01000001	10.00110000

♠

Chapter 3

Interpolation and extrapolation

Interpolation [31] is the process of fitting specified points/values $\{x_i, f(x_i) = y_i\}, i = 1, ..., n$ where $x_i \in (a, b)$, by a function, usually polynomial, $p(x) \in P(x)$. The purpose of the numerical interpolation is to

- Compute intermediate values of the sampled function $p(x) \approx f(x), x \in [x_0, x_n]$ for a given set of values $f(x_0) = f_0, f(x_1) = f_1, \ldots, f(x_n) = f_n$,
- Numerical differentiation, that is the foundation for finite difference and finite element method,
- Numerical integration.

The most frequently used interpolation methods [43] are the following:

- Linear interpolation,
- Polynomial interpolation,
- Spline interpolation,
- Nyquist-Shanon interpolation used when the number of data points is infinite,
- Hermite interpolation used when values and derivative of the function are known,

Extrapolation is the process of constructing new data points outside or between a given discrete set of data points. There are four extrapolation methods:

- Linear extrapolation: creation of tangent line from the known data and extending it to the next point or beyond the limit of known data,
- Extrapolation with known curvature: the curvature of extension beyond the known data can also be maintained,
- Conic extrapolation: can be created through five points near the end of known data,
- Polynomial extrapolation: polynomial curve can be created through all given data or only data near the end of limit and then can be extended further.

3.1 Polynomial interpolation

Many various formulas for polynomials are known but all of them are based on the same idea: fitting a set of discrete values

x	x_0	x_1	$\dots$	x_n
y	y_0	y_1	$\dots$	y_n

assuming that the data are obtained from the evaluation of a smooth function. Elements of the **x**-row are called nodes. The polynomial $p(x)$ takes the given values at the given points, i.e. $p(x_i) = y_i$ with $0 \leq i \leq n$. The polynomial is interpolating the above table, since the function is unknown. The interpolating polynomial can be written in several forms but the most popular are the Lagrange form and the Newton form.

3.1.1 Liner interpolation

The linear interpolation polynomial is defined by

$$p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_1 - x_0) \tag{3.1}$$

where $x_0, f(x_0)$ and $x_1, f(x_1)$ are known coordinates of two points. The difference between $f(x)$ and $p(x)$ is the approximation error defined by

$$E_{int} = f(x) - p(x). \tag{3.2}$$

As results from the Rolle's theorem, when $f(x) \in C^2$, the error E_{int} is bounded by

$$|E_{int}| \leq \frac{(x_1 - x_0)^2}{8} \max_{x_0 \leq x \leq x_1} |f^{"}| \tag{3.3}$$

3.1.2 Interpolating polynomial in Lagrange's form

So called the Lagrange interpolating polynomial (LIP), originally published by Waring [53] 16 years before Lagrange, is defined for a given set of n points $\{[x_1, y_1 = f(x_1)], [x_2, y_2 = f(x_2)], \ldots, [x_{n-1} = f(x_{n-1}]\}$ is the polynomial $p(x)$ of degree $(n-1)$ that passes through all n points. The set of cardinal functions $l_0, l_1, l_2, \ldots, l_n$ corresponding to the set of nodes $\{x_0, x_1, x_2, \ldots, x_n\}$ are defined as

$$l_i(x_j) = \delta_{ij} \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

Any function $f(x)$ can be interpolated by the Lagrange form of the interpolating polynomial of degree: deg $\leq n$

$$p_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i) \tag{3.4}$$

where $l_i(x)$ is of the order n. The cardinal function is given by

$$l_i(x) = \prod_{j \neq i, j=0}^{n} \frac{x - x_j}{x_i - x_j} \quad \text{with} \quad 0 \leq i \leq n \tag{3.5}$$

that explicitly is written as

$$l_i(x) \;\; = \left(\tfrac{x-x_0}{x_i-x_0}\right)\left(\tfrac{x-x_1}{x_i-x_1}\right)\ldots\left(\tfrac{x-x_{i-1}}{x_i-x_{i-1}}\right)\left(\tfrac{x-x_{i+1}}{x_i-x_{i+1}}\right)\ldots\left(\tfrac{x-x_n}{x_i-x_n}\right) \tag{3.6}$$

Equation 3.4 can be written explicitly as

$$\begin{aligned} p_n(x) \;\; &= \;\; f(x_0)\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)} \\ &+ \;\; f(x_1)\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)} \\ &+ \;\; f(x_2)\frac{(x-x_1)(x-x_3)\ldots(x-x_n)}{(x_2-x_0)(x_2-x_1)\ldots(x_2-x_n)} \\ &+ \;\; \ldots \\ &+ \;\; f(x_n)\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_n-x_0)(x_n-x_1)\ldots(x_n-x_{n-1})} \end{aligned} \tag{3.7}$$

The polynomial fulfils the so called "point exact" condition:

$$p_n(x_j) = \sum_{i=1}^{n} l_i(x_j) f(x_i) = l_j(x_j) f(x_j) = f(x_j) \equiv y_j \tag{3.8}$$

that can be easily verified, for the case with $n = 2$, that $p(x)$ fits perfectly all points $(x_i, y_i), i = 0, 1, 2$:

$$\begin{aligned}
p_2(x_0) &= y_0\frac{(x_0-x_1)(x_0-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1\frac{(x_0-x_0)(x_0-x_2)}{(x_1-x_0)(x_1-x_2)} \\
&+ y_2\frac{(x_0-x_0)(x_0-x_1)}{(x_2-x_0)(x_2-x_1)} = y_0 \\
p_2(x_1) &= y_0\frac{(x_1-x_1)(x_1-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1\frac{(x_1-x_0)(x_1-x_2)}{(x_1-x_0)(x_1-x_2)} \\
&+ y_2\frac{(x_1-x_0)(x_1-x_1)}{(x_2-x_0)(x_2-x_1)} = y_1 \\
p_2(x_2) &= y_0\frac{(x_2-x_1)(x_2-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1\frac{(x_2-x_0)(x_2-x_2)}{(x_1-x_0)(x_1-x_2)} \\
&+ y_2\frac{(x_2-x_0)(x_2-x_1)}{(x_2-x_0)(x_2-x_1)} = y_2
\end{aligned} \tag{3.9}$$

The data points fitting by LIP is perfect but the smoothness of interpolating polynomial goes worse for a higher degree $p(x)$. This smoothness/polynomial degree challenge can be seen in the following table:

more		higher		greater
data	$\Rightarrow$	degree of	$\Rightarrow$	oscillation
points		$p(x)$		between points

when constructing LIPs.

The existence of LIP is always guaranteed and it can be constructed for any table of data. Unfortunately, the evaluation of LIP is costly and tedious and this is why the Newton's method of a polynomial evaluation is preferred.

3.1.3 Interpolating polynomial in Newton's form

The main advantage of this interpolating method is the recurrence formula for constructing of a higher degree polynomial $p_{j+1}(x)$ on the basis of his predecessor $p_j(x)$ and the additional data point. Adding the new node (x_{j+1}, y_{j+1}) we can write the new interpolating polynomial

$$\begin{aligned}
p_{j+1}(x) &= p_j(x) + a(x-x_0)(x-x_1)\dots(x-x_{j-1})(x-x_j), \\
p_{j+1}(x_i) &= y_i \quad \text{for} \quad 0 \le i \le (j+1).
\end{aligned} \tag{3.10}$$

where a is undetermined polynomial coefficient that can be evaluated following the procedure:

- using Eq. (3.10), the equality $p_{j+1}(x_{j+1}) = y_{j+1}$, can be written as $p_j(x_{j+1}) + a(x_{j+1} - x_0)(x_{j+1} - x_1)\dots(x_{j+1} - x_j) = y_{j+1}$,
- and solved for $a = \frac{y_{j+1} - p_j(x_{j+1})}{(x_{j+1}-x_0)(x_{j+1}-x_1)\dots(x_{j+1}-x_j)}$,
- the above can be done assuming that all nodes $\{x_0, x_1, \dots, x_j, x_{j+1}\}$ are distinct.

The new polynomial is unique and it can be proved quite easily. Let $p_j(x)$ and $\pi_j(x)$ be interpolating polynomials constructed for the same data points $\{(x_0, y_0), (x_1, y_1), \dots, (x_j, y_j)\}$ for $0 \leq j \leq n$ such that $p_j(x_i) = \pi_j(x_i) = y_i$. Then the residual polynomial $r_j(x) = p_j(x) - \pi_j(x)$ of degree at most n is zero at all nodes $\{x_0, x_1, \dots, x_n\}$. The polynomial $r_n(x)$ has at most n roots $r_n(x) = 0$ that implies $p_n(x) - \pi_n(x) = 0$ and hence $p_n(x) = \pi_n(x)$. Therefore, the interpolating polynomial $p_n(x)$ is unique.

The above presented procedure for constructing $p_{j+1}(x)$ can be initiated from $p_1(x)$ and the interpolating polynomial can be written in one of the following forms:

- nested form

$$\begin{aligned} p_n(x) &= a_0 + a_1[x - x_0] + a_2[(x - x_0)(x - x_1)] \\ &+ a_3[(x - x_0)(x - x_1)(x - x_2)] \\ &+ \dots \\ &+ a_n[(x - x_0)(x - x_1)\dots(x - x_{n-1})] \end{aligned}$$

- with product notation

$$p_n(x) = a_0 + \sum_{i=1}^{n} a_i \left[\prod_{j=1}^{i-1} (x - x_j) \right]$$

- nested form after successive factorization

$$p_n(x) = \left(\dots \left[\left[a_n(x - x_{n-1}) + a_{n-1} \right] (x - x_{n-2}) + a_{n-2} \right] \dots \right) (x - x_0) + a_0$$

The coefficients a_i of $p_n(x)$ can be evaluated using the "ideal polynomial fit" concept, i.e. $p_n(x)$ is passing through each data point and $p_n(x_i) = f(x_i) = y_i$. This leads to the system of simultaneous equations:

$$\begin{aligned} y_0 &= a_0, \\ y_1 &= a_0 + a_1(x_2 - x_1) \\ y_2 &= a_0 + a_1(x_3 - x_1) + a_2(x_3 - x_1)(x_3 - x_2) \\ &\vdots \\ y_n &= a_0 + a_1(x_n - x_1) + \cdots + a_n(x_n - x_1)(x_n - x_2)\ldots(x_n - x_{n-1}) \end{aligned} \tag{3.11}$$

Introducing the concept of divided differences (DD) [3]

$$\begin{aligned} \delta_i^{(1)} &= \frac{y_i - y_0}{x_i - x_1}, \\ \delta_i^{(2)} &= \frac{\delta_i^{(1)} - \delta_2^{(1)}}{x_i - x_2}, \\ \delta_i^{(3)} &= \frac{\delta_i^{(2)} - \delta_3^{(2)}}{x_i - x_3}, \\ &\vdots \\ \delta_i^{(j)} &= \frac{\delta_i^{(j-1)} - \delta_j^{(j-1)}}{x_i - x_j}, \end{aligned} \tag{3.12}$$

the solution of Eq. (3.11) in terms of DD is

$$\begin{aligned} a_0 &= y_0, \\ a_1 &= \delta_1^{(1)}, \\ a_2 &= \delta_2^{(2)}, \\ &\vdots \\ a_n &= \delta_n^{(n)}. \end{aligned}$$

It is much more convenient to work with DD using tables following the example with n=5

x_0	y_0					
x_1	y_1	$\delta_1^{(1)}$				
			$\delta_2^{(2)}$			
x_2	y_2	$\delta_2^{(1)}$		$\delta_3^{(3)}$		
			$\delta_3^{(2)}$		$\delta_4^{(4)}$	
x_3	y_3	$\delta_3^{(1)}$		$\delta_4^{(3)}$		$\delta_5^{(5)}$
			$\delta_4^{(2)}$		$\delta_5^{(4)}$	
x_4	y_4	$\delta_4^{(1)}$		$\delta_5^{(3)}$		
			$\delta_5^{(2)}$			
x_5	y_5	$\delta_5^{(1)}$				

It can be written in more compact form

x_0	y_0					
x_1	y_1	$\delta_1^{(1)}$				
x_2	y_2	$\delta_2^{(1)}$	$\delta_2^{(2)}$			
x_3	y_3	$\delta_3^{(1)}$	$\delta_3^{(2)}$	$\delta_3^{(3)}$		
x_4	y_4	$\delta_4^{(1)}$	$\delta_4^{(2)}$	$\delta_4^{(3)}$	$\delta_4^{(4)}$	
x_5	y_5	$\delta_5^{(1)}$	$\delta_5^{(2)}$	$\delta_5^{(3)}$	$\delta_5^{(4)}$	$\delta_5^{(5)}$

where, unfortunately, the "influence zone" for the higher order difference is not so clear that in the first table.

We can easily notice that the terms situated on the upper diagonal side of the triangle in the above array are the coefficients of the polynomial $p_5(x)$.

The very important thing is to highlight the observation that DDs are invariant under all permutations of nodes x_{i-1}, x_i, x_{i+1}. It is issued from the observation that $\delta^{(k)}$ is equal to the coefficient of the term x^k of the polynomial with deg $\leq k$ that interpolates a table with the node order $\{x_0, x_1, x_2, \ldots, x_k\}$. Because the same polynomial can be constructed on another permutation of nodes, eg. $\{x_1, x_0, x_2, \ldots, x_k\}$, the coefficients of the same term x_k must be equal.

Example (3.1)♣

For the given data find DD

y	0.54030	0.45360	0.36236	0.26750	0.16997
x	1.0	1.1	1.2	1.3	1.4

Using Eq. (3.12), the DD are evaluated as follows

j	x_j	y_j	$\delta_j^{(1)}$	$\delta_j^{(2)}$	$\delta_j^{(3)}$	$\delta_j^{(4)}$
0	1.0	0.54030	-0.8670	-0.2270	0.1533	0.0125
1	1.1	0.45360	-0.9124	-0.1810	0.1583	
2	1.2	0.36236	-0.9486	-0.1335		
3	1.3	0.26750	-0.9753			
4	1.4	0.16997				

♠

At the end of this section we should also mention the so called: "inverse interpolation", i.e. evaluation of the inverse of a function $y = f(x)$. An interpolating polynomial

$$p(y) = \sum_{i=0}^{n} c_i \prod_{k=0}^{i-1} (y - y_k), \tag{3.13}$$

such that $p(y_i) = x_i$, can be constructed for given table

y	y_0	y_1	y_2	$\ldots$	y_n
x	x_0	x_1	x_2	$\ldots$	x_n

This IP can be also used to find the approximate location of a root of $f(x)$ as can be seen in the following example.

Example (3.2)♣

Given a table

y	0.585	0.347	0.183	0.042	-0.089
x	2.0	3.0	4.0	5.0	6.0

find the approximate location of a root of a function $f(x)$ The answer is: a root is in the subinterval $[5.0, 6.0]$. ♠

3.1.4 Neville's algorithm

The difference between Newton's method described in the previous subsection and Neville's method lies in the number of steps in evaluation of the polynomial. Newton's method involves two steps:

- computation of IP coefficients,
- evaluation of the polynomial,

but Neville's algorithm computes the interpolant in one step when only one point has to be interpolated. Neville's algorithm is a recursive way to construct a polynomial of order $(n-1)$ from n known function values. The

procedure for construction of IP for $(n-1)$ pairs of (x_i, y_i) consists of the following steps:

- find the n polynomials of order zero fitting
  - the n function values $\{y_0, y_1, y_2, \ldots, y_{n-1}\}$,
  - at nodes $\{x_0, x_1, x_2, \ldots, x_{n-1}\}$
- construct the $(n-1)$ polynomials of order one fitting all pairs (x_i, y_i) and (x_{i+1}, y_{i+1}),
- construct the $(n-2)$ polynomials of order two fitting all triplets (x_i, y_i), (x_{i+1}, y_{i+1}), and (x_{i+2}, y_{i+2}),
- continue up to the construction of a single polynomial of order $(n-1)$ fitting all given points $[(x_i, y_i)$ where $i \in [0, n-1]]$.

The recursive relation for the evaluation of a higher degree IP at the requested interpolation point x on the basis of two lower degree IP given at the same point is written as

$$\begin{aligned} \delta p_{[i(i+1)\ldots(i+m)]} &= \left(\frac{x - x_{i+m}}{x_i - x_{i+m}}\right) \delta p_{[i(i+1)\ldots(i+m-1)]} \\ &+ \left(\frac{x_i - x}{x_i - x_{i+m}}\right) \delta p_{[(i+1)(i+2)\ldots(i+m)]} \end{aligned} \tag{3.14}$$

The Neville's algorithm generates a table of the pyramidal form

$$\begin{array}{lllllll} x_0 & y_0 = \delta p_{[0]} & & & & & \\ & & \delta p_{[01]} & & & & \\ x_1 & y_1 = \delta p_{[1]} & & \delta p_{[012]} & & & \\ & & \delta p_{[12]} & & \delta p_{[0123]} & & \\ x_2 & y_2 = \delta p_{[2]} & & \delta p_{[123]} & & \delta p_{[01234]} & \\ & & \delta p_{[23]} & & \delta p_{[1234]} & & \delta p_{[012345]} \\ x_3 & y_3 = \delta p_{[3]} & & \delta p_{[234]} & & \delta p_{[12345]} & \\ & & \delta p_{[34]} & & \delta p_{[2345]} & & \\ x_4 & y_4 = \delta p_{[4]} & & \delta p_{[345]} & & & \\ & & \delta p_{[45]} & & & & \\ x_5 & y_5 = \delta p_{[5]} & & & & & \end{array}$$

Example (3.2) [60] ♣

Given values of a function $f(x)$ in the second column, find its interpolation at two points: at first at $x = 0.15$ and afterward at $x = 0.25$.

Neville's algorithm gives the following table:

0.1	−1.6228				
		−1.22230			
0.2	−0.8218		−1.18706		
		−1.08135		−1.17642	
0.3	−0.3027		−1.12320		−1.17186
		−0.91395		−1.13992	
0.4	0.1048		−1.02289		
		−0.76870			
0.5	0.4542				

The answer is

- $f(0.15) = -1.2223$, that is given in the third column. Other values given in the third column of the table are linear extrapolations of $f(x)$ back to $x = 0.15$ from interpolations on the other intervals.
- the first choice is $f(0.25) = -1.0813$ and the second one is taken from the five column $f(0.25) = -1.1764$.

♠

3.1.5 Hermite's interpolating polynomials

Hermite interpolation is closely related to Newton's divided difference method. It allows us to construct the Hermit interpolating polynomial (HIP) that not only fits the data points $\{x_i, y_i\}$ for $i \in [0, n-1]$, but also has specified derivatives at every node. The IP, that fulfils the above requirements, is called the Hermite IP or the osculating polynomial.

HIP as the generalization of LIP

For simplicity let us start with a third-order (cubic) polynomial

$$h(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \tag{3.15}$$

with four coefficients that fits two points $\{(x_0, y_0), (x_1, y_1)\}$ and has the specified first derivatives $y_0^{'}, y_1^{'}$ at nodes $\{x_0, x_1\}$. The four coefficients a_0, a_1, a_2, a_3, called also constraints of the interpolation, can be calculated by solving four

equations:

$$\begin{aligned} h(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 = y_0 \\ h(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 = y_1 \\ h^{'}(x_0) &= a_1 + 2a_2 x_0 + 3a_3 x_0^2 = y_0^{'} \\ h^{'}(x_1) &= a_1 + 2a_2 x_1 + 3a_3 x_1^2 = y_1^{'} \end{aligned} \tag{3.16}$$

The above procedure can be generalized for Hermite interpolating polynomials up to m degree and $n+1$ nodes. To this end we need to set up a general polynomial which is of degree $[(m+1)(n+1)-1]$ with $[(m+1)(n+1)]$ unknowns (constraints):

$$h(x) = \sum_{i=0}^{[(m+1)(n+1)-1]} a_i x^i \tag{3.17}$$

From the $[(m+1)(n+1)-1]$ requirements

$$\begin{aligned} h(x_i) &= y_i, \quad i = 0, \ldots, n \\ h^{'}(x_i) &= y_i^{'}, \quad i = 0, \ldots, n \\ h^{"}(x_i) &= y_i^{"}, \quad i = 0, \ldots, n \\ h^{(3')}(x_i) &= y_i^{(3')}, \quad i = 0, \ldots, n \\ &\vdots \\ h^{(m')}(x_i) &= y_i^{(m')}, \quad i = 0, \ldots, n \end{aligned} \tag{3.18}$$

we can evaluate $[(m+1)(n+1)-1]$ coefficients a_i of the HIP. Note that LIP is a special case of HIP with $m = 0$, i.e. only function values $y_i = f(x_i)$ are considered in the LIP construction.

Alternative method for setting HIP

This alternative method for setting of HIP consists of the identification of basis functions. A basis function is associated with each value of $y_i = f(x_i)$ and various derivative values at each node (data point).

To explain the method step by step, we can redo the example with the cubic Hermite interpolation. Assume now that the HIP passes through the points $\{(x_0 = 0, y_0), (x_1 = 1, y_1)\}$. We require a third degree polynomial

$$\begin{aligned} h(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3, \\ h^{'}(x) &= a_1 + 2a_2 x + 3a_3 x^2 \end{aligned} \tag{3.19}$$

that in nodes $x = 0$ and $x = 1$ fulfils the requirements

$$\begin{aligned} h(0) &= y_0, \\ h(1) &= y_1, \\ h^{'}(0) &= y_0^{'}, \\ h^{'}(1) &= y_1^{'}, \end{aligned}$$

from where the following set of equations can be derived

$$\begin{aligned} y_0 &= a_0, \\ y_1 &= a_0 + a_1 + a_2 + a_3, \\ y_0^{'} &= a_1, \\ y_1^{'} &= a_1 + 2a_2 + 3a_3, \end{aligned}$$

and written in the matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_0^{'} \\ y_1^{'} \end{bmatrix} \tag{3.20}$$

Solving this system for a_0, a_1, a_2, a_3 we obtain

$$\begin{aligned} a_0 &= y_0, \\ a_1 &= y_0^{'}, \\ a_2 &= 3y_1 - 3y_0 - y_1^{'} - 2y_0^{'}, \\ a_3 &= -2y_1 + 2y_0 + y_0^{'} + y_1^{'}. \end{aligned} \tag{3.21}$$

Substituting Eq. (3.21) into the first equation of 3.19, the HIP can be expressed as

$$\begin{aligned} h(x) &= y_0 + y_0^{'}x + (3y_1 - 3y_0 - y_1^{'} - 2y_0^{'})x^2 \\ &+ (-2y_1 + 2y_0 + y_0^{'} + y_1^{'})x^3. \end{aligned} \tag{3.22}$$

Factoring out the functional and derivative values Eq. (3.22) can be rewritten in the "generic" form

$$h(x) = y_0\alpha_0(x) + y_1\alpha_1(x) + y_0^{'}\beta_0(x) + y_1^{'}\beta_1(x), \tag{3.23}$$

with the following third degree polynomials as **basis functions** (BF)

$$\begin{aligned}\alpha_0(x) &= 2x^3 - 3x^2 + 1,\\ \alpha_1(x) &= -2x^3 + 3x^2,\\ \beta_0(x) &= x^3 - 2x^2 + x,\\ \beta_1(x) &= x^3 - x^2.\end{aligned}$$

The first and the third BF are associated with HIP and its first derivative, correspondingly, at the node x_0. The second and the fourth BF are associated with HIP and its first derivative at the node x_1. The basis functions for HIP at nodal points assume values given in the table of constraints

x	$\alpha_0(x)$	$\alpha_1(x)$	$\beta_0(x)$	$\beta_1(x)$
$x_0 = 0$	1	0	0	0
$x_1 = 1$	0	1	0	0

Following the same procedure as for HIP, the first derivative of HIP can be derived in terms of basis functions

$$h^{'} = y_0\alpha_0^{'}(x) + y_1\alpha_1^{'}(x) + y_0^{'}\beta_0^{'}(x) + y_1^{'}\beta_1^{'}(x). \tag{3.24}$$

The basic functions for the first derivative of HIP at nodes assume values given in the table of constraints

x	$\alpha_0^{'}(x)$	$\alpha_1^{'}(x)$	$\beta_0^{'}(x)$	$\beta_1^{'}(x)$
$x_0 = 0$	0	0	1	0
$x_1 = 1$	0	0	0	1

The above two tables can be shortly expressed as

$$\begin{aligned}\alpha_i(x_j) = \delta_{ij} \quad \beta_i(x_j) = 0 \quad i,j = 0,1\\ \alpha_i^{'}(x_j) = 0 \quad \beta_i^{'}(x_j) = \delta_{ij} \quad i,j = 0,1\end{aligned} \tag{3.25}$$

where δ_{ij} is the Kronecker delta.

General Hermite interpolation with basis functions

Generally, the Hermite interpolation can be set as:

$$h(x) = \sum_{i=0}^{n} \alpha_i(x)y_i + \sum_{i=0}^{n} \beta_i(x)y_i^{'} + \cdots + \sum_{i=0}^{n} \gamma_i(x)y_i^{(m)} \tag{3.26}$$

with the following constraints

$$\begin{aligned} h(x_j) &= y_j, \quad j = 0, \ldots, n \\ h^{'}(x_j) &= y_j^{'}, \quad j = 0, \ldots, n \\ &\vdots \\ h^{(m)}(x_j) &= y_j^{(m)}, \quad j = 0, \ldots, n \end{aligned}$$

where $y_j^{(m)}$ is the m-th derivative of y_j.
The requirement for the first derivative

$$\begin{aligned} h^{'}(x_j) &= y_j^{'}, \quad j = 0, \ldots, n \\ y_j^{'} &= \sum_{i=0}^{n} \alpha_i^{'}(x_j) y_i + \sum_{i=0}^{n} \beta_i^{'}(x_j) y_i^{'} + \cdots + \sum_{i=0}^{n} \gamma_i^{'}(x_j) y_i^{(m)} \end{aligned} \tag{3.27}$$

is associated with the following constraints for the basis functions:

$$\begin{aligned} \alpha_i^{'}(x_j) &= 0, \qquad i, j = 0, \ldots, n \\ \beta_i^{'}(x_j) &= \delta_{ij}, \qquad i, j = 0, \ldots, n \\ &\vdots \\ \gamma_i^{'}(x_j) &= 0 \qquad i, j = 0, \ldots, n. \end{aligned}$$

Analogically, in order to satisfy the m-th derivative conditions

$$\begin{aligned} h^{(m)}(x_j) &= y_j^{(m)}, \quad j = 0, \ldots, n \\ y_j^{(m)} &= \sum_{i=0}^{n} \alpha_i^{(m)}(x_j) y_i + \sum_{i=0}^{n} \beta_i^{(m)}(x_j) y_i^{'} + \cdots + \sum_{i=0}^{n} \gamma_i^{(m)}(x_j) y_i^{(m)} \end{aligned}$$

the following constraints are required for the basis functions:

$$\begin{aligned} \alpha_i^{(m)}(x_j) &= 0, \qquad i, j = 0, \ldots, n \\ \beta_i^{(m)}(x_j) &= 0, \qquad i, j = 0, \ldots, n \\ &\vdots \\ \gamma_i^{(m)}(x_j) &= \delta_{ij}, \qquad i, j = 0, \ldots, n. \end{aligned}$$

HIP basis functions in their general forms are expressed as

$$\begin{aligned}
\alpha_i(x) &= \textstyle\sum_{j=0}^{(m+1)(n+1)-1} a_{ij}x^j, \quad i = 0, \ldots, n \\
\beta_i(x) &= \textstyle\sum_{j=0}^{(m+1)(n+1)-1} b_{ij}x^j, \quad i = 0, \ldots, n \\
&\vdots \\
\gamma_i(x) &= \textstyle\sum_{j=0}^{(m+1)(n+1)-1} g_{ij}x^j, \quad i = 0, \ldots, n
\end{aligned}$$

3.2 Spline interpolation

Since very high order polynomials oscillate, the idea of the mathematical spline (MS) was developed following the observation of a flexible strip used to draw smooth curves through a finite number of knots situated on a flat surface. MS and a flexible strip has only one feature in common that is the continuity of a curve. Given points (x_i, y_i) for $i = 1, \ldots, n$, MS is built on the idea of a piecewise function - interpolant, $S(x)$, composed of the partial low-order polynomials, $s_i(x)$, evaluated for nodes $\{y_i, y_{i+1}\}$ on each subinterval determined by knots $\{x_i, x_{i+1}\}$. The partial polynomials, $s_i(x)$, may be of first, second or third order.

Admittedly, lower order splines do not oscillate but they are not very smooth at nodes where they may abruptly change their slopes, eg. the first order splines may not be continuous. The second derivative of the second order spline may not be continuous as well, so the curvature of the quadratic spline may drop at nodes. This is why, the first and the second order splines are not much popular in the contemporary IT and therefore, we are going to omit them and discuss the cubic splines (CS) only.

The another crucial idea for the construction of the spline interpolant $S(x)$ concerns the replacement of polynomial coefficients by values of the highest degree derivatives, eg. the 2-nd derivatives of partial CS, evaluated at ends of each two-knots subinterval. Therefore, the final formulation of the spline problem is expressed by a linear matrix equation where the vector of unknowns comprises values of the second derivatives of $s_i(x)$ at x_i.

Two approaches to the cubic spline interpolation will be presented here

- the first one starts with the identification of coefficients of a partial spline, a_i, b_i, c_i, d_i, goes through expressing them in terms of values of the second derivatives $m_i(x_i)$ and/or nodes y_i, and ends with the formulation of a matrix equation for $m_i(x_i)$,

- the second one starts with the twice integration of an interpolant, i.e. function $m_i(x)$ written in terms of values of the second derivatives of $s_i(x)$ at x_i and x_{i+1}, goes through the identification of integration constants and ends with the formulation of a matrix equation for $m_i(x_i)$.

3.2.1 The first approach to cubic spline interpolation

The piecewise function $S(x)$ is called the cubic spline interpolant (CSI) if there exists n partial cubic polynomials s_i such that following hold:

$$S(x) = \begin{cases} s_0(x) & \text{if } x_0 \le x \le x_1, \\ s_1(x) & \text{if } x_1 \le x \le x_2, \\ s_2(x) & \text{if } x_2 \le x \le x_3, \\ & \vdots \\ s_{n-1}(x) & \text{if } x_{n-1} \le x \le x_n, \end{cases}$$

where s_i is a third degree polynomial defined by

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{3.28}$$

for $i = 0, 1, \ldots, n$. Because the continuity of $S(x)$ includes requirements for the continuity of the first and second derivatives at $i = 1, \ldots, n-1$ knots, ie, everywhere apart of the ends: 0 and n-th knots, the following functions are essential to the smoothness of the interpolation process:

$$s^{'}(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i \tag{3.29}$$

$$s^{''}(x) = 6a_i(x - x_i) + 2b_i. \tag{3.30}$$

CSI needs to fulfil the following requirements

- $S(x)$ fits all data points, i.e. knots x_i and nodes y_i,
- $S(x), S^{'}(x), S^{''}(x)$ are continuous on the interval $[x_0, x_n]$.

Following the above assumptions and Eq. (3.28), the fundamental equality is concluded

$$S(x_i) = s_i(x_i) = y_i = a_i(x_i - x_i)^3 + b_i(x_i - x_i)^2 + c_i(x_i - x_i) + d_i = d_i \tag{3.31}$$

for each $i = 0, 1, 2, \ldots, n$. From the continuity and smoothness of $S(x)$ it can be concluded that in each subinterval

$$s_i(x_i) = s_{i-1}(x_i) \tag{3.32}$$

for $i = 1, 2, \ldots, n-1$. From Eq. (3.31)

$$s_i(x_i) = d_i$$

and

$$s_{i-1}(x_i) = a_{i-1}(x_i - x_{i-1})^3 + b_{i-1}(x_i - x_{i-1})^2 + c_{i-1}(x_i - x_{i-1}) + d_{i-1} \quad (3.33)$$

from where the recurrence formula for d_i can be concluded

$$d_i = a_{i-1}(x_i - x_{i-1})^3 + b_{i-1}(x_i - x_{i-1})^2 + c_{i-1}(x_i - x_{i-1}) + d_{i-1} \quad (3.34)$$

for $i = 1, 2, \ldots, n-1$. Sorting out the problem with the continuity of $s_i(x)$ we can apply the some procedure to study the continuity of its first derivative leading to the derivation of c_i. To this purpose we can write

$$s_i^{'}(x_i) = s_{i-1}^{'}(x_i), \quad (3.35)$$

but following Eq. (3.29)

$$s_i^{'}(x_i) = c_i, \quad (3.36)$$

and

$$s_{i-1}^{'}(x_i) = 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} \quad (3.37)$$

from where the recurrence formula for c_i can be derived

$$c_i = 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1}. \quad (3.38)$$

The continuity of the second derivative of $s_i(x)$ across the interval $[x_{i-1}, x_{i+1}]$ gives the equality $s_{i+1}^{''}(x_i) = s_i^{''}(x_i)$ for $i = 1, 2, \ldots, n-1$ that together with Eq. (3.30) leads to

$$s_{i+1}^{''}(x_i) = 2b_{i+1} = 6a_i(x_{i+1} - x_i) + 2b_i. \quad (3.39)$$

and subsequently results in the recurrence equation for b_i

$$b_{i+1} = 3a_i(x_{i+1} - x_i) + b_i. \quad (3.40)$$

The second stage of the derivation of the cubic spline interpolation procedure consists of the introduction of coefficients $m_i \equiv s_i^{''}(x_i)$ and expressing a_i, b_i, c_i in terms of m_i. The easiest think is to find b_i

$$s_i^{''}(x_i) \equiv m_i = 2b_i. \quad (3.41)$$

from where

$$b_i = \frac{m_i}{2}. \tag{3.42}$$

Rearranging Eq. (3.40), the coefficient a_i can be expressed by

$$\begin{aligned} 3a_i(x_{i+1} - x_i) &= b_{i+1} - b_i \\ a_i &= \frac{b_{i+1} - b_i}{3(x_{i+1} - x_i)} \\ a_i &= \frac{\frac{m_{i+1}}{2} - \frac{m_i}{2}}{3(x_{i+1} - x_i)} \end{aligned}$$

$$a_i = \frac{m_{i+1} - m_i}{6(x_{i+1} - x_i)} \tag{3.43}$$

The coefficient c_i can be find from the recurrence formula for d_{i+1}, i.e. from Eq. (3.34) taken with the subscript increased by one

$$d_{i+1} = a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i \tag{3.44}$$

from where c_i can be issued after several rearranging steps:

$$\begin{aligned} c_i(x_{i+1} - x_i) &= (d_{i+1} - d_i) - a_i(x_{i+1} - x_i)^3 - b_i(x_{i+1} - x_i)^2 \\ c_i &= \frac{d_{i+1} - d_i}{x_{i+1} - x_i} - a_i(x_{i+1} - x_i)^2 - b_i(x_{i+1} - x_i) \\ c_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{m_{i+1} - m_i}{6}(x_{i+1} - x_i) - \frac{m_i}{2}(x_{i+1} - x_i) \\ c_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{m_{i+1} - m_i + 3m_i}{6}(x_{i+1} - x_i), \end{aligned}$$

$$c_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{m_{i+1} + 2m_i}{6}(x_{i+1} - x_i) \tag{3.45}$$

Finally, all coefficients of s_i for $(n-1)$ equations, constituting the interpolating spline $S(x)$, can be expressed either in terms of m_i or known nodes y_i by the following relations

$$\begin{aligned} a_i &= \frac{m_{i+1} - m_i}{6(x_{i+1} - x_i)} \\ b_i &= \frac{m_i}{2} \\ c_i &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{m_{i+1} + 2m_i}{6}(x_{i+1} - x_i) \\ d_i &= y_i \end{aligned} \tag{3.46}$$

Assuming that knots x_i are equidistant, i.e. their common span is $h = x_{i+1} - x_i$, then Eq. (3.46) can be rewritten as

$$\begin{aligned} a_i &= \frac{m_{i+1} - m_i}{6h} \\ b_i &= \frac{m_i}{2} \\ c_i &= \frac{y_{i+1} - y_i}{h} - \frac{m_{i+1} + 2m_i}{6}h \\ d_i &= y_i \end{aligned} \tag{3.47}$$

Note that only a_i and c_i depend on h and three of them depend on m_i. The coefficients d_i are already known for all nodes y_i. The system of equations for evaluation of m_i can be issued from the recurrence formula for c_i, i.e. Eq. (3.36) written for the subscript $(i+1)$

$$c_{i+1} = 3a_ih^2 + 2b_ih + c_i. \tag{3.48}$$

Substituting the first three equations of Eq. (3.47) into Eq. (3.48) and after several transformations we can obtain the system of equations with unknowns m_i

$$\begin{aligned} \tfrac{y_{i+2} - y_{i+1}}{h} - \tfrac{m_{i+2} + 2m_{i+1}}{6}h = \tfrac{m_{i+1} - m_i}{2}h + m_ih &+ \tfrac{y_{i+1} - y_i}{h} - \tfrac{m_{i+1} + 2m_i}{6}h \\ -\tfrac{m_{i+2} + 2m_{i+1}}{6}h - \tfrac{m_{i+1} - m_i}{2}h - m_ih + \tfrac{m_{i+1} + 2m_i}{6}h &= \tfrac{y_{i+1} - y_i}{h} - \tfrac{y_{i+2} - y_{i+1}}{h} \\ h\big[\tfrac{m_{i+2} + 2m_{i+1}}{6} + \tfrac{m_{i+1} - m_i}{2} + m_i - \tfrac{m_{i+1} + 2m_i}{6}\big] &= \tfrac{y_{i+2} - y_{i+1}}{h} - \tfrac{y_{i+1} - y_i}{h} \\ \tfrac{h}{6}(m_i + 4m_{i+1} + m_{i+2}) &= \tfrac{y_i - 2y_{i+1} + y_{i+2}}{h} \end{aligned}$$

$$m_i + 4m_{i+1} + m_{i+2} = \frac{6}{h^2}(y_i - 2y_{i+1} + y_{i+2}) \tag{3.49}$$

for $i = 1, 2, \ldots, (n-1)$, that can be expressed in the matrix form

$$\begin{bmatrix} 1 & 4 & 1 & 0 & \ldots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \ldots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & \ldots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_{n-4} \\ m_{n-3} \\ m_{n-2} \\ m_{n-1} \\ m_n \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_0 - 2y_1 + y_2 \\ y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-5} - 2y_{n-4} + y_{n-3} \\ y_{n-4} - 2y_{n-3} + y_{n-2} \\ y_{n-3} - 2y_{n-2} + y_{n-1} \\ y_{n-2} - 2y_{n-1} + y_n \end{bmatrix}$$

(3.50)

where the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 4 & 1 \end{bmatrix} \tag{3.51}$$

has $(n-1)$ rows and $(n+1)$ columns. It means that we have two equations less that we need to find $(n+1)$ coefficients m_i, i.e. from Eq. (3.50) we can find only $(n-1)$ values of m_i. The missing values of m_i can be defined by arbitrary conditions defined at the endpoints, i.e. in knots: i_0 and i_n. Such "boundary" values of m_i constitute three types of cubic splines:

- natural spline,
- parabolic runout spline,
- cubic runout spline.

Natural spline

This type of interpolation splines includes the stipulation that the second derivative equals to zero at the endpoints, i.e.

$$m_0 = m_n = 0. \tag{3.52}$$

That leads to the matrix equation

$$\begin{bmatrix} 1 & 4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_{n-4} \\ m_{n-3} \\ m_{n-2} \\ m_{n-1} \\ 0 \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_0 - 2y_1 + y_2 \\ y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-5} - 2y_{n-4} + y_{n-3} \\ y_{n-4} - 2y_{n-3} + y_{n-2} \\ y_{n-3} - 2y_{n-2} + y_{n-1} \\ y_{n-2} - 2y_{n-1} + y_n \end{bmatrix}$$

(3.53)

Eliminating the first and the last columns, we obtain the $(n-2)\times(n-2)$ matrix equation

$$\begin{bmatrix} 4 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 4 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 4 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_{n-4} \\ m_{n-3} \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-5} - 2y_{n-4} + y_{n-3} \\ y_{n-4} - 2y_{n-3} + y_{n-2} \\ y_{n-3} - 2y_{n-2} + y_{n-1} \end{bmatrix} \tag{3.54}$$

which determines the remaining solutions for $\{m_1, m_2, \dots, m_{n-1}\}$ that makes the spline $S(x)$ unique.

Parabolic runout spline

The parabolic runout spline (PRS) is built on the basis of the assumption that the second derivatives at the endpoints, m_0 and m_n, are equal to m_2 and m_{n-1} respectively:

$$\begin{aligned} m_0 &= m_1, \\ m_n &= m_{n-1}. \end{aligned} \tag{3.55}$$

This assumption results in parabolic curves in both outermost subregions, i.e. $[x_1 - x_0]$ and $[x_n - x_{n-1}]$. This kind of spline is preferred for periodic and exponential data. The matrix equation for PRS is expressed by

$$\begin{bmatrix} 5 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & 4 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & 4 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 4 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 4 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 5 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_{n-4} \\ m_{n-3} \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-5} - 2y_{n-4} + y_{n-3} \\ y_{n-4} - 2y_{n-3} + y_{n-2} \\ y_{n-3} - 2y_{n-2} + y_{n-1} \end{bmatrix} \tag{3.56}$$

where the size of the quadratic matrix is $(n-2)\times(n-2)$.

Cubic runout spline

The cubic runout spline (CRS) is constructed on the basis of assumption of the shape of interpolating curves spanned over the two outermost subregions at each end of the domain $[x_0, x_1, \ldots, x_n]$. The stipulation is made that

$$\begin{aligned} m_0 &= 2m_1 - m_2, \\ m_n &= 2m_{n-1} - m_{n-2}. \end{aligned} \tag{3.57}$$

and this causes the curves at the subregions $[x_0, x_2]$ and $[x_{n-2}, x_n]$ to be a single cubic curve, rather than two separate curves. The matrix equation for CRS is

$$\begin{bmatrix} 6 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 1 & 4 & 1 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 4 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 4 & 1 & 0 \\ 0 & 0 & 0 & \ldots & 1 & 4 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 1 & 6 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ \vdots \\ m_{n-4} \\ m_{n-3} \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-5} - 2y_{n-4} + y_{n-3} \\ y_{n-4} - 2y_{n-3} + y_{n-2} \\ y_{n-3} - 2y_{n-2} + y_{n-1} \end{bmatrix} \tag{3.58}$$

Other type splines

Another two most popular cubic splines are

- periodic spline,
- clamped spline.

The periodic cubic spline is built on the basis of the following assumption:

$$\begin{aligned} s_0(x_0) &= s_n(x_n), \\ s_0^{'}(x_0) &= s_n^{'}(x_n), \\ m_0 &= m_n. \end{aligned} \tag{3.59}$$

The clamped cubic spline is constructed by imposing the condition that the first derivatives

$$\begin{aligned} s_0^{'}(x_0) &= u \\ s_n^{'}(x_n) &= v \end{aligned} \tag{3.60}$$

assumes the known values u and v. There are many other types of interpolating spline curves but the mentioned above are more widely used in engineering practice.

3.2.2 The second approach to cubic spline interpolation

Given points: knots (x_i) and nodes (y_i), i.e. $\{x_i, y_i\}, i = 0, 1, \ldots, n$, define a function $S(x)$ whose graph passes through the given points. The function $S(x)$ fulfils the condition $S(x) = s_i(x)$ if $x \in [x_i, x_{i+1}]$ for $i = 0, 1, \ldots, n-1$, where the partial functions, $s_i(x)$, are cubic polynomials that must be computed subject continuity requirements at least $s_i(x) \in C^2$. Since the graph of $s_i^{''}(x)$ is a straight line, the following interpolation formula is true:

$$s_i^{''}(x) = S^{''}(x_i)\frac{x - x_{i+1}}{x_i - x_{i+1}} + S^{''}(x_{i+1})\frac{x - x_i}{x_{i+1} - x_i} \tag{3.61}$$

where the values of $S^{''}(x_i)$ are not yet determined. Integrating the function twice $s_i^{''}(x)$ gives the function $s_i(x)$ and its first derivative, respectively:

$$s_i^{'}(x) = S^{''}(x_i)\frac{(x - x_{i+1})^2}{2(x_i - x_{i+1})} + S^{''}(x_{i+1})\frac{(x - x_i)^2}{2(x_{i+1} - x_i)} + C_1 \tag{3.62}$$

$$s_i(x) = S^{''}(x_i)\frac{(x - x_{i+1})^3}{6(x_i - x_{i+1})} + S^{''}(x_{i+1})\frac{(x - x_i)^3}{6(x_{i+1} - x_i)} + C_1 x + C_2 \tag{3.63}$$

The integration constants C_1 and C_2 can be find from Eq. (3.63) substituting at first $s_i(x_i) = y_i$ and then $s_i(x_{i+1}) = y_{i+1}$:

$$\begin{aligned} y_i &= S^{''}(x_i)\frac{(x_i - x_{i+1})^3}{6(x_i - x_{i+1})} + S^{''}(x_{i+1})\frac{(x_i - x_i)^3}{6(x_{i+1} - x_i)} + C_1 x_i + C_2 \\ y_{i+1} &= S^{''}(x_i)\frac{(x_{i+1} - x_{i+1})^3}{6(x_i - x_{i+1})} + S^{''}(x_{i+1})\frac{(x_{i+1} - x_i)^3}{6(x_{i+1} - x_i)} + C_1 x_{i+1} + C_2 \end{aligned}$$

from where the following relations can be derived

$$y_i = S^{''}(x_i)\frac{(x_i - x_{i+1})^2}{6} + C_1 x_i + C_2, \tag{3.64}$$

$$y_{i+1} = S^{''}(x_{i+1})\frac{(x_{i+1} - x_i)^2}{6} + C_1 x_{i+1} + C_2. \tag{3.65}$$

Assuming that knots x_i for $i = 0, 1, \ldots, n$ are equidistant, i.e. $h = x_{i+1} - x_i$, the above equations can be written as

$$y_i = \frac{S''(x_i)h^2}{6} + C_1 x_i + C_2, \tag{3.66}$$

$$y_{i+1} = \frac{S''(x_{i+1})h^2}{6} + C_1 x_{i+1} + C_2. \tag{3.67}$$

Subtracting Eq. (3.66) from Eq. (3.67) results in

$$\begin{aligned} y_{i+1} - y_i &= \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6} + C_1(x_{i+1} - x_i), \\ y_{i+1} - y_i &= \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6} + C_1 h, \end{aligned}$$

from where C_1 can be derived

$$C_1 = \frac{1}{h}\Big[y_{i+1} - y_i - \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}\Big] \tag{3.68}$$

Substituting this into Eq. (3.64) gives

$$y_i = \frac{S''(x_i)h^2}{6} + \frac{1}{h}\big[y_{i+1} - y_i - \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}\big]x_i + C_2$$

from where C_2 can be obtained

$$C_2 = y_i - \frac{S''(x_i)h^2}{6} - \frac{1}{h}\Big[y_{i+1} - y_i - \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}\Big]x_i \tag{3.69}$$

Substituting C_1 and C_2 into Eq. (3.63) we get

$$\begin{aligned} s_i(x) &= S''(x_i)\frac{(x - x_{i+1})^3}{6(x_i - x_{i+1})} + S''(x_{i+1})\frac{(x - x_i)^3}{6(x_{i+1} - x_i)} \\ &+ \frac{1}{h}\Big[y_{i+1} - y_i - \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}\Big]x \\ &+ y_i - \frac{S''(x_i)h^2}{6} - \frac{1}{h}\Big[y_{i+1} - y_i - \big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}\Big]x_i. \end{aligned}$$

Splitting some terms in above expression leads to

$$\begin{aligned} s_i(x) &= S''(x_i)\frac{(x - x_{i+1})^3}{6h} + S''(x_{i+1})\frac{(x - x_i)^3}{6h} \\ &+ \frac{y_{i+1} - y_i}{h}x - \frac{\big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}}{h}x \\ &+ y_i - \frac{S''(x_i)h^2}{6} - \frac{y_{i+1} - y_i}{h}x_i + \frac{\big(S''(x_{i+1}) - S''(x_i)\big)\frac{h^2}{6}}{h}x_i. \end{aligned}$$

Ordering two terms with $\frac{h^2}{6}$ results in

$$\begin{aligned} s_i(x) &= S^{''}(x_i)\frac{(x-x_{i+1})^3}{6h} + S^{''}(x_{i+1})\frac{(x-x_i)^3}{6h} \\ &+ \frac{y_{i+1}-y_i}{h}x - \frac{(S^{''}(x_{i+1})-S^{''}(x_i))h}{6}x \\ &+ y_i - \frac{S^{''}(x_i)h^2}{6} - \frac{y_{i+1}-y_i}{h}x_i + \frac{(S^{''}(x_{i+1})-S^{''}(x_i))h}{6}x_i. \end{aligned}$$

Splitting the fourth and the last term gives

$$\begin{aligned} s_i(x) &= -S^{''}(x_i)\frac{(x-x_{i+1})^3}{6h} + S^{''}(x_{i+1})\frac{(x-x_i)^3}{6h} \\ &+ \frac{y_{i+1}-y_i}{h}x - \frac{S^{''}(x_{i+1})h}{6}x + \frac{S^{''}(x_i)h}{6}x \\ &+ y_i - \frac{S^{''}(x_i)h^2}{6} - \frac{y_{i+1}-y_i}{h}x_i + \frac{S^{''}(x_{i+1})h}{6}x_i - \frac{S^{''}(x_i)h}{6}x_i. \end{aligned}$$

Grouping terms with $S^{''}(x_i)$ and $S^{''}(x_{i+1})$ produces

$$\begin{aligned} s_i(x) &= -S^{''}(x_i)\frac{(x-x_{i+1})^3}{6h} + \frac{S^{''}(x_i)h}{6}x - \frac{S^{''}(x_i)h^2}{6} - \frac{S^{''}(x_i)h}{6}x_i \\ &+ S^{''}(x_{i+1})\frac{(x-x_i)^3}{6h} - \frac{S^{''}(x_{i+1})h}{6}x + \frac{S^{''}(x_{i+1})h}{6}x_i \\ &+ y_i + \frac{y_{i+1}-y_i}{h}x - \frac{y_{i+1}-y_i}{h}x_i. \end{aligned}$$

Factoring out terms $S^{''}(x_i)$ and $S^{''}(x_{i+1})$ results in

$$\begin{aligned} s_i(x) &= \frac{S^{''}(x_i)}{6}\Big[-\frac{(x-x_{i+1})^3}{h} + hx - h^2 - h(x_{i+1}-h)\Big] \\ &+ \frac{S^{''}(x_{i+1})}{6}\Big[\frac{(x-x_i)^3}{h} - hx + hx_i\Big] \\ &+ y_i + \frac{y_{i+1}-y_i}{h}x - \frac{y_{i+1}-y_i}{h}x_i. \end{aligned}$$

Ordering further first two terms and splitting fractions with y_{i+1} and y_i leads to

$$\begin{aligned} s_i(x) &= \frac{S^{''}(x_i)}{6}\Big[\frac{(x_{i+1}-x)^3}{h} - h(x_{i+1}-x)\Big] \\ &+ \frac{S^{''}(x_{i+1})}{6}\Big[\frac{(x-x_i)^3}{h} - h(x-x_i)\Big] \\ &+ y_i + \frac{y_{i+1}}{h}x - \frac{y_i}{h}x - \frac{y_{i+1}}{h}x_i + \frac{y_i}{h}x_i. \end{aligned}$$

Factoring out y_{i+1} and y_i in the last five terms gives

$$\begin{aligned} s_i(x) &= \frac{S^{''}(x_i)}{6}\Big[\frac{(x_{i+1}-x)^3}{h} - h(x_{i+1}-x)\Big] \\ &+ \frac{S^{''}(x_{i+1})}{6}\Big[\frac{(x-x_i)^3}{h} - h(x-x_i)\Big] \\ &+ y_i\Big(1 - \frac{x}{h} + \frac{x_i}{h}\Big) + \frac{y_{i+1}}{h}(x-x_i). \end{aligned}$$

Rearranging the term next to y_i leads to

$$\begin{aligned} s_i(x) &= \frac{S^{''}(x_i)}{6}\Big[\frac{(x_{i+1}-x)^3}{h} - h(x_{i+1}-x)\Big] \\ &+ \frac{S^{''}(x_{i+1})}{6}\Big[\frac{(x-x_i)^3}{h} - h(x-x_i)\Big] \\ &+ y_i\Big(\frac{x_{i+1}-x_i+x_i-x}{h}\Big) + \frac{y_{i+1}}{h}(x-x_i). \end{aligned}$$

Finally, we evaluate the partial spline in the following form

$$\begin{aligned} s_i(x) &= \frac{S^{''}(x_i)}{6}\Big[\frac{(x_{i+1}-x)^3}{h} - h(x_{i+1}-x)\Big] \\ &+ \frac{S^{''}(x_{i+1})}{6}\Big[\frac{(x-x_i)^3}{h} - h(x-x_i)\Big] \\ &+ y_i\Big(\frac{x_{i+1}-x}{h}\Big) + \frac{y_{i+1}}{h}(x-x_i). \end{aligned} \tag{3.70}$$

Having partial spline functions $s_{i-1}(x)$ and s_i, both of them with C^2 regularity, we can verify the continuity of their first derivatives in the central knot of the subregion $[x_{i-1}, x_i, x_{i+1}]$, that can be also written as $[x_i - h, x_i, x_i + h]$. This amounts to requiring that $s_i^{'}(x_i) = s_{i-1}^{'}(x_i)$. Taking the first derivative of $s_i(x)$

$$\begin{aligned} s_i^{'}(x) &= -\frac{S^{''}(x_i)}{2h}(x_{i+1}-x)^2 + \frac{S^{''}(x_i)}{6}h + \frac{S^{''}(x_{i+1})}{2h}(x-x_i)^2 \\ &- \frac{S^{''}(x_{i+1})}{6}h - \frac{y_i}{h} + \frac{y_{i+1}}{h} \end{aligned} \tag{3.71}$$

and writing s_i and s_{i-1} at the knot x_i we obtain

$$\begin{aligned} s_i^{'}(x_i) &= -\frac{S^{''}(x_i)}{2h}(x_{i+1}-x_i)^2 + \frac{S^{''}(x_i)}{6}h + \frac{S^{''}(x_{i+1})}{2h}(x_i-x_i)^2 \\ &- \frac{S^{''}(x_{i+1})}{6}h - \frac{y_i}{h} + \frac{y_{i+1}}{h} \\ s_{i-1}^{'}(x_i) &= -\frac{S^{''}(x_i)}{2h}(x_i-x_i)^2 + \frac{S^{''}(x_{i-1})}{6}h + \frac{S^{''}(x_i)}{2h}(x_i-x_{i-1})^2 \\ &- \frac{S^{''}(x_i)}{6}h - \frac{y_{i-1}}{h} + \frac{y_i}{h}. \end{aligned}$$

The elimination of zero terms and putting in order terms with exponents of h gives

$$\begin{aligned} s_i^{'}(x_i) &= -\frac{S^{''}(x_i)}{2}h + \frac{S^{''}(x_i)}{6}h - \frac{S^{''}(x_{i+1})}{6}h - \frac{y_i}{h} + \frac{y_{i+1}}{h} \\ s_{i-1}^{'}(x_i) &= \frac{S^{''}(x_{i-1})}{6}h + \frac{S^{''}(x_i)}{2}h - \frac{S^{''}(x_i)}{6}h - \frac{y_{i-1}}{h} + \frac{y_i}{h}. \end{aligned}$$

Comparing both right hand sides leads to

$$\begin{aligned} - & \quad \frac{S^{''}(x_i)}{2}h + \frac{S^{''}(x_i)}{6}h - \frac{S^{''}(x_{i+1})}{6}h - \frac{y_i}{h} + \frac{y_{i+1}}{h} \\ = & \quad \frac{S^{''}(x_{i-1})}{6}h + \frac{S^{''}(x_i)}{2}h - \frac{S^{''}(x_i)}{6}h - \frac{y_{i-1}}{h} + \frac{y_i}{h}. \end{aligned}$$

Grouping terms with $S^{''}(x_i)$ and y_i results in

$$\begin{aligned} - & \quad \frac{1}{6}S^{''}(x_{i-1})h - \frac{3}{6}S^{''}(x_i)h + \frac{1}{6}S^{''}(x_i)h - \frac{3}{6}S^{''}(x_i)h + \frac{1}{6}S^{''}(x_i)h - \frac{1}{6}S^{''}(x_{i+1})h \\ = & \quad -\frac{y_{i-1}}{h} + \frac{y_i}{h} + \frac{y_i}{h} - \frac{y_{i+1}}{h}, \end{aligned}$$

from where the following relation is obtained

$$-\frac{1}{6}S^{''}(x_{i-1})h - \frac{4}{6}S^{''}(x_i)h - \frac{1}{6}S^{''}(x_{i+1})h = -\frac{y_{i-1}}{h} + 2\frac{y_i}{h} - \frac{y_{i+1}}{h}.$$

Finally, after further ordering we have

$$S^{''}(x_{i-1}) + 4S^{''}(x_i) + S^{''}(x_{i+1}) = \frac{6}{h^2}\Big[y_{i-1} - 2y_i + y_{i+1}\Big]. \tag{3.72}$$

Identifying m_i for $\{i-1, i, i+1\}$ by

$$\begin{aligned} m_{i-1} &= S^{''}(x_{i-1}), \\ m_i &= S^{''}(x_i), \\ m_{i+1} &= S^{''}(x_{i+1}), \end{aligned} \tag{3.73}$$

leads to

$$m_{i-1} + 4m_i + m_{i+1} = \frac{6}{h^2}\Big[y_{i-1} - 2y_i + y_{i+1}\Big]. \tag{3.74}$$

that, after adding 1 to all subscripts, can be rewritten in the form the same as Eq. (3.49)

$$m_i + 4m_{i+1} + m_{i+2} = \frac{6}{h^2}\Big[y_i - 2y_{i+1} + y_{i+2}\Big]. \tag{3.75}$$

For $i = 1, 2, \ldots, n-1$, this is the system of $(n-1)$ equations with $(n+1)$ unknowns m_i.

Example (3.3)♣

Find the natural spline function for the following data

i	0	1	2
x_i	-1	0	1
y_i	1	2	1

where $h = 1$.

Recalling Eq. (3.75)

$$m_0 + 4m_1 + m_2 = \frac{6}{h^2}(y_0 - 2y_1 + y_2)$$

and substituting the conditions for the natural spline $m_0 = m_2 = 0$ we obtain

$$4m_1 = \frac{6}{h^2}(y_0 - 2y_1 + y_2).$$

Inserting numbers y_i we obtain the value of m_1

$$m_1 = \frac{6}{4h^2}(y_0 - 2y_1 + y_2) = -3.$$

Using Eq. (3.70) with $h = 1$ we can write the system of two equations

$$\begin{aligned} s_0(x) &= \frac{m_0}{6}[(x_1 - x)^3 - (x_1 - x)] + \frac{m_1}{6}[(x - x_0)^3 - (x - x_0)] \\ &+ y_0(x_1 - x) + y_1(x - x_0) \\ s_1(x) &= \frac{m_1}{6}[(x_2 - x)^3 - (x_2 - x)] + \frac{m_2}{6}[(x - x_1)^3 - (x - x_1)] \\ &+ y_1(x_2 - x) + y_2(x - x_1) \end{aligned}$$

Inserting $m_0 = m_2 = 0$ into the above system results in

$$\begin{aligned} s_0(x) &= \frac{m_1}{6}[(x - x_0)^3 - (x - x_0)] + y_0(x_1 - x) + y_1(x - x_0), \\ s_1(x) &= \frac{m_1}{6}[(x_2 - x)^3 - (x_2 - x)] + y_1(x_2 - x) + y_2(x - x_1). \end{aligned}$$

Substituting the values of x_i and y_i, the cubic spline polynomials can be expressed by

$$\begin{aligned} s_0(x) &= -\frac{1}{2}[(x + 1)^3 - (x + 1)] - x + 2(x + 1), \\ s_1(x) &= -\frac{1}{2}[(1 - x)^3 - (1 - x)] + 2(1 - x) + x. \end{aligned}$$

♠

Example (3.4)♣

Find the natural spline function for the following data

i	0	1	2	3	4
x_i	-2	-1	0	1	2
y_i	0	1	2	1	0

where $h = 1$.

The system of equations that can be derived from Eq. (3.75) consists of three equations

$$\begin{aligned} 4m_1 + m_2 &= 6(y_0 - 2y_1 + y_2), \\ m_1 + 4m_2 + m_3 &= 6(y_1 - 2y_2 + y_3), \\ m_2 + 4m_3 &= 6(y_2 - 2y_3 + y_4). \end{aligned}$$

Substituting numbers it can be rewritten as

$$\begin{array}{rrrcr} 4m_1 & +m_2 & & = & 0, \\ m_1 & +4m_2 & +m_3 & = & -12, \\ & m_2 & +4m_3 & = & 0, \end{array}$$

This we can express in the matrix form

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -12 \\ 0 \end{bmatrix}$$

or expressed in the compact form by

$$\mathbf{A} \times \mathbf{m} = \mathbf{b}. \tag{3.76}$$

Knowing the inverse quadratic matrix

$$A^{-1} = \frac{1}{56} \begin{bmatrix} 15 & -4 & 1 \\ -4 & 16 & 4 \\ 1 & -4 & 15 \end{bmatrix},$$

we can solve the system Eq. (3.76)

$$\mathbf{m} = \mathbf{A^{-1}} \times \mathbf{b}$$

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \frac{1}{56} \begin{bmatrix} 15 & -4 & 1 \\ -4 & 16 & 4 \\ 1 & -4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ -12 \\ 0 \end{bmatrix}$$

and the system solution is

$$\begin{bmatrix} m_1 & m_2 & m_3 \end{bmatrix}^T = \begin{bmatrix} \frac{6}{7} & \frac{-24}{7} & \frac{6}{7} \end{bmatrix}^T$$

Knowing values of m_is and recalling Eq. (3.70), we can write four cubic polynomials $s_i(x)$ for $i = 0, 1, 2, 3$

$$\begin{aligned}
s_0(x) &= \frac{m_0}{6}\left[(x_1-x)^3-(x_1-x)\right]+\frac{m_1}{6}\left[(x-x_0)^3-(x-x_0)\right]+y_0(x_1-x)+y_1(x-x_0)\\
s_1(x) &= \frac{m_1}{6}\left[(x_2-x)^3-(x_2-x)\right]+\frac{m_2}{6}\left[(x-x_1)^3-(x-x_1)\right]+y_1(x_2-x)+y_2(x-x_1)\\
s_2(x) &= \frac{m_2}{6}\left[(x_3-x)^3-(x_3-x)\right]+\frac{m_3}{6}\left[(x-x_2)^3-(x-x_2)\right]+y_2(x_3-x)+y_3(x-x_2)\\
s_3(x) &= \frac{m_3}{6}\left[(x_4-x)^3-(x_4-x)\right]+\frac{m_4}{6}\left[(x-x_3)^3-(x-x_3)\right]+y_3(x_4-x)+y_4(x-x_3)
\end{aligned}$$

Inserting the values of m_i's together with values of knots and nodes we obtain the four cubic polynomials

$$\begin{aligned}
s_0(x) &= \frac{0}{6}\left[(-1-x)^3-(-1-x)\right]+\frac{m_1}{6}\left[(x-(-2))^3-(x-(-2))\right]\\
&+ 0(-1-x)+1(x-(-2)),\\
s_1(x) &= \frac{m_1}{6}\left[(0-x)^3-(0-x)\right]+\frac{m_2}{6}\left[(x-(-1))^3-(x-(-1))\right]\\
&+ 1(0-x)+2(x-(-1)),\\
s_2(x) &= \frac{m_2}{6}\left[(1-x)^3-(1-x)\right]+\frac{m_3}{6}\left[(x-0)^3-(x-0)\right]\\
&+ 2(1-x)+1(x-0),\\
s_3(x) &= \frac{m_3}{6}\left[(2-x)^3-(2-x)\right]+\frac{0}{6}\left[(x-1)^3-(x-1)\right]\\
&+ 1(2-x)+0(x-1),
\end{aligned}$$

that after some ordering can be finally rewritten in the form

$$\begin{array}{lllll}
s_0(x) = & & \frac{1}{7}[(x+2)^3-(x+2)] & & +x+2\\
s_1(x) = & \frac{1}{7}x(1-x^2) & -\frac{4}{7}[(x+1)^3-(x+1)] & & +x+1\\
s_2(x) = & -\frac{4}{7}[(1-x)^3-(1-x)] & +\frac{1}{7}x(x^2-1) & & +2-x\\
s_3(x) = & \frac{1}{7}[(2-x)^3-(2-x)] & & & +2-x
\end{array}$$

♠

3.3 Extrapolation

Extrapolation is the use of an interpolating formula for external nodes which are not situated within the interval. One of most popular extrapolation methods is Richardson's extrapolation applied not only in evaluation of a value of extrapolated function but mostly for upgrading of ordinary numerical methods ranging from the evaluation of derivatives to the evaluation of quadratures, i.e. integrals. The idea is to use low order formulas for which the expression of the truncation error is well known, and derive higher order

accuracy schemes from the low order formulas at the expense of higher computation. The method can be applied when the approximation generates a predictable error that depends on a systematic parameter, such as the step size h.

3.3.1 Richardson extrapolation

Richardson extrapolation is the method used to generate high accuracy by applying low-order formulas in discrete numerical methods with step size h. It is an excellent idea which can be used to upgrade a numerical method of order $O(h^p)$ to the method of the order $O(h^{p+1})$. In short, we can approximate some desired quantity A by $A_0(h)$. Note that A does not depend of h.

Assume that we know the explicit expression for an asymptotic error expansion (truncation error)

$$A = A_0(h) + K_0 h^{k_0} + K_1 h^{k_1} + K_2 h^{k_2} + \dots \tag{3.77}$$

where $K_0, K_1, K_2, \dots$ are usually unknown constants and k_i are known exponents such that $h^{k_i} > h^{k_{i+1}}$. The above equation may also be written in a more compact form

$$A = A_0(h) + K_0 h^{k_0} + O(h^{k_1}) \tag{3.78}$$

where $O(h^{k_1})$ is a sum of terms of order h^{k_1} and higher. Assuming that $O(h^{k_1})$ is the tiny term and can be dropped from Eq. (3.78), the linear equation is obtained

$$A = A_0(h) + K_0 h^{k_0} \tag{3.79}$$

with the two unknowns A and K. For each particular h, the Eq. (3.79) is a different equation and using just a different step size, eg. half of h, the following equation is obtained

$$A = A_0\left(\frac{h}{2}\right) + K_0\left(\frac{h}{2}\right)^{k_0} + O(h^{k_1}) \tag{3.80}$$

The two equations (3.79) and (3.80) may be solved, yielding approximate values of A and K_0. Note that in those equations the symbol $O(h^{k_1})$ stands for two different sums. Multiplying Eq. (3.80) by 2^{k_0} and then subtracting Eq. (3.78) yields

$$(2^{k_0} - 1)A = 2^{k_0} A_0\left(\frac{h}{2}\right) - A_0(h) + O(h^{k_1}) \tag{3.81}$$

Eq. (3.81) can be solved for A to give

$$A = \frac{2^{k_0} A_0\left(\frac{h}{2}\right) - A_0(h)}{(2^{k_0} - 1)} + O(h^{k_1}). \tag{3.82}$$

Following this process, a better approximation of A is achieved by subtracting the largest term $O(h^{k_0})$ in the error. This process can be repeated and then more error terms can be removed resulting in the better approximation. Defining the new $A_1(h)$

$$A_1(h) = \frac{2^{k_0} A_0\left(\frac{h}{2}\right) - A_0(h)}{(2^{k_0} - 1)} \tag{3.83}$$

the desired quantity A is approximated by

$$A = A_1(h) + O(h^{k_1}). \tag{3.84}$$

A general recurrence formula can be deduced from Eq. (3.83)

$$A_{i+1}(h) = \frac{2^{k_i} A_i\left(\frac{h}{2}\right) - A_i(h)}{(2^{k_i} - 1)}. \tag{3.85}$$

It can be even more generalized replacing 2^{k_i} by t^{k_i}

$$A_{i+1}(h) = \frac{t^{k_i} A_i\left(\frac{h}{t}\right) - A_i(h)}{(t^{k_i} - 1)}. \tag{3.86}$$

where t is some integer, and then Eq. (3.84) is expressed by

$$A = A_{i+1}(h) + O(h^{k_{i+1}}). \tag{3.87}$$

Examples of Richardson's extrapolation will be given in the section on finite differences.

Chapter 4

Direct solution of systems of linear equations

The system of n algebraic equations [50], [44]

$$\begin{array}{ccccccc}
a_{11}x_1 & +a_{12}x_2 & +a_{13}x_3 & +\ldots & +a_{1n}x_n & = & b_1 \\
a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & +\ldots & +a_{2n}x_n & = & b_2 \\
a_{31}x_1 & +a_{32}x_2 & +a_{33}x_3 & +\ldots & +a_{3n}x_n & = & b_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_{(n-2)1}x_1 & +a_{(n-2)2}x_2 & +a_{(n-2)3}x_3 & +\ldots & +a_{(n-2)n}x_n & = & b_{(n-2)} \\
a_{(n-1)1}x_1 & +a_{(n-1)2}x_2 & +a_{(n-1)3}x_3 & +\ldots & +a_{(n-1)n}x_n & = & b_{(n-1)} \\
a_{n1}x_1 & +a_{n2}x_2 & +a_{n3}x_3 & +\ldots & +a_{nn}x_n & = & b_n
\end{array}$$

can be represented in the matrix form [55]

$$\mathbf{Ax} = \mathbf{b} \tag{4.1}$$

and solved

$$\mathbf{x} = \mathbf{A^{-1}b} \tag{4.2}$$

subject that $\mathbf{A}$ fulfills several requirements:

- $\mathbf{A}$ is nonsingular if
  - $\mathbf{A}^{-1}$ exists,
  - $\mathbf{A}$ can be inverted,
  - the determinant of a matrix $\mathbf{A}$, i.e. $|\mathbf{A}|$ is not zero,
  - $\mathbf{Az} \neq \mathbf{0}$ if $\mathbf{z} \neq \mathbf{0}$

The number of solutions of Eq. (4.1) depends on properties of $\mathbf{A}$ and $\mathbf{b}$

- if $\mathbf{A}$ is non singular then Eq. (4.1) has one and only one solution,
- if $\mathbf{A}$ is non singular and vector $\mathbf{b} \in \text{span}\mathbf{A}$ then Eq. (4.1) has infinite solutions,
  - where span$\mathbf{A}$ is a linear span of columns of matrix $\mathbf{A}$,
  - Definition of linear span:
    * Given a vector space $\mathbf{C}$ over a field K and vectors are columns $\mathbf{c}_1, \ldots, \mathbf{c}_n$ of $\mathbf{A}$ then $\text{span}(\mathbf{c}_1, \ldots, \mathbf{c}_n) := \{a_1\mathbf{c}_1 + \cdots + a_n\mathbf{c}_n : a_1, \ldots, a_n \in K\}$ is a subspace S of $\mathbf{C}$ called the linear span of $\mathbf{c}_1, \ldots, \mathbf{c}_n$
    * The vectors are called spanning vectors and $\{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ is called a spanning set or generating set of S.
- if $\mathbf{A}$ is non singular and vector $\mathbf{b}$ is not in the span$\mathbf{A}$ than Eq. (4.1) has no solutions.

Solving Eq. (4.1) is central to many numerical algorithms. There are many methods available to this purpose. Some of them are algebraically correct, while others are iterative methods. The structure of $\mathbf{A}$ determines which method is the best subject the available computer resources: speed of a processor and an available memory.

4.1 Error bounds for solving $\mathbf{Ax} = \mathbf{b}$

The answer to this question can be given referring to the following textbooks: [54], [19]. The error bound can be derived assuming that the solution algorithm is numerically stable and knowing a condition number $\kappa(\mathbf{A})$ for the matrix $\mathbf{A}$. Sometimes the condition number is denoted as cond($\mathbf{A}$). To define the condition number $\kappa(\mathbf{A})$ of $\mathbf{A}$, we need to introduce several notions:

- Matrix norms
  - Definition of the "column" norm of a matrix: $\|\mathbf{A}\|_1 = \max_j \sum_i |a_{ij}|$ which is marked by the subscript $_1$,
  - definition of the "row" norm of a matrix: $\|\mathbf{A}\|_\infty = \max_i \sum_j |a_{ij}|$ which is marked by the subscript $_\infty$,
  - for each norm the following condition is fulfilled $\|\mathbf{AB}\| < \|\mathbf{A}\|\|\mathbf{B}\|$,

– definition of the infinity -norm of a vector: $\|\mathbf{x}\|_\infty = \max_i |x_i|$,

The condition number of a matrix is defined by

$$\kappa(\mathbf{A}) := \|\mathbf{A}\|_\infty \|\mathbf{A}^{-1}\|_\infty. \tag{4.3}$$

We wont to measure the difference between a numerical solution $\|\hat{\mathbf{x}}\|_\infty$ of Eq. (4.1)

$$(\mathbf{A} + \mathbf{E})\hat{\mathbf{x}} = \mathbf{b} \Rightarrow \hat{\mathbf{x}} = (\mathbf{A} + \mathbf{E})^{-1}\mathbf{b} \tag{4.4}$$

where $\mathbf{E}$ is taken as a small perturbation of $\mathbf{A}$, and an analytical (ideal) solution of Eq. (4.1):

$$\mathbf{A}\mathbf{x} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{4.5}$$

To this purpose we are going to derive a bound on the expression

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty}{\|\hat{\mathbf{x}}\|_\infty}, \tag{4.6}$$

which can be written by using the matrix condition number in the form

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty}{\|\hat{\mathbf{x}}\|_\infty} \leq \kappa(\mathbf{A}) \cdot \frac{\|\mathbf{E}\|_\infty}{\|\mathbf{A}\|_\infty}. \tag{4.7}$$

The above can be interpreted as follows:

- A small error is guaranteed for $\kappa(\mathbf{A})$ not too large, it can be shown that it is always at least 1, and for small $\frac{\|\mathbf{E}\|_\infty}{\|\mathbf{A}\|_\infty}$.
- $\mathbf{A}$ is well conditioned when $\kappa(\mathbf{A})$ is small,
- $\mathbf{A}$ is ill conditioned when $\kappa(\mathbf{A})$ is large,
- Numerical stability is guaranteed when $\frac{\|\mathbf{E}\|_\infty}{\|\mathbf{A}\|_\infty}$ is small and close to the machine precision ϵ that measures the roundoff error in individual floating point operations.
  - $\epsilon = 2^{-24} \approx 10^{-7}$ for IEEE standard single precision floating point arithmetics,
  - $\epsilon = 2^{-53} \approx 10^{-16}$ for IEEE standard double precision floating point arithmetics,

It is good to say that no matter what algorithm is applied to solve Eq. (4.1), we can compute $\frac{\|\mathbf{E}\|_\infty}{\|\mathbf{A}\|_\infty}$ quite cheaply given the approximate solution $\hat{\mathbf{x}}$ by using the following formula

$$\frac{\|\mathbf{E}\|_\infty}{\|\mathbf{A}\|_\infty} = \frac{\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_\infty}{\|\mathbf{A}\|_\infty \|\hat{\mathbf{x}}\|_\infty} \tag{4.8}$$

Costs of evaluation of this formula measured in flops is $O(n^2)$ in the case of dense matrices. This number is much less than $\frac{2n^3}{3}$ required by Gaussian elimination. Substituting Eq. (4.8) in Eq. (4.7) results in

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|_\infty}{\|\hat{\mathbf{x}}\|_\infty} \leq \kappa(\mathbf{A}) \cdot \frac{\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_\infty}{\|\mathbf{A}\|_\infty \|\hat{\mathbf{x}}\|_\infty}. \tag{4.9}$$

Now there is only one problem connected with the estimation of the condition number $\kappa(\mathbf{A})$. We know that it is related to computing of $\|\mathbf{A}^{-1}\|$, and it can be done quite cheaply computing the factorization $\mathbf{PA} = \mathbf{LU}$ that can be produced by Gaussian elimination, where $\mathbf{P}$ is a permutation matrix. Given $\mathbf{PA} = \mathbf{LU}$, $\|\mathbf{A}^{-1}\|$ can be estimated to within factor of two with just $O(n^2)$ flops. The computing of the condition matrix to within a factor of two is more than accurate enough for an error bound. The algorithm for estimating $\|\mathbf{A}^{-1}\|$ can be found in [22], [23] and [20].

4.2 Gauss elimination

The Gauss elimination [16] means the reduction of $\mathbf{Ax} = \mathbf{b}$ to the triangular form which could be easily solved by, so called, back-substitution. The solution starts from the last equation where we have only one unknown and the substitution process climbs up in the equation hierarchy subsequently giving the new x_i in each step. This method is based on the linear algebraic operation: any single equation of the system of linear equations can be replaced by a linear combination of this equation with any other belonging to the system.

4.2.1 Linear operations on the system of algebraic equations

Let us solve the system of N algebraic equations 4.1 replacing the original equation k by the linear combination of the j-th and k-th equations - rows in the system. The separate summation of LH sides and RH sides of j and

k equations

$$\begin{array}{rcl} a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n & = & b_j \\ a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kn}x_n & = & b_k \end{array}$$

leads to

$$(a_{j1} + a_{k1})x_1 + (a_{j2} + a_{k2})x_2 + \ldots + (a_{jn} + a_{kn})x_n = b_j + b_k. \quad (4.10)$$

From the linear algebra theorems we know that

- the new equation is satisfied subject that the equation-components are satisfied,
- the replacement of the equation k by the new one does not effect the solution of the system,
- we can replace any equation, eg. j, by the one produced by multiplying the original row by any scalar λ, eg.

$$\lambda a_{j1}x_1 + \lambda a_{j2}x_2 + \ldots + \lambda a_{jn}x_n = \lambda b_j \quad (4.11)$$

Performing two above mentioned linear operations the system Eq. (4.1) can be replaced by the following set of equations

$$\begin{array}{ccccc} a_{11}x_1 & +a_{12}x_2 & +\ldots & +a_{1n}x_n & = & b_1 \\ a_{21}x_1 & +a_{22}x_2 & +\ldots & +a_{2n}x_n & = & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (\lambda a_{j1} + a_{k1})x_1 & +(\lambda a_{j2} + a_{k2})x_2 & +\ldots & +(\lambda a_{jn} + a_{kn})x_n & = & \lambda b_j + b_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{k1}x_1 & +a_{k2}x_2 & +\ldots & +a_{kn}x_n & = & b_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1}x_1 & +a_{(n-1)2}x_2 & +\ldots & +a_{(n-1)n}x_n & = & b_{(n-1)} \\ a_{n1}x_1 & +a_{n2}x_2 & +\ldots & +a_{nn}x_n & = & b_n \end{array}$$

The above system can be expressed by the matrix equation $A^{'}x = b^{'}$

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ (\lambda a_{j1} + a_{k1}) & (\lambda a_{j2} + a_{k2}) & \ldots & (\lambda a_{jn} + a_{kn}) \\ \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & \ldots & a_{kn} \\ \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \ldots & a_{(n-1)n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_k \\ \vdots \\ x_{(n-1)} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \lambda b_j + b_k \\ \vdots \\ b_j \\ \vdots \\ b_{(n-1)} \\ b_n \end{bmatrix}$$

Operating on rows, it is better to use the concept of so called "augmented matrix" which contains the LHS quadratic matrix plus one column with the RHS elements, that can be written as follows

$$A_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \\ (\lambda a_{j1} + a_{k1}) & (\lambda a_{j2} + a_{k2}) & \dots & (\lambda a_{jn} + a_{kn}) & (\lambda b_{jn} + b_{kn}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kn} & b_k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

4.2.2 Gaussian elimination method

Working with the augmented matrix $A^{(aug)}$ for the system Eq. (4.1), we must check that the term a_{11} is not equal to 0. It can be done applying, so called, the "partial pivoting" technique. When $a_{11} \neq 0$, we can introduce the finite scalar

$$\lambda_{21} = -\frac{a_{21}}{a_{11}}, \tag{4.12}$$

and perform a row operation, where the summation of LH sides and RH sides are carried out separately,

$$\begin{array}{rrrrcl} \lambda_{21}(a_{11}x_1 & +a_{12}x_2 & +\dots & +a_{1n}x_n & = & b_1) \\ +(a_{21}x_1 & +a_{22}x_2 & +\dots & +a_{2n}x_n & = & b_2) \end{array}$$

that results in the replacement of the second row by the equation

$$(a_{21}+\lambda_{21}a_{11})x_1+(a_{22}+\lambda_{21}a_{12})x_2+\cdots+(a_{2n}+\lambda_{21}a_{1n})x_n = b_2+\lambda_{21}b_1. \tag{4.13}$$

Note that the coefficient next to x_1 equals to zero that can be seen after elementary algebra

$$a_{21} + \lambda_{21}a_{11} = a_{21} - \frac{a_{21}}{a_{11}}a_{11} = a_{21} - a_{21} = 0$$

Following that we can write the augmented matrix as

$$A^{(2,1)}_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & (a_{22} + \lambda_{21}a_{12}) & \dots & (a_{2n} + \lambda_{21}a_{1n}) & b_2 + \lambda_{21}b_1 \\ a_{31} & a_{32} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

This can be rewritten in the compact form

$$A^{(2,1)}_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a^{(2,1)}_{22} & \dots & a^{(2,1)}_{2n} & b^{(2,1)}_2 \\ a_{31} & a_{32} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

substituting $(a_{22} + \lambda_{21}a_{12})$ by $a^{(2,1)}_{22}$, etc.

Repeating the above operations for the new λ_{31} expressed by $\lambda_{31} = -\frac{a_{31}}{a_{11}}$, the augmented matrix can be written as

$$A^{(3,1)}_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a^{(2,1)}_{22} & \dots & a^{(2,1)}_{2n} & b^{(2,1)}_2 \\ 0 & a^{(3,1)}_{32} & \dots & a^{(3,1)}_{3n} & b^{(3,1)}_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

This process of row operations can be continued until all elements of the first column become zero except a_{11}. The augmented matrix corresponding to this stage will be

$$A^{(n,1)}_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a^{(2,1)}_{22} & \dots & a^{(2,1)}_{2n} & b^{(2,1)}_2 \\ 0 & a^{(3,1)}_{32} & \dots & a^{(3,1)}_{3n} & b^{(3,1)}_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a^{(n-1,1)}_{(n-1)2} & \dots & a^{(n-1,1)}_{(n-1)n} & b^{(n-1,1)}_{(n-1)} \\ 0 & a^{(n,1)}_{n2} & \dots & a^{(n,1)}_{nn} & b^{(n,1)}_n \end{bmatrix}$$

Getting all elements of the first column equal to zero apart of the first one, we can move to the second column and carry out row operations placing zeros below the element $(2, 2)$. If $a^{(2,1)}_{22} \neq 0$, then $\lambda_{32} = -\frac{a^{(3,1)}_{32}}{a^{(2,1)}_{22}}$ and after the row operation we obtain

$$a^{(3,2)}_{32} = a^{(3,1)}_{32} + \lambda_{32}a^{(2,1)}_{22} = 0.$$

Following this, the augmented matrix is

$$A^{(3,2)}_{(aug)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{13} & a_{1n} & b_1 \\ 0 & a^{(2,1)}_{22} & a^{(2,1)}_{23} & \dots & a^{(2,1)}_{2n} & b^{(2,1)}_2 \\ 0 & 0 & a^{(3,2)}_{33} & \dots & a^{(3,2)}_{3n} & b^{(3,2)}_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a^{(n,1)}_{n2} & a^{(n,1)}_{n3} & \dots & a^{(n,1)}_{nn} & b^{(n,1)}_n \end{bmatrix}$$

Continuing the row operation until replacing all elements of the second column situated below the element $(2,2)$ by zeros, we obtain

$$A_{(aug)}^{(n,2)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{13} & a_{1n} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \dots & a_{2n}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & a_{33}^{(3,2)} & \dots & a_{3n}^{(3,2)} & b_3^{(3,2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & a_{n3}^{(n,2)} & \dots & a_{nn}^{(n,2)} & b_n^{(n,2)} \end{bmatrix}$$

Then we can move to the third column and perform row operations placing zeros below the element $(3,3)$, and continuing such process step by step we can arrive to the $(n-1)$ column, where we can replace only one element below the position $((n-1),(n-1))$. The corresponding augmented matrix has the upper triangular form

$$A_{(aug)}^{(n,n-1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1(n-1)} & a_{1n} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \dots & a_{2(n-1)}^{(2,1)} & a_{2n}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & a_{33}^{(3,2)} & \dots & a_{3(n-1)}^{(3,2)} & a_{3n}^{(3,2)} & b_3^{(3,2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{(n-1)(n-1)}^{(n-1,n-2)} & a_{(n-1)n}^{(n-1,n-2)} & b_{n-1}^{(n-1,n-2)} \\ 0 & 0 & 0 & \dots & 0 & a_{nn}^{(n,n-1)} & b_n^{(n,n-1)} \end{bmatrix}$$

Writing out the system of equations on the base the above augmented matrix

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + \quad & a_{1n}x_n = \quad b_1 \\ a_{22}^{(2,1)}x_2 + a_{23}^{(2,1)}x_3 + \cdots + \quad & a_{2n}^{(2,1)}x_n = \quad b_2^{(2,1)} \\ a_{33}^{(3,2)}x_3 + \cdots + \quad & a_{3n}^{(3,2)}x_n = \quad b_3^{(3,2)} \\ \vdots \quad & \vdots \\ a_{n-1,n-1}^{(n-1,n-2)}x_{n-1} + \quad & a_{n-1,n}^{(n-1,n-2)}x_n = \quad b_{n-1}^{(n-1,n-2)} \\ & a_{n,n}^{(n,n-1)}x_n = \quad b_n^{(n,n-1)} \end{aligned}$$

we can note that the last equation can be immediately solved by dividing both sides by $a_{nn}^{(n,n-1)}$

$$x_n = \frac{b_n^{(n,n-1)}}{a_{nn}^{(n,n-1)}} \tag{4.14}$$

Working backward, we can solve

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

for x_{n-1}

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

This solution process is continued until the evaluation of x_1 and is called the backward substitution.

Example (4.1)♣

Compute the solution of a non-trivial system

$$\begin{array}{rcrcrcl} x_1 & + & 2x_2 & + & 3x_3 & = & 6 \\ 2x_1 & + & 2x_2 & + & 3x_3 & = & 7 \\ x_1 & + & 4x_2 & + & 4x_3 & = & 9 \end{array}$$

We can do this in three steps:

- subtract two times the first equation from the second equation and subtracting the first equation from the third equation and obtain

$$\begin{array}{rcrcrcr} x_1 & + & 2x_2 & + & 3x_3 & = & 6 \\ 0x_1 & - & 2x_2 & - & 3x_3 & = & -5 \\ 0x_1 & + & 2x_2 & + & x_3 & = & 3 \end{array}$$

- add the second equation to the third to obtain

$$\begin{array}{rcrcrcr} x_1 & + & 2x_2 & + & 3x_3 & = & 6 \\ 0x_1 & - & 2x_2 & - & 3x_3 & = & -5 \\ 0x_1 & + & 0x_2 & - & 2x_3 & = & -2 \end{array}$$

- substitute $x_3 = 1$ from the third equation back to the second equation gives the equation for x_2 and substituting both evaluated unknowns back into the first equation gives an equation for x_1, i.e.

$$\begin{array}{rcccccl} x_3 & = & 1 & & & & \\ x_2 & = & \frac{-5+3x_3}{-2} & & = & \frac{-5+3}{-2} = 1 & \\ x_1 & = & 6 - 2x_2 - 3x_3 & = & \frac{6-2-3}{1} & = & 1 \end{array}$$

We may express the above system in terms of a matrix $\mathbf{A}$ and two vectors: an unknown vector $\mathbf{x}$, and the right hand side $\mathbf{b}$, i.e. $\mathbf{Ax} = \mathbf{b}$, and repeat the row operations on the matrix and the RHS vector. From the system

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 1 & 4 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 7 \\ 9 \end{bmatrix}$$

we can subtract two times first row from the second row, and subtract the first row from the third row to obtain

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & -3 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -5 \\ 3 \end{bmatrix}$$

Dividing the second row through by -2 and adding the second and third rows, we get

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{3}{2} \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ \frac{5}{2} \\ 3 \end{bmatrix}$$

Subtracting the second row times $a_{32} = 2$ from the third row, we obtain

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{3}{2} \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ \frac{5}{2} \\ -2 \end{bmatrix}$$

Dividing the last row by $a_{33} = -2$ results in

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & \frac{3}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ \frac{5}{2} \\ 1 \end{bmatrix}$$

We can retrieve the value $x_3 = 1$ from the last row of the matrix equation. ♠

The second example is less intuitive and strictly follows the Gaussian elimination procedure

Example (4.2)♣

Consider the set of equations

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & = & 4 \\ 2x_1 & + & x_2 & + & 3x_3 & = & 7 \\ 3x_1 & + & x_2 & + & 6x_3 & = & 2 \end{array}$$

for which the augmented matrix is

$$(\mathbf{A}, \mathbf{b}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 2 & 1 & 3 & 7 \\ 3 & 1 & 6 & 2 \end{bmatrix}$$

Since $a_{11} \neq 0$ we can define $\lambda_{21} = -\frac{a_{21}}{a_{11}} = -2$ and perform the row operation

$$(\mathbf{A}^{(2,1)}, \mathbf{b}^{(2,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 2-(2)(1) & 1-(2)(1) & 3-(2)(1) & 7-(2)(4) \\ 3 & 1 & 6 & 2 \end{bmatrix}$$

resulting in

$$(\mathbf{A}^{(2,1)}, \mathbf{b}^{(2,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 3 & 1 & 6 & 2 \end{bmatrix}$$

Similarly as for the evaluation of λ_{21} we can define $\lambda_{31} = -\frac{a_{31}}{a_{11}} = -3$ and then the augmented matrix is

$$(\mathbf{A}^{(3,1)}, \mathbf{b}^{(3,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 3-(3)(1) & 1-(3)(1) & 6-(3)(1) & 2-(3)(4) \end{bmatrix}$$

that finally can be written as

$$(\mathbf{A}^{(3,1)}, \mathbf{b}^{(3,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & -2 & 3 & -10 \end{bmatrix}$$

Using $\lambda_{32} = -\frac{a_{32}^{(3,1)}}{a_{22}^{(2,1)}} = -\frac{-2}{-1} = -2$ in Eq. (**??**) we can implement zero at the position a_{32}

$$(\mathbf{A}^{(3,2)}, \mathbf{b}^{(3,2)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & -2-(2)(-1) & 3-(2)(1) & -10-(2)(-1) \end{bmatrix}$$

so that the final augmented matrix is

$$(\mathbf{A}^{(3,2)}, \mathbf{b}^{(3,2)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & -8 \end{bmatrix}$$

Following the backward substitution, we can find the last unknown from $x_3 = -\frac{b_3^{(3,2)}}{a_{33}^{(3,2)}} = \frac{8}{1} = 8$. Next, working backward, we can solve for x_2 by using

$$a_{22}^{(2,1)} x_2 + a_{23}^{(2,1)} x_3 = b_2^{(2,1)}$$

and reworking that we have

$$x_2 = \frac{b_2^{(2,1)} - a_{23}^{(2,1)} x_3}{a_{22}^{(2,1)}} = \frac{-1-(1)(-8)}{-1} = -7$$

Finally, we can solve for x_1 by using

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$

that results in

$$x_1 = \frac{4-(1)(-7)-(1)(-8)}{1} = 19$$

Therefore, the solution vector is

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ -7 \\ -8 \end{bmatrix}$$

♠

4.2.3 Pivoting

Pivoting is a process performed on a matrix in order to improve numerical stability of Gaussian elimination by placing a particularly "good" element in the diagonal position. The element in the diagonal of a matrix by which other elements are divided, is called the pivot element [56]. Dividing by small numbers is critical for the stability of the method. This problem can be eliminated quite easy by the row exchange. At first we will consider the example.

Example (4.3)♣ Conduct the partial pivoting for the following matrix

$$A_{(aug)}^{(n,n-1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1(n-1)} & a_{1n} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \dots & a_{2(n-1)}^{(2,1)} & a_{2n}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & p_1 & \dots & a_{3(n-1)}^{(3,2)} & a_{3n}^{(3,2)} & b_3^{(3,2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & p_{n-1} & \dots & a_{(n-1)(n-1)}^{(n-1,n-2)} & a_{(n-1)n}^{(n-1,n-2)} & b_{n-1}^{(n-1,n-2)} \\ 0 & 0 & p_n & \dots & 0 & a_{nn}^{(n,n-1)} & b_n^{(n,n-1)} \end{bmatrix}$$

In Gauss elimination we choose p_1 as the next pivot subject that it is nonzero and we must swap rows when $p_1 = 0$. In the Gaussian elimination with the partial pivoting we should bring the row with biggest $|p_i|$ into the pivoting position, i.e. if $|p_i| = max_{k=1,2,\dots,n}|p_k|$, then the row of p_1 must be swap with the row of p_i and p_i will be used as a pivot. This simple row operation result in significant improvement of the stability of the Gaussian elimination. We can repeat the above row operation with the real numbers

$$\begin{bmatrix} 1 & 5 & 2 & 6 & 8 & 3 & \dots & 5 \\ 0 & 10^{-7} & 1 & 11 & 231 & 13 & \dots & 6 \\ 0 & 8 & 5 & 6 & -9 & 3 & \dots & 17 \\ 0 & 4 & 3 & -4 & 4 & 7006 & \dots & 14 \\ 0 & 12 & 2 & 9 & 31 & -3 & \dots & 1 \\ 0 & -200 & 25 & 4 & 23 & 76 & \dots & 3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 9 & 61 & 6 & -5 & 3 & \dots & 91 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ \vdots \\ b_n \end{bmatrix}$$

The largest element in the second column is $|-200|$ placed at the sixth row. We can bring it to the diagonal at the position $(2, 2)$ by swapping the second

row with the sixth one. This operation results in

$$\begin{bmatrix} 1 & 5 & 2 & 6 & 8 & 3 & \dots & 5 \\ 0 & -200 & 25 & 4 & 23 & 76 & \dots & 3 \\ 0 & 4 & 3 & -4 & 4 & 7006 & \dots & 14 \\ 0 & 8 & 5 & 6 & -9 & 3 & \dots & 17 \\ 0 & 12 & 2 & 9 & 31 & -3 & \dots & 1 \\ 0 & 10^{-7} & 1 & 11 & 231 & 13 & \dots & 6 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 9 & 61 & 6 & -5 & 3 & \dots & 91 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_6 \\ b_3 \\ b_4 \\ b_5 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

♠

The pivoting technique is designed to avoid the problem, when $a_{11} = 0$ in $\lambda_{21} = \frac{a_{21}}{a_{11}}$ for the augmented matrix

$$(\mathbf{A}^{(2,1)}\mathbf{b}^{(2,1)}) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ (a_{21} - \lambda_{21}a_{11}) & (a_{22} - \lambda_{21}a_{12}) & \dots & (a_{2n} - \lambda_{21}a_{1n}) & b_2 - \lambda_{21}b_1 \\ a_{31} & a_{32} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

At first we can examine the first column of matrix $\mathbf{A}$ and find the row j that contains the value with the largest absolute value. The procedure can be written as in the above example or in the form:

$$|a_{ji}| \geq |a_{k1}| \quad \text{for all} \quad k = 1, 2, \dots, n.$$

Following this row identification, the first row can be replaced by the j-th one and the augmented matrix is now

$$(\hat{\mathbf{A}}, \hat{\mathbf{b}}) = \begin{bmatrix} a_{j1} & a_{j2} & \dots & a_{jn} & b_j \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & a_{(n-1)n} & b_{(n-1)} \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

The full pivoting is conducted by both row and columns. In the Example 4.3, the largest value occurs at the position $(3, 6)$ and is 7006. It can be brought onto the diagonal by interchanging the second and six columns and rows two and three. In addition the column number six should be exchanged

with the column fifth. The final result is

$$\begin{bmatrix} 1 & 3 & 2 & 6 & 5 & 58 & \dots & 5 \\ 0 & 7006 & 3 & -4 & 4 & 4 & \dots & 14 \\ 0 & 76 & 25 & 4 & -200 & 23 & \dots & 3 \\ 0 & 3 & 5 & 6 & 8 & -9 & \dots & 17 \\ 0 & -3 & 2 & 9 & 12 & 31 & \dots & 1 \\ 0 & 13 & 1 & 11 & 10^{-7} & 231 & \dots & 6 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 3 & 61 & 6 & 9 & -5 & \dots & 91 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_6 \\ b_3 \\ b_4 \\ b_5 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

4.2.4 Error analysis of Gaussian elimination

Following Wilkinson [59] (see also [42]) the relative error for the solution of a linear system by Gaussian elimination satisfies the inequality

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \leq n^{\mathsf{O}_{(1)}} \kappa(\mathbf{A}) \rho_L(\mathbf{A}) \rho_U(\mathbf{A}) \epsilon \tag{4.15}$$

where $\|\cdot\|$ are so called spectral norms which will be described in Appendix 1, $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$, $\rho_L(\mathbf{A})$ and $\rho_U(\mathbf{A})$ are the growth factors, ϵ is the machine precision, and $n^{\mathsf{O}_{(1)}}$ is the polynomial factor which depends on the norms of the condition number and growth factors. The condition number of matrix in Eq. (4.15) is given by Eq. (4.3) with spectral norms not necessary to be $\|\cdot\|_\infty$. The condition number measures how much the solution of $\mathbf{Ax} = \mathbf{b}$ varies respectively to slight changes of the matrix $\mathbf{A}$ and the RHS vector $\mathbf{b}$. The growth factors are contributing to the Gaussian elimination error and they are defined by

$$\begin{aligned} \rho_L(\mathbf{A}) &= \|\mathbf{L}\| \\ \rho_U(\mathbf{A}) &= \tfrac{\|\mathbf{U}\|}{\|\mathbf{A}\|} \end{aligned} \tag{4.16}$$

Factors $\rho_L(\mathbf{A})$ and $\rho_U(\mathbf{A})$ measure how large intermediate entries become as Gaussian elimination is progressing. We should mention that partial pivoting eliminates the growth of $\mathbf{L}$ because entries are bounded. In the worse case, the factor $\rho_U(\mathbf{A})$ could grow exponentially with n and a tight bound on this factor is 2^{n-1} in the max-norm. The definition of max-norm is given in Appendix 2.

4.3 Gauss-Jordan elimination

In the method of Gauss-Jordan elimination [36], the process of elimination terminated in the Gauss elimination method on the level illustrated by Eq. (**??**), is continued further by placing zeros above the diagonal. To replace the element $(n-1, n)$ by zero, the last two equation are used.

$$\begin{array}{rcrcl} t_{n-1,n-1}x_{n-1} & + & t_{n-1,n}x_n & = & d_{n-1} \\ & & t_{nn}x_n & = & d_n \end{array}$$

where t_{ij} are elements of the triangular augmented matrix and d_i are elements of the last column, i.e. the RHS vector for the system. Defining $\lambda_{n-1,n} = \frac{t_{n-1,n}}{t_{nn}}$, we can perform the row operation, i.e. subtract the LHS and RHS separately,

$$\begin{array}{rcl} t_{n-1,n-1}x_{n-1} + t_{n-1,n}x_n & = & d_{n-1} \\ \lambda_{n-1,n}t_{nn}x_n & = & \lambda_{n-1,n}d_n \end{array}$$

and replace the $(n-1)$-st equation by

$$t_{n-1,n-1}x_{n-1} + (t_{n-1,n} - \lambda_{n-1,n}t_{nn})x_n = d_{n-1} - d_n\lambda_{n-1,n} \tag{4.17}$$

Following this row operation, the set of equations becomes

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} & \dots & t_{1,n-1} & t_{1n} \\ & t_{22} & t_{23} & \dots & t_{2,n-1} & t_{2n} \\ & & t_{33} & \dots & t_{2,n-1} & t_{3n} \\ & & \vdots & \vdots & \vdots & \vdots \\ & & & & t^{(1)}_{(n-1),(n-1)} & 0 \\ & & & & & t_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(n-1)} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d^{(1)}_{(n-1)} \\ d_n \end{bmatrix}$$

where t_{ij} is one of non-zero elements of the first n columns of $A^{(n,n-1)}_{(aug)}$ and d_i is one of elements of the last column of this augmented matrix in Eq. (**??**). This elimination process can be continued until the set of equations will get the diagonal form

$$\begin{bmatrix} t^{(n-1)}_{11} & 0 & 0 & \dots & 0 & 0 \\ 0 & t^{(n-2)}_{22} & 0 & \dots & 0 & 0 \\ 0 & 0 & t^{(n-3)}_{33} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & t^{(1)}_{(n-1),(n-1)} & 0 \\ 0 & 0 & 0 & \dots & 0 & t_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(n-1)} \\ x_n \end{bmatrix} = \begin{bmatrix} d^{(n-1)}_1 \\ d^{(n-2)}_2 \\ d^{(n-3)}_3 \\ \vdots \\ d^{(1)}_{(n-1)} \\ d_n \end{bmatrix}$$

This can be converted into the form with the identity matrix after dividing each equation by its single coefficient

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(n-1)} \\ x_n \end{bmatrix} = \begin{bmatrix} \frac{d_1^{(n-1)}}{t_{11}^{(n-1)}} \\ \frac{d_2^{(n-2)}}{t_{22}^{(n-2)}} \\ \frac{d_3^{(n-3)}}{t_{33}^{(n-3)}} \\ \vdots \\ \frac{d_{(n-1)}^{(1)}}{t_{(n-1),(n-1)}^{(1)}} \\ \frac{d_n}{t_{nn}} \end{bmatrix}$$

that gives immediately the solution to the problem.

4.4 LU factorization

The LU procedure consist in the decomposition of a quadratic $n \times n$ matrix $\mathbf{A}$ into a product of a lower-triangular matrix $\mathbf{L}$, that has elements only on the diagonal and below, and an upper-triangular matrix $\mathbf{U}$, that has elements only on the diagonal and above. The procedure can be also applied for the inverting of a matrix $\mathbf{A}$. The matrix product can be written by

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{A} \tag{4.18}$$

The following theorem [19] is very important for the assessment of the existence of a unique $\mathbf{LU}$ factorization of matrix $\mathbf{A}$:

Theorem: There exists a unique $\mathbf{L} \cdot \mathbf{U}$ factorization of $\mathbf{A} \in \mathbf{R}^{n \times n}$ if and only if a sub-matrix $\mathbf{A}_k$ is non-singular for $k = 1, \dots, (n-1)$. If $\mathbf{A}_k$ is singular for some $1 \leq k \leq (n-1)$ then the factorization may not exist, but if so it is not unique.

Proof Given by induction [19].

The decomposition in Eq. (4.18) is used to solve the linear set of equations expressed by

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b} \tag{4.19}$$

At first, it can be solved for the vector $\mathbf{y}$ such that

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \tag{4.20}$$

and next solving

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \tag{4.21}$$

for $\mathbf{x}$. The advantage of breaking up one linear set into two successive sets lies in the reduction of the number of executions necessary for example for the inverting of a matrix.

For $n \times n$ matrix $\mathbf{A}$ the Eq. (4.18) would look like this

$$\begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ l_{41} & l_{42} & l_{43} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ 0 & 0 & 0 & \dots & u_{4n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ a_{41} & a_{42} & a_{42} & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n2} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

The Eq.(4.20) can be solved by the forward substitution

$$\begin{aligned} y_1 &= \tfrac{b_1}{l_{11}} \\ y_i &= \tfrac{1}{l_{ii}} \left[b_i - \textstyle\sum_{j=1}^{i-1} l_{ij} y_j \right] \quad i = 2, 3, \dots, n \end{aligned}$$

while the Eq.(4.21) can be solved by the back-substitution, that we know from the previous subsection,

$$\begin{aligned} x_n &= \tfrac{y_n}{u_{nn}} \\ x_i &= \tfrac{1}{u_{ii}} \left[y_i - \textstyle\sum_{j=1}^{i-1} u_{ij} y_j \right] \quad i = n-1, n-2, \dots, 1 \end{aligned}$$

The total execution count (TEC) for both Eq.(4.20) and Eq.(4.21) is n^2. But in the case of a matrix inverting, when we have n right hand sides which are the unit column vectors, TEC of Eq.(4.20) reduces from $\frac{1}{2}n^3$ to $\frac{1}{6}n^3$, while TEC of Eq. (4.21) remains unchanged and is still $\frac{1}{2}n^3$. The another advantage of LU decomposition of $\mathbf{A}$ is that once we have $\mathbf{L}$ and $\mathbf{U}$ we can solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}_{(k)}$ for as many RHS, i.e. $\mathbf{b}_{(k)}$, as we like, for one at a time.

The LU decomposition can be realized by one of two most popular direct factorization algorithms:

- Crout's algorithm with all diagonal elements of **U** required to be 1,
- Doolittle's method with all diagonal elements of **L** equal to 1.

Crout's algorithm for LU factorization

The procedure known as Crout's algorithm is the root finding algorithm and a recipe for solving for **L** and **U**, given **A**. Before formulating the algorithm, we should highlight some relations between components of **L**, **U** and **A** matrices. The a_{ij} component of **A** matrix is expressed as a sum beginning always with

$$l_{i1}u_{1j} + \cdots = a_{ij} \tag{4.22}$$

The number of terms in the above sum depends on whether i or j is the smaller number. Therefore, we have three cases

$$\begin{array}{lllll} i < j: & l_{i1}u_{1j} & +l_{i2}u_{2j} & \ldots & +l_{ii}u_{ij} & = a_{ij} \\ i = j: & l_{i1}u_{1j} & +l_{i2}u_{2j} & \ldots & +l_{ii}u_{jj} & = a_{ij} \\ i > j: & l_{i1}u_{1j} & +l_{i2}u_{2j} & \ldots & +l_{ii}u_{jj} & = a_{ij} \end{array}$$

We have $(n^2 + n)$ unknowns l_{ij} and u_{ij} in the above n^2 equations, where the diagonal elements are presented twice. The missing n elements we can assume arbitrary as follows

$$l_{ii} \equiv 1, \quad \text{for} \quad i = 1, \ldots, n \tag{4.23}$$

Since the diagonal elements of **L** matrix are all equal now to one, they do not need to be stored explicitly, and all elements of **L** and **U** can fit into one matrix with n^2 elements.

The Crout's algorithm is —

- set all $l_{ii} = 1$ for $i = 1, 2, \ldots, j$
- for each j in $j = 1, 2, \ldots, n$ execute the following routines:
  - Compute u_{ij} for all $i = 1, 2, \ldots, j$ with the formula

    $$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj},$$

    when $i = 1$ the summation term is taken to mean zero.
  - For all $i = j + 1, j + 2, \ldots, n$ solve for l_{ij}

    $$l_{ij} = \frac{1}{u_{jj}}(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}).$$

Both procedures must be done before increasing j by one. Performing few iterations we would note that every a_{ij} is used only once. That leads to the useful memory allocation: we can store l_{ij} and u_{ij} in the location where a_{ij}'s are stored, i.e.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ l_{31} & l_{32} & u_{33} & \dots & u_{3n} \\ l_{41} & l_{42} & l_{43} & \dots & u_{4n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & u_{nn} \end{bmatrix}$$

Performing Crout's algorithm, the combined matrix of l_{ij} and u_{ij} is filled by columns from left to right, and within each column from top to bottom.

Pivoting is essential for the stability of Crout's method but unfortunately only partial pivoting can be implemented efficiently but fortunately it is enough to ensure stability of the method.

The process of solution $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$ and the decomposition of the matrix is of the order of magnitude On^3. That means that doubling the number of rows and columns of the matrix, the number of arithmetic operations that must be performed will increase by a factor of 2^3. That is one of major problems with the application of the LU decomposition.

Doolittle's algorithm for LU factorization

The LU factorization is equivalent to solve following equations

$$A_{ij} = \sum_{k=1}^{min(i,j)} l_{ik}u_{kj}. \tag{4.24}$$

These equations can be easily solved when they are examined in the right order.

Generally, the problem can be solved for $\mathbf{A} \in \mathbf{R}^{m\times n}$ with $m \geq n$. Then, $\mathbf{L} \in \mathbf{R}^{m\times n}$, where $\mathbf{L}$ is lower triangular, and $\mathbf{U} \in \mathbf{R}^{n\times n}$. Setting $l_{ii} = 1$ and assuming that we know the first $(k-1)$ columns of $\mathbf{L}$ and the first $(k-1)$ rows of $\mathbf{U}$, we can write

$$a_{kj} = l_{k1}u_{1j} + \cdots + l_{k,k-1}u_{k-1,j} + u_{kj}, \quad \text{for} \quad j = k, \dots, n, \tag{4.25}$$

$$a_{ik} = l_{i1}u_{1k} + \cdots + l_{ik}u_{k,k}, \quad \text{for} \quad i = k+1, \dots, m, \tag{4.26}$$

The above equations we can solve for elements: u_{kj} (the first eq.) and l_{ik} (the second eq.), situated correspondingly in the k-th row of $\mathbf{U}$ and then in the k-th column of $\mathbf{L}$. This process is known as Doolittle's method and is written as the following routine with two loops:

for $k = 1, \ldots, n$

for $j = k, \ldots, n$

$$u_{kj} = a_{kj} - \sum_{i=1}^{k-1} l_{ki} u_{ij} \tag{4.27}$$

end

for $i = k + 1, \ldots, m$

$$l_{ik} = \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}}{u_{kk}} \tag{4.28}$$

end

end

The numerical cost of Doolittle's method can be estimated as $[n^2(m - \frac{n}{3})]$ flops.

Example for Crout's and Doolittle's direct factorization methods

The relationship between Doolittle's and Crout's method can be written comprehensively as follows:

$$\begin{array}{lll} \text{if} & \mathbf{L} \cdot \mathbf{U} = \mathbf{A}^T & \text{by Doolittle's method} \\ \text{then} & \mathbf{U}_{Crout} = \mathbf{U}^T & \\ \text{and} & \mathbf{L}_{Crout} = \mathbf{L}^T & \end{array}$$

Example (4.3)♣ Given that an $\mathbf{A}$ has a factorization, determine $\mathbf{L}$ and $\mathbf{U}$ and demonstrate how Crout's and Doolittle's direct factorization methods work in the case of 3×3 matrix $\mathbf{A}$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$= \begin{bmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} \end{bmatrix}$$

Equating appropriate elements of the left and the right matrices, we obtain 9 equations in 12 unknowns: l_{ij}, and u_{ij}. The unique solution can be obtained by

- setting all $l_{ii} = 1$ in Crout's method,
- setting all $u_{ii} = 1$ in Doolittle's method.

Doolittle's method gives

$$\begin{array}{rcccc}
a_{11} & = l_{11}u_{11} & \mapsto & l_{11} = a_{11} \\
a_{21} & = l_{21}u_{11} & \mapsto & l_{21} = a_{21} \\
a_{31} & = l_{31}u_{11} & \mapsto & l_{31} = a_{31} \\
a_{12} & = l_{11}u_{12} & \mapsto & u_{12} = \frac{a_{12}}{l_{11}} \\
a_{13} & = l_{11}u_{13} & \mapsto & u_{13} = \frac{a_{13}}{l_{11}} \\
a_{22} & = l_{21}u_{12} + l_{22}u_{22} & \mapsto & l_{22} = a_{22} - l_{21}u_{12} \\
a_{32} & = l_{31}u_{12} + l_{32}u_{22} & \mapsto & l_{32} = a_{32} - l_{31}u_{12} \\
a_{23} & = l_{21}u_{13} + l_{22}u_{23} & \mapsto & u_{23} = \frac{a_{23} - l_{21}u_{13}}{l_{22}} \\
a_{33} & = l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} & \mapsto & l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}
\end{array}$$

The above scheme can be seen in the order of solutions for the unknowns l_{ij} and u_{ij} as follows:

1. l_{11}, l_{21}, l_{31},
2. u_{12}, u_{13},
3. l_{22}, l_{32},
4. u_{23},
5. l_{33},

That can be generalized and represented by the following sequence

1. l_{11}, l_{21}, $\ldots$, l_{n1},
2. u_{12}, u_{13}, $\ldots$, u_{1n}
3. l_{22}, l_{32}, $\ldots$, l_{n2}
4. u_{23}, u_{24}, $\ldots$, u_{2n},
5. l_{33}, l_{43}, $\ldots$, l_{n3},
6. etc.

Crout's method gives

$$\begin{array}{lcll}
a_{11} & = l_{11}u_{11} & \mapsto & u_{11} = a_{11} \\
a_{12} & = l_{12}u_{11} & \mapsto & u_{12} = \frac{a_{12}}{u_{11}} \\
a_{13} & = l_{13}u_{11} & \mapsto & u_{13} = \frac{a_{13}}{u_{11}} \\
a_{21} & = l_{21}u_{11} & \mapsto & l_{21} = \frac{a_{21}}{u_{11}} \\
a_{22} & = l_{21}u_{12} + l_{22}u_{22} & \mapsto & u_{22} = a_{22} - l_{21}u_{12} \\
a_{23} & = l_{21}u_{13} + l_{22}u_{23} & \mapsto & u_{23} = a_{23} - l_{21}u_{13} \\
a_{31} & = l_{31}u_{11} & \mapsto & l_{31} = \frac{a_{31}}{u_11} \\
a_{32} & = l_{31}u_{12} + l_{32}u_{22} & \mapsto & l_{32} = \frac{a_{32} - l_{31}u_{12}}{u_{22}} \\
a_{33} & = l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} & \mapsto & u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}
\end{array}$$

In the order of solutions for the unknowns l_{ij} and u_{ij}, the above scheme can be seen as follows:

1. u_{11}, u_{12}, u_{13},
2. l_{21},
3. u_{22}, u_{23},
4. l_{31}, l_{32}
5. u_{33},

♠

Chapter 5

Stationary iterative methods for solving linear systems

We have the choice between direct and iterative methods for the solution of a linear system of equations written in the form of the matrix/vector equation $\mathbf{Ax} = \mathbf{b}$, with non-singular matrix $n \times n$, $\mathbf{A}$. The arguments for using iterative methods are based on economy of computer storage and/or CPU time. The usage of iterative methods requires some expertise. But it should be noted that it would be unreasonable to apply one of iterative methods for the solution of a given linear system when computer storage and CPU are not important for our computing. Dense linear systems and sparse systems with a suitable non-zero structure are usually solved by one of direct methods based on the Gaussian elimination. Such methods lead to the exact solution of a given linear system after finite and fixed CPU. Pivoting strategies help to control rounding errors. The problem arise when the direct solution scheme becomes too expensive as it is in the example described by H. Simon [46] which provides very interesting evaluation of CPU time. He predicted that solving a linear problem with 5×10^9 unknowns by the most efficient direct method would require the CPU time estimated up to 520,040 years, provided that the computation can be carried out at a speed of 1 TFLOPs. The same example can be solved by the method of preconditioned conjugate gradients during 575 seconds of CPU time with the same processing speed. The processing speed for the iterative methods can be lower than for direct methods, but obviously the CPU time savings are huge. The ration of the two times is of order n. The requirements for memory space for the iterative methods are smaller by orders of magnitude.

5.1 Jacobi's method

Generally, Jacobi's method is rarely used in practice but is good for illustration of more complicated iterative methods. Recently, it become a little bit more popular for parallel computing schemes. The Jacobi's strategy for solving the system $\mathbf{Ax} = \mathbf{b}$ can be summarized as

- Given the current approximation $\mathbf{x}_1^{(k)} = \{x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots, x_n^{(k)}\}$ evaluate $x_1^{(k+1)}$ from the first equation using the current values for $\{x_2^{(k)}, x_3^{(k)}, \ldots, x_n^{(k)}\}$,
- find the new value $x_i^{(k+1)}$ using old variables $\{x_1^{(k)}, x_2^{(k)}, \ldots, x_{i-1}^{(k)}, x_{i+1}^{(k)}, \ldots, x_n^{(k)}, \}$ in the i-th equation.

Following that the system can be written as

$$\begin{array}{ccccccc}
a_{11}x_1^{(k+1)} & +a_{12}x_2^{(k)} & +a_{13}x_3^{(k)} & +\ldots & +a_{1n}x_n^{(k)} & = & b_1 \\
a_{21}x_1^{(k)} & +a_{22}x_2^{(k+1)} & +a_{23}x_3^{(k)} & +\ldots & +a_{2n}x_n^{(k)} & = & b_2 \\
a_{31}x_1^{(k)} & +a_{32}x_2^{(k)} & +a_{33}x_3^{(k+1)} & +\ldots & +a_{3n}x_n^{(k)} & = & b_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
a_{n1}x_1^{(k)} & +a_{n2}x_2^{(k)} & +a_{n3}x_3^{(k)} & +\ldots & +a_{nn}x_n^{(k+1)} & = & b_n
\end{array}$$

The system in vector and matrix form can be written as

$$\begin{bmatrix}
a_{11} & 0 & \ldots & 0 & 0 \\
0 & a_{22} & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \ldots & a_{(n-1),(n-1)} & 0 \\
0 & 0 & \ldots & 0 & a_{nn}
\end{bmatrix}
\begin{bmatrix}
x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_{(n-1)}^{(k+1)} \\ x_n^{(k+1)}
\end{bmatrix} +$$

$$\begin{bmatrix}
0 & a_{12} & \ldots & a_{1(n-1)} & a_{1n} \\
a_{21} & 0 & \ldots & a_{2(n-1)} & a_{2n} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
a_{(n-1)1} & a_{(n-1)2} & \ldots & 0 & a_{(n-1)n} \\
a_{n1} & a_{n2} & \ldots & a_{n(n-1)} & 0
\end{bmatrix}
\begin{bmatrix}
x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_{(n-1)}^{(k)} \\ x_n^{(k)}
\end{bmatrix} =
\begin{bmatrix}
b_1 \\ b_2 \\ \vdots \\ b_{(n-1)} \\ b_n
\end{bmatrix}$$

Introducing three matrices:

- diagonal

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & \dots & 0 & 0 \\ 0 & a_{22} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a_{(n-1),(n-1)} & 0 \\ 0 & 0 & \dots & 0 & a_{nn} \end{bmatrix}$$

- strict lower triangular

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ a_{21} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & 0 & 0 \\ a_{n1} & a_{n2} & \dots & a_{n(n-1)} & 0 \end{bmatrix}$$

- strict upper triangular

$$\mathbf{U} = \begin{bmatrix} 0 & a_{12} & \dots & a_{1(n-1)} & a_{1n} \\ 0 & 0 & \dots & a_{2(n-1)} & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{(n-1)n} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

The Jacobi method can be written in the form

$$\mathbf{D}\mathbf{x}^{(k+1)} + (\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)} = \mathbf{b} \tag{5.1}$$

from where the unknown vector $\mathbf{x}^{(k+1)}$ can be evaluated as

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(-\mathbf{L} - \mathbf{U})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b} \tag{5.2}$$

The error $\mathbf{e}^{(k)}$ between the true solution $\mathbf{x}$ and the iterative solution $\mathbf{x}^{(k)}$ in each iteration (k) is measured as the difference in the vector space $\mathcal{X}$

$$\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$$

such that contains those three vectors, i.e. $\mathbf{x}, \mathbf{x}^{(k)}, \mathbf{e}^{(k)} \in \mathcal{X}$. To measure the error we need a single real number which can be issued by transforming $\mathbf{e}^{(k)}$ to the real number space $\mathcal{R}$ by applying the l^2 norm commonly used in linear algebra which is also known as the vector norm:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{(x_1^2 + x_2^2 + \cdots + x_n^2)}$$

The best idea for the illustration, how the evaluation of error practically works is to solve the example, by using Jacobi's method, for which we already know the

true solution $\mathbf{x}$. The iteration limit can be defined by limiting $\|\mathbf{e}\| \leq \varepsilon$ or limiting the number of iterations $k \leq \eta$.

{Example (5.1)♣

Solve the system $\mathbf{Ax} = \mathbf{b}$ where

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

and

$$\mathbf{b} = \begin{bmatrix} 0 \\ 2 \\ -3 \end{bmatrix}$$

by using the Jacobi method. The matrix $\mathbf{A}$ can be represented as a sum of three special matrices: lower diagonal, diagonal and upper-diagonal:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{bmatrix} + \begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

They are used to form the following two matrices:

$$\mathcal{D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{bmatrix}, \mathcal{N} = -(\mathbf{L} + \mathbf{U}) = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

The Jacobi's iterative process can be written as

$$\mathbf{x}^{(k+1)} = \mathcal{D}^{-1}\mathcal{N}\mathbf{x}^{(k)} + \mathcal{D}^{-1}\mathbf{b}$$

where

$$\mathcal{D}^{-1}\mathcal{N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix},$$

$$\mathcal{D}^{-1}\mathbf{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ -3 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix}.$$

The new vector of "unknowns" is

$$\mathbf{x}^{(k+1)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \mathbf{x}^{(k)} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix}$$

Assuming the initial approximation $\mathbf{x} = [0 \quad 0 \quad 0]^T$ and following the algorithm can produce five first iterations

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.6667 \\ 1.5 \end{bmatrix};$$

$$\mathbf{x}^{(2)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.6667 \\ 1.5 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 0.8333 \\ 0.6667 \\ 1.5 \end{bmatrix};$$

$$\mathbf{x}^{(3)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.8333 \\ 0.6667 \\ 1.5 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 0.8333 \\ 0.9444 \\ 1.9167 \end{bmatrix};$$

$$\mathbf{x}^{(4)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.8333 \\ 0.9444 \\ 1.9167 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 0.9723 \\ 0.9444 \\ 1.9167 \end{bmatrix};$$

$$\mathbf{x}^{(5)} = \begin{bmatrix} 0 & 1 & -1 \\ \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.9723 \\ 0.9444 \\ 1.9167 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{3}{2} \end{bmatrix} = \begin{bmatrix} 0.9723 \\ 0.9907 \\ 1.9861 \end{bmatrix};$$

♠

5.2 Gauss-Seidel method

This method was invented independently by two prominent German mathematicians: Carl Friedrich Gauss and Philipp Ludwig von Seidel. The method is based on Jacobi's method enriched by the replacement of elements of vector $\mathbf{x}^{(k)}$ by elements of the new vector $\mathbf{x}^{(k+1)}$ immediately when they become available. Complying with this directive the system of linear equation can be expressed by

$$\begin{array}{ccccccc} a_{11}x_1^{(k+1)} & +a_{12}x_2^{(k)} & +a_{13}x_3^{(k)} & +\ldots & +a_{1n}x_n^{(k)} & = & b_1 \\ a_{21}x_1^{(k+1)} & +a_{22}x_2^{(k+1)} & +a_{23}x_3^{(k)} & +\ldots & +a_{2n}x_n^{(k)} & = & b_2 \\ a_{31}x_1^{(k+1)} & +a_{32}x_2^{(k+1)} & +a_{33}x_3^{(k+1)} & +\ldots & +a_{3n}x_n^{(k)} & = & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1}x_1^{(k+1)} & +a_{n2}x_2^{(k+1)} & +a_{n3}x_3^{(k+1)} & +\ldots & +a_{nn}x_n^{(k+1)} & = & b_n \end{array}$$

The system in vector and matrix form can be written as

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 & 0 \\ a_{12} & a_{22} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1),1} & a_{(n-1),2} & \dots & a_{(n-1),(n-1)} & 0 \\ a_{n1} & a_{n2} & \dots & a_{n,(n-1)} & a_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_{(n-1)}^{(k+1)} \\ x_n^{(k+1)} \end{bmatrix} +$$

$$\begin{bmatrix} 0 & a_{12} & \dots & a_{1(n-1)} & a_{1n} \\ 0 & 0 & \dots & a_{2(n-1)} & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{(n-1)n} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_{(n-1)}^{(k)} \\ x_n^{(k)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{(n-1)} \\ b_n \end{bmatrix}$$

The matrix form of the above relation is

$$(\mathbf{L} + \mathbf{D})\mathbf{x}^{(k+1)} + \mathbf{U}\mathbf{x}^{(k)} = \mathbf{b} \tag{5.3}$$

from where the unknown vector $\mathbf{x}^{(k+1)}$ can be evaluated as

$$\mathbf{x}^{(k+1)} = (\mathbf{L} + \mathbf{D})^{-1}(-\mathbf{U}\mathbf{x}^{(k)} + \mathbf{b}) \tag{5.4}$$

where

- diagonal

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & \dots & 0 & 0 \\ 0 & a_{22} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a_{(n-1),(n-1)} & 0 \\ 0 & 0 & \dots & 0 & a_{nn} \end{bmatrix}$$

- strict lower triangular

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ a_{21} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \dots & 0 & 0 \\ a_{n1} & a_{n2} & \dots & a_{n(n-1)} & 0 \end{bmatrix}$$

- strict upper triangular

$$\mathbf{U} = \begin{bmatrix} 0 & a_{12} & \dots & a_{1(n-1)} & a_{1n} \\ 0 & 0 & \dots & a_{2(n-1)} & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{(n-1)n} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Example (5.1)♣

At first the GS method can be illustrated by a simple example with two-equation system

$$\begin{aligned} x_1 &= f_1(x_2) \\ x_2 &= f_2(x_1) \end{aligned}$$

The algorithm proceeds as follows:

1. The first iteration
   - Guess a value of x_2 and substitute this to the RHS of the first equation,
   - Solve the first equation for x_1,
     $$x_1^{(1)} = f_1(x_2^{(0)})$$
   - Insert the freshly evaluated $x_1^{(1)}$ in the RHS of the second equation,
   - Solve for $x_2^{(1)}$
     $$x_2^{(1)} = f_1(x_1^{(1)})$$
     discarding the initial value for x_2,
2. The second iteration
   - Solve
     $$x_1^{(2)} = f_1(x_2^{(1)})$$
     for $x_1^{(2)}$
   - Substitute $x_1^{(2)}$ to the RHS and solve
     $$x_2^{(2)} = f_2(x_1^{(2)})$$
     for $x_2^{(2)}$

3. n-th iteration

   - Solve

     $$x_1^{(n)} = f_1(x_2^{(n-1)})$$

     for $x_1^{(n)}$
   - Substitute $x_1^{(n)}$ to the RHS and solve

     $$x_2^{(n)} = f_2(x_1^{(n)})$$

     for $x_2^{(n)}$

The iteration process is terminated whether the number of iterations has reached a pre-set number or the evaluated values are changing by "very little", i.e. within a tolerance range. ♠

Example (5.1)♣

Solve the system $\mathbf{Ax} = \mathbf{b}$ where

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

and

$$\mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

by using the Gauss-Seidel method. Define matrices:

$$\mathcal{D} = \mathbf{L} + \mathbf{D} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Let assume arbitrary that the unknown vector in the beginning of the iteration process is

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix},$$

and following Eq. 7.31 the system of equations can be expressed as follows

$$\mathcal{D}\mathbf{x}^{(2)} = \mathbf{b} - \mathbf{U}\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}.$$

Solving the equation

$$\mathbf{x}^{(2)} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

by forward substitution results in

$$\mathbf{x}^{(2)} = \begin{bmatrix} 1.500 \\ 2.250 \\ 1.625 \end{bmatrix}$$

The third iteration gives

$$\mathbf{x}^{(3)} = \mathcal{D}^{-1}(\mathbf{b} + \mathbf{U}\mathbf{x}^{(2)}) = \begin{bmatrix} 1.6250 \\ 1.6250 \\ 1.3125 \end{bmatrix}$$

The next ten iterations produce the following results

$$\mathbf{x}^{(4)} = \begin{bmatrix} 1.3125 \\ 1.3125 \\ 1.1563 \end{bmatrix}; \mathbf{x}^{(5)} = \begin{bmatrix} 1.1563 \\ 1.1563 \\ 1.0781 \end{bmatrix}; \mathbf{x}^{(6)} = \begin{bmatrix} 1.0781 \\ 1.0781 \\ 1.0391 \end{bmatrix}; \ldots \mathbf{x}^{(13)} = \begin{bmatrix} 1.0002 \\ 1.0002 \\ 1.0001 \end{bmatrix}$$

Considering that the exact solution of the system is

$$\mathbf{x}_{exact} = \begin{bmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$$

the iteration process leads the sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \ldots$ to the exact solution ♠

5.3 Convergence of stationary iterative methods

Assessing the quality of the two above mentioned iterative methods one should attempt to answer two following questions:

- $\mathbf{Q_1}$: When or more precisely, under which conditions the iterative methods produce a sequence of approximations $\{\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ that converges to the true solution $\hat{\mathbf{x}}$ of linear system?

- **Q_2**: How quickly the approximations approach the true solution $\hat{\mathbf{x}}$?

The solution to the system $\mathbf{Ax} = \mathbf{b}$ has the general form

$$\mathbf{x}^{(k+1)} = \mathcal{N}\mathbf{x}^{(k)} + \mathcal{B} \tag{5.5}$$

where:

Iteration method	$\mathcal{N}$	$\mathcal{B}$
Gauss-Seidel	$-(\mathbf{L}+\mathbf{D})^{-1}\mathbf{U}$	$(\mathbf{L}+\mathbf{D})^{-1}\mathbf{b}$
Jacobi	$-\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$	$\mathbf{D}^{-1}\mathbf{b}$

and $\mathcal{N}$ is called as the iteration matrix. Assuming that the current approximation $\mathbf{x}^{(k)}$ is the exact solution, the iterative method should also produce a next iteration $\mathbf{x}^{(k+1)}$ that is also the next exact solution and then we can write

$$\hat{\mathbf{x}} = \mathcal{N}\hat{\mathbf{x}} + \mathcal{B} \tag{5.6}$$

Subtracting Eq.5.5 from Eq.5.6 results in

$$\hat{\mathbf{x}} - \mathbf{x}^{(k+1)} = \mathcal{N}(\hat{\mathbf{x}} - \mathbf{x}^{(k)}) \tag{5.7}$$

Introducing the current error as

$$\mathbf{e}^{(k)} = \hat{\mathbf{x}} - \mathbf{x}^{(k)} \tag{5.8}$$

we can write the recurrence formula

$$\mathbf{e}^{(k)} = \mathcal{N}\mathbf{e}^{(k-1)} = \mathcal{N}\mathcal{N}\mathbf{e}^{(k-2)} = \mathcal{N}^2\mathbf{e}^{(k-2)} = \cdots = \mathcal{N}^k\mathbf{e}^{(0)}. \tag{5.9}$$

The error $\mathbf{e}^{(k)}$ is a vector and using vector norm in Euclidean vector space we can get a single real number which measures the distance between a current $\mathbf{x}^{(k)}$ and the exact solution $\hat{\mathbf{x}}$, i.e.

$$\|\mathbf{e}^{(k)}\| = \|\mathcal{N}^k\mathbf{e}^{(0)}\| \leq \|\mathcal{N}\|^k\|\mathbf{e}^{(0)}\|. \tag{5.10}$$

It can be easily noted that if $\|\mathcal{N}\| \leq 1$ then $\|\mathbf{e}^{(k)}\| \Rightarrow 0$, i.e. converges to zero. So, what condition for the convergence of $\{\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ can be derived from there? The proper answer is: The smaller $\|\mathcal{N}\|$ is, the faster the error will converge to 0 and the faster the approximation will approach the true solution. But what will happen when $\|\mathcal{N}\| \geq 1$? Obviously, the error will simply increase and approximations will move away from the true solution.

Looking for the features of matrix $\mathcal{N}$ it could be encountered that the condition $\|\mathcal{N}\| \leq 1$ is guaranteed when the matrix $\mathbf{A}$ is strictly diagonally dominant. This means that, for each row of $\mathbf{A}$ the absolute value of the diagonal

Method	$\mathcal{N}$
Jacobi	$-\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}) = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ -a_{21} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 \end{bmatrix}$
Gauss-Seidel	$-(\mathbf{L}+\mathbf{D})^{-1}\mathbf{U} = \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ 0 & -\frac{a_{12}a_{21}}{a_{11}a_{22}} \end{bmatrix}$

Table 5.1: Iteration matrices

Method	Eigenvalues	Eigenvectors
Jacobi	$\lambda_1 = \sqrt{\frac{a_{12}a_{21}}{a_{11}a_{22}}}$, $\lambda_2 = -\sqrt{\frac{a_{12}a_{21}}{a_{11}a_{22}}}$	$\mathbf{v}_1 = \begin{bmatrix} 1 \\ -\sqrt{\frac{a_{11}a_{21}}{a_{12}a_{22}}} \end{bmatrix}$ $\mathbf{v}_2 = \begin{bmatrix} 1 \\ \sqrt{\frac{a_{11}a_{21}}{a_{12}a_{22}}} \end{bmatrix}$
Gauss-Seidel	$\lambda_1 = 0$, $\lambda_2 = -\frac{a_{12}a_{21}}{a_{11}a_{22}}$	$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\mathbf{v}_2 = \begin{bmatrix} 1 \\ -\frac{a_{21}}{a_{22}} \end{bmatrix}$

Table 5.2: Eigenvectors for the Jacobi and Gauss-Seidel methods

element is larger than the sum of the absolute values of the off-diagonal elements.

Following [48] the analysis for the Jacobi and Gauss-Seidel methods can be continued further for the general matrix $\mathbf{A}$ of the size 2×2

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

The iteration matrices for the Jacobi and Gauss-Seidel methods are given in the Table 5.1.

We can note that in both cases the diagonal elements of $\mathbf{A}$ must not be zero, i.e. $a_{11} \neq 0$ and $a_{22} \neq 0$.

It is known from the matrix algebra that if an $n \times n$ iteration matrix $\mathcal{N}$ has a full set of n distinct eigenvectors, then any norm of $\mathcal{N}$ equals to the same norm of eigenvalue of the greatest magnitude λ_{max}, i.e. $\|\mathcal{N}\| = \|\lambda_{max}\|$. The iteration matrices for the Jacobi and Gauss-Seidel iteration matrices always have two distinct eigenvectors, so each method is producing converging sequence of $\mathbf{x}^{(k)}$ if all eigenvalues of $\mathcal{N}$ are of magnitude less than 1, i.e. $|\lambda|_i < 1$ for all $i = 1, \ldots, n$. This is true both for λ being real or complex numbers.

Table 5.2 contains eigenvalues and corresponding eigenvectors for the Jacobi and Gauss-Seidel methods.

From this table we can draw that $\|\mathcal{N}\| = \|\lambda_{max}\| < 1$ when $|\frac{a_{12}a_{21}}{a_{11}a_{22}}| < 1$.

Now we can answer the two questions posed at the beginning of this section:

- answer to $\mathbf{Q_1}$: The sequence of $\mathbf{x}^{(k)}$ is converging to the true solution $\hat{\mathbf{x}}$ when the magnitude of the eigenvalues of the iteration matrix are less than unity.

- answer to $\mathbf{Q_2}$: The rate of convergence of $\mathbf{x}^{(k)}$ to the true solution $\hat{\mathbf{x}}$ is the same as the rate of convergence of $\|\mathcal{N}\|^k$ to zero.

Unfortunately, the above assessments for a general system of the magnitude $n \times n$, where $n > 2$, are more complicated as another two features important for matrices should be taken into the consideration: symmetry and positive definiteness. For the systems with $\mathbf{A}$ that is not positive definite or non-symmetric, both methods or one of them may not converge. Fortunately, many matrices arising from the well posed physical models are symmetric and positive definite.

5.4 The SOR method

The Successive Over-relaxation Method (SOR) can be seen as a generalization and improvement of the Gauss-Seidel method. The core idea of SOR is based on the postulate that we should search for the true solution $\hat{\mathbf{x}}$ further along the direction $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ subject that $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|$. Following [48] the SOR method can be derived from the Gauss-Seidel method in few steps:

- GS iteration equation is

$$\mathbf{D}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}$$

- that can be converted to the explicit form

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)})$$

- and after the subtraction $\mathbf{x}^{(k)}$ from both sides it takes the form

$$\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}\mathbf{x}^{(k)} - \mathbf{U}\mathbf{x}^{(k)})$$

- Introducing the new notion: GS-correction defined by

$$\omega(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})_{GS} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}\mathbf{x}^{(k)} - \mathbf{U}\mathbf{x}^{(k)})$$

- we can compute the new value

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}\mathbf{x}^{(k)} - \mathbf{U}\mathbf{x}^{(k)})$$

  where ω is selected from the range $1 < \omega < 2$. For the value $\omega = 1$ the above relation becomes the same as for the GS method.

- Multiplying both sides by $\mathbf{D}$ and dividing by ω results in

$$\frac{1}{\omega}\mathbf{D}\mathbf{x}^{(k+1)} = \frac{1}{\omega}\mathbf{D}\mathbf{x}^{(k)} + (\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}\mathbf{x}^{(k)} - \mathbf{U}\mathbf{x}^{(k)})$$

- Collecting all terms $\mathbf{x}^{(k+1)}$ on the LHS leads to

$$(\mathbf{L} + \frac{1}{\omega}\mathbf{D})\mathbf{x}^{(k+1)} = \frac{1}{\omega}\mathbf{D}\mathbf{x}^{(k)} + (\mathbf{b} - \mathbf{D}\mathbf{x}^{(k)} - \mathbf{U}\mathbf{x}^{(k)}) = (\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U})\mathbf{x}^{(k)} + \mathbf{b}.$$

- Finally solving this for $\mathbf{x}^{(k+1)}$ gives

$$\mathbf{x}^{(k+1)} = (\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}[(\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U})\mathbf{x}^{(k)} + \mathbf{b}].$$

- It can be also written in the same form as for the Jacobi and Gauss-Seidel methods, i.e.

$$\mathbf{x}^{(k+1)} = \mathcal{N}\mathbf{x}^{(k)} + \mathcal{B}$$

where

$$\mathcal{N} = (\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}(\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U})$$

and

$$\mathcal{B} = (\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}\mathbf{b}$$

The previous discussion above the convergence of the Jacobi and Gauss-Seidel methods applies also to the SOR method . So the optimal convergence of the sequence $\{\mathbf{x}^{(k)}\}$ can be achieved by choosing a value of ω that minimizes

$$\|(\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}(\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U})\|$$

5.4.1 Choosing the value of ω

The SOR method can be simplified to the Gauss-Seidel method assuming $\omega = 1$. The SOR fails to converge if $0 < \omega < 2$. Following that the term underrelaxation can be used when $0 < \omega < 1$ as was shown in [26]. Generally, an optimal value of ω cannot be computed in advance. In cases, when it is possible, the numerical cost of such evaluation is usually very high. Sometimes, a heuristic estimate can be used, e.g. $\omega = 2 - \mathcal{O}(h)$, where h is the mesh spacing of the physical domain.

Method	Algorithm for performing iteration $(k+1)$ for $i = 1$ to n
Jacobi	$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(-\mathbf{L} - \mathbf{U})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}$ $x_i^{(k+1)} = x^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)$
Gauss-Seidel	$\mathbf{x}^{(k+1)} = -(\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}\mathbf{x}^{(k)} + (\mathbf{L} + \mathbf{D})^{-1}\mathbf{b}$ $x_i^{(k+1)} = x^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)$
SOR	$\mathbf{x}^{(k+1)} = (\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}(\frac{1}{\omega}\mathbf{D} - \mathbf{D} - \mathbf{U})\mathbf{x}^{(k)} + (\mathbf{L} + \frac{1}{\omega}\mathbf{D})^{-1}\mathbf{b}$ $x_i^{(k+1)} = x^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)$

Table 5.3: Differences between the three already discussed methods

The SOR iteration is guaranteed to converge for any value of ω between 0 and 2 subject that the matrix $\mathbf{A}$ is symmetric and positive definite. In this case, the choice of ω can affect the rate of the SOR iteration convergence. The estimation of the optimal ω that ensures the rate, at which the iteration is converging, can be done employing schemes for adaptive parameter estimation . Adaptive methods are iterative methods that collect the information about the coefficient matrix during the iteration process, and use this for speeding convergence of the SOR iteration scheme.

For some matrices $\mathbf{A}$ which arise from the discretization of elliptic partial differential equations (PDE) there is a direct relationship between the spectra of the Jacobi and SOR iteration matrices. The theoretical optimal value of ω for the SOR method can be determined a' priori by

$$\omega_{opt} = \frac{2}{1 - \sqrt{1 - \rho^2}}, \tag{5.11}$$

where ρ is the spectral radius of the Jacobi iteration matrix. This estimation is seldom done, since the evaluation of the spectral radius of the Jacobi matrix is numerically quite expensive.

5.5 Comparison of three stationary methods

Following [48], the differences between the three already discussed methods is highlighted in the Table 5.3.

5.6 Symmetric Successive Over-Relaxation

The Symmetric Successive Over-Relaxation method (SSOR) combines two SOR steps together in such a way that the resulting iteration matrix is similar to a symmetric matrix subject that the coefficient matrix $\mathbf{A}$ is symmetric. The first step is carried out as in the original SOR method but in the second step the unknowns are updated in the reverse order. The SSOR can be seen also as the method consisting of two steps: a forward SOR step and a backward SOR step. The SSOR can be used as a pre-conditioner for other iterative schemes for symmetric matrices because their convergence rate with the optimal value ω is usually slower than the convergence rate of SOR with optimal ω.

The SSOR iteration can be expressed by the following equation

$$\mathbf{x}^{(k+1)} = \mathbf{B}_1\mathbf{B}_2\mathbf{x}^{(k)} + \omega(2-\omega)(\mathbf{D}-\omega\mathbf{U})^{-1}\mathbf{D}(\mathbf{D}-\omega\mathbf{L})^{-1}\mathbf{b}, \tag{5.12}$$

where

$$\mathbf{B}_1 = (\mathbf{D}-\omega\mathbf{U})^{-1}[\omega\mathbf{L} + (1-\omega)\mathbf{D}],$$

and

$$\mathbf{B}_1 = (\mathbf{D}-\omega\mathbf{L})^{-1}[\omega\mathbf{U} + (1-\omega)\mathbf{D}].$$

The matrix $\mathbf{B}_2$ is the same iteration matrix as for the SOR method and $\mathbf{B}_1$ is the same but with reversed roles of $\mathbf{L}$ and $\mathbf{U}$.

Chapter 6

Nonstationary iterative methods for large linear systems

6.1 Preliminaria

6.1.1 The quadratic form

The quadratic form is a scalar, quadratic function of a vector $\mathbf{x}$ defined by

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \tag{6.1}$$

where $\mathbf{A}$ is a matrix, $\mathbf{x}$ and $\mathbf{b}$ are vectors and c is a scalar constant. It will be shown that $q(\mathbf{x})$ is minimized by the solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ provided that $\mathbf{A}$ has two specific features: it is both symmetric, and positive definite. For the positive-definite $\mathbf{A}$, the surface defined by $\mathbf{q}$ is shaped like a paraboloid bowl. The gradient of a quadratic form is defined by

$$\frac{dq(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial q(\mathbf{x})}{\partial x_1} \\ \frac{\partial q(\mathbf{x})}{\partial x_2} \\ \cdots \\ \frac{\partial q(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{6.2}$$

The gradient is a vector field that points in the direction of greatest increase of $q(\mathbf{x})$ for a given point $\mathbf{x}$. The gradient is zero at the bottom of the paraboloid bowl. The quadratic form can be minimized by setting

$$\frac{dq(\mathbf{x})}{d\mathbf{x}} = \frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b} \tag{6.3}$$

equal to zero. This equation reduces to

$$\frac{dq(\mathbf{x})}{d\mathbf{x}} = \mathbf{A}\mathbf{x} - \mathbf{b} \tag{6.4}$$

subject that $\mathbf{A}$ is symmetric. Setting the above to zero, i.e. $\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$, gives the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ which we are going to solve. Therefore, the solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ is a critical point of $q(\mathbf{x})$. If $\mathbf{A}$ is symmetric and positive definite then the solution $\hat{\mathbf{x}}$ is the minimum of the quadratic form $q(\mathbf{x})$ and then $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be solved by finding such $\mathbf{x}$ that minimizes $q(\mathbf{x})$. When $\mathbf{A}$ appears to be nonsymmetric the solution can be find from the equation $\frac{1}{2}(\mathbf{A}^T + \mathbf{A})\mathbf{x} = \mathbf{b}$ that comes out from Eq.6.3, where the matrix $(\mathbf{A}^T + \mathbf{A})$ is always symmetric.

Following [45] we can explain, why the symmetry and positive definiteness provides matrices with this desired property. To this purpose we should consider the relationship between $q(\tilde{\mathbf{x}})$ and $q(\hat{\mathbf{x}})$, where $\tilde{\mathbf{x}}$ is the arbitrary point and $\hat{\mathbf{x}}$ is the solution of $\hat{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b}$. Writing Eq.6.1 for $\tilde{\mathbf{x}}$ and substituting $\tilde{\mathbf{x}} = \hat{\mathbf{x}} + \mathbf{e}$, where $\mathbf{e}$ is the error, we can get

$$\begin{aligned} q(\hat{\mathbf{x}} + \mathbf{e}) &= \frac{1}{2}(\hat{\mathbf{x}} + \mathbf{e})^T\mathbf{A}(\hat{\mathbf{x}} + \mathbf{e}) - \mathbf{b}^T(\hat{\mathbf{x}} + \mathbf{e}) + c \\ &= \frac{1}{2}\hat{\mathbf{x}}^T\mathbf{A}\hat{\mathbf{x}} + \mathbf{e}^T\mathbf{A}\hat{\mathbf{x}} + \frac{1}{2}\mathbf{e}^T\mathbf{A}\mathbf{e} - \mathbf{b}^T\hat{\mathbf{x}} - \mathbf{b}^T\mathbf{e} + c \\ &= \frac{1}{2}\hat{\mathbf{x}}^T\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}^T\hat{\mathbf{x}} + c + \mathbf{e}^T\mathbf{b} + \frac{1}{2}\mathbf{e}^T\mathbf{A}\mathbf{e} - \mathbf{b}^T\mathbf{e} \\ &= q(\hat{\mathbf{x}}) + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{e}^{\mathbf{T}}\mathbf{A}\mathbf{e} \end{aligned} \tag{6.5}$$

Finally coming back to $\hat{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ the last equation can be written as

$$q(\tilde{\mathbf{x}}) = q(\hat{\mathbf{x}}) + \frac{1}{2}(\tilde{\mathbf{x}} - \hat{\mathbf{x}})^T\mathbf{A}(\tilde{\mathbf{x}} - \hat{\mathbf{x}}) \tag{6.6}$$

if $\mathbf{A}$ is positive definite, then by inequality $\mathbf{d}^T\mathbf{A}\mathbf{d} > 0$ valid for any nonzero $\mathbf{d}$, the latter term is positive for all $\tilde{\mathbf{x}} \neq \hat{\mathbf{x}}$. Moreover, $\hat{\mathbf{x}}$ is a global minimum of $q(\mathbf{x})$.

Intuitively, the quadratic form can be seen as a paraboloid for a positive-definite matrix, but when $\mathbf{A}$ is negative-definite the quadratic form can be seen as the same paraboloid turned upside-down. For a singular $\mathbf{A}$ the solution is not unique but we could identify the set of solutions in the form of line or hyperplane having a uniform value for q. If $\mathbf{A}$ is indefinite, i.e. none of the above, then $\hat{\mathbf{x}}$ is a saddle point and techniques like Steepest Descent (SD) and Conjugate Gradient (CG) could fail. The respective illustrations can be seen eg. in [45] (see there Fig. 5). It should be emphasized that values of $\mathbf{b}$ and c determine the location of the minimum point of the paraboloid but they do not affect the shape of a surface representing the quadratic form.

6.1.2 Eigenvalues and eigenvectors and procedures always giving zero

The CG method needs an intuitive grasp of eigenvectors and eigenvalues and this is why we will discuss the mater following [45] where the concept of eigen-vectors and eigen-values is presented very comprehensible.

In general, multiplication of a positive definite and symmetric matrix $\mathbf{A}$ by a nonzero vector $\mathbf{v}$, produces a new vector $\mathbf{w}$. For the specific case, when $\mathbf{w}$ appears to be exactly $\mathbf{v}$, then looking for the situation from the mechanical point of view, the vector $\mathbf{v}$ does not rotate. The vector $\mathbf{v}$ may change length or reverse its direction but it would not turn sideways. Such specific vector $\mathbf{v}$ is called the eigenvector of the matrix $\mathbf{A}$. In addition, assuming that there exists a real number λ, the following relation is true

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}. \tag{6.7}$$

Eq.6.7 can be seen also as the scaling of an eigenvector. Note that an eigenvector scaled up or down still remains an eigenvector. A ***scaling number*** λ is called an eigenvalue of matrix $\mathbf{A}$.

In general, iterative methods often consist of applying $\mathbf{A}$ several times to a vector $\tilde{\mathbf{x}}^{(i)}$ which is evaluated in each iteration step as an approximate solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$. Note that the matrix $\mathbf{A}$ is applied to $\tilde{\mathbf{x}}^{(i)}$ only once. Vectors $\tilde{\mathbf{x}}^{(i)}$ are slightly different in length but their direction is the same and therefore, from now we can write $\tilde{\mathbf{x}}^{(i)} \approx \tilde{\mathbf{x}}$. Let assume that $\tilde{\mathbf{x}}$ is an eigenvector. When the matrix $\mathbf{A}$ is repeatedly applied to an eigenvector $\tilde{\mathbf{x}}$, the scaling factor λ, i.e. an eigenvalue of $\mathbf{A}$, can be either $|\lambda| < 1$ or $|\lambda| > 1$. If $|\lambda| < 1$, then $\mathbf{A}^i \quad \tilde{\mathbf{x}} = \lambda^i \quad \tilde{\mathbf{x}}$ will diminish as the power i, i.e. number of iterations, will approach infinity. Then $\mathbf{A}^i \quad \tilde{\mathbf{x}}$ will converge to zero. If $|\lambda| > 1$, then $\mathbf{A}^i \quad \tilde{\mathbf{x}}$ diverges to infinity. Each time $\mathbf{A}$ is applied, the vector $\tilde{\mathbf{x}}^{(i)}$ stretches or shrinks suitably to the value of $|\lambda|$ as can be seen in Figs. 6.1 and 6.2 following the corresponding figures shown in [45].

For a symmetric $\mathbf{A}$, there exists a set of n linearly independent eigenvectors of $\mathbf{A}$, denoted $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. Sometimes, also nonsymmetric matrices have the same property. Such set of vectors $\{\mathbf{v}_i\}$ can be recollected as the basis for the vector space $\mathbf{R}^n$. For both cases the set is not unique, because each eigenvector can be scaled by an arbitrary nonzero constant. Each eigenvector has a corresponding eigenvalue $\lambda_1, \lambda_2, \ldots, \lambda_n$ which are uniquely defined for a given matrix. Eigenvalues may or may not be equal to each other. For example, eigenvalue for the identity matrix $\mathbf{I}$ are all equal to one, and ***every nonzero vector is an eigenvector of*** $\mathbf{I}$.

Now the question is, what happen when $\mathbf{A}$ is repeatedly applied to a vector $\tilde{\mathbf{x}}$

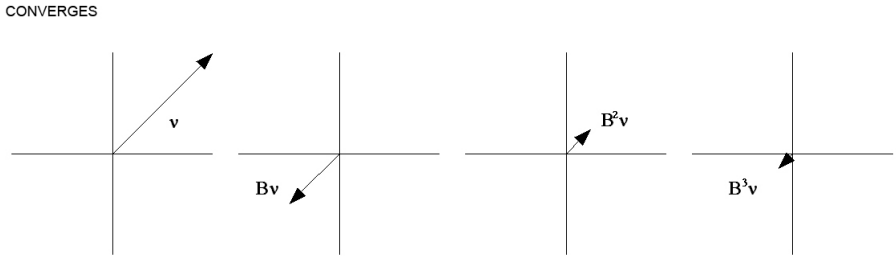

Figure 6.1: Convergence of initial vector $\mathbf{v}$ to $\mathbf{B}^n\mathbf{v}$

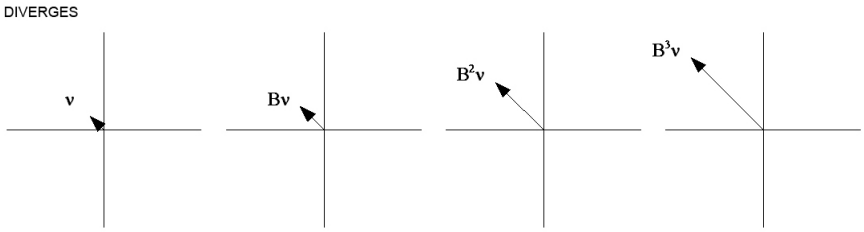

Figure 6.2: Divergence of $\mathbf{B}^n\mathbf{v}$ from the initial vector $\mathbf{v}$

that is not an eigenvector. The correct answer is based on the consideration of any vector as a sum of two eigenvectors: $\tilde{\mathbf{x}} = \mathbf{v}_1 + \mathbf{v}_2$. Then applying $\mathbf{A}$ to the sum of two eigenvectors, results in $\mathbf{A}^i(\mathbf{v}_1 + \mathbf{v}_2) = \mathbf{A}^i\mathbf{v}_1 + \mathbf{A}^i\mathbf{v}_2 = \lambda_1^i\mathbf{v}_1 + \lambda_2^i\mathbf{v}_2$. If the magnitude of all eigenvalues is smaller than one, then $\mathbf{A}^i\tilde{\mathbf{x}}$ is converging to zero, because eigenvectors that compose $\tilde{\mathbf{x}}$ converge to zero whenever $\mathbf{A}$ is repeatedly applied. When even one eigenvalue has magnitude greater than one, a vector $\tilde{\mathbf{x}}$ will diverge to infinity. Having two eigenvalues involved in our analysis, it is reasonable to introduce the notion of spectral radius of a matrix:

$$\rho(\mathbf{A}) = max|\lambda_i| \tag{6.8}$$

for all eigenvalues of $\mathbf{A}$ involved in the consideration of convergence. For convergent numerical methods, $\rho(\mathbf{A})$ should be less than one, and as small as possible.

The eigenvalues of a positive-definite matrix are all positive and it comes out from the rule

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \lambda\mathbf{v} \\ \mathbf{v}^T\mathbf{A}\mathbf{v} &= \lambda\mathbf{v}^T\mathbf{v} \end{aligned}$$

The number $\mathbf{v}^T\mathbf{A}\mathbf{v}$ is positive by definition, because it is a sum of products of corresponding components of two vectors, and hence, also λ must be positive.

6.1.3 Stationary points of Jacobi iterations

The fundamental Eq.(5.2) of the Jacobi method can be written as

$$\tilde{\mathbf{x}}^{(k+1)} = \mathbf{B}\tilde{\mathbf{x}}^{(k)} + \mathbf{z} \tag{6.9}$$

where $\mathbf{B} = \mathbf{D}^{-1}(-\mathbf{L} - \mathbf{U})$ and $\mathbf{z} = \mathbf{D}^{-1}\mathbf{b}$. The formula generates a sequence of vectors from the starting point $\tilde{\mathbf{x}}^{(0)}$, where a subsequent term, $\tilde{\mathbf{x}}^{(i+1)}$ is closer to the exact solution $\hat{\mathbf{x}}$ than a previous one. The vector $\hat{\mathbf{x}}$ is called a stationary point of Eq.(6.9), because if $\tilde{\mathbf{x}}^{(i)} \approx \hat{\mathbf{x}}$, then also $\tilde{\mathbf{x}}^{(i+1)} \approx \hat{\mathbf{x}}$. In particular, for the solution $\mathbf{x}$, Eq.(6.9) gives $\hat{\mathbf{x}} = \mathbf{B}\hat{\mathbf{x}} + \mathbf{z}$.

Let analyse the matter, what happens on each iteration. Expressing each iterate $\tilde{\mathbf{x}}^{(i)}$ as the sum of the exact solution $\hat{\mathbf{x}}$ and the error term $\mathbf{e}^{(i)}$, the Eq.(6.9) can be rewritten as

$$\begin{aligned} \tilde{\mathbf{x}}^{(k+1)} &= \mathbf{B}(\hat{\mathbf{x}} + \mathbf{e}^{(k)}) + \mathbf{z} \\ &= \mathbf{B}\hat{\mathbf{x}} + \mathbf{z} + \mathbf{B}\mathbf{e}^{(k)} \\ &= \hat{\mathbf{x}} + \mathbf{B}\mathbf{e}^{(k)}. \end{aligned}$$

Finally, moving $\hat{\mathbf{x}}$ into the LHS, the above relation becomes

$$\mathbf{e}^{(i+1)} = \mathbf{B}\mathbf{e}^{(i)}. \tag{6.10}$$

Eq.(6.10) can be interpreted as follows:

- Each iteration does not affect the correct part of $\tilde{\mathbf{x}}^{(i)}$, as $\hat{\mathbf{x}}$ is a stationary point,
- in each iteration the error term is effected.

Therefore, the initial guess $\tilde{\mathbf{x}}^{(0)}$ has no effect on the final outcome, but of course it affects the number of iterations necessary to converge to $\hat{\mathbf{x}}$ within arbitrary assumed tolerance. Moreover, the choice of $\tilde{\mathbf{x}}^{(0)}$ effects less the final solution than the spectral radius $\rho(\mathbf{B})$, which determines the speed of convergence. For example, assuming that $\mathbf{v}^{(k)}$ is the eigenvector of $\mathbf{B}$ corresponding to the largest eigenvalue, i.e. $\rho(\mathbf{B}) = \lambda_k$, the slowest convergence we can observe if the initial error $\mathbf{e}^{(0)}$, expressed as a linear combination of eigenvectors, includes a component in the direction of $\mathbf{v}^{(k)}$. The rate of convergence of the Jacobi method mostly depend on $\rho(\mathbf{B})$ which is related to $\mathbf{A}$. Unfortunately, Jacobi method does not provide the convergence of a sequence $\{\tilde{\mathbf{x}}^{(i)}\}$ to $\hat{\mathbf{x}}$ for every $\mathbf{A}$, or even for every positive-definite $\mathbf{A}$.

6.2 Method of steepest descent

In the steepest descent method, the solution process starts at an arbitrary point, i.e. initial guess, $\tilde{\mathbf{x}}^{(0)}$ and proceeds to the solution $\hat{\mathbf{x}}$ through a series of n steps $\{\tilde{\mathbf{x}}^{(1)}, \tilde{\mathbf{x}}^{(2)}, \ldots, \tilde{\mathbf{x}}^{(n)}\}$. Taking a step, we choose the direction in which the quadratic form f decreases most quickly. This direction is opposite $\frac{df(\tilde{\mathbf{x}}^{(i)})}{d\mathbf{x}} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}^{(i)}$. In the beginning let us introduce few definitions

Object	Expression	Definition
Error	$\mathbf{e}^{(i)} = \tilde{\mathbf{x}}^{(i)} - \hat{\mathbf{x}}$	a vector that indicates how far $\tilde{\mathbf{x}}^{(i)}$ is from the solution $\hat{\mathbf{x}}$.
Residual	$\mathbf{r}^{(i)} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}^{(i)}$	This value indicates how far $\mathbf{A}\tilde{\mathbf{x}}^{(i)}$ is from the correct value of RHS, i.e. $\mathbf{b}$. The direction of steepest descent.
Line search	$\tilde{\mathbf{x}}^{(1)} = \tilde{\mathbf{x}}^{(0)} + \alpha\tilde{\mathbf{x}}^{(0)}$	is a procedure that chooses α to minimize the quadratic form along a line.
Directional derivative	$f'(\tilde{\mathbf{x}}^{(\mathbf{1})}) = \frac{d}{d\alpha} f(\tilde{\mathbf{x}}^{(1)})$	the gradient

The residual can be seen as $\mathbf{r}^{(i)} = -\mathbf{A}\mathbf{e}^{(i)}$ and should be understand as the error transformed by $\mathbf{A}$ in to the space that comprises $\mathbf{b}$. Following the relation $\frac{df(\tilde{\mathbf{x}}^{(i)})}{d\mathbf{x}} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}^{(i)}$ the residual is $\mathbf{r}^{(i)} = \frac{df(\tilde{\mathbf{x}}^{(i)})}{d\mathbf{x}}$.

Therefore, the residual can be also understand as the direction of the steepest descent. We would like to know, what is the value of α that minimizes the directional derivative. The answer is: when the directional derivative is zero, i.e. $\frac{d}{d\alpha} f(\tilde{\mathbf{x}}^{(1)}) = 0$, the best choice of α is such that provides $\mathbf{r}^{(0)}$ to be orthogonal to $f'([\tilde{\mathbf{x}}^{(1)}]^T)$. That can be obtained by using the chain rule $\frac{d}{d\alpha} f(\tilde{\mathbf{x}}^{(1)}) = f'([\tilde{\mathbf{x}}^{(1)}]^T)\frac{d}{d\alpha}\tilde{\mathbf{x}}^{(1)} = f'([\tilde{\mathbf{x}}^{(1)}]^T)\mathbf{r}^{(0)}$ and setting the last term to zero.

Following [45] and substituting $f'(\tilde{\mathbf{x}}^{(1)}) = -\mathbf{r}^{(1)}$ we can determine α by the formula

$$\alpha = \frac{\left[\mathbf{r}^{(1)}\right]^T \mathbf{r}^{(0)}}{\left[\mathbf{r}^{(1)}\right]^T \mathbf{A}\mathbf{r}^{(0)}} \tag{6.11}$$

It is obtained from the set of transformations

$$\begin{aligned} \left[\mathbf{r}^{(1)}\right]^T \mathbf{r}^{(0)} &= 0 \\ \left[\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}^{(1)}\right]^T \mathbf{r}^{(0)} &= 0 \\ \left[\mathbf{b} - \mathbf{A}(\tilde{\mathbf{x}}^{(0)} + \alpha\mathbf{r}^{(0)})\right]^T \mathbf{r}^{(0)} &= 0 \\ [\mathbf{b} - \mathbf{A}(\tilde{\mathbf{x}}^{(0)})]^T \mathbf{r}^{(0)} - \alpha[\mathbf{A}\mathbf{r}^{(0)}]^T \mathbf{r}^{(0)} &= 0 \end{aligned} \tag{6.12}$$

Moving the second term from the last expression into RHS we can get

$$\begin{array}{lcl} [\mathbf{b} - \mathbf{A}(\tilde{\mathbf{x}}^{(0)})]^T \mathbf{r}^{(0)} & = & \alpha [\mathbf{A}\mathbf{r}^{(0)}]^T \mathbf{r}^{(0)} \\ [\mathbf{r}^{(1)}]^T \mathbf{r}^{(0)} & = & \alpha [\mathbf{r}^{(0)}]^T \mathbf{A}\mathbf{r}^{(0)} \end{array} \tag{6.13}$$

from where we can get Eq(6.11). Finally, the algorithm for the steepest descent consists of the following equations:

$$\begin{array}{lcl} \mathbf{r}^{(i)} & = & \mathbf{b} + \mathbf{A}\tilde{\mathbf{x}}^{(i)}, \\ \alpha^{(i)} & = & \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i-1)}}{[\mathbf{r}^{(i)}]^T \mathbf{A}\mathbf{r}^{(i-1)}} \\ \tilde{\mathbf{x}}^{(i+1)} & = & \tilde{\mathbf{x}}^{(i)} + \alpha^{(i)} \mathbf{r}^{(i)}. \end{array} \tag{6.14}$$

This algorithm requires two matrix-vector multiplications per iteration. One of them can be eliminated performing the following operations:

- multiply both sides of the last equation of Eq.(6.14) by $-\mathbf{A}$,
- adding $\mathbf{b}$ to both sides.

Then the last equation of Eq.(6.14) can be replaced by

$$\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} + \alpha^{(i)} \mathbf{A}\mathbf{r}^{(i)}. \tag{6.15}$$

Thus the multiplication appears in the algorithm twice but it is evaluated only once. The sequence of $\mathbf{r}^{(i)}$ defined by Eq.(6.15) is generated without any feedback from the value of $\tilde{\mathbf{x}}^{(i)}$ that following the accumulation of floating point roundoff errors may converge to some point situated near $\hat{\mathbf{x}}$. The quality of recurrence calculation can be upgraded by periodical use of the first equation of Eq.(6.14).

Figure (6.3) shows that each gradient is orthogonal to the previous gradient.

6.2.1 Convergence analysis for steepest descent

First, consider the vector error $\mathbf{e}^{(i)}$ as an eigenvector with a corresponding eigenvalue λ_e. Therefore, the residual is $\mathbf{r}^{(i)} = -\mathbf{A}\mathbf{e}^{(i)} = -\lambda_e \mathbf{e}^{(i)}$. Using $\mathbf{e}^{(i)}$ in Eq.(6.14) gives

$$\begin{array}{lcl} \mathbf{e}^{(i+1)} & = & \mathbf{e}^{(i)} + \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{r}^{(i)}]^T \mathbf{A}\mathbf{r}^{(i)}} \\ & = & \mathbf{e}^{(i)} + \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{\lambda_e [\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}} (-\lambda_e \mathbf{e}^{(i)}) \\ & = & 0. \end{array} \tag{6.16}$$

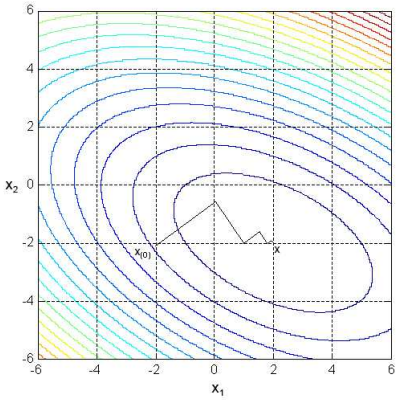

Figure 6.3: Each gradient is orthogonal to the previous gradient

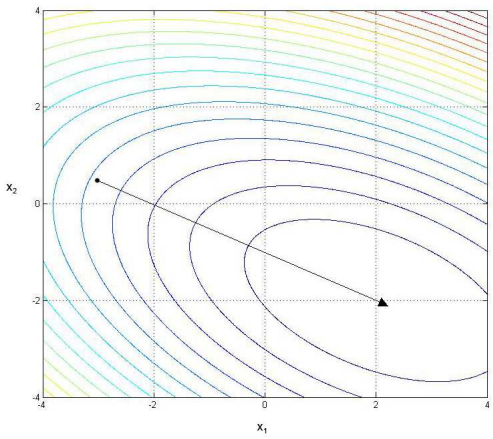

Figure 6.4: Convergence of the SD Method to the exact solution in the first iteration when the error term is en eigenvector

Figure shows how fast the SD procedure provides to converge to the exact solution. The instant convergence is ensured by choosing $\alpha^{(i)} = \lambda_e^{-1}$. Each error vector $\mathbf{e}^{(i)}$ can be expressed as a linear combination of eigenvectors

$$\mathbf{e}^{(i)} = \sum_{j=1}^{n} \kappa_j \mathbf{v}^{(j)} \tag{6.17}$$

where κ_j is the length of each component of vector $\mathbf{e}^{(i)}$. It is required that those eigenvectors are orthonormal. From the analysis we know that for symmetric $\mathbf{A}$, there exists a set of n orthogonal eigenvectors of $\mathbf{A}$ that can be scaled arbitrarily and in particular each of them could have the unit length. Following all of that, the selection of eigenvectors has useful property

$$\begin{aligned} [\mathbf{v}^{(j)}]^T \mathbf{v}^{(k)} &= 1 \quad \text{for} \quad j = k \\ [\mathbf{v}^{(j)}]^T \mathbf{v}^{(k)} &= 0 \quad \text{for} \quad j \neq k \end{aligned} \tag{6.18}$$

Following the last equation consisting the steepest descent and Eq.(6.17) we can obtain

$$\begin{aligned} \mathbf{e}^{(i+1)} &= \mathbf{e}^{(i)} + \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{r}^{(i)}]^T \mathbf{A} \mathbf{r}^{(i)}} \mathbf{r}^{(i)} \\ &= \mathbf{e}^{(i)} + \frac{\sum_j \kappa_j^2 \lambda_j^2}{\sum_j \kappa_j^2 \lambda_j^3} \mathbf{r}^{(i)}. \end{aligned} \tag{6.19}$$

We know from the previous subchapter that the convergence can be achieved in one step when error term $\mathbf{e}^{(i)}$ has only one eigenvector component. By choosing $\alpha^{(i)} = \frac{1}{\lambda_e}$, the Eq.(6.22) results in

$$\mathbf{e}^{(i+1)} = \mathbf{e}^{(i)} + \frac{\lambda^2 \sum_j \kappa_j^2}{\lambda^3 \sum_j \kappa_j^2} (-\lambda \mathbf{e}^{(i)}). \tag{6.20}$$

The above can be illustrated recalling a geometrical interpretation of the quadratic form $f(\mathbf{x})$ as an ellipsoid. In the case when all eigenvalues are equal, the ellipsoid becomes a sphere and the residual $\mathbf{r}^{(i)}$ points to the center of the sphere with no matter of a starting point. In other case, for several nonequal and nonzero eigenvalues, none of $\alpha^{(i)}$ eliminates all eigenvalue components. Then the choice of one $\alpha^{(i)}$ can be treated as a kind of compromise choice in the sense of a weighted average of values of $\frac{1}{\lambda_j}$. The weights λ_j^2 amplifies the influence of longer components of error $\mathbf{e}^{(i)}$ on $\mathbf{e}^{(i+1)}$. Following that, some shorter components of $\mathbf{e}^{(i)}$ might increase in length but not for long and for this reason, the method of SD and CG are called roughers in the contrast to Jacobi Method which is a smoother, because every eigenvector component is reduced on every iteration.

6.2.2 Convergence bound of Steepest Descent

Minimizing $\mathbf{e}^{(i)}$ is equivalent to minimizing Eq. (6.6)

$$q(\tilde{\mathbf{x}}) = q(\hat{\mathbf{x}}) + \frac{1}{2}(\tilde{\mathbf{x}} - \hat{\mathbf{x}})^T \mathbf{A}(\tilde{\mathbf{x}} - \hat{\mathbf{x}})$$

To show this it is convenient to introduce, so called **energy norm** $\|\mathbf{e}\|_{\mathbf{A}} = (\mathbf{e}^T\mathbf{A}\mathbf{e})$. Using the energy norm the error $\|\mathbf{e}^{(i+1)}\|$ can be expressed as follows

$\|\mathbf{e}^{(i+1)}\|_A^2 =$	Factoring reason
$= (\mathbf{e}^{(i+1)})^T\mathbf{A}\mathbf{e}^{(i+1)}$	
$= [(\mathbf{e}^{(i)})^T + \alpha^{(i)}(\mathbf{r}^{(i)})^T]\mathbf{A}[(\mathbf{e}^{(i)}) + \alpha^{(i)}(\mathbf{r}^{(i)})]$	$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)}\mathbf{r}^{(i)}$
$= \mathbf{e}^{(i)})^T\mathbf{A}\mathbf{e}^{(i)} + 2\alpha^{(i)}(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{e}^{(i)} + (\alpha^{(i)})^2(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}$	by symmetry of $\mathbf{A}$
$= \|\mathbf{e}^{(i)}\|_A^2 + 2\frac{(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)}}{(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}}[-(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)}] + \left[\frac{(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)}}{(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}}\right]^2 (\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}$	
$= \|\mathbf{e}^{(i)}\|_A^2 - \frac{[(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)}]^2}{(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}}$	
$= \|\mathbf{e}^{(i)}\|_A^2 \left(1 - \frac{[(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)}]^2}{[(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)}][(\mathbf{e}^{(i)})^T\mathbf{A}\mathbf{e}^{(i)}]}\right)$	
$= \|\mathbf{e}^{(i)}\|_A^2 \left(1 - \frac{\left[\sum_{(j)} \kappa_j^2\lambda_j^2\right]^2}{\left[\sum_{(j)} \kappa_j^2\lambda_j^3\right]\left[\sum_{(j)} \kappa_j^2\lambda_j\right]}\right)$	$(\mathbf{r}^{(i)})^T\mathbf{r}^{(i)} = \sum_{(j)} \kappa_j^2\lambda_j^2$ $(\mathbf{e}^{(i)})^T\mathbf{A}\mathbf{e}^{(i)} = \sum_{(j)} \kappa_j^2\lambda_j$ $(\mathbf{r}^{(i)})^T\mathbf{A}\mathbf{r}^{(i)} = \sum_{(j)} \kappa_j^2\lambda_j^3$
$= \|\mathbf{e}^{(i)}\|_A^2\omega^2$	

where the value which determines the rate of convergence of Steepest Descent is expressed by

$$\omega^2 = 1 - \frac{\left[\sum_{(j)} \kappa_j^2\lambda_j^2\right]^2}{\left[\sum_{(j)} \kappa_j^2\lambda_j^3\right]\left[\sum_{(j)} \kappa_j^2\lambda_j\right]}$$

.

Now the analysis of the convergence consists in finding an upper bound of ω. To facilitate convergence analysis in regard of weights and eigenvalues, we can limit the general consideration to the case with two eigenvalues, i.e. $n = 2$. Introducing two additional numbers: μ which is the slope of $\mathbf{e}^{(i)}$ and ϑ which is the condition number of the matrix $\mathbf{A}$, ω can be expressed by

$$\begin{aligned} \omega^2 &= 1 - \frac{(\kappa_1^2\lambda_1^2 + \kappa_2^2\lambda_2^2)^2}{(\kappa_1^2\lambda_1^3 + \kappa_2^2\lambda_2^3)(\kappa_1^2\lambda_1 + \kappa_2^2\lambda_2)} \\ &= 1 - \frac{(\vartheta^2 + \mu^2)^2}{(\vartheta^3 + \mu^2)(\vartheta + \mu^2)} \end{aligned} \tag{6.21}$$

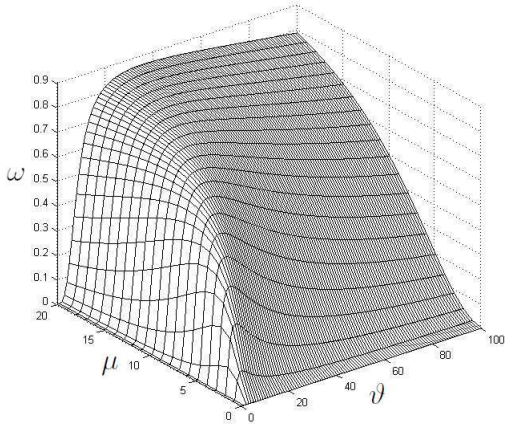

Figure 6.5: Convergence of the SD method as a function of μ and ϑ.

where

$$\vartheta = \frac{\lambda_1}{\lambda_2} \geq 1 \qquad \text{is the spectral condition number of } \mathbf{A}$$

$$\mu = \frac{\kappa_2}{\kappa_1} \quad \text{is the slope of } \mathbf{e}^{(i)} \text{ relative to the coordinate system defined by the eigenvectors}$$

Drawing a graph of ω as a function of μ and ϑ, makes the analysis of Eq.(6.21) more easy. Convergence is fast when μ and ϑ are small.

Let assume that $\mathbf{e}^{(0)}$ is an eigenvector and their slope μ is zero or infinity. Then ω can be determined from the graph as zero and that implies the instant convergence of SD. When eigenvalues are equal, the condition number $\vartheta = 1$ and again, $\omega = 0$.

For a fixed $\mathbf{A}$, ϑ is constant, and the upper bound for ω can be found by setting $\mu^2 = \vartheta^2$ that is equivalent to $\mu = |\vartheta|$. In the Figure (6.5), the line $\mu = |\vartheta|$ defines a faint ridge. Hence

$$\begin{aligned} \omega^2 &\leq 1 - \frac{4\vartheta^4}{\vartheta^5 + 2\vartheta^4 + \vartheta^3} \\ &= \frac{\vartheta^5 - 2\vartheta^4 + \vartheta^3}{\vartheta^5 + 2\vartheta^4 + \vartheta^3} \\ &= \left[\frac{\vartheta - 1}{\vartheta + 1}\right]^2 \end{aligned}$$

from where

$$\omega \leq \frac{\vartheta - 1}{\vartheta + 1}. \tag{6.22}$$

Eq. (6.22) is valid for more eigenvalues, i.e. $n \geq 2$, when the condition number of a symmetric, positive definite matrix is defined by

$$\vartheta = \frac{\lambda_{max}}{\lambda_{min}}.$$

For the larger condition number ϑ, the matrix $\mathbf{A}$ is more ill conditioned and the convergence SD is slower. Finally, the convergence results for SD are

$$\| \mathbf{e}^{(i)} \|_A \leq \left(\frac{\vartheta - 1}{\vartheta + 1} \right)^i \| \mathbf{e}^{(0)} \|_A \tag{6.23}$$

and following Eq.(6.6) and Eq.(6.6)

$$\begin{aligned} \frac{q(\mathbf{x}^{(i)}) - q(\hat{\mathbf{x}})}{q(\mathbf{x}^{(0)}) - q(\hat{\mathbf{x}})} &= \frac{\frac{1}{2}[\mathbf{e}^{(i)}]^T \mathbf{A} \mathbf{e}^{(i)}}{\frac{1}{2}[\mathbf{e}^{(0)}]^T \mathbf{A} \mathbf{e}^{(0)}} \\ &\leq \left(\frac{\vartheta - 1}{\vartheta + 1} \right)^{2i}. \end{aligned} \tag{6.24}$$

Starting points worse cases for SD are plotted in Fig.(6.6) as the straight lines defined by $\frac{\kappa_2}{\kappa_1} = |\vartheta|$.

6.3 Method of conjugate directions

The method of conjugate directions (CD) is based on the following ideas:

- choose a set of orthogonal search directions: $\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, ..., \mathbf{d}^{(n-1)}$,
- from a point $\mathbf{x}^{(i)}$ take a step $\mathbf{d}^{(i)}$ and choose a point $\mathbf{x}^{(i+1)}$, i.e.

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)} \mathbf{d}^{(i)} \tag{6.25}$$

- take exactly one step in each search direction,

The value of $\alpha^{(i)}$ can be found by using the orthogonality condition $[\mathbf{d}^{(i)}]^T \mathbf{e}^{(i+1)} = 0$ of $\mathbf{d}^{(i)}$ and the error vector $\mathbf{e}^{(i+1)}$, in a formula derived from Eq.(6.42)

$$[\mathbf{d}^{(i)}]^T (\mathbf{e}^{(i)} + \alpha^{(i)} \mathbf{d}^{(i)}) = 0 \tag{6.26}$$

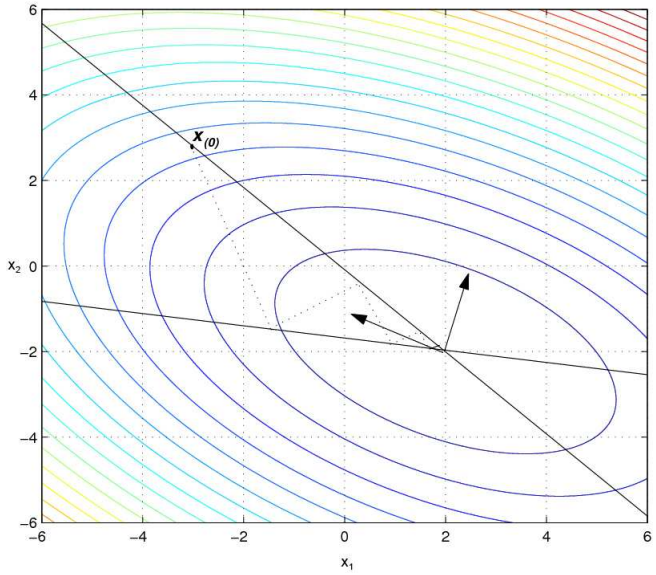

Figure 6.6: The worse convergence of SD are represented by solid lines

from where one can obtain

$$\alpha^{(i)} = -\frac{[\mathbf{d}^{(i)}]^T \mathbf{e}^{(i)}}{[\mathbf{d}^{(i)}]^T \mathbf{d}^{(i)}} \tag{6.27}$$

Unfortunately, $\alpha^{(i)}$, can not be computed without knowing $\mathbf{e}^{(i)}$. This can be improved by replacing orthogonality by A-orthogonality that is know also as the condition for conjugate vectors:

$$[\mathbf{d}^{(i)}]^T \mathbf{A} \mathbf{d}^{(j)}$$

The requirement for A-orthogonality of $\mathbf{e}^{(i+1)}$ to $\mathbf{d}^{(i)}$ is equivalent to finding the minimum point of the quadratic form $f(\mathbf{x}^{(i+1)})$ along the search direction $\mathbf{d}^{(i)}$. It can be seen in the following steps

- set the directional derivative to zero

$$\frac{d}{d\alpha} f(\mathbf{x}^{(i+1)}) = 0,$$

- and get

$$[f'(\mathbf{x}^{(i+1)})]^T \frac{d}{d\alpha} \mathbf{x}^{(i+1)} = 0,$$

- from where

$$-[\mathbf{r}^{(i+1)}]^T\mathbf{d}^{(i)} = 0,$$

- and taking the transposition of the LHS

$$[\mathbf{A}\mathbf{e}^{(i+1)}]^T\mathbf{d}^{(i)} = 0,$$

- obtain

$$[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{e}^{(i+1)} = 0.$$

Repeating the procedure for getting Eq.(6.44) we can obtain the expression for $\alpha^{(i)}$ for A-orthogonal search directions

$$\alpha^{(i)} = -\frac{[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{e}^{(i)}}{[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{d}^{(i)}} \tag{6.28}$$

$$= \frac{[\mathbf{d}^{(i)}]^T\mathbf{r}^{(i)}}{[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{d}^{(i)}} \tag{6.29}$$

that can be calculated. It can be noted that selecting a set of search vectors $\{\mathbf{d}^{(i)}\}$ the same as a set of residuals $\{\mathbf{r}^{(i)}\}$ leads to the same formula as obtained for SD.

The next task is to prove that this procedure really can evaluate $\mathbf{x}$ in n steps. It can be done in few following steps

1. express the error term by the linear combination of search directions

$$\mathbf{e}^{(n)} = \sum_{j=0}^{n-1} \delta^{(j)}\mathbf{d}^{(j)} \tag{6.30}$$

2. eliminate all the $\delta^{(j)}$ values but one from Eq.(6.29)
   - multiply Eq.(6.29) by $[\mathbf{d}^{(k)}]^T\mathbf{A}$

$$[\mathbf{d}^{(k)}]^T\mathbf{A}\mathbf{e}^{(n)} = \sum_{j=0}^{n-1} \delta^{(j)}[\mathbf{d}^{(k)}]^T\mathbf{A}\mathbf{d}^{(j)},$$

   - the RHS simplifies by A-orthogonality property of $\mathbf{d}$ vectors

$$[\mathbf{d}^{(k)}]^T\mathbf{A}\mathbf{e}^{(n)} = \delta^{(k)}[\mathbf{d}^{(k)}]^T\mathbf{A}\mathbf{d}^{(k)},$$

- due to orthogonality several terms from the summation operation will be eliminated

$$\delta^{(k)} = \frac{[\mathbf{d}^{(k)}]^T \mathbf{A}\mathbf{e}^{(n)}}{[\mathbf{d}^{(k)}]^T \mathbf{A}\mathbf{d}^{(k)}} = \frac{\mathbf{A}\left([\mathbf{d^{(k)}}]^{\mathbf{T}}\mathbf{e^{(0)}} + \sum_{\mathbf{i=0}}^{\mathbf{n-1}} \alpha^{(\mathbf{i})}[\mathbf{d^{(k)}}]^{\mathbf{T}}\mathbf{d^{(i)}}\right)}{[\mathbf{d}^{(k)}]^T \mathbf{A}\mathbf{d}^{(k)}},$$

- and only non-orthogonal terms to $\mathbf{d}^{(k)}$ will remain leading to the final expression for $\delta^{(k)}$ in the form

$$\delta^{(k)} = \frac{[\mathbf{d}^{(k)}]^T \mathbf{A}\mathbf{e}^{(k)}}{[\mathbf{d}^{(k)}]^T \mathbf{A}\mathbf{d}^{(k)}} \tag{6.31}$$

The comparison of Eq.(6.31)and Eq.(6.28) leads to the simple equality

$$\alpha^{(i)} = -\delta^{(k)}$$

that gives rise to the new idea of error analysis for CD related to the building up a solution $\mathbf{x}$ component by component gradually cutting down the error term with the some paste. Hence, the error can be expressed by

$$\begin{aligned} \mathbf{e}^{(i)} &= \mathbf{e}^{(0)} + \sum_{j=0}^{i-1} \alpha^{(j)} \mathbf{d}^{(j)} \\ &= \sum_{j=0}^{n-1} \delta^{(j)} \mathbf{d}^{(j)} - \sum_{j=0}^{i-1} \delta^{(j)} \mathbf{d}^{(j)} \\ &= \sum_{j=i}^{n-1} \delta^{(j)} \mathbf{d}^{(j)} \end{aligned} \tag{6.32}$$

Note that after n iterations, every component will be cut away, and $\mathbf{e}^{(n)} = 0$.

6.3.1 How to generate A-orthogonal search directions?

A-orthogonal search directions can be generated by the conjugate Gram-Schmidt process. To construct a set of $\{\mathbf{d}^{(j)}\}$ for $j = 0, ..., n$ choose a set of n linearly independent vectors $\mathbf{u}^{(o)}, \mathbf{u}^{(1)}, ..., \mathbf{u}^{(n-1)}$, and subtract out such components which are not A-orthogonal to the previous $\mathbf{d}^{(i)}$ vectors. The generation can be originated by setting $\mathbf{d}^{(0)} = \mathbf{u}^{(0)}$ and constructing a number $i > 0$ of the new vectors by following

$$\mathbf{d}^{(j)} = \mathbf{u}^{(j)} + \sum_{k=0}^{j-1} \beta_{jk} \mathbf{d}^{(k)} \tag{6.33}$$

where numbers β_{jk} are defined for $j > 0$ by the same method as was used to evaluate $\delta^{(j)}$, i.e. following the procedure:

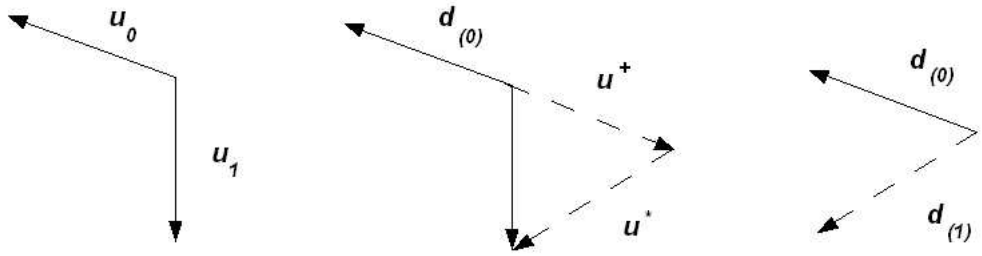

Figure 6.7: Gram-Schmidt orthogonality (conjugation) of two vectors

- multiply both sides of Eq.(6.33) by $\mathbf{Ad}^{(i)}$

$$[\mathbf{d}^{(j)}]^T\mathbf{Ad}^{(i)} = [\mathbf{u}^{(j)}]^T\mathbf{Ad}^{(i)} + \sum_{k=0}^{j-1}\beta_{jk}[\mathbf{d}^{(k)}]^T\mathbf{Ad}^{(i)}$$

- the above is reduced by A-orthogonality of $\mathbf{d}^{(i)}$ vectors

$$0 = [\mathbf{u}^{(j)}]^T\mathbf{Ad}^{(i)} + \beta_{ji}[\mathbf{d}^{(i)}]^T\mathbf{Ad}^{(i)}, \quad \text{for} \quad j > i$$

- and finally β_{ji} is evaluated by

$$\beta_{ji} = -\frac{[\mathbf{u}^{(j)}]^T\mathbf{Ad}^{(i)}}{[\mathbf{d}^{(i)}]^T\mathbf{Ad}^{(i)}} \tag{6.34}$$

Unfortunately, all old search vectors in CD method with Gram-Schmidt process must be kept in memory to evaluate each new vector. The full set of $\mathbf{d}^{(i)}$ vectors is generated by performing $O(n^3)$ operations.

Selecting the search directions as conjugated axial unit vectors, CD method becomes equivalent to performing the Gaussian elimination. It should be noted that CD was used rather seldom before the Conjugate Gradient (CG) method was invented.

6.3.2 Error term

To discuss the matter of CD method properties, it is convenient to introduce the following notions:

- The **linear span**, called also the linear hull, of a set of vectors in a vector space **is the intersection of all subspaces containing that set**. Therefore, the linear span of a set of vectors is a vector space.

- **Definition:** The span of a set $\mathcal{S}$ for a given vector space Ω over a field $\mathcal{R}$ is defined as the intersection $\mathcal{W}$ of all subspaces of Ω which contain $\mathcal{S}$. The span of the set of vectors $\mathbf{d}^{(1)}, \mathbf{d}^{2)}, \ldots, \mathbf{d}^{(n)} \in \Omega$, is

$$\text{span}(\mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(n)}) = \{\lambda_1 \mathbf{d}^{(1)} + \ldots + \lambda_n \mathbf{d}^{(n)} | \lambda_1, \ldots, \lambda_n \in \mathcal{R}\},$$

  where $\mathcal{S}$ is not necessarily finite.

- The spanning vectors need not be linearly independent, and therefore, a spanning set is not necessarily the basis for $\mathcal{S}$.

- A minimal spanning set for given $\mathcal{S}$ is a basis in a finite dimensional space. A spanning set in a finite dimensional space is a basis for $\mathcal{S}$ if and only if every vector in $\mathcal{S}$ can be expressed as a unique linear combination of elements in the spanning set.

- The span of $\mathcal{S}$ may also be defined as the collection of all finite combinations of elements of $\mathcal{S}$.

- Examples of the linear span:

  1. The real vector space $\mathcal{R}^3$ has three vectors $\{(1,0,0), (0,1,0), (0,0,1)\}$ as a spanning set.
  2. The spanning set, which is not the basis, can be given eg. by $\{(1,2,3), (0,1,2),$ $(-1, \frac{1}{2}, 3), (1,1,1)\}$,

CD method has very good property as at each step it finds the best solution within the subspace allowed for a solution exploration. It means that

- the value of $\mathbf{e}^{(i)}$ is chosen from $\mathbf{e}^{(0)} + \mathcal{W}_i$, where $\mathcal{W}_i$ is i-dimensional subspace span$\{\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(i-1)}\}$,

- the best solution is given in the sense that CD chooses the value from $\mathbf{e}^{(0)} + \mathcal{W}_i$ that minimizes $\|\mathbf{e}^{(i)}\|_A$.

Following Eq.(6.32), the quadratic or energy norm of the error term is expressed by

$$\begin{aligned} \|\mathbf{e}^{(i)}\|_A &= \sum_{j=i}^{n-1} \sum_{k=i}^{n-1} \delta^{(j)} \delta^{(k)} [\mathbf{d}^{(j)}]^T \mathbf{A} \mathbf{d}^{(\mathbf{k})} \\ &= \sum_{k=j}^{n-1} (\delta^{(j)})^2 [\mathbf{d}^{(j)}]^T \mathbf{A} \mathbf{d}^{(\mathbf{j})} \end{aligned} \tag{6.35}$$

where each therm in in this summation is associated with a search direction that has not yet been traversed. The prove of the minimum energy norm for $\mathbf{e}^{(i)}$ can be issued from the requirement that any other vector $\mathbf{e}$ chosen from $(\mathbf{e}^{(0)} + \mathcal{W}_i)$ must have the same terms in its expansion.

Another important property of CD method is that the hyperplane $\mathbf{x}^{(0)} + \mathcal{W}_i$ is tangent to the ellipsoid on which the momentary solution $\mathbf{x}^{(i)}$ lies. It is known from the chapter on SD method, that a residual $\mathbf{r}^{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(i)}$ at any point $\mathbf{x}^{(i)}$ is orthogonal to the ellipsoidal surface at that point. Hence a residual $\mathbf{r}^{(i)}$ is also orthogonal to $\mathcal{W}_i$ as well. This can be shown multiplying Eq.(6.32) by $-[\mathbf{d}^{(i)}]^T\mathbf{A}$

$$
\begin{aligned}
-[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{e}^{(j)} &= -\sum_{j=i}^{n-1}\delta^{(j)}[\mathbf{d}^{(i)}]^T\mathbf{A}\mathbf{d}^{(j)} && (6.36)\\
[\mathbf{d}^{(i)}]^T\mathbf{r}^{(j)} &= 0, \quad i<j && (6.37)
\end{aligned}
$$

the second relationship comes out from the A-orthogonality of $\mathbf{d}$-vector. The last relationship arises from the fact that because the error term is A-orthogonal to all the search directions and $\mathbf{r}^{(i)} = -\mathbf{A}\mathbf{e}^{(j)}$, hence the residual term is orthogonal to all previously tested search directions.

Recalling that the search directions are constructed from the set of $\{\mathbf{u}^{(i)}\}$ vectors and the subspace spanned by this set is $\mathcal{W}_i$, the residual $\mathbf{r}^{(i)}$ is orthogonal to $\mathbf{u}^{(i)}$ and the error $\mathbf{e}^{(j)}$ is A-orthogonal to $\mathcal{W}_i$. This can be proved by multiplying both sides of Eq.(6.33) by $\mathbf{r}^{(i)}$

$$
[\mathbf{d}^{(j)}]^T\mathbf{r}^{(i)} = [\mathbf{u}^{(j)}]^T\mathbf{r}^{(i)} + \sum_{k=0}^{j-1}\beta_{jk}[\mathbf{d}^{(k)}]^T\mathbf{r}^{(i)}. \tag{6.38}
$$

and applying Eq.(6.37)

$$
0 = [\mathbf{u}^{(j)}]^T\mathbf{r}^{(i)}, \quad i<j. \tag{6.39}
$$

From Eq.(6.38) the following identity results

$$
[\mathbf{d}^{(i)}]^T\mathbf{r}^{(i)} = [\mathbf{u}^{(i)}]^T\mathbf{r}^{(i)}. \tag{6.40}
$$

This section can be concluded that in CD method the number of matrix-vector products per iteration can be reduced to one by using a recurrence formula to find the residual

$$
\begin{aligned}
\mathbf{r}^{(i+1)} &= -\mathbf{A}\mathbf{e}^{(i+1)} = -\mathbf{A}(\mathbf{e}^{(i)} + \alpha^{(i)}\mathbf{d}^{(i)})\\
&= \mathbf{r}^{(i)} - \alpha^{(i)}\mathbf{A}\mathbf{d}^{(i)} && (6.41)
\end{aligned}
$$

6.4 Method of Conjugated Gradient for solving of linear systems

The method of Conjugated Gradient (CG) [21], [45], [52] can be defined as the method of conjugate directions where the search directions are constructed by setting the search directions $\mathbf{u}^{(i)}$ equal to residuals $\mathbf{r}^{(i)}$. The residuals are conjugate and following that they are orthogonal to previous search directions unless a residual is equal to zero. But then the problem is already solved. Constructing the search vectors from residual, the subspace span$\{\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(i-1)}\}$ is equal to $\mathcal{W}_i$. Residuals fulfil the following requirement

$$[\mathbf{r}^{(j)}]^T \mathbf{r}^{(i)} = 0, \quad i \neq j \tag{6.42}$$

The last equation of the previous subsection, Eq.(6.41), shows that each new residual $\mathbf{r}^{(i+1)}$ is a linear combination of the previous residual $\mathbf{r}^{(i)}$ and $\mathbf{A}\mathbf{d}^{(i)}$, where $\mathbf{d}^{(i)} \in \mathcal{W}_i$. This fact implies that each new subspace $\mathcal{W}_{i+1}$ is formed from the union of the previous subspace $\mathcal{W}_i$ and the subspace $\mathbf{A}\mathcal{W}_i$. Therefore, the subspace $\mathcal{W}_i$ can be span either on $\{\mathbf{d}^{(i)}\}$ or $\{\mathbf{r}^{(i)}\}$

$$\begin{aligned} \mathcal{W}_i &= \text{span}\{\mathbf{d}^{(0)}, \mathbf{A}\mathbf{d}^{(0)}, \mathbf{A}^2\mathbf{d}^{(0)}, \ldots, \mathbf{A}^{i-1}\mathbf{d}^{(0)}\} \\ &= \text{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)}, \ldots, \mathbf{A}^{i-1}\mathbf{r}^{(0)}\} \end{aligned} \tag{6.43}$$

This subspace $\mathcal{W}_i$ created by repeatedly applying a matrix $\mathbf{A}$ to a vector $\mathbf{d}^{(0)}$ or $\mathbf{r}^{(0)}$ is called a **Krylov subspace** (see the next section). This subspace has the following desirable property: $\mathbf{A}\mathcal{W}_i \subset \mathcal{W}_{i+1}$ and $\mathbf{r}^{(i+1)} \perp \mathcal{W}_{i+1}$ then Eq.(6.37) implies that $\mathbf{r}^{(i+1)}$ is A-orthogonal to $\mathcal{W}_i$. It can be shown by the conjugate Gram-Schmidt process that $\mathbf{r}^{(i+1)}$ is already A-orthogonal to all of previous search directions except $\mathbf{d}^{(i)}$.

The Gram-Schmidt constants in Eq.(6.34) can be simplified following the procedure:

- take the inner product of relation given by Eq.(6.41) and the residual $\mathbf{r}^{(j)}$

$$[\mathbf{r}^{(j)}]^T \mathbf{r}^{(i+1)} = [\mathbf{r}^{(j)}]^T \mathbf{r}^{(i)} - \alpha^{(i)} [\mathbf{r}^{(j)}]^T \mathbf{A}\mathbf{d}^{(i)}$$

- rearrange the above formula

$$\alpha^{(i)} [\mathbf{r}^{(j)}]^T \mathbf{A}\mathbf{d}^{(i)} = [\mathbf{r}^{(j)}]^T \mathbf{r}^{(i)} - [\mathbf{r}^{(j)}]^T \mathbf{r}^{(i+1)}$$

- by using the orthogonality of residuals and Eq.(6.42)

- when $i = j$

$$[\mathbf{r}^{(j)}]^T \mathbf{A}\mathbf{d}^{(i)} = \frac{1}{\alpha^{(i)}} [\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)},$$

- when $i = j + 1$

$$[\mathbf{r}^{(j)}]^T \mathbf{A}\mathbf{d}^{(i)} = -\frac{1}{\alpha^{(i-1)}} [\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)},$$

- for $i \neq j$ and $i \neq j + 1$

$$[\mathbf{r}^{(j)}]^T \mathbf{A}\mathbf{d}^{(i)} = 0.$$

- and following Eq.(6.34), parameters β_{ji} in Gram-Schmidt conjugation can be expressed in relation to $\mathbf{r}$ and $\mathbf{d}$

- for $i = j + 1$

$$\beta_{ji} = \frac{1}{\alpha^{(i-1)}} \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{d}^{(i-1)}]^T \mathbf{A}\mathbf{d}^{(i-1)}} \tag{6.44}$$

- or $\beta_{ji} = 0$ for $i > j + 1$.

Note that, because most of β_{ji} terms equals to zero, it is not necessary to store old search vectors to ensure the A-orthogonality of the new search vectors. This leads to the reduction of complexity both in space and time from the order $\mathcal{O}(n^2)$ to $\mathcal{O}(m)$, where m is the number of nonzero entries of matrix $\mathbf{A}$. Replacing $\alpha^{(i-1)}$ in Eq.(6.44) by $\frac{[\mathbf{d}^{(i-1)}]^T \mathbf{r}^{(i-1)}}{[\mathbf{d}^{(i-1)}]^T \mathbf{A}\mathbf{d}^{(i-1)}}$, that is RHS of Eq.(6.29), results in

$$\beta_{i,i-1} = \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{d}^{(i-1)}]^T \mathbf{r}^{(i-1)}}. \tag{6.45}$$

Noting that the superscript in the denominator is $j = i - 1$ and using the regularity of the non-zero terms of matrix $\mathbf{A}$, the following vector $\beta_{(i)} = \beta_{i,i-1}$ can replace the whole matrix.

Using $\mathbf{u}^{(i)} = \mathbf{r}^{(i)}$ in $[\mathbf{d}^{(i)}]^T \mathbf{r}^{(i)} = [\mathbf{u}^{(i)}]^T \mathbf{r}^{(i)}$ and substituting to Eq.(6.45) results in

$$\beta_{i,i-1} = \frac{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{r}^{(i-1)}]^T \mathbf{r}^{(i-1)}}.$$

Gathering all relations constituting the CG method leads to the algorithm for CG method:

set the initial vectors

$$\mathbf{r}^{(0)} = \mathbf{b} + \mathbf{A}\mathbf{x}^{(0)}$$

$$\mathbf{d}^{(0)} = \mathbf{r}^{(0)}$$

set a subscript

$$i = 0$$

repeat

$$\alpha^{(i)} = \frac{[\mathbf{d}^{(i)}]^T \mathbf{r}^{(i)}}{[\mathbf{d}^{(i)}]^T \mathbf{A}\mathbf{d}^{(i)}}$$

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha^{(i)}\mathbf{d}^{(i)}$$

$$\mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} + \alpha^{(i)}\mathbf{A}\mathbf{d}^{(i)}$$

if $\mathbf{r}^{(i+1)}$ is sufficiently small **then** exit loop **end if**

$$\beta_{(i+1)} = \frac{[\mathbf{r}^{(i+1)}]^T \mathbf{r}^{(i+1)}}{[\mathbf{r}^{(i)}]^T \mathbf{r}^{(i)}}$$

$$\mathbf{d}^{(i+1)} = \mathbf{r}^{(i+1)} + \beta^{(i+1)}\mathbf{d}^{(i)}$$

$$i = i + 1$$

end repeat

The result is $\mathbf{x}^{(i+1)}$.

It is important to mention [45] that the name "Conjugate Gradient" method is slightly incorrect because the gradients are not conjugate and not all the conjugate directions are gradients.

6.4.1 Convergence of CG

This subsection will be written on the basis of the following papers: [21], [47], [51],

6.5 Iterative methods based on Krylov subspace methods

6.5.1 Krylov subspace methods

The background of the Krylov method for solving linear algebraic systems

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{6.46}$$

is presented here following [34]. Symbols used in Eq. 6.46 have the same meanings as in previous sections i.e.: $\mathbf{A}$ is a real or complex nonsingular $n \times n$ matrix, and $\mathbf{b}$ is a real or complex vector with n elements. Assuming that $\mathbf{x}^{(0)}$ is the initial proposition for the true solution $\hat{\mathbf{x}}$, the initial residual vector $\mathbf{r}^{(0)}$ is defined by $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$. Krylov subspace method can be derived from the projection method following [38] and [39]. Let $\mathbf{x}^{(n)}$ be n-th iterate of $\hat{\mathbf{x}}$, where $n = 1, 2, ...$, and

$$\mathbf{x}^{(\mathbf{n})} \in \mathbf{x}^{(\mathbf{0})} + \mathcal{S}_n \tag{6.47}$$

where $\mathcal{S}_n$ is n-dimensional, so called, the search space. The $\mathbf{x}^{(n)}$ has n degrees of freedom and it can be uniquely determined subject to n constraints. Therefore, the constraints space $\mathcal{C}_n$ must be defined following the requirement that the n-th residual $\mathbf{r}^{(n)}$ is orthogonal to this space, i.e.,

$$\mathbf{r}^{(n)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(n)} \in \mathbf{r}^{(0)} + \mathbf{A}\mathcal{S}_n, \quad \text{and} \quad \mathbf{r}^{(n)} \perp \mathcal{C}_n. \tag{6.48}$$

The orthogonality is defined with respect to the Euclidean inner product, i.e., let $\mathbf{x}$ and $\mathbf{y} \in \Re^n$ then the inner (or dot) product is defined by

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i = 0. \tag{6.49}$$

Krylov subspaces are defined as follows

$$\mathcal{K}_n(\mathbf{A}, \mathbf{r}^{(0)}) \equiv \text{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \ldots, \mathbf{A}^{n-1}\mathbf{r}^{(0)}\}, \tag{6.50}$$

where $n = 1, 2, \ldots$. The linear span of a set of vectors in a vector space is the intersection of all subspaces containing that set.

Two Krylov subspace (KS) methods can be distinguished by considering basic relations between $\mathcal{S}_n$ and $\mathcal{C}_n$,:

Relation	KS Method
$\mathcal{S}_n = \mathcal{C}_n$	Galerkin method
$\mathcal{S}_n = \mathbf{A}\mathcal{C}_n$	Minimal residual method

Constructing residuals $\mathbf{r}^{(n)}$ orthogonal to all previously evaluated residuals $\mathbf{r}^{(i)}$ for $i \in [n-1, \ldots, 0]$, the Galerkin method can be seen as orthogonal residual (OR) method.

The Krylov subspaces $\mathcal{K}_j(\mathbf{A}, \mathbf{r}^{(0)})$ with $j \in [1, n]$ can be considered as a nested sequence

$$\mathcal{K}_1(\mathbf{A}, \mathbf{r}^{(0)}) \subset \ldots \subset \mathcal{K}_d(\mathbf{A}, \mathbf{r}^{(0)}) \subseteq \mathcal{K}_n(\mathbf{A}, \mathbf{r}^{(0)}) \tag{6.51}$$

that ends with a subspace of maximal dimension $d = \dim\mathcal{K}_n(\mathbf{A}, \mathbf{r}^{(0)})$ The number of steps of Krylov subspace methods is limited by the maximal

Krylov subspace dimension d. The projection process can be break down in the step n-th when the iterate $\mathbf{x}^{(n)}$ does not exist or $\mathbf{x}^{(n)}$ is not unique. The projection methods are **well defined** when they ensure existence and uniqueness of their iterates $\mathbf{x}^{(n)}$ for each step $n \leq d$ and they have the so called **finite termination property** with $n = d$. Such property is related to the properties of the matrix $\mathbf{A}$. In general, to analyze the subject of unique definition of each n iterates of the (OR) Krylow subspace method it is necessary to mention about the **field of values** of the matrix $\mathbf{A}$.

The **field of values** associated with the matrix $\mathbf{A}$ is the image of the unit sphere under the quadratic form induced by the matrix. Assuming that $\mathbf{A}$ is a square matrix with complex entries, the field of values for $\mathbf{A}$ is the set

$$\mathcal{F}(\mathbf{A}) = \{\mathbf{v}^H \mathbf{A}\mathbf{v} : \mathbf{v}^H \mathbf{v} = 1, \mathbf{v} \in \mathcal{C}_n\} \tag{6.52}$$

where $\mathbf{v}^H$ is Hermitian conjugate (or transconjugate). The Hermitian transpose or adjoint matrix for m by n matrix $\mathbf{A}$ with a complex entries is the n by m matrix $\mathbf{A}^H$ (or $\mathbf{A}^*$) obtained from $\mathbf{A}$ by taking the transpose and then taking the complex conjugate of each entry. The complex conjugate for one complex entry $a_j + ib_j$ is: $a_j - ib_j$, where a_j and b_j are reals. When b_j equals to zero for all entries of $\mathbf{A}$, the Hermitian conjugate $\mathbf{A}^H$ transfers to the transpose $\mathbf{A}^T$ in meaning of the standard linear algebra.

The further discussion of the (OR) Krylov subspace method will be limited only to the case of Hermitian positive definite matrices, because only then the matrix $\mathbf{A}$ supplies the norm in which the errors of the method can be minimized. Before going further in explanation of Krylov subspace methods two definitions: the Hermitian matrix, and the positive definite matrix, are presented below.

- **The Hermitian matrix A** or self-adjoint matrix is a square matrix with complex entries which is equal to its own conjugate transpose:
  1. the element of k-th row and l-th column is equal to the complex conjugate of the element in the l-th row and j-th column for all subscripts k and l,
  2. i.e., the following relation is valid $a_{kl} = a^*_{lk}$.
- **The Positive definite Hermitian matrix**
  - Suppose that $\mathbf{A}$ is $n \times n$ Hermitian matrix.
  - The transpose of a vector $\mathbf{v}$ is denoted by $\mathbf{v}^T$ and $\mathbf{v}^*$ stands for the conjugate transpose.

– The matrix $\mathbf{A}$ is **positive definite** if and only if it satisfies the following equivalent properties:

1. For the real quantity $\mathbf{v}^*\mathbf{A}\mathbf{v}$ the following relation is fulfilled $\mathbf{v}^*\mathbf{A}\mathbf{v} > 0$ for all non-zero vectors $\mathbf{v} \in \mathcal{C}_n$.
2. Suppose that all eigenvalues of $\mathbf{A}$ are positive. By the spectral theorem, any Hermitian matrix can be expressed as $\mathbf{A} = \mathbf{LDU}$ where $\mathbf{D}$ is the diagonal real matrix and $\mathbf{U}$ is the unitary matrix whose rows are orthonormal eigenvectors of $\mathbf{A}$ and they form a basis and $\mathbf{L} = \mathbf{U}^{-1}$. Therefore, $\mathbf{A}$ is positive definite if and only if elements of $\mathbf{D}$ which are also the eigenvalues of $\mathbf{A}$, are all positive.
3. The linear form $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^*\mathbf{A}\mathbf{v}$ defines the inner product on the space of $\mathcal{C}_n$.
4. The Sylvester criterion for determinants is fulfilled, i.e. all the leading principal minors of $\mathbf{A}$ are positive
   - the upper left 1×1 corner,
   - the upper left 2×2 corner,
   - the upper left 3×3 corner,
   - ...,
   - the upper left $n \times n$ corner,
5. Suppose that each entry of the matrix $\mathbf{A}$ is defined by $A_{ij} = \langle \mathbf{v}_i, \mathbf{v}_j \rangle = \mathbf{v}_i^*\mathbf{v}_j$ where vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathcal{C}_n$. That means that $\mathbf{A}$ is of the form $\mathbf{B}^*\mathbf{B}$, where $\mathbf{B}$ is injective but may not be a square matrix.

Obviously, the above described properties for a real symmetric matrix $\mathbf{A}$ can be simplified by replacing $\mathcal{C}_n$ by $\mathcal{R}^n$.

For a well defined the (OR) and (MR) Krylov subspace methods, the error $\mathbf{e}^{(n)} = \hat{\mathbf{x}} - \mathbf{x}^{(n)}$ where $\mathbf{x}^{(n)} \in \mathbf{x}^{(0)} + \mathcal{K}_n(\mathbf{A}, \mathbf{r}^{(n)})$ and the residual $\mathbf{r}^{(n)} \in \mathbf{r}^{(0)} + \mathbf{A}\mathcal{K}_n(\mathbf{A}, \mathbf{r}^{(n)})$ can be written in the polynomial form:

$$\begin{aligned} \mathbf{e}^{(n)} &= \mathcal{P}^{(n)}(\mathbf{A})\mathbf{e}^{(0)}, \\ \mathbf{r}^{(n)} &= \mathcal{P}_n\mathbf{r}^{(0)}, \end{aligned} \tag{6.53}$$

where $\mathcal{P}_n$ is the polynomial of degree at most n which is uniquely determined by the constraint conditions Eq. 6.48 and with the value one at the origin.

The Galerkin method (orthogonal residual OR) and the minimal residual (MR) method were implemented in various commonly used algorithms. The

CG method for Hermitian positive definite matrices is the example of a numerical implementation of the OR Krylov subspace method [21]. The MR Krylov subspace method was implemented in the MINRES method [35] for nonsingular Hermitian indefinite matrices and in the GMRES method [40]. for general nonsingular matrices.

Following [34], when the system matrix is a unitary normal matrix (unitary diagonalizable matrix), i.e., a complex square matrix $\mathbf{A}$ which commutes with its conjugate transpose: $\mathbf{A}^*\mathbf{A} = \mathbf{A}\mathbf{A}^* = \mathbf{I}$, the convergence behavior of CG, MINRES, and GMRES is completely determined by $\mathbf{A}$ spectrum and the convergence analysis reduces to the analysis of certain mini-max approximation problem on the matrix eigenvalues. When the system matrix is not normal, the convergence behavior of the GMRES method may not be related to the eigenvalues in simple way, and therefore, other properties of the input data should be considered for the assessment of the convergence. Unfortunately, the convergence analysis for GEMRES in the general non-normal case remains an open problem.

6.5.2 Minimal Residual Method (MINRES)

Preliminaria

The MINRES can be applied to symmetric indefinite systems. To formulate the definition of Hermitian indefinite matrix two intermediate definitions should be recalled:

1. Definition of the negative-semidefinite matrix:
   - The $n \times n$ Hermitian matrix is negative-semidefinite if $\mathbf{x}^*\mathbf{A}\mathbf{x} \leq 0$ for all $\mathbf{x} \in \mathcal{R}^n$, where $\mathbf{x}^*$ is conjugate transpose of $\mathbf{x}$.
2. Definition of the positive-semidefinite matrix:
   - The $n \times n$ Hermitian matrix is positive-semidefinite if $\mathbf{x}^*\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathcal{R}^n$.

Definition: A Hermitian matrix is called **indefinite** when it is neither positive-semidefinite nor negative-semidefinite.

The MINRES method, firstly proposed in [35], is a variant of the conjugate gradient method that avoids the LU decomposition where a breakdown of the algorithm could occur for a zero pivot of the indefinite matrix. The residuals in this method are minimized in the Euclidean norm called also "2-norm".

The best idea to describe the MINRES method is to follow [8] and firstly define the Lanczos process and then construct the MINRES as the relating process.

The Lanczos process

Assume as previously that $\mathbf{A} \in \mathbf{R}^{n\times n}$, $\mathbf{A} \neq 0$ and $\mathbf{b} \neq 0$. The Lanczos process transforms a symmetric matrix $\mathbf{A}$ to a tridiagonal matrix $\mathcal{T}_k$ with $(n+1)$-th additional row at the bottom. The matrix $\mathcal{T}_k \in \mathbf{R}^{n\times(n+1)}$ is defined as follows

$$\mathcal{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & \beta_3 & \alpha_3 & \beta_4 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \beta_{k-2} & \alpha_{k-2} & \beta_{k-1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \beta_k & \alpha_k \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \beta_{k+1} \end{bmatrix}$$

Introducing the new square and symmetric matrix $\mathbf{T}_k$ which consists of the first k rows of $\mathcal{T}_k$, then the $\mathcal{T}_k$ can be expressed as

$$\mathcal{T}_k = \begin{bmatrix} \mathbf{T}_k \\ \beta_{k+1}[\mathbf{e}^{(k)}]^T \end{bmatrix}$$

where $\mathbf{e}^{(k)}$ denotes the k-th unit vector and

$$\mathbf{T}_k = \begin{bmatrix} \mathbf{T}_{k-1} & \beta_k \mathbf{e}^{(k-1)} \\ \beta_k[\mathbf{e}^{(k-1)}]^T & \alpha_k \end{bmatrix}$$

In the Lanczos process, vectors $\mathbf{v}^{(k)}$ are computed as follows:

- assume $\mathbf{v}^{(0)}$ and normalize $\mathbf{v}^{(1)}$ by β_1, i.e. $\mathbf{v}^{(1)}\beta_1 = \mathbf{b}$, where $\beta_1 = \|\mathbf{v}^{(1)}\|_2$,
- evaluate a column vector $\mathbf{p}^{(k)} = \mathbf{A}\mathbf{v}^{(k)}$ and a scalar $\alpha_k = [\mathbf{v}^{(k)}]^T\mathbf{p}^{(k)}$,
- evaluate $\mathbf{v}^{(k+1)}$ from $\beta_{k+1}\mathbf{v}^{(k+1)} = \mathbf{p}^{(k)} - \alpha_k\mathbf{v}^{(k)} - \beta_k\mathbf{v}^{(k-1)}$, where β_{k+1} serves to normalize $\mathbf{v}^{(k+1)}$, i.e. $\beta_{k+1} = \|\mathbf{v}^{(k+1)}\|_2$.

The above procedure can be written in the matrix form

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_{k+1}\mathcal{T}_k \tag{6.54}$$

where $\mathbf{V}_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$.

Moreover, the columns of the matrix $\mathbf{V}_k$ are orthonormal, i.e. the inner product $\langle \cdot, \cdot \rangle$ of two vectors e.g. $\mathbf{v}^{(i)}$ and $\mathbf{v}^{(j)}$, equals to zero, when $i \neq j$. Thus, the Lanczos process stops, when $\beta_{k+1} = 0$ for $k \leq n$, and then

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_k \mathbf{T}_k \tag{6.55}$$

The memory allocation, in this procedure, is needed for the matrix $\mathbf{A}$, three vectors $(\mathbf{v}^{(k-1)}, \mathbf{v}^{(k)}, \mathbf{v}^{(k+1)})$, and three scalars $(\alpha_k, \beta_k, \beta_{k+1})$. The Lanczos process is terminated in iterations number of iterations $i = \min\{\text{rank}(\mathbf{A}) + 1, n\}$, where rank$(\mathbf{A})$ is a number of columns/rows of $\mathbf{A}$ linearly independent.

In each Lanczos step, a subproblem is solved to find $\mathbf{x}^{(k)} \in \mathcal{K}_k(\mathbf{A}, \mathbf{b})$ such that $\mathbf{x}^{(k)} = \mathbf{V}_k \mathbf{y}$ for some $\mathbf{y} \in \mathbf{R}^k$, where $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$ denotes the k-th Krylov subspace of $\mathbf{A}$ and $\mathbf{b}$. It follows that $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} = \mathbf{V}_{k+1}(\beta_1 \mathbf{e}^{(1)} - \mathcal{T}_k \mathbf{y})$, and all methods based on the Lanczos process attempt to make $(\beta_1 \mathbf{e}^{(1)} - \mathcal{T}_k \mathbf{y})$ small by using one or another method. Therefore, the MINRES is constructed upon the Lanczos process by solving the least-squares subproblem

$$\mathbf{y}^{(k)} = \arg \quad \min_{\mathbf{y} \in \mathbf{R}^k} \|\beta_1 \mathbf{e}^{(1)} - \mathcal{T}_k \mathbf{y}\|_2 \tag{6.56}$$

6.5.3 MINRES Convergence analysis

This subsection follows [34]. Krylov subspace methods stops in a finite number of steps and no limit of convergence or rate of convergence can be formed. Therefore, it is necessary to consider the convergence problem from the different point of view than in the analysis of classical fixed point iteration methods such as the Gauss-Seidel or SOR. The convergence of such methods was analyzed by applying the concept of the asymptotic convergence factor. Early stage of convergence can depend significantly on the right hand side b and the initial guess $\mathbf{x}^{(0)}$. Generally, the convergence analysis is a difficult nonlinear problem although the system $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b}$ is linear because of the non-existing limiting process, the relevance of the transient phase, and the dependence of this phase on the RHS and the first choice of the solution.

Polar decomposition of normal matrices

Assume that $\mathbf{A}$ is a nonsingular and normal matrix, and therefore it is possible to perform its spectral decomposition, as follows

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H \tag{6.57}$$

where $\mathbf{V}\mathbf{V}^H = \mathbf{I}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$.

Such decomposition leads to a simplification of the convergence analysis of Krylov subspace methods, because $\mathbf{A}^n = \mathbf{V}\mathbf{\Lambda}^n\mathbf{V}^H$ and writting the error and the residual in the polynomial form

$$\begin{aligned} \mathbf{x} - \mathbf{x}^{(n)} &= p_n(\mathbf{A})(\mathbf{x} - \mathbf{x}^{(0)}), \\ \mathbf{r}^{(n)} &= p_n(\mathbf{A})\mathbf{r}^{(0)}, \end{aligned} \tag{6.58}$$

results in to the following forms of the errors and residuals

$$\begin{aligned} \mathbf{x} - \mathbf{x}^{(n)} &= \mathbf{V}p_n(\Lambda)\mathbf{V}^H(\mathbf{x} - \mathbf{x}^{(0)}), \\ \mathbf{r}^{(n)} &= \mathbf{V}p_n(\Lambda)\mathbf{V}^H\mathbf{r}^{(0)}, \end{aligned} \tag{6.59}$$

where p_n is the polynomial of degree at most n and with the value one at the origin. In the worst case, the convergence speed of MINRES is determined by the value

$$\nu = \min_{p\in\pi_n}\max_k |p(\lambda_k)|, \tag{6.60}$$

where π_n is the set of polynomials of degree at most n with value one at origin. The value ν represents a min-max approximation problem on the discrete set of the matrix eigenvalues. It depends nonlinearly on the eigenvalue decomposition.

Min-max estimation in the case of the Hermitian matrix

Assuming that $\mathbf{A}$ is a Hermitian positive definite matrix, i.e. it is a square matrix with complex entries which is equal to its own conjugate transpose, that can be written as $a_{ij} = a^*_{ji}$. Therefore, the A-norm is defined as follows $\|\mathbf{x}\|_A = (\mathbf{x}^H\mathbf{A}\mathbf{x})^{1/2}$, and the Krylov subspace iterates $\mathbf{x}_n$ are uniquely defined in each iteration step n by using the CG method. Then the value ν in Eq. 6.60 can be expressed as an upper bound on the relative error

$$\frac{\|\mathbf{x} - \mathbf{x}^{(n)}\|_A}{\|\mathbf{x} - \mathbf{x}^{(0)}\|_A} \leq \min_{p\in\pi_n}\max_k |p(\lambda_k)|. \tag{6.61}$$

When the matrix $\mathbf{A}$ is nonsingular and normal, i.e. $\mathbf{A}$ is a complex square matrix, when $\mathbf{A}^*\mathbf{A} = \mathbf{A}\mathbf{A}^*$, where $\mathbf{A}^*$ is the conjugate transpose of $\mathbf{A}$. When $\mathbf{A}$ is a real matrix, then $\mathbf{A}^* = \mathbf{A}^T$, and it is normal if $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T$. In this case the relative residual norm is bounded similarly as in Eq. 6.61, i.e.

$$\frac{\|\mathbf{r}^{(n)}\|}{\|\mathbf{r}^{(0)}\|} \leq \min_{p\in\pi_n}\max_k |p(\lambda_k)|. \tag{6.62}$$

The worst-case MINRES convergence behavior for the case, when $\mathbf{A}$ is Hermitian indefinite, can be obtained from Eq. 6.62 by replacing the set of of eigenvalues by the union of two subintervals I^+ and I^- such that

$$I^+ \cup I^- = [\lambda_{min}, \lambda_s] \cup [\lambda_{s+1}, \lambda_{max}] \tag{6.63}$$

containing all of them and excluding the origin. Moreover, eigenvalues are ordered as follows $\lambda_{min} \leq \lambda_s < 0 < \lambda_{s+1} \leq \lambda_{max}$, and both subintervals of eigenvalues are of the same length, i.e. $\lambda_s - \lambda_{min} = \lambda_{max} - \lambda_{s+1}$. Then, the bound for the min-max value can be proposed,

$$\begin{aligned} \min_{p\in\pi_n}\max_k|p(\lambda_k)| &\leq \min_{p\in\pi_n}\max_{z\in(I^+\cup I^-)}|p(z)| \\ &\leq 2\left(\frac{|\lambda_{min}\lambda_{max}|^{1/2} - |\lambda_s\lambda_{s+1}|^{1/2}}{|\lambda_{min}\lambda_{max}|^{1/2} + |\lambda_s\lambda_{s+1}|^{1/2}}\right)^{[k/2]}, \end{aligned} \tag{6.64}$$

where $[k/2]$ denotes the integer part of $k/2$.

At the end of this subsection it must be emphasized that there is the principal difference between Eq. 6.62 and Eq. 6.64. The relative bounds describe different approximation problems and the corresponding values could be significantly different.

6.5.4 Generalized minimal residual method (GMRES)

This subsection follows the ideas presented in [57]. The generalized minimal residual method (GMRES) is an iterative method developed in [40] for the solution of a system of linear equations $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b}$, where $\mathbf{A}$ is an invertible matrix of the size $n \times n$, and the RHS of $\mathbf{b}$ is normalized, i.e. $\|\mathbf{b}\| = 1$, with the $\|\cdot\|$ Euclidean norm.

The numerical solution is expressed by the vector $\mathbf{x}^{(n)} \in \mathcal{K}_n$ in the Krylov subspace that minimizes the norm of the residual $\|\mathbf{r}^{(n)}\| = \|\mathbf{A}\mathbf{x}^{(n)} - \mathbf{b}\|$, where $\mathcal{K}_n = \{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2, \ldots, \mathbf{A}^{n-1}\mathbf{b}\}$ with minimal residual.

The vectors $\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{n-1}\mathbf{b}$ are almost linearly dependent and they hardly can be used as a basis for the linear space. Therefore, another corresponding set of vectors should be proposed for the basis of $\mathcal{K}_n$. The Arnoldi iteration is applied to find orthonormal vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ that are the basis for $\mathcal{K}_n$. The suitable algorithm is the following [58]:

- Choose an arbitrary vector $\mathbf{q}_1$, such that $\|\mathbf{q}_1\| = 1$.
- Repeat for $k = 2, 3, \ldots,$

- $\mathbf{q}_k \leftarrow \mathbf{A}_k \mathbf{q}_{k-1}$
- for j from 1 to $(k-1)$
  - $h_{j,k-1} \leftarrow \mathbf{q}_j^* \mathbf{q}_k$
  - $\mathbf{q}_k \leftarrow (\mathbf{q}_k - h_{j,k-1}\mathbf{q}_j)$
- $h_{k,k-1} \leftarrow \|\mathbf{q}_k\|$
- $\mathbf{q}_k \leftarrow \frac{\mathbf{q}_k}{h_{k,k-1}}$

The internal loop: "for j from 1 to $(k-1)$" projects out the vector $\mathbf{q}_k$ in the directions $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{k-1}$ and this provides the orthogonality of generated vectors.

Therefore, each vector $\mathbf{x}^{(n)}$ of $\mathcal{K}_n$ can be expressed by $\mathbf{x}^{(n)} = \mathbf{Q}_n \mathbf{y}^{(n)}$, where $\mathbf{y}^{(n)} \in \mathbf{R}^n$ and $\mathbf{Q}_n$ is the $m \times n$ matrix generated by the first n-Arnoldi vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$. Scalars produced by the Arnoldi algorithm $h_{j,k}$ are elements of so called upper Hessenberg matrix of the form

$$\mathbf{H}_n = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \ldots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \ldots & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \ldots & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & h_{n,n-1} & h_{n,n} \end{bmatrix}$$

The matrix $\mathbf{H}_n$ can be expressed by $\mathbf{H}_n = \mathbf{Q}_n^* \mathbf{A} \mathbf{Q}_n$. Therefore, the Arnoldi iteration can be seen as a partial orthogonal reduction of $\mathbf{A}$ to the Hessenberg form. The matrix $\mathbf{H}_n$ can be viewed as the representation in the basis formed by the Arnoldi vectors of the orthogonal projection of $\mathbf{A}$ onto $\mathcal{K}_n$.

The relation between $\mathbf{Q}_n$ and $\mathbf{Q}_{n+1}$ matrices and $\mathbf{A}$ is the following

$$\mathbf{A}\mathbf{Q}_n = \mathbf{Q}_{n+1}\tilde{\mathbf{H}}_n \tag{6.65}$$

where $\tilde{\mathbf{H}}_n$ is an $(n+1) \times n$ matrix formed by adding the $(n+1)$-th row to $\mathbf{H}_n$ and is defined by

$$\mathbf{H}_n = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \ldots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \ldots & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \ldots & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & h_{n,n-1} & h_{n,n} \\ 0 & \ldots & \ldots & 0 & h_{n,n+1} \end{bmatrix}$$

When $\mathbf{Q}_n$ is orthogonal, the following relation is fulfilled

$$\|\mathbf{A}\mathbf{x}^{(n)} - \mathbf{b}\| = \|\tilde{\mathbf{H}}_n\mathbf{y}^{(n)} - \mathbf{e}^{(1)}\| \tag{6.66}$$

where $\mathbf{e}^{(1)} = (1, 0, 0, \ldots, 0)$ is the first vector of the $\mathbf{R}^{n+1}$ basis. Therefore, the solution $\mathbf{x}^{(n)} \in \mathcal{K}_n$ can be evaluated by minimizing the residual vector

$$\|\mathbf{r}^{(n)}\| = \min_{\mathbf{y}^{(n)} \in \mathcal{K}_n} \|\tilde{\mathbf{H}}_n\mathbf{y}^{(n)} - \mathbf{e}^{(1)}\| \tag{6.67}$$

This problem can be solved as a linear least squares problem of size n where the objective function, S, is defined as a sum of squared residuals $r^{(i)} = \|\mathbf{r}^{(i)}\|$, i.e.

$$S = \sum_{1=1}^{i=m} [r^i]^2. \tag{6.68}$$

This is the basic concept of the GMRES procedure, where at each step of iteration the following operations are performed:

- do one step of the Arnoldi method,
- find $\mathbf{y}^{(n)}$ vector which minimizes $\|\mathbf{r}^{(n)}\|$,
- compute $\mathbf{x}^{(n)} = \mathbf{Q}_n\mathbf{y}^{(n)}$,
- repeat from the first step, when the residual is not yet small enough.

6.5.5 GMRES convergence analysis

The first result of convergence analysis of GMRES, which appeared in [40], was an extension of convergence analysis for MINRES applied to Hermitian systems. The analysis [61] is based on the spectral decomposition of matrix $\mathbf{A} \in \mathcal{C}^{n\times n}$, i.e.

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}, \tag{6.69}$$

where $\mathbf{\Lambda} = diag(\lambda)$, $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_n]^T$, and $\lambda_j \in \mathcal{C}, \lambda_j \neq 0$. The bond on the residual $\mathbf{r}_m$ at the m-th step relatively to the initial guess $\mathbf{r}_0$ is

$$\frac{\|\mathbf{r}^{(m)}\|}{\|\mathbf{r}^{(0)}\|} \leq \kappa_2(\mathbf{V}) \min_{p_m \in \pi_m} \max_{i=1,2,\ldots,n} |p_m(\lambda_i)|, \tag{6.70}$$

where π_m is the set of polynomials of degree m that equal one at zero, and $\kappa_2(\mathbf{V}) = \|\mathbf{V}\|\|\mathbf{V}^{-1}\|$ is the condition number of the eigenvector matrix. When matrix $\mathbf{A}$ is normal then $\kappa_2 = 1$ and Eq.6.70 becomes Eq.6.62. This estimation is less useful, when $\mathbf{A}$ is not normal, and then RHS can be large because of an almost singular matrix $\mathbf{A}$. It should be noted that for Hermitian $\mathbf{A}$, GMRES is equivalent to MINRES. Unfortunately, it is well known that eigenvalues alone cannot explain GMRES behavior.

Chapter 7

Numerical solutions of ordinary differential equations (ODE)

7.1 Finite differences

We already know that the differences could issued from the interpolation schemes. Now the finite differences derived from the Taylor series expansion will be discussed. The approximations of derivatives are forward difference, backward difference or the central difference approximations. The following table describes [6], [7], [10], [12], [17], [24], [32] the applicability of various finite difference schemes.

Finite difference	*When applied*
forward difference	values of the function are available only at the mesh point and some forward equally spaced points
backward difference	values of the function at forward points are not known
central difference	values of the function are available at both the forward and the backward points

These approximations are widely applied for solution of ordinary differential and partial differential equations. The finite differences can be also used for numerical differentiation of digital data when a generating function is unknown. The order of the approximation, i.e. the accuracy of the differential scheme, depends on the number of sampling points - number of "active" nodes used in a particular scheme. Generally, the greater is the number of active nodes, the higher is the accuracy of a differential scheme.

7.1.1 First finite difference

Assume that a function $f(x)$ is a continuous function of the class $f(x) \in \mathcal{C}^{n+1}$ on $x \in (a, b) \in \mathcal{R}$. The Taylor series expansion of $f(x)$ in the neighborhood of some point $x = x_0$ is expressed by

$$\begin{aligned} f(x) &= f_0 + \frac{\dot{f}_0}{1!}(x - x_0) + \frac{\ddot{f}_0}{2!}(x - x_0)^2 + \dots \\ &+ \frac{f_0^{(n)}}{n!}(x - x_0)^n + \frac{f_0^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \end{aligned} \tag{7.1}$$

where $f(x_0) = f_0$, $\dot{f}_0 = \dot{f}_{|x=x_0}$, $f_0^{(n)} = \frac{d^n}{dx^n}(f)_{|x=x_0}$, and $\xi \in (x - x_0)$. The Taylor series can be also written in a more compact form

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x - x_0)^n, \tag{7.2}$$

where $n!$ is the factorial of n, the zeroth derivative of $f(x)$ is defined to be $f(x)$ itself and $(x - x_0)^0$ and 0! are equal to one.

The first finite difference (FFD) is obtained considering the first three RHS terms of Eq.7.1, i.e.

$$f(x) = f_0 + \dot{f}_0(x - x_0) + \frac{\ddot{f}_0}{2}(x - x_0)^2 \tag{7.3}$$

taking $x = x_1$, $f(x_1) = f_1$ and moving f_0 in to the LHS

$$f_1 - f_0 = \dot{f}_0(x_1 - x_0) + \frac{\ddot{f}_0}{2}(x_1 - x_0)^2 \tag{7.4}$$

the approximation of $\dot{f}_0$ is obtained by dividing both sides by $(x_1 - x_0)$

$$\frac{f_1 - f_0}{x_1 - x_0} = \dot{f}_0 + \frac{\ddot{f}_0}{2}(x_1 - x_0). \tag{7.5}$$

Exchanging RHS with LHS, taking $h = x_1 - x_0$ and replacing $f_1 - f_0$ by $\triangle f_0$, the final form of the first finite difference can be expressed by

$$\dot{f}_0 = \frac{\triangle f_0}{h} + O(h) \tag{7.6}$$

where $O(h) = \frac{\ddot{f}_0}{2}h$. This is the first order difference respectively the power of h increment.

The second order difference can be obtained following the steps:

1. consider the truncation of the Taylor series with four terms, where the third derivative is included

$$\begin{aligned} f(x) &= f_0 + \frac{\triangle f_0}{h}(x - x_0) + \frac{\triangle^2 f_0}{2!h^2}(x - x_0)(x - x_1) \\ &+ \frac{f(\xi)^{(3)}}{3!}(x - x_0)(x - x_1)(x - x_2) \end{aligned} \tag{7.7}$$

2. assuming that the grid is equally spaced along the x-axis, i.e. $h = (x - x_0) = (x - x_1) = (x - x_2)$, take the first derivative of the above

$$\dot{f}(x) = \frac{\triangle f_0}{h} + \frac{\triangle^2 f_0}{h}(x - x_0) + 0(h^2) \tag{7.8}$$

3. express $\triangle$ and $\triangle^2$ explicitly by differences

$$\begin{aligned} \dot{f}(x) &= \frac{f_1 - f_0}{h} - \frac{f_2 - 2f_1 + f_0}{2h} + 0(h^2) && (7.9) \\ &= \frac{2f - 1 - 2f_0 - f_2 + 2f_1 - f_0}{2h} + 0(h^2) \\ &= \frac{1}{h}(-\frac{3}{2}f_0 + 2f_1 - \frac{1}{2}f_2) + 0(h^2), && (7.10) \end{aligned}$$

4. taking the second RHS term with plus, the another differential scheme of the second order can be obtained

$$\begin{aligned} \dot{f}(x) &= \frac{f_1 - f_0}{h} + \frac{f_2 - 2f_1 + f_0}{2h} + 0(h^2) \\ &= \frac{1}{2h}(f_2 - f_0) + 0(h^2), && (7.11) \end{aligned}$$

The last formula can be also obtained in the general form following another procedure:

- write the Taylor series expansion taken at two points: x_{i+1} and x_{i-1}

$$f_{i+1} = f_i + h\dot{f}_i + \frac{h^2}{2}\ddot{f}_i + \frac{h^3}{6}f_i^{(3)} + \ldots \tag{7.12}$$

$$f_{i-1} = f_i - h\dot{f}_i + \frac{h^2}{2}\ddot{f}_i - \frac{h^3}{6}f_i^{(3)} + \ldots \tag{7.13}$$

- subtract of the two Taylor series expansions

$$\dot{f}_i = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{6}f_i^{(3)} \equiv \frac{1}{2h}(f_{i+1} - f_{i-1}) + 0(h^2) \tag{7.14}$$

7.1.2 Second finite difference

The second finite difference (SFD) is obtained following the same procedure as for FFD where finally the summation of two Taylors series expansions like Eq.7.12 and Eq.7.13 with the fourth derivatives, i.e.

$$f_{i+1} = f_i + h\dot{f}_i + \frac{h^2}{2}\ddot{f}_i + \frac{h^3}{6}f_i^{(3)} + \frac{h^4}{24}f_i^{(4)} + \dots \tag{7.15}$$

$$f_{i-1} = f_i - h\dot{f}_i + \frac{h^2}{2}\ddot{f}_i - \frac{h^3}{6}f_i^{(3)} + \frac{h^4}{24}f_i^{(4)} + \dots \tag{7.16}$$

results in the following differential scheme

$$\ddot{f}_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + 0(h^2) \tag{7.17}$$

7.2 Advanced finite difference approximations based on the Taylor series expansions

The so called close-form expressions for the first and higher order derivatives will be presented here following three papers: [28], [29], [30]. The approximations of very high order and obviously high accuracy can be achieved quite easily.

7.2.1 Forward difference approximations

Assume that $f(t)$ is a time function evaluated at discrete points $t = kT$, where $k = 0, \pm 1, \pm 2, ...$ and T is the sampling period like h in the previous sections. The discrete time values of $f(t)$ and the value of the function and its derivatives at the origin $t = 0$ are related by the Taylor series expansion. This can be expressed by

$$f_k = f_0 + kTf_0^{(1)} + \frac{(kT)^2}{2!}f_0^{(2)} + \frac{(kT)^n}{n!}f_0^{(n)} + \ldots + 0(T^{n+1}), \tag{7.18}$$

where the term $0(T^{n+1})$ comes from the truncation of the series after $(n+1)$ terms. The above Taylor series can be written in the matrix form as

$$\mathbf{F}_F = \mathbf{A}_F \cdot \mathbf{D}_F + \mathbf{O}(T^{n+1}), \tag{7.19}$$

where

$$\mathbf{A}_F = \begin{bmatrix} T & \frac{T^2}{2!} & \dots & \frac{T^n}{n!} \\ 2T & \frac{(2T)^2}{2!} & \dots & \frac{(2T)^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ nT & \frac{(nT)^2}{2!} & \dots & \frac{(nT)^n}{n!} \end{bmatrix},$$

$$\mathbf{F}_F = \begin{bmatrix} f_1 - f_0 \\ f_2 - f_0 \\ \vdots \\ f_n - f_0 \end{bmatrix},$$

$$\mathbf{D}_F = \begin{bmatrix} f_0^{(1)} \\ f_0^{(2)} \\ \vdots \\ f_0^{(n)} \end{bmatrix},$$

where the subscript F denotes the forward difference approximation.

Neglecting the remainder terms, Eq.7.19 can be written as

$$\mathbf{F}_F \approx \mathbf{A}_F \cdot \mathbf{D}_F, \tag{7.20}$$

and the first derivative can be find from the relation

$$f_0^{(1)} \approx \frac{\det\hat{\mathbf{A}}_F}{\det\mathbf{A}} \equiv \frac{|\hat{\mathbf{A}}_F|}{|\mathbf{A}|} \tag{7.21}$$

where $\det\mathbf{A}$ and $\det\mathbf{A}_F$ means the determinant of the matrix $\mathbf{A}$ or $\mathbf{A}_F$ that can be also denoted by $|\mathbf{A}|$ and $|\mathbf{A}_F|$. This is defined by

$$\hat{\mathbf{A}}_F = \begin{bmatrix} f_1 - f_0 & \frac{T^2}{2!} & \dots & \frac{T^n}{n!} \\ f_2 - f_0 & \frac{(2T)^2}{2!} & \dots & \frac{(2T)^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ f_n - f_0 & \frac{(nT)^2}{2!} & \dots & \frac{(nT)^n}{n!} \end{bmatrix},$$

The determinant of $\mathbf{A}_F$ equals to unity whenever $T = 1$ for any order of the matrix. Therefore, Eq.7.21 can be expressed in the full form as

$$f_0^{(1)} = \frac{1}{T} \begin{vmatrix} f_1 - f_0 & \frac{1}{2!} & \dots & \frac{1}{n!} \\ f_2 - f_0 & \frac{2^2}{2!} & \dots & \frac{2^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ f_n - f_0 & \frac{n^2}{2!} & \dots & \frac{n^n}{n!} \end{vmatrix},$$

The coefficient of f_k is equal to the product of the coefficient $\frac{1}{T}$ and the minor M_{k1} of matrix $\mathbf{A}_F$ formed by removing the k-th row and the first column. The coefficient of f_0 is evaluated by multiplying $-\frac{1}{T}$ by determinant of $\mathbf{A}_F$.

The Eq.**??** can be written in the following forms

$$f_0^{(1)} = \frac{1}{T}\sum_{k=0}^{n} g_{k,n}^{F,1} f_k + O(T^n) \tag{7.22}$$

or

$$f_i^{(i)} = \frac{1}{T}\sum_{k=0}^{n} g_{k,n}^{F,1} f_{k+i} + O(T^n) \tag{7.23}$$

where $g_{k,n}^{F,1}$ is the coefficient of a term f_k in the forward difference approximation of the first derivative of order n. The coefficients $g_{k,n}^{F,1}$ were calculated in [28] for various orders of the approximation. It was find that for the forward difference approximations of the first derivative they can be explicitly expressed by the following formulas

$$g_{0,n}^{F,1} = -\sum_{j=1}^{n}\frac{1}{j} \quad \text{for} \quad k=0 \tag{7.24}$$

$$g_{k,n}^{F,1} = \frac{(-1)^{k+1}}{k} C_k^n \quad \text{for} k = \quad 1,2,\ldots,n \tag{7.25}$$

where $C_k^n = \frac{n!}{k!}(n-k)!$.

It was observed that $g_{0,n}^{F,1}$ is an additive inverse of $\sum_{k=1}^{n} g_{k,n}^{F,1}$ and the sum of coefficients $g_{0,n}^{F,1}$ for $k = 0,1,2,\ldots,n$ equals to zero. This satisfies the requirement that the differentiation of a constant function results in zero.

For higher values of n, the coefficients C_k^n can be evaluated by

$$\begin{aligned} g_{1,n}^{F,1} &= n \\ g_{k,n}^{F,1} &= -g_{k-1,n}^{F,1}\frac{k-1}{k^2}(n-k+1), \quad k=2,3,4,\ldots,n \end{aligned} \tag{7.26}$$

7.2.2 Backward difference approximations

The Taylor series with $(n+1)$ terms for the backward expansion of a function $f(x)$ is

$$\mathbf{F}_B = \mathbf{A}_B \cdot \mathbf{D}_B + \mathbf{O}(T^{n+1}), \tag{7.27}$$

where $\mathbf{F}_B$ is a vector of length n and $\mathbf{A}_B$ is a $n \times n$ matrix. The vector $\mathbf{D}_B$ is the same as $\mathbf{D}_F$. These objects are defined as follows

$$\mathbf{A}_B = \begin{bmatrix} -T & \frac{T^2}{2!} & \dots & -\frac{T^n}{n!} \\ -2T & \frac{(2T)^2}{2!} & \dots & -\frac{(2T)^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ -nT & \frac{(nT)^2}{2!} & \dots & -\frac{(nT)^n}{n!} \end{bmatrix},$$

$$\mathbf{F}_F = \begin{bmatrix} f_{-1} - f_0 \\ f_{-2} - f_0 \\ \vdots \\ f_{-n} - f_0 \end{bmatrix},$$

The (j,k)-th element of matrix $\mathbf{A}_B$ is defined as $\mathbf{A}_B(j,k) = -\frac{(-jT)^k}{k!}$.

Following the same procedure as in the case of the forward difference approximation, the first backward derivative can be evaluated by

$$f_0^{(1)} = -\frac{1}{T} \begin{vmatrix} f_{-1} - f_0 & \frac{1}{2!} & \dots & -\frac{1}{n!} \\ f_{-2} - f_0 & \frac{2^2}{2!} & \dots & -\frac{2^n}{n!} \\ \vdots & \vdots & \ddots & \vdots \\ f_{-n} - f_0 & \frac{n^2}{2!} & \dots & -\frac{n^n}{n!} \end{vmatrix},$$

It is mentioned in [28] that the coefficient $g_{k,n}^{B,1}$ of the term f_{-k} in the backward difference approximation is the additive inverse of the coefficient $g_{k,n}^{F,1}$ of the term f_k in the forward difference approximation, i.e. $g_{k,n}^{F,1} + g_{k,n}^{B,1} = 0$.

The backward difference approximation of order n can be written in the compact form as

$$f_{-i}^{(1)} = \frac{1}{T} \sum_{k=-n}^{0} g_{k,n}^{B,1} f_{k+1} + O(T^n), \tag{7.28}$$

where

$$g_{0,n}^{B,1} = -g_{0,n}^{F,1} = \sum_{j=1}^{n} \frac{1}{j}, \tag{7.29}$$

$$g_{k,n}^{B,1} = -g_{k,n}^{F,1} = \frac{(-1)^k}{k} C_k^n, \quad k = 1, 2, \dots, n. \tag{7.30}$$

Coefficients of the forward difference approximations can be evaluated as follows

$$\begin{aligned} g_{-1,n}^{B,1} &= -n \\ g_{-k,n}^{B,1} &= g_{-k+1,n}^{B,1} \frac{k-1}{k^2}(n-k+1), \quad k = 2, 3, 4, \dots, n. \end{aligned} \tag{7.31}$$

7.2.3 Central difference approximations

The central difference approximations of the current value of the derivative are calculated on the basis of both the backward and the forward values of the function. The derivative is evaluated by solving of $2n$ equations generated from the Taylor expansion containing $(2n+1)$ terms for the function $f(t)$ taken at $t = kT$ points for $k = \pm 1, \pm 2, \ldots, \pm n$. The set of these equations can be written as

$$\mathbf{F}_C = \mathbf{A}_C \cdot \mathbf{D}_C + \mathbf{O}(T^{2n+1}), \tag{7.32}$$

where

$$\mathbf{A}_C = \begin{bmatrix} T & \frac{T^2}{2!} & \frac{T^3}{3!} & \cdots & \frac{T^{2n}}{(2n)!} \\ -T & \frac{-T^2}{2!} & \frac{-T^3}{3!} & \cdots & \frac{-T^{2n}}{(2n)!} \\ 2T & \frac{(2T)^2}{2!} & \frac{(2T)^3}{3!} & \cdots & \frac{(2T)^{2n}}{(2n!)} \\ -2T & \frac{(-2T)^2}{2!} & \frac{(-2T)^3}{3!} & \cdots & \frac{(-2T)^{2n}}{(2n!)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ nT & \frac{(nT)^2}{2!} & \frac{(nT)^3}{3!} & \cdots & \frac{(nT)^{2n}}{2n!} \\ -nT & \frac{(-nT)^2}{2!} & \frac{(-nT)^3}{3!} & \cdots & \frac{(-nT)^{2n}}{2n!} \end{bmatrix},$$

$$\mathbf{F}_C = \begin{bmatrix} f_1 - f_0 \\ f_{-1} - f_0 \\ f_2 - f_0 \\ f_{-2} - f_0 \\ f_3 - f_0 \\ f_{-3} - f_0 \\ \vdots \\ f_n - f_0 \\ f_{-n} - f_0 \end{bmatrix},$$

$$\mathbf{D}_C = \begin{bmatrix} f_0^{(1)} \\ f_0^{(2)} \\ f_0^{(3)} \\ f_0^{(4)} \\ \vdots \\ f_0^{(2n-1)} \\ f_0^{(2n)} \end{bmatrix},$$

The (j,k)-th element of matrix $\mathbf{A}_C$ is defined by

$$\mathbf{A}_C(j,k) = \frac{1}{k!}\{\text{int}[\frac{j+1}{2}] \cdot (-1)^{j+1} \cdot T\}^k \tag{7.33}$$

where int$[\frac{j+1}{2}]$ rounds off $(i-\frac{1}{2})$ to the nearest integer value.

Using the central difference approximation, the first derivative can be expressed by

$$f_0^{(1)} \approx \frac{1}{T}\begin{vmatrix} f_1-f_0 & \frac{T^2}{2!} & \frac{T^3}{3!} & \cdots & \frac{T^{2n}}{(2n)!} \\ f_{-1}-f_0 & \frac{-T^2}{2!} & \frac{-T^3}{3!} & \cdots & \frac{-T^{2n}}{(2n)!} \\ f_2-f_0 & \frac{(2T)^2}{2!} & \frac{(2T)^3}{3!} & \cdots & \frac{(2T)^{2n}}{(2n!)} \\ f_{-2}-f_0 & \frac{(-2T)^2}{2!} & \frac{(-2T)^3}{3!} & \cdots & \frac{(-2T)^{2n}}{(2n!)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_n-f_0 & \frac{(nT)^2}{2!} & \frac{(nT)^3}{3!} & \cdots & \frac{(nT)^{2n}}{2n!} \\ f_{-n}-f_0 & \frac{(-nT)^2}{2!} & \frac{(-nT)^3}{3!} & \cdots & \frac{(-nT)^{2n}}{2n!} \end{vmatrix},$$

This can be written in the compact form as

$$f_i^{(1)} = \frac{1}{T}\sum_{k=-n}^{n} g_{k,2n}^{C,1} f_{k+i} + O(T^{2n}), \tag{7.34}$$

where

$$g_{0,2n}^{C,1} = 0, \tag{7.35}$$

$$g_{k,2n}^{C,1} = (-1)^{k+1}\frac{(n!)^2}{k(n-k)!(n+k)!} \quad k = \pm 1, \pm 2, \ldots, \pm n. \tag{7.36}$$

The above coefficients can be evaluated easily using the following formulas

$$\begin{aligned} g_{1,2n}^{C,1} &= \frac{n}{n+1} \\ g_{k,2n}^{C,1} &= -g_{k-1,2n}^{C,1}\frac{(k-1)(n-k+1)}{k(n+k)} \quad k = 2, 3, 4, \ldots, n. \\ g_{-k,2n}^{C,1} &= -g_{k,2n}^{C,1} \quad k = 1, 2, 3, \ldots, n. \end{aligned} \tag{7.37}$$

It can be seen that

$$\sum_{k=-n}^{n} g_{k,2n}^{C,1} = 0 \tag{7.38}$$

which ensures that the slope of a constant function equals to zero.

7.2.4 Approximations of the second and higher order derivatives

In this section, the second derivatives are defined by applying the forward, backward and central difference approximations.

The second derivatives can be expressed by using the *froward values* of function $f(x)$ in the following form

$$f_i^{(2)} = \frac{1}{T^2} \sum_{k=0}^{n} g_{k,n}^{F,2} f_{k+i} + O(T^n) \tag{7.39}$$

where

$$g_{k,n}^{F,2} = \frac{(-1)^k C_k^n}{k} \sum_{j=1, j\neq k}^{n} \frac{1}{j}, \quad k = 1, 2, 3, \ldots, n, \tag{7.40}$$

$$g_{0,n}^{F,2} = -\sum_{k=1}^{n} g_{k,n}^{F,2} \tag{7.41}$$

Basing the approximation scheme on the *backward values* of the function $f(x)$, the second derivatives are evaluated by the formula

$$f_i^{(2)} = \frac{1}{T^2} \sum_{k=-n}^{0} g_{k,n}^{B,2} f_{k+i} + O(T^n) \tag{7.42}$$

where $g_{k,n}^{B,2}$ is the coefficient of f_k corresponding to the k-th coefficient in the forward difference approximation which is defined by equations: Eq.7.40 and Eq.7.41, with F replaced by B.

The central difference approximations are used only for the higher derivatives because of their superiority over others. The p-th derivative of the $2n$-th order can be expressed by

$$f_i^{(p)} = \frac{1}{T^2} \sum_{k=-n}^{0} g_{k,2n}^{C,p} f_{k+i} + O(T^{2n}) \tag{7.43}$$

where

$$
\begin{aligned}
g^{C,p}_{0,2n} &= -2\sum_{k=1}^{n} g^{C,p}_{k,2n} \quad \text{for} \quad p = \text{even} && (7.44)\\
g^{C,p}_{0,2n} &= 0 \quad \text{for} \quad p = \text{odd} && (7.45)\\
\text{for} \quad k &= \pm 1, \pm 2, \ldots, \pm n \\
g^{C,2}_{k,2n} &= (-1)^{k+1}\frac{2!}{k^2}\frac{(n!)^2}{(n-k)(n+k)!}, && (7.46)\\
g^{C,3}_{k,2n} &= (-1)^{k}\frac{3!}{k}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\frac{1}{i^2}, && (7.47)\\
g^{C,4}_{k,2n} &= (-1)^{k}\frac{4!}{k^2}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\frac{1}{i^2}, && (7.48)\\
g^{C,5}_{k,2n} &= (-1)^{k+1}\frac{5!}{k}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\sum_{j=i+1,j\neq|k|}^{n}\frac{1}{(ij)^2}, && (7.49)\\
g^{C,6}_{k,2n} &= (-1)^{k+1}\frac{6!}{k}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\sum_{j=i+1,j\neq|k|}^{n}\frac{1}{(ij)^2}, && (7.50)\\
g^{C,7}_{k,2n} &= (-1)^{k}\frac{7!}{k}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\sum_{j=i+1,j\neq|k|}^{n}\sum_{l=j+1,l\neq|k|}^{n}\frac{1}{(ijl)^2}, && (7.51)\\
g^{C,8}_{k,2n} &= (-1)^{k}\frac{8!}{k}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1,i\neq|k|}^{n}\sum_{j=i+1,j\neq|k|}^{n}\sum_{l=j+1,l\neq|k|}^{n}\frac{1}{(ijl)^2}, && (7.52)\\
& && (7.53)
\end{aligned}
$$

The general expression for the k-th coefficient $g^{C,2}_{k,2n}$ in approximation of p-th derivative of order $2n$ can be find from Eqn. from 7.46 to 7.52 and can be written in the following form

$$
g^{C,p}_{k,2n} = (-1)^{k+d}\frac{p!}{k^{1+e}}\frac{(n!)^2}{(n-k)(n+k)!}\sum_{i=1}^{c}\frac{1}{\mathbf{X}(i)^2}, \tag{7.54}
$$

for $k = \pm 1, \pm 2, \ldots, \pm n$, where

$$\begin{aligned} c &= \text{int}[\frac{p-1}{2}] \\ d &= 1 \quad \text{for} \quad c = \text{even}, \\ d &= 0 \quad \text{for} \quad c = \text{odd}\ , \\ e &= 1 \quad \text{for} \quad p = \text{even}, \\ e &= 0 \quad \text{for} \quad p = \text{odd}\ . \end{aligned} \tag{7.55}$$

Vector $\mathbf{X}$ is generated as follows:

- Consider a vector $\mathbf{Y}$ containing all integers from 1 to n,
- The vector $\mathbf{X}$ of the length C_c^{n-1} contains the product of all possible combinations of length c in $\mathbf{Y}$.

Note that Eqs. 7.39, 7.43 and 7.54 define the explicit formulas for a central difference approximation of the first and any higher derivative of any arbitrary order.

7.3 First order ordinary differential equations

7.3.1 Concise introduction to ODE

In this section a brief repetition [1] of the ODE fundamentals will be given in a compact form. The basic definitions of order, linearity, initial conditions and solution will be recalled.

The first fundamental thing is distinguishing of a derivative type:

Derivatives	*Expressions*	*Description*
Ordinary derivative	$\frac{dw}{dt}$	$w(t)$ is a function of a single independent variable t
Partial derivatives	$\frac{\partial u}{\partial x}$	$u(x, y, \ldots, z)$ is a function of more than one independent variable

The diagnosis of a type of a differential equation is based on the examination of a number of independent variables, the assessment of a derivative type and a type of function:

- An ordinary differential equation (ODE) involves one or more ordinary derivatives of unknown functions with respect of a single independent variable.
- A partial differential equation (PDE) involves one or more partial derivatives of unknown functions taken with respect of various variables.

That is summarized in the following table:

Differential equation	*Expression*	*Description*
Ordinary differential equation	$a\frac{d^2w}{dt^2} + btw = c$ $a, b, c, t \in \mathbf{R}$	Involves one or more ordinary derivatives of unknown functions
Partial differential equation	$a\frac{\partial^2 u}{\partial y^2} - b\frac{\partial^2 u}{\partial x^2} = c$ $a, b, c, x, y \in \mathbf{R}$	Involves one or more partial derivatives of unknown functions

ODE examples:

- $$3\frac{dw(t)}{dt} + 2w(t) = 5e^{t+1},$$

  where $w(t)$ is unknown function and t is independent variable,

- $$2\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + 5y(t) = \frac{1}{2}\pi\sin(t),$$

  where $y(t)$ is unknown function and t is independent variable.

The order of an ordinary differential equation is determined by the order of a highest order derivative.

Examples illustrating the assessment of the order of ODE:

Examples of ODE	*Order*
$\frac{dy(t)}{dt} - 2y(t) = e^{t-1}$	first order
$\frac{d^2x(t)}{dt^2} - 3\frac{dx(t)}{dt} - 5x(t) = tg(t)$	second order
$3(\frac{d^2w(t)}{dt^2})^3 + 2\frac{dw(t)}{dt} - 5w^4(t) = 7$	second order

The solution of a differential equation is a function that satisfies the differential equation.

Example♣

Consider the following ODE:

$$\frac{dw(t)}{dt} + w(t) = 0.$$

The solution is $w(t) = e^{-t} + C$, where C is a constant such that $C \in \mathbf{R}$. This can be evaluated by substituting into the equation, i.e.

$$\frac{dw(t)}{dt} + w(t) = e^{t} - e^{t} = 0$$

♠

An ODE is linear when an unknown function and its derivatives, present in the equation, appear in a power one and there are neither products of a function and any derivative nor products of derivatives of any order.

ODE examples	*Linearity assessment*
$\frac{dy(t)}{dt} + y(t) = e^{t+1}$	Linear
$\frac{d^2z(t)}{dt^2} - 3\frac{dz(t)}{dt} + 3t^3 z(t) = sin(t)$	Linear
$(\frac{d^2z(t)}{dt^2})^2 - 2\frac{dz(t)}{dt} + 3z(t) = 6$	Non-linear because of $(\frac{d^2z(t)}{dt^2})^2$
$\frac{d^2z(t)}{dt^2} - 3z(t)\frac{dz(t)}{dt} = 1$	Nonlinear because of $z(t)\frac{dz(t)}{dt}$
$\frac{d^2z(t)}{dt^2} + \lvert\frac{dz(t)}{dt}\rvert = 3$	Nonlinear because of $\lvert\frac{dz(t)}{dt}\rvert$

Very important question is ***the uniqueness*** of ODE solution.

Consider the equation

$$\frac{d^2z(t)}{dt^2} + 4z(t) = 0$$

with the solution $z(t) = cos(2t))$. Is the solution unique? Obviously not! Why? Because in order to specify a solution to kth order differential equation n conditions are necessary. Therefore, two auxiliary conditions for the second order ODE , eg.

Expressions	*Condition type*
$z(t=0) = c$	Initial conditions
$\dot{z}(t=0) = d$	taken at the same point $t = 0$
$z(t=a) = g$	Boundary conditions
$z(t=b) = h$	taken at two different points $z \in [a, b]$

are necessary for the unique specification of the solution. Following above example it can be stressed that the initial conditions are taken at one point of the independent variable, but the boundary conditions are defined not at one point of the independent variable.

Respectively to auxiliary conditions, two problems can be specified for the same ODE

$$a\ddot{y} + b\dot{y} + cy = d$$

where coefficients are real, i.e. $a, b, c, d \in \mathbf{R}$,

Initial-Value problem	*Boundary-Value Problem*
$y(t=0) = g$, and $\dot{y}(t=0) = h$ where $g, h \in \mathbf{R}$	$y(t=k) = i$ and $y(t=l) = j$ where $i, j, k, l \in \mathbf{R}$ and $k \neq l$

Unfortunately, analytical solutions of ODEs are available only for linear ODEs and some classes of nonlinear equations. Therefore, the numerical methods are extensively used for solutions of ODEs and the solutions are given in forms of graphs or tables of the unknown function. Most of such numerical methods are based directly or indirectly on truncated Taylor series expansions.

7.3.2 Solution of ODE based on Taylor series method

The Taylor series method is based on the expansion of the given function $y = f(t)$ according to rules described in the previous sections devoted to the Taylor series expansion of a given function. In this method the subintervals $t_0 \leq t \leq t_m$ are required to be uniform. The implementation of the Taylor series method for solving the initial-value problem according to Eqs. 7.12 and 7.15 or Eqs. 7.13 and 7.16 is quite simple.

The algorithm for the Taylor series of order n method can be written as

- given $\dot{y} = f(t, y)$ with the initial-value condition $y_0 = y(t_0)$,
- apply m equal-width subintervals $h = \triangle t$,
- compute derivatives $\ddot{y}, y^{(3)}, \ldots, y^{(n)}$,
- for $i = 0$ to $m - 1$
- compute discrete values of $\ddot{y}_i, y_i^{(3)}, \ldots, y_i^{(n)}$
- compute y_{i+1} following Eq. 7.12 or Eq. 7.15
- Endfor

Example of the Taylor method ♣

Consider the initial-value problem for *ODE of the form:* $\dot{y} = t\cos 2y$, for $0 \le t \le 1$ divided with $h = \triangle t = 0.1$ and the initial condition $y(0) = -0.2$. *Find* y_i for $i = 1, \ldots, 10$ by applying the Taylor series method of order 3.

Solution

Write the Taylor series in the form

$$\begin{aligned} y_{i+1} &\approx y_i + h\dot{y}_i + \frac{h^2}{2!}\ddot{y}_i + \frac{h^3}{3!}y_i^{(3)} \\ &= y_i + 0.1\dot{y}_i + 0.005\ddot{y}_i + 0.000167y_i^{(3)} \end{aligned}$$

Analytical and discrete forms of derivatives of ODE RHS

$\dot{y} = t\cos 2y$	$\dot{y}_i = t_i \cos 2y_i$
$\ddot{y} = -2t\dot{y}\sin 2y + \cos 2y$	$\ddot{y}_i = -2t_i\dot{y}_i \sin 2y_i + \cos 2y_i$
$y^{(3)} = -2t\ddot{y}\sin 2y - 4\dot{y}\sin 2y$ $-4t(\dot{y})^2\cos 2y$	$y_i^{(3)} = -2t_i\ddot{y}_i\sin 2y_i - 4\dot{y}_i\sin 2y_i$ $-4t_i(\dot{y}_i)^2\cos 2y_i$

For $i = 0$ and $i = 1$ the discrete elements become

$i = 0$	$i = 1$
$\dot{y}_0 = t_0\cos 2y_0 = 0$	$\dot{y}_i = t_i\cos 2y_i$
$\ddot{y}_0 = -2t_0\dot{y}_0\sin 2y_0 + \cos 2y_0 = 0.921061$	$\ddot{y}_1 = -2t_i\dot{y}_1\sin 2y_1 + \cos 2y_1$
$y_0^{(3)} = -2t_0\ddot{y}_0\sin 2y_0 - 4\dot{y}_0\sin 2y_0$ $-4t(\dot{y})^2\cos 2y = 0$	$y_1^{(3)} = -2t_1\ddot{y}_1\sin 2y_1 - 4\dot{y}_1\sin 2y_1$ $-4t_1(\dot{y}_1)^2\cos 2y_1$
$y_0 = -0.2$	$y_1 = -0.19539$

The table of complete discrete solution of the ODE is the following

i	t_i	y_i
0	0	-0.20000
1	0.10000	-0.19539
2	0.20000	-0.18145
3	0.30000	-0.15788
4	0.40000	-0.12427
5	0.50000	-0.08022
6	0.60000	-0.02553
7	0.70000	0.03945
8	0.80000	0.11355
9	0.90000	0.19449
10	1.00000	0.27891

♠

Euler's method

The Euler's method (EM) is the special case of the Taylor's series method of order $n = 1$. The method for ODE of the form $\dot{y} = f(t, y(t))$ can be derived

using calculus

$$\int_{t_i}^{t_{i+1}} f(t, y(t))dt = \int_{t_i}^{t_{i+1}} \dot{y}(t)dt = y(t_{i+1}) - y(t_i). \tag{7.56}$$

Assuming that

$$\int_{t_i}^{t_{i+1}} \dot{y}(t)dt \approx (t_{i+1} - t_i)\dot{y}_i = h\dot{y}_i,$$

substituting $\dot{y}_i$ by the RHS of the ODE, i.e. $f(t, y(t))$ taken at the point t_i, and reordering Eq. 7.56, the $y(t_{i+1}) \equiv y_{i+1}$ can be evaluated from the following equation

$$y_{i+1} = y_i + hf(t_i, y_i). \tag{7.57}$$

The order of accuracy of the Euler's method is much lower than the Taylor's series method, because of only two-terms expansion of the series is used.

Example 1 of the Euler method ♣

Use Euler method to solve the initial value problem for

$$\frac{dy}{dt} = 1 + t^2,$$

with the initial condition

$$y(1) = -4,$$

to determine $y(1.01), y(1.02), y(1.03)$. Therefore, we have $t_0 = 1$, $y_0 = -4$ and $h = 0.01$.

Solution

Run the first three steps of EM:

Step number	*Finite difference*
1	$y_1 = y_0 + hf(x_0, y_0) = -4 + 0.01[1 + (1)^2] = -3.98$
2	$y_2 = y_1 + hf(x_1, y_1) = -3.98 + 0.01[1 + (1.01)^2] = -3.9598$
3	$y_3 = y_2 + hf(x_2, y_2) = -3.9598 + 0.01[1 + (1.02)^2] = -3.9394$

The summary of results is shown in the following table:

i	x_i	y_i
0	1.00	-4.00
1	1.01	-3.98
2	1.02	-3.9595
3	1.03	-3.9394

For this simple example we can compare the EM results with true values

i	y_i	*True value of* y_i
0	-4.00	-4.00
1	-3.98	-3.9799
2	-3.9595	-3.9595
3	-3.9394	-3.9309

♠

Following this table, the matter of numerical method errors should be mention. Three types of errors can be classified:

Type of error	*Description of error*
Local truncation error	The error produced due to the use of truncated Taylor series for a computation of $x(t+h)$ in one step.
Global truncation error	The error accumulated due to the truncation over many steps.
Round off error	The error which appears due to finite number of bits used in representation of numbers. This error could be accumulated and magnified in succeeding steps.

7.3.3 Solution of ODE by Runge-Kutta method of the second order

The Runge-Kutta (RK) method, invented by the German mathematicians C. Runge and M.W. Kutta in 1900, is an alternative for the Taylor series method and it is amongst the most popular method for the numerical solution for ODE. The method was invented due to seeking for accurate methods to solve ODE that do not require calculation of high order derivatives.

There is a class of RK methods which are single step numerical methods. The past approximations of the solution are not used, i.e. when getting the value of y_k as the numerical approximation of $y(t_k)$ the methods steps forward to the evaluation of y_{k+1} as an estimation of $y(t_{k+1}) = y(t_k + h)$ according to the following formula

$$y_{i+1} \approx y_0 + h \sum_{j=1}^{n} b_j k_j, \tag{7.58}$$

where n is the order of the RK equation, b_j is the weight, and k_j is the term of the Taylor series expansion of the equation $\dot{y} = f(t, x)$. The RK method of order 2 and 4 methods will be presented in this subsection, which have two and four terms in the summation term of Eq. 7.58.

The method is based on the approach to find a formula that involves unknown coefficients and then determine these coefficients that should match as many terms of the Taylor series expansions as possible.

The RK of the 2nd order method is a numerical technique, which is suitable only for the solution of the first order ODE of the general form:

$$\dot{y} = f(t, y), \quad y(0) = y_0. \tag{7.59}$$

For better understanding of the RK of the 2nd order method, the derivation of the Euler method from the Taylor series could be very helpful. For this purpose let repeat the derivation

$$y_{i+1} = y_i + \frac{dy}{dt}|_{t_i,y_i}(t_{i+1}-t_i) + \frac{1}{2!}\frac{d^2y}{dt^2}|_{t_i,y_i}(t_{i+1}-t_i)^2 + \frac{1}{3!}\frac{d^3y}{dt^3}|_{t_i,y_i}(t_{i+1}-t_i)^3 + \ldots, \tag{7.60}$$

and now replace derivatives of $y(t)$ by derivatives of RHS of ODE,

$$y_{i+1} = y_i + f(t_i, y_i)(t_{i+1}-t_i) + \frac{1}{2!}\dot{f}(t_i, y_i)(t_{i+1}-t_i)^2 + \frac{1}{3!}\ddot{f}(t_i, y_i)(t_{i+1}-t_i)^3 + \ldots. \tag{7.61}$$

The first two terms of the Taylor series

$$y_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i)$$

or written for equal width subintervals in the form

$$y_{i+1} = y_i + f(t_i, y_i)h$$

are the Euler method and hance can be considered as the RK method of the first order.

The true error of this approximation is given by

$$E_t = \frac{1}{2!}\dot{f}(t_i, y_i)h^2 + \frac{1}{3!}\ddot{f}(t_i, y_i)h^3 + \ldots. \tag{7.62}$$

Including one more term of the Taylor series

$$y_{i+1} = y_i + f(t_i, y_i)h + \frac{1}{2!}\dot{f}(t_i, y_i)h^2 \tag{7.63}$$

leads to the the RK method of the second order (RK2 method) expressed by

$$y_{i+1} = y_i + (b_1k_1 + b_2k_2)h \tag{7.64}$$

where

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + p_1h, y_i + q_{11}k_1h) \end{aligned} \tag{7.65}$$

The advantage of the RK2 method consists in the solution of ODE without having to calculate $\dot{f}(t,y)$. So, the next subject is how to find the unknowns b_1, b_2, p_1 and q_{11}. Equating Eq. 7.63 and Eq. 7.64 gives three equations

$$\begin{aligned} b_1 + b_2 &= 1 \\ b_2 p_1 &= \frac{1}{2} \\ b_2 q_{11} &= \frac{1}{2} \end{aligned}$$

Three values are generally used for b_2 and following that three methods are generated:

Heun's method	*Midpoint method*	*Ralston's method*
$b_2 = \frac{1}{2}$	$b_2 = 1$	$b_2 = \frac{2}{3}$
$b_1 = \frac{1}{2}$, $p_1 = 1$, $q_{11} = 1$	$b_1 = 0$, $p_1 = \frac{1}{2}$, $q_{11} = \frac{1}{2}$	$b_1 = \frac{1}{3}$, $p_1 = \frac{3}{4}$, $q_{11} = \frac{1}{3}$
$y_{i+1} = y_i + (\frac{1}{2}k_1 + \frac{1}{2}k_2)h$	$y_{i+1} = y_i + k_2 h$	$y_{i+1} = y_i + (\frac{1}{3}k_1 + \frac{2}{3}k_2)h$
$k_1 = f(t_i, y_i)$	$k_1 = f(t_i, y_i)$	$k_1 = f(t_i, y_i)$
$k_2 = f(t_i + h, y_i + k_1 h)$	$k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$	$k_2 = f(t_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h)$

The algorithm for the RK2 method is express as follows

- Given $\dot{y} = f(t,y)$, $y_0 = y(t_0)$, $h = \triangle t$, and integer m;
- for $i = 0$ to m
- compute $k_1 = f(t_i, y_i)$;
- compute $k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1)$;
- if $i < m$
- compute y_{i+1} using equation 7.64;
- update $t_{i+1} \leftarrow t_i + h$;
- Endif
- Endfor

Example 2 [27] ♣

A container at temperature of 1200 K is allowed to cool down in air at an ambient temperature of approximately 300 K. Assuming that cooling is driven only by the radiation, the differential equation for the temperature of the container is given by

$$\frac{d\theta}{dt} = -2.2067 \cdot 10^{-12}(\theta^4 - 81 \cdot 10^8)$$

where θ is in K and t-time is in seconds. Find the temperature after $t = 480$ seconds since the beginning of cooling process by using the RK2 method. Apply the step size $h = 240$ seconds.

Solution

The RHS of ODE describing cooling is

$$f(t, \theta) = -2.2067 \cdot 10^{-12}(\theta^4 - 81 \cdot 10^8).$$

Following the Heun's method

$i = 0$	$t_0 = 0$, $\theta_0 = 1200$
	$k_1 = f(t_0, \theta_0) = f(0, 1200) = -2.2067 \cdot 10^{-12}(1200^4 - 81 \cdot 10^8) = -4.5579$ $k_2 = f(t_0 + h, \theta_0 + k_1 h) = f(0 + 240, 1200 + (-4.5579)240) = 0.017595$ $\theta_1 = 1200 + [\frac{1}{2}(-4.5579) + \frac{1}{2}0.017595]240 = 655.16$K
$i = 1$	$t_1 = t_0 + h = 240$, $\theta_1 = 655.16$
	$k_1 = f(t_1, \theta_1) = -2.2067 \cdot 10^{-12}(655.16^4 - 81 \cdot 10^8) = -0.38869$ $k_2 = f(t_1 + h, \theta_1 + k_1 h) = f(480, 655.16 - 0.38869 \cdot 240) = -0.20206$ $\theta_2 = 655.16 + [\frac{1}{2}(-0.38869) + \frac{1}{2}(-0.20206)]240 = 584.27$K

Fortunately, the exact solution of the above ODE is available via solution of the following nonlinear equation

$$0.92593 \ln \frac{\theta - 300}{\theta + 300} - 1.8519 \arctan(0.00333\theta) = -0.22067 \cdot 10^{-3} t - 2.9282$$

and the exact solution at $t = 480$ seconds is $\theta(480) = 647.57$K. Following that, it is possible to evaluate the absolute relative true error by using the relation

$$|\epsilon_T| = \left| \frac{\text{exact solution} - \text{approximate solution}}{\text{approximate solution}} \right| \cdot 100 \tag{7.66}$$

The accuracy can be upgraded by using smaller time step size. The effect of the step size for the Heun's method is illustrated by data in the following table:

Step size h	$\theta(t = 480\text{s})$	E_T	$\|\epsilon\|_T\%$
480	-393.87	1041.40	160.82
240	584.27	63.30	9.78
120	651.35	-3.78	0.58
60	649.91	-2.34	0.36
30	648.21	-0.63	0.10

The comparison of the Euler method with three RK methods for the temperature θ at $t = 480$s is given in the following table

Step size h	*Euler*	*Heun*	*Midpoint*	*Ralston*
480	-987.84	-393.87	1208.40	449.78
240	110.32	584.27	976.87	690.01
120	546.77	651.35	690.20	667.71
60	614.97	649.91	654.85	652.25
30	632.77	648.21	649.02	648.61

The RK method of the second order is producing results with the local truncation error of the order $O(h^3)$. ♠

7.3.4 Solution of ODE by Runge-Kutta methods of higher order

The Runge-Kutta (RK) methods are until now the area of sophisticated research lead by many mathematicians with the world class leader Prof. John Butcher [4].

The classical RK method of order four, called also RK4, is derived from the Taylor series of order four by approximating the second, third, and forth derivatives. The classical method is defined by the following relations:

$$y_{i+1} \approx y_i + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4), \tag{7.67}$$

$$k_1 = f(t_i, y_i), \tag{7.68}$$

$$k_2 = f(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}), \tag{7.69}$$

$$k_3 = f(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}), \tag{7.70}$$

$$k_4 = f(t_i + h, y_i + k_3). \tag{7.71}$$

From the relationship 7.67 it can be seen that next value y_{i+1} is determined by the present value y_i increased by the product of the interval h and an estimated slope of the function $y(t)$. Numbers k_i, $i = 1, 2, \ldots, 4$ are slopes described in the following table

Slope	*Taken at a point*	*Evaluated by using*
k_1	the beginning of the interval	
k_2	the midpoint of the interval	slope k_2 for evaluation of $y(t_n + \frac{h}{2})$ by Euler's method
k_3	the midpoint of the interval	slope k_2 for evaluation of $y(t_n + \frac{h}{2})$
k_4	the end of the interval	slope k_3 for evaluation of $y(t_n + h)$

The error per step for RK4 method is of the order $O(h^5)$ but the total accumulated error is of the order $O(h^4)$.

The algorithm for the RK4 method for a single ODE is express as follows

- Given $\dot{y} = f(t, y)$, $y_0 = y(t_0)$, $h = \triangle t$, and integer m;
- for $i = 0$ to m
- compute $k_1 = f(t_i, y_i)$;
- compute $k_2 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1)$;
- compute $k_3 = f(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2)$;
- compute $k_4 = f(t_i + h, y_i + k_3)$;
- if $i < m$

- compute y_{i+1} using equation 7.67;
- update $t_{i+1} \leftarrow t_i + h$;
- Endif
- Endfor

The fundamental expression for the s-stage RK method 7.58 can be generalized for a non-unique interval h_k

$$y_{k+1} = y_k + h_k \sum_{i=1}^{s} b_i k_i \tag{7.72}$$

where

$$k_i = f(t_k + c_i h_k, y_k + h_k \sum_{j=1}^{s} a_{ij} k_j), \text{for} \quad i = 1, 2, \ldots, s. \tag{7.73}$$

Coefficients of the methods can be represented by the Butcher's tableau [15]

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_s & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline & b_1 & b_2 & \ldots & b_s \end{array} \tag{7.74}$$

that can be also written in the matrix form

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} \tag{7.75}$$

with $\mathbf{c} \in \mathrm{R}^s$, $\mathbf{b} \in \mathrm{R}^s$, and $\mathbf{A} \in \mathrm{R}^{s \times s}$.

The RK method can be explicit or implicit. For the implicit method the matrix $\mathbf{A}$ is full quadratic matrix and the tableau 7.74 looks as it is, but for the explicit method the tableau can be simplified because the matrix $\mathbf{A}$ is the triangular one and can be written in the following form

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \ldots & b_{s-1} & b_s \end{array} \tag{7.76}$$

The following examples serve as instructions how to operate the Butcher's tableau: ♣

•

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \mathbf{\frac{1}{2}} & \\ \hline & \mathbf{0} & \mathbf{1} \end{array} \quad \text{for} \quad y_1 = y_0 + \mathbf{0}hf(y_0) + \mathbf{1}hf(y_0 + \frac{\mathbf{1}}{\mathbf{2}}hf(y_0)) \qquad (7.77)$$

•

$$\begin{array}{c|cc} 0 & & \\ 1 & \mathbf{1} & \\ \hline & \mathbf{\frac{1}{2}} & \mathbf{\frac{1}{2}} \end{array} \quad \text{for} \quad y_1 = y_0 + \frac{\mathbf{1}}{\mathbf{2}}hf(y_0) + \frac{\mathbf{1}}{\mathbf{2}}hf(y_0 + \mathbf{1}hf(y_0)) \qquad (7.78)$$

♠

For RK methods the following sum condition is assumed

$$c_i = \sum_{j=1}^{s} a_{ij}, \text{for} \quad i = 1, 2, \ldots, s \qquad (7.79)$$

that can be also written in the matrix form

$$\mathbf{c} = \mathbf{A}\mathbf{i}$$

where $\mathbf{i}$ is a unit vector $\mathbf{i} = [1, 1, \ldots, 1]^T \in \mathrm{R}^s$. The condition ensures that all points, where the function $f(t)$ is evaluated, are first order approximations.

The RK4 method can be developed for the scalar autonomous IVP: $\dot{y} = f(t, y(t))$, $y(t_0) = y_0$, following the same procedure as for the RK2 method applied in the previous subsection. The Taylor series expended up to the fourth derivative is written as

$$y(t_{i+1}) = y(t_i) + h_i \left.\frac{dy}{dt}\right|_{t_i,y_i} + \frac{h_i^2}{2!}\left.\frac{d^2y}{dt^2}\right|_{t_i,y_i} + \frac{h_i^3}{3!}\left.\frac{d^3y}{dt^3}\right|_{t_i,y_i} + \frac{h_i^4}{4!}\left.\frac{d^4y}{dt^4}\right|_{t_i,y_i} + \ldots, \qquad (7.80)$$

and now replace derivatives of $y(t)$ by derivatives of RHS of ODE,

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + h_i f(t_i, y(t_i)) + \frac{h_i^2}{2!}\dot{f}(t_i, y(t_i)) + \\ &\quad + \frac{h_i^3}{3!}\ddot{f}(t_i, y(t_i)) + \frac{h_i^4}{4!}f^{(3)}(t_i, y(t_i)) + \ldots. \end{aligned} \qquad (7.81)$$

Assume now that $h = const.$ and introduce the following elementary differentials

$$\begin{aligned} \ddot{y} &= \frac{\partial f(y(t))}{\partial t} = \frac{\partial f}{\partial y} f = \dot{f} f, \\ y^{(3)} &= \frac{\partial^2 f}{\partial y^2}(f, f) + \frac{\partial f}{\partial y} \cdot \frac{\partial f}{\partial y} f = \ddot{f}(f, f) + \dot{f}\dot{f} f, \\ y^{(4)} &= \frac{\partial^3 f}{\partial y^3} \cdot (f, f, f) + \frac{\partial f}{\partial y} \cdot \frac{\partial f}{\partial y} \cdot \frac{\partial f}{\partial y} f + \frac{\partial f}{\partial y} \cdot \frac{\partial^2 f}{\partial y^2} \cdot (f, f) + 3\frac{\partial^2 f}{\partial y^2} \cdot (\frac{\partial f}{\partial y} \cdot f, f) \\ &= f^{(3)} \cdot (f, f, f) + \dot{f}\dot{f}\dot{f} f + \dot{f}\ddot{f}(f, f) + 3\ddot{f}(\dot{f} f, f) \end{aligned}$$

which are the Frechet's derivatives of $f(y)$. Using them in Eq. 7.81 results in

$$\begin{aligned} y_{i+1} &= y_i + hq_{11}f_i + h^2 q_{21}\dot{f}_i + h^3[q_{31}\ddot{f}_i(f,f) + q_{32}\dot{f}_i\dot{f}_i f_i] \\ &+ [q_{41}f_i^{(3)}(f,f,f) + q_{42}\dot{f}_i\ddot{f}_i(f,f) + q_{43}\ddot{f}_i(\dot{f}f,f) + q_{44}\dot{f}_i\dot{f}_i\dot{f}_i f_i] + \end{aligned} \quad (7.82)$$

where q_{ij} depend exclusively on the coefficients $\mathbf{A}, \mathbf{b}, \mathbf{c}$.

The local truncation error can be obtained by subtracting Eq. 7.82 from Eq. 7.81, i.e.

$$\begin{aligned} y(t_{k+1}) - y_{k+1} &= h(q_{11}-1)f + h^2(q_{21} - \frac{1}{2}) \cdot \frac{\partial f}{\partial y} f + \\ &+ h^3[(q_{31} - \frac{1}{6})\ddot{f}(f,f) + (q_{32} - \frac{1}{6})\dot{f}\dot{f}f] + \ldots \end{aligned} \quad (7.83)$$

The RK method is of order p when the truncation error is of the order $O(h^{p+1})$. That implies that in Eq.7.83 coefficients of h with the powers up to p must be equal to zero. The corresponding equations are called as the order equations.

Therefore, following some rather complicated operations shown in [5], the Butcher's tableau for famous RK4 is getting the well know form

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \quad (7.84)$$

It can be easily observed by verifying coefficients in Eq. 7.67 that values in the tableau are evaluated correctly.

Reevaluate now the example presented already for illustration of RK2 method by applying the RK4 method to the same problem solution

Example 3 [27] ♣

A container at temperature of 1200 K is allowed to cool down in air at an ambient temperature of approximately 300 K. Assuming that cooling is driven only by the radiation, the differential equation for the temperature of the container is given by

$$\frac{d\theta(t)}{dt} = -2.2067 \cdot 10^{-12}(\theta^4(t) - 81 \cdot 10^8)$$

where $\theta(t)$ is in K and t-time is in seconds. Find the temperature after $t = 480$ seconds since the beginning of cooling process by using the RK4 method. Apply the step size $h = 240$ seconds.

Solution

$i=0$	$t_0=0,\ \theta_0=1200$	
	$k_1=f(t_0,\theta_0)=f(0,1200)$	$=-2.2067\cdot 10^{-12}(1200^4-81\cdot 10^8)=-4.5579$
	$k_2=f(t_0+\frac{1}{2}h,\theta_0+\frac{1}{2}k_1h)$	$=f(0+\frac{1}{2}240,1200+\frac{1}{2}(-4.5579)240)=-0.3835$
	$k_3=f(t_0+\frac{1}{2}h,\theta_0+\frac{1}{2}k_2h)$	$=f(0+\frac{1}{2}240,1200+\frac{1}{2}(-0.3835)240)=-3.8954$
	$k_4=f(t_0+\frac{1}{2}h,\theta_0+k_3h)$	$=f(0+\frac{1}{2}240,1200-3.8954\cdot 240)=0.0069750$
	$\theta_1=\theta_0+\frac{1}{6}h(k_1+2k_2+2k_3+k_4)$	$=1200+[\frac{1}{6}240(-4.5579)+2(-0.3835)+$ $+2(-3.8954)+0.0069750]=675.65$ K
$i=1$	$t_1=t_0+h=240,\ \theta_1=675.65$ K	
	$k_1=f(t_1,\theta_1)$	$=-2.2067\cdot 10^{-12}(675.65^4-81\cdot 10^8)=-0.44199$
	$k_2=f(t_1+\frac{1}{2}h,\theta_1+\frac{1}{2}k_1h)$	$=f(240+\frac{1}{2}240,675.65-\frac{1}{2}0.44199\cdot 240)=-0.31372$
	$k_3=f(t_1+\frac{1}{2}h,\theta_1+\frac{1}{2}k_2h)$	$=f(240+\frac{1}{2}240,675.65-\frac{1}{2}-0.31372\cdot 240)=-0.34775$
	$k_4=f(t_1+\frac{1}{2}h,\theta_1+k_3h)$	$=f(240+\frac{1}{2}240,675.65-0.34775\cdot 240)=-0.25351$
	$\theta_2=\theta_1+\frac{1}{6}h(k_1+2k_2+2k_3+k_4)$	$=675.65+\frac{1}{6}240[-0.44199-2\cdot 0.31372$ $-2\cdot 0.34775-0.25351]=594.91$ K
$i=2$	$t_2=t_1+h=480,\ \theta_2=594.91$ K	

The effect of the step size for RK4 method is illustrated by numerical results in the following table:

Step size h	$\theta(t=480\text{s})$	E_T	$\lvert\epsilon\rvert_T\%$
480	-90.278	737.85	113.94
240	594.91	52.660	8.1319
120	646.16	1.4122	0.21807
60	647.54	0.033626	0.0051926
30	647.57	0.000869	0.00013419

♠

Example 4 [1] ♣

Solve the following BVP

$$\frac{dy}{dt}=1+y+t^2 \quad \text{with} \quad y(0)=0.5 \tag{7.85}$$

by using RK4 method to find $y(0.2)$ and $y(0.4)$.

Solution

$i=0$	$t_0=0,\ y_0=0.5,\ h=0.2$	
	$k_1=f(t_0,y_0)$	$=1+y_0+t_0^2=1.5$
	$k_2=f(t_0+\frac{1}{2}h,y_0+\frac{1}{2}k_1h)$	$=1+y_0+0.15+(t_0+0.1)^2=1.64$
	$k_3=f(t_0+\frac{1}{2}h,y_0+\frac{1}{2}k_2h)$	$=1+y_0+0.164+(t_0+0.1)^2=1.654$
	$k_4=f(t_0+\frac{1}{2}h,y_0+k_3h)$	$=1+y_0+0.16545+(t_0+0.1)^2=1.79$
	$y_1=y_0+\frac{1}{6}h(k_1+2k_2+2k_3+k_4)$	$=0.8293$
$i=1$	$t_1=t_0+h=0.2,\ y_1=0.8293$	
	$k_1=f(t_1,y_1)$	$=1.7893$
	$k_2=f(t_1+\frac{1}{2}h,y_1+\frac{1}{2}k_1h)$	$=1.9182$
	$k_3=f(t_1+\frac{1}{2}h,y_1+\frac{1}{2}k_2h)$	$=1.9311$
	$k_4=f(t_1+\frac{1}{2}h,y_1+k_3h)$	$=2.0555$
	$y_2=y_1+\frac{1}{6}h(k_1+2k_2+2k_3+k_4)$	$=1.2141$
$i=2$	$t_2=t_1+h=0.4,\ y_2=1.2141$	

♠

7.3.5 Solution of ODEs systems of the first-order

The system of ODE equations consists of two or more differential equations that share the same set of variables. Such equations can be obtained after conversion of higher order ODEs to a system of first order ODEs. The system can be solved as an initial value problem (IVP) or a boundary value problem (BVP). The number of conditions for IVP for n ODEs is equal to the number of equations.

The system of ODEs can be written in the form

$$\begin{aligned} \frac{dy_1(t)}{dt} &= f_1(t, y_1(t), y_2(t), \ldots, y_n(t)), \\ \frac{dy_2(t)}{dt} &= f_2(t, y_1(t), y_2(t), \ldots, y_n(t)), \\ &\vdots \\ \frac{dy_n(t)}{dt} &= f_n(t, y_1(t), y_2(t), \ldots, y_n(t)) \end{aligned} \tag{7.86}$$

where RHS functions: $f_1(t, y_1, y_2, \ldots, y_n)$, $f_2(t, y_1, y_2, \ldots, y_n)$,..., $f_n(t, y_1, y_2, \ldots, y_n)$ are given functions of t in the domain: $t \in [t_0, t_m]$. The initial values are given by the set of values: $y_1(t_0), y_2(t_0), \ldots, y_n(t_0)$. The domain consists of discrete equally-spaced points: $t_0, t_1, \ldots, t_m$, which are nodes of subintervals with length of $h = \triangle t$.

The solutions to an initial-value problem defined for a system of ODEs can be derived from the same methods used in the single ODE case. Therefore, the RK and Euler methods can be applied to solve IVP for the system of ODEs.

Consider two ODEs having two variables each. The IVP for the system can be written as

$$\begin{aligned} \frac{dy(t)}{dt} &= f(t, y(t), z(t)), \\ \frac{dz(t)}{dt} &= g(t, y(t), z(t)), \end{aligned}$$

with initial values

$$\begin{aligned} y(t_0) &= y_0, \\ z(t_0) &= z_0. \end{aligned}$$

The discrete values of t_i are given for $i = 0, 1, 2, \ldots, n$ and assuming the uniform intervals with $h = \triangle t$, the steps in t can be find by the formula

$$t_{i+1} = t_i + h = t_0 + ih.$$

The RK4 for the system Eq. 7.87 is expressed by

$$\begin{aligned}
k_1 &= hf(t_n, y_n, z_n), \\
l_1 &= hg(t_n, y_n, z_n), \\
k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}) \\
l_2 &= hg(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}) \\
k_3 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}) \\
l_3 &= hg(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}) \\
k_4 &= hf(x_n + h, y_n + k_3, z_n + l_3), \\
l_4 &= hg(x_n + h, y_n + k_3, z_n + l_3), \\
k &= \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
l &= \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4), \\
t_{n+1} &= t_n + h, \\
y_{n+1} &= y_n + k, \\
z_{n+1} &= z_n + l.
\end{aligned}$$

The algorithm the RK4 method applied for a system of two ODEs with three variables can be expressed as follows

- Given a system

$$\begin{aligned}
\frac{dy(t)}{dt} &= f(t, y(t), z(t)), \\
\frac{dz(t)}{dt} &= g(t, y(t), z(t)),
\end{aligned} \tag{7.87}$$

- Given $h = \triangle t$ and $m = i_{max}$,
- Given the initial values $y_0 = y(t_0)$ and $z_0 = z(t_0)$
- for $i = 0$ to m
- Evaluate $k_1 = hf(t_n, y_n, z_n)$,
- Evaluate $l_1 = hg(t_n, y_n, z_n)$,
- Evaluate $k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2})$
- Evaluate $l_2 = hg(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2})$

- Evaluate $k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2})$
- Evaluate $l_3 = hg(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2})$
- Evaluate $k_4 = hf(t_n + h, y_n + k_3, z_n + l_3)$,
- Evaluate $l_4 = hg(t_n + h, y_n + k_3, z_n + l_3)$,
- If $i < m$
- Compute $k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$,
- Compute $l = \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4)$,
- Compute $y_{n+1} = y_n + k$,
- Compute $z_{n+1} = z_n + l$,
- Update $t_{i+1} \leftarrow t_i + h$,
- Endif
- Endfor

The above algorithm is applied to solve the following example

Example 5 [41] ♣

Using the RK4 method, solve the IVP for two differential equations

$$\begin{aligned} \frac{dy}{dt} &= y - 2zt \\ \frac{dz}{dt} &= tyz \end{aligned}$$

with the initial values given by $t_0 = 0$, $y_0 = 2$, and $z_0 = -1$. Assume $h = \triangle t = 0.1$. Calculate three first steps only.

Solution

$$\begin{aligned} k_1 &= hy_0 - 2z_0t_0 = 0.2, \\ l_1 &= ht_0y_0z_0 = 0, \\ k_2 &= h(y_0 + \frac{k_1}{2}) - 2(z_0 + \frac{l_1}{2})(t_0 + \frac{h}{2}) = 0.220 \\ l_2 &= h(t_0 + \frac{h}{2})(y_0 + \frac{k_1}{2})(z_0 + \frac{l_1}{2}) = -0.0105 \\ k_3 &= h(y_0 + \frac{k_2}{2}) - 2(z_0 + \frac{l_2}{2})(t_0 + \frac{h}{2}) = 0.221053 \\ l_3 &= h(t_0 + \frac{h}{2})(y_0 + \frac{k_2}{2})(z_0 + \frac{l_2}{2}) = -0.010606 \\ k_4 &= h(y_0 + k_3) - 2(z_0 + l_3)(t_0 + h) = 0.242317 \\ l_4 &= h(t_0 + h)(y_0 + k_3)(z_0 + l_3) = -0.022446 \\ k &= \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = \frac{1}{6}(0.2 + 0.44 + 0.44106 + 0.242317) = 0.220737 \\ l &= \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4) = \frac{1}{6}(-2 \cdot 0.0105 - 2 \cdot 0.010605 - 0.022446) = -0.010776 \end{aligned}$$

$$
\begin{aligned}
y_1 &= y(0.1) = y_0 + k = 2 + 0.220737 = 2.220737 \\
z_1 &= z(0.1) = z_0 + l = -1 - 0.010776 = -1.010776
\end{aligned}
$$

Results for four steps of the RK4 method for this example are shown in the following table:

i	k_1	l_1	k_2	l_2	k_3	l_3	k_4	l_4
0	0.200000	0.000000	0.220000	-0.010500	0.221053	-0.010605	0.242317	-0.022446
1	0.242289	-0.022447	0.264848	-0.035901	0.266178	-0.036311	0.299575	-0.052080
2	0.242289	-0.022447	0.264848	-0.035901	0.266178	-0.036311	0.290575	-0.052080
3	0.242289	-0.022447	0.264848	-0.035901	0.266178	-0.036311	0.290575	-0.052080

Note that values of k_i and l_i converge very fast as can be seen from the second step.

♠

The RK4 method is not restricted to solve only the first order ODEs given in the form of Eqs.7.86 because it is well known that the higher order ODE can be transferred to a system of the first order ODEs. The procedure conversion procedure of a higher order ODE to the system of first order ODE consists of the following steps:

Action	*What to do*	*Comment*
Select dependent variables	Take the original dependent variable and its derivatives up to one degree less than the highest order derivative	Any n-th order ODE is convertible to a system of n first order ODEs.
Write differential equations in terms of the new variables	The new equations can be derived from the original equation or they come from the definitions of the new variables.	There are infinite number of ways for selection of the new variables. Therefore, there are infinite number of ODE systems equivalent to the higher order ODE.
Express the system of ODEs in the matrix form		Use a table to make conversion easier.

Example 6 ♣

Convert the second order ODE: $\ddot{y}(t) + 4\dot{y}(t) + 8y(t) = 2$ with initial conditions: $\dot{y}(0) = 1$ and $y(0) = 3$ to a system of the first order ODEs.

Solution

Select the new set of functions:

$$
\begin{aligned}
z_1(t) &= y(t), \\
z_2(t) &= \dot{y}(t).
\end{aligned}
$$

Write a table

Old function	*New function*	*Init. condition*	*New equation*
$y(t)$	$z_1(t)$	$z_1(0) = 3$	$\dot{z}_1 = z_2$
$\dot{y}(t)$	$z_2(t)$	$z_2(0) = 1$	$\dot{z}_2 = 2 - 4z_2 - 8z_1$

The matrix form of the new system is

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} z_2 \\ 1 - 4z_2 - 8z_1 \end{bmatrix}, \quad \mathbf{z}(0) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

♠

Example 7 ♣

Convert the third order ODE: $y^{(3)}(t) + 3\ddot{y}(t) + 6\dot{y}(t) + 9y(t) = 1$ with initial conditions: $\ddot{y}(0) = 8$, $\dot{y}(0) = 4$ and $y(0) = 2$ to a system of the first order ODEs.

Solution

Select the new set of functions:

$$\begin{aligned} z_1(t) &= y(t), \\ z_2(t) &= \dot{y}(t), \\ z_3(t) &= \ddot{y}(t). \end{aligned}$$

Write a table

Old function	*New function*	*Init. condition*	*New equation*
$y(t)$	$z_1(t)$	$z_1(0) = 2$	$\dot{z}_1 = z_2$
$\dot{y}(t)$	$z_2(t)$	$z_2(0) = 4$	$\dot{z}_2 = z_3$
$\ddot{y}(t)$	$z_3(t)$	$z_3(0) = 8$	$\dot{z}_3 = 1 - 3z_3 - 6z_2 - 9z_1$

The matrix form of the new system is

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 1 - 3z_3 - 6z_2 - 9z_1 \end{bmatrix}, \quad \mathbf{z}(0) = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

♠

Example 8 ♣

Convert the set of two ODEs consisting of the third order ODE and the second order ODE

$$y^{(3)}(t) + 3\ddot{y}(t) + 4\dot{y}(t) + 5x(t) = 1, \tag{7.88}$$

$$\ddot{x}(t) + 5y(t)x(t) + y(t) = 2 \tag{7.89}$$

with initial conditions: $\ddot{y}(0) = 7$, $\dot{y}(0) = 1$, $y(0) = 3$ and $x(0) = 2$, $\dot{x}(0) = -4$ to a system of higher order ODEs.

Solution

Select the new set of functions:

$$\begin{aligned} z_1(t) &= y(t), \\ z_2(t) &= \dot{y}(t), \\ z_3(t) &= \ddot{y}(t), \\ z_4(t) &= x(t), \\ z_5(t) &= \dot{x}(t). \end{aligned}$$

Write a table

Old function	*New function*	*Init. condition*	*New equation*
$y(t)$	$z_1(t)$	$z_1(0) = 3$	$\dot{z}_1 = z_2$
$\dot{y}(t)$	$z_2(t)$	$z_2(0) = 1$	$\dot{z}_2 = z_3$
$\ddot{y}(t)$	$z_3(t)$	$z_3(0) = 7$	$\dot{z}_3 = 1 - 3z_3 - 4z_2 - 5z_4$
$x(t)$	$z_4(t)$	$z_4(0) = 2$	$\dot{z}_4 = z_5$
$\dot{x}(t)$	$z_5(t)$	$z_5(0) = -4$	$\dot{z}_5 = 2 - z_2 - 5z_1 z_4$

The matrix form of the new system is

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \\ \dot{z}_5 \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ 1 - 3z_3 - 4z_2 - 5z_4 \\ z_5 \\ 2 - z_2 - 5z_1 z_4 \end{bmatrix}, \quad \mathbf{z}(0) = \begin{bmatrix} 3 \\ 1 \\ 7 \\ 2 \\ -4 \end{bmatrix}$$

♠

Applying RK4 method to a higher order ODE, the transformation from a higher order ODE to the system of the first order ODE may not be necessary as the RK4 direct solution can be used as is displayed in the following example

$$\begin{aligned} \ddot{y} &= f(t, y, \dot{y}), \\ k_1 &= h^2 \cdot f(t, y, \dot{y}), \\ k_2 &= \frac{h^2}{2} \cdot f(t + \frac{h}{2}, y + \frac{h}{2}\dot{y} + \frac{1}{4}k_1, \dot{y} + \frac{1}{h}k_1), \\ k_3 &= \frac{h^2}{2} \cdot f(t + \frac{h}{2}, y + \frac{h}{2}\dot{y} + \frac{1}{4}k_2, \dot{y} + \frac{1}{h}k_2), \\ k_4 &= \frac{h^2}{2} \cdot f(t + h, y + h\dot{y} + k_3, \dot{y} + \frac{2}{h}k_3). \\ \triangle y &= \frac{1}{3}(k_1 + k_2 + k_3), \\ \triangle \dot{y} &= \frac{1}{3h}(k_1 + k_2 + k_3 + k_4) \end{aligned} \tag{7.90}$$

The next values of y and $\dot{y}$ are defined as

$$\begin{aligned} y(t+h) &= y(t) + h\dot{y}(t) + \triangle y, \\ \dot{y}(t+h) &= \dot{y} + \triangle \dot{y}. \end{aligned} \tag{7.91}$$

The Euler method can be also applied to solve a higher order ODE. This is illustrated by the following example:

Example 9 ♣

Apply the Euler method to solve the IVP for the second order ODE

$$\ddot{y}(t) + 3\dot{y}(t) + 6y(t) = 2$$

with IV conditions

$$\begin{aligned} y(0) &= 1 \\ \dot{y}(0) &= -2 \end{aligned}$$

Solution Select the new set of functions

$$\begin{aligned} z_1(t) &= y(t) \\ z_2(t) &= \dot{y}(t) \end{aligned}$$

Convert the second order ODE to a set of two first order ODEs and represent them in the matrix form

$$\dot{\mathbf{z}} = \mathbf{f}(\mathbf{z}) = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} z_2 \\ 2 - 3z_2 - 6z_1 \end{bmatrix}, \quad \mathbf{z}(0) = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Assuming $h = 0.1$, calculate two steps of the Taylor method:

$$\dot{\mathbf{z}}(0.1) = \mathbf{z}(0) + h\mathbf{f}(\mathbf{z}(0)) = \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 0.1 \begin{bmatrix} -2 \\ 2 - 3(-2) - 6(1) \end{bmatrix} = \begin{bmatrix} 0.8 \\ -1.8 \end{bmatrix}$$

$$\dot{\mathbf{z}}(0.2) = \mathbf{z}(0.1) + h\mathbf{f}(\mathbf{z}(0.1)) = \begin{bmatrix} 0.8 \\ -1.8 \end{bmatrix} + 0.1 \begin{bmatrix} -1.8 \\ 2 + 3(1.8) - 6(0.8) \end{bmatrix} = \begin{bmatrix} 0.62 \\ -1.58 \end{bmatrix}$$

♠

7.3.6 Solution of ODE by predictor-corrector multi-step methods

The best understanding of multi-step methods of ODE integration can be built on the idea of quadrature formula that is related to the subject of numerical integration. This is why the following subsection appears in the section devoted to a solution of ODEs.

Quadrature and Newton-Cotes integration formulas

The solution to the IVP given by

$$\dot{y} = f(t, y(t)), \quad t \in [a, b], \quad \text{where} \quad a \leq t \leq b, \quad y(a) = \alpha \tag{7.92}$$

has a property that

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} \dot{y}(t)dt = \int_{t_k}^{t_{k+1}} f(t, y(t))dt. \tag{7.93}$$

where the partition $[t_k, t_{k+1}] \in [a, b]$.

Now the goal is to approximate the definite integral of $f(t, y(t))$ over the interval $[a, b]$ by evaluating $f(t, y(t))$ at a finite number of sample points.

♣ ***Quadrature definition***

Abbreviating $f(t, y(t))$ to the form of $f(t)$ and assuming that $a = t_0 < t_1 < t_2 < \ldots < t_n = b$, the following formula

$$Q_f = \sum_{k=0}^{n} w_k f(t_k) = w_0 f(t_0) + w_1 f(t_1) + \ldots + w_n f(t_n) \tag{7.94}$$

is called a numerical integral or ***the quadrature formula*** subject that it fulfills the condition

$$\int_a^b f(t)dt = Q_f + E_f,$$

where: $t_0, t_1, \ldots, t_n$ are quadrature nodes, $w_0, w_1, \ldots, w_n$ are weights, and E_f is the truncation error. ♠

The truncation error E_f can be anticipated replacing $f(t)$ by a polynomial. Any polynomial can be selected. Therefore consider the one of i-th degree:

$$p_i(t) = a_0 + a_1 t + \ldots + a_{i-1} t_{i-1} + a_i t_i$$

with the truncation error

$$E_f = K f^{(n+1)} c,$$

where K is chosen constant and n is the degree of precision.

The quadrature nodes can be chosen in various manner that will be considered in the next chapter when the approximate integration techniques will be discussed.

The function $f(t, y(t))$ cannot be integrated unless $y(t)$ is unknown. For this purpose, $f(t, y(t))$ can be replaced by the interpolating polynomial $p(t)$ that could be tailored for $f(t, y(t))$ by utilizing previously calculated points: $(t_0, w_0), \ldots, (t_n, w_n)$.

Adams-Bashforth-Moulton Method

For the derivation of an Adam-Bashforth explicit n-step method, the fundamental thing is to construct the backward-difference polynomial $p_{n-1}(t)$ fitting a set of points: $\{(t_i, f_i), (t_{i-1}, f_{i-1}), \ldots, (t_{i+1-n}, f_{i+1-n})\}$ where $f_i = f(t_i, y(t_i))$. Therefore, the RHS of ODE is approximated as follows

$$f(t, y(t)) = p_{n-1}(t) + \frac{f^{(n)}(\tau_i, y(\tau_i))}{n!}(t - t_i)(t - t_{i-1}) \ldots (t - t_{i+1-n}) \quad (7.95)$$

where $\tau_i \in (t_{i+1-n}, t_i)$. Expressing t as $t = t_i + sh$ and then $dt = hds$ and integrating both sides of Eq.7.95 results in

$$\begin{aligned}
\int_{t_i}^{t_{i+1}} f(t, y(t))dt &= \int_{t_i}^{t_{i+1}} \left[\sum_{k=0}^{n-1} (-1)^k \begin{pmatrix} -s \\ k \end{pmatrix} f_i^{(k)}\right] dt \\
&+ \int_{t_i}^{t_{i+1}} \frac{f^{(n)}(\tau_i, y(\tau_i))}{n!}(t - t_i)(t - t_{i-1}) \ldots (t - t_{i+1-n})dt \\
&= \sum_{k=0}^{n-1} f_i^{(k)} h(-1)^k \int_0^1 \begin{pmatrix} -s \\ k \end{pmatrix} ds \\
&+ \frac{h^{n+1}}{n!} \int_0^1 s(s+1) \ldots (s+n-1) f^{(n)}(\tau_i, y(\tau_i)) ds. \qquad (7.96)
\end{aligned}$$

Where

$$\begin{pmatrix} -s \\ k \end{pmatrix}$$

is not a binomial coefficients and integrals

$$\mathbf{I}_k^{-s} = (-1)^k \int_0^1 \begin{pmatrix} -s \\ k \end{pmatrix} ds$$

can be evaluated quite easy following the example for $k = 3$.

Example 10 ♣

$$\begin{aligned}
(-1)^3 \int_0^1 \begin{pmatrix} -s \\ 3 \end{pmatrix} ds &= -\int_0^1 \frac{(-3)(-s-1)(-s-2)}{1 \cdot 2 \cdot 3} ds = \frac{1}{6} \int_0^1 (s^3 + 3s^2 + 2s) ds \\
&= \frac{1}{6} \left[\frac{s^4}{4} + s^3 + s^2\right]_0^1 = \frac{1}{6} \cdot \frac{9}{4} = \frac{3}{8}.
\end{aligned}$$

♠

The table of $\mathbf{I}_k^{-s}$ with integrals is given below.

k	0	1	2	3	4	5
$\mathbf{I}_k^{-s}$	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$

Using the data from the table, Eq.7.96 can be rewritten as

$$\begin{aligned}\int_{t_i}^{t_{i+1}} f(t,y(t))dt &= h\left[f_i+\frac{1}{2}\dot{f}_i+\frac{5}{12}\ddot{f}_i+\ldots\right] \\ &+ \frac{h^{n+1}}{n!}\int_0^1 s(s+1)\cdots(s+n-1)f^{(n)}(\tau_i,y(\tau_i))ds. \end{aligned} \tag{7.97}$$

From the observation that the product $s(s+1)\cdots(s+n-1)$ does not change a sign for $s\in[0,1]$, it appears that for some number μ_i such that $t_{i-m+1}<\mu_i<t_{i+1}$, the error term in eq.7.97 can be expressed by

$$\frac{h^{n+1}}{n!}\int_0^1 s(s+1)\cdots(s+n-1)f^{(n)}(\tau_i,y(\tau_i))ds = h^{n+1}f^{(n)}(\mu_i,y(\mu_i))\int_0^1 \begin{pmatrix} -s \\ k \end{pmatrix} ds \tag{7.98}$$

Substituting RHS of Eqs 7.97 and 7.98 in to Eq.7.93 gives the final form of the Newton-Cotes integration formula

$$y(t_{i+1}) = y(t_i)+h\left[f_i+\frac{1}{2}\dot{f}_i+\frac{5}{12}\ddot{f}_i+\cdots\right]+h^{n+1}f^{(n)}(\mu_i,y(\mu_i))(-1)^n\int_0^1\begin{pmatrix} -s \\ k \end{pmatrix} ds \tag{7.99}$$

From the above above formula the three-steps Adams-Bashforth technique can be derived assuming $n=3$ and neglecting the error term, i.e.

$$\begin{aligned} y(t_{i+1}) &\approx y(t_i)+h\left[f_i+\frac{1}{2}\dot{f}_i+\frac{5}{12}\ddot{f}_i\right] \\ &= y(t_i)+h\left\{f_i+\frac{1}{2}\left(f_i-f_{i-1}\right)+\frac{5}{12}\left[f_i-2f_{i-1}+f_{i-2}\right]\right\} \\ &= y(t_i)+\frac{h}{12}\left(23f_i-16f_{i-1}+5f_{i-2}\right) \end{aligned} \tag{7.100}$$

The last part of Eq.7.100 can be rewritten in a different form

$$y(t_{i+1})=y(t_i)+\frac{h}{12}\left[23f(t_i,y(t_i))-16f(t_{i-1},y(t_{i-1}))+5f(t_{i-2},y(t_{i-2}))\right] \tag{7.101}$$

which is very useful for the derivation of explicit Adams-Bashforth (AB) and implicit Adams-Moulton (AM) methods. The coresponding explicit and implicit schemes together with the local truncation error (LTE) are given in the following table:

Method	Equations
AB two-step ***explicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1$ $w_{i+1} = w_i + \frac{h}{2}\left[3f(t_i, w_i) - f(t_{i-1}, w_{i-1})\right]$ $i = 1, 2, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{5}{12}y^{(3)}(\mu_i)h^2$ for $\mu_i \in (t_{i-1}, t_{i+1})$
AB three-step ***explicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2$ $w_{i+1} = w_i + \frac{h}{12}\left[23f(t_i, w_i) - 16f(t_{i-1}, w_{i-1}) + 5f(t_{i-2}, w_{i-2})\right]$ $i = 2, 3, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{3}{8}y^{(4)}(\mu_i)h^3$ for $\mu_i \in (t_{i-2}, t_{i+1})$
AB four-step ***explicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2,\ w_3 = \alpha_3$ $w_{i+1} = w_i + \frac{h}{24}\left[55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3})\right]$ $i = 3, 4, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{251}{720}y^{(5)}(\mu_i)h^4$ for $\mu_i \in (t_{i-3}, t_{i+1})$
AB five-step ***explicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2,\ w_3 = \alpha_3,\ w_4 = \alpha_4$ $w_{i+1} = w_i + \frac{h}{720}\left[1901f(t_i, w_i) - 2774f(t_{i-1}, w_{i-1}) + 2616f(t_{i-2}, w_{i-2})\right.$ $\left.-1274f(t_{i-3}, w_{i-3}) + 251f(t_{i-4}, w_{i-4})\right]$ where $i = 4, 5, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{95}{288}y^{(6)}(\mu_i)h^5$ for $\mu_i \in (t_{i-4}, t_{i+1})$
AM two-step ***implicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1$ $w_{i+1} = w_i + \frac{h}{12}\left[5f(t_{i+1}, w_{i+1}) + 8f(t_i, w_i) - f(t_{i-1}, w_{i-1})\right]$ $i = 1, 2, \ldots, N-1$ LTE $= \tau_{i+1}(h) = -\frac{1}{24}y^{(4)}(\mu_i)h^3$ for $\mu_i \in (t_{i-1}, t_{i+1})$
AM three-step ***implicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2$ $w_{i+1} = w_i + \frac{h}{24}\left[9f(t_{i+1}, w_{i+1}) + 19f(t_i, w_1) - 5f(t_{i-1}, w_{i-1}) + f(t_{i-2}, w_{i-2})\right]$ $i = 2, 3, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{19}{720}y^{(5)}(\mu_i)h^4$ for $\mu_i \in (t_{i-2}, t_{i+1})$
AM four-step ***explicit*** method	$w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2,\ w_3 = \alpha_3$ $w_{i+1} = w_i + \frac{h}{720}\left[251f(t_{i+1}, w_{i+1}) + 646f(t_i, w_i) - 264f(t_{i-1}, w_{i-1})\right.$ $\left.+106f(t_{i-2}, w_{i-2}) - 19f(t_{i-3}, w_{i-3})\right]$ where $i = 3, 4, \ldots, N-1$ LTE $= \tau_{i+1}(h) = \frac{3}{160}y^{(6)}(\mu_i)h^5$ for $\mu_i \in (t_{i-3}, t_{i+1})$

Example 11 ♡

Solve the following IVP

$$\dot{y} = y - t^2 \quad \text{with the initial condition} \quad y(0) = 1.$$

by using the Adams-Bashforth four-step method. Apply RK4 method for generation of starting points w_i. Assume the step size $h = 0.1$. The starting points evaluated from the RK4 are the following: $w_0 = 1$, $w_1 = 1.1048$, $w_2 = 1.2186$, $w_3 = 1.3402$. Taking the AB scheme from the third row of above table

$$w_{i+1} = w_i + \frac{h}{24}\left[55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3})\right]$$

Assuming that our starting point t_0 is chosen at the point with the smallest subscript, i.e. t_{i-3}, the corresponding scheme can be written as

$$w_4 = w_3 + \frac{h}{24}\left[55f(t_3, w_3) - 59f(t_2, w_2) + 37f(t_1, w_1) - 9f(t_0, w_0)\right].$$

Values $f(t_i, w_i)$ are the following

$f(t_i, w_i)$	Value
$f(0; 1)$	1.0000
$f(0.1; 1.1048)$	1.0948
$f(0.2; 1.2186)$	1.1786
$f(0.3; 1.3401)$	1.2501

Substituting the above values to the AB formula gives

$$w_4 = 1.3402 + \frac{0.1}{24}\left[55 \cdot 1.2501 - 59 \cdot 1.1786 + 37 \cdot 1.0948 - 9 \cdot 1\right] = 1.3401 + 0.1280 = 1.4681$$

♠

The implicit methods are more stable and have smaller roundoff errors because the coefficients of terms involving the function $f(t_i, w_i)$ in the expressions defining the local truncation error are smaller than for the explicit methods.

Implicit multi-step methods are used to improve results obtained form explicit methods. Such combination of methods is called a ***predictor-corrector method*** where an explicit method predicts an approximation and an implicit method corrects this prediction.

The effective algorithm for the predictor-corrector method, where three methods are used:

- Adams-Bashforth four-step explicit method as predictor,
- Adams-Moulton three-step implicit method as corrector,
- Runge-Kutta fourth order method as the generator of starting values w_i,

is written below for the solution of the general IVP formulated for the first order ODE.

Approximate the solution of the initial-value problem:

$$\dot{y} = f(t, y) \quad \text{for} \quad t \in [a, b] \quad \text{with the initial condition} \quad y(a) = \alpha_0.$$

by using the Adams fourth-order predictor-corrector method.

Formulas for the Adams-Moulton predictor-corrector methods are shown in the following table:

Order	Predictor-corrector formula	Local error
2	$w^p_{i+1} = w_i + \frac{h}{2}\left[3f(t_i, w_i) + f(t_{i-1}, w_{i-1})\right]$ $w^c_{i+1} = w_i + \frac{h}{2}\left[f(t_{i+1}, w^p_{i+1}) + f(t_i, w_i)\right]$	$\lvert e_{i+1}\rvert \approx \frac{1}{6}\lvert w^p_{i+1} - w^c_{i+1}\rvert$
3	$w^p_{i+1} = w_i + \frac{h}{12}\left[23f(t_i, w_i) - 16f(t_{i-1}, w_{i-1}) + 5f(t_{i-2}, w_{i-2})\right]$ $w^c_{i+1} = w_i + \frac{h}{12}\left[5f(t_{i+1}, w^p_{i+1}) + 8f(t_i, w_i) - f(t_{i-1}, w_{i-1})\right]$	$\lvert e_{i+1}\rvert \approx \frac{1}{10}\lvert w^p_{i+1} - w^c_{i+1}\rvert$
4	$w^p_{i+1} = w_i + \frac{h}{24}\left[55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2})\right.$ $\left. -9f(t_{i-3}, w_{i-3})\right]$ $w^c_{i+1} = w_i + \frac{h}{24}\left[9f(t_{i+1}, w^p_{i+1}) + 19f(t_i, w_i) - 5f(t_{i-1}, w_{i-1})\right.$ $\left. +f(t_{i-2}, w_{i-2})\right]$	$\lvert e_{i+1}\rvert \approx \frac{19}{270}\lvert w^p_{i+1} - w^c_{i+1}\rvert$

The algorithm:

- Calculate h and divide the interval $[a, b]$ into $(N+1)$ equally spaced numbers,
- **For** $i = 0, 1, 2$ **do**
  - $k_1 = hf(t_i, w_i)$,

- $k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{k_1}{2})$,
- $k_3 = hf(t_i + \frac{h}{2}, w_i + \frac{k_2}{2})$,
- $k_4 = hf(t_i + h, w_i + k_3)$,
- evaluate starting points w_i and t_i
  - $w_{i+1} = w_i + \frac{1}{6}(k_1 + k_2 + k_3 + k_4)$,
  - $t_{i+1} = a + (i+1)h$.

- **End do**
- **For** $i = 3, 4, \ldots, N-1$ **do**
  - Calculate $t^p = a + (i+1)h$,
  - **Iterate** with k
    - calculate the initial guess (predictor)
      $$w^{p\{k\}} = w_i^{\{k\}} + \frac{h}{24}\left[55f(t_i, w_i^{\{k\}}) - 59f(t_{i-1}, w_{i-1}^{\{k\}}) + 37f(t_{i-2}, w_{i-2}^{\{k\}}) - 9f(t_{i-3}, w_{i-3}^{\{k\}})\right]$$
    - correct the value $w_{i+1}^{c\{k\}}$
      $$w_{i+1}^{\{k\}} = w_i^{\{k\}} + \frac{h}{24}\left[9f(t_{i+1}, w^{p\{k\}}) + 19f(t_i, w_1^{\{k\}}) - 5f(t_{i-1}, w_{i-1}^{\{k\}}) + f(t_{i-2}, w_{i-2}^{\{k\}})\right]$$
  - **Stop iterate** when $|w_{i+1}^{p\{k\}} - w_{i+1}^{c\{k\}}| \leq \xi$ or $k =$ IterMax,

- **End do**

where ξ and IterMax, which means the maximum number of iterations, are given by the user arbitrary numbers, the superscript in curly brackets,$\{k\}$, is the iteration number, and p and c stands for "prediction" and "correction", correspondingly.

Example 12$\heartsuit$

Solve IVP for ODE

$$\frac{dy}{dt} = y - t^2,$$

with the initial condition $w_0 = y(0) = 1$. Assume the step size $h = 0.1$. Use the Adams-Moulton predictor-corrector method. Three starting points can be evaluated form the RK4: $w_0 = 1.0000$, $w_1 = 1.1048$, $w_2 = 1.2186$. Choosing the row two from the last table

$$w_{i+1}^p = w_i + \frac{h}{12}\left[23f(t_i, w_i) - 16f(t_{i-1}, w_{i-1}) + 5f(t_{i-2}, w_{i-2})\right]$$

and applying t_0 at the point with the smallest subscript, i.e. t_{i-2}, the third-order Adams-Bashforth scheme can be rewritten as

$$\begin{aligned} w_3^p &= w_2 + \frac{0.1}{12}\left[23f(t_2, w_2) - 16f(t_1, w_1) + 5f(t_0, w_0)\right] \\ &= 1.1786 + \frac{0.1}{12}\left[23 \cdot 1.1786 - 16 \cdot 1.0948 + 5 \cdot 1\right] \\ &= 1.2186 + 0.1216 = 1.3402 \end{aligned}$$

The implicit Adams-Moulton two-step scheme gives

$$w_3^c = 1.2186 + \frac{0.1}{12}\left[5 \cdot 1.2502 + 8 \cdot 1.1786 - 1 \cdot 1.0948\right] = 1.2186 + 0.1215 = 1.3401.$$

♠

Chapter 8

Solution methods for boundary value problems for ODEs

Usually, in physical problems, the initial conditions for derivatives of certain order are not know, but instead of that the initial and final values for the unknown or derivatives of some order can be known. Such type of problems are called as the boundary-value problems. this occurs more frequently with partial differential equations.

The difference between the IVP and BVP is depicted in the following table

Initial Value Problem (IVP)	Boundary Value Problem (BVP)
The initial conditions are given at the same point of the independent variable	The boundary conditions are given at two different points of the independent variable
Example of IVP $a\ddot{y}(t) + b\dot{y}(t) + cy(t) = d(t)$ $y(t_0) = \alpha$ and $\dot{y}(t_0) = \beta$	Example of BVP $a\ddot{y}(t) + b\dot{y}(t) + cy(t) = d(t)$ $y(t_l) = \alpha$ and $y(t_r) = \gamma$

In the case of IVP with given $y(t_0)$ and $\dot{t_0}$, there is a theorem of existence and uniqueness of the solution. But in the case of IVP the existence and uniqueness theorem is not available. Therefore, the BVP may have no solutions or may have non-unique solutions even when the problem is of the order N and N conditions are available.

Types of boundary conditions

Types of boundary conditions are described in the following table:

Name of condition	Value specified	Examples
Dirichlet condition	The value of $y(t)$ is specified at two particular locations of the independent variable	$y(0) = 2$ and $y(a) = 3$
Neumann condition	The derivative of $y(t)$ is specified at two points	$\dot{y}(0) = 5$ and $\dot{y}(b) = 7$
Mixed condition	Both, the value of $y(t)$ and its derivative $\dot{y}(t)$ are specified	$\dot{y}(c) = 2$ and $y(a) = -3$

8.1 Numerical solution of BVP by the shooting method

One of the most popular method for the solution of BVP is the shooting method which is based on the following idea:

1. transform the ODE of BVP to a system of ODE equations,
2. guess the initial conditions that are not available,
3. solve IVP,
4. check the known boundary conditions,
5. if necessary modify the guess and solve problem again and jump to the third point.

Example 13♡

Solve BVP for the second order ODE

$$\ddot{z}(t) - 4z(t) + 4t = 0$$

with the boundary conditions $z(0) = 0$ and $z(1) = 2$. The problem can be converted to a system of the first order ODEs

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} z_2 \\ 4(z_2 - t) \end{bmatrix}, \quad \begin{bmatrix} z_1(0) \\ z_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ ? \end{bmatrix}$$

and solved by the RK2 method with $h = 0.01$ for various values of $z_2(0)$ until the function $z(t)$ at $t = 1$ will satisfy the condition $z(1) = 2$. The following table (according to [1]) shows the guesses and results

guess	$\dot{z}(0)$	$z(1)$
1	0	-0.7688
2	1	0.9900

This table gives coordinates of two points in the space of $z(0)$ and $\dot{z}(1)$ and it is easy to fit a straight line through these points (see Fig. Al Amer Unit 7 Lesson 8 page 19). The third guess is less intuitive because of the simple geometry: choosing

the ordinate $z(1) = 2$, which is the solution to the BVP, correspond to abscissa $\dot{z}(0) = 1.5743$, which is the third and the final best guess.

♠

The shooting method can be ineffective for higher order BVP when boundary conditions for more than one variable should be invented.

8.2 Numerical solution of BVP by finite-difference method

Consider the second order ODE

$$\ddot{y}(t) + \dot{y}(t) = 0 \tag{8.1}$$

for $y(t)$, where $t \in [0, 1]$, with Neumann boundary conditions: $y(0) = 0$ and $\dot{y}(1) = -1$. The procedure to solve this BVP with the finite difference method consists the following steps:

- discretize the domain $t \in [0, 1]$,
- approximate ODE by the second order accurate finite difference scheme,
- approximate the boundary conditions,
- embed boundary conditions,
- set up the system of linear algebraic equations, called shortly: the linear system,
- solve the linear system.

These steps are described below in detail.

Discretization of the domain

The domain $t \in [0, 1]$ must be staggered about the boundary and therefore, for the Neumann's conditions it should be split into $(n-2)$ subintervals by n discrete points with the step $h = \frac{1}{n-2}$. Then the discretization of t is such that $t_i = h(i - \frac{3}{2})$.

For the Dirichlet conditions it should be split into $(n-1)$ subintervals with the step $h = \frac{1}{n-1}$.

Approximation of ODE

The Eq.8.1 can be approximated by the finite-differences and express in the form

$$\frac{1}{h}(y_{i-1} - 2y_i + y_{i+1} - y_i) = 0 \tag{8.2}$$

that can be also rewritten in the form of polynomial

$$a_i y_{i-1} + b_i y_i + c_i y_{i+1} = d_i \tag{8.3}$$

where coefficients are defined as follows: $a_i = c_i = \frac{1}{h^2}$, $b_i = -(1 + \frac{2}{h^2})$, and $d_i = 0$. The error term is neglected but we keep in mind that the scheme is the second order accurate in h. In addition it is assumed that the polynomial coefficients are not constant and $d_i \neq 0$. The Eq.8.3 is valid only for $i \in [2, \ldots, n-1]$ as the discrete value of the derivative is not defined at extreme points of a grid, i.e. $i = 1$ and $i = n$.

Approximation of boundary conditions

Following the approximation of ODE, the boundary conditions must be also approximated. Because the grid is staggered, the values of y_1 and y_2 are given at $t_1 = -\frac{h}{2}$ and $t_2 = \frac{h}{2}$ and therefore, the value y at $t = 0$ can be defined by interpolation between y_1 and y_2. Therefore, using the central interpolation at the point $t_{1+\frac{1}{2}}$

$$y_{1+\frac{1}{2}} = \frac{y_1 + y_2}{2} + \mathcal{O}(h^2) = 0$$

and solving for y_1 and neglecting the error of the finite-difference scheme, we obtain

$$y_1 = -y_2.$$

The boundary condition at $t = 1$ is approximated by the second-order accurate difference scheme

$$\frac{dy}{dt}|_{i=n-\frac{1}{2}} = \frac{y_n - y_{n-1}}{h} + \mathcal{O}(h^2) = -1.$$

taken at $t_{i=n-\frac{1}{2}}$. From where, neglecting the discretization error, we can obtain

$$y_n = y_{n-1} - h.$$

Embedding of boundary conditions

The approximation of ODE can be used only to find the solution at points $i \in [2, 3, \ldots, n-1]$ and therefore, the boundary conditions must be embedded in the range of those points. Writing the approximation of ODE at $i = 2$ and $i = n - 2$

$$\begin{aligned} a_2 y_1 + b_2 y_2 + c_2 y_3 &= d_2, \\ a_{n-1} y_{n-2} + b_{n-1} y_{n-1} + c_{n-1} y_n &= d_{n-1} \end{aligned}$$

and substituting the boundary conditions

$$\begin{aligned} y_1 &= -y_2 \\ y_n &= y_{n-1} - h \end{aligned}$$

results in two required equations embedded in the linear system for the ODE

$$\begin{aligned} (b_2 + a_2)y_2 + c_2 y_3 &= d_2 \\ a_{n-1}y_{n-2} + (b_{n-1} - c_{n-1})y_{n-1} &= d_{n-1} + c_{n-1}h. \end{aligned} \tag{8.4}$$

Setting the system of the linear equations

The system of linear equations approximating the ODE at $i \in [2, 3, \ldots, n-1]$ is written as follows

$$\begin{aligned} (b_2 - a_2)y_2 + c_2 y_3 &= d_2 \\ a_3 y_2 + b_3 y_3 + c_3 y_4 &= d_3 \\ a_4 y_3 + b_4 y_4 + c_4 y_5 &= d_4 \\ \vdots &\quad \vdots \\ a_{n-2}y_{n-3} + b_{n-2}y_{n-2} + c_{n-2}y_{n-1} &= d_{n-2} \\ a_{n-1}y_{n-2} + (b_{n-1} - c_{n-1})y_{n-1} &= d_{n-1} + c_{n-1}h \end{aligned} \tag{8.5}$$

This can be rewritten in the matrix form

$$\begin{bmatrix} \hat{b}_2 & a_2 & & & & & \\ a_3 & b_3 & c_3 & & & & \\ & a_4 & b_4 & c_4 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{n-3} & b_{n-3} & c_{n-3} & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & & a_{n-1} & \hat{b}_{n-1} \end{bmatrix} \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{n-3} \\ y_{n-2} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ \vdots \\ d_{n-3} \\ d_{n-2} \\ \hat{d}_{n-1} \end{bmatrix}$$

where the following substitutions have been applied

$$\begin{aligned} \hat{b}_2 &= b_2 - a_2, \\ \hat{b}_{n-1} &= b_{n-1} + c_{n-1}, \\ \hat{d}_{n-1} &= d_{n-1} + hc_{n-1}. \end{aligned}$$

Solution of the linear system

The liner system expressed by Eq.8.6 can be represented as

$$\mathbf{A} \cdot \mathbf{y} = \mathbf{d} \tag{8.6}$$

and one of the solvers appropriate for a tridiagonal system can be applied. The solution $\mathbf{y}$ is loaded by the error of the first-order accurate in h that is worse in comparison with the second order accurate shooting method with the predictor-corrector method.

8.3 Method of weighted residuals (MWR)

The differential equation can be expressed by using the differential operator $\mathcal{D}$ acting on the function $y(t)$, i.e.

$$\mathcal{D}y(t) = f(t), \quad \text{for} \quad t \in \Omega \subseteq \mathbf{R}^n \tag{8.7}$$

with boundary conditions on $\partial\Omega$. One of the possible forms of the differential operator is

$$\mathcal{D}y = (\frac{d^2}{dx^2} + k^2)y = 0 \quad \text{for} \quad 0 < t < 1. \tag{8.8}$$

The approximate solution $y(t)$ of Eq.8.7 can be expressed as

$$y(t) \approx v(t) = \sum_{j=1}^{n} c_j \varphi_j(t) \tag{8.9}$$

where the so called trial space is spanned on vectors $\{\varphi_j\}_{j=1}^n$. Therefore, the functions $v(t)$ are called the trial functions. Since vectors $\varphi_j(t)$ are known, the goal of MWR is to find n scalars $\{c_j\}_{j=1}^n$ that can be done by minimizing so called residuals

$$r(t) \equiv \mathcal{D}v(t_0) - f(t) \tag{8.10}$$

where r is the residual. The minimization of the residual is performed in the sense of the following integral relation defined over the entire domain Ω:

$$\int_\Omega r(t) w_i(t) dt = 0 \quad \text{for each} \quad i, \tag{8.11}$$

where $\{w_i\}_{i=1}^m$ are called the test functions or weights. The number of weight functions and the number of base functions for the trial space are related. There are many possibilities for the selection of these base functions. One of choices is related to the Lagrange polynomials which we already discussed in one of former chapters. The set of $(n-1)$ degree polynomials related to t_j, $j = 1, 2, \ldots, n$

$$p_j(t) = \prod_{i=1, i\neq j}^{n} \frac{t - t_i}{t_j - t_i} \tag{8.12}$$

forms the basis of the finite-dimensional linear space. Therefore, the trial functions can be expressed as

$$v(t) = \sum_{j=1}^{n} v_j p_j(t), \tag{8.13}$$

where $v_i \equiv v(t_i) = \sum_{j=1}^{n} c_j p_j(t_i) = c_i p_i(t) = c_i$ because $p_j(t_i) = \delta_{ij}$. These polynomials $p_j(t)$ are equal to zero only in a finite number of nodal points and otherwise are nonzero, and moreover, $p_j(t) \in C^{\infty}$.

The other set of base polynomials which could be chosen is the set of piecewise polynomials.

8.4 Galerkin method

The Galerkin method can be derived from MWR by choosing $w_i(t) = \varphi(t)$ as the basis of the trial space. Then the weighted average is written as

$$\int_{\Omega} r(t)\varphi_i(t)dt = 0. \tag{8.14}$$

where $r(t)$ is required to be orthogonal to $\varphi_i(x)$.

The so called Galerkin-Petrov method is constructed by applying another basis.

Example 13, [2], ♡

Solve the following BVP

$$\mathcal{D}y \equiv \left(\frac{d^2}{dt^2} + k^2\right) y = 0, \quad \text{for} \quad t \in (0,1) \tag{8.15}$$

$y(0) = 1$ and $y(1) = 0$.

Choose $h = 0.5$ and then calculate the solution at three nodes: $t = 0$, $t = 0.5$, $t = 1$. Hence, the trial function consists of three base functions $\varphi_j(t)$ of the trial space , i.e.

$$v(t) = \sum_{j=1}^{3} v_j \varphi_j(t), \tag{8.16}$$

with values v_1 and $v_3 = 0$ obtained from the boundary conditions. Therefore, the only unknown value of $v(0.5)$ correspond to the second node:

$$\int_0^1 \left(\frac{d^2 v}{dt^2} + k^2 v\right) \varphi_2(t)dt = 0 \tag{8.17}$$

Substituting Eq.8.16 into Eq.8.17 results in

$$\Sigma_{j=1}^{3} \int_0^1 \left(\frac{d^2\varphi_j}{dt^2} + k^2\varphi_j\right) \varphi_2(t)dt = 0 \tag{8.18}$$

It can be integrated by parts

$$\Sigma_{j=1}^{3} \int_0^1 \left(-\frac{dv}{dt}\frac{d\varphi_2}{dt} + k^2 v\varphi_j \right) dt + \frac{dv}{dt}\varphi_2 \,|_0^1 =$$
$$v_1 \left(\frac{1}{h} + \frac{1}{6}k^2 h \right) + v_2 \left(-\frac{2}{h} + \frac{2}{3}k^2 h \right) + v_3 \left(\frac{1}{h} + \frac{1}{6}k^2 h \right) = 0 \tag{8.19}$$

♠

8.5 Collocation method

The basic idea of the collocation method consists of the following steps:

- choose the finite-dimensional space of solution,
- choose the number of collocation points , i.e. the number of discrete points within the domain Ω,
- select the solution satisfying the given BVP at the collocation points $\{t_i^c\}$.

The last step can be done by minimizing the residual and enforcing it to assume the zero value at a finite number of collocation points. Assuming the test functions $w_i(t)$ in the form

$$w_i(t) = a(t - t_i^c),$$

the residual, defined by Eq. 8.11, is expressed by

$$\int_\Omega r(t)a(t - t_i^c)dt = r(t_i^c) = 0 \tag{8.20}$$

where t_i^c is the i-th collocation point.

Example 13, [2], ♡

Solve the following BVP

$$\mathcal{D}y \equiv \left(\frac{d^2}{dt^2} + k^2 \right) y = 0, \quad \text{for} \quad t \in (0, 1) \tag{8.21}$$

$y(0) = 1$ and $y(1) = 0$.

Choosing the piecewise cubic Hermitian trial space, the trial function approximation is expressed as

$$v(t) = \sum_{j=1}^{n} v_j \varphi_{0j}(t) + \frac{dv_j}{dt}\varphi_{1j}(t) \tag{8.22}$$

Then the second order ODE requires two collocation points per one segment of the domain called as the element. For example, taking $n = 2$, gives two nodes located at ends of the domain, i.e. $t = 0$ and $t = 1$. In this case there are $2n$ unknowns with totally $2n$ equations generated from the ODE and two boundary conditions. The accuracy of calculations of the order $O(h^4)$ can be achieved by choosing the collocation points as the Gauss quadrature points:

$$t_1^c = \frac{1}{6}(3 - \sqrt{3}),$$
$$t_2^c = \frac{1}{6}(3 + \sqrt{3}).$$

Then the residuals can be written as

$$\begin{aligned} r(t_1^c) &= \sum_{j=1}^{2} v_j \frac{d^2\varphi_{0j}}{dt}|_{t=t_1^c} + \frac{dv_j}{dt}\frac{d^2\varphi_{1j}}{dt^2}|_{t=t_1^c} \\ &+ k^2\left[\sum_{j=1}^{2} v_j\varphi_{0j}|_{t=t_1^c} + \frac{dv_j}{dt}\varphi_{1j}|_{t=t_1^c}\right] = 0 \qquad (8.23) \\ r(t_2^c) &= \sum_{j=1}^{2} v_j \frac{d^2\varphi_{0j}}{dt}|_{t=t_2^c} + \frac{dv_j}{dt}\frac{d^2\varphi_{1j}}{dt^2}|_{t=t_2^c} \\ &+ k^2\left[\sum_{j=1}^{2} v_j\varphi_{0j}|_{t=t_2^c} + \frac{dv_j}{dt}\varphi_{1j}|_{t=t_2^c}\right] = 0 \end{aligned}$$

Two Eqs. 8.23 and two boundary conditions compose the system than can be written in the matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ \frac{dv_1}{dt} \\ \frac{dv_2}{dt} \\ v_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad (8.24)$$

where values corresponding to elements of matrix $\mathbf{B}$ are the following

$$\begin{aligned} b_{21} &= b_{34} = -5.657 + 0.998k^2, \\ b_{22} &= -b_{33} = -3.868 + 0.027k^2, \\ b_{23} &= -b_{32} = -1.830 - 0.0008k^2, \\ b_{24} &= b_{31} = 5.657 + 0.002k^2, \end{aligned}$$

Unfortunately, the matrix $\mathbf{B}$ is nonsymmetric, but we can note very interesting combination of symmetry and skew-symmetry of elements which does not help to solve the linear system. The system can be solved by one of methods discussed earlier. ♠

8.6 Variational formulation of BVP for ODE

8.6.1 Preliminaries

Sobolev spaces

Elementary knowledge about the Sobolev spaces is necessary for understanding of variational formulation of BVP for ODE and the finite element method. The following table contains several definitions of norm, metrics, spaces and derivatives needed for the better understanding of the weak formulation of ODE and PDE problems.

Name	**Definition**
Lebesgue measurable set	A set which can be assigned a volume is called the measurable set.
Lebesgue norm	The Lebesgue norm is based on the Lebesgue integral interpreted as the area under a curve.
Lebesgue spaces	The Lebesgue spaces are defined by $L^p(\Omega) := \{f(t) : \|f\|_{L^p(\Omega)} < \infty\}$ for $f(t)$ is a real-valued functions given on the domain Ω, where $\|f\|_{L^p(\Omega)} := (\int_\Omega f^p dt)^{\frac{1}{p}}$ for $1 \leq p \leq \infty$.
Minkowski's inequality	$\|f+g\|_{L^p(\Omega)} \leq (\|f\|_{L^p(\Omega)} + \|g\|_{L^p(\Omega)})$ for $1 \leq p \leq \infty$ and $f, g \in L^p(\Omega)$
Holder's inequality	$\|f(t)g(t)\|_{L^1(\Omega)} \leq \|f\|_{L^p(\Omega)}\|g\|_{L^q(\Omega)}$ for $1 \leq p, q \leq \infty$ such that $\frac{1}{p} + \frac{1}{q} = 1$, and if $f \in L^p(\Omega)$ and $g \in L^q(\Omega)$ then $fg \in L^1(\Omega)$
Cauchy-Schwartz inequality	Cauchy-Schwartz inequality is the special case of the Holder's inequality when $p = q = 2$, i.e. $\int_\Omega \|f(t)g(t)\|dt \leq \|f\|_{L^2(\Omega)}\|g\|_{L^2(\Omega)}$
Norm $\|\cdot\|$ *Triangle inequality*	The norm $\|\cdot\|$ on a given vector space V is a function which has the following properties: 1. $\|v\| \geq 0$ for $\forall v \in 0$ and such that $\|v\| = 0$ for $v = 0$, 2. $\|c \cdot v\| = \|c\| \cdot \|v\|$ for $\forall c \in \mathcal{R}$ and $v \in V$, 3. $\|v + w\| \leq \|v\| + \|w\|$ for $\forall v, w \in V$.
Metric	Distance between two points $v, w \in V$ of the vector space V is defined as $d(v, w) = \|v - w\|$.
Complete metric space	A metric space is complete when every Cauchy sequence $\{v_j\} \in V$ has a limit $v \in V$.
Normed linear space	A Cauchy sequence for normed linear space is convergent $\|v_j - v_k\| \to 0$ for $j, k \to \infty$
Complete normed linear space	The normed linear space is complete, if $\|v - v_k\| \to 0$ as $j \to \infty$
Banach space	The Banach space is a complete normed linear vector space V over the real or complex numbers that is equipped with the norm $\|\cdot\|$ such that every Cauchy sequence in V has a limit in V. e.g $L^p(\Omega)$ for $1 \leq p \leq \infty$ is a Banach space.
Hilbert space	A Hilbert space is a real or complex space equipped with the inner product. The space is complete with the norm induced by the product, i.e $\|f\| = \sqrt{(f, f)}$. A Hilbert space is always a Banach space but the opposite conclusion may not be true. The space $L^2(\Omega)$ is an example of infinite-dimensional Hilbert space.

Name	**Definition**
Support	The support of a function $u \in \mathcal{C}$ defined on the domain $\Omega \subset \mathcal{R}^n$ is the closure of the open set $\{t : u(t) \neq 0\}$. The function u has a compact support if such set is compact and is the interior of Ω.
Functions with compact support	Denote $\mathbf{D}(\Omega) = \mathcal{C}_0^\infty(\Omega) \subset \mathcal{C}^\infty$ where $\mathcal{C}^\infty$ is the set of functions with compact support in $\Omega \subset \mathcal{R}^n$,
Locally integrable functions	The set of locally integrable functions is denoted by $L^1_{loc} := \{f : f \in L^1(K)\}$ for compact $\forall K \subset \Omega$
Weak derivative	The function $f \in L^1_{loc}$ has a weak derivative $D^\alpha_w f$, provided that a function $g \in L^1_{loc}(\Omega)$ exists and fulfils the following relation $\int_\Omega g(t)\varphi(t)dt = (-1)^{\lvert\alpha\rvert} \int_\Omega f(t)\varphi^{(\alpha)}(t)dt$ for $\forall \varphi \in D(\Omega)$, where $\varphi^{(\alpha)}$ means α derivative of φ. Then $D^\alpha_w f = g$ and g is called the weak derivative of f.
Weak derivative in $L^1([a,b])$ space	Assuming that f is a function in the Lebesgue space, i.e. $f \in L^1(\Omega) \equiv L^1([a,b])$ for $\Omega = [a,b]$, **the weak derivative** of f is a function g provided that $\int_a^b f(t)\dot{\varphi}(t)dt = -\int_a^b g(t)\varphi(t)dt$ for all continuously differentiable functions φ such that $\varphi(a) = \varphi(b) = 0$. Examples of weak derivatives are given below.
Sobolev space	The Sobolev space is a normed linear space defined by the following relation $W^k_p(\Omega) := \{f \in L^1_{loc}(\Omega) : \lVert f\rVert_{W^k_p}(\Omega) < \infty\}$ where $f \in L^1_{loc}(\Omega)$ and the norm in this space is given by one of the following definitions: if $1 \leq p \leq \infty$ then $\lVert f\rVert_{W^k_p(\Omega)} := \left(\sum_{\lvert\alpha\rvert \leq k} \lVert D^\alpha_w f\rVert^p_{L^p(\Omega)}\right)^{\frac{1}{p}}$ if $p = \infty$ then $\lVert f\rVert_{W^k_\infty(\Omega)} := max_{\lvert\alpha\rvert\leq k} \lVert D^\alpha_w f\rVert_{L^\infty(\Omega)}$ with $k \geq 0$. The above is true subject to the existence of the weak derivatives $D^\alpha_w f$ for all $\lvert\alpha\rvert \leq k$ The space $W^k_2(\Omega)$ is also called $H^k(\Omega)$.
Theorem:	The Sobolev space $W^k_p(\Omega)$ is also a Banach space.
Bilinear form on linear space	A bilinear form, $b(\cdot,\cdot)$, on a linear space V is a mapping $b : V \times V \to \mathcal{R}$ such that each of the maps: $v \to b(v,w)$, and $w \to b(v,w)$, is a linear form on V
Real inner product	A real inner product $(\cdot,\cdot)$ is a symmetric bilinear form on a linear space V that satisfies the following conditions: $(v,v) \geq 0$ for $\forall v \in V$ and $(v,v) = 0$ when $v = 0$.
Inner-product space	The inner-product space is a linear space V equipped with the inner-product $(\cdot,\cdot)$. It is denoted by $(V,(\cdot,\cdot))$.

Name	Definition
Examples of inner-product spaces	$V = \mathcal{R}^n$, $(x, y) := \sum_{i=1}^{n} x_i y_i$, $V = L^2(\Omega)$, $\Omega \subseteq \mathcal{R}^n$, $(f, g)_{L^2(\Omega)} := \int_\Omega f(t)g(t)dt$, $V = W_2^k$, $\Omega \subseteq \mathcal{R}^n$, $(f, g)_k := \sum_{\lvert\alpha\rvert \leq k} (D^\alpha f, D^\alpha g)_{L^2(\Omega)}$
Schwartz inequality in $V, (\cdot, \cdot)$	The Schwartz inequality $\lvert(f, g)\rvert \leq (f, f)^{\frac{1}{2}}(g, g)^{\frac{1}{2}}$ holds in $(V, (\cdot, \cdot))$ if and only if f and g are linearly dependent.
Norm in $(V, (\cdot, \cdot))$	The norm in $(V, (\cdot, \cdot))$ is defined by the inner-product as $\lVert f\rVert := \sqrt{(f, f)}$.
Hilbert spaces	The inner-product space $(V, (\cdot, \cdot))$ is called a Hilbert space if the associated normed linear space $(V, \lVert \cdot \rVert)$ exists and is complete.
Examples of Hilbert spaces	$V = \mathcal{R}^n$, $(x, y) := \sum_{i=1}^{n} x_i y_i$, $V = L^2(\Omega)$, $\Omega \subseteq \mathcal{R}^n$, $(f, g)_{L^2(\Omega)} := \int_\Omega f(t)g(t)dt$, $V = W_2^k$, $\Omega \subseteq \mathcal{R}^n$, $(f, g)_k := \sum_{\lvert\alpha\rvert \leq k} (D^\alpha f, D^\alpha g)_{L^2(\Omega)}$
Subspace of Hilbert space	Assume that V is a Hilbert space. A linear subset W of V is a subspace of V when any two functions $f, g \in W$ can produce a linear combination $f + \alpha g \in W$ with $\alpha \in \mathcal{R}$. Such subspace W is also a Hilbert space.

Example 14 ♡

The so called sign function $g : [-1, 1] \to [-1, 1]$ given by

$$g : t \to g(t) = \begin{cases} 1, & \text{if} \quad t > 0, \\ 0, & \text{if} \quad t = 0, \\ -1, & \text{if} \quad t < 0, \end{cases}$$

is a weak derivative of the absolute value function $f(t) = |t|$ such that $f : [-1, 1] \to [0, 1]$ that is not differentiable at $t = 0$. It should be emphasized here that it is not the only weak derivative that can be assigned to $f(t)$, because any function $p(t)$ that is equal to $g(t)$ almost everywhere is also a weak derivative for $f(t)$. ♠

Example 15 ♡

Show that the function $g(t)$ given by

$$g(t) := \begin{cases} 1, & \text{if} \quad t < 0, \\ -1, & \text{if} \quad t > 0, \end{cases}$$

is a weak derivative of the function $f(t) = 1 - |t|$, for $t \in \Omega = [-1, 1]$. Note that $f(t)$ is continuous at 0.

Solution

Apply the formula given by the first definition of the weak derivative in above table and break the interval $[-1, 1]$ into two parts where $f(t)$ is smooth and integrate by parts:

$$\begin{aligned}\int_{-1}^{1} f(t)\dot{\varphi}(t)dt &= \int_{-1}^{0} f(t)\dot{\varphi}(t)dt + \int_{0}^{1} f(t)\dot{\varphi}(t)dt \\ &= f\varphi|_{-1}^{0} - \int_{-1}^{0}(+1)\varphi(t)dt + f\varphi|_{0}^{1} - \int_{0}^{1}(-1)\varphi(t)dt \\ &= -\int_{-1}^{1} f(t)\varphi(t)dt - f(0+)\varphi(0+) + f(0-)\varphi(0-) \\ &= -\int_{-1}^{1} g(t)\varphi(t)dt.\end{aligned}$$

♠

Sturm-Liouville form of ODE

All second order ODE can be expressed as the LHS of the Sturm-Liouville (S-L) form called also the self-adjoint form. This observation is very useful because each BVP based of this kind of equation has a unique solution.

The classical S-L form for the real second order ODE is

$$-\frac{d}{dx}\left[p(t)\frac{dy(t)}{dt}\right] + q(t)y(t) = \lambda r(t)y(t) \tag{8.25}$$

where $q(t)$ and $r(t) \in C[a,b]$, $p(t) \in \mathcal{C}^1[a,b]$, $y(t) \in \mathcal{C}^2[a,b]$, and moreover $p(t) > 0$ and $q(t) > 0$ on $[a,b]$. The value λ is not specified by Eq. 8.25, however finding values of λ for which non-trivial solutions of BVP exist, is a part of the S-L problem. The above can be written as

$$\begin{aligned}\mathcal{D}y(t) &= \lambda r(t)y(t), \quad a \le t \le b \\ y(a) &= \alpha, \\ y(b) &= \beta,\end{aligned} \tag{8.26}$$

where

$$\mathcal{D}y(t) = -\frac{d}{dx}\left[p(t)\frac{dy(t)}{dt}\right] + q(t)y(t). \tag{8.27}$$

is the linear operator

$$\mathcal{D} : \mathcal{C}^2[a,b] \Rightarrow \mathcal{C}[a,b]$$

transfers objects from the space $\mathcal{C}^2$ to the space $\mathcal{C}$.

Following the first sentence of this section, each BVP for ODE with non-homogeneous conditions can be written as

$$\begin{aligned} \mathcal{D}y(t) &= r(t), \quad a \leq t \leq b \\ y(a) &= \alpha, \\ y(b) &= \beta. \end{aligned} \tag{8.28}$$

Moreover, it can be transferred to the BVP with homogeneous conditions (both of them equal to zero) subject to the existence of a function $w(t)$ such that takes the same boundary values as $y(t)$. Then the difference of these two functions $\mathcal{Z}(t) = y(t) - w(t)$ fulfils the following BVP

$$\begin{aligned} \mathcal{D}\mathcal{Z}(t) &= r(t) - \mathcal{D}w(t), \quad a \leq t \leq b \\ \mathcal{Z}(a) &= 0, \\ \mathcal{Z}(b) &= 0. \end{aligned} \tag{8.29}$$

The final solution $y(t) \in \mathcal{C}^2[a,b] = \{y \in \mathcal{C}^2[a,b], y(a) = \alpha, y(b) = \beta\}$ of the problem Eq. 8.28 can be done by solving at first the problem Eq. 8.29 for $\mathcal{Z}(t) \in \mathcal{C}_0^2[a,b] = \{y \in \mathcal{C}^2[a,b], y(a) = y(b) = 0\}$. It is important to note from which spaces these solutions are selected. Therefore, the problem Eq. 8.28 can be approximated by the problem Eq. 8.29 replacing $\mathcal{Z}(t)$ by $\hat{y}(t)$, i.e.

$$\begin{aligned} \mathcal{D}\hat{y}(t) &= r(t), \quad a \leq t \leq b \\ \hat{y}(a) &= 0, \\ \hat{y}(b) &= 0. \end{aligned} \tag{8.30}$$

Variational formulation of BVP for ODE

If $\hat{y}$ solves the problem Eq. 8.30, then the first step to convert the problem into its variational form is to select any smooth function $v(t)$ which satisfies the homogeneous boundary conditions, i.e. $v(a) = 0$ and $v(b) = 0$. The second step is to refer to one of quadratic functionals [11] defined in Sobolev spaces $H^1(a,b)$, e.g. inner product. Suppose that $\mathcal{D}\hat{y}(t)$ is expressed in simpler form: $\mathcal{D}\hat{y}(t) = \hat{\ddot{y}}(t)$. Then the problem expressed by Eq 8.30 and boundary conditions can be written as

$$\int_a^b r(t)v(t)dt = \int_a^b \hat{\ddot{y}}(t)v(t)dt. \tag{8.31}$$

Integration of the RHS of Eq. 8.31 gives

$$\begin{aligned} RHS &= \hat{\dot{y}}(t)v(t)|_a^b - \int_a^b \hat{\dot{y}}(t)\dot{v}(t)dt \\ &= -\int_a^b \hat{\dot{y}}(t)\dot{v}(t)dt \\ &= \varphi(\hat{y}, v) \end{aligned} \tag{8.32}$$

Both sides of Eq. 8.31 are the inner products in a Hilbert space.

The third step is to replace the infinite dimensional linear problem (IDLP)

$$\text{Find} \quad \hat{y} \in H_0^1 \quad \text{such that} \quad \forall v \in H_0^1, \quad -\varphi(\hat{y}) = \int_a^b r(t)v(t)dt \tag{8.33}$$

by a finite dimensional problem (FDP)

$$\text{Find} \quad \hat{y} \in H_0^1 \quad \text{such that} \quad \forall v \in V, \quad -\varphi(\hat{y}) = \int_a^b r(t)v(t)dt \tag{8.34}$$

where V is a finite dimensional subspace of H_0^1. The subspace V can be chosen as a space of piecewise linear functions.

Example 16 ♡

Consider the two-point boundary value problem

$$\begin{aligned} Ly &= -\ddot{y} + b\dot{y} + cy = f(t) \quad \text{for} \quad 0 < t < 1, \\ y(0) &= y_0, \\ y(1) &= y_1, \end{aligned} \tag{8.35}$$

where: $b, c \in \mathcal{R}$ with $c \geq 0$, $f(t) \in C[0,1]$, and $y_0, y_1 \in \mathcal{R}$. The weak formulation can be introduced at first assuming the following relations:

$$\begin{aligned} Y &= \{y(t) \in H^1(0,1) \quad \text{with} \quad y(0) = y_0, \quad y(1) = y_1\} \\ V &= \hat{H}^1(0,1), \end{aligned} \tag{8.36}$$

where $H^1(0,1)$ and $\hat{H}^1(0,1)$ are the Sobolev spaces specified as follows:

$$\begin{aligned} H^1(0,1) &= \{y(t) \in L_2(0,1) \quad \text{with generalized} \quad \dot{y}(t) \in L_2(0,1)\}, \\ \hat{H}^1(0,1) &= \{y(t) \in H^1(0,1) \quad \text{such that} \quad y(0) = y(1) = 0.\} \end{aligned} \tag{8.37}$$

These spaces provide the unique solution for the weak formulation of the second order BVP.

The weak form of Eq. 8.36 can be expressed as

Find $y \in Y$ such that

$$a(y,v) = (f,v), \quad \forall v \in V, \tag{8.38}$$

where

$$\begin{aligned} a(y,v) &= \int_0^1 (\dot{y}\dot{v} + b\dot{y}v + cyv)dt \\ (f,v) &= \int_0^1 fvdt. \end{aligned} \tag{8.39}$$

♠

8.7 Reyleigh-Ritz method

Our idea for this section is to follow the magnificent paper by R. Courant published in 1943 [11] with the presentation of the Rayleigh-Ritz (R-R) method. Courant's paper is fundamental for the development of the finite element method called by him as the finite difference method with some generalizations. The paper was one of the fundamental papers for the development of applied mathematics and computational mechanics.

8.7.1 Theory and the principle

Suppose that the minimum d of an integral expression or any variational expression $L(y)$ is seeking. The searching procedure can be initiated with ***the minimizing sequence*** $\phi_1, \phi_2, \ldots, \phi_n$ which is admissible for the variational problem

$$\lim_{n\to\infty} L(\phi_n) = d, \tag{8.40}$$

where d is the lower bound of the functional $L(\phi_i)$. The problem lies in the construction of the minimization sequence and the recipe for such construction is given by the Ritz procedure:

- start with an arbitrary chosen system of so called ***coordinate functions***: $\omega_1, \omega_2, \ldots, \omega_n, \ldots, \omega_z$, which fulfils the following conditions:
  1. Any linear combination of them is admissible in the variational problem, i.e.
  $$\phi_n = a_1\omega_1 + a_2\omega_2 + \ldots + a_n\omega_n \tag{8.41}$$
  2. Any admissible function ϕ_i may be approximated with any accuracy by a linear combination of coordinate functions. Similarly, any derivative of ϕ_i may be approximated by a linear combination of coordinate functions derivatives.
- For n sufficiently large and appropriate choice of coefficients $a_1, a_2, \ldots, a_n$, the admissible function ϕ_n, which differs arbitrary little from d can be find. It means that a minimizing sequence can be also find as the linear combination of the coordinate functions.
- Therefore, to construct required minimizing sequence, the coefficients a_i should be chosen in the following manner:
  1. Propose any function ϕ_n given by 8.41 and substitute it in the variational problem.

2. Then $L(\phi_n)$ becomes a function $F(a_1, a_2, \ldots, a_n)$ of n parameters a_i that can be evaluated form the minimum problem

$$L(\phi_n) = F(a_1, a_2, \ldots, a_n) = \min, \tag{8.42}$$

3. Because $L(\phi_n)$ is quadratic or bilinear functional, Eq.8.42 leads to a system of n linear equations for n parameters a_i and therefore the minimizing sequence ϕ_i is found.

Courant made the important remark regarding the convergence of the minimizing sequence:

- While the convergence of $L(\phi_i)$ to d is assured, it is associated with the convergence of ϕ_i to u which is the true solution of the original minimization problem or, at least, the derivatives of ϕ_i converge to the corresponding derivatives of u.
- The convergence of two sequences: $\phi_1, \phi_2, \ldots, \phi_n, \ldots$ and the sequence of derivatives of ϕ_i, is improved if the order of occurring derivatives becomes higher.
- The convergence becomes worse when the number of independent variables increases.

The better convergence in the R-R method can be assured by modifying the original variational problem by adding higher order terms which vanish for the actual solution u. From the theoretical point of view, the R-R method consists in the construction of the minimizing sequence but the main difficulty is to find suitable coordinate functions and to estimate the accuracy of the final result. The functions should be chosen considering two objectives:

- a_i terms of Eq. 8.42 situated on the main diagonal are dominating,
- the number of accounted terms should be kept small.

The completeness of the sequence ϕ_i is not important because only few of the coordinate functions are accounted in time. To achieve the successful evaluation of u the initial function should be fairly good approximation of u and in addition, the functions ϕ_i should be sufficiently different so that by increasing number of terms the quality of approximation could be increased. The choice of polynomials as coordinate functions leads to better results.

Referring that our original problem can be written in the form: $L(u) = f$, very useful method for the selection of coordinate functions consists of the following idea:

- start with choosing the function ω_1,

- then define $\omega_2 = L(\omega_1)$,
- and proceed further on with this pattern $\omega_3 = L(\omega_2) = LL(\omega_1)$, $\omega_4 = L()\omega_3 = LLL(\omega_1)$.

This idea results in the system of equations for a_i comparatively simple. Unfortunately, this selection of functions lead to some complications at boundaries because of higher derivatives.

8.7.2 Boundary conditions and R-R method objections

The approximation by the R-R method is quite good for the rigid boundary conditions and few admissible coordinate functions would give a solution close to the true one. Unfortunately, in general the rigid boundary conditions preclude the choice of simple coordinate function. The situation is different in the case of free and natural boundary conditions, when the boundary conditions need not to be stipulate in advance. Generally, polynomials with undetermined coefficients can be used as ϕ_i functions. However, the reasonable accuracy of solution requires many more terms. Therefore the chosen functions should satisfy the natural boundary conditions in advance. Nevertheless, the advantage of ϕ_i as polynomials might be decisive.

Courant noticed that rigid boundaries can be considered as a limiting case of free boundaries.

The R-R procedure does not contain a principle for estimating the accuracy of the approximation and this is a weak point of this method. The another disadvantage is related selection of selection of the coordinate functions that requires tedious computations.

Chapter 9

Numerical solutions of partial differential equations (PDE)

9.1 Classification of linear second order PDE's with two independent variables

The general linear equation of second order in two independent variables (x, y) and the function $u(x, y)$ is represented in the general form by the following expression

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu + G = 0, \tag{9.1}$$

where $A, \ldots, G = \text{const}$. This equation can be classified on the basis of discriminant

$$\text{dis} = B^2 - 4AC$$

that can be seen in the following table:

Discriminant value	*PDE type name*
$\text{dis} < 0$	Elliptic
$\text{dis} = 0$	Parabolic
$\text{dis} > 0$	Hyperbolic

It is well know from the courses of calculus, that only mathematically well posed problems can be solved correctly, i.e. problems for which

- the solution exists,
- the solution is unique,
- the solution depends continuously on boundary and initial conditions.

Unfortunately, in most cases of numerical models the existence of a solution does not pose a problem. The existence of a solution may become an issue in the case of nonconvergent iteration sequence and the reason of this could be the nonexistence of a solution. The uniqueness of the solution can also be a problem and it can appear in two cases when

- boundary conditions are irrelevant to the type or order of differential equation or a set of equations,
- the BVP is nonlinear and the numerical solution would provide only one of several solutions.

The continuous dependence of the solution on initial and boundary conditions is related to the uniqueness and in numerical methods this can pose a problem. A non-continuous dependence of the solution on the initial or boundary conditions could become apparent when small change of the initial conditions would lead to large variations in the solution. In the case of evolution problem, i.e. with time marching scheme, it can indicate that the scheme is marching simultaneously on several paths corresponding to various solutions. Then some chaotic results could appear even when the governing equations are deterministic.

For PDE three types of boundary conditions can be set a little bit differently than for ODE:

- Dirichlet conditions with $u = g$ on $\partial\Omega$. These conditions in two dimensional (2-D) space $\Omega = a \leq x \leq b, c \leq y \leq d$ are given at the ends of one of intervals and can be written as $u(a, y) = g_1(y)$ and $u(b, y) = g_2(y)$.
- Neumann conditions with $\frac{\partial u}{\partial n} = g$ or $\frac{\partial u}{\partial s} = f$ on $\partial\Omega$. In the case of 2-D the first derivatives are given at the ends of the intervals, i.e. $u_x(a, y) = g_3(y)$ and $u_x(b, y) = g_4(y)$
- Robin (mixed) conditions $\frac{\partial u}{\partial n} + ku = g$ with positive k on ∂R. These conditions in 2-D case of regular boundary can be written as $u_x(a, y)+k_1u(a, y) = p_1(y)$, and $u_x(b, y) + k_2u(b, y) = p_2(y)$

where Ω is a domain, $\partial\Omega$ is a domain boundary, and g, f and p are real functions given on the boundary ∂R.

Neumann conditions are more frequently used. Dirichlet conditions are applied only when the solution is known on the boundary and function f is analytic.

Some common second order PDE's are shown in the following table:

PDE	***Description***
$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$	Laplace equation
$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = v(x,y)$	Poisson's equation
$\frac{\partial^2 u}{\partial x^2} - \kappa \frac{\partial^2 u}{\partial t^2} =$	wave equation
$\sigma \frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial t} = 0$	diffusion equation
$\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = 0$	heat equation
$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(x,y)u = g(x,y)$	Helmholtz's equation
$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} - bu$	Klein-Gordon equation
$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} - e^{bu}$	modified Loiuville's equation
$\frac{1}{\kappa^2} \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$	vibrating membrane equation

9.2 Elliptic equation

Laplace's, Poisson's and Helmholtz's equations are the most popular elliptic equations. Poisson's equation which is used in modeling of heat distribution in a steady state, can be written in 2-D space in one of the forms

$$\nabla^2 u = -\frac{1}{\kappa} f(x,y), \tag{9.2}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = w(x,y) \tag{9.3}$$

where κ is the conductivity and $f(x,y)$ means the rate of heat distribution per unit area and $w(x,y)$ is the aggregation of the two former RHS elements: the parameter and the function. The objectivity of the boundary value problem with one of three types of boundary value conditions is to find the values of unknown function $u(x,y)$ inside Ω subject to given values on rectangular boundary $\partial\Omega$. It is assumed that both functions: $u(x,y)$ and $w(x,y)$, are continuous on the rectangular grid $\Omega : a < x < b, \quad c < y < d$.

By using finite difference (FD) technique, the derivatives in Eq. 9.3 can be replaced by finite differences based on the Taylor series expansion. In the 2-D case, the domain Ω is replaced by a rectangular grid $n \times m$, generated by so called central FD scheme, is shown in Fig.**??**. The central difference scheme spans over five nodal points: three along x axis and three along y axis with one common node called the central point situated at the cross section of the two directions. The objective of the Poisson's boundary value problem, approximated by using the finite difference technique, is to calculate the values of $u(x,y)$ inside $\Omega = x_0 < x < x_n, \quad y_0 < y < y_n$ for given values of this function on boundaries of the rectangular grid.

Example 17 ♡

Solve the Poisson's equation with the Dirichlet boundary conditions

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= w(x,y) && (9.4) \\ u(x_0,y) &= l_1(y), \quad \text{left boundary condition,} \\ u(x_n,y) &= r_1(y), \quad \text{right boundary condition,} && (9.5) \\ u(x,y_0) &= b_1(y), \quad \text{bottom boundary condition,} \\ u(x,y_m) &= u_1(y), \quad \text{upper boundary condition.} \end{aligned}$$

The solution consists of two steps:

- replacement the PDE by FD form by eliminating the derivatives by using the central difference scheme.
- solution of the system of linear equations by one of methods presented in Chapter 4 or 5, eg. Gaussian elimination or Crout's method.

The first step:

The PDF is replaced by the following FD formula

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2} = w_{i,j}$$

that after rearrangement of the denominator terms becomes

$$h^2(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + k^2(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) = k^2h^2w_{i,j},$$

and finally can be expressed in the form

$$h^2u_{i,j-1} + k^2u_{i,j-1} - 2(h^2+k^2)u_{i,j} + k^2u_{i+1,j} + h^2u_{i,j+1} = h^2k^2w_{i,j} \qquad (9.6)$$

Let us consider Eq. 9.6 for some specific areas of the rectangular domain with $n \times m$ nodes

Level $j = 1$	Left node $i = 1$	$-2(h^2+k^2)u_{1,1} + k^2u_{2,1} + h^2u_{1,2} =$ $h^2k^2w_{1,1} - h^2u_{1,0} - k^2_{0,1}$
Level $j = 1$	Interior nodes	$k^2u_{i-1,1} - 2(h^2+k^2)u_{i,1} + k^2u_{i+1,1} + h^2u_{i,2} =$ $h^2k^2w_{i,1} - h^2u_{i,0}$
Level $j = 1$	Right node, $i = n-1$	$k^2u_{n-2,1} - 2(h^2+k^2)u_{n-1,1} + h^2u_{n-1,2} =$ $h^2k^2w_{n-1,1} - h^2u_{n-1,0} - k^2u_{n,1}$
Middle level	Left node $i = 1$	$h^2u_{i,j-1} - 2(h^2+k^2)u_{1,j} + k^2u_{2,j} + h^2u_{1,j+1} =$ $h^2k^2w_{1,j} - k^2u_{0,j}$
Middle level	Interior nodes	$h^2u_{i,j-1} + k^2u_{i-1,j} - 2(h^2+k^2)u_{i,j} + k^2u_{i+1,j} + h^2u_{i,j+1} =$ $h^2k^2w_{i,j}$
Middle level	Right node $i = n-1$	$h^2u_{n-1,j-1} + k^2u_{n-2,j} - 2(h^2+k^2)u_{n-1,j} + h^2u_{n-1,j+1} =$ $h^2k^2w_{n-1,j} - k^2u_{n,j}$
Level $j = m-1$	Left node $i = 1$	$h^2u_{1,m-2} - 2(h^2+k^2)u_{1,m-1} + k^2u_{2,m-1} =$ $h^2k^2w_{1,m-1} - k^2u_{o,m-1} - h^2u_{1,m}$
Level $j = m-1$	Interior node	$h^2u_{i,m-2} + k^2u_{i-1,m-1} - 2(h^2+k^2)u_{i,m-1} + k^2u_{i+1,m-1} =$ $h^2k^2w_{i,m-1} - h^2u_{i,m}$
Level $j = m-1$	Right node $i = n-1$	$h^2u_{n-1,m-2} + k^2u_{n-2,m-1} - 2(h^2+k^2)u_{n-1,m-1} =$ $h^2k^2w_{n-1,m-1} - k^2u_{n,m-1} - h^2u_{n-1,m}$

The second step:

The system of linear FD equations is obtained by applying equations derived above for the grid $m \times n$. The system is also $m \times n$.

$$\begin{bmatrix} d^2 & k^2 & 0 & h^2 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ k^2 & d^2 & k^2 & 0 & h^2 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & k^2 & d^2 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ h^2 & 0 & 0 & d^2 & k^2 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & h^2 & 0 & k^2 & d^2 & \dots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & d^2 & k^2 & 0 & h^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & k^2 & d^2 & 0 & 0 & h^2 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & d^2 & k^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & h^2 & 0 & k^2 & d^2 & k^2 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & h^2 & 0 & k^2 & d^2 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{4,1} \\ u_{5,1} \\ \vdots \\ u_{n-5,m-1} \\ u_{n-4,m-1} \\ u_{n-3,m-1} \\ u_{n-2,m-1} \\ u_{n-1,m-1} \end{bmatrix} =$$

$$\begin{bmatrix} h^2k^2w_{1,1} - h^2u_{1,0} - k^2u_{0,1} \\ h^2k^2w_{2,1} - h^2u_{2,0} \\ h^2k^2w_{3,1} - h^2u_{3,0} \\ h^2k^2w_{4,1} - h^2u_{4,0} \\ h^2k^2w_{5,1} - h^2u_{5,0} \\ \vdots \\ h^2k^2w_{n-5,m-1} - h^2un - 5, m \\ h^2k^2w_{n-4,m-1} - h^2un - 4, m \\ h^2k^2w_{n-3,m-1} - h^2un - 3, m \\ h^2k^2w_{n-2,m-1} - h^2un - 2, m \\ h^2k^2w_{n-1,m-1} - k^2u_{n,m-1} - h^2un - 1, m \end{bmatrix}$$

where $d^2 = -2(h^2 + k^2)$.

The above system can be expressed by very compact form as

$$\mathbf{Au} = \mathbf{b} \tag{9.7}$$

where elements $a_{i,j}$ of the matrix $\mathbf{A}$ are determined as follows

$$a_{i,j} = \begin{cases} -2(h^2 + k^2) & \text{for} \quad i = j, \\ h^2 & \text{for} \quad a_{i,n-1+i} = a_{n-1+j,j}, \\ k^2 & \text{if} \quad i \mod (n-1) \neq 0 \quad \text{and} \quad i < (m-1)(n-1), \\ 0 & \text{elsewhere,} \end{cases}$$

The Matlab Program for the Second Order FD Solution to Poisson's Equation Code is given below:

1. % Numerical approximation to Poisson's equation over the square $[a, b]x[a, b]$ with
2. % Dirichlet boundary conditions. Uses a uniform mesh with $(n+2)x(n+2)$ total

3. % points (i.e, $n \times n$ interior grid points).
4. % Input:
5. % pfunc : the RHS of poisson equation (i.e. the Laplacian of u).
6. % bfunc : the boundary function representing the Dirichlet B.C.
7. % a, b : the interval defining the square
8. % $n : n + 2$ is the number of points in either direction of the mesh.
9. % Ouput:
10. % u : the numerical solution of Poisson equation at the mesh points.
11. % x, y : the uniform mesh.
12.
13. function $[u, x, y] = fd2poisson(pfunc, bfunc, a, b, n)$
14.
15. $h = (b - a)/(n + 1)$; % Mesh spacing
16.
17. $[x, y]$ = meshgrid$(a : h : b)$; % Uniform mesh, including boundary points.
18.
19. % Compute u on the boundary from the Dirichlet boundary condition
20. ub = zeros(n, n);
21. $idx = 2 : n + 1$;
22. $idy = 2 : n + 1$;
23. % West and East boundaries need special attention
24. $ub(:, 1) = feval(bfunc, x(idx, 1), y(idy, 1))$; % West Boundary
25. $ub(:, n) = feval(bfunc, x(idx, n + 2), y(idy, n + 2))$; % East Boundary
26. % Set the North and South boundaries
27. $ub(1, 1 : n) = ub(1, 1 : n) + feval(bfunc, x(1, idx), y(1, idy))$;
28. $ub(n, 1 : n) = ub(n, 1 : n) + feval(bfunc, x(n + 2, idx), y(n + 2, idy))$;
29. % Convert ub to a vector using column reordering

30. $ub = (1/h^2) * reshape(ub, n * n, 1);$

31.

32. % Evaluate the RHS of Poisson's equation at the interior points.

33. $f = feval(pfunc, x(idx, idy), y(idx, idy));$

34. % Convert f to a vector using column reordering

35. $f = reshape(f, n * n, 1);$

36.

37. % Create the $D2x$ and $D2y$ matrices

38. $z = [-2; 1; zeros(n - 2, 1)];$

39. $D2x = 1/h^2 * kron(toeplitz(z, z), eye(n));$

40. $D2y = 1/h^2 * kron(eye(n), toeplitz(z, z));$

41.

42. % Solve the system

43. $u = (D2x + D2y) \setminus (f - ub);$

44. % Convert u from a column vector to a matrix to make it easier to work with

45. % for plotting.

46. $u = reshape(u, n, n);$

47.

48. % Append on to u the boundary values from the Dirichlet condition.

49. $u = [[feval(bfunc, x(1, 1 : n + 2), y(1, 1 : n + 2))];...$

50. $[[feval(bfunc, x(2 : n + 1, 1), y(2 : n + 1, 1))]u \ ...$

51. $[feval(bfunc, x(2 : n + 1, n + 2), y(2 : n + 1, n + 2))]] \ ;...$

52. $[feval(bfunc, x(n + 2, 1 : n + 2), y(n + 2, 1 : n + 2))]];$

9.2.1 Multigrid solution of Poisson's equation

Introduction

The multigrid algorithm for solution of Poisson's equation is presented here following [37].

Consider Eq. 9.1 with an approximation to a solution $\hat{u}$. Each complete iteration of a multigrid scheme can be expressed in the form

$$u = \hat{u} + v, \tag{9.8}$$

where u is a better solution than $\hat{u}$ and v is the update that is also satisfying Eq. 9.2

$$\nabla^2 v = r \tag{9.9}$$

where $r = f - \nabla^2 \hat{u}$ is called the residual of the current approximation. The accuracy of the solution u is determined by by the accuracy of the spatial discretization in the computation of the residual. The transfer of information from a coarser to a finer grid involves the Jacobi iterations.

Diagonal multigrid for the 2D Poisson equation

Geometrically based multigrid algorithm uses a hierarchy of grid spacing which are proportional to 2^l where l the level of the grid. For 2D case the hierarchy of grids is proportional to $2^{-\frac{1}{2}l}$ with grids aligned at $\frac{1}{4}\pi$ to adjacent levels. The hierarchical grid is organized in the following way: assuming that spacing of the finest grid is h, the first coarser grid is constructed along the diagonals of the former with spacing $\sqrt{2}h$, and the second coarser is aligned with the finest grid with spacing $2h$. This pattern is repeated and is shown in Fig.**??**. The number of grid points halves when going from a fine level to the first coarser one.

The restriction operator is defined for the multigrid solution algorithm which is used for smoothing of the residual r from one grid to the next coarser grid. It is defined as the weighted average of values taken at a grid point and its four nearest neighbours situated on the finer grid. The appropriate scheme, known also as the blue-red scheme, is shown in Fig.**??**. The restriction operators are defined as follows:

$$r^{l-1}_{i,j} = \tfrac{1}{8}(4r^l_{i,j} + r^l_{i-1,j} + r^l_{i,j-1} + r^l_{i+1,j} + r^l_{i,j+1}) \quad \text{from green to red grid} \tag{9.10}$$

$$r^{l-1}_{i,j} = \tfrac{1}{8}(4r^l_{i,j} + r^l_{i-1,j-1} + r^l_{i+1,j-1} + r^l_{i+1,j+1} + r^l_{i-1,j+1}) \quad \text{from red to green grid} \tag{9.11}$$

Assuming that the finest grid is $n \times n$, the restriction to the next finer diagonal grid, marked in Fig. **??**, takes $K_r \approx 3n^2$ FLOPS, where FLOPS means a number

of FLoating point Operations Per Second and is measured in units $\frac{1}{s}$ that can be written as 1 FLOPS $= \frac{1}{s}$, where s stands for a second. The restriction to the next finer grid takes $K_r \approx \frac{3}{2}n^2$ FLOPS. Following that, the restriction of residuals on $l = 2L$ levels for the coarsest grid spacing $H = 2^L h$ takes

$$K_r \approx 6N(1 - \frac{1}{4^L}) \approx 6N \quad \text{FLOPS} \tag{9.12}$$

Jacobi prolongation of v

In the prolongation step the Jacobi iteration, known as "red-black" iteration, is used on the grid shown in Fig. **??**. The idea of such iteration is the following:

- compute the new values at the red points and then refine the values at the blue points,
- compute the new values at the green points and next refine those at the red points.

The hierarchy of diagonally oriented grids accounts for the operational savings because here is no need to interpolate in the prolongation step from one level to the next finer level. The prolongation from the blue grid to the finer diagonal red grid on the level l (see Fig. 3 a page E9, by A.J. Roberts) is realized by the weighted interpolation with the use of the Jacobi iteration and is given by

$$v^l_{i,j} = \frac{1}{4}(-2h^2 r^l_{i,j} + v^{l-1}_{i-1,j-1} + v^{l-1}_{i+1,j-1} + v^{l-1}_{i+1,j+1} + v^{l-1}_{i-1,j+1}). \tag{9.13}$$

The red grid spacing is assumed to be $\sqrt{2}h$. The values at the blue grid points are calculated afterwards by using weighted interpolation with the use of the Jacobi's iteration

$$v^l_{i,j} = \frac{1}{4}(-2h^2 r^l_{i,j} + v^{l}_{i-1,j-1} + v^{l}_{i+1,j-1} + v^{l}_{i+1,j+1} + v^{l}_{i-1,j+1}). \tag{9.14}$$

The same prolongation scheme can be applied to define values at finer grid shown in Fig. (Fig. 3b, page E9, by A.J. Roberts). Such prolongation transmission for one iteration and all green grid points takes $6N$ FLOPS. The total number of operations counts up to

$$K_p \approx 12N(1 - \frac{1}{4^L}) \quad \text{FLOPS} \approx 12N \quad \text{FLOPS} \tag{9.15}$$

for the prolongation to the grid spacing $H = 2^L h$.

The multigrid iteration can be improved by using the over-relaxation technique that is revealed here by introducing a small p parameter into Eqs. 9.14 and 9.15.

Therefore, these equations reads as follows

$$v_{i,j}^{l} = \frac{1}{4}(-2ph^2 r_{i,j}^{l} + v_{i-1,j-1}^{l-1} + v_{i+1,j-1}^{l-1} + v_{i+1,j+1}^{l-1} + v_{i-1,j+1}^{l-1}) \tag{9.16}$$
$$v_{i,j}^{l} = \frac{1}{4}(-2ph^2 r_{i,j}^{l} + v_{i-1,j-1}^{l} + v_{i+1,j-1}^{l} + v_{i+1,j+1}^{l} + v_{i-1,j+1}^{l}).$$

The optimal value of the parameter p is $p = 1.052$.

9.2.2 Laplace's equation

The Laplace's equation is the homogeneous form of the Poisson's equation, which is obtained setting the RHS of Eq.(9.3) to zero, i.e. $w(x, y) = 0$. This is reflected in the system Eq.(**??**) by eliminating $w_{i,j}$ from the RHS column that can be written as

$$\begin{bmatrix} -h^2u_{1,0} - k^2u_{0,1} \\ -h^2u_{2,0} \\ -h^2u_{3,0} \\ -h^2u_{4,0} \\ -h^2u_{5,0} \\ \vdots \\ -h^2un-5,m \\ -h^2un-4,m \\ -h^2un-3,m \\ -h^2un-2,m \\ -k^2u_{n,m-1} - h^2un-1,m \end{bmatrix}$$

9.3 Parabolic equations

The category of parabolic is always represented by the heat equation and describes a variation of parameters, controlling some process, with temperature. A more general form of this equation is known as the diffusion equation. It is very popular and meets applications in statistics and physics.

The heat equation in one dimensional case is expressed by

$$\frac{d\theta}{dt} = c^2 \frac{d^2\theta}{dx^2}, \tag{9.17}$$

where θ is temperature of a particle shown as a point $x \in [x_0, x_n]$ in a rod. The domain of this problem is $\{\Omega : x_0 < x < x_n, \quad t > 0\}$ where n is a number subintervals along x axis. Historically, the equation acquired its name from the model of heat flow along a metallic rod. Obviously, it can be also used for 2D and 3D heat transfer problems. The initial boundary value problem is specified by

- the initial condition

$$\theta(x,0) = f(\theta) \quad \text{for} \quad x \in [x_0, x_n] \tag{9.18}$$

- and the boundary conditions

$$\begin{aligned} \theta(x_0,t) &= g_1(t), \\ \theta(x_n,t) &= g_2(t) \quad \text{for} \quad t > 0. \end{aligned} \tag{9.19}$$

The heat equation can be solved by using one of three schemes:

- forward difference in time and centered difference in space, (FTCS),
- backward difference in time and centered difference in space, (BTCS),
- Crank-Nicolson (CN).

The table below consists of schemes for the solution of the heat equation

Scheme name	Time scheme	Space scheme	FD Formula
FTCS	$\frac{\theta_j^{k+1}-\theta_j^k}{\tau}$	Centered Space $C\frac{\theta_{j+1}^k-2\theta_j^k+\theta_{j-1}^k}{h^2}$	Time Explicit FD Formula $\theta_j^{k+1} = \theta_j^k + C\frac{\tau}{h^2}(\theta_{j+1}^k - 2\theta_j^k + \theta_{j-1}^k)$
BTCS	$\frac{\theta_j^{k+1}-\theta_j^k}{\tau}$	Centered Space $C\frac{\theta_{j+1}^{k+1}-2\theta_j^{k+1}+\theta_{j-1}^{k+1}}{h^2}$	Time Implicit FD Formula $\theta_j^{k+1} = \theta_j^k + C\frac{\tau}{h^2}(\theta_{j+1}^{k+1} - 2\theta_j^{k+1} + \theta_{j-1}^{k+1})$
NC	$\frac{\theta_j^{k+1}-\theta_j^k}{\tau}$	Average Centered $\frac{C}{2h^2}(\theta_{j+1}^{k+1} - 2\theta_j^{k+1} + \theta_{j-1}^{k+1} + \theta_{j+1}^k - 2\theta_j^k + \theta_{j-1}^k)$	Time Implicit FD Formula $\theta_j^{k+1} = \theta_j^k + C\frac{\tau}{h^2}(\theta_{j+1}^{k+1} - 2\theta_j^{k+1} + \theta_{j-1}^{k+1} + \theta_{j+1}^k - 2\theta_j^k + \theta_{j-1}^k)$

where C is the heat conduction, and the fraction $r = C\frac{\tau}{h^2}$ is the Courant number. The features of the heat equation solution schemes are summarized in the following table:

Scheme name	Stability	Local error
FTCS	Stable subject that $r = C\frac{\tau}{h^2} \leq 0.5$	Local Truncation Error $\varepsilon(h,\tau) = O(\tau) + O(h^2)$
BTCS	Unconditionally stable	Local Truncation Error the same as for FTCS
CN	Unconditionally stable	Local Truncation Error $\varepsilon(h,\tau) = O(\tau^2) + O(h^2)$

The Crank-Nicolson's (CN) scheme is the most popular implicit method for the solution of the heat problem. This scheme can be derived from the general implicit scheme

$$-sr\theta_{j-1}^{k+1} + (1+2sr)\theta_j^{k+1} - sr\theta_{j+1}^{k+1} = (1-s)r\theta_{j-1}^k + [1-2(1-s)r]\theta_j^k + (1-s)r\theta_{j-1}^k \tag{9.20}$$

as the special case for switch value $s = \frac{1}{2}$,

$$-r\theta_{j-1}^{k+1} + 2(1+r)\theta_j^{k+1} - r\theta_{j+1}^{k+1} = r\theta_{j-1}^k + 2(r-1)\theta_j^k + r\theta_{j+1}^k. \tag{9.21}$$

The other schemes can be also derived from Eq. 9.20

- FTCS for $q = 0$,
- BTCS for $q = 1$.

The scheme Eq.9.20 for the discrete domain results in the three-diagonal system of linear equations $\mathbf{Au} = \mathbf{D}$, that can be written as is shown correspondingly for the Dirichlet boundary conditions $\theta_e = T_e(t)$, $\theta_w = T_w(t)$, Eq. **??**, and the von Neumann boundary conditions $\frac{\partial\theta}{\partial x}|_e \approx \frac{\theta^1-\theta^2}{h}|_e = \partial T_e(t)$ and $\frac{\partial\theta}{\partial x}|_w \approx \frac{\theta^1-\theta^2}{h}|^w = \partial T_w(t)$, where subscripts e and w means "east" and "west" boundary conditions. The FD system for the von Neumann boundary conditions is written as follows

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ -sr & 1+2sr & -sr & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & -sr & 1+2sr & -sr & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & -sr & 1+2sr & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -sr & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1+2sr & -sr & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & -sr & 1+2sr & -sr & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & -sr & 1+2sr & -sr \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -sr & 1+2sr \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_0^{k+1} \\ \theta_1^{k+1} \\ \theta_2^{k+1} \\ \theta_3^{k+1} \\ \theta_4^{k+1} \\ \vdots \\ \theta_{n-4}^{k+1} \\ \theta_{n-3}^{k+1} \\ \theta_{n-2}^{k+1} \\ \theta_{n-1}^{k+1} \\ \theta_n^{k+1} \end{bmatrix} =$$

$$\begin{bmatrix} T_e^k \\ (1-s)r\theta_0^k + [1-2(1-s)r]\theta_1^k + (1-s)r\theta_2^k \\ (1-s)r\theta_1^k + [1-2(1-s)r]\theta_2^k + (1-s)r\theta_3^k \\ (1-s)r\theta_2^k + [1-2(1-s)r]\theta_3^k + (1-s)r\theta_4^k \\ (1-s)r\theta_3^k + [1-2(1-s)r]\theta_4^k + (1-s)r\theta_5^k \\ \vdots \\ (1-s)r\theta_{n-6}^k + [1-2(1-s)r]\theta_{n-5}^k + (1-s)r\theta_{n-4}^k \\ (1-s)r\theta_{n-5}^k + [1-2(1-s)r]\theta_{n-4}^k + (1-s)r\theta_{n-3}^k \\ (1-s)r\theta_{n-4}^k + [1-2(1-s)r]\theta_{n-3}^k + (1-s)r\theta_{n-2}^k \\ (1-s)r\theta_{n-3}^k + [1-2(1-s)r]\theta_{n-2}^k + (1-s)r\theta_{n-1}^k \\ (1-s)r\theta_{n-2}^k + [1-2(1-s)r]\theta_{n-1}^k + (1-s)r\theta_n^k \\ T_w^k \end{bmatrix}$$

The FD system for the Dirchlet boundary conditions is written in the form:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ -sr & 1+2sr & -sr & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & -sr & 1+2sr & -sr & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & -sr & 1+2sr & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -sr & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1+2sr & -sr & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & -sr & 1+2sr & -sr & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & -sr & 1+2sr & -sr \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -sr & 1+2sr \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \theta_0^{k+1} \\ \theta_1^{k+1} \\ \theta_2^{k+1} \\ \theta_3^{k+1} \\ \theta_4^{k+1} \\ \vdots \\ \theta_{n-4}^{k+1} \\ \theta_{n-3}^{k+1} \\ \theta_{n-2}^{k+1} \\ \theta_{n-1}^{k+1} \\ \theta_n^{k+1} \end{bmatrix} =$$

$$\left[\begin{array}{c} h\partial T_e^k \\ (1-s)r\theta_0^k + [1-2(1-s)r]\theta_1^k + (1-s)r\theta_2^k \\ (1-s)r\theta_1^k + [1-2(1-s)r]\theta_2^k + (1-s)r\theta_3^k \\ (1-s)r\theta_2^k + [1-2(1-s)r]\theta_3^k + (1-s)r\theta_4^k \\ (1-s)r\theta_3^k + [1-2(1-s)r]\theta_4^k + (1-s)r\theta_5^k \\ \vdots \\ (1-s)r\theta_{n-6}^k + [1-2(1-s)r]\theta_{n-5}^k + (1-s)r\theta_{n-4}^k \\ (1-s)r\theta_{n-5}^k + [1-2(1-s)r]\theta_{n-4}^k + (1-s)r\theta_{n-3}^k \\ (1-s)r\theta_{n-4}^k + [1-2(1-s)r]\theta_{n-3}^k + (1-s)r\theta_{n-2}^k \\ (1-s)r\theta_{n-3}^k + [1-2(1-s)r]\theta_{n-2}^k + (1-s)r\theta_{n-1}^k \\ (1-s)r\theta_{n-2}^k + [1-2(1-s)r]\theta_{n-1}^k + (1-s)r\theta_n^k \\ h\partial T_w^k \end{array}\right]$$

Both systems can be solved by applying so called Thomas algorithm known also as "progonka" method.

Example 17 ♡

Solve the heat conduction problem for a one dimensional rod. The problem is defined by the one dimensional heat equation

$$\frac{\partial\theta}{\partial t} = \frac{\lambda}{\rho c}\frac{\partial^2\theta}{\partial x^2} \tag{9.22}$$

with the initial condition

$$T(0 < x < 0.1, t = 0) = 0C = 273K \tag{9.23}$$

and the boundary conditions

$$\begin{aligned} T(x = 0, \text{all t}) &= 300C = 573K, \\ T(x = 0.1, \text{all t}) &= 100C = 373K. \end{aligned} \tag{9.24}$$

The conducting rod is made of copper with the following material parameters:

$$\begin{aligned} \triangle t &= 2\text{s} \\ \triangle x &= 0.025\text{m} \\ \lambda &= 400\frac{\text{W}}{\text{m K}} \\ \varrho &= 8300\frac{\text{kg}}{\text{m}^3} \\ c &= 400\frac{\text{J}}{\text{kg K}}. \end{aligned} \tag{9.25}$$

It is useful to evaluate the quantity k related to the discretization parameters and defined by

$$k = \frac{\lambda}{\varrho c}\frac{\triangle t}{(\triangle x)^2} = 0.386. \tag{9.26}$$

The problem can be solved by applying various schemes. As the first one we can choose the explicit scheme

$$\theta_i^{m+1} = \theta_i^m + k(\theta_{i-1}^m - 2\theta_i^m + \theta_{i+1}^m), \tag{9.27}$$

then we can obtain

$$\begin{aligned}
\text{for} \quad t &= 2\text{s} \\
\theta_1^1 &= \theta_1^0 + k(\theta_0^0 + \theta_1^0 + \theta_2^0) = 273 + 0.386[573 - 2 \cdot 273 + 273] = 388.66 \quad K \\
\theta_2^1 &= \theta_2^0 + k(\theta_1^0 + \theta_2^0 + \theta_3^0) = 273 \quad K \\
\theta_3^1 &= \theta_3^0 + k(\theta_2^0 + \theta_3^0 + \theta_4^0) = 311.5 \quad K \\
\text{for} \quad t &= 4\text{s} \\
\theta_1^2 &= \theta_1^1 + k(\theta_0^1 + \theta_1^1 + \theta_2^1) = 415.1 \quad K \\
\theta_2^2 &= \theta_2^1 + k(\theta_1^1 + \theta_2^1 + \theta_3^1) = 332.5 \quad K \\
\theta_3^2 &= \theta_3^1 + k(\theta_2^1 + \theta_3^1 + \theta_4^1) = 320.4 \quad K \\
\text{for} \quad t &= 6\text{s} \\
\theta_1^3 &= \theta_1^2 + k(\theta_0^2 + \theta_1^2 + \theta_2^2) = 444.1 \quad K \\
\theta_2^3 &= \theta_2^2 + k(\theta_1^2 + \theta_2^2 + \theta_3^2) = 359.7 \quad K \\
\theta_3^3 &= \theta_3^2 + k(\theta_2^2 + \theta_3^2 + \theta_4^2) = 345.3 \quad K
\end{aligned} \tag{9.28}$$

As the second choice we can consider the simple implicit method where two FD schemes are available: the centered FDD for approximation of the second space derivative of θ, and the backward FDD for approximation of time derivative.

$$\begin{aligned}
\frac{\partial^2\theta}{\partial x^2} &\approx \frac{\theta_{i-1}^{m+1} - 2\theta_i^{m+1} + \theta_{i+1}^{m+1}}{(\triangle x)^2} + O(\triangle x^2), \\
\frac{\partial\theta}{\partial t} &\approx \frac{\theta_i^{m+1} - \theta_i^m}{\triangle t} + O(\triangle t).
\end{aligned} \tag{9.29}$$

Then the combined scheme for the one-dimension heat equation is

$$-k\theta_{i-1}^{m+1} + (1+2k)\theta_i^{m+1} - k\theta_{i+1}^{m+1} = \theta_i^m \tag{9.30}$$

This scheme should be written for three specific cases:

- at the left boundary

$$(1+2k)\theta_1^{m+1} - k\theta_2^{m+1} = \theta_1^m + k\theta_0^{m+1} \tag{9.31}$$

- away from the boundary

$$-k\theta_{i-1}^{m+1} + (1+2k)\theta_i^{m+1} - k\theta_{i+1}^{m+1} = \theta_i^m \tag{9.32}$$

- at the right boundary

$$(1+2k)\theta_{N-1}^{m+1} - k\theta_{N-2}^{m+1} = \theta_{N-1}^{m} + \theta_{N}^{m+1} \tag{9.33}$$

Substituting initial and boundary values to Eq.9.31, Eq.9.32 and Eq.9.33 the system of three equations for $\theta_1^1, \theta_2^1, \theta_3^1$ is obtained

$$\begin{aligned} (1+2k)\theta_1^1 - k\theta_2^1 &= T_1^0 + kT_0^1 \\ -k\theta_{i-1}^1 + (1+2k)\theta_i^1 - k\theta_{i+1}^1 &= T_i^0 \\ (1+2k)\theta_{N-1}^1 - k\theta_{N-2}^1 &= T_{N-1}^0 + T_N^1 \end{aligned}$$

Values of RHSs for the above equations are shown in the following table

Equation	**RHS value**
at the left boundary	$T_1^0 + kT_0^1 = 273 + 0.39 \cdot 573 = 496.5$
away from the boundary	$T_i^0 = 273$
at the right boundary	$T_{N-1}^0 + T_N^1 = 273 + 0.39 \cdot 373 = 418.5$

Introducing a value of k and RHSs the system of equations for the first time step is given by

$$\begin{aligned} 1.78 \quad \theta_1^1 - 0,39 \quad \theta_2^1 &= 496.50 \\ -0.39 \quad \theta_{i-1}^1 + 1.78 \quad \theta_i^1 - 0.39 \quad \theta_{i+1}^1 &= 273 \\ 1.78 \quad \theta_{N-1}^1 - 0.39 \quad \theta_{N-2}^1 &= 418.50 \end{aligned}$$

It can be also written in the matrix form as follows

$$\begin{bmatrix} 1.78 & -0.39 & 0 \\ -0.39 & 1.78 & -0.39 \\ 0 & -0.39 & 1.78 \end{bmatrix} \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \theta_3^1 \end{bmatrix} = \begin{bmatrix} 496.5 \\ 273 \\ 418.5 \end{bmatrix}. \tag{9.34}$$

Finally, the solution of Eq. 9.34 for the first time step $t = 1$ can be written as follows

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \theta_3^1 \end{bmatrix} = \begin{bmatrix} 343.4 \\ 294.2 \\ 299.6 \end{bmatrix}. \tag{9.35}$$

The solution for the second time step, i.e.$t = 2$, can be obtained similarly. Therefore, the scheme consisting Eqs. 9.31, 9.32 and 9.33 should be written now for three specific cases correspondingly for the second time step:

- at the left boundary

$$(1+2k)\theta_1^2 - k\theta_2^2 = \theta_1^1 + kT_0^2 \tag{9.36}$$

- away from the boundary

$$-k\theta_{i-1}^2 + (1+2k)\theta_i^2 - k\theta_{i+1}^2 = \theta_i^1 \tag{9.37}$$

- at the right boundary

$$(1+2k)\theta_{N-1}^2 - k\theta_{N-2}^2 = \theta_{N-1}^1 + T_N^2 \tag{9.38}$$

Substituting numerical values for parameters of above equations, their RHS's can be evaluated sa follows

- at the left boundary

$$\theta_1^1 + 0.39T_0^2 = 343.4 + 0.39 \cdot 573 = 566.90 \tag{9.39}$$

- away from the boundary

$$\theta_i^1 = 294.20 \tag{9.40}$$

- at the right boundary

$$\theta_3^1 + T_4^2 = 299.6 + 0.39 \cdot 373 = 445.00 \tag{9.41}$$

and once again the matrix system can be written

$$\begin{bmatrix} 1.78 & -0.39 & 0 \\ -0.39 & 1.78 & -0.39 \\ 0 & -0.39 & 1.78 \end{bmatrix} \begin{bmatrix} \theta_1^2 \\ \theta_2^2 \\ \theta_3^2 \end{bmatrix} = \begin{bmatrix} 566.90 \\ 294.20 \\ 445.00 \end{bmatrix}. \tag{9.42}$$

The solution of Eq. 9.42 for the second time step $t = 2$ is

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \\ \theta_3^2 \end{bmatrix} = \begin{bmatrix} 388.70 \\ 320.60 \\ 320.20 \end{bmatrix}. \tag{9.43}$$

The third scheme which can be used for the heat conduction problem solution is the Crank-Nicolson (CR) implicit scheme which provides the second order accuracy in space approximation. The scheme approximating the second derivative is constructed as the linear average of two second derivatives taken in $m+1$ and m time steps

$$\frac{\partial^2\theta}{\partial x^2} \approx \frac{1}{6}\left[\frac{\theta_{i-1}^m - 2\theta_i^m + \theta_{i+1}^m}{(\triangle x)^2} + \frac{\theta_{i-1}^{m+1} - 2\theta_i^{m+1} + \theta_{i+1}^{m+1}}{(\triangle x)^2}\right] + O((\triangle x)^2). \tag{9.44}$$

The differential scheme for the time derivative is the same as given in relation Eq. 9.29. A combination of above two schemes results in the complex space-time CN's scheme

$$-k\theta_{i-1}^{m+1} + 2(1+k)\theta_i^{m+1} - k\theta_{i+1}^{m+1} = k\theta_{i-1}^m + 2(1-k)\theta_i^m + k\theta_{i+1}^m. \tag{9.45}$$

Applying the above scheme, i.e. Eq. 9.45, for the first time step $t = 2$s for three specific cases appropriately for a node position, we obtain

- at the left boundary

$$(1+2k)\theta_1^1 - k\theta_2^1 = kT_0^0 + 2(1-k)T_1^0 + kT_2^0 + kT_0^1, \tag{9.46}$$

- away from the boundary

$$-k\theta_1^1 + 2(1+k)\theta_2^1 - k\theta_3^1 = kT_1^0 + 2(1-k)T_2^0 + kT_3^0, \tag{9.47}$$

- at the right boundary

$$-k\theta_2^1 - 2(1-k)\theta_3^1 = kT_2^0 + 2(1-k)T_3^0 + kT_4^0 + k\theta_4^1. \tag{9.48}$$

Values of RHSs for the above equations are the following

- at the left boundary

$$\begin{aligned} &kT_0^0 + 2(1-k)T_1^0 + kT_2^0 + kT_0^1 = \\ &0.39 \cdot 573 + 2(1-0.39) \cdot 273 + 0.39 \cdot 273 + 0.39 \cdot 573 = 886.50, \end{aligned}$$

- away from the boundary

$$\begin{aligned} &kT_1^0 + 2(1-k)T_2^0 + kT_3^0 = \\ &0.39 \cdot 273 + 2(1-0.39) \cdot 273 + 0.39 \cdot 273 = 546.00, \end{aligned}$$

- at the right boundary

$$\begin{aligned} &kT_2^0 + 2(1-k)T_3^0 + kT_4^0 + k\theta_4^1 = \\ &0.39 \cdot 273 + 2(1-0.39)273 + 0.39 \cdot 373 + 0.39 \cdot 373 = 730.50. \end{aligned}$$

The system of linear equations derived form the CN scheme for the first time step for the domain with three internal nodal points can be written in the matrix form

$$\begin{bmatrix} 2.78 & -0.39 & 0 \\ -0.39 & 2.78 & -0.39 \\ 0 & -0.39 & 2.78 \end{bmatrix} \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \theta_3^1 \end{bmatrix} = \begin{bmatrix} 886.50 \\ 546.00 \\ 730.00 \end{bmatrix}. \tag{9.49}$$

Finally the solution for the first time step is

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \theta_3^1 \end{bmatrix} = \begin{bmatrix} 359.50 \\ 289.40 \\ 303.40 \end{bmatrix}. \tag{9.50}$$

Values of RHSs for the second time step $t = 4$s are the following

- at the left boundary

$$kT_0^1 + 2(1-k)\theta_1^1 + k\theta_1^2 + kT_2^0 + kT_0^2 =$$
$$0.39 \cdot 573 + 2(1-0.39) \cdot 359.50 + 0.39 \cdot 289.40 + 0.39 \cdot 573 = 998.40,$$

- away from the boundary

$$k\theta_1^1 + 2(1-k)\theta_2^1 + k\theta_3^1 =$$
$$0.39 \cdot 359.50 + 2(1-0.39) \cdot 289.40 + 0.39 \cdot 303.20 = 605.40,$$

- at the right boundary

$$k\theta_2^1 + 2(1-k)\theta_3^1 + k\theta_4^0 + k\theta_4^2 =$$
$$0.39 \cdot 289.40 + 2(1-0.39)303.20 + 0.39 \cdot 373 + 0.39 \cdot 373 = 773.70$$

The system of linear equations derived form the CN scheme for the second time step for the domain with three internal nodal points can be written in the matrix form

$$\begin{bmatrix} 2.78 & -0.39 & 0 \\ -0.39 & 2.78 & -0.39 \\ 0 & -0.39 & 2.78 \end{bmatrix} \begin{bmatrix} \theta_1^2 \\ \theta_2^2 \\ \theta_3^2 \end{bmatrix} = \begin{bmatrix} 998.40 \\ 605.40 \\ 773.70 \end{bmatrix}. \tag{9.51}$$

The solution for the second time step is

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \\ \theta_3^2 \end{bmatrix} = \begin{bmatrix} 404.00 \\ 319.80 \\ 323.20 \end{bmatrix}. \tag{9.52}$$

♠

9.4 The finite element method

The finite element method (FEM) is a specific Galerkin method such that the trial and spaces are the same and the residual is orthogonal to all elements in the space. Basis sets of FEM are compactly supported.

The advantages of FEM can be split into two groups: mathematical and computational, and they are listed in the following table:

Type	Advantage
Mathematical	1. The selection of function species support the intuition that the computed solution is close to the exact solution. 2. The Galerkin analysis provides an acceptable approximated solution for well posed partial (or ordinary) differential equations.
Computational	1. The technique supplies the solution not only at grid location but everywhere in the domain. 2. The FEM technique can be applied to very complex domains by using elements that fit domains better than simple uniform or rectangular lattices.

Equations most frequently solved by FEM in science and engineering can be grouped in three classes of problems:

PDE Class	Equation	Specific forms	Meaning of coefficients
Balance equation applied in mechanical engineering	$m\frac{\partial^2 u}{\partial t^2} + c\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}u = f(t)$		m - mass c - dumping k - stiffness u - displacement f - applied load
Laplace - Poisson equation	$-\nabla^2(\kappa\psi) = c$	Heat conduction $\nabla \cdot (\kappa \nabla T) = -q$	κ - thermal conductivity T - temperature q - internal heat generation rate
		Shaft torsion $\nabla^2(\frac{1}{G}\varphi) = -2\theta$	G - shear modulus of elasticity θ - twist per unit length φ - stress function defined by $\tau_{yz} = \frac{\partial \varphi}{\partial x}$, $\tau_{xz} = \frac{\partial \varphi}{\partial y}$
		Flow through porous media $\nabla^2(\kappa H) = -Q$	κ - permeability coefficient H - fluid head Q - internal flow injection rate
		Pressurized membranes $\nabla^2(Th) = -p$	T - membrane tension h - membrane displacement p - pressure imbalance
		Circulation of fluid flow $\nabla^2\varphi = -2\omega$	φ - stream function defined by $u = \frac{\partial \varphi}{\partial y}$, $v = \frac{\partial \varphi}{\partial x}$ ω - vorticity
		Ideal fluid flow $\nabla^2\varphi = 0$ $\nabla^2\psi = 0$	φ - potential function defined by $u = \frac{\partial \varphi}{\partial y}$, $v = \frac{\partial \varphi}{\partial x}$ φ - stream function defined by $u = \frac{\partial \psi}{\partial y}$, $v = \frac{\partial \psi}{\partial x}$
		Electrostatic field $\nabla^2(\varepsilon V) = -\rho$	ε - permittivity V - voltage (potential) ρ - charge density
		Diffusion $\nabla^2(Dc) = Q$	D - diffusion constant c - moisture concentration Q - production rate
Generalized fluid transport equation	$\frac{\partial \rho\varphi}{\partial t} + \nabla \cdot (\rho v \varphi) - \nabla \cdot (d\nabla\varphi) = R$		

Chapter 10

Appendices

10.1 Spectral norms

Assuming that the matrix **A** is real throughout, the properties of the spectral norms are the following:

- i.e. Wilkinson page 290

10.2 Max-norms

Bibliography

[1] AlAmer.....

[2] http://www.physics.arizona.edu/ restrepo/475B/Notes/source/node...

[3] C. de Boor, Divided differences, in: Surveys in Approximation Theory, 2005, **1**, 46-69,

[4] J. Butcher, Numerical equations for ordinary differential equations, J. Wiley, Chichester, 2003,

[5] J. Butcher, Runge-Kutta methods for ordinary differential equations, COE Workshop on Numerical Analysis, Kyushiu University, May 2005,

[6] R.L. Burden and J.D. Faires, Numerical Analysis, 5-th edition, PWS-Kent Pub. Co., Boston, 1993,

[7] Y.F. Chang and G. Corliss, ATOMFT: Solving ODEs and DAEs using Taylor series, Comp. Mth. Appl., 1994, **28**, 209-233,

[8] S-Ch. Choi, Iterative methods for singular linear equations and least-squares problems, PhD Thesis, Stanford University, ICME, 2006,

[9] W.J. Cody, J.T. Coonen, D.M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F.N. Ris, and D. Stevenson, A proposed radix-and-word-length-independent standard for floating-point arithmetics, IEEE Micro, 1984, **4**, 86-100, August,

[10] L. Collatz, The Numerical Treatmant of Differential Equation, 3rd edition, Springer, Berlin, 1996,

[11] R. Courant, Variational methods for solution of problems of equilibrium and vibrations, Bull. Amer. Math. Soc. 1943, **49**, 1-23,

[12] G. Dahlquist and A. Bjorck, Numerical Methods, Prentice-Hall, Inc., Englewood Clifs, NJ, 2003,

[13] http://en.wikipedia.org/wiki/Exponent_bias,

[14] http://www.cs.wisc.edu/ smoler/x86text/lect.notes/arith.flpt.html,

[15] I.Th. Famelis, S.N. Papakostas and Ch. Tsitouras, Symbolic derivation of Runge-Kutta order condition, J. Symbol. Comput., 2004, **37**, 311-327,

[16] J.E. Gentle, Numerical linear algebra for applications in statistics, Springer,

[17] C.F. Gerald and P.O. Wheatley, Applied Numerical Analysis, Addison-Wesley Pub. Co., Reading MA, 4th edition, 1989,

[18] D. Goldberg, Appendix D, What every computer scientist should know about floating-point arithmetic, Computing Surveys, 1991, March,

[19] G.H. Golub and C.F. Van Loan, Matrix computations, Johns Hopkins University Press, 1989,

[20] W.W. Hager, Condition estimates, SIAM J. Sci. Stat. Comput. 1984, **5**, 11-316,

[21] M.R. Hestenes and E. Stiefel, Methods of Conjugate gradients for solving linear equations, Journal of Research of the National Bureau of Standards, 1952, **49**, 409-436,

[22] N.J. Higham, A survey of condition number estimation for triangular matrices, SIAM Review 1987, **29**, 575-596,

[23] N.J. Higham, Experience with matrix norm estimation, SIAM J. Sci. Stat. Comput. 1990, **11**, 804-809,

[24] F.B. Hildebrand, Introduction to numerical Analysis, 2nd edition, McGraw-Hill, New York, 1987,

[25] W. Kahan, IEEE Standard 754 for binary floating-point arithmetic, Lecture Notes on the Status of IEEE 754, 1996, May,

[26] W. Kahan, Gauss-Seidel methods of solving large systems of linear equations, PhD Thesis, University of Toronto, 1958,

[27] A. Kaw, http://numericalmethods.eng.usf.edu,

[28] I.R. Khan and R. Ohba, Closed-form expressions for the finite difference approximations of first and higher derivatives based on Taylor series, J. Comput. Appl. Math., 1999, **107**, 179-193,

[29] I.R. Khan and R. Ohba, New finite difference formulas for numerical differentiation, J. Comput. Appl. Math., 2000, **126**, 269-276,

[30] I.R. Khan and R. Ohba, Taylor series based finite difference approximations of higher-degree derivatives, J. Comput. Appl. Math., 2003, **154**,, 115-124,

[31] D. Kidner, M. Dorey, and D. Smith, What's the point? Interpolation and extrapolation with a regular grid DEM. International Conference on Geo-computation, 1999, Fridriksburg, VA, USA, available on: http://www.geovista.psu.edu/sites/geocomp99/Gc99/082/gc_082.htm,

[32] E. Kreyszig, Advanced Engineering Mathematics, 8-th edition, Wiley, New yourk, 2001,

[33] V. Lefervre, J-M. Muller, Toward correctly rounded transcendentals, IEEE Transactions on Computers, 1998, **47**, 1235-1243,

[34] J. Liesen and P. Tichy, Convergence analysis of Krylov subspace methods, GAMM Mittailungen, 2004, **27**, Heft 2,

[35] C.C. Paige and M.A. Saunders, Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal., 1975, **12**, 617-629.

[36] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, Numerical recipies in FORTRAN, Cambridge University Press, 1992,

[37] A.J. Roberts, Simple and fast multigrid solution of poisson's equation using diagonally oriented grids, ANZIAM J. 2001, **43 E**, E1-E36,

[38] Y. Saad, Krylov subspace methods for solving large unsymmetric linear systems, Math. Comp., 1981, **37**, 105-126,

[39] Y. Saad, The Lanczos biorthogonalization algorithm and other oblique projection methods for solvinglarge unsymmetric systems, SIAM J. Numer. Anal., 1982, **19**, 485-506,

[40] Y. Saad and M.H. Schultz, GMRES: a generalized minimal residual algorithmfor solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 1986, **7**, 856-869.

[41] S. Salleh, A.Y. Zomaya and S.A, Bakar, Computing for numerical methods using Visual C++, Wiley-Interscience, 2007,

[42] A. Sankar, Smoothed analysis of Gaussian elimination, PhD. Thesis, Massachusetts Institute of Technology, February 2004,

[43] M. Schatzman, Numerical analysis: a mathematical introduction, 2002, chapters: 4 and 6, Clarendon Press, Oxford,

[44] H.R. Schwarz, Numerische mathematik, B.G. Teubner, zweite Auflage, 1988,

[45] J.R. Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213,

[46] H.D. Simon, Direct sparse matrix methods, In J.C. Almond and D.M. Young, editors, Modern Numerical Algorithms for Supercomputers, 325-444, Austin, 1989, The University of Texas at Austin, Center for High Performance Computing,

[47] A. van der Sluis and H.A. van der Vorst, Num. Math. 1986, **48**, 543-560,

[48] D.M. Strong, Iterative methods for solving $\mathbf{Ax} = \mathbf{b}$, Journal of Online Mathematics and its Applications, http://mathdl.maa.org/mathDL/

[49] G. Szegö, Orthogonal polynomials, Providence, Amer. Math. Soc., 1975, 329-332,

[50] L.N. Trefethen and D. Bau, Numerical linear algebra, SIAM, 1997,

[51] H.A. van der Vorst, The converegence behaviour preconditioned CG and CG-S in the presence of rounding errors, Lecture Notes in Mathematics, 1990, **1457**, 126-136,

[52] H.A. van der Vorst, Iterative methods for large linear systems, vorst@math.uu.nl,

[53] E. Waring, Problems concerning interpolations, Philosophical Transactions of the royal society of london, 1779, **69**, 59-67,

[54] D. Watkins, Fundamentals of matrix computations, Wiley, 1991,

[55] E.W. Weisstein, Linear system of equations, from MathWorld, A Wolfram Web Resource,

[56] E.W. Weisstein, Pivoting, from MathWorld, A Wolfram Web Resource, http://mathworld.wolfram.com/LinearSystemofEquations.html,

[57] http://en.wikipedia.org/wiki/Generalized _minimal_ residual_ method, Feb., 2008,

[58] http://en.wikipedia.org/wiki/Arnoldi_iteration, 2008,

[59] J.H. Wilkinson, Error analysis of direct methods of matrix inversion, J. Assoc. Comput. Mach., 1961, **8**, 261-330,

[60] http://www.physics.utah.edu/∼detar/phys6720/handouts/interpolation

[61] I. Zavorin, Spectral factorization of the Krylov matrix and convergence of GMRES, http://hdl.handle.net/1903/1168, 2001,