MATH 575A Numerical Analysis
Fall 2019 class notes

Misha Stepanov
Department of Mathematics and Program in Applied Mathematics
University of Arizona, Tucson, AZ 85721, USA

Part I
Rounding errors and numerical stability

Very often the study of numerical analysis starts with the topic of rounding errors. These class notes is not an exception.[1] Suggested reading: [TrBa97, Lecs. 12–15] and [Hig02, Chs. 1, 2].

1 Floating-point arithmetic

One can come up with 3 ways to represent information: I) analogue, II) digital, and III) algorithmical. Examples of information in analogue form would be: a length of a wooden stick, printed photo, audio tape recording,[2] a mass of an object. Examples of information in digital form are: post address, a text in a book, computer file, *etc.* Algorithmic way of presenting/storing information instead of giving information *per se*, just operates with a recipe how to produce the information. Information in digital form is easier to handle.[3] In computer the information is presented in bits that could be in 2 states: 0 and 1 (or low and high voltage). Having just 2 states makes circuit design easier.[4]

Let us say we want to devote N bits to store a [real] number. We have 2^N possible states, and thus no more than 2^N different numbers to play with. How would we spend these N bits? A possible [linear] way is to have numbers $\langle 1-2^{N-1}, 2-2^{N-1}, \ldots, -1, 0, 1, 2, \ldots, 2^{N-1}-1, 2^{N-1}\rangle / 2^E$, where E is not far from $N/2$. This way we have both large (about 2^{N-E}) and small (about 2^{-E}) numbers. It is easy to proceed with addition in this system. Long multiplication (more elaborate than addition) can be used. The problem is that largest and smallest non-zero numbers in this system are not that very large or small.[5]

Logarithmic quantization, with numbers $\pm(1+\varepsilon)^E$, where $\varepsilon \ll 1$ and $E \lesssim 2^N$. This way we could get very large and very small numbers easily, even with small relative change between the

[1] Here is an essay that advocates the point of view that numerical analysis is far from being just the study of rounding errors: L. N. Threfethen, *The definition of numerical analysis*, 1992.

[2] Later there were devices which wrote and read audio or video information on a magnetic tape in digital form.

[3] Copying without distortion is possible. Error correction is much more effective.

[4] For any base or radix B one can introduce a so called "radix economy" $E(B) := B/\ln B$. It is roughly the number of B-digits needed to represent a [large] number N multiplied by B (number of different B-digits) and divided by $\ln N$. Ternary system is more radix economic, $E(3) < E(2)$, and there were some ternary computers built.

[5] In many image formats (JPEG, PNG, *etc.*) a typical color depth is $N = 8$ bits. Often in digital photos some areas are solid white due to clipping, while some others are too dark. In Compact Disc (CD) $N = 16$ is used. This seems to be unsatisfactory for some, which gave rise to DVD-Audio (up to $N = 24$) and Super Audio CD (SACD).

consequtive numbers. It is very easy to multiply numberes in it (but addition becomes even harder than long multiplication in linear quantization).

A hybrid approach, combining the two, is to spend some bits (*fraction* or *mantissa*) in linear way, while some others (*exponent*) in logarithmic. One bit is reserved to store the sign (positive or negative) of the number. This is so called floating point numerical system. Eventually it was standardized as IEEE 754 [IEEE85]. Fraction part allows easy addition and [long] multiplication, while dynamic range is huge due to the exponent part.

Consider the following hypothetical floating-point numerical system $\mathbf{FP}_N$: the number in it is either 0, or a floating-point number with the fraction made of N bits:[6]

$$\pm 2^E \cdot \big(1.b_1 b_2 ... b_N\big)_2 = \pm 2^E \left(1 + \sum_{n=1}^{N} b_n 2^{-n}\right)$$

The exponent E could be any integer number, so $\mathbf{FP}_N$ has arbitrarily small or large numbers (one can think of 0 having exponent $E = -\infty$). Not considering possibility of overflows/underflows, IEEE 754 single and double precision formats correspond to $\mathbf{FP}_{23}$ and $\mathbf{FP}_{52}$.

There is an obvious mapping $\mathrm{inj} : \mathbf{FP}_N \to \mathbf{R}$. Let us construct the "best numerical representation" mapping $\mathrm{num} : \mathbf{R} \to \mathbf{FP}_N$ by choosing $\mathrm{num}(x)$ to be the closest to x number from $\mathbf{FP}_N$ (in cases where two numbers from $\mathbf{FP}_N$ are at the same minimal possible distance, [to imitate "round half to even" rule] the number with larger E or (if the exponents are the same) with $b_N = 0$ is chosen). Obviously, $\mathrm{num}\big(\mathrm{inj}(x)\big) = x$ for any $x \in \mathbf{FP}_N$. The reverse is only approximately true:

Theorem 1: For any $x \in \mathbf{R}$ we have $\mathrm{inj}\big(\mathrm{num}(x)\big) = x(1+\varepsilon)$, where $|\varepsilon| \le \varepsilon_{\text{machine}} := 2^{-(N+1)}$. In other words, it is possible to represent real numbers in floating-point numerical system $\mathbf{FP}_N$ with small (*machine epsilon* or *unit roundoff* $\varepsilon_{\text{machine}}$) relative error.

Proof: We have $\mathrm{num}(2x) = 2\,\mathrm{num}(x)$, so without loss of generality we can assume that $1 \le x < 2$. Within this interval $\mathbf{FP}_N$ contains numbers $1 + k2^{-N}$, where $0 \le k < 2^N$ and k is integer. Then $|\mathrm{num}(x) - x| \le 2^{-(N+1)}$, and $\varepsilon = (\mathrm{num}(x) - x)/x$.[7]

IEEE double precision arithmetic corresponds to $N = 52$, and $\varepsilon_{\text{machine}} = 2^{-53} \approx 1.11 \cdot 10^{-16}$, *i.e.*, it deals with numbers having approximately 16 decimal digits.[8]

Analysis of numerical algorithms is impossible without concrete knowledge of how basic arithmetic operations [or built-in functions like $\exp(\cdot)$ or $\sin(\cdot)$] are implemented. In hypothetical implementation, that we will call *virtual*, the numerical arithmetic operations of addition, multiplication, and division are defined as

$$x \oplus y := \mathrm{num}\big(\mathrm{inj}(x) + \mathrm{inj}(y)\big), \quad x \odot y := \mathrm{num}\big(\mathrm{inj}(x) \cdot \mathrm{inj}(y)\big), \quad x \oslash y := \mathrm{num}\big(\mathrm{inj}(x) / \mathrm{inj}(y)\big)$$

The operation of subtration $\ominus$ is defined similarly or, *e.g.*, as $x \ominus y := x \oplus (-y)$, where $(-y)$ is number y with the sign bit being flipped. For built-in functions we would assume $f(x) = \mathrm{num}\big(f\big(\mathrm{inj}(x)\big)\big)$.

This way defined arithmetic operations satisfy the following fundamental property:

$$\text{for all } x, y \in \mathbf{FP}_N \text{ we have } x \circledast y = \big(\mathrm{inj}(x) * \mathrm{inj}(y)\big)(1+\varepsilon), \text{ where } |\varepsilon| \le \varepsilon_{\text{machine}} \tag{1}$$

[6] One can come up with a floating-point numerical system with any *base* or *radix*. An early computer ENIAC was operating with [up to 20 digits] decimal numbers.

[7] For brevity, the mapping "inj" was skipped in several places.

[8] MATLAB® defines "`eps`" as $2\varepsilon_{\text{machine}} \approx 2.22 \cdot 10^{-16}$.

```
[...]/teaching/2019-4/math_575a/notes/C$ cat print_bits.c
#include <stdio.h>
void print_bits(double x, char* s) { int i; unsigned char c;
  for (i = 0; i < 8; i++) for (c = 128; c != 0; c >>= 1)
    if (*((char *)(&x) + 7 - i) & c) printf("1"); else printf("0");
  printf("  %s\n", s); }
int main() { int i; double x, x52;
  print_bits(0., " 0");
  print_bits(1., " 1");
  print_bits(-1., "-1");
  print_bits(0.5, "1/2");
  print_bits(0.25, "1/4");
  print_bits(1. / 3., "1/3");
  print_bits(2., " 2");
  print_bits(4., " 4");
  print_bits(1.5, " 1 + 1/2");
  print_bits(1. + 1. / 1024., " 1 + 2^(-10)");
  x52 = 1.; for (i = 0; i < 52; i++) x52 /= 2.;
  print_bits(1. + x52, " 1 + 2^(-52)");
  print_bits(1. - x52, " 1 - 2^(-52)");
  print_bits(1. + (2. * x52), " 1 + 2^(-51)");
  print_bits(2. + (2. * x52), " 2 + 2^(-51)");
  x = 1.; for (i = 0; i < 1022; i++) x /= 2.; print_bits(x, " 2^(-1022)");
  print_bits(x * x52, " 2^(-1074)");
printf("|\\___  ____/\\_________________________  _______________________/\n");
printf("|    \\/                               \\/\n");
printf("sign  exponent (11 bits)               fraction (52 bits)\n");
  return 0; }
[...]/teaching/2019-4/math_575a/notes/C$ cc print_bits.c
[...]/teaching/2019-4/math_575a/notes/C$ ./a.out
0000000000000000000000000000000000000000000000000000000000000000   0
0011111111110000000000000000000000000000000000000000000000000000   1
1011111111110000000000000000000000000000000000000000000000000000  -1
0011111111100000000000000000000000000000000000000000000000000000  1/2
0011111111010000000000000000000000000000000000000000000000000000  1/4
0011111111010101010101010101010101010101010101010101010101010101  1/3
0100000000000000000000000000000000000000000000000000000000000000   2
0100000000010000000000000000000000000000000000000000000000000000   4
0011111111111000000000000000000000000000000000000000000000000000   1 + 1/2
0011111111110000000001000000000000000000000000000000000000000000   1 + 2^(-10)
0011111111110000000000000000000000000000000000000000000000000001   1 + 2^(-52)
0011111111101111111111111111111111111111111111111111111111111110   1 - 2^(-52)
0011111111110000000000000000000000000000000000000000000000000010   1 + 2^(-51)
0100000000000000000000000000000000000000000000000000000000000001   2 + 2^(-51)
0000000000010000000000000000000000000000000000000000000000000000   2^(-1022)
0000000000000000000000000000000000000000000000000000000000000001   2^(-1074)
|\___  ____/\_________________________  _______________________/
|    \/                               \/
sign  exponent (11 bits)               fraction (52 bits)
[...]/teaching/2019-4/math_575a/notes/C$
```

where $*$ is any of four $+, -, \cdot, /$ operations, with $\oplus, \ominus, \odot, \oslash$ being their floating point analogues. We will call $\varepsilon_{\text{machine}}$-*valid* any floating-point arithmetic implementation, such that the numerical arithmetic operations of addition, subtraction, multiplication, and division do satisfy (1).

Problems and exercises

1. Find the minimal positive integer n in $\mathbf{FP}_N$ such that $n+1$ is already not in $\mathbf{FP}_N$.
2. Prove that if $\operatorname{num}(x) = 0$, then $x = 0$.
3. Prove that if $x \odot y = 0$, then at least one of x and y is equal to 0 ($\mathbf{FP}_N$ has no zero divisors).
4. Obviously, [in virtual implementation] addition $\oplus$ and multiplication $\odot$ operations in $\mathbf{FP}_N$ are commutative. Also 0 and 1 are indeed neutral elements for addition and multiplication, respectively [why?]. Prove of disprove by counterexample that addition $\oplus$ is (a) associative, (b) has opposites; multiplication $\odot$ is (c) associative, (d) has inverses; and (e) distibutive property holds.
5. Plot the polynomial $P(x) = (x-4)^6$ computed as

$$\Big(\Big(\big(((x^6 - 24x^5) + 240x^4) - 1280x^3\big) + 3840x^2\Big) - 6144x\Big) + 4096$$

for $x = 3.98, 3.98002, 3.98004, \ldots, 4.02$. Explain the quantization of the values of $P(x)$. Also plot $P(x)$ computed as $(x-4)^6$.

2 Stable and unstable numerical algorithms

Definition 2.1: A numerical *algorithm* is a mapping $F : \mathcal{X} \subseteq \mathbf{R}^m \to \mathbf{FP}_N^n$ from m-dimensional input data to n-dimensional output, with the way how the mapping is computed [including how the input is processed and how the output is interpreted] being clearly described. It is an attempt to numerically simulate an ideal mapping $F_{\text{exact}} : \mathcal{X} \to \mathbf{R}^n$ (which is the underlying [mathematical] *problem*, as in [TrBa97, Lec. 12]).

Example 2.1: Consider, *e.g.*, the problem of finding the point on a circle $x^2 + y^2 = 1$ that is the closest to a given point (x_0, y_0). A possible algorithm for solving it would be a mapping $(x_0, y_0) \in \mathbf{R}^2 \backslash (0,0) \mapsto \big(x_0/\sqrt{x_0^2 + y_0^2},\, y_0/\sqrt{x_0^2 + y_0^2}\big)$, here $m = n = 2$. Another algorithm would be a mapping $(x_0, y_0) \mapsto \varphi := \mathtt{atan2}(y_0, x_0)$, with $n = 1$, and the output is interpreted as the point $(\cos\varphi, \sin\varphi)$.[9]

Let us say, we want to calculate $F(\boldsymbol{x})$, with $\boldsymbol{x} \in \mathbf{R}^m$ being the input. Obviously, the components of $\boldsymbol{x}$ are not necessarily exactly representable in $\mathbf{FP}_N$. Inevitably one can expect a relative error about $\varepsilon_{\text{machine}}$ just from entering $\boldsymbol{x}$ into a computer. Instead of calculating $F(\boldsymbol{x})$, we are computing $F(\boldsymbol{x} + \Delta\boldsymbol{x})$, where $\Delta\boldsymbol{x}$, which is called *backward error*, could be just a round off error in $\boldsymbol{x}$.[10]

Definition 2.2: An algorithm F is called *backward stable*, if for any input $\boldsymbol{x}$ we have $F(\boldsymbol{x}) = F_{\text{exact}}(\boldsymbol{x} + \boldsymbol{\eta})$ with $\|\boldsymbol{\eta}\| \sim \varepsilon_{\text{machine}} \|\boldsymbol{x}\|$.[11] In other words, the output $F(\boldsymbol{x})$ is the *exact* answer to the problem with input relatively very close to $\boldsymbol{x}$.[12]

An algorithm being backward stable is the best one can hope for, as backward error is unavoidable. In Example 2.1 the `atan2` version is backward stable, while $m = n = 2$ version is not, as the

---

[9] `atan2` in C — arc tangent function of two variables.

[10] Sometimes the input is known only up to some uncertainty. In many cases, *e.g.*, weather prediction, people compute $F(\boldsymbol{x} + \Delta\boldsymbol{x})$, with several $\Delta\boldsymbol{x}$ of the order of the uncertainty, to estimate the resulting uncertainty in $F(\boldsymbol{x})$.

[11] Here $\|\cdot\|$ is a magnitude measured somehow. See definition of norm in MATH 527 (theory) course, and also Sec 4.1.

[12] It is assumed that all floating point arithmetical operations inside the algorithm F are done in [some] $\varepsilon_{\text{machine}}$-valid implementation. One can introduce a notion of *virtually backward stable* algorithm, if virtual implementation is used.

resulted point may be not exactly on the unit circle (whenever the exact answer lies in a lower than n dimensional submanifold, it is almost hopeless for the numerical answer to be in it and for the algorithm to be backward stable).

Definition 2.3: An algorithm F is called *stable*, if for any [reasonable of feasible] input $\boldsymbol{x}$ we have $F(\boldsymbol{x}) = F_{\text{exact}}(\boldsymbol{x}+\boldsymbol{\eta})+\boldsymbol{\varepsilon}$ with $\|\boldsymbol{\eta}\| \sim \varepsilon_{\text{machine}}\|\boldsymbol{x}\|$ and $\|\boldsymbol{\varepsilon}\| \sim \varepsilon_{\text{machine}}\|F(\boldsymbol{x})\|$. In other words, the output $F(\boldsymbol{x})$ is relatively close answer to the exact one for the problem with input very close to $\boldsymbol{x}$.

Any backward stable algorithm is stable, but not *vice versa*. Both algorithms in Example 2.1 are stable. Both definitions of backward stable and stable algorithms are about asymptotic behavior of errors in the limit $\varepsilon_{\text{machine}} \to 0$. Namely, for sufficiently small $\varepsilon_{\text{machine}}$, the relative errors are bounded by some constant multiplied by $\varepsilon_{\text{machine}}$.[13]

Example 2.2: computing $\exp(x)$ using Taylor series

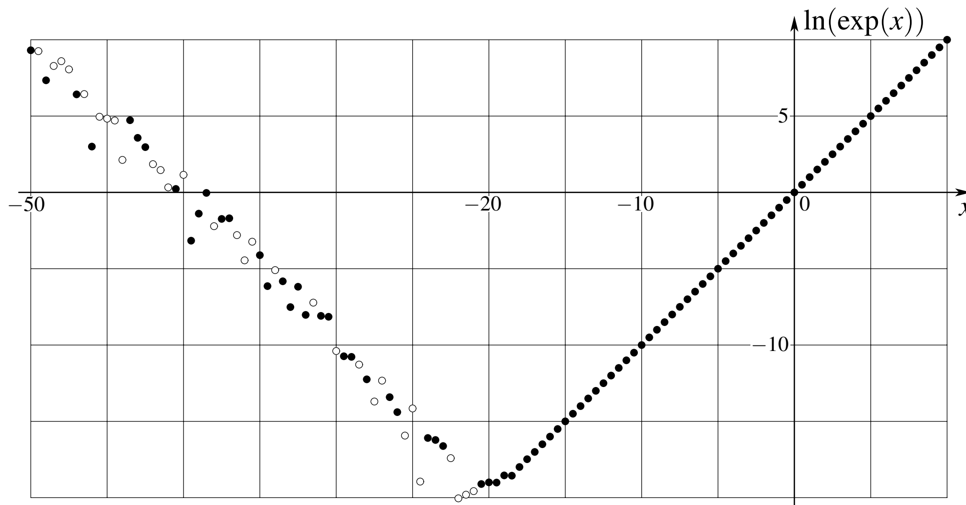
A typical scenario in which numerical accuracy is lost is when a small number is obtained as a sum/difference of almost opposite/same numbers.

Algorithm $\exp(x)\wedge$: Input: $x \in \mathbf{R}$. Output: $\exp(x)$, computed by summing $\sum_{n=0}^{\infty} x^n/n!$ from left to right, stopping when adding a term does not change the partial sum.

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat exp.py
from math import exp
for x in [-50., -40., -30., -20., -15., -10., -2., -1., 0., 1., 10., 30.]:
    taylor, sum, n, xnf = 0., 1., 1, x
    while taylor != sum:
        taylor, sum, n, xnf = sum, sum + xnf, n + 1, x * xnf / (n + 1.)
    print('exp({0: 3.0f}.) = {1: .15e} = {2:.15e}'.format(x, taylor, exp(x)))
[...]/teaching/2019-4/math_575a/notes/Python$ python exp.py
exp(-50.) =  1.107293338289197e+04 = 1.928749847963918e-22
exp(-40.) = -3.165731894063124e+00 = 4.248354255291589e-18
exp(-30.) = -3.066812356356220e-05 = 9.357622968840175e-14
exp(-20.) =  5.621884472130418e-09 = 2.061153622438558e-09
exp(-15.) =  3.059094197302007e-07 = 3.059023205018258e-07
exp(-10.) =  4.539992962303128e-05 = 4.539992976248485e-05
exp( -2.) =  1.353352832366128e-01 = 1.353352832366127e-01
exp( -1.) =  3.678794411714424e-01 = 3.678794411714423e-01
exp(  0.) =  1.000000000000000e+00 = 1.000000000000000e+00
exp(  1.) =  2.718281828459046e+00 = 2.718281828459045e+00
exp( 10.) =  2.202646579480671e+04 = 2.202646579480672e+04
exp( 30.) =  1.068647458152447e+13 = 1.068647458152446e+13
[...]/teaching/2019-4/math_575a/notes/Python$
```

The series $\sum_{n=0}^{\infty} x^n/n!$ converges for any x. If the operations with numbers are done exactly, then the algorithm (although never finishing, as the whole series needs to be went through) would output an exact answer. When x is large and negative, some terms in the series are large, although the answer is small. The small answer $\exp(-|x|)$ is produced as an almost cancellation of as large as $\exp(|x|)$ numbers. Whenever $|x|$ is so large, that $\exp(-|x|)$ is of the order of $2^{-N}\exp(|x|)$, one should not trust any digits of an outputted answer. For IEEE double precision ($N = 52$) this happens for $x \sim -\ln(2^{52})/2 \approx -18$. The algorithm is heavily unstable when applied to large negative x.

[13] For the algorithm to be practically useful, it doesn't necessarily have to be stable. Let us call an algorithm ζ-*semistable* with $0 < \zeta \leq 1$, if $\lim_{\varepsilon_{\text{machine}}\to 0^+} \ln(\text{relative errors})/\ln(\varepsilon_{\text{machine}}) = \zeta$. The Algorithm $\zeta(3)\wedge$ from Example 2.3 is $(2/3)$-semistable, while Algorithm $\pi\cap$ from Example 2.4 and possible algorithm from Example 3.2 are $(1/2)$-semistable. The Algorithm $\exp(1)\wedge$ from Example 2.2 that computes $\mathrm{e} \approx 2.72$ is 1-semistable, but [strictly speaking] is not stable.



Even when applied for $x > 0$ only (*i.e.*, when there are no sign changes in the term of the series), the algorithm is not [strictly speaking] backward stable. The larger is N, the larger is the number of terms that one needs to sum (although here it slowly grows with N, as $n!$ grows very fast). When the partial sum is already not much smaller than the answer, each addition potentially introduces a relative error of the order of $\varepsilon_{\text{machine}}$.

In practice, for large positive x, one would expect [why?] the algorithm to produce $\exp(x) \cdot \big(1 + O(x^{1/2}\varepsilon_{\text{machine}})\big)$, or maybe even $\exp(x) \cdot \big(1 + O(x^{1/4}\varepsilon_{\text{machine}})\big)$.

Example 2.3: computing $\zeta(3)$ by summing the series

Whenever one has a bunch of numbers to sum, it is more safe to add small numbers together first, and add large numbers later.

Let us consider two algorithms for computing

$$\text{Apéry's}^{14} \text{ constant } \zeta(3) := \sum_{n=1}^{\infty} \frac{1}{n^3} = 1.202056903159594285...$$

Algorithm $\zeta(3)\wedge$: Input: none. Output: $\zeta(3)$, computed by summing $\sum_{n=1}^{\infty} 1/n^3$ from left to right, stopping when adding a term does not change the partial sum.

Algorithm $\zeta(3)\vee$: Input: none. Output: $\zeta(3)$, computed by summing $\sum_{n=1}^{\infty} 1/n^3$ from right to left, starting at some sufficiently large n.

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat zeta3.py
zeta3, sum, n = 1., 0., 1
while zeta3 != sum:
    zeta3, sum, n = sum, sum + 1. / float(n**3), n + 1
print(' forward summation: zeta(3) = {0:.15f}'.format(zeta3))
zeta3, n = 0., 25000000
while n > 0:
    zeta3, n = zeta3 + 1. / float(n**3), n - 1
print('backward summation: zeta(3) = {0:.15f}'.format(zeta3))
[...]/teaching/2019-4/math_575a/notes/Python$ python3 zeta3.py
 forward summation: zeta(3) = 1.202056903150321
backward summation: zeta(3) = 1.202056903159594
[...]/teaching/2019-4/math_575a/notes/Python$
```

[14] R. Apéry, *Irrationalité de* $\zeta(2)$ *et* $\zeta(3)$, Astérisque **61**, 11–13 (1979). [Number $\zeta(3)$ is proved to be irrational.]

As it is not possible to represent the number $\zeta(3)$ exactly in $\mathbf{FP}_N$ [for any finite N], *any* numerical algorithm for computing $\zeta(3)$ *can not* be backward stable. The answer in Algorithm $\zeta(3)\wedge$ ignores the tail of the series starting at about $n_* \sim 1/\varepsilon_{\text{machine}}^{1/3}$, and the value of the ignored tail is about $1/n_*^2 \sim \varepsilon_{\text{machine}}^{2/3}$ — about $2/3$ of the significant digits should be correct.

Example 2.4: computing π by maximizing $\sin(\cdot)$

Another common source of accuracy loss is determining a quantity from a function which is not very sensitive to it, *e.g.*, finding position of an extremum of a function.

Algorithm $\pi\cap$: Input: none. Output: π, determined as 2 multiplied by the position of the first maximum of $\sin(x)$. The latter computed as the largest x such that $\sin(x)$ still grows locally:

```
[...]/teaching/2019-4/math_575a/notes/C$ cat find_pi.c
#include <stdio.h>
#include <math.h>
int main() { double pi2, step;
  for (pi2 = 0., step = 1.; pi2 + step != pi2; )
    if ((sin(pi2 + 2. * step) > sin(pi2 + step))
     && (sin(pi2 + step) > sin(pi2)))
      pi2 = pi2 + step; else step *= 0.5;
  printf("computed pi = %22.16e\n", 2. * pi2);
  printf("         pi = %22.16e\n", M_PI);
  return 0; }
[...]/teaching/2019-4/math_575a/notes/C$ cc find_pi.c -lm
[...]/teaching/2019-4/math_575a/notes/C$ ./a.out
computed pi = 3.1415926218032837e+00
         pi = 3.1415926535897931e+00
[...]/teaching/2019-4/math_575a/notes/C$
```

Here π is found from the maximization of $\sin(x)$ near $x = \pi/2$. We have $\sin(x) \approx 1 - (x-\pi/2)^2/2$ there. Whenever $(x-\pi/2)^2 \sim 2^{-N}$, the difference of $\sin(x)$ from 1 is too small for $\mathbf{FP}_N$ to handle. At the distance about $2^{-N/2}$ from $\pi/2$ the numerical function $\sin(x)$ stops to change, which causes only about half of the digits in computed value of π to be correct.

```
[...]/teaching/2019-4/math_575a/notes/C$ cat sin_near_1.c
#include <stdio.h>
#include <math.h>
double x53;
double f(double x) { return (x53 * (1. - sin(x))); }
int main() { int i;
  for (x53 = 1., i = 0; i < 53; i++) x53 *= 2.;
  printf("f(x) = 2^(53) * (1 - sin(x))\n");
  printf("f(pi / 2 - 2.e-8)   = %6.4f\n", f(M_PI / 2. - 2.e-8));
  printf("f(pi / 2 - 1.8e-8)  = %6.4f\n", f(M_PI / 2. - 1.8e-8));
  printf("f(pi / 2 - 1.1e-8)  = %6.4f\n", f(M_PI / 2. - 1.1e-8));
  printf("f(pi / 2 - 1.05e-8) = %6.4f\n", f(M_PI / 2. - 1.05e-8));
  printf("f(pi / 2 - 1.e-8)   = %6.4f\n", f(M_PI / 2. - 1.e-8));
  printf("f(pi / 2)           = %6.4f\n", f(M_PI / 2.));
  printf("f(pi / 2 + 1.e-8)   = %6.4f\n", f(M_PI / 2. + 1.e-8));
  return 0; }
[...]/teaching/2019-4/math_575a/notes/C$ cc sin_near_1.c -lm
[...]/teaching/2019-4/math_575a/notes/C$ ./a.out
f(x) = 2^(53) * (1 - sin(x))
f(pi / 2 - 2.e-8)   = 2.0000
```

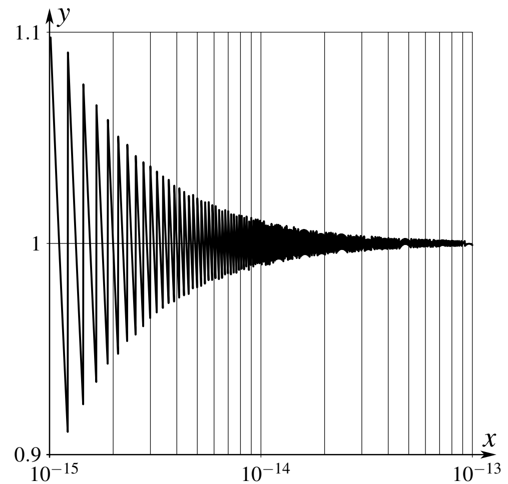
```
f(pi / 2 - 1.8e-8)   = 1.0000
f(pi / 2 - 1.1e-8)   = 1.0000
f(pi / 2 - 1.05e-8) = 0.0000
f(pi / 2 - 1.e-8)    = 0.0000
f(pi / 2)            = 0.0000
f(pi / 2 + 1.e-8)    = 0.0000
[...]/teaching/2019-4/math_575a/notes/C$
```

IEEE double precision, similar to $\mathbf{FP}_{52}$, contains numbers $1-k\cdot 2^{-53}$ and $1+k\cdot 2^{-52}$ with [not too large] non-negative k necessarily being integer. For up to $|x-\pi/2|<10^{-8}$ this integer number, in order to represent the value of $\sin(x)$ better, is equal to 0.

Problems and exercises

1. Consider the following algorithm: Input: two vectors $\boldsymbol{x},\boldsymbol{y}\in\mathbf{R}^n$. Output: geometric angle between the two vectors, computed as $\arccos\big((\boldsymbol{x}\cdot\boldsymbol{y})/\|\boldsymbol{x}\|\,\|\boldsymbol{y}\|\big)$. The norm $\|\boldsymbol{x}\|$ is calculated as $\sqrt{\boldsymbol{x}\cdot\boldsymbol{x}}$. It this algorithm backward stable, stable but not backward stable, or unstable?

2. The graph on the right is the result of numerical calculation of $(\exp(x)-1)/x$ (being computed as it is written). On the other hand, if the function would be computed as $(\exp(x)-1)/\ln(\exp(x))$, then the result would be very close to 1 (as it should be). Explain the difference between the two cases.[15] [16]



3 Condition number

Consider a numerical algorithm $F:\mathcal{X}\subseteq\mathbf{R}^m\to\mathbf{FP}^n_N$. As there are inevitable rounding off errors in processing the input (and in internal calculations), it is important to realize how sensitive F is to small perturbations of the input. A useful quantity to measure that is [relative] *condition number*, which is defined as

$$\kappa(F,\boldsymbol{x}):=\underbrace{\frac{\|F(\boldsymbol{x}+\Delta\boldsymbol{x})-F_{\text{exact}}(\boldsymbol{x})\|}{\|F_{\text{exact}}(\boldsymbol{x})\|}}_{\text{relative change of output due to perturbation }\Delta\boldsymbol{x}\text{ and numerical errors}}\Bigg/\frac{\overbrace{\|\Delta\boldsymbol{x}\|}^{\text{of the order of }\varepsilon_{\text{machine}}}}{\|\boldsymbol{x}\|},\qquad \kappa(F):=\max_{\boldsymbol{x}\in\mathcal{X}}\kappa(F,\boldsymbol{x})$$

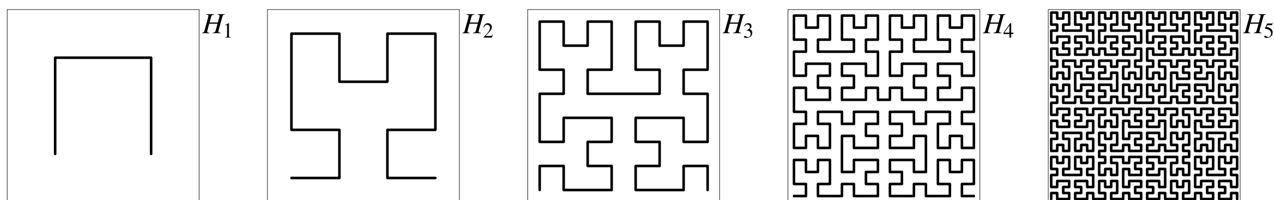
This definition is loose, as how the magnitude $\|\cdot\|$ of relative changes is measured is not specified. If all computations inside the algorithm are assumed to be exact, the condition number is the property of the mapping F_{exact} ifself (or of underlying the [mathematical] *problem*, as in [TrBa97, Lec. 12]). In the limit $\varepsilon_{\text{machine}}\to 0$ and in the case of that mapping being differentiable, the condition number is determined by Jacobian of F_{exact} mapping: $\kappa(F_{\text{exact}},\boldsymbol{x})=\|J(F_{\text{exact}})\big|_{\text{at }\boldsymbol{x}}\|\,\|\boldsymbol{x}\|/\|F_{\text{exact}}(\boldsymbol{x})\|$.

Example 3.1: Consider a function that is a^{th} power a number: $\cdot^a : x\mapsto x^a$, with $m=n=1$ (here $x>0$ and a is fixed). In the limit $\varepsilon_{\text{machine}}\to 0$, we have $\kappa(\cdot^a)=\big((x^a)'/x^a\big)/(1/x)=a$.

[Pathological] **Example 3.2**: Consider the mapping $H:[0,1]\to[0,1]^2$ which is the Hilbert curve.

[15] You can assume that both $\exp(\cdot)$ and $\ln(\cdot)$ are implemented as "numerical f"$(x)=\text{num}\big(f(\text{inj}(x))\big)$.

[16] Look up `expm1` in Python — e^x-1 to full precision, even for small x.



We have $H = \lim_{n\to\infty} H_n$. One has $\kappa(H) \sim \varepsilon_{\text{machine}}^{-1/2}$. Even if implemented in the best possible way, the algorithm would produce only half of the answer's signigicant digits right.

Example 3.3: Consider the algorithm $\ominus : (x,y) \in \mathbf{R}^2 \mapsto \big(\mathrm{num}(x) \ominus \mathrm{num}(y)\big) \in \mathbf{FP}_N$, with $m = 2$ and $n = 1$, which calculates the difference between the numbers x and y. We have

$$\kappa(\ominus, x, y) = \frac{\varepsilon_{\text{machine}}|x| + \varepsilon_{\text{machine}}|y|}{|x-y|} \cdot \frac{1}{\varepsilon_{\text{machine}}} = \frac{|x|+|y|}{|x-y|}$$

The condition number is large when x and y are relatively close, *i.e.*, when the result $x - y$ is much smaller than x or y. Addition/subtraction of large numbers resulting in small number could greatly deteriorate the relative accuracy. (See, *e.g.*, Example 2.2.)

Example 3.4: In "Lucky Numbers" part of "Surely, You are joking, Mr. Feynman" an "impossible" task to calculate $\tan(10^{100})$ [in one minute with 10% accuracy, no computer,] is posed. On a computer, if you use standard single or double accuracy floating point numerical system, you have only 8 or 16 signigicant [decimal] digits, while in order to find out where 10^{100} is inside the period of $\tan(\cdot)$, you need to know π with no less than 100 digits.

```
[...]/teaching/2019-4/math_575a/notes/C$ cat size_of.c
#include <stdio.h>
int main() { printf("sizeof(   float   ) = %2lu\n", sizeof(float));
  printf("sizeof(   double  ) = %2lu\n", sizeof(double));
  printf("sizeof(long double) = %2lu\n", sizeof(long double)); return 0; }
[...]/teaching/2019-4/math_575a/notes/C$ cc size_of.c
[...]/teaching/2019-4/math_575a/notes/C$ ./a.out
sizeof(   float   ) =  4
sizeof(   double  ) =  8
sizeof(long double) = 16
[...]/teaching/2019-4/math_575a/notes/C$
```

The C type `long double` is non-standard, and its size could be anything starting at 8 bytes. In 90s personal computers its size was typically 8 bytes. Later it was sometimes 12 bytes. FORTRAN has `real*16` and `complex*32` types, but [if you have concerns about the speed] you may check whether they are natively supported. Even with 16 bytes you get about $30 < 100$ digits. There is a lot of software to work with arbitrary- or multiple-precision, allowing very large integers and floats having plenty of significant digits.[17] Here is $\tan(10^{100})$ being calculated by Wolfram|Alpha and GP/PARI:

```
[...]/teaching/2019-4/math_575a/notes/gp-pari$ gp
[... technical stuff ...]
                  GP/PARI CALCULATOR Version 2.11.1 (released)
[... copyright notice and links ...]
parisize = 8000000, primelimit = 500000, nbthreads = 4
? tan(10^100)
```

[17] GNU MP, UBASIC, mpmath, and many, many more.

```
%1 = 0.4012319619908143541857543436532949583 2
```

```
%1 = 0.40123196199081435418575434365329495832
? tan(10.^100.)
  ***   at top-level: tan(10.^100.)
  ***                 ^-------------
  *** tan: precision too low in mpcosm1.
  ***   Break loop: type 'break' to go back to GP prompt
break> break

? \p 120
   realprecision = 134 significant digits (120 digits displayed)
? tan(10^100)
%2 = 0.401231961990814354185754343653294958323870261129244068319441538116871809822
2119121146726730974932083113492712621181822475
? tan(10.^100.)
%3 = 0.401231961990814354185754343653294315479678760973444893804899408956723411864
4413289517911232749262503335043779452140708 10
? \p 220
   realprecision = 231 significant digits (220 digits displayed)
? tan(10^100)
%4 = 0.401231961990814354185754343653294958323870261129244068319441538116871809822
21191211467267309749320831134927126211818224746837814909172552238624355491746554
57227844401117202350955319498957782439757435921759611849862972786 3
? tan(10.^100.)
%5 = 0.401231961990814354185754343653294958323870261129244068319441538116871809822
21191211467267309749320831134927126211818224746837814920164092668719583351245472 2
6603450075150599142102852532952089197823824500501316663684175312 5
? quit
Goodbye!
[...]/teaching/2019-4/math_575a/notes/gp-pari$
```

The expressions `tan(10^100)` and `tan(10.^100.)` are parsed differently by GP/PARI, and from the output for the latter one can deduce that about 100 last digits are wrong, consistently with $\kappa(\tan(x)) = 2x/\sin(2x) \sim |x|$ and $x = 10^{100}$.

Problems and exercises

1. Write a program that calculates the Hilbert curve $H(x)$, $0 \le x \le 1$ (Example 3.2). Compute $H(1/3)$ and $H(1/\sqrt{2})$ using single and double precision. How many digits are correct?

2. Consider an algorithm $f'_h : \mathbf{R} \to \mathbf{FP}_N$ that estimates the derivative of the function f (which we can compute at any point in stable way) at x as the finite difference $\big(f(x \oplus h) \ominus f(x)\big) \oslash h$. Assuming that $f(x)$ near the point of interest doesn't have any dramatic features (*e.g.*, f, f', f'', ... are of the order of f, f/L, f/L^2, ..., where L is characteristic scale of x) find how $\kappa(f'_h)$ depends on h and $\varepsilon_{\text{machine}}$. Which h would you choose? Estimate $f'(1)$ for $f(x) = 1/(1+x^2)$ using $h = 10^{-n}$, $n = 0, 1, 2, ..., 17$.

Part II
Numerical Linear Algebra

4 Matrices, singular value decomposition (SVD)

4.1 Matrices, vectors, orthogonality, norms

An $m \times n$ *matrix* is a rectangular table of numbers/matrix elements, with m rows and n columns. The matrix could originate from writing down in a neat way the coefficients of the system of m linear equations with n variables; or correspond to a linear transformation $\mathbf{C}^n \to \mathbf{C}^m$.

An $m \times 1$ matrix with just one column is an m-dimensional *vector*. An $m \times n$ matrix could be viewed as an [ordered] collection of n such vectors/columns. A 1×1 matrix $[a]$ could be identified with its only matrix element a as a number.

A matrix element of the matrix $\hat{A}$ ($\hat{B}$, $\hat{\Gamma}$, *etc.*) at i^{th} row and j^{th} column will be denoted as $(\hat{A})_{ij} = a_{ij}$ (b_{ij}, γ_{ij}, ...). We will call a [not necessarily square] $m \times n$ matrix $\hat{A}$ *diagonal*, if $(\hat{A})_{ij} = 0$ whenever $i \neq j$. A diagonal matrix is fully determined by its diagonal matrix elements, *e.g.*, $\hat{A} = \operatorname{diag}(a_{11}, a_{22}, ..., a_{ll})$, where $l = \min(m,n)$. We will denote $m \times n$ *zero matrix* (all matrix elements are zero) as $\hat{O}_{m,n}$, while $\hat{I}_n = \operatorname{diag}(1,1,...,1)$ will stand for $n \times n$ *identity matrix*, with $(\hat{I}_n)_{ij} = \delta_{ij}$.

A *dot product* or *scalar product* (making $\mathbf{C}^m$ an inner product space) of two vectors $\boldsymbol{x} = [x_i]$ and $\boldsymbol{y} = [y_i]$ is defined as $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x} \cdot \boldsymbol{y} \coloneqq \sum_{i=1}^m x_i^* y_i$, where $\cdot^*$ stands for complex conjugation. The latter is needed so that $\boldsymbol{x} \cdot \boldsymbol{x} = \sum_i |x_i|^2$ is always a non-negative real. A number $\|\boldsymbol{x}\| \coloneqq \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$ is called a [L^2-norm or] *length* of vector $\boldsymbol{x}$. *Unit vectors* are vectors of length 1. We say two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are *othogonal*, if $\boldsymbol{x} \cdot \boldsymbol{y} = 0$. Zero vector $\mathbf{0}$ is orthogonal to any vector.

For $\hat{A} : \mathbf{C}^n \to \mathbf{C}^m$, the *adjoint* $\hat{A}^\dagger$ is introduced through $\langle \hat{A}^\dagger \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{x}, \hat{A} \boldsymbol{y} \rangle = \sum_{i=1}^m x_i^* \sum_{j=1}^n a_{ij} y_j = \sum_{j=1}^n \sum_{i=1}^m a_{ij} x_i^* y_j = \sum_{j=1}^n \left(\sum_{i=1}^m a_{ij}^* x_i \right)^* y_j$, which gives $(\hat{A}^\dagger)_{ji} = a_{ij}^*$, or $n \times m$ matrix $\hat{A}^\dagger = (\hat{A}^{\mathrm{T}})^*$ is the complex conjugate of the transpose of $\hat{A}$, or *Hermitian conjugate* of $\hat{A}$. A [necessarily square] matrix $\hat{A}$ is called *Hermitian*, if $\hat{A}^\dagger = \hat{A}$; it is the complex analogue of real symmetric matrix. The dot product $\boldsymbol{x} \cdot \boldsymbol{y}$ is equal to $\boldsymbol{x}^\dagger \boldsymbol{y}$ as a 1×1 matrix, that is the product of $1 \times m$ matrix $\boldsymbol{x}^\dagger$ and $m \times 1$ matrix $\boldsymbol{y}$.

We will call an $m \times n$ matrix $\hat{A}$ *unitary* or *orthogonal*, if $\hat{A}^\dagger \hat{A} = \hat{I}_n$. This is not a standard terminology.[18] [19] Necessarily, we have $m \geq n$, as $\operatorname{rank} \hat{A} \leq \min(m,n)$. The statement $\hat{A}^\dagger \hat{A} = \hat{I}_n$ simply means that all columns of $\hat{A}$ are unit vectors (the diagonal of $\hat{I}_n$) that are pair-wise orthogonal (off-diagonal content of $\hat{I}_n$).

In some cases other than "L^2" norms are more convenient. Commonly used vector norms are

$$
\begin{aligned}
L^p\text{-norm}: &\quad \|\boldsymbol{x}\|_p \coloneqq \left(|x_1|^p + |x_2|^p + ... + |x_m|^p \right)^{1/p}, \qquad 1 \leq p < +\infty \\
L^\infty\text{-norm}: &\quad \|\boldsymbol{x}\|_\infty \coloneqq \max\left(|x_1|, |x_2|, ... |x_m| \right) = \lim_{p \to +\infty} \|\boldsymbol{x}\|_p \\
L^0\text{-“not a norm”}: &\quad \|\boldsymbol{x}\|_0 \coloneqq \left(\text{number of non-zero components of } \boldsymbol{x} \right) \\
\text{weighted norm}: &\quad \|\boldsymbol{x}\|_{\hat{W}} \coloneqq \|\hat{W} \boldsymbol{x}\|, \qquad \operatorname{rank} \hat{W} = m, \text{ arbitrary norm on the right}
\end{aligned}
$$

[18] In standard definition, a matrix $\hat{U}$ is *unitary* if it is square, invertible, and $\hat{U}^{-1} = \hat{U}^\dagger$.

[19] People sometimes consider complex matrices $\hat{Q}$ such that $\hat{Q}^{-1} = \hat{Q}^{\mathrm{T}}$, and call them *orthogonal*. All such matrices form a Lie group [why?]. Real orthogonal matrices are unitary.

For matrices / linear transformations / operators most useful are *induced* or *operator* norms:

$$\hat{A} : \mathcal{X} \to \mathcal{Y}, \qquad \|\hat{A}\| := \sup_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\hat{A}\boldsymbol{x}\|_{\text{in }\mathcal{Y}}}{\|\boldsymbol{x}\|_{\text{in }\mathcal{X}}} = \sup_{\boldsymbol{x}\in\mathcal{X},\|\boldsymbol{x}\|_{\text{in }\mathcal{X}}=1} \|\hat{A}\boldsymbol{x}\|_{\text{in }\mathcal{Y}}$$

When both $\mathcal{X}$ and $\mathcal{Y}$ are finite dimensional, the operator norm is finite (the transformation is continuos and the "sphere" $\|\boldsymbol{x}\|_{\text{in }\mathcal{X}} = 1$ is compact).

Plenty of other norms in the vector space $\mathbf{C}^{mn}$ of $m \times n$ matrices could be constructed, and some of them are in use. Let us mention *Frobenius* or *Hilbert–Schmidt* norm:

$$\hat{A} = [a_{ij}], \quad \boldsymbol{a}_j = [a_{ij}], \qquad \|\hat{A}\|_{\text{F}}^2 := \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2 = \sum_{j=1}^{n} \|\boldsymbol{a}_j\|_2^2 = \operatorname{tr}\big(\hat{A}^\dagger\hat{A}\big) = \operatorname{tr}\big(\hat{A}\hat{A}^\dagger\big) = \|\hat{A}^\dagger\|_{\text{F}}^2$$

Let $\hat{Q}$ be an $m \times n$ unitary matrix. Then for any $\boldsymbol{x} \in \mathbf{C}^n$ we have $\|\hat{Q}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$. More generally, for any $n \times l$ matrix $\hat{A}$ we have $\|\hat{Q}\hat{A}\|_{\text{F}} = \|\hat{A}\|_{\text{F}}$.[20]

4.2 Singular Value Decomposition (SVD)

For clearer geometrical images, let us consider a real $m \times n$ matrix $\hat{A}$, and a corresponding linear transformation $\hat{A} : \mathbf{R}^n \to \mathbf{R}^m$. An image of a $(n-1)$-sphere $\mathbf{S}^{n-1} \subset \mathbf{R}^n$ is an [hyper]ellipsoid (which could be degenerate) in $\mathbf{R}^m$, whose principal sizes (or the lengths of the principal semi-axes) and orientation are important characteristics of $\hat{A}$. Let us denote the jth, $1 \le j \le m$, principal semi-axis as $\sigma_j\boldsymbol{u}_j$, where $\|\boldsymbol{u}_j\|_2 = 1$, so the semi-axis length is $\sigma_j \ge 0$. Geometrically, all the semi-axes, *i.e.*, $\boldsymbol{u}$-vectors, are orthogonal to each other. (Obviously, no more than n σ's are non-zero.)

Definition / Theorem 4: Any [real or complex] $m \times n$ matrix $\hat{A}$ has a *[reduced] Singular Value Decomposition* (SVD) $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^\dagger$, where $m \times l$ matrix $\hat{U}$ and $l \times n$ matrix $\hat{V}$ are unitary, while $l \times l$ matrix $\hat{\Sigma}$ is real diagonal, with non-negative [and non-increasing] diagonal entries. Here $l = \min(m,n)$.[21] The diagonal entries of $\hat{\Sigma}$ are called *singular values*, while the columns of the matrix $\hat{U}$ / $\hat{V}$ are called *left / right singular vectors*.

Proof [and **Algorithm** $\text{SVD}_{\hat{A}^\dagger\hat{A}}$]:[22] Let us consider the case $m \ge n$ (otherwise construct SVD of $\hat{A}^\dagger$, and then Hermite conjugate). Construct the $n \times n$ matrix $\hat{A}^\dagger\hat{A}$. It is Hermitian, thus it is diagonalizable, with real eigenvalues and orthogonal eigenvectors. It is also positive definite,[23] so all its eigenvalues are non-negative. We can write $\hat{A}^\dagger\hat{A} = \hat{V}\hat{\Sigma}^2\hat{V}^\dagger$, where $\hat{V}$ is $n \times n$ unitary matrix; matrix $\hat{\Sigma}$ is $n \times n$ real diagonal. We can choose the diagonal entries of $\hat{\Sigma}$ to be non-negative. By permuting the columns of $\hat{V}$, we can reorder the diagonal entries of $\hat{\Sigma}$ in non-increasing order. The jth column of $\hat{U}$, the vector $\boldsymbol{u}_j$, is set to $\hat{A}\boldsymbol{v}_j/\sigma_j$ if $\sigma_j > 0$, or chosen arbitrarily from the orthogonal completion to the previous columns of $\hat{U}$ if $\sigma_j = 0$. We still have to prove two statements: 1) $\hat{U}$ is unitary; and 2) $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^\dagger$.

For 1) we need to show that columns of $\hat{U}$ (it is enough to consider only non-zero σ's) are unit vectors that are orthogonal to each other. We have $\langle\boldsymbol{u}_i,\boldsymbol{u}_j\rangle = \boldsymbol{u}_i^\dagger\boldsymbol{u}_j = (\hat{A}\boldsymbol{v}_i)^\dagger\hat{A}\boldsymbol{v}_j/\sigma_i\sigma_j = \boldsymbol{v}_i^\dagger\hat{A}^\dagger\hat{A}\boldsymbol{v}_j/\sigma_i\sigma_j = \boldsymbol{v}_i^\dagger\hat{V}\hat{\Sigma}^2\hat{V}^\dagger\boldsymbol{v}_j/\sigma_i\sigma_j = \boldsymbol{e}_i^\dagger\hat{\Sigma}^2\boldsymbol{e}_j/\sigma_i\sigma_j = \delta_{ij}$ (as $\hat{\Sigma}^2$ is diagonal).

---

[20] Whenever the product $\hat{U}\hat{B}\hat{V}^\dagger$ is defined, and the matrices $\hat{U}$ and $\hat{V}$ are unitary, we have $\|\hat{U}\hat{B}\hat{V}^\dagger\|_{\text{F}} = \|\hat{B}\|_{\text{F}}$.

[21] Not so rarely considered *full* SVD is a factorization $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^\dagger$, where square $m \times m$ matrix $\hat{U}$ and $n \times n$ matrix $\hat{V}$ are unitary, and $m \times n$ matrix $\hat{\Sigma}$ is diagonal (with non-negative numbers on the main diagonal).

[22] For a different proof, see [TrBa97, Theorem 4.1, p. 29].

[23] For any vector $\boldsymbol{v}$ we have $\langle\boldsymbol{v},\hat{A}^\dagger\hat{A}\boldsymbol{v}\rangle = \boldsymbol{v}^\dagger\hat{A}^\dagger\hat{A}\boldsymbol{v} = (\hat{A}\boldsymbol{v})^\dagger\hat{A}\boldsymbol{v} = \|\hat{A}\boldsymbol{v}\|_2^2 \ge 0$.

For 2), vectors $\boldsymbol{v}_j$, $j = 1, \ldots, n$, form a basis of $\mathbf{C}^n$ [why?], so any vector $\boldsymbol{v} \in \mathbf{C}^n$ is a [unique] linear combination of them. It is enough to check $\hat{A}\boldsymbol{v}_j = \hat{U}\hat{\Sigma}\hat{V}^\dagger \boldsymbol{v}_j = \hat{U}\sigma_j \boldsymbol{e}_j = \sigma_j \boldsymbol{u}_j$ for all $1 \le j \le n$. If $\sigma_j > 0$, then we have $\hat{A}\boldsymbol{v}_j = \sigma_j \boldsymbol{u}_j$ by construction of $\boldsymbol{u}_j$. If $\sigma_j = 0$, then $\boldsymbol{v}_j^\dagger \hat{A}^\dagger \hat{A} \boldsymbol{v}_j = \sigma_j^2 = 0 = \|\hat{A}\boldsymbol{v}_j\|_2^2$, thus $\hat{A}\boldsymbol{v}_j = \mathbf{0} = \sigma_j \boldsymbol{u}_j$.

Once the singular values are ordered, the matrix $\hat{\Sigma}$ is unique. For any j we can multiply $\boldsymbol{v}_j$ by any number c with absolute value $|c| = 1$ (only $c = -1$ is interesting in real case), and both $\|\boldsymbol{v}_j\|_2$ and $\|\hat{A}\boldsymbol{v}_j\|_2$ are not going to change. For some matrices (when singular values coincide) the choice for $\hat{V}$ is even richer. *I.e.*, for any square unitary matrix $\hat{V}$ we have $\hat{I} = \hat{V}^\dagger \hat{V}$ as the SVD of the identity matrix $\hat{I}$. Generally, we can make an arbitrary unitary rotation of $\hat{V}$'s (and simultaneously of $\hat{U}$'s) part that corresponds to the same singular value.

Example 4: Consider the matrix on the right. Both of its eigenvalues are equal to 1. Let us proceed with the SVD of $\hat{A}$ in Octave[24] and on paper:

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

```
[...]/teaching/2019-4/math_575a/notes/Octave$ octave-cli
GNU Octave, version 4.2.1
[... copyright notice and links ...]
octave:1> format long
octave:2> A = [1, 1; 0, 1]
A =

   1   1
   0   1

octave:3> [U, S, V] = svd(A)
U =

   0.850650808352040  -0.525731112119134
   0.525731112119134   0.850650808352040

S =

Diagonal Matrix

   1.618033988749895                   0
                   0   0.618033988749895

V =

   0.525731112119134  -0.850650808352040
   0.850650808352040   0.525731112119134

octave:4> B = U * S * V'
B =

   1.00000000000000e+00   1.00000000000000e+00
   1.11022302462516e-16   1.00000000000000e+00

octave:5>
```

$$\hat{A}^\dagger \hat{A} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\det\left(\lambda \hat{I}_2 - \hat{A}^\dagger \hat{A}\right) = \lambda^2 - 3\lambda + 1$$

$$\sigma_{1,2}^2 = \lambda_{1,2} = \tfrac{1}{2}\left(3 \pm \sqrt{5}\right)$$

$$\sigma_{1,2} = \sqrt{\lambda_{1,2}} = \tfrac{1}{2}\left(\sqrt{5} \pm 1\right) = \varphi^{\pm 1}$$

$$\hat{A}^\dagger \hat{A} \underbrace{\begin{bmatrix} 1 \\ \varphi \end{bmatrix}}_{\sqrt{1+\varphi^2}\,\boldsymbol{v}_1} = \left(\varphi^2 = \sigma_1^2\right) \begin{bmatrix} 1 \\ \varphi \end{bmatrix}$$

note: $1 + \varphi = \varphi^2$

$$\boldsymbol{u}_1 = \frac{\hat{A}\boldsymbol{v}_1}{\sigma_1} = \frac{\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ \varphi \end{bmatrix}}{\varphi\sqrt{1+\varphi^2}} = \frac{\begin{bmatrix} \varphi \\ 1 \end{bmatrix}}{\sqrt{1+\varphi^2}}$$

find $\boldsymbol{v}_2$ and $\boldsymbol{u}_2$ in a similar way, or from orthogonality

[24] MATLAB® is a commercial software, see MathWorks MATLAB licensing for UA Faculty, Staff & Students. GNU Octave is one of several (less effective) free alternatives to MATLAB, with mostly compatible syntax.

The Algorithm $\mathrm{SVD}_{\hat{A}^\dagger\hat{A}}$ is good for pen and paper calculations, but it is not numerically stable. The problem of diagonalization of $\hat{A}^\dagger\hat{A}$ has condition number $\kappa^2(\hat{A})$, instead of $\kappa(\hat{A})$.

We will not discuss numerical algorithms for the computation of SVD in detail. See, *e.g.*, [GoVa96, Sec. 8.6] and [Dem97, Sec. 5.4].

Here are some properties of SVD (here $\hat{A}$ is an $m\times n$ matrix, and $l=\min(m,n)$):

$$\hat{A}=\sum_{i=1}^{l}\sigma_i\boldsymbol{u}_i\boldsymbol{v}_i^\dagger,\qquad \|\hat{A}\|_{\mathrm{F}}^2=\sum_{i=1}^{l}\sigma_i^2,\qquad \|\hat{A}\|_2=\sigma_1,\qquad \big\langle\sigma_1,\boldsymbol{u}_1,\boldsymbol{v}_1\big\rangle=\underset{\langle\sigma_1,\boldsymbol{u}_1,\boldsymbol{v}_1\rangle}{\arg\min}\,\|\hat{A}-\sigma_1\boldsymbol{u}_1\boldsymbol{v}_1^\dagger\|_{\mathrm{F}}$$

$$\hat{A}_\downarrow:=\hat{A}-\sigma_1\boldsymbol{u}_1\boldsymbol{v}_1^\dagger,\qquad \big(\hat{A}_\downarrow\big)_\downarrow=\hat{A}-\sigma_1\boldsymbol{u}_1\boldsymbol{v}_1^\dagger-\sigma_2\boldsymbol{u}_2\boldsymbol{v}_2^\dagger,\qquad \sigma_1(\hat{A}_\downarrow)=\sigma_2(\hat{A})$$

$\sum_{i=1}^{r}\sigma_i\boldsymbol{u}_i\boldsymbol{v}_i^\dagger$ is the best approximation (in $\|\cdot\|_2$ and $\|\cdot\|_{\mathrm{F}}$ norms) of $\hat{A}$ by any matrix of rank $r\le l$

$$\big|\det\hat{A}\big|=\prod_{i=1}^{l=m=n}\sigma_i,\qquad \text{if } \hat{A}=\hat{A}^\dagger\text{, then its diagonalization is almost (up to signs of }\lambda\text{'s) its SVD}$$

4.3 Condition number of multiplication by a matrix

Consider a fixed $m\times n$ matrix $\hat{A}$. What is the [relative] condition number $\kappa(\hat{A}\cdot)$ of the problem of multiplying a vector by $\hat{A}$?[25]

If $n>m$ or the matrix $\hat{A}$ is not of full rank, then it has a non-trivial null space. Any errors in the computation of $\hat{A}\boldsymbol{x}=\boldsymbol{0}$ for $\boldsymbol{x}\in\mathrm{null}(\hat{A})$ would be considered infinite in relative sense, and thus $\kappa(\hat{A}\cdot)=\infty$ by definition. Otherwise we have

$$\kappa(\hat{A}\cdot,\boldsymbol{x})=\underbrace{\max_{\boldsymbol{\varepsilon}}\overbrace{\frac{\|\hat{A}(\boldsymbol{x}+\boldsymbol{\varepsilon})-\hat{A}\boldsymbol{x}\|}{\|\boldsymbol{\varepsilon}\|}}^{\text{sensitivity to }\boldsymbol{x}}}_{\|\hat{A}\|=\sigma_1}\frac{\|\boldsymbol{x}\|}{\|\hat{A}\boldsymbol{x}\|},\qquad \kappa(\hat{A}\cdot):=\max_{\boldsymbol{x}}\kappa(\hat{A}\cdot,\boldsymbol{x})=\frac{\sigma_1}{\sigma_n}$$

The denominator σ_n came from maximizing the ratio $\|\boldsymbol{x}\|/\|\hat{A}\boldsymbol{x}\|$.

Problems and exercises

1. Two norms $\|\cdot\|_{\mathrm{I}}$ and $\|\cdot\|_{\mathrm{II}}$ are called *equivalent* if there exist constants $0<C_1\le C_2$ such that $C_1\|\boldsymbol{x}\|_{\mathrm{I}}\le\|\boldsymbol{x}\|_{\mathrm{II}}\le C_2\|\boldsymbol{x}\|_{\mathrm{I}}$ for all $\boldsymbol{x}$. (a) Show that any two norms in $\mathbf{R}^m$ are equivalent. (b) Find constants C_1 and C_2 for any pair from $\|\cdot\|_1$, $\|\cdot\|_2$, and $\|\cdot\|_\infty$ norms.

2. Consider two operators $\hat{A}:X\to Y$ and $\hat{B}:Y\to Z$, where the vector spaces X, Y, and Z are normed. For induced norms, show that $\|\hat{B}\hat{A}\|\le\|\hat{A}\|\,\|\hat{B}\|$.

3. Consider "dot product with $\boldsymbol{u}$" operator $\boldsymbol{u}^\dagger:\mathbf{C}^n\to\mathbf{C}$, where $\boldsymbol{x}\mapsto\boldsymbol{u}\cdot\boldsymbol{x}=\boldsymbol{u}^\dagger\boldsymbol{x}$. Find its (a) induced L^2-norm and (b) Frobenius norm.

4. Consider "multiplying by $\boldsymbol{u}$" operator $\boldsymbol{u}:\mathbf{C}\to\mathbf{C}^m$, where $x\mapsto x\boldsymbol{u}$. Find its (a) induced L^2-norm and (b) Frobenius norm.

5. Let $\hat{A}$ be an $m\times n$ matrix, with an SVD $\hat{A}=\hat{U}\hat{\Sigma}\hat{V}^\dagger$. Find the SVD of $\hat{A}^\dagger$.

6. Let $\hat{A}$ be an $m\times n$ matrix. Consider $(m+n)\times(m+n)$ Hermitian matrix $\hat{B}:=\begin{bmatrix}\hat{O}_{n,n} & \hat{A}^\dagger\\ \hat{A} & \hat{O}_{m,m}\end{bmatrix}$. How the singular values of $\hat{A}$ and the eigenvalues of $\hat{B}$ are connected?

7. Show that $m\times n$, $m\ge n$, matrix is unitary if and only if all of its singular values are equal to 1.

[25] The so called *condition number* $\kappa(\hat{A})$ of a matrix $\hat{A}$ is defined in a similar but slightly different way: $\kappa(\hat{A})=\|\hat{A}\|_2\|\hat{A}^+\|_2$, where $\hat{A}^+$ is the Moore–Penrose pseudoinverse of $\hat{A}$.

5 Systems of linear equations

Consider a system of linear equations $\hat{A}\boldsymbol{x} = \boldsymbol{b}$, with $m \times n$ matrix $\hat{A}$. A way to interpret the system, which is often good while thinking theoretically, is: $\hat{A}$ is the matrix of a linear transformation $\mathbf{C}^n \to \mathbf{C}^m$, and we try to find such vector $\boldsymbol{x} \in \mathbf{C}^n$ that is transformed to $\boldsymbol{b}$, *i.e.*, $\hat{A} : \boldsymbol{x} \mapsto \boldsymbol{b}$.

Let us assume for simplicity that $\hat{A}$ is a full rank $m \times m$ matrix. Interpreting the system in way I), the solution could be found by computing the inverse matrix and reversing the transformation $\hat{A}$:

Algorithm $\hat{A}^{-1}\boldsymbol{b}$: Input: $\hat{A}$ and $\boldsymbol{b}$. Output: $\boldsymbol{x} := \hat{A}^{-1}\boldsymbol{b}$, *i.e.*, you compute the inverse matrix $\hat{A}^{-1}$ and multiply the r.h.s. $\boldsymbol{b}$ by it.

Example 5: "Failure" of $\hat{A}^{-1}\boldsymbol{b}$ in case of poorly conditioned matrix $\hat{A}$.

```
octave:1> format long
octave:2> A = [1 sqrt(2); sqrt(3) sqrt(6) + 1.e-13]
A =

   1.000000000000000   1.414213562373095
   1.732050807568877   2.449489742783278
```

$$\hat{A} = \begin{bmatrix} 1 & \sqrt{2} \\ \sqrt{3} & \sqrt{6} + 10^{-13} \end{bmatrix}$$

```
octave:3> cond(A)
ans =  119564591877097.7
octave:4> B = inverse(A)
B =

   24496352586638.59  -14142975760059.20
  -17321537028348.06   10000594066094.83
```

$$\sigma_1 \sim 1 \;\text{ and }\; \sigma_1\sigma_2 = \det\hat{A} = 10^{-13}$$

```
octave:5> b = [sqrt(3); 3]
b =

   1.732050807568877
   3.000000000000000
```

$$\boldsymbol{b} = \begin{bmatrix} \sqrt{3} \\ 3 \end{bmatrix}$$

```
octave:6> x = A \ b
x =

   1.721172241704469e+00
   7.692307692307693e-03
```

solution by $\backslash$ (or `mldivide`) operator

```
octave:7> y = B * b
y =

   1.726562500000000e+00
   3.906250000000000e-03
```

solution by Algorithm $\hat{A}^{-1}\boldsymbol{b}$

```
octave:8> A * x - b
ans =

  -2.220446049250313e-16
   0.000000000000000e+00

octave:9> A * y - b
ans =
```

```
   3.596415914275397e-05
   6.229175088678929e-05

octave:10> A * [sqrt(3); 0] - b
ans =

   0.000000000000000e+00
  -4.440892098500626e-16

octave:11>
```

residual $\hat{A}(\hat{A}^{-1}\boldsymbol{b}) - \boldsymbol{b}$ is not too small

$\boldsymbol{x} = \begin{bmatrix} \sqrt{3} \\ 0 \end{bmatrix}$ is exact solution

$\hat{A}^{-1}\boldsymbol{b}$ solution is numerically closer to it than solution by \\,
but $\hat{A}^{-1}\boldsymbol{b}$ provides orders of magnitude larger residual

This is a [forward] stable, but not a backward stable algorithm. The vector $\boldsymbol{x}$ is not far from the exact solution, but the discrepance is such that it is not very plausible:

Here the thick line is $\boldsymbol{x}_{\text{exact}} + t\boldsymbol{v}_2$, where $\boldsymbol{v}_1 \approx \begin{bmatrix} 1 & \sqrt{2} \end{bmatrix}^{\mathrm{T}}/\sqrt{3}$ and $\boldsymbol{v}_2 \approx \begin{bmatrix} \sqrt{2} & -1 \end{bmatrix}^{\mathrm{T}}/\sqrt{3}$ are the right singular vectors of $\hat{A}$. The vector $\hat{A}^{-1}\boldsymbol{b} - \boldsymbol{x}_{\text{exact}}$ has some $\boldsymbol{v}_1$ component, which gives rise to not so small residual $\hat{A}(\hat{A}^{-1}\boldsymbol{b}) - \boldsymbol{b}$.

Let us take the matrix $\hat{A}$ and the r.h.s. $\boldsymbol{b}$, put them into a computer with double precision (also solve $\hat{A}\boldsymbol{x} = \boldsymbol{b}$ by Gaussian elimination for illustration), and then *exactly* solve the system of 2 linear equations with floating point number representations of $\hat{A}$ and $\boldsymbol{b}$, thinking about them as *exact* input:

```
[...]/teaching/2019-4/math_575a/notes$ cat C/A_x_eq_b.c
#include <stdio.h>
#include <math.h>
void print_bits(double x, char* s) { int i; unsigned char c;
  for (i = 0; i < 8; i++) for (c = 128; c != 0; c >>= 1)
    if (*((char *)(&x) + 7 - i) & c) printf("1"); else printf("0");
  printf("  %s\n", s); }
int main() { int i; double A[2][2], b[2], x[2], det, G[2][2], x52;
  A[0][0] = 1.; A[0][1] = sqrt(2.); /* Example 5 */
  A[1][0] = sqrt(3.); A[1][1] = sqrt(6.) + 1.e-13;
  print_bits(A[0][0], "a_11"); print_bits(A[0][1], "a_12");
  print_bits(A[1][0], "a_21"); print_bits(A[1][1], "a_22");
  b[0] = sqrt(3.); b[1] = 3.;
  print_bits(b[0], "b_1"); print_bits(b[1], "b_2");
printf("|\\___  ____/\\_________________________  _______________________/\n");
printf("|    \\/                               \\/\n");
printf("sign  exponent (11 bits)               fraction (52 bits)\n\n");
  x52 = 1.; for (i = 0; i < 52; i++) x52 *= 2.; /* x52 = 2^52 */
  printf("a_12 = %18.1f / 2^52,   a_21 = b_1 = %18.1f / 2^52\n",
```

```
    x52 * A[0][1], x52 * A[1][0]);
  printf("a_11 = 1,   b_2 = 3,   a_22 = %18.1f / 2^52\n\n", x52 * A[1][1]);

  printf("solution by Cramer's rule:\n");
  det = 1. / (A[0][0] * A[1][1] - A[0][1] * A[1][0]);
  x[0] = (b[0] * A[1][1] - b[1] * A[0][1]) * det;
  x[1] = (A[0][0] * b[1] - A[1][0] * b[0]) * det;
  printf("   x = [%22.16e %22.16e]^T\n", x[0], x[1]);
  printf("   residual A * x - b: [%e %e]^T\n", x[0] + A[0][1] * x[1] - b[0],
    A[1][0] * x[0] + A[1][1] * x[1] - b[1]);

  printf("solution A^(-1) b:\n");
  G[0][0] = A[1][1] * det; G[0][1] = -A[0][1] * det;
  G[1][0] = -A[1][0] * det; G[1][1] = A[0][0] * det;
  x[0] = G[0][0] * b[0] + G[0][1] * b[1];
  x[1] = G[1][0] * b[0] + G[1][1] * b[1];
  printf("   x = [%22.16e %22.16e]^T\n", x[0], x[1]);
  printf("   residual A * x - b: [%e %e]^T\n", x[0] + A[0][1] * x[1] - b[0],
    A[1][0] * x[0] + A[1][1] * x[1] - b[1]);

  printf("solution by Gaussian elimination with complete pivoting:\n");
  G[1][0] = A[1][0] / A[1][1]; x[1] = b[1] / A[1][1];
  G[0][0] = A[0][0] - A[0][1] * G[1][0]; x[0] = b[0] - A[0][1] * x[1];
  x[0] /= G[0][0]; x[1] -= G[1][0] * x[0];
  printf("   x = [%22.16e %22.16e]^T\n", x[0], x[1]);
  printf("   residual A * x - b: [%e %e]^T\n", x[0] + A[0][1] * x[1] - b[0],
    A[1][0] * x[0] + A[1][1] * x[1] - b[1]);
  return 0; }
[...]/teaching/2019-4/math_575a/notes$ cc C/A_x_eq_b.c -lm ; ./a.out
0011111111110000000000000000000000000000000000000000000000000000  a_11
0011111111110110101000001001111001100110011111110011101111001101  a_12
0011111111111011101101100111101011101000010110000100110010101010  a_21
0100000000000011100110001000111000010100000010010010001000001111  a_22
0011111111111011101101100111101011101000010110000100110010101010  b_1
0100000000001000000000000000000000000000000000000000000000000000  b_2
|\___  _____/\__________________________  _______________________/
|    \/                                 \/
sign  exponent (11 bits)                fraction (52 bits)

a_12 = 6369051672525773.0 / 2^52,   a_21 = b_1 = 7800463371553962.0 / 2^52
a_11 = 1,   b_2 = 3,   a_22 = 11031521092846622.0 / 2^52

solution by Cramer's rule:
   x = [1.7321428571428570e+00 4.4642857142857140e-03]^T
   residual A * x - b: [6.405503e-03 1.109466e-02]^T
solution A^(-1) b:
   x = [1.7265625000000000e+00 3.9062500000000000e-03]^T
   residual A * x - b: [3.596416e-05 6.229175e-05]^T
solution by Gaussian elimination with complete pivoting:
   x = [1.7267759562841529e+00 3.7298831131791221e-03]^T
   residual A * x - b: [0.000000e+00 0.000000e+00]^T
[...]/teaching/2019-4/math_575a/notes$ gp
[... technical stuff ...]
                  GP/PARI CALCULATOR Version 2.11.1 (released)
```

```
[... copyright notice and links, technical parameters ...]
? a11 = 2^52; a12 = 6369051672525773; a21 = b1 = 7800463371553962; a22 = 1103152
1092846622; b2 = 3 * a11; det = a11 * a22 - a12 * a21
%1 = 2021703648790801886
? x1 = (b1 * a22 - b2 * a12) / det
%2 = 102697713703618710/59461872023258879
? x2 = (a11 * b2 - a21 * b1) / det
%3 = 3525341537980302/1010851824395400943
? 1. * x1
%4 = 1.7271187436454719105197015986197581099
? 1. * x2
%5 = 0.0034874958454854040546168999582954249611
```

The *exact* solution, $\tilde{\boldsymbol{x}}_{\text{exact}} \approx \left[\begin{array}{cc} 1.7271 & 0.0035 \end{array}\right]^{\mathrm{T}}$, of the "computerized" system (*i.e.*, how it looks like *after* the input data $\hat{A}$ and $\boldsymbol{b}$ are put into $\mathbf{FP}_{52}$), is as different from non-computerized $\boldsymbol{x}_{\text{exact}} = \left[\begin{array}{cc} \sqrt{3} & 0 \end{array}\right]^{\mathrm{T}}$ as solutions obtained by Gaussian elimination, $\boldsymbol{x}_{\text{GE}} \approx \left[\begin{array}{cc} 1.7268 & 0.0037 \end{array}\right]^{\mathrm{T}}$, or `mldivide` operator, $\hat{A}\backslash\boldsymbol{b} \approx \left[\begin{array}{cc} 1.7212 & 0.0077 \end{array}\right]^{\mathrm{T}}$.

5.1 System with orthogonal matrix

Consider a system $\hat{Q}\boldsymbol{x} = \boldsymbol{b}$, where $\hat{Q}$ is an $m \times m$ orthogonal or unitary matrix. The solution is $\boldsymbol{x} \coloneqq \hat{Q}^{\dagger}\boldsymbol{b}$, which could be viewed as the one obtained by the Algorithm $\hat{A}^{-1}\boldsymbol{b}$ (we have $\hat{Q}^{-1} = \hat{Q}^{\dagger}$). Here the matrix of the system $\hat{Q}$ is well conditioned though, $\kappa(\hat{Q}\cdot) = 1$. The L^2-norm of the residual is small: $\left\|\hat{Q}\big(\text{num}(\hat{Q}^{\dagger}) \odot \text{num}(\boldsymbol{b})\big) - \boldsymbol{b}\right\|_2 = \left\|\big(\text{num}(\hat{Q}^{\dagger}) \odot \text{num}(\boldsymbol{b})\big) - \hat{Q}^{\dagger}\boldsymbol{b}\right\|_2$ — we have $\hat{Q}\hat{Q}^{\dagger} = \hat{I}_m$, and multiplying a vector by unitary matrix $\hat{Q}^{\dagger}$ doesn't change its L^2-norm.

5.2 System with triangular matrix (see [TrBa97, Lec. 17])

Consider a system $\hat{R}\boldsymbol{x} = \boldsymbol{b}$, where $\hat{R}$ is an $n \times n$ upper triangular matrix, *i.e.*, $r_{ij} = 0$ whenever $i > j$. We assume that $\det\hat{R} = r_{11}r_{22}...r_{nn} \neq 0$, *i.e.*, the solution does exist and is unique. It can be found by a procedure called *back substitution* (the matrix of the system is already in echelon form, *i.e.*, most of the work of excluding variables is done): the last equation means $r_{nn}x_n = b_n$, so we immediately find $x_n \coloneqq b_n/r_{nn}$. In the equation $r_{n-1,n-1}x_{n-1} + r_{n-1,n}x_n = b_{n-1}$ only x_{n-1} is unknown, so we immediately find it: $x_{n-1} = (b_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1}$. Next we find x_{n-2}, and so on, till we finally find x_1.

Algorithm "Back Substitution": Input: upper triangular matrix $\hat{R}$ and the r.h.s. $\boldsymbol{b}$. Output: vector $\boldsymbol{x}$, computed according to the following pseudo-code:

```
for i = n, n−1, ..., 2, 1 do
    B := b_i
    for j = n, n−1, ..., i+1 do        (do nothing if n < i+1, i.e., if i = n)
        B := B ⊖ (r_ij ⊙ x_j)
    x_i := B ⊘ r_ii
return x
```

Alternatively, one can go from $i+1$ to n in `for` loop over j. There are $n(n-1)/2$ multiplications $r \odot x$, $n(n-1)/2$ subtractions $B \ominus (rx)$, and n divisions $B \oslash r$; overall n^2 floating point operations.

The pseudo-code could be written in a different form (which ruins the input vector $\boldsymbol{b}$, though):

```
for i = n, n−1, ..., 2, 1 do
    x_i := b_i ⊘ r_ii
    for j = i−1, i−2, ..., 1 do        (do nothing if i−1 < 1, i.e., if i = 1)
        b_j := b_j ⊖ (r_ji ⊙ x_i)
return x
```

Theorem 5: The Algorithm "Back Substitution" is backward stable. Moreover, we can interpret the output as the exact solution for the problem $(\hat{R}+\Delta\hat{R})\boldsymbol{x}=\boldsymbol{b}$ with $\|\Delta\hat{R}\| \sim \varepsilon_{\text{machine}}\|\hat{R}\|$, *i.e.*, we need to slightly change only the matrix $\hat{R}$, with the r.h.s. $\boldsymbol{b}$ being untouched. Moreover, we can show that $\Delta r_{ij} \sim \varepsilon_{\text{machine}} r_{ij}$, *i.e.*, we can choose the matrix $\Delta\hat{R}$ in such a way, that each of its matrix elements is a small, $\sim \varepsilon_{\text{machine}}$, change of the corresponding matrix element of $\hat{R}$. More specifically, $\left|(\Delta r_{ij})/r_{ij}\right| \leq (n+1-j)\varepsilon_{\text{machine}} + O(\varepsilon^2_{\text{machine}})$.[26]

Proof: Consider the case $n=3$ and the calculation of x_1:

$$x_1 := \Big(\big(b_1 \ominus (r_{13}\odot x_3)\big) \ominus (r_{12}\odot x_2)\Big) \oslash r_{11}$$

$$x_1 = \Big(\big(b_1 - r_{13}x_3(1+\varepsilon_{13})\big)(1+\delta_{13}) - r_{12}x_2(1+\varepsilon_{12})\Big)(1+\delta_{12})\cdot\frac{(1+\eta_1)}{r_{11}}$$

Here $|\varepsilon_{ij}|$, $|\delta_{ij}|$, and $|\eta_i|$ are all not greater than $\varepsilon_{\text{machine}}$: ε's are numerical errors introduced in calculation $r_{ij}x_j$, δ's are introduced while doing subtractions, and η's are due to divisions. The formula for x_1 could be rewritten as

$$x_1 = \frac{b_1 - r_{13}x_3(1+\varepsilon_{13}) - r_{12}x_2(1+\varepsilon_{12})(1+\delta_{13})^{-1}}{r_{11}(1+\eta_1)^{-1}(1+\delta_{12})^{-1}(1+\delta_{13})^{-1}}$$

Whenever we are doing subtraction, we push the numerical error as a change in r's on the left. Each non-diagonal r has potential need for change due to rx multiplications. Diagonal r's have additional potential need to change due to divisions.

Whenever we present the matrix $\hat{A}$ of the system $\hat{A}\boldsymbol{x}=\boldsymbol{b}$ as the product of unitary or triangular (partial case: diagonal) matrices, $\hat{A}=\hat{A}_1\hat{A}_2...\hat{A}_k$, we can find $\boldsymbol{x}$ as follows: First, solve $\hat{A}_1\boldsymbol{y_1}=\boldsymbol{b}$. This is easy, we have $\boldsymbol{y}_1=\hat{A}_1^{\dagger}\boldsymbol{b}$, if $\hat{A}_1$ is unitary; or we find $\boldsymbol{y}_1$ by back ($\hat{A}_1$ is upper triangular) or by forward ($\hat{A}_1$ is lower triangular, you subsequently find components of $\boldsymbol{y}_1$ from the first to the last) substitution. Then we solve $\hat{A}_2\boldsymbol{y}_2=\boldsymbol{y}_1$, $\hat{A}_3\boldsymbol{y}_3=\boldsymbol{y}_2$, and so on. Finally, we deal with $\hat{A}_k\boldsymbol{x}=\boldsymbol{y}_{k-1}$. We have

$$\boldsymbol{b}=\hat{A}_1\Big(\boldsymbol{y}_1=\hat{A}_2\Big(\boldsymbol{y_2}=\hat{A}_3\big(\boldsymbol{y}_3=...\hat{A}_{k-1}(\boldsymbol{y}_{k-1}=\hat{A}_k\boldsymbol{x})\big)\Big)\Big), \qquad \boldsymbol{b}=\hat{A}_1\hat{A}_2...\hat{A}_k\boldsymbol{x}=\hat{A}\boldsymbol{x}$$

The commonly used factorizations of $\hat{A}$ are:

QR factorization :	$\hat{A}=\hat{Q}\hat{R}$,	$\hat{Q}$ is unitary, $\hat{R}$ is upper triangular
with column pivoting :	$\hat{A}=\hat{Q}\hat{R}\hat{P}$,	$\hat{P}$ is [column] permutation matrix
Gaussian elimination, LU factorization :	$\hat{A}=\hat{L}\hat{U}$,	$\hat{L}/\hat{U}$ is lower / upper triangular
with partial pivoting :	$\hat{A}=\hat{P}\hat{L}\hat{U}$,	$\hat{P}$ is [row] permutation matrix
with complete pivoting :	$\hat{A}=\hat{P}\hat{L}\hat{U}\hat{Q}$,	$\hat{Q}$ is [column] permutation matrix

[26] This is a different pattern or $\varepsilon_{\text{machine}}$-perturbations of $\hat{R}$ than the one given in [TrBa97, p. 126], because there $r_{ij}x_j$ terms are subtracted from b_i in different order (from left to right).

Problems and exercises

1. Consider the system $\hat{A}\boldsymbol{x} = \boldsymbol{b}$, where

$$\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^\dagger = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix} \begin{bmatrix} 0.6 & 0.8 \\ -0.8 & 0.6 \end{bmatrix}, \qquad \boldsymbol{b} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Find $\boldsymbol{x}$ analytically. Solve the system numerically for $\varepsilon = k \cdot 10^{-14}$, $k = 1, 2, ..., 10$, by 1) computing $\hat{A}$, then applying the Cramer's rule; 2) $\boldsymbol{x} := \hat{V}\hat{\Sigma}^{-1}\hat{U}^\dagger\boldsymbol{b}$. Plot all 20 numerical solutions on one plot.

6 QR factorization

Definition/Theorem 6: Any [real or complex] $m \times n$, $m \geq n$, matrix $\hat{A}$ has a *[reduced] QR factorization* $\hat{A} = \hat{Q}\hat{R}$, where $m \times n$ matrix $\hat{Q}$ is unitary, while $n \times n$ matrix $\hat{R}$ is upper triangular.[27] If $\hat{A}$ is of full rank, *i.e.*, rank$\hat{A} = n$, then there is only one QR factorization with [strictly] positive diagonal entries of $\hat{R}$.

Proof: Matrices $\hat{Q}$ and $\hat{R}$ could be produced by the Gram–Schmidt process. In case of full rank, the columns of $\hat{Q}$ are defined up to a multiplicative factor with absolute value equal to 1, which is uniquely set if one requires $r_{ii} > 0$ for all $1 \leq i \leq n$.

There are two major approaches for obtaining the QR factorization of a matrix (here $\hat{A}$ is an $m \times n$ matrix with $m \geq n$, on schematic pictures the long/short sides have lengths m/n). One strategy, employed in Gram–Schmidt process, is to apply "upper triangular" column operations to matrix $\hat{A}$, in order to make it unitary:

$$\hat{A}\ \hat{I}_n \longrightarrow \underbrace{\hat{Q}_{k-1}\ \hat{U}_{k-1}}_{\hat{Q}_k}\ \underbrace{\hat{U}_k^{-1}\ \hat{R}_{k-1}}_{\hat{R}_k} \longrightarrow \hat{Q}\ \hat{R}$$

$$\hat{Q} = \hat{A}\hat{U}_1\hat{U}_2\hat{U}_3... \qquad \hat{R} = ...\hat{U}_3^{-1}\hat{U}_2^{-1}\hat{U}_1^{-1}$$

Another strategy, employed in Householder reflections and Givens rotations methods, is to act by unitary matrices $\hat{V}_1$, $\hat{V}_2$, $\hat{V}_3$, ..., on the matrix $\hat{A}$ until it becomes upper triangular:

$$\hat{I}_m\ \hat{A} \longrightarrow \underbrace{\hat{Q}_{k-1}\ \hat{V}_k^\dagger}_{\hat{Q}_k}\ \underbrace{\hat{V}_k\ \hat{R}_{k-1}}_{\hat{R}_k} \longrightarrow \hat{Q}\ \hat{R}$$

$$\hat{Q} = \hat{V}_1^\dagger\hat{V}_2^\dagger\hat{V}_3^\dagger... \qquad \hat{R} = ...\hat{V}_3\hat{V}_2\hat{V}_1\hat{A}$$

[27] A *full* QR factorization is $\hat{A} = \hat{Q}\hat{R}$, where square $m \times m$ matrix $\hat{Q}$ is unitary, and $m \times n$ matrix $\hat{R}$ is upper triangular (with zero matrix elements below the main diagonal).

6.1 Gram–Schmidt process

Algorithms "Classical / Modified Gram–Schmidt": Input: $m \times n$ matrix $\hat{A}$, $m \geq n$. Output: $m \times n$ unitary matrix $\hat{Q}$ and $n \times n$ upper triangular matrix $\hat{R}$, such that $\hat{A} = \hat{Q}\hat{R}$, according to the following pseudo-code:[28]

classical Gram–Schmidt

```
for j = 1, 2, ..., n do
    q_j := a_j
    for i = 1, 2, ..., j − 1 do
        r_ij := q_i† a_j          ⟶   modification: r_ij := q_i† q_j   ⟶
        q_j := q_j − r_ij q_i
    r_jj := ||q_j||_2
    q_j := q_j / r_jj
return Q̂, R̂
```

modified Gram–Schmidt[29]

```
for i = 1, 2, ..., n do
    r_ii := ||a_i||_2
    a_i := a_i / r_ii
    for j = i + 1, i + 2, ..., n do
        r_ij := a_i† a_j
        a_j := a_j − r_ij a_i
return Â, R̂
```

On the right is the pseudo-code for Modified Gram–Schmidt that is equivalent (just operations are done in different order) to the algorithm after $r_{ij} := \boldsymbol{q}_i^\dagger \boldsymbol{a}_j \to \boldsymbol{q}_i^\dagger \boldsymbol{q}_j$ modification.

Example 6: Let us compute the QR factorization of $\hat{A}$ using both classical (cGS) and modified (mGS) Gram–Schmidt methods. We will assume that $\varepsilon^2 \ll \varepsilon_{\text{machine}} \ll \varepsilon$ in our calculations (*i.e.*, we will drop ε^2 terms when added to something of the order of 1).

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

The 1st column is already as good as normalized ($\|\boldsymbol{a}_1\|_2^2 = 1 + \varepsilon^2 \approx 1$), so $r_{11} = 1$ and $\boldsymbol{q}_1 = \boldsymbol{a}_1$. We have $r_{12} = \boldsymbol{q}_1^\dagger \boldsymbol{a}_2 = 1$, and $r_{22} = \sqrt{2}\varepsilon$, $\boldsymbol{q}_2 = \begin{bmatrix} 0 & -1 & 1 & 0 \end{bmatrix}^{\mathrm{T}} / \sqrt{2}$ — so far there is no difference between classical and modified versions of the Gram–Schmidt processes. Now in classical version we have $r_{13} = \boldsymbol{q}_1^\dagger \boldsymbol{a}_3 = \boldsymbol{a}_1^\dagger \boldsymbol{a}_3 = 1$ and $r_{23} = \boldsymbol{q}_2^\dagger \boldsymbol{a}_3 = 0$, so we get $r_{33} = \sqrt{2}\varepsilon$, $\boldsymbol{q}_3 = \begin{bmatrix} 0 & -1 & 0 & 1 \end{bmatrix}^{\mathrm{T}} / \sqrt{2}$. The cGS QR factorization is

$$\hat{Q}_{\text{cGS}} = \begin{bmatrix} 1 & 0 & 0 \\ \varepsilon & -1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 1/\sqrt{2} \end{bmatrix}, \qquad \hat{R}_{\text{cGS}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \sqrt{2}\varepsilon & 0 \\ 0 & 0 & \sqrt{2}\varepsilon \end{bmatrix}$$

Indeed, we have $\hat{Q}_{\text{cGS}} \hat{R}_{\text{cGS}} = \hat{A}$, but the matrix $\hat{Q}_{\text{cGS}}$ is far from unitary: $\boldsymbol{q}_{\text{cGS},2} \cdot \boldsymbol{q}_{\text{cGS},3} = 1/2 \neq 0$.

In modified version we have $r_{13} = 1$, and the column $\boldsymbol{a}_3$ is then orthogonalized to $\boldsymbol{q}_1$, becoming $\begin{bmatrix} 0 & -\varepsilon & 0 & \varepsilon \end{bmatrix}^{\mathrm{T}}$. Only after that it is attempted to be orthogonalized to $\boldsymbol{q}_2$: we have $r_{23} = \varepsilon/\sqrt{2}$, the

---

[28] If for some $1 \leq i \leq n$ the matrix element r_{ii} is computed to be 0, then $\boldsymbol{q}_i$ is arbitrarily chosen from unit vectors orthogonal to $\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_{i-1}$. *E.g.*,

$$\begin{bmatrix} 4 & 5.6 \\ 3 & 4.2 \end{bmatrix} = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 5 & 7 \\ 0 & 0 \end{bmatrix}$$

Here, as $\boldsymbol{a}_2 = 7\boldsymbol{q}_1$, the matrix element r_{22} ends up to be zero, so $\boldsymbol{q}_2$ is chosen to be orthogonal to $\boldsymbol{q}_1$.

[29] If we do not want to overwrite $\hat{A}$, then we can copy $\hat{A}$ at the start of the algorithm and do calculations with the copy.

subtraction of $r_{23}\boldsymbol{q}_2$ brings the 3rd column to $\begin{bmatrix} 0 & -\varepsilon/2 & -\varepsilon/2 & \varepsilon \end{bmatrix}^{\mathrm{T}}$. Finally, we get

$$\hat{Q}_{\mathrm{mGS}} = \begin{bmatrix} 1 & 0 & 0 \\ \varepsilon & -1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 0 & 2/\sqrt{6} \end{bmatrix}, \qquad \hat{R}_{\mathrm{mGS}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \sqrt{2}\varepsilon & \varepsilon/\sqrt{2} \\ 0 & 0 & \sqrt{3/2}\varepsilon \end{bmatrix}$$

The matrix $\hat{Q}_{\mathrm{mGS}}$ is much closer to be unitary than $\hat{Q}_{\mathrm{cGS}}$, the dot products $\boldsymbol{q}_{\mathrm{mGS},1}\cdot\boldsymbol{q}_{\mathrm{mGS},2}$ and $\boldsymbol{q}_{\mathrm{mGS},1}\cdot\boldsymbol{q}_{\mathrm{mGS},3}$ are small, of the order of ε, but still non-zero.

6.2 Householder reflections

Consider the following problem: You have a vector $\boldsymbol{x}\in\mathbf{C}^m$. Find a unitary transformation $\hat{V}$ would transform $\boldsymbol{x}$ to a vector $\boldsymbol{y}$ with only non-zero component being the 1st one? (We have $|y_1| = \|\boldsymbol{x}\|_2$ then.)

There are many such transformations, and some of them are [Householder] reflections through a hyperplane orthogonal to $\boldsymbol{v} = \boldsymbol{x} - \mathrm{e}^{\mathrm{i}\theta}\|\boldsymbol{x}\|\boldsymbol{e}_1$, *i.e.*, $\hat{V} = \hat{I}_m - 2\boldsymbol{v}\boldsymbol{v}^\dagger/\|\boldsymbol{v}\|^2$. We have $\hat{V}^\dagger = \hat{V}$, and

$$\hat{V}^\dagger\hat{V} = (\hat{I}_m - 2\boldsymbol{v}\boldsymbol{v}^\dagger/\|\boldsymbol{v}\|^2)(\hat{I}_m - 2\boldsymbol{v}\boldsymbol{v}^\dagger/\|\boldsymbol{v}\|^2) = \hat{I}_m - 4\boldsymbol{v}\boldsymbol{v}^\dagger/\|\boldsymbol{v}\|^2 + 4\boldsymbol{v}\boldsymbol{v}^\dagger\boldsymbol{v}\boldsymbol{v}^\dagger/\|\boldsymbol{v}\|^4 = \hat{I}_m$$

Thus, $\hat{V}$ is unitary. It transforms vector $\boldsymbol{x}$ to

$$\hat{V}\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}\boldsymbol{v}^\dagger\boldsymbol{x}/\|\boldsymbol{v}\|^2 = \boldsymbol{x} - 2\boldsymbol{v}(\|\boldsymbol{x}\|^2 - \mathrm{e}^{-\mathrm{i}\theta}\|\boldsymbol{x}\|x_1)/\|\boldsymbol{v}\|^2 =$$
$$= \Big(\underbrace{\big(2\|\boldsymbol{x}\|^2 - x_1^*\mathrm{e}^{\mathrm{i}\theta}\|\boldsymbol{x}\| - x_1\mathrm{e}^{-\mathrm{i}\theta}\|\boldsymbol{x}\|\big)}_{\|\boldsymbol{v}\|^2}\boldsymbol{x} - 2\underbrace{\big(\boldsymbol{x} - \mathrm{e}^{\mathrm{i}\theta}\|\boldsymbol{x}\|\boldsymbol{e}_1\big)}_{\boldsymbol{v}}\underbrace{\big(\|\boldsymbol{x}\|^2 - \mathrm{e}^{-\mathrm{i}\theta}\|\boldsymbol{x}\|x_1\big)}_{\boldsymbol{v}^\dagger\boldsymbol{x}}\Big)/\|\boldsymbol{v}\|^2$$

In order for $\hat{V}\boldsymbol{x}$ to be proportional to $\boldsymbol{e}_1$, we need to have $\mathrm{e}^{2\mathrm{i}\theta} = x_1/x_1^*$, *i.e.*, $\theta = \arg x_1$ or $\theta = \arg x_1 + \pi$.

Definition 6: A *Householder reflection* for vector $\boldsymbol{x}\in\mathbf{C}^m$ is one of the two unitary transformations $\hat{H}_\pm(\boldsymbol{x}) \coloneqq \hat{I}_m - 2\boldsymbol{v}_\pm\boldsymbol{v}_\pm^\dagger/\|\boldsymbol{v}_\pm\|^2$, where $\boldsymbol{v}_\pm \coloneqq \boldsymbol{x} \pm \big(x_1/|x_1|\big)\|\boldsymbol{x}\|\boldsymbol{e}_1$.[30] We have $\hat{H}_\pm(\boldsymbol{x})\boldsymbol{x} = \mp\big(x_1/|x_1|\big)\|\boldsymbol{x}\|\boldsymbol{e}_1$.

Householder QR factorization

```
for i = 1, 2, ..., n do
    x := Â_{i:m,i}                                   (m − i + 1) × 1 matrix or a vector
    v_i := x ± (x_1/|x_1|) ||x|| e_1                 + sign is better for numerical stability
    v_i := v_i / ||v_i||_2                           normalize, so we don't need to divide by ||v_i||_2^2 later
    Â_{i:m,i:n} := Â_{i:m,i:n} − 2 v_i (v_i^† Â_{i:m,i:n})
return Â, v_1, v_2, ..., v_n
```

The vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ could be used to reconstruct the matrix $\hat{Q}$.

Example 6, continued: Let us now compute the QR factorization of $\hat{A}$ by Householder reflections. For the 1st column the Householder reflector H_- is formed from the vector $\boldsymbol{v}_- = \begin{bmatrix} 1 & \varepsilon & 0 & 0 \end{bmatrix}^{\mathrm{T}} - \sqrt{1+\varepsilon^2}\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & \varepsilon & 0 & 0 \end{bmatrix}^{\mathrm{T}}$. Thus $\hat{H}_-(\boldsymbol{a}_1)$ reflection is just changing the sign of the 2nd component. We have $\hat{H}_-(\boldsymbol{a}_1)\boldsymbol{a}_1 = \begin{bmatrix} 1 & -\varepsilon & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, *i.e.*, not all components below the 1st one become zero. That is because in our computation the small vector $\boldsymbol{v}_-$ is resulted in almost cancellation of two close vectors, so the direction of $\boldsymbol{v}_-$ suffers from large numerical errors.

[30] If $x_1 = 0$, then $x_1/|x_1|$ could be set to any number with absolute value 1 (*e.g.*, the number 1).

We have $\boldsymbol{v}_+ = \begin{bmatrix} 2 & \varepsilon & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, and

$$\hat{H}_+(\boldsymbol{a}_1)\hat{A} = \left(\hat{I}_4 - 2\frac{\boldsymbol{v}_+\boldsymbol{v}_+^\dagger}{\|\boldsymbol{v}_+\|^2 = 4}\right)\hat{A} = \underbrace{\begin{bmatrix} -1 & -\varepsilon & 0 & 0 \\ -\varepsilon & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\hat{V}_1} \begin{bmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & -\varepsilon & -\varepsilon \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix}}_{\hat{R}_1}$$

Then, for the 2nd column we have $\boldsymbol{x} = \begin{bmatrix} -\varepsilon & \varepsilon & 0 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{v}_+ = \varepsilon\begin{bmatrix} -(\sqrt{2}+1) & 1 & 0 \end{bmatrix}^{\mathrm{T}}$, and

$$\hat{V}_2\hat{V}_1\hat{A} = \hat{V}_2\hat{R}_1 = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\hat{V}_2} \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & -\varepsilon & -\varepsilon \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix}}_{\hat{R}_1} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & \sqrt{2}\varepsilon & \varepsilon/\sqrt{2} \\ 0 & 0 & -\varepsilon/\sqrt{2} \\ 0 & 0 & \varepsilon \end{bmatrix}}_{\hat{R}_2}$$

Finally, for the 3rd column we have $\boldsymbol{x} = \begin{bmatrix} -\varepsilon/\sqrt{2} & \varepsilon \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{v}_+ = \varepsilon\begin{bmatrix} -(\sqrt{3}+1)/\sqrt{2} & 1 \end{bmatrix}^{\mathrm{T}}$, and

$$\hat{V}_3\hat{V}_2\hat{V}_1\hat{A} = \hat{V}_3\hat{R}_2 = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/\sqrt{3} & \sqrt{2/3} \\ 0 & 0 & \sqrt{2/3} & 1/\sqrt{3} \end{bmatrix}}_{\hat{V}_3} \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & \sqrt{2}\varepsilon & \varepsilon/\sqrt{2} \\ 0 & 0 & -\varepsilon/\sqrt{2} \\ 0 & 0 & \varepsilon \end{bmatrix}}_{\hat{R}_2} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & \sqrt{2}\varepsilon & \varepsilon/\sqrt{2} \\ 0 & 0 & \sqrt{3/2}\varepsilon \\ 0 & 0 & 0 \end{bmatrix}}_{\hat{R}}$$

6.3 Givens rotations

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

where $r = \sqrt{a^2+b^2}$, and $(c,s) = (a,b)/r$. We have $c = \cos\varphi$ and $s = \sin\varphi$, where $\varphi = \mathtt{atan2}(b,a)$.

Problems and exercises

1. Compute the QR factorization of $\hat{A}$ using both classical and modified Gram–Schmidt methods, and by Householder reflections. Assume that $\varepsilon^2 \ll \varepsilon_{\mathrm{machine}} \ll \varepsilon$ in your calculations (*i.e.*, drop ε^2 terms when added to something of the order of 1).

$$\hat{A} = \begin{bmatrix} 3 & 3+4\varepsilon & 7 \\ 4 & 4-3\varepsilon & 1 \\ 5\varepsilon & -7\varepsilon & -12 \end{bmatrix}$$

2. Write programs that compute the QR factorization $\hat{A} = \hat{Q}\hat{R}$ of a matrix by classical and modified Gram–Schmidt methods, and by Householder reflections (with generation of the matrix $\hat{Q}$). Test them (how big is the residual $\hat{A} - \hat{Q}\hat{R}$, how close is $\hat{Q}^\dagger\hat{Q}$ to the identity matrix) on 10×10 Hilbert matrix $\hat{H}_{ij} = 1/(i+j-1)$.

3. Consider 1001×5 matrix $\hat{A}$ with $A_{ij} = x_i^{j-1}\exp(-x_i^2/2)$, where $x_i = 0.01(i-501)$, $1 \le i \le 1001$, $1 \le j \le 5$. Find the QR factorization $\hat{A} = \hat{Q}\hat{R}$ by Gram–Schmidt method, and plot the vectors $\boldsymbol{q}_j/\sqrt{0.01} = 10\boldsymbol{q}_j$, $1 \le j \le 5$ as functions of x_i.[31]

[31] The result is related to Hermite functions [and Hermite polynomials].

4. Consider Lie groups of all complex/real invertible $n \times n$ matrices $\mathrm{GL}(n,\mathbf{C})/\mathrm{GL}(n,\mathbf{R})$. They contain subgroups of all unitary/orthogonal matrices $\mathrm{U}(n)/\mathrm{O}(n)$ and all complex/real upper triangular matrices with strictly positive diagonal entries $\mathrm{T}_+(n,\mathbf{C})/\mathrm{T}_+(n,\mathbf{R})$. Find [real] dimensions of all the 6 mentioned Lie groups. Calculate $\dim\mathrm{GL}(n,\mathbf{C}) - \dim\mathrm{U}(n) - \dim\mathrm{T}_+(n,\mathbf{C})$ and $\dim\mathrm{GL}(n,\mathbf{R}) - \dim\mathrm{O}(n) - \dim\mathrm{T}_+(n,\mathbf{R})$.

7 Gaussian elimination, LU factorization

$$\boxed{\hat{I}_n}\ \boxed{\hat{A}} \longrightarrow \underbrace{\boxed{\hat{L}_{j-1}}\ \boxed{\hat{T}_j^{-1}}}_{\hat{L}_j}\ \underbrace{\boxed{\hat{T}_j}\ \boxed{\hat{U}_{j-1}}}_{\hat{U}_j} \longrightarrow \boxed{\hat{L}}\ \boxed{\hat{U}}$$

$$\hat{L} = \hat{T}_1^{-1}\hat{T}_2^{-1}\hat{T}_3^{-1}... \qquad \hat{U} = ...\hat{T}_3\hat{T}_2\hat{T}_1\hat{A}$$

The process is similar to QR factorization by Householder reflections, but instead of unitary transformation $\hat{V}_k$ the lower triangular $\hat{T}_j$ is employed here. Column by column of $\hat{A}$ we make its content below the main diagonal being 0. Here is an example of a pseudo-code that produces LU factorization:

for $j = 1, 2, ..., n-1$ do
  for $i = j+1,\ j+2, ..., n$ do — application of $\hat{T}_j$
    $l_{ij} := a_{ij}/a_{jj}$
    for $k = j,\ j+1, ..., n$ do — row operation $\hat{A}_{i,j:n} := \hat{A}_{i,j:n} - l_{ij}\hat{A}_{j,j:n}$
      $a_{ik} := \hat{a}_{ik} - l_{ij}a_{jk}$ — a_{ij} becomes $a_{ij} - (l_{ij} = a_{ij}/a_{jj})a_{jj} = 0$
return $\hat{L}, \hat{A}$

The matrices $\hat{T}_j$ and $\hat{T}_j^{-1}$ look like (empty spaces correspond to zero matrix elements)[32]

$$\hat{T}_j = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & -l_{j+1,j} & 1 & & \\ & & -l_{ij} & & 1 & \\ & & -l_{nj} & & & 1 \end{bmatrix}, \qquad \hat{T}_j^{-1} = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & l_{j+1,j} & 1 & & \\ & & l_{ij} & & 1 & \\ & & l_{nj} & & & 1 \end{bmatrix}$$

$$\hat{T}_j \begin{bmatrix} \cdot \\ \cdot \\ a_{jj} \\ a_{j+1,j} \\ a_{ij} \\ a_{nj} \end{bmatrix} = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & -l_{j+1,j} & 1 & & \\ & & -l_{ij} & & 1 & \\ & & -l_{nj} & & & 1 \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \\ a_{jj} \\ a_{j+1,j} \\ a_{ij} \\ a_{nj} \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ a_{jj} \\ -(a_{j+1,j}/a_{jj})a_{jj} + a_{j+1,j} \\ -(a_{ij}/a_{jj})a_{jj} + a_{ij} \\ -(a_{nj}/a_{jj})a_{jj} + a_{nj} \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ a_{jj} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

[32] The simplest case to self-check the formula for $\hat{T}_j^{-1}$ is $\begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ -l & 1 \end{bmatrix} = \begin{bmatrix} 1\cdot 1 + 0\cdot(-l) & 1\cdot 0 + 0\cdot 1 \\ l\cdot 1 + 1\cdot(-l) & l\cdot 0 + 1\cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

$$
\hat{L}_j = \underbrace{\begin{bmatrix} 1 & & & & & \\ l_{21} & 1 & & & & \\ l_{j1} & l_{j,j-1} & 1 & & & \\ l_{j+1,1} & l_{j+1,j-1} & & 1 & & \\ l_{i1} & l_{i,j-1} & & & 1 & \\ l_{n1} & l_{n,j-1} & & & & 1 \end{bmatrix}}_{\hat{L}_{j-1}=\hat{T}_1^{-1}\hat{T}_2^{-1}\ldots\hat{T}_{j-1}^{-1}} \underbrace{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & l_{j+1,j} & 1 & & \\ & & l_{ij} & & 1 & \\ & & l_{nj} & & & 1 \end{bmatrix}}_{\hat{T}_j^{-1}} = \underbrace{\begin{bmatrix} 1 & & & & & \\ l_{21} & 1 & & & & \\ l_{j1} & l_{j,j-1} & 1 & & & \\ l_{j+1,1} & l_{j+1,j-1} & l_{j+1,j} & 1 & & \\ l_{i1} & l_{i,j-1} & l_{ij} & & 1 & \\ l_{n1} & l_{n,j-1} & l_{nj} & & & 1 \end{bmatrix}}_{\hat{L}_j}
$$

Example 7: Consider the matrix $\hat{A}$ and its LU factorization:[33]

$$
\hat{A} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 4 & 1 & -5 & 0 \\ 3 & -4 & 0 & 1 \\ 1 & -2 & 4 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & & 1 & \\ 1 & & & 1 \end{bmatrix}}_{\hat{L}_1=\hat{T}_1^{-1}} \overbrace{\underbrace{\begin{bmatrix} 1 & & & \\ -4 & 1 & & \\ -3 & & 1 & \\ -1 & & & 1 \end{bmatrix}}_{\hat{T}_1} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 4 & 1 & -5 & 0 \\ 3 & -4 & 0 & 1 \\ 1 & -2 & 4 & 1 \end{bmatrix}}^{\hat{U}_1} =
$$

$$
= \begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & & 1 & \\ 1 & & & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -4 & 3 & 1 \\ 0 & -2 & 5 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & -4 & 1 & \\ 1 & -2 & & 1 \end{bmatrix}}_{\hat{L}_2=\hat{L}_1\hat{T}_2^{-1}} \overbrace{\underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & 4 & 1 & \\ & 2 & & 1 \end{bmatrix}}_{\hat{T}_2} \begin{bmatrix} 1 & 0 & -1 & 0 \\ & 1 & -1 & 0 \\ & -4 & 3 & 1 \\ & -2 & 5 & 1 \end{bmatrix}}^{\hat{U}_2} =
$$

$$
= \begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & -4 & 1 & \\ 1 & -2 & & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 & 0 \\ & 1 & -1 & 0 \\ & 0 & -1 & 1 \\ & 0 & 3 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & -4 & 1 & \\ 1 & -2 & -3 & 1 \end{bmatrix}}_{\hat{L}_3=\hat{L}_2\hat{T}_3^{-1}} \overbrace{\underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & 3 & 1 \end{bmatrix}}_{\hat{T}_3} \begin{bmatrix} 1 & 0 & -1 & 0 \\ & 1 & -1 & 0 \\ & & -1 & 1 \\ & & 3 & 1 \end{bmatrix}}^{\hat{U}_3} =
$$

$$
= \begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & -4 & 1 & \\ 1 & -2 & -3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 & 0 \\ & 1 & -1 & 0 \\ & & -1 & 1 \\ & & 0 & 4 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ 4 & 1 & & \\ 3 & -4 & 1 & \\ 1 & -2 & -3 & 1 \end{bmatrix}}_{\hat{L}} \underbrace{\begin{bmatrix} 1 & 0 & -1 & 0 \\ & 1 & -1 & 0 \\ & & -1 & 1 \\ & & & 4 \end{bmatrix}}_{\hat{U}} = \hat{L}\hat{U}
$$

[33] The matrix $\hat{A}$ is chosen in such a way that the matrices $\hat{L}$ and $\hat{U}$ end up being integer. Here $\kappa(\hat{A}) \approx 129.1$, $\kappa(\hat{L}) \approx 414.4$, and $\kappa(\hat{U}) \approx 8.18$.

LU factorization with partial pivoting:[34]

$$
\hat{A} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 4 & 1 & -5 & 0 \\ 3 & -4 & 0 & 1 \\ 1 & -2 & 4 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \begin{bmatrix} 4 & 1 & -5 & 0 \\ 1 & 0 & -1 & 0 \\ 3 & -4 & 0 & 1 \\ 1 & -2 & 4 & 1 \end{bmatrix} =
$$

$$
= \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{1}{4} & 1 & & \\ \frac{3}{4} & & 1 & \\ \frac{1}{4} & & & 1 \end{bmatrix}}_{\hat{L}_1 = \hat{T}_1^{-1}} \underbrace{\begin{bmatrix} 4 & 1 & -5 & 0 \\ 0 & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & -\frac{19}{4} & \frac{15}{4} & 1 \\ 0 & -\frac{9}{4} & \frac{21}{4} & 1 \end{bmatrix}}_{\hat{U}_1} =
$$

$$
= \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{1}{4} & 1 & & \\ \frac{3}{4} & & 1 & \\ \frac{1}{4} & & & 1 \end{bmatrix}}_{\hat{L}_1 = \hat{T}_1^{-1}} \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & -5 & 0 \\ & -\frac{19}{4} & \frac{15}{4} & 1 \\ & -\frac{1}{4} & \frac{1}{4} & 0 \\ & -\frac{9}{4} & \frac{21}{4} & 1 \end{bmatrix} =
$$

$$
= \underbrace{\begin{bmatrix} & & 1 & \\ 1 & & & \\ & 1 & & \\ & & & 1 \end{bmatrix}}_{\hat{P}_2} \begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{4} & & 1 & \\ \frac{1}{4} & & & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & \frac{1}{19} & 1 & \\ & \frac{9}{19} & & 1 \end{bmatrix}}_{\hat{T}_2^{-1}} \underbrace{\begin{bmatrix} 4 & 1 & -5 & 0 \\ & -\frac{19}{4} & \frac{15}{4} & 1 \\ & 0 & \frac{1}{19} & -\frac{1}{19} \\ & 0 & \frac{66}{19} & \frac{10}{19} \end{bmatrix}}_{\hat{U}_2} =
$$

$$
= \begin{bmatrix} & & 1 & \\ 1 & & & \\ & 1 & & \\ & & & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{4} & \frac{1}{19} & 1 & \\ \frac{1}{4} & \frac{9}{19} & & 1 \end{bmatrix}}_{\hat{L}_2} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{bmatrix} \begin{bmatrix} 4 & 1 & -5 & 0 \\ & -\frac{19}{4} & \frac{15}{4} & 1 \\ & & \frac{66}{19} & \frac{10}{19} \\ & & \frac{1}{19} & -\frac{1}{19} \end{bmatrix} =
$$

$$
= \underbrace{\begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{bmatrix}}_{\hat{P}_3} \begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{4} & \frac{9}{19} & 1 & \\ \frac{1}{4} & \frac{1}{19} & & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & \frac{1}{66} & 1 \end{bmatrix}}_{\hat{T}_3^{-1}} \begin{bmatrix} 4 & 1 & -5 & 0 \\ & -\frac{19}{4} & \frac{15}{4} & 1 \\ & & \frac{66}{19} & \frac{10}{19} \\ & & 0 & -\frac{2}{33} \end{bmatrix} =
$$

$$
= \underbrace{\begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{bmatrix}}_{\hat{P}} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{4} & \frac{9}{19} & 1 & \\ \frac{1}{4} & \frac{1}{19} & \frac{1}{66} & 1 \end{bmatrix}}_{\hat{L}} \underbrace{\begin{bmatrix} 4 & 1 & -5 & 0 \\ & -\frac{19}{4} & \frac{15}{4} & 1 \\ & & \frac{66}{19} & \frac{10}{19} \\ & & & -\frac{2}{33} \end{bmatrix}}_{\hat{U}}
$$

[34] We get $\kappa(\hat{L}) \approx 2.46$ and $\kappa(\hat{U}) \approx 144.8$.

LU factorization with complete pivoting:[35]

$$
\begin{aligned}
\hat{A} &= \begin{bmatrix} 1 & 0 & -1 & 0 \\ 4 & 1 & -5 & 0 \\ 3 & -4 & 0 & 1 \\ 1 & -2 & 4 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \begin{bmatrix} -5 & 1 & 4 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -4 & 3 & 1 \\ 4 & -2 & 1 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}_1} = \\
&= \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{1}{5} & 1 & & \\ 0 & & 1 & \\ -\frac{4}{5} & & & 1 \end{bmatrix}}_{\hat{L}_1=\hat{T}_1^{-1}} \underbrace{\begin{bmatrix} -5 & 1 & 4 & 0 \\ 0 & -\frac{1}{5} & \frac{1}{5} & 0 \\ 0 & -4 & 3 & 1 \\ 0 & -\frac{6}{5} & \frac{21}{5} & 1 \end{bmatrix}}_{\hat{U}_1} \underbrace{\begin{bmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}_1} = \\
&= \underbrace{\begin{bmatrix} & 1 & & \\ 1 & & & \\ & & 1 & \\ & & & 1 \end{bmatrix}}_{\hat{P}_1} \underbrace{\begin{bmatrix} 1 & & & \\ \frac{1}{5} & 1 & & \\ 0 & & 1 & \\ -\frac{4}{5} & & & 1 \end{bmatrix}}_{\hat{L}_1=\hat{T}_1^{-1}} \begin{bmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{bmatrix} \begin{bmatrix} -5 & 4 & 1 & 0 \\ & \frac{21}{5} & -\frac{6}{5} & 1 \\ & 3 & -4 & 1 \\ & \frac{1}{5} & -\frac{1}{5} & 0 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \underbrace{\begin{bmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}_1} = \\
&= \underbrace{\begin{bmatrix} & & & 1 \\ 1 & & & \\ & & 1 & \\ & 1 & & \end{bmatrix}}_{\hat{P}_2} \begin{bmatrix} 1 & & & \\ -\frac{4}{5} & 1 & & \\ 0 & & 1 & \\ \frac{1}{5} & & & 1 \end{bmatrix} \begin{bmatrix} -5 & 4 & 1 & 0 \\ & \frac{21}{5} & -\frac{6}{5} & 1 \\ & 3 & -4 & 1 \\ & \frac{1}{5} & -\frac{1}{5} & 0 \end{bmatrix} \underbrace{\begin{bmatrix} & & 1 & \\ 1 & & & \\ & 1 & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}_2} = \\
&= \underbrace{\begin{bmatrix} & & & 1 \\ 1 & & & \\ & & 1 & \\ & 1 & & \end{bmatrix}}_{\hat{P}_2} \underbrace{\begin{bmatrix} 1 & & & \\ -\frac{4}{5} & 1 & & \\ 0 & \frac{5}{7} & 1 & \\ \frac{1}{5} & \frac{1}{21} & & 1 \end{bmatrix}}_{\hat{L}_2} \underbrace{\begin{bmatrix} -5 & 4 & 1 & 0 \\ & \frac{21}{5} & -\frac{6}{5} & 1 \\ & 0 & -\frac{22}{7} & \frac{2}{7} \\ & 0 & -\frac{1}{7} & -\frac{1}{21} \end{bmatrix}}_{\hat{U}_2} \underbrace{\begin{bmatrix} & & 1 & \\ 1 & & & \\ & 1 & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}_2} = \\
&= \underbrace{\begin{bmatrix} & & & 1 \\ 1 & & & \\ & & 1 & \\ & 1 & & \end{bmatrix}}_{\hat{P}=\hat{P}_2} \underbrace{\begin{bmatrix} 1 & & & \\ -\frac{4}{5} & 1 & & \\ 0 & \frac{5}{7} & 1 & \\ \frac{1}{5} & \frac{1}{21} & \frac{1}{22} & 1 \end{bmatrix}}_{\hat{L}} \underbrace{\begin{bmatrix} -5 & 4 & 1 & 0 \\ & \frac{21}{5} & -\frac{6}{5} & 1 \\ & & -\frac{22}{7} & \frac{2}{7} \\ & & 0 & -\frac{2}{33} \end{bmatrix}}_{\hat{U}} \underbrace{\begin{bmatrix} & & 1 & \\ 1 & & & \\ & 1 & & \\ & & & 1 \end{bmatrix}}_{\hat{Q}=\hat{Q}_2} =
\end{aligned}
$$

Problems and exercises

1. Write a program that solves the square system $\hat{A}\boldsymbol{x} = \boldsymbol{b}$ by Gaussian elimination with partial pivoting. Test it on 10×10 Hilbert matrix $\hat{H}_{ij} = 1/(i+j-1)$: First compute the r.h.s. vector $\boldsymbol{b} = \hat{H}\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}^{\mathrm{T}}$, and then solve the system $\hat{H}\boldsymbol{x} = \boldsymbol{b}$.

[35] We get $\kappa(\hat{L}) \approx 2.92$ and $\kappa(\hat{U}) \approx 121.2$.

8 Eigenvalues

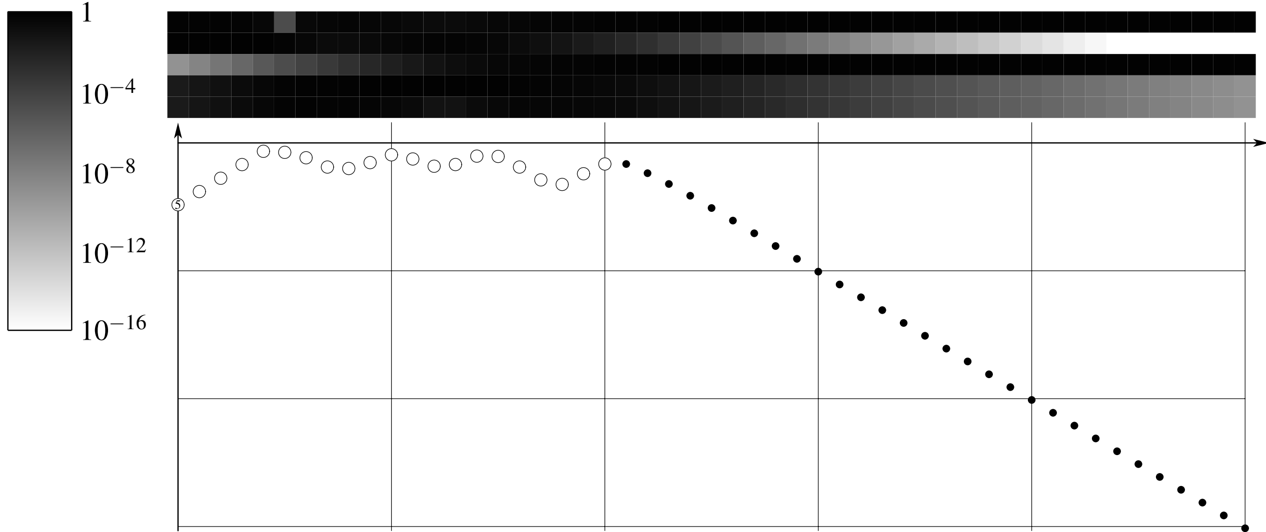
Algorithm $\det(\lambda\hat{I}-\hat{A})=0$: Calculate the characteristic polynomial of the matrix, then find its roots. This is an unstable algorithm.

Algorithm "power iteration": Choose arbitrarily a vector $\boldsymbol{x}_0$, then calculate $\boldsymbol{x}_n := \hat{A}\boldsymbol{x}_{n-1}$ for $n=1$, 2, ... Renormalize the vector $\boldsymbol{x}_n$ when needed. This method finds an eigenvector for the eigenvalue with the largest absolute value.

Exapmpe 8: Consider the following matrix:

$$\hat{A}=\begin{bmatrix} 8 & 7 & -8 & -7 & -1 \\ 6 & 7 & 6 & -3 & 3 \\ -8 & -8 & 8 & 8 & 0 \\ -2 & -2 & -2 & 6 & 2 \\ -6 & -6 & -6 & 6 & 2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 \end{bmatrix}}_{\hat{V}} \underbrace{\begin{bmatrix} 16 & & & & \\ & 8 & & & \\ & & 4 & & \\ & & & 2 & \\ & & & & 1 \end{bmatrix}}_{\hat{D}} \underbrace{\begin{bmatrix} 1 & 1 & 0 & -1 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ 2 & 2 & 2 & -1 & 1 \\ -1 & -1 & -1 & 1 & -1 \\ -2 & -1 & -2 & 1 & -1 \end{bmatrix}}_{\hat{V}^{-1}}$$

The columns of $\hat{V}$ are the eigenvectors of $\hat{A}$: $\hat{A}\boldsymbol{v}_j = 2^{5-j}\boldsymbol{v}_j$, $j=1, 2, 3, 4, 5$. Let us choose $\boldsymbol{x}_0 := \boldsymbol{v}_1 + 2^{16}\boldsymbol{v}_2 + 2^{28}\boldsymbol{v}_3 + 2^{36}\boldsymbol{v}_2 + 2^{40}\boldsymbol{v}_5$, normalize it $\boldsymbol{x}_0 := \boldsymbol{x}_0/\|\boldsymbol{x}_0\|_2$, and then do power iterations $\boldsymbol{x}_n = \hat{A}\boldsymbol{x}_{n-1}$, $\boldsymbol{x}_n := \boldsymbol{x}_n/\|\boldsymbol{x}_n\|_2$ for $n=1, 2, 3, ..., 50$:



The graph shows the smallest angle to one the eigenvalues $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$, $\boldsymbol{v}_4$, $\boldsymbol{v}_5$. Here the vector $\boldsymbol{x}_0$ is chosen in such a way, that each eigenvector is dominant at some iteration. Eventually the iterations converge to $\boldsymbol{v}_1$, the eigenvector with the largest in absolute value eigenvalue.

Algorithm "inverse iteration": Choose a number μ. Do power iteration for $(\hat{A}-\mu\hat{I})^{-1}$. This method finds an eigenvector for the eigenvalue closest to μ.

Definition 8.1: Let $\hat{A}$ be an $n\times n$ matrix. The *Rayleigh quotient* of a vector $\boldsymbol{x}$ is the $R(\hat{A},\boldsymbol{x}) := \big(\boldsymbol{x}^\dagger\hat{A}\boldsymbol{x}\big)/(\boldsymbol{x}^\dagger\boldsymbol{x})$. If $\boldsymbol{x}$ is an eigenvector of the matrix $\hat{A}$ with eigenvalue λ, then $R(\hat{A},\boldsymbol{x})=\lambda$.

Algorithm "Rayleigh quotient iteration": Choose arbitrarily a vector $\boldsymbol{x}_0$, then calculate $\boldsymbol{x}_n := \big(\hat{A} - R(\hat{A},\boldsymbol{x}_{n-1})\hat{I}\big)^{-1}\boldsymbol{x}_{n-1}$ for $n=1$, 2, ... Renormalize the vector $\boldsymbol{x}_n$ when needed.

Definition/Theorem 8.2: Any square $n\times n$ matrix $\hat{A}$ has a *Schur decomposition* $\hat{A}=\hat{Q}\hat{T}\hat{Q}^\dagger$, where matrix $\hat{Q}$ is unitary, while matrix $\hat{T}$ is upper triangular. This is a similarity transformation, and diagonal elements of $\hat{T}$ are the eigenvalues of $\hat{A}$.

Definition 8.3: A *Hessenberg decomposition* of a square $n \times n$ matrix $\hat{A}$ is $\hat{A} = \hat{Q}\hat{H}\hat{Q}^\dagger$, where matrix $\hat{Q}$ is unitary, and matrix $\hat{H}$ is such that $H_{ij} = 0$ if $i > j+1$. This is a similarity transformation, and the eigenvalues of $\hat{A}$ and of $\hat{H}$ are the same.

The Hessenberg form of a matrix $\hat{A}$ can be ontained with Householder reflections $O(n^3)$ operations. The algorithm is simialar to QR factorization, but one leaves one more component non-zero. This allows the zeros of the formed matrix being not destroyed by multiplying by $\hat{V}^\dagger$ from the right.

Problems and exercises

1. Consider the 100×100 matrix $\hat{A}$ with $a_{ii} = -2$, and $a_{i+1,i} = a_{i-1,i} = 1$ for all $i = 1, 2, \ldots 100$. The values $0/101$ of the index are identified with $100/1$. The matrix is cyclic shift invariant ($a_{ij} = a_{i+k,j+k}$), so discrete Fourier transform diagonalizes it. Consider $\boldsymbol{x}_0 = \boldsymbol{e}_{50}$. (a) Do power iterations $\boldsymbol{x}_n := \hat{A}\boldsymbol{x}_{n-1}$, $\boldsymbol{x}_n := \boldsymbol{x}_n/\|\boldsymbol{x}_n\|_2$. Plot $\boldsymbol{x}_{100}$, $\boldsymbol{x}_{1000}$, and $\boldsymbol{x}_{10000}$. Guess the maximal in absolute value eigenvalue $\lambda_{\max}$ and the eigenvector $\boldsymbol{x}_{\max}$ corresponding to it. (b) Let $\mu = -4.001$. Plot $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, and $\boldsymbol{x}_4$ for the inverse power iteration $\boldsymbol{x}_n := (\hat{A} - \mu\hat{I})^{-1}\boldsymbol{x}_{n-1}$, $\boldsymbol{x}_n := \boldsymbol{x}_n/\|\boldsymbol{x}_n\|_2$ ($\boldsymbol{x}_n$ is found as the solution of the system $(\hat{A} - \mu\hat{I})\boldsymbol{x}_n = \boldsymbol{x}_{n-1}$). (c) Do Rayleigh quotient iterations $\mu_0 = -4.001$, $\boldsymbol{x}_n := (\hat{A} - \mu_{n-1}\hat{I})^{-1}\boldsymbol{x}_{n-1}$, $\boldsymbol{x}_n := \boldsymbol{x}_n/\|\boldsymbol{x}_n\|_2$, $\mu_n := R(\hat{A}, \boldsymbol{x}_n) = \boldsymbol{x}_n^\dagger\hat{A}\boldsymbol{x}_n$ up to $n = 5$. Print μ_n and $\|\boldsymbol{x}_n \pm \boldsymbol{x}_{n-1}\|_2$, $n = 1, 2, ..., 5$.

2. Consider the matrix $\hat{A}$ on the right. Its eigenvalues are complex. (a) Do 100 and 101 QR iterations $\hat{Q}\hat{R} := \hat{A}$, $\hat{A} := \hat{R}\hat{Q}$. Can you easily extract eigenvalues of $\hat{A}$, *e.g.*, from $\hat{A}_{1:2,1:2}$? (b) Do 100 shifted QR iterations $\hat{Q}\hat{R} := \hat{A} - \hat{I}_4$, $\hat{A} := \hat{R}\hat{Q} + \hat{I}_4$. Calculate the eigenvalues of the upper left $\hat{A}_{1:2,1:2}$ and lower right $\hat{A}_{3:4,3:4}$ 2×2 corners of the resulted matrix, compare them with the eigenvalues of $\hat{A}$. (c) Do 100 shifted QR iterations $\hat{Q}\hat{R} := \hat{A} - \mu\hat{I}_4$, $\hat{A} := \hat{R}\hat{Q} + \mu\hat{I}_4$ with $\mu = (1+\mathrm{i})/2$. Is $\hat{A}$ close to being an upper triangular?

$$\hat{A} = \begin{bmatrix} 30000 & -29999 & -29999 & 30000 \\ 30001 & -30000 & -30000 & 30001 \\ 9999 & -10000 & 30000 & -29999 \\ 10000 & -10001 & 30001 & -30000 \end{bmatrix}$$

Part III

Systems of nonlinear equations

Consider you are to solve the equation $f(x) = 0$, where f is the continuous real function of one real variable. This problem could be solved by simple but powerful *bisection method*. The idea is the following: If you find such $x_{\text{left}} < x_{\text{right}}$ that $f(x_{\text{left}})$ and $f(x_{\text{right}})$ are of different sign, then there is such x_*, $x_{\text{left}} < x_* < x_{\text{right}}$, that $f(x_*) = 0$. Try $x = (x_{\text{left}} + x_{\text{right}})/2$. If $f(x) = 0$, then you solved the equation. Otherwise check which $f(x_{\text{left}})$ or $f(x_{\text{right}})$ is of different sign with $f(x)$ and narrow the interval [inside which at least one solution lies in] $(x_{\text{left}}, x_{\text{right}})$ to either (x_{left}, x) or (x, x_{right}). We can systematically reduce, each time by factor or 2, the width of the interval containing a solution, until the width of the interval [or uncertainty in solution] is small enough.

Here is how the solution of $x = \cos x$ equation inside the interval $[0, 1]$ is found (we define $f(x) = x - \cos(x)$, we have $f(0) = -1 < 0$ and $f(1) = 1 - \cos(1) > 0$):

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat x_eq_cos_x.py
from math import cos
xl, xr = 0., 1.
print('{0:.8e}                    {1:.8e}'.format(xl, xr))
while (xr - xl > 1.e-6):
```

```
    xm = 0.5 * (xl + xr)
    if (cos(xm) > xm):
        print('               {0:.15e} {1:.8e}'.format(xm, xr))
        xl = xm
    else:
        print('{0:.8e} {1:.15e}'.format(xl, xm))
        xr = xm
[...]/teaching/2019-4/math_575a/notes/Python$ python3 x_eq_cos_x.py
0.00000000e+00                       1.00000000e+00
               5.000000000000000e-01 1.00000000e+00
5.00000000e-01 7.500000000000000e-01
               6.250000000000000e-01 7.50000000e-01
               6.875000000000000e-01 7.50000000e-01
               7.187500000000000e-01 7.50000000e-01
               7.343750000000000e-01 7.50000000e-01
7.34375000e-01 7.421875000000000e-01
               7.382812500000000e-01 7.42187500e-01
7.38281250e-01 7.402343750000000e-01
7.38281250e-01 7.392578125000000e-01
               7.387695312500000e-01 7.39257812e-01
               7.390136718750000e-01 7.39257812e-01
7.39013672e-01 7.391357421875000e-01
               7.390747070312500e-01 7.39135742e-01
7.39074707e-01 7.391052246093750e-01
7.39074707e-01 7.390899658203125e-01
               7.390823364257812e-01 7.39089966e-01
7.39082336e-01 7.390861511230469e-01
               7.390842437744141e-01 7.39086151e-01
7.39084244e-01 7.390851974487305e-01
[...]/teaching/2019-4/math_575a/notes/Python$
```

9 Functional iteration

Quite often a general system of non-linear equations arises (or can be rewritten) in the form $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{f}(\cdot)$ is a n-component vector function, with an n-component vector as an argument.

A common method of solving such a system is by iterations $\boldsymbol{x}^{(n+1)} := \boldsymbol{f}\big(\boldsymbol{x}^{(n)}\big)$ starting from some initial guess $\boldsymbol{x}^{(0)}$. If such iterations do converge, then they converge to a solution. Here is how the equation $x = \cos(x)$ is solved by functional iteration, starting from initial guess $x^{(0)} = 0$:

```
[...]/teaching/2019-4/math_575a/notes/C$ cat x_eq_cos_x.c
#include <stdio.h>
#include <math.h>
int main() { int i; double x;
  for (x = 0., i = 0; i <= 34; i++, x = cos(x)) printf("%8.6f ", x);
  printf("\n"); return 0; }
[...]/teaching/2019-4/math_575a/notes/C$ cc x_eq_cos_x.c -lm
[...]/teaching/2019-4/math_575a/notes/C$ ./a.out
0.000000 1.000000 0.540302 0.857553 0.654290 0.793480 0.701369 0.763960 0.722102
 0.750418 0.731404 0.744237 0.735605 0.741425 0.737507 0.740147 0.738369 0.73956
7 0.738760 0.739304 0.738938 0.739184 0.739018 0.739130 0.739055 0.739106 0.7390
71 0.739094 0.739079 0.739089 0.739082 0.739087 0.739084 0.739086 0.739085
[...]/teaching/2019-4/math_575a/notes/C$
```

```
octave:1> format long
octave:2> options = optimset('TolX', 1.e-13, 'TolFun', 1.e-13);
octave:3> x = fsolve(@(x) (x - cos(x)), 0., options)
x =    7.390851332151714e-01
octave:4> y = cos(x)
y =    7.390851332151533e-01
```

The stability of the functional iterations at the solution $\boldsymbol{x}_* = \boldsymbol{f}(\boldsymbol{x}_*)$ could be obtained from the linearization of $\boldsymbol{f}$ at $\boldsymbol{x} = \boldsymbol{x}_*$.

10 Newton–Raphson method

Let us try to solve $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ system of equations. It is complicated as the vector function $\boldsymbol{f}$ is non-linear. Consider we have a guess for a solution, $\boldsymbol{x}_0$. We can approximate $\boldsymbol{f}$ by its tangent line approximation: $\boldsymbol{f}(\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{\delta}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (\nabla \boldsymbol{f})(\boldsymbol{x}_0) \cdot \boldsymbol{\delta}$. The truncation of the Taylor series in $\boldsymbol{\delta}$ is motivated by our expectation that $\boldsymbol{\delta}$ is small. Our system of equations becomes $\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (\nabla \boldsymbol{f})(\boldsymbol{x}_0) \cdot \boldsymbol{\delta} = \boldsymbol{0}$, which is a system of linear equations for the components of vector $\boldsymbol{\delta}$. We have

$$\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{\delta} \approx \boldsymbol{x}_0 + \big(\text{solution of the } (\nabla \boldsymbol{f})(\boldsymbol{x}_0) \cdot \boldsymbol{\delta} = -\boldsymbol{f}(\boldsymbol{x}_0) \text{ system}\big) = \boldsymbol{x}_0 - \big((\nabla \boldsymbol{f})(\boldsymbol{x}_0)\big)^{-1} \boldsymbol{f}(\boldsymbol{x}_0)$$

The method of solving the system of equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ based on iterations $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \big((\nabla \boldsymbol{f})(\boldsymbol{x}_n)\big)^{-1} \boldsymbol{f}(\boldsymbol{x}_n)$ is called the Newton–Raphson method.

There is no guarantee that these iterations are going to converge, but when they do they converge very fast: $\boldsymbol{f}(\boldsymbol{x}_{n+1}) = \boldsymbol{f}\Big(\boldsymbol{x}_n - \big((\nabla \boldsymbol{f})(\boldsymbol{x}_n)\big)^{-1} \boldsymbol{f}(\boldsymbol{x}_n)\Big) = \boldsymbol{f}(\boldsymbol{x}_n) - (\nabla \boldsymbol{f})(\boldsymbol{x}_n) \cdot \big((\nabla \boldsymbol{f})(\boldsymbol{x}_n)\big)^{-1} \boldsymbol{f}(\boldsymbol{x}_n) + \frac{1}{2}(\nabla\nabla \boldsymbol{f})(\boldsymbol{x}_n) \cdot \Big(\big((\nabla \boldsymbol{f})(\boldsymbol{x}_n)\big)^{-1} \boldsymbol{f}(\boldsymbol{x}_n)\Big)^2 \propto (\boldsymbol{f}(\boldsymbol{x}_n))^2$. The "error" in the next step is the square of the error in previous step.

Example 10.1: Let us compute $\sqrt{2}$. We may construct a function $f(x)$ such that $f(\sqrt{2}) = 0$, and then find the root by the Newton–Raphson method. Let $f(x) := x^2 - 2$. Then $f'(x) = 2x$, and the update rule reads as $x_{n+1} = x_n - (x_n^2 - 2)/2x_n = x_n/2 + 1/x_n$. If we start from $x_0 = 1$ or $x_0 = 2$, we have $x_1 = 1/2 + 1/1 = 2/2 + 1/2 = 3/2$. Then $x_2 = x_1/2 + 1/x_1 = 3/4 + 2/3 = (9+8)/12 = 17/12 = 1.41666...$ (notice that $17^2 = 289 \approx 288 = 2 \cdot 12^2$). We have $x_3 = 17/24 + 12/17 = (17^2 + 24 \cdot 12)/24 \cdot 17 = 577/408 = 1.41421568...$ (notice that $577^2 = 332929 \approx 332928 = 2 \cdot 408^2$). Next $x_4 = 665857/470832 = 1.41421356237468...$, while $\sqrt{2} = 1.41421356237309...$ The Newton–Raphson iterations very quickly converge to $\sqrt{2}$, at each iteration the number of correct significant digits is doubled.

Example 10.2: Consider the system $y = x^2$, $xy = 1$. We can write it as $\boldsymbol{f}(x, y)$ by setting $f_1(x, y) = y - x^2$ and $f_2(x, y) = xy - 1$. The matrix $\nabla \boldsymbol{f}$ and the iterations look like

$$\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} -2x & 1 \\ y & x \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} -2x & 1 \\ y & x \end{bmatrix}^{-1} \begin{bmatrix} y - x^2 \\ xy - 1 \end{bmatrix} = \begin{bmatrix} y - x^2 \\ xy - 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \frac{1}{2x^2 + y} \begin{bmatrix} x & -1 \\ -y & -2x \end{bmatrix} \begin{bmatrix} y - x^2 \\ xy - 1 \end{bmatrix} =$$
$$= \frac{1}{2x^2 + y} \begin{bmatrix} 1 + xy + x^3 \\ x(2 + xy) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{\alpha}{3 + 4\alpha + \beta + 2\alpha^2} \begin{bmatrix} \beta + \alpha + \alpha^2 \\ 2\beta - \alpha + \alpha\beta \end{bmatrix}$$

where $x = 1+\alpha$, $y = 1+\beta$. We have $x = y = 1$ being a solution, with the deviation from it being about squared in magnitude with each iteration:

```
octave:1> format long
octave:2> f = @(x) ([1 + x(1) * x(2) + (x(1))^3, x(1) * (2 + x(1) * x(2))] / (2
* (x(1))^2 + x(2)));
octave:3> x0 = [2, 0];
octave:4> [x0 ; f(x0); f(f(x0)); f(f(f(x0))); f(f(f(f(x0)))); f(f(f(f(f(x0)))))]
ans =

   2.000000000000000e+00   0.000000000000000e+00
   1.125000000000000e+00   5.000000000000000e-01
   9.851804123711341e-01   9.510309278350515e-01
   1.000325727687672e+00   1.000422180897834e+00
   1.000000081169684e+00   1.000000056293719e+00
   1.000000000000004e+00   1.000000000000001e+00
```

Problems and exercises

1. Consider the system $y = x^2$, $xy = 1$ (Example 10.2). Find all of its solutions analytically. Do several Newton-Raphson iterations starting from $x_0 = -1+\mathrm{i}$, $y_0 = 0$. Do you converge to a solution, and if yes, to which one?

2. Consider the [transcendental] equation $\mathrm{e}^x = kx$. When $k > \mathrm{e}$, there are two solutions, $x_{\text{small}}(k) < 1$ and $x(k) > 1$. For $\mathrm{e}^2 \le k \le \mathrm{e}^{10}$ log-log plot the solution $x(k)$ found by (a) bisection method, (b) functional iteration (you need to rewrite the equation in $x = F(x)$ form with iterations being converging), and (c) Newton–Raphson method.

Part IV
Numerical ODEs

Suggested reading: [AsPe98].

11 Interpolation, basic integration schemes

Consider you are to compute $I = \int_a^b \mathrm{d}x\, f(x)$. Here we think of generic (*i.e.*, not specified) function $f(x)$. We would like to 1) compute I accurately enough, and 2) spend less of an effort (which we will measure in at how many points the function $f(\cdot)$ is computed).[36] In a general recipe, where $f(x)$ is not specified, [due to linearity of integration] an algorithm of computing I

Example 11: Let us compute $\int_0^{\pi/2} \mathrm{d}x\, \sin(x)$. We will do it in several ways:

(a) Analytical: $\int_0^{\pi/2} \mathrm{d}x\, \sin(x) = -\cos(x)\big|_0^{\pi/2} = \cos(0) - \cos(\pi/2) = 1$. This is the exact answer.

(b) Taylor series: $\displaystyle\int_0^{\pi/2} \mathrm{d}x\, \sin(x) = \int_0^{\pi/2} \mathrm{d}x \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = \sum_{n=0}^{\infty} \frac{(-1)^n (\pi/2)^{2n+2}}{(2n+2)!}$. Let us calculate this series numerically:

[36] It is not necessary that the integral is estimated through values of $f(\cdot)$ at some points. Possible situations could be using an analytical formula for I, or presenting f as a linear combination of functions

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat int_sin_b.py
from math import pi
taylor, sum, n, pi2f = 1., 0., 0, -1.
while taylor != sum:
    pi2f = -pi2f * (pi / 2.)**2 / ((2. * n + 1.) * 2. * (n + 1.))
    taylor, sum, n = sum, sum + pi2f, n + 1
print('series = {0:.15e}, {1:d} terms are summed'.format(taylor, n))
[...]/teaching/2019-4/math_575a/notes/Python$ python3 int_sin_b.py
series = 9.999999999999999e-01, 11 terms are summed
[...]/teaching/2019-4/math_575a/notes/Python$
```

(c) Midpoint rule: for large N we have $\int_0^{\pi/2} \mathrm{d}x\ \sin(x) \approx \underbrace{\frac{\pi/2}{N}}_{\Delta x} \sum_{i=0}^{N-1} \sin\Big(\underbrace{\frac{\pi(2i+1)}{4N}}_{x_i}\Big)$:

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat int_sin_c.py
from math import pi, sin
for N in [10, 100, 1000]:
    S = sum(map(sin, [(pi / 2.) * (i + 0.5) / N for i in range(0, N)]))
    print('N = {0:4d}, midpoint = {1:.15e}'.format(N, (pi / 2.) * S / N))
[...]/teaching/2019-4/math_575a/notes/Python$ python3 int_sin_c.py
N =   10, midpoint = 1.001028824142709e+00
N =  100, midpoint = 1.000010280911905e+00
N = 1000, midpoint = 1.000000102808387e+00
[...]/teaching/2019-4/math_575a/notes/Python$
```

(d) As in (c), but Simpson's rule is used: here N is necessarily even, $\Delta x := (b-a)/N$, $x_j := a + j\Delta x$ (so $x_0 = a$ and $x_N = b$), and $\int_a^b \mathrm{d}x\ f(x) \approx \frac{\Delta x}{3} \sum_{j=0}^{N/2-1} \Big(f(x_j) + 4f(x_{j+1}) + f(x_{j+2})\Big) = \Delta x\big(f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 2f(x_{N-2}) + 4f(x_{N-1}) + f(b)\big)/3$:

```
[...]/teaching/2019-4/math_575a/notes/Python$ cat int_sin_d.py
from math import pi, sin
for N in [10, 100, 1000]:
    S, w = sin(0.) + sin(pi / 2.), 4.
    for i in range(1, N):
        S, w = S + w * sin((pi / 2.) * i / N), 6. - w
    print('N = {0:4d}, Simpson = {1:.15e}'.format(N, pi * S / (6. * N)))
[...]/teaching/2019-4/math_575a/notes/Python$ python3 int_sin_d.py
N =   10, Simpson = 1.000003392220900e+00
N =  100, Simpson = 1.000000000338236e+00
N = 1000, Simpson = 1.000000000000033e+00
[...]/teaching/2019-4/math_575a/notes/Python$
```

[37]

Problems and exercises

[37] Chebfun — computational software using Chebyshev nodes.

1. Consider the integral $1 = \frac{1}{\mathrm{e}-1}\int\limits_0^1 \mathrm{d}x\ \exp(x)$. Compute it by left-, right-sum, trapezoidal, mid-point, and Simpson's rules. Log-log plot errors vs. Δx, find the order of accuracy of these numerical integration schemes.

2. (a) Compute the integral $\int\limits_0^1 \mathrm{d}x\ \left(1-x^2\right)^2$ using trapezoidal rule. (b) Compute the integral $\int\limits_0^1 \frac{\mathrm{d}x}{1+x^2}$ using Simpson's rule. In both (a) and (b) log-log plot errors vs. Δx, speculate about the order of accuracy.

3. Compute $\int\limits_0^1 \frac{\mathrm{d}x}{\sqrt{x+x^3}}$ up to 10 significant digits.

12 Euler method, stability

The solution of the system of ODEs $\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \boldsymbol{f}\big(t,\boldsymbol{x}(t)\big)$ can be written as[38]

$$\boldsymbol{x}(t_{\text{end}}) = \boldsymbol{x}(t_{\text{start}}) + \int\limits_{t_{\text{start}}}^{t_{\text{end}}} \mathrm{d}t\ \underbrace{\boldsymbol{f}\big(t,\boldsymbol{x}(t)\big)}_{\mathrm{d}\boldsymbol{x}/\mathrm{d}t}$$

We assume $\boldsymbol{x}(t_{\text{start}})$ to be known, and our task is to find $\boldsymbol{x}(t_{\text{end}})$ or the whole trajectory $\boldsymbol{x}(t)$. We divide the interval of integration $[t_{\text{start}},t_{\text{end}}]$ into N subintervals

$$t_{\text{start}} = t_0 < t_1 < t_2 < \ldots < t_{N-1} < t_N = t_{\text{end}}, \qquad n^{\text{th}} \text{ step size } h_n = t_{n-1} - t_n$$

and approximate the integral of the r.h.s. $\boldsymbol{f}$ over $[t_{n-1},t_n]$ using some integration scheme. We will denote $\boldsymbol{x}(t_n)$ as $\boldsymbol{x}_n$. Estimating the integral of $\boldsymbol{f}$ over $[t_0,t_1]$ would give us the difference between $\boldsymbol{x}_1$ and $\boldsymbol{x}_0$, and (as $\boldsymbol{x}_0$ is known) that will give us the value of $\boldsymbol{x}_1$.[39] Next, from the esimation of the integral of $\boldsymbol{f}$ over $[t_1,t_2]$ we will find $\boldsymbol{x}_2$. This way, one by one we find all the values $\boldsymbol{x}_n$, $1 \le n \le N$.

If we estimate the integral of $\boldsymbol{f}$ over subinterval $[t_{n-1},t_n]$ using left sums rule (with just one subdivision) we get the [forward] Euler method:

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\boldsymbol{f}\big(t,\boldsymbol{x}(t)\big)$$

From the known $\boldsymbol{x}(t_{\text{start}}) = \boldsymbol{x}_0$ we calculate $\boldsymbol{x}_1 = \boldsymbol{x}_0 + h_1\boldsymbol{f}(t_0,\boldsymbol{x}_0)$. Note that this is a straightforward calculation, we get $\boldsymbol{x}_1$ right away. Such numerical schemes for solving ODEs are called *explicit*.

[38] Such a rewrite is not very much useful by itself, as the function $\boldsymbol{x}(t)$ inside the integrand $\boldsymbol{f}(\cdot,\cdot)$ is unknown.
[39] Estimation of the integral may depend on the value of $\boldsymbol{x}_1$, then finding $\boldsymbol{x}_1$ is not straightforward.

13 Runge–Kutta methods

A general Runge–Kutta method is typically defined by writing down its *Butcher tableau*:

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}
\qquad
\begin{aligned}
\boldsymbol{k}_i &= h\boldsymbol{f}\Big(t + c_j h, \boldsymbol{x}(t) + \sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j\Big) \\
\boldsymbol{x}(t+h) &\coloneqq \boldsymbol{x}(t) + \sum_{j=1}^{s} b_j \boldsymbol{k}_j
\end{aligned}
$$

The number s is called a number of stages. The quantity $\boldsymbol{x}(t) + \sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j$ could be thought as a preliminary estimation of $\boldsymbol{x}(t + c_i h)$. The method is explicit if $a_{ij} = 0$ whenever $i \le j$. In this case the preliminary data $\boldsymbol{k}_i$, $i = 1, 2, ..., s$ can be calculated in staightforward computation.

For a method to be at least 1st order of accuracy, we need to have $b_1 + b_2 + ... + b_s = 1$.

It is physically reasonable to have $c_i = \sum_{j=1}^{s} a_{ij}$, as it implies $\boldsymbol{k}_i = h\boldsymbol{f}\big(t + c_i h, \boldsymbol{x}(t + c_i h)\big) + O(h^3)$. Then an explicit method necessarily would have $c_1 = 0$, and $\boldsymbol{k}_1 = h\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$, *i.e.*, the 1st stage in an explicit Runge–Kutta method is always a forward Euler step.

A celebrated classical Runge–Kutta method of the 4th order of accuracy (RK4) is given by

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
\qquad
\begin{aligned}
\boldsymbol{k}_1 &\coloneqq h\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big) \\
\boldsymbol{k}_2 &\coloneqq h\boldsymbol{f}\big(t + \tfrac{h}{2}, \boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{k}_1\big) \\
\boldsymbol{k}_3 &\coloneqq h\boldsymbol{f}\big(t + \tfrac{h}{2}, \boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{k}_2\big) \\
\boldsymbol{k}_4 &\coloneqq h\boldsymbol{f}\big(t + h, \boldsymbol{x}(t) + \boldsymbol{k}_3\big) \\
\boldsymbol{x}(t+h) &\coloneqq \boldsymbol{x}(t) + \tfrac{1}{6}\big(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\big)
\end{aligned}
$$

It is an explicit method with 4 stages. Let us demonstrate that it is indeed of the 4th order of accuracy. It will be convenient to use the following notation: $\boldsymbol{F}^{(m,n)}_{j_1 j_2 \cdots j_n} \coloneqq h^{m+1} \frac{\partial^m}{\partial t^m} \frac{\partial^n}{\partial X_{j_1} \partial X_{j_2} \ldots \partial X_{j_n}} \boldsymbol{f}\big(t, \boldsymbol{X}\big)\big|_{\boldsymbol{X}=\boldsymbol{x}(t)}$. We will write $\boldsymbol{F}$ instead of $\boldsymbol{F}^{(0,0)} = h\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$. For the dynamics $\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$, we would have (this

is general, and not about RK4)

$$
\begin{aligned}
\Delta \boldsymbol{x} &= \boldsymbol{x}(t+h)-\boldsymbol{x}(t) = \Big(h\tfrac{\mathrm{d}}{\mathrm{d}t}+\tfrac{h^2}{2}\tfrac{\mathrm{d}^2}{\mathrm{d}t^2}+\tfrac{h^3}{6}\tfrac{\mathrm{d}^3}{\mathrm{d}t^3}+\tfrac{h^4}{24}\tfrac{\mathrm{d}^4}{\mathrm{d}t^4}\Big)\boldsymbol{x}(t)+O(h^5)=\boldsymbol{F}+\\
&+\Big(\tfrac{h^2}{2}\tfrac{\mathrm{d}}{\mathrm{d}t}+\tfrac{h^3}{6}\tfrac{\mathrm{d}^2}{\mathrm{d}t^2}+\tfrac{h^4}{24}\tfrac{\mathrm{d}^3}{\mathrm{d}t^3}\Big)\boldsymbol{f}\big(t,\boldsymbol{x}(t)\big)+\ldots=\boldsymbol{F}+\tfrac{1}{2}\boldsymbol{F}^{(1,0)}+\tfrac{1}{2}\boldsymbol{F}^{(0,1)}_j F_j+\\
&+\Big(\tfrac{h^3}{6}\tfrac{\mathrm{d}}{\mathrm{d}t}+\tfrac{h^4}{24}\tfrac{\mathrm{d}^2}{\mathrm{d}t^2}\Big)\Big(\tfrac{\partial \boldsymbol{f}(t,\boldsymbol{X})}{\partial t}+\tfrac{\partial \boldsymbol{f}(t,\boldsymbol{X})}{\partial X_j}f_j\big(t,\boldsymbol{X}\big)\Big)\Big|_{\boldsymbol{X}=\boldsymbol{x}(t)}=\boldsymbol{F}+\tfrac{1}{2}\boldsymbol{F}^{(1,0)}+\tfrac{1}{2}\boldsymbol{F}^{(0,1)}_j F_j+\\
&+\underbrace{\tfrac{1}{6}\boldsymbol{F}^{(2,0)}+\tfrac{1}{6}\boldsymbol{F}^{(1,1)}_j F_j}_{\frac{h^3}{6}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial}{\partial t}\boldsymbol{f}(t,\boldsymbol{X})\big|_{\boldsymbol{X}=\boldsymbol{x}(t)}}+\underbrace{\tfrac{1}{6}\boldsymbol{F}^{(1,1)}_j F_j+\tfrac{1}{6}\boldsymbol{F}^{(0,2)}_{jk}F_jF_k+\tfrac{1}{6}\boldsymbol{F}^{(0,1)}_j F^{(1,0)}_j+\tfrac{1}{6}\boldsymbol{F}^{(0,1)}_j F^{(0,1)}_{j;k}F_k}_{\frac{h^3}{6}\frac{\mathrm{d}}{\mathrm{d}t}f_j(t,\boldsymbol{X})\frac{\partial}{\partial X_j}\boldsymbol{f}(t,\boldsymbol{X})\big|_{\boldsymbol{X}=\boldsymbol{x}(t)}}+\\
&+\tfrac{h^4}{24}\tfrac{\mathrm{d}}{\mathrm{d}t}\Big[\tfrac{\partial^2\boldsymbol{f}(t,\boldsymbol{X})}{\partial t^2}+2\tfrac{\partial^2\boldsymbol{f}(t,\boldsymbol{X})}{\partial t\partial X_j}f_j\big(t,\boldsymbol{X}\big)+\tfrac{\partial^2\boldsymbol{f}(t,\boldsymbol{X})}{\partial X_j\partial X_k}f_j\big(t,\boldsymbol{X}\big)f_k\big(t,\boldsymbol{X}\big)+\tfrac{\partial\boldsymbol{f}(t,\boldsymbol{X})}{\partial X_j}\tfrac{\partial f_j(t,\boldsymbol{X})}{\partial t}+\\
&+\tfrac{\partial\boldsymbol{f}(t,\boldsymbol{X})}{\partial X_j}\tfrac{\partial f_j(t,\boldsymbol{X})}{\partial X_k}f_k\big(t,\boldsymbol{X}\big)\Big]\Big|_{\boldsymbol{X}=\boldsymbol{x}(t)}=\boldsymbol{F}+\tfrac{1}{2}\boldsymbol{F}^{(1,0)}+\tfrac{1}{2}\boldsymbol{F}^{(0,1)}_j F_j+\tfrac{1}{6}\boldsymbol{F}^{(2,0)}+\tfrac{1}{3}\boldsymbol{F}^{(1,1)}_j F_j+\tfrac{1}{6}\boldsymbol{F}^{(0,2)}_{jk}F_jF_k+\\
&+\tfrac{1}{6}\boldsymbol{F}^{(0,1)}_j F^{(1,0)}_j+\tfrac{1}{6}\boldsymbol{F}^{(0,1)}_j F^{(0,1)}_{j;k}F_k+\tfrac{1}{24}\boldsymbol{F}^{(3,0)}{}_{\boxed{1}}+\tfrac{1}{8}\boldsymbol{F}^{(2,1)}_j F_{j\boxed{12}}+\tfrac{1}{8}\boldsymbol{F}^{(1,2)}_{jk}F_jF_{k\boxed{23}}+\\
&+\tfrac{1}{24}\boldsymbol{F}^{(0,3)}_{jkl}F_jF_kF_{l\boxed{3}}+\tfrac{1}{8}\boldsymbol{F}^{(1,1)}_j F^{(1,0)}_j{}_{\boxed{24}}+\tfrac{1}{8}\boldsymbol{F}^{(1,1)}_j F^{(0,1)}_{j;k}F_{k\boxed{25}}+\tfrac{1}{8}\boldsymbol{F}^{(0,2)}_{jk}F^{(1,0)}_j F_{k\boxed{34}}+\\
&+\tfrac{1}{8}\boldsymbol{F}^{(0,2)}_{jk}F^{(0,1)}_{j;l}F_kF_{l\boxed{35}}+\tfrac{1}{24}\boldsymbol{F}^{(0,1)}_j F^{(2,0)}_j{}_{\boxed{4}}+\tfrac{1}{12}\boldsymbol{F}^{(0,1)}_j F^{(1,1)}_{j;k}F_{k\boxed{45}}+\tfrac{1}{24}\boldsymbol{F}^{(0,1)}_j F^{(0,2)}_{j;kl}F_kF_{l\boxed{5}}+\\
&+\tfrac{1}{24}\boldsymbol{F}^{(0,1)}_j F^{(0,1)}_{j;k}F^{(1,0)}_k{}_{\boxed{5}}+\tfrac{1}{24}\boldsymbol{F}^{(0,1)}_j F^{(0,1)}_{j;k}F^{(0,1)}_{k;l}F_{l\boxed{5}}
\end{aligned}
$$

The digits in boxes indicate which of the 5 terms in big square brackets do contribute to the adjacent part of the expression.

For the classical Runge–Kutta method we have $\boldsymbol{k}_1 = \boldsymbol{F}$, and

$$\boldsymbol{k}_2 = h\boldsymbol{f}\big(t+\tfrac{h}{2}, \boldsymbol{x}(t)+\tfrac{1}{2}\boldsymbol{F}\big) = \boldsymbol{F} + \underbrace{\tfrac{1}{2}\boldsymbol{F}^{(1,0)} + \tfrac{1}{2}\boldsymbol{F}_j^{(0,1)}F_j}_{O(h^2)\text{ terms}} + \underbrace{\tfrac{1}{8}\boldsymbol{F}^{(2,0)} + \tfrac{1}{4}\boldsymbol{F}_j^{(1,1)}F_j + \tfrac{1}{8}\boldsymbol{F}_{jk}^{(0,2)}F_jF_k}_{O(h^3)\text{ terms}} +$$
$$+ \underbrace{\tfrac{1}{48}\boldsymbol{F}^{(3,0)} + \tfrac{1}{16}\boldsymbol{F}_j^{(2,1)}F_j + \tfrac{1}{16}\boldsymbol{F}_{jk}^{(1,2)}F_jF_k + \tfrac{1}{48}\boldsymbol{F}_{jkl}^{(0,3)}F_jF_kF_l}_{O(h^4)\text{ terms}} + O(h^5)$$

$$\boldsymbol{k}_3 = h\boldsymbol{f}\big(t+\tfrac{h}{2}, \boldsymbol{x}(t)+\tfrac{1}{2}\boldsymbol{k}_2\big) = \boldsymbol{F} + \tfrac{1}{2}\boldsymbol{F}^{(1,0)} + \tfrac{1}{2}\boldsymbol{F}_j^{(0,1)}k_{2j} + \tfrac{1}{8}\boldsymbol{F}^{(2,0)} + \tfrac{1}{4}\boldsymbol{F}_j^{(1,1)}k_{2j} + \tfrac{1}{8}\boldsymbol{F}_{jk}^{(0,2)}k_{2j}k_{2k} +$$
$$+ \tfrac{1}{48}\boldsymbol{F}^{(3,0)} + \tfrac{1}{16}\boldsymbol{F}_j^{(2,1)}k_{2j} + \tfrac{1}{16}\boldsymbol{F}_{jk}^{(1,2)}k_{2j}k_{2k} + \tfrac{1}{48}\boldsymbol{F}_{jkl}^{(0,3)}k_{2j}k_{2k}k_{2l} + O(h^5) = \boldsymbol{F} + \tfrac{1}{2}\boldsymbol{F}^{(1,0)} +$$
$$+ \tfrac{1}{2}\boldsymbol{F}_j^{(0,1)}\Big(F_j + \tfrac{1}{2}F_j^{(1,0)} + \tfrac{1}{2}F_{j;k}^{(0,1)}F_k + \tfrac{1}{8}F_j^{(2,0)} + \tfrac{1}{4}F_{j;k}^{(1,1)}F_k + \tfrac{1}{8}F_{j;kl}^{(0,2)}F_kF_l + O(h^4)\Big) + \tfrac{1}{8}\boldsymbol{F}^{(2,0)} +$$
$$+ \tfrac{1}{4}\boldsymbol{F}_j^{(1,1)}\Big(F_j + \tfrac{1}{2}F_j^{(1,0)} + \tfrac{1}{2}F_{j;k}^{(0,1)}F_k + O(h^3)\Big) + \tfrac{1}{8}\boldsymbol{F}_{jk}^{(0,2)}\Big(F_jF_k + \tfrac{1}{2}F_j^{(1,0)}F_k + \tfrac{1}{2}F_{j;l}^{(0,1)}F_lF_k +$$
$$+ \tfrac{1}{2}F_jF_k^{(1,0)} + \tfrac{1}{2}F_jF_{k;l}^{(0,1)}F_l + \ldots\Big) + \tfrac{1}{48}\boldsymbol{F}^{(3,0)} + \tfrac{1}{16}\boldsymbol{F}_j^{(2,1)}F_{2j} + \tfrac{1}{16}\boldsymbol{F}_{jk}^{(1,2)}F_jF_k + \tfrac{1}{48}\boldsymbol{F}_{jkl}^{(0,3)}F_jF_kF_l + \ldots$$

$$\boldsymbol{k}_4 = h\boldsymbol{f}\big(t+h, \boldsymbol{x}(t)+\boldsymbol{k}_3\big) = \boldsymbol{F} + \boldsymbol{F}^{(1,0)} + \boldsymbol{F}_j^{(0,1)}k_{3j} + \tfrac{1}{2}\boldsymbol{F}^{(2,0)} + \boldsymbol{F}_j^{(1,1)}k_{3j} + \tfrac{1}{2}\boldsymbol{F}_{jk}^{(0,2)}k_{3j}k_{3k} +$$
$$+ \tfrac{1}{6}\boldsymbol{F}^{(3,0)} + \tfrac{1}{2}\boldsymbol{F}_j^{(2,1)}k_{3j} + \tfrac{1}{2}\boldsymbol{F}_{jk}^{(1,2)}k_{3j}k_{3k} + \tfrac{1}{6}\boldsymbol{F}_{jkl}^{(0,3)}k_{3j}k_{3k}k_{3l} + \ldots = \boldsymbol{F} + \boldsymbol{F}^{(1,0)} +$$
$$+ \boldsymbol{F}_j^{(0,1)}\Big(F_j + \tfrac{1}{2}F_j^{(1,0)} + \tfrac{1}{2}F_{j;k}^{(0,1)}\big(F_k + \tfrac{1}{2}F_k^{(1,0)} + \tfrac{1}{2}F_{k;l}^{(0,1)}F_l\big) + \tfrac{1}{8}F_j^{(2,0)} + \tfrac{1}{4}F_{j;k}^{(1,1)}F_k + \tfrac{1}{8}F_{j;kl}^{(0,2)}F_kF_l\Big) +$$
$$+ \tfrac{1}{2}\boldsymbol{F}^{(2,0)} + \boldsymbol{F}_j^{(1,1)}\Big(F_j + \tfrac{1}{2}F_j^{(1,0)} + \tfrac{1}{2}F_{j;k}^{(0,1)}F_k\Big) + \tfrac{1}{2}\boldsymbol{F}_{jk}^{(0,2)}\Big(F_jF_k + \tfrac{1}{2}F_j^{(1,0)}F_k + \tfrac{1}{2}F_{j;l}^{(0,1)}F_lF_k +$$
$$+ \tfrac{1}{2}F_jF_k^{(1,0)} + \tfrac{1}{2}F_jF_{k;l}^{(0,1)}F_l\Big) + \tfrac{1}{6}\boldsymbol{F}^{(3,0)} + \tfrac{1}{2}\boldsymbol{F}_j^{(2,1)}F_j + \tfrac{1}{2}\boldsymbol{F}_{jk}^{(1,2)}F_jF_k + \tfrac{1}{6}\boldsymbol{F}_{jkl}^{(0,3)}F_jF_kF_l + \ldots$$

	$\boldsymbol{F}$	$\boldsymbol{F}^{(1,0)}$	$\boldsymbol{F}_j^{(0,1)}F_j$	$\boldsymbol{F}^{(2,0)}$	$\boldsymbol{F}_j^{(1,1)}F_j$	$\boldsymbol{F}_{jk}^{(0,2)}F_jF_k$	$\boldsymbol{F}_j^{(0,1)}F_j^{(1,0)}$	$\boldsymbol{F}_j^{(0,1)}F_{j;k}^{(0,1)}F_k$	$\boldsymbol{F}^{(3,0)}$	$\boldsymbol{F}_j^{(2,1)}F_j$	$\boldsymbol{F}_{jk}^{(1,2)}F_jF_k$	$\boldsymbol{F}_{jkl}^{(0,3)}F_jF_kF_l$	$\boldsymbol{F}_j^{(1,1)}F_j^{(1,0)}$	$\boldsymbol{F}_j^{(1,1)}F_{j;k}^{(0,1)}F_k$	$\boldsymbol{F}_{jk}^{(0,2)}F_j^{(1,0)}F_k$	$\boldsymbol{F}_{jk}^{(0,2)}F_{j;l}^{(0,1)}F_kF_l$	$\boldsymbol{F}_j^{(0,1)}F_j^{(2,0)}$	$\boldsymbol{F}_j^{(0,1)}F_{j;k}^{(1,1)}F_k$	$\boldsymbol{F}_j^{(0,1)}F_{j;kl}^{(0,2)}F_kF_l$	$\boldsymbol{F}_j^{(0,1)}F_{j;k}^{(0,1)}F_k^{(1,0)}$	$\boldsymbol{F}_j^{(0,1)}F_{j;k}^{(0,1)}F_{k;l}^{(0,1)}F_l$
$\Delta\boldsymbol{x}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$
$\boldsymbol{k}_1$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\boldsymbol{k}_2$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$	0	0	$\frac{1}{48}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{48}$	0	0	0	0	0	0	0	0	0
$\boldsymbol{k}_3$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{48}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{48}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$	0	0
$\boldsymbol{k}_4$	1	1	1	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{4}$

We have $\Delta\boldsymbol{x} = \boldsymbol{x}(t+h) - \boldsymbol{x}(t) = (\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)/6 + O(h^5)$, and the classical Runge–Kutta method RK4 is indeed of the 4th order of accuracy.

Problems and exercises

1. Consider the system of equations $\mathrm{d}x/\mathrm{d}t = v$, $\mathrm{d}v/\mathrm{d}t = -\sin(x) - 0.02v$. Solve the system with the initial condition $x(0) = 0$, $v(0) = 2.125$ numerically, using forward Euler, explicit midpoint (RK2), and classical Runge–Kutta (RK4) methods. Find out how the error in

$$\begin{bmatrix} x(20) & v(20) \end{bmatrix} = \begin{bmatrix} 6.8426504104428864014... & 1.7912033841288853138... \end{bmatrix}$$

scales with h.

2. Consider the system of equations $\mathrm{d}x/\mathrm{d}t = p$, $\mathrm{d}p/\mathrm{d}t = x - x^2$.[40] Solve the system with the initial condition $x(0) = 0.01$, $p(0) = 0.009$ numerically, using forward Euler, explicit midpoint (RK2), and classical Runge–Kutta (RK4) methods. Log-log plot the error in

$$\begin{bmatrix} x(20) & p(20) \end{bmatrix} = \begin{bmatrix} 0.48859294559329852479... & 0.40118050259290873684... \end{bmatrix}$$

vs. the step size h for all the three methods.

3. Consider the system of N ordinary differential equations $(1 < n < N)$

$$\frac{\mathrm{d}u_1(t)}{\mathrm{d}t} = \frac{-u_1(t) + u_2(t)}{2(\Delta x)^2}, \quad \frac{\mathrm{d}u_n(t)}{dt} = \frac{u_{n-1}(t) - 2u_n(t) + u_{n+1}(t)}{2(\Delta x)^2}, \quad \frac{\mathrm{d}u_N(t)}{\mathrm{d}t} = \frac{u_{N-1}(t) - u_N(t)}{2(\Delta x)^2}$$

where $\Delta x = 1/N$, and $N = 100$. (This is a discretization of $\partial u/\partial t = \frac{1}{2}\partial^2 u/\partial x^2$ diffusion equation on $0 < x < 1$ segment, with zero flux boundary at the walls located at $x = 0$ and $x = 1$. The quantity $u_n(t)$ could be thought as the concentration of diffusing particles inside $\big((n-1)/N, n/N\big)$ interval.) Consider the initial condition $u_n(0) \coloneqq C_N x_n^2(1 - x_n)$, where $x_n \coloneqq (n - \frac{1}{2})/N$ and $C_N \coloneqq N / \sum_{n=1}^{N} x_n^2(1 - x_n)$. Solve the system to find $u_n(1)$ by (a) forward Euler using the time step $\tau = 1/9992$; (b) forward Euler with $\tau = 1/10000$; (c) backward Euler, $\tau = 1/100$. Plot $u_n(t = 1)$ as a function of n.

14 Adaptive step size

To reduce computational cost / improve accuracy of computation we would like to increase / decrease the step size. The former reduces the number of steps, while the latter shrinks the local (and then the global) error in each step. We would like to go with large steps through dull, uninteresting parts of our dynamics, while it is desirable to make small steps in tricky parts of the dynamics in order not to lose accuracy. To do so, we constantly need to be aware of whether we are satisfied with the current, intantaneous quality of solution.

An easy way to estimate the accuracy of numerical solution is to compare it with another solution, of comparable or even better quality. With whatever numerical scheme you are using, one possibility is to compare $\boldsymbol{x}(t+h)$, obtained from $\boldsymbol{x}(t)$ by one full step h, with $\boldsymbol{x}(t+h)$ obtained from $\boldsymbol{x}(t+\frac{1}{2}h)$, which itself is an $(h/2)$-update of $\boldsymbol{x}(t)$. Then we compare the two versions of $\boldsymbol{x}(t+h)$, and if, let us say, it is greater than some tolerance level, we do not accept such an update of $\boldsymbol{x}(t)$. The next thing is to tune the size step in such a way that the predicted difference between the two versions of $\boldsymbol{x}\big(t + \text{new } h\big)$ would be close to the tolerance level:

$$\text{new } h \coloneqq \left(\text{frac} \cdot \frac{\text{tolerance level}}{\text{difference of the two } \boldsymbol{x}(t+h) \text{ versions}}\right)^{1/(p+1)} \cdot h$$

[40] This is a Hamiltonian system, with $\mathcal{H}(x,p) = \frac{1}{2}p^2 - \frac{1}{2}x^2 + \frac{1}{3}x^3$.

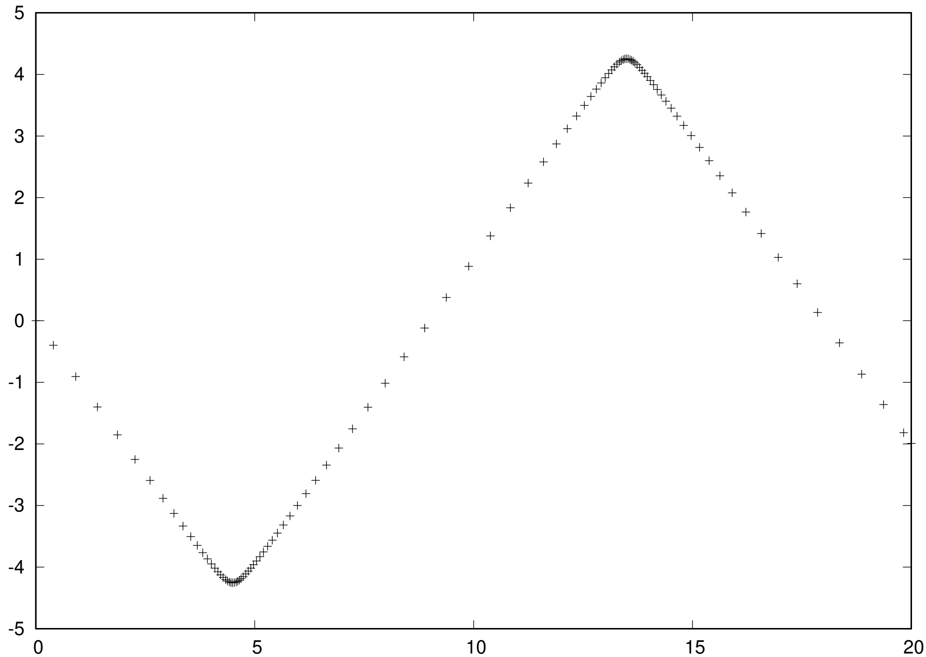
Here p is the order of accuracy of our scheme, and frac is the so called *safety fraction*.

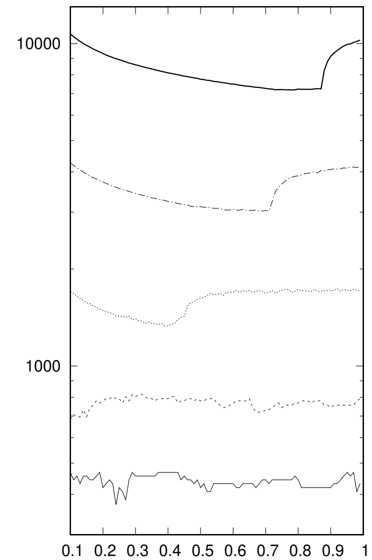
Example 14: Consider the following system of ODEs $\mathrm{d}x/\mathrm{d}t = p$, $\mathrm{d}p/\mathrm{d}t = -x/\sqrt{1+x^2}$ with initial condition $x(0) = 10$, $p(0) = 0$.[41] We would like to construct the trajectory $x(t)$, $p(t)$ for $0 \leq t \leq t_{\mathrm{end}} = 20$. Let us employ RK4 method and choose the step size adaptively, as described above:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define m 2
int counter; /* number of calculations of the r.h.s. f */
void RHS(double t, double *x, double *f)
  { f[0] = x[1]; f[1] = -x[0] / sqrt(1. + pow(x[0], 2.)); counter++; }

/* classical Runge--Kutta method (RK4) */
#define s 4
double X[s][m], K[s][m], a[4][4] = { {   0.,    0.,   0.,   0.},
                                     {1./2.,    0.,   0.,   0.},
                                     {   0., 1./2.,   0.,   0.},
                                     {   0.,    0.,   1.,   0.} },
  b[s] = {1./6., 1./3., 1./3., 1./6.}, c[s] = {0., 1./2., 1./2., 1.};

void explicit_Runge_Kutta(double h, double t, double *x0, double *x1) {
  int i, j, l;
  for (i = 0; i < s; RHS(t + c[i] * h, X[i], K[i]), i++) for (l = 0; l < m; l++)
    for (X[i][l] = x0[l], j = 0; j < i; j++) X[i][l] += h * a[i][j] * K[j][l];
  for (l = 0; l < m; l++)
    for (x1[l] = x0[l], i = 0; i < s; i++) x1[l] += h * b[i] * K[i][l]; }

int main(int argc, char **argv)
 {
  double t, dt, t_end = 20., x[m], xh[m], xhh1[m], xhh2[m];
  double local_error, tolerance = 15. * atof(argv[2]), frac = atof(argv[1]);

  x[0] = 10.; x[1] = 0.; t = 0.; dt = atof(argv[3]);
  printf("%22.16e % 22.16e % 22.16e 0\n", t, x[0], x[1]);
  for (counter = 0; t < t_end;)
   {
    if (t + dt > t_end) dt = t_end - t;
/* Runge--Kutta, full step */
    explicit_Runge_Kutta(dt, t, x, xh);
/* Runge--Kutta, two half steps */
    explicit_Runge_Kutta(0.5 * dt, t, x, xhh1);
    explicit_Runge_Kutta(0.5 * dt, t, xhh1, xhh2);
/* estimating new time step from the mismatch between the two updates */
    local_error = sqrt(pow(xh[0] - xhh2[0], 2.) + pow(xh[1] - xhh2[1], 2.));
/* checking whether the time step is accepted or rejected */
    if (local_error < tolerance) { t += dt; x[0] = xhh2[0]; x[1] = xhh2[1];
      printf("%22.16e % 22.16e % 22.16e %d\n", t, x[0], x[1], counter); }
    dt = dt * pow(frac * tolerance / local_error, 0.2);
   } return 0; }
```

[41] This system is Hamiltonian, with $\mathcal{H}(x,p) = \frac{1}{2}p^2 + \sqrt{1+x^2}$.

Here is the graph of $p(t)$, the tolerance level is 10^{-8}, the points on the graph are actual consecutive points of the obtained numerical solution (one can see that near $x = 0$, where the "velocity" $p(t)$is the maximal, the time step is a lot smaller):





On the right is the graph that shows the dependence of the total number of the r.h.s. $\boldsymbol{f}(t,\boldsymbol{x})$ calculations to reach the final time $t_{\text{end}} = 20$ as a function of frac. The curves correspond (from bottom to top) to tolerance levels 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, and 10^{-12}. Whenever frac is too small, we propagate forward with smaller steps, so we need more steps. On the other hand, if frac is too large, we expect rejections of the step to happen more ofter, so we'll still need many r.h.s. calculations.

Problems and exercises

1. Consider the system of ODEs $\mathrm{d}x/\mathrm{d}t = p$, $\mathrm{d}p/\mathrm{d}t = -x/\sqrt{1+x^2}$ with initial condition $x(0) = 10$, $p(0) = 0$ (Example 14). Find $x(20)$, $p(20)$ by Dormand–Prince method with adaptive step size. Plot how the number of the r.h.s. evaluations changes with the tolerance level.

15 Boundary value problems

15.1 Quasi-linearization

Discretizing the ODEs somehow, write down the BVP as a (linear or non-linear) system of equations, which then solve, *e.g.*, by Newton–Raphson method (in this case the method is called quasi-linearization).

15.2 Shooting method

Consider the BVP $\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$ with boundary conditions $\boldsymbol{g}_0\big(\boldsymbol{x}(t_0)\big) = \boldsymbol{0}$ and $\boldsymbol{g}_1\big(\boldsymbol{x}(t_1)\big) = \boldsymbol{0}$. Construct the system of equations for $\boldsymbol{x}(t_0)$ being ($\boldsymbol{g}_0 = \boldsymbol{0}$ and $\boldsymbol{g}_1 = \boldsymbol{0}$), in which the argument of $\boldsymbol{g}_1$, the vector $\boldsymbol{x}(t_1)$ is treated as a vector function of $\boldsymbol{x}(t_0)$ that is computed by an ODE solver. The resulting system of equations is solved by methhods of Part III. The parts of $\boldsymbol{x}(t_0)$ which are not immediately determined from $\boldsymbol{g}_0\big(\boldsymbol{x}(t_0)\big) = \boldsymbol{0}$ are called *shooting parameters*.

15.3 Petviashvili factor

15.4 Galerkin method

Problems and exercises

1. Consider the BVP $u'' + u^2 = 0$ with $u(\pm 1) = 0$ boundary conditions. There are two solutions: $u(x) \equiv 0$ and a "non-trivial" one. Find the latter solution by (a) quasi-linearization method $\boldsymbol{u}_{n+1} \coloneqq \boldsymbol{u}_n - \big((\nabla \boldsymbol{f})(\boldsymbol{u}_n)\big)^{-1} \boldsymbol{f}(\boldsymbol{u}_n)$:

$$\boldsymbol{u} = \begin{bmatrix} u(-1) \\ u(-1+h) \\ u(-1+2h) \\ \vdots \\ u(x) \\ \vdots \\ u(1-h) \\ u(1) \end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{u}) = \begin{bmatrix} u(-1) \\ u(-1) - 2u(-1+h) + u(-1+2h) + h^2 u^2(-1+h) \\ u(-1+h) - 2u(-1+2h) + u(-1+3h) + h^2 u^2(-1+2h) \\ \vdots \\ u(x-h) - 2u(x) + u(x+h) + h^2 u^2(x) \\ \vdots \\ u(1-2h) - 2u(1-h) + u(1) + h^2 u^2(1-h) \\ u(1) \end{bmatrix}$$

(b) simple shooting method (there is only one shooting parameter here, *e.g.*, $u'(-1)$); and (c) functional iteration with Petviashvili factor:

$$v_n(x) \coloneqq A_n \cdot (x+1) - \int_{-1}^{x} \mathrm{d}\xi_1 \int_{-1}^{\xi_1} \mathrm{d}\xi_2 \, u_n^2(\xi_2), \quad \text{by construction } v_n'' = -u_n^2 \text{ and } v_n(-1) = 0$$

$$A_n \text{ is chosen to enforce } v_n(1) = 0$$

$$u_{n+1}(x) \coloneqq v_n(x) \cdot \left(\frac{\int_{-1}^{1} \mathrm{d}\xi \, \big(-v_n''(\xi)\big)}{\int_{-1}^{1} \mathrm{d}\xi \, v_n^2(\xi)} \right)^{\alpha}, \qquad \alpha \coloneqq 1 \text{ for faster convergence}$$

References

[AsPe98] U. M. Ascher, L. R. Petzold, *Computer methods for ODEs and DAEs* (SIAM, 1998).

[Dem97] J. W. Demmel, *Applied numerical linear algebra* (SIAM, Philadelphia, 1997).

[GoVa96] G. H. Golub, C. F. Van Loan, *Matrix computations*, 3rd ed. (Johns Hopkins U. Press, Baltimore, 1996).

[Hig02] N. J. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed. (SIAM, Philadelphia, 2002).

[IEEE85] *IEEE standard for binary floating-point arithmetic, ANSI/IEEE Standard 754-1985* (IEEE, New York, 1985).

[Knu98] D. E. Knuth, *The art of computer programming. Vol. 2. Seminumerical algorithms*, 3rd ed. (Addison Wesley Longman, 1998).

[TrBa97] L. N. Trefethen, D. Bau III, *Numerical linear algebra* (SIAM, 1997).

Index

adjoint, 11
algorithm
  backward stable, 4
  semistable, 5
  stable, 5

back substitution, 18
backward error, 4
backward stable algorithm, 4
bisection method, 29
Butcher tableau, 35

condition number, 8

diagonal matrix, 11
dot product, 11

equivalent norms, 14

Frobenius norm, 12

Hermitian conjugate, 11
Hermitian matrix, 11
Hessenberg decomposition, 29
Hilbert–Schmidt norm, 12
Householder reflection, 22

identity matrix, 11
implementation
  ε-valid, 4
  virtual, 2
induced norm, 12

machine epsilon, 2
matrix, 11
  diagonal, 11
  Hermitian, 11
  identity matrix, 11
  orthogonal, 11
  unitary, 11
  zero matrix, 11

norm
  L^0-“norm”, 11
  L^2-norm, 11
  L^∞-norm, 11
  L^p-norm, 11
  Frobenius, 12
  Hilbert–Schmidt, 12
  induced, 12
  operator, 12
  weighted, 11

operator norm, 12
orthogonal matrix, 11
orthogonal vectors, 11

QR factorization, 20

Rayleigh quotient, 28

safety fraction, 38
scalar product, 11
Schur decomposition, 28
semistable algorithm, 5
shooting parameter, 41
singular value, 12
singular value decomposition, 12
singular vector, 12
stable algorithm, 5
SVD, 12

unit roundoff, 2
unit vector, 11
unitary matrix, 11

ε-valid implementation, 4
vector, 11
  unit, 11
virtual implementation, 2

weighted norm, 11

zero matrix, 11