Základy počítačové fyziky

Příručka studentů kombinovaného studia oboru PTA

Stanislav Hledík

Ústav fyziky, Filozoficko-přírodovědecká fakulta, Slezská univerzita v Opavě

Opava 2008

Abstrakt

Distanční kurs Základy počítačové fyziky seznamuje se základními numerickými algoritmy, s tvorbou a používáním programů ve vědě a technice obecně a s numerickými simulacemi fyzikálních jevů. Jednotlivé partie jsou ilustrovány jednoduchými programy.

Program Aplikovaná fyzika

Obor B 1702 Počítačová technika a její aplikace

Klíčová slova Numerické programování; simulace; počítačová fyzika

Anotace Tato elektronická skripta pro studenty kombinovaného studia oboru „Počítačová technika a její aplikace“ je uzpůsobena ke konzumaci na počítačovém monitoru. Jejich cílem je provést čtenáře základy numerického programování a počítačové fyziky. Kvůli všeobecné dosažitelnosti, jednoduchosti a nezávislosti na komerčních produktech je jako základní programovací prostředek použit programovací jazyk C. Jednotlivé partie jsou ilustrovány úlohami v textu a jednoduchými programy.

Autor RNDr. Stanislav Hledík, Ph.D.

Recenzenti doc. Ing. Petr Čermák, CSc. (ÚI FPF SU v Opavě)
doc. RNDr. Jan Obdržálek, CSc. (ITF MFF UK v Praze)

ISBN X-XXXX-XXXX

Obsah

Úvod

Toto je učební text k semestrálnímu distančnímu kursu „Základy počítačové fyziky“. Klade důraz na osvojení matematických základů používaných algoritmů, avšak bez zatěžujících detailů. Z toho důvodu je text doprovázen množstvím odkazů na ilustrační příklady shromážděné v Dodatku 9. Neváže se na konkrétní programovací jazyk; lze používat libovolný nízkoúrovňový jazyk běžný v oblasti numerického programování (obvykle Fortran 77, Fortran 90/95, C nebo C++). Kvůli všeobecné dosažitelnosti, jednoduchosti a nezávislosti na komerčních produktech je jako základní programovací prostředek použit programovací jazyk C.

Výklad je převážně založen na druhém vydání knihy [Press et al., 1997a] (existují mutace pro Fortran 77 a 90 [Press et al., 1997b, Press et al., 1997c]), jejíž online verzi lze stáhnout z http://www.nr.com/oldverswitcher.html a po nainstalování dekryptovacího pluginu (dostupného tamtéž) pro Adobe Reader prohlížet nebo tisknout.

V Opavě 23. listopadu 2008

Stanislav Hledík
stanislav.hledik@fpf.slu.cz

0. Jak příručku používat

Rychlý náhled kapitoly: Tato kapitola slouží jako úvodní a má za úkol seznámit čtenáře se způsobem používání jak příručky samotné, tak programů nezbytných pro praktické programování simulací fyzikálních jevů a jejich vizualizaci.

Cíle kapitoly:

- Naučit se pracovat s hypertextovým elektronickým skriptem.
- Naučit se instalaci a základní práci s vývojovým prostředím pro jazyk C a C++ Bloodshed Dev-C++.
- Naučit se instalaci a základní práci s vizualizačním programem gnuplot.

Klíčová slova kapitoly: Portable Document Format, PDF; Adobe Reader; Ghostscript, GSview, GV, KGhostview; Xpdf, KPDF; hypertext; IDE, gcc, Bloodshed Dev-C++; vizualizace, gnuplot

0.1. Prohlížeč

Pro čtení příručky je zapotřebí prohlížeče dokumentů ve formátu PDF (Portable Document Format). Protože pravděpodobně nevlastníte komerční programový balík Adobe Acrobat, nejlepší volbou je stáhnout si zdarma Adobe Reader, který existuje ve verzi pro Windows, MacOS a pro různé „odrůdy“ Unixu včetně Linuxu. Pokud si tento text prohlížíte pomocí tohoto programu, odkazuje text zvýrazněný fialovou barvou přímo na příslušné stránky – kliknutí na něj otevře okno prohlížeče na příslušné stránce.[1]

Další možností je použít prohlížeče založené na jádru Ghostscript s frontendem GSview pro Windows a GV nebo KGhostview pro Unix a Linux. Tyto programy však nepodporují hypertextové prvky, o nichž bude řeč dále. Uživatelé operačního systému Linux mohou použít prohlížeče Xpdf, případně jeho klonu KPDF, jenž hypertextové prvky částečně podporují. Konečně uživatelé MacOS X od Apple mohou využít prohlížeče Preview integrovaného do systému, jenž rovněž částečně podporuje hypertextové prvky.

---

1. Nestane-li se tak, musíte si v menu Edit→Preferences→Internet Adobe Readeru nastavit váš oblíbený webový prohlížeč.

0.2. Prvky učebního textu

Pro snazší orientaci v textu zobrazeném na monitoru počítače obsahuje tato učebnice hypertextové prvky. V levém horním rohu je umístěna aktuální stránková číslice a celkový počet stran, vlevo dole červený „ukazatel postupu“ názorně graficky zobrazující polohu aktuálního textu v dokumentu (za koncovou stranu se bere strana 263). Stránková číslice a prostřední „čárka“ v ukazateli postupu jsou aktivní a po kliknutí otevírají dialog „GoToPage“. Nalezení některých dalších aktivních zón ukazatele postupu přenecháváme čtenáři.

Každá strana je vpravo dole vybavena vlastním navigačním menu obsahujícím všechny hlavní prvky pro pohyb v dokumentu – přechod na první a poslední stranu, na předchozí a další stranu, skok na naposledy prohlíženou stranu a zpět, hledání podle čísla strany, celoobrazovkový režim, zavření dokumentu a ukončení prohlížeče; navíc zde nalezneme tlačítko pro přechod na obsah. Hlavní využití navigace je v celoobrazovkovém módu. Příslušné tlačítko navigace pak funguje jako přepínač do a z celoobrazovkového režimu. Připomeňme, že navigace je plně funkční pouze v prohlížeči Adobe Reader resp. v komerčním Adobe Acrobatu, částečně v prohlížečích Xpdf a KPDF.

V textu se kromě vlastního výkladu nacházejí následující prvky:

- Na začátku každé kapitoly je box označený symbolem  obsahující rychlý náhled kapitoly, dále box označený symbolem  deklarující cíle kapitoly a konečně box označený symbolem  s výčtem klíčových slov kapitoly.
- Boxy označené symbolem  obsahující popis volání odvozovaných rutin a přehled vstupů, výstupů a závislostí na jiných rutinách.
- Boxy označené symbolem  s otázkami k procvičení; odpovědi některých z nich jsou uvedeny v tomtéž boxu vzhůru nohama.
- Boxy označené symbolem  s pobídkou k vyzkoušení ilustračních programů (jenž jsou kvůli portabilitě až na výjimky psány v ANSI C). Většina příkladů je převzata z knihy [Press et al., 1997a], další byly vytvořeny speciálně pro tento kurs, některé jsou přejaty (s uvedením zdroje); několik pochází z výzkumných úkolů řešených autorem a spolupracovníky. Názvy rutin v boxu tvoří link do Dodatku 9, kde je uveden podrobný popis programu ilustrujícího funkci rutiny.
- Každá kapitola je uzavřena boxem označeným symbolem  obsahujícím shrnutí učiva kapitoly.

Výklad je vybaven následujícími druhy hypertextových odkazů:

Interní linky (na kapitoly, sekce, klíčová slova, stránky, zpětné reference v seznamu literatury apod.) jsou červené.

Linky na literaturu uvedenou v seznamu na konci textu mají následující podobu: [Press et al., 1997a].

Externí linky (typicky URL otevírající nakonfigurovaný webový prohlížeč) mají fialovou barvu.

Některé strany mají inkrementální zobrazování, kdy se text při použití povelu pro zobrazení další stránky (tlačítka „Další“ nebo klávesy PgDn) doplňuje na tutéž stranu, a teprve až je naplněna podle záměru autora, vyvolá „obrácení“ na další stranu s číslem o jedničku větším. Můžete si to rovnou vyzkoušet – přejděte na další stranu...

Některé strany mají inkrementální zobrazování, kdy se text při použití povelu pro zobrazení další stránky (tlačítka „Další“ nebo klávesy PgDn) doplňuje na tutéž stranu, a teprve až je naplněna podle záměru autora, vyvolá „obrácení“ na další stranu s číslem o jedničku větším. Můžete si to rovnou vyzkoušet – přejděte na další stranu... a další...

Některé strany mají inkrementální zobrazování, kdy se text při použití povelu pro zobrazení další stránky (tlačítka „Další“ nebo klávesy PgDn) doplňuje na tutéž stranu, a teprve až je naplněna podle záměru autora, vyvolá „obrácení“ na další stranu s číslem o jedničku větším. Můžete si to rovnou vyzkoušet – přejděte na další stranu... a další... a ještě další...

Některé strany mají inkrementální zobrazování, kdy se text při použití povelu pro zobrazení další stránky (tlačítka „Další“ nebo klávesy PgDn) doplňuje na tutéž stranu, a teprve až je naplněna podle záměru autora, vyvolá „obrácení“ na další stranu s číslem o jedničku větším. Můžete si to rovnou vyzkoušet – přejděte na další stranu... a další... a ještě další... a samozřejmě tlačítko „Předchozí“ nebo klávesa PgUp pak text ze stránky ubírá.

Tento způsob zobrazování je užit záměrně zejména v situacích, kdy přemíra informací z celé strany by u čtenáře mohla vyvolat saturaci vjemů, nebo při skrývání řešení otázek k procvičení. Poněkud méně triviální příklad lze nalézt na straně 64.

0.3. Vývojové prostředí pro jazyk C

Počítačová fyzika se nedá naučit bez praktického vyzkoušení. Proto byl tento učební text od začátku koncipován jako elektronický, aby si student mohl přímo při čtení na počítači prakticky zkoušet probíranou látku. Tím se dostáváme k prostředí, v němž budeme programovat.

Aby tento kurs nebyl vázán na komerční překladač nebo IDE (Integrated Development Environment, Integrované vývojové prostředí), doporučuji použít známého svobodného překladače jazyka GCC, jenž bývá nativní součástí operačního systému GNU/Linux, uživatelé Linuxu jej mohou použít. Uživatelům Windows doporučuji nainstalovat volně dostupné IDE Bloodshed Dev-C++ založené na Mingw portu GCC.

Základní práce s IDE Bloodshed Dev-C++ bude vysvětlena v průběhu konzultací na Ústavu fyziku FPF SU v Opavě.

0.4. Vizualizace výsledků

Je rozumné psát numerické programy pro simulaci fyzikálních procesů tak, že jejich výstupem jsou datové soubory v textovém formátu. Tím programy oprostíme od grafických knihoven. Výstupní datové soubory mohou být následně načteny programy, které umějí data graficky zpracovat podle našich instrukcí.

Typickým zástupcem takových programů je gnuplot. Jedná se podobně jako v případě vývojového prostředí Bloodshed Dev-C++ o svobodný software a funguje pod operačním systémem Windows a Linux.

Σ **Shrnutí kapitoly:** Naučili jste se základním dovednostem nutným pro vývoj numerických programů a vizualizaci získaných výsledků. Můžete se proto směle pustit do další kapitoly.

1. Počítačová reprezentace čísel a aritmetika

Rychlý náhled kapitoly: V této kapitole budou vysvětleny principy ukládání a manipulace s čísly v počítači, a to jak celých, tak reálných. Standardní počítačová reprezentace čísel a počítačová aritmetika obsahují důležité body, jichž bychom si při tvorbě počítačových modelů měli být vědomi. Ušetříme si tak dlouhé hodiny přemýšlení, proč něco funguje jinak, než se domníváme, že by mělo fungovat.

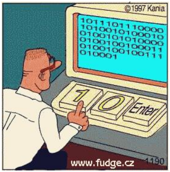
Cíle kapitoly:

- Naučit se pracovat s binární, oktalovou a hexadecimální reprezentací čísel.
- Porozumět reprezentaci celých čísel v počítači a vyhnout se případným problémům s přetečením nebo podtečením.
- Porozumět reprezentaci reálných čísel s pohyblivou desetinnou čárkou a její implementaci normou ANSI/IEEE 754-1985.
- Porozumět aritmetice v pohyblivou desetinné čárce a jejím úskalím.

Klíčová slova kapitoly: Celá čísla, znaménková, neznaménková; reálná čísla; bit; byte, bajt; binární, oktalová, hexadecimální reprezentace; pohyblivá desetinná čárka; FP čísla, normalizovaná, subnormální; norma ANSI/IEEE 754-1985; jednoduchá, dvojnásobná přesnost; strojové epsilon; zaokrouhlování; výjimky; stabilita

Motto: *10.0 × 0.1 se zřídkakdy rovná 1.0*

Celá čísla (*integers*) včetně znaků (*characters*) a reálná čísla (*real*) mají odlišný způsob počítačové reprezentace a provádění aritmetických operací. Počítač disponuje konečným prostorem pro uskladnění číslic; proto je nutné jak spočetnou množinu celých čísel, tak reálné kontinuum vhodně reprezentovat.

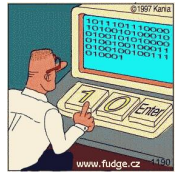


1.1. Binární čísla

Dekadická soustava – báze (*basis*) 10, deset číslic (*digits*) 0, . . . , 9:

$$(109.375)_{10} = 1 \times 10^2 + 0 \times 10^1 + 9 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$$

Motto: *10.0×0.1 se zřídkakdy rovná 1.0*



Celá čísla (*integers*) včetně znaků (*characters*) a reálná čísla (*real*) mají odlišný způsob počítačové reprezentace a provádění aritmetických operací. Počítač disponuje konečným prostorem pro uskladnění číslic; proto je nutné jak spočetnou množinu celých čísel, tak reálné kontinuum vhodně reprezentovat.

1.1. Binární čísla

Dekadická soustava – báze (*basis*) 10, deset číslic (*digits*) $0, \ldots, 9$:

$$(109.375)_{10} = 1 \times 10^2 + 0 \times 10^1 + 9 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$$

Technologie hardwaru předurčuje vyjádření čísel ve dvojkové soustavě (*binary system*) – báze 2, dvě číslice (*bit* = ***bi**nary digi**t***) 0, 1:

$$\begin{aligned}(109.375)_{10} &= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &\quad + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= (1101101.011)_2\end{aligned}$$

1	1	0	1	1	0	1	0	1	1	LSB = least significant bit
MSB						LSB	MSB		LSB	MSB = most significant bit

1.1.1. Aritmetické operace s binárními čísly

Analogicky jako v dekadické soustavě:

$1 + 0 = 1, 1 + 1 = (10)_2 = (2)_{10}, (10)_2 + 1 = (11)_2 = (3)_{10}$, atd.

$$\begin{array}{cccccccc} & & 1 & 1 & 1 & 1 & 0 \\ + & & & 1 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

$$\begin{array}{cccccccc} & & & & & 1 & 1 & 1 \\ \times & & & & & 1 & 1 & 0 \\ \hline & & & & & 0 & 0 & 0 \\ & & & & 1 & 1 & 1 & \\ & & & 1 & 1 & 1 & & \\ \hline & & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

1.1.1. Aritmetické operace s binárními čísly

Analogicky jako v dekadické soustavě:

$1 + 0 = 1$, $1 + 1 = (10)_2 = (2)_{10}$, $(10)_2 + 1 = (11)_2 = (3)_{10}$, atd.

$$\begin{array}{rcccccc} & & 1 & 1 & 1 & 1 & 0 \\ + & & & 1 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

$$\begin{array}{rcccccc} & & & & 1 & 1 & 1 \\ \times & & & & 1 & 1 & 0 \\ \hline & & & & 0 & 0 & 0 \\ & & & 1 & 1 & 1 & \\ & & 1 & 1 & 1 & & \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

1.1.2. Konverze z dekadické do dvojkové soustavy

Celočíselná část:

109	Kvocient	54	27	13	6	3	1	0
: 2	Zbytek	1	0	1	1	0	1	1
		LSB						MSB

Zlomková část:

0.375	Zlomek	0.75	0.5	0
× 2	Celé č.	0	1	1
		MSB		LSB

Pozor! Opačné pořadí: LSB → MSB.

1.1.3. Oktalová a hexadecimální reprezentace

Oktalová: báze 8, osm číslic $0, \ldots, 7$, pokryto skupinou tří bitů. Binární číslo se rozdělí na skupiny tří bitů od tečky doleva a doprava (a přidají se nulové vycpávky, je-li třeba). Každá trojice se pak přepíše jako jediná oktalová číslice $0, \ldots, 7$:

$$(109.375)_{10} = (\boxed{001}\boxed{101}\boxed{101}.\boxed{011})_2 = (155.3)_8$$

1.1.3. Oktalová a hexadecimální reprezentace

Oktalová: báze 8, osm číslic $0, \ldots, 7$, pokryto skupinou tří bitů. Binární číslo se rozdělí na skupiny tří bitů od tečky doleva a doprava (a přidají se nulové vycpávky, je-li třeba). Každá trojice se pak přepíše jako jediná oktalová číslice $0, \ldots, 7$:

$$(109.375)_{10} = (\boxed{001}\boxed{101}\boxed{101}.\boxed{011})_2 = (155.3)_8$$

Hexadecimální: báze 16, šestnáct číslic $0, \ldots, 9, \mathrm{A} \equiv 10, \mathrm{B} \equiv 11, \mathrm{C} \equiv 12, \mathrm{D} \equiv 13, \mathrm{E} \equiv 14, \mathrm{F} \equiv 15$, pokryto skupinou čtyř bitů. Binární číslo se rozdělí na skupiny čtyř bitů od tečky doleva a doprava (a přidají se nulové vycpávky, je-li třeba). Každá čtveřice se pak přepíše jako jediná hexadecimální číslice $0, \ldots, 9, \mathrm{A}, \ldots, \mathrm{F}$:

$$(109.375)_{10} = (\boxed{0110}\boxed{1101}.\boxed{0110})_2 = (6\mathrm{D}.6)_{16}$$

1.2. Reprezentace dat

Data a programy jsou v paměti ukládány v binárním formátu. Paměť je organizována do skupin po 8 bitech (b) – bajtů (*byte*, *bytes*, B) – nejmenší adresovatelná jednotka: $1\,\mathrm{B} = 8\,\mathrm{b}$.

1 KB	1024 B	2^{10} B
1 MB	1024 KB = 1048576 B	2^{20} B
1 GB	1024 MB = 1073741824 B	2^{30} B
1 TB	1024 GB = 1099511627776 B	2^{40} B

Různé druhy fyzické paměti (hierarchie paměti):

Typ paměti	**Velikost**	**Přístupová doba**
Registry CPU	8 B	1 clock cycle
Cache, Level 1	126 KB–512 KB	1 ns
Cache, Level 2	512 KB–8 MB	10 ns
Hlavní paměť (RAM)	8 MB–4 GB	60 ns
Pevný disk (HDD)	2 GB–100 GB	10 ms

Několik bajtů (obvykle 4, tj. 32 b) tvoří slovo (*word*).

1.2.1. Znaky (characters)

Písmena abecedy (velká i malá), interpunkce, různé další symboly. ASCII (American Standard Code for Information Interchange): 7 bitů pro 1 znak $\Rightarrow$ $2^7 = 128$ reprezentovatelných znaků (0–127: ‚dolní polovina ASCII tabulky‘).

Zbylé pozice do 1 B: 128–255 (příp. (-128)–(-1) v případě znaménkového znaku): ‚horní polovina‘, obvykle pro znaky národních abeced, není dodržován standard ISO 8859-2 $\Rightarrow$ problémy s kódováním češtiny (další cp1250, PC-latin2, Kamenických, Mac OS, . . .).

Vyzkoušejte program `char2ascii` (zobrazí ASCII kód zadaného znaku).

1.2.1. Znaky (characters)

Písmena abecedy (velká i malá), interpunkce, různé další symboly. ASCII (American Standard Code for Information Interchange): 7 bitů pro 1 znak $\Rightarrow$ $2^7 = 128$ reprezentovatelných znaků (0–127: ‚dolní polovina ASCII tabulky‘).

Zbylé pozice do 1 B: 128–255 (příp. (-128)–(-1) v případě znaménkového znaku): ‚horní polovina‘, obvykle pro znaky národních abeced, není dodržován standard ISO 8859-2 $\Rightarrow$ problémy s kódováním češtiny (další cp1250, PC-latin2, Kamenických, Mac OS, . . .).

Vyzkoušejte program `char2ascii` (zobrazí ASCII kód zadaného znaku).

Znaky v paměti Umístění znaku v RAM (chápané jako dlouhá sekvence bajtů): jednoznačně určeno adresou. Např. máme-li k dispozici 512 MB RAM, tj. 2^{30} B, adresy pokrývají $0, \ldots, 2^{30} - 1 = (3\mathrm{FFFFFFF})_{16}$.

Vyzkoušejte program `charaddr` (zobrazí adresu, na níž je uložen zadaný znak).

Pro uložení posloupnosti znaků (znakového řetězce, *character string*) ‚Ahoj!‘ potřebujeme alokovat pět po sobě jdoucích bajtů, např.:

Adresa	Obsah	Poznámka
BFFFF260	’A’	první bajt
BFFFF261	’h’	
BFFFF262	’o’	
BFFFF263	’j’	
BFFFF264	’!’	poslední bajt

Pro přístup potřebujeme znát adresu prvního znaku (zde BFFFF260) a délku řetězce (zde 5).

Vyzkoušejte program `straddr` (zobrazí adresy, na nichž začíná a končí zadaný znakový řetězec).

1.2.2. Celá čísla (integers)

Neznaménková (unsigned) 1 B může reprezentovat celá čísla $0, \ldots, 2^8 - 1 = 255$, pro běžné aplikace málo. Standardní datové typy obvykle rezervují pro celé číslo 2, 4, 8 po sobě jdoucích bajtů. Závisí na platformě!! Např. v Unixu je `unsigned int` v jazyce C 4-bajtové celé číslo, v MS DOS 2-bajtové.

Příklad: 4-bajtové (32 bitové) neznaménkové celé číslo 37 je zobrazeno jako

00000000	00000000	00000000	00100101

Rozsah 32 bitových neznaménkových celých čísel je od 0

00000000	00000000	00000000	00000000

do $2^{32} - 1 = 4294967295$

11111111	11111111	11111111	11111111

Rozsah p-bitového neznaménkového celého čísla je $0, 1, \ldots, 2^p - 1$.

Pro demonstraci přetečení/podtečení (*overflow/underflow*) neznaménkových celých čísel vyzkoušejte programy `unsigover` a `unsigunder` [Sandu, 2001].

Znaménková (signed) Používají dvojkový doplněk (*two's complement*). p-bitové znaménkové celé číslo v rozsahu $-2^{p-1}, \ldots, -1, 0, 1, \ldots, 2^{p-1} - 1$ je reprezentováno nejmenším celým kladným číslem, s nímž je kongruentní modulo 2^p.

(číslo)$_{10}$	**(dvojkový doplněk)**$_{10}$	**(dvojkový doplněk)**$_2$
-2147483648	2147483648	10000000 00000000 00000000 00000000
-2147483647	2147483649	10000000 00000000 00000000 00000001
-2147483646	2147483650	10000000 00000000 00000000 00000010
⋮	⋮	⋮
-2	4294967294	11111111 11111111 11111111 11111110
-1	4294967295	11111111 11111111 11111111 11111111
0	0	00000000 00000000 00000000 00000000
1	1	00000000 00000000 00000000 00000001
2	2	00000000 00000000 00000000 00000010
⋮	⋮	⋮
2147483645	2147483645	01111111 11111111 11111111 11111101
2147483646	2147483646	01111111 11111111 11111111 11111110
2147483647	2147483647	01111111 11111111 11111111 11111111

Je-li k reprezentováno určitým bitovým vzorkem, pak $-(k+1)$ je reprezentováno inverzním bitovým vzorkem s $0 \to 1$, $1 \to 0$.

Důvod pro užívání dvojkového doplňku pro znaménková celá čísla: odčítání a operace se zápornými čísly jsou nahrazeny operacemi jen s kladnými čísly. Jiné způsoby (sign/magnitude a biased) viz [Sandu, 2001].

Pro demonstraci přetečení/podtečení znaménkových celých čísel vyzkoušejte programy `sigover` a `sigunder` [Sandu, 2001].

Pozor na přiřazování např. hodnoty neznaménkové celočíselné proměnné do znaménkové celočíselné proměnné, jako např. v tomto fragmentu:

```
unsigned int u;
signed int s;
s=u;
```

Pro demonstraci konverze neznaménkové celočíselné proměnné do znaménkové vyzkoušejte program `unsig2sig`.

Celá čísla v paměti Jak je např. uloženo nějaké 32-bitové (ne)znaménkové celé číslo, třeba $(37)_{10} = (\underbrace{\boxed{00000000}}_{B_0}\underbrace{\boxed{00000000}}_{B_1}\underbrace{\boxed{00000000}}_{B_2}\underbrace{\boxed{00100101}}_{B_3})_2$?

Potřebujeme alokovat čtyři po sobě jdoucí bajty $B_0, \ldots, B_3$.

Ale pozor! Pořadí ukládání bajtů vícebajtových datových typů (tj. cokoli kromě znaků), tak řečená endianita (*endianity, byte sex*) závisí na systému. Např. IBM a Sun užívají adresování Big Endian, Intel adresování Little Endian.

→ vzrůst adresy →

	B_0	B_1	B_2	B_3	

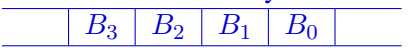
→ vzrůst adresy →

	B_3	B_2	B_1	B_0	

Celá čísla v paměti Jak je např. uloženo nějaké 32-bitové (ne)znaménkové celé číslo, třeba $(37)_{10} = (\underbrace{\boxed{00000000}}_{B_0}\underbrace{\boxed{00000000}}_{B_1}\underbrace{\boxed{00000000}}_{B_2}\underbrace{\boxed{00100101}}_{B_3})_2$?

Potřebujeme alokovat čtyři po sobě jdoucí bajty $B_0, \ldots, B_3$.

Ale pozor! Pořadí ukládání bajtů vícebajtových datových typů (tj. cokoli kromě znaků), tak řečená endianita (*endianity, byte sex*) závisí na systému. Např. IBM a Sun užívají adresování Big Endian, Intel adresování Little Endian.



Podle Gulliverových cest – Liliputáni se dělili na dvě skupiny, které se přely o to, na kterém konci se mají naklepávat vajíčka (Little Endians a Big Endians). Pro přístup potřebujeme znát nejnižší adresu (bajtu B_0 pro Big Endian, bajtu B_3 pro Little Endian) a velikost datového typu (zde 4 B).

Prostudujte zdrojový kód programu `byte_sex` [Kasprzak, 2004], jenž vypíše endianitu vašeho systému v době běhu.

Vyzkoušejte programy `uintaddr`/`sintaddr` (zobrazí počáteční a koncovou adresu zadaného (ne)znaménkového celého čísla).

Aritmetika počítačových celých čísel (s výjimkou celočíselného dělení) je, až na situace pře/podtečení, přesná, tj. numerický výsledek = matematický výsledek v $\mathbb{Z}$!! (Spočetná množina je ‚oseknuta‘ na konečnou.)

Reálná čísla $\mathbb{R}$ tvoří kontinuum (mohutnost $\aleph$), které musí v počítači být reprezentováno a vhodně aproximováno pomocí konečného počtu čísel. Lze reprezentovat jen čísla do určité meze, a v tomto rozsahu jen vhodně rozloženou konečnou podmnožinu $\mathbb{F} \subset \mathbb{R}$.

1.2.3. Reálná čísla

Obvykle reprezentována čísly v pohyblivé řádové čárce/tečce (*floating-point numbers*, FP). Historicky: čísla v pevné řádové čárce/tečce (*fixed-point numbers*) $\Rightarrow$ volba měřítka v rukou programátora (John von Neumann).

Motivace FP – hračkový model Pro jednoduchost volíme bázi $\beta = 10$. Každé číslo $x \in \mathbb{R}$ lze jednoznačně psát v normalizovaném tvaru

$$x = \underbrace{\sigma}_{\substack{\pm 1 \\ \text{znam., 1 b}}} \times \underbrace{m}_{\substack{1 \le m < 10 \\ \text{mantisa} \in \mathbb{R},\, d_\mathrm{m}\ \text{č.}}} \times 10 \overbrace{e}^{\substack{\exp. \in \mathbb{Z},\, d_\mathrm{e}\ \text{č.}}} \tag{1.1}$$

Např. $109.375 = +1 \times 1.09375 \times 10^2$. Nerovnost omezující mantisu zajišťuje jedinou nenulovou číslici před její desetinnou tečkou (proto pohyblivá, $\Rightarrow$ jednoznačnost); nelze třeba $+1 \times 0.109375 \times 10^3$ nebo $+1 \times 0.00109375 \times 10^5$.

Mějme pro uložení 6 dekadických cifer: $\boxed{\sigma}\boxed{m_1 m_2 m_3}\boxed{e_1 e_2} = \boxed{+}\boxed{109}\boxed{02}$. Zde $d_\mathrm{m} = 3$, $d_\mathrm{e} = 2$, $m_1 \neq 0$. Takových FP čísel je již konečně mnoho.

Lze v našem hračkovém modelu uložit $x = 0.000123$? Ne: $\boxed{+}\boxed{000}\boxed{00}$, úplná ztráta informace! Exponent je totiž nezáporný! Co s tím?

Lze v našem hračkovém modelu uložit $x = 0.000123$? Ne: $\boxed{+}\boxed{000}\boxed{00}$, úplná ztráta informace! Exponent je totiž nezáporný! Co s tím?

Použít bias: jsou-li jako exponent uloženy cifry e_1e_2, skutečný exponent je $e_1e_2 - 49$ (49 je bias). Skutečný exponent má rozsah -49 až 50, ale je uložen jako 00 až 99. Nemusíme pak ukládat znaménko exponentu, za cenu zmenšení rozsahu. Číslo $x = 0.000123 = +1 \times 1.23 \times 10^{-4}$ je tedy uloženo jako $\boxed{+}\boxed{123}\boxed{45}$. Obvykle se nejmenší/největší hodnota exponentu (zde 00 a 99) rezervuje pro reprezentaci speciální čísel jako $\pm\infty$, 0, subnormální čísla, `NaN` (viz sekci 1.2.3), tj. $e_1e_2 \in \{01, \ldots, 98\}$. Volba biasu je určena podmínkou $1/x_{\min} < x_{\max}$ a opačně.

Lze v našem hračkovém modelu uložit $x = 0.000123$? Ne: $\boxed{+}\boxed{000}\boxed{00}$, úplná ztráta informace! Exponent je totiž nezáporný! Co s tím?

Použít bias: jsou-li jako exponent uloženy cifry e_1e_2, skutečný exponent je $e_1e_2 - 49$ (49 je bias). Skutečný exponent má rozsah -49 až 50, ale je uložen jako 00 až 99. Nemusíme pak ukládat znaménko exponentu, za cenu zmenšení rozsahu. Číslo $x = 0.000123 = +1 \times 1.23 \times 10^{-4}$ je tedy uloženo jako $\boxed{+}\boxed{123}\boxed{45}$. Obvykle se nejmenší/největší hodnota exponentu (zde 00 a 99) rezervuje pro reprezentaci speciální čísel jako $\pm\infty$, 0, subnormální čísla, `NaN` (viz sekci 1.2.3), tj. $e_1e_2 \in \{01, \ldots, 98\}$. Volba biasu je určena podmínkou $1/x_{\min} < x_{\max}$ a opačně.

Max. absolutní hodnota normalizovaného FP čísla: pro $m_1m_2m_3 = 999$, $e_1e_2 = 98$ je $x_{\max} = 9.99 \times 10^{49}$.

Min. absolutní hodnota normalizovaného FP čísla: pro $m_1m_2m_3 = 100$, $e_1e_2 = 01$ je $x_{\min} = 1.00 \times 10^{-48}$.

Nula je reprezentována $m_1m_2m_3 = 000$, $e_1e_2 = 00$. Obě (± 0) ekvivalentní.

Subnormální čísla: upustíme-li od normalizovaného tvaru ($m_1 \neq 0$), lze reprezentovat čísla menší než $x_{\min}$: $m_1m_2m_3 = 010$, $e_1e_2 = 00$ dává $x = 10^{-49}$, $m_1m_2m_3 = 001$, $e_1e_2 = 00$ dává $x = 10^{-50}$.

Lze v našem hračkovém modelu uložit $x = 0.000123$? Ne: $\boxed{+}\boxed{000}\boxed{00}$, úplná ztráta informace! Exponent je totiž nezáporný! Co s tím?

Použít bias: jsou-li jako exponent uloženy cifry e_1e_2, skutečný exponent je $e_1e_2 - 49$ (49 je bias). Skutečný exponent má rozsah -49 až 50, ale je uložen jako 00 až 99. Nemusíme pak ukládat znaménko exponentu, za cenu zmenšení rozsahu. Číslo $x = 0.000123 = +1 \times 1.23 \times 10^{-4}$ je tedy uloženo jako $\boxed{+}\boxed{123}\boxed{45}$. Obvykle se nejmenší/největší hodnota exponentu (zde 00 a 99) rezervuje pro reprezentaci speciální čísel jako $\pm\infty$, 0, subnormální čísla, `NaN` (viz sekci 1.2.3), tj. $e_1e_2 \in \{01, \ldots, 98\}$. Volba biasu je určena podmínkou $1/x_{\min} < x_{\max}$ a opačně.

Max. absolutní hodnota normalizovaného FP čísla: pro $m_1m_2m_3 = 999$, $e_1e_2 = 98$ je $x_{\max} = 9.99 \times 10^{49}$.

Min. absolutní hodnota normalizovaného FP čísla: pro $m_1m_2m_3 = 100$, $e_1e_2 = 01$ je $x_{\min} = 1.00 \times 10^{-48}$.

Nula je reprezentována $m_1m_2m_3 = 000$, $e_1e_2 = 00$. Obě (± 0) ekvivalentní.

Subnormální čísla: upustíme-li od normalizovaného tvaru ($m_1 \neq 0$), lze reprezentovat čísla menší než $x_{\min}$: $m_1m_2m_3 = 010$, $e_1e_2 = 00$ dává $x = 10^{-49}$, $m_1m_2m_3 = 001$, $e_1e_2 = 00$ dává $x = 10^{-50}$.

Subnormální čísla zlepšují přesnost v okolí 0: uvažujme fragment kódu `if (x!=y) z=1.0/(x-y);`, a $x = 1.02 \times 10^{-48}$, $y = 1.01 \times 10^{-48}$. Podmínka je v rámci normalizovaných FP splněna, ale rozdíl $x - y$ je pod rozsahem normalizovaných FP ($\Rightarrow$ dělení nulou!), nikoli však subnormálních FP, $x - y = 10^{-50}$.

Volíme-li bázi $\beta = 2$, máme normalizovaný tvar

$$x = \underbrace{\sigma}_{\substack{0 \equiv +, 1 \equiv - \\ \text{znam., 1 bit}}} \times \underbrace{m}_{\substack{1 \le m < 2 \\ \text{mantisa} \in \mathbb{R},\ d_\text{m} \text{ bitů}}} \times 2^{\overbrace{e}^{\substack{\text{exp.} \in \mathbb{Z},\ d_\text{e} \text{ bitů}}}} \tag{1.2}$$

Např. $(109.375)_{10} = (1101101.011)_2 = +1 \times 1.101101011 \times 2^6$. Mějme pro uložení 11 bitů: $d_\text{m} = 6$, $d_\text{e} = 4$, jeden bit pro znaménko. FP reprezentace je (zatím bez biasu) $\boxed{\sigma}\boxed{m_1m_2m_3m_4m_5m_6}\boxed{e_1e_2e_3e_4} = \boxed{0}\boxed{110110}\boxed{0110}$. Požadavek na normalizaci ve dvojkové soustavě ($m_1 \neq 0$) znamená $m_1 = 1$, je tedy zbytečné m_1 ukládat a místo toho uložit další, nejméně významný bit, navíc. Toto se nazývá technika skrytého bitu (*hidden bit technique*): FP reprezentace je pak $\boxed{0}\boxed{101101}\boxed{0110}$.

Volíme-li bázi $\beta = 2$, máme normalizovaný tvar

$$x = \underbrace{\sigma}_{\substack{0 \equiv +, 1 \equiv - \\ \text{znam., 1 bit}}} \times \underbrace{m}_{\substack{1 \le m < 2 \\ \text{mantisa} \in \mathbb{R},\ d_\text{m} \text{ bitů}}} \times 2^{\overbrace{e}^{\text{exp.} \in \mathbb{Z},\ d_\text{e} \text{ bitů}}} \tag{1.2}$$

Např. $(109.375)_{10} = (1101101.011)_2 = +1 \times 1.101101011 \times 2^6$. Mějme pro uložení 11 bitů: $d_\text{m} = 6$, $d_\text{e} = 4$, jeden bit pro znaménko. FP reprezentace je (zatím bez biasu) $\boxed{\sigma}\boxed{m_1m_2m_3m_4m_5m_6}\boxed{e_1e_2e_3e_4} = \boxed{0}\boxed{110110}\boxed{0110}$. Požadavek na normalizaci ve dvojkové soustavě ($m_1 \neq 0$) znamená $m_1 = 1$, je tedy zbytečné m_1 ukládat a místo toho uložit další, nejméně významný bit, navíc. Toto se nazývá technika skrytého bitu (*hidden bit technique*): FP reprezentace je pak $\boxed{0}\boxed{101101}\boxed{0110}$.

IEEE standard Standardizuje FP čísla, jak provádět aritmetické operace, a ošetření výjimek (*exceptions handling*). Vyvinut v 80 letech 20. století (IEEE, W. Kahan), nyní respektován všemi výrobci CPU.

ANSI/IEEE 754-1985 (pro $\beta = 2$), ANSI/IEEE 854-1987 (pro obecné β)
ISO standard: IEC 559: 1989

Číslo $x \in \mathbb{F}$ je binárně reprezentováno jako

$$x = \underbrace{\sigma}_{\text{1-bitové znam.}} \times \underbrace{m}_{d_{\mathrm{m}}\text{-bitová mantisa}} \times 2^{\overbrace{e}^{d_{\mathrm{e}}\text{-bitový exp.}} - \overbrace{E}^{\text{bias}}}$$

Číslo $x \in \mathbb{F}$ je binárně reprezentováno jako

$$x = \underbrace{\sigma}_{\text{1-bitové znam.}} \times \underbrace{m}_{d_\mathrm{m}\text{-bitová mantisa}} \times 2^{\overbrace{e}^{d_\mathrm{e}\text{-bitový exp.}} - \overbrace{E}^{\text{bias}}} \tag{1.3}$$

Jednoduchá přesnost (single precision)

Délka:	4 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záporné
Exponent:	$d_\mathrm{e} = 8$, bias $E = (127)_{10}$	Mant.:	$d_\mathrm{m} = 23$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	1.18×10^{-38}, 3.40×10^{38}	ε, ε_-:	1.1921×10^{-7}, 5.9605×10^{-8}

σ	$e_1 e_2 e_3 \cdots e_8$	$m_1 m_2 m_3 \cdots m_{23}$

Číslo $x \in \mathbb{F}$ je binárně reprezentováno jako

$$x = \underbrace{\sigma}_{\text{1-bitové znam.}} \times \underbrace{m}_{d_\mathrm{m}\text{-bitová mantisa}} \times 2^{\overbrace{e}^{d_\mathrm{e}\text{-bitový exp.}} - \overbrace{E}^{\text{bias}}} \tag{1.3}$$

Jednoduchá přesnost (single precision)

Délka:	4 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záporné
Exponent:	$d_\mathrm{e} = 8$, bias $E = (127)_{10}$	Mant.:	$d_\mathrm{m} = 23$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	1.18×10^{-38}, 3.40×10^{38}	ε, ε_-:	1.1921×10^{-7}, 5.9605×10^{-8}

σ	$e_1e_2e_3\cdots e_8$	$m_1m_2m_3\cdots m_{23}$

$e_1e_2e_3\cdots e_8$	**Hodnota**	**Poznámka**
$(00000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{23})_2 \times 2^{-126}$	$m_1 = \cdots = m_{23} = 0$: nula ($\pm 0$)
	$\pm(0.m_1\cdots m_{23})_2 \times 2^{-126}$	aspoň jedno $m_i \neq 0$: subnormální
$(00000001)_2 = (1)_{10}$	$\pm(1.m_1\cdots m_{23})_2 \times 2^{-126}$	normalizované číslo
$\cdots$	$\cdots$	$\cdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.m_1\cdots m_{23})_2 \times 2^{0}$	normalizované číslo
$(10000000)_2 = (128)_{10}$	$\pm(1.m_1\cdots m_{23})_2 \times 2^{1}$	normalizované číslo
$\cdots$	$\cdots$	$\cdots$
$(11111110)_2 = (254)_{10}$	$\pm(1.m_1\cdots m_{23})_2 \times 2^{127}$	normalizované číslo
$(11111111)_2 = (255)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{23} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 11$, bias $E = (1023)_{10}$	Mant.:	$d_\mathrm{m} = 52$, technika skrytého bitu
$\lvert x_{\mathrm{min,max}}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1 e_2 e_3 \cdots e_{11}$	$m_1 m_2 m_3 \cdots m_{52}$

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 11$, bias $E = (1023)_{10}$	Mant.:	$d_\mathrm{m} = 52$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1e_2e_3\cdots e_{11}$	$m_1m_2m_3\cdots m_{52}$

$e_1e_2e_3\cdots e_{11}$	**Hodnota**	**Poznámka**
$(00000000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	$m_1 = \cdots = m_{52} = 0$: nula ($\pm 0$)
	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	aspoň jedno $m_i \neq 0$: subnormální
$(00000000001)_2 = (1)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{-1022}$	normalizované číslo
$\cdots$	$\cdots$	$\cdots$
$(01111111111)_2 = (1023)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{0}$	normalizované číslo
$(10000000000)_2 = (1024)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1}$	normalizované číslo
$\cdots$	$\cdots$	$\cdots$
$(11111111110)_2 = (2046)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1023}$	normalizované číslo
$(11111111111)_2 = (2047)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{52} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_e = 11$, bias $E = (1023)_{10}$	Mant.:	$d_m = 52$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1e_2e_3\cdots e_{11}$	$m_1m_2m_3\cdots m_{52}$

$e_1e_2e_3\cdots e_{11}$	**Hodnota**	**Poznámka**
$(00000000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	$m_1 = \cdots = m_{52} = 0$: nula ($\pm 0$)
	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	aspoň jedno $m_i \neq 0$: subnormální
$(00000000001)_2 = (1)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{-1022}$	normalizované číslo
...	...	...
$(01111111111)_2 = (1023)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{0}$	normalizované číslo
$(10000000000)_2 = (1024)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(11111111110)_2 = (2046)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1023}$	normalizované číslo
$(11111111111)_2 = (2047)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{52} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Jaké číslo reprezentuje v jedn. přes. |0|10000000|00110000000000000000000|?

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_e = 11$, bias $E = (1023)_{10}$	Mant.:	$d_m = 52$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1e_2e_3\cdots e_{11}$	$m_1m_2m_3\cdots m_{52}$

$e_1e_2e_3\cdots e_{11}$	**Hodnota**	**Poznámka**
$(00000000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$ $\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	$m_1 = \cdots = m_{52} = 0$: nula ($\pm 0$) aspoň jedno $m_i \neq 0$: subnormální
$(00000000001)_2 = (1)_{10}$ ... $(01111111111)_2 = (1023)_{10}$ $(10000000000)_2 = (1024)_{10}$ ... $(11111111110)_2 = (2046)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{-1022}$ ... $\pm(1.m_1\cdots m_{52})_2 \times 2^{0}$ $\pm(1.m_1\cdots m_{52})_2 \times 2^{1}$ ... $\pm(1.m_1\cdots m_{52})_2 \times 2^{1023}$	normalizované číslo ... normalizované číslo normalizované číslo ... normalizované číslo
$(11111111111)_2 = (2047)_{10}$	‚**+inf**' a ‚**-inf**' ‚**NaN**'	$m_1 = \cdots = m_{52} = 0$: $\pm\infty$ aspoň jedno $m_i \neq 0$: not a number

Jaké číslo reprezentuje v jedn. přes. |0|10000000|00110000000000000000000|? $(1 + 2^{-3} + 2^{-4}) \times 2^1 = \frac{19}{8}$

Jak je zobrazeno $x = 5$ jako 4 B integer?

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_e = 11$, bias $E = (1023)_{10}$	Mant.:	$d_m = 52$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1e_2e_3\cdots e_{11}$	$m_1m_2m_3\cdots m_{52}$

$e_1e_2e_3\cdots e_{11}$	**Hodnota**	**Poznámka**
$(00000000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	$m_1 = \cdots = m_{52} = 0$: nula ($\pm 0$)
	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	aspoň jedno $m_i \neq 0$: subnormální
$(00000000001)_2 = (1)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{-1022}$	normalizované číslo
...	...	...
$(01111111111)_2 = (1023)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{0}$	normalizované číslo
$(10000000000)_2 = (1024)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(11111111110)_2 = (2046)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1023}$	normalizované číslo
$(11111111111)_2 = (2047)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{52} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Jaké číslo reprezentuje v jedn. přes. |0|10000000|00110000000000000000000|? $\frac{8}{19} = 2^{1} \times (2^{-4} + 2^{-5} + 1)$

Jak je zobrazeno $x = 5$ jako 4 B integer? |00000000000000000000000000000101|

Jak je zobrazeno $x = 5.0$ jako single prec.?

Dvojnásobná přesnost (double precision)

Délka:	8 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_e = 11$, bias $E = (1023)_{10}$	Mant.:	$d_m = 52$, technika skrytého bitu
$\lvert x_{\min,\max}\rvert$:	2.23×10^{-308}, 1.79×10^{308}	ε, ε_-:	2.22×10^{-16}, 1.11×10^{-16}

σ	$e_1e_2e_3\cdots e_{11}$	$m_1m_2m_3\cdots m_{52}$

$e_1e_2e_3\cdots e_{11}$	**Hodnota**	**Poznámka**
$(00000000000)_2 = (0)_{10}$	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	$m_1 = \cdots = m_{52} = 0$: nula ($\pm 0$)
	$\pm(0.m_1\cdots m_{52})_2 \times 2^{-1022}$	aspoň jedno $m_i \neq 0$: subnormální
$(00000000001)_2 = (1)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{-1022}$	normalizované číslo
...	...	...
$(01111111111)_2 = (1023)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{0}$	normalizované číslo
$(10000000000)_2 = (1024)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(11111111110)_2 = (2046)_{10}$	$\pm(1.m_1\cdots m_{52})_2 \times 2^{1023}$	normalizované číslo
$(11111111111)_2 = (2047)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{52} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Jaké číslo reprezentuje v jedn. přes. |0|10000000|00110000000000000000000|? $\frac{8}{61} = 2^1 \times (2^{-4} + 2^{-3} + 1)$

Jak je zobrazeno $x = 5$ jako 4 B integer? |00000000000000000000000000000101|

Jak je zobrazeno $x = 5.0$ jako single prec.? |00000000000000000000001|10000001|0| $\equiv 2^2 \times (2^{-2} + 1)$

Rozšířená dvojnásobná přesnost (double-extended precision)

Doporučeno normou ANSI/IEEE 754-1985 (mají registry mikroprocesorů Intel).

Délka:	10 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 15$, bias $E = (16383)_{10}$	Mantisa:	$d_\mathrm{m} = 64$, bez skrytého bitu

σ	$e_1 e_2 e_3 \cdots e_{15}$	$m_1 m_2 m_3 \cdots m_{64}$

Rozšířená dvojnásobná přesnost (double-extended precision)

Doporučeno normou ANSI/IEEE 754-1985 (mají registry mikroprocesorů Intel).

Délka:	10 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 15$, bias $E = (16383)_{10}$	Mantisa:	$d_\mathrm{m} = 64$, bez skrytého bitu

σ	$e_1e_2e_3\cdots e_{15}$	$m_1m_2m_3\cdots m_{64}$

$e_1e_2e_3\cdots e_{15}$	**Hodnota**	**Poznámka**
$(000000000000000)_2 = (0)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	$m_1 = \cdots = m_{64} = 0$: nula ($\pm 0$)
$(000000000000001)_2 = (1)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	normalizované číslo
$\ldots$	$\ldots$	$\ldots$
$(011111111111111)_2 = (16383)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{0}$	normalizované číslo
$(100000000000000)_2 = (16384)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{1}$	normalizované číslo
$\ldots$	$\ldots$	$\ldots$
$(111111111111110)_2 = (32766)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{16363}$	normalizované číslo
$(111111111111111)_2 = (32767)_{10}$	„`+inf`“ a „`-inf`“	$m_1 = \cdots = m_{64} = 0$: $\pm\infty$
	„`NaN`“	aspoň jedno $m_i \neq 0$: not a number

Užitečné pro dočasné uchování mezivýsledků (v registrech). Existuje také rozšířená jednoduchá přesnost (*single-extended precision*). Podrobně viz [Goldberg, 1991].

Rozšířená dvojnásobná přesnost (double-extended precision)

Doporučeno normou ANSI/IEEE 754-1985 (mají registry mikroprocesorů Intel).

Délka:	10 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 15$, bias $E = (16383)_{10}$	Mantisa:	$d_\mathrm{m} = 64$, bez skrytého bitu

σ	$e_1e_2e_3\cdots e_{15}$	$m_1m_2m_3\cdots m_{64}$

$e_1e_2e_3\cdots e_{15}$	**Hodnota**	**Poznámka**
$(000000000000000)_2 = (0)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	$m_1 = \cdots = m_{64} = 0$: nula ($\pm 0$)
$(000000000000001)_2 = (1)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	normalizované číslo
...	...	...
$(011111111111111)_2 = (16383)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{0}$	normalizované číslo
$(100000000000000)_2 = (16384)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(111111111111110)_2 = (32766)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{16363}$	normalizované číslo
$(111111111111111)_2 = (32767)_{10}$	‚`+inf`‘ a ‚`-inf`‘	$m_1 = \cdots = m_{64} = 0$: $\pm\infty$
	‚`NaN`‘	aspoň jedno $m_i \neq 0$: not a number

Užitečné pro dočasné uchování mezivýsledků (v registrech). Existuje také rozšířená jednoduchá přesnost (*single-extended precision*). Podrobně viz [Goldberg, 1991].

Co představuje |0|10000000|10000000000000000000000| jako single prec. FP číslo a co jako 4 B integer?

Rozšířená dvojnásobná přesnost (double-extended precision)

Doporučeno normou ANSI/IEEE 754-1985 (mají registry mikroprocesorů Intel).

Délka:	10 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_{\mathrm{e}} = 15$, bias $E = (16383)_{10}$	Mantisa:	$d_{\mathrm{m}} = 64$, bez skrytého bitu

σ	$e_1 e_2 e_3 \cdots e_{15}$	$m_1 m_2 m_3 \cdots m_{64}$

$e_1 e_2 e_3 \cdots e_{15}$	**Hodnota**	**Poznámka**
$(000000000000000)_2 = (0)_{10}$	$\pm(m_1.m_2 \cdots m_{64})_2 \times 2^{-16362}$	$m_1 = \cdots = m_{64} = 0$: nula ($\pm 0$)
$(000000000000001)_2 = (1)_{10}$	$\pm(m_1.m_2 \cdots m_{64})_2 \times 2^{-16362}$	normalizované číslo
...	...	...
$(011111111111111)_2 = (16383)_{10}$	$\pm(m_1.m_2 \cdots m_{64})_2 \times 2^{0}$	normalizované číslo
$(100000000000000)_2 = (16384)_{10}$	$\pm(m_1.m_2 \cdots m_{64})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(111111111111110)_2 = (32766)_{10}$	$\pm(m_1.m_2 \cdots m_{64})_2 \times 2^{16363}$	normalizované číslo
$(111111111111111)_2 = (32767)_{10}$	‚`+inf`' a ‚`-inf`' ‚`NaN`'	$m_1 = \cdots = m_{64} = 0$: $\pm\infty$ aspoň jedno $m_i \neq 0$: not a number

Užitečné pro dočasné uchování mezivýsledků (v registrech). Existuje také rozšířená jednoduchá přesnost (*single-extended precision*). Podrobně viz [Goldberg, 1991].

Co představuje |0|10000000|10000000000000000000000| jako single prec. FP číslo a co jako 4 B integer?

3.0 a 1077936128

Jak jsou v single prec. zobrazena čísla 2^{-126} a $(2 - 2^{-23}) \times 2^{127}$?

Rozšířená dvojnásobná přesnost (double-extended precision)

Doporučeno normou ANSI/IEEE 754-1985 (mají registry mikroprocesorů Intel).

Délka:	10 B (podléhají endianitě)	Znam.:	1 bit; 0 pro kladné, 1 pro záp.
Exponent:	$d_\mathrm{e} = 15$, bias $E = (16383)_{10}$	Mantisa:	$d_\mathrm{m} = 64$, bez skrytého bitu

σ	$e_1e_2e_3\cdots e_{15}$	$m_1m_2m_3\cdots m_{64}$

$e_1e_2e_3\cdots e_{15}$	**Hodnota**	**Poznámka**
$(000000000000000)_2 = (0)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	$m_1 = \cdots = m_{64} = 0$: nula ($\pm 0$)
$(000000000000001)_2 = (1)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{-16362}$	normalizované číslo
...	...	...
$(011111111111111)_2 = (16383)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{0}$	normalizované číslo
$(100000000000000)_2 = (16384)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{1}$	normalizované číslo
...	...	...
$(111111111111110)_2 = (32766)_{10}$	$\pm(m_1.m_2\cdots m_{64})_2 \times 2^{16363}$	normalizované číslo
$(111111111111111)_2 = (32767)_{10}$	‚`+inf`' a ‚`-inf`' ‚`NaN`'	$m_1 = \cdots = m_{64} = 0$: $\pm\infty$ aspoň jedno $m_i \neq 0$: not a number

Užitečné pro dočasné uchování mezivýsledků (v registrech). Existuje také rozšířená jednoduchá přesnost (*single-extended precision*). Podrobně viz [Goldberg, 1991].

?? Co představuje |0|10000000|10000000000000000000000| jako single prec. FP číslo a co jako 4 B integer?
3.0 a 1077936128
Jak jsou v single prec. zobrazena čísla 2^{-126} a $(2-2^{-23}) \times 2^{127}$?
|0|00000001|00000000000000000000000| a |0|11111110|11111111111111111111111|

Strojové epsilon Jak probíhá např. sečtení dvou FP čísel? Pokud nejsou exponenty stejné, zvětší se exponent menšího čísla na hodnotu většího čísla, a zároveň se zmenšuje mantisa posunem cifer o potřebný počet míst doprava (tj. dělením β, *right-shifting*).

Např. v hračkovém modelu sčítáme $1 = 1.00 \times 10^0$ a $0.01 = 1.00 \times 10^{-2}$. Upravíme druhý sčítanec na 0.01×10^0 a sečteme: $1.00 \times 10^0 + 0.01 \times 10^0 = 1.01 \times 10^0$. OK!

Strojové epsilon Jak probíhá např. sečtení dvou FP čísel? Pokud nejsou exponenty stejné, zvětší se exponent menšího čísla na hodnotu většího čísla, a zároveň se zmenšuje mantisa posunem cifer o potřebný počet míst doprava (tj. dělením β, *right-shifting*).

Např. v hračkovém modelu sčítáme $1 = 1.00 \times 10^0$ a $0.01 = 1.00 \times 10^{-2}$. Upravíme druhý sčítanec na 0.01×10^0 a sečteme: $1.00 \times 10^0 + 0.01 \times 10^0 = 1.01 \times 10^0$. OK!

Kdyby byl druhý sčítanec nejbližší menší FP číslo než 0.01, tj. 9.99×10^{-3}, dostaneme srovnáním jeho exponentu a posunem mantisy o tři pozice $1.00 \times 10^0 + 0.00999 \times 10^0 = 1.00 \times 10^0$! Ztráta signifikantní informace!

Číslo 0.01 je nejmenší kladné FP číslo ε, pro které platí $1 + \varepsilon > 1$, nebo jinak, $1 + \varepsilon$ je nejbližší FP číslo větší než 1. Nazývá se strojové epsilon (*machine epsilon*). V našem hračkovém modelu $\varepsilon = 0.01$.

Strojové epsilon Jak probíhá např. sečtení dvou FP čísel? Pokud nejsou exponenty stejné, zvětší se exponent menšího čísla na hodnotu většího čísla, a zároveň se zmenšuje mantisa posunem cifer o potřebný počet míst doprava (tj. dělením β, *right-shifting*).

Např. v hračkovém modelu sčítáme $1 = 1.00 \times 10^0$ a $0.01 = 1.00 \times 10^{-2}$. Upravíme druhý sčítanec na 0.01×10^0 a sečteme: $1.00 \times 10^0 + 0.01 \times 10^0 = 1.01 \times 10^0$. OK!

Kdyby byl druhý sčítanec nejbližší menší FP číslo než 0.01, tj. 9.99×10^{-3}, dostaneme srovnáním jeho exponentu a posunem mantisy o tři pozice $1.00 \times 10^0 + 0.00999 \times 10^0 = 1.00 \times 10^0$! Ztráta signifikantní informace!

Číslo 0.01 je nejmenší kladné FP číslo ε, pro které platí $1 + \varepsilon > 1$, nebo jinak, $1 + \varepsilon$ je nejbližší FP číslo větší než 1. Nazývá se strojové epsilon (*machine epsilon*). V našem hračkovém modelu $\varepsilon = 0.01$.

Analogicky v binární IEEE aritmetice je strojové epsilon $\varepsilon = 2^{-d_\mathrm{m}}$ (při použití skrytého bitu) a $\varepsilon = 2^{1-d_\mathrm{m}}$ (bez použití skrytého bitu). (Lze definovat rovněž strojové epsilon ε_- pro odečítání od 1.)

Demonstrace pro single precision:

$$1.0 \equiv |0|01111111|00000000000000000000000|$$

$$\varepsilon \equiv |0|\underbrace{01101000}_{(104)_{10}}|00000000000000000000000|$$

Zvětšíme-li exponent ε o 1, musíme mantisu dělit dvojkou, čímž se skrytý bit 1 posune na první místo v mantise (teď už ale žádný další skrytý bit u ε není!):

$$\varepsilon = |0|\underbrace{01101001}_{(105)_{10}}|10000000000000000000000|$$

Demonstrace pro single precision:

$$1.0 \equiv |0|01111111|00000000000000000000000|$$

$$\varepsilon \equiv |0|\underbrace{01101000}_{(104)_{10}}|00000000000000000000000|$$

Zvětšíme-li exponent ε o 1, musíme mantisu dělit dvojkou, čímž se skrytý bit 1 posune na první místo v mantise (teď už ale žádný další skrytý bit u ε není!):

$$\varepsilon = |0|\underbrace{01101001}_{(105)_{10}}|10000000000000000000000|$$

Dalším zvětšením exponentu o 1 a posunem mantisy doprava máme:

$$\varepsilon = |0|\underbrace{01101010}_{(106)_{10}}|01000000000000000000000|$$

Po $127 - 104 = 23 = d_{\mathrm{m}}$ takových operacích je mantisa srovnána a skrytý bit vytlačen na pravou krajní pozici:

$$\varepsilon = |0|\underbrace{01111111}_{(127)_{10}}|00000000000000000000001|$$

Vezmeme-li místo čísla 1.0 libovolné FP číslo x s obecným exponentem $e - E$ a mantisou m nejvýše $|111\ldots10|$, bude příslušné strojové epsilon mít exponent menší o d_{m} a v mantise samé nuly, tj. nejbližší FP číslo bude o $\varepsilon \times 2^{e-E}$ větší. Pro $1 \leq x < 2$ je $e - E = 0$, pro $2 \leq x < 4$ je $e - E = 1$ (mezera mezi FP čísly 2ε), pro $0.5 \leq x < 1$ je $e - E = -1$ (mezera mezi FP čísly $\varepsilon/2$), apod. Čím větší je FP číslo, tím větší je mezera mezi nimi; FP čísla mají vysokou hustotu v blízkosti nuly a čím více se od nuly vzdalujeme, tím se stávají řidšími – viz obrázek na následující straně.

Jaká je mezera mezi 1024 a nejbližším větším IEEE single precision FP číslem?

Vezmeme-li místo čísla 1.0 libovolné FP číslo x s obecným exponentem $e - E$ a mantisou m nejvýše $|111\ldots10|$, bude příslušné strojové epsilon mít exponent menší o d_{m} a v mantise samé nuly, tj. nejbližší FP číslo bude o $\varepsilon \times 2^{e-E}$ větší. Pro $1 \leq x < 2$ je $e - E = 0$, pro $2 \leq x < 4$ je $e - E = 1$ (mezera mezi FP čísly 2ε), pro $0.5 \leq x < 1$ je $e - E = -1$ (mezera mezi FP čísly $\varepsilon/2$), apod. Čím větší je FP číslo, tím větší je mezera mezi nimi; FP čísla mají vysokou hustotu v blízkosti nuly a čím více se od nuly vzdalujeme, tím se stávají řidšími – viz obrázek na následující straně.

Jaká je mezera mezi 1024 a nejbližším větším IEEE single precision FP číslem?
$2^{-13} \approx 0.00012$

Uvažujte číslo $(0.1)_{10}$. Jaká je jeho single precision FP reprezentace?

Vezmeme-li místo čísla 1.0 libovolné FP číslo x s obecným exponentem $e - E$ a mantisou m nejvýše $|111\ldots10|$, bude příslušné strojové epsilon mít exponent menší o d_{m} a v mantise samé nuly, tj. nejbližší FP číslo bude o $\varepsilon \times 2^{e-E}$ větší. Pro $1 \leq x < 2$ je $e - E = 0$, pro $2 \leq x < 4$ je $e - E = 1$ (mezera mezi FP čísly 2ε), pro $0.5 \leq x < 1$ je $e - E = -1$ (mezera mezi FP čísly $\varepsilon/2$), apod. Čím větší je FP číslo, tím větší je mezera mezi nimi; FP čísla mají vysokou hustotu v blízkosti nuly a čím více se od nuly vzdalujeme, tím se stávají řidšími – viz obrázek na následující straně.

Jaká je mezera mezi 1024 a nejbližším větším IEEE single precision FP číslem?

$1024_{\varepsilon} \approx 0.0001_2$

Uvažujte číslo $(0.1)_{10}$. Jaká je jeho single precision FP reprezentace?

$$|0|\overbrace{01111011}^{(123)_{10}}|10011001100110011001100|$$

$$= (1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21} + \cdots) \times 2^{-4}$$

Má technika skrytého bitu v tomto případě za důsledek přesnější reprezentaci?

Vezmeme-li místo čísla 1.0 libovolné FP číslo x s obecným exponentem $e - E$ a mantisou m nejvýše $|111\ldots10|$, bude příslušné strojové epsilon mít exponent menší o d_{m} a v mantise samé nuly, tj. nejbližší FP číslo bude o $\varepsilon \times 2^{e-E}$ větší. Pro $1 \leq x < 2$ je $e-E=0$, pro $2 \leq x < 4$ je $e-E=1$ (mezera mezi FP čísly 2ε), pro $0.5 \leq x < 1$ je $e - E = -1$ (mezera mezi FP čísly $\varepsilon/2$), apod. Čím větší je FP číslo, tím větší je mezera mezi nimi; FP čísla mají vysokou hustotu v blízkosti nuly a čím více se od nuly vzdalujeme, tím se stávají řidšími – viz obrázek na následující straně.

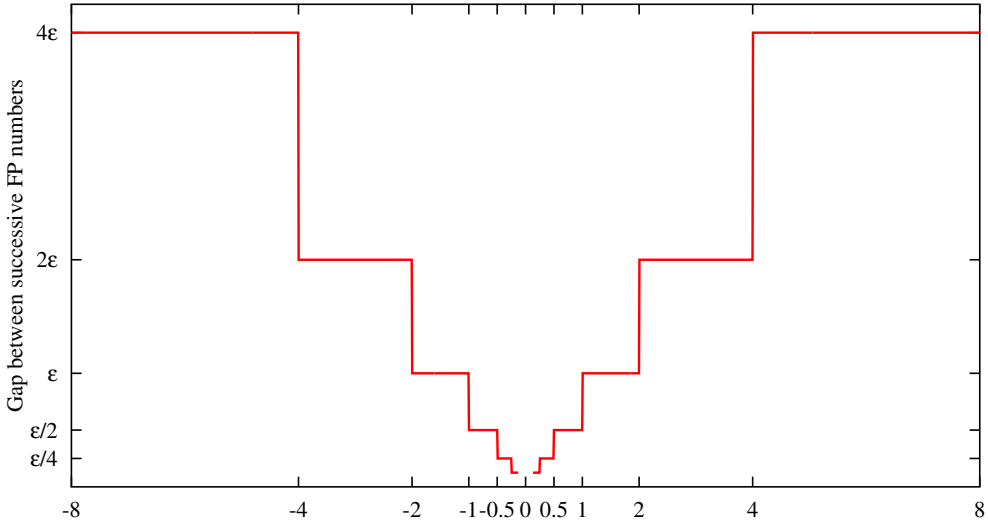
Jaká je mezera mezi 1024 a nejbližším větším IEEE single precision FP číslem?
$1024\varepsilon \approx 0.00012$

Uvažujte číslo $(0.1)_{10}$. Jaká je jeho single precision FP reprezentace?

$$|0|\overbrace{01111011}^{(123)_{10}}|10011001100110011001100|$$
$$= (1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21} + \cdots) \times 2^{-4}$$

Má technika skrytého bitu v tomto případě za důsledek přesnější reprezentaci? Ne.

Program `epsilon` [Sandu, 2001] po zadání exponentu p sečte $1 + 2^{-p}$ a od výsledku odečte 1 (pro single i double precision). Je-li $2^{-p} < \varepsilon$, výsledek bude nula. Spusťte program pro $p \in \{20, \ldots, 60\}$ a experimentálně zjistěte strojové epsilon pro jednoduchou i dvojnásobnou přesnost.



Grafické znázornění rozložení FP čísel. V každém intervalu $\langle 2^n, 2^{n+1})$ leží stejný počet FP čísel, takže mezera mezi nimi se zvětšuje a hustota zmenšuje.

Množina FP čísel: $\mathbb{F} = \{\pm 0,$ normální, subnormální, $\pm\infty\}$.

Pomocí programů `machar_single` a `machar_double` [Press et al., 1997a] diagnostikujte FP parametry vašeho systému. Položky na výstupu znamenají:

ibeta $\equiv \beta$, báze používané číselné soustavy.

it $\equiv d_\mathrm{m} + 1$ (při použití skrytého bitu), d_m bez něj; počet cifer mantisy.

irnd Vrací $0, \ldots, 5$ informující o způsobu zaokrouhlování při sčítání a jak je ošetřeno podtečení. 2, 5: váš systém zaokrouhluje podle IEEE standardu. 1, 4: váš systém provádí zaokrouhlování, ale ne podle IEEE standardu. 0, 3: nezaokrouhluje, usekává. 0, 1, 2: bez subnormálních čísel, 3, 4, 5: užívá subnormálních čísel.

ngrd Počet ‚guard digits'.

machep Nejmenší (nejzápornější) exponent $e - E$ báze β takový, že $1 + \beta^{e-E} > 1$.

negep Nejmenší (nejzápornější) exponent $e - E$ báze β takový, že $1 - \beta^{e-E} < 1$.

iexp $\equiv d_\mathrm{e}$, počet bitů exponentu.

minexp Nejmenší (nejzápornější) exponent $e - E$ báze β konzistentní s normalizovanými čísly.

maxexp Největší (kladný) exponent $e - E$ báze β.

eps $\equiv \varepsilon$, strojové epsilon.

epsneg $\equiv \varepsilon_-$, strojové epsilon ε_-.

xmin $\equiv \beta^{\mathtt{minexp}}$, nejmenší kladné normalizované FP číslo.

xmax $\equiv (1 - \varepsilon_-)\beta^{\mathtt{maxexp}}$, největší normalizované FP číslo.

Zaokrouhlování Není-li $x \in \mathbb{R}$ a zároveň $x \notin \mathbb{F}$, musíme je reprezentovat vhodným FP číslem – zaokrouhlování (*rounding*). ANSI/IEEE definuje čtyři typy zaokrouhlování: nahoru, dolů, k nule, k nejbližšímu FP číslu.

Nahoru/dolů (up or down) Označme x_- (x_+) nejbližší menší nebo rovné (větší nebo rovné) FP číslo, a definujme

$$[x] \equiv \begin{cases} x_- & \text{pro zaokrouhlení dolů} \\ x_+ & \text{pro zaokrouhlení nahoru} \end{cases} \qquad [0] \equiv 0 \tag{1.4}$$

Zaokrouhlení reprezentuje určitou zaokrouhlovací chybu (*roundoff error*). Relativní zaokrouhlovací chybu δ definujeme

$$[x] = x(1+\delta) \tag{1.5}$$

Protože $|[x] - x| \leq |x_+ - x_-|$ a zároveň $x \geq x_-$, je

$$|\delta| \leq \frac{|x_+ - x_-|}{|x_-|} = \frac{\varepsilon 2^{e-E}}{m 2^{e-E}} \leq \varepsilon \tag{1.6}$$

Nalezněte v našem hračkovém FP modelu příklad čísla takového, že při zaokrouhlení dolů $\delta \approx \varepsilon$.

Zaokrouhlování Není-li $x \in \mathbb{R}$ a zároveň $x \notin \mathbb{F}$, musíme je reprezentovat vhodným FP číslem – zaokrouhlování (*rounding*). ANSI/IEEE definuje čtyři typy zaokrouhlování: nahoru, dolů, k nule, k nejbližšímu FP číslu.

Nahoru/dolů (up or down) Označme x_- (x_+) nejbližší menší nebo rovné (větší nebo rovné) FP číslo, a definujme

$$[x] \equiv \begin{cases} x_- & \text{pro zaokrouhlení dolů} \\ x_+ & \text{pro zaokrouhlení nahoru} \end{cases} \qquad [0] \equiv 0 \tag{1.4}$$

Zaokrouhlení reprezentuje určitou zaokrouhlovací chybu (*roundoff error*). Relativní zaokrouhlovací chybu δ definujeme

$$[x] = x(1+\delta) \tag{1.5}$$

Protože $|[x] - x| \leq |x_+ - x_-|$ a zároveň $x \geq x_-$, je

$$|\delta| \leq \frac{|x_+ - x_-|}{|x_-|} = \frac{\varepsilon 2^{e-E}}{m 2^{e-E}} \leq \varepsilon \tag{1.6}$$

Nalezněte v našem hračkovém FP modelu příklad čísla takového, že při zaokrouhlení dolů $\delta \approx \varepsilon$.
$\cdots 6666\dot{}0 = x$

K nule (chopping, toward zero) Zahození (odseknutí) bitů nenulových které se nevejdou do mantisy. Zřejmě $0 \leq x_- \leq x$ pro kladná x a $x \leq x_+ \leq 0$ pro záporná x. Zaokrouhlené číslo není dále od nuly než zaokrouhlované číslo. Zaokrouhlovací chyba

$$-\varepsilon \leq \delta_{\text{chopping}} \leq 0 \tag{1.7}$$

K nejbližšímu FP číslu (to nearest) Používá většina procesorů; pod zaokrouhlováním (*rounding*) budeme dále rozumět tento způsob.

$$[x] \equiv \begin{cases} x_- & \text{je-li} \quad x_- \leq x < \frac{1}{2}(x_+ - x_-) \\ x_+ & \text{je-li} \quad x_+ \geq x > \frac{1}{2}(x_+ - x_-) \end{cases} \tag{1.8}$$

Zřejmě

$$-\tfrac{1}{2}\varepsilon \leq \delta_{\text{rounding}} \leq \tfrac{1}{2}\varepsilon \tag{1.9}$$

Chyby mají obě znaménka, takže se pravděpodobně zruší a nebudou se akumulovat jako u zaokrouhlení k nule. Je-li $x = \frac{1}{2}(x_+ + x_-)$, IEEE standard vyžaduje vybrat zaokrouhlení se sudým (nulovým) posledním bitem. Např. pro šestibitovou mantisu $x = 1.0000001$ může být zaokrouhleno na $x_- = 1.000000$ nebo na $x_+ = 1.000001$; v tomto případě x_-. To garantuje polovinu zaokrouhlení nahoru a polovinu dolů. Podrobně viz [Goldberg, 1991, Práger a Sýkorová, 2004].

FP čísla jsou přesná až na faktor $(1 \pm \varepsilon/2)$. Pro jednoduchou přesnost to znamená 1 ± 10^{-7}, tj. cca 7 desetinných míst pro čísla řádu jednotek, u čísel řádu 10^7 jsou nepřesností ‚postiženy' jednotky, u čísel menších než 1 je nepřesností ‚postižena' sedmá cifra po první nenulové číslici.

Podobně pro dvojnásobnou přesnost je přesných cca 16 desetinných míst pro čísla řádu jednotek.

Jaké jsou v IEEE single precision zaokrouhlené hodnoty těchto čísel:

$x = 4 + 2^{-20}$

FP čísla jsou přesná až na faktor $(1 \pm \varepsilon/2)$. Pro jednoduchou přesnost to znamená 1 ± 10^{-7}, tj. cca 7 desetinných míst pro čísla řádu jednotek, u čísel řádu 10^7 jsou nepřesností ‚postiženy' jednotky, u čísel menších než 1 je nepřesností ‚postižena' sedmá cifra po první nenulové číslici.

Podobně pro dvojnásobnou přesnost je přesných cca 16 desetinných míst pro čísla řádu jednotek.

Jaké jsou v IEEE single precision zaokrouhlené hodnoty těchto čísel:

$x = 4 + 2^{-20}$ $\qquad$ $x = 2^2(1 + 2^{-22})$, je FP číslo, $[x] = x$

$x = 8 + 2^{-20}$

FP čísla jsou přesná až na faktor $(1 \pm \varepsilon/2)$. Pro jednoduchou přesnost to znamená 1 ± 10^{-7}, tj. cca 7 desetinných míst pro čísla řádu jednotek, u čísel řádu 10^7 jsou nepřesností ‚postiženy' jednotky, u čísel menších než 1 je nepřesností ‚postižena' sedmá cifra po první nenulové číslici.

Podobně pro dvojnásobnou přesnost je přesných cca 16 desetinných míst pro čísla řádu jednotek.

??

Jaké jsou v IEEE single precision zaokrouhlené hodnoty těchto čísel:

$x = 4 + 2^{-20}$ $x = [x]$, je FP číslo, $x = 2^2(1 + 2^{-22})$

$x = 8 + 2^{-20}$ $x = [x]$, je FP číslo, $x = 2^3(1 + 2^{-23})$

$x = 16 + 2^{-20}$

FP čísla jsou přesná až na faktor $(1 \pm \varepsilon/2)$. Pro jednoduchou přesnost to znamená 1 ± 10^{-7}, tj. cca 7 desetinných míst pro čísla řádu jednotek, u čísel řádu 10^7 jsou nepřesností ‚postiženy' jednotky, u čísel menších než 1 je nepřesností ‚postižena' sedmá cifra po první nenulové číslici.

Podobně pro dvojnásobnou přesnost je přesných cca 16 desetinných míst pro čísla řádu jednotek.

Jaké jsou v IEEE single precision zaokrouhlené hodnoty těchto čísel:

$x = 4 + 2^{-20}$ $x = [x]$, je FP číslo, $x = 2^2(1 + 2^{-22})$

$x = 8 + 2^{-20}$ $x = [x]$, je FP číslo, $x = 2^3(1 + 2^{-23})$

$x = 16 + 2^{-20}$ $x = 2^4(1 + 2^{-24})$, není FP, $[x] = x_- = 2^4$, $x_+ = 2^4(1 + 2^{-23})$, (sudý posl. bit)

$x = 32 + 2^{-20}$

FP čísla jsou přesná až na faktor $(1 \pm \varepsilon/2)$. Pro jednoduchou přesnost to znamená 1 ± 10^{-7}, tj. cca 7 desetinných míst pro čísla řádu jednotek, u čísel řádu 10^7 jsou nepřesností ‚postiženy' jednotky, u čísel menších než 1 je nepřesností ‚postižena' sedmá cifra po první nenulové číslici.

Podobně pro dvojnásobnou přesnost je přesných cca 16 desetinných míst pro čísla řádu jednotek.

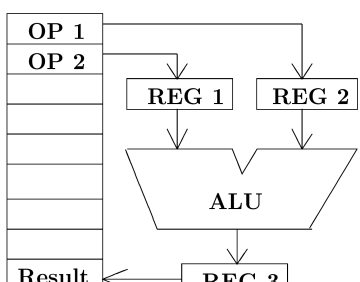
Jaké jsou v IEEE single precision zaokrouhlené hodnoty těchto čísel:

$x = 4 + 2^{-20}$	$x = [x]$, je FP číslo, $x = 2^2(1 + 2^{-22})$
$x = 8 + 2^{-20}$	$x = [x]$, je FP číslo, $x = 2^3(1 + 2^{-23})$
$x = 16 + 2^{-20}$	$x = 2^4(1 + 2^{-24})$, není FP, $[x] = x_- = 2^4$, $x_+ = 2^4(1 + 2^{-23})$, (sudý poslední bit)
$x = 32 + 2^{-20}$	$x = 2^5(1 + 2^{-25})$, není FP, x je blízké k x_-, takže $[x] = x_-$

Vyzkoušejte program `round` s čísly $1.0, 2.0, 1.1, 0.5, 0.0625, 0.6, 0.62$ a podle vlastního výběru. Vysvětlete získané výsledky.

1.3. Aritmetické operace

Hodnoty operandů jsou načteny z paměti do registrů CPU, Arithmetic and Logic Unit (ALU) provede operaci, výsledek vrátí do třetího registru, poté je hodnota uložena do paměti.



Operandy: FP čísla. Ale výsledek aritmetické operace nemusí být FP číslo! Bude zaokrouhleno $\Rightarrow$ výsledek aritmetické operace je obecně poškozen zaokrouhlovací chybou.

V našem hračkovém modelu mějme dvě FP čísla $a = 97.2 = 9.72 \times 10^1$ a $b = 6.43 = 6.43 \times 10^0$. Mantisa druhého se posune tak, aby se exponenty rovnaly.

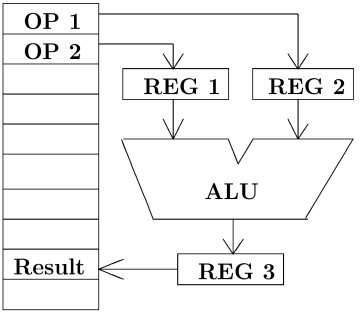
$$\begin{array}{rl} 9.72 & \times 10^1 \\ 0.643 & \times 10^1 \\ \hline 10.363 & \times 10^1 \end{array}$$

Výsledek 1.0363×10^2 není FP číslo! Bude zaokrouhlen na $1.04 \times 10^2 = 104$. Matematický výsledek se liší od numerického! Označme FP aritmetické operace $\oplus, \ominus, \otimes, \oslash$; obecně např. $a \oplus b \neq a + b$.

FP sčítání není asociativní! Pro $c = 0.999 = 9.99 \times 10^{-1}$ je $(a \oplus b) \oplus c = 1.04 \times 10^2 = 104$, ale $a \oplus (b \oplus c) = 1.05 \times 10^2 = 105$.

1.3. Aritmetické operace

Hodnoty operandů jsou načteny z paměti do registrů CPU, Arithmetic and Logic Unit (ALU) provede operaci, výsledek vrátí do třetího registru, poté je hodnota uložena do paměti.



Operandy: FP čísla. Ale výsledek aritmetické operace nemusí být FP číslo! Bude zaokrouhleno $\Rightarrow$ výsledek aritmetické operace je obecně poškozen zaokrouhlovací chybou.

V našem hračkovém modelu mějme dvě FP čísla $a = 97.2 = 9.72 \times 10^1$ a $b = 6.43 = 6.43 \times 10^0$. Mantisa druhého se posune tak, aby se exponenty rovnaly.

$$\begin{array}{rr} 9.72 & \times 10^1 \\ 0.643 & \times 10^1 \\ \hline 10.363 & \times 10^1 \end{array}$$

Výsledek 1.0363×10^2 není FP číslo! Bude zaokrouhlen na $1.04 \times 10^2 = 104$. Matematický výsledek se liší od numerického! Označme FP aritmetické operace $\oplus, \ominus, \otimes, \oslash$; obecně např. $a \oplus b \neq a + b$.

FP sčítání není asociativní! Pro $c = 0.999 = 9.99 \times 10^{-1}$ je $(a \oplus b) \oplus c = 1.04 \times 10^2 = 104$, ale $a \oplus (b \oplus c) = 1.05 \times 10^2 = 105$.

Ukažte, že FP sčítání je komutativní.

1.3.1. IEEE aritmetika

Nejprve exaktně, pak zaokrouhlit na nejbližší:

$$a \oplus b \equiv [a + b] \quad a \ominus b \equiv [a - b] \quad a \otimes b \equiv [a \times b] \quad a \oslash b \equiv [a/b] \tag{1.10}$$

Totéž platí pro druhou odmocninu, zbytek, a konverzi mezi celočíselným a FP typem. Operace prováděné tímto způsobem se nazývají exaktně zaokrouhlené (*exactly rounded*) nebo správně zaokrouhlené (*correctly rounded*).

Hardware kompatibilní s IEEE FP aritmetikou zlepšuje portabilitu programů.

Požadavek na exaktně zaokrouhlené operace se jeví jako přirozený, ale je obtížné hardwarově jej implementovat. Důvodem je, že pro implementaci korektní IEEE aritmetiky potřebujeme dodatečné hardwarové zdroje (registry dostatečné šířky).

Truncation error – na rozdíl od zaokrouhlovací chyby plně pod kontrolou programátora.

1.3.2. Speciální aritmetické operace

Znaménková nula Obě legální, ale ne zcela ekvivalentní. Je-li $x = +0$, $y = -0$ (ve smyslu FP reprezentace single, double a extended double), pak `(x==y)` vrací logickou pravdu.

Konzistence s oběma nekonečny: podle IEEE $1/(+0) = +\infty$ a $1/(-0) = -\infty$. Kdyby byla jen jedna nula a $1/0 = +\infty$, pak $1/(1/(-\infty)) = 1/0 = +\infty$.

Např. pro $\mathrm{tg}(\pi/2 - x)$ můžeme konzistentně definovat funkční hodnotu pro $x = \pm 0$ jako $\mp\infty$.

Nevýhoda: pro $x = +0$, $y = -0$ sice `(x==y)` vrací logickou pravdu, ale `(1/x==1/y)` logickou nepravdu.

1.3.2. Speciální aritmetické operace

Znaménková nula Obě legální, ale ne zcela ekvivalentní. Je-li $x = +0$, $y = -0$ (ve smyslu FP reprezentace single, double a extended double), pak `(x==y)` vrací logickou pravdu.

Konzistence s oběma nekonečny: podle IEEE $1/(+0) = +\infty$ a $1/(-0) = -\infty$. Kdyby byla jen jedna nula a $1/0 = +\infty$, pak $1/(1/(-\infty)) = 1/0 = +\infty$.

Např. pro $\mathrm{tg}(\pi/2 - x)$ můžeme konzistentně definovat funkční hodnotu pro $x = \pm 0$ jako $\mp\infty$.

Nevýhoda: pro $x = +0$, $y = -0$ sice `(x==y)` vrací logickou pravdu, ale `(1/x==1/y)` logickou nepravdu.

Porovnání

`(a<b).OR.(a==b).OR.(a>b)`	True, jsou-li $a, b \in \mathbb{F}$. False, je-li jedno `NaN`.
`+0==-0`	True
`+inf==-inf`	False

Operace s nekonečnem Nekonečna poskytují možnost pokračovat ve výpočtu, objeví-li se přetečení rozsahu normalizovaných FP čísel.

$a/\infty =$	0	a konečné FP
	`NaN`	$a = \infty$
$a \times \infty =$	$+\infty$	$a > 0$
	$-\infty$	$a < 0$
	`NaN`	$a = 0$
$\pm\infty + a =$	$\pm\infty$	a konečné FP
	$\pm\infty$	$a = \pm\infty$
	`NaN`	$a = \mp\infty$

Operace s nekonečnem Nekonečna poskytují možnost pokračovat ve výpočtu, objeví-li se přetečení rozsahu normalizovaných FP čísel.

$a/\infty =$	0	a konečné FP
	`NaN`	$a = \infty$
$a \times \infty =$	$+\infty$	$a > 0$
	$-\infty$	$a < 0$
	`NaN`	$a = 0$
$\pm\infty + a =$	$\pm\infty$	a konečné FP
	$\pm\infty$	$a = \pm\infty$
	`NaN`	$a = \mp\infty$

Operace s `NaN` Jakákoli operace, jejímž aspoň jedním operandem je `NaN`, má výsledek `NaN`. Navíc je `NaN` výsledkem: $\infty + (-\infty)$, $0 \times \infty$, $0/0$, ∞/∞, $\sqrt{-|x|}$, $x \bmod 0$, $\infty \bmod x$.

1.3.3. Výjimky, návěští, zachycování

(*exceptions, flags, exception trapping*)

IEEE definuje 5 výjimek: dělení nulou, přetečení (*overflow*), podtečení (*underflow*), neplatná operace (*invalid operation*), nepřesná operace (*inexact operation*).

Trap handlers jsou užitečné pro zpětnou kompatibilitu programů.

Dělení nulou Je-li $a \in \mathbb{F}$, IEEE vyžaduje:

$$a/0.0 = \begin{cases} +\infty & \text{je-li} \quad a > 0 \\ -\infty & \text{je-li} \quad a < 0 \\ \texttt{NaN} & \text{je-li} \quad a = 0 \end{cases}$$

1.3.3. Výjimky, návěští, zachycování

(*exceptions, flags, exception trapping*)

IEEE definuje 5 výjimek: dělení nulou, přetečení (*overflow*), podtečení (*underflow*), neplatná operace (*invalid operation*), nepřesná operace (*inexact operation*).

Trap handlers jsou užitečné pro zpětnou kompatibilitu programů.

Dělení nulou Je-li $a \in \mathbb{F}$, IEEE vyžaduje:

$$a/0.0 = \begin{cases} +\infty & \text{je-li} \quad a > 0 \\ -\infty & \text{je-li} \quad a < 0 \\ \texttt{NaN} & \text{je-li} \quad a = 0 \end{cases} \tag{1.11}$$

Přetečení Když je výsledek aritmetické operace konečný, ale větší co do velikosti než největší reprezentovatelné FP číslo. Standardní IEEE ošetření: výsledek $\pm\infty$ (round to nearest) nebo největší reprezentovatelné FP číslo (round toward 0). Některé kompilátory přetečení zachytí a ukončí vykonávání programu s chybovou hláškou.

Příklad eliminace přetečení: [Press et al., 1997a] poskytuje knihovnu `complex.c`, v němž jsou definovány funkce pro práci s komplexními čísly, mj. funkci `Cabs` pro výpočet absolutní hodnoty. Pro datový typ `fcomplex` definovaný jako
`typedef struct FCOMPLEX {float r,i;} fcomplex;`

Chybná implementace:

```
float Cabs(fcomplex z)
{
    return sqrt(z.r*z.r+z.i*z.i);
}
```

Zde mohou kvadráty přetéci! Špatně!

Příklad eliminace přetečení: [Press et al., 1997a] poskytuje knihovnu `complex.c`, v němž jsou definovány funkce pro práci s komplexními čísly, mj. funkci `Cabs` pro výpočet absolutní hodnoty. Pro datový typ `fcomplex` definovaný jako `typedef struct FCOMPLEX {float r,i;} fcomplex;`

Chybná implementace:

```
float Cabs(fcomplex z)
{
    return sqrt(z.r*z.r+z.i*z.i);
}
```

Zde mohou kvadráty přetéci! Špatně!

Tento výpočet nepřeteče! Tak to má být!

Správná implementace:

```
float Cabs(fcomplex z)
{
    float x,y,ans,temp;
    x=fabs(z.r);
    y=fabs(z.i);
    if (x == 0.0)
        ans=y;
    else if (y == 0.0)
        ans=x;
    else if (x > y) {
        temp=y/x;
        ans=x*sqrt(1.0+temp*temp);
    } else {
        temp=x/y;
        ans=y*sqrt(1.0+temp*temp);
    }
    return ans;
}
```

Podtečení Když je výsledek aritmetické operace menší než nejmenší normalizované FP číslo. Podle IEEE je výsledek subnormální číslo (tzv. postupné podtečení, *gradual underflow*) nebo 0, je-li výsledek dostatečně malý. Subnormální čísla mají menší přesnost než normalizovaná, takže vedou ke ztrátě přesnosti. To je ale lepší než bez jejich použití.

Příklad ztráty přesnosti: v našem hračkovém modelu $x = 1.99 \times 10^{-40}$, $y = 1.00 \times 10^{-11}$, $z = 1.00 \times 10^{11}$ spočtěme $(x \times y) \times z$. Matematický výsledek je roven $t_{\mathrm{exact}} = x$. Podle teorie zaokrouhlovacích chyb očekáváme

$$t_{\mathrm{FP}} = (x \otimes y) \otimes z = (1 + \delta) t_{\mathrm{exact}}, \qquad |\delta| \approx \varepsilon \tag{1.12}$$

(Pro každé FP násobení je relativní chyba $|\delta_{\otimes}| \leq \varepsilon/2$.) V hračkovém modelu je $\varepsilon = 0.01$, takže očekáváme

$$t_{\mathrm{FP}} \in \langle (1 - 2\varepsilon) t_{\mathrm{exact}}, (1 + 2\varepsilon) t_{\mathrm{exact}} \rangle = \langle 1.98 \times 10^{-40}, 2.00 \times 10^{-40} \rangle \tag{1.13}$$

Avšak součin $x \otimes y = 1.99 \times 10^{-51}$ podtéká, a má v subnormálních číslech reprezentaci 0.01×10^{-49}, což vynásobeno se z dá $t_{\mathrm{FP}} = 1.00 \times 10^{-40}$. Relativní chyba je pak $|\delta_{\mathrm{subnorm}}| = 0.99 = 99\varepsilon$.

Podtečení Když je výsledek aritmetické operace menší než nejmenší normalizované FP číslo. Podle IEEE je výsledek subnormální číslo (tzv. postupné podtečení, *gradual underflow*) nebo 0, je-li výsledek dostatečně malý. Subnormální čísla mají menší přesnost než normalizovaná, takže vedou ke ztrátě přesnosti. To je ale lepší než bez jejich použití.

Příklad ztráty přesnosti: v našem hračkovém modelu $x = 1.99 \times 10^{-40}$, $y = 1.00 \times 10^{-11}$, $z = 1.00 \times 10^{11}$ spočtěme $(x \times y) \times z$. Matematický výsledek je roven $t_{\text{exact}} = x$. Podle teorie zaokrouhlovacích chyb očekáváme

$$t_{\text{FP}} = (x \otimes y) \otimes z = (1 + \delta) t_{\text{exact}}, \qquad |\delta| \approx \varepsilon \tag{1.12}$$

(Pro každé FP násobení je relativní chyba $|\delta_{\otimes}| \leq \varepsilon/2$.) V hračkovém modelu je $\varepsilon = 0.01$, takže očekáváme

$$t_{\text{FP}} \in \langle (1 - 2\varepsilon) t_{\text{exact}}, (1 + 2\varepsilon) t_{\text{exact}} \rangle = \langle 1.98 \times 10^{-40}, 2.00 \times 10^{-40} \rangle \tag{1.13}$$

Avšak součin $x \otimes y = 1.99 \times 10^{-51}$ podtéká, a má v subnormálních číslech reprezentaci 0.01×10^{-49}, což vynásobeno se z dá $t_{\text{FP}} = 1.00 \times 10^{-40}$. Relativní chyba je pak $|\delta_{\text{subnorm}}| = 0.99 = 99\varepsilon$.

Neplatná operace Výsledkem je `NaN`.

Podtečení Když je výsledek aritmetické operace menší než nejmenší normalizované FP číslo. Podle IEEE je výsledek subnormální číslo (tzv. postupné podtečení, *gradual underflow*) nebo 0, je-li výsledek dostatečně malý. Subnormální čísla mají menší přesnost než normalizovaná, takže vedou ke ztrátě přesnosti. To je ale lepší než bez jejich použití.

Příklad ztráty přesnosti: v našem hračkovém modelu $x = 1.99 \times 10^{-40}$, $y = 1.00 \times 10^{-11}$, $z = 1.00 \times 10^{11}$ spočtěme $(x \times y) \times z$. Matematický výsledek je roven $t_{\text{exact}} = x$. Podle teorie zaokrouhlovacích chyb očekáváme

$$t_{\text{FP}} = (x \otimes y) \otimes z = (1 + \delta) t_{\text{exact}}, \qquad |\delta| \approx \varepsilon \tag{1.12}$$

(Pro každé FP násobení je relativní chyba $|\delta_{\otimes}| \leq \varepsilon/2$.) V hračkovém modelu je $\varepsilon = 0.01$, takže očekáváme

$$t_{\text{FP}} \in \langle (1 - 2\varepsilon) t_{\text{exact}}, (1 + 2\varepsilon) t_{\text{exact}} \rangle = \langle 1.98 \times 10^{-40}, 2.00 \times 10^{-40} \rangle \tag{1.13}$$

Avšak součin $x \otimes y = 1.99 \times 10^{-51}$ podtéká, a má v subnormálních číslech reprezentaci 0.01×10^{-49}, což vynásobeno se z dá $t_{\text{FP}} = 1.00 \times 10^{-40}$. Relativní chyba je pak $|\delta_{\text{subnorm}}| = 0.99 = 99\varepsilon$.

Neplatná operace Výsledkem je `NaN`.
Nepřesná operace Výsledkem je hodnota zaokrouhlená podle IEEE.

Vyzkoušejte program `messy-c`.

1.3.4. Systémové aspekty

Design počítačových systémů vyžaduje hluboké znalosti FP čísel. Moderní procesory mají speciální FP instrukce, kompilátory musí generovat takové FP instrukce, a operační systémy musí ošetřit výjimky generované FP instrukcemi.

Zde uvedeme pouze vybrané aspekty, podrobná diskuse je uvedena např. v [Goldberg, 1991, Sandu, 2001].

1.3.4. Systémové aspekty

Design počítačových systémů vyžaduje hluboké znalosti FP čísel. Moderní procesory mají speciální FP instrukce, kompilátory musí generovat takové FP instrukce, a operační systémy musí ošetřit výjimky generované FP instrukcemi.

Zde uvedeme pouze vybrané aspekty, podrobná diskuse je uvedena např. v [Goldberg, 1991, Sandu, 2001].

Optimalizace Následující kódy jsou matematicky ekvivalentní:

eps1single

```
#include <stdio.h>
int main(void)
{
    float eps=1.0;
    do {
        eps /= 2.0;
    } while (1.0+eps>1.0);
    printf("\n%.12e\n",eps);
    return 0;
}
```

eps2single

```
#include <stdio.h>
int main(void)
{
    float eps=1.0;
    do {
        eps /= 2.0;
    } while (eps>0.0);
    printf("\n%.12e\n",eps);
    return 0;
}
```

eps3single

```
#include <stdio.h>
int main(void)
{
    float eps=1.0,x;
    do {
        eps /= 2.0; x=1.0+eps;
    } while (x>1.0);
    printf("\n%.12e\n",eps);
    return 0;
}
```

Spouštějte demonstrační programy `epsXsingle`, `X = 1,2,3` a optimalizovanou verzi `eps3single-optcpu`, a pokuste se vysvětlit výsledky. (Double varianty `epsXdouble` a `eps3double-optcpu` se liší jen záměnou `float` za `double`.)

1.3.5. Dlouhé sumace

Problém s dlouhými sumacemi: každá individuální sumace vnáší do částečného součtu určitou chybu, takže celková suma může být značně nepřesná.

Položíme-li v sumě $\sum_{j=1}^{n} x_j$: $s_1 = (1+\delta_1)x_1$, $s_2 = (1+\delta_2)(s_1+x_2) = (1+\delta_2)[(1+\delta_1)x_1 + x_2]$, $s_3 = (1+\delta_3)(s_2+x_3) = (1+\delta_3)\{(1+\delta_2)[(1+\delta_1)x_1 + x_2] + x_3\} = (1+\delta_1+\delta_2+\delta_3)x_1 + (1+\delta_2+\delta_3)x_2 + (1+\delta_3)x_3 + \mathcal{O}(\varepsilon^2)$, atd.

- Výpočet ve vyšší přesnosti (např. žádáme-li s_n v single, počítáme v double). Problém nastává, žádáme-li výsledek v double.

1.3.5. Dlouhé sumace

Problém s dlouhými sumacemi: každá individuální sumace vnáší do částečného součtu určitou chybu, takže celková suma může být značně nepřesná.

Položíme-li v sumě $\sum_{j=1}^{n} x_j$: $s_1 = (1+\delta_1)x_1$, $s_2 = (1+\delta_2)(s_1+x_2) = (1+\delta_2)[(1+\delta_1)x_1 + x_2]$, $s_3 = (1+\delta_3)(s_2+x_3) = (1+\delta_3)\{(1+\delta_2)[(1+\delta_1)x_1 + x_2] + x_3\} = (1+\delta_1+\delta_2+\delta_3)x_1 + (1+\delta_2+\delta_3)x_2 + (1+\delta_3)x_3 + \mathcal{O}(\varepsilon^2)$, atd.

- Výpočet ve vyšší přesnosti (např. žádáme-li s_n v single, počítáme v double). Problém nastává, žádáme-li výsledek v double.
- Vhodně seřadit členy – zřejmě je výhodnější seřadit sčítance vzestupně.

1.3.5. Dlouhé sumace

Problém s dlouhými sumacemi: každá individuální sumace vnáší do částečného součtu určitou chybu, takže celková suma může být značně nepřesná.

Položíme-li v sumě $\sum_{j=1}^{n} x_j$: $s_1 = (1+\delta_1)x_1$, $s_2 = (1+\delta_2)(s_1+x_2) = (1+\delta_2)[(1+\delta_1)x_1 + x_2]$, $s_3 = (1+\delta_3)(s_2+x_3) = (1+\delta_3)\{(1+\delta_2)[(1+\delta_1)x_1 + x_2] + x_3\} = (1+\delta_1+\delta_2+\delta_3)x_1 + (1+\delta_2+\delta_3)x_2 + (1+\delta_3)x_3 + \mathcal{O}(\varepsilon^2)$, atd.

- Výpočet ve vyšší přesnosti (např. žádáme-li s_n v single, počítáme v double). Problém nastává, žádáme-li výsledek v double.
- Vhodně seřadit členy – zřejmě je výhodnější seřadit sčítance vzestupně.
- Použít Kahanovu sumační formuli [Goldberg, 1991]:

```
sum=x(1)
c=0.0
DO j=2,n
  y=x(j)-c
  t=sum+y
  c=(t-sum)-y
  sum=t
END DO
```

Pak vypočítaná suma

$$s_n = \sum_{j=1}^{n} x_j(1+\delta_j) + \mathcal{O}(n\varepsilon^2)\sum_{j=1}^{n}|x_j| \qquad (1.14)$$

kde $|\delta_j| \leq 2\varepsilon$. Důkaz viz [Goldberg, 1991].

Uvažujme sumu ($b \in \mathbb{N}$)

$$s = 1 + \underbrace{\frac{1}{b} + \ldots + \frac{1}{b}}_{b \text{ členů}} + \underbrace{\frac{1}{b^2} + \ldots + \frac{1}{b^2}}_{b^2 \text{ členů}} + \ldots + \underbrace{\frac{1}{b^{p_{\max}}} + \ldots + \frac{1}{b^{p_{\max}}}}_{b^{p_{\max}} \text{ členů}} \tag{1.15}$$

Každý ze členů ve svorkách je roven jedné, exaktní matematická suma je proto $p_{\max} + 1$. Počet sčítanců v sumě je

$$1 + b + b^2 + \ldots + b^{p_{\max}} = \frac{b^{p_{\max}} - 1}{b - 1} \tag{1.16}$$

Vyzkoušejte program `longsum`, který vykonává sumaci (1.15) pro $b = 10$, $p_{\max} = 7$ třemi způsoby: v sestupném pořadí, ve vzestupném pořadí a s použitím Kahanovy sumační formule. Jak se chovají optimalizované verze `longsum-O1` a `longsum-O2`?

Uvažujme sumu ($b \in \mathbb{N}$)

$$s = 1 + \underbrace{\frac{1}{b} + \ldots + \frac{1}{b}}_{b \text{ členů}} + \underbrace{\frac{1}{b^2} + \ldots + \frac{1}{b^2}}_{b^2 \text{ členů}} + \ldots + \underbrace{\frac{1}{b^{p_{\max}}} + \ldots + \frac{1}{b^{p_{\max}}}}_{b^{p_{\max}} \text{ členů}} \tag{1.15}$$

Každý ze členů ve svorkách je roven jedné, exaktní matematická suma je proto $p_{\max} + 1$. Počet sčítanců v sumě je

$$1 + b + b^2 + \ldots + b^{p_{\max}} = \frac{b^{p_{\max}} - 1}{b - 1} \tag{1.16}$$

Vyzkoušejte program `longsum`, který vykonává sumaci (1.15) pro $b = 10$, $p_{\max} = 7$ třemi způsoby: v sestupném pořadí, ve vzestupném pořadí a s použitím Kahanovy sumační formule. Jak se chovají optimalizované verze `longsum-O1` a `longsum-O2`?

Typický výstup v single precision vypadá (GNU/Linux 2.4.20, GCC 3.3 20030226):

```
basis = 10,  maximum_power = 7
number_of_terms_in_the_sum = 11111111
exact:      8.00000000,  rel_error =   0.00000000 %
sort down: 6.95631695,  rel_error = -13.04603815 %
sort up:    8.01876831,  rel_error =   0.23460388 %
Kahan:      8.00000000,  rel_error =   0.00000000 %
```

U optimalizované verze je poslední řádek
`Kahan: 6.95631695, rel_error = -13.0460377 %` Proč?

1.3.6. Patologie, nástrahy a pasti

Konverze mezi binárním a dekadickým formátem

Vyzkoušejte program `storeprt`, jehož první část pouze uloží číslo $x = 1.0 \times 10^{-4}$ a poté jej vytiskne na terminál. Vysvětlete pozorovanou ‚anomálii‘.

1.3.6. Patologie, nástrahy a pasti

Konverze mezi binárním a dekadickým formátem

Vyzkoušejte program `storeprt`, jehož první část pouze uloží číslo $x = 1.0 \times 10^{-4}$ a poté jej vytiskne na terminál. Vysvětlete pozorovanou ‚anomálii‘.

Porovnávání FP čísel

Vyzkoušejte opět program `storeprt`, jehož druhá část se větví podle výsledku srovnání čísel $1.0 \times 10^8 \times x^2$ a 1.0. Vysvětlete, proč se skutečnost liší od očekávání.

Vyzkoušejte upravený program `storeprt2`, jenž má upravenou podmínku větvení. Proč tento program pracuje v souladu s očekáváním?

1.3.6. Patologie, nástrahy a pasti

Konverze mezi binárním a dekadickým formátem

Vyzkoušejte program `storeprt`, jehož první část pouze uloží číslo $x = 1.0 \times 10^{-4}$ a poté jej vytiskne na terminál. Vysvětlete pozorovanou ‚anomálii‘.

Porovnávání FP čísel

Vyzkoušejte opět program `storeprt`, jehož druhá část se větví podle výsledku srovnání čísel $1.0 \times 10^8 \times x^2$ a 1.0. Vysvětlete, proč se skutečnost liší od očekávání.

Vyzkoušejte upravený program `storeprt2`, jenž má upravenou podmínku větvení. Proč tento program pracuje v souladu s očekáváním?

Vyzkoušejte programy `quiz_cor` a `quiz_inc`. Proč první z nich vybere ‚správnou‘ a druhý ‚nesprávnou‘ větev?

Zvláštní konverze Někdy je nepřesnost v FP přenesena prostřednictvím konverzí do celočíselných typů.

Vyzkoušejte program `convers` a jeho optimalizovanou verzi `convers-optcpu`. Vysvětlete výsledky.

Zvláštní konverze Někdy je nepřesnost v FP přenesena prostřednictvím konverzí do celočíselných typů.

Vyzkoušejte program `convers` a jeho optimalizovanou verzi `convers-optcpu`. Vysvětlete výsledky.

Jiný problém nastává při konverzi single do double precision. Taková konverze nezlepšuje přesnost – bity registru jsou ‚vycpány‘ nulami a číslo zůstává jinak stejné jako v single.

Ověřte si uvedené tvrzení pomocí programu `sing2dbl`.

Zvláštní konverze Někdy je nepřesnost v FP přenesena prostřednictvím konverzí do celočíselných typů.

Vyzkoušejte program `convers` a jeho optimalizovanou verzi `convers-optcpu`. Vysvětlete výsledky.

Jiný problém nastává při konverzi single do double precision. Taková konverze nezlepšuje přesnost – bity registru jsou ‚vycpány' nulami a číslo zůstává jinak stejné jako v single.

Ověřte si uvedené tvrzení pomocí programu `sing2dbl`.

Operandy v paměti a v registrech Je-li tentýž výpočet prováděn s operandy v registru (FP registry CPU Pentium mají rozšířenou dvojnásobnou přesnost, 80 bitů) a s operandy v paměti, může se výsledek lišit.

Tento jev demonstrujte pomocí programu `mem_reg-O0` a jeho optimalizované verzi (stupeň `-O2`) `mem_reg-O2`.

Neplatné číslice Odečteme-li v single precision $100000.1 - 100000.0$, očekáváme výsledek 0.1.

Jaký výsledek poskytne program `insignid`? Jak jej vysvětlíte?

Neplatné číslice Odečteme-li v single precision $100000.1 - 100000.0$, očekáváme výsledek 0.1.

Jaký výsledek poskytne program `insignid`? Jak jej vysvětlíte?

Single precision zaručuje 7 decimálních číslic, a výše uvedené odečítání zruší nejvýznamnějších 6; výsledek obsahuje jen jednu. Připojené ‚smetí‘ jsou nesignifikantní číslice, které pocházejí z nepřesné binární reprezentace operandů.

Neplatné číslice Odečteme-li v single precision $100000.1 - 100000.0$, očekáváme výsledek 0.1.

Jaký výsledek poskytne program `insignid`? Jak jej vysvětlíte?

Single precision zaručuje 7 decimálních číslic, a výše uvedené odečítání zruší nejvýznamnějších 6; výsledek obsahuje jen jednu. Připojené ‚smetí' jsou nesignifikantní číslice, které pocházejí z nepřesné binární reprezentace operandů.

Na pořadí operací záleží! Matematicky ekvivalentní výrazy mohou dávat odlišné výsledky podle toho, v jakém pořadí se provádí vyčíslování v FP.

Demonstrujte tuto skutečnost pomocí programu `ordofop`, případně jeho optimalizované verze `ordofop-optcpu`.

Proč dává optimalizovaná verze jiné výsledky než neoptimalizovaná?

Katastrofické vyrušení (*catastrophic cancellation*, *loss-of-significance errors*). Odečítáme-li velmi blízká čísla, nejvýznamnější číslice operandů se rovnají a vzájemně se zruší. Pokud jsou operandy poškozeny zaokrouhlovací chybou, může dojít k úplnému zamaskování výsledku.

Řešíme-li kvadratickou rovnici

$$ax^2 + bx + c = 0 \tag{1.17}$$

s takovými koeficienty, že $4ac \ll b^2$, bude jeden z kořenů

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.18}$$

podléhat katastrofickému vyrušení. Pro $b > 0$ to bude kořen x_1 odpovídající $+$. V takovém případě rozšíříme rovnici (1.18) výrazem $-b - \sqrt{b^2 - 4ac}$:

$$x_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} \tag{1.19}$$

Vyzkoušejte program `cancerr`, který řeší kvadratickou rovnici s koeficienty $a = 1.0$, $b = c = 1.0 \times 10^8$. Podmínka $4ac \ll b^2$ je splněna a kořen (1.18) odpovídající znaménku $+$ trpí katastrofickým vyrušením. Použitím (1.19) dostaneme podstatně přesnější výsledek.

1.3.7. Stabilita

Někdy je zaokrouhlovací chyba ‚vmixována' do výpočtu v jeho rané fázi a zvětšuje se tak dlouho, až zcela překryje správný výsledek.

Příklad [Press et al., 1997a]: Kladným řešením kvadratické rovnice

$$x^2 = 1 - x \tag{1.20}$$

je ‚Zlatý řez' (*Golden Mean*) $\varphi = \frac{1}{2}(\sqrt{5} - 1) \approx 0.61803398875$. Celočíselné mocniny φ^n lze počítat pomocí $\varphi^{n+1} = \varphi\varphi^n$, $n = 0, 1, \ldots$, nebo z rekurentní formule

$$\varphi^{n+1} = \varphi^{n-1} - \varphi^n, \quad \varphi^0 = 1, \quad \varphi^1 = 0.61803398875, \quad n = 1, 2, \ldots \tag{1.21}$$

jen pomocí odčítání. (Důkaz (1.21): $\varphi^{n+1} - \varphi^{n-1} = \varphi^{n-1}(\varphi^2 - 1) = -\varphi^n$, protože podle (1.20) je výraz v závorce roven $-\varphi$.)

1.3.7. Stabilita

Někdy je zaokrouhlovací chyba ‚vmixována' do výpočtu v jeho rané fázi a zvětšuje se tak dlouho, až zcela překryje správný výsledek.

Příklad [Press et al., 1997a]: Kladným řešením kvadratické rovnice

$$x^2 = 1 - x \tag{1.20}$$

je ‚Zlatý řez' (*Golden Mean*) $\varphi = \frac{1}{2}(\sqrt{5} - 1) \approx 0.61803398875$. Celočíselné mocniny φ^n lze počítat pomocí $\varphi^{n+1} = \varphi\varphi^n$, $n = 0, 1, \ldots$, nebo z rekurentní formule

$$\varphi^{n+1} = \varphi^{n-1} - \varphi^n, \quad \varphi^0 = 1, \quad \varphi^1 = 0.61803398875, \quad n = 1, 2, \ldots \tag{1.21}$$

jen pomocí odčítání. (Důkaz (1.21): $\varphi^{n+1} - \varphi^{n-1} = \varphi^{n-1}(\varphi^2 - 1) = -\varphi^n$, protože podle (1.20) je výraz v závorce roven $-\varphi$.)

Rovnice (1.20) má ještě jedno řešení: $\psi = -\frac{1}{2}(\sqrt{5} + 1) \approx -1.61803398875$. Toto řešení splňuje stejnou rekurzivní relaci (1.21). Jelikož (1.21) je lineární, a $|\psi| > 1$, malá příměs zavlečená zaokrouhlovací chybou bude exponenciálně růst.

Tento příklad nestability je implementován v programech `unstable_single` a `unstable_double` (varianty pro jednoduchou a dvojnásobnou přesnost).

1.4. Havárie způsobené chybným použitím FP čísel

- Kolaps řízení obranného protiraketového systému Patriot ve válce v Zálivu r. 1991. Systém fungoval efektivně, ale v jednom případě selhal: Identifikace střel Scud probíhala tak, že radarový systém po zachycení podezřelého objektu předpověděl jeho budoucí polohu po uplynutí nějakého časového intervalu, a jestliže v tom místě v příslušnou dobu objekt zjistil, dal signál k jeho zničení. Právě v předpovědi nové polohy docházelo k nepřesnostem, a to tím větším, čím déle systém pracoval. Po 20 hodinách byla chyba v určení polohy zhruba 137 metrů, po 100 hodinách 687 metrů. Příčina: vnitřní hodiny počítače uchovávaly čas v celých násobcích 0.1 sec a program je převáděl na FP číslo v sekundách. Při tom se dekadické číslo 0.1, které má v binární soustavě nekonečný rozvoj $0.0001100110011\ldots$, zaokrouhlovalo.
- Převod FP čísla na celé číslo způsobil destrukci rakety Ariane 5 v ceně $\sim$ 1 G$ v červnu 1996. Příčina: třicet sedm vteřin po startu se program pokusil konvertovat horizontální komponentu rychlosti rakety z double precision na krátké (16bitové) celé číslo, které přeteklo, počítačový program ohlásil neplatnou operaci a řídicí systém provedl samodestrukci rakety.

Více viz [Práger a Sýkorová, 2004] a
http://www.fas.org/starwars/gao/im92026.htm
http://www.siam.org/siamnews/general/patriot.htm
http://www.ima.umn.edu/~arnold/disasters/

Shrnutí kapitoly: První kapitola měla úvodní charakter a seznámila vás s principy využívaných ve výpočetní technice při reprezentaci a manipulaci s čísly. Naučili jste se rozumět problémům celočíselné aritmetiky a hlavně aritmetiky reálných čísel. Seznámili jste se s důležitými pojmy jako jsou FP čísla, normalizovaná a subnormální čísla; norma ANSI/IEEE 754-1985; jednoduchá a dvojnásobná přesnost, strojové epsilon, zaokrouhlování výjimky a stabilita výpočtu

Další zdroje:

- Výklad nevyžadující zvláštní předběžné znalosti je uveden např. v [Práger a Sýkorová, 2004].
- Úvod do Fortranu 95 a numerických metod [Sandu, 2001]. Obsahuje kapitolu o reprezentaci čísel a počítačové aritmetice. Dostupné na http://www.cs.mtu.edu/~asandu/Courses/CS2911/fortran_notes/main.html
- Podrobnou analýzu s mnoha odkazy na původní prameny podává např. článek [Goldberg, 1991].
- Sekce 1.3 (1.2) v [Press et al., 1997a]. Pro jazyk C dostupné na http://www.library.cornell.edu/nr/bookcpdf/c1-3.pdf, pro Fortran na http://www.library.cornell.edu/nr/bookfpdf/c1-2.pdf.

2. Řešení lineárních algebraických rovnic

Rychlý náhled kapitoly: V této kapitole se budeme věnovat jedné ze základních úloh numerické matematiky – řešení soustav lineárních algebraických rovnic (SLAR). Důležitost této úlohy tkví v tom, že mnoho jiných úloh může být převedeno na řešení SLAR. Ukážeme si základní algoritmy řešení a naučíme se včas detekovat možné problémy.

Cíle kapitoly:

- Naučit se řešit SLAR metodou Gaussovy–Jordanovy eliminace.
- Naučit se řešit SLAR metodou Gaussovy eliminace se zpětnou substitucí.
- Naučit se řešit SLAR metodou metodou trojúhelníkové faktorizace.
- Naučit se řešit SLAR s některými speciálními maticemi: tridiagonální a pásově diagonální maticí.
- Naučit se nepřímou metodu iteračního zpřesnění.

Klíčová slova kapitoly: Matice, regulární, singulární; degenerace, řádková, sloupcová; metoda přímá, iterativní; Gaussova–Jordanova eliminace; pivotace; Gaussova eliminace se zpětnou substitucí; trojúhelníková faktorizace, LU dekompozice; Croutův algoritmus; inverze matice, determinant matice; tridiagonální soustava, pásově diagonální soustava; iterační zpřesnění

2.1. Základní definice

Soustava lineárních algebraických rovnic:

$$\mathsf{A} \cdot \mathsf{x} = \mathsf{b} \tag{2.1}$$

$$\mathsf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{pmatrix}, \mathsf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}, \mathsf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix} \tag{2.2}$$

Na N neznámých $x_1, \ldots, x_N$ je naloženo M rovnic. Koeficienty a_{ij} a pravá strana b_i jsou známá čísla. Uložení vektorů a matic v paměti – viz dodatek 7.

Je-li $N = M$ (čtvercová matice), je soustava (2.1) jednoznačně analyticky řešitelná, pokud je matice regulární ($\det \mathsf{A} \neq 0$). Soustava s $\det \mathsf{A} = 0$ se nazývá singulární (degenerovaná). Řádková degenerace: jedna nebo více rovnic je lineární kombinací ostatních. Sloupcová degenerace: všechny rovnice obsahují určité neznámé v téže lineární kombinaci.

Numericky mohou nastat při řešení čtvercové soustavy patologie:

Algoritmické selhání Soustava je velmi blízko (ale ne přesně) singulárnosti, takže díky zaokrouhlovací chybě je od určitého stadia řešení považována za singulární.

Nealgoritmické selhání Akumulovaná zaokrouhlovací chyba zcela přebije správné řešení, zvláště je-li N velké. Lze odhalit zpětnou substitucí.

Numericky mohou nastat při řešení čtvercové soustavy patologie:

Algoritmické selhání Soustava je velmi blízko (ale ne přesně) singulárnosti, takže díky zaokrouhlovací chybě je od určitého stadia řešení považována za singulární.

Nealgoritmické selhání Akumulovaná zaokrouhlovací chyba zcela přebije správné řešení, zvláště je-li N velké. Lze odhalit zpětnou substitucí.

Linear equation-solving packages obsahují sofistikované metody pro detekci a/nebo korekci těchto dvou patologií.

- Soustavy s $N \sim 20$–50 lze rutinně řešit v single precision bez použití sofistikovaných metod, pokud nejsou blízko singulární. Pro double precision pak N je řádu několik stovek (limituje strojový čas).
- Soustavu s N řádu 10^3 a více lze řešit, je-li jejich matice řídká (*sparse*).
- Je-li soustava blízká k singulární, je potřeba sofistikovaných metod dokonce při $N \sim 10$.

Numericky mohou nastat při řešení čtvercové soustavy patologie:

Algoritmické selhání Soustava je velmi blízko (ale ne přesně) singulárnosti, takže díky zaokrouhlovací chybě je od určitého stadia řešení považována za singulární.

Nealgoritmické selhání Akumulovaná zaokrouhlovací chyba zcela přebije správné řešení, zvláště je-li N velké. Lze odhalit zpětnou substitucí.

Linear equation-solving packages obsahují sofistikované metody pro detekci a/nebo korekci těchto dvou patologií.

- Soustavy s $N \sim 20$–50 lze rutinně řešit v single precision bez použití sofistikovaných metod, pokud nejsou blízko singulární. Pro double precision pak N je řádu několik stovek (limituje strojový čas).
- Soustavu s N řádu 10^3 a více lze řešit, je-li jejich matice řídká (*sparse*).
- Je-li soustava blízká k singulární, je potřeba sofistikovaných metod dokonce při $N \sim 10$.

Metody mohou být

- přímé – počet operací je předem exaktně definován
- iterativní – počet operací závisí na přesnosti a rychlosti konvergence

2.1.1. Úkoly numerické lineární algebry

Pro čtvercové matice $N \times N$:

- Řešení maticové rovnice $\mathsf{A} \cdot \mathsf{x} = \mathsf{b}$.
- Současné řešení více maticových rovnic $\mathsf{A} \cdot \mathsf{x}_j = \mathsf{b}_j$, $j = 1, 2, \ldots$.
- Výpočet inverzní matice A^{-1}. Tento úkol je ekvivalentní předchozímu úkolu s $j = 1, 2, \ldots, N$ a $(\mathsf{b}_j)_k = \delta_{jk}$.
- Výpočet determinantu $\det \mathsf{A}$.

2.1.1. Úkoly numerické lineární algebry

Pro čtvercové matice $N \times N$:

- Řešení maticové rovnice $\mathsf{A} \cdot \mathsf{x} = \mathsf{b}$.
- Současné řešení více maticových rovnic $\mathsf{A} \cdot \mathsf{x}_j = \mathsf{b}_j$, $j = 1, 2, \ldots$.
- Výpočet inverzní matice A^{-1}. Tento úkol je ekvivalentní předchozímu úkolu s $j = 1, 2, \ldots, N$ a $(\mathsf{b}_j)_k = \delta_{jk}$.
- Výpočet determinantu $\det \mathsf{A}$.

Pro $M < N$ nebo degenerovanou soustavu s $M = N$: buď žádné řešení, nebo více než jedno řešení – součet partikulárního řešení x_p a řešení x_h z nulového podprostoru (*nullspace*) matice A o dimenzi $N - M$ (též homogenního řešení).

- Tento úkol řeší singular value decomposition.

2.1.1. Úkoly numerické lineární algebry

Pro čtvercové matice $N \times N$:

- Řešení maticové rovnice $\mathsf{A} \cdot \mathsf{x} = \mathsf{b}$.
- Současné řešení více maticových rovnic $\mathsf{A} \cdot \mathsf{x}_j = \mathsf{b}_j$, $j = 1, 2, \ldots$.
- Výpočet inverzní matice A^{-1}. Tento úkol je ekvivalentní předchozímu úkolu s $j = 1, 2, \ldots, N$ a $(\mathsf{b}_j)_k = \delta_{jk}$.
- Výpočet determinantu $\det \mathsf{A}$.

Pro $M < N$ nebo degenerovanou soustavu s $M = N$: buď žádné řešení, nebo více než jedno řešení – součet partikulárního řešení x_p a řešení x_h z nulového podprostoru (*nullspace*) matice A o dimenzi $N - M$ (též homogenního řešení).

- Tento úkol řeší singular value decomposition.

Pro $M > N$: soustava je předeterminována (*overdetermined*). Často se hledá ‚kompromisní‘ řešení blízké vyhovění všem rovnicím současně: suma čtverců rozdílů mezi levou a pravou stranou (2.1) se minimalizuje.

- Toto je lineární problém nejmenších čtverců.

2.1.1. Úkoly numerické lineární algebry

Pro čtvercové matice $N \times N$:

- Řešení maticové rovnice $\mathsf{A} \cdot \mathsf{x} = \mathsf{b}$.
- Současné řešení více maticových rovnic $\mathsf{A} \cdot \mathsf{x}_j = \mathsf{b}_j$, $j = 1, 2, \ldots$.
- Výpočet inverzní matice A^{-1}. Tento úkol je ekvivalentní předchozímu úkolu s $j = 1, 2, \ldots, N$ a $(\mathsf{b}_j)_k = \delta_{jk}$.
- Výpočet determinantu $\det \mathsf{A}$.

Pro $M < N$ nebo degenerovanou soustavu s $M = N$: buď žádné řešení, nebo více než jedno řešení – součet partikulárního řešení x_p a řešení x_h z nulového podprostoru (*nullspace*) matice A o dimenzi $N - M$ (též homogenního řešení).

- Tento úkol řeší singular value decomposition.

Pro $M > N$: soustava je předeterminována (*overdetermined*). Často se hledá ‚kompromisní‘ řešení blízké vyhovění všem rovnicím současně: suma čtverců rozdílů mezi levou a pravou stranou (2.1) se minimalizuje.

- Toto je lineární problém nejmenších čtverců.

Standardní knihovny rutin – viz Dodatek 8.

2.2. Cramerovo pravidlo (CR)

CR je matematicky dobře definováno. Pro $\det \mathsf{A} \neq 0$ je

$$x_j = \frac{\sum_{i=1}^{N} (-1)^{i+j} M_{ij} b_i}{\det \mathsf{A}} \tag{2.3}$$

kde M_{ij} je minor (z A vyškrtnut i-tý řádek a j-tý sloupec). Ale! Počet operací CR $\mathcal{O}((N+1)!)$. Proto je CR nepoužitelné pro větší N.

2.2. Cramerovo pravidlo (CR)

CR je matematicky dobře definováno. Pro $\det \mathsf{A} \neq 0$ je

$$x_j = \frac{\sum_{i=1}^{N} (-1)^{i+j} M_{ij} b_i}{\det \mathsf{A}} \tag{2.3}$$

kde M_{ij} je minor (z A vyškrtnut i-tý řádek a j-tý sloupec). Ale! Počet operací CR $\mathcal{O}((N+1)!)$. Proto je CR nepoužitelné pro větší N.

2.3. Gaussova–Jordanova eliminace (GJE)

Řešíme současně soustavy se stejnou maticí

$$\mathsf{A} \cdot \mathsf{x}_1 = \mathsf{b}_1, \quad \mathsf{A} \cdot \mathsf{x}_2 = \mathsf{b}_2, \quad \ldots \quad , \mathsf{A} \cdot \mathsf{x}_M = \mathsf{b}_M, \quad \mathsf{A} \cdot \mathsf{Y} = 1 \tag{2.4}$$

kde 1 je jednotková matice $N \times N$. Ekvivalentní zápis:

$$\mathsf{A} \left[\mathsf{x}_1 \sqcup \mathsf{x}_2 \sqcup \cdots \sqcup \mathsf{x}_M \sqcup \mathsf{Y} \right] = \left[\mathsf{b}_1 \sqcup \mathsf{b}_2 \sqcup \cdots \sqcup \mathsf{b}_M \sqcup 1 \right] \tag{2.5}$$

kde $\sqcup$ značí sloučení sloupců (a odstranění přilehlých závorek).

Dále pro jednoduchost položíme ve výkladu $N = 4$ a $M = 2$; numerické rutiny jsou obecné.

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{c} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \end{array}\right] = \left[\begin{array}{c} b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\left[\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \cdot \left[\begin{array}{c|c} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{array}\right] = \left[\begin{array}{c|c} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{c|c|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{c|c|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{c|c|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{c|c|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Podobně zůstane řešení nezměněno, nahradíme-li libovolný řádek A a celé pravé strany netriviální lineární kombinací sebe sama a ostatních řádků.

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\left[\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] . \left[\begin{array}{c|c|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{c|c|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Podobně zůstane řešení nezměněno, nahradíme-li libovolný řádek A a celé pravé strany netriviální lineární kombinací sebe sama a ostatních řádků.
3. Výměna libovolných dvou sloupců matice A a současná výměna odpovídajících řádků řešení x_j a inverze Y (pravá strana zůstává beze změny). Toto míchá pořadím řádků v řešení. Na konci je třeba obnovit správné pořadí.

$$\left[\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] . \left[\begin{array}{c|} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \end{array}\right| \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right] = \left[\begin{array}{c|} b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \end{array}\right| \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{cc|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{cc|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Podobně zůstane řešení nezměněno, nahradíme-li libovolný řádek A a celé pravé strany netriviální lineární kombinací sebe sama a ostatních řádků.
3. Výměna libovolných dvou sloupců matice A a současná výměna odpovídajících řádků řešení x_j a inverze Y (pravá strana zůstává beze změny). Toto míchá pořadím řádků v řešení. Na konci je třeba obnovit správné pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{cc|} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{array}\right] = \left[\begin{array}{cc|} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{c|c|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{c|c|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Podobně zůstane řešení nezměněno, nahradíme-li libovolný řádek A a celé pravé strany netriviální lineární kombinací sebe sama a ostatních řádků.
3. Výměna libovolných dvou sloupců matice A a současná výměna odpovídajících řádků řešení x_j a inverze Y (pravá strana zůstává beze změny). Toto míchá pořadím řádků v řešení. Na konci je třeba obnovit správné pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{c|c|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{c|c|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

1. Výměna libovolných dvou řádků A a odpovídajících řádků všech b_j a jednotkové matice 1 (tj. pravé strany) nemění řešení x_j a inverzní matici Y. Jde o přepsání rovnic v jiném pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{cc|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{cc|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Podobně zůstane řešení nezměněno, nahradíme-li libovolný řádek A a celé pravé strany netriviální lineární kombinací sebe sama a ostatních řádků.
3. Výměna libovolných dvou sloupců matice A a současná výměna odpovídajících řádků řešení x_j a inverze Y (pravá strana zůstává beze změny). Toto míchá pořadím řádků v řešení. Na konci je třeba obnovit správné pořadí.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{array}{cc|cccc} x_{11} & x_{12} & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & x_{22} & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & x_{32} & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & x_{42} & y_{41} & y_{42} & y_{43} & y_{44} \end{array}\right] = \left[\begin{array}{cc|cccc} b_{11} & b_{12} & 1 & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 1 & 0 & 0 \\ b_{31} & b_{32} & 0 & 0 & 1 & 0 \\ b_{41} & b_{42} & 0 & 0 & 0 & 1 \end{array}\right]$$

GJE využívá jedné nebo více těchto operací k redukci matice A na jednotkovou matici $\Rightarrow$ na pravé straně se objeví řešení a inverzní matice.

2.3.1. Algoritmus GJE

Pouze pomocí operace 2:

- Dělit první řádek a_{11}, je-li $a_{11} \neq 0$ (tj. lineární kombinace).
- Odečíst příslušný násobek prvního řádku od ostatních, aby $a'_{i1} = 0, i = 2, 3, \ldots, N$, takže první sloupec A souhlasí s prvním sloupcem 1.
- Dělit druhý řádek a'_{22} (už obecně není rovno a_{22}).
- Odečíst příslušný násobek druhého řádku od ostatních, aby $a'_{i2} = 0, i = 1, 3, \ldots, N$, takže druhý sloupec A souhlasí s druhým sloupcem 1. Přitom první sloupec nebude narušen, neboť $a'_{i1} = 0, i = 2, 3, \ldots, N$.
- Atd., $\ldots, N$
- Stejné operace se provádějí s pravou stranou!

Diagonální element, kterým se dělí, se nazývá hlavní prvek (*pivot*). Co když narazíme na nulový pivot?

2.3.1. Algoritmus GJE

Pouze pomocí operace 2:

- Dělit první řádek a_{11}, je-li $a_{11} \neq 0$ (tj. lineární kombinace).
- Odečíst příslušný násobek prvního řádku od ostatních, aby $a'_{i1} = 0, i = 2, 3, \ldots, N$, takže první sloupec A souhlasí s prvním sloupcem 1.
- Dělit druhý řádek a'_{22} (už obecně není rovno a_{22}).
- Odečíst příslušný násobek druhého řádku od ostatních, aby $a'_{i2} = 0, i = 1, 3, \ldots, N$, takže druhý sloupec A souhlasí s druhým sloupcem 1. Přitom první sloupec nebude narušen, neboť $a'_{i1} = 0, i = 2, 3, \ldots, N$.
- Atd., $\ldots, N$
- Stejné operace se provádějí s pravou stranou!

Diagonální element, kterým se dělí, se nazývá hlavní prvek (*pivot*). Co když narazíme na nulový pivot? Pak je třeba přibrat operaci 1 (prohazování řádků, částečná pivotace, *partial pivoting*) nebo operaci 1 i 3 (prohazování řádků i sloupců, plná pivotace, *full pivoting*). GJE bez pivotace je numericky nestabilní, i když nenarazíme na nulový pivot!

Abychom ‚nepokazili' již vybudované sloupce jednotkové matice 1, volíme při pivotaci řádky (sloupce) pod nebo na řádku (sloupci), který eliminujeme.

Plná pivotace je obtížnější, protože míchá pořadím řádků v řešení. Operace sloupcových pivotací je nutno zaznamenat a použít na závěr při rekonstrukci správného pořadí. Ukazuje se, že částečná pivotace je ‚téměř' stejně dobrý jako plná pivotace.

Jak vybrat pivot?

Jak vybrat pivot? Nejlepší volba: vybrat pivot s největší absolutní hodnotou. To závisí na škálování rovnic – co když některou vynásobíme velkým číslem a ovlivníme výběr pivotu?

Můžeme rovnice normalizovat tak, aby největší element v každé z nich byl roven 1 – implicitní pivotace. Vyžaduje navíc záznamy o škálovacích faktorech, jimiž rovnice násobíme.

Jak vybrat pivot? Nejlepší volba: vybrat pivot s největší absolutní hodnotou. To závisí na škálování rovnic – co když některou vynásobíme velkým číslem a ovlivníme výběr pivotu?

Můžeme rovnice normalizovat tak, aby největší element v každé z nich byl roven 1 – implicitní pivotace. Vyžaduje navíc záznamy o škálovacích faktorech, jimiž rovnice násobíme.

Rutina pro GJE s plnou pivotací (C) [Press et al., 1997a]
`void gaussj(float **a, int n, float **b, int m)`
IN: matice `a[1..n][1..n]`, její velikost `n`, pravé strany `b[1..n][1..m]` a jejich počet `m`.
OUT: `a[1..n][1..n]` a `b[1..n][1..m]`.
Řeší soustavu (2.5). Na výstupu bude matice `a` přepsána inverzní maticí a v `b` budou sloupečky odpovídajících řešení.

Rutina pro GJE s plnou pivotací (Fortran 77) [Press et al., 1997b]
`SUBROUTINE gaussj(a, n, np, b, m, mp)`
IN: `a(1:n,1:n)`, její velikost `n`/`np`, pravé strany `b(1:n,1:m)` a jejich počet `m`/`mp`.
OUT: `a(1:n,1:n)` a `b(1:n,1:m)`.
Řeší soustavu (2.5). Na výstupu bude matice `a` přepsána inverzní maticí a v `b` budou sloupečky odpovídajících řešení.

Rutina pro GJE s plnou pivotací (Fortran 90) [Press et al., 1997c]
`SUBROUTINE gaussj(a, b)`
IN: `a`, pravé strany `b`.
OUT: `a` a `b`.
Řeší soustavu (2.5). Na výstupu bude matice `a` přepsána inverzní maticí a v `b` budou sloupečky odpovídajících řešení.

2.3.2. Poznámky ke GJE

- Pro uskladnění stačí alokovat místo jen pro matici A a vektory b_j na pravé straně. Jak je postupně budována A^{-1}, tak je ničena původní matice. Podobně řešení x_j postupně nahrazují pravostranné vektory b_j.
- Počet operací je $\mathcal{O}(N^3 + N^2 M)$.
- Výhody:
  - Základní, solidní metoda, vhodná pro testování, selžou-li ostatní.
  - Stejně nebo mírně více stabilní jako ostatní přímé metody.

2.3.2. Poznámky ke GJE

- Pro uskladnění stačí alokovat místo jen pro matici A a vektory b_j na pravé straně. Jak je postupně budována A^{-1}, tak je ničena původní matice. Podobně řešení x_j postupně nahrazují pravostranné vektory b_j.
- Počet operací je $\mathcal{O}(N^3 + N^2M)$.
- Výhody:
  - Základní, solidní metoda, vhodná pro testování, selžou-li ostatní.
  - Stejně nebo mírně více stabilní jako ostatní přímé metody.
- Nevýhody:
  - Pravá strana musí být známa a uložena před aplikací GJE. Vynásobení dodatečné pravé strany A^{-1} je náchylné na kontaminaci zaokrouhlovací chybou.
  - Nepotřebujeme-li A^{-1}, je GJE $3\times$ pomalejší než nejlepší alternativní technika (LUD).

Vyzkoušejte demonstrační program metody GJE `xgaussj` z knihovny [Press et al., 1997a, Vetterling et al., 1993]. (Netiskne vstupy, na ty je nutno podívat se do souboru `x/DAT/matrx1.dat`.)

2.4. Gaussova eliminace se zpětnou substitucí (GEBS)

Na rozdíl od GJE neredukuje A na 1, nýbrž na trojúhelníkovou matici (v GJE algoritmu odečítáme od řádků pod aktuálním pivotem), a provádíme jen částečnou pivotaci (prvky a'_{ii} jsou nenulové, jinak by $\det \mathsf{A} = 0$):

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix} \tag{2.6}$$

To je Gaussova eliminace (GE).

2.4. Gaussova eliminace se zpětnou substitucí (GEBS)

Na rozdíl od GJE neredukuje A na 1, nýbrž na trojúhelníkovou matici (v GJE algoritmu odečítáme od řádků pod aktuálním pivotem), a provádíme jen částečnou pivotaci (prvky a'_{ii} jsou nenulové, jinak by $\det \mathsf{A} = 0$):

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix} \tag{2.6}$$

To je Gaussova eliminace (GE). Nyní zpětná substituce (*backsubstitution, BS*):

$$x_4 = \frac{b'_4}{a'_{44}}, \quad x_3 = \frac{1}{a'_{33}} \left(b'_3 - a'_{34} x_4 \right), \quad \ldots, \quad x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{N} a'_{ij} x_j \right) \tag{2.7}$$

Výhody: počet operací $\mathcal{O}(\frac{1}{3}N^3 + \frac{1}{2}N^2 M) \Rightarrow$ cca 1.5–3× rychlejší než GJE.
Nevýhody: Jako u GJE, pravá strana musí být známa a uložena před aplikací GEBS.

Následující metoda (LUD) má stejný počet operací jako GEBS, ale nesdílí s ní nevýhodu GEBS. Proto rutiny nejsou k dispozici.

2.5. LU dekompozice (LUD, též trojúhelníková faktorizace)

Přímo motivováno metodou GEBS. Kdyby se podařil rozklad matice A na součin dolní trojúhelníkové (*lower triangular*) matice L a horní trojúhelníkové (*upper triangular*) matice U

$$\mathsf{L} \cdot \mathsf{U} = \mathsf{A} \tag{2.8}$$

nebo v komponentách

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{2.9}$$

pak se řešení soustavy (2.1) rozpadá na řešení dvou soustav

$$\mathsf{L} \cdot \mathsf{y} = \mathsf{b}, \qquad \mathsf{U} \cdot \mathsf{x} = \mathsf{y} \tag{2.10}$$

jejichž řešení se provede přímou substitucí (*forward substitution*), $\mathcal{O}(\frac{1}{6}N^3)$ operací

$$y_1 = \frac{b_1}{\alpha_{11}}, \quad y_2 = \frac{1}{\alpha_{22}}\left(b_2 - \alpha_{21} y_1\right), \quad \ldots, \quad y_i = \frac{1}{\alpha_{ii}}\left(b_i - \textstyle\sum_{j=1}^{i-1} \alpha_{ij} y_j\right) \tag{2.11}$$

a zpětnou substitucí (*backsubstitution*), $\mathcal{O}(\frac{1}{2}N^3)$ operací

$$x_N = \frac{b_N}{\beta_{NN}}, \; x_{N-1} = \frac{1}{\beta_{N-1,N-1}}\left(y_{N-1} - \beta_{N-1,N} x_N\right), \; \ldots, \; x_i = \frac{1}{\beta_{ii}}\left(y_i - \textstyle\sum_{j=i+1}^{N} \beta_{ij} x_j\right) \tag{2.12}$$

Jakmile známe LU dekompozici matice A, můžeme řešit soustavu (2.1) pro libovolný počet pravých stran, na rozdíl od GJE a GEBS.

2.5.1. Jak provést LU dekompozici?

Rozpis rovnic (2.8) a (2.9) je

$$\begin{aligned} i<j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ii}\beta_{ij}=a_{ij}\\ i=j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ii}\beta_{jj}=a_{ij}\\ i>j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ij}\beta_{jj}=a_{ij} \end{aligned} \tag{2.13}$$

což je N^2 rovnic pro N^2+N neznámých. Proto připojíme podmínku

$$\alpha_{ii}\equiv 1, \quad i=1,2,\ldots,N \tag{2.14}$$

Jakmile známe LU dekompozici matice A, můžeme řešit soustavu (2.1) pro libovolný počet pravých stran, na rozdíl od GJE a GEBS.

2.5.1. Jak provést LU dekompozici?

Rozpis rovnic (2.8) a (2.9) je

$$\begin{aligned} i<j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ii}\beta_{ij}=a_{ij} \\ i=j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ii}\beta_{jj}=a_{ij} \\ i>j: &\quad \alpha_{i1}\beta_{1j}+\alpha_{i2}\beta_{2j}+\cdots+\alpha_{ij}\beta_{jj}=a_{ij} \end{aligned} \tag{2.13}$$

což je N^2 rovnic pro N^2+N neznámých. Proto připojíme podmínku

$$\alpha_{ii}\equiv 1, \quad i=1,2,\ldots,N \tag{2.14}$$

Soustavu (2.13) a (2.14) řeší Croutův algoritmus pouhým přestavěním v určitém pořádku.

- Položme $\alpha_{ii}=1 \quad \forall i=1,2,\ldots,N$.
- $\forall j=1,2,\ldots,N$ proveďme tyto dvě procedury: $\forall i=1,2,\ldots,j$ použijme první dvě rovnice (2.13) a (2.14) pro vyjádření β_{ij}

  $$\beta_{ij}=a_{ij}-\textstyle\sum_{k=1}^{i-1}\alpha_{ik}\beta_{kj} \qquad \text{nebo} \qquad \beta_{1j}=a_{1j} \quad \text{pro} \quad i=1 \tag{2.15}$$

  a $\forall i=j+1,j+2,\ldots,N$ použijeme třetí rovnici (2.13) pro vyjádření α_{ij}

  $$\alpha_{ij}=\frac{1}{\beta_{ij}}\left(a_{ij}-\textstyle\sum_{k=1}^{j-1}\alpha_{ik}\beta_{kj}\right) \tag{2.16}$$

  Proveďte obě procedury před přechodem k dalšímu j.

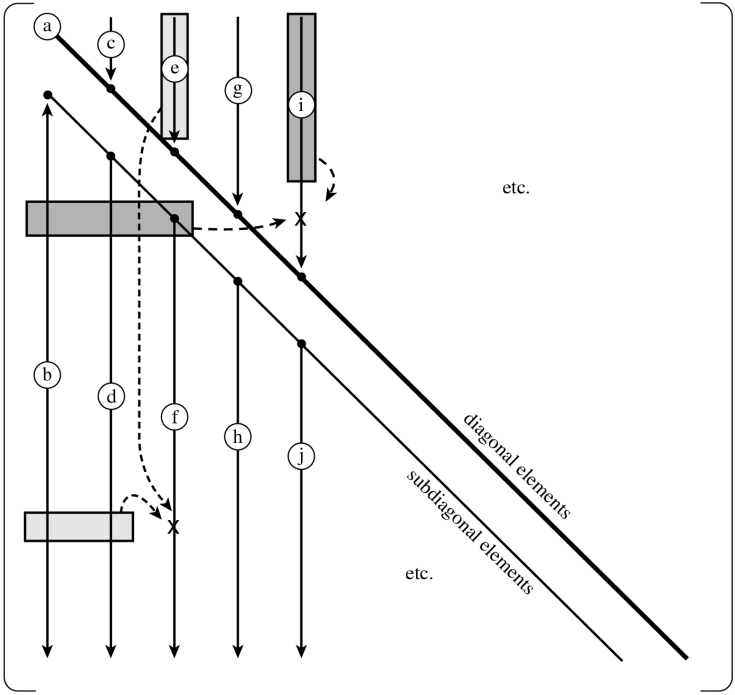
Croutova metoda umožňuje postupné přepisování elementů původní matice a_{ij} elementy LU rozkladu α_{ij} a βij. Díky (2.14) je můžeme N^2 elementů α_{ij} a βij uložit místo původní matice:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \xrightarrow{\text{LU}} \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ \alpha_{21} & \beta_{22} & \beta_{23} & \beta_{24} \\ \alpha_{31} & \alpha_{32} & \beta_{33} & \beta_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \beta_{44} \end{bmatrix} \tag{2.17}$$

Pro funkci a stabilitu Croutova algoritmu je $\alpha\omega$ pivotace. V následujících rutinách je implementován částečná implicitní pivotace, tj. přehazování řádků a normalizace největšího elementu rovnic na 1. To znamená, že ve skutečnosti se nerozkládá matice A, ale matice vytvořená permutací jejich řádků. Rutina musí kromě rozkladu dodat informaci o této permutaci.

LU dekompozice v následující rutině obsahuje $\mathcal{O}(\frac{1}{3}N^3)$ operací.

Na následující stránce je grafická ilustrace Croutova algoritmu pro LU dekompozici matice. Elementy původní matice jsou modifikovány v pořadí označeném písmeny a, b, c, atd. Stínované boxy ukazují již modifikované elementy, které jsou použity pro modifikaci dvou typických elementů označených $\times$.


a
c
e
g
i
etc.
b
d
f
h
j
diagonal elements
subdiagonal elements
etc.

Rutina pro LU dekompozici (C) [Press et al., 1997a]
`void ludcmp(float **a, int n, int *indx, float *d)`
IN: matice `a[1..n][1..n]` a `n`.
OUT: matice `a[1..n][1..n]`, permutační vektor `indx[1..n]` a `d`=± 1.
Zadanou matici přepíše LU dekompozicí její řádkové permutace. Vektor `indx` obsahuje záznam o permutaci, `d` je ± 1 podle toho, zda permutace je sudá nebo lichá.

Rutina pro LU dekompozici (Fortran 77) [Press et al., 1997b]
`SUBROUTINE ludcmp(a, n, np, indx, d)`
IN: matice `a(1:n,1:n)` a `n`/`np`.
OUT: matice `a(1:n,1:n)`, permutační vektor `indx(1:n)` a `d`=± 1.
Zadanou matici přepíše LU dekompozicí její řádkové permutace. Vektor `indx` obsahuje záznam o permutaci, `d` je ± 1 podle toho, zda permutace je sudá nebo lichá.

Rutina pro LU dekompozici (Fortran 90) [Press et al., 1997c]
`SUBROUTINE ludcmp(a, indx, d)`
IN: matice `a`.
OUT: matice `a`, permutační vektor `indx` a `d`=± 1.
Zadanou matici přepíše LU dekompozicí její řádkové permutace. Vektor `indx` obsahuje záznam o permutaci, `d` je ± 1 podle toho, zda permutace je sudá nebo lichá.

Rutina pro přímou a zpětnou substituci (C) [Press et al., 1997a]
`void lubksb(float **a, int n, int *indx, float b[])`
IN: matice `a[1..n][1..n]`, `n`, vektory `indx[1..n]` a `b[1..n]`.
OUT: vektor `b[1..n]`.
Matice `a` a vektor `indx` jsou LU dekompozice a perm. vektor dodané rutinou `ludcmp`. Pravá strana `b` je na výstupu přepsána řešením, `a`, `n` a `indx` se nemění a mohou být použity pro následná volání.

Rutina pro přímou a zpětnou substituci (Fortran 77) [Press et al., 1997b]
`SUBROUTINE lubksb(a, n, np, indx, b)`
IN: matice `a(1:n,1:n)`, `n`/`np`, vektory `indx(1:n)` a `b(1:n)`.
OUT: vektor `b(1:n)`.
Matice `a` a vektor `indx` jsou LU dekompozice a perm. vektor dodané rutinou `ludcmp`. Pravá strana `b` je na výstupu přepsána řešením, `a`, `n` a `indx` se nemění a mohou být použity pro následná volání.

Rutina pro přímou a zpětnou substituci (Fortran 90) [Press et al., 1997c]
`SUBROUTINE lubksb(a, indx, b)`
IN: matice `a`, vektory `indx` a `b`.
OUT: vektor `b`.
Matice `a` a vektor `indx` jsou LU dekompozice a perm. vektor dodané rutinou `ludcmp`. Pravá strana `b` je na výstupu přepsána řešením, `a`, `n` a `indx` se nemění a mohou být použity pro následná volání.

Rutiny berou v úvahu skutečnost, že vektor `b` může začínat mnoha nulovými elementy, takže je efektivní pro inverzi matice. Celkový počet operací LU a substitucí: $\mathcal{O}\left[\left(\frac{1}{3}+\frac{1}{6}+\frac{1}{2}\right)N^3\right] = \mathcal{O}(N^3)$.

Typické použití:

```
float **a,*b,d;
int n,*indx;
...
ludcmp(a,n,indx,&d);  /* LU dekompozice ‚jednou provždy‘. */
lubksb(a,n,indx,b);   /* Řešíme soustavu s pravou stranou b ... */
lubksb(a,n,indx,b_2); /* ... a s jinou pravou stranou ... */
...
lubksb(a,n,indx,b_m); /* ... a s další a další pravou stranou. */
```

2.5.2. Aplikace LU dekompozice

Inverze matice

Matice `y` obsahuje inverzi původní matice `a`. Počet operací je prakticky stejný jako u GJE. Potřebujeme-li spočítat $A^{-1} \cdot B$, dosadíme místo jednotkových sloupců 1 sloupce B. Je to přesnější a ušetří to celé maticové násobení.

```
#define N ...
float **a,**y,d,*b;
int i,j,*indx;
...
ludcmp(a,N,indx,&d);  /* LU dekompozice pouze jednou. */
for (j=1;j<=N;j++) {  /* Inverze po sloupcích. */
    for(i=1;i<=N;i++) b[i]=0.0;
    b[j]=1.0;
    lubksb(a,N,indx,b);
    for (i=1;i<=N;i++) y[i][j]=b[i];
}
```

2.5.2. Aplikace LU dekompozice

Inverze matice

Matice `y` obsahuje inverzi původní matice `a`. Počet operací je prakticky stejný jako u GJE. Potřebujeme-li spočítat $A^{-1} \cdot B$, dosadíme místo jednotkových sloupců 1 sloupce B. Je to přesnější a ušetří to celé maticové násobení.

```
#define N ...
float **a,**y,d,*b;
int i,j,*indx;
...
ludcmp(a,N,indx,&d);  /* LU dekompozice pouze jednou. */
for (j=1;j<=N;j++) {  /* Inverze po sloupcích. */
    for(i=1;i<=N;i++) b[i]=0.0;
    b[j]=1.0;
    lubksb(a,N,indx,b);
    for (i=1;i<=N;i++) y[i][j]=b[i];
}
```

Determinant matice

Determinant je $\det A = \prod_{j=1}^{N} \beta_{jj}$ (proč?). Rutina `ludcmp` počítá dekompozici řádkové permutace matice A, je třeba výsledek násobit `d`. Rutina `lubksb` není potřebná. Hrozí-li přetečení, lze použít vhodné škálování nebo místo násobení použít sčítání/odčítání logaritmů.

```
#define N ...
float **a,d;
int j,*indx;
...
ludcmp(a,N,indx,&d);  /* Vrátí d=+-1. */
for (j=1;j<=N;j++) d *= a[j][j];
```

Vyzkoušejte demonstrační programy metody LUD `xludcmp` a substituce `xlubksb` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

2.6. Tridiagonální a pásově diagonální systémy rovnic

Tridiagonální matice: nenulové elementy jsou jen na diagonále $\pm$ jeden sloupec.
Pásově diagonální matice: nenulové elementy pouze na několika diagonálních liniích přilehlých k hlavní diagonále.

2.6.1. Tridiagonální soustavy

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & & & & \\ a_2 & b_2 & c_2 & 0 & \cdots & & & & \\ 0 & a_3 & b_3 & c_3 & \cdots & & & & \\ 0 & 0 & a_4 & b_4 & \cdots & & & & \\ & & & & \ddots & & & & \\ & & & & \cdots & b_{N-3} & c_{N-3} & 0 & 0 \\ & & & & \cdots & a_{N-2} & b_{N-2} & c_{N-2} & 0 \\ & & & & \cdots & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & \cdots & 0 & 0 & a_N & b_N \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{N-3} \\ u_{N-2} \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ \vdots \\ r_{N-3} \\ r_{N-2} \\ r_{N-1} \\ r_N \end{bmatrix} \tag{2.18}$$

LU dekompozice, přímá a zpětná substituce jsou jen $\mathcal{O}(N)$ operace. Pivotace se neimplementuje, protože můžeme narazit na nulový pivot i u regulární matice. V praxi mají úlohy vedoucí k tridiagonální soustavě obvykle dodatečné vlastnosti garantující neselhání algoritmu (např. diagonální dominanci $|b_j| > |a_j| + |c_j|$). V případě problémů je vždy možno použít obecnější metody pro řešení pásových soustav s pivotací.

Rutina pro řešení tridiagonální soustavy (C) [Press et al., 1997a]
`void tridag(float a[],float b[],float c[],float r[],float u[],unsigned long n)`
IN: subdiagonála `a[1..n]`, diagonála `b[1..n]`, superdiagonála `c[1..n]`, pr. strana `r[1..n]`, `n`.
OUT: řešení `u[1..n]`.
Vektory `a`, `b`, `c`, `r` nejsou přepsány. Prvky a_1 a c_N jsou ‚dummy' a nejsou rutinou použity.

Rutina pro řešení tridiagonální soustavy (Fortran 77) [Press et al., 1997b]
`SUBROUTINE tridag(a, b, c, r, u, n)`
IN: subdiagonála `a(1:n)`, diagonála `b(1:n)`, superdiagonála `c(1:n)`, pr. strana `r(1:n)`, `n`.
OUT: řešení `u(1:n)`.
Vektory `a`, `b`, `c`, `r` nejsou přepsány. Prvky a_1 a c_N jsou ‚dummy' a nejsou rutinou použity.

Rutina pro řešení tridiagonální soustavy (Fortran 90) [Press et al., 1997c]
`SUBROUTINE tridag_ser(a, b, c, r, u)`
`RECURSIVE SUBROUTINE tridag_par(a, b, c, r, u)`
IN: subdiagonála `a`, diagonála `b`, superdiagonála `c`, pravá strana `r`, `n`.
OUT: řešení `u`.
Vektory `a`, `b`, `c`, `r` nejsou přepsány. Prvky a_1 a c_N jsou ‚dummy' a nejsou rutinou použity.

Vyzkoušejte demonstrační program pro řešení tridiagonální soustavy `xtridag` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

2.6.2. Pásově diagonální soustavy (band diagonal)

Kromě diagonály má matice $m_1 \geq 0$ subdiagonál a $m_2 \geq 0$ superdiagonál; přesně, $a_{ij} = 0$, když $j > i + m_2$ nebo $i > j + m_1$. (Přitom $\max\{m_1, m_2\} \ll N$.)

Kompaktní uložení v paměti v matici $(m_1 + 1 + m_2) \times N$. Příklad pro $N = 7, m_1 = 2, m_2 = 1$:

$$\begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 5 & 0 & 0 & 0 & 0 \\ 9 & 2 & 6 & 5 & 0 & 0 & 0 \\ 0 & 3 & 5 & 8 & 9 & 0 & 0 \\ 0 & 0 & 7 & 9 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 8 & 4 & 6 \\ 0 & 0 & 0 & 0 & 2 & 4 & 4 \end{bmatrix} \quad \text{je uložena jako} \quad \begin{bmatrix} \bullet & \bullet & 3 & 1 \\ \bullet & 4 & 1 & 5 \\ 9 & 2 & 6 & 5 \\ 3 & 5 & 8 & 9 \\ 7 & 9 & 3 & 2 \\ 3 & 8 & 4 & 6 \\ 2 & 4 & 4 & \bullet \end{bmatrix} \tag{2.19}$$

(Zde • označuje nevyužité místo.)

V této sekci jsou k dispozici tři rutiny: první pro násobení sloupcového vektoru pásově diagonální maticí zleva, zbývající dvě jsou analogií rutin pro LU dekompozici a přímou/zpětnou substituci v případě pásově diagonální matice.

Rutina pro násobení vektoru pásově diagonální maticí zleva (C) [Press et al., 1997a]
`void banmul(float **a, unsigned long n, int m1, int m2, float x[], float b[])`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE banmul(a, n, m1, m2, np, mp, x, b)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE banmul(a, m1, m2, x, b)`
IN: (C: `a[1..n][1..m1+m2+1]`), (F77: `a(1:n,1:m1+m2+1)`), (F90: `a`), (C: `n`), (F77: `n/np`, `m/mp`), `m1`, `m2`, násobený vektor `x`.
OUT: vektor `b=Ax`.
Ostatní argumenty zůstávají nezměněny.

Rutina pro LU dekompozici pásově diagonální matice – analogie `ludcmp` (C) [Press et al., 1997a]
`void bandec(float **a, unsigned long n, int m1, int m2, float **al, unsigned long indx[], float *d)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE bandec(a, n, m1, m2, np, mp, al, mpl, indx, d)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE bandec(a, m1, m2, al, indx, d)`
IN: (C: `a[1..n][1..m1+m2+1]`), (F77: `a(1:n,1:m1+m2+1)`), (F90: `a`), (C: `n`), (F77: `n/np`, `m/mp`, `mpl`), `m1`, `m2`.
OUT: (C: `a[1..n][1..m1+m2+1]`, `al[1..n][1..m1]`, `indx[1..n]`), (F77: `a(1:n,1:m1+m2+1)`, `al(1:n,1:m1)`, `indx(1:n)`),
(F90: `a`, `al`, `indx`), `d`.
Vstupní matice `a` je přepsána horní trojúhelníkovou maticí, dolní trojúhelníková matice je uložena v `al`. V `indx` je záznam o permutacích, `d=+1(-1)` pro sudou (lichou) permutaci.

Rutina pro řešení pásově diagonální soustavy – analogie `lubksb` (C) [Press et al., 1997a]
`void banbks(float **a, unsigned long n, int m1, int m2, float **al, unsigned long indx[], float b[])`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE banbks(a, n, m1, m2, np, mp, al, mpl, indx, b)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE banbks(a, m1, m2, al, indx, b)`
IN: (C: `a[1..n][1..m1+m2+1]`, `al[1..n][1..m1]`, `indx[1..n]`, `b[1..n]`),
(F77: `a(1:n,1:m1+m2+1)`, `a(1:n,1:m1)`, `indx(1:n)`, `b(1:n)`),
(F90: `a`, `al`, `indx`, `b`), (C: `n`), (F77: `n/np`, `m/mp`, `mpl`), `m1`, `m2`.
OUT: (C: `b[1..n]`), (F77: `b(1:n)`), (F90: `b`).
Matice `a`, `al` a permutační vektor `indx` dodá rutina `bandec`. Řešení přepíše pravou stranu `b`, ostatní jsou zachovány a mohou být použity v následujících voláních.

2.7. Iterační zpřesnění

Při použití přímých metod má zaokrouhlovací chyba tendenci k akumulaci. Tento jev je tím větší, čím je matice blíže k singulární. V takovém případě lze získané nepřesné řešení zpřesnit pomocí iteračního zpřesnění (*iteration improvement*).

Předpokládejme, že x je přesným řešením soustavy (2.1). Místo toho získáme nepřesné řešení $\mathsf{x} + \delta\mathsf{x}$. Po vynásobení maticí A dostaneme odlišný pravostranný vektor:

$$\mathsf{A} \cdot (\mathsf{x} + \delta\mathsf{x}) = \mathsf{b} + \delta\mathsf{b} \tag{2.20}$$

Odečtením (2.20) a (2.1) máme

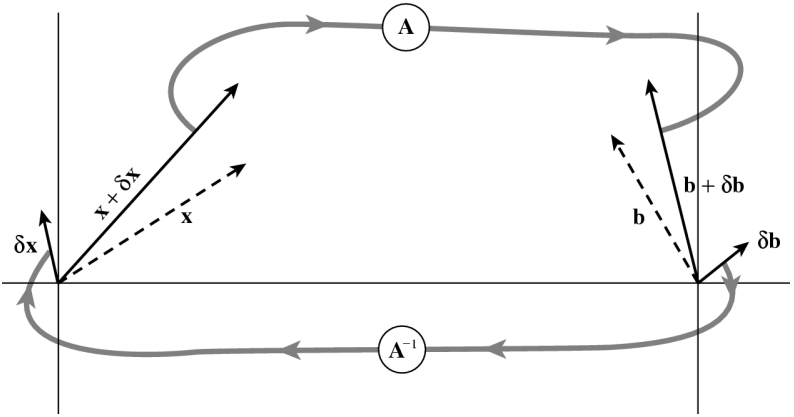
$$\mathsf{A} \cdot \delta\mathsf{x} = \delta\mathsf{b} \tag{2.21}$$

Vypočteme-li z (2.20) $\delta\mathsf{b}$ a dosadíme-li do pravé strany (2.21), obdržíme

$$\mathsf{A} \cdot \delta\mathsf{x} = \mathsf{A} \cdot (\mathsf{x} + \delta\mathsf{x}) - \mathsf{b} \tag{2.22}$$

Pravá strana (2.22) je známá, a řešením dostaneme chybu $\delta\mathsf{x}$. Odečtením chyby od původního nepřesného řešení dostaneme zpřesnění. Je nezbytné počítat pravou stranu (2.22) v double precision. Máme-li LU dekompozici matice A, můžeme ji použít.

Iterační zpřesnění má $\mathcal{O}(N^2)$ operací. Rutinu `mprove` lze volat libovolný počet krát; pokud nejsme daleko od skutečného řešení, jedno nebo dvě volání obvykle dostačují.



Rutina pro iterační zpřesnění (C) [Press et al., 1997a]
`void mprove(float **a, float **alud, int n, int indx[], float b[], float x[])`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE mprove(a, alud, n, np, indx, b, x)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE mprove(a, alud, indx, b, x)`
IN: (C: `a[1..n][1..n]`, `alud[1..n][1..n]`, `indx[1..n]`, `b[1..n]`, `x[1..n]`),
(F77: `a(1:n,1:n)`, `alud(1:n,1:n)`, `indx(1:n)`, `b(1:n)`, `x(1:n)`),
(F90: `a`, `alud`, `indx`, `b`, `x`), (C, F77: `n`), (F77: `np`).
OUT: (C: `x[1..n]`), (F77: `x(1:n)`, (C: `x`)).
DEP: `mprove` — `lubksb`
Vstupní matice `alud` a vektor `indx` jsou LU dekompozice a permutační vektor dodané rutinou `ludcmp`. Vstupní nepřesný vektor `x` bude na výstupu přepsán zpřesněním.

Úlohy ke kapitole 2

1. Ze zdrojových kódů rutiny odvoďte počet operací $\mathcal{O}(N^3 + N^2 M)$ pro metodu GJE.
2. Uvažujte soustavu lineárních algebraických rovnic (2.1) s maticí A naplněnou elementy

$$a_{ij} = \frac{1}{i + j - 1} \tag{2.23}$$

(tzv. Hilbertova matice). Pravou stranu naplňte elementy

$$b_i = \sum_{j=1}^{N} a_{ij} \tag{2.24}$$

Exaktní řešení soustavy je evidentně $x_1 = x_2 = \ldots = x_N = 1$ pro libovolné N. Řešte (2.1) s uvedenými elementy s použitím GJE a LUD pro $N = 2, 3, 5, 7, 10, 40$. Vysvětlete získané výsledky.

3. Hilbertovu matici z předešlé úlohy nahraďte maticí s elementy

$$\tilde{a}_{ij} = \frac{1}{\cos(i + j) - 1} \tag{2.25}$$

(kosinus celého čísla nemůže být 1) a analogicky definovanou pravou stranou

$$\tilde{b}_i = \sum_{j=1}^{N} \tilde{a}_{ij} \tag{2.26}$$

Exaktní řešení soustavy je evidentně opět $x_1 = x_2 = \ldots = x_N = 1$ pro libovolné N. Řešte (2.1) s uvedenými elementy s použitím GJE a LUD pro $N = 2, 3, 5, 7, 10, 40$. Vysvětlete získané výsledky.

Shrnutí kapitoly: Naučili jste se řešit soustavy lineárních algebraických rovnic (SLAR). Ačkoliv tato úloha sama o sobě může připadat na první pohled nudná, má ve skutečnosti zásadní důležitost pro řešení jiných problémů numerické matematiky a fyzikálních simulací.

Další zdroje:

- Kapitola 2 v [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c]. Pro jazyk C dostupné na
  http://www.library.cornell.edu/nr/bookcpdf/,
  pro Fortran na
  http://www.library.cornell.edu/nr/bookfpdf/.
- Úvod do Fortranu 95 a do numerických metod [Sandu, 2001]. Obsahuje Kapitolu 15 o soustavách lineárních algebraických rovnic. Dostupné online na
  http://www.cs.mtu.edu/~asandu/Courses/CS2911/fortran_notes/main.html.

3. Interpolace a extrapolace

Rychlý náhled kapitoly: V této kapitole budou vysvětleny principy interpolace a extrapolace datových souborů. Je třeba striktně odlišit inter- a extrapolaci od modelování (fitování) dat. U inter- a extrapolace prochází hledaná křivka vždy všemi datovými body, což u modelování, např. metodou nejmenších čtverců, nepožadujeme. Ukážeme si různé metody interpolace.

Cíle kapitoly:

- Naučit se metodě polynomiální inter- a extrapolace.
- Naučit se metodě racionální inter- a extrapolace.
- Naučit se metodě interpolace kubickými splajny.
- Naučit se pracovat s pomocnými rutinami pro prohledávání uspořádané tabulky.
- Naučit se metodě nalezení koeficientů interpolačního polynomu.

Klíčová slova kapitoly: Interpolace, extrapolace, polynomiální, racionální; řád interpolace; Nevilleův rekurzivní algoritmus; Bulirschův–Stoerův rekurzivní algoritmus; kubický splajn, přirozený kubický splajn; prohledávání uspořádané tabulky; koeficienty interpolačního polynomu

Motto: *Extrapolujte ceny benzínu za uplynulé čtyři měsíce do čtyř následujících let. (Psáno v říjnu 2008.)*

Mějme N dvojic $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ (předpokládejme $x_1 < x_2 < \ldots < x_N$), např. výsledky měření nebo výpočtů; x_i mohou, ale nemusí být ekvidistantní. Předpokládejme dále, získaná závislost je popsána nějakou funkcí f tak, že $y_i = f(x_i)$.

Úkol: odhadnout hodnotu $y = f(x)$ pro $x \notin \{x_i\}_{i=1,2,\ldots,N}$. Je-li $x_1 \leq x \leq x_N$ ($x < x_1$ nebo $x > x_N$), mluvíme o interpolaci (extrapolaci – ‚hazardnější‘).

Interpolace/extrapolace modeluje funkci určitého typu (polynom, racionální lomená funkce, trigonometrická funkce, . . .) mezi/za známými body.

Motto: *Extrapolujte ceny benzínu za uplynulé čtyři měsíce do čtyř následujících let. (Psáno v říjnu 2008.)*

Mějme N dvojic $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ (předpokládejme $x_1 < x_2 < \ldots < x_N$), např. výsledky měření nebo výpočtů; x_i mohou, ale nemusí být ekvidistantní. Předpokládejme dále, získaná závislost je popsána nějakou funkcí f tak, že $y_i = f(x_i)$.

Úkol: odhadnout hodnotu $y = f(x)$ pro $x \notin \{x_i\}_{i=1,2,\ldots,N}$. Je-li $x_1 \leq x \leq x_N$ ($x < x_1$ nebo $x > x_N$), mluvíme o interpolaci (extrapolaci – ‚hazardnější‘).

Interpolace/extrapolace modeluje funkci určitého typu (polynom, racionální lomená funkce, trigonometrická funkce, . . .) mezi/za známými body.

Koncepčně má inter/extrapolace dvě fáze:

1. Přizpůsobení interpolační funkce datům.
2. Vyčíslení interpolační funkce v bodě x.

Tento dvoufázový algoritmus je výpočetně méně efektivní a náchylnější k zaokrouhlovací chybě než algoritmy, které konstruují $f(x)$ přímo z N tabelovaných hodnot, začínajíce z nejbližších bodů za přidávání postupně klesajících korekcí přibíráním dalších a dalších bodů, dokud nevezmou v úvahu všech N bodů.

Počet operací je $\mathcal{O}(N^2)$, a poslední korekci lze použít jako odhad chyby. Tato metoda poskytuje interpolační funkci, jejíž první a vyšší derivace obecně nejsou spojité (při průchodu x tabelovanými hodnotami x_i).

Počet operací je $\mathcal{O}(N^2)$, a poslední korekci lze použít jako odhad chyby. Tato metoda poskytuje interpolační funkci, jejíž první a vyšší derivace obecně nejsou spojité (při průchodu x tabelovanými hodnotami x_i).

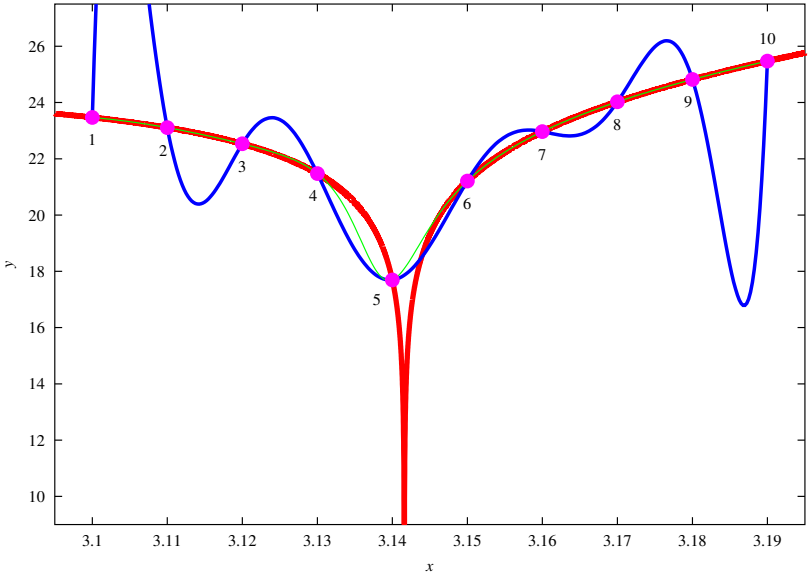
Potřebujeme-li spojité derivace, použijeme splajny (*spline functions*). Splajn je polynom spojující po sobě následující páry bodů tak, aby byla garantována spojitost splajnu do určité derivace, včetně bodů napojení. Nejrozšířenější jsou kubické splajny, které garantují spojitost do druhé derivace včetně. Splajny jsou stabilnější než obyčejná polynomiální interpolace, s menší možností ‚divokých oscilací' mezi tabelovanými body.

Počet operací je $\mathcal{O}(N^2)$, a poslední korekci lze použít jako odhad chyby. Tato metoda poskytuje interpolační funkci, jejíž první a vyšší derivace obecně nejsou spojité (při průchodu x tabelovanými hodnotami x_i).

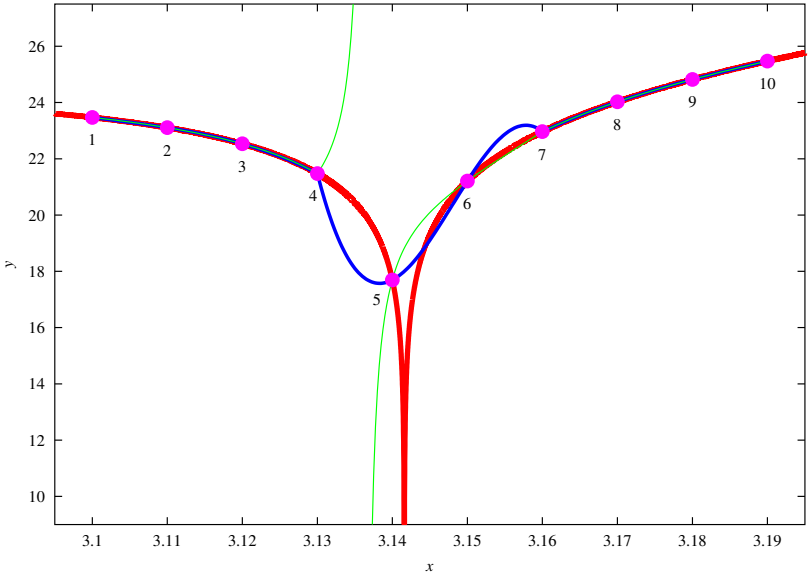
Potřebujeme-li spojité derivace, použijeme splajny (*spline functions*). Splajn je polynom spojující po sobě následující páry bodů tak, aby byla garantována spojitost splajnu do určité derivace, včetně bodů napojení. Nejrozšířenější jsou kubické splajny, které garantují spojitost do druhé derivace včetně. Splajny jsou stabilnější než obyčejná polynomiální interpolace, s menší možností ‚divokých oscilací' mezi tabelovanými body.

Globální interpolace přes všechna data může dát zcela chybné výsledky. Je lepší dát přednost lokální interpolaci přes data ‚sousedící' s x, ačkoli i v takovém případě nelze v okolí určitého bodu interpolovat ze znalosti několika okolních hodnot. Inter/extrapolace (dále jen interpolace) vždy předpokládá jistý stupeň hladkosti. Příklady jsou na obrázku na následující a další straně.

Občas je zapotřebí (v případě polynomiální interpolace) znát koeficienty interpolačního polynomu (není zapotřebí pro vyčíslení $f(x)$!). Jde-li však jen o znalost $f(x)$ mezi x_i, je lépe použít shora popsaného způsobu.



Body $(x_1, y_1), (x_2, y_2), \ldots, (x_{10}, y_{10})$ s ekvidistantními x_i leží na grafu funkce $f(x) = 3x^2 + \frac{1}{\pi^4} \ln\left[(\pi - x)^2\right] + 1$ (o tom interpolace ale ‚nic neví‘). Globální polynomiální interpolace (v tomto případě polynomem 9. stupně) je nepoužitelná; globální racionální interpolace dává s výjimkou intervalu mezi body 4 a 6 (v okolí singularity) rozumné výsledky.



Rozdělíme deset bodů z předchozího obrázku do tří skupin po čtyřech se společnými sousedními body, a provedeme lokálně v každé čtveřici kubickou polynomiální interpolaci a racionální interpolaci. V hladkých částech funkce oba typy interpolace věrně kopírují průběh $f(x)$, v okolí singularity jsou nepoužitelné.

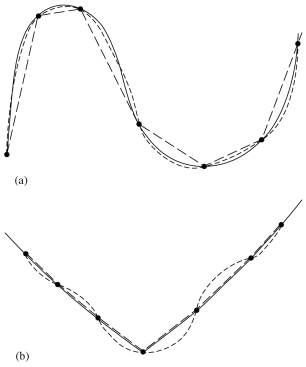
Počet bodů použitých pro interpolaci minus jedna je řád (*order*) interpolace – v případě polynomiální interpolace je řád roven stupni interpolačního polynomu.

Vyšší řád neznamená automaticky vyšší přesnost! Přidáním bodů vzdálených od x dostaneme polynom vyššího řádu, který má tendenci k oscilacím. Přidání blízkých bodů, pokud jsou k dispozici, je naopak vhodné.

Počet bodů použitých pro interpolaci minus jedna je řád (*order*) interpolace – v případě polynomiální interpolace je řád roven stupni interpolačního polynomu.

Vyšší řád neznamená automaticky vyšší přesnost! Přidáním bodů vzdálených od x dostaneme polynom vyššího řádu, který má tendenci k oscilacím. Přidání blízkých bodů, pokud jsou k dispozici, je naopak vhodné.

Doporučení: používejte interpolaci se 3 a 4 body (kvadratický a kubický polynom), nejvýše s 5 až 6 body (bikvadratický až 5. řádu).



(a) Hladká funkce (plná čára) je přesněji interpolována polynomem vyššího řádu (kratší čárky) než 1. řádu (delší čárky).

(b) Funkce s ostrými hranami nebo rychle se měnící derivací je přesněji interpolována polynomem nižšího řádu (delší čárky) než vyššího řádu (kratší čárky).

3.1. Polynomiální interpolace a extrapolace

$N \geq 1$ body $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ prochází jednoznačný polynom $N-1$ stupně (interpolační polynom $N-1$ stupně). Je dán explicitně Lagrangeovou formulí

$$\begin{aligned} P(x) = & \frac{(x-x_2)(x-x_3)\cdots(x-x_N)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_N)} y_1 \\ & + \frac{(x-x_1)(x-x_3)\cdots(x-x_N)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_N)} y_2 \\ & + \cdots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)\cdots(x_N-x_{N-1})} y_N \end{aligned} \tag{3.1}$$

Platí $P(x_i) = y_i$. Lagrangeova formule je v praxi neužitečná: neposkytuje odhad chyby, obtížně programovatelná, sklon k zaokrouhlovací chybě.

Nevilleův algoritmus konstruuje tentýž interpolační polynom rekurzivní cestou: místo abychom do hotového polynomu (3.1) dosazovali x, ze známého x teprve rekurzivně budujeme polynom $P(x)$.

Pokuste se vysvětlit příčinu náchylnosti Lagrangeovy formule (3.1) k zaokrouhlovací chybě.

Rekurzivní Nevilleův algoritmus funguje takto (ilustrace pro $N = 4$ a $x \in (x_1, x_2)$):

$x_1 : \; P_1 \equiv y_1$

$x_2 : \; P_2 \equiv y_2$

$x_3 : \; P_3 \equiv y_3$

$x_4 : \; P_4 \equiv y_4$

$$P_{12} = \frac{(x-x_2)P_1+(x_1-x)P_2}{x_1-x_2}$$

$$P_{23} = \frac{(x-x_3)P_2+(x_2-x)P_3}{x_2-x_3}$$

$$P_{34} = \frac{(x-x_4)P_3+(x_3-x)P_4}{x_3-x_4}$$

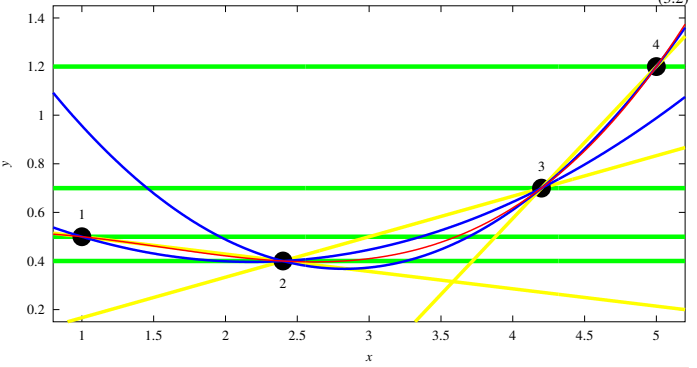
$$P_{123} = \frac{(x-x_3)P_{12}+(x_1-x)P_{23}}{x_1-x_3}$$

$$P_{234} = \frac{(x-x_4)P_{23}+(x_2-x)P_{34}}{x_2-x_4}$$

$$P_{1234} = \frac{(x-x_4)P_{123}+(x_1-x)P_{234}}{x_1-x_4}$$

Z ‚rodičovských‘ polynomů P_1 a P_2 vzniká ‚dceřiný‘ polynom P_{12} stupně o 1 vyššího. Podobně P_{23} je potomkem P_2 a P_3, P_{123} je potomkem P_{12} a P_{23}, atd., až nakonec výsledný interpolační polynom P_{1234} je potomkem P_{123} a P_{234}.

(3.2)



Ověřte, že $P_{1234}(x)$ je identický s Lagrangeovou formulí (3.1) pro $N = 4$.

Obecný tvar rekurzivní formule pro výpočet ‚dceřiného' polynomu z ‚rodičovských' polynomů je

$$P_{i,i+1,\ldots,i+m} = \frac{(x - x_{i+m})P_{i,i+1,\ldots,i+m-1} + (x_i - x)P_{i+1,i+2,\ldots,i+m}}{x_i - x_{i+m}} \tag{3.3}$$

Pro diference mezi ‚dceřiným' polynomem a ‚rodičovskými' polynomy

$$C_{m,i} \equiv P_{i,i+1,\ldots,i+m} - P_{i,i+1,\ldots,i+m-1} \tag{3.4}$$

$$D_{m,i} \equiv P_{i,i+1,\ldots,i+m} - P_{i+1,i+2,\ldots,i+m} \tag{3.5}$$

snadno pomocí (3.3) odvodíme rekurzivní formule pro

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \tag{3.6}$$

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \tag{3.7}$$

Podle (3.4) a (3.5) dostaneme ‚dceřiný' polynom přičítáním korekcí C a D k ‚rodičovským' polynomům. Finální interpolační polynom (poslední potomek) pak dostaneme jako sumu libovolného y_i plus soubor korekcí C a D, které k němu tvoří cestu.

Poslední přičtená korekce pak slouží jako odhad chyby.

Rutina pro polynomiální interpolaci (C) [Press et al., 1997a]
`void polint(float xa[], float ya[], int n, float x, float *y, float *dy)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE polint(xa, ya, n, x, y, dy)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE polint(xa, ya, x, y, dy)`
IN: (C: `xa[1..n]`, `ya[1..n]`), (F77: `xa(1:n)`, `ya(1:n)`), (F90: `xa`, `ya`), (C, F77: `n`), `x`.
OUT: `y` a `dy`.
V proměnné `y` vrátí hodnotu interpolačního polynomu P stupně `n`, $P($`xa[i]`$) =$`ya[i]`, v bodě `x`.
V proměnné `dy` vrátí odhad chyby.

Typické použití:

```
#define N 100 /* number of data points */
#define NP 4  /* number of points for (cubic) interpolation */
...
float x,y,dy,*xa,*ya;
...
xa=vector(1,N); /* allocate memory for all data */
ya=vector(1,N);
...
polint(xa,ya,NP,x,&y,&dy); /* thru pts 1,2,3,4, xa[1]<=x<=xa[4] */
polint(xa+9,ya+9,NP,x,&y,&dy); /* thru pts 10,..,13, xa[10]<=x<=xa[13] */
polint(xa+96,ya+96,NP,x,&y,&dy); /* thru last 4 pts, xa[97]<=x<=xa[100] */
```

Pro C-programátory: Proč je pro interpolaci přes body $10, \ldots, 13$ v rutině `polint` argument `xa+9`?

Vyzkoušejte demonstrační program pro polynomiální interpolaci `xpolint` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

3.2. Racionální interpolace a extrapolace

Racionální funkce procházející $m+1$ body $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_{i+m}, y_{i+m})$ tvaru

$$R_{i,i+1,\ldots,i+m} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu} \tag{3.8}$$

obsahuje $\mu + \nu + 1$ neznámých koeficientů p_i a q_i (q_0 je libovolné), proto musí platit

$$m = \mu + \nu \tag{3.9}$$

Je třeba specifikovat stupeň polynomu jak ve jmenovateli, tak v čitateli.

Racionální funkce jsou někdy lepší než polynomy, neboť mohou modelovat póly. Tyto póly mohou nastávat pro reálné x, častěji však v komplexní rovině v analytickém pokračování. Takové póly mohou zcela zruinovat polynomiální aproximaci. Nakreslíme-li v komplexní rovině kruh obsahující všechny x_i, polynomiální interpolace nebude fungovat, pokud nejbližší pól nebude dostatečně daleko vně kruhu. Naproti tomu, racionální interpolace bude fungovat, jestliže má Q_ν dostatečný stupeň pro vystižení blízkých pólů.

Bulirsch a Stoer nalezli algoritmus Nevilleova typu – vytváří interpolační diagonální racionální funkci, pro kterou platí $\mu = \nu$ (je-li m sudé) nebo $\nu = \mu + 1$ (je-li m liché).

Detaily Bulirschova–Stoerova algoritmu: generující rekurence je

$$R_{i,i+1,\ldots,i+m} = R_{i+1,\ldots,i+m} + \frac{R_{i+1,\ldots,i+m} - R_{i,\ldots,i+m-1}}{\left(\frac{x-x_i}{x-x_{i+m}}\right)\left(1 - \frac{R_{i+1,\ldots,i+m} - R_{i,\ldots,i+m-1}}{R_{i+1,\ldots,i+m} - R_{i+1,\ldots,i+m-1}}\right) - 1} \tag{3.10}$$

s počátečními podmínkami

$$R_i = y_i, \qquad R \equiv [R_{i,i+1,\ldots,i+m} \text{ pro } m = -1] = 0 \tag{3.11}$$

Vztahy analogická vztahům (3.4), (3.5), (3.6) a (3.7) pro polynomiální interpolaci mají tvar

$$C_{m,i} \equiv R_{i,i+1,\ldots,i+m} - R_{i,i+1,\ldots,i+m-1} \tag{3.12}$$

$$D_{m,i} \equiv R_{i,i+1,\ldots,i+m} - R_{i+1,i+2,\ldots,i+m} \tag{3.13}$$

a

$$D_{m+1,i} = \frac{C_{m,i+1}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \tag{3.14}$$

$$C_{m+1,i} = \frac{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \tag{3.15}$$

kde jsme použili

$$C_{m+1,i} - D_{m+1,i} = C_{m,i+1} - D_{m,i} \tag{3.16}$$

Na těchto relacích je založena rutina pro racionální interpolaci.

Rutina pro racionální interpolaci (volání analogické jako u `polint`) (C) [Press et al., 1997a]
`void ratint(float xa[], float ya[], int n, float x, float *y, float *dy)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE ratint(xa, ya, n, x, y, dy)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE ratint(xa, ya, x, y, dy)`
IN: (C: `xa[1..n]`, `ya[1..n]`), (F77: `xa(1:n)`, `ya(1:n)`), (F90: `xa`, `ya`), (C, F77: `n`), `x`.
OUT: `y` a `dy`.
V proměnné `y` vrátí hodnotu interpolační diagonální racionální funkce R, $R($`xa[i]`$)=$`ya[i]`, v bodě `x`.
V proměnné `dy` vrátí odhad chyby.

Typické použití:

```
#define N 100 /* number of data points */
#define NP 4  /* number of points for rational interpolation */
...
float x,y,dy,*xa,*ya;
...
xa=vector(1,N); /* allocate memory for all data */
ya=vector(1,N);
...
ratint(xa,ya,NP,x,&y,&dy); /* thru pts 1,2,3,4, xa[1]<=x<=xa[4] */
ratint(xa+9,ya+9,NP,x,&y,&dy); /* thru pts 10,..,13, xa[10]<=x<=xa[13] */
ratint(xa+96,ya+96,NP,x,&y,&dy); /* thru last 4 pts, xa[97]<=x<=xa[100] */
```

Pro C-programátory: Proč je pro interpolaci přes body $97,\ldots,100$ v rutině `ratint` argument `xa+96`?

Vyzkoušejte demonstrační program pro racionální interpolaci `xratint` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

3.3. Interpolace kubickými splajny

Lineární interpolace mezi N body $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ má tvar po částech lineární funkce (speciální případ Lagrangeovy formule (3.1))

$$y = Ay_j + By_{j+1}, \quad A(x) \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j}, \quad B(x) \equiv 1 - A(x) = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.17}$$

Zřejmě $y'' = 0$ uvnitř každého intervalu $\langle x_j, x_{j+1} \rangle$, ale obecně neexistuje y'' v bodech x_j.

Kubické splajny poskytují interpolační formuli, která má

- hladké první derivace y',
- spojité druhé derivace y'',

a to jak uvnitř každého intervalu $\langle x_j, x_{j+1} \rangle$, tak v hraničních bodech x_j.

3.3. Interpolace kubickými splajny

Lineární interpolace mezi N body $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ má tvar po částech lineární funkce (speciální případ Lagrangeovy formule (3.1))

$$y = Ay_j + By_{j+1}, \quad A(x) \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j}, \quad B(x) \equiv 1 - A(x) = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.17}$$

Zřejmě $y'' = 0$ uvnitř každého intervalu $\langle x_j, x_{j+1} \rangle$, ale obecně neexistuje y'' v bodech x_j.

Kubické splajny poskytují interpolační formuli, která má

- hladké první derivace y',
- spojité druhé derivace y'',

a to jak uvnitř každého intervalu $\langle x_j, x_{j+1} \rangle$, tak v hraničních bodech x_j.

Předpokládejme, že v hraničních bodech x_j jsou kromě y_j zadány také hodnoty druhé derivace y''_j, a přidejme k pravé straně (3.17) v každém intervalu kubický polynom $S(x)$:

$$y(x) = A(x)y_j + B(x)y_{j+1} + S(x), \qquad S(x) = s_0 + s_1 x + s_2 x^2 + s_3 x^3 \tag{3.18}$$

s těmito vlastnostmi:

$$S(x_j) = S(x_{j+1}) = 0 \qquad \text{(souhlas } y(x) \text{ s hodnotami } y_j) \tag{3.19}$$

$$S''(x_j) = y''_j, \quad S''(x_{j+1}) = y''_{j+1} \qquad \text{(souhlas } y''(x) \text{ s hodnotami } y''_j) \tag{3.20}$$

Vztahy (3.19) a (3.20) tvoří čtyři rovnice pro výpočet čtyř koeficientů s_j polynomu $S(x)$.

Po jejich vyřešení dostaneme přepis (3.18) ve tvaru

$$y(x) = A(x)y_j + B(x)y_{j+1} + C(x)y''_j + D(x)y''_{j+1} \tag{3.21}$$

kde koeficienty

$$C(x) \equiv \tfrac{1}{6}(A^3(x) - A(x))(x_{j+1} - x_j)^2 \tag{3.22}$$

$$D(x) \equiv \tfrac{1}{6}(B^3(x) - B(x))(x_{j+1} - x_j)^2 \tag{3.23}$$

obsahují kubickou část závislosti na x. Evidentně $C(x_j) = C(x_{j+1}) = D(x_j) = D(x_{j+1}) = 0$ (díky $A(x_j) = B(x_{j+1}) = 1$, $A(x_{j+1}) = B(x_j) = 0$), čímž jsou splněny (3.19), a snadno ověříme $C''(x) = A(x)$, $D''(x) = B(x)$. Proto

$$y''(x) = A(x)y''_j + B(x)y''_{j+1} \tag{3.24}$$

takže (3.20) jsou také splněny. První derivace je

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

Po jejich vyřešení dostaneme přepis (3.18) ve tvaru

$$y(x) = A(x)y_j + B(x)y_{j+1} + C(x)y''_j + D(x)y''_{j+1} \tag{3.21}$$

kde koeficienty

$$C(x) \equiv \tfrac{1}{6}(A^3(x) - A(x))(x_{j+1} - x_j)^2 \tag{3.22}$$

$$D(x) \equiv \tfrac{1}{6}(B^3(x) - B(x))(x_{j+1} - x_j)^2 \tag{3.23}$$

obsahují kubickou část závislosti na x. Evidentně $C(x_j) = C(x_{j+1}) = D(x_j) = D(x_{j+1}) = 0$ (díky $A(x_j) = B(x_{j+1}) = 1$, $A(x_{j+1}) = B(x_j) = 0$), čímž jsou splněny (3.19), a snadno ověříme $C''(x) = A(x)$, $D''(x) = B(x)$. Proto

$$y''(x) = A(x)y''_j + B(x)y''_{j+1} \tag{3.24}$$

takže (3.20) jsou také splněny. První derivace je

$$y'(x) = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{3.25}$$

Hodnoty y''_j v hraničních bodech však nejsou známy! Použijeme-li však podmínku spojitosti první derivace, pak z rovnosti (3.25) aplikované na intervaly $\langle x_{j-1}, x_j\rangle$ a $\langle x_j, x_{j+1}\rangle$ pro x rovno společnému hraničnímu bodu x_j dostaneme

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \tag{3.26}$$

To je $N - 2$ lineárních rovnic pro N neznámých y''_i, $i = 1, \ldots, N$.

Dvě z nich můžeme proto volit libovolně – obvykle y_1'' a y_N''. Nejčastější volby jsou

- $y_1'' = 0$ a/nebo $y_N'' = 0$, tzv. přirozený kubický splajn, který má nulovou křivost na jednom nebo obou okrajích dat
- na jednom z obou okrajů dat se zadá první derivace y_1' a/nebo y_N' a pomocí (3.25) se dopočítají y_1'' a y_N''.

Soustava (3.26) je (vzhledem k neznámým y_j'') tridiagonální, takže se snadno řeší. Jakmile z (3.26) určíme $N-2$ zbývajících y_j'', stává se (3.21) determinovanou funkcí s garantovanou spojitostí druhé derivace (a tím garantovanou hladkostí první derivace) v celém intervalu $\langle x_1, x_N \rangle$ – splajnem.

Rutina pro výpočet $(N-2)$ 2. derivací y_j'' (C) [Press et al., 1997a]
`void spline(float x[], float y[], int n, float yp1, float ypn, float y2[])`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE spline(x, y, n, yp1, ypn, y2)`
(Fortran 90) [Press et al., 1997c]
`SUBROUTINE spline(x, y, yp1, ypn, y2)`
IN: (C: `x[1..n]`, `y[1..n]`), (F77: `x(1:n)`, `y(1:n)`), (F90: `x`, `y`), (C, F77: `n`), `yp1`, `ypn`.
OUT: (C: `y2[1..n]`), (F77: `y2(1:n)`), (F90: `y2`).
Vektory `x` a `y` obsahují soubor dat (x_j monotónně roste s j), v proměnných `yp1` a `ypn` zadáváme požadovanou hodnotu 1. derivace na okraji souboru dat. Je-li tato hodnota $\geq 1.0 \times 10^{30}$, rutina použije podmínku přirozeného splajnu (2. derivace, tj. křivost, nulová na příslušném konci). Druhé derivace v hraničních bodech jsou zapsány do vektoru `y2`.

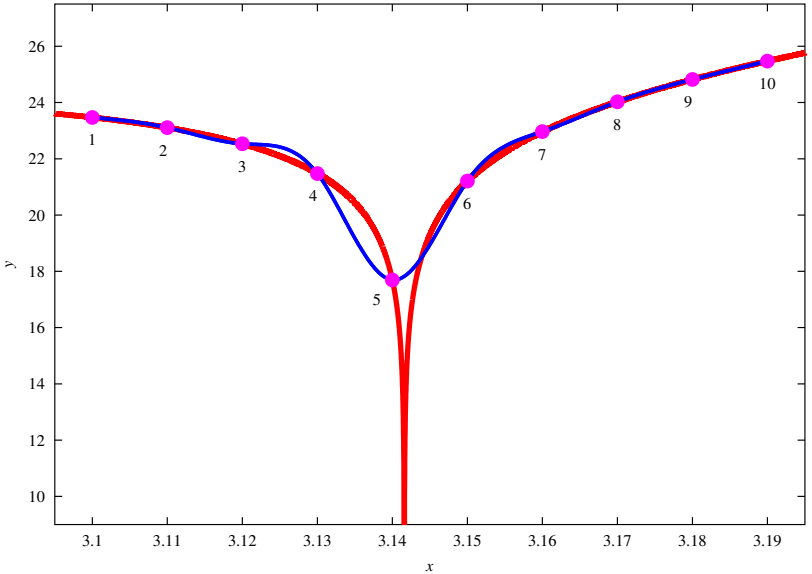
Předchozí rutinu (`spline`) stačí volat jednou. Pro výpočet interpolované hodnoty v bodě x pak voláme rutinu `splint`, jež použije hodnoty `y2[1..n]`:

Rutina pro výpočet kubického splajnu v bodě x (C) [Press et al., 1997a]
`void splint(float xa[], float ya[], float y2a[], int n, float x, float *y)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE splint(xa, ya, y2a, n, x, y)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION splint(xa, ya, y2a, x)`
IN: (C: `xa[1..n]`, `ya[1..n]`, `y2a[1..n]`), (F77: `xa(1:n)`, `ya(1:n)`, `y2a(1:n)`), (F90: `xa`, `ya`, `y2a`), (C, F77: `n`), `x`.
OUT: (C, F77: `y`).
Vektor `y2a` je z rutiny `spline`. Interpolovaná hodnota příslušná hodnotě `x` nezávislé proměnné je vrácena (F90) nebo zapsána do `y` (C, F77).

Typické použití:

```
#define YPRIME1 2.0e30 /* natural spline */
#define NP 10 /* number of points */
...
float x,yn,*xa,*ya,*y2n;
...
xa=vector(1,NP);
ya=vector(1,NP);
y2n=vector(1,NP);
...
spline(xa,ya,NP,YPRIME1,YPRIME1,y2n); /* generates y2n as output */
splint(xa,ya,y2n,NP,x,&yn); /* uses y2n as input */
...
```

Vyzkoušejte demonstrační programy pro interpolaci kubickými splajny `xspline` a `xsplint` z knihovny [Press et al., 1997a, Vetterling et al., 1993].



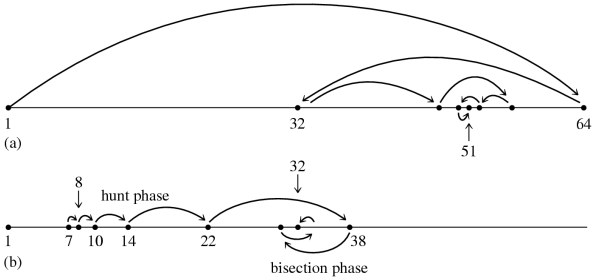
Body $(x_1, y_1), (x_2, y_2), \ldots, (x_{10}, y_{10})$ s ekvidistantními x_i leží na grafu funkce $f(x) = 3x^2 + \frac{1}{\pi^4} \ln\left[(\pi - x)^2\right] + 1$ a jsou interpolovány přirozeným kubickým splajnem.

3.4. Pomocné rutiny

Máme-li data s velkým počtem bodů N, je vhodné použít interpolační schéma (polynomiální, racionální) s menším počtem bodů m, obvykle $m = 4$. Předpokládejme abscisy x_j monotónně rostoucí nebo klesající. Potom musíme řešit následující úkoly:

1. Pro zadané x určit $j = 1, \ldots, N-1$ tak, že $x_j \leq x < x_{j+1}$.
2. Pro takto lokalizované x ošetřit ‚okrajové efekty‘: leží-li x např. v (x_1, x_2) a používáme-li čtyřbodovou interpolaci, vezmeme body $1, 2, 3, 4$. Avšak pro x uprostřed dat daleko od obou konců použijeme ‚dva body vlevo, dva vpravo‘, pro $x \in (x_{N-1}, x_N)$ použijeme body $N-3, N-2, N-1, N$.

První problém je řešen rutinami `locate` a `hunt`. Rutina `locate` řeší problém ab initio pomocí bisekce (a) v $\mathcal{O}(\log_2 N)$ pokusech, rutina `hunt` předřadí bisekci ‚lov‘ startující z předešlé pozice (b) – může být rychlejší faktorem $\log_2 N$, přinejhorším je dvakrát pomalejší.



Rutina pro prohledávání uspořádané tabulky (C) [Press et al., 1997a]
`void locate(float xx[], unsigned long n, float x, unsigned long *j)`
`void hunt(float xx[], unsigned long n, float x, unsigned long *jlo)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE locate(xx, n, x, j)`
`SUBROUTINE hunt(xx, n, x, jlo)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION locate(xx, x)`
`SUBROUTINE hunt(xx, x, jlo)`
IN: (C: `xx[1..n]`), (F77: `xx(1:n)`), (F90: `xx`), (C, F77: `n`), `x`.
OUT: `j` nebo `jlo` (s výjimkou `locate` (F90)).
Je-li zadán (monotónní) vektor `xx` abscis a hodnota `x`, rutina vrátí hodnotu `j` nebo `jlo` tak, že `x` leží mezi x_j a x_{j+1}. Je-li `x` mimo rozsah $\langle x_1, x_n \rangle$, vrátí 0 nebo n.

Máme-li lokalizovánu polohu x v tabulce dat v intervalu $\langle x_j, x_{j+1} \rangle$, určíme nejlevější prvek x_k vstupující do interpolačního procesu (nejpravější bude x_{k+m-1}) takto:

$$k = \min\left\{\max\left\{j - (m-1)/2, 1\right\}, N + 1 - m\right\} \tag{3.27}$$

Vyzkoušejte fungování (3.27) pro hodnoty $N = 20$, $m = 4$ s různými polohami x.

Nezaměňujte interpolaci a fitování:

- Fitování (např. metodou nejmenších čtverců) je vyhlazovací proces – počet koeficientů bývá $\ll$ než počet dat a hodnoty nafitované funkce v x_j nejsou přesné.
- U interpolace je počet koeficientů a dat stejný, takže hodnoty interpolantu v x_j jsou přesné, ale statistické chyby v datech způsobují oscilace.

3.5. Koeficienty interpolačního polynomu

Občas je potřeba znát koeficienty interpolačního polynomu (3.1) procházejícího malým počtem bodů. (Připomeňme, že při proceduře určování hodnoty interpolačního polynomu pomocí Nevilleova algoritmu jeho koeficienty znát nepotřebujeme!) Určování koeficientů interpolačního polynomu je vhodné se vyhnout, protože

- koeficienty mohou být určeny mnohem méně přesně než hodnota interpolačního polynomu v zadaném x,
- hodnoty v zadaném x počítané ze známých koeficientů neprochází přesně tabelovanými body.

V odůvodněných případech, jako například

- současný výpočet interpolovaných hodnot funkce a jejích derivací,
- výpočet konvoluce segmentu tabelované funkce s nějakou funkcí, jejíž momenty (tj. její konvoluce s mocninami x) jsou známy analyticky.

Pro určení $N+1$ koeficientů interpolačního polynomu

$$y = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N \tag{3.28}$$

potřebujeme $N+1$ bodů $(x_0, y_0), \ldots, (x_N, y_N)$:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix} \tag{3.29}$$

To je Vandermondeova matice, pro jejíž řešení existuje speciální metoda s počtem operací $\mathcal{O}(N^2)$. Vandermondeův systém může být numericky špatně definován – žádné numerická metoda nedá přesný výsledek, na rozdíl od určení interpolované hodnoty.

Druhá metoda využívá rutiny polint: inter/extrapolujeme pro $x = 0$, čímž dostaneme c_0. Odečteme c_0 od všech y_j a dělíme každé y_j odpovídajícím x_j. Zahodíme jeden bod (nejlépe s nejmenším x_j), a inter/extrapolací pro $x = 0$ dostaneme c_1, atd. Poněkud stabilnější než předchozí metoda, ale $\mathcal{O}(N^3)$. Jinak podobné problémy jako u Vandermondeovy metody. Rozumné výsledky v single precision do $N \sim 8$–10, v double precision do $N \sim 15$–20.

Rutina `polcoe` implementuje Vandermondeovu metodu, rutina `polcof` druhou metodu.

Rutina pro určení koeficientů interpolačního polynomu (C) [Press et al., 1997a]
`void polcoe(float x[], float y[], int n, float cof[])`
`void polcof(float x[], float y[], int n, float cof[])`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE polcoe(x, y, n, cof)`
`SUBROUTINE polcof(x, y, n, cof)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION polcoe(x, y)`
`FUNCTION polcof(x, y)`
IN: (C: `x[0..n]`, `y[0..n]`), (F77: `x(1:n)`, `y(1:n)`), (F90: `x`, `y`), (C, F77: `n`).
OUT: (C: `cof[0..n]`), (F77: `cof(1:n)`).
DEP: `polcof`↔polint
Z vektorů `x` a `y` obsahujících tabelované hodnoty (x_j, y_j) vypočte a vrátí vektor koeficientů interpolačního polynomu (F90) nebo je zapíše do vektoru `cof` (C, F77).

Vyzkoušejte demonstrační programy pro prohledávání uspořádané tabulky `xlocate` a `xhunt` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Vyzkoušejte demonstrační programy pro určení koeficientů interpolačního polynomu `xpolcoe` a `xpolcof` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Shrnutí kapitoly: Naučili jste se interpolovat (nebo extrapolvat) pomocí různých metod a rozumně tyto metody využívat – víte, kdy která dává užitečné výsledky a kdy je naopak kontraproduktivní.

Další zdroje:

- Kapitola 3 v [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c]. Pro jazyk C dostupné na http://www.library.cornell.edu/nr/bookcpdf/, pro Fortran na http://www.library.cornell.edu/nr/bookfpdf/.
- Úvod do Fortranu 95 a do numerických metod [Sandu, 2001]. Obsahuje Kapitoly 18 a 20 o polynomiální interpolaci a splajnech. Dostupné online na http://www.cs.mtu.edu/~asandu/Courses/CS2911/fortran_notes/main.html.

Úlohy ke kapitole 3

Uvažujte Rungeovu funkci

$$r(x) = \frac{1}{1 + 10x^2} \tag{3.30}$$

na definičním oboru $\langle -1, 1 \rangle$, a $N = 9$ Čebyševových bodů

$$x_j = -\cos\left[\frac{(2j-1)\pi}{2N}\right], \qquad y_j = r(x_j), \qquad j = 1, \ldots, N \tag{3.31}$$

1. Interpolujte body $(x_j, y_j), j = 1, \ldots, N$ na intervalu $\langle -1, 1 \rangle$ postupně dvoubodovou polynomiální interpolací (tj. po částech lineární funkcí), tříbodovou polynomiální interpolací (tj. kvadratickými polynomy), čtyřbodovou polynomiální interpolací (tj. kubickými polynomy), ..., osmibodovou polynomiální interpolací, devítibodovou polynomiální interpolací.
2. Interpolujte body $(x_j, y_j), j = 1, \ldots, N$ na intervalu $\langle -1, 1 \rangle$ postupně dvoubodovou, tříbodovou, ..., devítibodovou racionální interpolací.
3. Interpolujte body $(x_j, y_j), j = 1, \ldots, N$ na intervalu $\langle -1, 1 \rangle$ přirozeným kubickým splajnem.
4. Interpolujte body $(x_j, y_j), j = 1, \ldots, N$ na intervalu $\langle -1, 1 \rangle$ kubickým splajnem s prvními derivacemi v x_1 a x_N nastavenými na hodnoty analyticky spočtené derivace (3.30) v těchto bodech.
5. Diskutujte získané výsledky.

4. Numerická integrace

Rychlý náhled kapitoly: V této kapitole budou vysvětleny principy další ze základních úloh numerického modelování: numerické kvadratury. Existuje mnoho různých přístupů, zde se budeme věnovat jen těm nejzákladnějším jako jsou lichoběžníkové (trapezoidální) pravidlo, Rombergova kvadratura a dotkneme se způsobu numerického výpočtu nevlastních integrálů.

Cíle kapitoly:

- Naučit se numericky počítat určité integrály pomocí lichoběžníkového pravidla.
- Naučit se numericky počítat určité integrály pomocí Simpsonova pravidla.
- Naučit se numericky počítat určité integrály Rombergovou metodou.
- Naučit se numericky počítat některé typy nevlastních určitých integrálů.

Klíčová slova kapitoly: Kvadratura; uzavřené formule, otevřené formule; lichoběžníkové pravidlo, Simpsonovo pravidlo; otevřené extrapolace; rozšířené uzavřené formule; rozšířené otevřené a polootevřené formule; Rombergova kvadratura; nevlastní integrály

Motto: *Metoda brutální síly: (1) nakresli graf funkce na ocelový plech známé tloušťky a vystřihni, (2) zvaž vystřižený kus, (3) poděl hmotnost hustotou oceli a tloušťkou plechu, (4) raduj se z výsledku.*

Historie numerické integrace (kvadratury) sahá k objevu diferenciálního a integrálního počtu. Na rozdíl od derivací elementárních funkcí a jejich kombinací, jež lze vždy derivovat analyticky, integrály obecně analyticky vyjádřit nelze.

Vyčíslení určitého integrálu

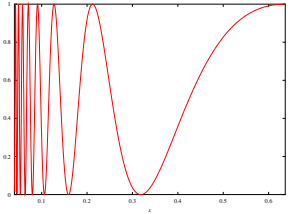
$$\mathcal{I} = \int_a^b f(x)\,\mathrm{d}x \tag{4.1}$$

je ekvivalentní řešení diferenciální rovnice

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x) \tag{4.2}$$

s počáteční podmínkou $y(a) = 0$ a dosazení $\mathcal{I} \equiv y(b)$.

K numerické kvadratuře lze proto použít metody řešení obyčejných diferenciálních rovnic, jež jsou vhodné pro kvadraturu funkcí koncentrovaných do jednoho nebo více ‚píků‘, nebo funkcí, jejichž tvar není charakterizován jedinou délkovou škálou, jako např. $f(x) = \sin^2(x^{-1})$ na intervalu $\langle \delta, 2/\pi \rangle$, kde δ je malé kladné číslo.



Kromě metod této kapitoly a pomocí ODE jsou další metody kvadratury:

- Metody založené na aproximaci funkcí. Viz kapitola 5 v [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c]. Explicitní diskuse integrace funkcí pomocí Čebyševovy aproximace je v sekci 5.9 uvedené knihy.
- Analytická integrace kubického splajnu (3.21) s použitím výstupu rutiny `spline` pro y''_j [Forsythe et al., 1977].
- Integrály konvolučního typu lze počítat pomocí metod rychlé Fourierovy transformace v kapitole 5.
- Důležitá metoda výpočtu multidimenzionálních integrálů je metoda Monte Carlo.

4.1. Klasické formule pro ekvidistantní abscisy

Mějme ekvidistantní posloupnost abscis $x_0, x_1, \ldots, x_N, x_{N+1}$ s krokem h,

$$x_j = x_0 + jh, \qquad j = 0, 1, \ldots, N, N+1 \tag{4.3}$$

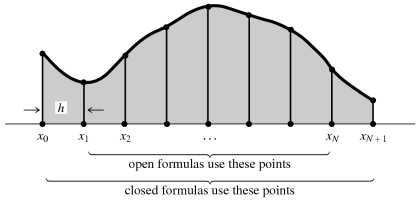
Funkční hodnoty v nich označíme $f(x_j) \equiv f_j$. Úkolem je numericky spočítat určitý integrál

$$\int_a^b f(x)\,\mathrm{d}x \tag{4.4}$$

kde $a = x_0$ a $b = x_{N+1}$.

Uzavřené formule používají funkční hodnoty v koncových bodech $f(a)$ a $f(b)$.

Otevřené formule aproximují (4.4) pomocí x_j ležících striktně mezi a a b (např. f má v a a/nebo b singularitu, nelze vyčíslit ap.). Blíže viz sekci 4.4.



Formule pro integraci funkce přes malý počet intervalů prezentované v následujících dvou subsekcích 4.1.1 a 4.1.2 jsou základní stavební bloky pro konstrukci rozšířených formulí používajících větší počet intervalů (subsekce 4.1.3 a 4.1.4).

4.1.1. Uzavřené Newtonovy–Cotesovy formule

Obecný tvar n-bodové uzavřené Newtonovy–Cotesovy formule je

$$\int_{x_1}^{x_n} f(x)\,\mathrm{d}x = h\left[\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n\right] + \mathcal{O}(h^{m+2} f^{(m+1)}(\xi)) \tag{4.5}$$

kde α_j, $j = 1, \ldots, n$ jsou váhové koeficienty a $m \geq n-1$ je nejvyšší stupeň polynomu, pro který (4.5) platí exaktně. Notace $\mathcal{O}(h^{m+2} f^{(m+1)}(\xi))$ označuje, že pro jiné funkce než $f(x) \equiv P_0(x), \ldots, P_m(x)$ se exaktní výsledek od vypočteného liší o součin numerického koeficientu (vyčíslitelného!) $\times\, h^{m+2} \times$ hodnota $(m+1)$-derivace f v některém bodě (neznámém!) $\xi \in (x_1, x_n)$.

4.1.1. Uzavřené Newtonovy–Cotesovy formule

Obecný tvar n-bodové uzavřené Newtonovy–Cotesovy formule je

$$\int_{x_1}^{x_n} f(x)\,\mathrm{d}x = h\left[\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n\right] + \mathcal{O}(h^{m+2} f^{(m+1)}(\xi)) \tag{4.5}$$

kde α_j, $j = 1, \ldots, n$ jsou váhové koeficienty a $m \geq n-1$ je nejvyšší stupeň polynomu, pro který (4.5) platí exaktně. Notace $\mathcal{O}(h^{m+2} f^{(m+1)}(\xi))$ označuje, že pro jiné funkce než $f(x) \equiv P_0(x), \ldots, P_m(x)$ se exaktní výsledek od vypočteného liší o součin numerického koeficientu (vyčíslitelného!) $\times\, h^{m+2} \times$ hodnota $(m+1)$-derivace f v některém bodě (neznámém!) $\xi \in (x_1, x_n)$.

Hodnoty koeficientů α_j se jednoduše odvodí tak, že položíme (bez újmy na obecnosti) $x_1 = 0$, takže $x_j = (j-1)h$, do n-bodové formule (4.5) dosadíme za $f(x)$ postupně n polynomů $f(x) = 1, x/h, (x/h)^2, \ldots, (x/h)^{n-1}$ a vyčíslíme $\int_0^{(n-1)h} f(x)\,\mathrm{d}x$ a f_j. Dostaneme tak pro n koeficientů $\alpha_1, \ldots, \alpha_n$ soustavu n lineárních algebraických rovnic.

Díky náhodnému zkrácení však může (4.5) platit exaktně i pro polynom stupně $m > n-1$. V případě obecné funkce použijeme Taylorův polynom se středem v x_1, $f(x) = P_m(x) + f^{(m+1)}(\xi)x^{m+1}/(m+1)!$, kde $\xi \in (x_1, x_n)$. Integrál polynomiální části se zkrátí s $h[\ldots]$, integrál zbytku dá $(n-1)^{m+2} h^{m+2} f^{(m+1)}(\xi)/(m+2)!$. To je chybový člen na konci formule (4.5) včetně zmíněného koeficientu.

Odvoďte klasické kvadraturní formule uvedené na následující straně položením po řadě $n = 2, 3, 4, 5$ v obecném vzorci (4.5).

Dvoubodové lichoběžníkové (trapezoidální) pravidlo

$$\int_{x_1}^{x_2} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{2}f_1 + \tfrac{1}{2}f_2\right] + \mathcal{O}(h^3 f''(\xi)) \tag{4.6}$$

Exaktní pro polynomy do stupně $m = 1 = n - 1$: lineární funkce.

Dvoubodové lichoběžníkové (trapezoidální) pravidlo

$$\int_{x_1}^{x_2} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{2}f_1 + \tfrac{1}{2}f_2\right] + \mathcal{O}(h^3 f''(\xi)) \tag{4.6}$$

Exaktní pro polynomy do stupně $m = 1 = n - 1$: lineární funkce.

Tříbodové Simpsonovo pravidlo

$$\int_{x_1}^{x_3} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{1}{3}f_3\right] + \mathcal{O}(h^5 f^{(4)}(\xi)) \tag{4.7}$$

Exaktní až po kubické polynomy včetně ($m = 3 > 2 = n - 1$).

Čtyřbodové Simpsonovo 3/8 pravidlo

$$\int_{x_1}^{x_4} f(x)\,\mathrm{d}x = h\left[\tfrac{3}{8}f_1 + \tfrac{9}{8}f_2 + \tfrac{9}{8}f_3 + \tfrac{3}{8}f_4\right] + \mathcal{O}(h^5 f^{(4)}(\xi)) \tag{4.8}$$

Exaktní až po kubické polynomy včetně ($m = 3 = n - 1$).

Dvoubodové lichoběžníkové (trapezoidální) pravidlo

$$\int_{x_1}^{x_2} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{2}f_1 + \tfrac{1}{2}f_2\right] + \mathcal{O}(h^3 f''(\xi)) \tag{4.6}$$

Exaktní pro polynomy do stupně $m = 1 = n - 1$: lineární funkce.

Tříbodové Simpsonovo pravidlo

$$\int_{x_1}^{x_3} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{1}{3}f_3\right] + \mathcal{O}(h^5 f^{(4)}(\xi)) \tag{4.7}$$

Exaktní až po kubické polynomy včetně ($m = 3 > 2 = n - 1$).

Čtyřbodové Simpsonovo 3/8 pravidlo

$$\int_{x_1}^{x_4} f(x)\,\mathrm{d}x = h\left[\tfrac{3}{8}f_1 + \tfrac{9}{8}f_2 + \tfrac{9}{8}f_3 + \tfrac{3}{8}f_4\right] + \mathcal{O}(h^5 f^{(4)}(\xi)) \tag{4.8}$$

Exaktní až po kubické polynomy včetně ($m = 3 = n - 1$).

Pětibodové Bodeho pravidlo

$$\int_{x_1}^{x_5} f(x)\,\mathrm{d}x = h\left[\tfrac{14}{45}f_1 + \tfrac{64}{45}f_2 + \tfrac{24}{45}f_3 + \tfrac{64}{45}f_4 + \tfrac{14}{45}f_5\right] + \mathcal{O}(h^7 f^{(6)}(\xi)) \tag{4.9}$$

Exaktní až po polynomy stupně 5 včetně ($m = 5 > 4 = n - 1$). Další vícebodové formule viz §25.4 v [Abramowitz a Stegun, 1964].

4.1.2. Otevřené extrapolace pro jeden interval

Místo na koncích otevřených elementárních Newtonových–Cotesových formulí typu $\int_{x_0}^{x_{n+1}} f(x)\,\mathrm{d}x = h[\alpha_1 f_1 + \cdots + \alpha_n f_n] + \mathcal{O}(\ldots)$, které se (a) obtížně zřetězují do rozšířených formulí a (b) pro ostatní použití jsou překonány Gaussovou kvadraturou, uveďme formule extrapolující integrál přes interval $\langle x_0, x_1\rangle$ pomocí funkčních hodnot v $x_1, x_2, \ldots, x_n$:

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = h\left[\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n\right] + \mathcal{O}(h^{m+2} f^{(m+1)}(\xi)) \tag{4.10}$$

Komentář a důkaz je analogický jako v případě uzavřených Newtonových–Cotesových formulí s těmito rozdíly: $\xi \in (x_0, x_1)$, $x_j = jh$, vyčíslujeme integrál $\int_0^h f(x)\,\mathrm{d}x$, a chybový člen je $h^{m+2} f^{(m+1)}(\xi)/(m+2)!$. Volbou n postupně dostaneme:

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = h[f_1] + \mathcal{O}(h^2 f'(\xi)) \tag{4.11}$$

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = h[\tfrac{3}{2} f_1 - \tfrac{1}{2} f_2] + \mathcal{O}(h^3 f''(\xi)) \tag{4.12}$$

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = h[\tfrac{23}{12} f_1 - \tfrac{16}{12} f_2 + \tfrac{5}{12} f_3] + \mathcal{O}(h^4 f^{(3)}(\xi)) \tag{4.13}$$

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = h[\tfrac{55}{24} f_1 - \tfrac{59}{24} f_2 + \tfrac{37}{24} f_3 - \tfrac{9}{24} f_4] + \mathcal{O}(h^5 f^{(4)}(\xi)) \tag{4.14}$$

Pro tyto formule je nejvyšší stupeň polynomu m, pro který platí exaktně, roven postupně $m = 0, 1, 2, 3$.

Odvoďte extrapolační formule (4.11)–(4.14) položením po řadě $n = 1, 2, 3, 4$ v obecném vzorci (4.10).

4.1.3. Rozšířené uzavřené formule

Dostaneme je většinou zřetězením uzavřených Newtonových–Cotesových formulí na po sobě jdoucích, nepřekrývajících se intervalech (páru intervalů, trojici intervalů, ...) pro $a = x_1, x_2, x_3, \ldots, x_{N-1}, x_N = b$. Krok je $h = (b-a)/N$, a počet takových multiintervalů je $\sim N$, takže chybový člen je úměrný

$$\sum_{j=1}^{\sim N} \mathcal{O}\left(\frac{(b-a)^{m+2}}{N^{m+2}} f^{(m+1)}(\xi_j)\right) \sim \mathcal{O}\left(\frac{(b-a)^{m+2}}{N^{m+1}} f^{(m+1)}(\xi)\right) \tag{4.15}$$

kde $\xi \in (x_1, x_N)$ je hodnota maxima $(m+1)$-derivace přes jednotlivé multiintervaly. Protože $b - a$ je obvykle konstantní a zajímá nás hlavně závislost chyby na počtu bodů, budeme psát chybový člen zjednodušeně

$$\mathcal{O}\left(\frac{1}{N^{m+1}}\right)$$

4.1.3. Rozšířené uzavřené formule

Dostaneme je většinou zřetězením uzavřených Newtonových–Cotesových formulí na po sobě jdoucích, nepřekrývajících se intervalech (páru intervalů, trojici intervalů, ...) pro $a = x_1, x_2, x_3, \ldots, x_{N-1}, x_N = b$. Krok je $h = (b-a)/N$, a počet takových multiintervalů je $\sim N$, takže chybový člen je úměrný

$$\sum_{j=1}^{\sim N} \mathcal{O}\left(\frac{(b-a)^{m+2}}{N^{m+2}} f^{(m+1)}(\xi_j)\right) \sim \mathcal{O}\left(\frac{(b-a)^{m+2}}{N^{m+1}} f^{(m+1)}(\xi)\right) \tag{4.15}$$

kde $\xi \in (x_1, x_N)$ je hodnota maxima $(m+1)$-derivace přes jednotlivé multiintervaly. Protože $b - a$ je obvykle konstantní a zajímá nás hlavně závislost chyby na počtu bodů, budeme psát chybový člen zjednodušeně

$$\mathcal{O}\left(\frac{1}{N^{m+1}}\right) \tag{4.16}$$

Rozšířené lichoběžníkové (trapezoidální) pravidlo

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{N-1} + \tfrac{1}{2}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^2}\right) \tag{4.17}$$

Exaktní pro polynomy do stupně $m = 1$: po částech lineární funkce.

Rozšířené Simpsonovo pravidlo

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{2}{3}f_3 + \tfrac{4}{3}f_4 + \cdots + \tfrac{2}{3}f_{N-2} + \tfrac{4}{3}f_{N-1} + \tfrac{1}{3}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^4}\right) \tag{4.18}$$

Exaktní až po ‚po částech kubické polynomy' včetně ($m = 3$).

Rozšířená formule řádu $1/N^3$

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{5}{12}f_1 + \tfrac{13}{12}f_2 + \sum_{k=3}^{N-2} f_k + \tfrac{13}{12}f_{N-1} + \tfrac{5}{12}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^3}\right) \tag{4.19}$$

Exaktní až po ‚po částech kvadratické polynomy' včetně ($m = 2$).

Dokažte (4.19) aritmetickým zprůměrováním rozšířeného Simpsonova pravidla (4.18) a modifikace rozšířeného Simpsonova pravidla, na jejíž první a poslední krok je použito lichoběžníkové pravidlo (4.6). Chybový člen je součtem chybových členů obou elementárních pravidel násobených počtem intervalů, na nichž jsou použity, $\sim 2 \times \mathcal{O}(h^3 f'') + N \times \mathcal{O}(h^5 f^{(4)}) \sim \mathcal{O}(1/N^3)$.

Rozšířené Simpsonovo pravidlo

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{2}{3}f_3 + \tfrac{4}{3}f_4 + \cdots + \tfrac{2}{3}f_{N-2} + \tfrac{4}{3}f_{N-1} + \tfrac{1}{3}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^4}\right) \tag{4.18}$$

Exaktní až po ‚po částech kubické polynomy' včetně ($m = 3$).

Rozšířená formule řádu $1/N^3$

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{5}{12}f_1 + \tfrac{13}{12}f_2 + \sum_{k=3}^{N-2} f_k + \tfrac{13}{12}f_{N-1} + \tfrac{5}{12}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^3}\right) \tag{4.19}$$

Exaktní až po ‚po částech kvadratické polynomy' včetně ($m = 2$).

Dokažte (4.19) aritmetickým zprůměrováním rozšířeného Simpsonova pravidla (4.18) a modifikace rozšířeného Simpsonova pravidla, na jejíž první a poslední krok je použito lichoběžníkové pravidlo (4.6). Chybový člen je součtem chybových členů obou elementárních pravidel násobených počtem intervalů, na nichž jsou použity, $\sim 2 \times \mathcal{O}(h^3 f'') + N \times \mathcal{O}(h^5 f^{(4)}) \sim \mathcal{O}(1/N^3)$.

Další rozšířená formule řádu $1/N^4$

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{3}{8}f_1 + \tfrac{7}{6}f_2 + \tfrac{23}{24}f_3 + f_4 + f_5 + \cdots + f_{N-4} + f_{N-3} + \tfrac{23}{24}f_{N-2} + \tfrac{7}{6}f_{N-1} + \tfrac{3}{8}f_N\right] + \mathcal{O}\left(\tfrac{1}{N^4}\right) \tag{4.20}$$

Dokáže se fitováním kubických polynomů na posloupnost po sobě jdoucích skupin čtyř bodů; podrobnosti viz §18.3 v [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c].

4.1.4. Rozšířené otevřené a polootevřené formule

Jsou konstruovány ze zřetězených uzavřených Newtonových–Cotesových formulí na jednom nebo obou koncích modifikovaných otevřenými extrapolacemi (po příslušném přečíslování). Z cvičení pod rovnicí (4.19) plyne konzistence ‚vnitřní' uzavřené formule s ‚okrajovou' extrapolativní otevřenou formulí o jednotku nižšího řádu.

- Kombinace (4.11) a (4.17) dává

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\tfrac{3}{2}f_2 + f_3 + \cdots + f_{N-2} + \tfrac{3}{2}f_{N-1}\right] + \mathcal{O}\left(\tfrac{1}{N^2}\right) \tag{4.21}$$

- Kombinace (4.12) a (4.19) dává

$$\begin{aligned}\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\Big[&\tfrac{23}{12}f_2 + \tfrac{7}{12}f_3 + f_4 + f_5 + \cdots \\ &+ f_{N-4} + f_{N-3} + \tfrac{7}{12}f_{N-2} + \tfrac{23}{12}f_{N-1}\Big] + \mathcal{O}\left(\tfrac{1}{N^3}\right)\end{aligned} \tag{4.22}$$

- Kombinace (4.13) a (4.18) dává

$$\begin{aligned}\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\Big[&\tfrac{27}{12}f_2 + 0 + \tfrac{13}{12}f_4 + \tfrac{4}{3}f_5 + \tfrac{2}{3}f_6 + \cdots \\ &+ \tfrac{2}{3}f_{N-5} + \tfrac{4}{3}f_{N-4} + \tfrac{13}{12}f_{N-3} + 0 + \tfrac{27}{12}f_{N-1}\Big] + \mathcal{O}\left(\tfrac{1}{N^4}\right)\end{aligned} \tag{4.23}$$

- Kombinace (4.14) a (4.20) dává

$$\begin{aligned}\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\Big[&\tfrac{55}{24}f_2 - \tfrac{1}{6}f_3 + \tfrac{11}{8}f_4 + f_5 + f_6 + f_7 + \cdots \\ &+ f_{N-5} + f_{N-4} + \tfrac{11}{8}f_{N-3} - \tfrac{1}{6}f_{N-2} + \tfrac{55}{24}f_{N-1}\Big] + \mathcal{O}\left(\tfrac{1}{N^4}\right)\end{aligned} \tag{4.24}$$

- Důležitá je rozšířená středová formule (*extended midpoint rule*):

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h[f_{3/2} + f_{5/2} + \cdots + f_{N-3/2} + f_{N-1/2}] + \mathcal{O}\left(\tfrac{1}{N^2}\right) \tag{4.25}$$

Důkaz je jednoduchý: $\int_{x_j}^{x_{j+1}} f(x)\,\mathrm{d}x$, $j = 1, \ldots, N-1$ aproximujeme hodnotou $f_{j+1/2}$ uprostřed intervalu $\times\, h$. Každá taková aproximace je přesná až do lineární funkce včetně, tj. chybové členy odpovídají lichoběžníkovému pravidlu (4.6) a celkový chybový člen (4.25) rozšířenému lichoběžníkovému pravidlu (4.17).

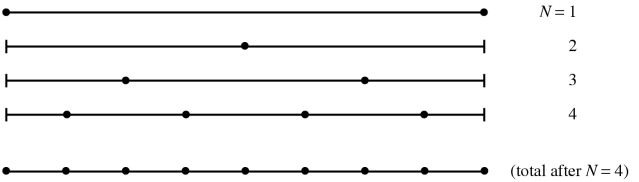
Dokažte rozšířenou středovou formuli (4.25).

Polootevřené formule vzniknou kombinací uzavřených Newtonových–Cotesových formulí s předchozími formulemi (4.21) až (4.24). Na uzavřeném konci použijeme váhy z první sady, na otevřeném konci z druhé sady formulí.

- Příklad polootevřené formule: kombinace (4.12) na levém konci a (4.19) dává

$$\begin{aligned}\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\,\big[&\tfrac{23}{12}f_2 + \tfrac{7}{12}f_3 + f_4 + f_5 + \cdots \\ &+ f_{N-2} + \tfrac{13}{12}f_{N-1} + \tfrac{5}{12}f_N\big] + \mathcal{O}\left(\tfrac{1}{N^3}\right)\end{aligned} \tag{4.26}$$

Ilustrace k sekvenčnímu volání rutiny `trapzd`. ⟶



4.2. Elementární algoritmy

Výchozí rovnice: rozšířené lichoběžníkové pravidlo (4.17). Důvod: pro fixní funkci $f(x)$ a meze a a b můžeme zdvojnásobit počet intervalů bez ztráty předchozích výsledků. Nejhrubější odhad je s $N = 2$ ($[f(a) + f(b)]/2$), 1. krok vede ke zjemnění zavedením $f(\frac{a+b}{2})$, 2. krok přidává ‚$1/4$' a ‚$3/4$' body, atd. Proces je schematicky zachycen na předchozí straně.

Tato rutina počítá n-tý stupeň zjemnění rozšířeného lichob. pravidla (C) [Press et al., 1997a]
`float trapzd(float (*func)(float), float a, float b, int n)`
(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]
`SUBROUTINE trapzd(func, a, b, s, n)`
IN: `func, a, b, n.`
OUT: (F77, F90: `s`).
Když je volána s $n = 1$, vrátí nejhrubší odhad $[f(a) + f(b)]/2$. Následná volání s $n = 2, 3, \ldots$ zlepšují přesnost přidáním 2^{n-2} vnitřních bodů.

Typické použití je `for(j=1;j<=m+1;j++) s=trapzd(func,a,b,j);`. Lepší je ale:

Rutina pro výpočet určitého integrálu lichoběžníkovým pravidlem (C) [Press et al., 1997a]
`float qtrap(float (*func)(float), float a, float b)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE qtrap(func, a, b, s)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION qtrap(func, a, b)`
IN: `func, a, b,` vnitřní proměnné (pro C preprocesorové konstanty) `EPS=1.0e-5` a `JMAX=20`.
OUT: (F77: `s`).
DEP: `qtrap`↩`trapzd`
Vrátí integrál funkce `func` od `a` do `b`. Parametr `EPS` se nastaví na požadovanou relativní přesnost (ne méně než 10^{-6} pro single prec.), $2^{\texttt{JMAX}-1}$ je maximální povolený počet zjemňovacích kroků.

Trik, jak z rozšířeného lichoběžníkového pravidla (4.17) řádu $1/N^2$ udělat rozšířené Simpsonovo pravidlo (4.18) řádu $1/N^4$: Eulerova–Maclaurinova sumační formule má tvar

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{N-1} + \frac{1}{2}f_N\right] - \frac{B_2 h^2}{2!}(f'_N - f'_1) - \cdots - \frac{B_{2k} h^{2k}}{(2k)!}(f_N^{(2k-1)} - f_1^{(2k-1)}) - \cdots \tag{4.27}$$

kde B_{2k} jsou Bernoulliova čísla definovaná generující funkcí

$$\frac{t}{\mathrm{e}^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \tag{4.28}$$

V (4.27) se vyskytují jen sudé mocniny h, na rozdíl od ostatních formulí v subsekci 4.1.3. Vezměme výsledky dvou po sobě následujících volání rutiny `trapzd` jako v typickém použití na předchozí straně. Tím se počet intervalů N zdvojnásobí na $2N$, h se změní na $h/2$, a rutina vrátí výsledky S_N a S_{2N}. Vedoucí (kvadratický) člen chybové expanze v (4.27) se změní faktorem $1/4$. V kombinaci

$$S \equiv \tfrac{4}{3}S_{2N} - \tfrac{1}{3}S_N \tag{4.29}$$

se proto tento člen zruší! Zůstane člen řádu $h^4 \sim 1/N^4$, stejný jako u (4.18). Snadno se ověří, že (4.29) je ve skutečnosti Simpsonovo rozšířené pravidlo (4.18):

$$
\begin{aligned}
-\tfrac{1}{3}S_N &= -\tfrac{2}{3} \times \tfrac{h}{2} \times [\tfrac{1}{2}f_1 \quad + \quad f_3 \quad + \quad f_5 + \cdots + \quad f_{2N-2} \quad + \tfrac{1}{2}f_{2N}] \\
\tfrac{4}{3}S_{2N} &= \quad \tfrac{4}{3} \times \tfrac{h}{2} \times [\tfrac{1}{2}f_1 + \quad f_2 + \quad f_3 + \quad f_4 + \quad f_5 + \cdots + \quad f_{2N-2} + \quad f_{2N-1} + \tfrac{1}{2}f_{2N}] \\
S &= \qquad \tfrac{h}{2} \times [\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{2}{3}f_3 + \tfrac{4}{3}f_4 + \tfrac{2}{3}f_5 + \cdots + \tfrac{2}{3}f_{2N-2} + \tfrac{4}{3}f_{2N-1} + \tfrac{1}{3}f_{2N}]
\end{aligned}
$$

$$
\begin{aligned}
-\tfrac{1}{3}S_N &= -\tfrac{2}{3} \times \tfrac{h}{2} \times [\tfrac{1}{2}f_1 \quad + \quad f_3 \quad + \quad f_5 + \cdots + \quad f_{2N-2} \quad + \tfrac{1}{2}f_{2N}] \\
\tfrac{4}{3}S_{2N} &= \quad \tfrac{4}{3} \times \tfrac{h}{2} \times [\tfrac{1}{2}f_1 + \quad f_2 + \quad f_3 + \quad f_4 + \quad f_5 + \cdots + \quad f_{2N-2} + \quad f_{2N-1} + \tfrac{1}{2}f_{2N}] \\
S &= \qquad \tfrac{h}{2} \times [\tfrac{1}{3}f_1 + \tfrac{4}{3}f_2 + \tfrac{2}{3}f_3 + \tfrac{4}{3}f_4 + \tfrac{2}{3}f_5 + \cdots + \tfrac{2}{3}f_{2N-2} + \tfrac{4}{3}f_{2N-1} + \tfrac{1}{3}f_{2N}]
\end{aligned}
$$

Rutina pro výpočet určitého integrálu Simpsonovým pravidlem (C) [Press et al., 1997a]
`float qsimp(float (*func)(float), float a, float b)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE qsimp(func, a, b, s)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION qsimp(func, a, b)`
IN: `func`, `a`, `b`, vnitřní proměnné (pro C preprocesorové konstanty) `EPS=1.0e-6` a `JMAX=20`.
OUT: (F77: `s`).
DEP: `qsimp`↩`trapzd`
Vrátí integrál funkce `func` od `a` do `b`. Parametr `EPS` se nastaví na požadovanou relativní přesnost (ne méně než 10^{-6} pro single prec.), $2^{\texttt{JMAX}-1}$ je maximální povolený počet zjemňovacích kroků.

Vyzkoušejte demonstrační program pro n-té zjemnění integrace pomocí lichoběžníkového pravidla `xtrapzd` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Vyzkoušejte demonstrační program pro integraci pomocí rozšířeného lichoběžníkového pravidla `xqtrap` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Vyzkoušejte demonstrační program pro integraci pomocí rozšířeného Simpsonova pravidla `xqsimp` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

4.3. Rombergova kvadratura

Idea 1 Rozšířené Simpsonovo pravidlo (4.18) lze získat z rozšířeného lichoběžníkového pravidla (4.17) pomocí triku (4.29), který odstraní z (4.27) vedoucí chybový člen $\mathcal{O}(1/N^2) \sim \mathcal{O}(h^2)$. Zkusme aplikovat takové zjemnění $(k-1)$-krát za sebou a odstranit tak z chybové řady (4.27) členy až do, ale nezahrnující $\mathcal{O}(1/N^{2k})$; rozšířené Simpsonovo pravidlo (4.18) pak představuje speciální případ $k = 2$.

4.3. Rombergova kvadratura

Idea 1 Rozšířené Simpsonovo pravidlo (4.18) lze získat z rozšířeného lichoběžníkového pravidla (4.17) pomocí triku (4.29), který odstraní z (4.27) vedoucí chybový člen $\mathcal{O}(1/N^2) \sim \mathcal{O}(h^2)$. Zkusme aplikovat takové zjemnění $(k-1)$-krát za sebou a odstranit tak z chybové řady (4.27) členy až do, ale nezahrnující $\mathcal{O}(1/N^{2k})$; rozšířené Simpsonovo pravidlo (4.18) pak představuje speciální případ $k = 2$.

Idea 2 Richardsonova extrapolace (*Richardson's deferred approach to the limit*): provést určitý numerický algoritmus pro různé (klesající) hodnoty parametru h, a potom extrapolovat výsledky pro $h = 0$; v tomto případě provést několik postupných zjemnění intervalu (např. $k = 5$) a pro extrapolaci použít rutinu pro polynomiální extrapolaci.

Rutina pro výpočet určitého integrálu Rombergovou kvadraturou (C) [Press et al., 1997a]
`float qromb(float (*func)(float), float a, float b)`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE qromb(func, a, b, ss)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION qromb(func, a, b)`
IN: `func`, `a`, `b`, vnitřní proměnné (pro C preprocesor. konstanty) `EPS=1.0e-6`, `JMAX=20`, `K=5`.
OUT: (F77: `ss`).
DEP: `qromb`↔`trapzd`, `polint`
Vrátí integrál funkce `func` od `a` do `b`. Parametr `EPS` se nastaví na požadovanou relativní přesnost (ne méně než 10^{-6} pro single prec.), $2^{\texttt{JMAX}-1}$ je max. povolený počet zjemňovacích kroků, `K` je počet bodů použitých pro Richardsonovu extrapolaci. Velmi výkonná pro dostatečně hladké funkce.

4.4. Nevlastní integrály

Integrál nazveme nevlastním, vyskytne-li se v něm jakákoliv z těchto ‚patologií‘:

1. $\lim_{x\to a_+} f(x)$ nebo $\lim_{x\to b_-} f(x)$ mají konečné hodnoty, ale $f(a)$ nebo $f(b)$ nelze vyčíslit, např. $f(x) = \sin x/x$ v bodě $x = 0$.
2. $b = \infty$ nebo $a = -\infty$.
3. Na některé mezi se vyskytne integrabilní singularita, např. $1/\sqrt{x}$ pro $a = 0$.
4. Na známém místě $a < c < b$ se vyskytne integrabilní singularita.
5. Na neznámém místě v intervalu (a, b) se vyskytne integrabilní singularita.

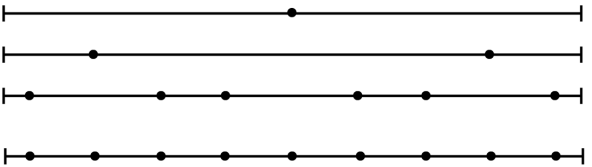
Integrály typu $\int_1^\infty x^{-1}\,\mathrm{d}x = \infty$ nebo $\int_{-\infty}^\infty \cos x\,\mathrm{d}x$ nemají pro numerické počítání smysl a nebudeme se jimi zabývat.

Roli ‚tažného koně‘ hraje v případě nevlastních integrálů místo rozšířeného lichoběžníkového pravidla (4.17) rozšířená středová formule (4.25), neboť pro ni platí analogie (4.27), druhá Eulerova–Maclaurinova sumační formule:

$$\int_{x_1}^{x_N} f(x)\,\mathrm{d}x = h\left[f_{3/2} + f_{5/2} + f_{7/3} + \cdots + f_{N-3/2} + f_{N-1/2}\right] + \frac{B_2h^2}{4}(f'_N - f'_1) + \frac{B_{2k}h^{2k}}{(2k)!}(1 - 2^{-2k+1})(f_N^{(2k-1)} - f_1^{(2k-1)}) + \cdots \tag{4.30}$$

Dokažte (4.30) přepsáním (4.27) pro h a $h/2$ a odečtením $2\times$ druhá minus první.

Na rozdíl od sekvenčního volání rutiny `trapzd` se zdvojnásobování počtu kroků, v tomto případě se počet kroků ztrojnásobuje. ⟶



Tato rutina počítá n-tý stupeň zjemnění rozšířeného středového pravidla (C) [Press et al., 1997a]
`float midpnt(float (*func)(float), float a, float b, int n)`
(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]
`SUBROUTINE midpnt(func, a, b, s, n)`
IN: `func`, `a`, `b`, `n`.
OUT: (F77, F90: `s`).
Když je volána s $n = 1$, vrátí nejhrubší odhad $f((a + b)/2)$. Následná volání s $n = 2, 3, \ldots$ zlepšují přesnost přidáním $(2/3) \times 3^{n-1}$ vnitřních bodů. Mezi voláními nesmí být `s` modifikováno.

Rutina `midpnt` řeší ‚patologii č. 1‘ a můžeme jí okamžitě nahradit rutinu `trapzd` v rutinách `qtrap` a `qsimp` s tím, že v nich kromě toho

- snížíme parametr `JMAX` tak, aby $3^{\texttt{JMAX}-1}$ (ztrojnásobování) nebyl příliš větší než $2^{\texttt{JMAX}-1}$, tj. $\texttt{JMAX} = 14$,
- u rutiny `qsimp` nahradíme ze stejného důvodu výraz `s=(4.0*st-ost)/3.0` výrazem `s=(9.0*st-ost)/8.0`.

Modifikujte ‚ručně‘ obě rutiny a přejmenujte je např. na `qtrao` a `qsimo`. Podobně v tomto smyslu modifikujte rutinu `qromb` na rutinu `qromo`.

Rutina pro Rombergovu kvadraturu na otevřeném intervalu (C) [Press et al., 1997a]

```
float qromo(float (*func)(float), float a, float b,
            float (*choose)(float (*)(float), float, float, int))
```

(Fortran 77) [Press et al., 1997b]

```
SUBROUTINE qromo(func, a, b, ss, choose)
```

(Fortran 90) [Press et al., 1997c]

```
FUNCTION qromo(func, a, b, choose)
```

IN: `func`, `choose`, `a`, `b`, vnitřní proměnné (pro C preproc. konst.) `EPS=1.0e-6`, `JMAX=14`, `K=5`.

OUT: (F77: `ss`).

DEP: `qromo`↩`polint`, `choose` = `midpnt`, `midinf`, `midsql`, `midsqu`, `midexp` (viz dále).

Vrátí integrál funkce `func` od `a` do `b` s použitím specifické rutiny `choose`. Parametr `EPS` se nastaví na požadovanou relativní přesnost (ne méně než 10^{-6} pro single prec.), $2^{\texttt{JMAX}-1}$ je max. povolený počet zjemňovacích kroků, `K` je počet bodů použitých pro Richardsonovu extrapolaci.

Typické použití je `answer=qromo(bessy0,0.0,2.0,midpnt);`.

Rutina pro Rombergovu kvadraturu na otevřeném intervalu (C) [Press et al., 1997a]
`float qromo(float (*func)(float), float a, float b,`
`float (*choose)(float (*)(float), float, float, int))`
(Fortran 77) [Press et al., 1997b]
`SUBROUTINE qromo(func, a, b, ss, choose)`
(Fortran 90) [Press et al., 1997c]
`FUNCTION qromo(func, a, b, choose)`
IN: `func`, `choose`, `a`, `b`, vnitřní proměnné (pro C preproc. konst.) `EPS=1.0e-6`, `JMAX=14`, `K=5`.
OUT: (F77: `ss`).
DEP: `qromo`↩`polint`, `choose` = `midpnt`, `midinf`, `midsql`, `midsqu`, `midexp` (viz dále).
Vrátí integrál funkce `func` od `a` do `b` s použitím specifické rutiny `choose`. Parametr `EPS` se nastaví na požadovanou relativní přesnost (ne méně než 10^{-6} pro single prec.), $2^{\texttt{JMAX}-1}$ je max. povolený počet zjemňovacích kroků, `K` je počet bodů použitých pro Richardsonovu extrapolaci.

Typické použití je `answer=qromo(bessy0,0.0,2.0,midpnt);`.

‚Patologie č. 2‘: substituce $x = 1/t$ pro $f(x)$ klesající do ∞ rychleji než $1/x^2$:

$$\int_a^b f(x)\,\mathrm{d}x = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right)\,\mathrm{d}t \tag{4.31}$$

kde $ab > 0$ (tj. buď $b \to \infty$ a $a > 0$, nebo $a \to -\infty$ a $b < 0$).

Ekvivalent rutiny `midpnt`, ale pro ‚patologii č. 2‘ (C) [Press et al., 1997a]
`float midinf(float (*funk)(float), float aa, float bb, int n)`
(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]
`SUBROUTINE midinf(funk, aa, bb, s, n)`
IN: `funk`, `aa`, `bb`, `n`.
OUT: (F77, F90: `s`).
Horní (dolní) mez `bb` (`aa`) může být ∞ ($-\infty$), tj. ‚velmi velké číslo‘, jako $\pm 1.0 \times 10^{30}$. Musí platit `aa` $\times$ `bb` > 0. Mezi voláními nesmí být `s` modifikováno.

Potřebujeme-li např. zápornou dolní mez a $b = \infty$, použijeme
`answer=qromo(funk,-5.0,2.0,midpnt)+qromo(funk,2.0,1.0e30,midinf)`
kde ‚dělicí bod' položíme do dostatečně velké kladné hodnoty, v níž `funk` začíná asymptotický pokles k ∞.

Jiným příkladem ‚patologie č. 2' implementovaným v následující rutině je integrand klesající exponenciálně k ∞. Substituce $t = \mathrm{e}^{-x}$ $(x = -\ln t)$ dává

$$\int_{x=a}^{x=\infty} f(x)\,\mathrm{d}x = \int_{t=0}^{t=\mathrm{e}^{-a}} f(-\ln t)\,\frac{\mathrm{d}t}{t} \tag{4.32}$$

Ekvivalent rutiny `midpnt`, ale pro ‚patologii č. 2' s `funk` exponenciálně klesající k ∞ (C) [Press et al., 1997a]
`float midexp(float (*funk)(float), float aa, float bb, int n)`
(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]
`SUBROUTINE midexp(funk, aa, bb, s, n)`
IN: `funk`, `aa`, (`bb` nepoužito), `n`.
OUT: (F77, F90: `s`).
Horní mez `bb` je kvůli kompatibilitě s `midpnt`, nepoužije se. Mezi voláními nesmí být `s` modifikováno.

Potřebujeme-li např. zápornou dolní mez a $b = \infty$, použijeme
`answer=qromo(funk,-5.0,2.0,midpnt)+qromo(funk,2.0,1.0e30,midinf)`
kde ‚dělicí bod' položíme do dostatečně velké kladné hodnoty, v níž `funk` začíná asymptotický pokles k ∞.

Jiným příkladem ‚patologie č. 2' implementovaným v následující rutině je integrand klesající exponenciálně k ∞. Substituce $t = \mathrm{e}^{-x}$ $(x = -\ln t)$ dává

$$\int_{x=a}^{x=\infty} f(x)\,\mathrm{d}x = \int_{t=0}^{t=\mathrm{e}^{-a}} f(-\ln t)\,\frac{\mathrm{d}t}{t} \tag{4.32}$$

Ekvivalent rutiny `midpnt`, ale pro ‚patologii č. 2' s `funk` exponenciálně klesající k ∞ (C) [Press et al., 1997a]
`float midexp(float (*funk)(float), float aa, float bb, int n)`
(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]
`SUBROUTINE midexp(funk, aa, bb, s, n)`
IN: `funk`, `aa`, (`bb` nepoužito), `n`.
OUT: (F77, F90: `s`).
Horní mez `bb` je kvůli kompatibilitě s `midpnt`, nepoužije se. Mezi voláními nesmí být `s` modifikováno.

‚Patologie č. 3': diverguje-li $f(x)$ u a jako $(x-a)^{-\gamma}$, kde $0 \le \gamma < 1$:

$$\int_a^b f(x)\,\mathrm{d}x = \frac{1}{1-\gamma}\int_0^{(b-a)^{1-\gamma}} t^{\frac{\gamma}{1-\gamma}} f(t^{\frac{1}{1-\gamma}} + a)\,\mathrm{d}t \tag{4.33}$$

resp. pro singularitu v okolí horní meze

$$\int_a^b f(x)\,\mathrm{d}x = \frac{1}{1-\gamma}\int_0^{(b-a)^{1-\gamma}} t^{\frac{\gamma}{1-\gamma}} f(b - t^{\frac{1}{1-\gamma}})\,\mathrm{d}t \tag{4.34}$$

V případě divergence u obou mezí rozdělíme integrál na dva v některém vnitřním bodě (a, b).

Vztahy (4.33)a (4.34) mají jednoduchý tvar pro případ $\gamma = 1/2$:

$$\int_a^b f(x)\,\mathrm{d}x = \int_0^{\sqrt{b-a}} 2tf(a+t^2)\,\mathrm{d}t \tag{4.35}$$

$$\int_a^b f(x)\,\mathrm{d}x = \int_0^{\sqrt{b-a}} 2tf(b-t^2)\,\mathrm{d}t \tag{4.36}$$

Vztahy (4.35)a (4.36) implementují rutiny

Ekvivalenty rutiny `midpnt`, ale pro ‚patologii č. 3' s `funk` mající na dolní (horní) mezi integrabilní divergenci typu $(x-a)^{-1/2}$ ($(b-x)^{-1/2}$). (C) [Press et al., 1997a]

```
float midsql(float (*funk)(float), float aa, float bb, int n)
float midsqu(float (*funk)(float), float aa, float bb, int n)
```

(Fortran 77, 90) [Press et al., 1997b, Press et al., 1997c]

```
SUBROUTINE midsql(funk, aa, bb, s, n)
SUBROUTINE midsqu(funk, aa, bb, s, n)
```

IN: `funk`, `aa`, `bb`, `n`.
OUT: (F77, F90: `s`).
Mezi voláními nesmí být `s` modifikováno.

‚Patologie č. 4': převést na č. 3 rozdělením integračního intervalu bodem divergence.

Výše uvedené rutiny `mid*` lze s `qromo` aplikovat na vhodné podintervaly pokrývající integrační interval, popřípadě je lze modifikovat pro konkrétní potřeby (např. (4.33) a (4.34) pro $\gamma = 1/3$).

Vyzkoušejte demonstrační program pro n-té zjemnění pomocí otevřeného středobodového pravidla `xmidpnt` z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Vyzkoušejte demonstrační program pro uzavřenou (otevřenou) Rombergovu integraci `xqromb` (`xqromo`) z knihovny [Press et al., 1997a, Vetterling et al., 1993].

Úlohy ke kapitole 4

Uzavřená kvadratura

1. Uvažujte integrál

$$I_{m,n} = \int_0^{\pi/2} \sin^{2m-1}\theta \cos^{2n-1}\theta \, \mathrm{d}\theta \tag{4.37}$$

kde m, n jsou kladná celá čísla.

(a) Nakreslete graf integrandu pro $m = n = 1$ a pro $m = 2$, $n = 4$.

(b) Vypočtěte $I_{1,1}$ a $I_{2,4}$ pomocí rozšířeného lichoběžníkového pravidla (rutina `qtrap`).

(c) Vypočtěte $I_{1,1}$ a $I_{2,4}$ pomocí rozšířeného Simpsonova pravidla (rutina `qsimp`).

(d) Vypočtěte $I_{1,1}$ a $I_{2,4}$ pomocí Rombergovy kvadratury (rutina `qromb`).

(e) Porovnejte získané výsledky s exaktními hodnotami

$$I_{1,1} = \tfrac{1}{2}$$

$$I_{2,4} = \tfrac{1}{40}$$

získanými z analytické formule pro integrál (4.37) (viz např. [Prudnikov et al., 1981], str. 402, formule 26)

$$I_{m,n} = \frac{1}{2}B(m,n) \equiv \frac{\Gamma(m)\Gamma(n)}{2\Gamma(m+n)} \overset{m,n\in\mathbb{N}}{=} \frac{(m-1)!\,(n-1)!}{2(m+n-1)!}$$

Pokračování na další straně. . .

Otevřená kvadratura

2. Uvažujte integrál

$$I_p = \int_0^\infty \frac{x^{p-1}}{1+x}\,\mathrm{d}x \tag{4.38}$$

kde $p \in (0, 1)$.

(a) Nakreslete graf integrandu pro $p = 0.5$.

(b) Modifikujte rutiny `qtrap` a `qsimp` podle návodu v textu na `qtrao` a `qsimo`, a vypočtěte $I_{0.5}$.

(c) Vypočtěte $I_{0.5}$ s použitím rutiny `qromo` a příslušných subrutin `mid*`.

3. Uvažujte integrál (distribuční funkce normálního rozdělení)

$$F(x) = \frac{1}{\sqrt{2\pi}\,\sigma}\int_{-\infty}^{x} \exp\left(-\frac{t^2}{2\sigma^2}\right)\mathrm{d}t \tag{4.39}$$

(a) Vypočtěte $F(x)$ pro $x \in \langle -4\sigma, 4\sigma\rangle$ s krokem 0.1σ a výsledek vykreslete do grafu.

(b) Vypočtěte $\lim\limits_{x\to\infty} F(x)$ a porovnejte s exaktní hodnotou 1.

Shrnutí kapitoly: Naučili jste se další ze základních úloh numerického modelování: numerické kvadratuře. Setkáte-li se s určitým integrálem, který nelze spočítat analyticky, nemusíte už stříhat křivku integrandu z plechu jak bylo ze žertu naznačeno v mottu na začátku kapitoly. Místo toho můžete použít některé v této kapitole popisovaných metod.

Další zdroje:

- Kapitola 4 v [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c]. Pro jazyk C dostupné na
  http://www.library.cornell.edu/nr/bookcpdf/,
  pro Fortran na
  http://www.library.cornell.edu/nr/bookfpdf/.
- Úvod do Fortranu 95 a do numerických metod [Sandu, 2001]. Obsahuje Kapitolu 19 o numerické kvadratuře. Dostupné online na
  http://www.cs.mtu.edu/~asandu/Courses/CS2911/fortran˙notes/main.html.

5. Fourierovská spektrální analýza

Rychlý náhled kapitoly: V této kapitole bude objasněna důležité numerická metoda – rychlá Fourierova transformace (FFT). Ačkoli Fourierova transformace je známa relativně dlouho, algoritmus umožňující její rychlou numerickou implementaci vznikl teprve v padesátých létech minulého století. S nástupem počítačů se FFT a na ní založené spektrální metody staly jedním z nejmocnějších nástrojů matematické fyziky a analýzy signálů.

Cíle kapitoly:

- Naučit se základům spojitých Fourierových řad a spojité Fourierovy transformace a jejich základnám vlastnostem.
- Naučit se základům diskrétní Fourierovy transformace a jejím základnám vlastnostem, Shannonovu vzorkovacímu teorému.
- Naučit se algoritmus FFT.
- Naučit se základním aplikacím – filtrování signálu ve frekvenční doméně.

Klíčová slova kapitoly: Spojitá, diskrétní Fourierova řada, transformace; periodický, neperiodický signál; Diracova δ-funkce; výkonová spektrální hustota; Parsevalův teorém; konvoluce, korelace; Shannonův vzorkovací teorém; Danielsonův–Lanczosův algoritmus; filtrování; Wienerova optimální filtrace

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} \tag{5.1}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} \tag{5.1}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a sudým průběhem ($s(-t) = s(t)$) lze rozložit do řady s koeficienty $a_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \underbrace{a_1 \cos 2\pi f t}_{\text{základní frekv.}} \tag{5.2}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a sudým průběhem ($s(-t) = s(t)$) lze rozložit do řady s koeficienty $a_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \underbrace{a_1 \cos 2\pi f t}_{\text{základní frekv.}} + \underbrace{a_2 \cos 4\pi f t}_{\text{2. harmonická}} \tag{5.2}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a sudým průběhem ($s(-t) = s(t)$) lze rozložit do řady s koeficienty $a_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \underbrace{a_1 \cos 2\pi f t}_{\text{základní frekv.}} + \underbrace{a_2 \cos 4\pi f t}_{\text{2. harmonická}} + \underbrace{a_3 \cos 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.2}$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a sudým průběhem ($s(-t) = s(t)$) lze rozložit do řady s koeficienty $a_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \underbrace{a_1 \cos 2\pi f t}_{\text{základní frekv.}} + \underbrace{a_2 \cos 4\pi f t}_{\text{2. harmonická}} + \underbrace{a_3 \cos 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.2}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ lze rozložit do řady s koeficienty $a_n, b_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \sum\nolimits_{n=1}^{\infty} (a_n \cos 2\pi n f t + b_n \sin 2\pi n f t)$$

5.1. Fourierova transformace pro nezasvěcené

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a lichým průběhem ($s(-t) = -s(t)$) lze rozložit do řady s koeficienty $b_n \in \mathbb{R}$

$$s(t) = \underbrace{b_1 \sin 2\pi f t}_{\text{základní frekv.}} + \underbrace{b_2 \sin 4\pi f t}_{\text{2. harmonická}} + \underbrace{b_3 \sin 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.1}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ a sudým průběhem ($s(-t) = s(t)$) lze rozložit do řady s koeficienty $a_n \in \mathbb{R}$

$$s(t) = \frac{a_0}{2} + \underbrace{a_1 \cos 2\pi f t}_{\text{základní frekv.}} + \underbrace{a_2 \cos 4\pi f t}_{\text{2. harmonická}} + \underbrace{a_3 \cos 6\pi f t}_{\text{3. harmonická}} + \cdots \tag{5.2}$$

Periodický reálný signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ lze rozložit do řady s koeficienty $a_n, b_n \in \mathbb{R}$ (resp. $c_n \in \langle 0, \infty)$, $\varphi_n \in \langle -\pi/2, \pi/2 \rangle$)

$$s(t) = \frac{a_0}{2} + \sum\nolimits_{n=1}^{\infty} (a_n \cos 2\pi n f t + b_n \sin 2\pi n f t) \tag{5.3}$$

$$s(t) = \frac{a_0}{2} + \sum\nolimits_{n=1}^{\infty} c_n \cos(2\pi n f t - \varphi_n) \tag{5.4}$$

Koeficienty c_n (resp. φ_n) tvoří amplitudové (resp. fázové) spektrum signálu $s(t)$. Řada (5.3) resp. (5.4) se nazývá Fourierova.

Koeficienty Fourierovy řady (5.3) se určí podle

$$a_n = \frac{2}{T}\int_0^T s(t)\cos 2\pi n f t \,\mathrm{d}t, \quad b_n = \frac{2}{T}\int_0^T s(t)\sin 2\pi n f t \,\mathrm{d}t \tag{5.5}$$

$n = 0, 1, 2, \ldots$. Spektrální koeficienty Fourierovy řady (5.4) se určí podle

$$c_n = \sqrt{{a_n}^2 + {b_n}^2}, \quad \cos\varphi_n = \frac{a_n}{c_n}, \quad \sin\varphi_n = \frac{b_n}{c_n}$$

(5.9)

Koeficienty Fourierovy řady (5.3) se určí podle

$$a_n = \frac{2}{T}\int_0^T s(t)\cos 2\pi n f t\,\mathrm{d}t, \quad b_n = \frac{2}{T}\int_0^T s(t)\sin 2\pi n f t\,\mathrm{d}t \tag{5.5}$$

$n = 0, 1, 2, \ldots$. Spektrální koeficienty Fourierovy řady (5.4) se určí podle

$$c_n = \sqrt{{a_n}^2 + {b_n}^2}, \quad \cos\varphi_n = \frac{a_n}{c_n}, \quad \sin\varphi_n = \frac{b_n}{c_n} \tag{5.6}$$

Periodický komplexní signál $s(t)$ na $\mathbb{R}$ s frekvencí $f = 1/T$ lze rozložit do řady s koeficienty $\hat{s}_n \in \mathbb{C}$

$$s(t) = \sum\nolimits_{n=-\infty}^{\infty} \hat{s}_n\,\mathrm{e}^{-2\pi\mathrm{i}nft} \tag{5.7}$$

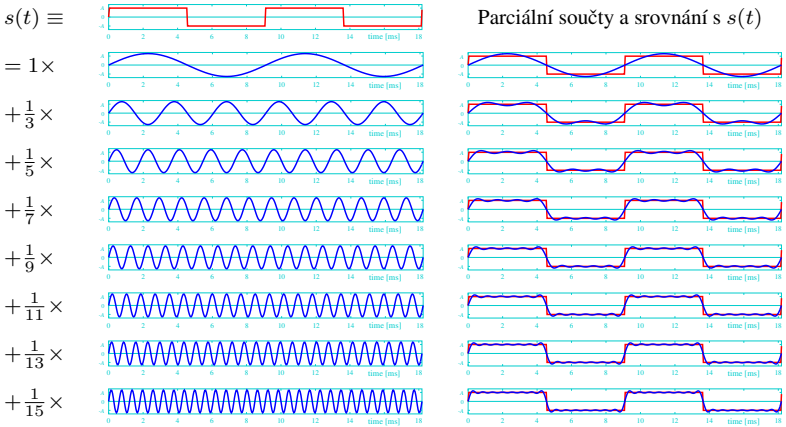
$$\hat{s}_n = \frac{1}{T}\int_0^T s(t)\,\mathrm{e}^{2\pi\mathrm{i}nft}\,\mathrm{d}t \tag{5.8}$$

$n = 0, \pm1, \pm2, \ldots$. Analogicky spektrální koeficienty jsou

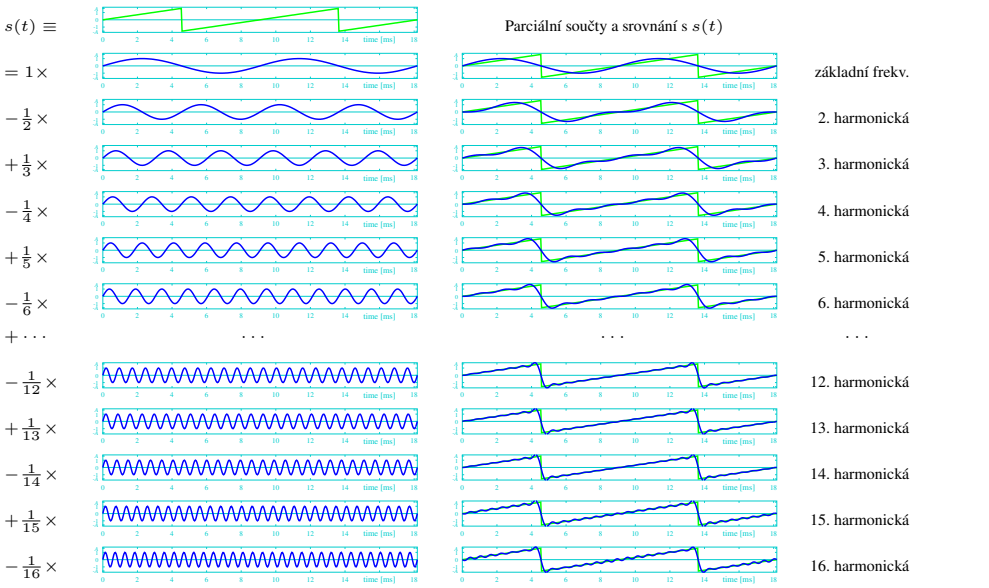
$$c_n = |\hat{s}_n|, \quad \cos\varphi_n = \frac{\Re\hat{s}_n}{c_n}, \quad \sin\varphi_n = \frac{\Im\hat{s}_n}{c_n} \tag{5.9}$$

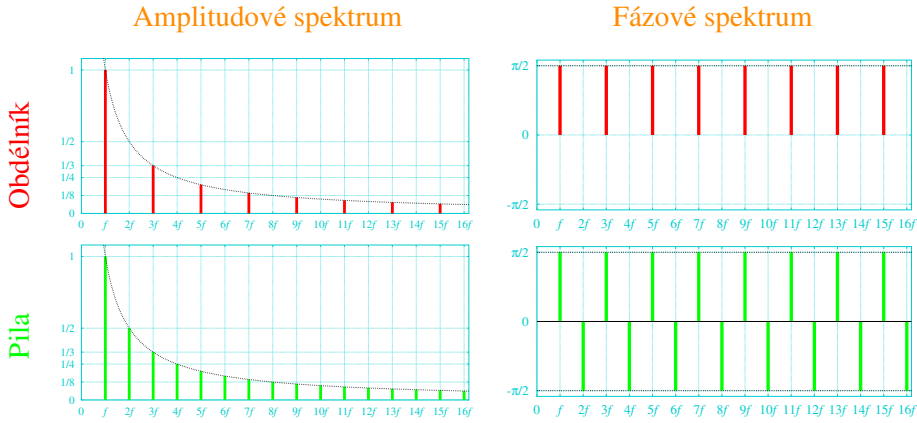
Ilustrace – Fourierův rozklad obdélníkového signálu:

Mějme signál o frekvenci $f = 110\,\mathrm{Hz}$ (perioda $T = 9.09091\,\mathrm{ms}$) s lichým obdélníkovým průběhem a amplitudou $A = \pi/4 \approx 0.785398$. Takový signál lze rozložit do sinové Fourierovy řady (5.1) s koeficienty $b_1 = 1, b_3 = 1/3, b_5 = 1/5, \ldots$ (sudé nulové). Pro poslech signálu klikni na jeho graf.

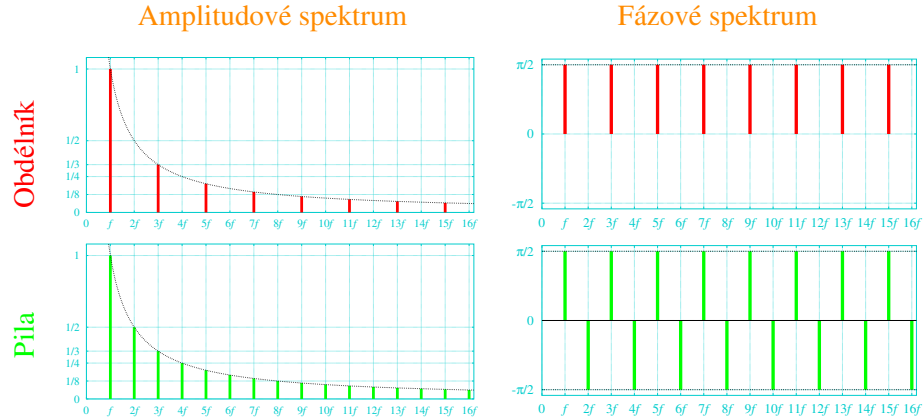


Pilový signál: frekvence stejná jako u předchozí ilustrace, ale lichý pilový průběh a amplituda $A = \pi/2 \approx 1.570796$. Nyní $b_1 = 1, b_2 = -1/2, b_3 = 1/3, \mathrm{b}_4 = -1/4, b_5 = 1/5, \ldots$. Pro poslech signálu klikni na jeho graf.



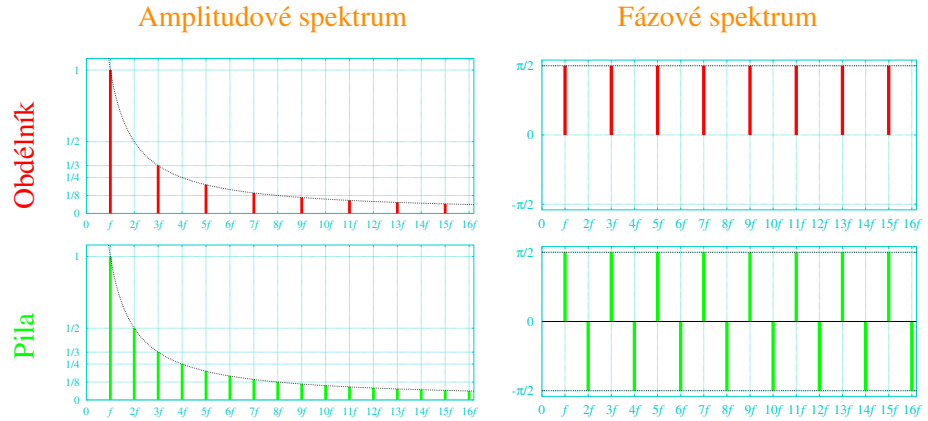


Spektrum obsahuje kompletní informaci o signálu $s(t)$, jež může být ze spektra zrekonstruována podle vztahů (5.3), (5.4) nebo (5.7).



Spektrum obsahuje kompletní informaci o signálu $s(t)$, jež může být ze spektra zrekonstruována podle vztahů (5.3), (5.4) nebo (5.7).

Neperiodický signál $s(t)$ na $\mathbb{R}$ lze chápat jako limitní případ periodického signálu s periodou $T \to \infty$. Potom $f \to 0$, čáry spekter se k sobě přibližují a diskrétní spektrum přechází na spojité. Vztahy (5.8) a (5.7) přecházejí na přímou a inverzní Fourierovu transformaci dané vztahem (5.21).



Spektrum obsahuje kompletní informaci o signálu $s(t)$, jež může být ze spektra zrekonstruována podle vztahů (5.3), (5.4) nebo (5.7).

Neperiodický signál $s(t)$ na $\mathbb{R}$ lze chápat jako limitní případ periodického signálu s periodou $T \to \infty$. Potom $f \to 0$, čáry spekter se k sobě přibližují a diskrétní spektrum přechází na spojité. Vztahy (5.8) a (5.7) přecházejí na přímou a inverzní Fourierovu transformaci dané vztahem (5.21).

Jak vypadají spektra harmonických složek a parciálních součtů obdélníku a pily?

Jsou spektra obou uvedených signálů skutečně striktně diskrétní (čárová)?

Jsou spektra obou uvedených signálů skutečně striktně diskrétní (čárová)?

Nikoliv! Striktně diskrétní spektrum by měl pouze nekonečně dlouho trvající signál, periodický na celém $\mathbb{R}$. Skutečný signál je konečný v čase, a mimo dobu trvání může být dodefinován nulou, takže ve skutečnosti není periodický na $\mathbb{R}$, a jeho spektrum dané Fourierovou transformací (5.21) je spojité.

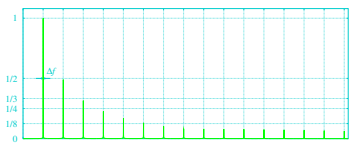
Spektrum dostatečně dlouho trvajícího signálu – signálu, jehož trvání Δt je mnohonásobkem jeho ‚periody' $T = 1/f$ (tj. $f\Delta t \gg 1$) – je však blízké diskrétnímu.

Budeme-li dobu trvání signálu Δt zkracovat, budou se úzké píky spektra čím dál víc rozšiřovat a odlišovat od diskrétního spektra.

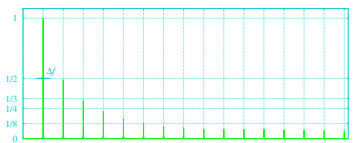
V extrémním případě, kdy signál obsahuje jen několik málo ‚period' $T = 1/f$, rozšíří se původně úzké píky spektra natolik, že jsou výrazně zastoupeny i frekvence ležící mezi píky – spektrum se rozmaže. Dá se ukázat že ‚neurčitost frekvence' Δf je spojena s trváním signálu Δt vztahem

$$\Delta f \Delta t \approx 1 \tag{5.10}$$

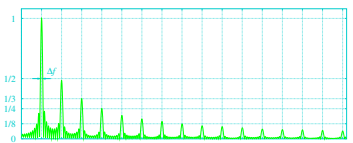
nazývaným relací neurčitosti. Tento fundamentální výsledek ilustrují na příkladu pilovitého signálu následující obrázky. (Srovnej je s diskrétním spektrem.)



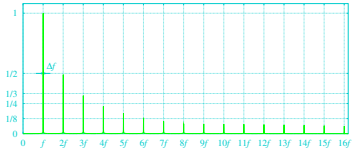
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



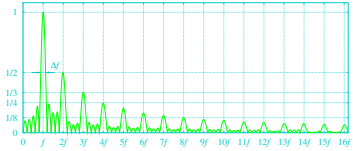
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



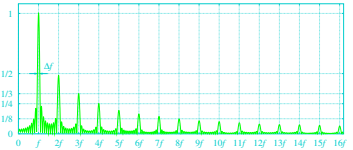
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 92.88\,\mathrm{ms}$ (počet samplů 2048=2^{11}) a obsahujícího cca 10 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 11\,\mathrm{Hz}$.



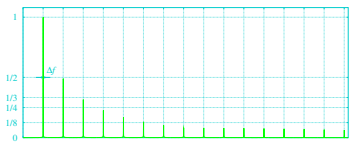
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



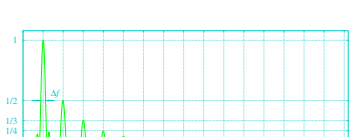
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 46.44\,\mathrm{ms}$ (počet samplů 1024=2^{10}) a obsahujícího cca 5 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 22\,\mathrm{Hz}$, píky se rozšiřují.



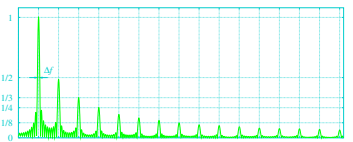
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 92.88\,\mathrm{ms}$ (počet samplů 2048=2^{11}) a obsahujícího cca 10 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 11\,\mathrm{Hz}$.



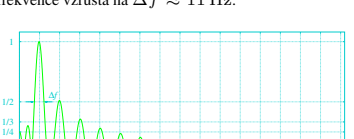
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



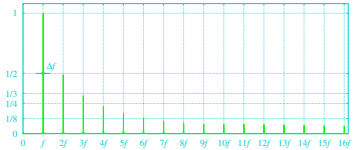
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 46.44\,\mathrm{ms}$ (počet samplů 1024=2^{10}) a obsahujícího cca 5 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 22\,\mathrm{Hz}$, píky se rozšiřují.



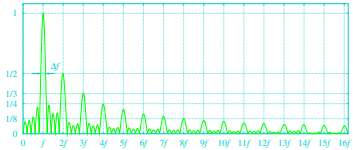
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 92.88\,\mathrm{ms}$ (počet samplů 2048=2^{11}) a obsahujícího cca 10 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 11\,\mathrm{Hz}$.



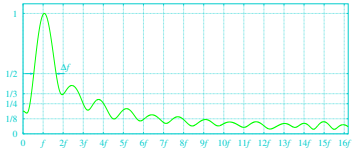
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 23.22\,\mathrm{ms}$ (počet samplů 512=2^{9}) a obsahujícího necelé 3 periody. Neurčitost frekvence vzrůstá na $\Delta f \approx 43\,\mathrm{Hz}$ – srovnatelná se základní frekvencí.



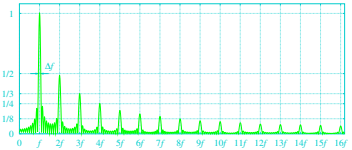
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



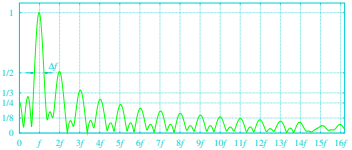
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 46.44\,\mathrm{ms}$ (počet samplů 1024=2^{10}) a obsahujícího cca 5 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 22\,\mathrm{Hz}$, píky se rozšiřují.



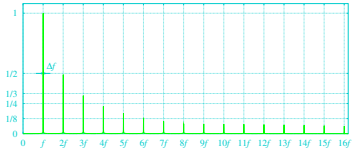
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 11.61\,\mathrm{ms}$ (256=2^8 samplů) obsahujícího zhruba jednu a čtvrt periody. Neurčitost frekvence $\Delta f \approx 86\,\mathrm{Hz} \sim f = 110\,\mathrm{Hz}$ potlačuje rozlišitelnost píků.



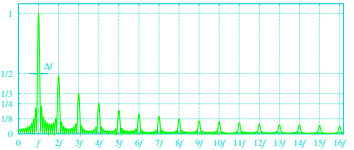
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 92.88\,\mathrm{ms}$ (počet samplů 2048=2^{11}) a obsahujícího cca 10 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 11\,\mathrm{Hz}$.



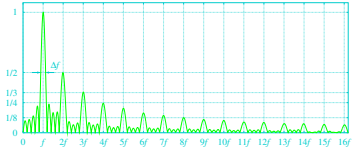
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 23.22\,\mathrm{ms}$ (počet samplů 512=2^9) a obsahujícího necelé 3 periody. Neurčitost frekvence vzrůstá na $\Delta f \approx 43\,\mathrm{Hz}$ – srovnatelná se základní frekvencí.



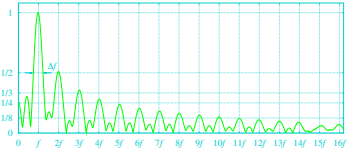
Amplitudové spektrum původního pilovitého signálu s délkou trvání $\Delta t = 2.97215\,\mathrm{s}$ (počet samplů 65536=2^{16}) a obsahujícího cca 327 period. Neurčitost frekvence $\Delta f \approx 0.34\,\mathrm{Hz}$ je nepatrná.



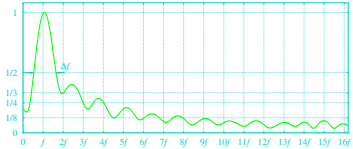
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 92.88\,\mathrm{ms}$ (počet samplů 2048=2^{11}) a obsahujícího cca 10 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 11\,\mathrm{Hz}$.



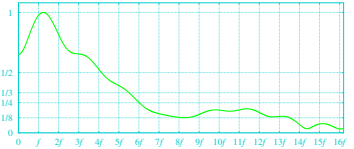
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 46.44\,\mathrm{ms}$ (počet samplů 1024=2^{10}) a obsahujícího cca 5 period. Neurčitost frekvence vzrůstá na $\Delta f \approx 22\,\mathrm{Hz}$, píky se rozšiřují.



Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 23.22\,\mathrm{ms}$ (počet samplů 512=2^{9}) a obsahujícího necelé 3 periody. Neurčitost frekvence vzrůstá na $\Delta f \approx 43\,\mathrm{Hz}$ – srovnatelná se základní frekvencí.



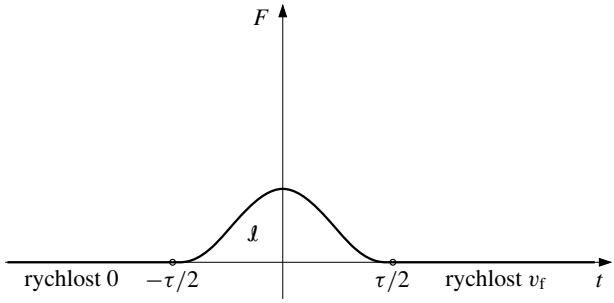
Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 11.61\,\mathrm{ms}$ (256=2^{8} samplů) obsahujícího zhruba jednu a čtvrt periody. Neurčitost frekvence $\Delta f \approx 86\,\mathrm{Hz} \sim f = 110\,\mathrm{Hz}$ potlačuje rozlišitelnost píků.



Amplitudové spektrum pilovitého signálu zkráceného na $\Delta t = 5.805\,\mathrm{ms}$ (počet samplů 128=2^{7}) a obsahujícího něco přes půl periody. Neurčitost frekvence $\Delta f \approx 172\,\mathrm{Hz} > f = 110\,\mathrm{Hz}$ zcela smazává periodicitu.

5.1.1. Diracova δ-funkce

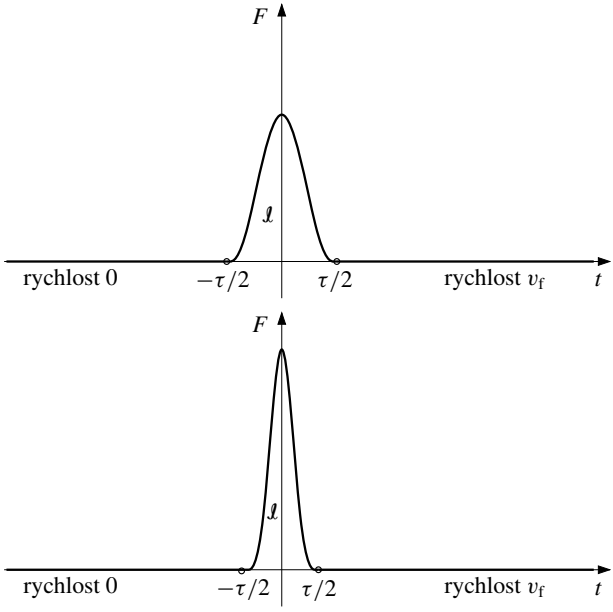
Motivace: jednorozměrný pohyb bodové hmoty m řídící se II. Newtonovým zákonem $m\,\mathrm{d}v/\mathrm{d}t = F(t)$, na počátku v klidu, působení síly omezeno na konečný časový interval délky τ:



Po odeznění silového impulsu ($t \geq \tau/2$) má hmota rychlost

$$v_\mathrm{f} = \frac{1}{m}\int_{-\tau/2}^{\tau/2} F(t)\,\mathrm{d}t = \frac{1}{m}\underbrace{\int_{-\infty}^{\infty} F(t)\,\mathrm{d}t}_{\mathcal{I}\,\equiv\,\text{impuls síly}} = \frac{1}{m}\,\mathcal{I} \tag{5.11}$$

Zkracujeme dobu působení τ, zvětšujeme F tak, aby celkový impuls $\mathcal{I}$ (plocha pod grafem $F(t)$) zůstal stejný. Rychlost v_f zůstane podle (5.11) stejná:



Limitní případ: síla působící po nekonečně krátkou dobu v čase $t = 0$, ale je ‚nekonečně veliká' tak, aby integrál $\int_{-\infty}^{\infty} F(t)\,\mathrm{d}t = \mathcal{I}$, čímž částice opět získá rychlost $v_\mathrm{f} = \mathcal{I}/m$:

$$\delta(t) = \begin{cases} 0 & \text{pro } t \neq 0\,, \\ \infty & \text{pro } t = 0 \end{cases} \quad \text{tak, že} \quad \int_{-\infty}^{\infty} \delta(t)\,\mathrm{d}t = 1 \tag{5.12}$$

Idealizovaný, nekonečně krátkou dobu působící silový impuls, který udělí částici rychlost v_f, lze psát jako $F(t) = \mathcal{I}\delta(t)$.

Korektní definice δ-funkce: teorie distribucí (např. [Schwartz, 1972, Kap. II]).

Limitní případ: síla působící po nekonečně krátkou dobu v čase $t = 0$, ale je ‚nekonečně veliká' tak, aby integrál $\int_{-\infty}^{\infty} F(t)\,\mathrm{d}t = \mathcal{I}$, čímž částice opět získá rychlost $v_\mathrm{f} = \mathcal{I}/m$:

$$\delta(t) = \begin{cases} 0 & \text{pro } t \neq 0\,, \\ \infty & \text{pro } t = 0 \end{cases} \quad \text{tak, že} \quad \int_{-\infty}^{\infty} \delta(t)\,\mathrm{d}t = 1 \tag{5.12}$$

Idealizovaný, nekonečně krátkou dobu působící silový impuls, který udělí částici rychlost v_f, lze psát jako $F(t) = \mathcal{I}\delta(t)$.

Korektní definice δ-funkce: teorie distribucí (např. [Schwartz, 1972, Kap. II]).

Některé vlastnosti δ-funkce:

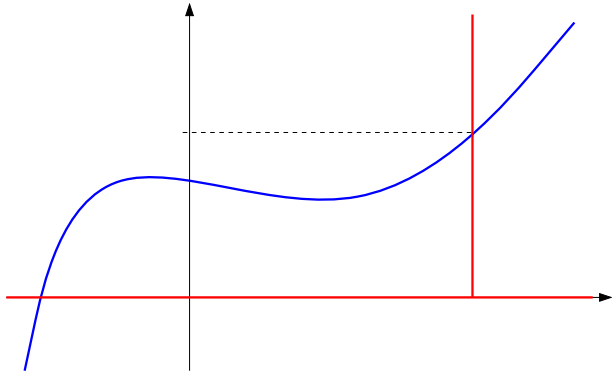
$$\delta(-t) = \delta(t) \qquad \text{(sudá funkce)} \tag{5.13}$$

$$\delta(at) = \frac{1}{|a|}\,\delta(t) \quad a \neq 0 \qquad \text{(škálování času)} \tag{5.14}$$

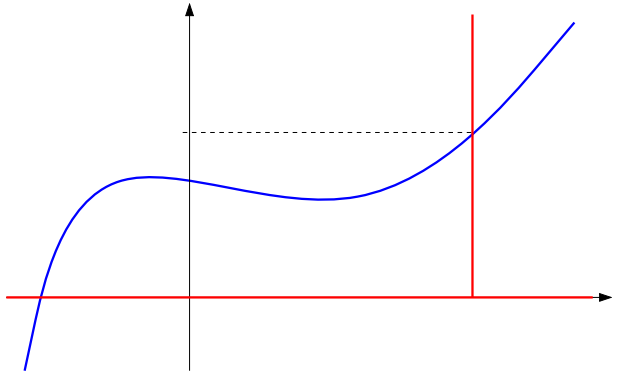
$$\delta(\varphi(t)) = \sum_i \frac{1}{|\varphi'(t_i)|}\,\delta(t - t_i), \quad t_i \text{ jsou kořeny } \varphi(t) = 0 \tag{5.15}$$

Dokažte vztahy (5.13), (5.14), (5.15). U prvních dvou zvolte vhodnou substituci, u třetího si uvědomte, kde a pod jakým úhlem protíná graf funkce $\varphi(t)$ osu t.

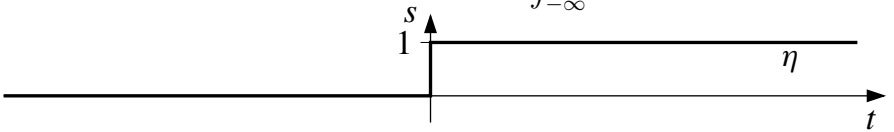
$$\int_{-\infty}^{\infty} f(t)\delta(t)\,\mathrm{d}t = f(0), \qquad \int_{-\infty}^{\infty} f(t)\delta(t-t_0)\,\mathrm{d}t = f(t_0) \tag{5.16}$$



$$\int_{-\infty}^{\infty} f(t)\delta(t)\,\mathrm{d}t = f(0), \qquad \int_{-\infty}^{\infty} f(t)\delta(t-t_0)\,\mathrm{d}t = f(t_0) \tag{5.16}$$



Jednotkový skok (Heavisideova funkce): $\eta(t) \equiv \int_{-\infty}^{t} \delta(t')\,\mathrm{d}t'$



Jiná motivace: jak zapsat hustotu bodové hmoty m umístěné v bodě s polohovým vektorem $\boldsymbol{r}_0 = (x_0, y_0, z_0)$?

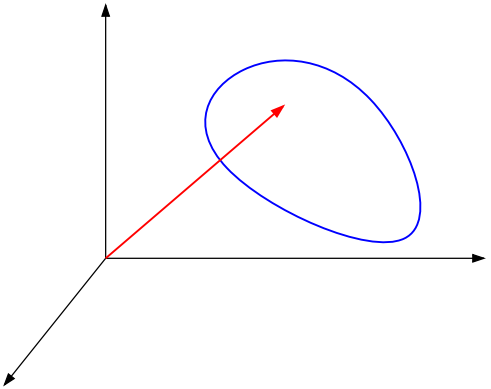
Jiná motivace: jak zapsat hustotu bodové hmoty m umístěné v bodě s polohovým vektorem $\boldsymbol{r}_0 = (x_0, y_0, z_0)$?

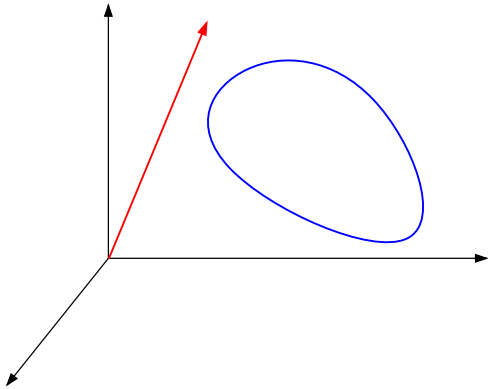
$$\rho(\boldsymbol{r}) = m\delta(\boldsymbol{r} - \boldsymbol{r}_0), \quad \text{kde} \quad \delta(\boldsymbol{r}) \equiv \delta(x)\delta(y)\delta(z)$$

Celková hmota je

$$\int_\Omega \rho(\boldsymbol{r})\,\mathrm{d}V = m\int_\Omega \delta(\boldsymbol{r} - \boldsymbol{r}_0)\,\mathrm{d}V = m$$

pokud $\boldsymbol{r}_0 \in \Omega$, a nulová, pokud $\boldsymbol{r}_0 \notin \Omega$.





V diskrétních aplikacích má vlastnosti analogické Diracově δ-funkci Kroneckerova δ definovaná pro celočíselné indexy $i, k \in \mathbb{Z}$ zcela regulárně jako

$$\delta_{ik} = \begin{cases} 0 & \text{pro } i \neq k\,, \\ 1 & \text{pro } i = k\,. \end{cases} \tag{5.17}$$

Vztahy (5.12) a (5.16) mají diskrétní ekvivalent

$$\sum_{k=-\infty}^{\infty} \delta_{ik} = 1\,, \tag{5.18}$$

$$\sum_{k=-\infty}^{\infty} \delta_{ik} f_k = f_i\,. \tag{5.19}$$

Diskrétní analogie Heavisideovy funkce je

$$\eta_i = \sum_{k=-\infty}^{i} \delta_{ik} = \begin{cases} 0 & \text{pro } i < 0\,, \\ 1 & \text{pro } i \geq 0\,. \end{cases} \tag{5.20}$$

Vztahy (5.18) a (5.19) se opět dají zobecnit v tom smyslu, že sumační množinu rovnou všem celým číslům nahradíme libovolným intervalem celých čísel, přičemž (5.18) resp. (5.19) platí, patří-li 0 resp. i do sumačního intervalu, jinak je hodnota pravých stran (5.18) a (5.19) nulová.

5.2. Spojitá Fourierova transformace

Fyzikální procesy a signály:

- v časové doméně, obecně komplexní funkce (signál) $s(t)$ (t může reprezentovat i jinou než časovou souřadnici, typicky délkovou, příp. také dvě nebo více proměnných, opět typicky délkové – obraz).
- ve frekvenční doméně, obecně komplexní signál $\hat{s}(f)$

Funkce $s(t)$ a $\hat{s}(f)$ jsou jen různé reprezentace (obrazy) téhož signálu. Jsou vázány přímou Fourierovou transformací (tvoří transformační pár):

$$\underbrace{\hat{s}(f)}_{\text{fr. dom.}} = \int_{-\infty}^{\infty} \underbrace{s(t)}_{\text{čas. dom.}} \mathrm{e}^{2\pi \mathrm{i} f t}\, \mathrm{d}t$$

5.2. Spojitá Fourierova transformace

Fyzikální procesy a signály:

- v časové doméně, obecně komplexní funkce (signál) $s(t)$ (t může reprezentovat i jinou než časovou souřadnici, typicky délkovou, příp. také dvě nebo více proměnných, opět typicky délkové – obraz).
- ve frekvenční doméně, obecně komplexní signál $\hat{s}(f)$

Funkce $s(t)$ a $\hat{s}(f)$ jsou jen různé reprezentace (obrazy) téhož signálu. Jsou vázány přímou a inverzní Fourierovou transformací (tvoří transformační pár):

$$\underbrace{\hat{s}(f)}_{\text{fr. dom.}} = \int_{-\infty}^{\infty} \underbrace{s(t)}_{\text{čas. dom.}} \mathrm{e}^{2\pi \mathrm{i} f t}\,\mathrm{d}t \quad \overset{\text{FT}}{\Longleftrightarrow} \quad \underbrace{s(t)}_{\text{čas. dom.}} = \int_{-\infty}^{\infty} \underbrace{\hat{s}(f)}_{\text{fr. dom.}} \mathrm{e}^{-2\pi \mathrm{i} f t}\,\mathrm{d}f$$

5.2. Spojitá Fourierova transformace

Fyzikální procesy a signály:

- v časové doméně, obecně komplexní funkce (signál) $s(t)$ (t může reprezentovat i jinou než časovou souřadnici, typicky délkovou, příp. také dvě nebo více proměnných, opět typicky délkové – obraz).
- ve frekvenční doméně, obecně komplexní signál $\hat{s}(f)$

Funkce $s(t)$ a $\hat{s}(f)$ jsou jen různé reprezentace (obrazy) téhož signálu. Jsou vázány přímou a inverzní Fourierovou transformací (tvoří transformační pár):

$$\underbrace{\hat{s}(f)}_{\text{fr. dom.}} = \int_{-\infty}^{\infty} \underbrace{s(t)}_{\text{čas. dom.}} \mathrm{e}^{2\pi \mathrm{i} f t}\,\mathrm{d}t \quad \overset{\text{FT}}{\Longleftrightarrow} \quad \underbrace{s(t)}_{\text{čas. dom.}} = \int_{-\infty}^{\infty} \underbrace{\hat{s}(f)}_{\text{fr. dom.}} \mathrm{e}^{-2\pi \mathrm{i} f t}\,\mathrm{d}f \tag{5.21}$$

Fourierův obraz $\hat{s}(f)$: váhová funkce, součin $\hat{s}(f)\,\mathrm{d}f$ udává, jak mnoho ‚vlnek' $\mathrm{e}^{-2\pi \mathrm{i} f t}$ obsahuje signál $s(t)$ ve frekvenčním intervalu $\langle f, f + \mathrm{d}f \rangle$.

Ekvivalentní přetlumočení do složek:

Snadno se lze přesvědčit, že pro reálnou a imaginární část signálu $s(t)$ platí

$$\Re s(t) = \int_{-\infty}^{\infty} S(f)\cos[2\pi f t - \alpha(f)]\,\mathrm{d}f \tag{5.22}$$

$$\Im s(t) = -\int_{-\infty}^{\infty} S(f)\sin[2\pi f t - \alpha(f)]\,\mathrm{d}f \tag{5.23}$$

kde reálné funkce $S(f)$ a $\alpha(f)$ jsou dány vztahy

$$S(f) \equiv |\hat{s}(f)|, \quad \cos\alpha(f) = \frac{\Re\hat{s}(f)}{S(f)}, \quad \sin\alpha(f) = \frac{\Im\hat{s}(f)}{S(f)} \tag{5.24}$$

Jinými slovy, reálná (imaginární) složka signálu je ‚poskládána' z kosinových (sinových) vlnek s toutéž reálnou frekvenčně závislou váhovou funkcí (amplitudou) $S(f)$ a tímtéž frekvenčně závislým fázovým posuvem $\alpha(f)$.

Přesvědčete se o platnosti (5.22) a (5.23).

Někdy se místo frekvence f používá kruhová frekvence $\omega \equiv 2\pi f$. Jak se změní definice (5.21) a vztahy (5.22), (5.23)?

Poznámka k záporným frekvencím: Na vzorcích (5.21), (5.22) nebo (5.23) možná zaráží, že frekvence f v nich probíhá všechny reálné hodnoty, ačkoliv fyzikální význam mají pouze nezáporné frekvence $f \geq 0$.

Pomocí symetrií Fourierovy transformace se lze snadno přesvědčit, že pro reálný signál $s(t)$ lze vztah (5.22) přepsat do ekvivalentního tvaru

$$s(t) = \int_0^\infty 2S(f)\cos[2\pi ft - \alpha(f)]\,\mathrm{d}f \tag{5.25}$$

v němž frekvence prochází jen nezáporné hodnoty, s nimiž bychom proto pro reálné signály vystačili i v matematickém formalismu. Symetrický integrační obor ve vzorcích (5.21), (5.22) a (5.23) je v případě reálných signálů čistě formální a protože usnadňuje matematické manipulace, budeme se této konvence přidržovat a mít na paměti, že fyzikálně $f \geq 0$.

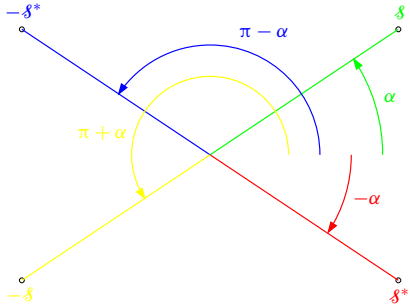
Poznámka k záporným frekvencím: Na vzorcích (5.21), (5.22) nebo (5.23) možná zaráží, že frekvence f v nich probíhá všechny reálné hodnoty, ačkoliv fyzikální význam mají pouze nezáporné frekvence $f \geq 0$.

Pomocí symetrií Fourierovy transformace se lze snadno přesvědčit, že pro reálný signál $s(t)$ lze vztah (5.22) přepsat do ekvivalentního tvaru

$$s(t) = \int_0^\infty 2S(f)\cos[2\pi ft - \alpha(f)]\,\mathrm{d}f \tag{5.25}$$

v němž frekvence prochází jen nezáporné hodnoty, s nimiž bychom proto pro reálné signály vystačili i v matematickém formalismu. Symetrický integrační obor ve vzorcích (5.21), (5.22) a (5.23) je v případě reálných signálů čistě formální a protože usnadňuje matematické manipulace, budeme se této konvence přidržovat a mít na paměti, že fyzikálně $f \geq 0$.

Poznámka k výpočtu fáze: Někdy se pro výpočet fáze $\alpha(f)$ místo korektních vztahů (5.24) uvádí nesprávně zjednodušený vztah $\alpha(f) = \mathrm{arctg}[\Im\hat{s}(f)/\Re\hat{s}(f)]$. Zaměníme-li v něm $\hat{s}(f) \to \hat{s}(f)^*$, dostaneme správně $\alpha(f) \to -\alpha(f)$. Avšak při záměně $\hat{s}(f) \to -\hat{s}(f)^*$ dává tento vztah nesprávně $\alpha(f) \to -\alpha(f)$; ve skutečnosti má být $\alpha(f) \to \pi - \alpha(f)$. Situaci ilustruje obrázek vpravo.



Fourierovým spektrem signálu $s(t)$ nazýváme:

- Buď komplexní funkci frekvence $\hat{s}(f)$ definovanou (5.21), $f \in \mathbb{R}$;
- nebo její modul $S(f) \equiv |\hat{s}(f)|$ (amplitudové spektrum) a fázový posun $\alpha(f)$ (fázové spektrum) definované (5.24), $f \in \mathbb{R}$.

Obě tyto alternativy nesou úplnou informaci o signálu $s(t)$.

Fourierovým spektrem signálu $s(t)$ nazýváme:

- Buď komplexní funkci frekvence $\hat{s}(f)$ definovanou (5.21), $f \in \mathbb{R}$;
- nebo její modul $S(f) \equiv |\hat{s}(f)|$ (amplitudové spektrum) a fázový posun $\alpha(f)$ (fázové spektrum) definované (5.24), $f \in \mathbb{R}$.

Obě tyto alternativy nesou úplnou informaci o signálu $s(t)$.

V některých aplikacích nás zajímá hlavně modul $S(f)$ nebo jeho kvadrát

$$P(f) \equiv S(f)^2, \qquad f \in \mathbb{R} \tag{5.26}$$

nazývaný oboustranná výkonová spektrální hustota (*two-sided power spectral density*). Častěji se používá jednostranná výkonová spektrální hustota (*one-sided power spectral density*)

$$\mathrm{PSD}(f) \equiv S(f)^2 + S(-f)^2, \qquad f \geq 0 \tag{5.27}$$

Pokud nebude řečeno jinak, budeme používat jednostrannou výkonovou spektrální hustotu $\mathrm{PSD}(f)$. Díky symetriím FT pro reálné signály lze ve všech spektrech omezit na $f \geq 0$, a platí $\mathrm{PSD}(f) = 2S(f)^2$.

Výraz $\mathrm{PSD}(f)\mathrm{d}f$ udává, ‚jak mnoho výkonu' signálu je lokalizováno do frekvenčního intervalu $\langle f, f + \mathrm{d}f \rangle$.

5.2.1. Základní vlastnosti FT

V následujícím textu budeme transformační pár mezi signálem v časové doméně $s(t)$ a frekvenční doméně $\hat{s}(f)$ označovat $s(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}(f)$.

Linearita Je-li $s_1(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}_1(f)$, $s_2(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}_2(f)$, pak

$$c_1 s_1(t) + c_2 s_2(t) \overset{\text{FT}}{\Longleftrightarrow} c_1 \hat{s}_1(f) + c_2 \hat{s}_2(f) \tag{5.28}$$

pro libovolné konstanty $c_1, c_2 \in \mathbb{C}$. Důkaz: plyne z linearity (5.21). Linearita je důležitou vlastností Fourierovy transformace.

5.2.1. Základní vlastnosti FT

V následujícím textu budeme transformační pár mezi signálem v časové doméně $s(t)$ a frekvenční doméně $\hat{s}(f)$ označovat $s(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}(f)$.

Linearita Je-li $s_1(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}_1(f)$, $s_2(t) \overset{\text{FT}}{\Longleftrightarrow} \hat{s}_2(f)$, pak

$$c_1 s_1(t) + c_2 s_2(t) \overset{\text{FT}}{\Longleftrightarrow} c_1 \hat{s}_1(f) + c_2 \hat{s}_2(f) \tag{5.28}$$

pro libovolné konstanty $c_1, c_2 \in \mathbb{C}$. Důkaz: plyne z linearity (5.21). Linearita je důležitou vlastností Fourierovy transformace.

Změna měřítka času (time scaling)

$$s(at) \overset{\text{FT}}{\Longleftrightarrow} |a|^{-1} \hat{s}(f/a), \qquad a \neq 0 \tag{5.29}$$

Důkaz: substitucí $at = t'$ v (5.21). Formule (5.29) říká, že dojde-li ke „zhuštění“ signálu ($a > 1$), jeho spektrum se roztáhne koeficientem a^{-1} (všechny frekvence se vynásobí koeficientem $a^{-1} < 1$), přičemž se stejným koeficientem sníží, a obráceně (viz obrázek 1).

Změna měřítka frekvence (frequency scaling)

$$\hat{s}(af) \overset{\mathrm{FT}}{\Longleftrightarrow} |a|^{-1} s(t/a), \qquad a \neq 0 \tag{5.30}$$

Důkaz: analogicky jako (5.29). Interpretace formule (5.30) je analogická jako formule (5.29) pro změnu měřítka času (viz obrázek 1).

?? Dokažte vlastnosti (5.28), (5.29) a (5.30).

Změna měřítka frekvence (frequency scaling)

$$\hat{s}(af) \overset{\mathrm{FT}}{\Longleftrightarrow} |a|^{-1} s(t/a), \qquad a \neq 0 \tag{5.30}$$

Důkaz: analogicky jako (5.29). Interpretace formule (5.30) je analogická jako formule (5.29) pro změnu měřítka času (viz obrázek 1).

?? Dokažte vlastnosti (5.28), (5.29) a (5.30).

Posun v čase (time shifting)

$$s(t - t_0) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}(f) \mathrm{e}^{2\pi \mathrm{i} f t_0} \tag{5.31}$$

Důkaz: substitucí $t - t_0 = t'$ v (5.21). Formule (5.31) říká, že při posunutí signálu o t_0 doprava se spektrum vynásobí komplexní jednotkou $\mathrm{e}^{2\pi \mathrm{i} f t_0}$, takže modul $S(f) = |\hat{s}(f)|$ zůstává nezměněn. Posun v čase ilustruje obrázek 3. Důsledkem (5.31) a linearity (5.28) je vztah

$$\tfrac{1}{2}[s(t - t_0) + s(t + t_0)] \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}(f) \cos(2\pi f t_0) \tag{5.32}$$

použitý při konstrukci obrázku 2.

Posun ve frekvenci (modulační věta, frequency shifting)

$$\hat{s}(f - f_0) \overset{\mathrm{FT}}{\Longleftrightarrow} s(t)\mathrm{e}^{-2\pi\mathrm{i}f_0 t} \tag{5.33}$$

Důkaz: substitucí $f - f_0 = f'$ v (5.21). Při frekvenčním posunu Fourierova obrazu $\hat{s}(f)$ zůstává modul signálu $|s(t)|$ nezměněn. Důsledkem (5.33) a linearity (5.28) je vztah

$$\tfrac{1}{2}[\hat{s}(f - f_0) + \hat{s}(f + f_0)] \overset{\mathrm{FT}}{\Longleftrightarrow} s(t)\cos(2\pi f_0 t)\,. \tag{5.34}$$

Je-li tedy $s(t) = 1$, máme pro Fourierův obraz kosinu sudý pár „polovičních“ δ-funkcí (viz obrázek 2).

Posun ve frekvenci (modulační věta, frequency shifting)

$$\hat{s}(f - f_0) \overset{\text{FT}}{\Longleftrightarrow} s(t)\mathrm{e}^{-2\pi\mathrm{i}f_0 t} \tag{5.33}$$

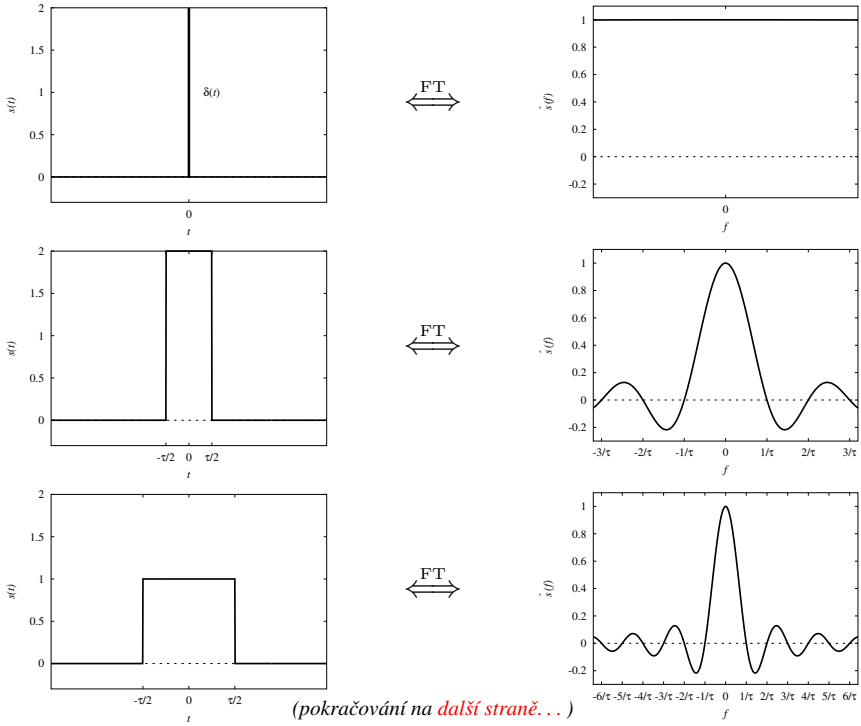
Důkaz: substitucí $f - f_0 = f'$ v (5.21). Při frekvenčním posunu Fourierova obrazu $\hat{s}(f)$ zůstává modul signálu $|s(t)|$ nezměněn. Důsledkem (5.33) a linearity (5.28) je vztah

$$\tfrac{1}{2}[\hat{s}(f - f_0) + \hat{s}(f + f_0)] \overset{\text{FT}}{\Longleftrightarrow} s(t)\cos(2\pi f_0 t)\,. \tag{5.34}$$

Je-li tedy $s(t) = 1$, máme pro Fourierův obraz kosinu sudý pár „polovičních“ δ-funkcí (viz obrázek 2).

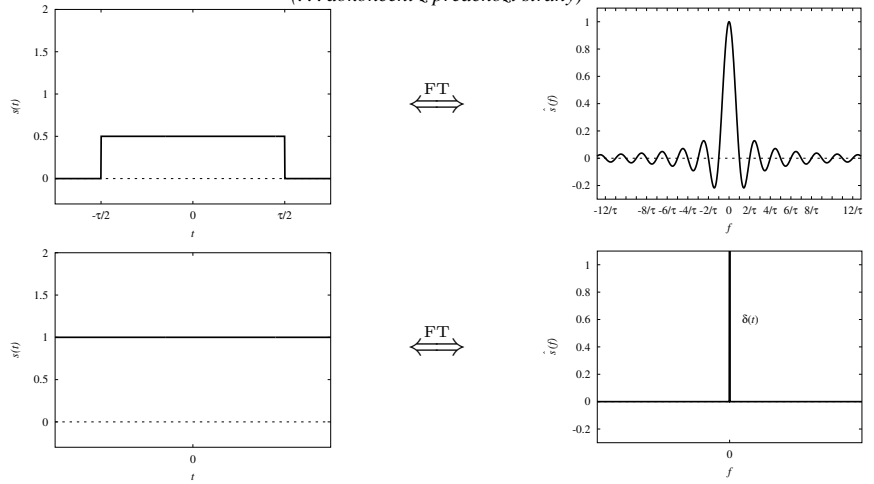
Dualita Fourierovy transformace Aplikujeme-li na signál $s(t)$ formálně FT (levá část 5.21) dvakrát po sobě, dostaneme $s(-t)$.
Důkaz: plyne ihned z (5.21).

Dokažte vlastnosti (5.31), (5.33) a dualitu.

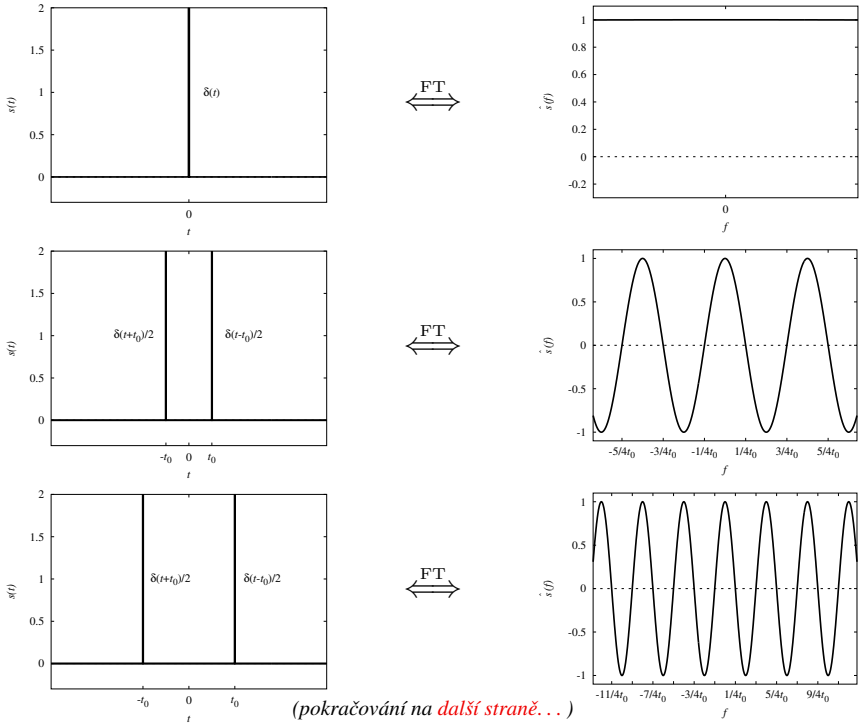


(pokračování na další straně. . .)

(. . . dokončení z předchozí strany)

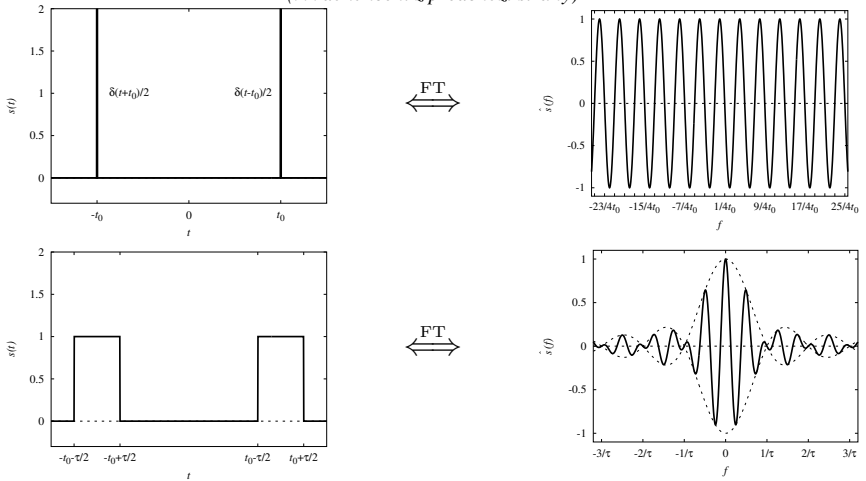


Obrázek 1: Ilustrace změny měřítka času a frekvence. V levém sloupci je časový průběh signálu (časová doména), v pravém sloupci jeho Fourierův obraz (frekvenční doména). Čtyři horní páry obsahují reálný sudý obdélníkový signál, jehož součin $A\tau$ amplitudy A a délky trvání τ je konstantní. Fourierův obraz je podle tabulky na str. 160 také reálný a sudý a je roven $\hat{s}(f) = A\tau\,\mathrm{sinc}(\pi\tau f)$ (definice sinc viz (5.44)) o maximu rovném $A\tau$. Při „roztažení“ v časové doméně se Fourierův obraz „stlačuje“ a naopak; v limitě, kdy signál v časové doméně přejde na δ-funkci, bude jeho Fourierův obraz konstantní funkcí (viz první pár). Výjimkou je poslední pár, kdy není zachována konstantnost $A\tau$, místo toho $A = 1$ a $\tau \to \infty$. Fourierovým obrazem konstantního signálu je δ-funkce.



(pokračování na další straně. . .)

(. . . dokončení z předchozí strany)



Obrázek 2: Ilustrace posunu v čase. V levém sloupci je časový průběh signálu (časová doména), v pravém sloupci jeho Fourierův obraz (frekvenční doména). Čtyři horní páry ukazují „rozštěpení“ δ-funkce na sudý pár „polovičních“ δ-funkcí s postupně vzrůstající odlehlostí $2t_0$, čemuž podle formule (5.32) odpovídá Fourierův obraz $\hat{s}(f)\cos(2\pi f t_0)$ s postupně vzrůstající „frekvencí“ t_0. V posledním řádku je obdélníkový impuls z druhého řádku obrázku 1 „rozštěpený“ na sudý pár „polovičních“ impulsů, čemuž podle formule (5.32) odpovídá Fourierův obraz původního impulsu (obalová křivka) násobený faktorem $\hat{s}(f)\cos(2\pi f t_0)$.

Limitní vlastnosti Je-li signál $s(t)$ absolutně integrovatelný na celém $\mathbb{R}$

$$\int_{-\infty}^{\infty} |s(t)|\,\mathrm{d}t < \infty \tag{5.35}$$

potom

$$\lim_{f\to\pm\infty} \hat{s}(f) = 0, \quad \text{resp.} \quad \lim_{f\to\pm\infty} S(f) = 0 \tag{5.36}$$

Tuto vlastnost budeme v dalším předpokládat. Důkaz viz např. [Rektorys, 1995]. Triviální důsledek (5.35): také $\lim_{t\to\pm\infty} s(t) = 0$.

Limitní vlastnosti Je-li signál $s(t)$ absolutně integrovatelný na celém $\mathbb{R}$

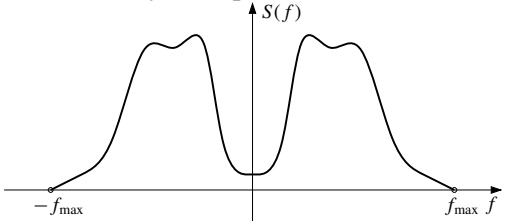
$$\int_{-\infty}^{\infty} |s(t)|\,\mathrm{d}t < \infty \tag{5.35}$$

potom

$$\lim_{f\to\pm\infty} \hat{s}(f) = 0, \quad \text{resp.} \quad \lim_{f\to\pm\infty} S(f) = 0 \tag{5.36}$$

Tuto vlastnost budeme v dalším předpokládat. Důkaz viz např. [Rektorys, 1995]. Triviální důsledek (5.35): také $\lim_{t\to\pm\infty} s(t) = 0$.

Signál, jehož spektrum splňuje $\hat{s}(f) = 0$ ($S(f) = 0$) pro $\forall f > f_c$, se nazývá signál s omezenou šířkou pásma (*bandwidth limited*). Takový signál zjevně splňuje (5.36) a má zásadní význam pro vzorkování.



Derivace originálu

$$\frac{\mathrm{d}}{\mathrm{d}t}s(t) \overset{\mathrm{FT}}{\Longleftrightarrow} -2\pi \mathrm{i} f \hat{s}(f) \tag{5.37}$$

Důkaz: integrací per partes s využitím důsledku absolutní integrovatelnosti $s(t)$

$$\int_{-\infty}^{\infty} |s(t)| \, \mathrm{d}t < \infty \tag{5.38}$$

Derivace obrazu

$$2\pi \mathrm{i} t s(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \frac{\mathrm{d}\hat{s}(f)}{\mathrm{d}t} \tag{5.39}$$

Důkaz: integrací per partes s využitím limitní vlastnosti obrazu při absolutní integrovatelnosti $s(t)$.

Ověřte vlastnosti (5.37), (5.39).

Integrace originálu Je-li $\sigma(t)$ primitivní funkce k $s(t)$

$$\sigma(t) = \int_{-\infty}^{t} s(t')\,\mathrm{d}t'$$

pak

$$\sigma(t) \overset{\mathrm{FT}}{\Longleftrightarrow} -\frac{1}{2\pi \mathrm{i} f}\hat{s}(f) \tag{5.40}$$

Důkaz: plyne z derivace originálu (5.37), kde položíme $\sigma' = s$.

Integrace obrazu Je-li $\Sigma(f)$ primitivní funkce k $\hat{s}(f)$

$$\Sigma(f) = \int_{-\infty}^{f} \hat{s}(f')\,\mathrm{d}f'$$

pak

$$\frac{1}{2\pi \mathrm{i} t}s(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \Sigma(f) \tag{5.41}$$

Důkaz: plyne z derivace obrazu (5.39), kde položíme $\Sigma' = \hat{s}$.

?? Ověřte vlastnosti (5.40) a (5.41).

Symetrie Fourierovy transformace

Má-li signál $s(t)$ určitou vlastnost symetrie, odrazí se to v symetrii jeho Fourierova obrazu $\hat{s}(f)$. Následující tabulka podává přehled základních symetrií.

Signál $s(t)$	Fourierovo sp. $\hat{s}(f)$	Ampl. sp. $S(f)$	Fázové sp. $\alpha(f)$	PSD(f)
reálný: $s(t)^* = s(t)$	$\hat{s}(-f) = \hat{s}(f)^*$	$S(-f) = S(f)$	liché: $\alpha(-f) = -\alpha(f)$	$2S(f)^2$
imag.: $s(t)^* = -s(t)$	$\hat{s}(-f) = -\hat{s}(f)^*$	$S(-f) = S(f)$	$\alpha(-f) = \pi - \alpha(f)$	$2S(f)^2$
sudý: $s(-t) = s(t)$	sudé: $\hat{s}(-f) = \hat{s}(f)$	$S(-f) = S(f)$	sudé: $\alpha(-f) = \alpha(f)$	$2S(f)^2$
lichý: $s(-t) = -s(t)$	lich.: $\hat{s}(-f) = -\hat{s}(f)$	$S(-f) = S(f)$	$\alpha(-f) = \pi + \alpha(f)$	$2S(f)^2$
reálný a sudý	reálné a sudé	$S(-f) = S(f)$	$\alpha(-f) = \alpha(f) = 0, \pi$	$2S(f)^2$
reálný a lichý	imagin. a liché	$S(-f) = S(f)$	$\alpha(f) = -\alpha(-f) = \pm\pi/2$	$2S(f)^2$
imagin. a sudý	imagin. a sudé	$S(-f) = S(f)$	$\alpha(-f) = \alpha(f) = \pm\pi/2$	$2S(f)^2$
imagin. a lichý	reálné a liché	$S(-f) = S(f)$	$\alpha(f) = 0, \pi$	$2S(f)^2$
			$\alpha(-f) = \pi, 0$	

Amplitudové spektrum je ve všech těchto speciálních případech sudé – viz obrázek u definice bandwidth limited. Pro názornou ilustraci se zvláště hodí signály ze zdůrazněných řádků.

Ověřte symetrie Fourierovy transformace s použitím (5.21), (5.22), (5.23) a (5.24).

Parsevalův teorém říká, že energii (anglicky total power) signálu $\int_{-\infty}^{\infty} |s(t)|^2 \,\mathrm{d}t$ lze počítat analogicky i ve frekvenční doméně se stejným výsledkem:

$$\int_{-\infty}^{\infty} |s(t)|^2 \,\mathrm{d}t = \int_{-\infty}^{\infty} |\hat{s}(f)|^2 \,\mathrm{d}f \,. \tag{5.42}$$

?? Ověřte Parsevalův teorém (5.42).

5.2.2. Elementární Fourierovy transformace

V této sekci až na výjimky s výhodou využijeme faktu, že Fourierova transformace reálného sudého signálu je rovněž reálná a sudá.

FT obdélníkového pulsu a její limitní případy Transformace reálného sudého obdélníkového pulsu je znázorněna třemi prostředními transformačními páry na obrázku 1. Sudý obdélníkový puls $s(t)$ výšky A a šířky τ má Fourierův obraz

$$\hat{s}(f) = A\tau \operatorname{sinc}(\pi\tau f)\,, \tag{5.43}$$

kde 'sinus cardinalis' sinc je definován jako

$$\operatorname{sinc} x = \begin{cases} \sin x/x & \text{pro } x \neq 0\,, \\ 1 & \text{pro } x = 0\,. \end{cases} \tag{5.44}$$

Je-li puls posunut o $t_0 \neq 0$, pozbude signál sudosti a podle vztahu pro časový posun (5.31) bude Fourierův obraz roven

$$\hat{s}(f) = A\tau \operatorname{sinc}(\pi\tau f)\, \mathrm{e}^{2\pi \mathrm{i} f t_0}\,. \tag{5.45}$$

Modulus přitom $S(f)$ zůstane nezměněn. Posun obdélníkového pulsu ilustruje obrázek 3. Limitní případy, tj. zúžení do δ-funkce a roztažení do konstantní funkce, jsou ilustrovány na obrázku 1. Symetrickým „rozštěpením“ obdélníkového pulsu dostaneme sudý Fourierův obraz, jak je znázorněno na dolním páru obrázku 2.
Transformační páry lze obrátit; například sudý signál v časové doméně tvarovaný podle funkce sinc (5.44) má obraz tvaru obdélníka.

FT Gaussovy funkce Transformace reálné normalizované Gaussovy funkce[2] se střední hodnotou 0 a směrodatnou odchylkou σ

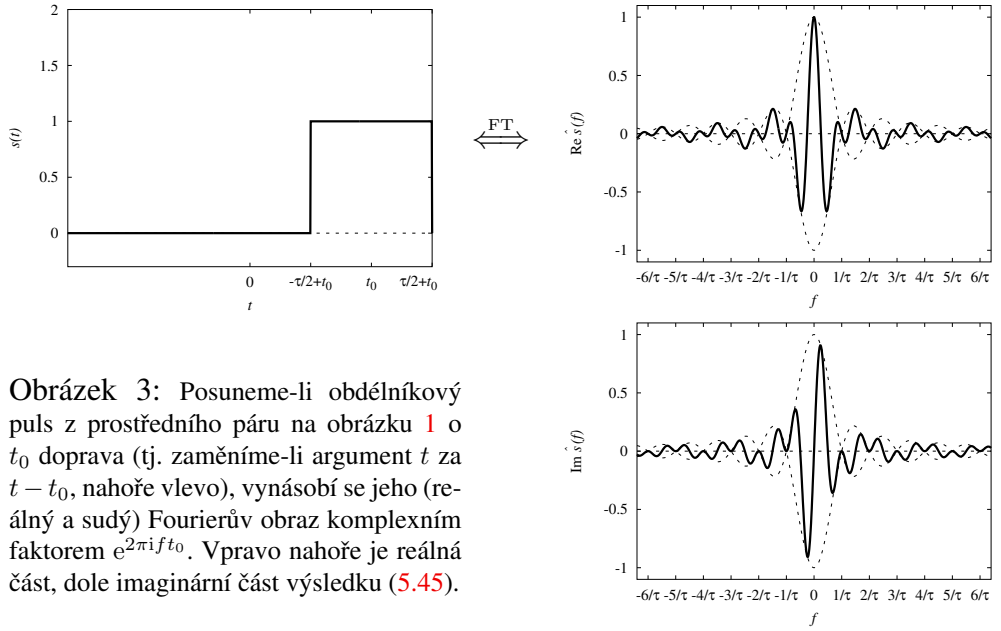
$$s(t) = \gamma(t;\sigma) \equiv \frac{1}{\sqrt{2\pi}\,\sigma}\,\mathrm{e}^{-\frac{t^2}{2\sigma^2}}\,, \tag{5.46}$$

je znázorněna třemi páry na obrázku 4. Její obraz je

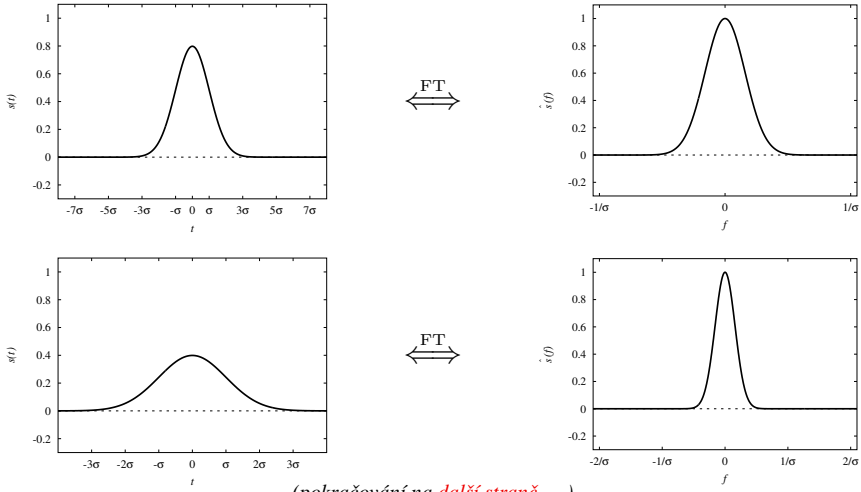
$$\hat{s}(f) = \sqrt{2\pi}\,\hat{\sigma}\,\gamma(f;\hat{\sigma})\,, \quad \text{kde} \quad \hat{\sigma} = \frac{1}{2\pi\sigma}\,. \tag{5.47}$$

Je-li gaussovský puls posunut o $t_0 \neq 0$, pozbude sudosti a podle vztahu pro časový posun (5.31) se Fourierův obraz vynásobí komplexní exponencielou

2. Normalizovaná Gaussova funkce má $\int_{-\infty}^{\infty} \gamma(t;\sigma)\,\mathrm{d}t = 1$.

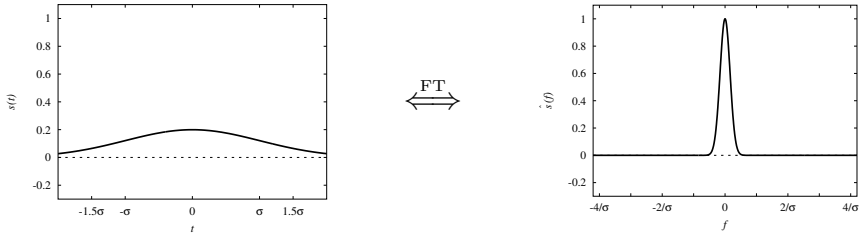


Obrázek 3: Posuneme-li obdélníkový puls z prostředního páru na obrázku 1 o t_0 doprava (tj. zaměníme-li argument t za $t - t_0$, nahoře vlevo), vynásobí se jeho (reálný a sudý) Fourierův obraz komplexním faktorem $e^{2\pi i f t_0}$. Vpravo nahoře je reálná část, dole imaginární část výsledku (5.45).

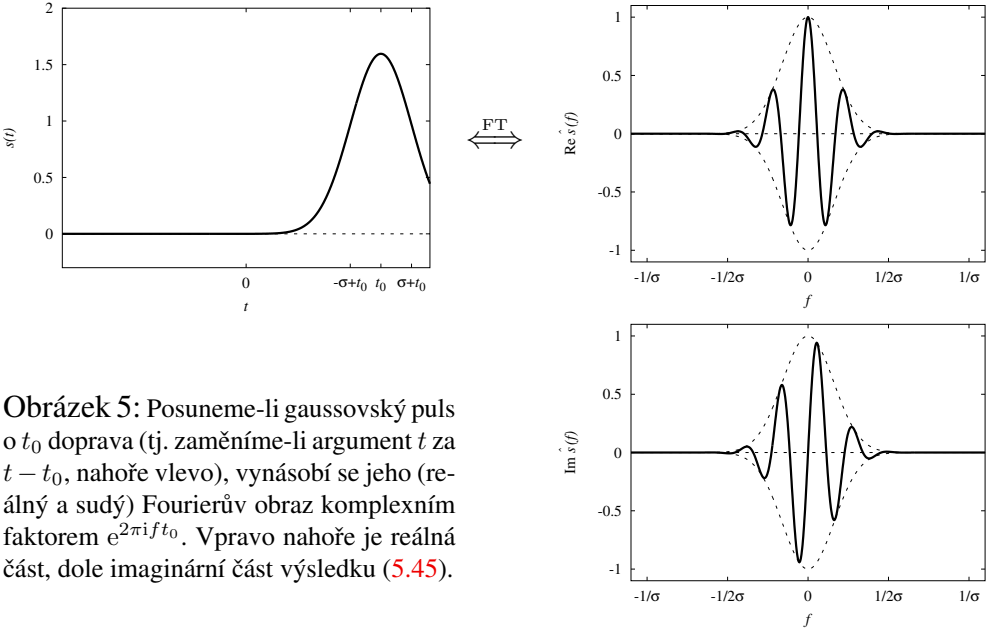


(pokračování na další straně. . .)

(. . . dokončení z předchozí strany)



Obrázek 4: Ilustrace Fourierovy transformace Gaussovy funkce. V levém sloupci je časový průběh signálu (časová doména), v pravém sloupci jeho Fourierův obraz (frekvenční doména). Všechny tři signály v časové doméně obsahují reálnou sudou normalizovanou Gaussovu funkci (5.46). Fourierův obraz (5.47) je podle tabulky na str. 160 také reálný a sudý. Při „roztažení“ v časové doméně se Fourierův obraz „stlačuje“ a naopak. Limitní případy vypadají analogicky jako na obrázku 4 a nejsou proto zobrazeny. Za povšimnutí stojí absence oscilačních laloků funkce sinc (5.44). Gaussova funkce se velice rychle blíží k nule a pro čtyřnásobek směrodatné odchylky je již prakticky nulová, což má dalekosáhlé praktické upotřebení.



Obrázek 5: Posuneme-li gaussovský puls o t_0 doprava (tj. zaměníme-li argument t za $t - t_0$, nahoře vlevo), vynásobí se jeho (reálný a sudý) Fourierův obraz komplexním faktorem $\mathrm{e}^{2\pi \mathrm{i} f t_0}$. Vpravo nahoře je reálná část, dole imaginární část výsledku (5.45).

$e^{2\pi i f t_0}$, přičemž modulus $S(f)$ zůstane nezměněn. Časový posun gaussovského pulsu ilustruje obrázek 5.

Kvalitativně popište Fourierův obraz signálu s průběhem daným funkcí sinus cardinalis (5.44).

5.2.3. Konvoluce a korelace

Konvoluce $s * r$ signálů v časové doméně $s(t)$ a $r(t)$ je definována jako

$$(s * r)(t) \equiv \int_{-\infty}^{\infty} s(\tau) r(t - \tau)\, \mathrm{d}\tau\,. \tag{5.48}$$

Konvoluce je zřejmě komutativní, $s * r = r * s$, takže lze psát

$$(s * r)(t) \equiv \int_{-\infty}^{\infty} s(t - \tau) r(\tau)\, \mathrm{d}\tau\,. \tag{5.49}$$

Konvoluční teorém říká, že je-li $s(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}(f)$ a $r(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{r}(f)$, pak

$$(s * r)(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}(f)\hat{r}(f)\,. \tag{5.50}$$

Jinými slovy, Fourierův obraz konvoluce dvou signálů je roven součinu Fourierových obrazů obou signálů. Ačkoli funkce r a s jsou z matematického hlediska v (5.48) rovnoprávné, v aplikacích obvykle jedna z nich hraje roli signálu s a druhá roli odezvy (anglicky response function) popisující deformaci signálu po průchodu přístrojem (podrobněji viz např. [Press et al., 1997a], názorná demonstrace konvoluce je na obrázku 6).

Z vlastností Diracovy δ-funkce plyne, že konvoluce libovolného signálu s Diracovou δ-funkcí (5.12) dá přesnou kopii signálu; konvoluce s Diracovou δ-funkcí posunutou o t_0, $\delta(t - t_0)$, dá kopii signálu posunutou o t_0. Naopak, signál mající tvar δ-funkce bude po konvoluci s libovolnou odezvovou funkcí roven této odezvové funkci. Konvoluce je tak vhodným nástrojem pro popis (časově invariantních) vlastností přístrojů pro zpracování signálu.

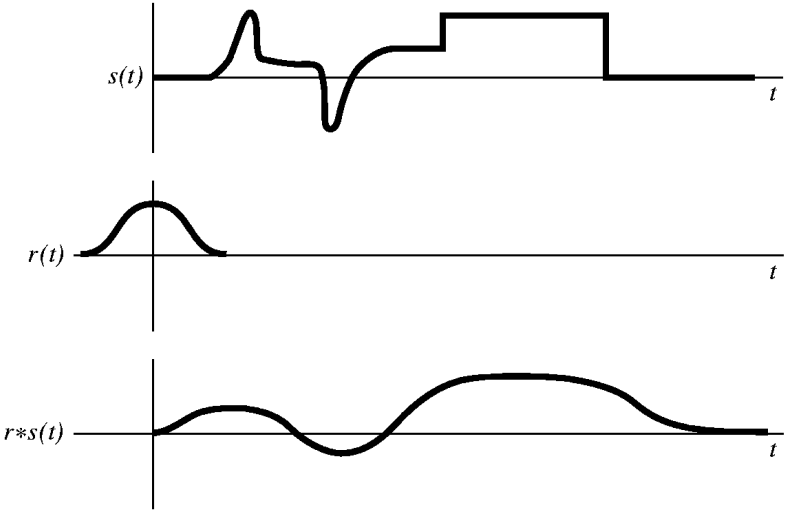
Užitečné je obrácení konvolučního teorému

$$s(t)\, r(t) \overset{\text{FT}}{\Longleftrightarrow} (\hat{s} * \hat{r})(f)\,, \tag{5.51}$$

které říká, že Fourierovým obrazem součinu dvou signálů v časové doméně je konvoluce jejich Fourierových obrazů ve frekvenční doméně

$$(\hat{s} * \hat{r})(f) \equiv \int_{-\infty}^{\infty} \hat{s}(\varphi)\hat{r}(f - \varphi)\,\mathrm{d}\varphi\,. \tag{5.52}$$

Užitečnost obrácení konvolučního teorému tkví v následující aplikaci: vyřízneme-li z dlouhotrvajícího stacionárního signálu $s(t)$ se spektrem $\hat{s}(f)$ časové okno délky T, je tato operace ekvivalentní násobení signálu $s(t)$ odezvovou funkcí $r(t)$ rovnou jedné v intervalu okna a rovnou nule jinde; spektrum takového okna odpovídá (až na



Obrázek 6: Příklad konvoluce dvou funkcí. Signál $s(t)$ je konvolvován s odezvovou funkcí $r(t)$. Jelikož odezvová funkce je širší než některé detaily v původním signálu, budou tyto detaily konvolucí „vymyty“. Není-li přítomen šum, proces konvoluce lze invertovat ve formě dekonvoluce. Obrázek je převzat z [Press et al., 1997a].

násobení komplexní exponencielou z (5.31)) funkci (5.43). Podle (5.51) je Fourierův obraz tohoto součinu roven konvoluci spektra původního signálu a funkce (5.43). Je-li např. původní spektrum tvořeno úzkými píky podobně jako spektrum na obrázku (a) na straně 139, budou — jak plyne z vlastnosti (5.16) Diracovy δ-funkce — úzké píky původního spektra nahrazeny funkcemi (5.43), a to tím širšími, čím užší bude vyříznuté okno. Tuto skutečnost ilustrují grafy (b)–(f) na straně 139. Nahradíme-li obdélníkové okno jiným, např. gaussovským (5.46), odpadnou oscilační laloky (5.43), avšak rozšíření původně úzkých píků zůstane zachováno.

Určíme-li spektrum pro celý, dostatečně dlouho trvající signál, dostaneme maximum informace o obsažených frekvencích, ale nebudeme mít žádnou informaci o tom, kdy se tyto frekvence vyskytly. Aplikujeme-li okénkovou Fourierovu transformaci,[3] jejíž princip byl popsán v předchozím odstavci, budeme mít k dispozici informaci o lokalizaci, ale ztratíme přesnost informace o frekvenci. To je opět projev principu neurčitosti (5.10).

3. Také nazývanou short-time Fourier transform.

Uvedené skutečnosti mají dalekosáhlé důsledky pro analýzu signálu. Fourierova spektrální analýza je vhodná pro stacionární signály, jejichž charakter je časově invariantní. Máme-li co do činění s nestacionárními signály a časově lokalizovanými přechodnými jevy, je vhodné použít metod waveletové analýzy, jejíž základní ideje popisuje Kapitola 6.

Korelace dvou signálů $s_1(t)$ a $s_2(t)$ je definována jako

$$\mathrm{Corr}(s_1, s_2)(t) = \int_{-\infty}^{\infty} s_1(\tau + t) s_2(\tau)\, \mathrm{d}\tau\,. \tag{5.53}$$

Korelace je funkcí prodlevy (anglicky lag) t. Korelační teorém říká, že je-li $s_1(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_1(f)$ a $s_2(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_2(f)$, pak

$$\mathrm{Corr}(s_1, s_2)(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_1(f)\hat{s}_2(-f)\,, \tag{5.54}$$

resp. pro reálné signály

$$\mathrm{Corr}(s_1, s_2)(t) \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_1(f)\hat{s}_2(f)^*\,. \tag{5.55}$$

Jinými slovy, Fourierův obraz korelace dvou reálných signálů je roven součinu jejich Fourierových obrazů, přičemž druhý bereme komplexně sdružený.

Korelace signálu se sebou samotným se nazývá autokorelace a platí pro ni (je-li signál reálný) Wienerův–Khinchinův teorém

$$\mathrm{Corr}(s,s)(t) \overset{\mathrm{FT}}{\Longleftrightarrow} |\hat{s}(f)|^2 . \tag{5.56}$$

??? Popište, jak některá zařízení (zesilovač, magnetofon, dalekohled) zkreslují signál, a uveďte kvalitativně jejich responsní funkce.

5.2.4. Fourierova transformace ve dvou a více dimenzích

Fourierův obraz dvourozměrného signálu $s(x, y)$ (typicky se jedná o spojitou formu obrazových dat) můžeme definovat přímočaře na základě (5.21) jako

$$\hat{s}(f_x, f_y) = \int\int_{-\infty}^{\infty} s(x, y)\, \mathrm{e}^{2\pi \mathrm{i}(f_x x + f_y y)}\, \mathrm{d}x\mathrm{d}y \quad \overset{\mathrm{FT}}{\Longleftrightarrow}$$

$$s(x, y) = \int\int_{-\infty}^{\infty} \hat{s}(f_x, f_y)\, \mathrm{e}^{-2\pi \mathrm{i}(f_x x + f_y y)}\, \mathrm{d}f_x\mathrm{d}f_y\,. \tag{5.57}$$

Analogicky Fourierův obraz trojrozměrného signálu $s(x, y, z)$ je

$$\hat{s}(f_x, f_y, f_z) = \int\int\int_{-\infty}^{\infty} s(x, y, z)\, \mathrm{e}^{2\pi \mathrm{i}(f_x x + f_y y + f_z z)}\, \mathrm{d}x\mathrm{d}y\mathrm{d}z \quad \overset{\mathrm{FT}}{\Longleftrightarrow}$$

$$s(x, y, z) = \int\int\int_{-\infty}^{\infty} \hat{s}(f_x, f_y, f_z)\, \mathrm{e}^{-2\pi \mathrm{i}(f_x x + f_y y + f_z z)}\, \mathrm{d}f_x\mathrm{d}f_y\mathrm{d}f_z\,. \tag{5.58}$$

Všechny vlastnosti shrnuté v sekcích 5.2.1 a 5.2.3 se snadno dají zobecnit na dvourozměrnou a trojrozměrnou Fourierovou transformaci (5.57) a (5.58).

Podobně bychom mohli definovat vícerozměrnou Fourierovu transformaci. To je opět přímočará technická záležitost a zde ji vynecháme.

5.3. Diskrétní a rychlá Fourierova transformace

5.3.1. Diskretizace časová a amplitudová

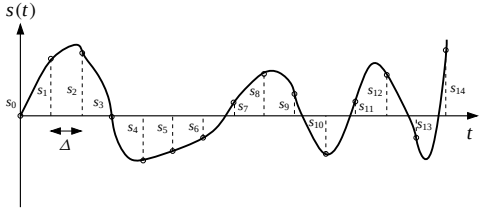
Digitalizace analogového signálu digitalizovat znamená

- jednak provést časovou diskretizaci neboli vzorkování (anglicky sampling), tj. určit hodnoty signálu $s(t)$ v diskrétních, obvykle ekvidistantních, časových intervalech [Press et al., 1997a],

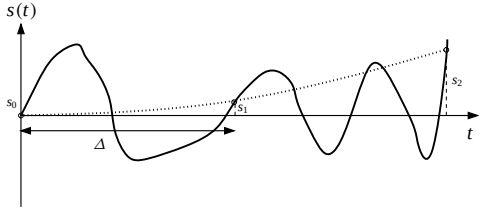
$$s_n = s(n\Delta)\,, \quad n = \ldots -3, -2, -1, 0, 1, 2, 3, \ldots\,, \tag{5.59}$$

  jak je znázorněno na obrázku 7,
- jednak provést diskretizaci hodnot s_n, tj. reprezentovat je vhodným datovým typem, např. desetinnými čísly s plovoucí tečkou v jednoduché přesnosti, dvoubajtovými neznaménkovými celými čísly apod. Této diskretizaci se říká kvantizace nebo amplitudová diskretizace a dále se jí nebudeme zabývat.

Časová odlehlost sousedních vzorků Δ se nazývá vzorkovací interval, její převrácená hodnota $f_{\mathrm{samp}} \equiv 1/\Delta$ se nazývá vzorkovací frekvence (anglicky sampling rate). U vícerozměrného signálu budeme mít odpovídající počet vzorkovacích intervalů a vzorkovacích frekvencí.



Obrázek 7: Ilustrace vzorkování analogového signálu $s(t)$. Je-li Nyquistova kritická frekvence signálu (5.60) větší než maximální frekvence $f_{\max}$ obsažená ve spektru (obrázek na straně 157), lze z hodnot s_n signál zrekonstruovat.



Obrázek 8: Ilustrace vzniku aliasu na stejném signálu jako na obrázku 7. Vzhledem k podvzorkování (nesplnění Shannonovy podmínky (5.61)) je signál rekonstruovaný ze vzorků zcela odlišný od originálního signálu.

5.3.2. Shannonův vzorkovací teorém a alias

Diskretizujeme-li signál vzorkovacím intervalem Δ, hraje klíčovou roli Nyquistova kritická frekvence

$$f_{\mathrm{c}} = \frac{1}{2\Delta} = \frac{1}{2} f_{\mathrm{samp}} \,. \tag{5.60}$$

Má-li spojitý signál $s(t)$ vzorkovaný frekvencí f_{samp} omezenou šířku pásma s maximální frekvencí f_{max} (viz obrázek na straně 157) a platí-li Shannonova podmínka[4]

$$f_{\mathrm{max}} < f_{\mathrm{c}} \,, \tag{5.61}$$

nebo ekvivalentně

$$f_{\mathrm{samp}} > 2 f_{\mathrm{max}} \,, \tag{5.62}$$

(tj. je-li vzorkovací frekvence větší než dvojnásobek maximální frekvence obsažené v signálu), pak signál $s(t)$ je úplně určen svými vzorky s_n (podrobnosti viz např. [Press et al., 1997a]).

4. V literatuře uváděná také jako Shannonova–Kotělnikovova podmínka.

Nemá-li spojitý signál $s(t)$ omezenou šířku pásma nebo $f_{\max} \geq f_c$ (tj., není-li vzorkovací frekvence více než dvojnásobkem maximální frekvence obsažené v signálu), nastává jev zvaný alias, kdy část spektra ležící vně intervalu $\langle -f_c, f_c \rangle$ je mapována dovnitř tohoto intervalu. Alias — což je v podstatě důsledek podvzorkování — je všeobecně známý jev; stačí připomenout zdánlivě „stojící“ nebo „pomalu a v opačném směru se otáčející“ kola auta pozorovaná při výbojkovém pouličním osvětlení, letecká vrtule ve filmu či televizi apod., viz obrázek 8.

5.3.3. Diskrétní Fourierova transformace

Uvažujme konečný signál s N vzorky. Diskrétní Fourierova transformace (DFT) dá na výstupu zřejmě právě N nezávislých hodnot. Místo spojitého Fourierova obrazu $\hat{s}(f)$ na intervalu $\langle -f_c, f_c \rangle$ proto hledáme diskrétní hodnoty

$$f_n \equiv \frac{n}{N\Delta}, \quad n = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2}. \tag{5.63}$$

Ve vztahu (5.63) máme celkem $N + 1$ hodnot; ukazuje se, že hodnoty v extrémních frekvencích $f_{-N/2}$ a $f_{N/2}$ jsou stejné, takže ve skutečnosti dostáváme jen N nezávislých hodnot pro $n = -N/2 + 1, \ldots, 0, \ldots, N/2$.

Aproximujeme-li první integrál (5.21) diskrétní sumou v diskrétních frekvencích (5.63), obdržíme diskrétní Fourierovu transformaci N vzorků s_k [Press et al., 1997a]

$$\hat{s}(f_n) \approx \Delta \hat{s}_n\,, \quad \text{kde} \quad \hat{s}_n \equiv \sum_{k=0}^{N-1} s_k\, \mathrm{e}^{2\pi \mathrm{i} k n/N}\,. \tag{5.64}$$

V sumě (5.64) můžeme místo $n = -N/2+1, \ldots, 0, \ldots, N/2$ sčítat přes $n = 0, \ldots, N-1$, protože (5.64) je periodický s periodou N. Nulová frekvence odpovídá $n = 0$, kladné frekvence $0 < f < f_\mathrm{c}$ odpovídají hodnotám $1 \le n \le N/2-1$, záporné frekvence $-f_\mathrm{c} < f < 0$ odpovídají hodnotám $N/2+1 \le n \le N-1$. Hodnota $n = N/2$ pak odpovídá jak $f = f_\mathrm{c}$, tak $f = -f_\mathrm{c}$.

Diskrétní Fourierova transformace (5.64) má tytéž vlastnosti symetrie jako spojitá; stačí v tabulce na str. 160 nahradit $s(t) \to s_k$, $\hat{s}(f) \to \hat{s}_n$, $\hat{s}(-f) \to \hat{s}_{N-n}$, a pojmy „sudý“ resp. „lichý“ odkazují na stejné resp. opačné znaménko ve vzorcích s indexy k a $N-k$.

Inverzní diskrétní Fourierova transformace má tvar (povšimněme si normalizačního koeficientu $1/N$)

$$s_k = N^{-1} \textstyle\sum_{n=0}^{N-1} \hat{s}_n\, \mathrm{e}^{-2\pi \mathrm{i} k n/N}\,. \tag{5.65}$$

5.3.4. Rychlý algoritmus Fourierovy transformace (FFT)

Z přepisu formule (5.64) do maticového tvaru

$$\underbrace{\begin{pmatrix}\hat{s}_0\\ \hat{s}_1\\ \hat{s}_2\\ \vdots\\ \hat{s}_{N-1}\end{pmatrix}}_{\text{výstup}} = \underbrace{\begin{pmatrix}1 & 1 & 1 & \dots & 1\\ 1 & W^1 & W^2 & \dots & W^{N-1}\\ 1 & W^2 & W^4 & \dots & W^{2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & W^{N-1} & W^{2(N-1)} & \dots & W^{(N-1)(N-1)}\end{pmatrix}}_{W} \underbrace{\begin{pmatrix}s_0\\ s_1\\ s_2\\ \vdots\\ s_{N-1}\end{pmatrix}}_{\text{vstup}}, \tag{5.66}$$

kde $W \equiv \mathrm{e}^{2\pi \mathrm{i}/N}$, je vidět, že sloupcový vektor s_k je násoben maticí W elementů W^{nk}. Toto maticové násobení, které vytváří sloupcový vektor $\hat{s}_n$, zahrnuje $\mathcal{O}(N^2)$ násobení v plovoucí desetinné tečce.

Ačkoli existuje více optimalizovaných schémat, jak výpočet (5.64) urychlit, uvedeme jedno z nich, tzv. Danielsonův–Lanczosův algoritmus pocházející z r. 1942. Počet vzorků N v Danielsonově–Lanczosově algoritmu je sice omezen na celočíselné mocniny 2, ale princip metody je velmi jednoduchý a univerzální.

Naznačíme zde podstatu algoritmu. Diskrétní Fourierova transformace posloupnosti vzorků o délce N může být rozepsána jako suma dvou DFT o délkách $N/2$, jedné zformované ze sudých vzorků, druhé z lichých vzorků. Tento postup aplikujeme rekurzivně, tj. postupujeme k DFT o délkách $N/4$, $N/8$, ... Nyní je jasné, proč algoritmus funguje jen pro $N = 2^M$, pak totiž po určitém počtu „rozpůlení“ (rovnému binárnímu logaritmu N) dospějeme k DFT o délce 1, což je identita! Stačí tedy najít (vzájemně jednoznačnou) korespondenci mezi těmito „DFT délky 1“ a původními vzorky s_k. Ukazuje se, že přepermutujeme-li původní pořadí s_k v bitově reverzním pořadí (tj. index vyjádříme ve dvojkové soustavě a převrátíme pořadí bitů.), dostaneme potřebnou korespondenci.

Nyní můžeme celý postup obrátit: nejprve provést „zpřeházení“ původního pořadí s_k v bitově reverzním pořadí (lze provést velmi rychle), čímž dostaneme N triviálních DFT o délkách 1, a z nich poté konstruujeme DFT délky $2, 4, 8, \ldots$, až dospějeme k celkové DFT signálu délky N. Protože v každém z $\log_2 N$ „půlení“ provádíme $\mathcal{O}(N)$ násobení, obsahuje celý proces $\mathcal{O}(N \log_2 N)$ násobení v plovoucí desetinné tečce. Tento algoritmus se nazývá rychlou Fourierovou transformací (anglicky Fast Fourier Transform, FFT).

To je oproti „maticové“ aritmetice (5.66) urychlení faktorem $\mathcal{O}(N/\log_2 N)$. Máme-li např. $N = 10^6$ (okolo 20 s hudby), je faktor urychlení řádu 10^4. Kdyby DFT počítaná FFT algoritmem trvala zhruba několik sekund, zabral by výpočet pomocí původní maticové podoby (5.66) zhruba jeden den!

5.3.5. Diskrétní konvoluce a korelace

Diskrétní verze vztahu (5.49) pro odezvovou funkci nenulovou pouze v rozmezí vzorků $-M/2 < k \leq M/2$ pro M dostatečně velké sudé celé číslo,[5] je

$$(s * r)_j = \sum_{k=-M/2+1}^{M/2} s_{j-k}\, r_k \,. \tag{5.67}$$

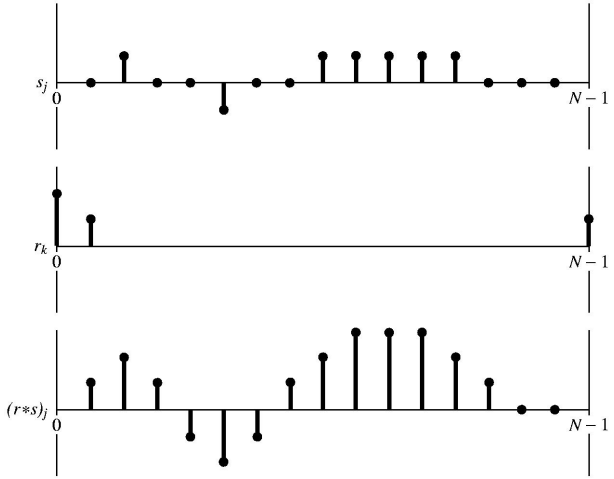
Způsob ošetření části odezvové funkce pro záporný čas objasňuje obrázek 9. Diskrétní konvoluční teorém pak říká: je-li vzorkovaný signál s_j periodický s periodou N, takže je úplně popsán N vzorky $s_0, s_1, \ldots, s_{N-1}$, pak DFT jeho diskrétní konvoluce s odezvovou funkcí r_j o délce N je rovna součinu DFT signálu $\hat{s}_n$ a odezvové funkce $\hat{r}_n$, neboli

$$\sum_{k=-N/2+1}^{N/2} s_{j-k}\, r_k \overset{\text{FT}}{\Longleftrightarrow} \hat{s}_n \hat{r}_n \,. \tag{5.68}$$

Hodnoty r_k pro $k = 0, \ldots, N-1$ jsou tytéž jako pro $k = -N/2+1, \ldots, N/2$, ale v cyklicky přetočeném pořadí, jak je popsáno v odstavci za vztahem (5.64).

---

5. Pro takovou odezvovou funkci se používá označení FIR (finite impulse response). V praktických aplikacích vystačíme s odezvovými funkcemi typu FIR, buď proto, že máme do do činění s funkcí tohoto typu, nebo jimi skutečnou odezvovou funkci aproximujeme.



Obrázek 9: Příklad konvoluce dvou diskrétně vzorkovaných funkcí. Signál s_j je konvolvován s odezvovou funkcí r_j. Pro záporný čas je odezvová funkce cyklicky přetočena na pravý konec pole r_j. Obrázek je převzat z [Press et al., 1997a].

Problém diskrepance mezí sum v (5.67) a v (5.68) se vyřeší snadno — téměř vždy máme odezvovou funkci mnohem kratší než vlastní signál, takže ji M můžeme dodefinovat na plnou délku N nulami. Obtížnější je problém splnění periodicity signálu skutečnými daty. Podrobný rozbor (viz např. [Press et al., 1997a]) ukazuje, že stačí doplnit na jeden konec dat „nulovou vycpávku“ o délce rovné většímu z čísel vyjadřujících trvání odezvové funkce v kladném či záporném čase. Pro symetrickou odezvovou funkci prostě na konec signálu přidáme $M/2$ nul.

Diskrétní forma korelace (5.53) má pro vzorkované signály s_j, v_j, periodické s periodou N, tvar

$$\mathrm{Corr}(s,v)_j = \sum_{k=0}^{N-1} s_{j+k}\, v_k \,. \tag{5.69}$$

Diskrétní korelační teorém pro reálné signály říká, že jsou-li $s_j \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_k$ a $v_j \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{v}_k$ transformační páry, pak

$$\mathrm{Corr}(s,v)_j \overset{\mathrm{FT}}{\Longleftrightarrow} \hat{s}_k \hat{v}_k^* \,. \tag{5.70}$$

Ošetření koncových efektů ukazuje, že pro prodlevu (lag) mezi signály definovanou za rovnicí (5.53) zahrnující $\pm K$ vzorků je třeba na konec obou signálů alespoň K nul.

Jak konvoluce, tak korelace, počítané podle levých stran vztahů (5.68) a (5.70), potřebují $\mathcal{O}(N^2)$ násobení v plovoucí desetinné tečce, zatímco jejich pravé strany jich potřebují pouze $\mathcal{O}(N)$. Fourierova transformace mezi oběma doménami vyžaduje $\mathcal{O}(N \log_2 N)$, takže výpočet konvoluce nebo korelace nepřímo přes násobení Fourierových obrazů je výpočetně efektivnější řádově stejným faktorem, jako rychlá Fourierova transformace vůči maticové formě DFT (viz konec sekce 5.3.4).

5.3.6. FFT ve dvou a více dimenzích

Podobně jako v případě spojité dvou nebo vícerozměrné Fourierovy transformace (sekce 5.2.4), také DFT může být přímočaře zobecněna na dvě a tři dimenze (více dimenzemi se nebudeme zabývat). Máme-li komplexní signál s_{k_1,k_2} definovaný na dvourozměrné mřížce $0 \leq k_1 \leq N_1 - 1, 0 \leq k_2 \leq N_2 - 1$, definujeme dvourozměrnou DFT jako

$$\hat{s}_{n_1,n_2} = \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} s_{k_1,k_2} \, \mathrm{e}^{2\pi \mathrm{i} k_1 n_1/N_1} \, \mathrm{e}^{2\pi \mathrm{i} k_2 n_2/N_2} \,, \tag{5.71}$$

máme-li komplexní signál s_{k_1,k_2,k_3} definovaný na trojrozměrné mřížce $0 \leq k_1 \leq N_1 - 1, 0 \leq k_2 \leq N_2 - 1, 0 \leq k_3 \leq N_3 - 1$, definujeme trojrozměrnou DFT jako

$$\hat{s}_{n_1,n_2,n_3} = \sum_{k_3=0}^{N_3-1} \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} s_{k_1,k_2,k_3} \, \mathrm{e}^{2\pi \mathrm{i} \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} + \frac{k_3 n_3}{N_3} \right)} \,. \tag{5.72}$$

Pro výpočet dvou- a trojdimenzionální diskrétní Fourierovy transformace (5.71) a (5.72) lze zobecnit rychlý Danielsonův–Lanczosův algoritmus popsaný v předešlé sekci (5.3.4). Inverzní transformaci dostaneme záměnou s za $\hat{s}$, změnou znamének v exponencielách a násobením normalizačním faktorem $1/N_1N_2$ resp. $1/N_1N_2N_3$ analogicky jako v (5.65).

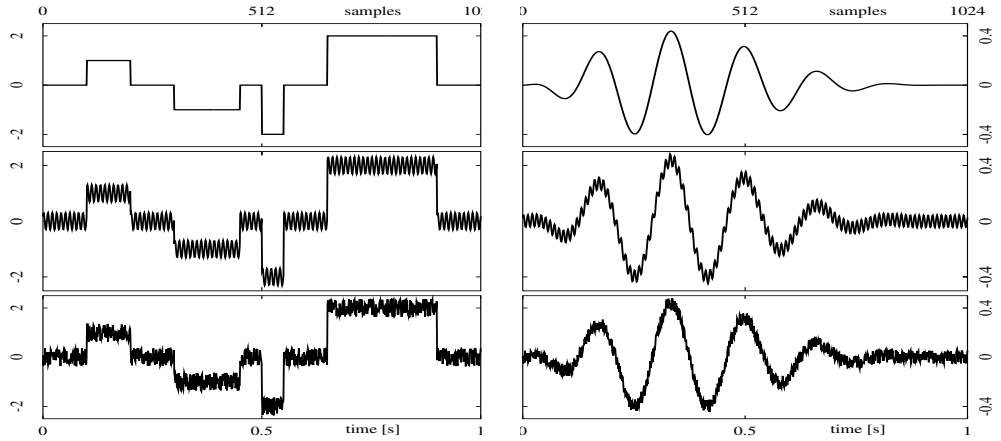
Existují knihovny numerických rutin pro výpočet jednodimenzionální a vícedimenzionální diskrétní Fourierovy transformace [Frigo a Johnson, 0000, Press et al., 1997a], včetně jejích variant, jako např. sinová transformace, kosinová transformace apod. (viz [Press et al., 1997a]).

5.4. Filtrování ve frekvenční doméně

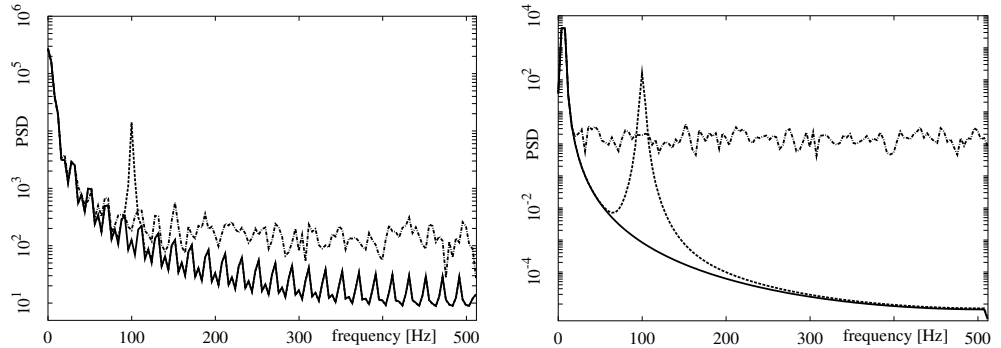
Na závěr ilustrujme použití fourierovské spektrální metody na jednoduchém příkladě filtrování signálu za účelem odstranění kontaminace v něm obsažené. Pro tento záměr byly sestrojeny dva testovací signály; jeden nespojitý a druhý hladký (viz obrázek 10). Tyto testovací signály byly podrobeny umělé kontaminaci dvojího druhu — v prvním případě se jedná o úzkopásmovou kontaminaci, tj. kontaminaci, jejíž spektrum zasahuje jen úzký rozsah frekvencí, ve druhém případě se jedná o širokopásmovou kontaminaci, jejíž spektrum zasahuje podstatnou část spektra původního signálu. Konkrétně byl v roli úzkopásmové kontaminace použit sinusový signál o frekvenci 100 Hz, v roli širokopásmové kontaminace byl použit bílý šum, tj. šum, jehož spektrum je (přibližně) je frekvenci nezávislé, a jsou v něm rovnoměrně zastoupeny všechny frekvence až do poloviny vzorkovací frekvence.[6]

Pro oba signály a jejich kontaminované verze byly sestrojeny tzv. odhady výkonové spektrální hustoty (PSD estimates, viz obrázek 11).

6. Pro umělou kontaminaci byl použit program `noisify` z kolekce [Hledík, 2007], jenž pro kontaminaci bílým šumem využívá generátoru náhodných čísel.



Obrázek 10: Testovací signály a jejich kontaminace. Ve všech případech mají délku trvání 1 s a jsou vzorkovány na $N = 1024$ vzorků. V levém sloupci je nespojitý „obdélníkový“ signál, v pravém sloupci spojitý hladký signál. Nahoře jsou signály zobrazeny v čisté podobě, v prostřední řadě jsou kontaminovány úzkopásmovým signálem (sinusovým o kmitočtu 100 Hz), v dolní řadě jsou kontaminovány bílým šumem. Odstup sumárního signálu od kontaminace je v obou případech $\mathrm{SNR} = 15\,\mathrm{dB}$ (tj. signálový výkon je $10^{15/10} = 31.6\times$ větší než kontaminační výkon).



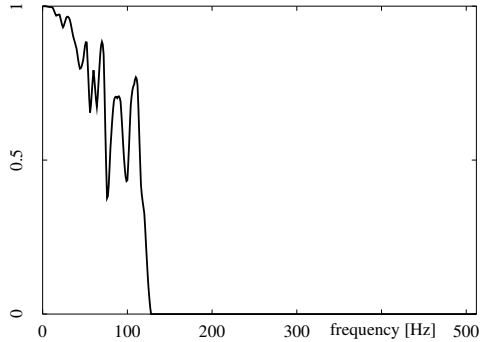
Obrázek 11: Odhad výkonové spektrální hustoty nespojitého (vlevo) a hladkého signálu (vpravo). Šířka Welchova okna [Press et al., 1997a] s 50% překryvem je 128 vzorků. Nepřerušovanou čarou je vyznačena PSD původního, nekontaminovaného signálu (horní dvojice na obrázku 10). Tečkovaně je vyznačena PSD signálu kontaminovaného úzkopásmovým rušením (prostřední dvojice na obrázku 10), jež se od PSD nekontaminovaného signálu liší píkem v okolí kmitočtu 100 Hz. Čerchovaně je vyznačena PSD signálu kontaminovaného širokopásmovým rušením (dolní dvojice na obrázku 10), jež se od PSD nekontaminovaného signálu liší téměř v celém frekvenčním rozsahu.

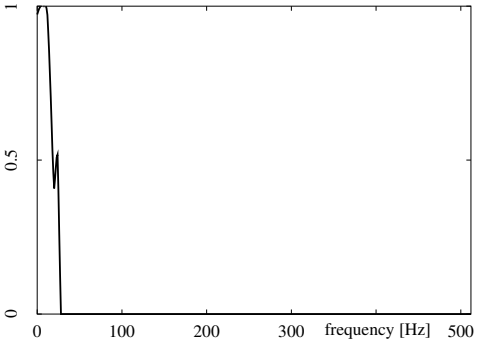
Odhad PSD (5.27) se určuje tak, že se provede Fourierova transformace vždy jen kousku signálu (tzv. okna), ať už obdélníkového nebo vytvarovaného do příhodnějšího tvaru (obvykle blízkého Gaussově křivce[7]), a PSD ze všech těchto oken, pokrývajících celý signál (okna se mohou či nemusejí vzájemně překrývat) se sečte. Čím je okno užší, tím je křivka odhadu PSD hladší, ale za cenu menšího rozlišení, a naopak. Podrobně o odhadu PSD viz např. [Press et al., 1997a].

Z odhadů PSD na obrázku 11 je zřejmé, že potlačení úzkopásmové kontaminace pomocí filtrace ve frekvenční doméně není obtížné. Postup zahrnuje tři kroky:

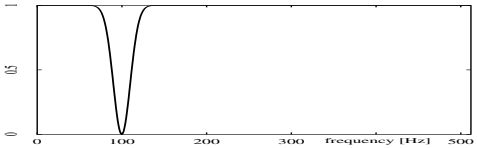
1. Spočteme Fourierovu transformaci $\hat{s}(f)$ kontaminovaného signálu $s(t)$.
2. Vynásobíme ji (vzorek po vzorku) filtrační funkcí $\Phi(f)$ sestrojenou na základě odhadu PSD kontaminovaného signálu. V tomto konkrétním případě bude mít filtrační funkce tvar tzv. zářezového filtru, tj. bude rovna jedné pro všechny frekvence až na úzké frekvenční pásmo, v němž se vyskytuje kontaminace. Filtrační funkci je vhodné volit dostatečně hladkou, opět z důvodů, jenž byly ozřejmeny v sekci 5.2.3. Filtrační funkce použitá pro odstranění úzkopásmové kontaminace našeho příkladu je zobrazena na obrázku 12 vpravo.
3. Na přefiltrované spektrum $\hat{s}_{\mathrm{clean}}(f) \equiv \Phi(f)\hat{s}(f)$ aplikujeme inverzní Fourie-

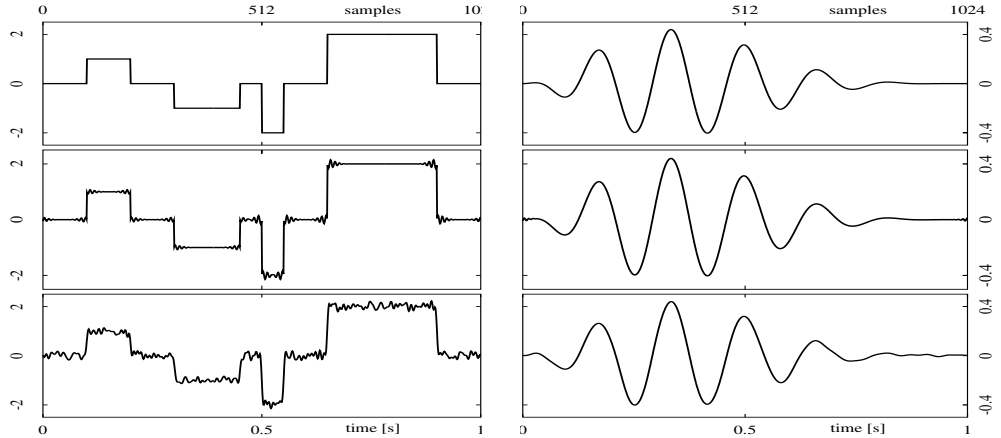
7. Tvar blízký Gaussově křivce je výhodný z důvodů naznačených v sekci 5.2.3.





Obrázek 12: Filtry použité pro dekontaminaci signálů. Vpravo je zářezový filtr centrovaný na 100 Hz, jímž byla odstraněna úzkopásmová kontaminace harmonickým signálem. Nahoře vlevo (vpravo) je Wienerův optimální filtr použitý k odstranění širokopásmové kontaminace nespojitého (spojitého) signálu.





Obrázek 13: V horní řadě jsou pro srovnání zopakovány původní, nekontaminované signály z horní řady obrázku 10. Prostřední a dolní řada odpovídají prostřední a dolní řadě na obrázku 10 s tím rozdílem, že kontaminované signály byly filtrovány. Úzkopásmová kontaminace (prostřední řada) byla odstraněna pomocí zářezového filtru z obrázku 12 vpravo. Širokopásmová kontaminace (dolní řada) byla filtrována pomocí Wienerovy optimální filtrace (viz obrázek 12 nahoře a [Press et al., 1997a]).

rovu transformaci, čímž dostaneme signál $s_{\mathrm{clean}}(t)$ v časové doméně zbavený kontaminace, ale bohužel také části odpovídající frekvenčnímu spektru ležícímu ve frekvenčním pásmu kontaminace. Právě v případě, kdy je toto pásmo úzké, se tímto způsobem dá dosáhnout velmi dobrých výsledků.

Pro filtraci byl vytvořen zářezový filtr znázorněný na obrázku 12 vpravo. Tento filtr odstraní ze spektra kontaminovaného signálu úzké pásmo, v němž se projevuje úzkopásmová kontaminace. Výsledek filtrace znázorněný v prostřední řadě obrázku 13 ukazuje, že úzkopásmová dekontaminace je velmi uspokojivá pro hladký signál, méně již pro nespojitý signál. Proč je tomu tak objasní pohled na spektra na obrázku 11. U hladkého signálu jsou dominantní frekvence zhruba do $\sim 20\,\mathrm{Hz}$, v oblasti kontaminace okolo $100\,\mathrm{Hz}$ jsou příspěvky do energie signálu $10^4\times$ až $10^5\times$ menší. Proto jejich potlačení zářezovým filtrem z obrázku 12 vpravo ovlivní tvar původního signálu jen velmi málo a výsledkem je téměř dokonale restaurovaný signál. Nespojitý signál má však díky svým ostrým hranám mnohem vyrovnanější, plošší spektrum, a aplikací zářezového filtru se zbavíme i důležité části spektra původního signálu. To se projeví oscilacemi v „rozích“ signálu.

Situace u širokopásmové kontaminace je složitější. Použití filtru je delikátnější, neboť musíme citlivě odstranit část spektra považovanou za šum. Obvykle to bývá vysokofrekvenční „chvost“, takže jsou vhodné filtry typu dolní propust. V našem případě jsme použili tzv. Wienerovy (optimální) filtrace [Press et al., 1997a]. Její praktické užití spočívá v tom, že vysokofrekvenční šumový „chvost“ (na obrázku 11 konstantní funkce $N^2 \approx 100$ pro nespojitý signál a $N^2 \approx 1$ pro spojitý signál) se extrapoluje doleva na zbytek odhadu spektra, přičemž filtrační funkce $\Phi(f)$ se určí jako

$$\Phi(f) = 1 - \frac{N^2}{C^2}\,, \tag{5.73}$$

kde C^2 je skutečný odhad spektra.

Výsledek optimální filtrace pomocí filtrů z obrázku 12 nahoře je znázorněn v dolní řadě obrázku 13. Širokopásmová dekontaminace je vcelku uspokojivá pro hladký signál, avšak výsledky pro nespojitý signál jsou špatné. Důvod je opět v tom, že nespojitý signál má mnohem plošší spektrum než hladký, spojitý signál — stačí pohled na obrázek 11 a srovnání odstupu PSD širokopásmově kontaminovaného od PSD původního čistého signálu v obou případech.

Pro všechny filtrace popsané a zobrazené v této sekci byl použit jednoúčelový program `foufires` z kolekce [Hledík, 2007].

Jiné metody odstraňování širokopásmové kontaminace založené na waveletové transformaci jsou naznačeny v Kapitole 6.

Shrnutí kapitoly: Naučili jste se důležité numerické metodě – rychlá Fourierově transformaci (FFT). Nyní chápete podstatu různých algoritmů na ní postavených, jež se uplatňují při zpracování jednorozměrného signálu (obvykle zvuku, ale obecně jakýchkoli experimentálně získaných jednorozměrných signálů) a dvourozměrného signálu (obvykle obrazových dat).

Další zdroje:

- Velmi pěkné animace jsou dostupné na http://www.math.muni.cz/~plch/nkpm/.
- Rozsahem přijatelná monografie o Fourierových řadách je např. [Hardy a Rogosinski, 1971].
- Základní informace o Fourierových řadách a Fourierově transformaci jsou uvedeny v každé knize o matematických metodách ve fyzice a technice, např. [Grega et al., 1974].
- Matematicky korektní teorie Fourierových řad je hezky podána v [Schwartz, 1972, Kap. V].

6. Waveletová analýza

Rychlý náhled kapitoly: V této kapitole bude objasněna moderní numerická metoda – waveletová (jinak též vlnková) transformace. Podobně jako u fourierovských metod byly matematické základy waveletové transformace známy relativně dlouho, avšak teprve nástup počítačů umožnil, aby se stala důležitým nástrojem matematické fyziky a analýzy signálů.

Cíle kapitoly:

- Pochopit motivaci pro zavedení waveletové transformace.
- Naučit se používat diskrétní waveletovou transformaci typu Daub.
- Naučit se používat diskrétní waveletovou transformaci typu Daub na filtraci signálů prahováním.

Klíčová slova kapitoly: Wavelet, DTWT; filtrační koeficienty, daubechiovské; vyhlazující filtr, detailní filtr; decimace; momenty; DaubN; waveletové koeficienty, koeficienty škálovací funkce; pyramidální algoritmus; energie signálu, energetická mapa signálu, kompaktifikace energie; práh, prahování

6.1. Motivace

Fourierova transformace a rychlá diskrétní implementace Fourierovy transformace (FFT), jejichž základy jsou probrány v Kapitole 5, patří k základním matematickým nástrojům pro analýzu a zpracování repetičních, časově invariantních nebo stacionárních signálů. Pro přechodné, nestacionární nebo v čase proměnné signály je Fourierova transformace vhodná méně, což je podmíněno přímo její podstatou — bázové funkce $\mathrm{e}^{-2\pi \mathrm{i} f t}$, z nichž Fourierova transformace (5.21) pomocí váhových koeficientů $\hat{s}(f)\,\mathrm{d}f$ „sestavuje“ signál $s(t)$, jsou nenulové na celé reálné ose. Je-li v signálu přítomen přechodový jev omezený na krátký časový úsek, odpovídající fourierovský obraz $\hat{s}(f)$ definovaný (5.21) konverguje pomalu a je rozprostřen přes široký rozsah frekvencí. Přetlumočeno v řeči principu neurčitosti (5.10): čím je přechodný jev v signálu časově omezenější, tím je jeho fourierovské spektrum širší. Nelze mít signál lokalizovaný jak v časové, tak ve frekvenční doméně.

Přirozeně se proto nabízí nahradit bázové funkce $e^{\pm 2\pi i f t}$ jinými funkcemi, jež by byly nenulové pouze na omezeném časovém intervalu (tzv. funkcemi s omezeným nosičem), nebo jež by alespoň dostatečně rychle konvergovaly k nule mimo určitý omezený interval. Takové bázové funkce by byly, na rozdíl od sinů a kosinů obsažených v komplexní exponenciele $e^{\pm 2\pi i f t}$, lokalizovány jak v čase, tak ve frekvenci, nebo přesněji řečeno, v charakteristické škále. Další odlišnost od Fourierovy transformace využívající jednoznačně definovanou bázi spočívá v tom, že waveletová transformace má k dispozici nekonečně mnoho bází. Zhruba se dá říci, že různé waveletové báze poskytují různý kompromis mezi kompaktností časové lokalizace a hladkostí bázových funkcí. Zatímco ve Fourierově analýze jsou bázové funkce předem dány a teprve poté se zkoumají vlastnosti transformace, ve waveletové analýze na základě požadovaných vlastností konstruujeme bázové funkce.

Pokuste se kvalitativně zdůvodnit, proč časově lokalizované signály mají široké Fourierovo spektrum.

Waveletová[8] analýza, jejíž základ (Haarův rozklad) historicky vznikl již v roce 1910 [Burrus et al., 1998], se v 80. letech 20. století dočkala prudkého rozvoje a přiřadila se k důležitým nástrojům v oblasti zpracování a analýzy signálu, numerické analýzy, matematického modelování, statistiky i v dalších vědeckých a technických aplikacích. Od těch dob jí bylo věnováno mnoho monografií a nespočet časopiseckých článků a bylo vyvinuta řada aplikací ve všech výše zmíněných oblastech.

V tomto dodatku se v zájmu kompaktnosti inspirujeme přístupem použitým v sekci 13.10 knihy [Press et al., 1997a], jenž je jasný, přímočarý a jeví se optimálním pro rychlé pochopení a aplikaci. Pro další studium doporučujeme knihu [Walker, 1999], hlubší vhled poskytnou např. knihy [Burrus et al., 1998, Percival a Walden, 2000].

8. Termín wavelet se do češtiny překládá jako vlnka. Vyjadřuje skutečnost, že bázové funkce jsou „malé“ (mají omezený nosič, jsou tedy lokalizovány v čase), a jejich střední hodnota je nulová, mají tedy „vlnový“ charakter.

6.2. Waveletové koeficienty typu Daub4

Podobně jako FFT je diskrétní waveletová transformace (dále DTWT)[9] lineární operací na reálném vzorkovaném vstupním signálu $s_1, s_2, \ldots, s_N$ o délce (počtu vzorků) $N \geq 2$ rovné celočíselné mocnině dvou, jež převádí na numericky odlišný výstupní signál $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_N$ téže délky. V dalším ukážeme, že počet operací násobení je podobně jako u rychlé Fourierovy transformace $\mathcal{O}(N \log_2 N)$. Dále budeme požadovat, aby DTWT byla ortogonální a tím také invertibilní — jinými slovy, vyjádříme DTWT v maticovém tvaru[10]

$$\underbrace{\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \vdots \\ \hat{s}_N \end{pmatrix}}_{\text{výstup}} = \underbrace{\begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1,N} \\ w_{21} & w_{22} & \ldots & w_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N,1} & w_{N,2} & \ldots & w_{N,N} \end{pmatrix}}_{W} \underbrace{\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}}_{\text{vstup}} . \tag{6.1}$$

9. Zkratka DTWT znamená discrete-time wavelet transform a zdůrazňuje skutečnost, že se jedná o transformaci časově diskretizovaného (vzorkovaného) signálu podobně jako FFT.
10. Skutečně používaný algoritmus, jehož podstata bude vysvětlena dále v sekci 6.3, se však liší od prostého maticového násobení s počtem operací násobení $\mathcal{O}(N^2)$.

Budeme požadovat, aby matice W reprezentující transformaci byla regulární a matice k ní inverzní byla rovna matici transponované,

$$W^{-1} = W^{\mathrm{T}}. \tag{6.2}$$

Existuje mnoho transformací typu (6.1), waveletová transformace se zabývá takovými typy, jimž odpovídající báze jsou — na rozdíl od sinů a kosinů ve fourierovské bázi $\mathrm{e}^{\pm 2\pi \mathrm{i} f t}$ — lokalizovány v čase, jak bylo naznačeno v úvodu. Pro DTWT to znamená, že matice W bude řídká (v každém řádku a sloupci obsahovat jen „několik málo“ nenulových komponent), na rozdíl od diskrétní fourierovské matice (5.66), jejíž prvky jsou obecně nenulové všechny.

Zavedeme nyní z praktického hlediska důležitou třídu waveletových transformací definovaných pomocí tzv. filtračních koeficientů, poprvé popsanou belgickou matematičkou INGRID DAUBECHIES, konkrétně typ Daub4 (číslo 4 udává počet filtračních koeficientů, jež budou zavedeny v následujících odstavcích).

Uvažujme čtyři reálná čísla c_0, c_1, c_2, c_3 a matici zkonstruovanou z nich následujícím způsobem (nulové komponenty matice nejsou explicitně vypsány a jsou nahrazeny prázdnými pozicemi):

$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}_3 \\ \hat{s}_4 \\ \vdots \\ \hat{s}_{N-3} \\ \hat{s}_{N-2} \\ \hat{s}_{N-1} \\ \hat{s}_N \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & & & & & \\ & c_0 & c_1 & c_2 & c_3 & & & & \\ & & c_0 & c_1 & c_2 & c_3 & & & \\ & & & c_0 & c_1 & c_2 & c_3 & & \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & \\ & & & & & c_0 & c_1 & c_2 & c_3 \\ c_3 & & & & & & c_0 & c_1 & c_2 \\ c_2 & c_3 & & & & & & c_0 & c_1 \\ c_1 & c_2 & c_3 & & & & & & c_0 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_{N-3} \\ s_{N-2} \\ s_{N-1} \\ s_N \end{pmatrix}. \tag{6.3}$$

Kdybychom položili $c_0 = c_1 = c_2 = c_3 = \frac{1}{4}$, dostali bychom výstupní signál v sloupečku na levé straně rovný vstupnímu signálu ze sloupečku na pravé straně vyhlazenému přes čtyři sousední vzorky.[11]

11. Poslední čtyři řádky matice jsou cyklicky „přetočeny“ na začátek, což znamená, že vstupní signál by měl mít periodické počáteční podmínky; to se dá vždy zajistit dodefinováním dostatečného počtu vzorků na začátku a konci signálu nulami.

Matice ve vztahu (6.3) reprezentuje vyhlazující filtr konvolučního typu (5.67), který ze signálu odstraní detaily kratší než čtyři vzorky a jehož výstupem je vyhlazený signál na levé straně. V případě všech čtyř koeficientů rovných $\frac{1}{4}$ by se jednalo o prostý klouzavý aritmetický průměr. Kdybychom naopak vzali místo filtru definovaného koeficienty c_0, c_1, c_2, c_3 filtr definovaný koeficienty $c_3, -c_2, c_1, -c_0$ (pořadí koeficientů je obráceno a každý sudý má opačné znaménko), bude jeho účinek opačný: pro konstantní signál dostaneme zřejmě nulový výstup, pro měnící se signál naopak obecně nenulový výstup. Takový filtr bude ze signálu extrahovat detaily ve smyslu doplňkových informací k výstupu vyhlazenému filtrem c_0, c_1, c_2, c_3, budeme je proto v dalším nazývat detailním filtrem.[12] Kdybychom v matici (6.3) nahradili vyhlazující filtr detailním filtrem, dostali bychom ve sloupečku na levé straně detaily signálu ze sloupečku na pravé straně.

Nyní oba popsané filtry, vyhlazující a detailní, zkombinujeme do jediné matice za cenu přeškálování vyhlazeného i detailního signálu na poloviční délku.

12. V kontextu teorie zpracování signálu se oba filtry nazývají kvadraturními zrcadlovými filtry.

V matici (6.3) a v odpovídajících pozicích výstupního signálu ve sloupečku na levé straně (6.3) škrtneme všechny sudé řádky (tato operace se nazývá decimace), a uprázdněné řádky v matici nahradíme detailním filtrem $c_3, -c_2, c_1, -c_0$ působícím na stejné vzorky signálu jako předcházející vyhlazující filtr, tj. všechny čtyři koeficienty detailního filtru budou ležet přesně pod koeficienty předcházejícího vyhlazujícího filtru. Výsledek bude vypadat takto:

$$\begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}_3 \\ \hat{s}_4 \\ \vdots \\ \hat{s}_{N-3} \\ \hat{s}_{N-2} \\ \hat{s}_{N-1} \\ \hat{s}_N \end{pmatrix} = \underbrace{\begin{pmatrix} c_0 & c_1 & c_2 & c_3 & & & & & & & \\ c_3 & -c_2 & c_1 & -c_0 & & & & & & & \\ & & c_0 & c_1 & c_2 & c_3 & & & & & \\ & & c_3 & -c_2 & c_1 & -c_0 & & & & & \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & & & \\ & & & & & c_0 & c_1 & c_2 & c_3 & & \\ & & & & & c_3 & -c_2 & c_1 & -c_0 & & \\ c_2 & c_3 & & & & & & c_0 & c_1 & & \\ c_1 & -c_0 & & & & & & c_3 & -c_2 & & \end{pmatrix}}_{F_N} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_{N-3} \\ s_{N-2} \\ s_{N-1} \\ s_N \end{pmatrix}. \tag{6.4}$$

Výstupem na levé straně (6.4) je vyhlazený signál decimovaný na poloviční délku na lichých pozicích $(\hat{s}_1, \hat{s}_3, \ldots, \hat{s}_{N-3}, \hat{s}_{N-1})$ a doplňující detailní signál obsažený rovněž v polovičním počtu vzorků na sudých pozicích $(\hat{s}_2, \hat{s}_4, \ldots, \hat{s}_{N-2}, \hat{s}_N)$.

Matice F_N v (6.4) musí splňovat podmínku ortogonality (6.2); snadno se ověří, že to nastane právě tehdy, když filtrační koeficienty budou splňovat dvě rovnice

$$\begin{aligned} c_0^2 + c_1^2 + c_2^2 + c_3^2 &= 1\,, \\ c_0 c_2 + c_1 c_3 &= 0\,. \end{aligned} \tag{6.5}$$

Místo toho, abychom explicitně specifikovali hodnoty koeficientů vyhlazujícího filtru, jak jsme zkusmo učinili bezprostředně za rovnicí (6.3), využijeme zbylé dva stupně volnosti naopak k určení nejprve detailního filtru, a teprve na jeho základě filtru vyhlazujícího. Konkrétně budeme požadovat, aby detailní filtr $c_3, -c_2, c_1, -c_0$ měl určité tzv. nulové momenty. Má-li filtr nulových prvních p momentů, znamená to, že akce filtru na signál tvořený polynomy nultého, prvního, ..., $(p-1)$-ho stupně dá nulu.

Proto budeme požadovat první dva momenty detailního filtru nulové ($p = 2$), čímž dostaneme zbývající dvě rovnice:

$$\begin{aligned} c_3 - c_2 + c_1 - c_0 &= 0\,, \\ 0c_3 - 1c_2 + 2c_1 - 3c_0 &= 0\,. \end{aligned} \tag{6.6}$$

Tyto podmínky znamenají, že pokud je signál konstantní nebo lineárně se měnící, je detailní signál extrahovaný z něj detailním filtrem roven nule.

Soustava čtyř rovnic pro čtyři neznámé (6.5) a (6.6) má jednoznačné (až na pravolevou reverzi) řešení

$$\begin{aligned} c_0 &= \frac{1+\sqrt{3}}{4\sqrt{2}} \doteq +0.48296291314453416\,, \\ c_1 &= \frac{3+\sqrt{3}}{4\sqrt{2}} \doteq +0.83651630373780794\,, \\ c_2 &= \frac{3-\sqrt{3}}{4\sqrt{2}} \doteq +0.22414386804201339\,, \\ c_3 &= \frac{1-\sqrt{3}}{4\sqrt{2}} \doteq -0.12940952255126037\,. \end{aligned} \tag{6.7}$$

Řešení (6.7) definuje filtr nazývaný Daub4. Počet nulových momentů p, a tím i počet filtračních koeficientů rovný $2p$ lze měnit. Pro $p = 1$ bychom dostali dvoukoeficientový filtr typu Daub2 (totožný s tzv. Haarovým–Welchovým filtrem),[13] hodnoty koeficientů pro p až do hodnoty 10, tj. typu Daub6, Daub8, ..., Daub20, lze najít např. v [Burrus et al., 1998].

Odvoďte podmínky ortogonality (6.5).

13. Podmínky ortogonality (6.5) se v případě Daub2 redukují na jedinou rovnici $c_0^2 + c_1^2 = 1$, podmínky vymizení momentů (6.6) se redukují na jedinou podmínku $c_1 - c_0 = 0$ zajišťující vymizení jediného (nultého) momentu, tj. vymizení akce filtru pouze pro konstantní polynom. Řešení této soustavy je triviální a dá (až na pravolevou reverzi) $c_0 = -c_1 = \sqrt{2}/2$.

6.3. Diskrétní waveletová transformace Daub4

Waveletová transformace typu Daub4 je definována hierarchickou aplikací matice (6.4) s filtračními koeficienty (6.7) na vstupní signál. Konkrétní algoritmus vypadá následovně:

1. Sloupeček s N vzorky vstupního signálu $s_1, s_2, \ldots, s_N$ násobíme zleva ortogonální maticí F_N o rozměru $N \times N$ z (6.4).
2. $N/2$ koeficientů jedenkrát vyhlazeného (a decimovaného) signálu přeskupíme z lichých pozic na prvních $N/2$ pozic, $N/2$ koeficientů detailního signálu přeskupíme ze sudých pozic na zbývajících $N/2$ pozic.
3. Na prvních $N/2$ pozic obsahujících jedenkrát vyhlazený signál aplikujeme ortogonální matici $F_{N/2}$ z (6.4), avšak poloviční rozměru $N/2 \times N/2$; zbylých $N/2$ koeficientů detailního signálu z bodu 2 podržíme beze změny.
4. $N/4$ dvakrát vyhlazených koeficientů přeskupíme na prvních $N/4$ pozic, na dalších $N/4$ pozic dáme „detaily z jednou vyhlazeného“ signálu, zbývajících $N/2$ (detailů původního signálu) zůstává nezměněných z bodu 2.

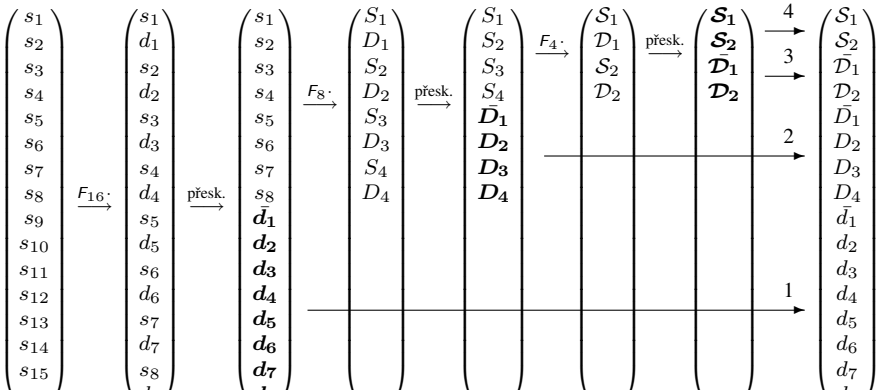
5. Na prvních $N/4$ pozic dvakrát vyhlazeného (a decimovaného) signálu aplikujeme ortogonální matici $F_{N/4}$ z (6.4), avšak čtvrtinového rozměru $N/4 \times N/4$.
6. $N/8$ třikrát vyhlazených (a decimovaných) koeficientů přeskupíme na prvních $N/8$ pozic, na dalších $N/8$ pozic dáme „detaily z dvakrát vyhlazeného“ signálu, zbývajících $N/4$ pozic (detailů jednou vyhlazeného signálu) a $N/2$ pozic (detailů původního signálu) zůstává nezměněných z bodů 2 a 4.
7. Tuto proceduru opakujeme dokud nedostaneme triviální počet (obvykle 2 nebo 1) koeficientů „vyhlazeného z vyhlazeného ... z vyhlazeného“ signálu.

Popsaná procedura se nazývá pyramidální algoritmus [Press et al., 1997a]. Nyní je zřejmé, proč je nutné, aby vstupní signál byl vzorkován na počet vzorků $N = 2^M$, kde $M \in \mathbb{N}$. Zároveň je jasný počet operací $\mathcal{O}(N \log_2 N)$: v každém stupni pyramidálního algoritmu provedeme $\mathcal{O}(N)$ násobení, přičemž počet stupňů pyramidy je $\mathcal{O}(\log_2 N)$. Pyramidální algoritmus je pro délku signálu $N = 16$ názorně demonstrován diagramem znázorněným na obrázku 14.

Ortogonální matice $F_{N/2}, F_{N/4}, F_{N/8}, \ldots, F_4$ postupně redukované na polovinu a vystupující v pyramidálním algoritmu lze chápat jako submatice umístěné v levém horním rohu plné matice F_N a doplněné na plnou velikost $N \times N$ jedničkami na diagonále a nulami na ostatních pozicích, např. matice F_8 doplněná na $N \times N$ je

$$\tilde{F}_8 = \underbrace{\begin{pmatrix} \boxed{F_8} & & \text{nuly} \\ & 1 & \\ & & \ddots \\ \text{nuly} & & 1 \end{pmatrix}}_{N \times N} . \tag{6.8}$$

Ortogonalita takto doplněných matic $\tilde{F}_N \equiv F_N, \tilde{F}_{N/2}, \tilde{F}_{N/4}, \tilde{F}_{N/8}, \ldots, \tilde{F}_4$ je zřejmá. Operaci přeskupování vyhlazených a detailních koeficientů lze rovněž popsat pomocí ortogonálních matic $P_N, P_{N/2}, P_{N/4}, \ldots, P_4$, jež lze doplnit na ortogonální matice $\tilde{P}_N \equiv P_N, \tilde{P}_{N/2}, \tilde{P}_{N/4}, \ldots, \tilde{P}_4$ plné velikosti $N \times N$ analogicky jako v (6.8).



Obrázek 14: Diagramatický popis pyramidálního algoritmu waveletové transformace pro $N = 16$. Komponenty výsledné waveletové transformace jsou vyznačeny tučným řezem. Po první aplikaci (6.4) a přeskupení získáme osm detailních koeficientů $d_1, \ldots, d_8$ tvořících druhou polovinu waveletové transformace (šipka 1), po druhé aplikaci (6.4) na horní polovinu a přeskupení získáme čtyři detaily z vyhlazení $D_1, \ldots, D_4$ tvořící druhou čtvrtinu waveletové transformace (šipka 2), po třetí aplikaci (6.4) na horní čtvrtinu a přeskupení získáme dva detaily z vyhlazených vyhlazení $\mathcal{D}_1, \mathcal{D}_2$ tvořící druhou osminu waveletové transformace (šipka 3) a dva koeficienty škálovací funkce $\mathcal{S}_1, \mathcal{S}_2$ tvořící první osminu waveletové transformace (šipka 4). Ve sloupečku zcela vpravo je vypsána výsledná waveletová transformace sestávající shora dolů ze dvou koeficientů škálovací funkce $\mathcal{S}_1, \mathcal{S}_2$ reprezentující třikrát vyhlazenou hodnotu vstupního signálu a ze tří úrovní detailů (od nejhrubších po nejjemnější) $\mathcal{D}_1, \mathcal{D}_2$, $D_1, \ldots, D_4, d_1, \ldots, d_8$.

Např. ortogonální přeskupovací matice P_8 a její doplnění na $\tilde{P}_8$ mají tvar

$$P_8 = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}, \quad \tilde{P}_8 = \underbrace{\begin{pmatrix} \boxed{P_8} & & \text{nuly} \\ & 1 & \\ & & \ddots \\ \text{nuly} & & 1 \end{pmatrix}}_{N\times N}. \tag{6.9}$$

Celý pyramidální algoritmus pak můžeme popsat jako násobení sloupečku se vzorky vstupního signálu zleva posloupností ortogonálních matic

$$\underbrace{\tilde{P}_4\tilde{F}_4\cdots\tilde{P}_{N/4}\tilde{F}_{N/4}\tilde{P}_{N/2}\tilde{F}_{N/2}\tilde{P}_N\tilde{F}_N}_{W}\begin{pmatrix} s_1\\ s_2\\ \vdots\\ s_N \end{pmatrix}. \tag{6.10}$$

Výsledná waveletová matice W daná součinem ortogonálních matic je pak sama také ortogonální, jak jsme požadovali v úvodu. Je však nutné upozornit, že maticový přístup, jehož počet operací je $\mathcal{O}(N^2)$, byl použit pouze k důkazu ortogonality a pro skutečnou implementaci je tedy příliš pomalý; vtip pyramidálního algoritmu je, jak jsme ukázali dříve, v rychlosti algoritmu dané počtem operací $\mathcal{O}(N \log_2 N)$.[14]

Detailních koeficienty všech úrovní (tj. všechny koeficienty až na první dva; na obrázku 14 jsou označeny jako různými variantami písmene „d“) se nazývají waveletové koeficienty, zatímco poslední dva koeficienty[15] označené na obrázku 14 jako $\mathcal{S}_1$ a $\mathcal{S}_2$ se nazývají koeficienty škálovací funkce.[16] Popsaný přístup se nazývá multiresolution analysis [Walker, 1999].

---

14. Faktor urychlení definovaný jako poměr počtu operací obou přístupů je $\mathcal{O}(N/\log_2 N)$. Pro $N \approx 10^6$, což odpovídá $\sim 20\,\mathrm{s}$ hudby v CD kvalitě, je faktor urychlení $\sim 5 \times 10^4$. Výpočetnímu času $\sim 1\,\mathrm{s}$ pomocí pyramidálního algoritmu by odpovídalo mnoho hodin maticového násobení!
15. Případně jediný poslední koeficient; záleží na tom, zda pyramidální algoritmus prodloužíme ještě o další stupeň.
16. Často se termín waveletové koeficienty volně užívá pro označení všech koeficientů přetransformovaného signálu, typu „d“ i $\mathcal{S}$.

K inverzi popsané DTWT stačí obrátit pyramidální algoritmus: začne se na nejmenší škále a diagram znázorněný na obrázku 14 se projde zprava doleva. Waveletová transformace se dá, podobně jako Fourierova transformace v sekci 5.2.4 Kapitoly 5, zobecnit na dvourozměrný nebo vícerozměrný případ [Press et al., 1997a, Walker, 1999]. Dvourozměrné wavelety se používají ve zpracování digitálního obrazu a hrají tak důležitou roli mj. v medicínské diagnostice.

6.4. Jak vypadají wavelety a škálovací funkce?

Nyní bychom rádi viděli, jak vlastně vypadají bázové funkce — wavelety a škálovací funkce — jakožto analogie sinusovek a kosinusovek komplexní exponenciely $\mathrm{e}^{\pm 2\pi \mathrm{i} f t}$. Libovolný signál o délce N můžeme vyjádřit v triviální ortogonální bázi tvořené N jednotkovými vektory o délce N

$$\boldsymbol{e}^1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{e}^2 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \quad \ldots, \quad \boldsymbol{e}^{N-1} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}, \quad \boldsymbol{e}^N = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{6.11}$$

takto:

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix} = \underbrace{\left(\boldsymbol{e}^1, \boldsymbol{e}^2, \ldots, \boldsymbol{e}^N\right)}_{\text{jednot. matice } N \times N} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix}. \tag{6.12}$$

Tentýž signálový vektor lze vyjádřit v jiné (netriviální) bázi $\boldsymbol{w}_1, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_N$ odpovídající hodnotám $\hat{s}_j$ transformovaným pomocí (6.1), tj.

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix} = \left(\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^N\right) \begin{pmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \vdots \\ \hat{s}_N \end{pmatrix} . \tag{6.13}$$

Porovnáním (6.12), (6.13) a s využitím vztahu (6.1) dostáváme ihned

$$\left(\boldsymbol{e}^1, \boldsymbol{e}^2, \ldots, \boldsymbol{e}^N\right) = \left(\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^N\right) W , \tag{6.14}$$

a s využitím (6.2) pak

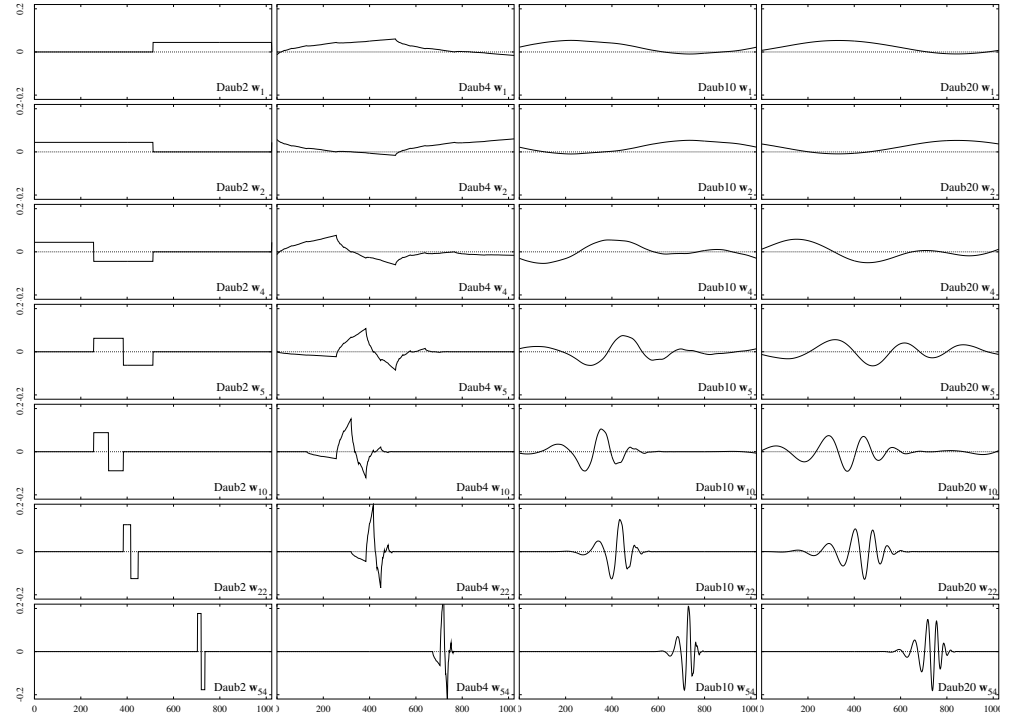
$$\left(\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^N\right) = \underbrace{\left(\boldsymbol{e}^1, \boldsymbol{e}^2, \ldots, \boldsymbol{e}^N\right)}_{\text{jednotková matice}} W^{\mathrm{T}} . \tag{6.15}$$

Uvědomíme-li si, že triviální bázové vektory sestavené do řádku tvoří jednotkovou matici, vidíme, že můžeme rovnou psát

$$\left(\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^N\right) \equiv W^{\mathrm{T}} . \tag{6.16}$$

Bázové vektory $\boldsymbol{w}^j$ jsou tvořeny sloupci transponované transformační matice W^{T}, neboli řádky transformační matice W. Odtud také plyne postup výpočtu vektorů nové báze $\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^N$: jednoduše pomocí inverzní DTWT zobrazíme signálový vektor odpovídající postupně vektorům triviální báze (6.11). Tento postup byl uplatněn při konstrukci grafů na obrázku 15 znázorňujícím příklady waveletů a škálovacích funkcí pro typ Daub2, Daub4, Daub10 a Daub20, jenž byly připraveny s pomocí programu `powersp` [Hledík, 2007] implementujícího waveletové rutiny z kolekce [Press et al., 1997a].

V praxi se používají další druhy waveletové transformace, např. typu Coiflet, Lemarie, B-spline a jiné, jejichž popis leží za možnostmi tohoto stručného úvodu. Bližší poučení lze najít v [Burrus et al., 1998, Percival a Walden, 2000].


Daub2 w1
Daub4 w1
Daub10 w1
Daub20 w1
Daub2 w2
Daub4 w2
Daub10 w2
Daub20 w2
Daub2 w4
Daub4 w4
Daub10 w4
Daub20 w4
Daub2 w5
Daub4 w5
Daub10 w5
Daub20 w5
Daub2 w10
Daub4 w10
Daub10 w10
Daub20 w10
Daub2 w22
Daub4 w22
Daub10 w22
Daub20 w22
Daub2 w54
Daub4 w54
Daub10 w54
Daub20 w54
0.2
0
-0.2
0
200
400
600
800
1000

(. . . popis k obrázku na předchozí straně)

Obrázek 15: Ukázka diskrétních waveletů a škálovacích funkcí o délce $N = 1024$ pro typy (zleva doprava) Daub2 (Haar–Welch), Daub4, Daub10 a Daub20. Dvě horní řady obsahují škálovací funkce $\boldsymbol{w}_1, \boldsymbol{w}_2$, ostatní řady pak vybrané wavelety ($\boldsymbol{w}_4, \boldsymbol{w}_5, \boldsymbol{w}_{10}, \boldsymbol{w}_{22}, \boldsymbol{w}_{54}$). Také ostatní wavelety mají stejný tvar (bráno shora dolů), liší se různou pozicí na horizontální ose a škálou (rozlišením). Střední hodnota škálovacích funkcí je nenulová, zatímco střední hodnota waveletů je nulová (pro verifikaci je v grafech čárkovaně vyznačena nulová hodnota). Při srovnávání sloupců zleva doprava je patrná vzrůstající hladkost. Dvoukoeficientové wavelety Daub2 v levém sloupci jsou nespojité a odpovídají $p = 1$, tj. nechávají vymizet detaily pouze při konstantním signálu. Hodí se pro transformaci nespojitých signálů à la Houston Skyline [Burrus et al., 1998]. Čtyřkoeficientové wavelety Daub4 v druhém sloupci zleva, jež jsme využili k zavedení waveletové transformace, jsou spojité, odpovídají $p = 2$, tj. dávají nulové detaily při lineárně se měnícím signálu, a mají bizarní tvary a zajímavé matematické vlastnosti. Mohou například reprezentovat po částech lineární funkci libovolných směrnic — ve správné kombinaci se všechny hroty patrné na waveletech vyruší. Desetikoeficientové wavelety Daub10 ve třetím sloupci zleva odpovídají $p = 5$ (vymizí nultý až čtvrtý moment, tj. dávají nulové detaily při nejvýše kvarticky se měnícím signálu). Konečně dvacetikoeficientové wavelety Daub20 v pravém sloupci odpovídají $p = 10$ (vymizí nultý až devátý moment, tj. dávají nulové detaily při signálu měnícím se podle polynomu až devátého stupně). Čím je vstupní signál hladší, tím je vhodnější pro reprezentaci vícekoeficientovou waveletovou transformací. Neznamená to ale, že čím více koeficientů, tím lépe; např. Daub4 je efektivnější při reprezentaci po částech lineárních signálů než Daub$N$ pro $N \geq 6$.

6.5. Nástin aplikace waveletové transformace ve zpracování signálu

Waveletová transformace se v praktickém zpracování signálu používá hlavně ke dvěma úzce souvisejícím postupům — kompresi dat a odstraňování šumu [Walker, 1999]. Jelikož se zde nelze pouštět do systematického výkladu, budeme pouze demonstrovat podstatu waveletového odstraňování šumu ze signálu na příkladu, včetně srovnání s metodou založenou na FFT.

Nejprve ukážeme, že ortogonální waveletová transformace (6.1) zachovává energii signálu definovanou pro diskrétní reálný signál[17]

$$E = \sum_{k=1}^{N} s_k^2 = \begin{pmatrix} s_1, s_2, \ldots, s_N \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix} = \mathbf{s}^{\mathrm{T}} \mathbf{s}\,, \tag{6.17}$$

kde $\mathbf{s}^{\mathrm{T}} \equiv (s_1, s_2, \ldots, s_N)$ je řádek transponovaný ke sloupečku $\mathbf{s}$.

17. Ve fyzice je energie či intenzita obvykle úměrná kvadrátu odpovídající dynamické proměnné. Vhodnou volbou jednotek energie vždy můžeme dosáhnout rovnosti kvadrátu dynamické proměnné.

Toto tvrzení je přímým důsledkem ortogonality transformační matice (6.2), jelikož pro energii transformovaného signálu můžeme psát

$$\hat{E} = \hat{\mathbf{s}}^{\mathrm{T}}\hat{\mathbf{s}} = \mathbf{s}^{\mathrm{T}}\underbrace{\mathbf{W}^{\mathrm{T}}\mathbf{W}}_{\mathbf{I}}\mathbf{s} = \mathbf{s}^{\mathrm{T}}\mathbf{s} = E\,. \tag{6.18}$$

Dále definujme energetickou mapu vyjadřující rozdělení energie ve vzorcích signálu. Energetickou mapu dostaneme tak, že kvadráty jednotlivých vzorků s_k^2 uspořádáme sestupně podle velikosti a z takto definované uspořádané posloupnosti $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$ vytvoříme relativní částečné součty

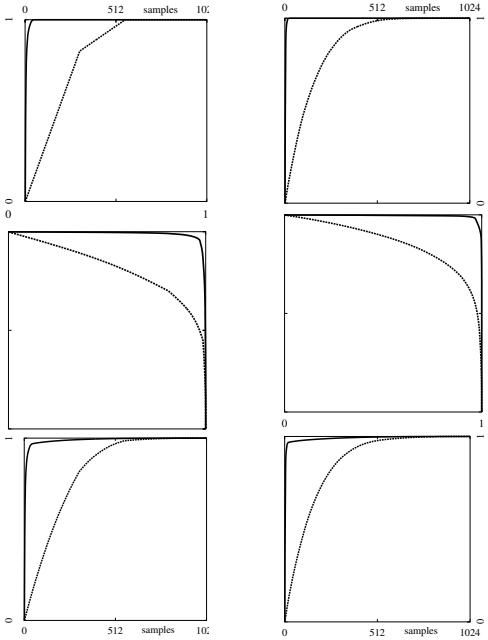
$$\mathcal{E}_j \equiv \left(\sum_{k=1}^{j}\varepsilon_k\right)/E\,, \quad j = 1, 2, \ldots, N\,. \tag{6.19}$$

Energetická mapa nenulového signálu (jenž má alespoň jeden vzorek nenulový) je neklesající posloupností čísel mezi 0 a 1,

$$0 < \mathcal{E}_1 \leq \mathcal{E}_2 \leq \cdots \leq \mathcal{E}_N = 1\,, \tag{6.20}$$

a podle rychlosti, jakou se přibližuje k jedničce, lze usuzovat na lokalizaci energie ve vzorcích. Posloupnost $\mathcal{E}_j$ nejprve prudce stoupá (započítává největší energie), posléze se její směrnice zmenšuje a zezdola konverguje k jedničce. Čím má energetická mapa zpočátku strmější průběh, tím více je energie lokalizována (kompaktifikována) v menším počtu vzorků.

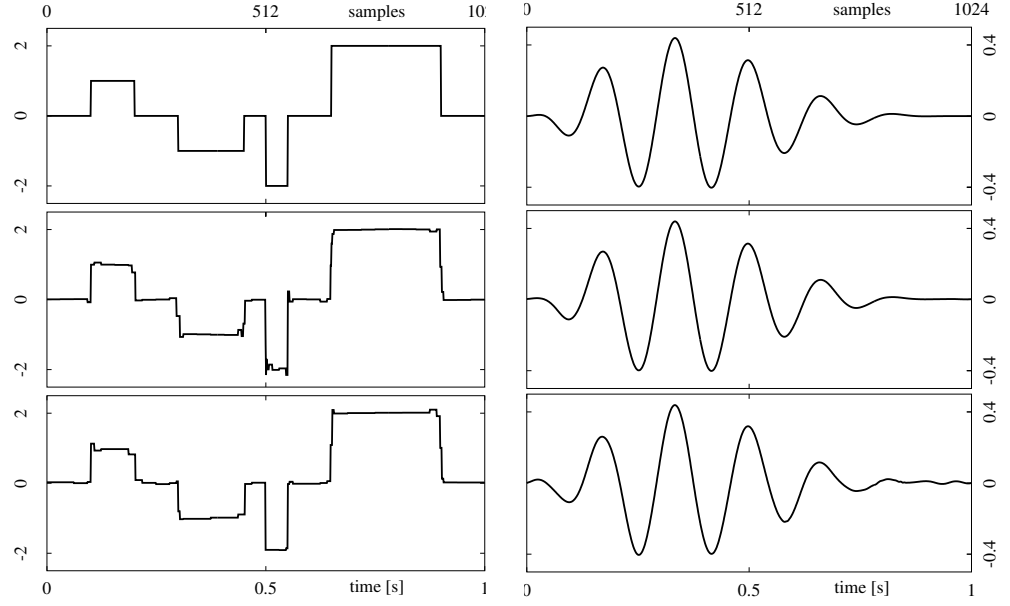
Diskrétní waveletová transformace typu Daub má schopnost zvyšovat kompaktifikaci energie. Znamená to, že jestliže v původním signálu dosáhne hladina energie $\mathcal{E}_L$ např. $99\,\%$ z celkové energie E pro nějaké $1 \leq L \leq N$, pak po waveletové transformaci dosáhne hladina stejné úrovně pro $1 \leq \hat{L} < L$, jinými slovy, $99\,\%$ energie signálu je po waveletové transformaci signálu „zahuštěno“ v méně ($\hat{L}$) vzorcích, než je tomu u původního signálu (v L vzorcích). Tuto důležitou vlastnost demonstruje obrázek 16. Míra kompaktifikace závisí na několika faktorech, mj. např. na výběru typu waveletové transformace. Obecně platí (viz obrázek 15), že pro reprezentaci nespojitých signálů (jako náš testovací obdélníkový signál) dávají lepší výsledky wavelety Daub s menším počtem koeficientů (zde konkrétně „krabicovité“ Daub2), a pro reprezentaci hladkých signálů dávají lepší výsledky hladké wavelety Daub s větším počtem koeficientů (zde konkrétně Daub20). To je ilustrace waveletového přístupu, kdy podle typu signálu (např. jeho hladkosti) vybíráme vhodnou waveletovou bázi, v níž bude signál optimálně reprezentován.



Obrázek 16: Energetické mapy testovacích signálů z obrázku 10. Řady i sloupce odpovídají obrázku 10, tj. v levém sloupci jsou energetické mapy pro nespojitý „obdélníkový“ signál, v pravém sloupci pro spojitý hladký signál; řady shora dolů odpovídají původnímu čistému signálu, úzkopásmové kontaminaci sinusovkou 100 Hz a širokopásmové kontaminaci bílým šumem. Čárkovaně jsou vyznačeny průběhy $\mathcal{E}_j$ pro netransformovaný signál, plně pro signál transformovaný. (Nespojitý signál byl transformován pomocí Daub2, jež se lépe hodí pro tento druh signálu, zatímco hladký spojitý signál byl transformován pomocí Daub20 — viz diskuse v popisu obrázku 15). Transformované signály mají mnohem větší kompaktifikaci energie než původní signály.

Vraťme se k záměru odstranění šumu z testovacích signálů na obrázku 10. Idea je velmi jednoduchá — nejprve se pomocí waveletové transformace kompaktifikuje energie signálu do relativně malého počtu vzorků. Poté se provede tzv. prahování (thresholding), kdy se z transformovaného signálu odstraní vzorky, jejichž absolutní hodnota je menší než práh (threshold) $T > 0$, a jejichž počet je roven číslu $\hat{L}$ odpovídajícímu podle energetické mapy zadanému procentu energie.[18] Jelikož energie šumu, jehož amplituda je menší než charakteristická amplituda signálu, je koncentrována do malých (podprahových) vzorků, a naopak, jelikož energie signálu je kompaktifikována do malého počtu relativně velkých vzorků, dojde k potlačení šumové kontaminace. Prahování lze provést selektivně pro každou úroveň pyramidálního algoritmu počínaje koeficienty škálovací funkce přes postupně se zjemňující detaily (viz obrázek 14). Tento přístup implementuje program `foufires` z kolekce [Hledík, 2007].

18. Popsaný algoritmus je tzv. tvrdé prahování (hard thresholding). Existuje několik druhů měkkého prahování (soft thresholding), při němž se podprahové vzorky potlačí méně nespojitým způsobem. Podrobně viz [Walker, 1999, Burrus et al., 1998].



Obrázek 17: Výsledek odstraňování šumu waveletovým prahováním. V horní řadě jsou pro srovnání zopakovány původní, nekontaminované signály z horní řady obrázku 10. Prostřední a dolní řada odpovídají prostřední a dolní řadě na obrázku 10 po aplikaci waveletového prahování. Výsledky pro hladký signál jsou srovnatelné nebo mírně lepší, u nespojitého signálu je výsledek zřetelně lepší.

Po prahování se obnoví signál pomocí inverzní waveletové transformace. Obnovený signál bude „chudší“ o malé procento energie převážně šumových složek. Výsledek aplikace prahování na testovací signály z obrázku 10 je demonstrován na obrázku 17.

Shrnutí kapitoly: Naučili jste se další důležité numerické metodě – waveletové analýze.

Další zdroje:

- Pro počáteční studium doporučujeme knihu [Walker, 1999].
- Hlubší záběr mají knihy [Burrus et al., 1998, Percival a Walden, 2000]

7. Poznámky

7.1. Proč nesignalizuje první bit mantisy speciální číslo?

V případě báze $\beta = 10$ by mohla být použit pro signalizaci nuly nebo subnormálních čísel nulovost prvního bitu mantisy. Tuto techniku však nelze použít v případě $\beta = 2$ kvůli skrývání prvního bitu normalizovaných čísel, který může být roven pouze 1. Příznakem speciálního čísla je vyhrazená speciální hodnota exponentu, tj. $00\ldots0$ nebo $11\ldots1$.

7.2. Strojové ε_-

Strojové ε_- je definováno jako nejmenší kladné FP číslo, pro které platí $1 - \varepsilon_- < 1$, nebo jinak, $1 - \varepsilon_-$ je nejbližší FP číslo menší než 1. V binární IEEE aritmetice je strojové epsilon $\varepsilon_- = 2^{-d_\mathrm{m}-1} = \varepsilon/2$ (při použití skrytého bitu) a $\varepsilon_- = 2^{-d_\mathrm{m}} = \varepsilon/2$ (bez použití skrytého bitu).

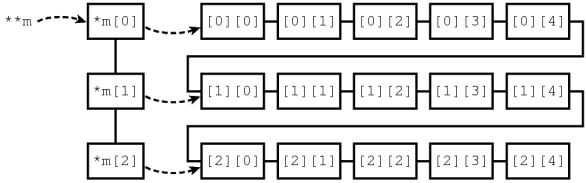
Analogicky strojové epsilon na intervalu $\langle 2^n, 2^{n+1})$ je $2^n\varepsilon$, speciálně pro $n = -1$ je na intervalu $\langle 1/2, 1)$ $\varepsilon_- = \varepsilon/2$.

7.3. Jak jsou matice uloženy v paměti?

Jazyk C Flexibilní způsob reprezentace matic v jazyce C je pointer na pole pointerů, tak jak je implementován ve vysokoúrovňových funkcích pro dynamickou alokaci a dealokaci paměti v souboru `nrutil.c` v [Press et al., 1997a]. Matice jsou ukládány po řádcích.

Příklad alokace paměti s použitím rutiny `matrix`:

```
float **m;
/* matice 3 ř./5 sl. */
m=matrix(0,2,0,4);
...
free_matrix(m,0,2,0,4);
```



Funkce z `nrutil.c` umožňují snadné definování vektorů, matic, resp. třídimenzionálních matic s libovolným rozsahem indexů. Např. v předcházejícím kódu je místo *zero-offset* matice `m` výhodnější použít *unit-offset* matici:

```
float **m;
/* matice 3 ř./5 sl. */
m=matrix(1,3,1,5);
...
free_matrix(m,1,3,1,5);
```

Detailní výklad alokace paměti pro jedno- až tří-rozměrná pole podává sekce 1.2 knihy [Press et al., 1997a].

Fortran 90/95 Jsou podporována až sedmirozměrná pole [Metcalf a Reid, 1999]. Na rozdíl od jazyka C jsou ve Fortranu 90/95 pole ukládána v takovém pořádku, že první index se mění nejrychleji, poslední nejpomaleji. Pro dvourozměrná pole (matice) to znamená, že jsou ukládána po sloupcích. Příkaz

```
REAL, DIMENSION(3,5) :: m
```

deklaruje *unit-offset* matici `m` s 3 řádky a 5 sloupci; zatímco *zero-offset* matice téže velikosti by byla deklarována příkazem

```
REAL, DIMENSION(0:2,0:4) :: m
```

Detailní výklad alokace paměti pro vícerozměrná pole podává [Metcalf a Reid, 1999].

8. Numerické knihovny

8.1. Svobodný software

GSL (GNU Scientific Library) Všeobecná knihovna pro C a C++. Dostupné na http://www.gnu.org/software/gsl/. Šířeno pod licencí GPL (GNU Public Licence).

LINPACK (LINear algebra PACKage) Knihovna pro lineární algebru vyvinutá v Aragonne National Laboratories.

LAPACK (Linear Algebra PACKage) Následovník knihovny LINPACK.

BLAS (Basic Linear Algebra Subroutines) Knihovna pro lineární algebru.

8.2. Komerční software

NR (Numerical Recipes) Všeobecná knihovna dodávaná ve formě zdrojových kódů v udržovaných jazykových mutacích C, Fortran 77, 90, C++; v neudržovaných starších verzích taky pro Pascal a BASIC. Dokumentace (knihy [Press et al., 1997a, Press et al., 1997b, Press et al., 1997c]) jsou dostupné online na http://www.nr.com/.

NAG libraries (Numerical Algorithm Group) Všeobecná knihovna.

IMSL ??

CERN Dokumentace dostupná online na http://wwwasdoc.web.ch/wwwasdoc.cernlib.html.

9. Popisy a zdrojové texty příkladů

Krátké zdrojové texty jsou zobrazeny přímo; předchází jim stručný popis a ‚neonový box' s názvem, z něhož lze program kliknutím spustit, např. char2ascii níže. Někdy je vypuštěn popis, spouští se stejným způsobem, např. storeprt. Dlouhé zdrojové texty nejsou zobrazeny, lze je prohlížet kliknutím na ‚neonový box' s názvem a spouštět kliknutím na symbol ➶ vlevo od názvu, např. longsum. Jakýkoli název v popisu v ‚neonové barvě' je prohlížecí link.

9.1. Ke kapitole 1

char2ascii Program zobrazí ASCII kód zadaného znaku v decimální, oktalové a hexadecimální soustavě.

```
#include <stdio.h>
int main(void)
{
    char a;
    printf("%s","\nPlease input character: ");
    scanf("%c",&a);
    printf("ASCII code of character \'%c\' is %3u (decimal)\n",a,a);
    printf("                                %3o (octal)\n",a);
    printf("                                 %2X (hexadecimal)\n",a);
    return 0;
}
```

charaddr Program zobrazí adresu (v hexadecimálním tvaru), na kterou v době běhu umístil zadaný znak.

```
#include <stdio.h>
int main(void)
{
    char a;
    printf("%s","\nPlease input character: ");
    scanf("%c",&a);
    printf("Character \'%c\' is located at address %p\n",a,&a);
    return 0;
}
```

straddr Program zobrazí hexadecimální adresy prvního a posledního znaku zadaného řetězce. Je-li zadaný řetězec delší než 8 znaků, bude oříznut.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char a[16];
    unsigned int i;
    printf("%s","\nPls input string of chars (will be truncated to 8 chars): ");
    scanf("%8s",a);
    printf("String length is:          %u\n",(i=strlen(a)));
    printf("Address of 1st/last char:  %p / %p\n",a,a+i-1);
    return 0;
}
```

unsigover Program přičte zadané kladné celé číslo i k maximálnímu neznaménkovému celému číslu zmenšenému o 3. Pro $i > 3$ dojde k přetečení.

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    unsigned int k=UINT_MAX-3,i;
    printf("\nMaximum unsigned int is: m = %u\n",k+3);
    printf("The same - 3 is:          k = %u\n",k);
    printf("Input small pos. integer i = ");
    scanf("%u",&i);
    printf("Sum                    k + i = %u\n",k+i);
    return 0;
}
```

unsigunder Varianta předchozího příkladu: program odečte zadané kladné celé číslo i od neznaménkové 3. Pro $i > 3$ dojde k podtečení.

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    unsigned int k=3,i;
    printf("\nMinimum unsigned int is: m = %u\n",k-3);
    printf("The same + 3 is:         k = %u\n",k);
    printf("Input small pos. integer i = ");
    scanf("%u",&i);
    printf("Difference           k - i = %u\n",k-i);
    return 0;
}
```

sigover Program přičte zadané kladné celé číslo i k maximálnímu znaménkovému celému číslu zmenšenému o 3. Pro $i > 3$ dojde k přetečení.

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    signed int k=INT_MAX-3,i;
    printf("\nMaximum signed int is:   m = %d\n",k+3);
    printf("The same - 3 is:         k = %d\n",k);
    printf("Input small pos. integer i = ");
    scanf("%d",&i);
    printf("Sum                  k + i = %d\n",k+i);
    return 0;
}
```

sigunder Varianta předchozího příkladu: program odečte zadané kladné celé číslo i od minimálního znaménkového celého čísla zvětšeného o 3. Pro $i > 3$ dojde k podtečení.

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    signed int k=INT_MIN+3,i;
    printf("\nMinimum signed int is:   m = %d\n",k-3);
    printf("The same + 3 is:         k = %d\n",k);
    printf("Input small pos. integer i = ");
    scanf("%d",&i);
    printf("Difference           k - i = %d\n",k-i);
    return 0;
}
```

unsig2sig Program přiřadí hodnotu zadaného neznaménkového celého čísla v proměnné u do znaménkové celočíselné proměnné s. Pozorujte chování programu v závislosti na tom, jestli je v u hodnota nejvýše rovná nebo větší než maximální znaménkové celé číslo.

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    unsigned int u;
    signed int s;
    printf("\nInput pos. integer le/gt %d: u = ",INT_MAX);
    scanf("%u",&u);
    s=u;
    printf("After conversion to signed:          s = %d\n",s);
    return 0;
}
```

byte_sex Vypíše endianitu systému v době běhu. Upraveno podle `bytesex.c`, © 2001 Jan „Yenya“ Kasprzak, ¡kas@fi.muni.cz¿).

```
#include <stdio.h>
int main(void)
{
    union {int i; char c[sizeof(int)];} u;
    u.i = 0; u.c[0] = 1;
    puts(u.i == 1 ? "\nLittle Endian" : "\nBig Endian");
    return 0;
}
```

uintaddr Program vypíše adresy, na něž v době běhu umístil neznaménkové celé číslo.

```
#include <stdio.h>
int main(void)
{
    unsigned int a;
    printf("%s","\nPlease input nonnegative number: ");
    scanf("%u",&a);
    printf("%u-bit number \'%u\' occupies\naddresses "
     "from %p to 0x%x\n",
     8*sizeof(a),a,&a,((unsigned int)(&a+1))-1);
    return 0;
}
```

sintaddr Program vypíše adresy, na něž v době běhu umístil znaménkové celé číslo.

```
#include <stdio.h>
int main(void)
{
    signed int a;
    printf("%s","\nPlease input number: ");
    scanf("%d",&a);
    printf("%u-bit number \'%d\' occupies\naddresses "
     "from %p to 0x%x\n",
     8*sizeof(a),a,&a,((unsigned int)(&a+1))-1);
    return 0;
}
```

epsilon Pro zadané kladné celé číslo p vypočte a zobrazí $(1.0 \oplus 2^{-p}) \ominus 1.0$ pro jednoduchou a dvojnásobnou přesnost. Pro dostatečně veliké p je $2^{-p} < \varepsilon$ a uvedený výraz bude nulový. Metodou pokusu a omylu umožňuje stanovit strojové epsilon pro obě přesnosti.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    float a;
    double b;
    int p;
    printf("\n%s","Please provide exponent: ");
    scanf("%d",&p);
    a = 1.0f + (float)pow(2.0,-p);
    printf("Single: %E\n",a-1.0);
    b = 1.0 + pow(2.0,-p);
    printf("Double: %E\n",b-1.0);
    return 0;
}
```

round Zadané číslo uloží do reálné proměnné v jednoduché přesnosti. Obsah proměnné pak zobrazí. Vyzkoušejte čísla 1.0, 2.0, 1.1, 0.5, 0.0625, 0.6, 0.62 a podle vlastního výběru a pozorujte zaokrouhlování.

```
#include <stdio.h>
int main(void)
{
    float x;
    printf("\n%s","Please provide a number: ");
    scanf("%f",&x);
    printf("Rounded to nearest FP:   %.12f\n",x);
    return 0;
}
```

messy-c Ilustrace některých aritmetických operací s výsledky $\pm\infty$ a `NaN`; verze v C.

```
#include <stdio.h>
int main(void)
{
    float a,b,c,d;
    a=0.0;
    b=-0.0;
    printf("\na = %f, b = %f\n",a,b);
    c=1.0/a;
    d=1.0/b;
    printf("1.0/a = %f,  1.0/b = %f\n",c,d);
    printf("1.0/a + 1.0/b   = %f\n",c+d);
    printf("1.0/a - 1.0/b   = %f\n",c-d);
    printf("1.0/b - 1.0/a   = %f\n",d-c);
    printf("(1.0/a)/(1.0/b) = %f\n",c/d);
    printf("(1.0/a)*(1.0/b) = %f\n",c*d);
    printf("a*(1.0/a)       = %f\n",a*c);
    return 0;
}
```

messy-f Ilustrace některých aritmetických operací s výsledky $\pm\infty$ a `NaN`; verze ve Fortranu 90.

```
PROGRAM messy
  IMPLICIT NONE
  REAL :: a,b,c,d
  a=0.0
  b=-0.0
  PRINT*
  PRINT '(a5,f9.6,a6,f9.6)', &
        " a = ", a, ", b = ", b
  c=1.0/a
  d=1.0/b
  PRINT '(a9,f9.6,a10,f10.6)', &
       " 1.0/a = ", c, ", 1.0/b = ", d
  PRINT*, "1.0/a + 1.0/b   = ", c+d
  PRINT*, "1.0/a - 1.0/b   = ", c-d
  PRINT*, "1.0/b - 1.0/a   = ", d-c
  PRINT*, "(1.0/a)/(1.0/b) = ", c/d
  PRINT*, "(1.0/a)*(1.0/b) = ", c*d
  PRINT*, "a*(1.0/a)       = ", a*c
  STOP
END PROGRAM messy
```

eps3single-optcpu Optimalizovaná verze (-O3) programu eps3single.

```
/* Code identical to eps3single.c. */
```

eps1double Verze programu eps1single ve dvojnásobné přesnosti.

```
/* Same as eps1single.c except of 'float' -> 'double'. */
```

eps2double Verze programu eps2single ve dvojnásobné přesnosti.

```
/* Same as eps2single.c except of 'float' -> 'double'. */
```

eps3double Verze programu eps3single ve dvojnásobné přesnosti.

```
/* Same as eps3single.c except of 'float' -> 'double'. */
```

eps3double-optcpu Optimalizovaná verze (-O3) programu eps3double.

```
/* Same as eps3single.c except of 'float' -> 'double'. */
```

machar_single Program pro diagnostiku FP parametrů systému v jednoduché přesnosti podle [Press et al., 1997a]. Význam položek výstupu je uveden v rámečku s popisem programu.

machar_double Program pro diagnostiku FP parametrů systému ve dvojnásobné přesnosti podle [Press et al., 1997a]. Význam položek výstupu je uveden v rámečku s popisem programu.

longsum Demonstrace dlouhé sumace při sestupném, vzestupném uspořádání a při použití Kahanovy sumační formule. Neoptimalizovaná verze (`-O0`).

longsum-O1 Demonstrace dlouhé sumace při sestupném, vzestupném uspořádání a při použití Kahanovy sumační formule. Stupeň optimalizace `-O1`.

longsum-O2 Demonstrace dlouhé sumace při sestupném, vzestupném uspořádání a při použití Kahanovy sumační formule. Stupeň optimalizace `-O2`.

storeprt

```
PROGRAM storeprt
  IMPLICIT NONE
  REAL :: x=1.0E-4
  PRINT*
  PRINT*, x
  IF ((1.0E+8 * x**2)==1.0) THEN
     PRINT*, 'Correct'
  ELSE
     PRINT*, 'Incorrect'
  END IF
  STOP
END PROGRAM storeprt
```

storeprt2

```
PROGRAM storeprt2
  IMPLICIT NONE
  REAL :: x=1.0E-4, epsilon=1.0E-7, w
  PRINT*
  PRINT*, x
  w=1.0E+8 * x**2
  IF (ABS(w-1.0) .LE. 0.5*epsilon*(ABS(w)+ABS(1.0))) THEN
     PRINT*, 'Correct'
  ELSE
     PRINT*, 'Incorrect'
  END IF
  STOP
END PROGRAM storeprt2
```

quiz_cor

```
PROGRAM quiz_cor
  IMPLICIT NONE
  REAL :: x=1.0/2.0
  PRINT*
  IF ((2.0*x) .EQ. 1.0) THEN
     PRINT*, 'Correct'
  ELSE
     PRINT*, 'Incorrect'
  END IF
  STOP
END PROGRAM quiz_cor
```

quiz_inc

```
PROGRAM quiz_inc
  IMPLICIT NONE
  REAL :: x=1.0/3.0
  PRINT*
  IF ((3.0*x) .EQ. 1.0) THEN
     PRINT*, 'Correct'
  ELSE
     PRINT*, 'Incorrect'
  END IF
  STOP
END PROGRAM quiz_inc
```

convers

```
PROGRAM convers
  IMPLICIT NONE
  REAL :: x=1.0E-4
  INTEGER :: i
  i=10000*x
  PRINT*
  PRINT*, i
  STOP
END PROGRAM convers
```

convers-optcpu

```
PROGRAM convers
  IMPLICIT NONE
  REAL :: x=1.0E-4
  INTEGER :: i
  i=10000*x
  PRINT*
  PRINT*, i
  STOP
END PROGRAM convers
```

sing2dbl

```
#include <stdio.h>
int main(void)
{
    float x=1.234567f;
    double y,z=1.234567;
    y=x;
    printf("\nx = %.12f\n",x);
    printf("y = %.12f\n",y);
    printf("z = %.12f\n",z);
    return 0;
}
```

insignid

```
PROGRAM insignid
  IMPLICIT NONE
  REAL :: x=100000.0,y=100000.1,z
  z=y-x
  PRINT*
  PRINT*, 'z = ', z
  STOP
END PROGRAM insignid
```

mem_reg-O0

```
SUBROUTINE ratio(r,s,t)
  IMPLICIT NONE
  REAL :: r,s,t
  t=s/r
END SUBROUTINE ratio

SUBROUTINE subtr(r,s,t)
  IMPLICIT NONE
  REAL :: r,s,t
  t=s-r
END SUBROUTINE subtr

PROGRAM mem_reg
  IMPLICIT NONE
  REAL :: a,b,x,y,z,c,direct, bycall
  DATA a /3.0/, b /10.0/
  DATA x /3.0/, y /10.0/
  direct=(y/x)-(b/a)
  CALL ratio(x,y,z)
  CALL ratio(a,b,c)
  CALL subtr(c,z,bycall)
  PRINT*
  PRINT*, direct-bycall
  STOP
END PROGRAM mem_reg
```

mem_reg-O2

```
SUBROUTINE ratio(r,s,t)
  IMPLICIT NONE
  REAL :: r,s,t
  t=s/r
END SUBROUTINE ratio

SUBROUTINE subtr(r,s,t)
  IMPLICIT NONE
  REAL :: r,s,t
  t=s-r
END SUBROUTINE subtr

PROGRAM mem_reg
  IMPLICIT NONE
  REAL :: a,b,x,y,z,c,direct, bycall
  DATA a /3.0/, b /10.0/
  DATA x /3.0/, y /10.0/
  direct=(y/x)-(b/a)
  CALL ratio(x,y,z)
  CALL ratio(a,b,c)
  CALL subtr(c,z,bycall)
  PRINT*
  PRINT*, direct-bycall
  STOP
END PROGRAM mem_reg
```

ordofop

```
PROGRAM ordofop
  IMPLICIT NONE
  REAL :: x=1.23456789,y=4.56789123,z=9.999999,r1,r2
  r1=(x*y)/z
  r2=(1.0/z)*x*y
  PRINT*
  PRINT* 'r1 - r2 = ',r1-r2
  r1=(x*y)
  r1=r1/z
  r2=1.0/z
  r2=r2*x
  r2=r2*y
  PRINT* 'r1 - r2 = ',r1-r2
  STOP
END PROGRAM ordofop
```

ordofop-optcpu

```
PROGRAM ordofop
  IMPLICIT NONE
  REAL :: x=1.23456789,y=4.56789123,z=9.999999,r1,r2
  r1=(x*y)/z
  r2=(1.0/z)*x*y
  PRINT*
  PRINT* 'r1 - r2 = ',r1-r2
  r1=(x*y)
  r1=r1/z
  r2=1.0/z
  r2=r2*x
  r2=r2*y
  PRINT* 'r1 - r2 = ',r1-r2
  STOP
END PROGRAM ordofop
```

unstable_single

```
#include <stdio.h>
#include <math.h>
#define DBL 0
#if (DBL==1)
#define MAX 46
#define REAL double
#else /* DBL */
#define MAX 26
#define REAL float
#endif /* DBL */
int main(void)
{
    REAL g=(sqrt(5.0)-1.0)/2.0,m=g,s=g,prevs=1.0,ss;
    int i;
    printf("\n%s\n",
           " i    s_(i+1) = s_1 * s_i   s_(i+1) ="
           " s_(i-1) - s_i   frac.err.[%]");
    printf("%s\n",
           "-------------------------------"
           "-----------------------------");
    for (i=0;i<=MAX;i++) {
  if (i==0)
      printf("%2d    %+.16f    %+.16f     %+9.2f\n",
       i,1.0,1.0,0.0);
  else {
      printf("%2d    %+.16f    %+.16f     %+9.2f\n",
       i,m,s,100.*(s-m)/m);
      m=m*g;
      ss=s;
      s=prevs-s;
      prevs=ss;
  }
    }
    return 0;
}
```

unstable_double

```
#include <stdio.h>
#include <math.h>
#define DBL 1
#if (DBL==1)
#define MAX 46
#define REAL double
#else /* DBL */
#define MAX 26
#define REAL float
#endif /* DBL */
int main(void)
{
    REAL g=(sqrt(5.0)-1.0)/2.0,m=g,s=g,prevs=1.0,ss;
    int i;
    printf("\n%s\n",
           " i    s_(i+1) = s_1 * s_i   s_(i+1) ="
           " s_(i-1) - s_i   frac.err.[%]");
    printf("%s\n",
           "-------------------------------"
           "-----------------------------");
    for (i=0;i<=MAX;i++) {
  if (i==0)
      printf("%2d    %+.16f    %+.16f     %+9.2f\n",
                  i,1.0,1.0,0.0);
  else {
      printf("%2d    %+.16f    %+.16f     %+9.2f\n",
       i,m,s,100.*(s-m)/m);
      m=m*g;
      ss=s;
      s=prevs-s;
      prevs=ss;
  }
    }
    return 0;
}
```

cancerr

```
PROGRAM cancerr
  IMPLICIT NONE
  REAL :: a=1.0,b=1.0E+8,c=1.0E+8,d,x1,x2
  d=SQRT(b**2-4.0*a*c)
  x1=(-b-d)/(2.0*a) ! this is ok
  x2=(-b+d)/(2.0*a) ! suffers from cancellation
  PRINT*
  PRINT*, 'x1 = ', x1, ', x2 = ', x2
  x2=(2.0*c)/(-b-d) ! this is much better
  PRINT*, 'x1 = ', x1, ', x2 = ', x2
  STOP
END PROGRAM cancerr
```

9.2. Ke kapitole 2

xgaussj Volá rutinu gaussj implementující Gaussovu–Jordanovu eliminaci s plnou pivotací, a aplikuje ji na řešení soustavy lineárních rovnic s maticemi a pravými stranami definovanými v souboru `matrx1.dat`. Každá matice je invertována, vynásobena svou vlastní inverzí pro kontrolu, že vyjde jednotková matice. Poté jsou vektory řešení vynásobeny původní maticí a porovnány s exaktním pravými stranami načteným z `matrx1.dat`.

xludcmp Volá rutinu ludcmp implementující LU dekompozici na sadu matic definovaných v souboru `matrx1.dat`. Ke každé vstupní matici zobrazí příslušnou dolní a horní matici dekompozice a pro kontrolu také jejich součin. (Permutace řádků způsobená částečnou implicitní pivotací je pro snadné porovnání vrácena do původní podoby. Toto v součinnosti s rutinou xlubksb není nutno provádět; této rutině se předá informace o permutaci.)

xlubksb Volá rutinu lubksb implementující přímou a zpětnou substituci (jež používá rozklad dodaný rutinou ludcmp implementující LU dekompozici). Aplikuje ji na řešení soustavy lineárních rovnic s maticemi a pravými stranami definovanými v souboru `matrx1.dat`. Pro každou matici je porovnán vektor pravé strany a součin matice s vektorem řešení.

xtridag Volá rutinu tridag pro řešení tridiagonálních soustav definovaných v souboru `matrx2.dat`. Pro každou matici je pro porovnání ukázán vektor pravé strany, vektor řešení a pro porovnání s pravou stranou také součin matice s vektorem řešení.

9.3. Ke kapitole 3

xpolint Mějme ekvidistantní datový soubor s N body ($\texttt{xa[i]} = i\pi/N, \texttt{ya[i]} = \sin(\texttt{xa[i]})$) na intervalu $\langle 0, \pi\rangle$ a datový soubor ($\texttt{xa[i]} = i/N, \texttt{ya[i]} = \exp(\texttt{xa[i]})$) na intervalu $\langle 0, 1\rangle$; $i \in \{1, \ldots, N\}$. Počet bodů N si můžete interaktivně volit. Rutina polint implementující Nevilleův algoritmus pro konstrukci polynomiálního interpolačního polynomu $N-1$ stupně je použita pro výpočet interpolovaných hodnot v deseti bodech odpovídajících $N = 10$ a s hodnotami x mírně ‚posunutými‘, aby ani pro $N = 10$ nebyly totožné s datovým souborem. Vyzkoušejte postupně $N = 2, 3, 4, 5, 6, 7, 10, 20, 30, 50, 99, 110$ a vysvětlete chování interpolované hodnoty a odhadu chyby.

xratint Funguje podobně jako předchozí ukázka. Datový soubor je tvořen 6 body ($\texttt{xa[i]} = i/3, \texttt{ya[i]} = f(\texttt{xa[i]})$), $i \in \{1, \ldots, 6\}$, kde $f(x) = x\mathrm{e}^{-x}/[(x-1)^2+1]$. Rutina ratint implementující Bulirschův–Stoerův algoritmus pro konstrukci diagonální racionální interpolační funkce je použita pro výpočet interpolovaných hodnot v deseti bodech $x = i/5$, $i \in \{1, \ldots, 10\}$. Pozorujte vztah mezi skutečnou a interpolovanou hodnotou a odhadem chyby poskytnutým rutinou.

xspline Datový soubor je tvořen 20 body ($\texttt{xa[i]} = i\pi/20, \texttt{ya[i]} = \sin(\texttt{xa[i]})$), $i \in \{1, \ldots, 20\}$; zadány jsou hodnoty první derivace $f(x) \equiv \sin x$ na obou koncích souboru. Rutina spline počítá hodnoty f'' pro každou hodnotu `x[i]`. Porovnejte skutečné a vypočtené hodnoty f''.

xsplint Funguje na tomtéž datovém souboru jako `xpolint` pro fixní $N = 10$, pro tytéž ‚posunuté' hodnoty x, a pro tytéž funkce f, ale s využitím interpolace splajny. Vlastní interpolaci provádí rutina splint, jíž dodává potřebné druhé derivace f'' rutina spline testovaná v předchozí ukázce.

xlocate Máme-li monotónně seřazená data `x[j]`, $j \in \{1, \ldots, N\}$, konkrétně `x[j]` $= \exp(j/20) - 74$ a $N = 100$, rutina locate pro zadané x nalezne index j tak, že x se nachází mezi x_j a x_{j+1}. V prvním sloupci je hodnota x k lokalizaci, ve druhém příslušný index j, ve třetím a čtvrtém pak ‚obklopující' (*bracketing*) hodnoty x_j a x_{j+1}. Je-li $j = 0$ ($j = N$), není x v tabelovaném rozsahu. Program potom oznamuje ‚lower lim' pro $x < x_1$ a ‚upper lim' pro $x > x_N$.

xhunt Funguje analogicky jako předchozí ukázka, ale místo rutiny locate používá hunt. Rozdíl mezi oběma rutinami je vyložen v sekci 3.4 o pomocných rutinách.

xpolcoe Opět zcela stejné vstupy jako program `xpolint`, ale místo rekurentní konstrukce polynomiálního interpolačního polynomu pomocí Nevilleova algoritmu používá rutinu polcoe implementující Vandermondeovu metodu pro určení koeficientů interpolačního polynomu, z něhož posléze počítá interpolované hodnoty $f(x)$.

xpolcof Funguje stejně jako předchozí ukázka, ale používá rutinu polcof implementující druhou metodu popsanou v sekci 3.5.

9.4. Ke kapitole 4

xtrapzd Tato ukázka volá rutinu trapzd.

xqtrap Tato ukázka volá rutinu qtrap.

xqsimp Tato ukázka volá rutinu qsimp.

xmidpnt Tato ukázka volá rutinu midpnt.

xqromb Tato ukázka volá rutinu qromb.

xqromo Tato ukázka volá rutinu qromo.

10. Výsledky úloh

10.1. Ke kapitole 2

Výsledky pro Hilbertovu matici (2.23) s pravou stranou (2.24) (v levém sloupci jsou výsledky metody GJE, v pravém metody LUD); výpočty v single precision:

$n = 2$

1.000000	1.000000
1.000000	1.000000

$n = 3$

0.999999	1.000000
1.000003	1.000001
0.999997	1.000000

$n = 5$

1.000093	1.000077
0.998328	0.998631
1.007029	1.005741
0.989579	0.991502
1.005027	1.004095

$n = 7$

1.003043	1.001827
0.879234	0.929520
2.158665	1.664267
-3.489593	-1.542318
9.208048	5.605855
-6.075985	-2.942873
3.318666	2.284817

$n = 10$

1.004925	1.001638
0.763174	0.910963
3.698094	2.131162
-11.197721	-4.745139
26.268555	14.838495
-21.886929	-14.954043
7.301528	7.918557
-2.563211	1.743061
10.784071	0.710839
-4.176050	0.443014

$n = 40$

1.098166	0.974728
-3.117915	2.125518
40.871414	-11.567856
-135.787582	58.316116
142.124435	-123.446800
104.189758	141.746704
-152.453537	-105.790329
-144.891083	45.021286
136.653976	89.682381
117.013100	-87.071922
-351.817566	-29.869486
348.688385	-10.388809
-211.423004	-21.820902
748.234863	117.836197
-769.546997	-156.579575
-381.192932	188.822083
951.063477	-77.117409
160.762695	57.782467
-824.813293	-28.273844
223.029053	26.366032
-448.803192	2.331474
871.923340	-75.117470
-561.326050	-14.417043
-230.847702	-92.698799
-189.062378	97.934837
-213.439301	-4.205263
818.300537	102.137138
170.662384	-98.220963
502.512787	62.345520
-320.290253	-128.182449
953.731384	229.899002
-652.625000	-271.073608
-292.389893	268.862915
-1140.913086	-246.733063
92.301315	244.324707
-356.277039	-48.134335
1157.578491	-54.620586
-283.460693	78.414551
619.719849	-241.446243
-455.924164	151.842148

Obě metody dávají s výjimkou $n \leq 3$ chybné výsledky, pro $n \geq 7$ zcela nepoužitelné. Klíč k porozumění: vypočtěte determinant Hilbertovy matice (2.23) pro uvedená n.

Výsledky pro ‚kosinovou matici (2.25)‘ s pravou stranou (2.26) (v levém sloupci jsou výsledky metody GJE, v pravém metody LUD); výpočty v single precision:

$n = 2$

```
1.000000    1.000000
1.000000    1.000000
```

$n = 3$

```
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
```

$n = 5$

```
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
```

$n = 7$

```
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
```

$n = 10$

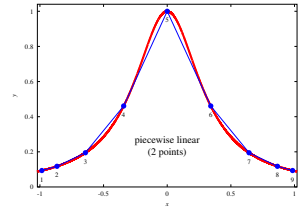
```
1.000000    0.999984
1.000000    0.999991
1.000000    0.999999
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000003
0.999999    1.000039
1.000000    1.000000
1.000000    1.000000
```

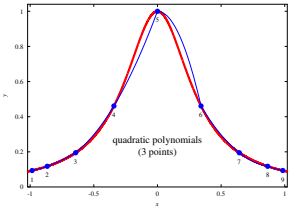
$n = 40$

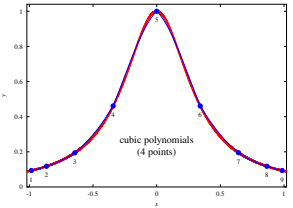
```
1.000133    0.999922
0.999764    1.000006
1.000008    0.999999
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000001    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
0.999998    1.000002
1.000004    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
0.999998    1.000001
1.000003    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000000    1.000000
1.000001    1.000000
```

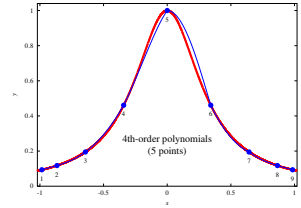
Obě metody dávají pro všechna n očekávané výsledky blízké matematicky exaktním výsledkům. Proč?

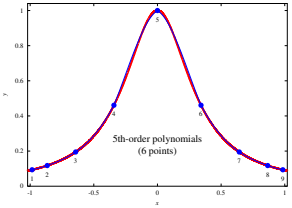
10.2. Ke kapitole 3

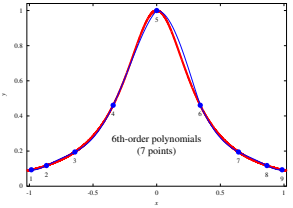


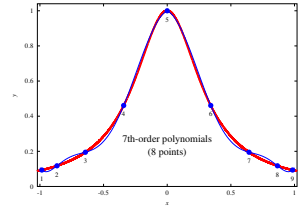


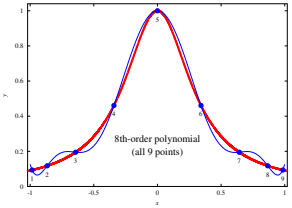


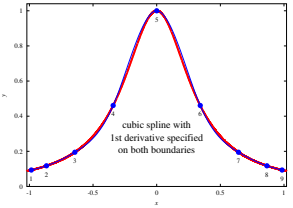


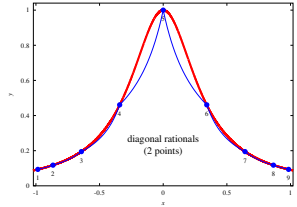


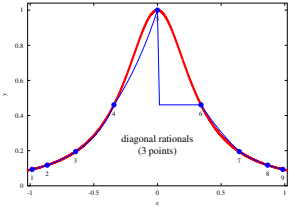


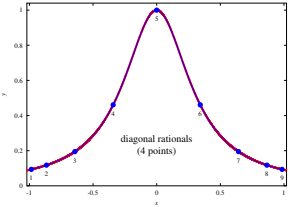


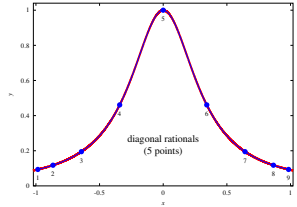


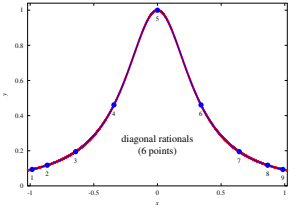


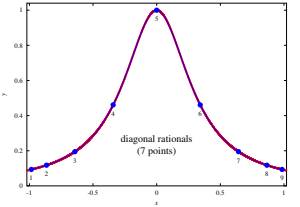

diagonal rationals
(2 points)
x
y

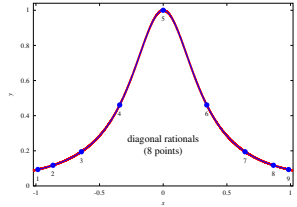

diagonal rationals
(3 points)
x
y

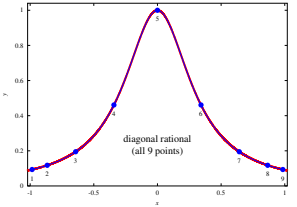

diagonal rationals
(4 points)
x
y

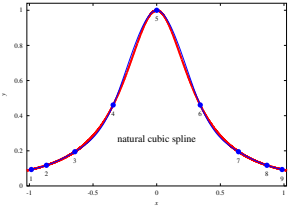

diagonal rationals
(5 points)
x
y


diagonal rationals
(6 points)
x
y


diagonal rationals
(7 points)
x
y


diagonal rationals
(8 points)
x
y


diagonal rational
(all 9 points)
x
y


natural cubic spline
x
y

Reference

[Abramowitz a Stegun, 1964] Abramowitz, M. a Stegun, I. A. (1964).
Handbook of Mathematical Functions, svazek 55 *Applied Mathematics Series*.
National Bureau of Standards, Washington.
Reprinted 1968 by Dover Publications, New York.

[Bratley et al., 1983] Bratley, P., Fox, B. L., a Schrage, E. L. (1983).
A Guide to Simulation.
Springer-Verlag, New York.

[Burrus et al., 1998] Burrus, C. S., Gopinath, R. A., a Guo, H. (1998).
Introduction to Wavelets and Wavelets Transforms. A Primer.
Prentice Hall, New Jersey.

[Dahlquist a Bjorck, 1974] Dahlquist, G. a Bjorck, A. (1974).
Numerical Methods.
Prentice Hall, Englewood Cliffs, NJ.

[Došlá et al., 2002] Došlá, Z., Plch, R., a Sojka, P. (2002).
Matematická analýza s programem Maple. 2. Nekonečné řady.
Masarykova univerzita v Brně, Brno.
CD-ROM.

[Forsythe et al., 1977] Forsythe, G. E., Malcolm, M. A., a Moler, C. B. (1977).
Computer Methods for Mathematical Computations.
Prentice-Hall, Englewood Cliffs, NJ.

[Frigo a Johnson, 0000] Frigo, M. a Johnson, S. G. (0000).
FFTW – the Fastest Fourier Transform in the West.

[Goldberg, 1991] Goldberg, D. (1991).
What every computer scientist should know about floating-point arithmetic.
ACM Comput. Surveys, 23(1):5–48.

[Grega et al., 1974] Grega, A., Kluvanec, D., a Rajčan, E. (1974).

Matematika pre fyzikov.
Slovenské pedagogické nakladateľstvo, Bratislava, 1. vydání.

[Hardy a Rogosinski, 1971] Hardy, G. H. a Rogosinski, W. W. (1971).
Fourierovy řady.
SNTL/Alfa, Praha, 1. vydání.

[Hledík, 2007] Hledík, S. (2007).
FoSpAToX — Fourier and Spectral Applications Tools/Toys for Experiments.
collections of programs for experiments in 1D spectral and wavelet analysis.

[Kasprzak, 2004] Kasprzak, J. (2004).
Unix.
http://uf.fpf.slu.cz/.

[Knuth, 1981] Knuth, D. E. (1981).
Seminumerical Algorithms, svazek 2 *The Art of Computer Programming*.
Addison-Wesley, Reading, MA, 2. vydání.

[Kopeček a Kučera, 1989] Kopeček, I. a Kučera, J. (1989).
Programátorské poklesky.
Mladá fronta – program Start, Praha.

[Metcalf a Reid, 1999] Metcalf, M. a Reid, J. (1999).
Fortran 90/95 Explained.
Oxford University Press, New York, 2nd vydání.

[Percival a Walden, 2000] Percival, D. B. a Walden, A. T. (2000).
Wavelet Methods for Time Series Analysis.
Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge.

[Press et al., 1997a] Press, W. H., Teukolsky, S. A., Vetterling, W. T., a Flannery, B. P. (1997a).
Numerical Recipes in C: The Art of Scientific Computing.
Cambridge University Press, Cambridge, 2nd vydání.

[Press et al., 1997b] Press, W. H., Teukolsky, S. A., Vetterling, W. T., a Flannery, B. P. (1997b).
Numerical Recipes in Fortran 77: The Art of Scientific Computing.

Cambridge University Press, Cambridge, 2nd vydání.
[Fortran Numerical Recipes, Vol 1].

[Press et al., 1997c] Press, W. H., Teukolsky, S. A., Vetterling, W. T., a Flannery, B. P. (1997c).
Numerical Recipes in Fortran 90: The Art of Parallel Scientific Computing.
Cambridge University Press, Cambridge.
[Fortran Numerical Recipes, Vol 2, with foreword by M. Metcalf].

[Prudnikov et al., 1981] Prudnikov, A. P., Bryčkov, J. A., a Maričev, O. I. (1981).
Integraly i rjady. Elementarnyje funkcii.
Nauka, Moskva.

[Práger a Sýkorová, 2004] Práger, M. a Sýkorová, I. (2004).
Jak počítače počítají.
Pokroky Mat. Fyz. Astronom., 49(1):32–45.

[Ralston, 1978] Ralston, A. (1978).
Základy numerické matematiky.
Academia, Praha.
2. vydání.

[Rektorys, 1995] Rektorys, K. (1995).
Přehled užité matematiky.
Prometheus, Praha.

[Sandu, 2001] Sandu, A. (2001).
Lecture Notes: Introduction to Fortran 95 and Numerical Computing. A Jump-Start for Scientists and Engineers.
http://www.cs.mtu.edu/~asandu/Courses/CS2911/fortran_notes/main.html.

[Schwartz, 1972] Schwartz, L. (1972).
Matematické metody ve fyzice.
SNTL – Nakladatelství technické literatury, Praha.

[Segeth, 1998] Segeth, K. (1998).
Numerický software.
Karolinum – nakladatelství Univerzity Karlovy, Praha.

[Stroud a Secrest, 1966] Stroud, A. H. a Secrest, D. (1966).
Gaussian Quadrature Formulas.
Prentice Hall, Englewood Cliffs, NJ.

[Vetterling et al., 1993] Vetterling, W. T., Teukolsky, S. A., Press, W. H., a Flannery, B. P. (1993).
Numerical Recipes Example Book (C).
Cambridge University Press, Cambridge, 2nd vydání.

[Walker, 1999] Walker, J. S. (1999).
A Primer on Wavelets and Their Scientific Applications.
Chapman & Hall/CRC Press, Boca Raton.