LJETNI SEMESTAR 2012/2013

NUMERIČKE METODE I MATEMATIČKO MODELIRANJE

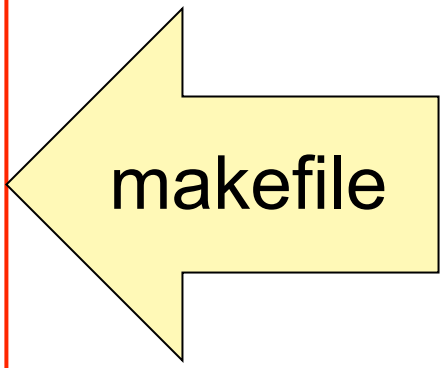
2. PREDAVANJE

"makefile" za kompajliranje programa u Linux/Unix-u

- U Linuxu/Unixu je za kompajliranje programa često praktično napraviti tzv. "makefile" → skripta koja uključuje naredbe i opcije za kompajliranje

- Korištenjem "makefile" izbjegava se ponovno utipkavanje naredbi za Kompajliranje nakon svake promjene u programu, dovoljno je samo otipkati make

```
# General makefile for c - choose PROG =   name of given program

# Here we define compiler option, libraries and the  target
CC= c++ -Wall
PROG= myprogram

# Here we make the executable file
${PROG} :           ${PROG}.o

                    ${CC} ${PROG}.o -o ${PROG}

# whereas here we create the object file

${PROG}.o :         ${PROG}.cpp
                    ${CC} -c ${PROG}.cpp
```

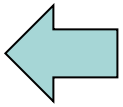


"makefile" za kompajliranje programa u Linux/Unix-u

- "makefile" se može koristiti za različite programske jezike i kompajlere, moguće su i kombinacije npr. Fortrana i C programskog jezika
- upotreba "makefile" je posebno korisna u slučajevima kada treba povezati više različitih izvornih datoteka koda
- Npr. različite funkcije su u različitim datotekama i koriste različite "header" datoteke (*.h)

"makefile" koji povezuje fortran i C++ podprograme

```
FCOMP = ifort -assume nounderscore -warn alignments
CCOMP = icpc -Wno-deprecated
LIBS = -lm -cxxlib

OBJ = ftchrpa.o edmonds.o

.cpp.o:
        $(CCOMP) -c  $<

.f.o:
        $(FCOMP) -c  $<

run : $(OBJ)
        $(FCOMP) -o run $(OBJ) $(LIBS)

ftchrpa.o : ftchrpa.f

edmonds.o : edmonds.cpp common.h edmonds.h sla.h
```



Primjer sa
Intel kompajlerom

REALNI BROJEVI I NUMERIČKA PRECIZNOST

- jedan od ključnih aspekata matematičkog modeliranja i računalne fizike je numerička preciznost
- u stvaranju dobrog algoritma treba voditi računa o nepreciznostima i greškama do kojih može doći u proračunima
- računalo nije u mogućnosti manipulirati sa realnim brojevima izraženim sa više od određenog fiksnog broja znamenki
- to znači da su realni brojevi u zapisu sa pomičnom točkom uvijek ograničeni na preciznost koja ovisi o računalu
- Uvijek treba voditi računa da računalo može prikazati realni broj samo do neke određene preciznosti

TIPOVI ZA POHRANJIVANJE PODATAKA U RAČUNALU

type in C/C++ and Fortran 90/95	bits	range
char/CHARACTER	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int/INTEGER (2)	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned short int	16	0 to 65535
signed short int	16	-32768 to 32767
int/long int/INTEGER(4)	32	-2147483648 to 2147483647
signed long int	32	-2147483648 to 2147483647
float/REAL(4)	32	$3.4e^{-38}$ to $3.4e^{+38}$
double/REAL(8)	64	$1.7e^{-308}$ to $1.7e^{+308}$
long double	64	$1.7e^{-308}$ to $1.7e^{+308}$

REALNI BROJEVI I NUMERIČKA PRECIZNOST

- Primjer: pretpostavimo da računalo može raditi sa realnim brojevima preciznosti od 5 znamenki iza decimalne točke
- Treba izračunati vrijednosti funkcije f(x) za male vrijednosti x:

$$f(x) = \frac{1-cos(x)}{sin(x)}$$

Ako pomnožimo brojnik i nazivnik sa 1+cos(x), dobijemo ekvivalentni izraz:

$$f(x) = \frac{sin(x)}{1+cos(x)}$$

Izaberimo x=0.007, što daje uz zadanu preciznost,

$$sin(0.007) \approx 0.69999 \cdot 10^{-2}$$
$$cos(0.007) \approx 0.99998$$

REALNI BROJEVI I NUMERIČKA PRECIZNOST

Prvi izraz za f(x) daje

$$f(x) = \frac{1-0.99998}{0.69999 \cdot 10^{-2}} = \frac{0.2 \cdot 10^{-4}}{0.69999 \cdot 10^{-2}} = 0.28572 \cdot 10^{-2}$$

a drugi izraz rezultira sa

$$f(x) = \frac{0.69999 \cdot 10^{-2}}{1+0.99998} = \frac{0.69999 \cdot 10^{-2}}{1.99998} = 0.35000 \cdot 10^{-2}$$

U prvom izrazu zbog ograničene preciznosti u brojniku nakon oduzimanja ostaje samo jedna relevantna znamenka→ gubitak preciznosti rezultira netočnim rezultatom zbog približnog poništavanja dva vrlo bliska broja

da smo imali preciznost zadanu na 6 znameniki, oba izraza bi dala isti rezultat

Gubitak numeričke preciznosti zbog greške zaokruživanja, nekoliko vodećih znamenki je izgubljeno zbog oduzimanja približno jednakih brojeva

REALNI BROJEVI I NUMERIČKA PRECIZNOST

- Potencijalni izvori greške u radu sa realnim brojevima:

- 1) Pozitivno prekoračenje("overflow") – pozitivni eksponent prelazi maksimalnu dozvoljenu vrijednost (npr. 308 za double precision (64 bita) gdje je max. vrijednost $1.7e^{308}$)
- Moguće rješenje → npr. korištenje logaritma brojeva

- 2) Negativno prekoračenje("underflow") – negativni eksponent je manji od minimalne dozvoljene vrijednosti (npr. -308 za double precision (64 bita))

- 3) Greška zaokruživanja ("Roundoff error") – nema dovoljno mjesta da se pohrane sve znamenke broja (npr. za double precision, znamenke iza 15te su izgubljene u zapisivanju)

$$x = 1.2345678911121314\,68 = 0.1234567891112131\,468 \times 10^1$$

REALNI BROJEVI I NUMERIČKA PRECIZNOST

- ako sumiramo alternirajuće nizove velikih brojeva, oduzimanje dva velika broja može voditi do greške zaokruživanja jer nisu zadržane sve relevantne znamenke

- 4) Gubitak preciznosti → preporučljiv zapis jednadžbi preko bezdimenzionalnih varijabli (npr. u nuklearnoj fizici gdje su udaljenosti mjerene u 10^{-15} m, prikladno je zamijeniti varijable bezdimenzionalnom veličinom iznosa 1)

- ***Primjer:*** prepostavimo da imamo dva realna broja 1 i 10^{-8} koje želimo zbrojiti u zapisu sa jednostrukom preciznošću (single precision)

→ informacija o drugom broju je jednostavno izgubljena (kod zbrajanja računalo izjednačava eksponente dva broja koja treba zbrojiti)

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

- **Primjer:** Primjeniti tri različita algoritma u proračunu funkcije e^{-x} za x=0-100 u koracima po 10 i istražiti moguće izvore greške u numeričkom modeliranju funkcije:
- a) jednostavnim razvojem

$$\exp(-x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!}$$

- b) primjenom rekurzivne relacije

$$\exp(-x) = \sum_{n=0}^{\infty} s_n = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!} \qquad s_n = -s_{n-1}\frac{x}{n}$$

- c) računanjem e^x rekurzijom, i zatim invertirati

$$\exp(x) = \sum_{n=0}^{\infty} s_n \qquad \exp(-x) = \frac{1}{\exp(x)}$$

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

```
#include <iostream>
// type float:  32 bits precision
// type double: 64 bits precision
#define    TYPE              double
#define    PHASE(a)          (1 - 2 * (abs(a) % 2))
#define    TRUNCATION        1.0E-10
// function declaration
TYPE factorial(int);

int main()
{
    int    n;
    TYPE   x, term, sum;
    for(x = 0.0; x < 100.0; x += 10.0)  {
      sum  = 0.0;                       //initialization
```

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

```
    n     = 0;
    term  = 1;
    while(fabs(term) > TRUNCATION)  {
        term =  PHASE(n) * (TYPE) pow((TYPE) x,(TYPE) n) / factorial(n);
        sum += term;
        n++;
    }  // end of while() loop
    cout << '' x =’’ << x << '' exp = '' << exp(-x) << '' series = '' <<
        sum;
    cout  << '' number of terms = " << n << endl;
  } // end of for() loop
  return 0;
} // End: function main()
```

```
//      The function factorial()
//      calculates and returns n!

TYPE factorial(int n)
{
   int  loop;
   TYPE fac;
   for(loop = 1, fac = 1.0; loop <= n; loop++)  {
      fac *= loop;
   }
   return fac;
} // End: function factorial()
```

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

Rezultat algoritma za računanje eksponencijalne funkcije:

x	$\exp(-x)$	Series	Number of terms in series
0.0	0.100000E+01	0.100000E+01	1
10.0	0.453999E-04	0.453999E-04	44
20.0	0.206115E-08	0.487460E-08	72
30.0	0.935762E-13	-0.342134E-04	100
40.0	0.424835E-17	-0.221033E+01	127
50.0	0.192875E-21	-0.833851E+05	155
60.0	0.875651E-26	-0.850381E+09	171
70.0	0.397545E-30	NaN	171
80.0	0.180485E-34	NaN	171
90.0	0.819401E-39	NaN	171
100.0	0.372008E-43	NaN	171

Za male vrijednosti x slaganje je dobro, za veće vrijednosti dolazi do katastrofalnog učinka zbog gubitka preciznosti u proračunu

Za vrijednosti x veće od 70 dolazi do greške prekoračenja ("overflow")
NaN → "not a number"

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

- 171! se nalazi izvan limita zadanog za "double precision" varijablu, računalo postavlja factorial(171) na nulu i kao posljedicu imamo dijeljenje sa nulom

- općenito treba izbjegavati varijable sa jednostrukom preciznošću (float) u matematičkom modeliranju fizikalnih problema, preciznost nije dovoljna i veća je mogućnost greške

- problem prekoračenja može se izbjeći korištenjem rekurzivne formule u proračunu,

$$\exp(-x) = \sum_{n=0}^{\infty} s_n = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!}$$

$$s_n = -s_{n-1}\frac{x}{n}$$

Rekurzivne formule se često koriste u aproksimiranju funkcija, primjer su Besselove funkcije, Hermite-ovi i Laguerre-ovi polinomi

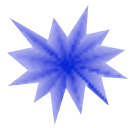
GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

```
// program to compute exp(-x) without factorials
using namespace std;
#include <iostream>
#define  TRUNCATION      1.0E-10

int main()
{
    int         loop, n;
    double      x, term, sum;

    for(loop = 0; loop <= 100; loop += 10){
      x     = (double) loop;            // initialization
      sum   = 1.0;
      term  = 1;
      n     = 1;
      while(fabs(term) > TRUNCATION){
           term *= -x/((double) n);
           sum  += term;
           n++;
      } // end while loop
      cout << ''x ='' << x << ''exp = '' << exp(-x) << ''series = '' << sum;
      cout  << ''number of terms = " << n << endl;
    } // end of for loop
}  //     End: function main()
```

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

Rezultat unaprijeđene verzije algoritma sa rekurzivnim računanjem članova niza

x	$\exp(-x)$	Series	Number of terms in series
0.000000	0.10000000E+01	0.10000000E+01	1
10.000000	0.45399900E-04	0.45399900E-04	44
20.000000	0.20611536E-08	0.56385075E-08	72
30.000000	0.93576230E-13	-0.30668111E-04	100
40.000000	0.42483543E-17	-0.31657319E+01	127
50.000000	0.19287498E-21	0.11072933E+05	155
60.000000	0.87565108E-26	-0.33516811E+09	182
70.000000	0.39754497E-30	-0.32979605E+14	209
80.000000	0.18048514E-34	0.91805682E+17	237
90.000000	0.81940126E-39	-0.50516254E+22	264
100.000000	0.37200760E-43	-0.29137556E+26	291



Iako je uklonjen problem prekoračenja, za veći x rezultati i dalje daju grešku, smanjenje "truncation" neće pomoći

GREŠKE U NUMERIČKOM PRORAČUNU e^{-x}

- problem u računu je što se za veći n dobivaju veliki brojevi različitih predznaka, što generira grešku zbog ograničenja u duljini zapisa realnog broja
- Rješenje u ovom slučaju je jednostavno: s obzirom da je $e^{x}=1/e^{-x}$, problem alternirajućeg predznaka u članovima niza se može riješiti izračunom funkcije e^{x}, rezultat na kraju treba samo invertirati
- Međutim, treba i dalje voditi računa o tome da funkcija e^{x} vrlo brzo raste, pa može preći doseg dozvoljen za varijablu tipa double

GREŠKE U NUMERIČKOM PRORAČUNU ∑(1/n)

- **Primjer:** treba izračunati red $s_1 = \sum_{n=1}^{N} \frac{1}{n}$ koji je konačan za konačni N.
- Ista suma može se izračunati i primjenom alternativnog zapisa:

$$s_2 = \sum_{n=N}^{1} \frac{1}{n}$$

- Kada sumiramo ova dva reda analitički, dobije se $s_2 = s_1$
- Međutim, u numeričkom proračunu $s_2 \neq s_1$ *zbog akumulirane greške u zaokruživanju !*
- *Npr. u proračunu sa jednostrukom preciznošću (float) za N=1000000:*

$s_1 = 14.35736$

$s_2 = 14.39265$

GREŠKE U NUMERIČKOM PRORAČUNU ∑(1/n)

- prijelazom na zapis broja sa dvostrukom preciznošću (double) *rezultati postaju egzaktno podudarni*, međutim razlike su ipak i dalje moguće u slučaju većeg N, npr. za $N=10^8$ (ali tek nakon 9. znamenke)

- Odabrana preciznost varijabli u računu je od iznimne važnosti za rezultat i grešku do koje može doći

- Preporučljivo je koristiti dvostruku preciznost u svim proračunima koji koriste realne brojeve

- Odabir odgovarajućeg algoritma je jednako važan (kao što je ilustrirano u primjeru funkcije e^{-x})

Zadatak 2:

- Primjeniti tri različita algoritma u proračunu funkcije e^{-x} za x=0-100 u koracima po 10 i istražiti moguće izvore greške u numeričkom modeliranju funkcije. Računati posebno slučajeve sa float i double tipovima varijabli i rezultate ispisivati tablično.

- 1) direktnim razvojem

$$\exp(-x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!}$$

- 2) primjenom rekurzivne relacije

$$\exp(-x) = \sum_{n=0}^{\infty} s_n = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!} \qquad s_n = -s_{n-1}\frac{x}{n}$$

- 3) računanjem e^x rekurzijom, i zatim invertirati

$$\exp(x) = \sum_{n=0}^{\infty} s_n \qquad \exp(-x) = \frac{1}{\exp(x)}$$