

MAT 202E

Numerical Methods

Assist.Prof.Dr. Onur TUNÇER
tuncero@itu.edu.tr



- Introduction
- Errors
- Representation of real numbers in a computer
- When can we solve a problem?
- Numerical Disasters
- Flowcharting
- Code Compilation
- Modular Programming

- Many problems can be formulated in simple mathematical equations
This does not mean, that they are solved easily!
- For any application, you need numbers
⇒ Numerical Methods!
- Needed for most basic things:
 $\exp(3)$, $\sqrt{7}$, $\sin(42^\circ)$, $\log(5)$, π
- Often, modeling and numerical calculations can help in design, construction, safety
- Note: many everyday's problems are so complicated that they cannot be solved yet
⇒ Efficiency is crucial

- Numerical methods: Numerical approximation of solutions to understood problems
- Numerical representation of real numbers has far-reaching consequences
- Two main objectives
 - **quantify errors**
Approximation without error estimation is useless
 - **increase efficiency**
Solutions which take years or need more resources than you have are useless
- Nowadays, many fields depend on numerics

Partial Differential Equations in Conservation Form to Solve for the Non-Reacting Compressible Flow of an Ideal Gas

$$\text{Mass: } \frac{\partial \rho}{\partial t} + \text{div}(\rho u) = 0$$

$$x\text{-momentum: } \frac{\partial(\rho u)}{\partial t} + \text{div}(\rho u u) = -\frac{\partial p}{\partial x} + \text{div}(\mu \text{ grad } u) + S_{Mx}$$

$$y\text{-momentum: } \frac{\partial(\rho v)}{\partial t} + \text{div}(\rho v u) = -\frac{\partial p}{\partial y} + \text{div}(\mu \text{ grad } v) + S_{My}$$

$$z\text{-momentum: } \frac{\partial(\rho w)}{\partial t} + \text{div}(\rho w u) = -\frac{\partial p}{\partial z} + \text{div}(\mu \text{ grad } w) + S_{Mz}$$

$$\text{Internal energy: } \frac{\partial(\rho i)}{\partial t} + \text{div}(\rho i u) = -p \text{ div } u + \text{div}(k \text{ grad } T) + \Phi + S_i$$

$$\text{Equations of state: } p = p(\rho, T) \text{ and } i = i(\rho, T)$$

$$\text{e.g. for perfect gas: } p = \rho R T \text{ and } i = C_v T$$

Errors!

- Round-off Error: Finite precision
numerical calculations are almost always approximations
- Truncation error: a calculation has to stop

Examples:

- Approximation (e.g. finite Taylor series)
- Discretization

It is crucial to know when to stop (i.e. when a calculation is converged!). To check this, change parameters (e.g. step size, number of basis states) and check result.

- Modelling error

- Truncation errors are problem specific
- Often, every step involves an approximation, e.g. a finite Taylor series
- The truncation errors accumulate
- Often, truncation errors can be calculated

- Precision of representation of numbers is finite
 - Errors accumulate!
- a real number x can be represented as

$fl(x) = x \cdot (1 + \varepsilon)$: floating point computer representation

$|fl(x) - x| = \varepsilon x$ absolute error (often also Δx)

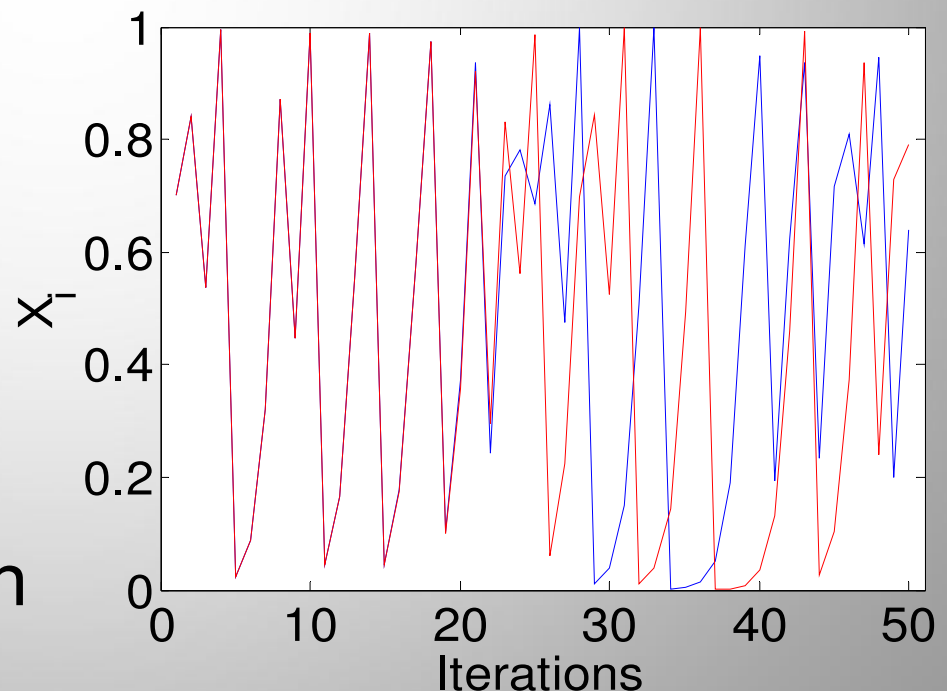
$|fl(x) - x| / x = \varepsilon$ relative error

Example: Logistic Map

$$x_{i+1} = r * x_i * (1 - x_i)$$

$$x_0 = 0.7 ; r = 4$$

single and double precision



Digital Representation of Numbers

- Real numbers: floating point representation
 - double (64 bit) standard!
 - single (32 bit) less precision, half memory
- Integer: signed and unsigned (8,16,32 bit)
 - `a=uint8(255), b=int16(32767), c=int32(231-1)`
 - Integers use less space, calculations are precise!
- Complex:
 - `w=2+3i; v=complex(x,y)` x,y can be matrices!
- Boolean: `true (=1)` or `false (=0)`
 - Strings: `s='Hello World'`
- special values:
 - `+inf, -inf, NaN` Infinity, Not-a-Number
 - check with `isinf(x), isnan(x)`

Floating Point Standard IEEE 754

- Normalized:

$$N = 1.f \cdot 2^p$$

Denormalized:

$$N = 0.f \cdot 2^p$$

bit (double)	use
63	sign
62 – 52	Exponent p
51 – 0	Significant f

- Limits of double precision (64bit) FP numbers
realmax (10^{308}), realmin (10^{-308}) largest, smallest real number
eps (10^{-16}) accuracy
- strictly speaking, FP numbers are **not associative** and **not distributive**, but they are to a very good approximation if used reasonably
- mathematically floating point numbers are **not a field!**
Many fundamental theorems of analysis and algebra have to be used with care!

- not all real numbers can be represented as FP
 $0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1-1$ $-1\text{E-}16$
- Floating point calculations are not precise!
 $\sin(\pi)$ is not 0, $1\text{E}20+1-1\text{E}20$ is not 1
- never compare two floats ($a==b$) directly
 try $100*(1/3)/100 == 1/3$ FALSE!!!
 use $\text{abs}(a-b)<\text{eps}$ ($\text{eps}=2.2\text{E-}16$)
- be careful with mixing integer and FP
 $i=\text{int32}(1);$
 $i/2$ produces 1 as $i/2$ stays integer!
 $i/3$ produces 0

! This is much more dangerous in C and FORTRAN etc.

Solution: explicit type conversion

$\text{double}(i)/2$ produces 0.5

What can be solved
numerically?

- Suppose we want to evaluate $f(x)$ with *perfect algorithm*
- we have FP number $x+\Delta x$ with error Δx

$$\Delta f(x) = f(x+\Delta x) - f(x) \approx f'(x)\Delta x \quad (\text{if } f \text{ differentiable})$$

Relative error:

Definition: condition number

$$\frac{\Delta f(x)}{f(x)} \approx \frac{x f'(x)}{f(x)} \frac{\Delta x}{x}$$

- $\gamma \gg 1$: problem ill-conditioned
- γ small: problem well-conditioned

$$\gamma(x) = \left| \frac{x f'(x)}{f(x)} \right|$$

- Let's try a simple calculation:
 $99 - 70 * \sqrt{2}$ (≈ 0.00505)
- Suppose we have 1.4 as approximation for $\sqrt{2}$
- We have 2 mathematically equivalent methods:

$$f_1: 99 - 70 * \sqrt{2} \qquad f_1(1.4) = 1$$

$$f_2: 1 / (99 + 70 * \sqrt{2}) \qquad f_2(1.4) \approx 0.0051$$

Condition numbers:

$$f_1(x) = 99 - 70x \qquad \gamma_1(\sqrt{2}) \approx 20000$$

$$f_2(x) = 1 / (99 + 70x) \qquad \gamma_2(\sqrt{2}) \approx 0.5$$

$$f_1: 99 - 70 * \sqrt{2}$$

$$f_2: 1 / (99 + 70 * \sqrt{2})$$

- Condition number of subtraction, addition:

$$f(x) = x - a \quad \gamma_- = |-x / (x - a)| \quad \text{ill-conditioned for } x - a \approx 0$$

$$f(x) = x + a \quad \gamma_+ = |x / (x + a)| \quad \text{ill-conditioned for } x + a \approx 0$$

- Condition number for multiplication, division:

$$f(x) = ax \quad \gamma = |xa / (ax)| = 1$$

$$f(x) = 1/x \quad \gamma = |xx^{-2} / (x^{-1})| = 1 \quad \text{well-conditioned}$$

NUMERICAL DISASTERS

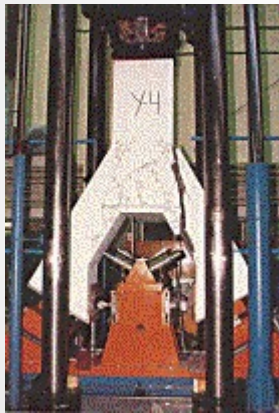
- Patriot system hit by SCUD missile
 - position predicted from time and velocity
 - the system up-time in $1/10$ of a second was converted to seconds using 24bit precision (by multiplying with $1/10$)
 - $1/10$ has non-terminating binary expansion
 - after 100h, the error accumulated to 0.34s
 - the SCUD travels 1600 m/s so it travels $>500\text{m}$ in this time (28 soldiers DEAD!)
- Ariane 5
 - A 64bit FP number containing the horizontal velocity was converted to 16bit signed integer
 - Range overflow followed (Rocket LOST!)



from
<http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html>

The sinking of the Sleipner A offshore platform

The Sleipner A platform produces oil and gas in the North Sea and is supported on the seabed at a water depth of 82 m. It is a Condeep type platform with a concrete gravity base structure consisting of 24 cells and with a total base area of 16 000 m². Four cells are elongated to shafts supporting the platform deck. The first concrete base structure for Sleipner A sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway on 23 August 1991. Immediately after the accident, the owner of the platform, Statoil, a Norwegian oil company appointed an investigation group, and SINTEF was contracted to be the technical advisor for this group. The investigation into the accident is described in 16 reports... The conclusion of the investigation was that the loss was caused by a failure in a cell wall, resulting in a serious crack and a leakage that the pumps were not able to cope with. The wall failed as a result of a combination of a serious error in the finite element analysis and insufficient anchorage of the reinforcement in a critical zone. A better idea of what was involved can be obtained from this photo and sketch of the platform. The top deck weighs 57,000 tons, and provides accommodation for about 200 people and support for drilling equipment weighing about 40,000 tons. When the first model sank in August 1991, the crash caused a seismic event registering 3.0 on the Richter scale, and left nothing but a pile of debris at 220m of depth. The failure involved a **total economic loss of about \$700 million.**








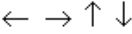
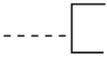
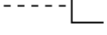



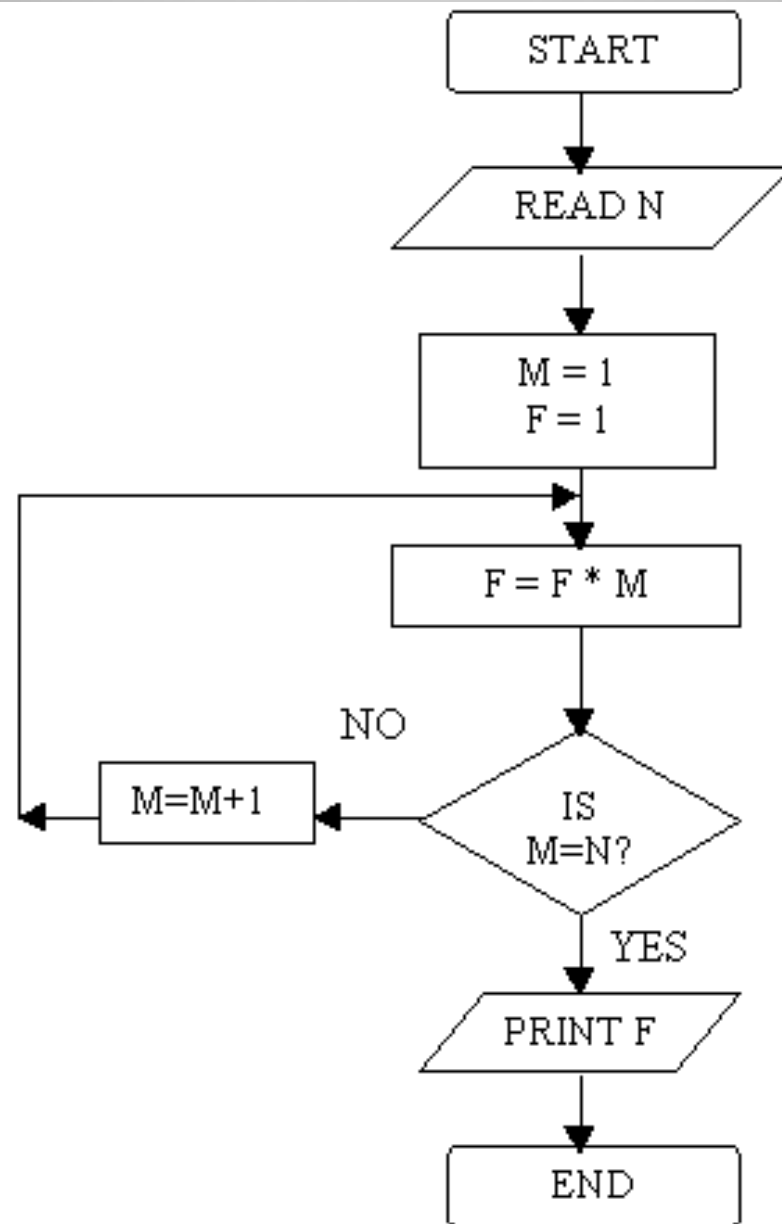
The 24 cells and 4 shafts referred to above are shown to the left while at the sea surface.

The cells are 12m in diameter. The cell wall failure was traced to a tricell, a triangular concrete frame placed where the cells meet. At right one is pictured undergoing failure testing. The post accident investigation traced the error to inaccurate finite element approximation of the linear elastic model of the tricell (**using the popular finite element program NASTRAN**). The shear stresses were underestimated by 47%, leading to insufficient design. In particular, certain concrete walls were not thick enough. More careful finite element analysis, made after the accident, predicted that failure would occur with this design at a depth of 62m, which matches well with the actual occurrence at 65m



FLOWCHARTING SYMBOLS

	Start or end of the program
	Computational steps or processing function of a program
	Input or output operation
	Decision making and branching
	Connector or joining of two parts of program
	Magnetic Tape
	Magnetic Disk
	Off-page connector
	Flow line
	Annotation
	Display



CODE COMPILATION SEQUENCE

List of Languages Supported by GCC Compiler

- C (gcc command)
- C++ (g++ command)
- Objective-C / Objective-C++ (gcc command)
- Fortran (g77 command)
- Java (gcj command)
- Ada (gnat command)
- Assembly language of every supported processor (as command)
- Other languages are supported with “front-end” interfaces that are not built-in to GCC. User must obtain these front-ends from other sources.
 - Pascal
 - Mercury
 - COBOL

Compiler Front-End (gcc / g++ Commands)

- Wrapper that coordinates compile sequence
- Provides single (language dependent) command to interface to all the compile steps
- Reduces complexity to one command
- Chooses correct compile steps based on input file extension
- Checks file extensions for non-default output filenames
- Layered user interface approach allows simple usage as well as more powerful complex usage
- Example below compiles source file into an executable and warns of any major issues.

```
gcc -Wall hello.c -o hello.exe
```


VIRTUES OF MODULAR CODING

```
/*  
    This is a demonstration program for the  
    rootfinding subroutine 'bisect'  
*/
```

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

```
float fcn(float);  
float sign(float, float);  
void bisect(float(*f)(float), float, float, float, float  
*, int *);
```

```

main()
{

    float a, b, epsilon, root;
    int ier;

    while (1) {

        /* Input problem parameters */

        printf("\n\n What are a,b,epsilon");
        printf("\n to stop, let epsilon=0 : \n");
        scanf("%f %f %f", &a, &b, &epsilon);
        if (epsilon == 0.0)
            return 0;

        /* Calculate root */

        bisect(fcn, a ,b ,epsilon, &root, &ier);

        /* Print answers */

        printf("\n\n a = %11.4e    b = %11.4e    epsilon =
%9.3e",
            a, b, epsilon);
        printf("\n root = %14.7e    ier = %1d", root, ier);
    }
    system ("pause");
    return 0;
}

```

```
float fcn(float x)
{
    float result;

    result = x - exp(-x);
    return(result);
}
```

```
float sign(float a, float b)
{
    if (b < 0.0)
        return(-fabs(a));
    else
        return(fabs(a));
}
```

```
void bisect(float(*f)(float), float a, float b, float eps,  
           float *root, int *ier)
```

```
{  
/*
```

The program uses the bisection method to solve
the equation

$$f(x) = 0.$$

The solution is to be in $[a,b]$ and it is assumed
that

$$f(a)*f(b) \leq 0.$$

The solution is returned in root, and it is to
be in error by at most eps.

ier is an error indicator.

If ier=0 on completion of the routine, then the
solution has been computed satisfactorily.

If ier=1, then $f(a)*f(b)$ was greater than 0,
contrary to assumption.

```
*/
```

```
const float zero = 0.0, one = 1.0, two = 2.0;  
float c, fa, fb, fc, sfa, sfb, sfc;
```

```
/* Initialize */
```

```
fa = (*f)(a);  
fb = (*f)(b);  
sfa = sign(one, fa);  
sfb = sign(one, fb);  
if (sfa*sfb > 0.0)  
{
```

```
/* The choice of a and b is in error */
```

```
*ier = 1;  
return;  
}
```

```
/* Create a new value of c, the midpoint of [a,b] */
```

```
while (1) {  
    c = (a + b)/two;  
    if (fabs(b-c) <= eps)  
    {
```

```
        /* c is an acceptable solution of f(x)=0 */
```

```
        *root = c;  
        *ier = 0;  
        return;  
    }
```

```
/* The value of c was not sufficiently accurate.  
   Begin a new iteration */
```

```
    fc = (*f)(c);  
    if (fc == zero)  
    {
```

```
        /* c is an acceptable solution of f(x)=0 */
```

```
        *root = c;  
        *ier = 0;  
        return;  
    }
```

```
    sfc = sign(one, fc);  
    if (sfb*sfc > zero)  
    {
```

```
        /* The solution is in [a,c] */
```

```
        b = c;  
        sfb = sfc;  
    }
```

```
    else  
    {
```

```
        /* The solution is in [c,b] */
```

```
        a = c;  
        sfa = sfc;  
    }
```

```
    }  
}
```

