

From Mathematical Formula to Scientific Software

Outline of the lecture

- Introduction
- Mathematical formula \Rightarrow complete algorithm \Rightarrow working code
- Two types of programming languages
- Some software issues
- Project assignment

Scientific computing

- Motivations
 - Computer simulation of physical processes
 - Physical process \rightarrow mathematical model \rightarrow software program \rightarrow simulation result
- Application of numerical algorithms
(discrete approximations of analytical solutions)
- Widely used
 - Simulation of natural phenomena
 - Applications in industry
 - Applications in medicine
 - Applications in finance

Scientific software

- Desired properties
 - Correct
 - Efficient (speed, memory, storage)
 - Easily maintainable
 - Easily extendible
- Important skills
 - Understanding numerics
 - Designing data structures
 - Using libraries and programming tools
 - (Quick learning of new programming languages)

A typical scientific computing code

- Starting point
 - Numerical problem
- Pre-processing
 - Data input and preparation
 - Build-up of internal data structure
- Main computation
- Post-processing
 - Result analysis
 - Display, output and visualization

Computational kernels

- Integration
- Differentiation
- Interpolation
- Discretization
- Systems of linear equations
- Systems of nonlinear equations
- Ordinary differential equations
- Partial differential equations

Building blocks

- Variables
- Arrays
- Branches
- Loops
- User-defined functions/subroutines/methods
- New advanced data types (classes/structs/modules)

Example: Euclidean norm of a vector

- Given a vector $\mathbf{v} = (v_1, v_2, \dots, v_n) \in \mathbb{R}^n$

- Euclidean norm: $\|\mathbf{v}\| \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n v_i^2}$

- \mathbf{v} as a one-dimensional array (syntax in Java)

```
double[] v;
```

```
v = new double[n];
```

- Computing the Euclidean norm by a for-loop:

```
double norm_v = 0.;
int i;
for (i=0; i<n; i++) {
    norm_v += v[i]*v[i];
}
norm_v = Math.sqrt(norm_v);
```

Example: matrix-vector product

- Given matrix $\mathbf{A} \in \mathbb{R}^{m,n}$, two vectors: $\mathbf{w} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$
- Matrix-vector product $\mathbf{w} = \mathbf{A} \mathbf{v}$ where

$$w_i \stackrel{\text{def}}{=} \sum_{j=1}^n A_{i,j} v_j \quad \text{for } i = 1, 2, \dots, m$$

- \mathbf{A} as a two-dimensional array
- \mathbf{v} and \mathbf{w} as one-dimensional arrays
- Main computation by a two-level for-loop:

```
for (i=0; i<m; i++) {  
    w[i] = 0.;  
    for (j=0; j<n; j++)  
        w[i] += A[i][j]*v[j];  
}
```

A two-step strategy

- Correct implementation of a complicated numerical problem is a challenging task
- Divide the task into two steps:
 - Express the numerical problem as a **complete algorithm**
 - Translate the algorithm into a computer code using a specific programming language

Advantages

- Small gap between the numerical method and the complete algorithm (few software issues to consider)
- Easy translation from the complete algorithm to a computer code (no numerical issues)
- An effective approach
- Easy to debug
- Easy to switch to another programming language

Writing complete algorithms

- Complete algorithm = mathematical pseudo code:
programming language independent!
- Rewrite a compact mathematical formula as a set of simple operations (e.g., replace \sum with a for-loop or do-loop in Fortran)
- Identify input and output
- Give names to mathematical entities and make them variables/arrays
- Introduce intermediate variables (if necessary)

Example: Trapezoidal integration

- Want to approximate $\int_a^b f(x)dx$
- Use n Trapezoids

$$\int_a^b f(x)dx \approx \frac{h}{2}f(a) + \frac{h}{2}f(b) + h \sum_{i=1}^{n-1} f(a + ih)$$

- $h = \frac{b-a}{n}$

A complete algorithm

```
trapezoidal ( $a, b, f, n$ )  
   $h = \frac{b-a}{n}$   
   $s = 0$   
  for  $i = 1, \dots, n - 1$   
     $s \leftarrow s + h f(a + ih)$   
  end for  
   $s \leftarrow s + \frac{h}{2} f(a) + \frac{h}{2} f(b)$   
  return  $s$ 
```

- Input: a, b, f, n
- Output: s
- \sum is expressed by a for-loop
- The symbol “ \leftarrow ” means an update

Comments

- Strictly speaking, the two-step strategy is not necessary for this very simple example
- The purpose is for demonstrating the ideas
- For more complicated numerical problems, the mathematical pseudo code is useful for developing and checking a software code

Efficiency improvement

```
trapezoidal ( $a, b, f, n$ )
```

$$h = \frac{b-a}{n}$$

$$s = 0 \quad x = a$$

```
for  $i = 1, \dots, n - 1$ 
```

$$x \leftarrow x + h$$

$$s \leftarrow s + f(x)$$

```
end for
```

$$s \leftarrow s + 0.5 \cdot (f(a) + f(b))$$

$$s \leftarrow hs$$

```
return  $s$ 
```

- Reduction of the number of arithmetic operations
 - Factorization of the factor h
 - Introduction of intermediate variable x
- Multiplication ($0.5 \cdot f(a)$) instead of division ($f(a)/2$)

Costly operations

- Evaluation of complicated mathematical functions
(e.g. $f(x) = e^{-x^2}$)
- Divisions
- If-test inside loops
- Read from memory and write to memory

Optimization; rule of thumb

- Adopt good programming habits
- Maintain the clear structure of the numerical method
- Avoid “premature optimization”
- Leave part of the optimization work to a compiler

Example: Simpson's rule

- Want to approximate $\int_a^b f(x)dx$
- Similar idea as Trapezoidal rule, better accuracy

$$\int_a^b f(x)dx \approx \frac{h}{6} \sum_{i=1}^n \left\{ f(x_{i-1}) + 4f(x_{i-\frac{1}{2}}) + f(x_i) \right\}$$

- $h = \frac{b-a}{n}$, $x_i = a + ih$, $x_{i-\frac{1}{2}} = \frac{1}{2}(x_{i-1} + x_i)$

Complete algorithm (I)

```
simpson ( $a, b, f, n$ )
```

$$h = \frac{b-a}{n}$$

$$s = 0$$

```
for  $i = 1, \dots, n$ 
```

$$x^- = a + (i-1)h$$

$$x^+ = a + ih$$

$$x = \frac{1}{2}(x^- + x^+)$$

$$s \leftarrow s + f(x^-) + 4f(x) + f(x^+)$$

```
end for
```

$$s \leftarrow \frac{h}{6}s$$

```
return  $s$ 
```

- Input: a, b, f, n
- Output: s
- Intermediate variables: x^-, x, x^+

Efficiency consideration

- $f(x^+)$ in iteration i is the same as $f(x^-)$ in iteration $i + 1$

$$\begin{aligned} & f(x_0) + 4f(x_{\frac{1}{2}}) + f(x_1) + \\ & f(x_1) + 4f(x_{1+\frac{1}{2}}) + f(x_2) + \\ & \dots \\ & f(x_{n-1}) + 4f(x_{n-\frac{1}{2}}) + f(x_n) \end{aligned}$$

- Unnecessary function evaluations should be avoided for efficiency!
- Rewrite Simpson's rule

$$\int_a^b f(x)dx \approx \frac{h}{6} \left[f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_i) + 4 \sum_{i=1}^n f(x_{i-\frac{1}{2}}) \right]$$

Complete algorithm (II)

```
simpson (a, b, f, n)
  h =  $\frac{b-a}{n}$ 
  s1 = 0 x = a
  for i = 1, ..., n - 1
    x ← x + h
    s1 ← s1 + f(x)
  end for
  s2 = 0 x = a + 0.5 · h
  for i = 1, ..., n
    s2 ← s2 + f(x)
    x ← x + h
  end for
  s =  $\frac{h}{6}(f(a) + f(b) + 2s_1 + 4s_2)$ 
  return s
```

- New intermediate variables s_1 and s_2
- Two for-loops (can we combine them into one loop?)

Choosing a programming language

- Many programming languages exist
- We examine 7 languages: Fortran 77, C, C++, Java, Maple, Matlab & Python
- Issues that influence the choice of a programming language
 - Static typing vs. dynamic typing
 - Computational efficiency
 - Built-in high-performance utilities
 - Support for user-defined data types

Static typing vs. dynamic typing

- Statically typed programming languages
 - Each variable must be given a specific type (int, char, float, double etc.)
 - Compiler is able to detect obvious syntax errors
 - Special rules for transformation between different types
- Dynamically typed programming language
 - No need to give a specific type to a variable
 - Typing is dynamic and adjusts to the context
 - Great flexibility and more “elegant” syntax
 - Difficult to detect certain “typos”

Computational efficiency

- Compiled languages run normally fast
 - Program code $\xrightarrow{\text{compilation \& linking}}$ executable (machine code)
- Interpreted languages run normally slow
 - Statements are interpreted directly as function calls in a library
 - Translation takes place “on the fly”
- Different compiled languages may have different efficiency

Built-in utilities

- Compiled languages have very fast loop-instructions
- Plain loops in interpreted languages (Maple, Matlab & Python) are very slow
- Important for interpreted languages to have built-in numerical libraries
- Need to “break” a complicated numerical method into a series of simple steps when using an interpreted language

User-defined data types

- Built-in primitive data types may not be enough for complicated numerical programming
- Need to “group” primitive variables into a new data type
 - `struct` in C (only data, no function)
 - `class` in C++, Java & Python
 - Class hierarchies \Rightarrow powerful tool \Rightarrow **object-oriented programming**

Different programming languages

- Different syntax
- Similar structure for main computation
- Different ways for function transfer
- Different I/O
- Different ways for writing comments
- No need to learn all the details at once!
- Learn from the examples!

Example implementations

- Trapezoidal rule for

$$f(x) = e^{-x^2} \log(1 + x \sin(x)), \quad a = 0, \quad b = 2, \quad n = 1000.$$

- Using six programming languages
 - Treating C as a subset of C++

Trapezoidal rule in Fortran 77 (I)

```
real*8 function trapezoidal (a, b, f, n)
real*8 a, b, f
external f
integer n

real*8 s, h, x
integer i
h = (b-a)/float(n)
s = 0
x = a
do i = 1, n-1
    x = x + h
    s = s + f(x)
end do
s = 0.5*(f(a) + f(b)) + s
trapezoidal = h*s
return
end
```

Trapezoidal rule in Fortran 77 (II)

C test function to integrate:

```
real*8 function f1 (x)
real*8 x
f1 = exp(-x*x)*log(1+x*sin(x))
return
end
```

C main program:

```
program integration
integer n
real*8 a, b, result
external f1
a = 0
b = 2
n = 1000
result = trapezoidal (a, b, f1, n)
write (*,*) result
end
```

Trapezoidal rule in C++ (I)

File: Trapezoidal.h

```
typedef double (*fptr) (double x);

double Trapezoidal (double a, double b, fptr f, int n)
{
    double h = (b-a)/double(n);
    double s = 0, x = a;
    for (int i=1; i<=n-1; i++) {
        x = x + h;
        s = s + f(x);
    }
    s = 0.5*(f(a)+f(b)) + s;
    return h*s;
}
```

Trapezoidal rule in C++ (II)

```
#include "Trapezoidal.h"
#include <cmath>
#include <iostream>

double f1 (double x)
{
    return exp(-x*x)*log(1.0+x*sin(x));
}

int main()
{
    double a = 0, b = 2;
    int n = 1000;
    double result = Trapezoidal (a, b, f1, n);
    std::cout << result << std::endl;
}
```

Trapezoidal rule in Java (I)

```
interface Func { // base class for function f(x)
    public double f (double x); // default empty implementation
}

class Trapezoidal {
    public static double integrate (double a, double b,
                                    Func f, int n)
    {
        double h = (b-a)/((double) n);
        double s = 0, x = a;
        int i;
        for (i=1; i<=n-1; i++) {
            x = x + h;
            s = s + f.f(x);
        }
        s = 0.5*(f.f(a)+f.f(b)) + s;
        return h*s;
    }
}
```

Trapezoidal rule in Java (II)

```
class f1 implements Func {
    public double f (double x)
    { return Math.exp(-x*x)*Math.log(1.0+x*Math.sin(x)); }
}

class Demo {
    public static void main (String argv[])
    {
        double a = 0, b = 2;
        int n = 1000;
        double result;
        f1 my_func = new f1();
        result = Trapezoidal.integrate(a, b, my_func, n);
        System.out.println(result);
    }
}
```

Trapezoidal rule in Matlab (I)

File: Trapezoidal.m

```
function r = Trapezoidal(a, b, f, n)
%TRAPEZOIDAL Numerical integration from a to b using trapezoids

f = fcnchk(f);
h = (b-a)/n;
s = 0;
x = a;

for i = 1:n-1
    x = x + h;
    s = s + feval(f,x);
end
s = 0.5*(feval(f,a) + feval(f,b)) + s;
r = h*s;
```

Trapezoidal rule in Matlab (II)

File: f1.m

```
function y = f1(x)
y = exp(-x*x)*log(1+x*sin(x));
```

File: main.m

```
a = 0;
b = 2;
n = 1000;
result = Trapezoidal(a, b, @f1, n);
disp(result);
```

```
unix> matlab
matlab> main
```

Trapezoidal rule in Python

```
#!/usr/bin/env python
from math import *

def Trapezoidal(a, b, f, n):
    h = (b-a)/float(n);
    s = 0
    x = a
    for i in range(1,n,1):
        x = x + h
        s = s + f(x)
    s = 0.5*(f(a) + f(b)) + s
    return h*s

def f1(x):
    f = exp(-x*x)*log(1+x*sin(x))
    return f

a = 0; b = 2; n = 1000;
result = Trapezoidal(a, b, f1, n)
print result
```

Trapezoidal rule in Maple

File: Trapezoidal.mpl

```
Trapezoidal := proc(a,b,f,n)
local h, s, x, i:
h := (b-a)/n:
s := 0: x := a:
for i from 1 to n-1 do
    x := x + h:
    s := s + f(x):
od:
s := s + 0.5*(f(a)+f(b)):
s := h*s:
s;
end:
```

```
unix> maple
```

```
> read "Trapezoidal.mpl";
> f1:=x->exp(-x*x)*log(1+x*sin(x));
                                2
f1 := x -> exp(-x ) log(1 + x sin(x))

> q=Trapezoidal(0,2.0,f1,1000);
```

Measuring CPU-time

- On Unix/Linux, command `time` can be used
 - Easy to use
 - Low timing resolution
- There are programming language dependent timing functions
 - High timing resolution
 - No standard name or syntax

Vectorization

- Loops are very slow in interpreted languages
- Should use built-in vector functionality when possible

```
trapezoidal_vec (a,b,f,n)
```

$$h = \frac{b-a}{n}$$

$$\mathbf{x} = (a, a+h, \dots, b)$$

$$\mathbf{v} = f(\mathbf{x})$$

$$s = h \cdot (\text{sum}(\mathbf{v}) - 0.5 \cdot (v_1 + v_{n+1}))$$

```
return s
```

Guidelines on implementation

- Understand the numerics (make use of literature)
- Close resemblance between mathematical pseudo code and numerical method
- Test the implementation on first problems with known solutions
- No premature optimization before code verification
- During later optimization, refer to the “non-optimized” code as reference for checking

Project assignment No. 1

- Motivation
 - Getting some initial experience with several new programming languages
 - Learning how to write a report (using \LaTeX)
- Submission
 - Every student submits her/his own report (own implementation)
- Important links:
`http://www.ifi.uio.no/~xingca/inf165`

Project assignment No. 1 (cont'd)

- Implementation of Simpson's rule

$$\int_a^b f(x)dx \approx \frac{h}{6} \sum_{i=1}^n \left\{ f(x_{i-1}) + 4f(x_{i-\frac{1}{2}}) + f(x_i) \right\},$$

- Efficiency comparison
- Convergence study
- Non-uniform spacing