CS601: Software Development for Scientific Computing

Autumn 2024

Week1: Overview

Course Takeaways (intended)

- Non-CS majors:
  1. Write code (mostly in C/C++) and
  2. Develop software (not just write standalone code)
     - Numerical software
- CS-Majors:

  In addition to the above two:

  3. Learn to face mathematical equations and implement them with confidence

Why C++ ?

- C/C++/Fortran codes form the majority in scientific computing codes
- Catch a lot of errors early (e.g. at *compile-time* rather than at *run-time)*
- Has features for *object-oriented* software development
- Known to result in codes with better *performance*

What is this course about?

Software Development

+

Scientific Computing

Software Development

- *Software development is the process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in creating and maintaining applications, frameworks, or other software components.*

Software development is a process of writing and maintaining the source code, but in a broader sense, it includes all that is involved between the conception of the desired software through to the final manifestation of the software, …

- Wikipedia on “Software Development”

Scientific Computing

- Also called computational science
  - *Development of models to understand systems (biological, physical, chemical, engineering, humanities)*

Collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in science and engineering

This course NOT about..

- Software Engineering
  - Systematic study of Techniques, Methodology, and Tools to build correct software within time and price budget (topics covered in CS305)
    - People, Software life cycle and management etc.
- Scientific Computing
  - Rigorous exploration of numerical methods, their analysis, and theories
  - Programming models (topics covered in CS410)

Who this course is for?

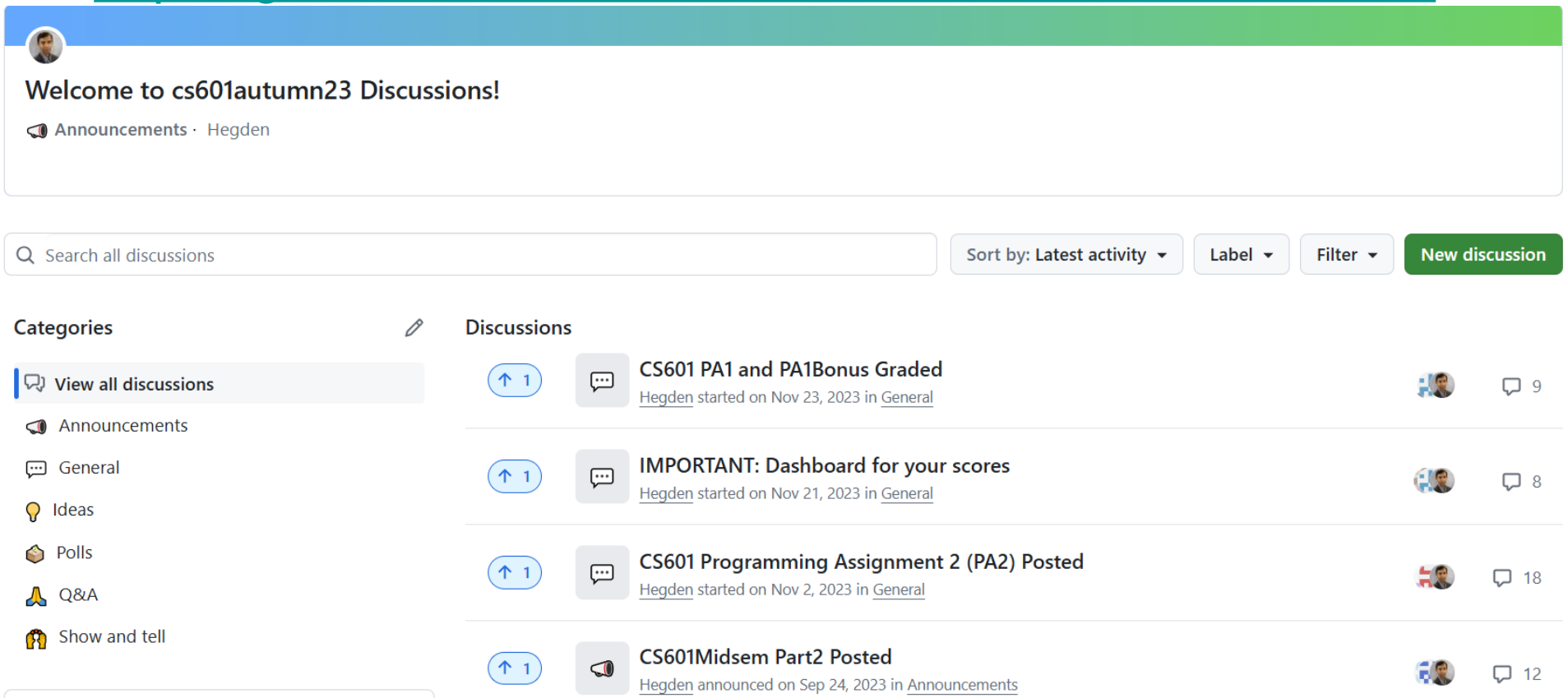
- You are interested in scientific computing
- You are interested in high-performance computing
- You want to build / add to a large software system

Course Webpage

- https://hegden.github.io/cs601

GitHub Discussions

- We will use the Discussion feature in GitHub. E.g. https://github.com/IITDhCSE/cs601autumn23/discussions



Provide your github IDs here: https://docs.google.com/forms/d/e/1FAIpQLSdWIaAXddcs0yED3-p6230-DotoaxhNrPEtnDiCbGPsfF7clw/viewform?usp=sf_link

Let us try an exam question (from this course) on ChatGPT

- Question:

Computing $\sqrt{x^2 + y^2}$ is a common problem. A common implementation strategy is as follows:

```
double ComputeHypotenuse(double x, double y) {
    return sqrt(x*x + y*y);
}
```

However, the above strategy is not robust. How would you implement a robust code?

- https://chat.openai.com/share/bcf4d871-21cf-4799-9275-1486345ce6dd

Takeaways:

- You still need to know the right questions to ask.
- Know if the answer provided makes sense.

Develop intuition

Let us dive into an example….

Example - Factorial

- $n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

  $(n-1)! = (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

  therefore,

  Definition1: $n! = n \times (n-1)!$

  is this definition complete?

  - plug 0 to n and the equation breaks.

  Definition2:

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

Exercise 1

- Does this code implement the definition of factorial correctly?

```
int fact(int n){
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Example - Factorial

Definition2:

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

is this definition complete?

- `n!` is not defined for negative `n`

Solution - Factorial

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Exercise 2

- In how many `flops` does the code execute?

  assume 1 `flop` = 1 step executing ***any*** arithmetic operation

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Exercise 3

- Does the code yield correct results for any n?

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Computational Thinking

- We just did an exercise in "computational thinking"
  - A skill necessary for everyone, not just computer programmers
  - An approach to problem solving, designing systems, and understanding human behavior that draws on concepts fundamental to computer science.

Computational Thinking - Examples

- How difficult is the problem to solve? And what is the best way to solve?
- Modularizing something in anticipation of multiple users
- Prefetching and caching in anticipation of future use
- Thinking recursively
- Reformulating a seemingly difficult problem into one which we know how to solve by reduction, embedding, transformation, simulation
  - Are approximate solutions accepted?
  - False positives and False negatives allowed? etc.
- Using abstraction and decomposition in tackling large problem
- …

Computational Thinking – 2 As

- Abstractions
  - Our "mental" tools
  - Includes: choosing right abstractions, operating at multiple layers of abstractions, and defining relationships among layers
- Automation
  - Our "metal" tools that amplify the power of "mental" tools
  - Is mechanizing our abstractions, layers, and relationships
    - Need precise and exact notations / models for the "computer" below ("computer" can be human or machine)

Computing - 2 As Combined

- Computing is the automation of our abstractions
- Provides us the ability to scale
  - Make infeasible problems feasible
    - E.g. SHA-1 not safe anymore
  - Improve the answer's precision
    - E.g. capture the image of a black-hole

Summary: choose the right abstraction and computer

Computational Thinking – applied to the factorial exercise

- Need to be precise (formulating)
  - recall: `n! = 1` for `n=0`, not defined for negative `n`
- Choosing right abstractions
  - recall: use of recursion, correct data type
- Ability to define the complexity
  - recall: `flop` calculation
- What else?

Computational Thinking – applied to the factorial exercise

- Need to be precise (formulating)
  - recall: `n! = 1` for `n=0`, not defined for negative `n`
- Choosing right abstractions
  - recall: use of recursion, correct data type
- Ability to define the complexity
  - recall: `flop` calculation
- Choose the right “computer” for mechanizing the abstractions chosen

General Approach to Solving a Computational Problem

1. **Problem statement:** more precise this is, the easier it is to design and implement
2. **Solution Algorithm:** exactly how is the problem going to be solved
3. **Implementation:** breaking the algorithm into manageable pieces and putting it all together to solve the problem using a language of choice.
4. **Verification:** checking that the implementation solves the original problem.
   - For numerical software this is often most difficult step, because you don't know the correct answer.

Scientific Software - Characteristics

- The answer is not a typical yes/no, red/blue/green
- The answer varies continuously. Think of computing the value of pi = 3.141592…
- Uses approximations. Think of *discretization*
- Employs efficient *kernels*
  - Kernels are core operations that are executed very frequently
- Should be able to adapt to change.
  - Writing everything from scratch is not an option
- Deals with large-scale problems
  - Lot of input/output data or both
  - Computationally hard

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Exercise

Compute root(s) of:

$$x = \cos x;\ x \in \mathbb{R}$$

`roots`, also called `zeros`, is the value of the argument/input to the function when the function output vanishes i.e. becomes zero

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

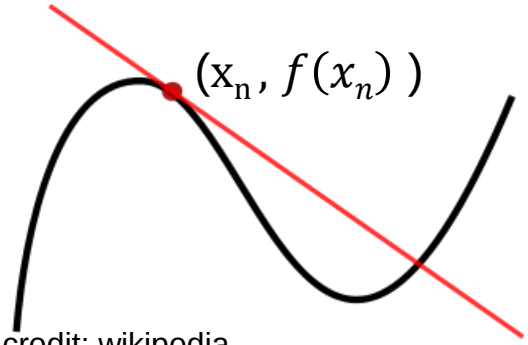
Mathematical Problem

- let $y = f(x)$

  $f(x) = \cos(x) - x$

- At $\mathrm{x} = \mathrm{x_n}$, the value of y is $f(x_n)$. The coordinates of the point are $(\mathrm{x_n}, f(x_n))$ = known point.

- From calculus: ***derivative*** of a function of single variable at a chosen input value, when it exists, is the ***slope of the tangent*** to the graph at that input value.
  - $f'(x_n)$ is the slope of the line that is tangent to $f(x)$ at $\mathrm{x_n}$



credit: wikipedia

Mathematical Problem

- From high-school math: point-slope formula for equation of a line

$$y - y_1 = m(x - x_1),$$

given the slope m and any known point (x_1, y_1)

- Substituting with:
  - $(x_n, f(x_n))$ = known point
  - $f'(x_n)$ = slope

Equation of the tangent line to graph of $f(x)$ at x_n :

$$y - f(x_n) = f'(x_n)(x - x_n)$$

Mathematical Problem

- Interested in finding roots i.e. value of x at y=0 i.e. at point $(x_{np1}, 0)$.
- Substituting in the equation of the tangent line,

$$y - f(x_n) = f'(x_n)(x - x_n)$$

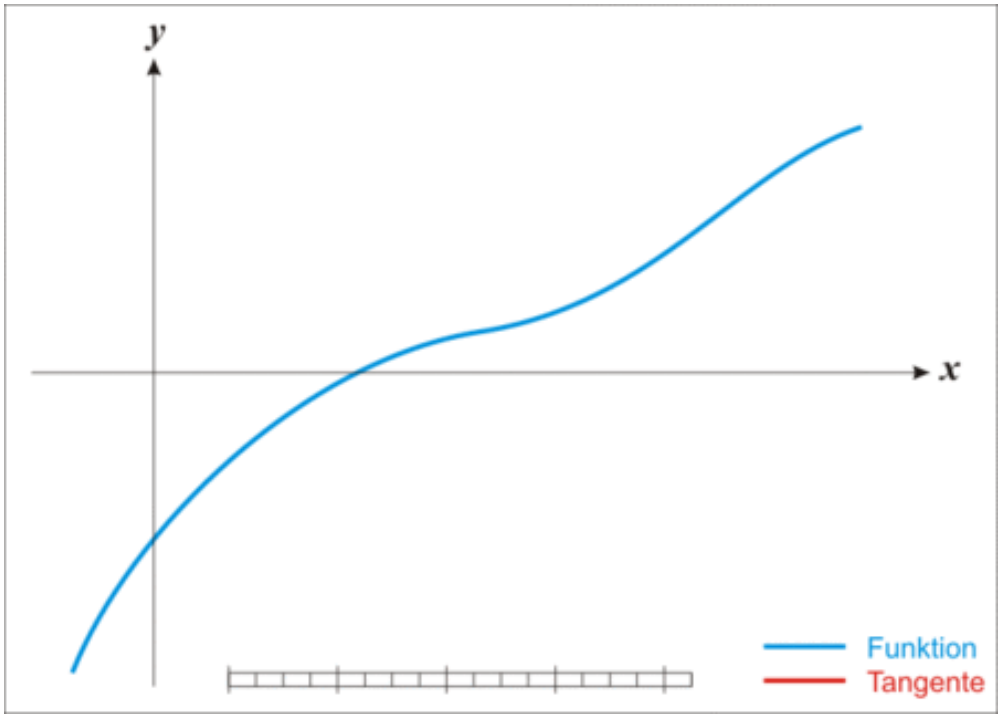
$$= \quad -f(x_n) = f'(x_n)(x_{np1} - x_n)$$

$$= \quad \boldsymbol{x_{np1} = x_n - f(x_n) \,/\, f'(x_n)}$$

Mathematical Problem

- Visualizing

  (source: https://en.wikipedia.org/wiki/Newton's_method) :



The function *f* is shown in blue and the tangent line is in red. We see that x_{n+1} is a better approximation than x_n for the root *x* of the function *f*.

Mathematical Problem

$$x_2 = x_1 - f(x_1) \,/\, f'(x_1)$$
$$x_3 = x_2 - f(x_2) \,/\, f'(x_2)$$
$$x_4 = x_3 - f(x_3) \,/\, f'(x_3)$$
$$\ldots$$

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Numerical Analysis

Talk to domain experts

- Choosing the initial value of x
- Does the method converge ?
- What is an acceptable approximation?
- etc.

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Designing Algorithms and Data Structures

- Start with x_1

$$\boldsymbol{x_2 = x_1 - f(x_1) \,/\, f'(x_1)}$$

$$\boldsymbol{x_3 = x_2 - f(x_2) \,/\, f'(x_2)}$$

$$\boldsymbol{x_4 = x_3 - f(x_3) \,/\, f'(x_3)}$$

…

- Repeat for up to `maxIterations`
- Check for $x_{n+1} - x_n$ to be "sufficiently small"
- Choose appropriate data types for x

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Selecting libraries and tools

- E.g. use the math library in C++ (cmath)

Toward Scientific Software

- Necessary Skills:
  1. Understanding the mathematical problem
  2. Understanding numerics
  3. Designing algorithms and data structures
  4. Selecting language and using libraries and tools
  5. Verify the correctness of the results
  6. Quick learning of new programming languages

Verify the correctness of results

- Compare with ‘gold’ code / benchmark
- Compare with empirical data

Scientific Software - Examples

Entertainment

- Toy Story, Shrek rendered using data-center nodes

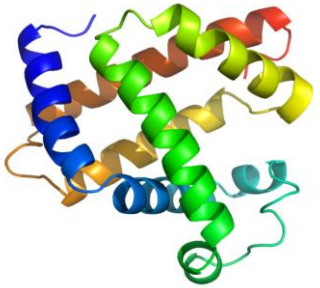
Economics

- ad-placement

Scientific Software - Examples

Biology

- Shotgun algorithm expedites sequencing of human genome



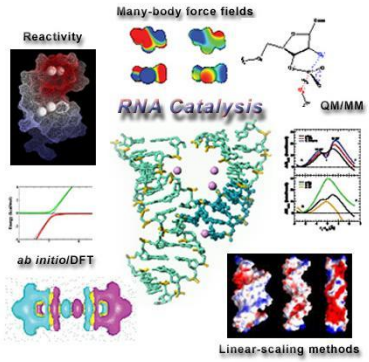
Credit: Wikipedia

- Analyzing fMRI data

Credit: Wikipedia

Chemistry

- optimization and search algorithms to identify best chemicals for improving reaction conditions to improve yields

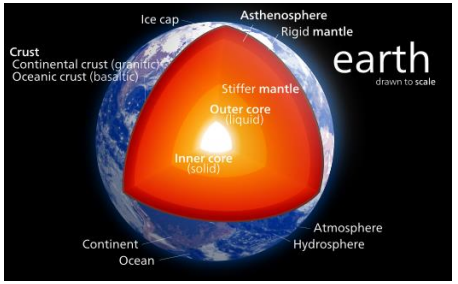


Credit: University of Minnesota

Scientific Software - Examples

Geology

- Modeling the Earth's surface to the core



Credit: Wikipedia

Astronomy

- kd-trees help analyze very large multi-dimensional data sets

Credit: Kaggle.com

Engineering

- Boeing 777 tested via computer simulation (not via wind tunnel)

Recap: Scientific Computing