Summer School for Integrated Computational Materials Education 2019

Introduction to Computation & Fundamentals of Simulations

Katsuyo Thornton
Materials Science & Engineering
University of Michigan

THIS LECTURE WILL COVER

1. Approach to Computational Modeling
2. Numerical Method (Example)
3. Errors & Uncertainties
4. Validation and Verification
5. Programming Practice

1. Introduction: Approach to Computational Modeling

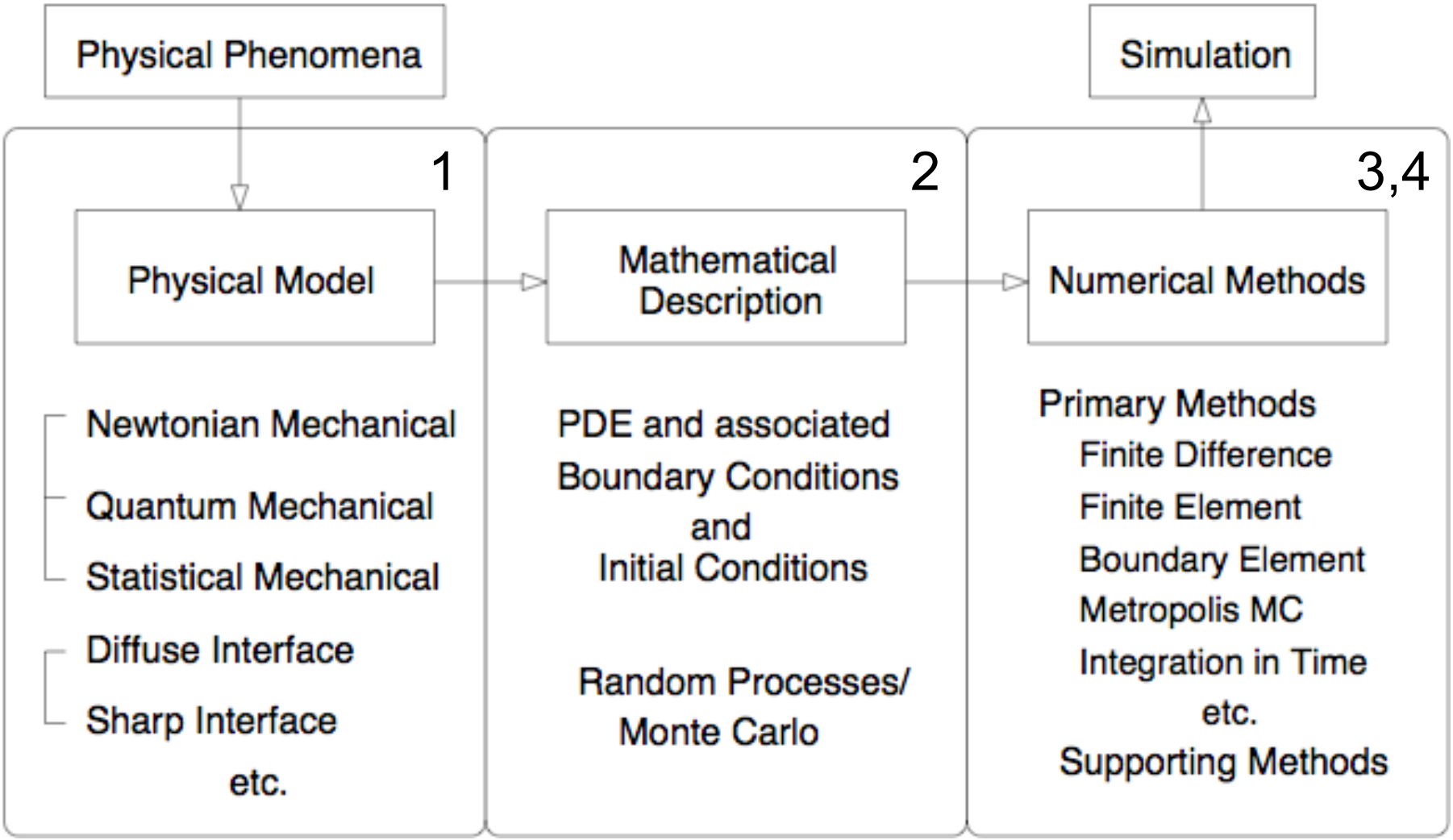
Why Computational Materials Science?

- Versatility; same method can be applied to many problems
- Increasing computational resources; many more problems can be solved
- Physical insights and understanding; applications beyond what is studied

How?

Develop a model of a materials system/process by identifying the underlying physics; solve the corresponding partial differential equations.

Approaches to Computational Modeling



Fundamentals of Simulation

INGREDIENTS OF NUMERICAL SIMULATIONS

A simulation of a complex physical system often requires a combination of powerful numerical methods. There are several layers in a complete simulation development process.

1a. Physical Model

- Quantum Models (solving Schrödinger equation)
- Atomistic Models (Monte Carlo, Molecular Dynamics, etc.)
- Continuum Models (e.g., Diffuse & Sharp Interface Models)

1b. Time Dependence

- Dynamics (explicit/implicit time evolution)
- Steady State (relaxation to an equilibrium or self-similar state)

INGREDIENTS OF NUMERICAL SIMULATIONS (CONTINUED)

2. Mathematical Description: e.g., PDE
   - Eulerian vs. Lagrangian (e.g., in MD)

3. Numerical Method: e.g., Discretization of the Physical Domain and/or Solving PDE
   - Finite Difference Method
   - Finite Element Method
   - Boundary Integral Method

4. Supporting Numerical Methods
   - Fast Fourier Transform
   - Multigrid Method

2. Numerical Methods

Example: Finite Difference Method

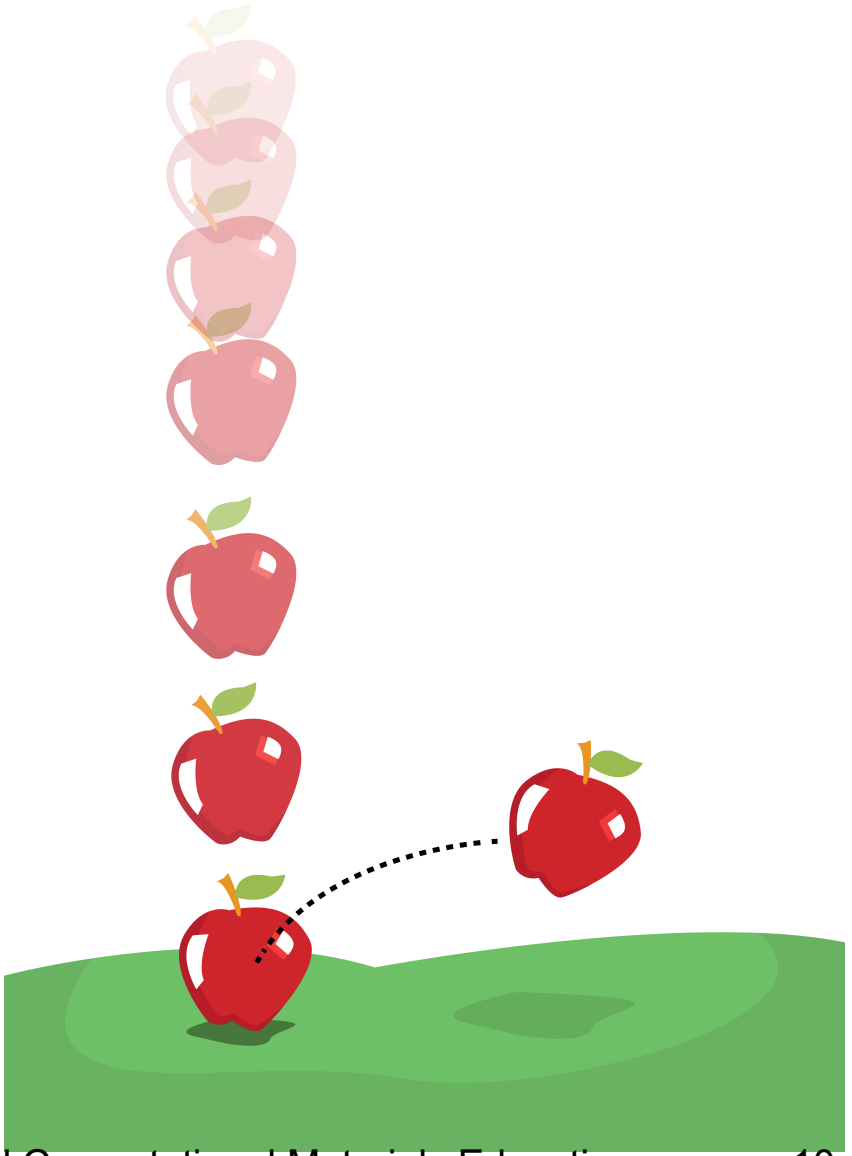
Finite Difference Method Allows You to Predict What Will Happen

- If you know the rate of change, you can predict what it will be after a certain amount of time

$$\text{Rate of change} = \frac{\text{Change in something}}{\text{Change in time}}$$

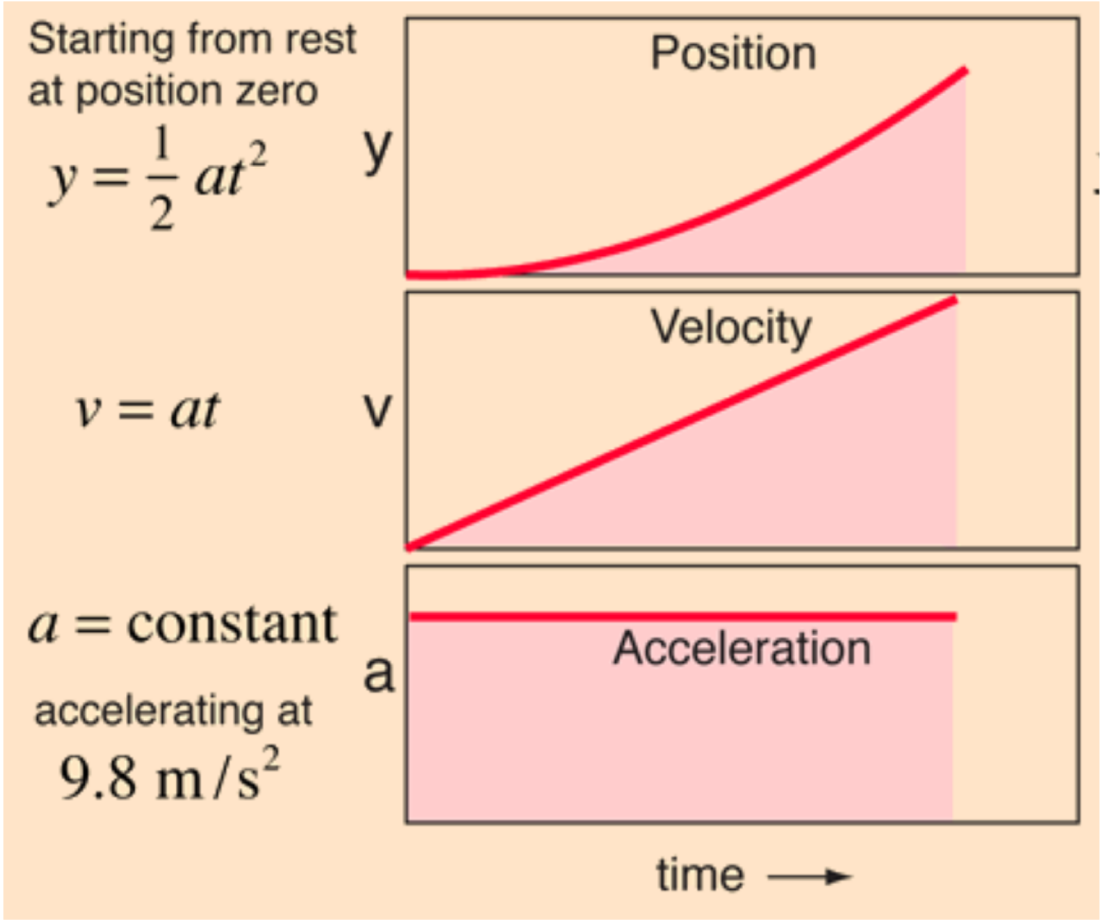
Predicting the future sometimes requires math

- If you know the rate of change, you can predict what it will be after a certain amount of time
- Example: an apple falling from a tree



Predicting the future sometimes requires math

- If you know the rate of change, you can predict what it will be after a certain amount of time
- Example: an apple falling from a tree

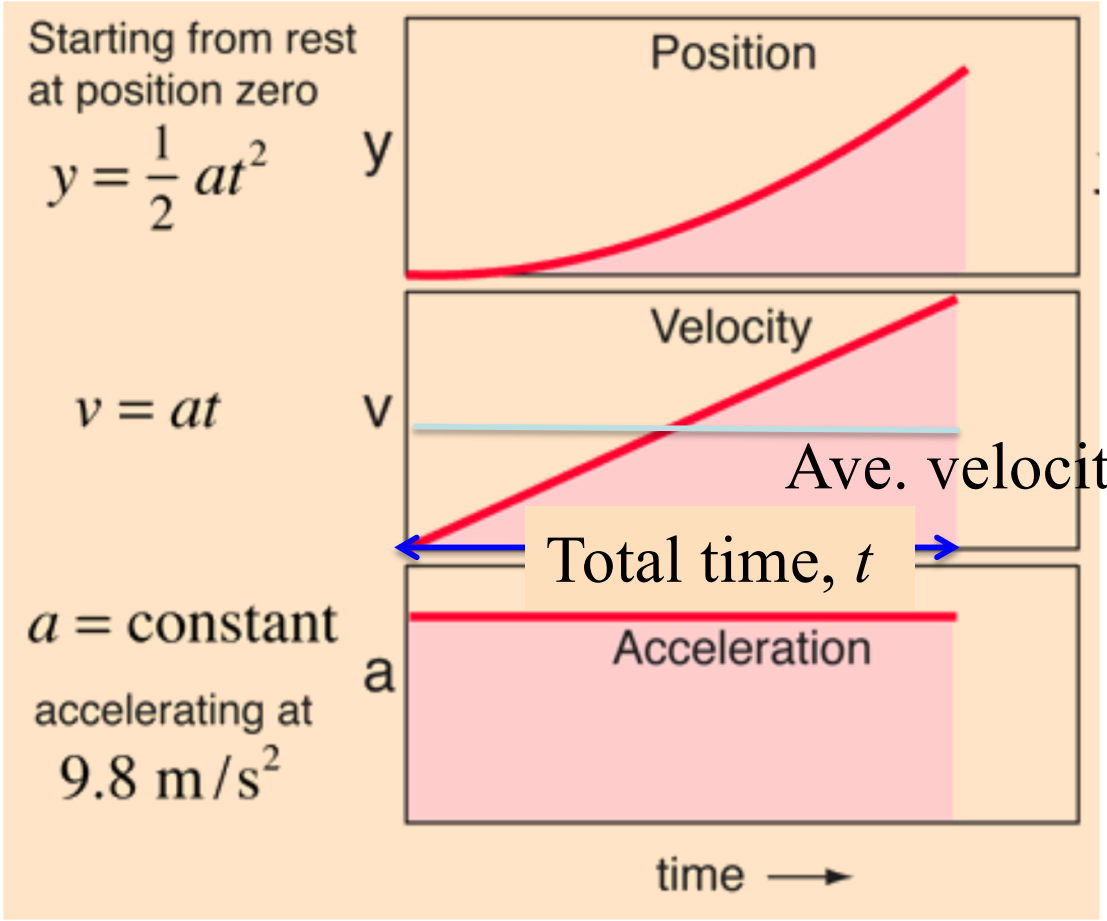


Predicting the future sometimes requires math

- If you know the rate of change, you can predict what it will be after a certain amount of time
- Example: an apple falling from a tree

$$v_{ave} = \frac{1}{2}at$$

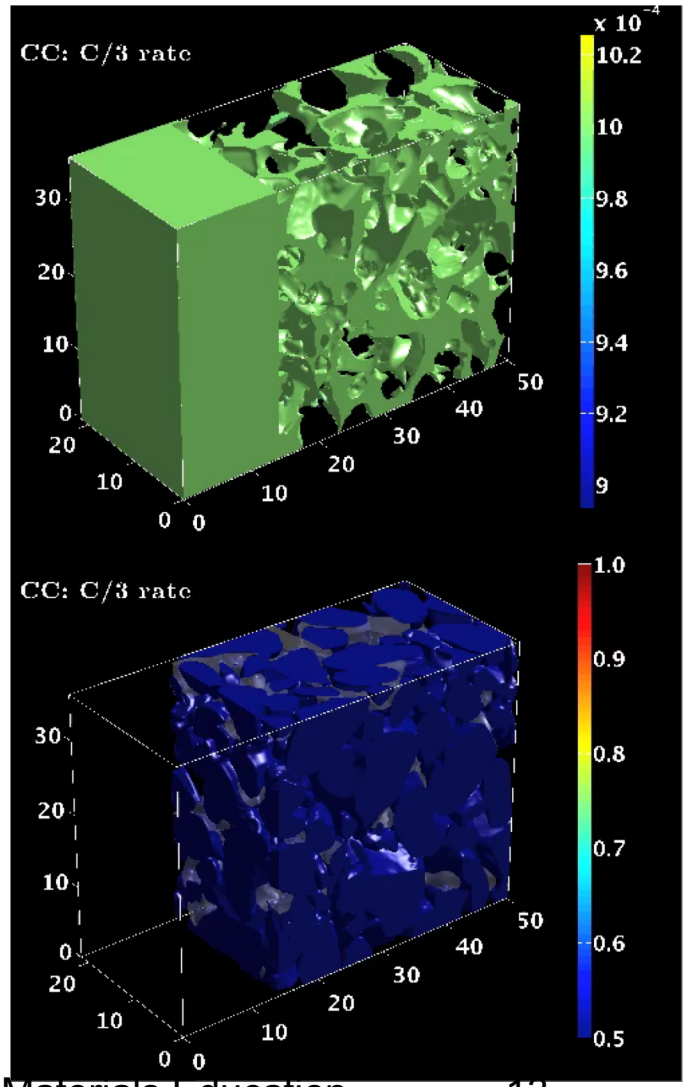
$$y = v_{ave}t = \frac{1}{2}at^2$$



But as Things Get More Complicated, You Need a Computer

- Or a computational cluster with thousands of CPUs
- Models often results in PDEs
- Discretization is required to solve it on a computer

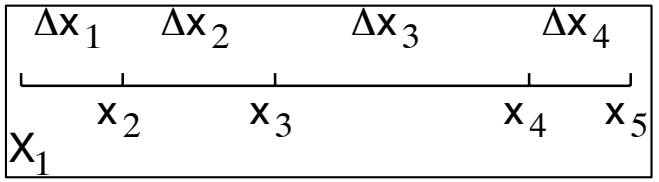
The Flux cluster at the UM Modular Data Center



DISCRETE REPRESENTATION OF A CONTINUOUS VARIABLE

Consider an independent continuous variable $x \in [X_1, X_2]$. We take the discrete representation, x_i, of x to be

$$x_i = X_1 + \sum_{\alpha=1}^{i-1} \Delta x_\alpha, 1 < i \leq N$$



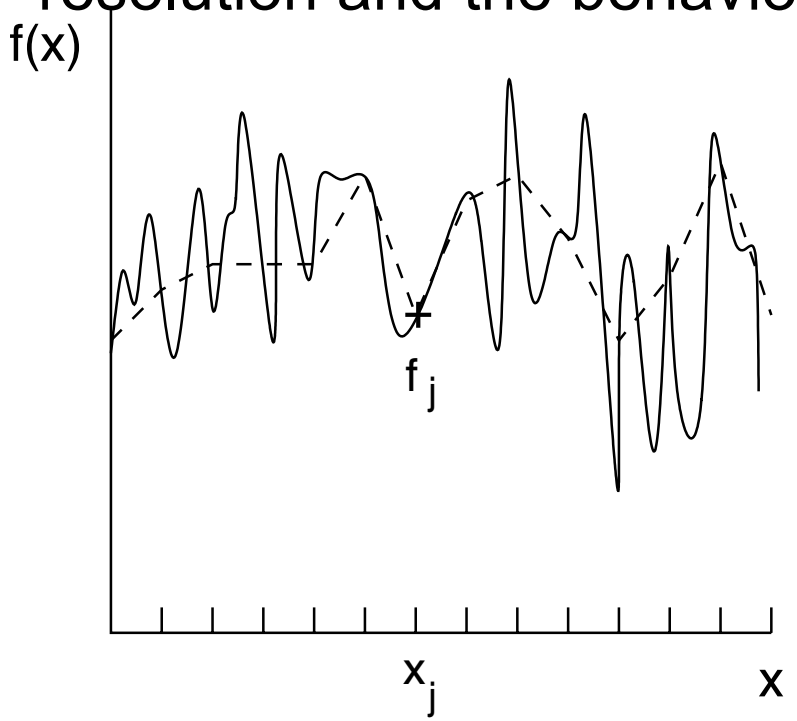
Given a dependent function f(x) defined on the domain, a discrete approximation of f(x) is given by

$$f_i = f(x_i)$$

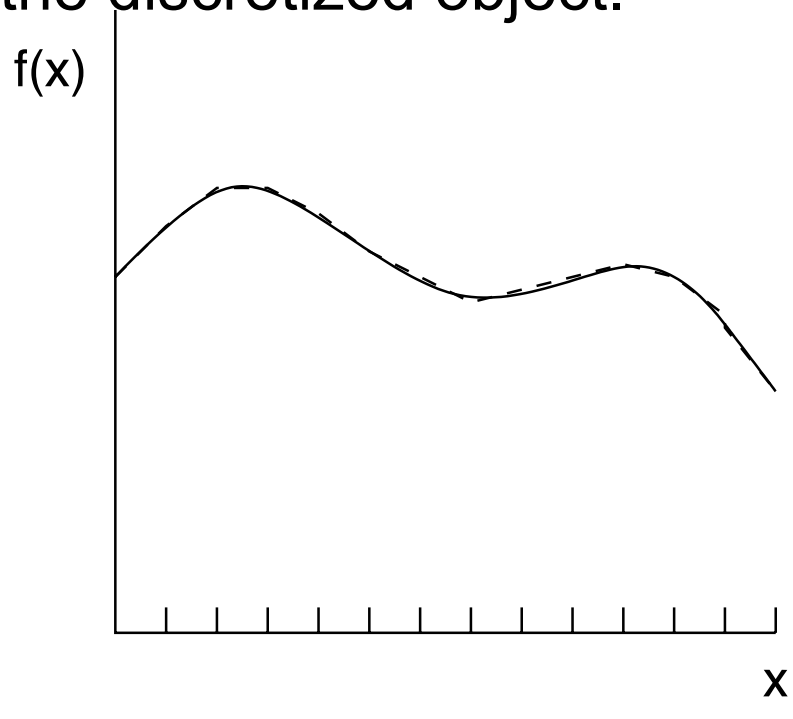
Similarly, given a differential equation on the domain, you can find a *discrete representation of the equation. Such representation is* called *the difference equation.*

QUALITY OF DISCRETE REPRESENTATION

The quality of the discrete representation depends on the resolution and the behavior of the discretized object.



Poor Approximation



Good Approximation

Ensure adequate resolution by representing the smallest features by at least several points.

WHAT ARE FINITE DIFFERENCE METHODS?

- Used to solve ODE or PDE.
- Based on the Taylor expansion.
- Probably the most simple method to understand.
- Discretization:
  - Spatial: Divide space $x \in [0,L]$ into N segments, $\Delta x = L/N$. Calculate derivatives of a function to use in solving a differential equation. Fix spatial coordinates (the Eulerian description).
  - Temporal: Set a time step of size Δt, and calculate the new value of a function at $t + \Delta t$.
  - There are many ways to discretize an equation; accuracy and stability depends on it.
- Turns differential equations into algebraic equations.

CHARACTERIZATION OF DISCRETIZATION ACCURACY

An approximation to a quantity is nth order accurate if nth order term in the Taylor expansion of the quantity is correctly reproduced.

Example: First Derivative of *f(x)*

The Taylor expansion of *f(x)* around x_i gives *f'(x)*, *f''(x)*, etc.

Method 1. One-sided (forward) differencing on *x*

$$f(x_{i+1}) - f(x_i) = \overbrace{f(x_i) + f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 + \ldots}^{\text{Taylor expansion for } f(x_{i+1}) \text{ around } x_i} - f(x_i)$$

$$\approx f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2$$

$$\longrightarrow f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + \underbrace{\mathcal{O}(\Delta x)}_{\text{truncation error}}$$

Method 2. One-sided (backward) differencing on *x*

$$f(x_i) - f(x_{i-1}) = f(x_i) - (f(x_i) - f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 - ...)$$

$$\approx f'(x_i)\Delta x - \frac{1}{2}f''(x_i)\Delta x^2$$

$$\longrightarrow f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + \underbrace{\mathcal{O}(\Delta x)}_{\text{truncation error}}$$

Method 3. Center differencing on *x*

$$f(x_{i+1}) - f(x_{i-1}) \approx f(x_i) + f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 + \frac{1}{6}f'''(x_i)\Delta x^3$$

$$- (f(x_i) - f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 - \frac{1}{6}f'''(x_i)\Delta x^3)$$

$$= 2f'(x_i)\Delta x + \frac{1}{3}f'''(x_i)\Delta x^3$$

$$\longrightarrow f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + \underbrace{\mathcal{O}(\Delta x^2)}_{\text{truncation error}}$$

INTEGRATION OF ODE/INITIAL VALUE PROBLEM

First-order ODEs occur very commonly in physics, with the variable being time. An initial value problem is specified by a differential equation (or a set) for a function $u(t)$ and its initial value, $u(t = t_o)$.

Consider an example: $$\frac{du}{dt} = f(u,t)$$

We want to find an approximate solution. We'll do this by step in time with size Δt, i.e., find $u(t + \Delta t)$ given $u(t)$. Let's use the results we obtained for forward and centered differencing:

$$\left.\frac{du}{dt}\right|_{t=t^n} = \begin{cases} [u(t^{n+1}) - u(t^n)]/\Delta t \ , & \text{Forward} \\ [u(t^{n+1}) - u(t^{n-1})]/2\Delta t \ , & \text{Centered} \end{cases}$$

Take the forward differencing case (denoting the current time and value of u with t^n and u^n, respectively):

$$u^{n+1} = u^n + f(u,t) \cdot \Delta t$$

You have a choice of which t to evaluate $f(u, t)$. Since we do not know the value of $u(t)$ at t^{n+1}, let's simply evaluate $f(u, t)$ at $t = t^n$.

$$u^{n+1} = u^n + f(u^n, t^n) \cdot \Delta t$$

This method is called the forward Euler time stepping.

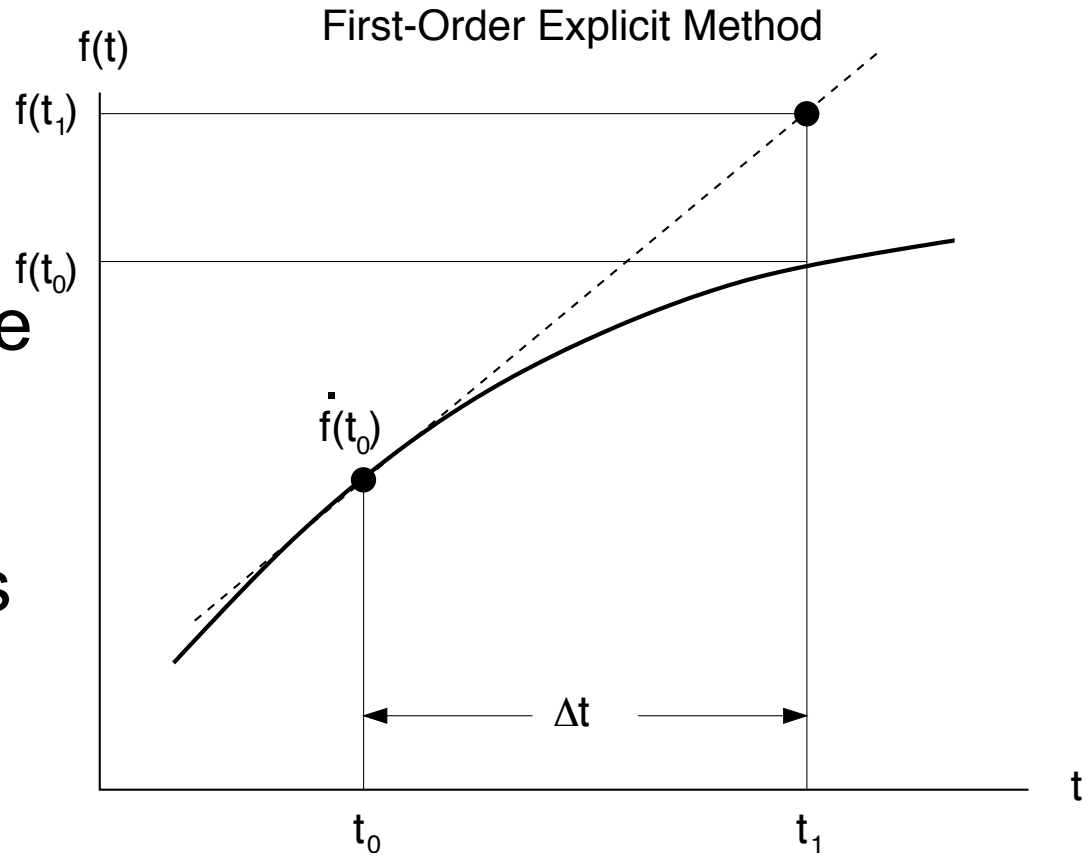
The centered differencing leads to the Leap-Frog method, which is unstable.

For PDEs, the derivatives on the right hand side are also discretized in the same manner.

TEMPORAL INTEGRATION SCHEMES

The schemes for integrating a differential equation in time can be divided into three categories.

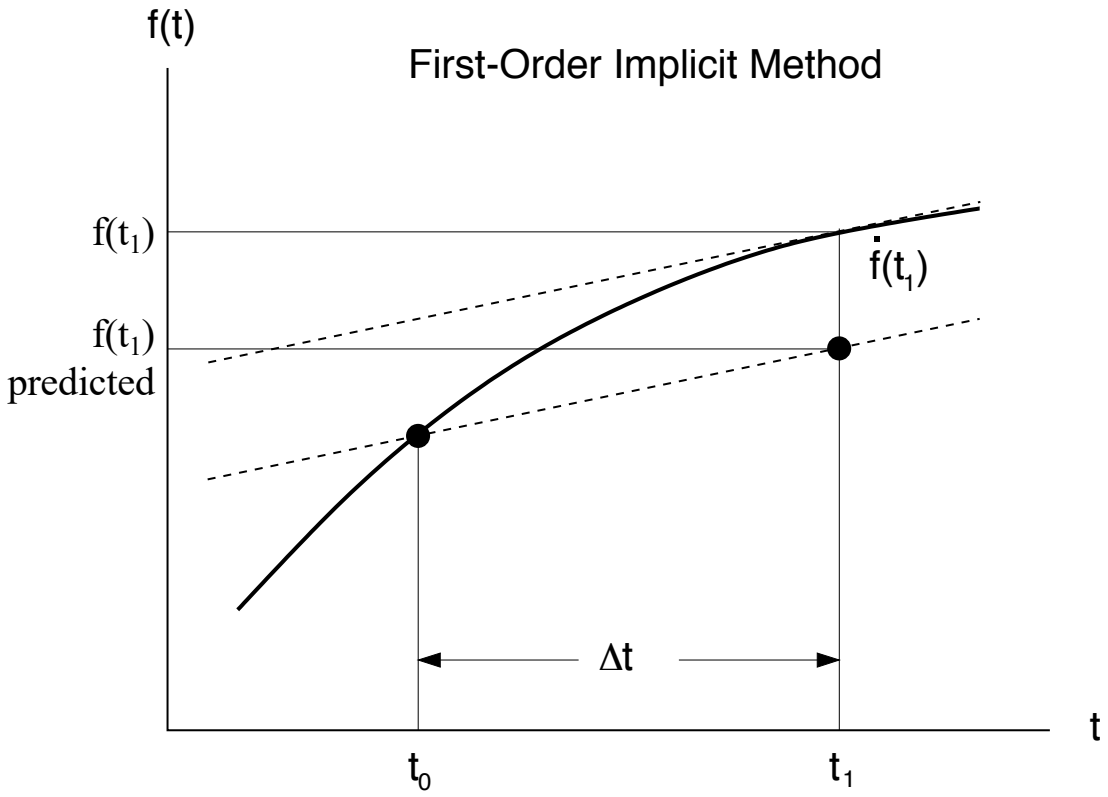
Explicit Method The evaluation of a future quantity does not require the future value itself. The schemes on the previous page fall in this category.



Implicit Method The evaluation of a future quantity requires the future value itself. This usually results in solving a matrix equation or iteration.

Semi-implicit Method A PDE is often in the form

$$\frac{\partial u}{\partial t} = (L_1 + L_2)u$$



where L1 and L2 are spatial differential operators. Sometimes only one of the differential operators benefit significantly from implicit treatment. In such cases, a technique called “operator splitting” is used which allows a different time-stepping scheme to each of the differential operators.

3. Errors & Uncertainties

SOURCES OF ERRORS

Model Error: Differences between the physical system and the "equivalent" as described by governing equations, initial condition and boundary conditions.

Truncation or Discretization Error: This error results from converting the analytical form of equations to a discretized form that can be solved numerically. Error analysis gives you an estimate of error. Accuracy depends on the resolution (in both spatial and time coordinates).

Rounding Error: This error results from floating operations in digital equipment. Accuracy of computational operations generally receives little attention. Often, carrying enough precision provides sufficient accuracy. However, in special cases, care must be taken.

MODELS

This is probably the most difficult issue to address and examine.

Questions:

- What approximations are made in deriving the governing equations? (e.g., sharp interface model). How good or bad are they?
- What effects excluded in the model are there, and what are the magnitude of the effects?
- Are you using physically consistent parameters? If not, what are the effects? (e.g., gradient coefficient)
- If there are input parameter/function to a model, how good are they? What are the range of errors, and how sensitive is your model to the changes in the input? (e.g., free energy, potential, etc.)

DISCRETIZATION ERROR

The magnitude and behavior of error can be estimated by error analysis.

Ex. Verlet algorithm (for MD calculation)

As MD is a Lagrangian formulation (i.e., we follow the location of mass), the location of an atom needs to be evolved in time as prescribed by the force field. Given $f(t)$ on the mass m at $r(t)$, Verlet method gives the updated location by

$$r(t + \Delta t) \approx 2r(t) - r(t - \Delta t) + (f(t)/m)\Delta t^2$$

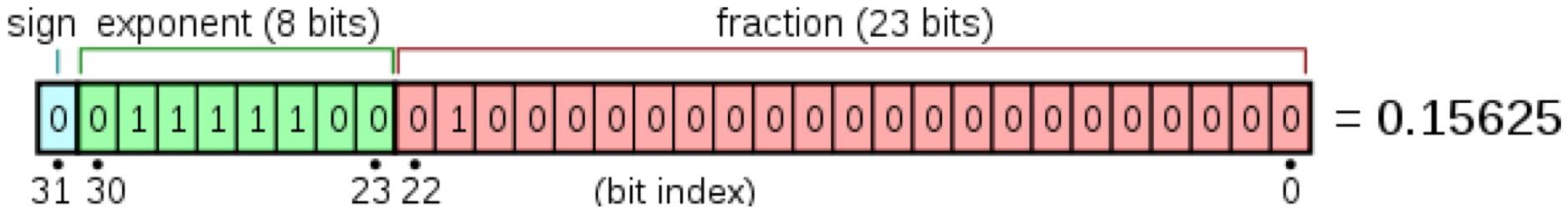
How much error is there in this approximation? $O(\Delta t^4)$.

There are in fact many ways to discretize a single equation. The accuracy (and stability) must be checked for each method-equation combination.

ROUNDING ERROR

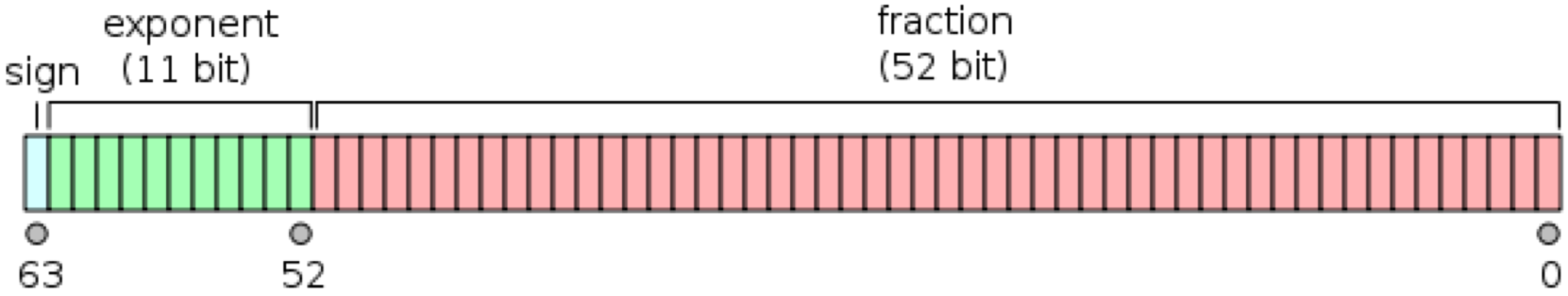
Computers represent numbers in binary

Single precision binary floating-point format (in a 32bit machine)



From Wikipedia

Double precision binary floating-point format



$$(-1)^{\text{sign}} \times 2^{\text{exponent}-\text{exponent bias}} \times 1.\text{mantissa}$$

Each has specific maximum and minimum numbers & rounding error

ROUNDING ERROR

Can be important!

- Cancellation error, e.g., a small difference such as $1 - \exp(-\delta)$ when δ is small.
- Recursive relations, where errors can propagate/accumulate and numerical instabilities may occur.
- Sums of terms with greatly varied magnitude, e.g., $\sum_{k=1}^{\infty} k^{-2} (= \pi^2/6)$. If summed from $k = 1$ until the change is small, F90 single precision calculation gives only 4 digits of accuracy (out of 9). If summed large to small, you get 8 digits of accuracy.
- Dynamical evolution or iteration not associated with minimization of residual error

4. Validation & Verification

Validation & Verification: Why?

Why is it important? A simulation tool can become a black box.

Black box

- Dictionary definition: a usually complicated electronic device that functions and is packaged as a unit and whose internal mechanism is usually hidden from or mysterious to the user; *broadly* **:** anything that has mysterious or unknown internal functions or mechanisms
- As simulation codes get complex, it can become a black box. The input may be of good quality, but it does not guarantee good output quality.

Validation & Verification (V&V)

1. **Numerical Accuracy (assuming bug-free)** Easy to address. The simplest way is to increase the resolution in space and time and the change is within the acceptable range (i.e., numerically converged).
2. **Physical Assumptions (in the model)** Sometimes difficult. If ignoring some effects (e.g., terms in equations), verify that the magnitude of the effect is small compared to other effects that are included (e.g., compare energies associated with different effects). Also check that this is true throughout the simulation if possible. Dimensionless numbers are often useful in determining the regimes that certain effects must be considered/can be ignored. Ultimately, a direct comparison to experiment is the best, but it is not always possible.

Dimensionless numbers in fluid dynamics

Name	Standard symbol	Definition
Archimedes number	Ar	$\mathrm{Ar} = \frac{gL^3\rho_\ell(\rho-\rho_\ell)}{\mu^2}$
Atwood number	A	$\mathrm{A} = \frac{\rho_1-\rho_2}{\rho_1+\rho_2}$
Bejan number (fluid mechanics)	Be	$\mathrm{Be} = \frac{\Delta P L^2}{\mu\alpha}$
Bingham number	Bm	$\mathrm{Bm} = \frac{\tau_y L}{\mu V}$
Biot number	Bi	$\mathrm{Bi} = \frac{hL_C}{k_b}$
Blake number	Bl or B	$\mathrm{B} = \frac{u\rho}{\mu(1-\epsilon)D}$
Bond number	Bo	$\mathrm{Bo} = \frac{\rho a L^2}{\gamma}$
Brinkman number	Br	$\mathrm{Br} = \frac{\mu U^2}{\kappa(T_w - T_0)}$
Brownell–Katz number	N_{BK}	$\mathrm{N_{BK}} = \frac{u\mu}{k_{rw}\sigma}$
Capillary number	Ca	$\mathrm{Ca} = \frac{\mu V}{\gamma}$
Chandrasekhar number	C	$\mathrm{C} = \frac{B^2 L^2}{\mu_o \mu D_M}$
Colburn J factors	J_M, J_H, J_D	
Damkohler number	Da	$\mathrm{Da} = k\tau$
Darcy friction factor	C_f or f_D	
Dean number	D	$\mathrm{D} = \frac{\rho V d}{\mu}\left(\frac{d}{2R}\right)^{1/2}$
Deborah number	De	$\mathrm{De} = \frac{t_c}{t_p}$
Drag coefficient	c_d	$c_\mathrm{d} = \frac{2F_\mathrm{d}}{\rho v^2 A}$,
Eckert number	Ec	$\mathrm{Ec} = \frac{V^2}{c_p \Delta T}$
Eötvös number	Eo	$\mathrm{Eo} = \frac{\Delta\rho\, g\, L^2}{\sigma}$
Ericksen number	Er	$\mathrm{Er} = \frac{\mu v L}{K}$
Euler number	Eu	$\mathrm{Eu} = \frac{\Delta p}{\rho V^2}$
Excess		

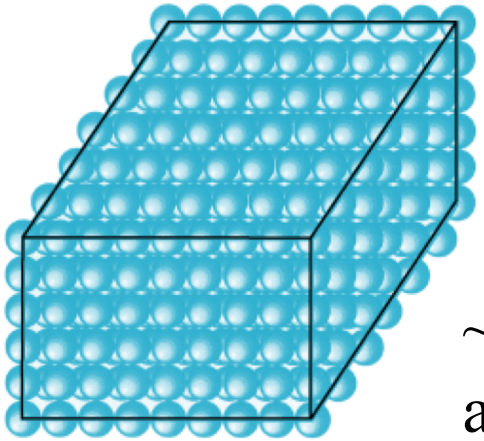
Validation & Verification (Continued)

3. System Size System size is often limited by the computational resources. Most physical systems have a much larger system size than those that can be simulated.

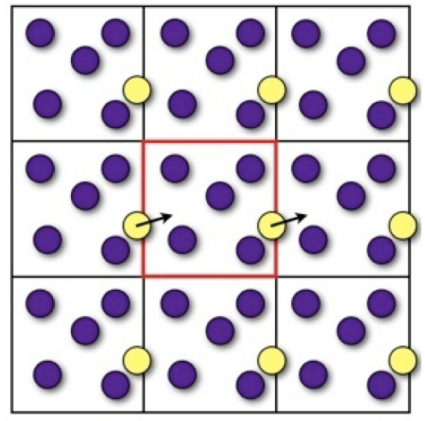
a. Edge Effect

• A simple cubic crystal of 1000 atoms has 49% (in 3D) of them on the surface, and the edge effect becomes important.

• Consider using periodic boundary conditions at the cell edges.



~500 atoms



Validation & Verification (Continued)

b. Size Effect

- If the periodic unit cell is too small compared to the feature within it, or if a point can see its own image, the results become influenced by the system size.
- This manifests as domination and damping of features with certain wavelengths as any fluctuation must be compatible with the imposed periodicity.
- Unless the size of the lattice is physically consistent, it may lead to unphysical effects, such as an artificial ordering.
- Verification: Study the effects of limited system size by using a smaller or (preferably) larger system and compare.

Validation & Verification (Continued)

4. Code/method verification It is difficult to write a bug-free code! Test runs can often give indications for bugs.

a. Using analytical solutions

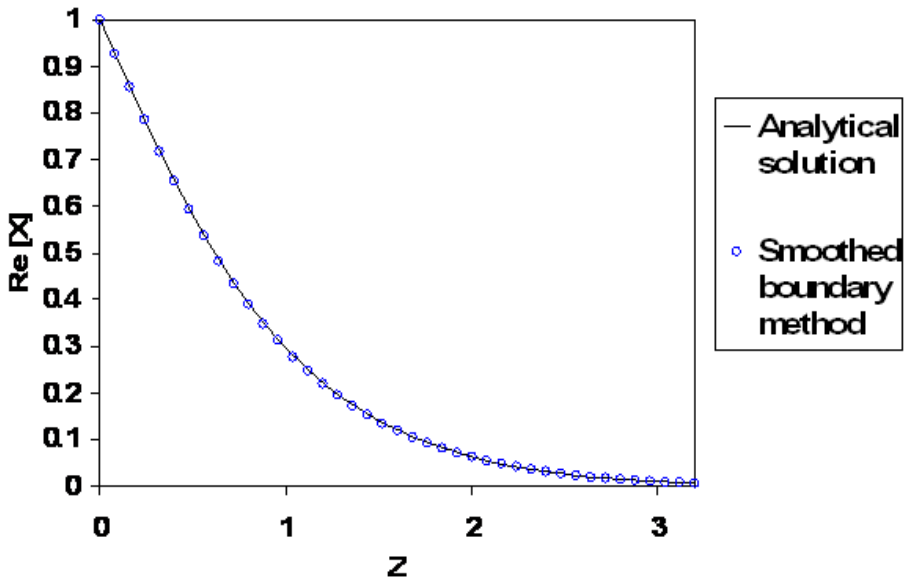
- Set up simulations that corresponds to setups that have analytical solutions (Ex. Reaction-diffusion in cylinder); more later.

b. Using your physical intuition

- Always think about what you would expect to see from a simulation. Does the result make sense?

c. Unit tests

- Test piece by piece.



Validation & Verification (Continued)

4. Code/method verification (continued)

d. Method of Manufactured Solutions

- For the given PDE, add a forcing term and assume an analytical solution. Obtain the analytical form of the forcing function.

e. Debugging

- Always start with small calculations!!
- Use debugger & check points
- Be detailed

Use of Analytical Methods

Even if your goal is to study a system that is very complicated, it is always a good idea to simplify the problem and study it analytically.

The results can be used to validate your code, as it is always easy to set up a simple case in robust numerical simulation. This is important as it is not very difficult to leave a bug in a big codes.

It also gives you an appreciation and understanding of the physical problem at a higher level. You may learn something new and get an extra publication on the side.

5. Programming

Good coding practice

Efficiency

Pointers on Programming (Discussion)

- Key: Break down the task into smaller pieces
  - Outlines are helpful in doing this
- Identify repeated tasks and make a subroutine or function
- Stream-line the tasks by looking from the bird's eye view (for the big picture)
- Get the details right by being meticulous
- Make the variable names something anyone can understand
- Write in comments
- Use version control software
- Learn to use debugger
- Know how to check your solutions

Pointers on Programming (Discussion)

- Key: Break down the task into smaller pieces
  - Outlines are helpful in doing this
- Identify repeated tasks and make a subroutine or function
- Stream-line the tasks by looking from the bird's eye view (for the big picture)
- Get the details right by being meticulous
- Make the variable names something anyone can understand
- Write in comments
- Use version control software
- Learn to use debugger
- Know how to check your solutions

Efficiency in Computation

- CPU time efficiency
- Storage efficiency
- Programming efficiency (human time)

Ask:

1. How long would a calculation take?
2. How much memory/storage does it take? (Use 8 byte per double precision)
3. How much time do I want to spend programming?

Find the limiting factor. If the computation takes too long using a simple method, you would have to pay more attention to this issue and compromise on others.

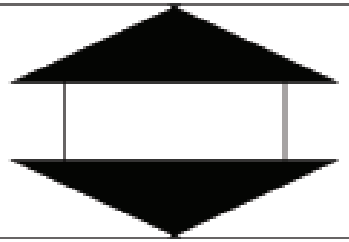
Trade-offs: Example for Time stepping

Higher Order Scheme (using previous data)

Higher CPU time efficiency (larger Δt)

Lower storage efficiency

Lower programming efficiency
(more complicated, esp. for adaptive mesh)



Lower CPU time efficiency

Higher storage efficiency

Higher programming efficiency

Lower Order Scheme