Scientific Computing

Maastricht Science Program

Week 2

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Recap

- What is scientific programming?
- Programming
  - Arithmetic, IF, conditions, WHILE, FOR
  - Matlab Cheat Sheet

- General form of linear equations $a_0 + a_1 x_1 + a_2 x_2 + ... = 0$
- Finding the zeros of non-linear equations
  - bisection
  - Newton

This Lecture

- A very short introduction linear algebra
  - Vectors & Matrices in Matlab
  - LU factorization
- Floating Point Numbers
- Computation
  - Computation Errors
  - Computational Costs

A Very Short Introduction to Linear Algebra

Linear Algebra (LA)

- Linear Algebra deals with linear functions
  - You know what that is!
  - but higher dimensions $R^n \to R^m$
- I can only give a very brief introduction
  - covering only basic things
- Please:
  - get a linear algebra book, open it!
  - Watch some video lectures.
    - E.g., the first couple at:

      http://web.mit.edu/18.06/www/videos.shtml

Motivation

- LA is the basis of **many** methods in science
- For us:
  - Important to solve systems of linear equations

$$a_1 x_1 + a_2 x_2 + \ldots = c$$

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n &= c_1 \\ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n &= c_2 \\ &\ldots \\ a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n &= c_m \end{aligned}$$

- Arise in many problems, e.g.:
  - Identifying gas mixture from peaks in spectrum
  - fitting a line to data. (Next week)

Motivation

- LA is the basis of **many** methods in science

- x_j - the amount of gas of type j
- a_{ij} - how much a gas of type j contributes to wavelength i
- c_i - the height of the peak of wavelength i

ems of linear equations

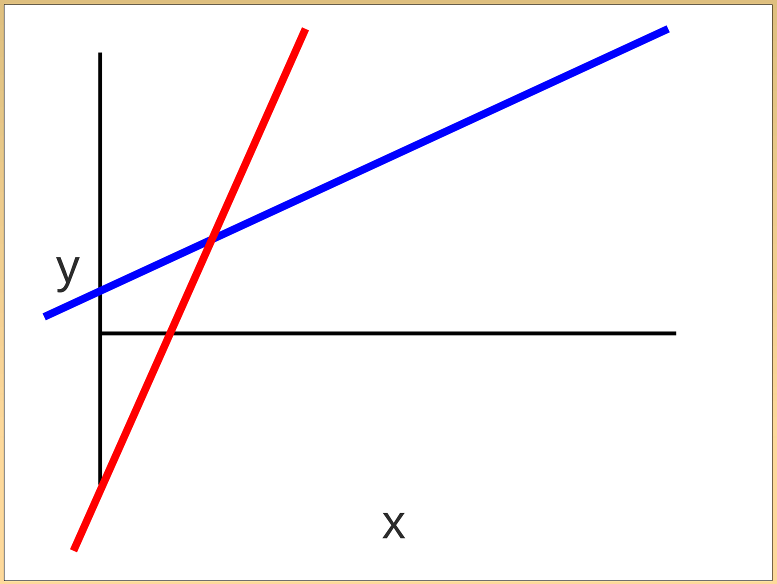
$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= c_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= c_2 \\
&\ldots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= c_m
\end{aligned}
$$

- Arise in many problems, e.g.:
  - Identifying gas mixture from peaks in spectrum
  - fitting a line to data. (Next week)

Linear System of Equations

- Example

$$y = 0.5x + 1$$
$$y = 2x - 3$$



- Infinitely many, 1 or no solution

Matrices

- A **matrix** with
  - m rows,
  - n columns

  is a collection of numbers
  - represented as a table

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 54 & 6 \\ 75 & 24 & 81 \\ 25 & 5 & 435 \end{bmatrix}$$

- A **vector** is a matrix that is
  - 1 row (row vector), or
  - 1 column (column vector)

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix}$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix}$$

Matrices

- A **matrix** with
  - m rows,
  - n columns

  is a collection of numbers
  - represented as a table

- A **vector** is a matrix that is
  - 1 row (row vector), or
  - 1 column (column vector)

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

```
octave:1> A = [3, -2, 6; 5, 2, -8]
A =

   3  -2   6
   5   2  -8

octave:2> w = [5;75;25]
w =

    5
   75
   25
```

$$B = \begin{bmatrix} 5 & 54 & 6 \\ 75 & 24 & 81 \\ 25 & 5 & 435 \end{bmatrix}$$

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix}$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix}$$

Matrices

- A **matrix** with
  - m rows,
  - n columns

  is a collection of numbers
  - represented as a table
- A **vector** is a matrix that is
  - 1 row (row vector), or
  - 1 column (column vector)

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

```
octave:1> A = [3, -2, 6; 5, 2, -8]
A =

   3  -2   6
   5   2  -8

octave:2> w = [5;75;25]
w =

    5
   75
   25
```

$$B = \begin{bmatrix} 5 & 54 & 6 \\ 75 & 24 & 81 \\ 25 & 5 & 435 \end{bmatrix}$$

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix}$$

```
octave:3> a1 = [4:8]
a1 =

   4   5   6   7   8

octave:4> a2 = [4:2:8]
a2 =

   4   6   8
```

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix}$$

Some Special Matrices

- **Square** matrix: m=n
- **Identity** matrix - 'eye(3)'
- **Zero** matrix – 'zeros(m,n)'

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Types: diagonal, triangular (upper & lower)

$$D = \begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix} \quad TU = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix} \quad TL = \begin{bmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{bmatrix}$$

  - '*' denotes any number

Operations on Vectors - 1

- We can perform operations on them!
  - First: vectors. Next: generalization to matrices.

- **Transpose**: convert row ↔ column vector

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix} \qquad v^T = \begin{bmatrix} 3 \\ -2 \\ 6 \end{bmatrix}$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix} \qquad w^T = \begin{bmatrix} 5 & 75 & 25 \end{bmatrix}$$

Operations on Vectors - 1

- We can perform operations on them!
  - First: vectors. Next: generalization to matrices.

- **Transpose**: convert row ↔ column vector

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix} \qquad v^T = \begin{bmatrix} 3 \\ -2 \\ 6 \end{bmatrix}$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix} \qquad w^T = \begin{bmatrix} 5 & 75 & 25 \end{bmatrix}$$

```
octave:9> a = [1,4,-2498, 12.4]
a =

      1.0000      4.0000  -2498.0000      12.4000

octave:10> a'
ans =

      1.0000
      4.0000
  -2498.0000
     12.4000

octave:11> a''
ans =

      1.0000      4.0000  -2498.0000      12.4000
```

Operations on Vectors - 2

- Sum $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} = \begin{bmatrix} 11 & 22 & 33 \end{bmatrix}$
- Product with scalar $5 * \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 5 & 10 & 15 \end{bmatrix}$

- Inner product (also: 'scalar product' or 'dot product')

$$(v, w) = v^T w = \sum_{k=1}^{n} v_k w_k$$

Operations on Vectors - 2

- Sum $\begin{bmatrix}1 & 2 & 3\end{bmatrix}+\begin{bmatrix}10 & 20 & 30\end{bmatrix}=\begin{bmatrix}11 & 22 & 33\end{bmatrix}$
- Product with scalar $5*\begin{bmatrix}1 & 2 & 3\end{bmatrix}=\begin{bmatrix}5 & 10 & 15\end{bmatrix}$

- Inner product (also: 'scalar product' or 'dot product')

$$(v,w)=v^T w=\sum_{k=1}^{n} v_k w_k$$

$$v=\begin{bmatrix}1\\2\\3\end{bmatrix}, w=\begin{bmatrix}10\\20\\30\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}10\\20\\30\end{bmatrix}=1*10+2*20+3*30=10+40+90=140$$

Operations on Vectors - 2

- Sum $\begin{bmatrix}1 & 2 & 3\end{bmatrix}+\begin{bmatrix}10 & 20 & 30\end{bmatrix}=\begin{bmatrix}11 & 22 & 33\end{bmatrix}$
- Product with scalar $5*\begin{bmatrix}1 & 2 & 3\end{bmatrix}=\begin{bmatrix}5 & 10 & 15\end{bmatrix}$
- Inner product (also: 'scalar product' or 'dot product')

$$(v,w)=v^T w=\sum_{k=1}^{n} v_k w_k$$

$$v=\begin{bmatrix}1\\2\\3\end{bmatrix}, w=\begin{bmatrix}10\\20\\30\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}10\\20\\30\end{bmatrix}=1*10+2*20+3*30=10+40+90=140$$

```
octave:4> a = [1;2;3]
a =

   1
   2
   3

octave:5> b = [4;5;6]
b =

   4
   5
   6

octave:6> dot(a,b)
ans =  32
octave:7> a'*b
ans =  32
```

Operations on Vectors - 2

- Sum $$\begin{bmatrix}1 & 2 & 3\end{bmatrix}+\begin{bmatrix}10 & 20 & 30\end{bmatrix}=\begin{bmatrix}11 & 22 & 33\end{bmatrix}$$
- Product with scalar $$5*\begin{bmatrix}1 & 2 & 3\end{bmatrix}=\begin{bmatrix}5 & 10 & 15\end{bmatrix}$$
- Inner product (also: 'scalar product' or 'dot product')

$$(v,w)=v^T w=\sum_{k=1}^{n} v_k w_k$$

$$v=\begin{bmatrix}1\\2\\3\end{bmatrix}, w=\begin{bmatrix}10\\20\\30\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}10\\20\\30\end{bmatrix}=1*10+2*20+3*30=10+40+90=140$$

- Outer product (also: 'vector product')

Vector Indexing

- Retrieve parts of vectors

```
octave:12> a = [10, 20, 30, 40, 50, 60, 70]
a =

   10   20   30   40   50   60   70

octave:13> a(3)
ans =  30
octave:14> a([2,4])
ans =

   20   40

octave:16> a([4:end])
ans =

   40   50   60   70
```

Vector Indexing

- Retrieve parts of vectors

```
octave:12> a = [10, 20, 30, 40, 50, 60, 70]
a =

   10   20   30   40   50   60   70

octave:13> a(3)
ans =  30
octave:14> a([2,4])
ans =

   20   40

octave:16> a([4:end])
ans =

   40   50   60   70
```

indexing with another vector

special 'end' index

Operations on Matrices - 1

- Now matrices!
- **Transpose**:
  - convert each row → column vector
    (or convert each column → row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

Operations on Matrices - 1

- Now matrices!
- **Transpose**:
  - convert each row → column vector
    (or convert each column → row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

Operations on Matrices - 1

- Now matrices!
- **Transpose**:
  - convert each row → column vector
    (or convert each column → row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

Operations on Matrices - 1

- Now matrices!
- **Transpose**:
  - convert each row → column vector
    (or convert each column → row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix} \qquad B^T = \begin{bmatrix} 1 & 10 \\ 2 & 20 \\ 3 & 30 \end{bmatrix}$$

Operations on Matrices - 2

- **Sum** and **product with scalar**: pretty much the same

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix} = \begin{bmatrix} 11 & 22 & 33 \\ 44 & 55 & 66 \end{bmatrix}$$

$$5 * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 10 & 15 \\ 20 & 25 & 30 \end{bmatrix}$$

Matrix Product

- Inner product → **Matrix product**

$$C = AB$$

  - C = m x n, A = m x p, B = p x n,
  - Each entry of C is an inner product: $c_{ij} = r_i^A c_j^B$

$$\begin{bmatrix} \dots & \dots & \dots \\ \mathbf{190} & \dots & \dots \\ \dots & \dots & .. \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ \mathbf{30} & \mathbf{40} \\ 50 & 60 \end{bmatrix} \begin{bmatrix} \mathbf{1} & 2 & 3 \\ \mathbf{4} & 5 & 6 \end{bmatrix}$$

Matrix Product

- Inner product → **Matrix product**

$$C = AB$$

  - C = m x n,   A = m x p,   B = p x n,
  - Each entry of C is an inner product: $c_{ij} = r_i^A \cdot c_j^B$

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ 190 & \cdots & \cdots \\ \cdots & \cdots & .. \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
octave:22> A = [10, 20; 30, 40; 50, 60]
A =

   10   20
   30   40
   50   60

octave:23> B = [1,2,3;4,5,6]
B =

   1   2   3
   4   5   6

octave:24> A*B
ans =

    90   120   150
   190   260   330
   290   400   510
```

Matrix Product

- Inner product → **Matrix product**

$$C = AB$$

- C = m x n, A = m x p, B = p x n,
- Each entry of C is an inner product: $c_{ij} = r_i c_j$

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ 190 & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
octave:22> A = [10, 20; 30, 40; 50, 60]
A =

   10   20
   30   40
   50   60

octave:25> Btrans = B'
Btrans =

   1   4
   2   5
   3   6

octave:26> A*Btrans
error: operator *: nonconformant arguments (op1 is 3x2, op2 is 3x2)
```

Matrix size is important

Matrix-Vector Product

- Matrix-vector product is just a (frequently occurring) special case:

$$Ab = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_1 \\ \dots \\ b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \dots \\ c_m \end{bmatrix}$$

Matrix-Vector Product

- Also represents a system of equations!

$$Ax=\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}\begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}=\begin{bmatrix} c_1 \\ \dots \\ c_m \end{bmatrix}$$

$$\begin{gathered} a_{11}x_1+a_{12}x_2+\dots+a_{1n}x_n=c_1 \\ a_{21}x_1+a_{22}x_2+\dots+a_{2n}x_n=c_2 \\ \dots \\ a_{m1}x_1+a_{m2}x_2+\dots+a_{mn}x_n=c_m \end{gathered}$$

Matrix Inverse

- Matrix **inverse**
  - a square matrix A has an inverse A^{-1}, **if** $A^{-1}A=I$
  - A is called 'invertible'
  - generalization of scalar inverse $a^{-1}a=\frac{a}{a}=1$

- Why?
  - Solution of linear system of equations:

$$Ax=b$$

$$Ax=b$$
$$A^{-1}Ax=A^{-1}b$$
$$Ix=A^{-1}b$$
$$x=A^{-1}b$$

Matrix Inverse

- Matrix **inverse**
  - a square matrix A has an inverse A^{-1}, **if** $A^{-1}A = I$
  - A is called 'invertible'
  - generalization of scalar inverse $a^{-1}a = \frac{a}{a} = 1$
- Special case: diagonal matrix

$$A = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1/a_{11} & 0 & 0 \\ 0 & 1/a_{22} & 0 \\ 0 & 0 & 1/a_{33} \end{bmatrix}$$

Existence of Matrix Inverse

- Inverse does exist for every square matrix...
  - (there is a more general procedure, but can get divisions by 0 when following it.)

$$A^{-1}=\begin{bmatrix} 1/a_{11} & 0 & 0 \\ 0 & 1/a_{22} & 0 \\ 0 & 0 & 1/a_{33} \end{bmatrix}$$

- A^{-1} exists
  ↔ A is 'non singular'
  ↔ 'determinant' is not zero
  ↔ columns of A are **linearly independent**

- $\{v_{1,} \ldots, v_k\}$ are linearly independent if

$$a_1 v_1 + \ldots + a_k v_k = 0 \quad \Rightarrow \quad a_1 = 0, \ldots, a_k = 0$$

Solving Linear Systems

- So how to solve a linear system?
- 'inv'
  - only for square matrices
- '\' (left division)
  - careful! Will also find a solution if none exists!

```
octave:9> A = rand(4);
octave:10> c = rand(4,1);
octave:11> inv(A)*c
ans =

   0.905965
  -0.032969
   0.109202
   0.430893

octave:12> A\c
ans =

   0.905965
  -0.032969
   0.109202
   0.430893
```

Floating Point Numbers

How are number represented?

- Matlab represents numbers using a **floating point representation**

$$\left(-1\right)\cdot\left(0.b_1 b_2 \ldots b_{53}\right)\cdot 2^e$$

sign

mantissa
•53 bits - 2^{53}=9.0072e+15
•normalized: b_1 ~= 0
(unique representation)

Exponent
-1021< e < 1024

How are number represented?

- Matlab represents numbers using a **floating point representation**

$$(-1)\cdot(0.b_1 b_2 \ldots b_{53})\cdot 2^e$$

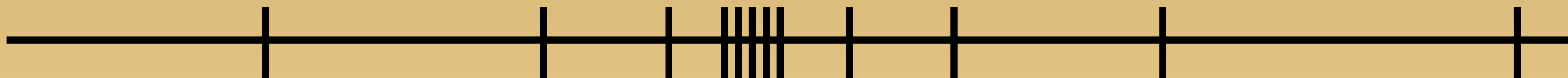
sign

mantissa
- 53 bits - 2^{53}=9.0072e+15
- normalized: b_1 ~= 0 (unique representation)

Exponent
-1021< e < 1024

- Smallest
  - normalized $(0.100\ldots00)\cdot 2^{-1021} = 2.2251\text{e-}308$
  - non-norm. $(0.000\ldots01)\cdot 2^{-1021} = 4.9407\text{e-}324$
- Largest $(0.111\ldots11)\cdot 2^{1024} = 1.7977\text{e+}308$

Spacing between numbers



- Spacing for the largest numbers

$$(0.000...001)\cdot 2^{1024}$$

$$(0.000...010)\cdot 2^{1024}$$

$$diff = (0.000...001)\cdot 2^{1024} = 1\cdot 2^{(1024-53)} = 1.9958e+292$$

- Spacing for smallest numbers 4.9407e-324

- “eps(n)” gives spacing around n
  - eps(realmax), eps(0)

Round Off Errors

- set of floating point numbers *F*
- when real number *x* is replaced by number *fl(x)* in *F*

  → round off error

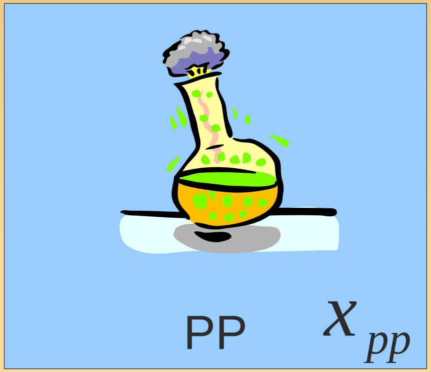
- Absolute error can be large: 0.5 *eps(realmax)

- However: *relative error* is bounded $$\frac{|x - fl(x)|}{|x|} \leqslant \frac{1}{2} \epsilon$$
  - where $\epsilon = eps(1) = 2.2204\text{e-}16$

Computation

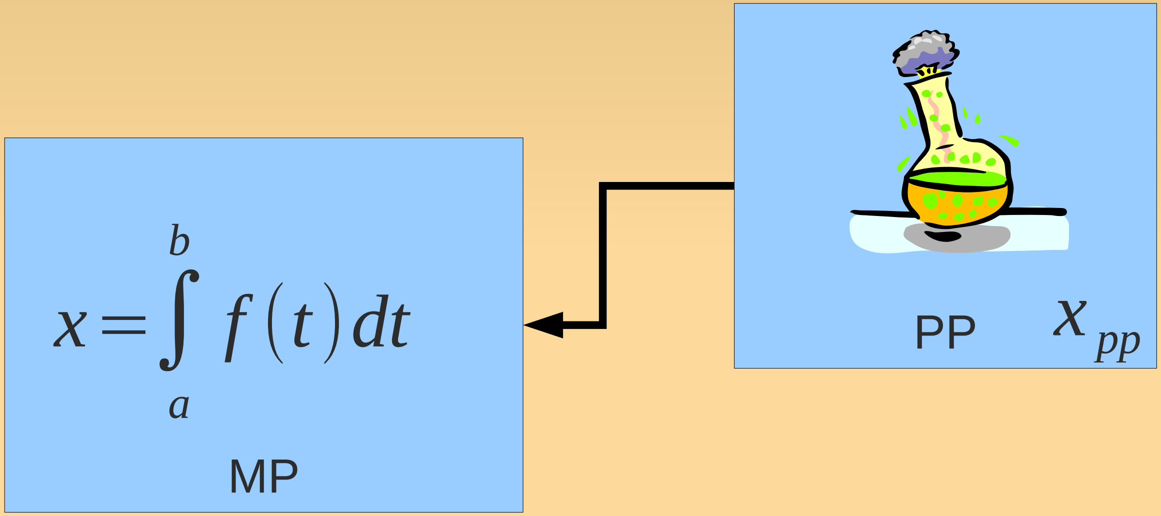
More on Errors

- Round off errors are only part of the story



More on Errors

- Round off errors are only part of the story



More on Errors

- Round off errors are only part of the story

PP x_{pp}

$$x = \int_a^b f(t)\,dt$$

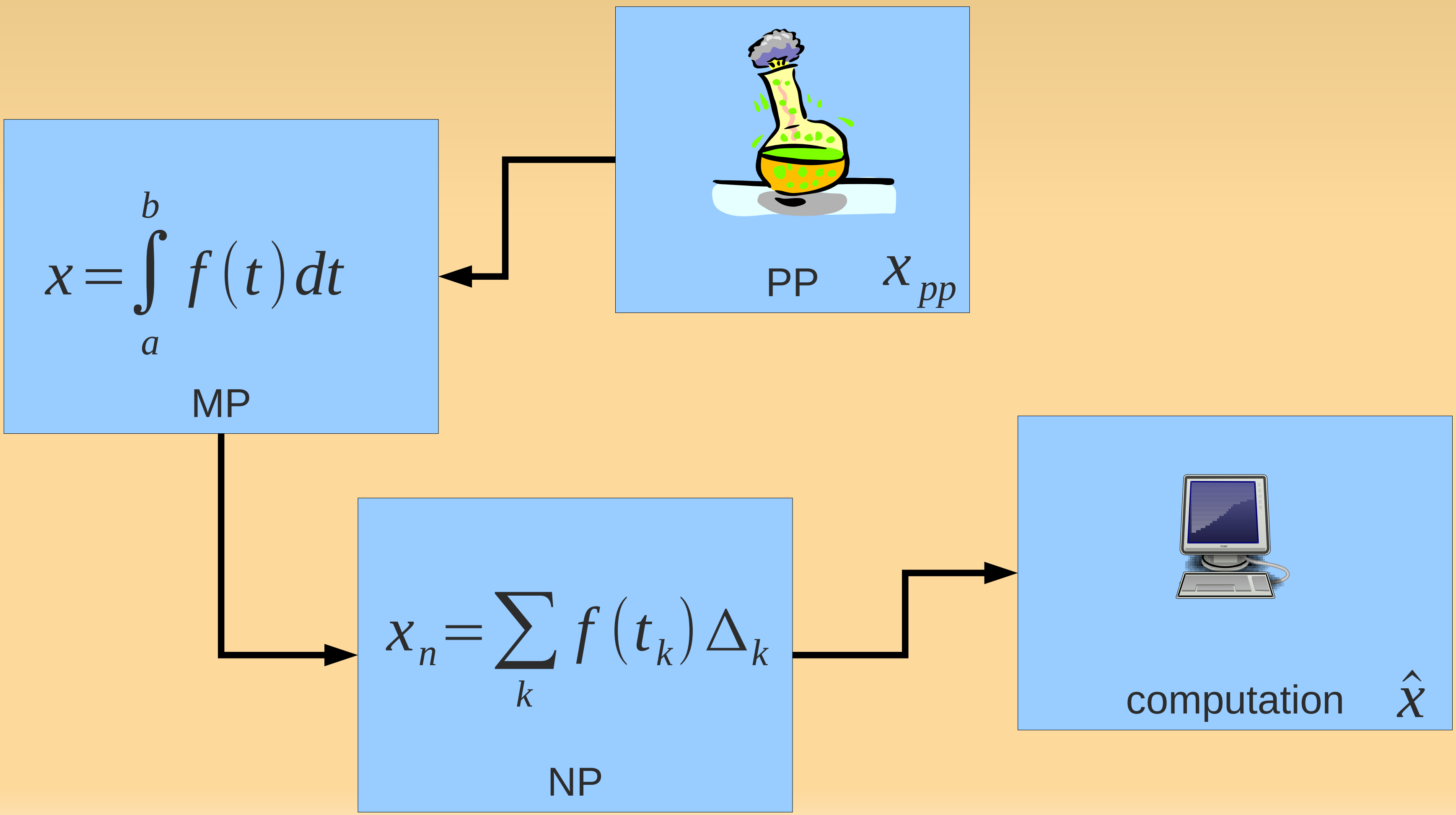
MP

$$x_n = \sum_k f(t_k)\,\Delta_k$$

NP

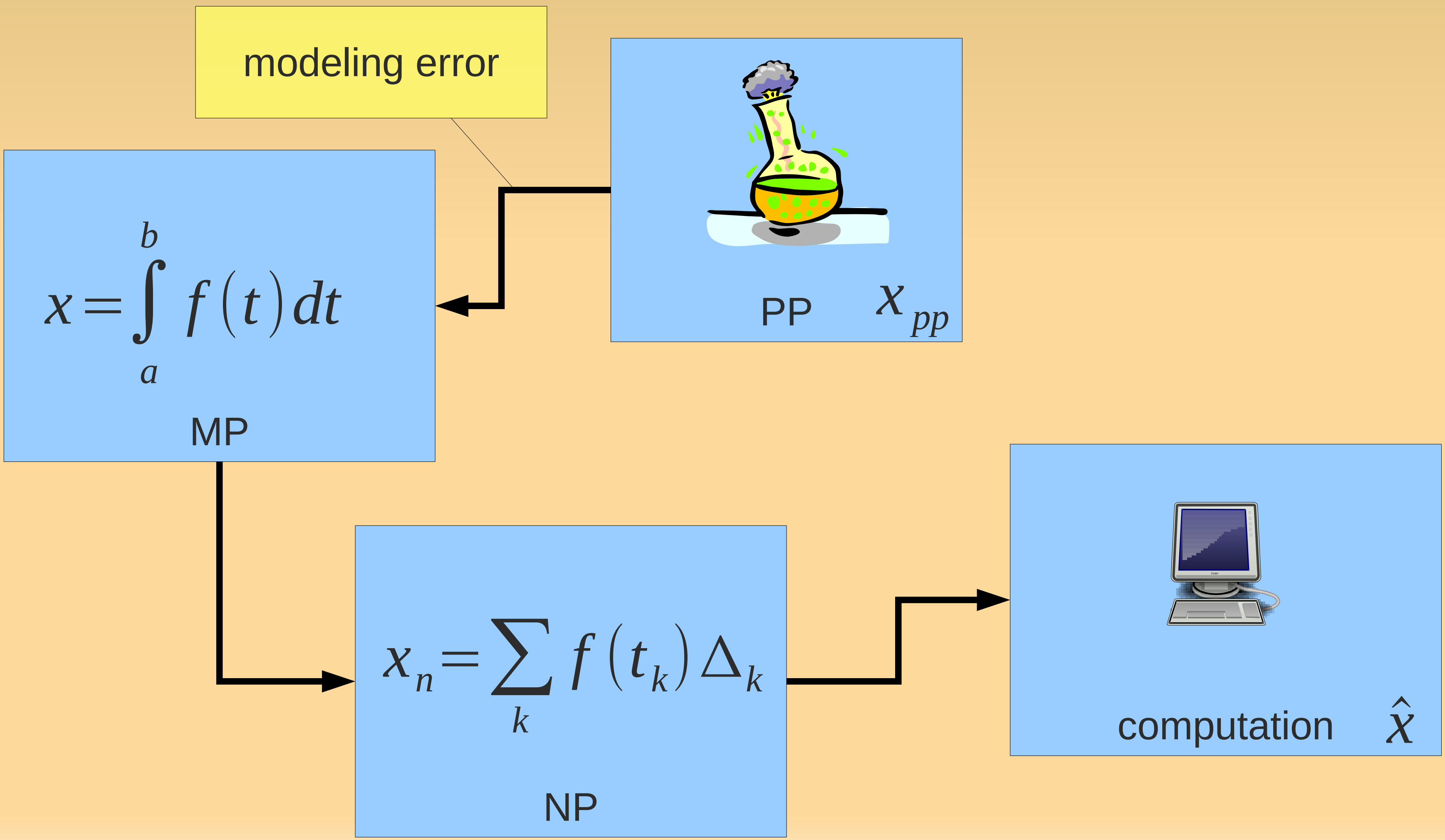
More on Errors

- Round off errors are only part of the story



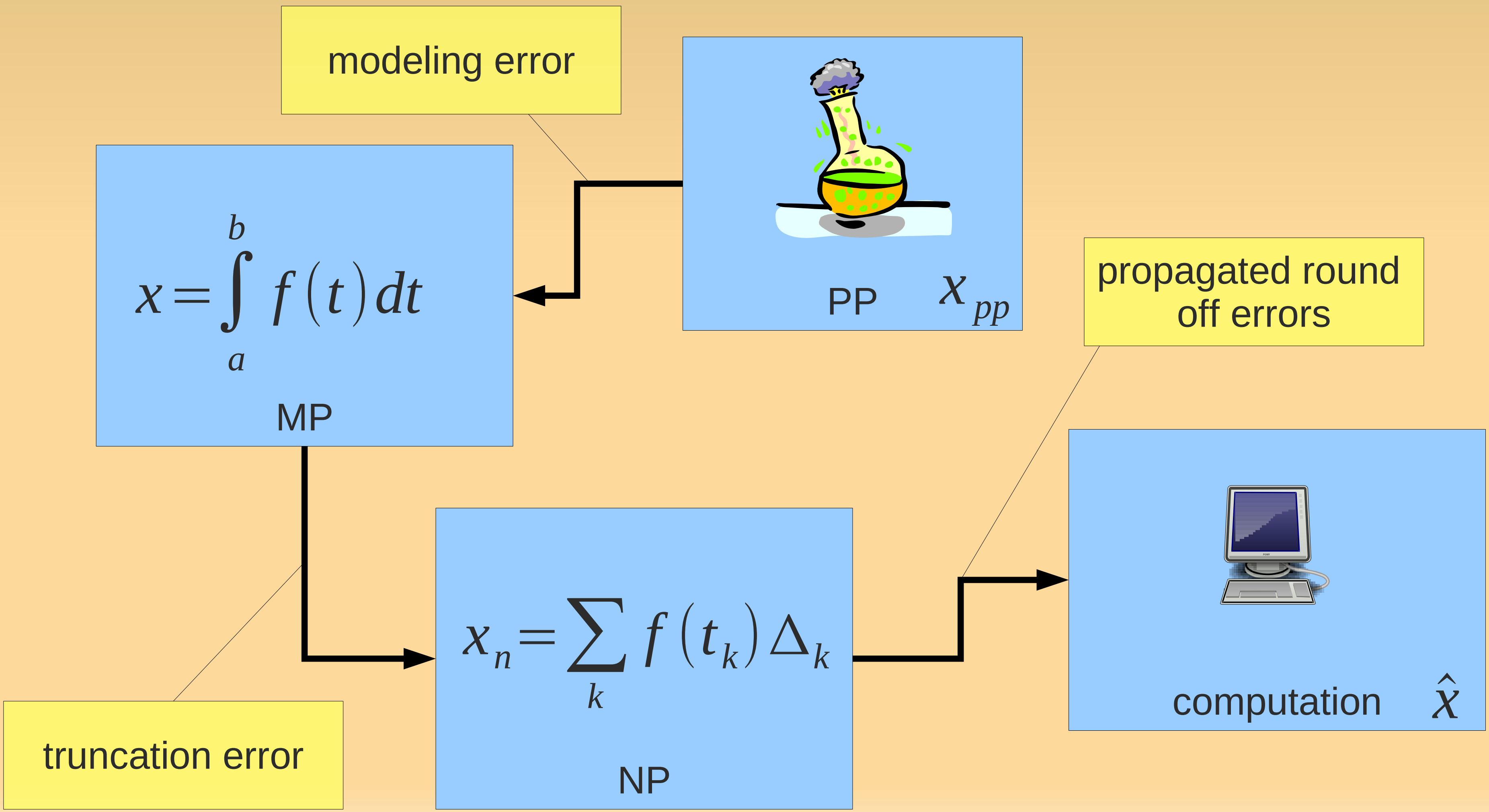
More on Errors

- Round off errors are only part of the story



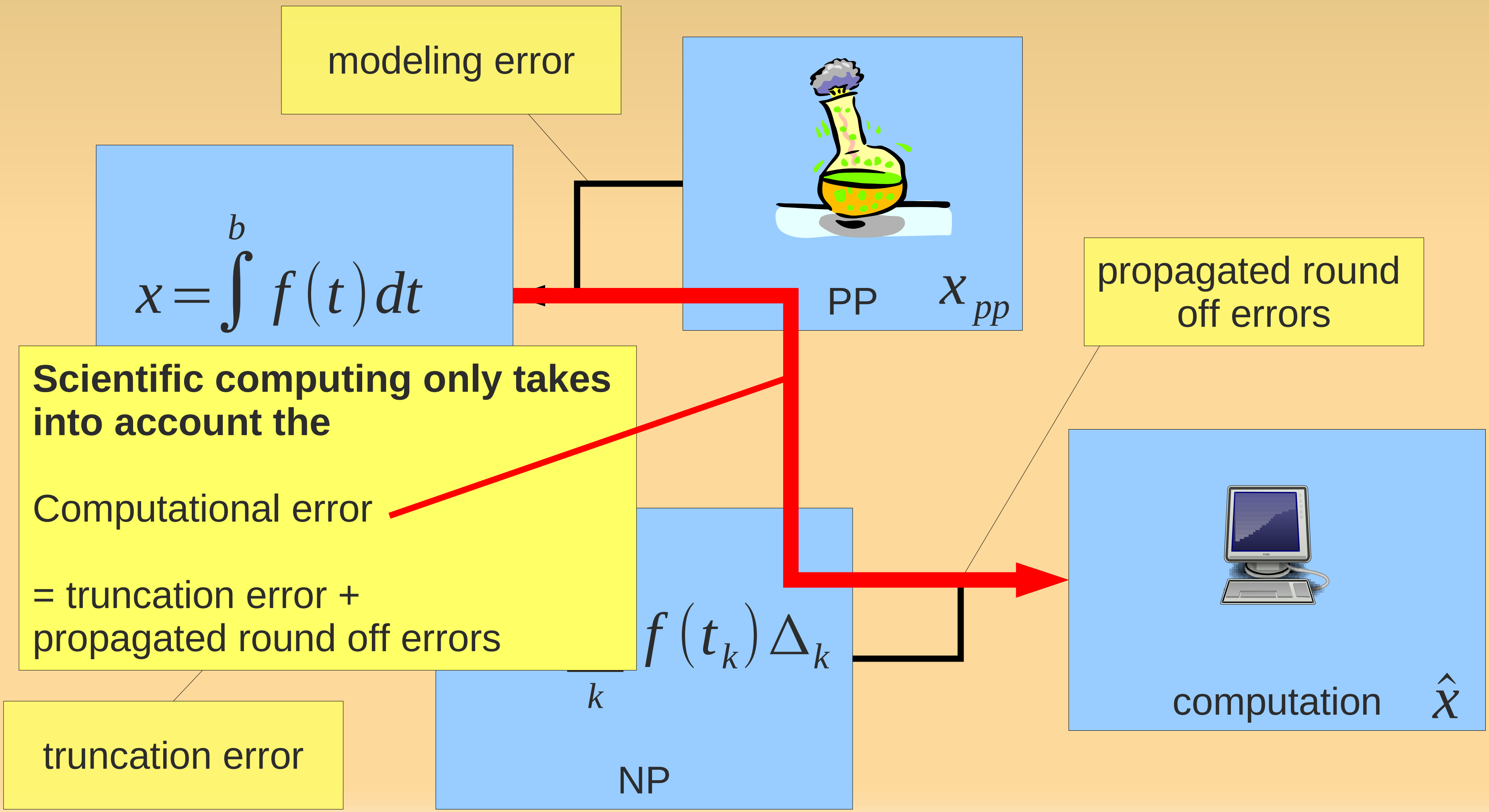
More on Errors

- Round off errors are only part of the story



More on Errors

- Round off errors are only part of the story



Convergence of Numerical Methods

- Discretization parameter h
  - e.g. `bin size' Δ_k

$$x_n = \sum_k f(t_k)\Delta_k$$

- A method is **convergent** IFF

$$h \to 0 \quad \Rightarrow \quad e_c \to 0$$

- **Order** of convergence $e_c < C \cdot h^p$
  - how fast the error reduces (when *h* decreases)

Iterative order

- **Iterative order** of convergence
  - says something about iterative methods
  - E.g., we said Newton's method is “fast”
- iterative order is p:

$$|x^{(n+1)} - x^*| \leq |x^{(n)} - x^*|^p$$

$$|e^{(n+1)}| \leq |e^{(n)}|^p$$

  - Newton is order 2
- In QSG: $|e^{(n)}| \leq \rho^{n^p} e^0$
  - basically unrolling the recursive equation above

Computational Cost

- We discussed of how fast we approach an answer
  - per iteration.
- Did not mention the cost of an iteration.

- **Computational complexity** gives a assessment of the complexity of an algorithm.
  - as a function of the size of the input.

Complexity of Matrix Multiplication

- As an example consider matrix multiplication

$$C = AB$$

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product … ?

Complexity of Matrix Multiplication

- As an example consider matrix multiplication

$$C = AB$$

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & n \times n & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij} = r_i^A c_j^B$

Complexity of Matrix Multiplication

- As an example consider matrix mu

$$C = AB$$

$$\begin{bmatrix} \dots & \dots & \dots \\ \dots & n \times n & \dots \\ \dots & \dots & \dots \end{bmatrix} = \begin{bmatrix} \dots & \dots & \dots \\ \dots & n \times n & \dots \\ \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dots \\ \dots \\ \dots \end{bmatrix}$$

Inner product of 2 *n*-vectors:

- n multiplications
- (n-1) additions

→ 2n-1 operations

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij} = r_i^A c_j^B$

Complexity of Matrix Multiplication

Often we are not interested in the exact number of computations.

→ “**Big-oh**” **notation**

“*f* has order of at most *g*”: $f(n)=O(g(n))$

IF
Exist a positive constant *c*, such that for sufficiently large *n*

$$f(n)\leq c\cdot|g(n)|$$

Inner product of 2 *n*-vectors:

- n multiplications
- (n-1) additions

→ 2n-1 operations

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij}=r_i^A c_j^B$

Complexity of Matrix Multiplication

Often we are not interested in the exact number of computations.

→ "**Big-oh**" **notation**

"*f* has order of at most *g*": $f(n)=O(g(n))$

IF
Exist a positive constant *c*, such that for sufficiently large *n*

$$f(n) \leq c \cdot |g(n)|$$

Inner product of 2 *n*-vectors:

- n multiplications
- (n-1) additions

→ 2n-1 operations

$$2n-1=O(n)$$

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij}=r_i^A c_j^B$

Complexity of Matrix Multiplication

Often we are not interested in the exact number of computations.

→ “**Big-oh**” **notation**

“*f* has order of at most *g*”: $f(n) = O(g(n))$

IF
Exist a positive constant *c*, such that for sufficiently large *n*

$$f(n) \leq c \cdot |g(n)|$$

Inner product of 2 *n*-vectors:

- n multiplications
- (n-1) additions

→ 2n-1 operations

$$2n - 1 = O(n)$$

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij} = r_i^A c_j^B$

Complexity of simplest algorithm?

Complexity of Matrix Multiplication

Often we are not interested in the exact number of computations.

→ “**Big-oh**” **notation**

“*f* has order of at most *g*”: $f(n)=O(g(n))$

IF
Exist a positive constant *c*, such that for sufficiently large *n*

$$f(n)\leq c\cdot|g(n)|$$

Inner product of 2 *n*-vectors:

- n multiplications
- (n-1) additions

→ 2n-1 operations

$$2n-1=O(n)$$

- Simplest algorithm:
  - for each of the n^2 entries c_{ij}
  - compute the inner product $c_{ij}=r_i^A c_j^B$

Complexity of simplest algorithm?

$$O(n^3)$$

Practical Time Measuring

- Theoretic analysis is useful to predict run-time.
- But in order to figure out where in a complex program the time is spend

  → measuring usually more informative

- 'cputime'

```
octave:> [TOTAL, USER, SYSTEM] = cputime ()
TOTAL =  0.44003
USER =  0.34802
SYSTEM =  0.092005
octave:> inv(rand(50));
octave:> [TOTAL2, USER2, SYSTEM2] = cputime ()
TOTAL2 =  0.50003
USER2 =  0.38402
SYSTEM2 =  0.11601
octave:> USER2 - USER
ans =  0.036003
```

Solving Linear Systems & LU factorization

Easy cases: Diagonal Matrices

- In case of a diagonal matrix A, the system is easy!

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Easy cases: Diagonal Matrices

- In case of a diagonal matrix A, the system is easy!

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$x_1 = c_1 / a_{11}$$
$$x_2 = c_2 / a_{22}$$
$$x_3 = c_3 / a_{33}$$

Easy cases: Diagonal Matrices

- In case of a diagonal matrix A, the system is easy!

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\begin{aligned} x_1 &= c_1 / a_{11} \\ x_2 &= c_2 / a_{22} \\ x_3 &= c_3 / a_{33} \end{aligned}$$

$$A^{-1} = \begin{bmatrix} 1/a_{11} & 0 & 0 \\ 0 & 1/a_{22} & 0 \\ 0 & 0 & 1/a_{33} \end{bmatrix}$$

Easy cases: Triangular Matrices

- Triangular systems are also is easy

$$\begin{bmatrix} 3 & 0 & 0 \\ 6 & 4 & 0 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ -5 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 4 & 0 \end{bmatrix} \begin{bmatrix} 5.5 \\ x_2 \\ x_3 \end{bmatrix} = 12$$

$$x_1 = 5.5$$

$$33 + 4\mathrm{x}_2 = 12$$

$$x_2 = (12 - 33)/4 = 3.75$$

Easy cases: Triangular Matrices

- Triangular systems are also is easy

$$\begin{bmatrix} 3 & 0 & 0 \\ 6 & 4 & 0 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ -5 \end{bmatrix}$$

Book (5.9) expresses this in 1 line:

$$x_2 = \frac{1}{4}\left(12 - \left(6 * 5.5\right)\right)$$

$$x_1 = 5.5$$

$$\begin{bmatrix} 6 & 4 & 0 \end{bmatrix} \begin{bmatrix} 5.5 \\ x_2 \\ x_3 \end{bmatrix} = 12$$

$$33 + 4\mathrm{x}_2 = 12$$

$$x_2 = \left(12 - 33\right)/4 = 3.75$$

Easy cases: Triangular Matrices

- Triangular systems are also is easy

$$\begin{bmatrix} 3 & 0 & 0 \\ 6 & 4 & 0 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ -5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} 5.5 \\ 3.75 \\ x_3 \end{bmatrix} = -5$$

$$x_1 = 5.5$$
$$x_2 = 3.75$$

$$26 + 5\mathrm{x}_3 = -5$$
$$x_3 = (-5 - 26)/5 = -6.2$$

Easy cases: Triangular Matrices

- Triangular systems are also is easy

$$\begin{bmatrix} 3 & 0 & 0 \\ 6 & 4 & 0 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ -5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} 5.5 \\ 3.75 \\ x_3 \end{bmatrix} = -5$$

$$26 + 5x_3 = -5$$

$$x_3 = (-5 - 26)/5 = -6.2$$

$$x_1 = 5.5$$
$$x_2 = 3.75$$
$$x_3 = 6.2$$

called
'forward substitution'

Easy cases: Triangular Matrices

- Upper triangular matrices work the same.
  - but start at the bottom
  - 'backward substitution'

- Now basic idea: use these simple case to solve general linear systems!

- LU factorization:
  - first decompose a matrix A in L, U
  - then use that to solve the original system

(L)ower and **(U)**pper diagonal

LU factorization

- We want to solve $A\,x = b$
- If $A = LU$
  we get...

find x

LU factorization

- We want to solve $A x = b$
- If $A = LU$
  we get...

$$A x = b$$
$$L U x = b$$
$$L (U x) = b$$

LU factorization

- We want to solve $Ax = b$
- If $A = LU$
  we get...

$$Ax = b$$
$$LUx = b$$
$$L(Ux) = b$$

define:
$$Ux \equiv y$$

$$\longrightarrow$$

$$Ly = b$$

LU factorization

- We want to solve $Ax=b$
- If $A=LU$
  we get...

$$Ax=b$$
$$LUx=b$$
$$L(Ux)=b$$

define:

$$Ux\equiv y$$

$$Ly=b$$

solve

$$y$$

LU factorization

- We want to solve $Ax = b$
- If $A = LU$ we get...

$$Ax = b$$
$$LUx = b$$
$$L(Ux) = b$$

define: $Ux \equiv y$ → $Ly = b$

solve

y → $Ux = y$

LU factorization

- We want to solve $Ax = b$
- If $A = LU$ we get...

$$Ax = b$$
$$LUx = b$$
$$L(Ux) = b$$

define: $Ux \equiv y$ → $Ly = b$

solve → y → $Ux = y$ solve → x

LU factorization

- How to compute L,U?

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

- "Gauss factorization"
  - many ways to chose L, U... → arbitrary assignment

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

  - Now solve the resulting systems of equations
    → $u_{11}=a_{11}$

    → $u_{12}=a_{12}$, etc.
  - see QSG.

Homework Reading

- Recap:
  - CH1: 1.2, 1.5.2, 1.6.
  - LU factorization p. 129-142
    - don't worry if you don't get all the examples

- Preparation for next time:
  - CH3: p. 75--81, 93--103 (sec. 3.5 is optional)