

PHSA-CC-1-1-P : Mathematical Physics I

Instructor : Amit Kumar Bhattacharjee (AKB)

Course Webpage : https://amitbny.github.io/akb.github.io/sem1H_numerlab.html

Course timeline : Jul–Nov, 2019

Evaluation : Assignments/Classtest followed by Semester examination

Ebook resources : National digital library: https://ndl.iitkgp.ac.in
http://nlist.inflibnet.ac.in

AKB

Course Marks : TBD; Credits – 2

- Introduction and overview ➔ Computer architecture and organization, memory and Input/output devices.
- Basics of scientific computing ➔ Binary and decimal arithmetic, Floating point numbers, algorithms, Sequence, Selection and Repetition, single and double precision arithmetic, underflow & overflow - importance of making equations in terms of dimensionless variables, Iterative methods.

AKB

History of Computer ➡ Invented by Dr. C. Babbage, a Mathematics Professor in 19th century.

1st Generation (1937 – 1946) ➡ Dr. J.V. Atanasoff & C. Berry – "Atanasoff Berry Computer" (ABC) → Collosus-1943 (first computer for millitary) → Electronic Numerical Integrator & Computer (ENIAC)-1946 (28k Kg weight, 18k vaccum tubes, single task machine without OS).

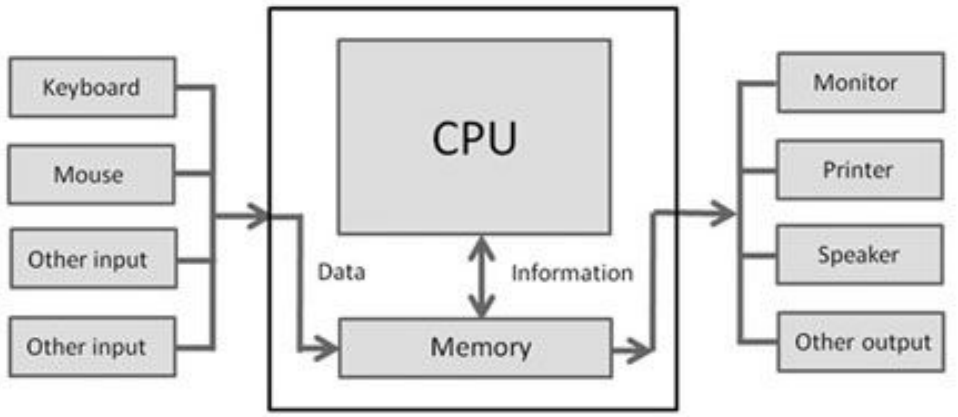
2nd Generation (1947 – 1962) ➡ Universal Automatic Computer (UNIVAC1)-1951 for public using Registers → International Business Machine (IBM) 650/700 (memory & OS).

3rd Generation (1963 -) ➡ MicroSoft Disk Operating System (MS-DOS)-1981, IBM introduced Personal Computer (PC) using Inegrated Circuit (IC) → Apple Introduced Macintosh → Windows in 1990.

AKB

What is a Computer?

A computer is a device that can be instructed to carry out sequences of arithmetic or logical set of operations automatically via computer programming.



AKB

What is a Computer?

A computer is a device that can be instructed to carry out sequences of arithmetic or logical set of operations automatically via computer programming.

AKB

What is a Computer?

CPU consists of Arithmetic Logic Unit (ALU), Memory, Input/Output (I/O)

HDD

RAM

Graphics Card

& other components, like motherboard, heat-sink, power supply, Fan etc.

HDD (data-storage) is the secondary memory while RAM (volatile memory) is the primary memory. Graphics card is necessary for data-heavy applications.

AKB

- ALU performs 2 class of operations (arithmetic & logic), e.g. +, -, *, /, sin(), sqrt() etc. Some machines can operate only on whole numbers (integers), while others use floating point (real) numbers, but with limited precision. Also Boolean logic operations (AND, OR, NOT, XOR) are performed in ALU.

ALU, Memory, I/O

- ALU performs 2 class of operations (arithmetic & logic), e.g. +, -, *, /, sin(), sqrt() etc. Some machines can operate only on whole numbers (integers), while others use floating point (real) numbers, but with limited precision. Also Boolean logic operations (AND, OR, NOT, XOR) are performed in ALU.

- Memory cell can store binary numbers in groups of 8 bits (or 1 byte). Each byte represents 256 different numbers ($2^8 = 256$): $\mathbb{R} \in [0,255], [-128,127]$. CPU contains memory cells (Registers) which are read/written more rapidly than RAM.

ALU, Memory, I/O

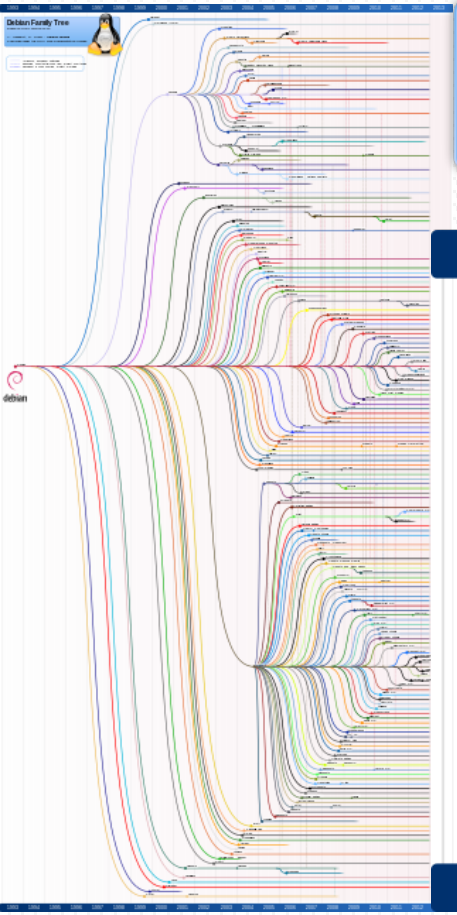
- ALU performs 2 class of operations (arithmetic & logic), e.g. +, -, *, /, sin(), sqrt() etc. Some machines can operate only on whole numbers (integers), while others use floating point (real) numbers, but with limited precision. Also Boolean logic operations (AND, OR, NOT, XOR) are performed in ALU.

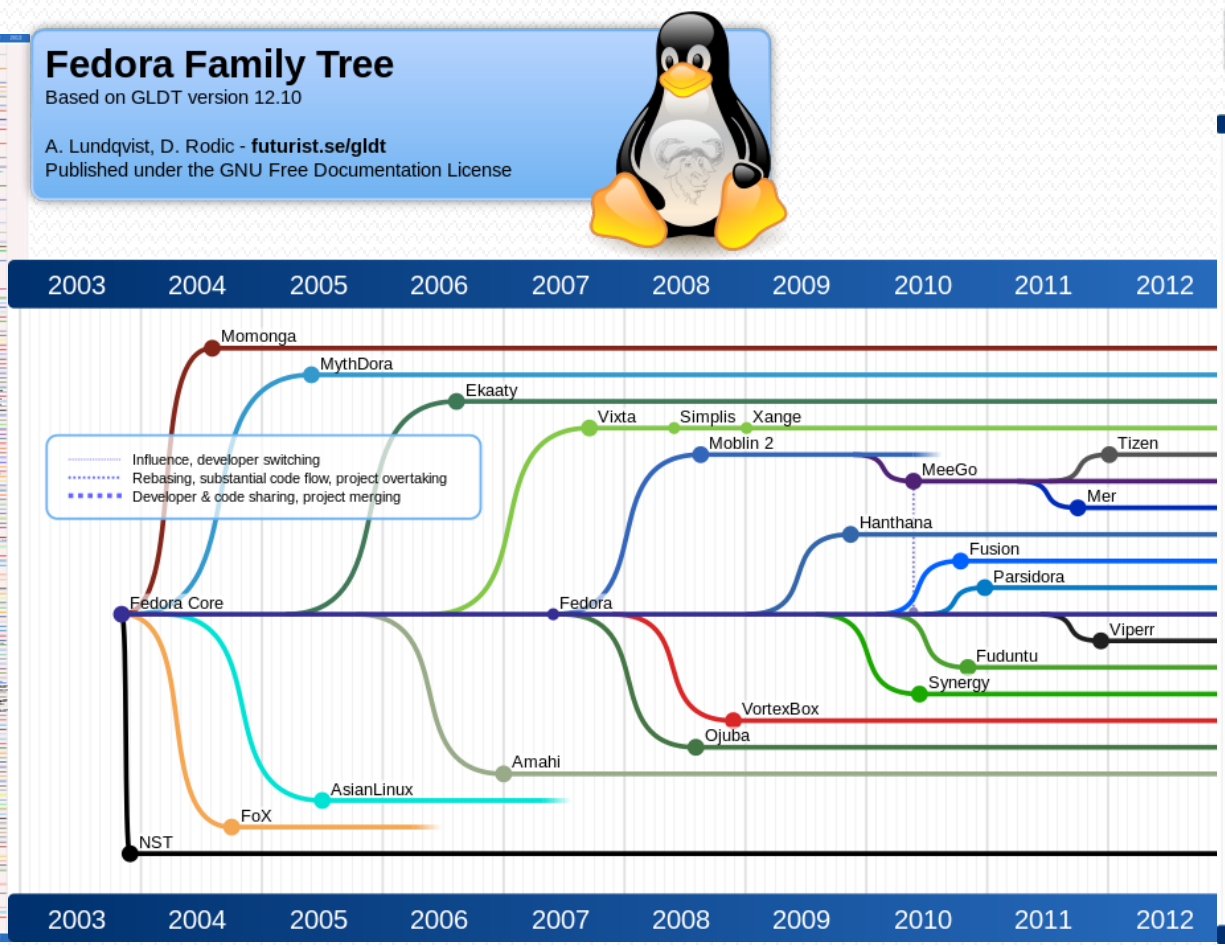
- Memory cell can store binary numbers in groups of 8 bits (or 1 byte). Each byte represents 256 different numbers ($2^8 = 256$): $\mathbb{R} \in [0,255], [-128,127]$. CPU contains memory cells (Registers) which are read/written more rapidly than RAM.

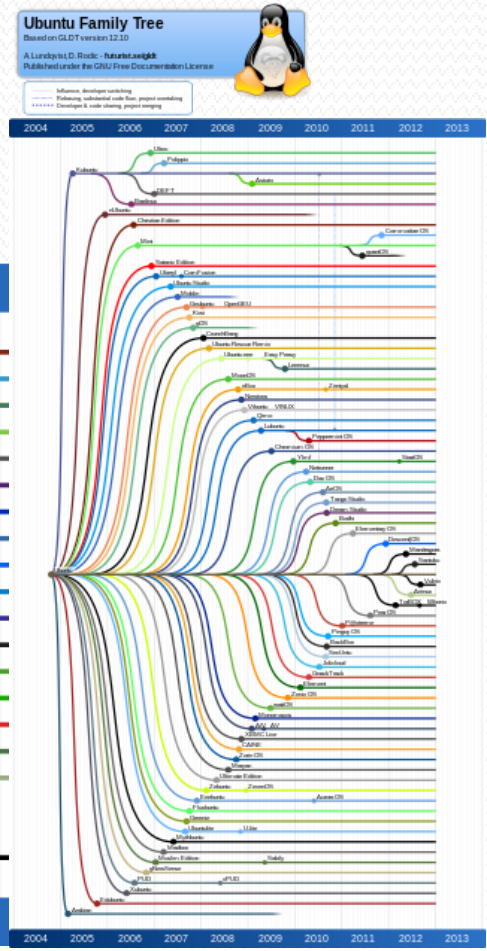
- I/O is the way a CPU exchanges information with the outside world, through Peripherals e.g. keyboard, mouse etc (input devices) & display, printer etc (output devices). HDD, optical disk drives, computer networking serve as both input and output devices.

What is a Computer?

Computer Programs, libraries, Operating Systems (OS) etc. OS has many Variant : (i) Unix distro (Solaris Sun OS), IRIX etc,
(ii) GNU/Linux [CentOS, Fedora (Redhat), SUSE, Ubuntu/Mint]







What is a Computer?

Computer Programs, libraries, Operating Systems (OS) etc. OS has many Variant : (i) Unix distro (Solaris Sun OS), IRIX etc,
(ii) GNU/Linux [CentOS, Fedora (Redhat), SUSE, Ubuntu/Mint]

Library : (i) Multimedia [DirectX, OpenGL, OpenAL, Vulkan (API)]
(ii) Programming Library (GSL, NRCP etc)

Data : (i) Protocol (TCP/IP, FTP, HTTP, SMTP etc),
(ii) File format (HTML, XML, JPEG, MPEG, PNG etc)

User Interface : GUI

Application Software : Office-suite, Graphics, Audio, Games, Software Engineering (Compiler, Assembler, Interpreter, Debugger, Text editor etc).

What is a Computer?

Programming Languages : (i) Low-level (e.g. Assembly language),
(ii) High-level (e.g. Basic, C/C++, Fortran 90/95 Java, Pascal),
(iii) Scripting (Python, Ruby, Perl).

Mathematical Softwares :
(i) Coding : LAPACK, LINPACK.

(ii) Coding/Visualization/Post-processing : Mathematica, Matlab/Octave, Maple.

(iii) Visualization : OpendX, Ovito, Paraview, VisIt, PyMol.

(iv) Supercomputing : LAMMPS, BoxLib, PETSc, Sundials.

Scientific Computing

17 Equations That Changed the World
by Ian Stewart

1.	**Pythagoras's Theorem**	$a^2 + b^2 = c^2$	Pythagoras,530 BC
2.	**Logarithms**	$\log xy = \log x + \log y$	John Napier, 1610
3.	**Calculus**	$\frac{\mathrm{d}f}{\mathrm{d}t} = \lim_{h \to 0} = \frac{f(t+h) - f(t)}{h}$	Newton, 1668
4.	**Law of Gravity**	$F = G\frac{m_1 m_2}{r^2}$	Newton, 1687
5.	**The Square Root of Minus One**	$i^2 = -1$	Euler, 1750
6.	**Euler's Formula for Polyhedra**	$V - E + F = 2$	Euler, 1751
7.	**Normal Distribution**	$\Phi(x) = \frac{1}{\sqrt{2\pi}\rho} e^{\frac{(x-\mu)^2}{2\rho^2}}$	C.F. Gauss, 1810
8.	**Wave Equation**	$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$	J. d'Almbert, 1746
9.	**Fourier Transform**	$f(\omega) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \omega} \mathrm{d}x$	J. Fourier, 1822
10.	**Navier-Stokes Equation**	$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$	C. Navier, G. Stokes, 1845
11.	**Maxwell's Equations**	$\nabla \cdot \mathbf{E} = 0$ $\quad \nabla \cdot \mathbf{H} = 0$ $\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{H}}{\partial t}$ $\quad \nabla \times \mathbf{H} = \frac{1}{c}\frac{\partial E}{\partial t}$	J.C. Maxwell, 1865
12.	**Second Law of Thermodynamics**	$\mathrm{d}S \geq 0$	L. Boltzmann, 1874
13.	**Relativity**	$E = mc^2$	Einstein, 1905
14.	**Schrodinger's Equation**	$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi$	E. Schrodinger, 1927
15.	**Information Theory**	$H = -\sum p(x) \log p(x)$	C. Shannon, 1949
16.	**Chaos Theory**	$x_{t+1} = k x_t (1 - x_t)$	Robert May, 1975
17.	**Black-Scholes Equation**	$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} - rV = 0$	F. Black, M. Scholes, 1990

= ?

AKB

Scientific Computing

- Domain beyond analytical approach (due to non-linearity, integrability, non-inversion and many other reasons): Numerical Mathematics/Applied Mathematics/Computational Science. Applications include, Computational Finance, Computational Biology, Computational Engineering, Computational Physics, Computational Chemistry, Computational Materials Science & so on.

Scientific Computing

- Domain beyond analytical approach (due to non-linearity, integrability, non-inversion and many other reasons): Numerical Mathematics/Applied Mathematics/Computational Science. Applications include, Computational Finance, Computational Biology, Computational Engineering, Computational Physics, Computational Chemistry, Computational Materials Science & so on.

- A well-executed computation can reproduce lab-based experiments quantitatively, and therefore can predict new phenomena by “numerical experiments” often hard to realize on a lab due to financial / timeframe / workforce restrictions.

AKB

Scientific Computing

- Domain beyond analytical approach (due to non-linearity, integrability, non-inversion and many other reasons): Numerical Mathematics/Applied Mathematics/Computational Science. Applications include, Computational Finance, Computational Biology, Computational Engineering, Computational Physics, Computational Chemistry, Computational Materials Science & so on.

- A well-executed computation can reproduce lab-based experiments quantitatively, and therefore can predict new phenomena by “numerical experiments” often hard to realize on a lab due to financial / timeframe / workforce restrictions.

- There goes the catch! Given a computer, is every computation is a well-executed computation? Answer is NO.

AKB

Binary and Decimal

- Decimal (or denary) numeral system represents integer and non-integer numbers in base-10 positional number system. Decimal refers to digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in the decimal system containing a “decimal separator”, e.g. 3.14. In general,

$$a_m a_{m-1} \ldots a_0 . b_1 b_2 \ldots b_n = a_m 10^m + a_{m-1} 10^{m-1} + \ldots + a_0 10^0 + \frac{b_1}{10^1} + \frac{b_2}{10^2} + \ldots + \frac{b_n}{10^n}.$$

Binary and Decimal

- Decimal (or denary) numeral system represents integer and non-integer numbers in base-10 positional number system. Decimal refers to digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in the decimal system containing a “decimal separator”, e.g. 3.14. In general,

$$a_m a_{m-1} \ldots a_0 . b_1 b_2 \ldots b_n = a_m 10^m + a_{m-1} 10^{m-1} + \ldots + a_0 10^0 + \frac{b_1}{10^1} + \frac{b_2}{10^2} + \ldots + \frac{b_n}{10^n}.$$

- Binary numeral system represents only two numbers 0 and 1 in base-2 number system. A human-understood decimal is converted to computer-understood binary to perform computation and back converted to decimal to decipher.

Binary and Decimal

- Decimal (or denary) numeral system represents integer and non-integer numbers in base-10 positional number system. Decimal refers to digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in the decimal system containing a “decimal separator”, e.g. 3.14. In general,

$$a_m a_{m-1} \ldots a_0 . b_1 b_2 \ldots b_n = a_m 10^m + a_{m-1} 10^{m-1} + \ldots + a_0 10^0 + \frac{b_1}{10^1} + \frac{b_2}{10^2} + \ldots + \frac{b_n}{10^n}.$$

- Binary numeral system represents only two numbers 0 and 1 in base-2 number system. A human-understood decimal is converted to computer-understood binary to perform computation and back converted to decipher.
- Conversion binary ➔ decimal: $a_m a_{m-1} \ldots a_1 = a_m 2^{m-1} + a_{m-1} 2^{m-2} + \ldots + a_1 2^0$.

Most Significant Bit (MSB)

Least Significant Bit (LSB)

$$101100101_2 = 1*2^8 + 0*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 357_{10}.$$

- Conversion decimal ➔ binary: Repeated division-by-2 method:

294_{10} : divide by 2 ➔ 147 {remainder 0 (*LSB*)}, divide by 2 ➔ 73 (remainder 1),
divide by 2 ➔ 36 (remainder 1), divide by 2 ➔ 18 (remainder 0),
divide by 2 ➔ 9 (remainder 0), divide by 2 ➔ 4 (remainder 1),
divide by 2 ➔ 2 (remainder 0), divide by 2 ➔ 1 (remainder 0),
divide by 2 ➔ 0 {remainder 1 (*MSB*)} ➔ 100100110_2.

- Conversion decimal ➔ binary: Repeated division-by-2 method:

294_{10} : **divide by 2 ➔ 147 {remainder 0 (*LSB*)}, divide by 2 ➔ 73 (remainder 1),**
divide by 2 ➔ 36 (remainder 1), divide by 2 ➔ 18 (remainder 0),
divide by 2 ➔ 9 (remainder 0), divide by 2 ➔ 4 (remainder 1),
divide by 2 ➔ 2 (remainder 0), divide by 2 ➔ 1 (remainder 0),
divide by 2 ➔ 0 {remainder 1 (*MSB*)} ➔ 100100110_2.

- Fractions in binary terminate, if the denominator has 2 as the only prime factor. 1/10 doesn't have a finite binary representation which causes 10×0.1 not to be precisely equal to 1 in floating point arithmetic. To interpret the binary expression for $\frac{1}{3} = .010101\ldots$ means $= 0*2^{-1}+1*2^{-2}+0*2^{-3}+1*2^{-4}+\ldots = 0.3125+\ldots$. So 1 and 0's alternate forever, if we want to reach the exact expression as a sum of inverse powers of 2 ➔ *source of Error* !!

Scientific Computing

The Patriot Missile Failure ➡ On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Saudi Arabia failed to track & intercept an incoming Iraqi Scud missile. It killed 28 soldiers & injured 100s of people. Cause of the failure turned out to be inaccurate calculation of the time due to arithmetic errors!! How????

Scientific Computing

The Patriot Missile Failure ➡ On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Saudi Arabia failed to track & intercept an incoming Iraqi Scud missile. It killed 28 soldiers & injured 100s of people. Cause of the failure turned out to be inaccurate calculation of the time due to arithmetic errors!! How????

- Time in tenths of second (measured by system's internal clock) was multiplied by 0.1 to produce the time in seconds, using a 24-bit Register. Specifically, value of 1/10 (having non-terminating binary expansion) was truncated at 24-bits. This small chopping error when multiplied by large number led to significant error. $\frac{1}{10} = \frac{1}{2^4}+\frac{1}{2^5}+\frac{1}{2^8}+\frac{1}{2^9}+\frac{1}{2^{12}}+\frac{1}{2^{13}} \rightarrow$ binary expansion $\rightarrow$

$$0.0001100110011001100110011001100$$

AKB

Scientific Computing

The Patriot Missile Failure ➡ On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Saudi Arabia failed to track & intercept an incoming Iraqi Scud missile. It killed 28 soldiers & injured 100s of people. Cause of the failure turned out to be inaccurate calculation of the time due to arithmetic errors!! How????

- Time in tenths of second (measured by system's internal clock) was multiplied by 0.1 to produce the time in seconds, using a 24-bit Register. Specifically, value of 1/10 (having non-terminating binary expansion) was truncated at 24-bits. This small chopping error when multiplied by large number led to significant error. $\frac{1}{10} = \frac{1}{2^4}+\frac{1}{2^5}+\frac{1}{2^8}+\frac{1}{2^9}+\frac{1}{2^{12}}+\frac{1}{2^{13}} \rightarrow$ binary expansion $\rightarrow$ $0.0001100110011001100110011001100$

  What patriot saved in the system ➡ $0.00011001100110011001100$

  error in binary 0.00000000000000000000011001100 lead to 0.000000095 in decimal. Multiplying by the number of tenths of a second in 100 hours gives 0.000000095×100×60×60×10=0.34 seconds.

AKB

Floating Point Number System

Significant Digits These are the first nonzero digit & all succeeding digits, *e.g.*

1.7320 has 5 significant digits, while 0.0491 has only 3.

A floating point (real) number system have elements of the form $y = \pm m \times \beta^{e-t}$, is characterized with 4 integer parameters:

- Base (or radix) β, precision t, exponent e & significand (or mantissa) m.

Here $e_{min} \le e \le e_{max}$ & $0 \le m \le \beta^t - 1$. This gives the range of nonzero floating point numbers $\beta^{e_{min}-1} \le y \le \beta^{e_{max}}\left(1-\beta^{-t}\right)$.

AKB

Floating Point Number System

Significant Digits ➔ These are the first nonzero digit & all succeeding digits, *e.g.* 1.7320 has 5 significant digits, while 0.0491 has only 3.

A floating point (real) number system have elements of the form $y = \pm m \times \beta^{e-t}$, is characterized with 4 integer parameters:

- Base (or radix) β, precision t, exponent e & significand (or mantissa) m. Here $e_{min} \le e \le e_{max}$ & $0 \le m \le \beta^t - 1$. This gives the range of nonzero floating point numbers $\beta^{e_{min}-1} \le y \le \beta^{e_{max}}\left(1-\beta^{-t}\right)$.
- Floating point numbers aren't equally spaced !! If $\beta = 2, t = 3, e_{min} = -1, e_{max} = 3$ then non-negative numbers are $0, 0.25, 0.3125, 0.3750, 0.4375, 0.5, 0.625, 0.750, 0.875,$ $1.0, 1.25, 1.50, 1.75, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0$.

AKB

Floating Point Number System

Significant Digits These are the first nonzero digit & all succeeding digits, *e.g.*

1.7320 has 5 significant digits, while 0.0491 has only 3.

A floating point (real) number system have elements of the form $y = \pm m \times \beta^{e-t}$,

is characterized with 4 integer parameters:

- Base (or radix) β, precision t, exponent e & significand (or mantissa) m.

```
amitb@amit-softmat:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Spacing of the floating point numbers jumps by a factor 2 at each power of 2. Spacing can be characterized in terms of machine epsilon, which is the distance from 1.0 to the next larger floating point number. In Python, this can be seen as :

"import numpy as np", and then, "np.spacing(1)" yields 2.220446049250313e-16.

AKB

Floating Point Number System

In MATLAB/Octave, “eps” gives the same value. “realmax” & “realmin” represent the largest positive & smallest positive normalized floating point number. In Python, “import numpy as np”, and then, “np.finfo(np.double).max” yields 1.7976931348623157e+308, while “np.finfo(np.double).tiny” yields 2.2250738585072014e-308.

Floating Point Number System

In MATLAB/Octave, “eps” gives the same value. “realmax” & “realmin” represent the largest positive & smallest positive normalized floating point number. In Python, “import numpy as np”, and then, “np.finfo(np.double).max” yields 1.7976931348623157e+308, while “np.finfo(np.double).tiny” yields 2.2250738585072014e-308.

IEEE Arithmetic IEEE standard defines a binary floating point system. The standard specifies floating point number formats, results of the basic floating point operations & comparisons, rounding modes, floating point exceptions & handling, conversion between different arithmetic formats.

Two main floating point formats are defined:

Type	Size	Significand	Exponent	Unit roundoff	Range
(precision) Single	32 bits	23+1 bits	8 bits	$2^{-24} \approx 5.96 \times 10^{-8}$	$10^{\pm 38}$
Double	64 bits	52+1 bits	11 bits	$2^{-53} \approx 1.11 \times 10^{-16}$	$10^{\pm 308}$

AKB

Floating Point Number System

In both formats one bit is reserved as a sign bit. The most significant bit is always 1 & not stored. This hidden bit accounts for the "+1" in the table.

Floating Point Number System

In both formats one bit is reserved as a sign bit. The most significant bit is always 1 & not stored. This hidden bit accounts for the "+1" in the table.

- NaN (Not a number) is a special bit pattern with arbitrary significand. It's generated by operations such as $0/0, 0\times\infty, \infty/\infty, (+\infty)+(-\infty)$. Infinity symbol is represented by zero significand & same exponent field as NaN, sign bit distinguishes between $\pm\infty$ with property, $\infty+\infty=\infty, (-1)\times\infty=-\infty, \text{finite}/\infty=0$. Zero is represented by a zero exponent field & zero significand, with $+0=-0$.
- In MATLAB/Fortran 90/95, A(p:q, r:s) denotes submatrix of A formed of rows p to q & columns r to s. A(:, j) is the jth column of A, and A(i, :) the ith row of A.

AKB

Floating Point Number System

In both formats one bit is reserved as a sign bit. The most significant bit is always 1 & not stored. This hidden bit accounts for the "+1" in the table.

- NaN (Not a number) is a special bit pattern with arbitrary significand. It's generated by operations such as $0/0, 0\times\infty, \infty/\infty, (+\infty)+(-\infty)$. Infinity symbol is represented by zero significand & same exponent field as NaN, sign bit distinguishes between $\pm\infty$ with property, $\infty+\infty=\infty, (-1)\times\infty=-\infty, \text{finite}/\infty=0$. Zero is represented by a zero exponent field & zero significand, with $+0=-0$.
- In MATLAB/Fortran 90/95, A(p:q, r:s) denotes submatrix of A formed of rows p to q & columns r to s. A(:, j) is the jth column of A, and A(i, :) the ith row of A.
- Evaluation of an expression in floating point arithmetic denoted by $fl(.)$ is

$$fl(x\,op\,y)=(x\,op\,y)(1+\delta), \quad |\delta|\le u$$

u is called the ***unit roundoff*** (machine precision) $\approx 10^{-8}(\text{single}), 10^{-16}(\text{double}),$ $10^{-10}-10^{-12}(\text{pocket calculators})$.

- Computed quantities are denoted with ***hat***. So, $\hat{x}$ is the computed approximation of x.

AKB

Floating Point Number System

- $\lfloor x \rfloor$ (floor x) is the largest integer $\leq x$ & $\lceil x \rceil$ (ceil x) is the smallest integer $\geq x$.
  Check with Python: “import math; math.floor(1.9) = 1.0”, “math.ceil(1.9)=2.0”.
- Remember, we compute single precision arithmetic ($\mathrm{u} \approx 6 \times 10^{-8}$) by rounding, say, a double precision result with *unit roundoff* ($\mathrm{u} \approx 1.1 \times 10^{-16}$) to single precision as well rounding result of every elementary operation to single precision.

AKB

Floating Point Number System

- $\lfloor x \rfloor$ (floor x) is the largest integer $\leq x$ & $\lceil x \rceil$ (ceil x) is the smallest integer $\geq x$.
  Check with Python: “import math; math.floor(1.9) = 1.0”, “math.ceil(1.9)=2.0”.
- Remember, we compute single precision arithmetic ($u \approx 6 \times 10^{-8}$) by rounding, say, a double precision result with *unit roundoff* ($u \approx 1.1 \times 10^{-16}$) to single precision as well rounding result of every elementary operation to single precision.

Absolute & Relative Error ➡ If $\hat{x}$ is an approximation to real number x, then

$$E_{abs}(\hat{x}) = |x - \hat{x}|, \quad E_{rel}(\hat{x}) = \frac{|x - \hat{x}|}{|x|}$$

Note that relative error is scale independent: $x \rightarrow \alpha x$, $\hat{x} \rightarrow \alpha \hat{x}$, doesn't change $E_{rel}(\hat{x})$.

- Relative error is connected with the notion of *Correct significant digits*, however relative error is a more precise, base independent measure.
- Sources of Error ➡ (i) rounding, (ii) data uncertainty & (iii) truncation. Uncertainty in data can arise in several ways ➡ from errors of measurement, storing data on

AKB

computer. Data errors can be analysed using perturbation theory, while intermediate rounding errors require an analysis specific to the given method & thus harder to understand.

computer. Data errors can be analysed using perturbation theory, while intermediate rounding errors require an analysis specific to the given method & thus harder to understand.

- Truncation/discretization errors is when in Taylor's series to derive numerical methods, such as Trapezium rule for Quadrature, Euler's method for differential equations etc, finite terms are kept and later are omitted. This depends on choice of "h": $$f(x+h)=f(x)+hf'(x)+\frac{h^2}{2!}f''(x)+\frac{h^3}{3!}f'''(x)+O(h^4)$$
- "Rounding errors and instability are important & numerical analysts will always be the experts in these subjects & at pains to ensure that the unwary are not tripped up by them. But our central mission is to compute quantities that are typically uncomputable, from an analytic point of view, and to do it with lightning speed".

– Nick Trefethen, FRS, Univ. of Oxford.

Precision vs Accuracy ➡ Accuracy refers to the absolute/relative error of an approximate quantity. Precision is the accuracy with which the basic arithmetic operations (+, -, *, /) are performed & for floating point arithmetic is measured by the *unit roundoff* u. Accuracy & precision are the same for the scalar computation c = a x b, but accuracy can be much worse than precision in the solution of a linear system of equations, e.g. stiff equations.

Precision vs Accuracy Accuracy refers to the absolute/relative error of an approximate quantity. Precision is the accuracy with which the basic arithmetic operations (+, -, *, /) are performed & for floating point arithmetic is measured by the *unit roundoff* u. Accuracy & precision are the same for the scalar computation c = a x b, but accuracy can be much worse than precision in the solution of a linear system of equations, e.g. *stiff* equations.

Forward and Backward Errors Suppose that an approximation $\hat{y}$ of $y=f(x)$ is computed in an arithmetic of precision u with $E_{\text{rel}}(\hat{y})\approx u$. This doesn't mean we know for any $\triangle x$, $\hat{y}=f(x+\triangle x)$. The absolute and relative errors of $\hat{y}$ are called forward errors and $min|\triangle x|$ and $\frac{min|\triangle x|}{|x|}$ is called the backward error. Stability of numerical recipe lies on backward stable algorithm where rounding errors are most significant. Recipe for cosine functions do not satisfy $\hat{y}=f(x+\triangle x)$ but $\hat{y}+\triangle y=f(x+\triangle x)$, $\triangle y\leq\epsilon|y|$, $\triangle x\leq\eta|x|$, are called mixed forward-backward error result.

AKB

Conditioning Forward & Backward error is governed by sensitivity of solution to perturbations in the data or "conditioning" of the problem. Then,

$$\hat{y}-y=f(x+\triangle x)-f(x)=f'(x)\triangle x+\frac{(\triangle x)^2}{2}f''(x+\triangle x)+O((\triangle x)^3),$$

or $\frac{\hat{y}-y}{y}=\left(\frac{xf'(x)}{f(x)}\right)\frac{\triangle x}{x}+O((\triangle x)^2)$. Here $c(x)=\left|\frac{xf'(x)}{f(x)}\right|$ measures the relative change in output for relative change in input or condition number of f. For example, consider $f(x)=\log x$, $c(x)=|1/\log x|$ → ∞ for $x\approx 1$. So a small relative change in x can produce large relative change in log x for x~1.

AKB

Conditioning Forward & Backward error is governed by sensitivity of solution to perturbations in the data or "conditioning" of the problem. Then,

$$\hat{y}-y=f(x+\triangle x)-f(x)=f'(x)\triangle x+\frac{(\triangle x)^2}{2}f''(x+\triangle x)+O((\triangle x)^3),$$

or $\frac{\hat{y}-y}{y}=(\frac{xf'(x)}{f(x)})\frac{\triangle x}{x}+O((\triangle x)^2)$. Here $c(x)=\left|\frac{xf'(x)}{f(x)}\right|$ measures the relative change in output for relative change in input or condition number of f. For example, consider $f(x)=\log x$, $c(x)=|1/\log x|$ → ∞ for $x\approx 1$. So a small relative change in x can produce large relative change in log x for x~1.

Rule of thumb: forward error ≤ condition number × backward error. So, computed solution to an ill-conditioned problem can have a *large* forward error, even if computed solution has *small* backward error. Backward stability implies forward stability. Cramer's rule for solving 2x2 linear system is forward stable but not backward stable.

AKB

Cancellation Consider the function $f(x)=\dfrac{(1-\cos x)}{x^2}$ which for all $x\neq 0$ is $0\leq f(x)<1/2$. However, say, for $x=1.2\times 10^{-5}$, $\cos x=0.9999999999$ rounded to 10 significant digits, so as $1-\cos x=0.0000\,0000\,01$ and then, $\dfrac{(1-\cos x)}{x^2}=\dfrac{10^{-10}}{1.44\times 10^{-10}}$ = 0.6944..., which is wrong !!

```
>> x=eps:(pi/40):pi/2; f = (1- cos(x))./x.^2; [x(:) f(:)]
ans =

  2.22044604925031e-16                         0
     0.078539816339745     0.499743031894047
      0.15707963267949     0.498972761400788
     0.235619449019235     0.497691087900341
      0.31415926535898     0.495901170055451
     0.392699081698724     0.493607415383329
     0.471238898038469     0.490815465712315
     0.549778714378214     0.487532178579061
     0.628318530717959     0.483765604637539
     0.706858347057704     0.479524961166377
     0.785398163397449     0.474820601775892
     0.863937979737193     0.469663982430643
     0.942477796076938      0.46406762391727
      1.02101761241668     0.458045070900783
      1.09955742875643      0.45161084772531
      1.17809724509617     0.444780411127427
      1.25663706143592      0.43757010004169
      1.33517687777566     0.429997082688671
      1.41371669411541     0.422079301145707
      1.49225651045515     0.413835413609684
       1.5707963267949     0.405284734569351

>> x=1.2e-5; [cos(x) 1-cos(x) x^2]
ans =

       0.999999999928  7.19999615483857e-11                1.44e-10
```

AKB

Cancellation Consider the function $f(x)=\frac{(1-\cos x)}{x^2}$ which for all $x\neq 0$ is $0\leq f(x)<1/2$. However, say, for $x=1.2\times 10^{-5}$, $\cos x=0.9999999999$ rounded to 10 significant digits, so as $1-\cos x=0.0000\,0000\,01$ and then, $\frac{(1-\cos x)}{x^2}=\frac{10^{-10}}{1.44\times 10^{-10}}$ = 0.6944..., which is wrong !!

The problem lies in the fact that, even though 1-c is exact, it has only 1 significant figure, so subtraction produces a result of the same size as the error in c. However, if the subtraction is avoided by rewriting $\cos x=1-2\sin^2(x/2)$, $f(x)=\frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2$. The same procedure now yields f(x) = 0.5 correct to 10 significant digits.

AKB

Cancellation Consider the function $f(x)=\frac{(1-\cos x)}{x^2}$ which for all $x\neq 0$ is $0\leq f(x)<1/2$. However, say, for $x=1.2\times 10^{-5}, \cos x=0.9999999999$ rounded to 10 significant digits, so as $1-\cos x=0.0000\,0000\,01$ and then, $\frac{(1-\cos x)}{x^2}=\frac{10^{-10}}{1.44\times 10^{-10}}$ = 0.6944..., which is wrong !!

The problem lies in the fact that, even though 1-c is exact, it has only 1 significant figure, so subtraction produces a result of the same size as the error in c. However, if the subtraction is avoided by rewriting $\cos x=1-2\sin^2(x/2)$, $f(x)=\frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2$. The same procedure now yields f(x) = 0.5 correct to 10 significant digits.

- Error in cancellation can be avoided by *estimating the damage*, it can't be unavoidable. Or computing ratio of differences of the same order of error so that numerator & denominator cancels out. Or, for example computing x + (y – z) for $x\gg y\approx z>0$.

AKB

Roots of a Quadratic Equation Depending on the sign of the remainder b^2-4ac, for $a\neq 0, ax^2+bx+c=0$ have two roots (real-unequal, real-equal, imaginary) $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. If $b^2\gg 4ac$, then $x=\frac{-b\pm b}{2a}$ and for "+" sign it suffers massive cancellation that brings prominence of earlier rounding errors. To avoid, the largest (in absolute value) root is chosen $x_1=\frac{-b-\sqrt{b^2-4ac}}{2a}$ and the other from $x_1 x_2=\frac{c}{a}$. But when $b^2\approx 4ac$, accuracy is lost & only way to guarantee accuracy is to use extended precision.

AKB

Roots of a Quadratic Equation Depending on the sign of the remainder b^2-4ac, for $a\neq 0, ax^2+bx+c=0$ have two roots (real-unequal, real-equal, imaginary) $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. If $b^2\gg 4ac$, then $x=\frac{-b\pm b}{2a}$ and for "+" sign it suffers massive cancellation that brings prominence of earlier rounding errors. To avoid, the largest (in absolute value) root is chosen $x_1=\frac{-b-\sqrt{b^2-4ac}}{2a}$ and the other from $x_1 x_2=\frac{c}{a}$. But when $b^2\approx 4ac$, accuracy is lost & only way to guarantee accuracy is to use extended precision.

Overflow & Underflow If we apply $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ in single precision arithmetic to equation $10^{20}x^2-3\times 10^{20}x+2\times 10^{20}=0$, even when the roots are not harmful (x=1 & x=2), overflow occurs since the maximum floating point number is $\approx 10^{38}$.

AKB

Roots of a Quadratic Equation Depending on the sign of the remainder b^2-4ac, for $a\neq 0, ax^2+bx+c=0$ have two roots (real-unequal, real-equal, imaginary) $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. If $b^2\gg 4ac$, then $x=\frac{-b\pm b}{2a}$ and for "+" sign it suffers massive cancellation that brings prominence of earlier rounding errors. To avoid, the largest (in absolute value) root is chosen $x_1=\frac{-b-\sqrt{b^2-4ac}}{2a}$ and the other from $x_1 x_2=\frac{c}{a}$. But when $b^2\approx 4ac$, accuracy is lost & only way to guarantee accuracy is to use extended precision.

Overflow & Underflow If we apply $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ in single precision arithmetic to equation $10^{20}x^2-3\times 10^{20}x+2\times 10^{20}=0$, even when the roots are not harmful (x=1 & x=2), overflow occurs since the maximum floating point number is $\approx 10^{38}$. Analytically/numerically dividing by maximum (|a|, |b|, |c|) = 3×10^{20} is OK, but same strategy doesn't work for, say, $10^{-20}x^2-3x+2\times 10^{20}=0$ whose roots are 10^{20} & 2×10^{20}.

AKB

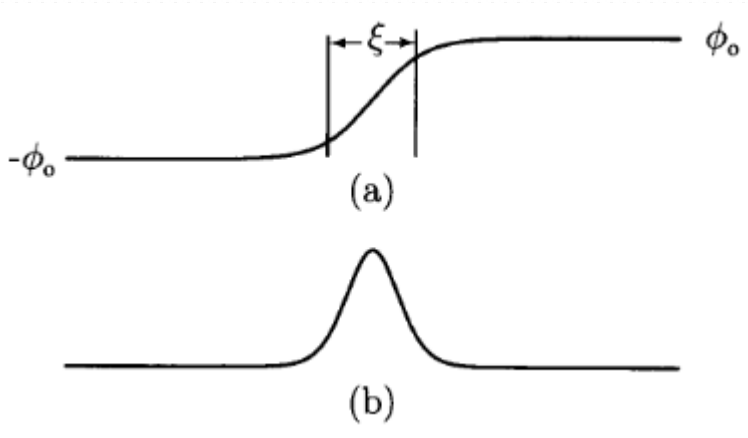
Roots of a Quadratic Equation Depending on the sign of the remainder b^2-4ac, for $a\neq 0, ax^2+bx+c=0$ have two roots (real-unequal, real-equal, imaginary) $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. If $b^2\gg 4ac$, then $x=\frac{-b\pm b}{2a}$ and for "+" sign it suffers massive cancellation that brings prominence of earlier rounding errors. To avoid, the largest (in absolute value) root is chosen $x_1=\frac{-b-\sqrt{b^2-4ac}}{2a}$ and the other from $x_1x_2=\frac{c}{a}$. But when $b^2\approx 4ac$, accuracy is lost & only way to guarantee accuracy is to use extended precision.

Overflow & Underflow If we apply $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ in single precision arithmetic to equation $10^{20}x^2-3\times 10^{20}x+2\times 10^{20}=0$, even when the roots are not harmful (x=1 & x=2), overflow occurs since the maximum floating point number is $\approx 10^{38}$. Analytically/numerically dividing by maximum (|a|, |b|, |c|) = 3×10^{20} is OK, but same strategy doesn't work for, say, $10^{-20}x^2-3x+2\times 10^{20}=0$ whose roots are 10^{20} & 2×10^{20}. Scaling the variable $x=10^{20}y$ yields, $10^{20}y^2-3\times 10^{20}y+2\times 10^{20}=0$ which is the initial equation we started from.

AKB

Need for non-dimensionalization

A well-known example in Condensed Matter Physics is the ϕ^4 kink, that gives a *tanh* solution of the domain wall formed between liquid-gas interface/magnetic domain walls having diverse consequences in many branches of physics.



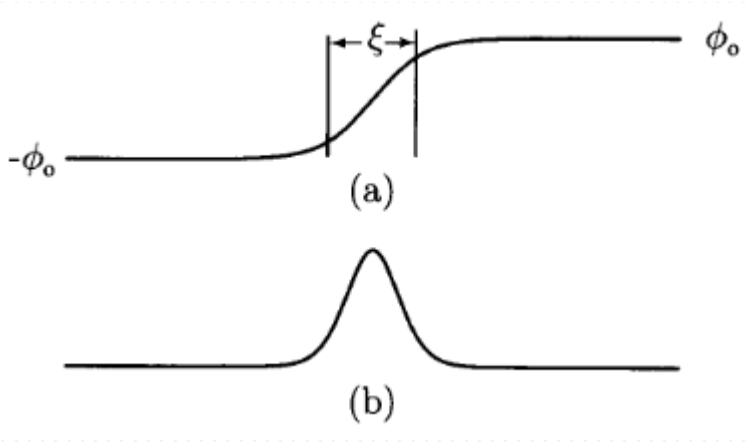
$$\phi = \pm\phi_0 \tanh\left[\frac{x - x_0}{\sqrt{2\,c/r}}\right] \qquad \phi = \text{density (say)}$$

$$\mathcal{F} = -\frac{1}{2} r \phi^2 + u\phi^4 + \frac{1}{2} c (\nabla \phi)^2$$

AKB

Need for non-dimensionalization

A well-known example in Condensed Matter Physics is the ϕ^4 kink, that gives a *tanh* solution of the domain wall formed between liquid-gas interface/magnetic domain walls having diverse consequences in many branches of physics.



$$\phi = \pm\phi_0 \tanh\left[\frac{x - x_0}{\sqrt{2\,c/r}}\right] \qquad \phi = \text{density (say)}$$

$$\mathcal{F} = -\frac{1}{2} r \phi^2 + u\phi^4 + \frac{1}{2} c (\nabla \phi)^2$$

- To have a control over the dynamics of density variation, clearly one needs to control these parameters, r, u, c or a multi-dimensional diagram which is nearly impossible to control often because of many parameters with less-known activity.

AKB

Need for non-dimensionalization

- Notice that $\mathcal{F} = -\frac{1}{2} r \phi^2 + u \phi^4$ have two minima, that we see from $\frac{\partial \mathcal{F}}{\partial \phi} = 0 = -r\phi + 4u\phi^3$ or $\phi = \pm \phi_0 = \sqrt{\frac{r}{4u}}$. Then $\mathcal{F}_0 = -\frac{1}{2} r \phi^2 + u \phi^4 = -\frac{r^2}{16u} = -\frac{r \phi_0^2}{4}$

Need for non-dimensionalization:

- Notice that $\mathcal{F} = -\frac{1}{2} r \phi^2 + u \phi^4$ have two minima, that we see from $\frac{\partial \mathcal{F}}{\partial \phi} = 0 = -r\phi + 4u\phi^3$ or $\phi = \pm \phi_0 = \sqrt{\frac{r}{4u}}$. Then $\mathcal{F}_0 = -\frac{1}{2} r \phi^2 + u \phi^4 = -\frac{r^2}{16u} = -\frac{r \phi_0^2}{4}$
- But now notice that, $\mathcal{F} = -\frac{1}{2} r \phi^2 + u \phi^4 = -\frac{1}{2} r \frac{\phi^2}{\phi_0^2} \phi_0^2 + u \frac{\phi^4}{\phi_0^4} \phi_0^4 = -\frac{r^2}{8u} \hat{\phi}^2 + \frac{r^2}{16u} \hat{\phi}^4$

$$= -\frac{r^2}{8u} \hat{\phi}^2 + \frac{r^2}{16u} \hat{\phi}^4 = -\frac{r^2}{16u} \left(2\hat{\phi}^2 - \hat{\phi}^4 \right) = \mathcal{F}_0 \left(2\hat{\phi}^2 - \hat{\phi}^4 \right)$$

AKB

Need for non-dimensionalization:

- Notice that $\mathcal{F} = -\frac{1}{2} r \phi^2 + u \phi^4$ have two minima, that we see from $\frac{\partial \mathcal{F}}{\partial \phi} = 0 = -r\phi + 4u\phi^3$ or $\phi = \pm\phi_0 = \sqrt{\frac{r}{4u}}$. Then $\mathcal{F}_0 = -\frac{1}{2} r \phi^2 + u \phi^4 = -\frac{r^2}{16u} = -\frac{r\phi_0^2}{4}$
- But now notice that, $\mathcal{F} = -\frac{1}{2} r \phi^2 + u \phi^4 = -\frac{1}{2} r \frac{\phi^2}{\phi_0^2}\phi_0^2 + u \frac{\phi^4}{\phi_0^4}\phi_0^4 = -\frac{r^2}{8u}\hat{\phi}^2 + \frac{r^2}{16u}\hat{\phi}^4$

$$= -\frac{r^2}{8u}\hat{\phi}^2 + \frac{r^2}{16u}\hat{\phi}^4 = -\frac{r^2}{16u}\left(2\hat{\phi}^2 - \hat{\phi}^4\right) = \mathcal{F}_0\left(2\hat{\phi}^2 - \hat{\phi}^4\right)$$

- Therefore, $\frac{\mathcal{F}}{\mathcal{F}_0} = \hat{\mathcal{F}} = 2\hat{\phi}^2 - \hat{\phi}^4$ & so, $\partial_t \hat{\phi} = 4\left(\hat{\phi} - \hat{\phi}^3\right)$. The dynamics is completely free of parameter. According to the scale, we can choose what to compute !!

Beautiful Example : Kibble mechanism in Cosmology → LCD screen defect applications. Both are governed by the same equation !!

AKB

Solving Linear Systems

Suppose we have a set of linear equations A*x = b, where $x = (x_1, x_2, x_3, \ldots, x_n)^T$ and

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \end{pmatrix}$$

, A is a nonsingular matrix. Of interest are tridiagonal, symmetric positive (semi)-definite (SPD), triangular etc matrix.

AKB

Solving Linear Systems

Suppose we have a set of linear equations A*x = b, where $x = (x_1, x_2, x_3, \ldots, x_n)^T$ and

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \end{pmatrix}$$

, A is a nonsingular matrix. Of interest are tridiagonal, symmetric positive (semi)-definite (SPD), triangular etc matrix. A square matrix is lower (upper) triangular if all elements above main diagonal are zero. So,

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 16 & 21 \\ 4 & 28 & 73 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}}_{L} x \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{pmatrix}}_{U}$$

AKB

Solving Linear Systems

Suppose we have a set of linear equations A*x = b, where $x=(x_1, x_2, x_3, \ldots, x_n)^T$ and

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \end{pmatrix}$$

, A is a nonsingular matrix. Of interest are tridiagonal, symmetric positive (semi)-definite (SPD), triangular etc matrix. A square matrix is lower (upper) triangular if all elements above main diagonal are zero. So,

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 16 & 21 \\ 4 & 28 & 73 \end{pmatrix} = \underset{L}{\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}} x \underset{U}{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{pmatrix}}$$

- Standard method to solve a linear (tridiagonal) system is to use "**Gaussian Elimination**", meaning, first stage of eliminating all component of A below the main diagonal (or reducing tridiagonal A to row-triangular form) and second stage of backward solution (backsolve).

AKB

Gaussian Elimination

- So, in tridiagonal system $Ax=b\,;A=\text{tridiag}(l_i,d_i,u_i)$, where $l_i=a_{i,i-1}, d_i=a_{i,i}$, $u_i=a_{i,i+1}$

$$[A|b]=\left(\begin{array}{ccccc|c} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ l_2 & d_2 & u_2 & \dots & 0 & b_2 \\ 0 & l_3 & d_3 & \dots & 0 & \\ \vdots & & \ddots & \ddots & u_{n-1} & b_{n-1} \\ 0 & \dots & 0 & l_n & d_n & b_n \end{array}\right)$$

. Now we transform from tridiag to U matrix, as,

$$d_1x_1+u_1x_2=b_1, \qquad \text{or } x_1=(b_1-u_1x_2)/d_1$$

$$l_2x_1+d_2x_2+u_2x_3=b_2, \qquad \text{or } (d_2-u_1l_2/d_1)x_2+u_2x_3 = b_2-b_1l_2/d_1$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

AKB

Gaussian Elimination

- So, in tridiagonal system $Ax=b\,;A=\text{tridiag}(l_i,d_i,u_i)$, where $l_i=a_{i,i-1}, d_i=a_{i,i}$, $u_i=a_{i,i+1}$

$$[A|b]=\left(\begin{array}{ccccc|c} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ l_2 & d_2 & u_2 & \dots & 0 & b_2 \\ 0 & l_3 & d_3 & \dots & 0 & \\ \vdots & \ddots & \ddots & & u_{n-1} & b_{n-1} \\ 0 & \dots & 0 & l_n & d_n & b_n \end{array}\right)$$

. Now we transform from tridiag to U matrix, as,

$$d_1x_1+u_1x_2=b_1, \qquad \text{or } x_1=(b_1-u_1x_2)/d_1$$

$$l_2x_1+d_2x_2+u_2x_3=b_2, \qquad \text{or } (d_2-u_1l_2/d_1)x_2+u_2x_3 = b_2-b_1l_2/d_1$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

So,

$$[A|b]\sim\left(\begin{array}{ccccc|c} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ 0 & d_2-u_1(l_2/d_1) & u_2 & \dots & 0 & b_2-b_1(l_2/d_1) \\ 0 & l_3 & d_3 & \dots & 0 & \vdots \\ \vdots & \ddots & \ddots & & u_{n-1} & b_{n-1} \\ 0 & \dots & 0 & l_n & d_n & b_n \end{array}\right)$$

$$d_1\neq 0$$

$$d_2-u_2(l_2/d_1)\neq 0$$

Row Equivalent

AKB

Gaussian Elimination

- So, in tridiagonal system $Ax=b\,;A=\text{tridiag}(l_i,d_i,u_i)$, where $l_i=a_{i,i-1},d_i=a_{i,i}$, $u_i=a_{i,i+1}$

$$[A|b]=\left(\begin{array}{ccccc|c} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ l_2 & d_2 & u_2 & \dots & 0 & b_2 \\ 0 & l_3 & d_3 & \dots & 0 & \\ \vdots & \ddots & \ddots & & u_{n-1} & b_{n-1} \\ 0 & \dots & 0 & l_n & d_n & b_n \end{array}\right)$$

. Now we transform from tridiag to U matrix, as,

$$d_1x_1+u_1x_2=b_1, \qquad \text{or } x_1=(b_1-u_1x_2)/d_1$$

$$l_2x_1+d_2x_2+u_2x_3=b_2, \qquad \text{or } (d_2-u_1l_2/d_1)x_2+u_2x_3=b_2-b_1l_2/d_1$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

$$\dots,\dots,\dots,\dots,\dots$$

So,

$$[A|b]\sim\left(\begin{array}{ccccc|c} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ 0 & d_2-u_1(l_2/d_1) & u_2 & \dots & 0 & b_2-b_1(l_2/d_1) \\ 0 & l_3 & d_3 & \dots & 0 & \vdots \\ \vdots & \ddots & \ddots & & u_{n-1} & b_{n-1} \\ 0 & \dots & 0 & l_n & d_n & b_n \end{array}\right)$$

$$d_1\neq 0$$

$$d_2-u_2(l_2/d_1)\neq 0$$

Row Equivalent

So general form is $Diagonal_k=d_k-u_{k-1}(l_k/d_{k-1})\,;\;\; Vector_k=b_k-b_{k-1}(l_k/d_{k-1})$ and by reducing the last equation, we get, $d_n-u_{n-1}(l_n/d_{n-1})x_n=b_n-b_{n-1}(l_n/d_{n-1})$, so

AKB

we can easily solve the last equation to find x_n and using this value to find x_{n-1} and so on (backward).

we can easily solve the last equation to find x_n and using this value to find x_{n-1} and so on (backward).

To illustrate the process, let's look at a concrete example that we will work through in detail. Consider the system of equations

$$\begin{aligned} 4x_1 + 2x_2 - x_3 &= 5 \\ x_1 + 4x_2 + x_3 &= 12 \\ 2x_1 - x_2 + 4x_3 &= 12 \end{aligned}$$

which can be written in matrix–vector form as

$$\begin{bmatrix} 4 & 2 & -1 \\ 1 & 4 & 1 \\ 2 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 12 \end{bmatrix}.$$

We write this as an augmented matrix:

$$A' = \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 1 & 4 & 1 & 12 \\ 2 & -1 & 4 & 12 \end{array}\right].$$

Then the elimination algorithm proceeds as follows:

$$A' = \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 1 & 4 & 1 & 12 \\ 2 & -1 & 4 & 12 \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 2 & -1 & 4 & 12 \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & -2 & \frac{9}{2} & \frac{19}{2} \end{array}\right].$$

The first step was accomplished by multiplying the first row by $\frac{1}{4}$ and subtracting the result from the second row; the second step was accomplished by multiplying the first row by $\frac{1}{2}$ and subtracting the result from the third row. To finish the job, we have (by multiplying the second row by $-\frac{4}{7}$ and subtracting from the third row)

$$A' \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & -2 & \frac{9}{2} & \frac{19}{2} \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & 0 & \frac{73}{14} & \frac{219}{14} \end{array}\right] = A''.$$

This augmented matrix represents a triangular system—meaning that the coefficient matrix is triangular—as follows:

$$A'' = [U \mid c] \Rightarrow Ux = c,$$

that is,

$$\begin{bmatrix} 4 & 2 & -1 \\ 0 & \frac{7}{2} & \frac{5}{4} \\ 0 & 0 & \frac{73}{14} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ \frac{43}{4} \\ \frac{219}{14} \end{bmatrix};$$

and we can now solve by interpreting each row as follows:

Third Row: $\frac{73}{14}x_3 = \frac{219}{14} \Rightarrow x_3 = 3;$

Second Row: $\frac{7}{2}x_2 + \frac{5}{4}x_3 = \frac{43}{4} \Rightarrow x_2 = 2;$

First Row: $4x_1 + 2x_2 - x_3 = 5 \Rightarrow x_1 = 1.$

AKB

we can easily solve the last equation to find x_n and using this value to find x_{n-1} and so on (backward).

To illustrate the process, let's look at a concrete example that we will work through in detail. Consider the system of equations

$$\begin{aligned} 4x_1 + 2x_2 - x_3 &= 5 \\ x_1 + 4x_2 + x_3 &= 12 \\ 2x_1 - x_2 + 4x_3 &= 12 \end{aligned}$$

which can be written in matrix–vector form as

$$\begin{bmatrix} 4 & 2 & -1 \\ 1 & 4 & 1 \\ 2 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 12 \end{bmatrix}.$$

We write this as an augmented matrix:

$$A' = \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 1 & 4 & 1 & 12 \\ 2 & -1 & 4 & 12 \end{array}\right].$$

Then the elimination algorithm proceeds as follows:

$$A' = \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 1 & 4 & 1 & 12 \\ 2 & -1 & 4 & 12 \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 2 & -1 & 4 & 12 \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & -2 & \frac{9}{2} & \frac{19}{2} \end{array}\right].$$

The first step was accomplished by multiplying the first row by $\frac{1}{4}$ and subtracting the result from the second row; the second step was accomplished by multiplying the first row by $\frac{1}{2}$ and subtracting the result from the third row. To finish the job, we have (by multiplying the second row by $-\frac{4}{7}$ and subtracting from the third row)

$$A' \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & -2 & \frac{9}{2} & \frac{19}{2} \end{array}\right] \sim \left[\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 0 & \frac{7}{2} & \frac{5}{4} & \frac{43}{4} \\ 0 & 0 & \frac{73}{14} & \frac{219}{14} \end{array}\right] = A''.$$

This augmented matrix represents a triangular system—meaning that the coefficient matrix is triangular—as follows:

$$A'' = [U \mid c] \Rightarrow Ux = c,$$

that is,

$$\begin{bmatrix} 4 & 2 & -1 \\ 0 & \frac{7}{2} & \frac{5}{4} \\ 0 & 0 & \frac{73}{14} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ \frac{43}{4} \\ \frac{219}{14} \end{bmatrix};$$

and we can now solve by interpreting each row as follows:

Third Row: $\frac{73}{14}x_3 = \frac{219}{14} \Rightarrow x_3 = 3;$

Second Row: $\frac{7}{2}x_2 + \frac{5}{4}x_3 = \frac{43}{4} \Rightarrow x_2 = 2;$

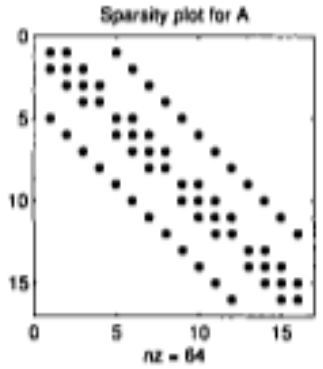
First Row: $4x_1 + 2x_2 - x_3 = 5 \Rightarrow x_1 = 1.$

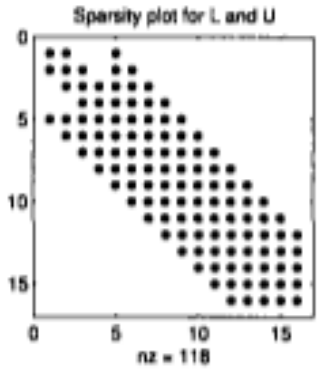
- For an ill-conditioned matrix, Gaussian elimination is adversely affected by rounding error. This lead to iterative refinement/improvement of the algorithm.

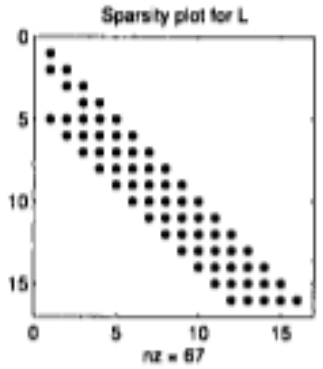
AKB

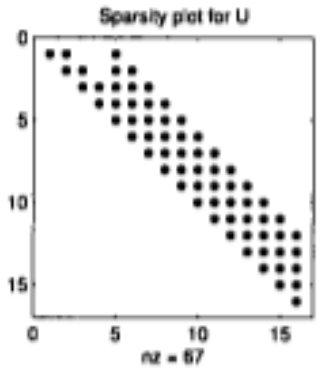
- For a large sparse matrix, Gaussian elimination is bad because L & U is dense!!

$$A = \left[\begin{array}{cccc|cccc|cccc|cccc}
-4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4
\end{array}\right]$$





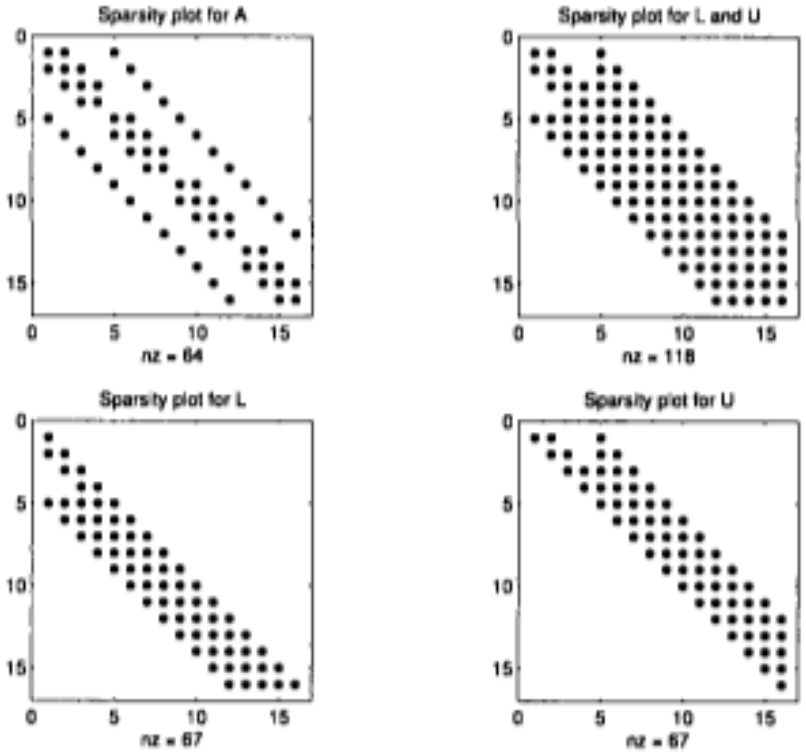




- In MATLAB/Octave, $spy(A)$ checks sparsity.

- For a large sparse matrix, Gaussian elimination is bad because L & U is dense!!

$$A = \left[\begin{array}{cccc|cccc|cccc|cccc}
-4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4
\end{array}\right]$$



- In MATLAB/Octave, $spy(A)$ checks sparsity.

Iterative (Inexact/approximate) Methods

Splitting methods, method of conjugate gradients, Jacobi iteration, Gauss-Seidel iteration and so on ....

- For a linear system $Ax=b$, we split $A=M-N$, such that linear system of form $Mx'=b'$ is easy to solve, or $x'=M^{-1}b'$ is easy to compute.

- It follows, $Ax=b$, or $(M-N)x=b$, or $Mx=Nx+b$, or $x=M^{-1}Nx+M^{-1}b$. So, given an initial guess x^0, iteration is to compute the sequence of vectors $x^{k+1}=M^{-1}Nx^k+M^{-1}b$.

iteration matrix T

- It follows, $Ax=b$, or $(M-N)x=b$, or $Mx=Nx+b$, or $x=\underbrace{M^{-1}N}_{\text{iteration matrix } T}x+M^{-1}b$. So, given an initial guess x^0, iteration is to compute the sequence of vectors $x^{k+1}=M^{-1}Nx^k+M^{-1}b$.
- Note, $T=M^{-1}N=M^{-1}(M-A)=I-M^{-1}A \leq I$ (always) so the method works best when $M^{-1}\approx A^{-1}$ or $M=A$. This is the backbone of Iterative methods.

- It follows, $Ax=b$, or $(M-N)x=b$, or $Mx=Nx+b$, or $x=\underbrace{M^{-1}N}_{\text{iteration matrix } T}x+M^{-1}b$. So, given an initial guess x^0, iteration is to compute the sequence of vectors $x^{k+1}=M^{-1}Nx^k+M^{-1}b$.
- Note, $T=M^{-1}N=M^{-1}(M-A)=I-M^{-1}A \leq I(\text{always})$ so the method works best when $M^{-1}\approx A^{-1}$ or $M=A$. This is the backbone of Iterative methods.

Jacobi Iteration Its called "diagonal inversion" as it involves inversion of diag(A) with A = D - (D – A), so that iteration becomes $x^{k+1}=(I-D^{-1}A)x^k+D^{-1}b,$

$$=x^k-D^{-1}Ax^k+D^{-1}b.$$

- It follows, $Ax=b$, or $(M-N)x=b$, or $Mx=Nx+b$, or $x=\underbrace{M^{-1}N}_{\text{iteration matrix } T}x+M^{-1}b$. So, given an initial guess x^0, iteration is to compute the sequence of vectors $x^{k+1}=M^{-1}Nx^k+M^{-1}b$.
- Note, $T=M^{-1}N=M^{-1}(M-A)=I-M^{-1}A \leq I\,(\text{always})$ so the method works best when $M^{-1}\approx A^{-1}$ or $M=A$. This is the backbone of Iterative methods.

Jacobi Iteration Its called "diagonal inversion" as it involves inversion of diag(A) with A = D - (D – A), so that iteration becomes $x^{k+1}=(I-D^{-1}A)x^k+D^{-1}b$,

$$=x^k-D^{-1}Ax^k+D^{-1}b.$$

Example: $\begin{pmatrix}4&1&0&0\\1&5&1&0\\0&1&6&1\\1&0&1&4\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}=\begin{pmatrix}1\\7\\16\\14\end{pmatrix}$ has exact solution, $x=(0,1,2,3)^T$. If we start from 1st Jacobi iteration for $x=(0,0,0,0)^T$ with D = diag(4,5,6,4), then in 19 iterations, $x^{k+1}-x^k<10^{-6}$.

$x^1=x^0-D^{-1}Ax^0+D^{-1}b=(0.2500,1.4000,2.6667,3.5000)^T$

$x^2=x^1-D^{-1}Ax^1+D^{-1}b=(-0.1000,0.8167,1.8500,2.7708)^T$

$x^3=x^2-D^{-1}Ax^2+D^{-1}b=(0.0458,1.0500,2.0688,3.0625)^T$

Gauss-Seidel Iteration An obvious extension of Jacobi iteration is to invert the entire lower triangular part of A by A = L – (L – A) so that iteration becomes

$$x^{k+1}=(I-L^{-1}A)x^k+L^{-1}b=x^k-L^{-1}Ax^k+L^{-1}b.$$

$A=\begin{pmatrix}4&1&0&0\\1&5&1&0\\0&1&6&1\\1&0&1&4\end{pmatrix}$ $L=\begin{pmatrix}4&0&0&0\\1&5&0&0\\0&1&6&0\\1&0&1&4\end{pmatrix}$. Here we do not compute L inverse but solve the linear system Lz = Ax (easy).

$x^1=(0.2500,1.3500,2.4417,2.8271)^T$, $x^2=(-0.0875,0.9292,2.0406,3.0117)^T$,

$x^3=(0.0177,0.9883,2.0000,2.9956)^T$.

- Gauss-Seidel converges faster (11 iterations) than Jacobi, as more of the matrix is inverted at each step, especially for SPD matrices.

Syllabus

- Introduction to programming in python ➡ Introduction to programming, constants, variables and data types, dynamical typing, operators and expressions, modules, I/O statements, file handling, iterables, compound statements, indentation in python, the if-elif-else block, for and while loops, nested compound statements.
- Programs ➡ (a) Elementary calculations with different type of data e.g., area & volume of regular shapes using formula. Creation and handling one dimensional array. Sum and average of a list of numbers stored in array, finding the largest and lowest number from a list, swapping two data in a list, sorting of numbers in an array using bubble sort, insertion sort method. Calculation of term value in a series and finding the other terms with a seed (value of particular term) and calculation of different quantities with series. Convergence & accuracy of series. Introduction of three dimensional array. Simple calculations of matrices e.g., addition, subtraction, multiplication.

  (b) Curve fitting, Least square fit, Goodness of fit, standard deviation,

  i. Ohms law to calculate R,

  ii. Hooke's law to calculate spring constant

AKB

Introduction to Programming in Python

- Python is a scripting language, one of the most handy, easy-to-implement & open-source languages available. Unlike Fortran 77, 90/95 or C (high level), it directly interacts with the interpreter on every sentence executed on the command prompt (>>) [Python REPL-interpreter]. We'll however learn to write standard programs (py-scripts) in a file (suitably_choiced_name.py) & then execute it without compilation.

- Name REPL (Read-Evaluate-Print-Loops) is because it (a) reads what a user types, (b) evaluates what it reads, (c) prints out the return value after evaluation, and (d) loops back and does it all over again.

- Object oriented programming: deals with computable data as an object on which different methods act to produce a desired result. Its nature is defined as its properties. These properties can be probed by functions, which are called methods.

AKB

Introduction to Programming in Python

Field of Study	Python Module
Scientific Computation	numpy, scipy, sympy
Plotting/Visualization	matplotlib
Image Processing	scikit-image
Graphic User Interface (GUI)	pyQT
Statistics	pandas
Game Development	PyGame
Networking	networkx
Cryptography	pyOpenSSL
Database	SQLAlchemy
Language Processing	nltk
Testing	nose
HTML/XML parsing	BeautifulSoup
Machine Learning	scikit-learn, tensorflow

AKB

Introduction to Programming in Python

- Create a directory of your Group. This is where throughout the course your python scripts (codes) will be stored.
- Open **IDLE** editor & create a new file, say, hello.py.

  Within it, write: **print('Hello World')** Save & Run the code and welcome yourself !!
- Getting out of terminal : exit() or Ctrl + D.
- print ('Hello World')

  print 'I am a first year UG student'

  print "I'm new to Python Programming"

  print 'It is my first' + ' ' + 'Python code'
- ...code fragment... # Single line comments are commented out

  """

  Multiple line comment can be

  commented out like this.

  """

AKB

Introduction to Programming in Python

Syntax, Variables, Numbers, Operators:

Variables ➔ Name of a variable (or any identifier such as class, function, module or other object) can be anything combined with alphabets, letters & some symbols in the keyboard (except @, $, % etc.), beginning with a letter (the upper case or lower case letters from A to Z). But the names cannot be the words given in the following table. Those are Python keywords (in lower case letters) reserved for the use by system. Also, names are case sensitive. Suppose, we define a variable as ‘ABC’ and later we call as ‘Abc’, then it will not work.

Reserved words ➔ Using the module keyword, you can obtain the list of keywords.

import keyword; print ("Python keywords:", keyword.kwlist)

'Python keywords:', ['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield']

AKB

Introduction to Programming in Python

Syntax, Variables, Numbers, Operators

- Operators:

=	v = a + b	
+=	v = v + b	
-=	v = v – b	
/=	v = v / a	
//=	v = v // a	(roundoff)
*=	v = v * a	
**=	v = v ** a	(to the power)
%=	v = v % a	(remainder)

For **a=10, b=2**, Addition: **a+b = 12**, Substraction: **a–b = 8**, Multiplication: **a*b = 20**, Division: **a/b = 5**, Modulus [Remainder, when a is divided by b] **a%b = 0**, Power or exponent **a**b = 100**, Rounding off, e.g. if **a=100, b=3, 100/3= 33** also **100//3=33** but **100.0/3=33.3333**.....whereas **100.0//3 = 33.0**

I/O (Input/Output) Statements

```
a=input()  # Wait for the input value of 'a'.
a=input("Enter a: \n")
a, b, c = input()  # For many inputs.
```

- Multiple assignments within same statement: >> **a = b = c = 10**
- Order of usage: **PEMDAS (Parenthesis → Exponents → Multiplication → Division → Addition → Substracton.**

AKB

Introduction to Programming in Python

Comparison Operators:

Operator Symbol	Operator Meaning	Example
==	equal to	1==1 is True, 1==2 is False
!=	not equal to	1!=1 is False, 1!=2 is True
<>	not equal to	1<>1 is False, 1<>2 is True
<	less than	1<2 is True, 2<1 is False
>	greater than	1>2 is False, 2>1 is True
<=	less than equal to	1<=1 is True, 1<=2 is True
>=	greater than equal to	1>=1 is True, 1>=2 is False

```
>> not True
False
>> a = True; b = False; a and b
False
>> a or b : True
```

```
>> 1>=2 == 2>=1
False
>> 2>=1 == 1<=2
True
```

```
>> False > True
False
>> True > False
True
```

AKB

Introduction to Programming in Python

Array & List A list data type stores a sequence of values. All elements of list can be accessed by their index, but individual list elements can belong to any data type. Array on the other hand stores only numeric value & is not built in python interpreter need module "numpy" (Semester-3).

We can make a list of numbers or names or anything mixed, in the following way:

```
>>> X = [3, 2, 4, 1, 5, 0]                    # List of numbers
>>> Y = ["AKB", "MC", "SSB", "PD", "AD"]      # List of names
>>> Z = ["Good", 10, "Bad", 50]               # Mixed list
```

Display List **>>> list(X)** → [3, 2, 4, 1, 5, 0]

>>> list(Y) → "AKB", "MC", "SSB", "PD", "AD"

- To view, on the command prompt, **>>>X** Or **>>>print X** both works.
- Note, if we write **>>>list()**, we get back an empty list []. Also, while writing, the amount of gap between 'list' and the parenthesis () does not matter. You may write >>>list() & get back the same response.

AKB

Introduction to Programming in Python

>>> X+Y # sum of two lists (ans: **[2,3,4,1,5,0,'AKB','MC','SSB','PD','AD']**)

>>> X*3 # repeat 3 times (ans: **[2,3,4,1,5,0,2,3,4,1,5,0,2,3,4,1,5,0]**)

>>> len(X) # length of list (ans: **6**); **>>> sum(X)** # adding all element (ans: **15**)

>>> max(X) # maximum of X (ans: **5**), **>>> min(X)** # minimum of X (ans: **0**)

>>> X.index(max(X)) # Location of the maximum in list 'X' (ans: **4**)

>>> X.index(3) # Location of '3' in the list 'X' (ans: **1**)

- Positions are counted from left (like C programming): 0, 1, 2, 3,…

Numbers

int : Integers with +ive or -ive sign : Examples: 1245, -234 etc

long : Long Integers of unlimited size. Examples: 1245L

float : Floating point real numbers with a decimal point. Examples: 0.0, 2.24, -3.1e-10

complex : Complex numbers of the form a+bj, j = √-1, Examples: 1.2j, 1+2j, 1.2e-10j

Type conversion

>>> X = 15.56; int(X) # returns 15.

>>> X = 15; float(X) # returns 15.0

AKB

Introduction to Programming in Python

Playing with List

```
>>> X.append(10); list(X)          # Adding '10' at the right end of the list 'X' (ans: 2,3,4,1,5,0,10])
>>> Y.append('RM'); list(Y)        # Adding 'RM' at the right end of the list 'Y' (ans:
                                   ['AKB','MC','SSB','PD','AD','RM'])
>>> mean = sum(X)/len(X)           # Mean value from a list of numbers (ans: 15/6=2.5~2, as it
                                   is integer).
>>> mean=float(sum(X))/len(X)      # type-casting to float / make any entry in list float, say, 1.0.
>>> M = [3, ['a', -3.0, 5] ]       # (list within list) nested lists (for entering matrices)
>>> M[1][2]                        # returns 5
>>> range(5) or L = list(range(5))  # generate numbers from 0 to 4 [0,1,2,3,4]
>>> range(1,10)                    # returns [0,1,2,3,4,5,6,7,8,9]
>>> range(1, 10, 2)                # numbers from 1 to 9 with stride 2 [1, 3, 5, 7, 9].
>>> L=list(range(17,29,4))         # returns [17, 21, 25]
```

AKB

Introduction to Programming in Python

Slicing Slicing a list between i and j creates a new list containing the elements starting at index i and ending just before j. L[i : j] means create a list by taking all elements from L starting at L[i] until L[j-1] .

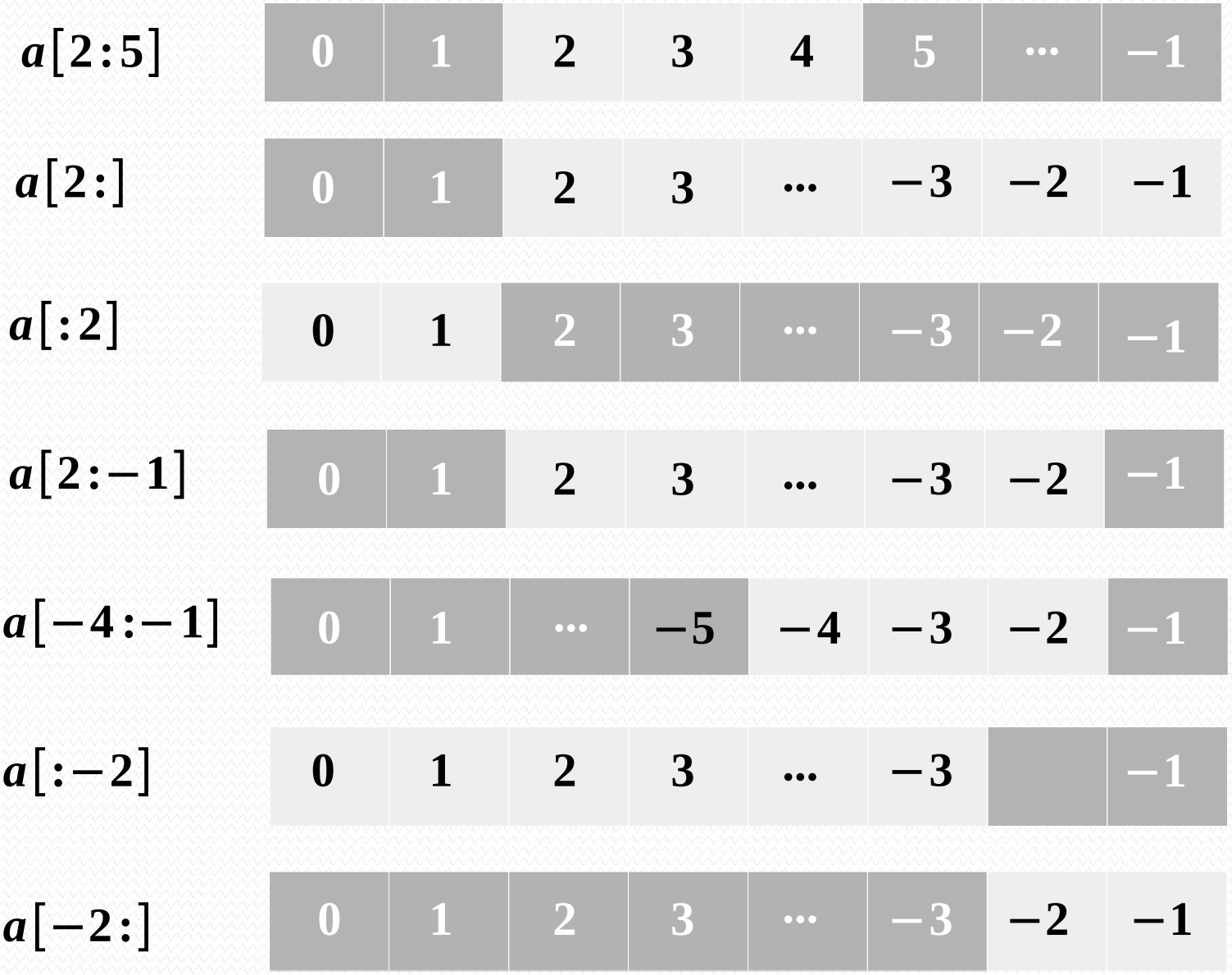
```
>>> a = list( [2, 3, 1, 4, 5, 6, 9, 10, 1] ); len(a)   # returns length of a is 9.
>>> a[:]     # returns entire list.
>>> a[1:]   # returns [3, 1, 4, 5, 6, 9, 10, 1]
>>> a[:1]   # returns 2
>>> a[3 : 6]  # returns [4, 5, 6]
```

Python allows the use of negative indexes for counting from the right. Element a[-1] is the last element in the list a.

- → a[i:] amounts to taking all elements except the i first ones,
- → a[:i] amounts to taking the first i elements,
- → a[-i:] amounts to taking the last i elements,
- → a[:-i] amounts to taking all elements except the i last ones.

AKB

Introduction to Programming in Python



AKB

Strides Length of the step from one index to the other (default stride is 1).

```
>>> L = list(range(100))   # generate 0 to 99
>>> L[:10:2]               # [0, 2, 4, 6, 8]
>>> L[::20]                # [0, 20, 40, 60, 80]
>>> L[10:20:3]             # [10, 13, 16, 19]
>>> L[20:10:-3]            # [20, 17, 14, 11] (negative stride).
```

- We can create a new list that is reversed using a negative stride →

```
>>> L = [1, 2, 3]
>>> R = L[::-1]            # R = [3, 2, 1]
```

Altering lists Insertion, deletion of elements & list concatenation. With slicing notation, list insertion & deletion is done, deletion is just replacing a part of a list by an empty list []

```
>>> L = ['a', 1, 2, 3, 4]
>>> L[2:3] = []            # ['a', 1, 3, 4]
>>> L[3:] = []             # ['a', 1, 3]
>>> L[1:1] = [1000, 2000]  # ['a', 1000, 2000, 1, 3]
```

Introduction to Programming in Python

- Two lists are concatenated by the plus operator + : >>> L = [1, -17];
  M = [-23.5, 18.3, 5.0]; L + M # returns [1, -17, -23.5, 18.3, 5.0]
- Concatenating a list n times with itself using multiplication operator * :
  >>> n = 3; n * [1.,17,3] # returns [1., 17, 3, 1., 17, 3, 1., 17, 3]
  >>> [0] * 5 # returns [0,0,0,0,0]

list.append(x) → Add x to end of the list.	L = [0, 1, 2, 3, 4] L.append(5) # [0, 1, 2, 3, 4, 5]
list.remove(x) → Remove first item from list with x.	L.remove(0) # [1, 2, 3, 4, 5]
list.insert(i,x) → Insert x at position i.	L.insert(0,0) # [0, 1, 2, 3, 4, 5]
list.count(x) → Number of times x appears in list.	L.count(2) # 1
list.sort() → Sort items of the list.	L.sort() # [0, 1, 2, 3, 4, 5]
list.reverse() → Reverse the elements of the list.	L.reverse() # [5, 4, 3, 2, 1, 0]
list.pop() → Remove last element of the list.	L.pop() # [0, 1, 2, 3, 4] L.pop() # [0, 1, 2, 3]

AKB

Introduction to Programming in Python

Tuples A tuple is an immutable list (cannot be modified). A tuple is just a comma-separated sequence of objects (a list without brackets or encloses a tuple in a pair of parentheses).

```
my_tuple = 1, 2, 3      # Our first tuple
my_tuple = (1, 2, 3)    # Same as previous
my_tuple = 1, 2, 3,     # Same as previous
len(my_tuple)           # 3, same as for lists
my_tuple[0] = 'a'       # error! Tuples are immutable
```

The comma indicates that the object is a tuple:

```
singleton = 1,          # Note the comma
len(singleton)          # 1
```

- Tuples are useful when a group of values goes together, e.g. they are used to return multiple values from functions. One may assign several variables at once by unpacking a list or tuple:

```
a, b = 0, 1     # a gets 0 and b gets 1
a, b = [0, 1]   # exactly the same effect
(a, b) = 0, 1   # same
[a,b] = [0,1]   # same thing
```

AKB

Introduction to Programming in Python

Array An array object strores a group of elements (values) of same datatype. Know more from ">>> help()" & then "help>>> array". To import array module, either, "import array" or "from array import *".

```
from array import *
a = array('i', [5, 6, -7, 8])        # i for signed integer, f for floating point (single dimension)
arr = array('d', [1.5, -2.2, 3.0, 5.75]  # d for double precision
m = array('d', [1.5, -2.2, 3.0, 5.75]   # d for double precision
```

Loops To execute a set of command repeatedly, in python "for", "while" loop is used.

For loop	While loop

sum=0
for x in range(begin, end, step):
  print "We are within a loop"
  sum += 1
print 'Sum of ', begin, ' to ', end, ' is = ', sum
	sum=0; n=5;
	while sum<n:
	  print sum
	  sum += 1
	print 'Summing 1 ', n-1, ' times = ', sum

AKB

Introduction to Programming in Python

Decision making (Logical statements) If, If … else, If … elif ...else, Nested if ...are used for logical decision making.

If:
```
if x==10: print ‘x = ’, x
OR
if x==10:
   print “value of x is 10”
```

If-else:
```
if x==10:
      print ‘x is 10’
   else:
      print ‘x is not 10’
```

If-elif-else:
```
a, b, c = input(‘Enter a,b,c: \n’)
x = b*b - 4*a*c
if x<0:
   print 'x is negative.'
elif x>0:
   print 'x is positive.'
else:
   print 'x is zero.'
```

AKB