Data Representation and Introduction to Visualization

Massimo Ricotti
ricotti@astro.umd.edu

University of Maryland
Visualization is useful for:
1. Data entry (initial conditions).
2. Code debugging and performance analysis.
3. Interpretation and display of results.

Our focus will be #3. The computational astrophysicist can either:
1. Develop new visualization software tailored to problem under study.
2. Use an existing software package.
Plotting 1-D data

- Function of one variable only: $f(x)$ vs. $x$.
- Examples: sm, gnuplot, xgobi, IDL, etc.
- Minimum requirements:
  - Read data from file.
  - Perform arithmetic manipulation of data.
  - Multiple data sets on plot.
  - Multiple plots on page.
  - Add text to plots.
Plotting 2-D data

- Function of 2 variables, i.e. \( f(x, y) \).

- If \( f \) is a scalar quantity, can:
  
  1. Make image.
     - Represent each \((x, y)\) data point by one or more pixels on screen.
     - Use integer value to represent data value at \((x, y)\) point (8 bit: 0–255; 24-bit: 0–16.8 million).
  
  2. Make contour plot.
     - Contours are isosurfaces of data.
  
  3. Make 3-D surface plot.
     - Use \((x, y)\) as 2 coordinates, \( f \) as third coordinate, plot surface.
If $f$ is a vector quantity, i.e. $f(x, y)$, can:

1. Plot vectors directly (as arrows).
   - Can be hard to see.

2. Plot streamlines.
   - Contours of $\Phi$, where $f = \nabla \Phi$.

2-D plotting packages include sm, gnuplot, xgobi, IDL, ximage, NCAR graphics, etc.
Plotting 3-D data

- Function of 3 variables, i.e. $f(x, y, z)$.

- If $f$ is a scalar quantity, can:
  1. Plot 2-D slices.
     - E.g. faces of cube.
  2. Plot isosurfaces.
     - These are now 3-D surfaces. Can use wireframe of polygons. Can shade with second variable $g(x, y, z)$.
  3. Plot volumetric rendering.
     - Solve transfer equation ("ray tracing") using emissivity proportional to data value.
Standard algorithms exist for 3-D rendering, including shadowing, hidden surface removal, etc. Often implemented in hardware. Also have “dynamic/interactive” visualization: rotation, etc.

If $f$ is a vector quantity, i.e. $f(x, y, z)$, can:

1. Plot 3-D vectors on 2-D slice.
2. Plot streamlines in 3-D.

3-D plotting packages include tipsy, xgobi, IDL, NCAR graphics, xdataslice, etc.
**Animation**

- If any one of the coordinates of data in a plot is time, it makes sense to render images as a time sequence, e.g. make animation.

- The eye is very sensitive to motion, can discover much detail using animations.

- **Animation formats include** MPEG, FLI, QT, AVI, GIF, plus many custom formats.

- **Animation players include** mpeg_play, xanim, quicktime, gifview, etc.

  - Often built into web browsers.
DATA REPRESENTATION

- Computers store data as different variable types, e.g. integer, floating point, complex, etc.

- Different machines have different wordlengths, e.g. 4-byte ints on a 32-bit machine (Pentium), 8-byte ints on a 64-bit machine (G5).

- This makes (binary) data non-portable.
Integers

- All data types represented by 0’s and 1’s.

- An integer value:

\[ j = \sum_{i=1}^{N} s_i \times 2^{N-i} \]

- \( N = \# \) of bits in word.
- \( s_i = \) value of bit \( i \) in binary string \( s \).

- E.g., \( 0 0 0 0 0 1 1 0 = 2^2 + 2^1 = 6 \) for 8-bit word.

- Use “two’s complement” method for sign (see below).

- Largest value that can be represented is \( 2^N - 1 \).

- For 32-bit word this is 4,294,967,295.
Arithmetic with integers is exact, except:

- when division results in remainder, or
- result exceeds largest representable integer.

E.g. $2 \times 10^9 + 3 \times 10^9 = \text{overflow error}$.

Note multiplication (division) by 2’s can be achieved by left-shift (right-shift), which is very fast (in C, use the $<<$ ($>>$) operator).
Two’s complement

- Exploits finite size of data representations (cyclic groups) and properties of binary arithmetic.

- To get negative of binary integer, invert all bits and add 1 to the result.

  E.g., $1 = 0 0 0 0 0 0 0 1$ in 8-bit.
  
  invert bits: $1 1 1 1 1 1 1 0$
  
  add 1: $0 0 0 0 0 0 0 1$
  
  result: $1 1 1 1 1 1 1 1 = -1$

- In 8 bits, signed char ranges from $-128$ to $+127$. 
Negative powers of 2

Binary notation can be extended to cover negative powers of 2, e.g. “110.101” is:

\[ 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 6.625. \]

Can represent real numbers by specifying some location in the word as the “binary point” (“fixed-point representation”).

In practice, use some bits for an exponent (“floating-point representation”).
**Floats**

For most machines these days, real numbers are represented by floating-point format:

\[ x = s \times M \times B^{e-E} \]

- \( s \) = sign
- \( B \) = base (usually 2, sometimes 16)
- \( M \) = mantissa
- \( e \) = exponent
- \( E \) = bias, usually 127.

In past, manufacturers used different number of bits for each of \( M \) and \( e \), resulting in non-portable code.
Currently, most manufacturers adopt IEEE standard:

- \( s \) = first bit.
- Next 8 bits are \( e \). (\( e = 255 \) reserved for inf & NaN.)
- Last 23 bits are \( M \), expressed as a binary fraction, either 1.F, or, if \( e = 0 \), 0.F (in which case \( E = 126 \)), where F is in base 2.

E.g., 0 10000001 1010000000000000000000000 = 
\[ (+1) \left(2^{129-127}\right) \left(1 + 0.5 + 0.125\right) = 6.5. \]

Largest single-precision float
\[ f_{\text{max}} = 2^{127} \times \left(1 + 1/2 + 1/4 + \cdots + 1/2^{23}\right) \approx 3.4028235 \times 10^{38} \] (just under \( 2^{128} \)).

Smallest (and least precise!) \( f_{\text{min}} = 2^{-149} \approx 10^{-45} \).
Round-off error

- Not all values along real axis can be represented.
- There are $10^{38}$ integers between $f_{\text{min}}$ and $f_{\text{max}}$, but only $2^{32} \approx 10^9$ bit patterns.
- Values $< |10^{-45}|$ result in “underflow” error.
- If value cannot be represented, next nearest value is produced. Difference between desired and actual value is called “round-off error” (RE).
- Smallest value $e_m$ for which $1 + e_m > 1$ is called “machine accuracy,” typically $2^{-23} \sim 10^{-7}$ for 32 bits.
- Double precision greatly reduces $e_m$ ($\sim 10^{-16}$). (In this case the 64 bits are divided into 1 sign bit, 11 exponent bits, and 52 mantissa bits; the bias is 1023.)
RE accumulates in a calculation:

- Random walk: total error $\sqrt{N}e_m$ after $N$ operations.
- But algorithms rarely random, giving linear error $Ne_m$.

Subtraction of two very nearly equal numbers can give rise to large RE.

E.g., solution of quadratic equation...

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

...can go badly wrong whenever $ac \ll b^2$ (Cf. PS#2).

RE cannot be avoided—it is a consequence of using a finite number of bits to represent real values.
Truncation error

- In practice, most numerical algorithms approximate desired solution with a finite number of arithmetic operations, e.g.,
  - evaluating integral by quadrature;
  - summing series using finite number of terms.

- Difference between true solution and numerical approximation to solution is called “truncation error” (TE).

- TE exists even on “perfect” machine with no RE.

- TE is under programmer’s control; much effort goes into reducing it.

- Usually RE and TE do not interact.

- Sometimes TE can amplify RE until it swamps calculation. The solution is then called unstable.
  
  E.g., integer powers of Golden Mean (Cf. PS#2).