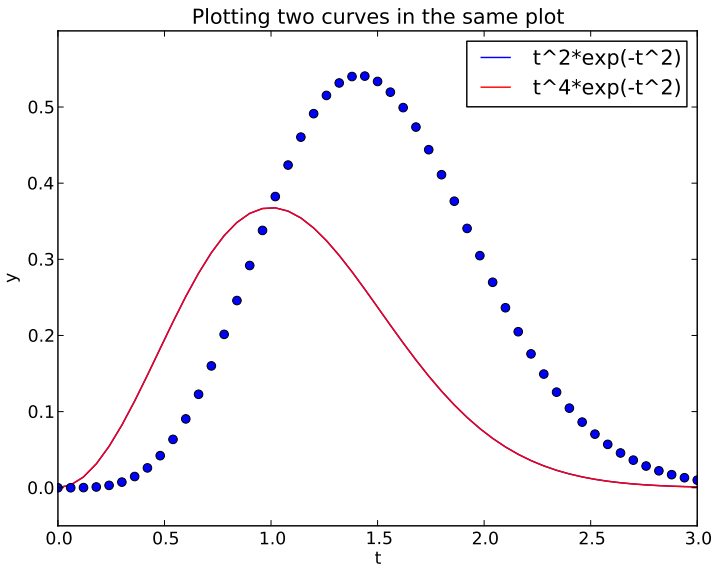
Ch.5: Array computing and curve Plotting

Hans Petter Langtangen[1,2]

[1]Simula Research Laboratory
[2]University of Oslo, Dept. of Informatics

Sep 9, 2014

Goal: learn to visualize functions



We need to learn about a new object: array

- Curves $y = f(x)$ are visualized by drawing straight lines between points along the curve
- Meed to store the coordinates of the points along the curve in lists or *arrays* `x` and `y`

- Arrays $\approx$ lists, but computationally much more efficient
- To compute the `y` coordinates (in an array) we need to learn about *array computations* or *vectorization*
- Array computations are useful for much more than plotting curves!

The minimal need-to-know about vectors

- Vectors are known from high school mathematics, e.g., point (x, y) in the plane, point (x, y, z) in space
- In general, a vector v is an n-tuple of numbers: $v = (v_0, \ldots, v_{n-1})$
- Vectors can be represented by lists: v_i is stored as `v[i]`, but we shall use arrays instead

Vectors and arrays are key concepts in this chapter. It takes separate math courses to understand what vectors and arrays really are, but in this course we only need a small subset of the complete story. A learning strategy may be to just start using vectors/arrays in programs and later, if necessary, go back to the more mathematical details in the first part of Ch. 5.

The minimal need-to-know about arrays

Arrays are a generalization of vectors where we can have multiple indices: $A_{i,j}$, $A_{i,j,k}$

Example: table of numbers, one index for the row, one for the column

$$\begin{bmatrix} 0 & 12 & -1 & 5 \\ -1 & -1 & -1 & 0 \\ 11 & 5 & 5 & -2 \end{bmatrix} \qquad A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,n-1} \\ \vdots & \ddots & \vdots \\ A_{m-1,0} & \cdots & A_{m-1,n-1} \end{bmatrix}$$

- The no of indices in an array is the *rank* or *number of dimensions*
- Vector = one-dimensional array, or rank 1 array
- In Python code, we use Numerical Python arrays instead of nested lists to represent mathematical arrays (because this is computationally more efficient)

Storing (x,y) points on a curve in lists

Collect points on a function curve $y = f(x)$ in lists:

```
>>> def f(x):
...     return x**3       # sample function
...
>>> n = 5                 # no of points
>>> dx = 1.0/(n-1)        # x spacing in [0,1 ]
>>> xlist = [i*dx for i in range(n)]
>>> ylist = [f(x) for x in xlist]

>>> pairs = [[x, y] for x, y in zip(xlist, ylist)]
```

Turn lists into Numerical Python (NumPy) arrays:

```
>>> import numpy as np        # module for arrays
>>> x2 = np.array(xlist)      # turn list xlist into array
>>> y2 = np.array(ylist)
```

Make arrays directly (instead of lists)

The pro drops lists and makes NumPy arrays directly:

```
>>> n = 5                     # number of points
>>> x2 = np.linspace(0, 1, n) # n points in [0, 1]
>>> y2 = np.zeros(n)          # n zeros (float data type)
>>> for i in xrange(n):
...     y2[i] = f(x2[i])
...
```

Note:

- `xrange` is like `range` but faster (esp. for large `n` - `xrange` does not explicitly build a list of integers, `xrange` just lets you loop over the values)
- Entire arrays must be made by `numpy` (`np`) functions

Arrays are not as flexible as list, but computational much more efficient

- List elements can be any Python objects
- Array elements can only be of one object type
- Arrays are very efficient to store in memory and compute with if the element type is `float`, `int`, or `complex`
- Rule: use arrays for sequences of numbers