# Week 8: misc. numeric computation

*Ben Bolker*

*20:03 07 March 2017*

## Integration

- How should we integrate $\int_0^1 \sqrt{1-x^2}\,dx$?

```
## /usr/local/lib/python3.4/dist-packages/matplotlib/backends/backend_gtk3.py:219: Warning: Source ID 8
##   GLib.source_remove(self._idle_draw_id)
```

- What about something horrible like $\int_0^1 \exp(-x^2)\log(1+x)\,dx$ ?
- Let's write a couple of programs:
  - brute force (without array functions, only `for` loops)
  - with array functions (we shouldn't need any `for` loops)
- How could we make this better?
  - better integration rules (trapezoid, Simpson's?)
  - choice of `n`
  - adaptive integration (i.e., choice of tolerance): loops within loops ...
- Hard things
  - high dimensions
  - weird shapes/limits of integration

## (Pseudo)random numbers

- From Wikipedia: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin" (von Neumann) (original paper) ... "We are here dealing with mere 'cooking recipes' for making digits; probably they can not be justified, but should merely be judged by their results ..."
- *linear congruential generators*:
  - $x_n = (ax_{n-1} + c) \mathrm{mod} m$
  - or `x = (a*x +c ) % m`
  - from here:

```
x = [5]
(a,c,m) = (2,3,10)
for i in range(9):
    newx = (a*x[-1]+c) % m
    x.append(newx)
print(x)
```

```
## [5, 3, 9, 1, 5, 3, 9, 1, 5, 3]
```

- Park-Miller *minimal standard generator*:

```
import numpy as np
(a,c,m) = (16807,0,2147483647)
x = [5]
for i in range(9):
    newx = (a*x[-1]+c) % m
    x.append(newx)
print(np.array(x)/m)
```

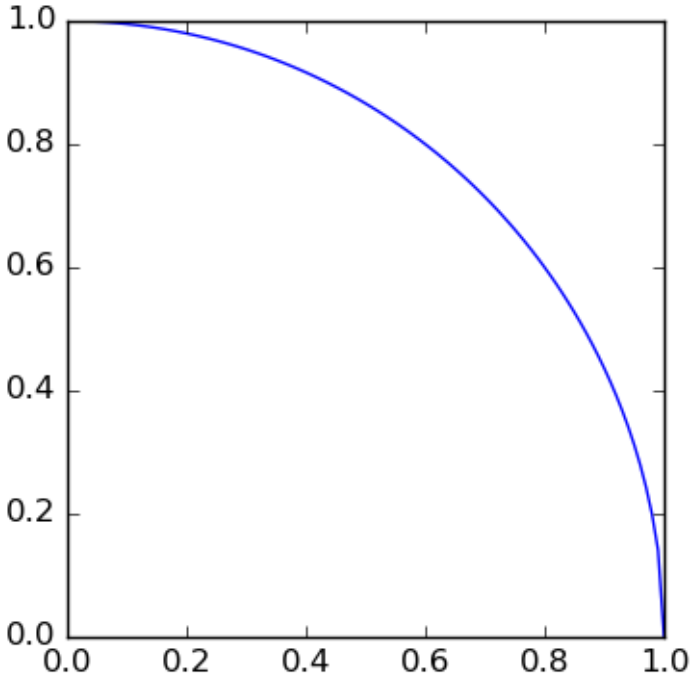


Figure 1: quarter circle

```
## [  2.32830644e-09   3.91318463e-05   6.57688941e-01   7.78026611e-01
##    2.93250660e-01   6.63836187e-01   9.47959316e-02   2.35223081e-01
##    3.94323584e-01   3.96482029e-01]
```

```
## /usr/local/lib/python3.4/dist-packages/matplotlib/backends/backend_gtk3.py:219: Warning: Source ID 8
##   GLib.source_remove(self._idle_draw_id)
```

- using `numpy`: reference

```
import numpy.random as rand
a = rand.rand(1000)
```

- can also do useful things like
- pick from a list: `choice()` (with or without replacement)
- shuffle values: `shuffle()` (in-place)
- pick values from different distributions
- sample from a large range of non-uniform distributions (Poisson, Normal, binomial ...)
- using random number generators for serious work:
- know what generator is used (Mersenne twister is OK)
- set the seed: `seed()`
- using random numbers for cryptography: be super-paranoid: see e.g. this

## Monte Carlo integration

- *Monte Carlo* techniques (Ulam)
- Monte Carlo integration
  - pick uniform numbers in a simple region (e.g. square)

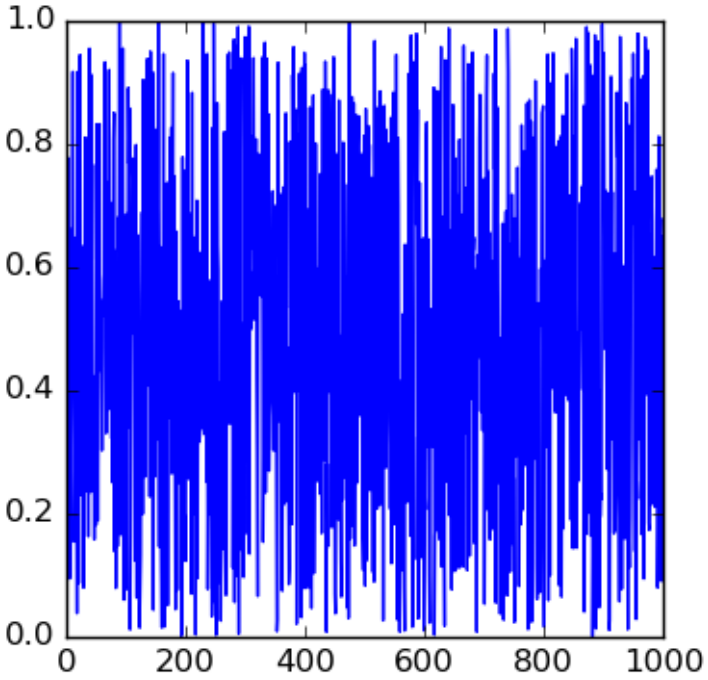


Figure 2: random values

  - what fraction fall under the curve?
  - also called *rejection sampling* in this context
- let's write the program