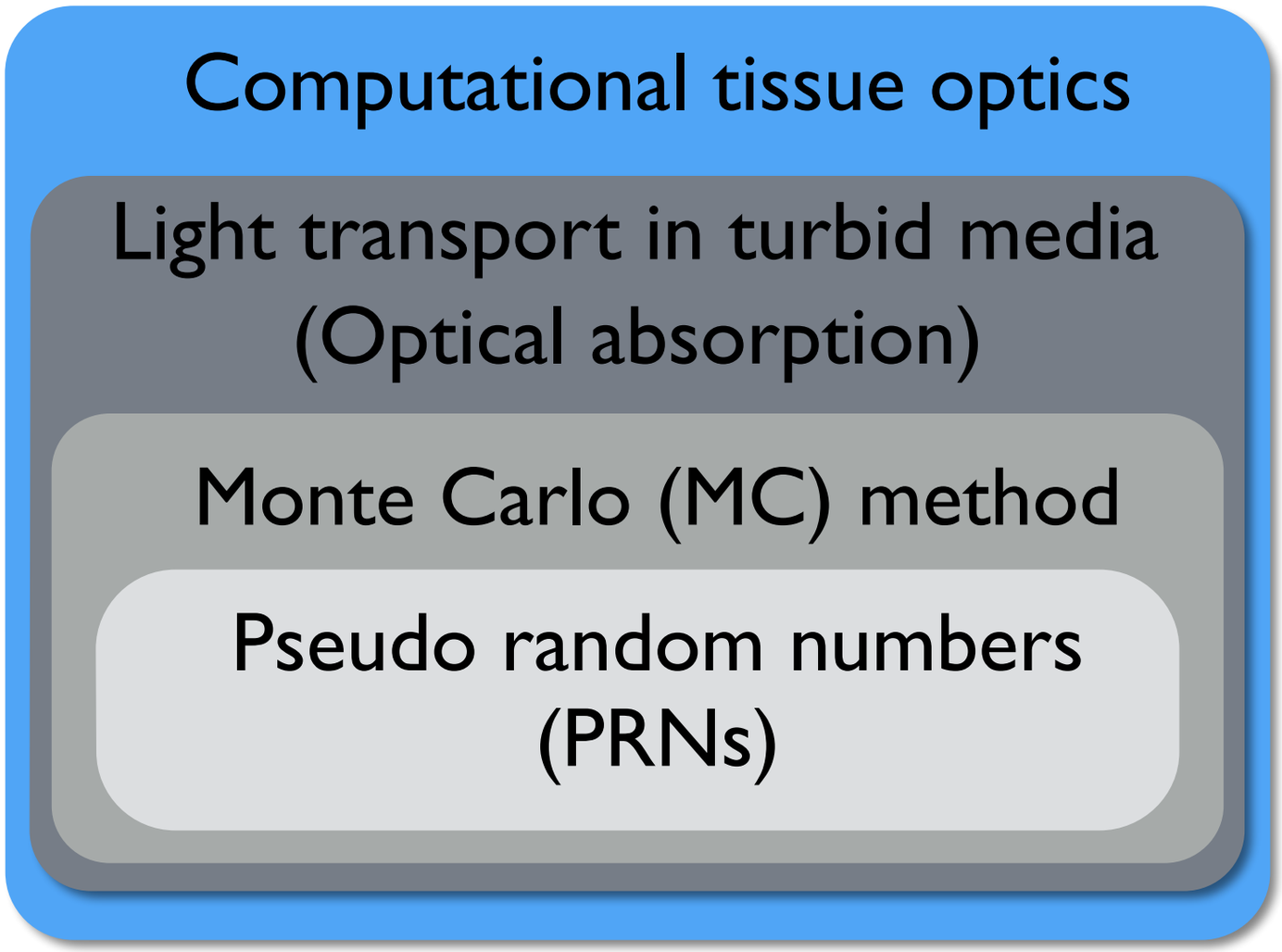
Motivation — Why computer simulations?

Optical Technologies.

- Theory dilemma:
  - Analytic solution only for rare special cases
  - System simply too complex for analytic solution
- Experiment dilemma:
  - Controlling all parameters is not an easy task
  - Measurement noise inevitable
  - New measurement protocols might involve substantial changes on experimental setup
- Simulation:
  - Computer simulations = numerical experiments
  - Complements theory & experiment and can be used to model, efficiently explore and predict phenomena on all scales ranging from Quark-Gluon plasma to cosmology

Lecture focus on tissue optics: outline



Optical Technologies.

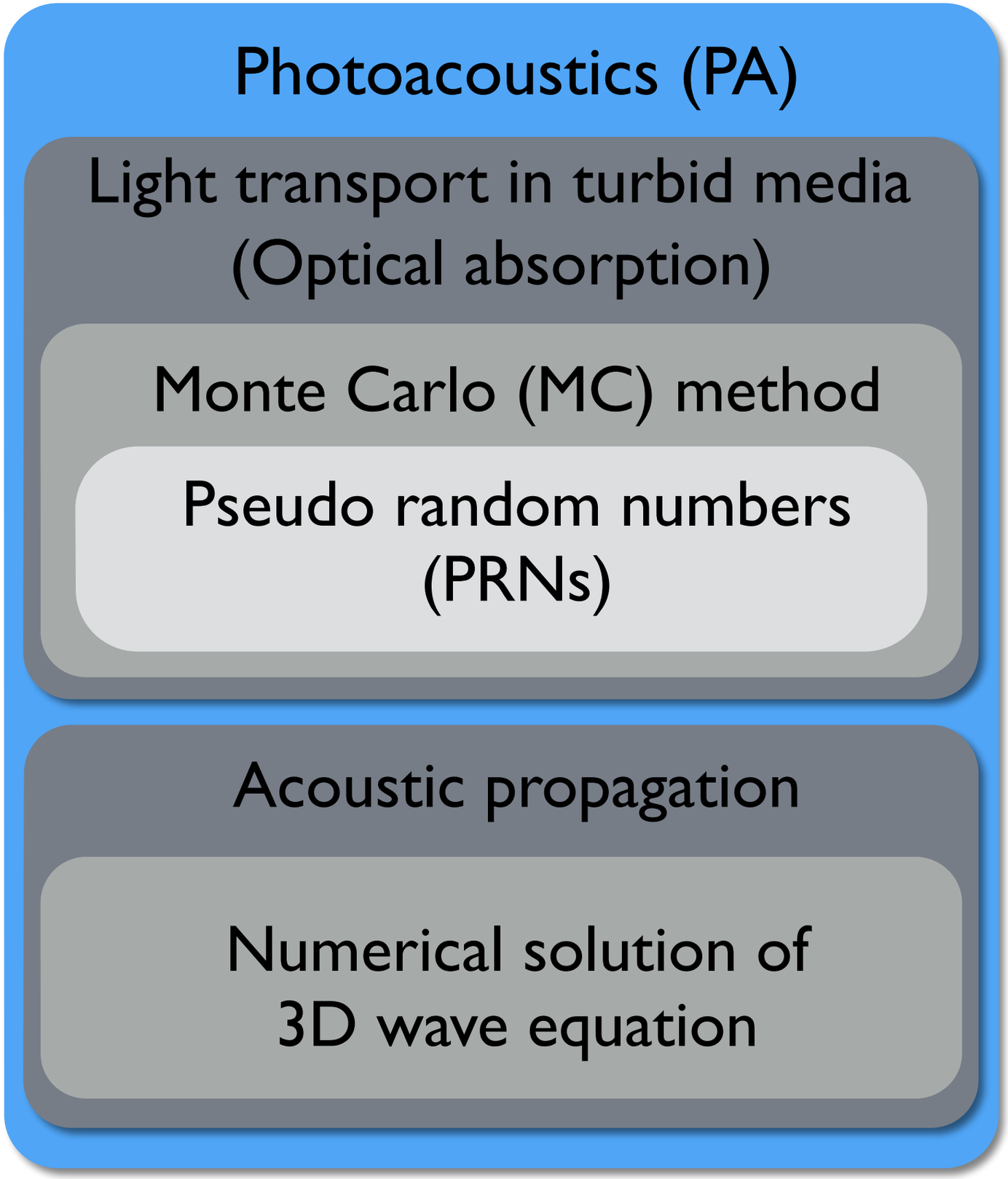
Top-down approach:

- to understand light transport in general turbid media you need to *master the MC method*
- to master MC method you need to *sample PRNs*
- to sample PRNs you need to actually *code*

Bottom-up approach:

- Lecture 1:
  PRN generation / quality control
- Lecture 2:
  MC sampling strategies
- Lecture 3:
  MC simulation in turbid media
- Exercise 1:
  extensive Python *code examples*

Lecture focus on tissue optics: outline



Optical Technologies.

Top-down approach:

The photoacoustic effect requires:

- understanding
  optical absorption
- understanding
  acoustic propagation

excellent testbed for *reductionism*

Bottom-up approach:

- Lecture 4:
  PA stress field propagation
- Exercise 2:
  extensive Python *code examples*
- Lecture 5 (BONUS):
  Finite-difference scheme 1D WE

+ *best practices for scientific software engineering and computing!*

Lecture focus on tissue optics: outline

Photoacoustics (PA)

Light transport in turbid media (Optical absorption)

Monte Carlo (MC) method

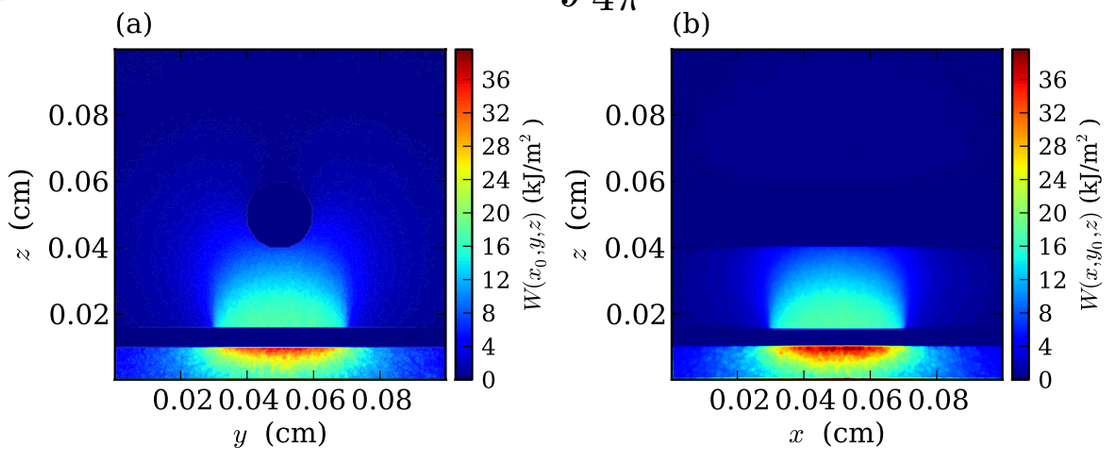
Pseudo random numbers (PRNs)

Acoustic propagation

Numerical solution of 3D wave equation

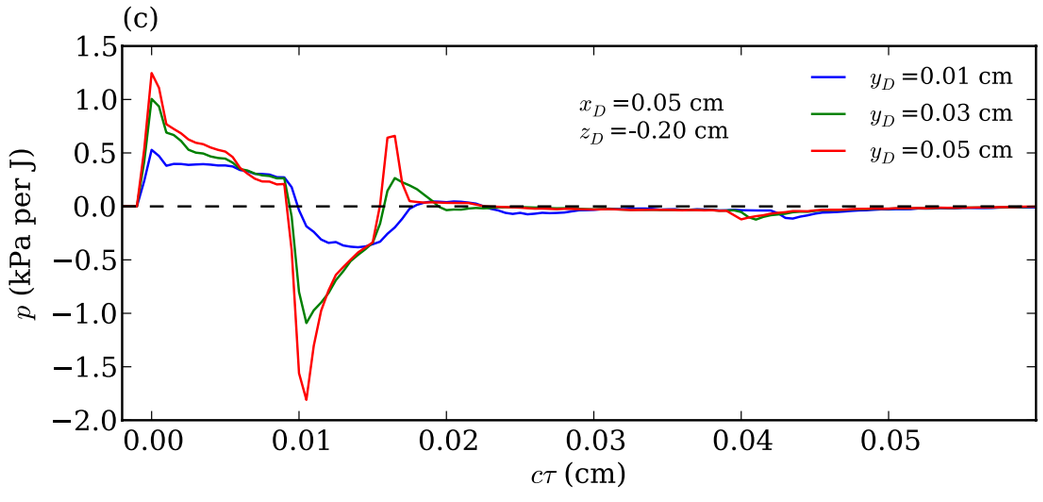
Optical absorption:

$$\mathbf{s}\ \nabla I(\mathbf{r},\mathbf{s}) = -(\mu_a + \mu_s) I(\mathbf{r},\mathbf{s}) + \mu_s \int_{4\pi} p(\mathbf{s},\mathbf{s}') I(\mathbf{r},\mathbf{s}')\ \mathrm{d}\omega'$$



Acoustic propagation:

$$\left[\frac{1}{v^2}\frac{\partial^2}{\partial t^2} - \nabla^2\right] p = \left[c_{\mathrm{th}}\frac{\partial}{\partial t} - c_{\mathrm{el}}\frac{\partial^2}{\partial t^2}\right] I$$



Optical Technologies.

Divide each difficulty into as many parts as is feasible and necessary to resolve it.

R. Descartes

Optical Technologies.

Lecture 1

Random Variables

Outline

Optical Technologies.

- Motivation - Why computer simulations?
- Lecture focus on tissue optics: outline

Lecture 1: Random Variables (RVs)

1.1 - Distribution of RVs

1.2 - Basic statistical summary measures

1.3 - Pseudo Random Number Generators (PRNGs)

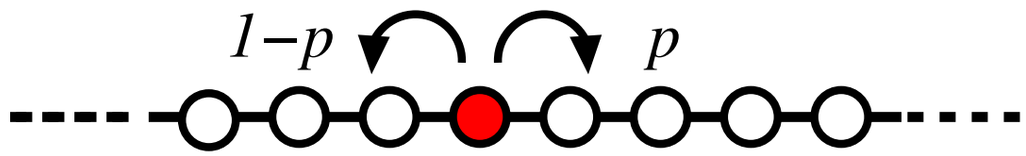
1.4 - Sampling nonuniform random numbers

1.1 Distribution of Random Variables

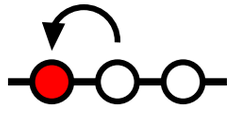
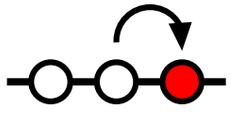
Optical Technologies.

Random experiment:

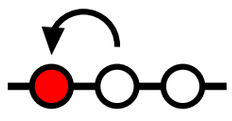
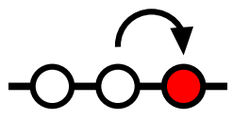
- outcome is not predictable
- e.g.: 1*D* random walk:



Sample space Ω:

- set of elementary events
- e.g.: 1*D* random walk: $\Omega = \{$  ,  $\}$

Random variable (RV):

- function $X : \Omega \to \mathbb{R}$ that relates a numerical value $x = X(\omega)$ to each elementary event $\omega \in \Omega$
- e.g.: 1*D* random walk: $X($  $) = -1$, $X($  $) = 1$

1.1 Distribution of Random Variables

Optical Technologies.

Combination of several RVs

- new RV $Y = f(X^{(0)}, \ldots, X^{(k)})$
- use outcomes $x^{(i)}$ to yield $y = f(x^{(0)}, \ldots, x^{(k)})$

Example: symmetric $1D$ random walk starting at $x_0 = 0$

- probability to step right: $p = 0.5$
- determine endposition x_N after N steps
- random experiment: take one step, repeat N-times
- new random variable $Y = \sum_{i=0}^{N-1} X^{(i)}$ yields endposition $x_N = \sum_{i=0}^{N-1} x^{(i)}$

Relevance of the random walk model:

- continuum limit yields diffusion equation
- simplified model for polymers

1.1 Distribution of Random Variables

Optical Technologies.

Probability function P:

- $P(X = x)$ signifies probability to observe RV with value x

Probability mass function (PMF):

- $p_x : \mathbb{R} \to [0, 1]$, where $p_X(x) = P(X = x)$
- description of discrete RV:
  - map numerical values to probabilities
  - discrete state space: $p_X(x) = 0$ except for finite set $\{x_i\}$
  - normalized: $\sum_{x_i} p_X(x_i) = 1$

Cumulative distribution function (CDF):

- $F_X : \mathbb{R} \to [0, 1]$, where $F_X(x) = P(X \leq x)$
- properties:
  - non-decreasing: if $x_1 < x_2$, then $F_X(x_1) \leq F_X(x_2)$
  - normalized: $\lim_{x \to -\infty} F_X(x) = 0$, $\lim_{x \to \infty} F_X(x) = 1$
  - relation to PMF: $F_X(x) = \sum_{x_i < x} p_X(x_i)$

1.2 Basic summary measures

Optical Technologies.

Features of a distribution function:

- moments of the distribution

$$E[X^k] = \begin{cases} \sum_i x_i^k \; p_X(x_i), & \text{for } X \textit{ discrete}, \\ \int_{-\infty}^{\infty} x^k \; p_X(x) \; dx, & \text{for } X \textit{ continuous}. \end{cases}$$

  $E[\cdot]$ signifies the *expectation operator*.

- Basic parameters related to a finite dataset:

  $\mathrm{av}(x) = \frac{1}{N} \sum_{i=0}^{N-1} x_i$ (average/mean value)
  - central tendency of sample

  $\mathrm{Var}(x) = \frac{1}{N-1} \sum_{i=0}^{N-1} [x_i - av(x)]^2$ (corrected variance)
  - unbiased estimator for the spread of the $x_i \in x$
  - proper implementation: corrected two-pass algorithm

  $\mathrm{sDev}(x) = \sqrt{\mathrm{Var}(x)}$ (standard deviation)

  $\mathrm{sErr}(x) = \frac{1}{\sqrt{N}} \mathrm{sDev}(x)$ (standard error)
  - signifies how accurate sample mean approximates the true mean

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

Why pseudo … what about true ones?

1.3.1 True random numbers

- Oldest true random number generator (TRNG):

  Coin tossing

  - random bit generator: 0 (head), 1(tail)
  - 32-bit integer needs 32 coin tosses
  - unbiased if coin is fair (1:1 head-to-tail ratio)

[http://imperialcoins.com]

- TRNG based on atmospheric noise:

  Random.org

  - rate: 3000 bits per second
  - limited quota: 1000000 free bits per day

[https://www.random.org]

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

Why pseudo … what about true ones?

1.3.1 True random numbers

- TRNG exploiting the randomness of quantum physics:

Quantis QRNG

  - „delivering true randomness with quantum random number generation“
  - hardware device with rate up to 16 Mbits/sec [http://www.idquantique.com]

Pros:

- generate true RNs
- yield uncorrelated RN sequences
- RN sequence not reproducible on purpose

Cons:

- may require hardware
- generally slow
- hinder debugging of code since RN sequence not reproducible

1.4 Sampling nonuniform random numbers

Optical Technologies.

1.3.2 Pseudo random numbers

Pros:

- fast (based on algorithms)
- portable (no additional hardware)
- RN sequence reproducible (facilitates proper debugging)

Cons:

- finite sequence length (numbers may repeat)
- successive RNs might be correlated

Know your PRNG's limits: How long is the period? […]
Are there correlations at any time during the sequence?

H. G. Katzgraber

Testing PRNGs is of paramount importance before use in scientific applications can even be considered.

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

1.3.3 Types of PRNGs

Algorithmic principle behind PRNGs:

$$x_{i+1} = f(x_i, x_{i-1}, \ldots, x_{i-n})$$

- seed block: $x_i, x_{i-1}, \ldots, x_{i-n}$ (needed to start the recurrence)
- problem: find f that yields x_{i+1} behaving as random as possible

Linear congruential generators (LCGs):

Deterministic algorithmic procedure of the form:

$$x_{i+1} = (a * x_i + c) \bmod m$$

- defining parameters:

  a (int): multiplier

  c (int): increment

  m (int): modulus (defines the period of the PRNG)

Optical Technologies.

You learn to be a better programmer by reading (*wading through*) other people's code.

R. C. Martin

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

Code Listing 0: **`python`** implementation of LCG

```
1 def LCG(a,c,m,x0):
2     """linear congruential generator
3     """
4     x = x0*m
5     while 1:
6         x = (a*x + c)%(m)
7         yield float(x)/m
```

- Note: for our exemplary use cases correct only if `maxint` $\geq 2^{49} - 1$

  [Park, S. K. and Miller, K.W., Commun. ACM, 31 (1988) 1192]

```
>>> import sys
>>> sys.maxint >= 2**49 - 1
True
```

Best practice 1: Write programs for people, not computers.
Understandability is key!

Best practice 2: Extensively document design and purpose.
(see module `linearCongruentialGenerator.py`)

[Wilson, G. and many others, PLOS Biology, 12 (2014) e1001745, open access, i.e. FREELY AVAILABLE ONLINE!]

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

```
 9 def LCG(a,c,m,x0):
10     """ linear congruential generator
11
12     Implements linear congruential generator following Ref. [1]
13
14     Args:
15         a (int): multiplier
16         c (int): increment
17         m (int): modulus
18         x0 (float): starging value in range [0,1)
19
20     Returns:
21         nothing, used as generator
22
23     Notes:
24         The starting value in Ref. [1] is taken as a positive integer and the
25         resulting sequence of random numbers is divided by the modulus m in
26         order to obain numbers in the interval [0,1). In contrast, the given
27         implementation expects a starting value from the range [0,1) and
28         returns already scaled samples.
29
30         Subsequent pseudo random samples can be obtained by calling the next()
31         method of the generator.
32
33     Refs:
34         [1] Random Number Generators
35             Hull, T.E. and Dobell, A.R.
36             SIAM Review, 4 (1962) 230
37     """
```

(see?)

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

Example:

Code Listing 1: sample RNs

```
from linearCongruentialGenerator import LCG

def main():
    # PARAMETER SETTING DEFINING RANDU GENERATOR
    N = int(sys.argv[1])
    s = float(sys.argv[2])
    a = 16807
    c = 0
    m = 2147483647

    # INITIALIZE INSTANCE OF PRNG
    r = LCG(a, c, m, s)

    # SAMPLE RNs
    for i in range(N):
        print r.next()

main()
```

Test output for given N, s:

```
0.494169999999
0.515189983936
0.79806001869
0.994734120076
0.496356120604
0.257318995245
0.76035308627
0.254320940925
0.372054130169
0.113765744127
0.0608615428311
0.899950361986
0.465733897896
0.589621943627
0.776006540743
0.341930269994
0.822047790353
0.157212465889
0.269914199473
0.447950539377
```

- observation: different seed, different sequence

Looks random … but is it?

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

1.3.4 Quality control - A simple PRNG test

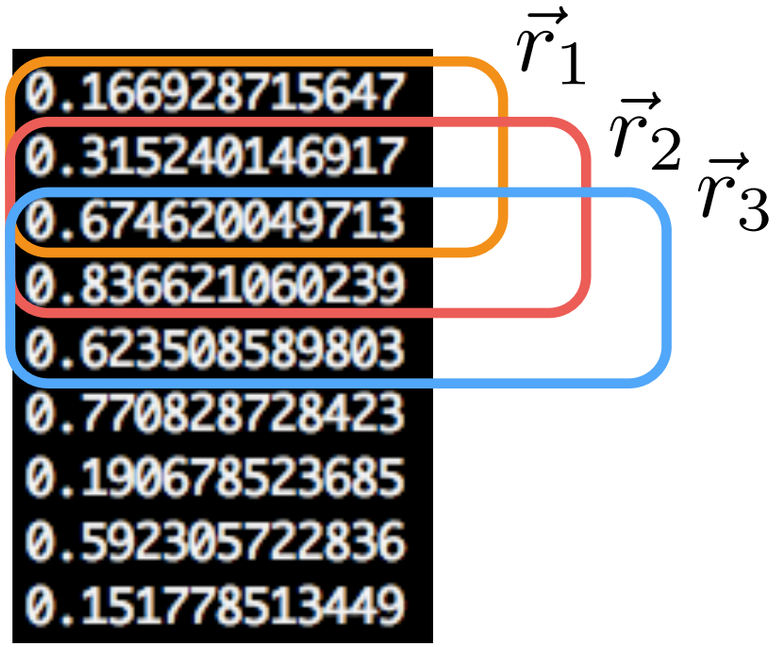
Spectral test:

- group stream of PRNs into *k*-tuples
- plot *k*-tuples (*k*=2, 3)
- check for spatial correlations

Note: RNs obtained via LCGs fall mainly in the planes. (That's bad!)

[Marsaglia, G., Proc. N.A. S., 61 (1968) 25]

Example considering *k*=3:



1.3 Pseudo random number generators (PRNGs)

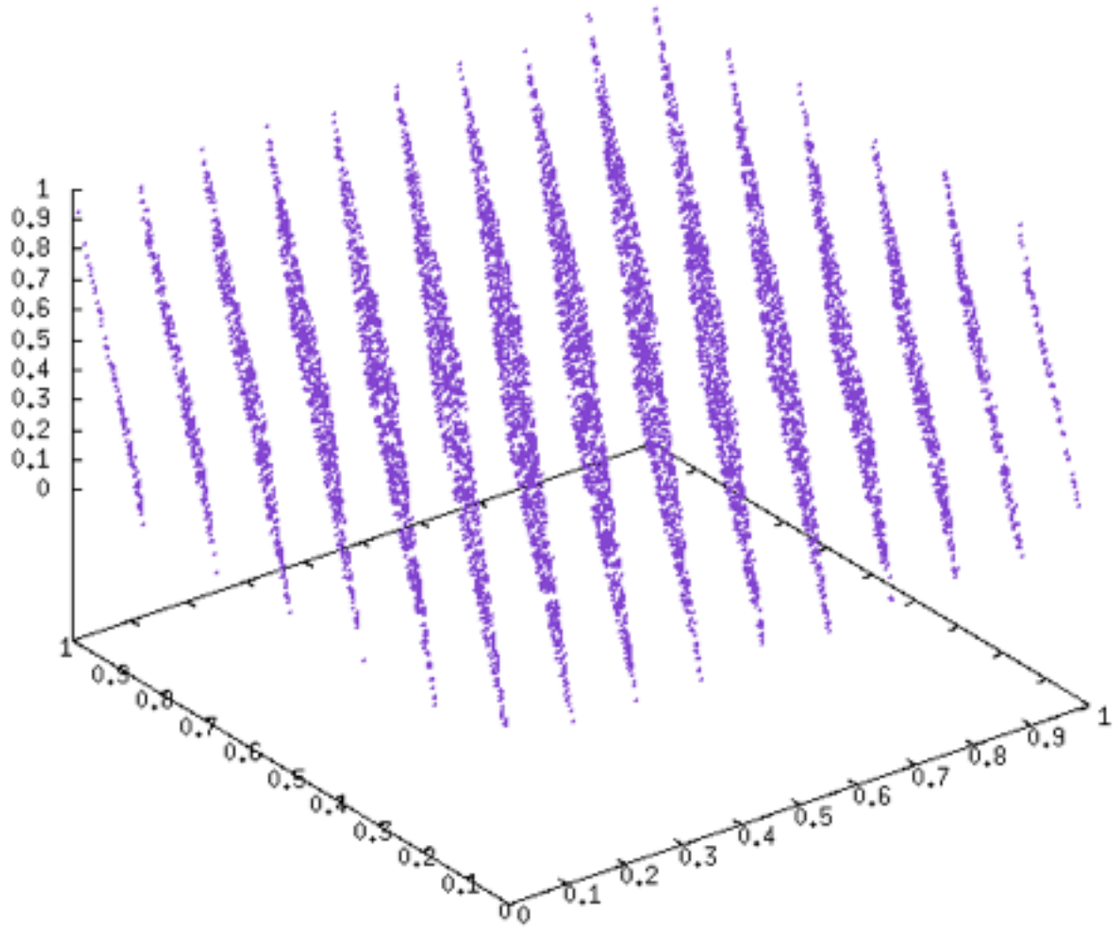
Optical Technologies.

Test 1 (using k=3): Park-Miller type generator (c=0) RANDU

Code Listing 2: RANDU

```
14 from linearCongruentialGenerator import LCG
15
16 def main():
17     # PARAMETER SETTING DEFINING RANDU GENERATOR
18     a = 65539
19     c = 0
20     m = 2**31
21     s = 323.
22
23     # INITIALIZE INSTANCE OF PRNG
24     r = LCG(a, c, m, s/m)
25
26     # SAMPLE RN TRIPLETS
27     r1, r2, r3 = r.next(), r.next(), r.next()
28     for i in range(10000):
29         print r1, r2, r3
30         r1, r2, r3 = r2, r3, r.next()
31
32 main()
```

3-tuple viewed from proper angle (using `gnuplot`) [www.gnuplot.org]



- standard generator on IBM machines during 1960's
- example of a very, very bad generator

1.3 Pseudo random number generators (PRNGs)

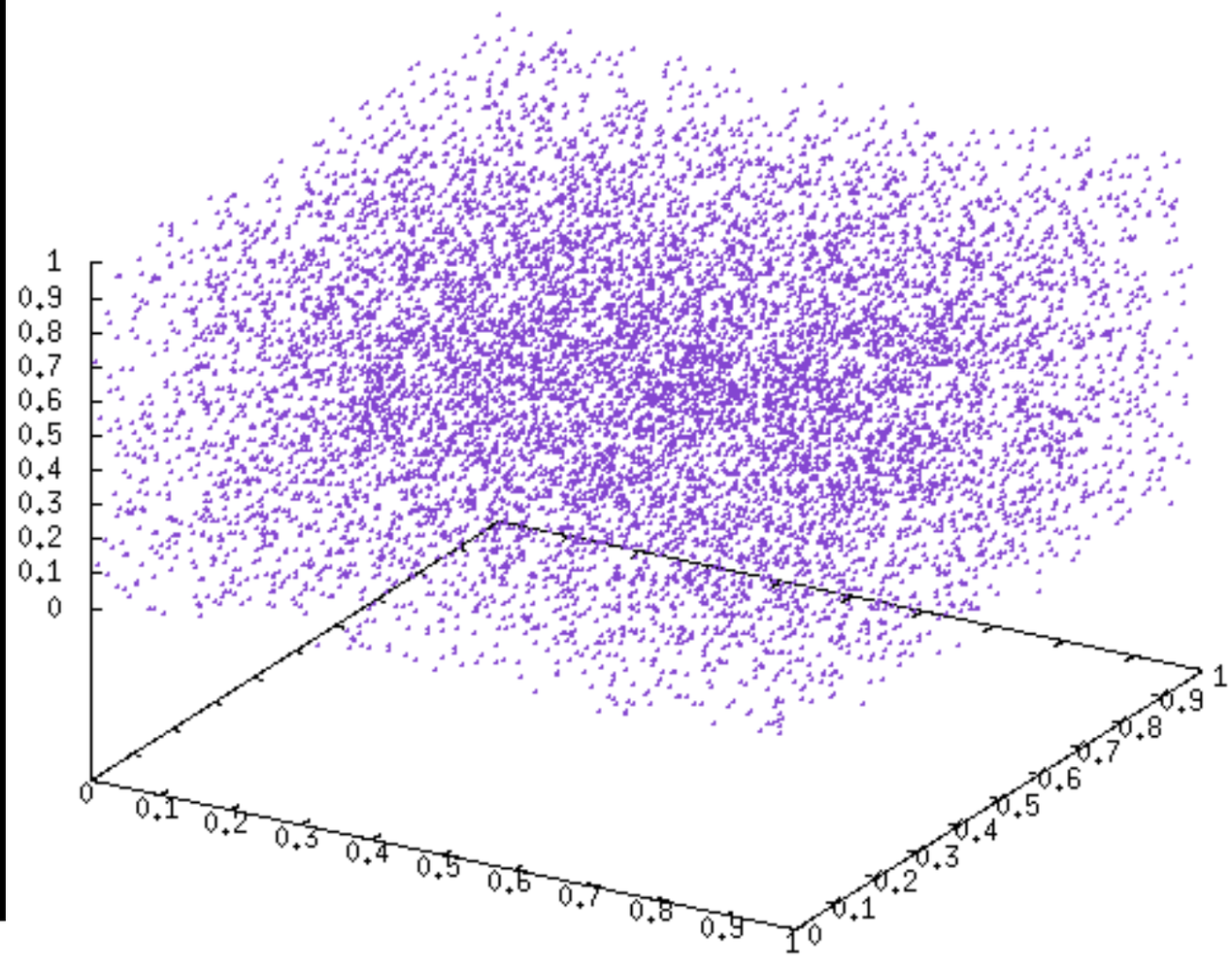
Optical Technologies.

Test 2 (using *k*=3): The „minimal standard“

Code Listing 3: Minimal Standard

```
23 from linearCongruentialGenerator import LCG
24
25 def main():
26     # PARAMETER SETTING DEFINING MINIMAL STANDARD
27     a = 16807
28     c = 0
29     m = 2147483647
30     s = 323.
31
32     # INITIALIZE INSTANCE OF PRNG
33     r = LCG(a, c, m, s/m)
34
35     # SAMPLE RN TRIPLETS
36     r1, r2, r3 = r.next(), r.next(), r.next()
37     for i in range(10000):
38         print r1, r2, r3
39         r1, r2, r3 = r2, r3, r.next()
40
41 main()
```

3-tuple viewed via `gnuplot`



- Park-Miller type prime modulus linear congruential generator (PMMLCG)

1.3 Pseudo random number generators (PRNGs)

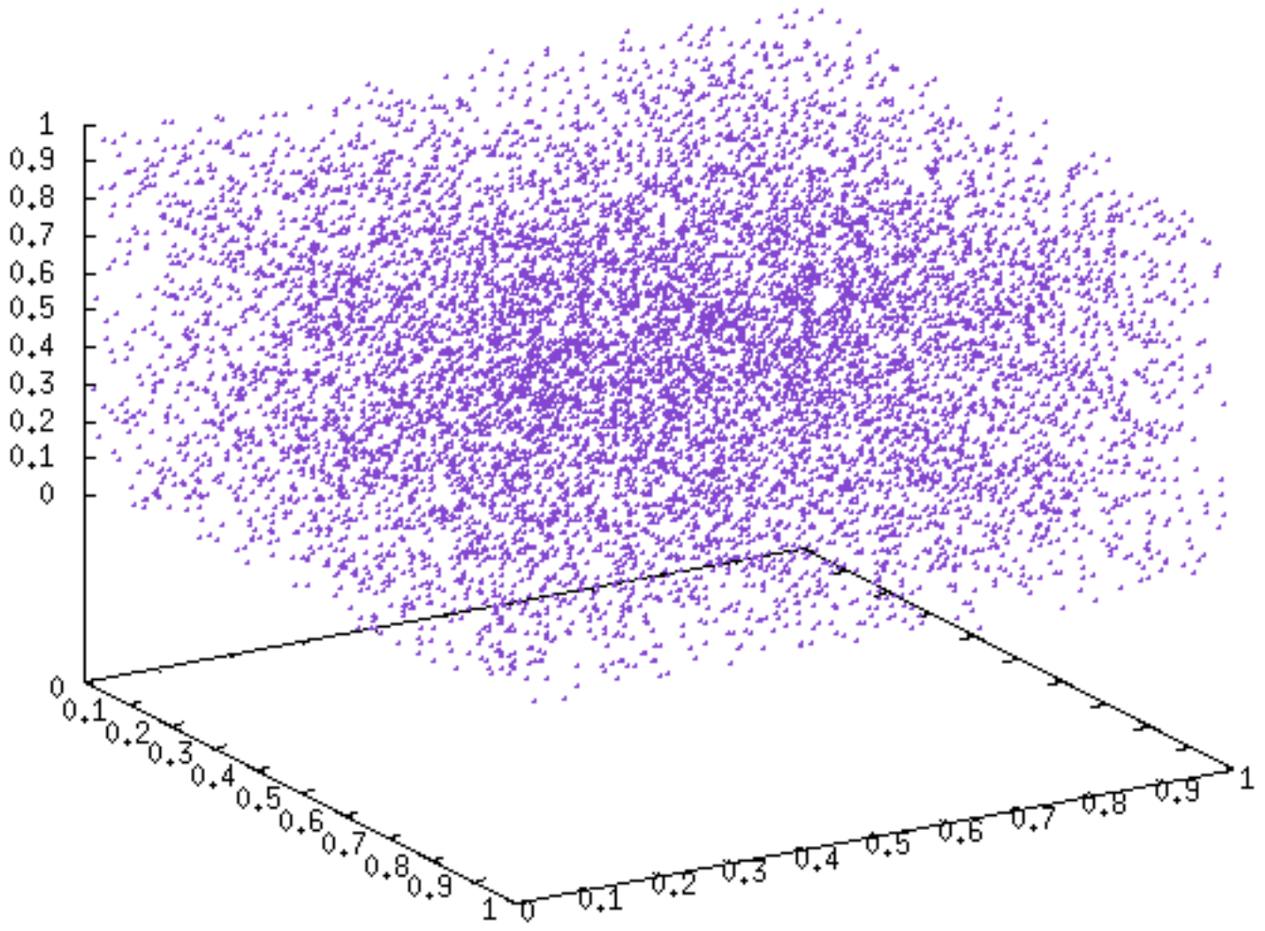
Optical Technologies.

Test 3 (using k=3): Mersenne Twister (MT; **`python`** standard)

Code Listing 4: Mersenne Twister

```
12 import random
13
14 def main():
15     random.seed(0)
16     r = random.random
17
18     r1, r2, r3 = r(), r(), r()
19     for i in range(10000):
20         print r1, r2, r3
21         r1, r2, r3 = r2, r3, r()
22
23 main()
```

3-tuple viewed via `gnuplot`



- very fast
- extremely large period: $m = 2^{19937} - 1$ (i.e. a Mersenne prime)
- ridiculously small correlations

1.3 Pseudo random number generators (PRNGs)

Optical Technologies.

1.3.5 Elaborate test suites

- Old battery of statistical tests (1995) for PRNGs:

  Diehard

  - reads in sequences of RNs and performs tests
  - contains 16 standard tests
  - more recent version: *DieHarder*

    [http://www.phy.duke.edu/~rgb/General/dieharder.php]

- US National Institute of Standards and Technology (NIST):

  NIST Statistical Test Suite

  - contains 15 up-to-date tests and elaborate discussion

    [http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html]

Optical Technologies.

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.

J. von Neumann

1.4 Sampling nonuniform RNs

Optical Technologies.

- we know how to sample uniform PRNs in [0,1)
- but: MC for light transport requires random 3D orientations
- central issue is thus:

How to sample from the surface of the unit 3-sphere?

Why … at all?

after scattering: new propagation direction

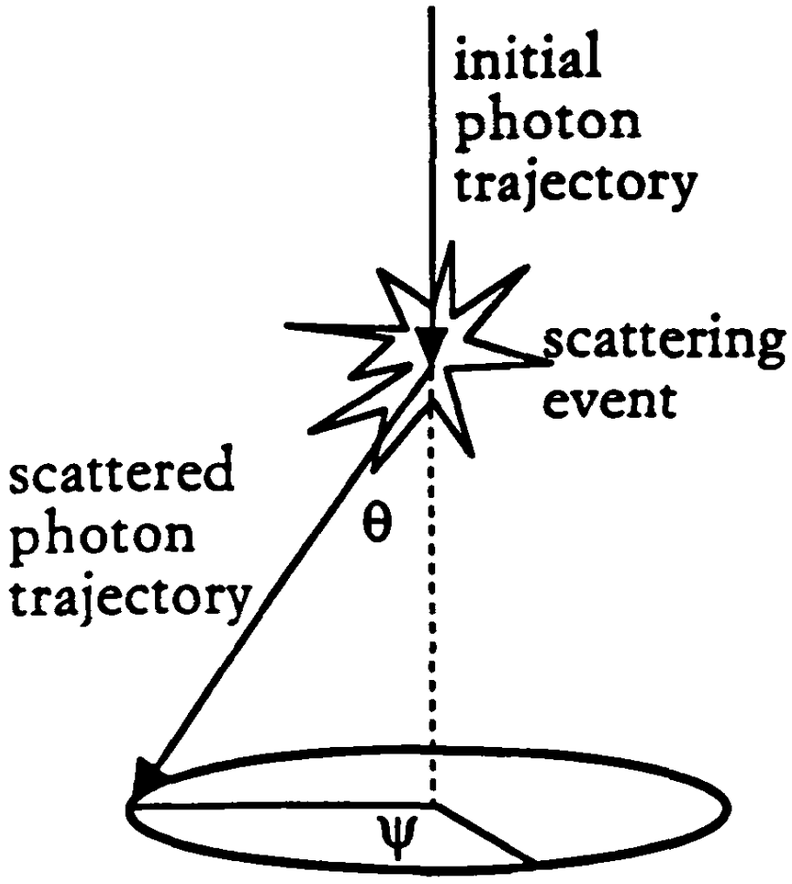


Figure 4.1. Deflection of a photon by a scattering event. The angle of deflection, θ, and the azimuthal angle, ψ, are indicated.

[Jacques, S. L. and Wang, L., in *Optical-Thermal Response of Laser-Irradiated Tissue*, Plenum Press, (1995)]

1.4 Sampling nonuniform RNs

- we know how to sample uniform PRNs in [0,1)
- but: MC for light transport requires random 3D orientations
- central issue is thus:

How to sample from the surface of the unit 3-sphere?

1.4.1 Uniform sampling from surface of unit 3-sphere

Generate unit vectors via *acceptance-rejection method*:

- generate trial point, accept if it can be projected to desired surface
- way more efficient than naive but more intuitive methods (see Exercises)
- efficiency: $\epsilon_{\text{samp}} = \frac{V_{\text{circle}}}{V_{\text{square}}} = \frac{\pi}{4}$

  (on average 2.55 RNs per unit vector)

Code Listing 5: 3-sphere sampler

```
2  class UnitSphere(object):
3
4      def __init__(self,r=random.random):
5          self.r = r
6
7      def generate(self):
8          x1 = x2 = 1.
9          while(x1*x1 + x2*x2 >= 1):
10             x1 = 2.*self.r() - 1.
11             x2 = 2.*self.r() - 1.
12
13         t = x1*x1 + x2*x2
14         fac = sqrt(1.-t)
15         return 2.*x1*fac, 2.*x2*fac, 1.-2.*t
16
```

[Marsaglia, G., Ann. Math. Stat., 43 (1972) 645]

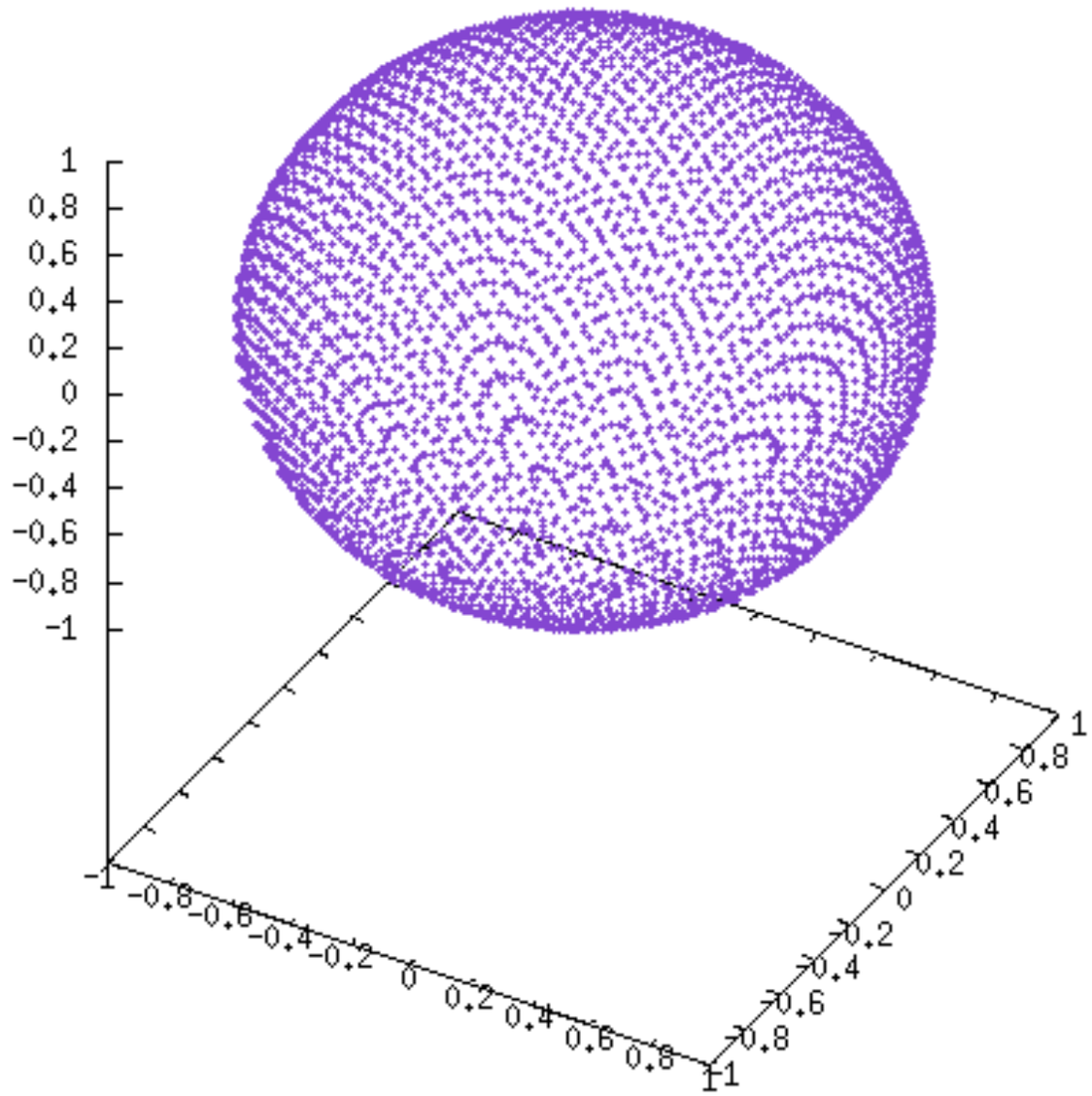
1.4 Sampling nonuniform RNs

Optical Technologies.

A graphical check of generated unit vectors:

Code Listing 6: sampling via bad LCG

```
11 from randomVariateGenerator import UnitSphere
12 from linearCongruentialGenerator import LCG
13 from math import pi
14
15 def main():
16     # SIMULATION PARAMETERS
17     N = int(sys.argv[1])
18     seed = 323.
19     a, c, m = 106, 1283, 6075
20
21     # INITIALIZE INSTANCE OF PRNG
22     r = LCG(a, c, m, seed/m)
23
24     # INITIALIZE INSTANCE OF UNIT SPHERE SAMPLER
25     mySamp = UnitSphere(r.next)
26
27     # SAMPLE RN TRIPLETS
28     for i in range(N):
29         x1,x2,x3 = mySamp.generate()
30         print x1,x2,x3
31
32 main()
```



(script: `main_sampleUnitVectors_BAD.py`)

Not random at all!

1.4 Sampling nonuniform RNs

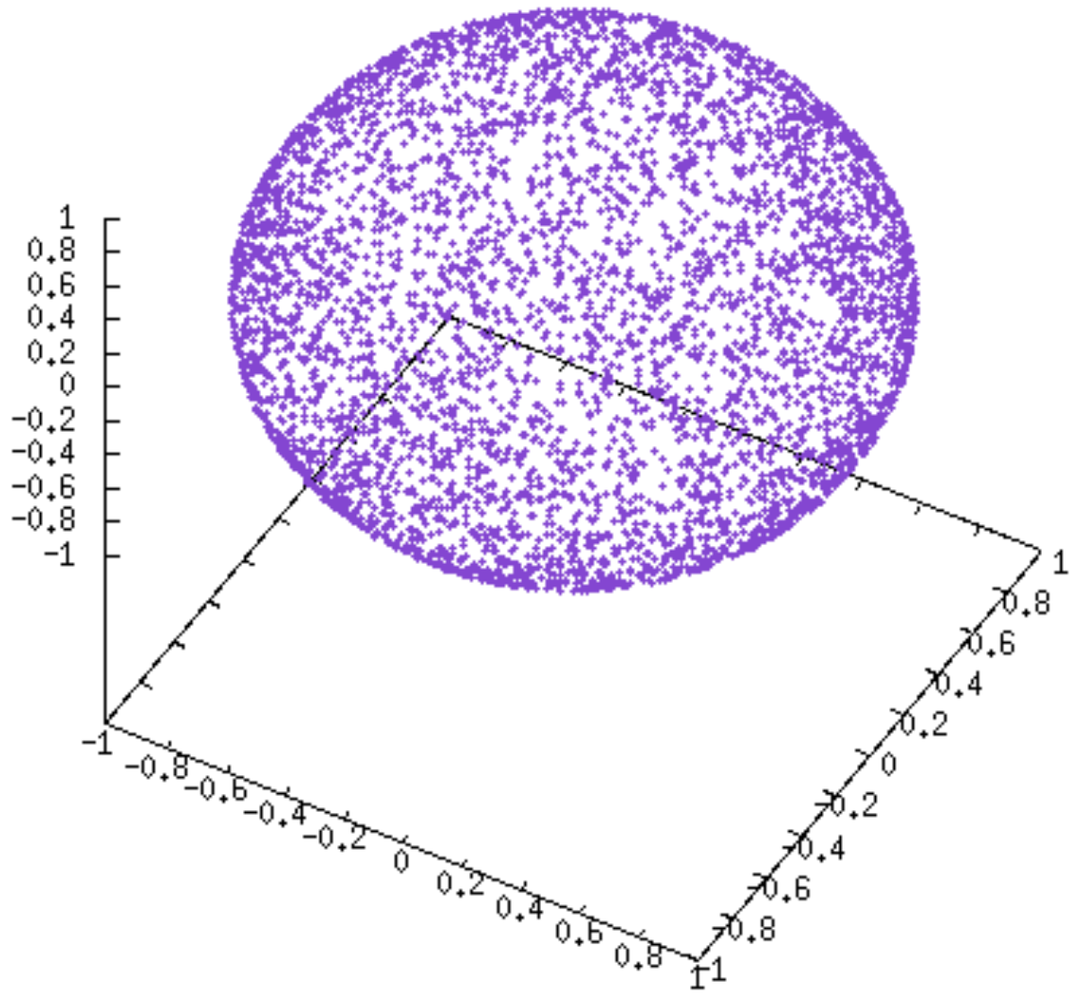
Optical Technologies.

A graphical check of generated unit vectors:

Code Listing 7: sampling via MT

```
11 from randomVariateGenerator import UnitSphere
12
13 def main():
14     # SIMULATION PARAMETERS
15     N = int(sys.argv[1])
16
17     # INITIALIZE INSTANCE OF UNIT SPHERE SAMPLER
18     mySamp = UnitSphere()
19
20     # SAMPLE RN TRIPLETS
21     for i in range(N):
22         x1,x2,x3 = mySamp.generate()
23         print x1,x2,x3
24
25 main()
```

(script: `main_sampleUnitVectors.py`)



Looks random … but is it?

1.4 Sampling nonuniform RNs (Quality control)

Optical Technologies.

How to check whether the unit vectors are truly random?

Null hypothesis: N given vectors are from a population uniformly distributed in $3D$

- find the length of the resultant vector R of the unit vectors $\vec{r}_i$:

$$R = |\sum_{i=1}^{N} \vec{r}_i| \quad \text{distributed following} \quad f(R) \approx \frac{3\sqrt{6}R^2}{\sqrt{\pi N^3}} \exp\{-3R^2/2N\}$$

- individual x_i, y_i, z_i are distributed normally as $N \to \infty$
- find reference χ^2 statistics for desired confidence interval $\alpha = 0.05$

$$R_0^2 = \frac{N}{3}\chi^2(\alpha = 0.05, \mathrm{dof} = 3)$$

- If $R > R_0$, reject null hypothesis at confidence interval α

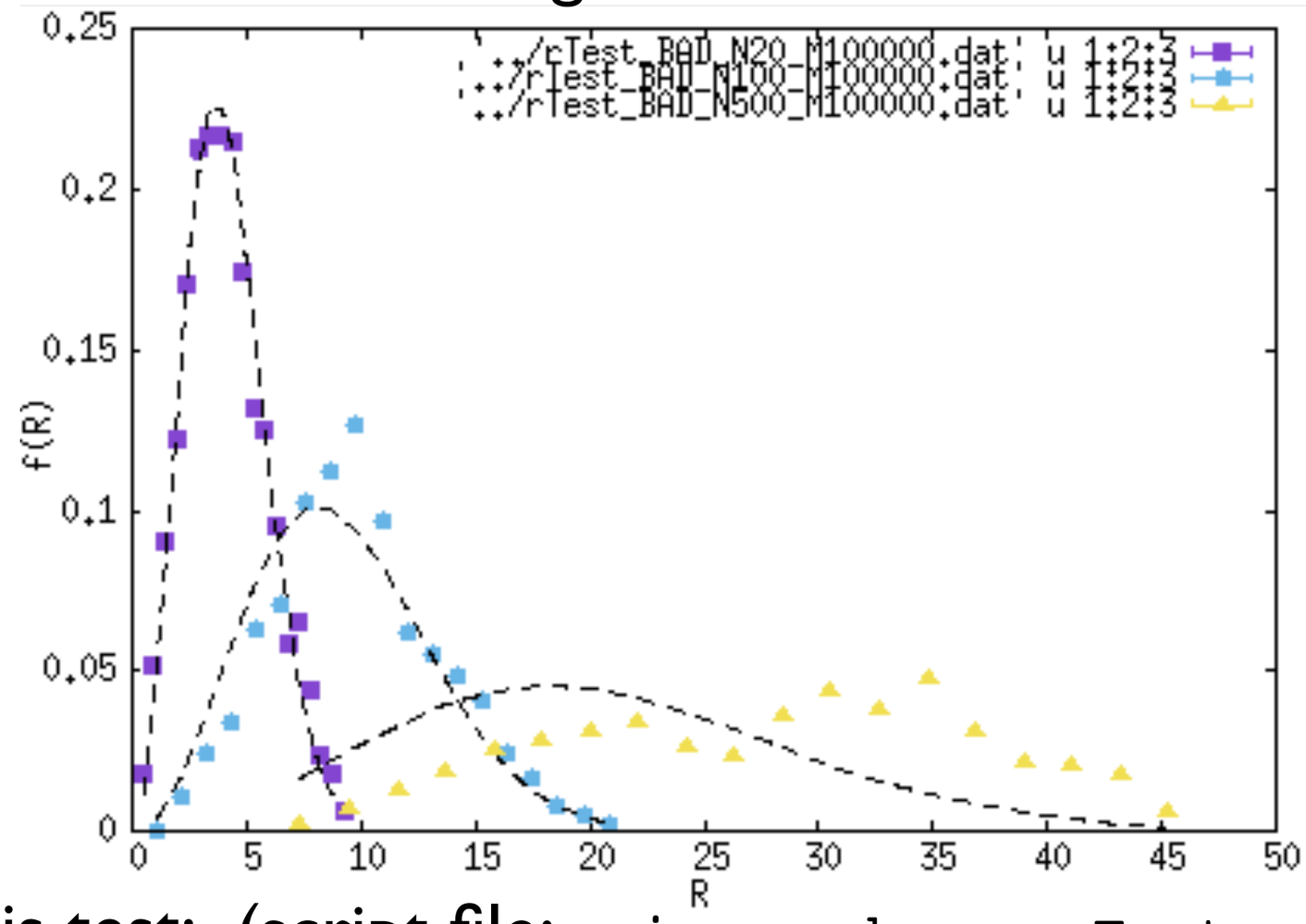
(see module file: `unitVectorTest.py`) [Stephens, M.A., J. Amer. Statist. Assoc., 59 (1964) 160]

1.4 Sampling nonuniform RNs

Optical Technologies.

Test null hypothesis for unit sphere sampler using LCG:

- distribution of resultant length:



- hypothesis test: (script file: `main_randomnessTest_unitVectors_BAD.py`)

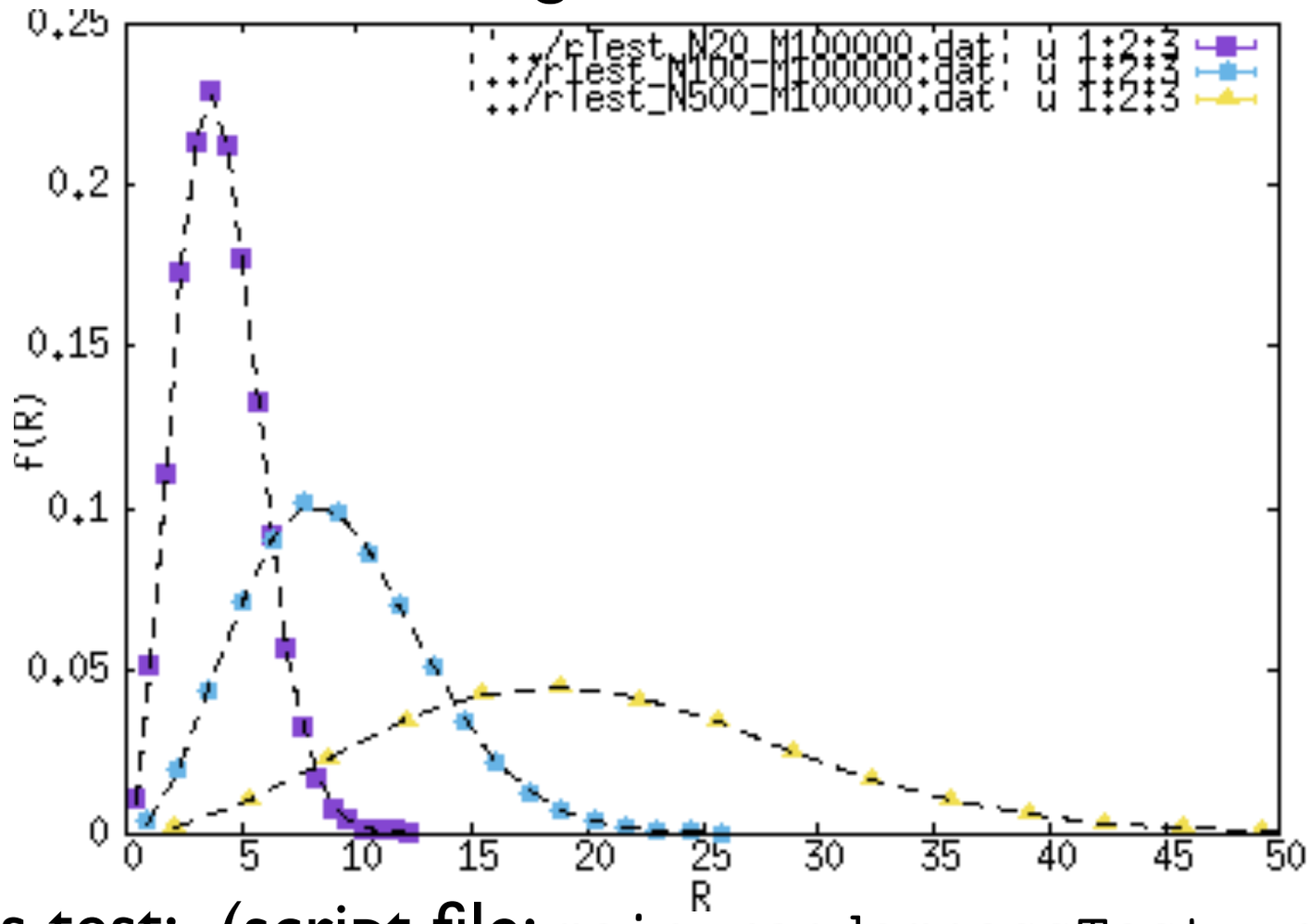
```
25 # TETS NULL HYPOTHESIS FOR SAMPLER: UnitSphere
26 # av(R) +- sErr(R) = 31.687298 +- 1.089658
27 # R0 = 30.468435
28 # REJECT NULL HYPOTHESIS (SIGNIFICANCE LEVEL 0.20): True
29 # TEST RESULT: FAILED
```

1.4 Sampling nonuniform RNs

Optical Technologies.

Test null hypothesis for unit sphere sampler using MT:

- distribution of resultant length:



- hypothesis test: (script file: `main_randomnessTest_unitVectors.py`)

```
25 # TETS NULL HYPOTHESIS FOR SAMPLER: UnitSphere
26 # av(R) +- sErr(R) = 22.463892 +- 0.919587
27 # R0 = 30.468435
28 # REJECT NULL HYPOTHESIS (SIGNIFICANCE LEVEL 0.20): False
29 # TEST RESULT: OK
```

1.4 Sampling nonuniform RNs

Optical Technologies.

- uniform sampling from unit 3-sphere = isotropic scattering
- however: most biological tissue exhibit non-isotropic scattering

1.4.2 Sampling from Henyey-Greenstein (HG) phase function

- commonly used single-scattering function for biological tissue
- analytic expression with only one parameter (conceptually simple):

pdf: $$p(\mu) = \frac{1}{2}\frac{(1-g^2)}{(1+g^2-2g\mu)^{3/2}}$$ where $$\mu = \cos(\theta)$$

$$\int_{-1}^{1} p(\mu)\ \mathrm{d}\mu = 1$$

cdf: $$F(\mu) = \frac{(1-g^2)}{2g}\left[(1+g^2-2g\mu)^{-1/2} - (1+g)^{-1}\right]$$

1.4 Sampling nonuniform RNs

Optical Technologies.

Efficient sampling possible via *transformation method*:

- Step 1: *invert* to get μ as function of g

$$\mu = \frac{1}{2g}\left[1 + g^2 - \left(\frac{1-g^2}{1+gu}\right)^2\right] \qquad \text{where} \quad u = 2F - 1$$

Note: inversion breaks down as $g \to 0$

Remedy: expansion in powers of g

$$\mu \approx u + \frac{3}{2}g(1-u^2) - 2g^2u(1-u^2) + \ldots$$

- Step 2: sample nonuniform HG random variates *directly*

Note: additional azimuthal angle sampled uniformly!

1.4 Sampling nonuniform RNs

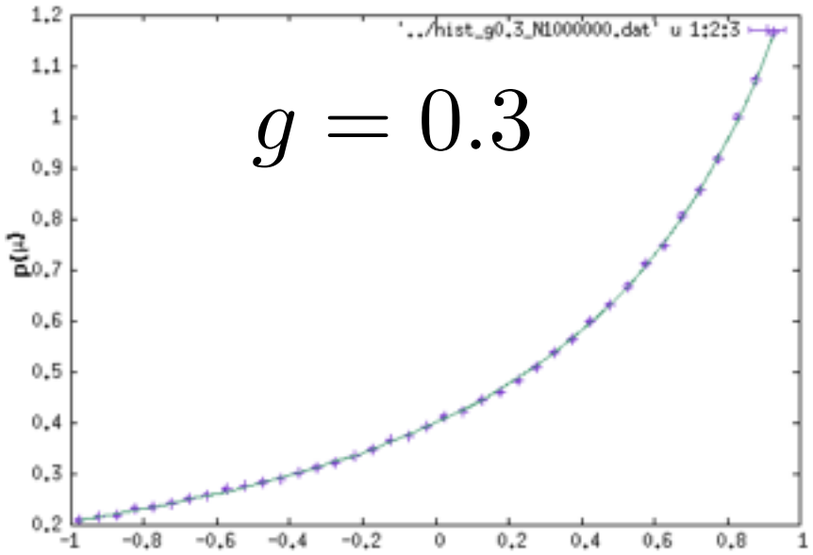
Code Listing 8: Henyey-Greenstein sampler

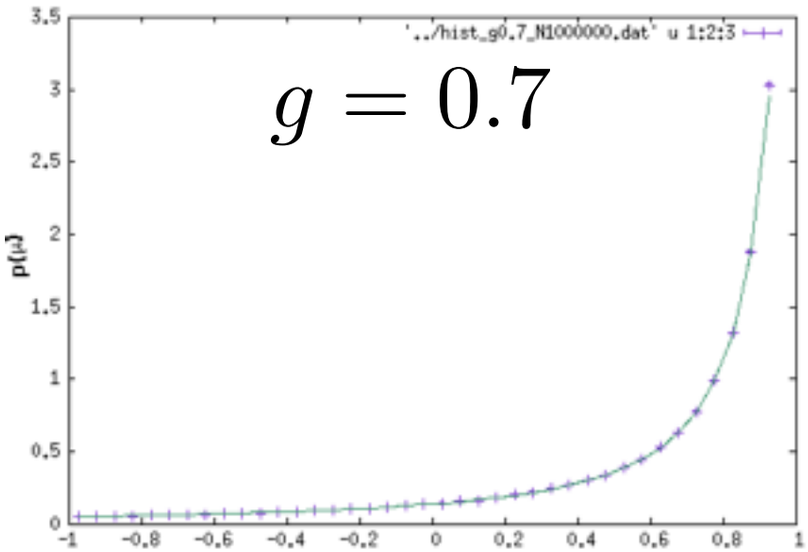
```
class HenyeyGreenstein(object):

    def __init__(self, g, r=random.random, MING=1e-3):
        self.g = g
        self.r = r
        self.MING = MING

    def generate(self):
        s = 2.*self.r()-1.
        g = self.g

        if abs(g) < self.MING:
            mu = s + 3.*g*(1-s*s)/2 - 2.*g*g*s*(1-s*s)
        else:
            mu = (1. + g*g - ((1.-g*g)/(1.+g*s))**2)/2./g

        fac = sqrt(1-mu*mu)
        phi = pi*(2*self.r()-1)
        cosPhi = cos(phi)
        sinPhi = sqrt(1.-cosPhi*cosPhi) if phi>=0 else -sqrt(1.-cosPhi*cosPhi)
        return fac*cosPhi, fac*sinPhi, mu
```

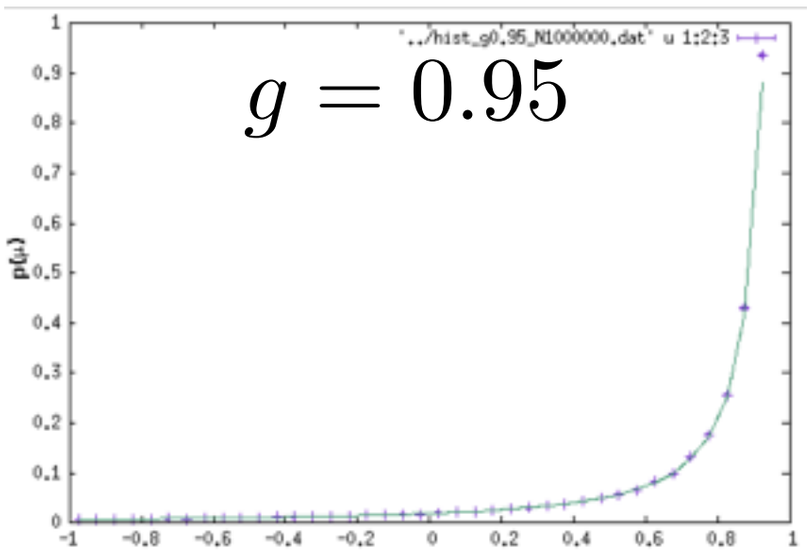
1.4 Sampling nonuniform RNs

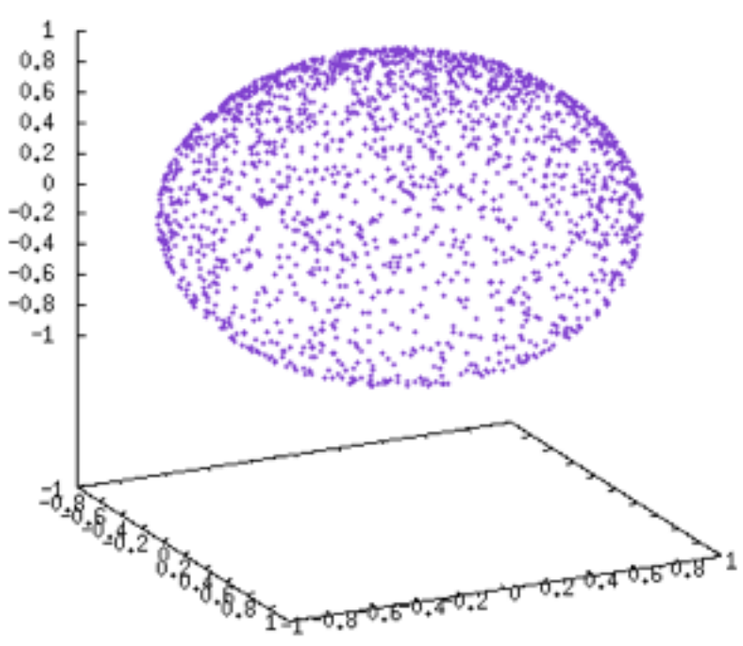
Optical Technologies.

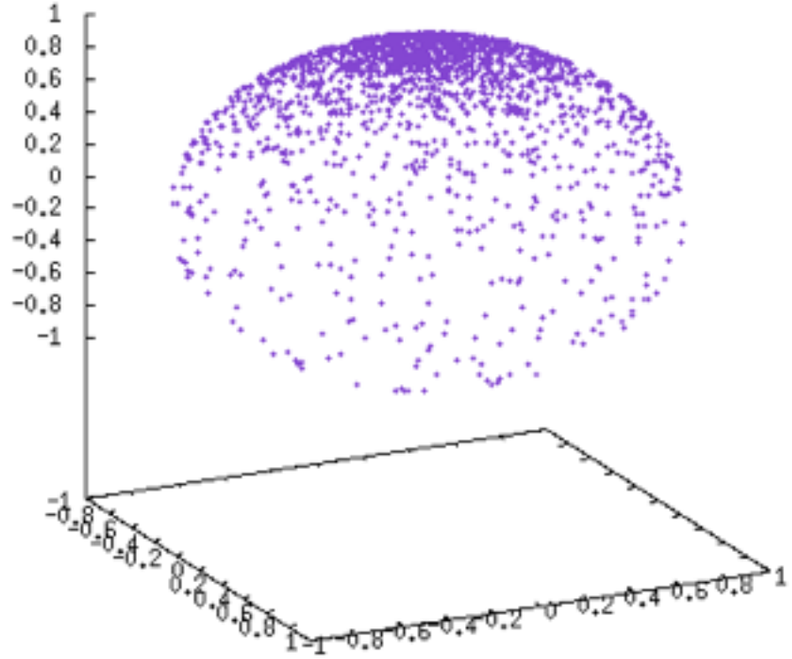
Exemplary simulations for different scattering anisotropy g:

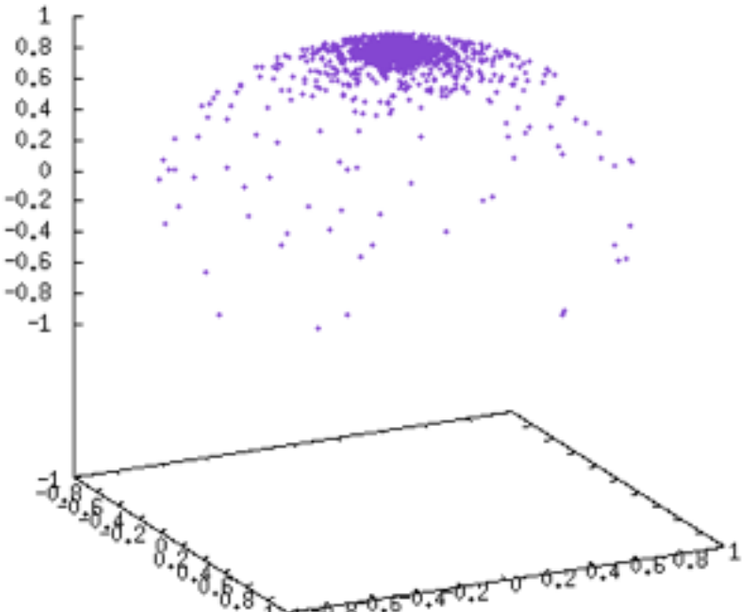












Looks reasonable … but is it?

1.4 Sampling nonuniform RNs (Quality control)

Optical Technologies.

How to check whether the unit vectors are drawn from HG phase function?

Null hypothesis: N given vectors are drawn from HG phase function at g

- compute approximate cdf from sample, compare to true cdf
- Kolmogorov Smirnov (KS) test

Code Listing 9: KS test (file: `main_HGPhaseFunction_KS.py`)

```
def F(g,u):
    return (1-g*g)*( 1./np.sqrt(1+g*g-2*g*u) - 1./(1+g)  )/2/g

def main():

    N = int(sys.argv[1])
    Dcrit = 1.628/np.sqrt(N)
    g, dg = 0.05, 0.05

    while g < 0.95:
        cdf = lambda x: F(g,x)
        myGen = HenyeyGreenstein(g)
        uList = sorted([myGen.generate()[2] for i in range(N)])

        D,p = scipy.stats.kstest(uList,cdf)
        print "g = %3.2lf, p-value = %4.3lf, Hypot. = %s  "%(g, p, D < Dcrit)

        g+=dg
```

Testing:

```
g = 0.05, p-value = 0.329, Hypot. = True
g = 0.10, p-value = 0.855, Hypot. = True
g = 0.15, p-value = 0.974, Hypot. = True
g = 0.20, p-value = 0.442, Hypot. = True
g = 0.25, p-value = 0.758, Hypot. = True
g = 0.30, p-value = 0.317, Hypot. = True
g = 0.35, p-value = 0.581, Hypot. = True
g = 0.40, p-value = 0.969, Hypot. = True
g = 0.45, p-value = 0.048, Hypot. = True
g = 0.50, p-value = 0.088, Hypot. = True
g = 0.55, p-value = 0.609, Hypot. = True
g = 0.60, p-value = 0.546, Hypot. = True
g = 0.65, p-value = 0.124, Hypot. = True
g = 0.70, p-value = 0.224, Hypot. = True
g = 0.75, p-value = 0.900, Hypot. = True
g = 0.80, p-value = 0.405, Hypot. = True
g = 0.85, p-value = 0.993, Hypot. = True
g = 0.90, p-value = 0.060, Hypot. = True
```

1.4 Sampling nonuniform RNs

Optical Technologies.

Notes on HG phase function:

- pro: simple expansion in Legendre polynomials (good for analytics)
- con: does not reduce to physically meaningful phase function for small particles illuminated by unpolarized light
- Remedy: more complex phase functions (see exercises)

After lecture: example programs available at

https://github.com/omelchert/CompTissueOpt-2017.git