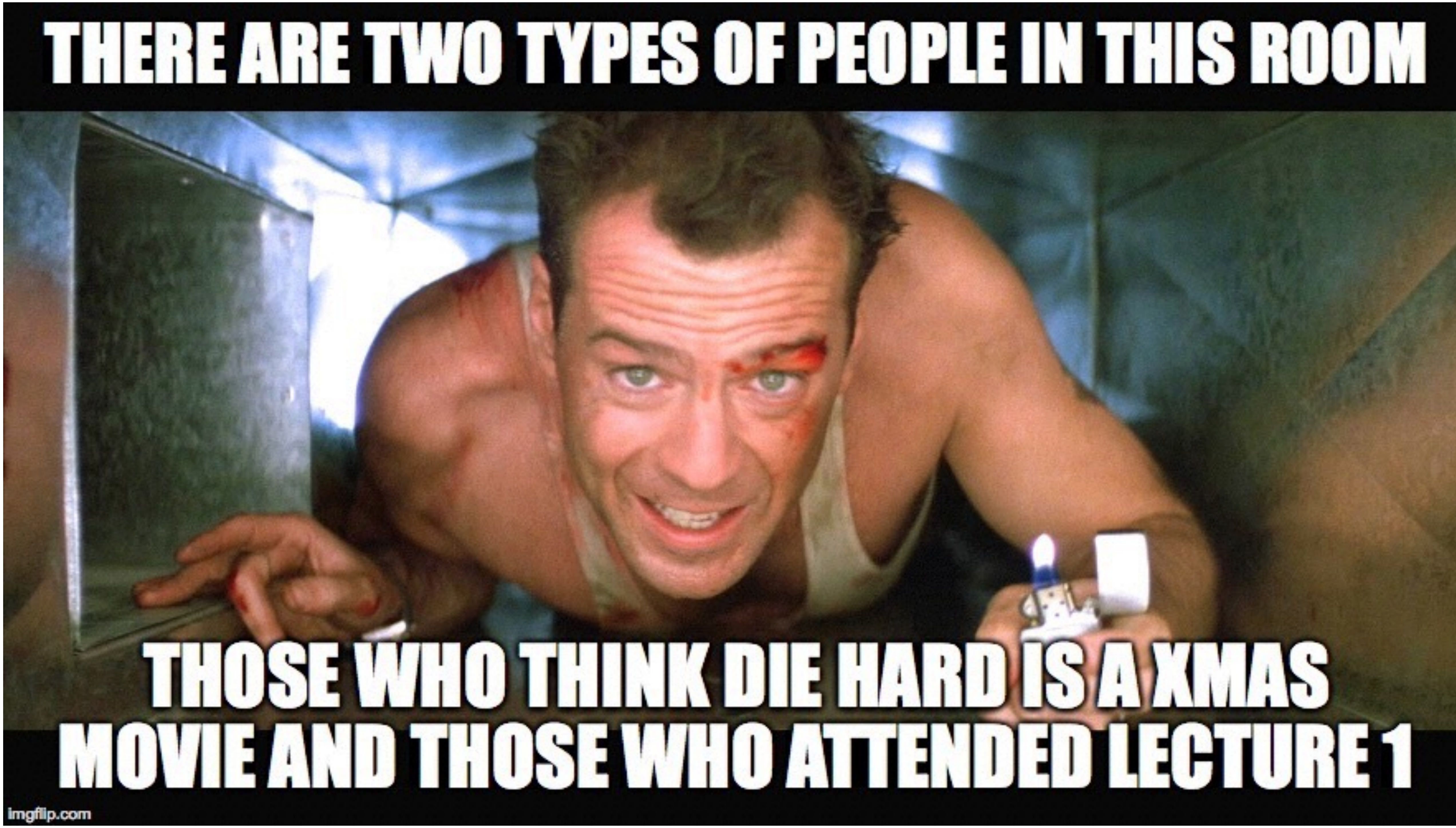
Recap — Lecture 1



Lecture focus on tissue optics: outline

Optical Technologies.

Computational tissue optics

Light transport in turbid media (Optical absorption)

Monte Carlo (MC) method

Pseudo random numbers (PRNs)

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 2

Monte Carlo (MC) Basics

Outline

- Recap - Lecture 1

Lecture 2: Monte Carlo (MC) Basics

2.1 - The Monte Carlo method

2.2 - Paradigmatic MC simulations

2.2.1 Estimating pi

2.2.2 The $1\mathrm{D}$ random walk

2.2.3 The $3\mathrm{D}$ random walk

Optical Technologies.

Number 247 SEPTEMBER 1949 *Volume 44*

THE MONTE CARLO METHOD

NICHOLAS METROPOLIS AND S. ULAM
Los Alamos Laboratory

We shall present here the motivation and a general description of a method dealing with a class of problems in mathematical physics. The method is, essentially, a statistical approach to the study of differential equations, or more generally, of integro-differential equations that occur in various branches of the natural sciences.

[Metropolis, N. and Ulam, S., J. Amer. Stat. Assoc., 44 (1949) 335]

Definition of GAME CHANGER

: a newly introduced element or factor that changes an existing situation or activity in a significant way

[https://www.merriam-webster.com/dictionary/gamechanger]

Interlude: goal-driven (software) development

Best practice 3:

Balance long-term wishes (top) with existing options (bottom)

- Model:

  specify the requirements of the model you are interested in

- Method:

  review and opt for viable numerical approach

- Mapping:

  assemble software that maps the requirements on the technical basis

3M procedure applies to all kinds of projects (software dev., thesis, …)

2.2 Paradigmatic MC simulations

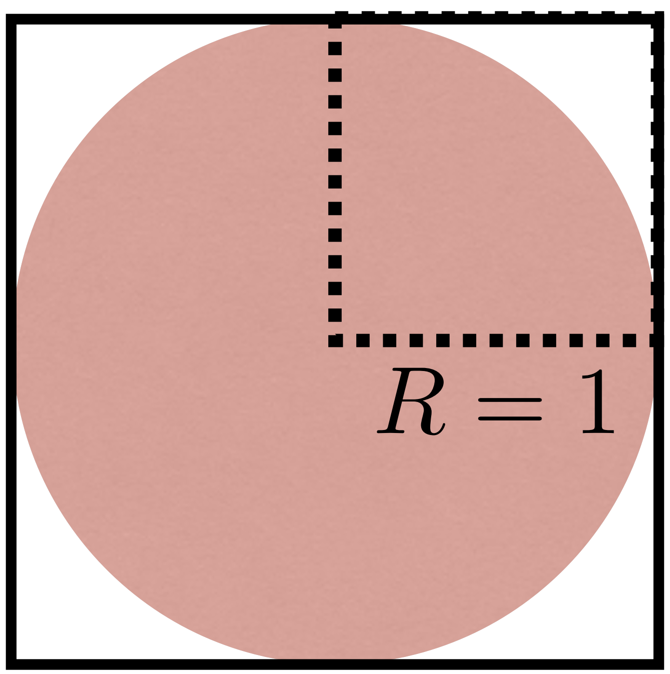
2.2.1 Estimating π

- **M**odel:

  obtain π by integration of unit circle

- **M**ethod:

  simple-sampling MC strategy: estimate π by uniformly sampling unit square



Idea: if

$$V_{\mathrm{sq}} \equiv n_{\mathrm{tries}}$$

then

$$V_{\mathrm{circ}} \equiv \frac{n_{\mathrm{succ}}}{n_{\mathrm{tries}}}$$

$$\pi \equiv 4 \times V_{\mathrm{circ}}$$

- **M**apping:

  see module `estimatePi.py`

Code Listing 1: simple sampling MC

```
def simpleSampling(r, nTries):
    """simple sampling estimator

    Implements simple-sampling Monte Carlo (MC)
    strategy to estimate pi

    Args:
        r (object) pseudo random number generator
        nTries (int) number of trial points

    Returns:
        pi (float) simple sampling estimate of pi
    """
    nSucc = 0
    for n in xrange(nTries):
        x, y = r(), r()
        if (x*x + y*y <= 1):
            nSucc += 1
    return 4*float(nSucc)/nTries
```

Exemplary results:

```
# SIMPLE SAMPLING MC INTEGRATION:
#  ['10000', '20170503', 'MT']
# estimate: 3.1428
# exact: 3.14159265359
```

(script `main_esimatePi_simpleSampling.py`)

Best practice 4: Functions should be short!

hack

1. A clever or elegant technical accomplishment, especially **one** with a playful or prankish bent. A clever routine in a computer program, especially one which uses **tools** for purposes other than those for which they were intended, might be

[http://www.urbandictionary.com/define.php?term=hack]

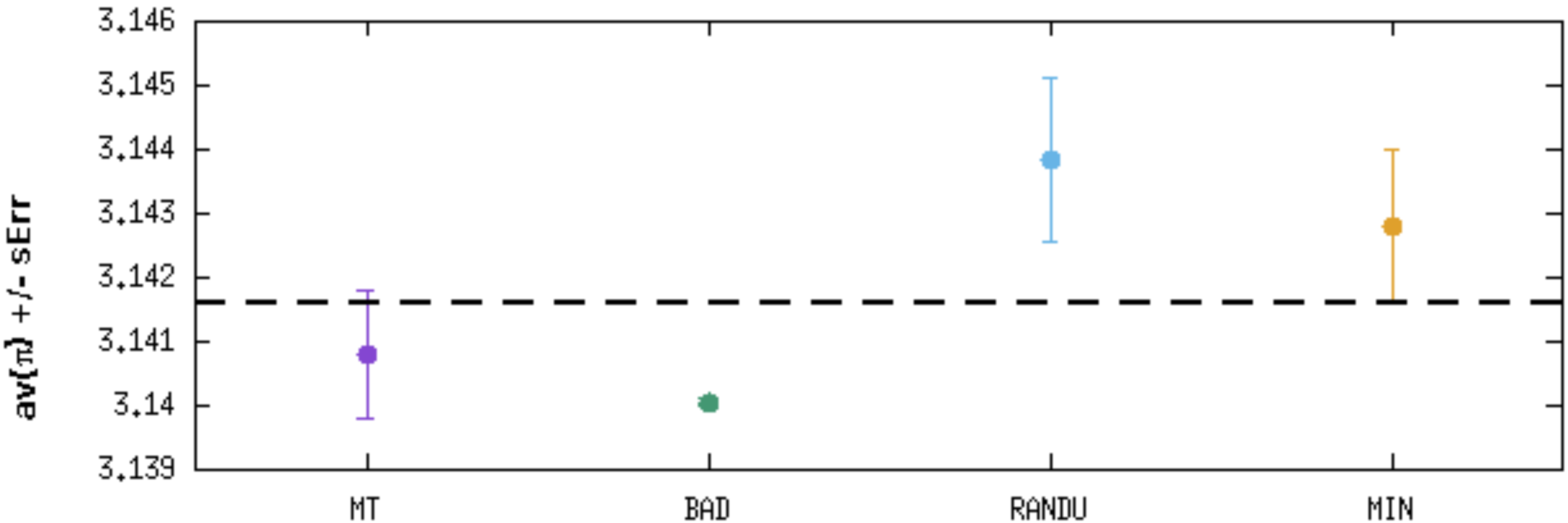
```python
def fetchPRNG(mode=None, seed = 323):
    """pseudo random number generator (PRNG) handler

    Hack emulating switch statement of other languages for easy change of PRNG

    Args:
        mode (str) PRNG type (choices: BAD, MIN, RANDU; default: MT)
        seed (int) integer seed for prng

    Returns:
        r (object) random number generator
    """
    random.seed(seed)
    return {
        'BAD': LCG(106, 1283, 6075, float(seed)/6075 ).next,
        'MIN': LCG(16807, 0, 2147483647, float(seed)/2147483647).next,
        'RANDU': LCG(65539, 0, 2147483648, float(seed)/2147483648).next,
    }.get(mode, random.random)
```

(see skript: `rngSwitch.py`)

2.2 Paradigmatic MC simulations

Optical Technologies.

Numerical experiments using $n_{\mathrm{tries}} = 100\mathrm{k}$ and different PRNGs:



(skript: cmp_estimPi.gpi)

(builds upon script: `main_esimatePi_siSaStats.py`)

- Notes:
  - simple-sampling MC strategy for acceptance-rejection type integration
  - naturally extends to integration of actual functions (see exercises)
  - know the limits of your PRNG (see Lecture 1)

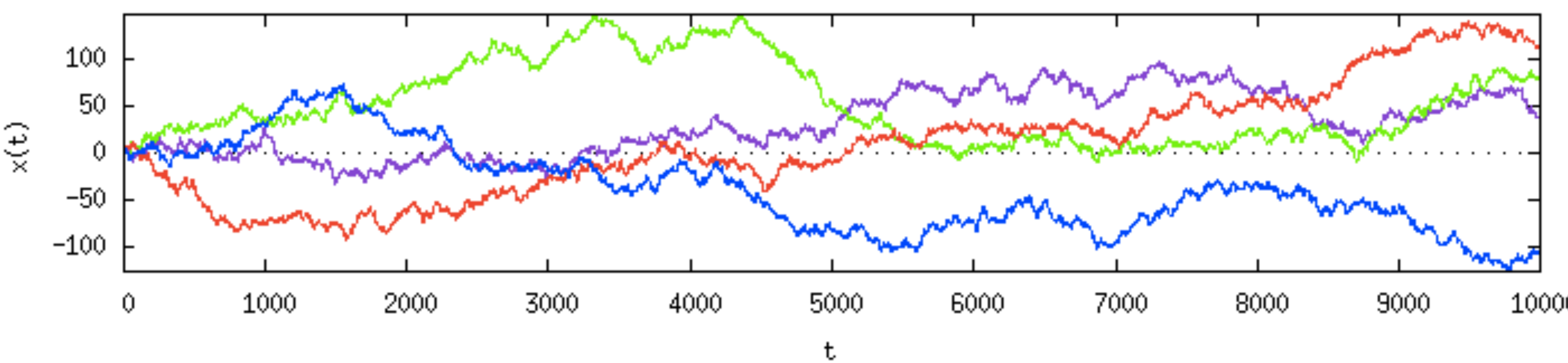
2.2 Paradigmatic MC simulations

2.2.2 1D Random walks

- (Subjective) simplicity ranking of stochastic problems
  - 1st: 1D percolation: configurational statistics; static
  - 2nd: 1D random walk: configurational statistics; dynamic
- paradigmatic example for illustrating the MC method
  - many physically relevant limiting cases
  - can be solved by analytic means

Example:

Four independent discrete symmetric 1D random walks with N=100 steps

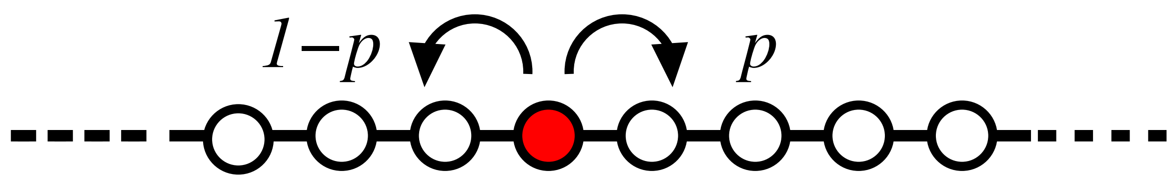


Optical Technologies.

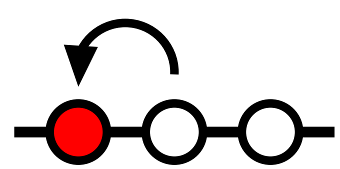
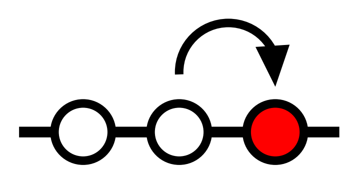
2.2.2.1 Simulation of discrete symmetric 1D random walks

Model:

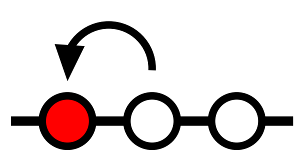
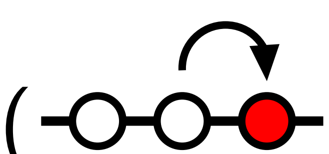
- Random experiment:



- Sample space:

$\Omega = \{$  ,  $\}$

- Random variable:

$X($  $) = -1$, $X($  $) = 1$

Method:

- agent-based simulation approach (highly extendible!)

Code Listing 1: a simple random walk class

```
class RandomWalker1D(object):
    def __init__(self, x0=0., r = StepSampler().discreteSymmetric):
        """instance of 1D random walker class

        Args:
            x0 (float) starting point of the walk (default: x0=0.)
            r (function) step sampler (default: discrete sample space [-1,1])

        Attrib:
            x (float) current walker position
            nSteps (int) number of steps taken
        """
        self.x0 = x0
        self.x  = x0
        self.nSteps = 0
        self.r  = r

    def step(self):
        """perform single step
        """
        self.x += self.r()
        self.nSteps += 1
```

(module: `randomWalker1D.py`)

```
from randomWalker1D import *

def main():
    # PARSE COMMAND LINE ARGUMENTS
    N = int(sys.argv[1])

    # INITIALIZE RANDOM WALKER AT ORIGIN
    rw = RandomWalker1D()

    # SINGLE MC SIMULATION
    for i in range(N):
        print rw.nSteps, rw.x
        rw.step()

main()
```

(skript: `main_1DRW.py`)

Optical Technologies.

Mapping:

- implement discrete symmetric random walk

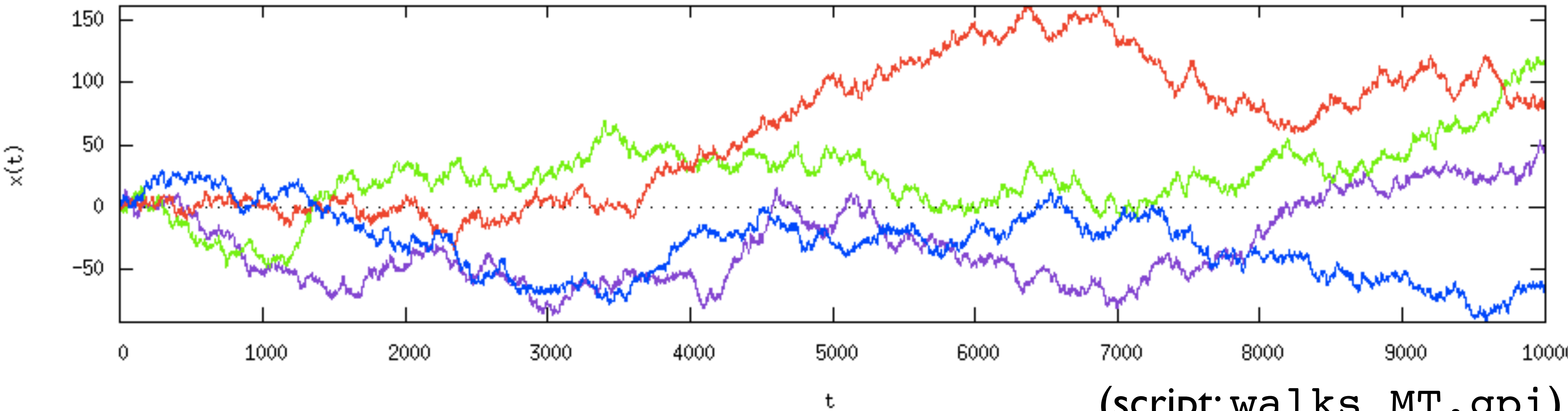
Code Listing 2: an adequate step size sampler (module: `randomWalker1D.py`)

```python
class StepSampler(object):
    def __init__(self, r=random.random):
        self.r = r

    def discreteSymmetric(self):
        """discrete symmetric step size sampler

        Returns:
            dx (int) discrete random variable
               uniformly taken from [-1,1]
        """
        return 1 if self.r() < 0.5 else -1

```

… essentially a very fast coin tosser!

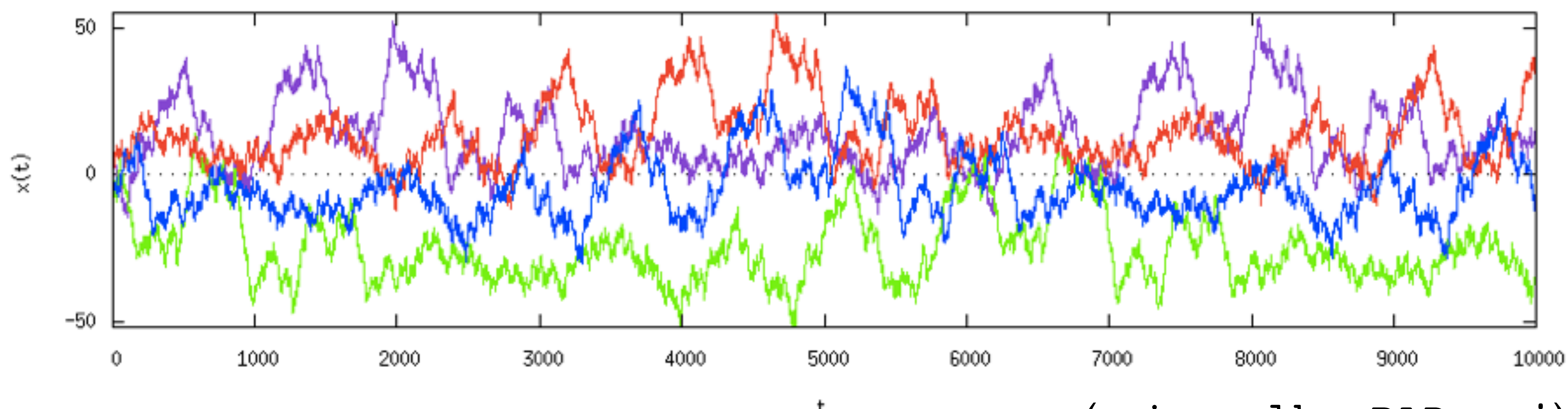
Note: use a maximally reliable PRNG!

Numerical experiments 1: PRNG - Mersenne Twister (MT)



(script: `walks_MT.gpi`)

Numerical experiments 2: PRNG - Park-Miller type LCG (‚BAD')



(script: `walks_BAD.gpi`)

Skeptical snake is skeptical!

2.2 Paradigmatic MC simulations (Quality control)

2.2.2.2 Endpoint distribution

- model requirements:
  - endpoint (i.e. resultant) of discrete symmetric RW of N steps
- method (numerical approach):
  - sample endpoints via new RV: $Y_N = f(X_1, \ldots, X_N) = \sum_{i=1}^{N} X_i$
  - accumulate probability mass function $p(Y_N = y)$
- *Code Listing 3:* (partial) mapping to technical basis

```
    # INITIALIZE STEP SAMPLER
    sampler = StepSampler(fetchPRNG(mode))

    # INITIALIZE PMF TO ACCUMULATE ENDPOINTS
    pmf = PMF()

    # MC SIMULATION TO SAMPLE ENDPOINTS
    for m in range(M):
        rw = RandomWalker1D(0.0, sampler.discreteSymmetric)
        rw.walk(N)
        pmf.add(rw.x)

    # WRITE DATA TO STDOUT
    pmf.dump()
```

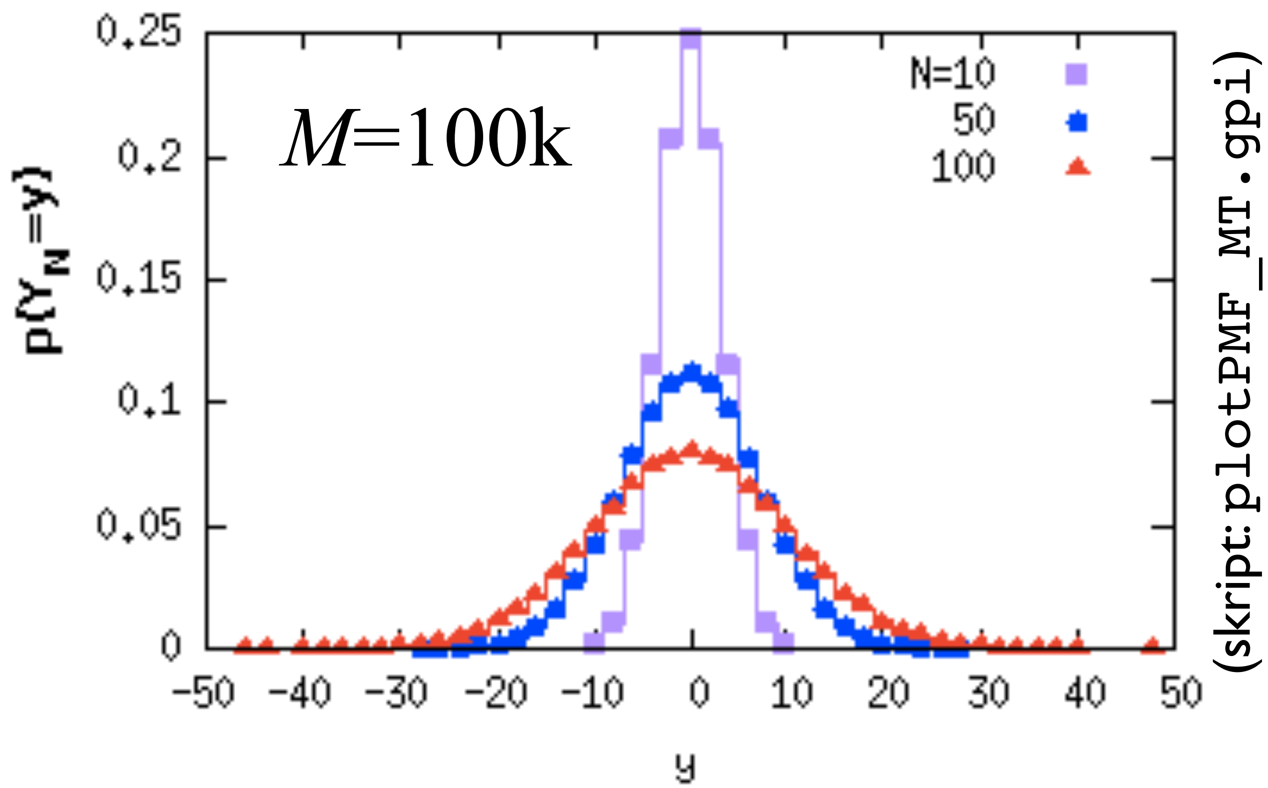
```
class PMF(object):
    def __init__(self):
        self.pmf = {}
        self.n = 0

    def add(self,k):
        self.pmf[k] = self.pmf.get(k,0) + 1
        self.n += 1

    def dump(self):
        print "# (x) (f(x)) (p(x))"
        for (k,v) in sorted(self.pmf.items()):
            print k, v, float(v)/self.n
```

- configuration statistic: exact pmf given by Bernoullian distribution

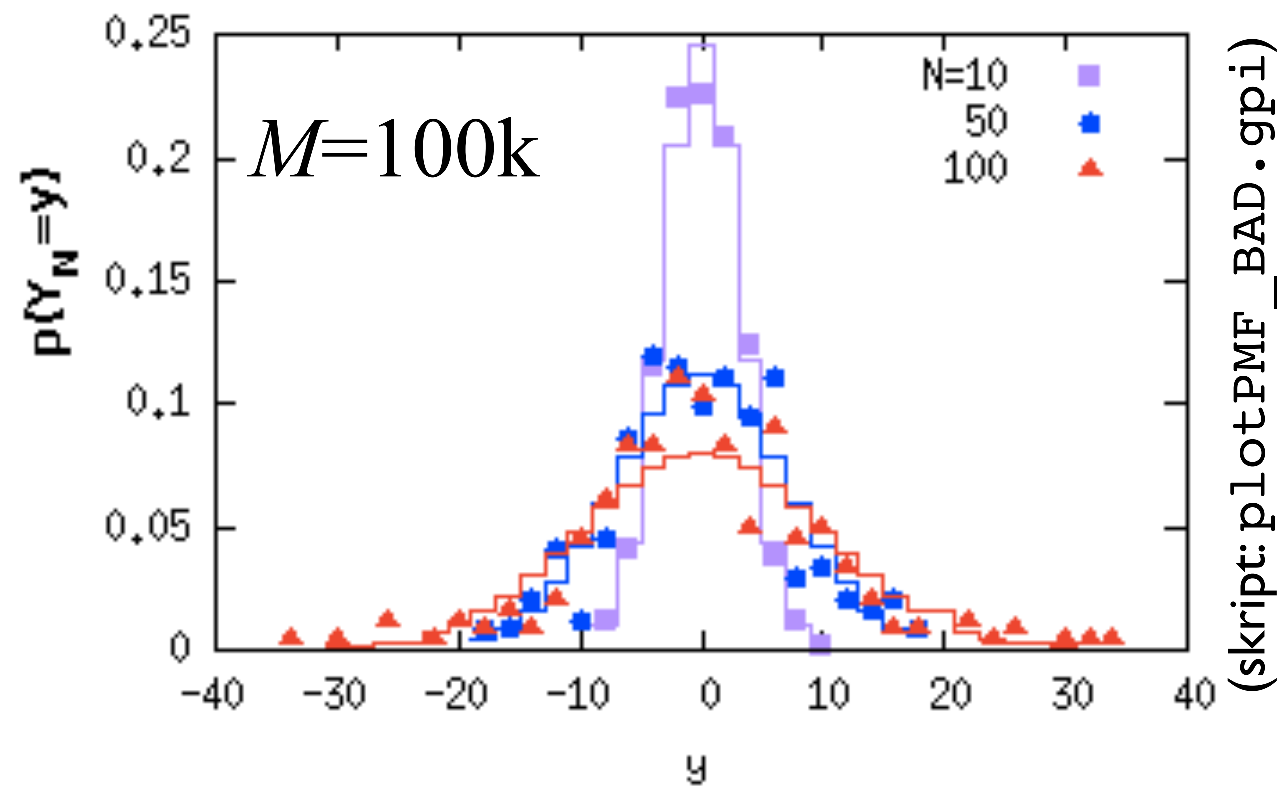
  [Chandrasekhar, S., Rev. Mod. Phys. 15 (1943) 1]

$$W(m,N) = \frac{N!}{[(N+m)/2]![(N-m)/2]!}\left(\frac{1}{2}\right)^N$$

Numerical experiments 1: MT



Numerical experiments 2: BAD



Chi-by-eye assessment:

Looks reasonable … but is it?

(raw data generated via script `main_1DRW_endpointPMF_compare.py`)

Skeptical hippo is still skeptical!

Optical Technologies.

χ^2— test: balancing observed vs. expected

- *chi-square test*: hypothesis test using chi-square statistics as test statistics
- three-step procedure to test for statistical significance:
  - *formulate null hypothesis*

    model based on assumption that observed effect was due to chance
  - *compute p-value*

    probability of observed effect under null hypothesis
  - *interpret results*

    conclusion whether effect is statistically significant
- *test statistics:* measure of total deviation from expected frequencies

$$\chi^2 = \sum_i \frac{(f_i^{\mathrm{obs}} - f_i^{\mathrm{exp}})^2}{f_i^{\mathrm{exp}}}$$ (χ^2—square statistics)

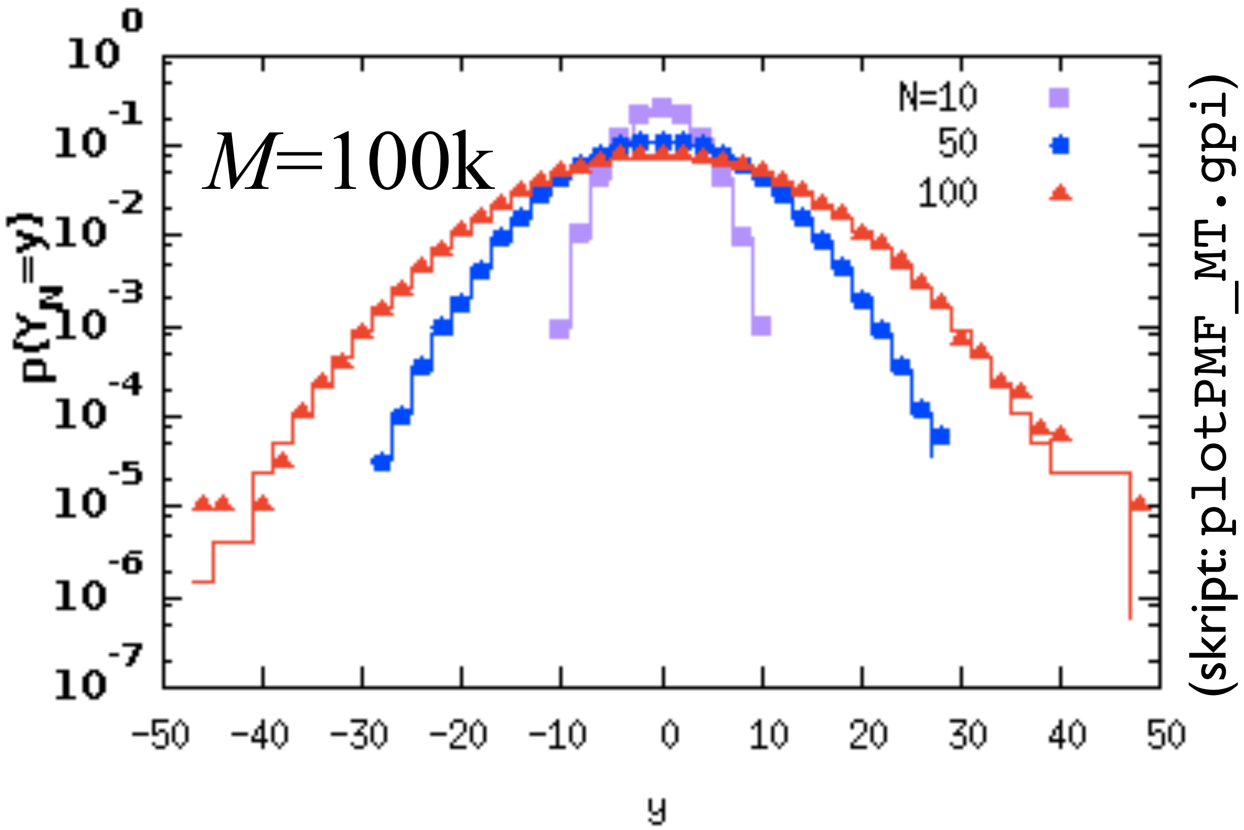
Best practice 5: restrict test to categories with frequency at least 5

(implementation using scipy, see script: `main_1DRW_endpointPMF_chi2py`)

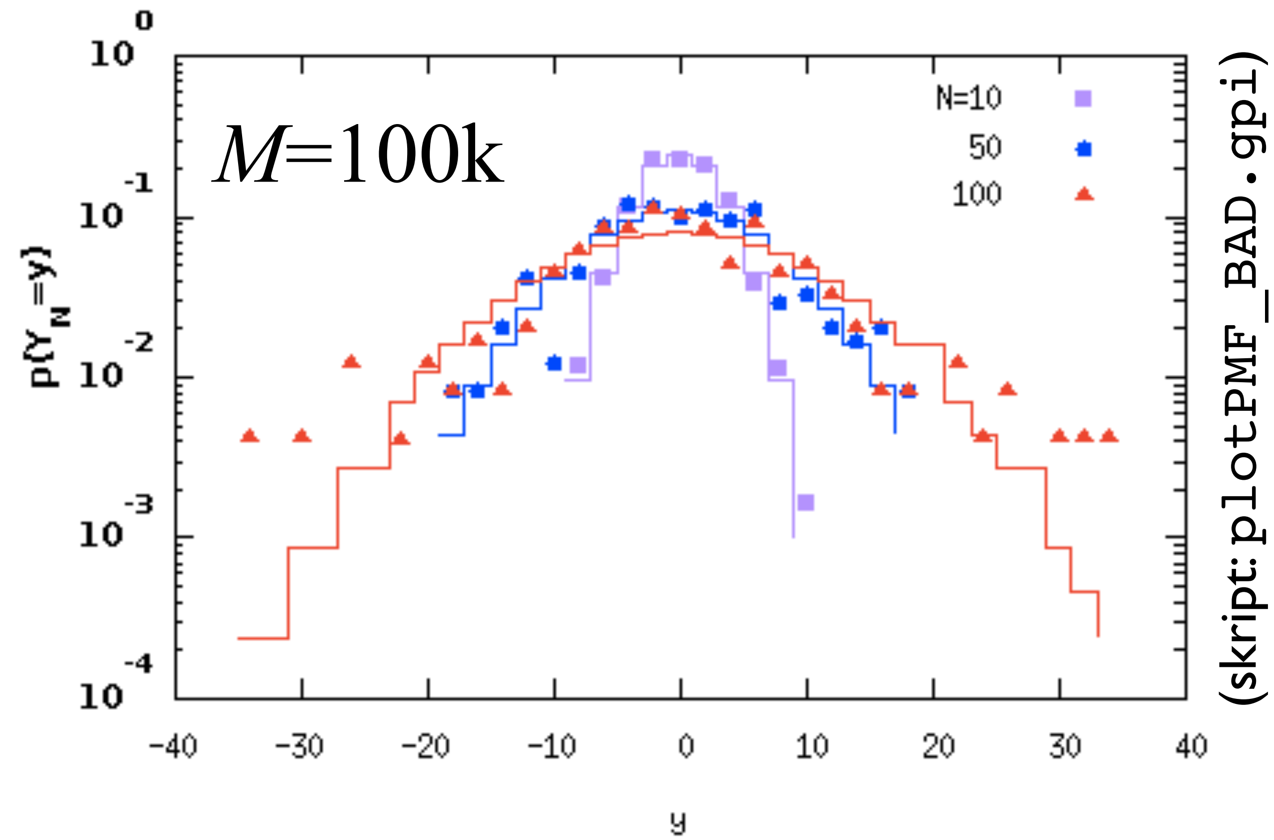
Null hypothesis for χ^2—square test:

1DRW N-step endpoints are drawn from Bernoullian distribution!

Numerical experiments 1: MT



Numerical experiments 2: BAD




```
# TEST NULL HYPOTHESIS FOR SETUP:
# ['main_1DRW_endpointPMF_chi2.py', '20', '100000', 'MT']
# CHI2-stats: 12.506976
# p-Value:  0.708405934178
# TEST RESULT:  PASSED
```

```
# TEST NULL HYPOTHESIS FOR SETUP:
# ['main_1DRW_endpointPMF_chi2.py', '20', '100000', 'BAD']
# CHI2-stats: 646.603277
# p-Value:  6.31403847938e-129
# TEST RESULT:  FAILED
```

(script `main_1DRW_endpointPMF_compare.py`)

Optical Technologies.

2.2.2.3 The 1DRW as limiting case

- close relation between 1D random walk and 1D diffusion:
  - compute probability to find random walk at given position
  - compare to finite-difference variant of 1D diffusion equation

$$\partial_t \; u(x,t) \;=\; D\,\partial_x^2 u(x,t)$$

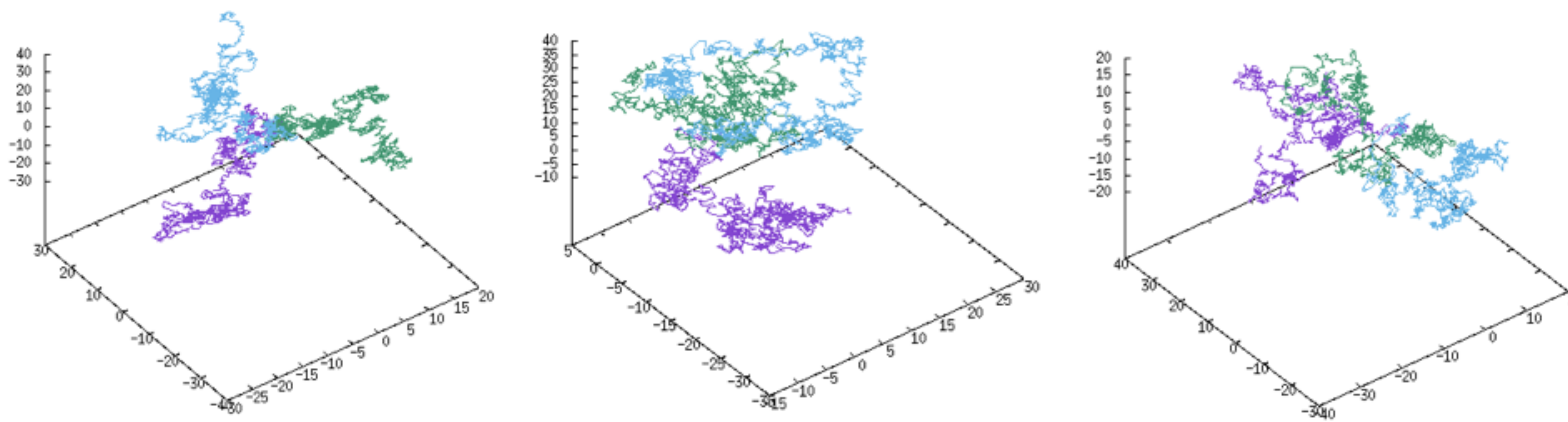
- see blackboard!
- … and exercises!

Best practice 6: Compare to limiting cases whenever possible! (verification testing)

2.2.3 The 3D RW

- intriguing statistical properties:
  - statistical fractal: self-similar curve with scaling dimension $d_\mathrm{f} = 2$
  - short lengthscales: jagged curve
- characterizing random walks - *typical* observables:
  - N-step endpoint distribution
  - radius of gyration, i.e. the moment of inertia of the structure

Examples: three 3DRWs with N=1000 (script: `main_3DRW_iso.py`)



(script: `sample3DRW_iso.gpi`)

2.2 Paradigmatic MC simulations

Optical Technologies.

2.2.3.1 The „isotropic“ 3D RW - *easy* special case

- Model:
  - step length: unity
  - directional cosines: uniformly sampled from surface of 3-sphere (see 1.4.1)
- Method:
  - idea: generate sequence of points $\vec{r}_0 \to \vec{r}_1 \to \ldots \to \vec{r}_N$ via

$$\vec{r}_0 = (0,0,0) \qquad \vec{\omega}_0 = (0,0,1)$$

$$\vec{r}_1 = \vec{r}_0 + \vec{\omega}_0 \qquad \vec{\omega}_1 \leftarrow \text{new}$$

  - *acceptance-rejection* sampling to yield directional cosines $\vec{\omega}_i$ (*Lect. 1, CL 5*)
- Mapping:

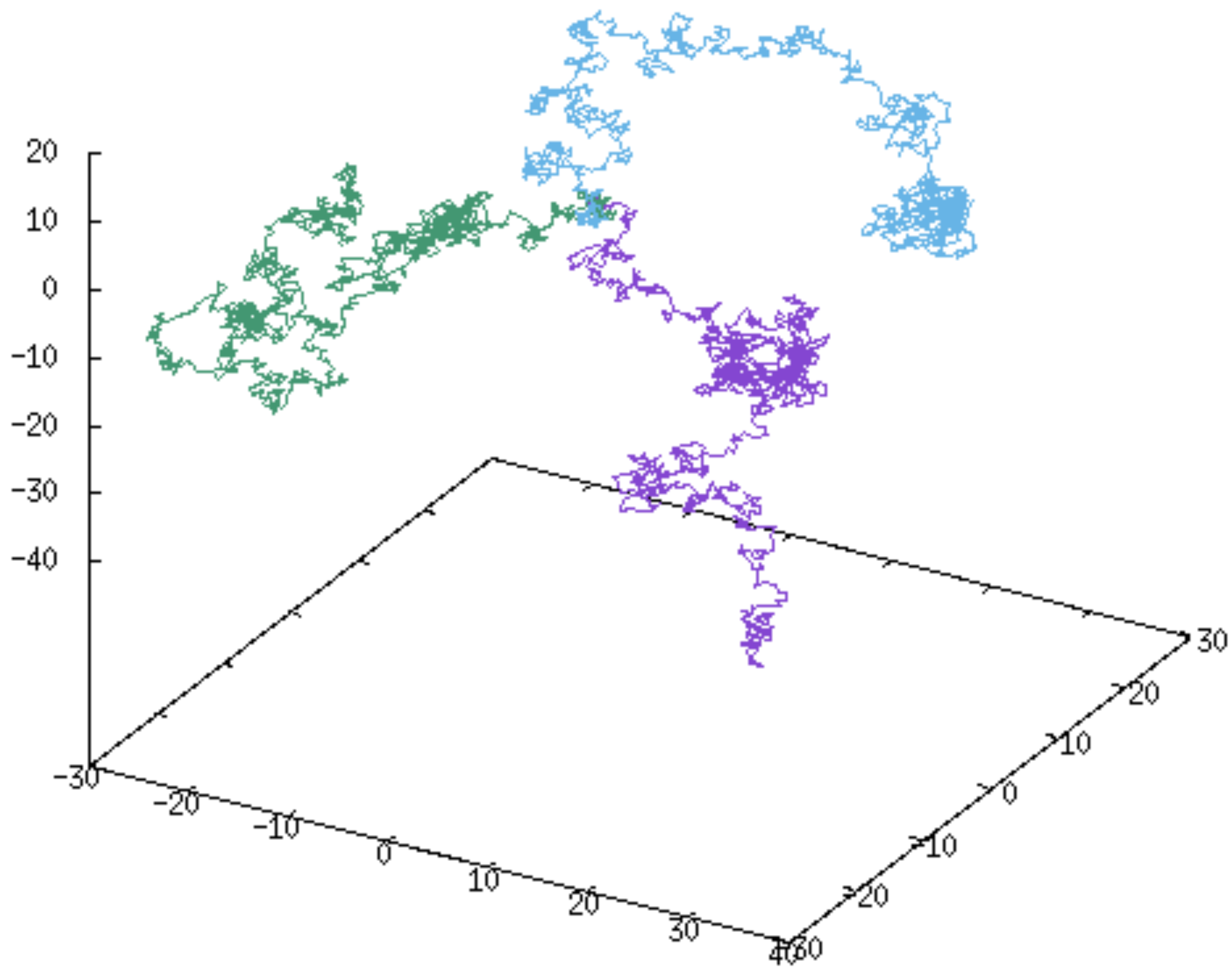
  module: `randomWalker3D_iso.py`

Code Listing 4: isotropic walker

```python
class RandomWalker3D_iso(object):
    """isotropic 3D random walker """
    def __init__(self, r=random.random,
                        x0=(0,0,0), w0=(0,0,1) ):
        """instance of 3D rand wlkr"""
        self.x0 = x0
        self.w = w0
        self.x  = x0
        self.nSteps = 0
        self.wSamp = UnitSphere(r).generate

    def step(self):
        """perform single step
        """
        ux, uy, uz = self.x
        wx, wy, wz = self.w

        self.x = (ux+wx, uy+wy, uz+wz)
        self.w = self.wSamp()
        self.nSteps += 1
```

easy since $\vec{\omega}_i$ can be considered unrelated!

Numerical experiment - isotropic 3DRWs (N=1000):



(scripts: `main_3DRW_iso.py; sample3DRW_iso.gpi`)

2.2.3.2 The „non-isotropic“ 3D RW - *tricky* general case

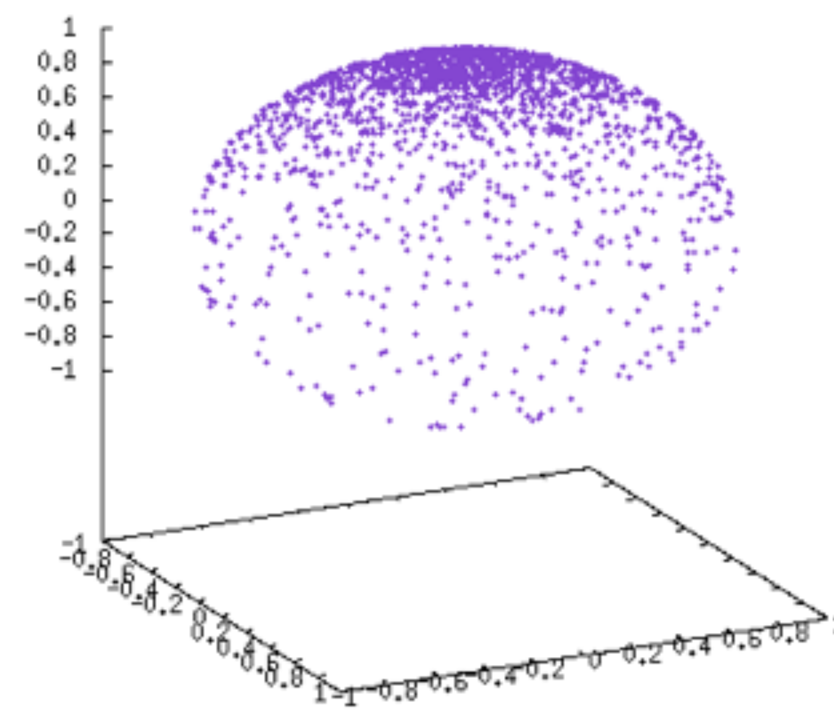
- Model:
  - step length: unity
  - directional cosines: sampled from HG phase function (see 1.4.2)
- Method:
  - idea: generate sequence of points $\vec{r}_0 \to \vec{r}_1 \to \ldots \to \vec{r}_N$ via

$$\vec{r}_0 = (0,0,0) \qquad \vec{\omega}_0 = (0,0,1)$$

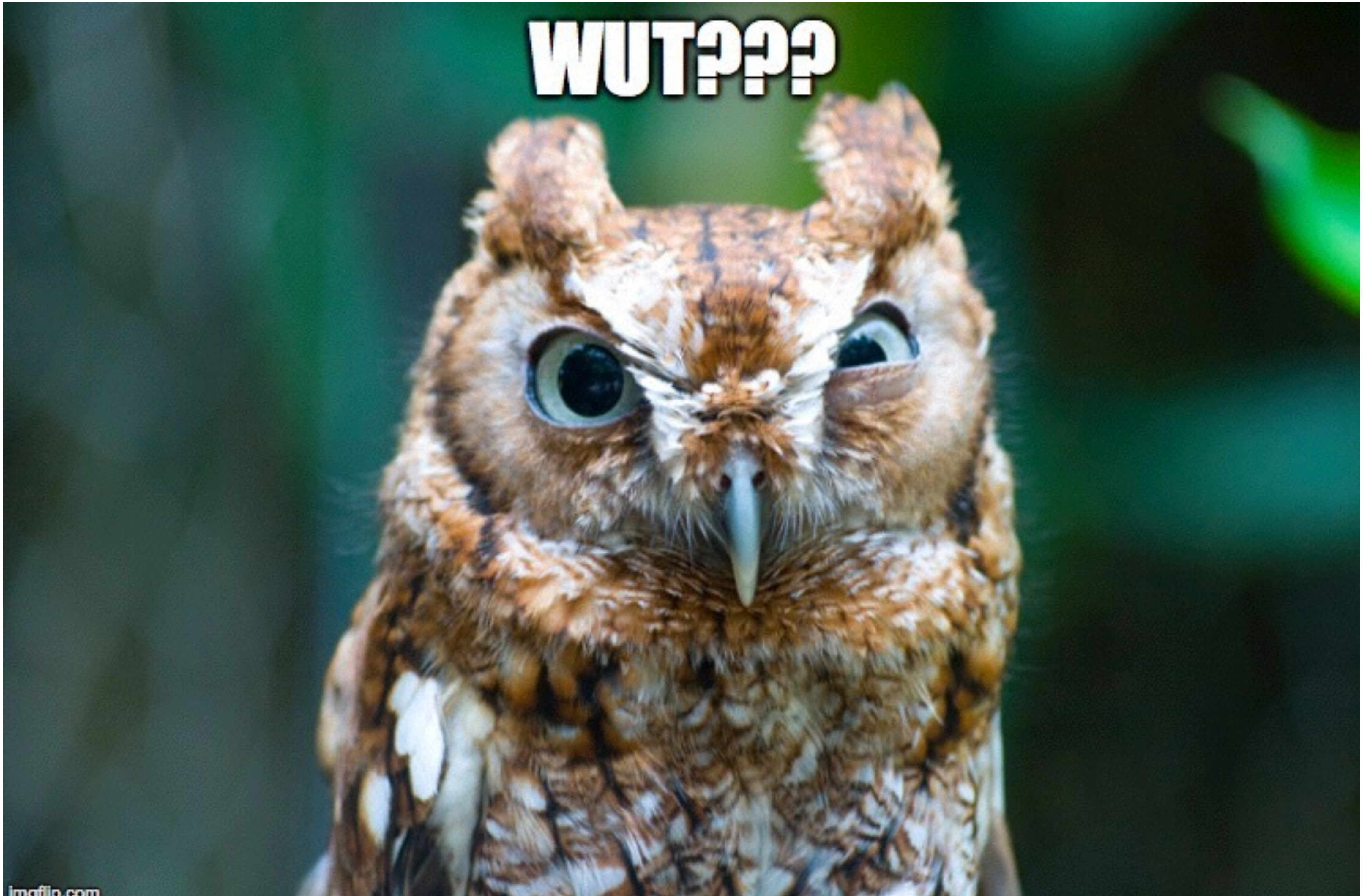
$$\vec{r}_i = \vec{r}_{i-1} + \vec{\omega}_{i-1} \qquad \vec{\omega}_i = \vec{f}(\vec{\omega}_{i-1}, \vec{\omega}_i')$$

  - *direct* sampling to yield directional cosines $\vec{\omega}_i'$ (*Lect. 1, CL 8*) in fixed frame of reference

Note:
sampling yields directional cosines in <u>fixed</u> frame of reference … but we need them in <u>local</u> frame of reference of $\vec{\omega}_{i-1}$



Confused owl is confused!



Computing new directional cosines in local frame of reference:

- conversion from fixed to local frame of reference is purely geometric problem
- Result (see blackboard):

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

given

$$\vec{\omega}_{i-1} = (\omega_x, \omega_y, \omega_z)^T$$

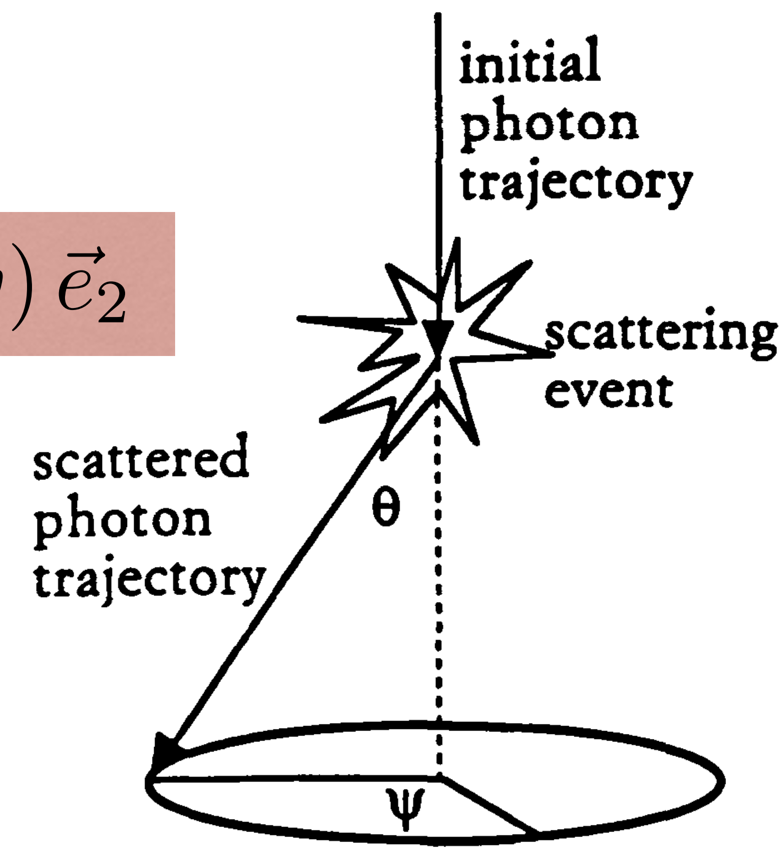
$$\vec{\omega}_i' = (\sin(\theta)\cos(\psi), \sin(\theta)\sin(\psi), \cos(\theta))^T$$

then

$$\vec{\omega}_i = \cos(\theta)\,\vec{\omega}_{i-1} + \sin(\theta)\cos(\psi)\,\vec{e}_1 + \sin(\theta)\cos(\psi)\,\vec{e}_2$$

$$\vec{e}_1 = \frac{(-\omega_y, \omega_x, 0)^T}{\sqrt{1-w_z^2}}$$

$$\vec{e}_2 = \frac{(\omega_x\omega_z, \omega_y\omega_z, -(1-\omega_z^2))^T}{\sqrt{1-w_z^2}}$$



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

2.2 Paradigmatic MC simulations

- Mapping: module `randomWalker3D.py`

  Code Listing 5: 3D random walker step method with custom cosine sampler

```python
    def step(self):
        """perform single step

        Implements single step for 3D random walker by computing new
        directions following Ref [1]

        Refs:
            [1] Multiple Scattering in Reflection Nebulae
                Witt, A. N.
                Astrophys. J., 35 (1977) 1
        """
        ux, uy, uz = self.x
        wx, wy, wz = self.w
        sinTcosP, sinTsinP, cosT = self.wSamp()

        if abs(wz) < 0.99999:
            fac = 1./np.sqrt(1.- wz*wz)
            wxp = wx*cosT - fac*(wy*sinTcosP - wx*wz*sinTsinP)
            wyp = wy*cosT + fac*(wx*sinTcosP + wy*wz*sinTsinP)
            wzp = wz*cosT - sinTsinP*np.sqrt(1-wz*wz)
        else:
            wxp = sinTcosP
            wyp = sinTsinP
            wzp = cosT*wz

        self.x = (ux+wx, uy+wy, uz+wz)
        self.w = (wxp, wyp, wzp)
        self.nSteps += 1
```

limiting case for when ω_z gets too dominant

Best practice 7:
make incremental chances!
(cf. *Code Listing 4*)

2.2 Paradigmatic MC simulations

(Quite) general 3D random walk:

Code Listing 6: 3D random walk driver script (see `main_3DRW.py`)

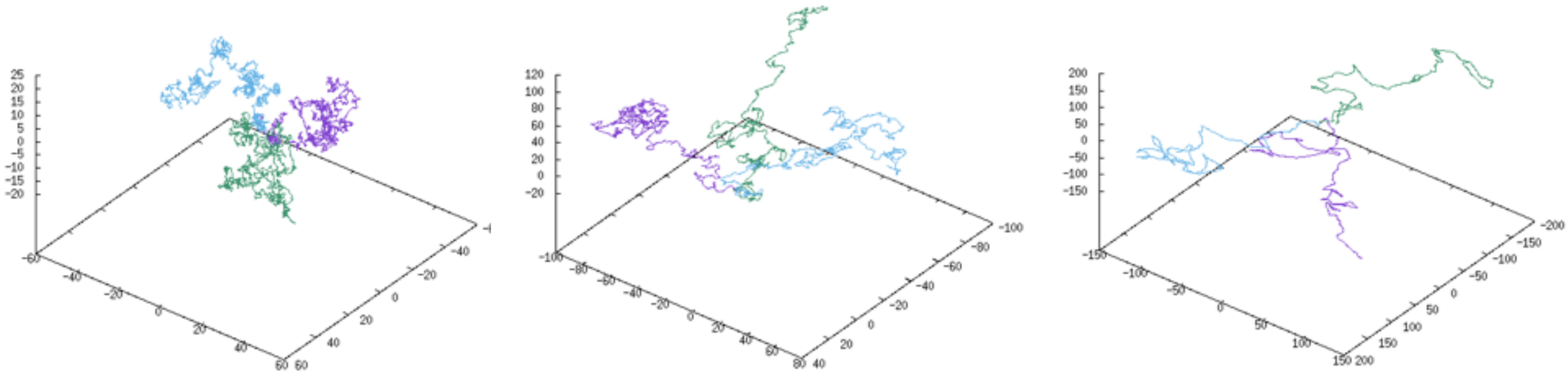
```
from randomVariateGenerator import *
from randomWalker3D import RandomWalker3D

def main():
    N = int(sys.argv[1])
    g = float(sys.argv[2])
    mySamp = HenyeyGreenstein(g)
    myRW = RandomWalker3D(mySamp.generate)

    for i in range(N):
        x,y,z = myRW.x
        print myRW.nSteps, x, y, z
        myRW.step()

main()
```

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

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

Numerical experiments - non-isotropic 3DRWs (N=1000):

$g = 0.3$ $\qquad$ $g = 0.7$ $\qquad$ $g = 0.95$



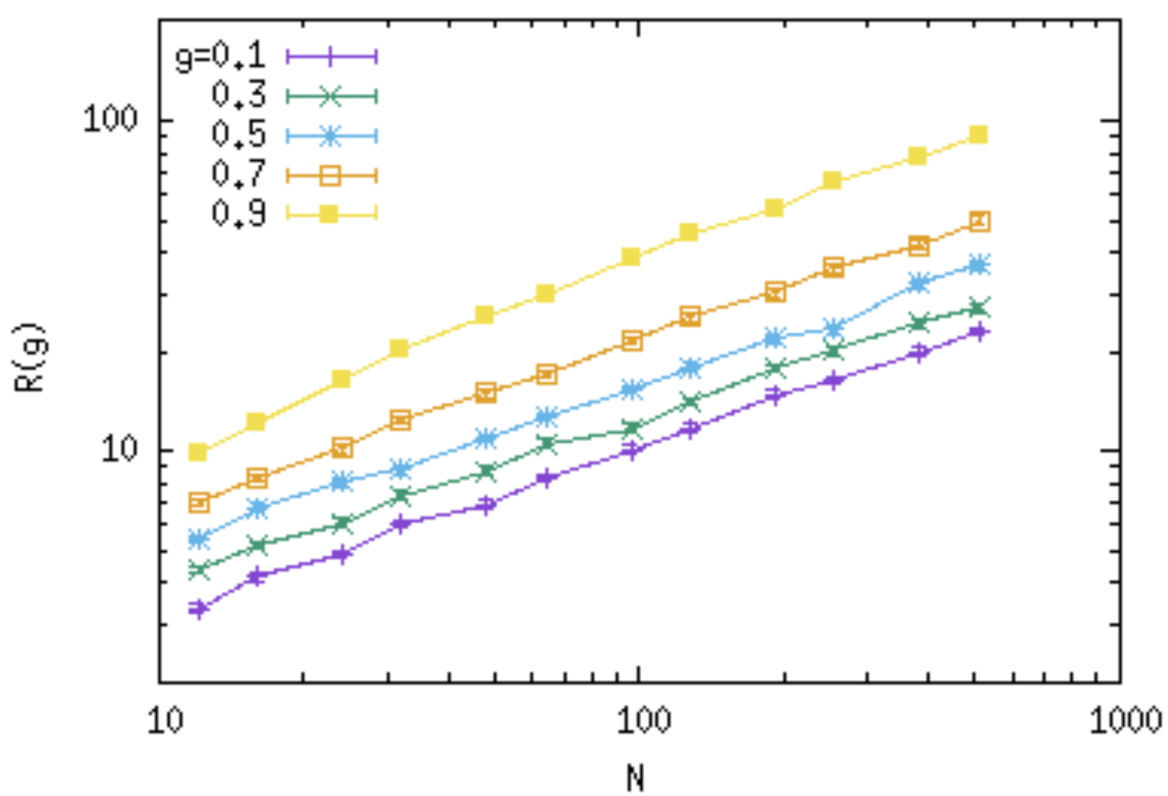
(scripts: `main_3DRW.py; sample3DRW.gpi`)

- directional cosines sampled from Henyey-Greenstein phase function
- observation for increasing anisotropy g:
  - walks look less rugged!
  - walks tend to cover more distance!

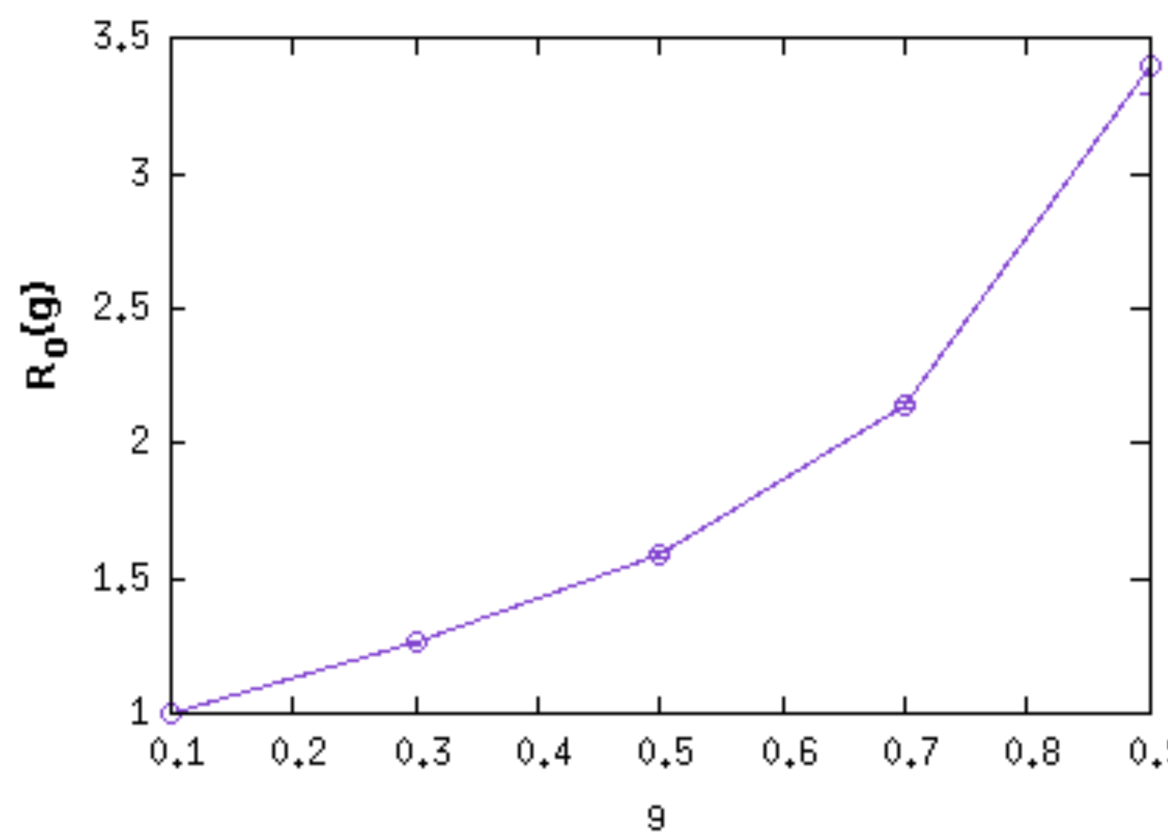
Optical Technologies.

Analysis: characterizing the endpoint distribution

- Notes:
  - usually one cares about the asymptotic limit, i.e. $N \to \infty$
  - our applications: due to *optical absorption* (photon) walks are short
- here: statistical properties of rather short walks
  - self-similar scaling: $$R(g) = R_0(g) \cdot N^{1/d_\mathrm{f}} \qquad d_\mathrm{f} = 2$$



(script: resultantScaling.gpi)

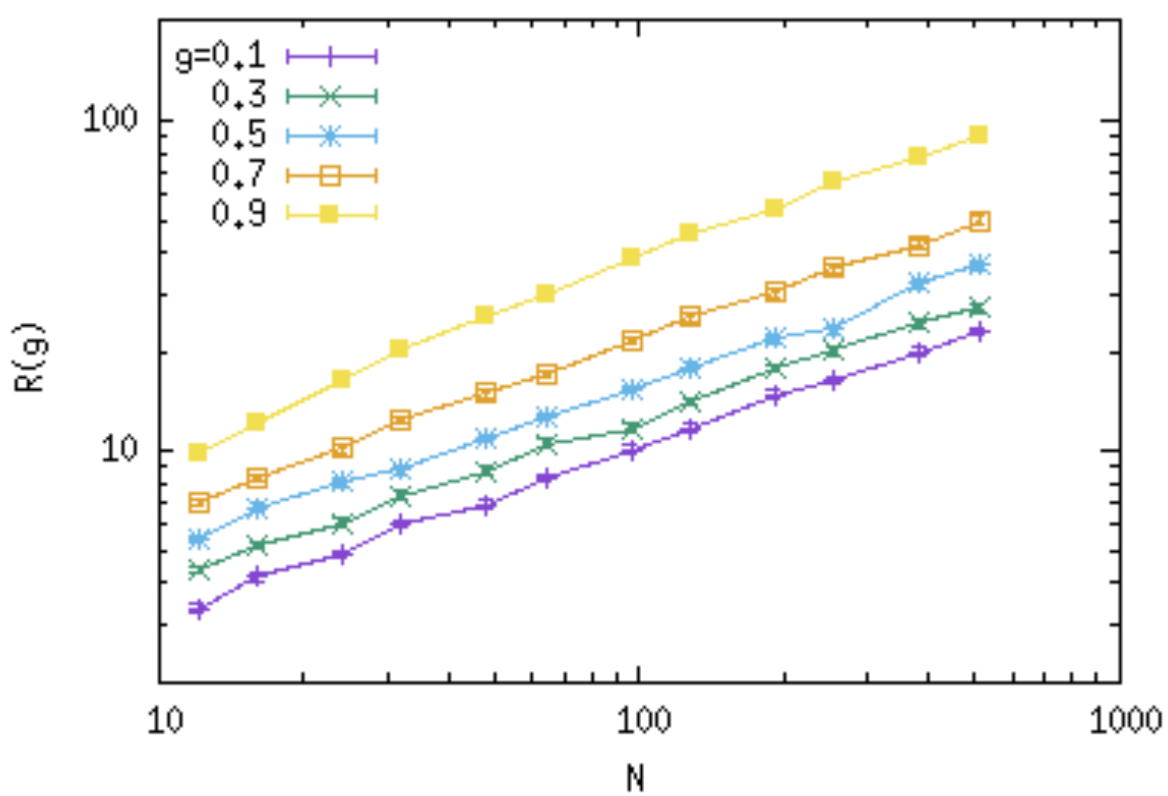


(script: resultantPref.gpi)

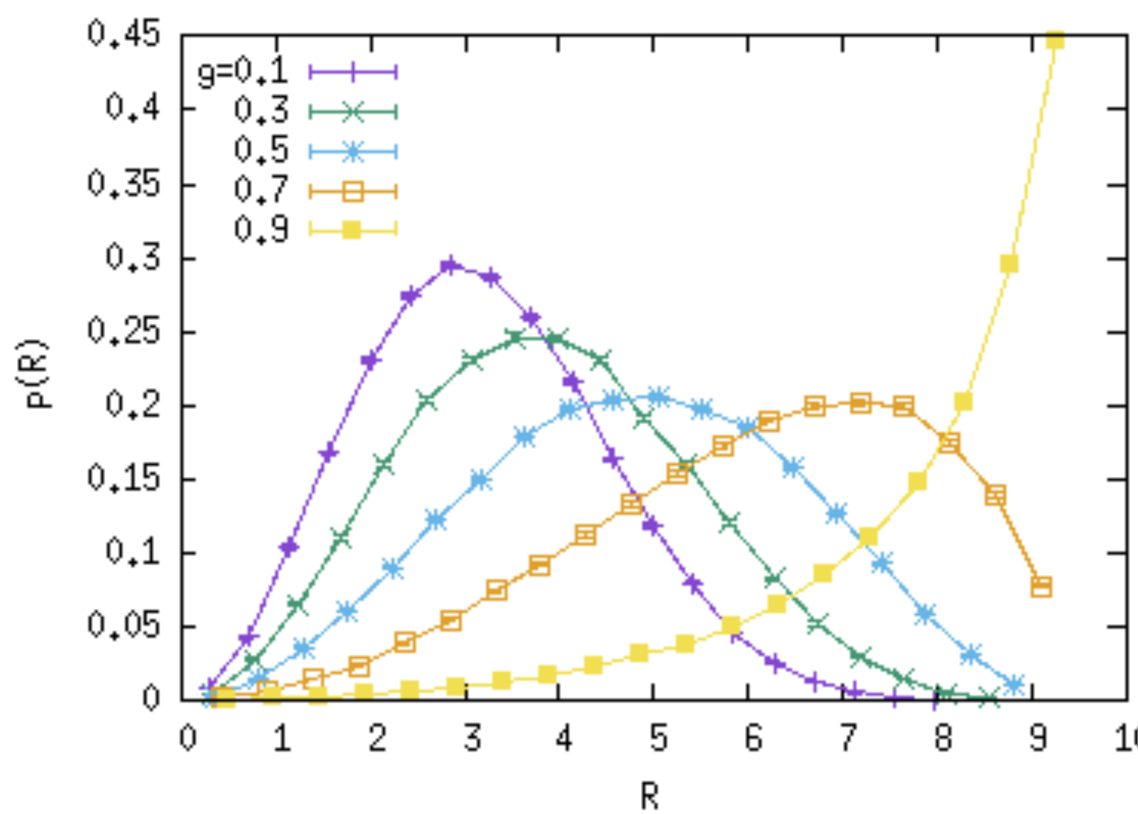
Analysis: characterizing the endpoint distribution

- Notes:
  - usually one cares about the asymptotic limit, i.e. $N \to \infty$
  - our applications: due to *optical absorption* (photon) walks are short
- here: statistical properties of rather short walks
  - self-similar scaling:

$$R(g) = R_0(g) \cdot N^{1/d_{\mathrm{f}}} \qquad d_{\mathrm{f}} = 2$$



(script: resultantScaling.gpi)



(script: resultantHist.gpi)

Notes on lecture 2:

- *upcoming exercises*: a numerical study on
  - random number generation,
  - Monte Carlo integration, and,
  - „small particle scattering“

- *next*: photon migration in turbid media is just a $3\mathrm{D}$ random walk … with a peculiar step size distribution … within a (probably) disordered medium! We will implement all that in lecture 3

After lecture: example programs available at

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