

Integration in higher dimensions

Next: What about integration of high-dimensionality objects?

- We've done 1D and 2D, what about ND?
- Application: Dealing with multivariate probability distribution functions. “Marginalize” a joint PDF – integrate out one variable.
- Application: Statistical Mechanics (physics).

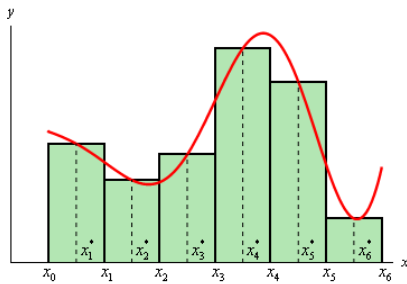
$$Z(N, V, T) = \frac{1}{N!} \prod_{j=1}^N \left[\frac{1}{h^3} \int d^3 p_j \int_V d^3 q_j \right] e^{-\beta \mathcal{H}(p_{3N}, q_{3N})}$$

Partition function for ideal gas

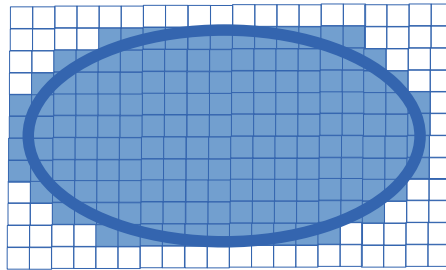
- Integration complexity tends to grow as $O(N^D)$ (or worse), where D = dimensionality.
 - For high dimensionality problems, this stinks.
- Integration of high-dimensional objects is a research area.

Problems with integration of high-dimensionality objects

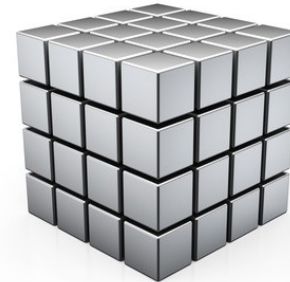
- Time complexity of the usual integration methods grows as $O(N^D)$.
 - You must evaluate a function at each point in an ND grid.



1D: $O(N)$



2D: $O(N^2)$



3D: $O(N^3)$

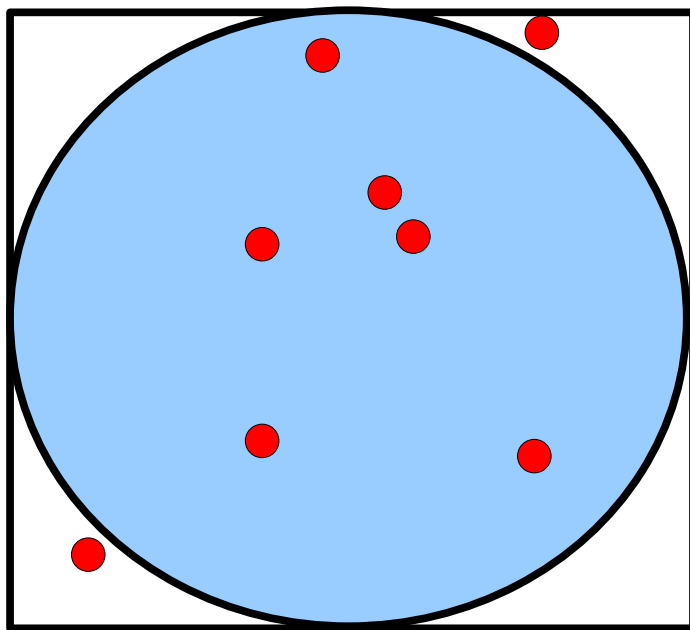
- Starting in one corner of the cube means you waste lots of time integrating over the periphery of your function.

Features of a good ND integration method

- Start getting results immediately.
 - Don't waste time integrating unimportant regions in your domain.
- Convergent method
 - The longer you work, the more accurate your result.
- Complexity growing more slowly than $O(N^D)$

Monte Carlo integration

- Example in 2D: Compute area of circle
 - Famous way to compute π
- Throw random darts at circular target

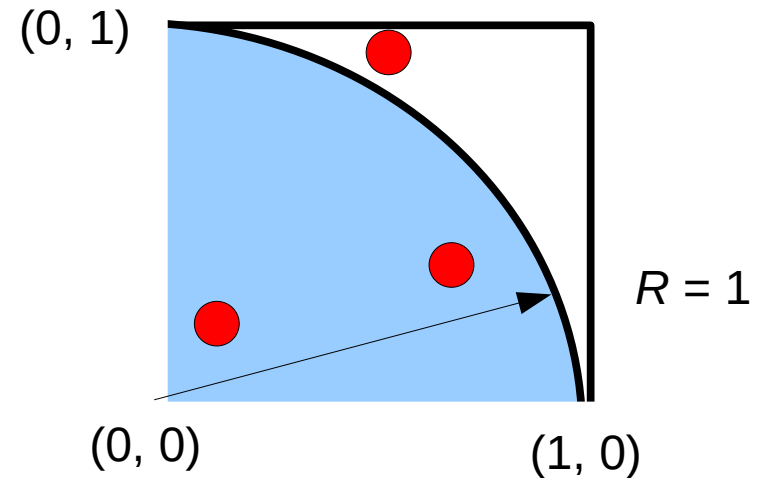


Monte Carlo – *famous part of Monaco with popular gambling casino (south of France).*

- Count hits inside circle to get π

Algorithm

1. $S = 0$ % Hit counter
 2. Loop N times
 3. Generate random x, y in interval $[0, 1]$
 4. If $x^2 + y^2 < 1$ then $S = S+1$
 5. End Loop
 6. Return $\pi = 4*S/N$
- Algorithm relies on **uniform** random sampling of square area in the limit of large N.



$$A_{\text{circle}} = \frac{\pi R^2}{4}$$

$$A_{\text{square}} = R^2$$

$$\pi = 4 \frac{A_{\text{circle}}}{A_{\text{square}}}$$

```
function p = monte_carlo_pi()
    % This fcn computes a value of pi based on the
    % dart-throwing Monte Carlo algorithm

    N = 1000000; % Number of trials
    S = 0;       % Number of hits inside circle

    for i = 1:N
        rx = rand();
        ry = rand();
        if (rx*rx + ry*ry < 1)
            S = S + 1;
        end
    end

    p = 4*S/N;
end
```

```
>> monte_carlo_pi  
ans =  
3.1373600000000000
```

```
>> monte_carlo_pi  
ans =  
3.1388800000000000
```

Results obtained with
N = 100000 trials

```
>> monte_carlo_pi  
ans =  
3.1408000000000000
```

```
>> monte_carlo_pi  
ans =  
3.1485600000000000
```

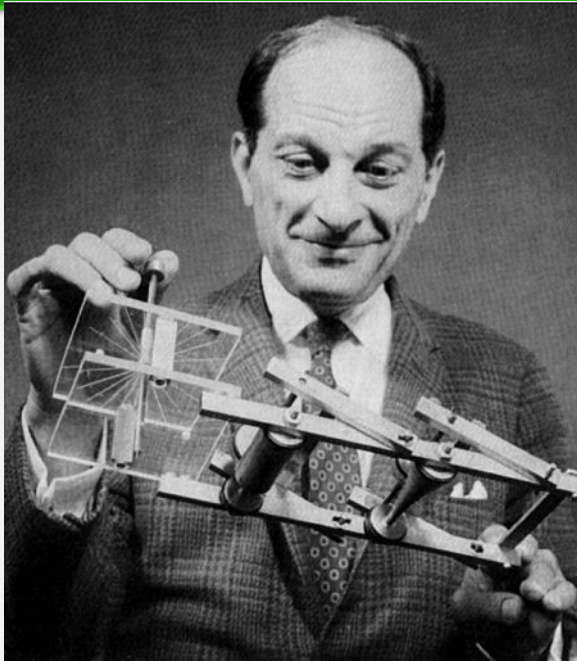
```
>> monte_carlo_pi  
ans =  
3.1347200000000000
```


Monte Carlo pi computation



- Simple demonstration of a stochastic simulation.
- Convergence to π is very slow – $1/\sqrt{N}$
- Extension of integration technique to odd-shaped objects is obvious.

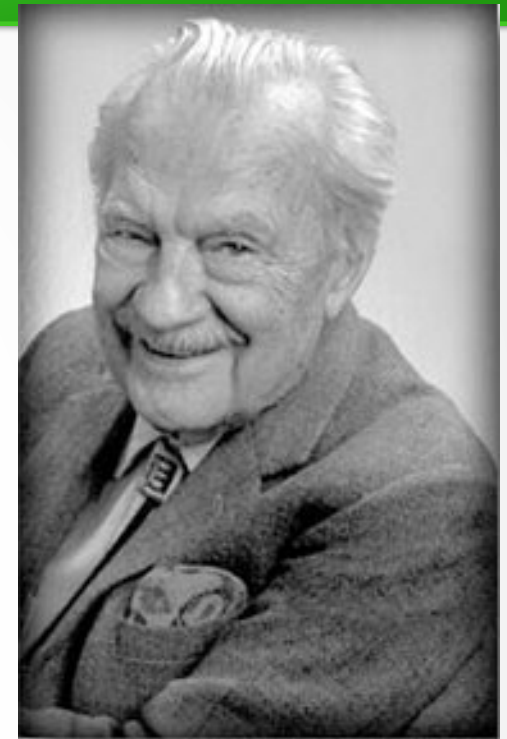
Origins of Monte Carlo method



Stanislaw Ulam



John von Neumann



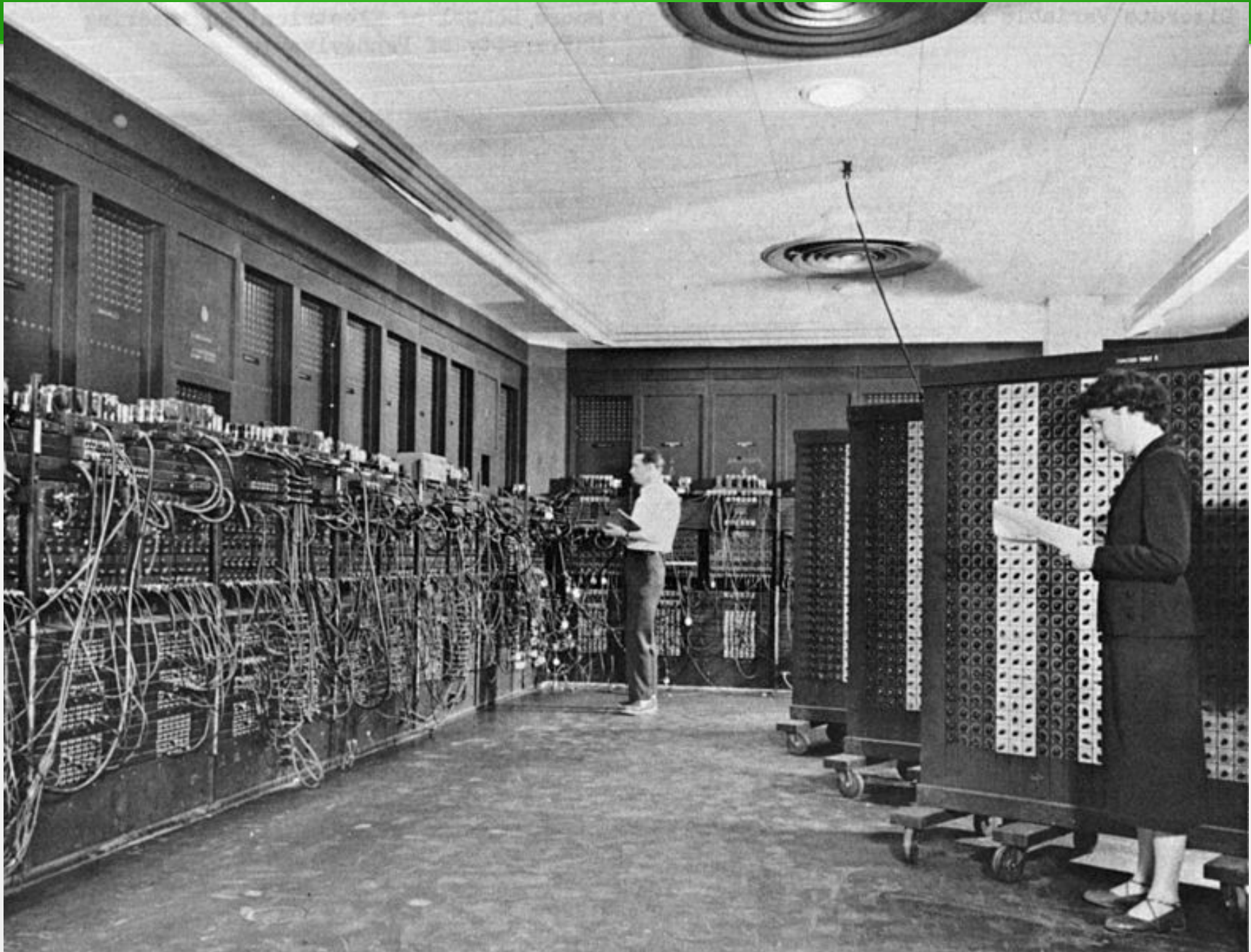
Nicholas Metropolis

- Method developed at Los Alamos.
- First implemented on ENIAC computer (late 1940s, early 1950s)
- Really requires computer to be possible (i.e. impossible prior to development of computers).



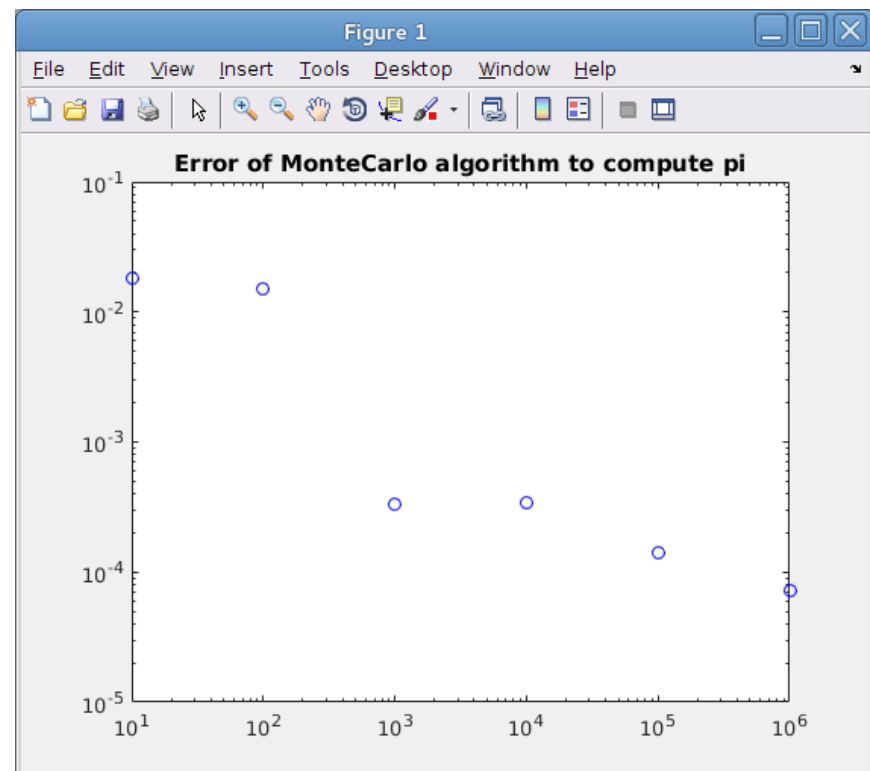
Ivy Mike, 31 October 1952

ENIAC



Convergence of Monte Carlo pi calculation

- Convergence to pi is very slow – $1/\sqrt{N}$
- Why $1/\sqrt{N}$?



Slope of $\log(\text{err})$ vs. $\log(N) = -0.518425$

Integrating a function

- The above computation of pi integrated the 1-function over an area A:

$$\iint_{(x,y) \in A} dx dy 1$$

- What about an arbitrary function over a volume in N dimensions?

$$\int_{\vec{x} \in V} d\vec{x} f(\vec{x}) = ?$$

Integrating a function

- Recall the expression for finding the average of a function over a domain V :

$$\langle f(x) \rangle_V = \frac{1}{V} \int_{\vec{x} \in V} d\vec{x} f(\vec{x})$$

- To find average, what if we took N samples of $f(x)$, at uniformly distributed points x_i in V , and added them up?

$$\langle f(x) \rangle_V = \frac{1}{N} \sum_{i=1}^N f(\vec{x}_i)$$

Monte Carlo method

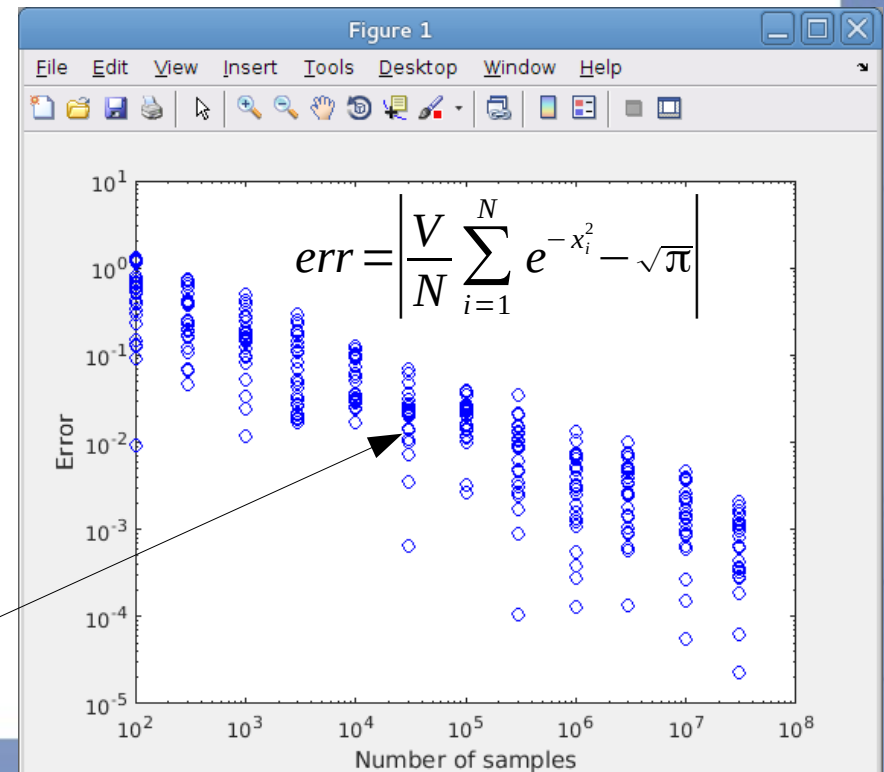
- This suggests

$$\int_{\vec{x} \in V} d\vec{x} f(\vec{x}) = \frac{V}{N} \sum_{i=1}^N f(\vec{x}_i) \quad \text{The Monte Carlo method}$$

- Example runs computing the area under the Gaussian:

$$\int_{-\infty}^{\infty} dx e^{-x^2} = \sqrt{\pi}$$

$1/\sqrt{N}$ error scaling



Back to error: Why $1/N^{1/2}$?

- Monte Carlo involves the following approximation for an integral:

$$\int_V dx f(x) \approx \frac{V}{N} \sum_{i=1}^N f(x_i) \quad \text{where } x_i \in \text{uniform random value in } V \text{ and } N \text{ is large number.}$$

- Central limit theorem asserts:

If X_i is random variable with mean μ , then

$$\lim_{n \rightarrow \infty} \left[\left(\frac{1}{n} \sum_{i=1}^n X_i \right) - \mu \right] \Rightarrow \frac{\sigma}{\sqrt{n}} N(0, 1)$$

where $N(0, 1)$ denotes a Gaussian (normal) distribution with zero mean and unit variance.

Put two previous statements together

- Monte Carlo approximation is same as finding average

$$\lim_{N \rightarrow \infty} \frac{V}{N} \sum_{i=1}^N f(x_i) \rightarrow \int_V dx f(x) \\ \rightarrow V \langle f(x) \rangle_V$$

- Therefore, error after N trials is

$$e_N = \left(\frac{V}{N} \sum_{i=1}^N f(u_i) \right) - V \langle f(u) \rangle_V$$

Looks a lot like statement of Central Limit Theorem on last slide.

The CLT implies $1/\text{sqrt}(n)$

- Therefore, for large n ,

$$\lim_{N \rightarrow \infty} [e_N] \Rightarrow \frac{\sigma}{\sqrt{N}} N(0, 1)$$

- So in the limit of large N , the error of Monte Carlo integration decreases as

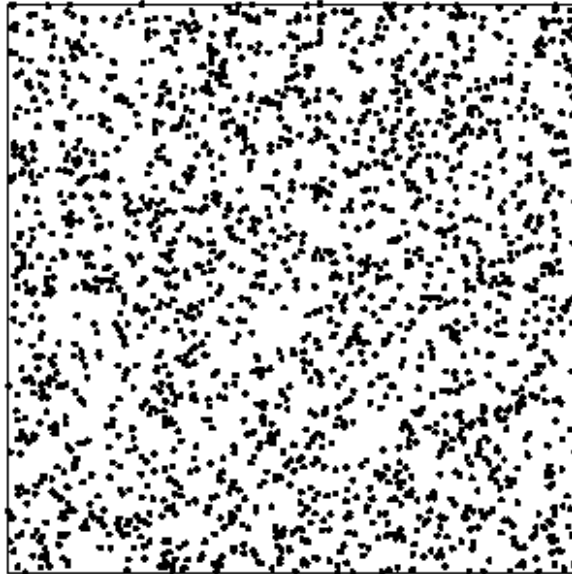
$$e_N \rightarrow \frac{1}{\sqrt{N}} \quad \text{QED.}$$

Can we improve convergence?

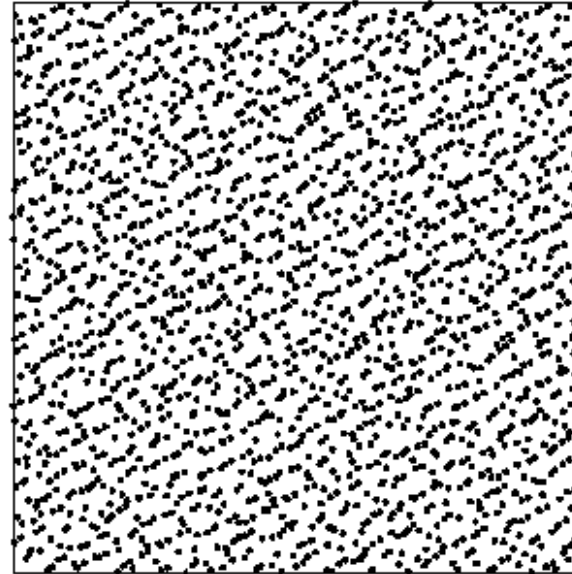
- $1/N^{1/2}$ stinks. Can we do better?
- We use random number and Monte Carlo as a technique to **sample** the object under study.
- The key is sampling. We use `rand()` because it creates a uniform distribution of real numbers on domain $[0, 1]$.
- The distribution is uniform, but because each successive random number is uncorrelated with all the others, convergence is $1/N^{1/2}$.
- Since all we want is a way to sample the object, can we find a better method to sample it?

Quasi-Monte Carlo methods

Random



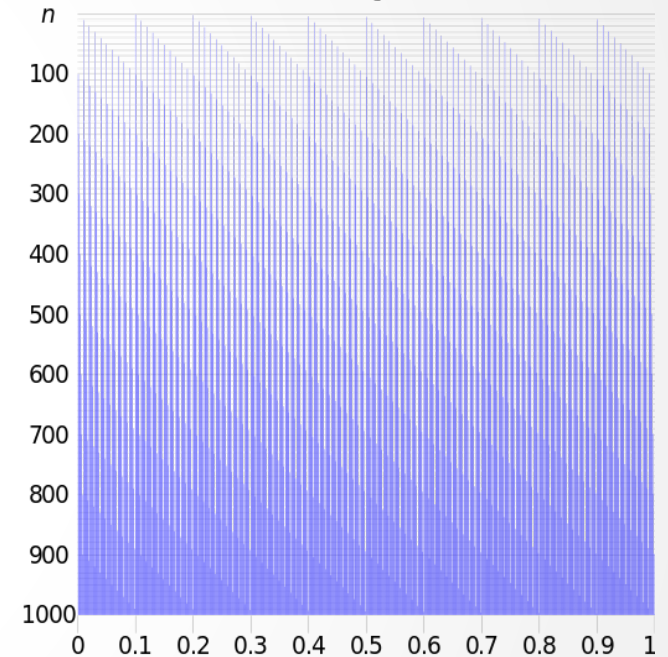
Quasi-Random



- We want a way to sample our domain which is uniform.
- A uniform random distribution is not the ideal way to sample.
 - Prone to “clumping”.
- Consider a quasi-random (low-discrepancy) distribution.
- Such a distribution also uniformly samples our domain.
- Note: points in quasi-random distribution are correlated.

How to create quasi-random sequence?

- Goal: create way to generate sequence which covers domain $[0, 1]$ uniformly, and increases density with increasing N .
- Examples:
 - van der Corput sequence
 - Halton sequence
 - Sobol Sequence
 - Etc.....



Such a sequence is extremely useful in computational problems where numbers are computed on a grid, but it is not known in advance how fine the grid must be to obtain accurate results. Using a quasirandom sequence allows stopping at any point where convergence is observed, whereas the usual approach of halving the interval between subsequent computations requires a huge number of computations between stopping points.

-- Trandafir, Aurel and Weisstein, Eric W. "Quasirandom Sequence." From MathWorld--A Wolfram Web Resource. <http://mathworld.wolfram.com/QuasirandomSequence.html>

van der Corput sequence

- 1D sequence.
- Elements form dense set on unit interval $[0, 1]$.
- Set has nice property that unit interval is covered uniformly with increasing N

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0, 1]/ \sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	

Base = 2


Max sequence length = 16

van der Corput sequence

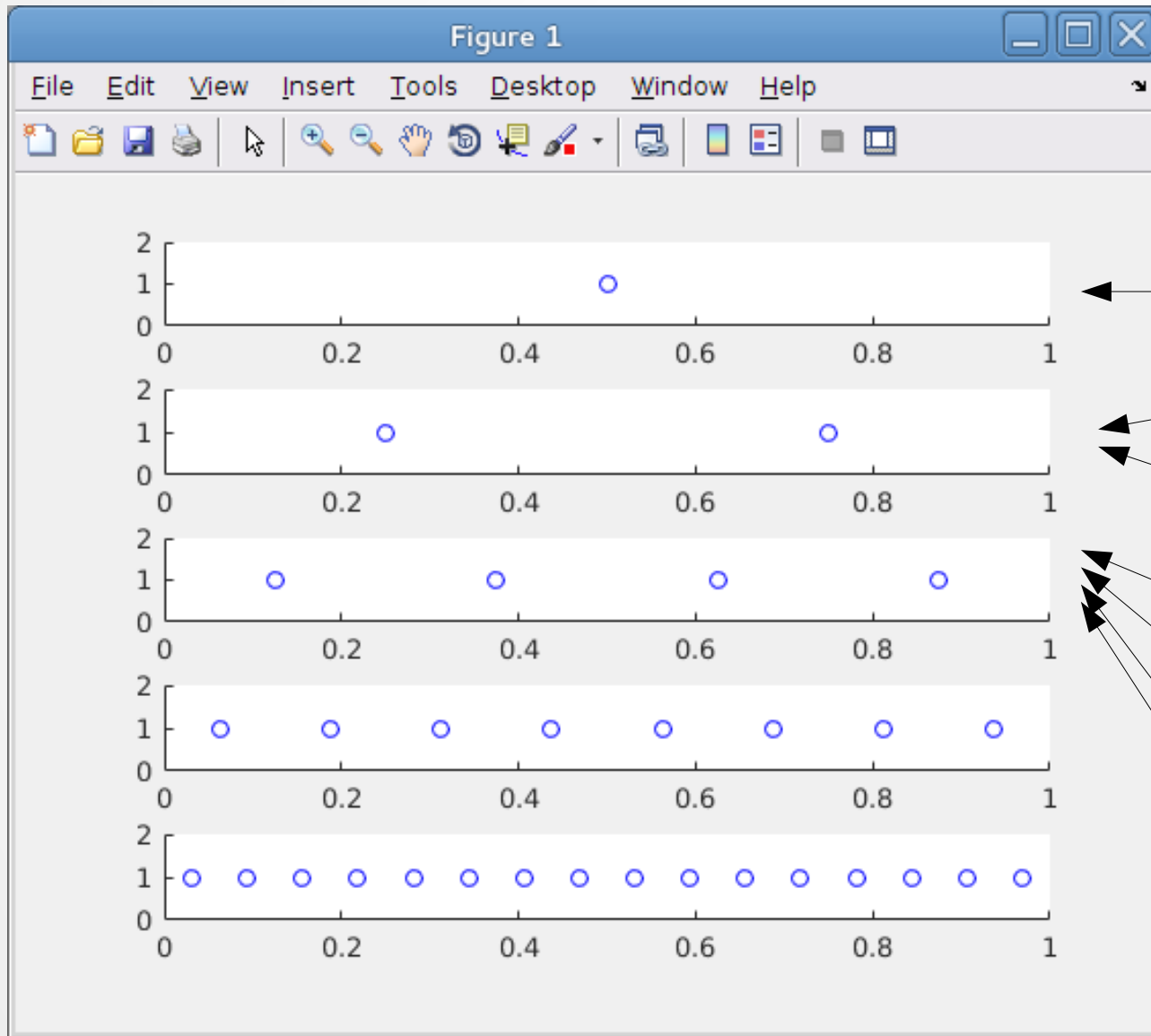
1. Choose base b
2. Choose max sequence length k
3. Write down sequence $1, 2, 3, \dots$ in base b using numbers of length $\log_b(k)$
4. Reverse the digits of your sequence
5. Translate back to your favorite working base (e.g. 10).

VanDerCorput algo for base 2, 3 digits

- Goal: Fill unit interval with sample points

Start position (dec)	Binary representation		Flipped binary representation	Output sequence (dec)
1/8	0.001		0.100	4/8 =1/2
2/8	0.010		0.010	2/8 =1/4
3/8	0.011		0.110	6/8 =3/4
4/8	0.100		0.001	1/8
5/8	0.101		0.101	5/8
6/8	0.110		0.011	3/8
7/8	0.111		0.111	7/8


Demo – base 2



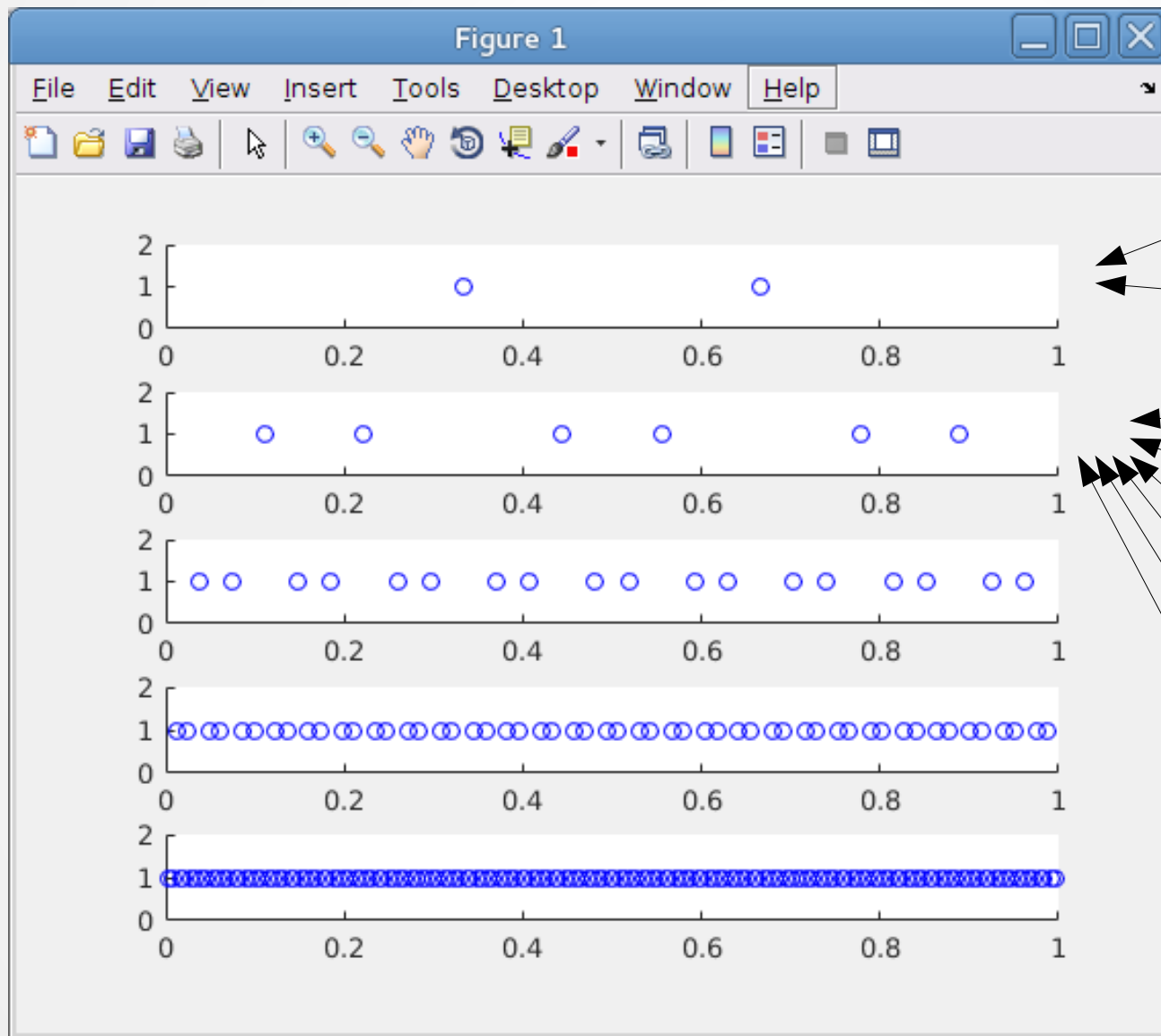
Flipped binary representation Output sequence

0.100	$\frac{4}{8}$ $=\frac{1}{2}$
0.010	$\frac{2}{8}$ $=\frac{1}{4}$
0.110	$\frac{6}{8}$ $=\frac{3}{4}$
0.001	$\frac{1}{8}$
0.101	$\frac{5}{8}$
0.011	$\frac{3}{8}$
0.111	$\frac{7}{8}$

VanDerCorput algo for base 3, 2 digits

Start position (dec)	Base 3 representaiton		Flipped base 3 representation	Output sequence (dec)
1/9	0.01		0.10	3/9 =1/3
2/9	0.02		0.20	6/9 =2/3
3/9	0.10		0.01	1/9
4/9	0.11		0.11	4/9
5/9	0.12		0.21	7/9
6/9	0.20		0.02	2/9
7/9	0.21		0.12	5/9
8/9	0.22		0.22	8/9

Base 3



Flipped base 3 representation Output sequence (dec)

0.10	$\frac{3}{9}$ $=\frac{1}{3}$
0.20	$\frac{6}{9}$ $=\frac{2}{3}$
0.01	$\frac{1}{9}$
0.11	$\frac{4}{9}$
0.21	$\frac{7}{9}$
0.02	$\frac{2}{9}$
0.12	$\frac{5}{9}$
0.22	$\frac{8}{9}$

Halton sequence

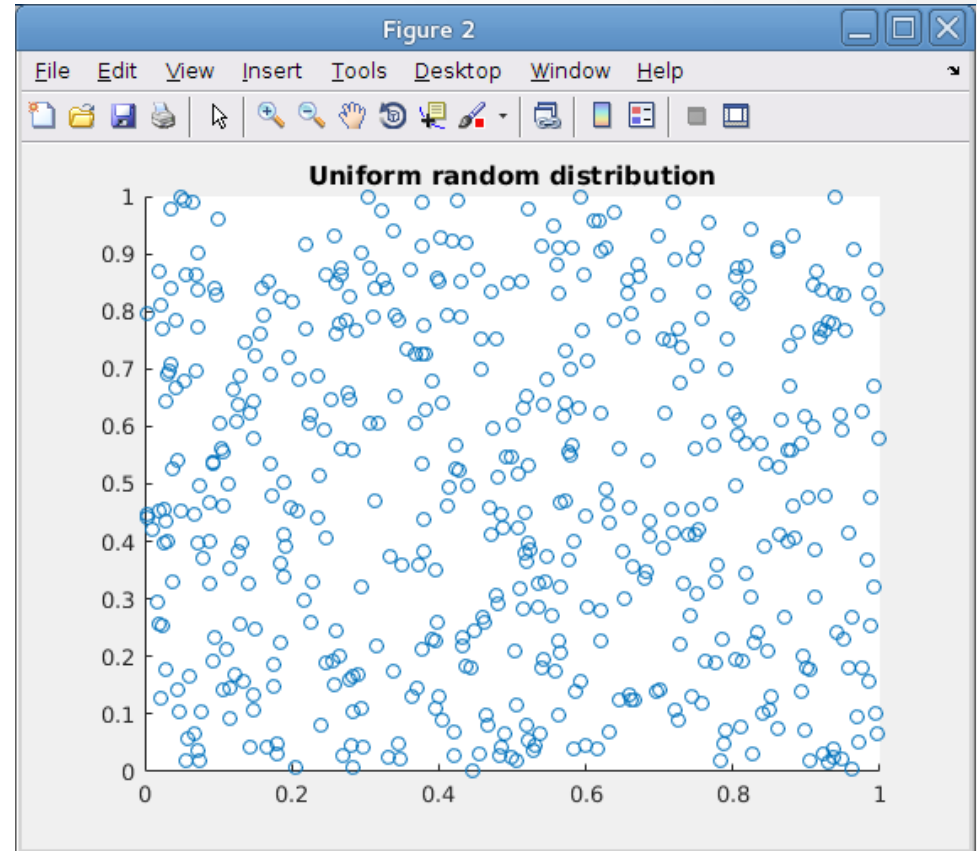
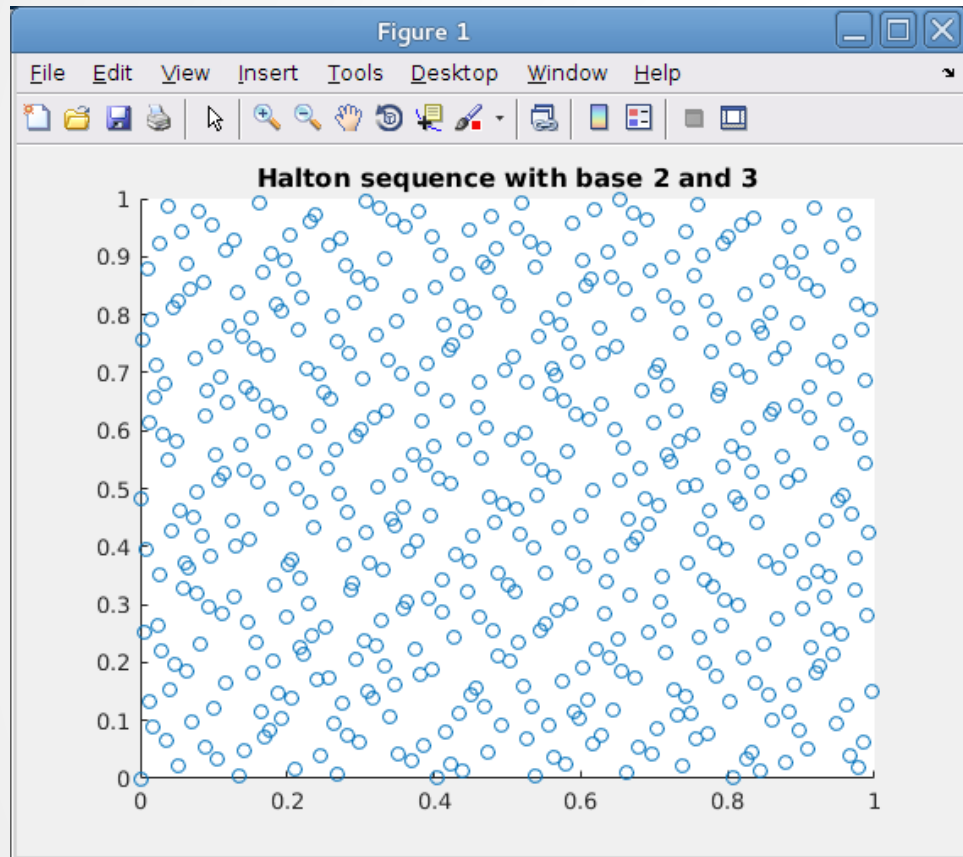
- van der Corput is nice 1D sequence which fills unit interval uniformly. But what about multiple dimensions?
- Halton sequence: use successive prime bases to fill successive dimensions.

```
function [x, y] = halton2(N)
    % Returns the first N elements of a Halton
    % sequence in 2 dimensions. We use the first
    % two primes, 2 and 3, to generate the sequence.

    x = vdcorput(N, 2);
    y = vdcorput(N, 3);

end
```

Halton vs. uniform random sampling



Another pi computation: Buffon needle problem

- You have a bunch of needles (sticks) of length l .
- You have a floor with boards separated by lines.
- Throw the needles on the floor. They land in random orientations.
- What is the probability a needle will cross a line in the floor?

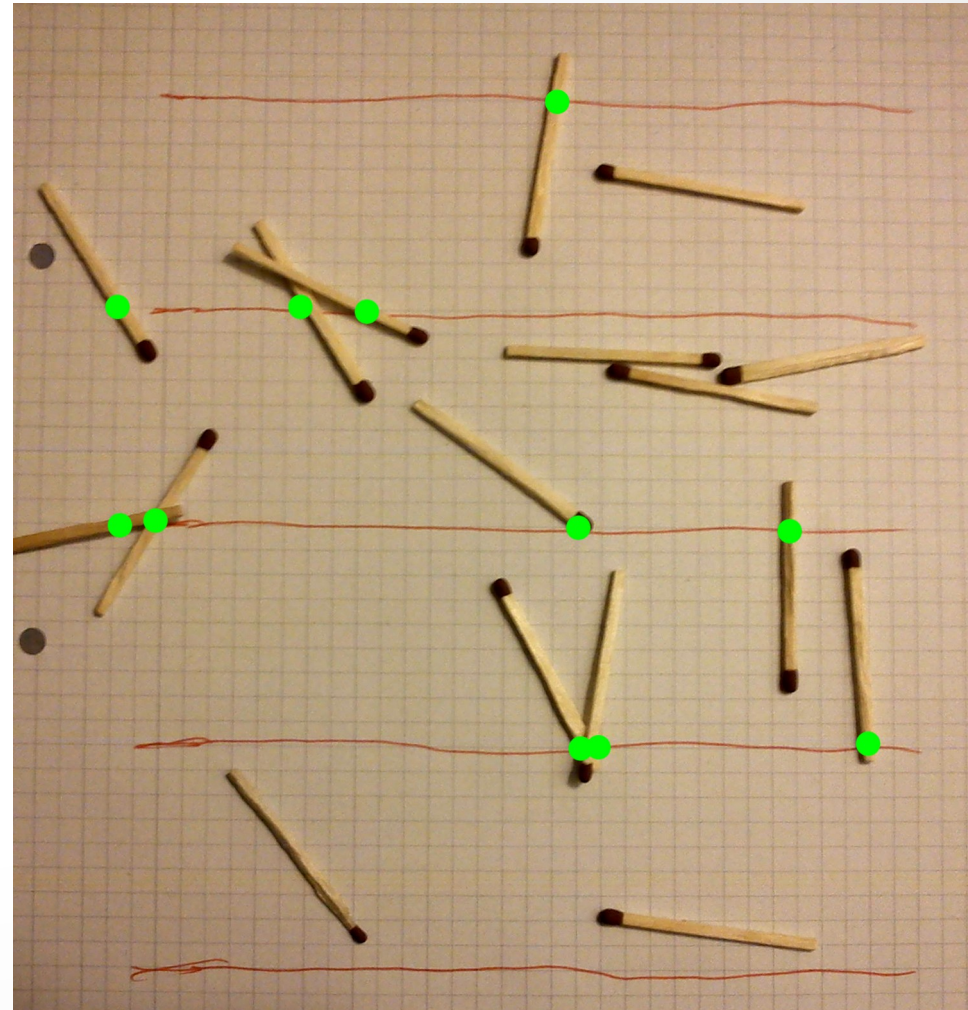


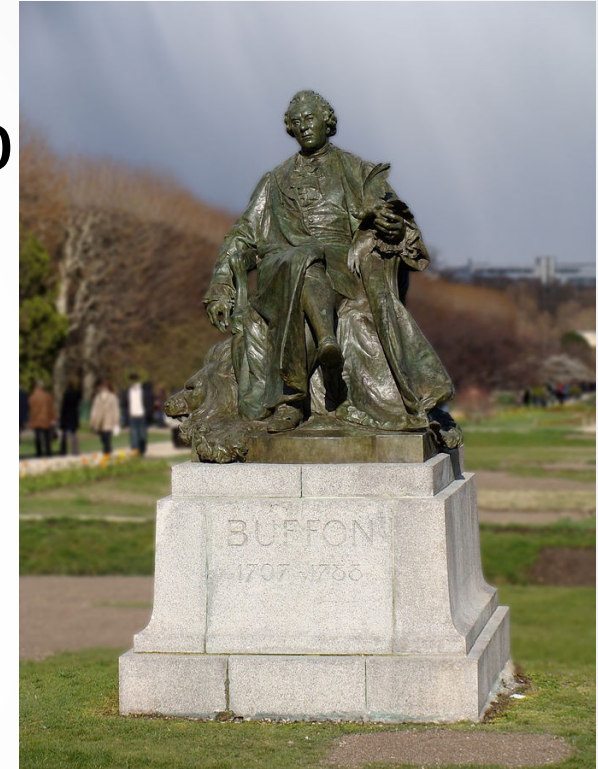
Image source: Wikipedia

The Comte de Buffon and his problem



Georges-Louis Leclerc, Comte de Buffon
7 September 1707 – 16 April 1788

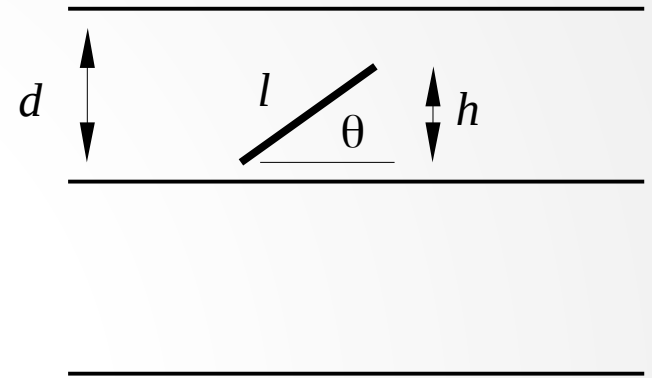
- Ingredients for a good Monte Carlo simulation:
 - Inspired by complicated physics (i.e. not trivial to solve analytically).
 - Process can be sampled using randomness.



Statue in Jardin des Plantes, Paris.

Analytic solution

- Take d = line separation,
 l = needle length.
- Take $l < d$ for simplicity
- Given an angle, probability
the needle touches one of
the lines:



$$p_{touch}(\theta) = \frac{h}{d} = \frac{l \sin \theta}{d}$$

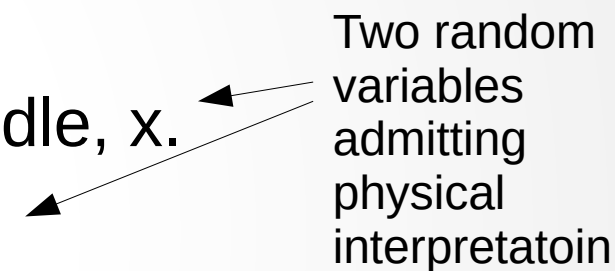
- Integrate over all angles to get total probability

$$p_{touch}(total) = \frac{1}{\pi} \int_0^{\pi} \frac{l \sin \theta}{d}$$

$$\begin{aligned}
 p_{touch}(total) &= \frac{1}{\pi} \int_0^{\pi} \frac{l \sin \theta}{d} \\
 &= \frac{l}{\pi d} (-\cos \theta) \Big|_0^{\pi} \\
 &= \frac{l}{\pi d} (-(-1 - 1)) \\
 &= \frac{2l}{\pi d}
 \end{aligned}$$

- Note that if I can find $p_{touch}(total)$ via experiment, I can determine the value of pi experimentally.

Algorithm for Monte Carlo Buffon needle

1. Initialize $s = 0$ % Running sum of needles straddling line in floor.
 2. Loop:
 3. Generate random center position of needle, x .
 4. Generate random orientation of needle,
 5. Compute positions of ends, x_1 and x_2 .
 6. Check if $\text{floor}(x_1)$ and $\text{floor}(x_2)$ are different. If they straddle a line, then the values are different.
 7. If they straddle, $s = s+1$
 8. Continue to next iteration
 9. At end of iterations, return number of straddles/number of trials.
- 
- Two random variables admitting physical interpretation

Code

```
function cnt = buffon_needle_trials(N, L, T)

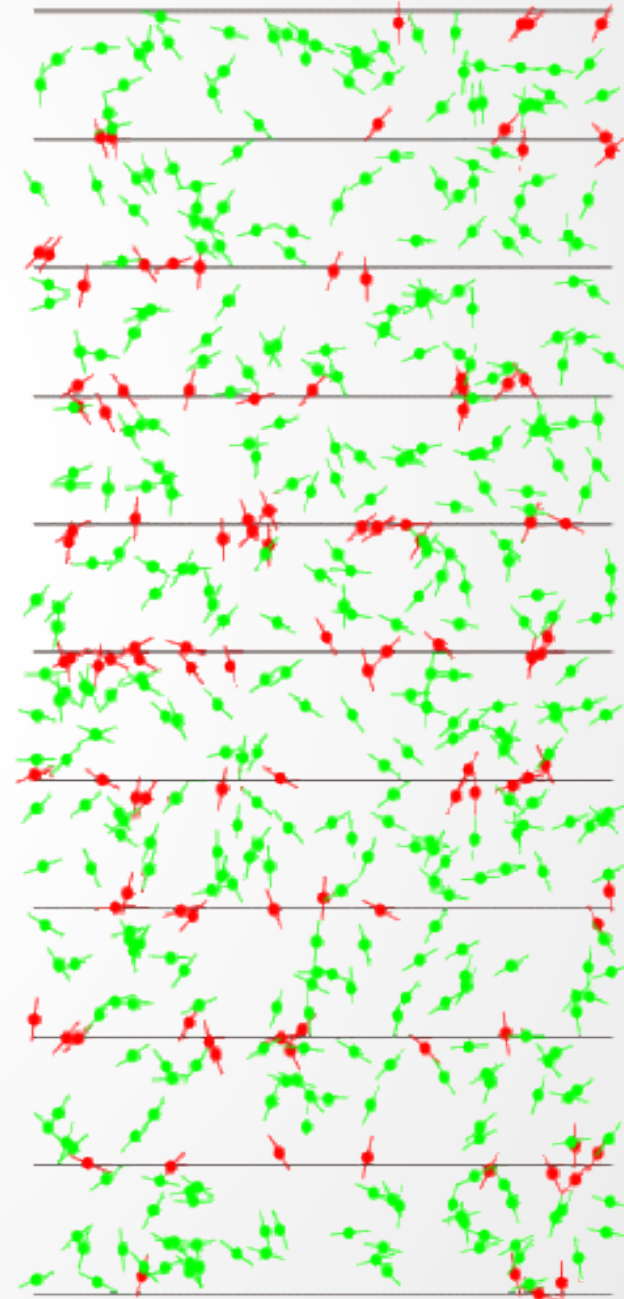
% We assume the floor has 10 lines in it at positions
% 0, 1, 2, ... 9.
% Create random vector of needle center positions
% after N throws.
x = 9*T*rand(N, 1);

% Now create random vector of needle angles (between
% 0 and 2*pi)
theta = 2*pi*rand(N, 1);

% Create vector which contains start and end x coords
% of all needles.
x1 = x - (L/2)*cos(theta);
x2 = x + (L/2)*cos(theta);

% Count number of needles whose ends cross a line in
% the floor.
hits = floor(x1) ~= floor(x2);
cnt = sum(hits);

end
```



Test

```
function mypi = test_buffon_needle()
    Ns = floor(power(10, 1:.5:7))';
    errs = zeros(length(Ns), 1);

    for i = 1:length(Ns)
        N = Ns(i);      % Number of trials
        L = 0.8;         % Length of needle
        T = 1;           % Distance between floor cracks

        p = buffon_needle_trials(N, L, T)/N;

        mypi = (2*L)/(T*p);

        fprintf('For N = %d, computed value of pi = %f\n', N, mypi)
        errs(i) = abs(mypi - pi);
    end

    figure(1)
    loglog(Ns, errs, 'bo')
    xlabel('Number of trials')
    ylabel('Error')
    title('Buffon needle Monte Carlo simulation')

    % Do linear regression to fit line to data
    logNs = log(Ns);
    logErrs = log(errs);
    PP = polyfit(logNs, logErrs, 1);

    fprintf('This trial set converges to pi as N^%f\n', PP(1))

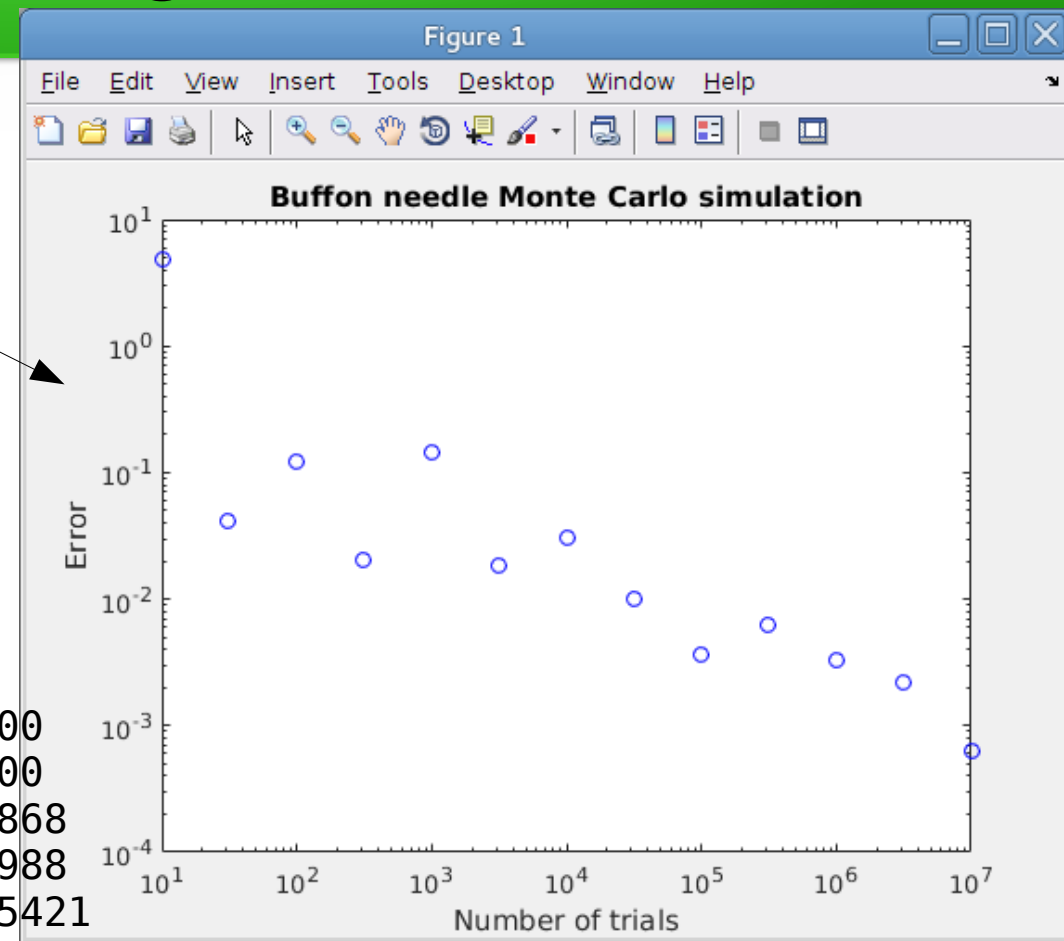
end
```

Monte Carlo convergence behavior

Relationship looks roughly linear on log-log plot. By eye, number of trials varies by 6 orders-of-magnitude, error varies by 3 orders-of-magnitude. Slope is negative. This implies:

$$err \propto N^{-1/2}$$

```
>> test_buffon_needle;
For N = 10, computed value of pi = 8.000000
For N = 31, computed value of pi = 3.100000
For N = 100, computed value of pi = 3.018868
For N = 316, computed value of pi = 3.120988
For N = 1000, computed value of pi = 3.285421
For N = 3162, computed value of pi = 3.122963
For N = 10000, computed value of pi = 3.172086
For N = 31622, computed value of pi = 3.151563
For N = 100000, computed value of pi = 3.145210
For N = 316227, computed value of pi = 3.147947
For N = 1000000, computed value of pi = 3.144877
For N = 3162277, computed value of pi = 3.139389
For N = 10000000, computed value of pi = 3.142219
This trial set converges to pi as N^-0.450292
```



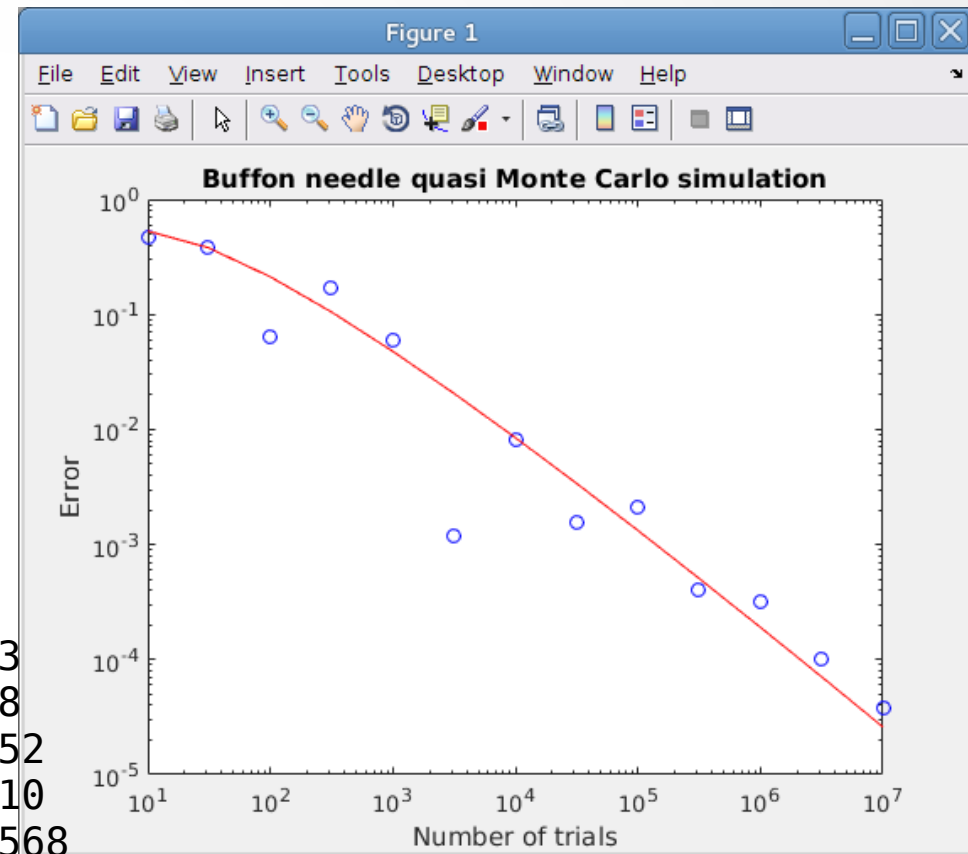
Program computes exponent using linear regression

Quasi-Monte Carlo Buffon needle

- Convergence:

$$err \propto \frac{(\log N)^D}{N}$$

```
>> test_buffon_needle
For N = 10, computed value of pi = 2.666667
For N = 31, computed value of pi = 2.755556
For N = 100, computed value of pi = 3.076923
For N = 316, computed value of pi = 2.974118
For N = 1000, computed value of pi = 3.082852
For N = 3162, computed value of pi = 3.140410
For N = 10000, computed value of pi = 3.133568
For N = 31622, computed value of pi = 3.143145
For N = 100000, computed value of pi = 3.139471
For N = 316227, computed value of pi = 3.141185
For N = 1000000, computed value of pi = 3.141277
For N = 3162277, computed value of pi = 3.141490
For N = 10000000, computed value of pi = 3.141555
This trial set converges to pi as N^-0.684361
Linear fit to theoretical err has slope = -0.741736
```



Quasi-Monte Carlo

- Uniform sampling of domain using low-discrepancy sequences.

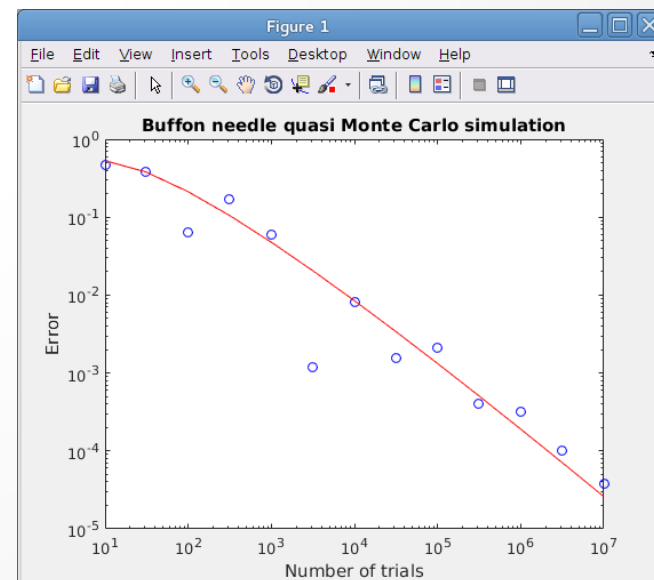
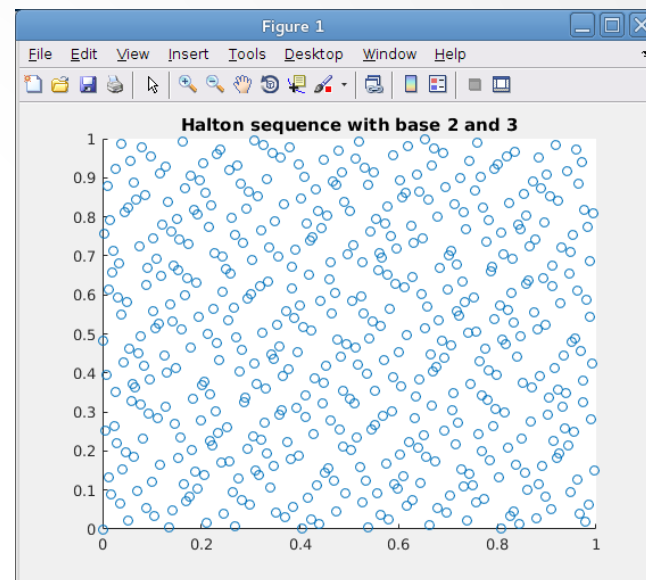
- Convergence:
$$err \propto \frac{(\log N)^D}{N}$$

- Trade-off in terms of algorithm performance.

- Generating the quasi-random sequence impacts performance.

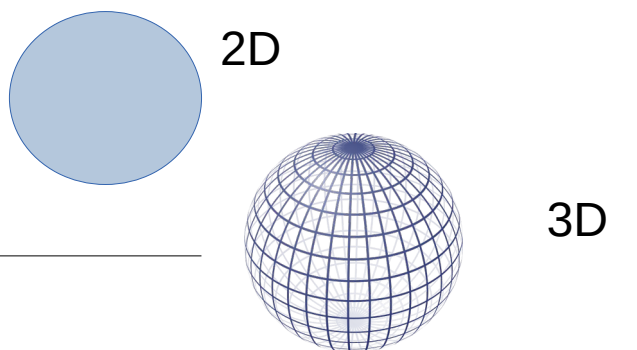
- Other sequences exist too...

- Sobol
 - Faure
 - Niederreiter
 - Etc.



Monte Carlo integration in higher dimensions

- 2 D case is trivial.
 - Don't actually need Monte Carlo – just used as example.
- However, going to ND is more interesting.
- Consider computing volume of N dimensional ball:



The diagram illustrates the progression from 2D to 3D and then to 4D. A solid blue circle represents a 2D ball, with an arrow pointing from the formula $V_2 = \pi R^2$ to it. A wireframe sphere represents a 3D ball, with an arrow pointing from the formula $V_3 = \frac{4}{3} \pi R^3$ to it. Below these, the text $V_4 = ???$ is followed by a long arrow pointing to the right, where the text '?????' and '4D' are located.

$$V_2 = \pi R^2 \quad \leftarrow \quad \text{2D}$$
$$V_3 = \frac{4}{3} \pi R^3 \quad \leftarrow \quad \text{3D}$$
$$V_4 = ??? \quad \leftarrow \quad \text{?????} \quad \text{4D}$$

Volume of n-dimensional ball

- Volume is known analytically:

$$V_d = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} R^d$$

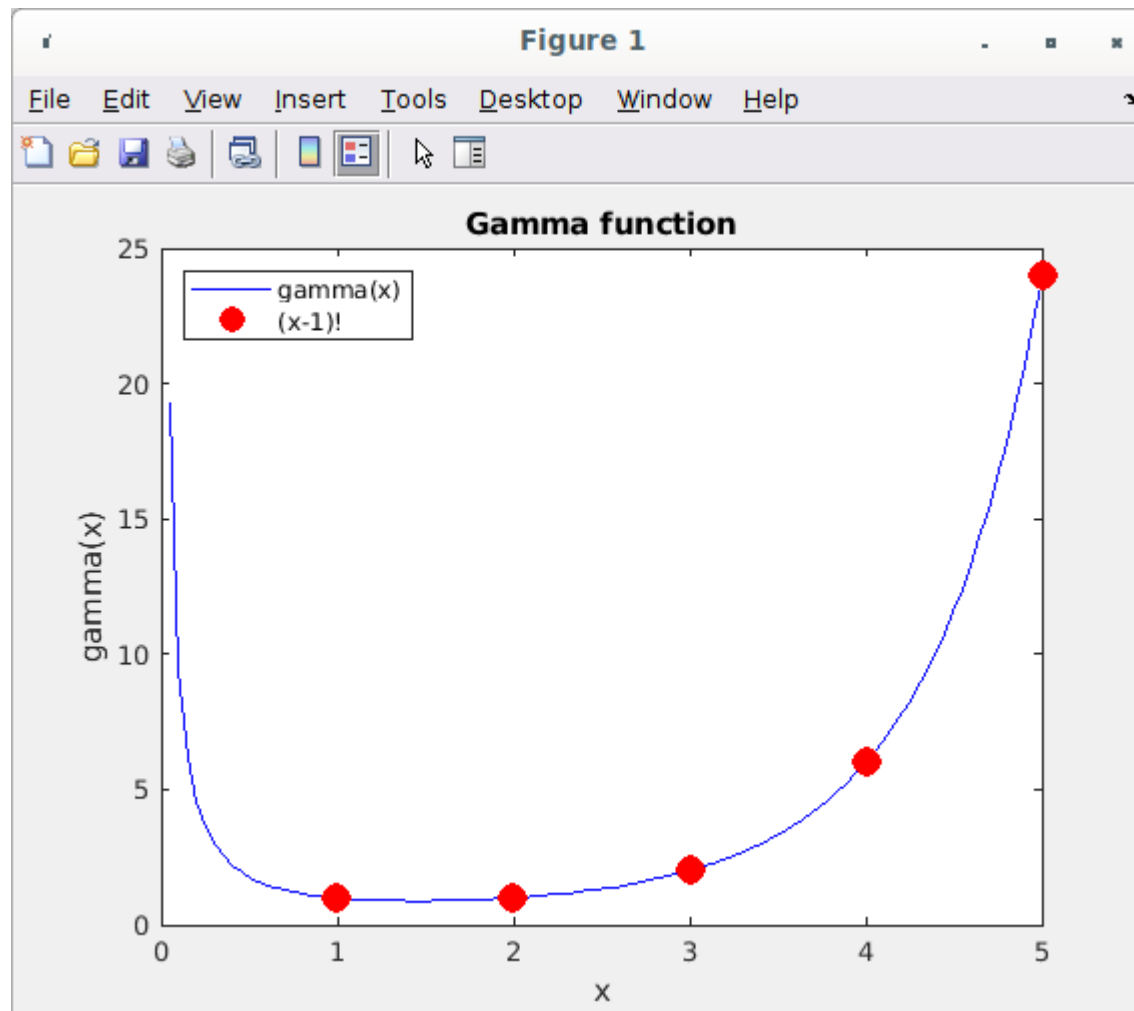
Note: Box counting will scale badly -- $O(N^d)$

- Expression involves gamma function

$$\Gamma(x) = \int_0^{\infty} dt t^{x-1} e^{-t}$$

Gamma function

- Generalization of the factorial to non-integers.



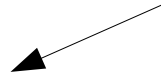
A few properties of the gamma function

- Generalization of factorial to non-integers:

$$\Gamma(n) = (n-1)! \quad \text{for integer } n$$

- Recursion relation:

$$\Gamma(x) = (x-1) \Gamma(x-1)$$

$$n! = n(n-1)!$$


- Some useful values:

$$\begin{array}{lll} \Gamma(1) = 1 & \Gamma\left(\frac{1}{2}\right) = \sqrt{\pi} & \Gamma\left(\frac{5}{2}\right) = \left(\frac{3}{2}\right)\left(\frac{1}{2}\right)\sqrt{\pi} \\ \Gamma(2) = 1 & & \\ \Gamma(3) = 2 & \Gamma\left(\frac{3}{2}\right) = \frac{1}{2}\sqrt{\pi} & \end{array}$$

Check the volume expression....

$$V_d = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} R^d$$

- $n = 2$

$$V_2 = \frac{\pi^{2/2}}{\Gamma\left(\frac{2}{2} + 1\right)} R^2 = \pi R^2$$

- $n = 3$


$$V_3 = \frac{\pi^{3/2}}{\Gamma\left(\frac{3}{2} + 1\right)} R^3 = \frac{4}{3} \pi R^3$$

- $n = 4$

$$V_4 = \frac{\pi^{4/2}}{\Gamma\left(\frac{4}{2} + 1\right)} R^4 = \frac{1}{2} \pi^2 R^4$$

Find volume of N-ball

- Use Monte Carlo approach
- Algorithm:
 1. Input: dimension to compute N
 2. $S = 0$ % Hit counter
 3. Loop T times
 4. Generate random vector p with elements in $[0, 1]$
 5. If $\text{norm}(p) < 1$ then $S = S+1$
 6. End Loop
 7. Return $Pi = 2^N * S/T$



Need 2^N factor because we are working in an N-cube whose sides have length 2.

Code

```
function V = monte_carlo_vol(N)
    % This fcn computes the volume of the unit ball in
    % N dimensions using Monte Carlo integration.

    T = 100000000; % Number of trials
    S = 0;          % Number of hits inside N-ball

    for i = 1:T
        p = rand(N, 1); % point is described by N-vector
        if (norm(p, 2) < 1) % Euclidian (2-norm) by default
            S = S + 1;
        end
    end

    V = (2^N)*S/T;

end
```

Test

```
function test_monte_carlo_vol()

    for n = 2:20
        Vcomp = monte_carlo_vol(n);
        Vtrue = pi^(n/2)/gamma(n/2+1);

        diff = Vcomp - Vtrue;
        relerr = diff/Vtrue;
        fprintf('n = %d, Vcomp = %f, Vtrue = %f diff = %f,\n', n, Vcomp, Vtrue, diff, relerr)
    end

end
```

T = 10,000,000

```
>> test_monte_carlo_vol
n = 2, Vcomp = 3.141030, Vtrue = 3.141593 diff = -0.000562, relerr = -1.789709e-04
n = 3, Vcomp = 4.190934, Vtrue = 4.188790 diff = 0.002143, relerr = 5.116979e-04
n = 4, Vcomp = 4.937830, Vtrue = 4.934802 diff = 0.003028, relerr = 6.136415e-04
n = 5, Vcomp = 5.264234, Vtrue = 5.263789 diff = 0.000445, relerr = 8.446123e-05
n = 6, Vcomp = 5.170573, Vtrue = 5.167713 diff = 0.002860, relerr = 5.534402e-04
n = 7, Vcomp = 4.726554, Vtrue = 4.724766 diff = 0.001788, relerr = 3.783531e-04
n = 8, Vcomp = 4.072627, Vtrue = 4.058712 diff = 0.013915, relerr = 3.428446e-03
n = 9, Vcomp = 3.293235, Vtrue = 3.298509 diff = -0.005274, relerr = -1.598814e-03
n = 10, Vcomp = 2.561843, Vtrue = 2.550164 diff = 0.011679, relerr = 4.579768e-03
n = 11, Vcomp = 1.878630, Vtrue = 1.884104 diff = -0.005473, relerr = -2.905084e-03
n = 12, Vcomp = 1.324237, Vtrue = 1.335263 diff = -0.011026, relerr = -8.257527e-03
n = 13, Vcomp = 0.944538, Vtrue = 0.910629 diff = 0.033909, relerr = 3.723674e-02
n = 14, Vcomp = 0.619315, Vtrue = 0.599265 diff = 0.020051, relerr = 3.345880e-02
n = 15, Vcomp = 0.416154, Vtrue = 0.381443 diff = 0.034710, relerr = 9.099733e-02
n = 16, Vcomp = 0.209715, Vtrue = 0.235331 diff = -0.025615, relerr = -1.088487e-01
n = 17, Vcomp = 0.157286, Vtrue = 0.140981 diff = 0.016305, relerr = 1.156559e-01
n = 18, Vcomp = 0.131072, Vtrue = 0.082146 diff = 0.048926, relerr = 5.956003e-01
n = 19, Vcomp = 0.052429, Vtrue = 0.046622 diff = 0.005807, relerr = 1.245603e-01
n = 20, Vcomp = 0.104858, Vtrue = 0.025807 diff = 0.079051, relerr = 3.063163e+00
```

- Good results for small N.
- Bad results for higher dimensionality
 - Need to run more trials
 - Illustrates difficulties of working in high dimensions

$T = 100,000,000$

```
>> tic; test_monte_carlo_vol; toc
```

```
n = 2, Vcomp = 3.141843, Vtrue = 3.141593 diff = 0.000251, relerr = 7.975140e-05
n = 3, Vcomp = 4.189234, Vtrue = 4.188790 diff = 0.000444, relerr = 1.059483e-04
n = 4, Vcomp = 4.935257, Vtrue = 4.934802 diff = 0.000455, relerr = 9.215353e-05
n = 5, Vcomp = 5.264762, Vtrue = 5.263789 diff = 0.000973, relerr = 1.847692e-04
n = 6, Vcomp = 5.168634, Vtrue = 5.167713 diff = 0.000921, relerr = 1.781871e-04
n = 7, Vcomp = 4.726095, Vtrue = 4.724766 diff = 0.001329, relerr = 2.813662e-04
n = 8, Vcomp = 4.053893, Vtrue = 4.058712 diff = -0.004819, relerr = -1.187324e-03
n = 9, Vcomp = 3.300695, Vtrue = 3.298509 diff = 0.002186, relerr = 6.627653e-04
n = 10, Vcomp = 2.559570, Vtrue = 2.550164 diff = 0.009406, relerr = 3.688343e-03
n = 11, Vcomp = 1.879552, Vtrue = 1.884104 diff = -0.004552, relerr = -2.415939e-03
n = 12, Vcomp = 1.345372, Vtrue = 1.335263 diff = 0.010109, relerr = 7.571087e-03
n = 13, Vcomp = 0.906199, Vtrue = 0.910629 diff = -0.004430, relerr = -4.864457e-03
n = 14, Vcomp = 0.589988, Vtrue = 0.599265 diff = -0.009277, relerr = -1.548012e-02
n = 15, Vcomp = 0.387318, Vtrue = 0.381443 diff = 0.005874, relerr = 1.540066e-02
n = 16, Vcomp = 0.262799, Vtrue = 0.235331 diff = 0.027469, relerr = 1.167240e-01
n = 17, Vcomp = 0.144179, Vtrue = 0.140981 diff = 0.003198, relerr = 2.268455e-02
n = 18, Vcomp = 0.083886, Vtrue = 0.082146 diff = 0.001740, relerr = 2.118418e-02
n = 19, Vcomp = 0.047186, Vtrue = 0.046622 diff = 0.000564, relerr = 1.210424e-02
n = 20, Vcomp = 0.010486, Vtrue = 0.025807 diff = -0.015321, relerr = -5.936837e-01
Elapsed time is 6015.447327 seconds.
```

Error should – on average – be $1/\sqrt{10}$ lower than for
 $T = 10,000,000$ (last page)

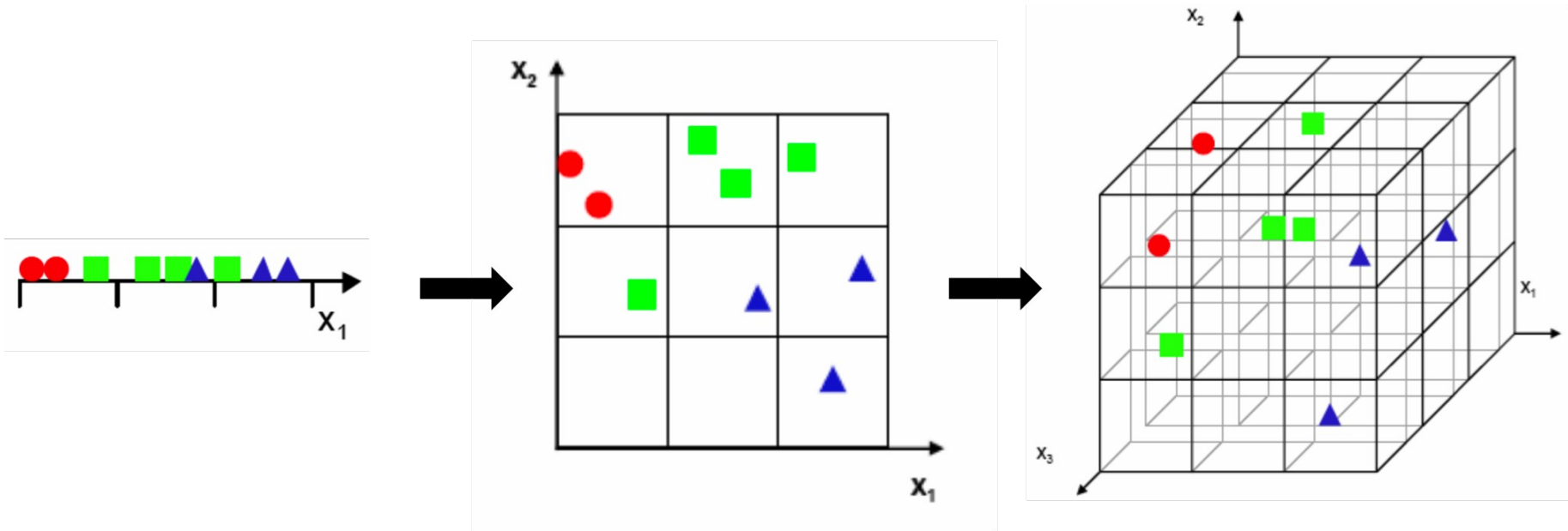
The Curse of Dimensionality

- Some problems scale as $O(N^D)$. (Think of box counting.)
- Volume of space increases as N^D .
- This is a big problem for machine learning algorithms, where it's common to work problems in high-dimensional spaces.

The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance.

-- Wikipedia

The curse of dimensionality



- Objects are densely packed in 1D
- Objects are spread out in 3D
- Imagine what happens in ND, for large D
- Box counting scales as $O(N^D)$
- Monte Carlo scales better, but sampling is sparse.
 - Sparse sampling is sometimes OK.

Session summary

- Compute pi using Monte Carlo (integrating the 1-function)
 - Monte Carlo convergence: $1/\sqrt{N}$
- Quasi-Monte Carlo methods
 - Importance of uniform sampling
 - Generating low-discrepancy sequences for uniform sampling
- Buffon needle problem computed using naive- and quasi-Monte Carlo method.
 - Quasi-Monte Carlo convergence: $(\log N)^D / N$
- Computed volume of N-ball using Monte Carlo
 - Curse of dimensionality