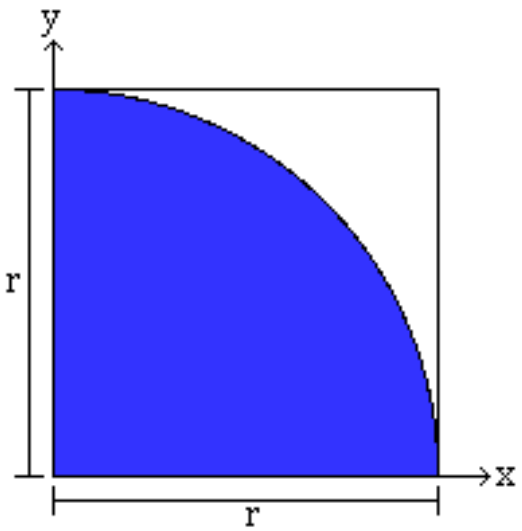
Monte Carlo Algorithms

- Broad class of algorithms
- Rely on repeated use of random numbers
- Often used in simulations of real-world events
- Also used for numerical integration, particularly for higher order integrals

Motivating Monte Carlo algorithms: Estimating π

- Randomly generate n (x,y) values in the range (0,1) from a uniform distribution
  - Set radius r = 1
  - Determine the ratio:
    - d = (number within the circle)/n
  - The ratio of the areas is: $\pi/4$
  - Estimate π by 4d.



- Estimate improves as n increases.

Consider $\int_0^1 f(x)dx$

- By the Mean Value Theorem for Integrals, there exists c Є(0,1) such that

$$\int_0^1 f(x)dx = f(c).$$

- f(c) is the average value of f over (0,1)
- How does this help?

For example, $\int_0^1 x \sin x \, dx$

- Actual solution: $(-x\cos x + \sin x)_0{}^1$=0.3011
- Midpoint rule: 0.2397
- Trapezpoid rule: 0.4207
- Simpson's rule: 0.3001

Some Monte Carlo Results (actual answer = 0.3011…)

n					
1	0.1715	0.0399	0.2025	0.1887	0.6316
3	0.0124	0.2382	0.1334	0.0886	0.1714
5	0.4766	0.0725	0.2886	0.3901	0.1994
10	0.2267	0.3179	0.2786	0.2165	0.3101
100	0.3307	0.2826	0.2144	0.2980	0.2845
1000	0.3105	0.3003	0.3214	0.2987	0.3034
10000	0.3036	0.3026	0.3069	0.3058	0.3058

Convergence Results

- It will converge:

$$\widehat{I_n} = \frac{1}{n}\sum_{j=1}^{n} f(x_j)$$

$$\lim_{n\to\infty} \widehat{I_n} = I = \int_0^1 f(x)dx$$

- However, the error for Monte Carlo is proportional to $1/\sqrt{n}$

Consider the double integral: $\int_0^1 \int_0^1 f(x,y)dxdy$

- Use simple Simpson's Rule to approximate over each variable individually
- … (board work)

- Follow-up:
- How many function evaluations are needed for composite rule, if we want to divide x-interval into n pieces, and y-interval into n pieces?

Monte Carlo for Multiple Integrals

- $\int_0^1 \int_0^1 f(x,y)dxdy \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i)$
- $\int_0^1 \int_0^1 \int_0^1 f(x,y,z)dxdydz \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i, z_i)$

- For the double integral:
  - Generate 2n random values: r_k over (0,1)
  - Choose $x_i=r_{2k}$, $y_i = r_{2k+1}$
  - Evaluate function at n points and compute average, as indicated.
- Generalize for more variables

How about other intervals?

- Recall from before: for some c Є(0,1)

$$\int_0^1 f(x)dx = f(c).$$

- This can be generalized to: for some c Є(a,b)

$$\int_a^b f(x)dx = (b-a)f(c)\,.$$

- In the algorithm: random values should be uniformly distributed in the interval (a,b) rather than (0,1).

Multiple integrals with Monte Carlo

- $\int_a^b \int_c^d f(x,y)dxdy \approx (b-a)(d-c)\frac{1}{n}\sum_{i=1}^{n} f(x_i,y_i)$
- $\int_a^b \int_c^d \int_p^q f(x,y,z)dxdydz$

$$\approx (b-a)(d-c)(q-p)\frac{1}{n}\sum_{i=1}^{n} f(x_i,y_i,z_i)$$

- Error remains proportional to $1/\sqrt{n}$
- As $n \to \infty$, the approximation will converge.

Example: $\int_0^{9/10}\int_0^{1}\int_0^{11/10}\sqrt{4-x^2-y^2-z^2}\,dxdydz$

$$\approx \frac{9}{10}\times 1\times\frac{11}{10}\times\frac{1}{n}\sum_{i=1}^{n}\sqrt{4-{x_i}^2-{y_i}^2-{z_i}^2}$$

Where

- $x_i\epsilon(0,\frac{9}{10}),\quad y_i\epsilon(0,1),\quad z_i\epsilon(0,\frac{11}{10}),$
- for *i = 1:n*

Some Monte Carlo Results
(actual answer = 1.705759…)

n					
10	1.6721	1.8058	1.6777	1.7846	1.7235
100	1.6938	1.6910	1.6905	1.7020	1.7056
1000	1.7000	1.7084	1.7082	1.7067	1.7105
10000	1.7068	1.7050	1.7062	1.7070	1.7016
100000	1.7051	1.7049	1.7062	1.7058	1.7054
1000000	1.7057	1.7057	1.7056	1.7059	1.7056

"Random" Numbers?

- True random numbers
  - Can generate "true" random numbers based on events such as flipping a coin or throwing dice
  - Generally, time-consuming.
- Pseudo random numbers
  - Algorithms exist to generate numbers which appear random and hard to predict.
  - These sequences are generally sufficient for Monte Carlo techniques.
  - The algorithms are "seeded" to start the sequence.

Matlab's random number generator

- R = rand() returns a pseudorandom value drawn from the standard uniform distribution on the open interval (0,1).

Variations:

- R = rand(M,N) returns an M-by-N matrix
- R = rand(N) returns an N-by-N matrix
- R = rand(..., 'double') or R = rand(..., 'single') returns an array of uniform values of the specified class.

Notes:

- The sequence of numbers produced by rand is determined by the settings of the uniform random number generator that underlies rand, RANDI, and RANDN.
- Control that shared random number generator using RNG.

More on Matlab's **`rand`**

Example 1: Generate values from the uniform distribution on the interval [a, b]:

```
r = a + (b-a).*rand(100,1);
```

Example 2: Use the RANDI function, instead of rand, to generate integer values from the uniform distribution on the set 1:100.

```
r = randi(100,1,5);
```

Example 3: Reset the random number generator used by rand, RANDI, and RANDN to its default startup settings, so that rand produces the same random numbers as if you restarted MATLAB.

```
rng('default')
rand(1,5)
```

More on Matlab's **`rand`**

Example 4: Save the settings for the random number generator used by rand, RANDI, and RANDN, generate 5 values from rand, restore the settings, and repeat those values.

```
s = rng
u1 = rand(1,5)
rng(s);
u2 = rand(1,5) % contains same values as u1
```

Example 5: Reinitialize the random number generator used by rand, RANDI, and RANDN with a seed based on the current time. rand will return different values each time you do this.

```
rng('shuffle');
rand(1,5)
```