CO65: Monte Carlo Integration

Jintian Wang ( )
Date: 2026-04-30

1 AIM and Method

The aim of this project was to use Monte Carlo integration to estimate several quantities which are difficult or inconvenient to calculate directly. The three calculations were the volume of a unit sphere in ten dimensions, the mass, centre of mass and moments of inertia of a bounded toroidal section with constant density, and a two-dimensional integral over a disk.

The basic Monte Carlo method used here was to generate uniformly distributed random points inside a simple enclosing box, then test whether each point lay inside the required region. The required integral was then estimated by multiplying the volume or area of the enclosing box by the average value of the integrand over all generated points. Points outside the required region were treated as contributing zero.

For the ten-dimensional sphere, I generated N = 100000 points inside the hypercube -1 <= x_i <= 1. Points satisfying sum(x_i^2) <= 1 were accepted. The volume was calculated from the accepted fraction multiplied by the hypercube volume, 2^10.

For the toroidal section, I generated N = 10000 points inside the bounding box 1 <= x <= 4, -3 <= y <= 4, and -1 <= z <= 1, with volume 42. A point was accepted if it satisfied
z^2 + (sqrt(x^2 + y^2) - 3)^2 <= 1.

Assuming density rho = 1, the mass was estimated from the accepted fraction. The centre of mass was calculated from the sums of x, y and z over the accepted points, divided by the mass. The moments of inertia were calculated using the corresponding integrands y^2 + z^2, z^2 + x^2, and x^2 + y^2.

For the surface integral, I generated N = 100000 points in the square -0.5 <= x <= 0.5 and -0.5 <= y <= 0.5. Points inside the disk x^2 + y^2 <= 0.25 were accepted, and the integral of sin(ln(x + y + 1)) over the disk was estimated by summing the function values at accepted points and dividing by the total number of generated points.

2 Results

The numerical results from one run of the program are shown below. Since the method uses random sampling, the final digits vary between runs.

```
Part 1: 10D unit sphere
Estimated volume = 2.324480 +/- 0.154106
```

The ten-dimensional sphere calculation gave a value close to the expected order of magnitude. The uncertainty is relatively large compared with ordinary low-dimensional examples because the accepted fraction in ten dimensions is small. This is a useful feature of the problem, since it shows the slow convergence of simple Monte Carlo integration in high dimensions.

```
Part 2: toroidal section
Mass = 21.932400 +/- 0.209793
Centre of mass:
  x_cm = 2.397850 +/- 0.024990
  y_cm = 0.209765 +/- 0.025583
  z_cm = -0.003961 +/- 0.006897
Moments of inertia:
  Ixx = 80.908998 +/- 1.210270
  Iyy = 142.823577 +/- 1.723004
  Izz = 212.835789 +/- 2.228183
```

For the toroidal section, the center of mass has a positive x-coordinate because the section is restricted to x >= 1. The z-coordinate is close to zero, as expected from the symmetry of the shape about the plane z = 0. The moment of inertia values are different because the accepted volume is not symmetric in x and y after applying the bounding planes.

```
Part 3: disk integral
Integral estimate = -0.046820
```

The disk integral gave a small value because the integrand sin(ln(x + y + 1)) changes sign over the disk. The result is therefore sensitive to cancellation between positive and negative contributions, but it should become more stable as N is increased.

3 Conclusions

This project showed that a simple Monte Carlo method can be used to estimate volumes, masses, centres of mass, moments of inertia and surface integrals using the same general structure. The method only required random point generation, an acceptance condition, and a function to be summed over the accepted points.

The results were physically reasonable. The ten-dimensional sphere volume was obtained from the fraction of points lying inside the sphere. The toroidal section had a centre of mass consistent with its geometric restrictions, and its moment of inertia components reflected the non-symmetric bounding region. The disk integral was also estimated using the same accepted-point method.

The main limitation of this simple method is statistical error. Increasing the number of random points improves the result, but only approximately as 1/sqrt(N), so much larger sample sizes are needed for noticeably higher precision. This was most apparent in the ten-dimensional calculation, where only a small fraction of the generated points lay inside the sphere.

Appendix: my Code

```
% CO65_main.m
% Author: Jintian Wang, Date: 30/04/2026
%
% Monte Carlo integration for CO65.
% This script estimates:
%   1. the volume of a unit 10-dimensional sphere
%   2. the mass, centre of mass, and moments of inertia of a toroidal section
%   3. an integral over a disk
%
% Required helper functions:
%   get_accepted_points.m
%   calculate_sum.m


%% Part 1: volume of a unit 10-dimensional sphere

n = 100000;
D = 10;

% Points inside [-1, 1]^10
pst = 2 * rand(n, D) - 1;

%apply MC
acc_pst = get_accepted_points(pst, @(p) sum(p.^2) <= 1);

vol = 2^10;
R10 = size(acc_pst, 1) / n;
vol10 = vol * R10;

% err
err10 = vol * sqrt(R10 * (1 - R10) / n);

fprintf("\nPart 1: 10D unit sphere\n");
fprintf("Estimated volume = %.6f +/- %.6f\n", vol10, err10);



%% Part 2: toroidal section
fprintf("\nPart 2: toroidal section\n");
N = 10000;
```

```
% Bounding box:
x = 1 + 3 * rand(N, 1);
y = -3 + 7 * rand(N, 1);
z = -1 + 2 * rand(N, 1);

V = 42;
points = [x, y, z];

% MC applied
torus_func = @(p) p(3)^2 + (sqrt(p(1)^2 + p(2)^2) - 3)^2 <= 1;
acc_pts = get_accepted_points(points, torus_func); %accepted points

% CALCULATIONS
s_rho = size(acc_pts, 1);
s_rho2 = s_rho;
% rho = 1 inside, = 0 outside

% @(p)=acc_pts      p(1)=x     p(2)=y       p(3)=z
sx = calculate_sum(acc_pts, @(p) p(1));
sy = calculate_sum(acc_pts, @(p) p(2));
sz = calculate_sum(acc_pts, @(p) p(3));

sx2 = calculate_sum(acc_pts, @(p) p(1)^2);
sy2 = calculate_sum(acc_pts, @(p) p(2)^2);
sz2 = calculate_sum(acc_pts, @(p) p(3)^2);

sIxx = calculate_sum(acc_pts, @(p) p(2)^2 + p(3)^2);
sIyy = calculate_sum(acc_pts, @(p) p(3)^2 + p(1)^2);
sIzz = calculate_sum(acc_pts, @(p) p(1)^2 + p(2)^2);

sIxx2 = calculate_sum(acc_pts, @(p) (p(2)^2 + p(3)^2)^2);
sIyy2 = calculate_sum(acc_pts, @(p) (p(3)^2 + p(1)^2)^2);
sIzz2 = calculate_sum(acc_pts, @(p) (p(1)^2 + p(2)^2)^2);

% M
M = V * s_rho / N;
M_err = V * sqrt((s_rho2 / N - (s_rho / N)^2) / N);
fprintf("Mass = %.6f +/- %.6f\n", M, M_err);

% COM
xcm = V * sx / N / M;
ycm = V * sy / N / M;
zcm = V * sz / N / M;

xcm_err = V * sqrt((sx2 / N - (sx / N)^2) / N) / M;
ycm_err = V * sqrt((sy2 / N - (sy / N)^2) / N) / M;
zcm_err = V * sqrt((sz2 / N - (sz / N)^2) / N) / M;

fprintf("Centre of mass:\n");
fprintf("  x_cm = %.6f +/- %.6f\n", xcm, xcm_err);
fprintf("  y_cm = %.6f +/- %.6f\n", ycm, ycm_err);
fprintf("  z_cm = %.6f +/- %.6f\n", zcm, zcm_err);

% Inertia
Ixx = V * sIxx / N;
Iyy = V * sIyy / N;
Izz = V * sIzz / N;

Ixx_err = V * sqrt((sIxx2 / N - (sIxx / N)^2) / N);
Iyy_err = V * sqrt((sIyy2 / N - (sIyy / N)^2) / N);
Izz_err = V * sqrt((sIzz2 / N - (sIzz / N)^2) / N);

fprintf("Moments of inertia:\n");
fprintf("  Ixx = %.6f +/- %.6f\n", Ixx, Ixx_err);
fprintf("  Iyy = %.6f +/- %.6f\n", Iyy, Iyy_err);
fprintf("  Izz = %.6f +/- %.6f\n", Izz, Izz_err);


%% Part 3: surface integral over disk

N = 100000;

% -0.5 <= x <= 0.5, -0.5 <= y <= 0.5
xy = rand(N, 2) - 0.5;

%apply MC
acc_func= @(p) p(1)^2 + p(2)^2 <= 0.25;
acc_pst = get_accepted_points(xy,acc_func);

%integral
int_func = @(p) sin(log(p(1) + p(2) + 1));
sf = calculate_sum(acc_pst,int_func);

I = sf / N;

fprintf("\nPart 3: disk integral\n");
fprintf("Integral estimate = %.6f", I);
```

```
function [accepted_pts] = get_accepted_points(pts, cond_func)
% Author: JINTIAN WANG , Date: 30/04/2026
% Returns the points from pts that returns True in cond_func.
% Input:
% * pts: (N x D) matrix that contains N points with D dimension
% * cond_func: a function that accepts a point as a D-dimensional vector and returns true
% if a certain condition is fulfilled and false otherwise (see hint).
%
% Output:
% * accepted_pts: (M x D) that only returns D-dimensional points that fulfills the conditions
% by the cond_func function
%
% Example use:
% >> pts = rand(10000, 4); % 10000 points with 4 dimensions
% >> cond_func = @(pt) (sum(pt.^2, 2) < 1); % only accepts with radius < 1
% >> acc_pts = get_accepted_points(pts, cond_func);

n = size(pts, 1);
keep = false(n, 1);

for k = 1:n
    p = pts(k, :);
    keep(k) = cond_func(p);
end

accepted_pts = pts(keep, :);

end
```

```
function [val] = calculate_sum(acc_pts, func)
% Author: JINTIAN WANG , Date: 30/04/2025
% Calculate the sum of func(pt) for every pt in acc_pts and returns the value.
% Input:
% * acc_pts: the list of points accepted for the calculation with the form of a (M x D) matrix,
% and each row of the matrix corresponds to each point.
% * func: a function that accept a point and returns a value to be calculated.
%
% Output:
% * val: sum of func(pt) for every pt in acc_pts
%
% Example use:
% >> % Calculating the x-distance of centre of mass of the positive quadrant of a
% >> % 3-dimensional sphere with radius 1
% >> pts = rand(10000, 3); % 10000 points with 4 dimensions
% >> cond_func = @(pt) (sum(pt.^2, 2) < 1); % only accepts with radius < 1
% >> acc_pts = get_accepted_points(pts, cond_func);
% >> func = @(pt) pt(1); % get the x-coordinate of the poi

val = 0;

for k = 1:size(acc_pts, 1)
    p = acc_pts(k, :);
    val = val + func(p);
end

end
```