Московский Авиационный Институт
(национальный исследовательский университет)

Курсовая работа по курсу
«Численные методы»

Метод Монте-Карло для вычисления кратных интегралов

Студент:
Макаров Никита, 8о-306Б

Руководитель:
Ревизников Д.Л.

Москва
2015

Содержание

1 Введение

Методами Монте-Карло называют численные методы решения математических задач при помощи моделирования случайных величин. Однако, решать методами Монте-Карло можно любые математические задачи, а не только задачи вероятностного происхождения, связанные со случайными величинами.

Важнейшим приемом построения методов Монте-Карло является сведение задачи к расчету математических ожиданий. Так как математические ожидания чаще всего представляют собой обычные интегралы, то центральное положение в теории метода Монте-Карло занимают методы вычисления интегралов.

Преимущества недетерминированных методов особенно ярко проявляются при решении задач большой размерности, когда применение традиционных детерминированных методов затруднено или совсем невозможно.

До появления ЭВМ методы Монте-Карло не могли стать универсальными численными методами, ибо моделирование случайных величин вручную - весьма трудоемкий процесс. Развитию методов Монте-Карло способствовало бурное развитие ЭВМ. Алгоритмы Монте-Карло сравнительно легко программируются и позволяют производить расчеты во многих задачах, недоступных для классических численных методов. Так как совершенствование ЭВМ продолжается, есть все основания ожидать дальнейшего развития методов Монте-Карло и дальнейшего расширения области их применения.

2 Идея метода Монте-Карло

Важнейший прием построения методов Монте-Карло — сведение задачи к расчету математических ожиданий. Пусть требуется найти значение m некоторой изучаемой величины. С этой целью выбирают такую случайную величину X, математическое ожидание которой равно $m : M[X] = m$. Практически же поступают так: вычисляют N возможных значений x_i случайной величины X и находят их среднее арифметическое

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i.$$

Так как последовательность одинаково распределенных случайных величин, у которых существуют математические ожидания, подчиняется закону больших чисел, то при $N \to \infty$ среднее арифметическое этих величин сходится по вероятности к математическому ожиданию. Таким образом, при больших N величина $\bar{x} \approx m$.

В методе Монте-Карло данные вырабатываются искусственно путем использования некоторого генератора случайных чисел в сочетании с функцией распределения вероятностей для исследуемого процесса.

3 Погрешность метода Монте-Карло

Для оценки величины m смоделируем случайную величину X с математическим ожиданием $M[X] = m$. Выберем N независимых реализаций $x_1, ..., x_N$ случайной величины X и вычислим среднее арифметическое:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Также предположим, что случайная величина X имеет конечную дисперсию:

$$D[X] = M[X^2] - (M[X])^2.$$

Из центральной предельной теоремы имеем:

$$P\left\{|\bar{X} - m| < t_\beta\sqrt{\frac{D[X]}{N}}\right\} \approx 2\Phi(t_\beta) = \beta. \tag{1}$$

Из уравнения (1) получаем верхнюю границу ошибки с коэффициентом доверия β:

$$\varepsilon = t_\beta\sqrt{\frac{D[X]}{N}}. \tag{2}$$

Задав некоторое значение ε и β при известном $\sigma = \sqrt{D[X]}$ можно определить необходимое количество испытаний, обеспечивающее точность ε с надежностью β:

$$N = \frac{\sigma^2 \cdot t_\beta^2}{\varepsilon^2}. \tag{3}$$

Обычно при решении реальных задач значение дисперсии неизвестно, а следовательно неизвестен параметр σ. Чтобы определить приближенное значение σ проводят некоторое начальное количество испытаний N_0. Затем по результатам этих испытаний определяется приближенное значение дисперсии:

$$D[X] = \frac{1}{N_0}\sum_{i=1}^{N_0} x_i^2 - \left(\frac{1}{N_0}\sum_{i=1}^{N_0} x_i\right)^2. \tag{4}$$

Зная значение $D[X]$ можем получить приближенное значение N:

$$N = \frac{(D[X])^2 \cdot t_\beta^2}{\varepsilon^2}. \tag{5}$$

4 Вычисление интегралов усреднением подынтегральной функции

Пусть требуется вычислить интеграл

$$I = \int\limits_a^b \varphi(x)dx. \tag{6}$$

Предположим, что имеется случайная величина X, равномерно распределенная на интервале (a, b) с плотностью вероятности $f(x) = \frac{1}{b-a}$. Тогда имеем математическое ожидание:

$$M[\varphi(x)] = \int\limits_a^b \varphi(x)f(x)dx = \frac{1}{b-a}\int\limits_a^b \varphi(x)dx. \tag{7}$$

Из уравнения (4) следует:

$$\int\limits_a^b \varphi(x)dx = (b-a)M[\varphi(x)]. \tag{8}$$

Если теперь заменить математическое ожидание $M[\varphi(x)]$ его оценкой — выборочным средним, то получим оценку интеграла (3):

$$I = (b-a)\frac{\sum\limits_{i=1}^{N}\varphi(x_i)}{N}, \tag{9}$$

где x_i — значения случайной величины X, а N — количество испытаний. Так как $X \sim R(a, b)$, то очевидно, что $x_i = a + (b-a)\eta$, где η — случайное число из интервала $(0, 1)$.

5 Описание программы

В ходе выполнения данной работы была разработанна программа на языке `MATLAB`, вычисляющая кратные интегралы произвольной размерности.

Функции

- `ndIntegral(f,a,b,G,N,t)` — вычисляет значение интеграла от функции f по области G с ограниченями области интегрирования a и b за N испытаний при коэффициенте доверия t;
- `randInRange(a,b)` — возвращает случайный вектор в интервале (a, b);
- `checkPoint(x,G)` — проверяет принадлежность точки x области G;
- `test1d(N,error,t)` — тест 1-мерного интеграла с начальным количеством испытаний `N` и погрешностью `error` при коэффициенте доверия `t`;
- `test2d(N,error,t)` — тест 2-мерного интеграла с начальным количеством испытаний `N` и погрешностью `error` при коэффициенте доверия `t`;
- `test3d(N,error,t)` — тест 3-мерного интеграла с начальным количеством испытаний `N` и погрешностью `error` при коэффициенте доверия `t`;
- `test6d(N,error,t)` — тест 6-мерного интеграла с начальным количеством испытаний `N` и погрешностью `error` при коэффициенте доверия `t`;

6 Тестирование разработанного ПО

6.1 1-мерный интеграл

Вычислим интеграл, не имеющий первообразной в классе элементарных функций:

$$\frac{1}{\sqrt{2\pi}} \int\limits_0^3 e^{-\frac{t^2}{2}}\, dt. \qquad (10)$$

Его значение известно и соответствует $\Phi(3)$ в таблице значений функции Лапласа, то есть 0.49865.

Вычислим значение с помощью разработанного ПО.

```
>> MonteCarlo.test1d(50,0.05,3);
Error of integration:
    0.1792

First approximation of integral value:
    0.5087

Minimal necessary N:
    643

Error of integration:
    0.0486

Integral value:
    0.4992
```

Разность с точным решением составила 0.0005.

6.2 2-мерный интеграл

Вычислим интеграл

$$I = \iint\limits_{G} (x + y)\, dx\, dy, \tag{11}$$

$$G : 0 \le x \le 2,\ x^2 \le y \le 2x.$$

Для него известно аналитическое решение: $I = 3.4(6)$.

Вычислим значение с помощью разработанного ПО.

```
>> MonteCarlo.test2d(100,0.1,3);
Error of integration:
     3.9595

First approximation of integral value:
     3.5285

Minimal necessary N:
     156781

Error of integration:
     0.0922

Integral value:
     3.4582
```

Разность с точным решением составила 0.008.

6.3 3-мерный интеграл

Вычислим интеграл

$$I = \iiint\limits_G 10x\,dx\,dy\,dz, \tag{12}$$

$$G: 0 \le x \le 1,\ 0 \le y \le \sqrt{1-x^2},\ 0 \le z \le \sqrt{\frac{x^2+y^2}{2}}.$$

Для него известно аналитическое решение: $I = 1$.

Вычислим значение с помощью разработанного ПО.

```
>> MonteCarlo.test3d(1000,0.1,3);
Error of integration:
     0.2990

First approximation of integral value:
     0.9655

Minimal necessary N:
     943

Error of integration:
     0.1025

Integral value:
     1.0056
```

Разность с точным решением составила 0.0056.

6.4 6-мерный интеграл

Решим задачу о вычислении силы притяжения Земли и Луны. Искомое значение силы

$$F = \sqrt{F_x^2 + F_y^2 + F_z^2}, \tag{13}$$

где

$$F_x = G \iiint\iiint_{D\times D'} \frac{\rho(x,y,z)\rho'(x',y',z')}{r^2}(x-x')\,dx\,dy\,dz\,dx'\,dy'\,dz',$$

$$F_y = G \iiint\iiint_{D\times D'} \frac{\rho(x,y,z)\rho'(x',y',z')}{r^2}(y-y')\,dx\,dy\,dz\,dx'\,dy'\,dz',$$

$$F_z = G \iiint\iiint_{D\times D'} \frac{\rho(x,y,z)\rho'(x',y',z')}{r^2}(z-z')\,dx\,dy\,dz\,dx'\,dy'\,dz',$$

$$r = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2},$$

Здесь под D и D' подразумеваются области интегрирования, то есть Земля и Луна.

Точное значение можно получить, если принять Землю и Луну за материальные точки ввиду расстояния между ними. Тогда искомая величина выражается формулой

$$F = G\frac{m_1 \cdot m_2}{r^2}. \tag{14}$$

и равна $1.98997 \cdot 10^{20}$ Н.

Вычислим значение с помощью разработанного ПО.

```
>>MonteCarlo.test6d(1000000,0.11e+20,3)
Integral value:
  1.9731e+20

Difference with correct answer:
  1.6823e+18
```

Разность с точным решением составила $1.6823 \cdot 10^{18}$, что очень неплохо, учитывая масштабы задачи.

7 Исходный код программы

```
classdef MonteCarlo
    % Integration by Monte Carlo method

    methods(Static)

        %% randInRange: Returns random value in range [a,b].
        function [x] = randInRange(a,b)
            n = length(a);
            x = zeros(1,n);
            for i = 1:n
                x(i) = a(i) + (b(i) - a(i)) .* rand();
            end
        end

        %% checkPoint: Checks, is point x in G area, or not.
        function [inArea] = checkPoint(x,G)
            n = length(G); % old version: n = length(x) !!!
            inArea = 1;
            for i = 1:n
                inArea = inArea && G{i}(x);
            end
        end

        %% ndIntegral: Computing n-dimensional definite integral
         % at G area which no more than
         % n-dimensional parallelepiped with properties a and b.
        function [I,c] = ndIntegral(f,a,b,G,N,t_beta)
            %t_beta = 3; % beta = 0.997
            fSum = 0; % sum of computed values f(x)
            fSumSquared = 0; % squared sum of f(x)
            n = 0; % amount of points found in G

            % generating N random vectors x
            for i = 1:N
                x = MonteCarlo.randInRange(a,b);
                % check conditions, x must be in [a,b]
                inArea = MonteCarlo.checkPoint(x,G); % bool value
                % adding f(x)
                if (inArea)
                    fSum = fSum + f(x);
                    fSumSquared = fSumSquared + f(x)^2;
                    n = n + 1;
                end
            end
            c = inf;

            % computing n-dimensional volume of figure
            V = prod(b-a);
            % computing integral value
            I = V * fSum / N;

            if (nargin == 6)
```

```
                fAvg = fSum / n;
                fSquaredAvg = fSumSquared / n;
                Omega = n / N;
                % computing standard deviation
                S1 = sqrt(fSquaredAvg - fAvg^2);
                S2 = sqrt(Omega * (1 - Omega));
                % computing error
                error = V*t_beta*(Omega*S1/sqrt(n) + abs(fAvg)*S2/sqrt(N));
                c = V*t_beta*(sqrt(Omega)*S1 + fAvg*S2);
                disp('Error of integration:');
                disp(error);
            end
        end

        %% TESTS

        %% test1d: computing 1-dimensional definite integral
        function [I] = test1d(N0, maxError, t_beta)
            f = @(x) 1/sqrt(2*pi) * exp(-(x^2)/2);

            % conditions of G area
            x1Cond = @(x) (0<=x(1) && x(1)<=3);
            G = {x1Cond};

            % limitations for every element in x
            a(1) = 0; b(1) = 3;

            [I,c] = MonteCarlo.ndIntegral(f,a,b,G,N0,t_beta);
            disp('First approximation of integral value:');
            disp(I);

            % computing min necessary N
            N = ceil((c/maxError)^2);

            if (N > N0)
                disp('Minimal necessary N:');
                disp(N);

                % computing more correct integral value
                [I] = MonteCarlo.ndIntegral(f,a,b,G,N,t_beta);
                disp('Integral value:');
                disp(I);
            end
        end

        %% test2d: computing 2-dimensional definite integral
        function [I] = test2d(N0, maxError, t_beta)
            f = @(x) x(1) + x(2);

            % conditions of G area
            x1Cond = @(x) (0<=x(1) && x(1)<=2);
            x2Cond = @(x) (x(1)^2<=x(2) && x(2)<=2*x(1));
            G = {x1Cond,x2Cond};

            % limitations for every element in x
```

```
        a(1) = 0; b(1) = 2;
        a(2) = 0; b(2) = 4;

        [I,c] = MonteCarlo.ndIntegral(f,a,b,G,N0,t_beta);
        disp('First approximation of integral value:');
        disp(I);

        % computing min necessary N
        N = ceil((c/maxError)^2);

        if (N > N0)
            disp('Minimal necessary N:');
            disp(N);

            % computing more correct integral value
            [I] = MonteCarlo.ndIntegral(f,a,b,G,N,t_beta);
            disp('Integral value:');
            disp(I);
        end
    end

    %% test3d: computing 3-dimensional definite integral
    function [I] = test3d(N0, maxError, t_beta)
        f = @(x) 10*x(1);

        % conditions of G area
        x1Cond = @(x) (0<=x(1) && x(1)<=1);
        x2Cond = @(x) (0<=x(2) && x(2)<=sqrt(1-x(1)^2));
        x3Cond = @(x) (0<=x(3) && x(3)<=((x(1)^2+x(2)^2)/2));
        G = {x1Cond,x2Cond,x3Cond};

        % limitations for every element in x
        a(1) = 0; b(1) = 1;
        a(2) = 0; b(2) = 1;
        a(3) = 0; b(3) = 1;

        [I,c] = MonteCarlo.ndIntegral(f,a,b,G,N0,t_beta);
        disp('First approximation of integral value:');
        disp(I);

        % computing min necessary N
        N = ceil((c/maxError)^2);

        if (N > N0)
            disp('Minimal necessary N:');
            disp(N);

            % computing more correct integral value
            [I] = MonteCarlo.ndIntegral(f,a,b,G,N,t_beta);
            disp('Integral value:');
            disp(I);
        end
    end

    %% test6d: computing 6-dimensional definite integral
```

```
        function [I] = test6d(N0, maxError, t_beta)
            % Solving the problem of the
            % mutual attraction of two material bodies.

            gravityConst = 6.67e-11; % gravitational constant
            m1 = 6e+24; % mass of the Earth
            m2 = 7.35e+22; % mass of the Moon
            r = 384467000; % distance between Earth and Moon
            p1 = 5520; % avg density of Earth
            p2 = 3346; % avg density of Moon
            R1 = 6367000; % Earth radius
            R2 = 1737000; % Moon radius

            dist = @(x) sqrt((x(1)-(r+x(4)))^2 + ...
                             (x(2)-x(5))^2 + ...
                             (x(3)-x(6))^2);

            fx = @(x) (x(1)-(r+x(4))) / ((dist(x))^3);
            fy = @(x) (x(2)-x(5)) / ((dist(x))^3);
            fz = @(x) (x(3)-x(6)) / ((dist(x))^3);

            % conditions of G area
            x1Cond = @(x) (x(1)^2+x(2)^2+x(3)^2<=R1^2);
            x2Cond = @(x) (x(4)^2+x(5)^2+x(6)^2<=R2^2);
            G = {x1Cond,x2Cond};

            % limitations for every element in x
            a(1) = -R1; b(1) = R1;
            a(2) = -R1; b(2) = R1;
            a(3) = -R1; b(3) = R1;
            a(4) = -R2; b(4) = R2;
            a(5) = -R2; b(5) = R2;
            a(6) = -R2; b(6) = R2;

            [FxI,c] = MonteCarlo.ndIntegral(fx,a,b,G,N0,t_beta);
            Fx = gravityConst * p1 * p2 * FxI;

            [FyI,c] = MonteCarlo.ndIntegral(fy,a,b,G,N0,t_beta);
            Fy = gravityConst * p1 * p2 * FyI;

            [FzI,c] = MonteCarlo.ndIntegral(fz,a,b,G,N0,t_beta);
            Fz = gravityConst * p1 * p2 * FzI;

            I = sqrt(Fx^2 + Fy^2 + Fz^2);
            disp('Integral value:');
            disp(I);

            F_correct = gravityConst * m1 * m2 / r^2;

            diff = abs(I - F_correct);
            disp('Difference with correct answer:');
            disp(diff);
        end
    end
end
```

8 Список литературы

- Кетков Ю.Л., Кетков А.Ю., Шульц М.М. MATLAB 7, программирование, численные методы. - СПб.:БХВ-Петербург, 2005г. 752 стр. с илл.
- Г.М. Фихтенгольц Курс дифференциального и интегрального исчисления. Т. III, М.: Наука, 1956.- 656 с. с илл.
- Соболь И.М. Численные методы Монте-Карло, М. ФИЗМАТЛИТ, 1973.-312 с.
- Ермаков С.М. Метод Монте-Карло и смежные вопросы, М.: ФИЗМАТЛИТ, 1975. - 2-е изд.