**Московский авиационный институт**

**(национальный исследовательский университет)**

Факультет: «Компьютерные науки и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Курсовой проект**

по курсу «Численные методы»

Тема: «Решение СЛАУ методом Монте-Карло»

Студент: Мариничев И. А.

Группа: М8О-308Б-19

Преподаватель: Пивоваров Д. Е.

Оценка:

Москва

2022

**1. Описание метода.**

Рассмотрим систему линейных алгебраических уравнений (СЛАУ):

Постановка задачи решения СЛАУ



А – разреженная матрица, n > 100000.

Численные методы решения СЛАУ:

1. Прямые методы - позволяют найти решение за определенное количество шагов:

• Метод Ньютона, LU-разложение, …

2. Итерационные – устанавливают процедуру уточнения определенного начального приближения к решению

• Метод простой итерации, Гаусса-Зейделя, …

Все методы используют детерминированный алгоритм. Большинство плохо поддаются распараллеливанию. Для итерационных методов существует проблема сходимости.

Метод Монте-Карло – стохастический численный метод

Особенности:

• Использует случайные числа

• Позволяет найти решение любого количества корней, не решая СЛАУ целиком

• Обладает естественным параллелизмом

• Время нахождения одного корня практически не зависит от размерности СЛАУ и зависит только от свойств СЛАУ

Ограничения: • Матрица А коэффициентов СЛАУ должна обладать диагональным преобладанием:

Изображение выглядит как текст, часы

Автоматически созданное описание

• Нужен качественный генератор случайных чисел

Схема метода Монте-Карло

Для решения СЛАУ методом Монте-Карло необходимо привести ее к следующему виду:

Изображение выглядит как текст

Автоматически созданное описание

Метод Монте-Карло заключается в моделировании цепи Маркова, что можно представить как случайное блуждание по матрице коэффициентов. Матрица вероятностей переходов строится по матрице коэффициентов:

Изображение выглядит как текст

Автоматически созданное описание

В итоге получим следующую последовательность действий:

1. Получить матрицу коэффициентов. Рассчитать матрицу переходов P. Рассчитать вектор весов W
2. Начало цикла расчета корней СЛАУ
3. Выбор i-го корня для расчета
4. Начало реализации N итераций поиска оценок корня
5. Построить цепь Маркова для оценки E iго корня
6. X[i] += E
7. Конец реализации N итераций поиска оценок корня
8. X[i] /= N
9. Конец цикла расчета корней СЛАУ

**2. Реализация на С++.**

#ifndef MONTE\_CARLO\_METHOD\_HPP

#define MONTE\_CARLO\_METHOD\_HPP

#include <math.h>

#include <algorithm>

#include "matrix.hpp"

template <typename Type>

const Matrix<Type> get\_non\_absorbing\_transition\_probability(const Matrix<Type> &A, Matrix<Type> &P)

{

size\_t row = A.rows();

size\_t col = A.cols();

Matrix<Type> T(row, col);

Type sum = 0.0;

for (size\_t i = 0; i < row; i++)

{

sum = 0.0;

for (size\_t j = 0; j < col; j++)

sum += abs(A[i][j]);

for (size\_t j = 0; j < col; j++)

{

if (sum > 1e-6)

P[i][j] = abs(A[i][j]) / sum;

else

P[i][j] = 0.0;

}

}

for (size\_t i = 0; i < row; i++)

{

T[i][0] = P[i][0];

for (size\_t j = 1; j < col; j++)

T[i][j] = T[i][j - 1] + P[i][j];

T[i][col - 1] = 1.0;

}

return T;

}

template <typename Type>

const Matrix<Type> get\_absorbing\_transition\_probability(const Matrix<Type> &A, double scale, Matrix<Type> &P)

{

size\_t row = A.rows();

size\_t col = A.cols();

Matrix<Type> T(row + 1, col + 1);

Type sum = 0.0;

for (size\_t i = 0; i < row; i++)

{

sum = 0.0;

double sum1 = 0.0;

for (size\_t j = 0; j < col; j++)

sum += abs(A[i][j]);

sum = double((double(col) + scale)) / col \* sum;

for (size\_t j = 0; j < col; j++)

{

if (sum > 1e-6)

P[i][j] = abs(A[i][j]) / sum;

else

P[i][j] = 0.0;

sum1 += P[i][j];

}

P[i][col] = 1 - sum1;

}

for (size\_t i = 0; i < row; i++)

{

T[i][0] = P[i][0];

for (size\_t j = 1; j < col; j++)

T[i][j] = T[i][j - 1] + P[i][j];

T[i][col] = 1.0;

}

return T;

}

template <typename Type>

class MonteCarloMethod

{

private:

size\_t row;

size\_t col;

double err, sum1, sum2, x, err\_w;

size\_t next, step, times;

size\_t hops;

public:

void init()

{

err = 10000.0;

sum1 = 0.0;

sum2 = 0.0;

x = 0.0;

next = 0;

step = 20000;

times = 1;

err\_w = 1e-6;

hops = 0;

}

std::vector<Type> absorbing(const Matrix<Type> &A, const std::vector<Type> &b, double \_err = 0.1)

{

row = A.rows();

col = A.cols();

Matrix<Type> P(row + 1, col + 1);

Matrix<Type> t = get\_absorbing\_transition\_probability(A, 0.2, P);

size\_t size = b.size();

std::vector<Type> res(size);

srand((unsigned)time(NULL));

size\_t total = 0, \_total = 0, \_hops = 0;

for (size\_t i = 0; i < size; i++)

{

init();

total = 0;

std::cout << "\nCalculating x[" << i << "]... " << std::endl;

while (err > \_err)

{

size\_t cc = step;

while (step--)

{

double v = 1.0;

size\_t index = i, next = 0;

while (next != col)

{

double r = double(rand()) / RAND\_MAX;

next = upper\_bound(t[index].begin(), t[index].end(), r) - t[index].begin();

if (next == col)

continue;

if (abs(P[index][next]) > 1e-6)

v = v \* A[index][next] / P[index][next];

else

v = 0;

hops++;

\_hops++;

index = next;

}

v = v \* b[index] / P[index][col];

sum1 += v;

sum2 += v \* v;

}

step = 1;

total = total + cc;

if (total % 200000 == 0)

std::cout << total << " random walks generated" << std::endl;

x = sum1 / total;

double \_\_err = (sum2 - sum1 / total) / total / total;

times++;

err = sqrt(\_\_err) / x;

}

std::cout << "\nTotal random walks: " << total << std::endl;

std::cout << "Average hops: " << hops / total << std::endl;

res[i] = x;

\_total += total;

}

std::cout << "\nAvegage total random walks: " << \_total << std::endl;

std::cout << "Average hops: " << \_hops / \_total << std::endl;

return res;

}

std::vector<Type> non\_absorbing(const Matrix<Type> &A, const std::vector<Type> &b, double \_err = 0.1)

{

row = A.rows();

col = A.cols();

Matrix<Type> P(row, col);

Matrix<Type> t = get\_non\_absorbing\_transition\_probability(A, P);

size\_t size = b.size();

std::vector<Type> res(size);

srand((unsigned)time(NULL));

unsigned long long total = 0, \_total = 0, \_hops = 0;

for (size\_t i = 0; i < size; i++)

{

init();

total = 0;

std::cout << "\nCalculating x[" << i << "]... " << std::endl;

while (err > \_err)

{

unsigned long long cc = step;

while (step--)

{

double v = 0.0, w = 1.0;

unsigned long long index = i, next = 0;

while (abs(w) > err\_w)

{

double r = double(rand()) / RAND\_MAX;

next = upper\_bound(t[index].begin(), t[index].end(), r) - t[index].begin();

if (P[index][next] > 1e-6)

w = w \* A[index][next] / P[index][next];

else

w = 0;

hops++;

\_hops++;

v = v + w \* b[next];

index = next;

}

v = v + b[i];

sum1 += v;

sum2 += v \* v;

}

step = 1;

total = total + cc;

if (total % 200000 == 0)

std::cout << total << " random walks generated" << std::endl;

x = sum1 / total;

double \_\_err = (sum2 - sum1 / total) / total / total;

times++;

err = sqrt(\_\_err) / x;

if (total >= 5000000)

break;

}

std::cout << std::endl;

std::cout << "Total random walks: " << total << std::endl;

std::cout << "Average hops: " << hops / total << std::endl;

\_total += total;

res[i] = x;

}

std::cout << "Avegage total random walks: " << \_total << std::endl;

unsigned long long avg\_total\_hops = \_hops / \_total;

std::cout << "Average hops: " << avg\_total\_hops << std::endl;

return res;

}

};

#endif /\* MONTE\_CARLO\_METHOD\_HPP \*/

**3. Демонстрация работы программы.**

ivan@asus-laptop:~/nm\_cp$ ./solution ./tests/4x4.txt -p 0.005 -a

~Monte Carlo method with absorbing matrix~

Calculating x[0]...

Total random walks: 20000

Average hops: 19

Calculating x[1]...

Total random walks: 20000

Average hops: 19

Calculating x[2]...

Total random walks: 20000

Average hops: 20

Calculating x[3]...

200000 random walks generated

400000 random walks generated

600000 random walks generated

800000 random walks generated

1000000 random walks generated

1200000 random walks generated

Total random walks: 1393903

Average hops: 20

Avegage total random walks: 1453903

Average hops: 20

Total time: 744834 microsec

Success! Answer was written to the file 'answers/answer\_4x4.txt'

ivan@asus-laptop:~/nm\_cp$ cat ./tests/4x4.txt

4 4

14 -4 -2 3

-3 23 -6 -9

-7 -8 21 -5

-2 -2 8 18

38 -195 -27 142

ivan@asus-laptop:~/nm\_cp$ cat ./answers/answer\_4x4.txt

System solution:

x[0] = -1.448031

x[1] = -6.491393

x[2] = -1.941859

x[3] = 8.052057

**4. Тест производительности.**

|  |  |
| --- | --- |
| **Размерность квадратной матрицы** | **Время в секундах** |
| 4 | 1.678067 |
| 10 | 10.712754 |
| 100 | 161.723802 |
| 1000 | 6452.137604 |

**5. Выводы.**

В ходе данной лабораторной работы я изучил метод Монте-Карло в применении к задаче решения систем линейных уравнений. Одним из главных преимуществ данного метода является высокий потенциал распараллеливания, так как каждый корень системы находится независимо от остальных. А главной особенностью является то, что он дает гарантию решения лишь для матриц с диагональным преобладанием.

Стоит отметить, что тот вариант реализации, который был написан мной на С++ носит скорее учебный характер, так как передо мной стояла задача понять алгоритм при реализации, а не написать максимально оптимальный решатель для данной задачи.

**Список источников**

1. Алгоритм Монте-Карло для решения систем линейных алгебраических уравнений методом Зейделя [Электронный ресурс]:

https://cyberleninka.ru/article/n/algoritm-monte-karlo-dlya-resheniya-sistem-lineynyh-algebraicheskih-uravneniy-metodom-zeydelya/viewer

1. Revisit of Monte Carlo Methods on Solving LargeScale Linear Sytems

[Электронный ресурс]: https://math.nist.gov/mcsd/Seminars/2014/2014-11-04-Li-presentation.pdf

1. Метод Монте-Карло и его точность

[Электронный ресурс]: https://habr.com/ru/post/274975/