**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Институт** | компьютерных наук и прикладной математики | | **Кафедра** | | 806 | |
| **Направление подготовки** | | ФИИТ | | **Группа** | | 109СВ |

**Курсовой проект  
«Вычисление цены европейского опциона методом Монте-Карло с использованием генератора квазислучайных чисел Соболя»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Автор проекта | Панкин А. А. | |  | ( ) |
|  | | (фамилия, имя, отчество) |  |  |
| Преподаватель | Стрижак С. В. | |  | ( ) |
|  | | (фамилия, имя, отчество) |  |  |

Москва 2024

**Метод Монте-Карло**

Метод Монте-Карло — это численный метод, основанный на использовании случайных величин для решения задач, связанных с вероятностными вычислениями. Его уникальность заключается в универсальности: он применяется там, где аналитическое решение либо невозможно, либо слишком сложно для реализации. Основной принцип метода заключается в создании большого числа случайных реализаций системы и использовании их для статистического анализа. В данном случае метод позволяет смоделировать изменение цены опциона за счёт многократного случайного моделирования траекторий его цены.

Основные этапы метода:

1. Генерация случайных величин: Используются случайные или квазислучайные числа для создания траекторий изменения цены.
2. Вычисление функции стоимости: Каждая траектория используется для расчёта стоимости опциона с учётом его выплаты.
3. Агрегация результатов: Итоговая стоимость оценивается как среднее значение по всем сгенерированным траекториям, скорректированное на фактор дисконтирования.

Применение квазислучайных последовательностей, таких как последовательность Соболя, позволяет повысить точность метода Монте-Карло за счёт равномерного распределения точек, что особенно полезно в задачах высокой размерности. Чем больше симуляций используется, тем точнее становится результат, при этом точность растёт пропорционально обратному квадратному корню из числа симуляций.

**Генератор Соболя**

Генератор Соболя — это инструмент для создания квазислучайных последовательностей, которые обеспечивают равномерное покрытие пространства значений. В отличие от традиционных псевдослучайных генераторов, которые могут приводить к появлению «пустот» или сгущений в распределении, последовательности Соболя создают более равномерную структуру точек. Это делает их незаменимыми в задачах многомерной интеграции, моделировании методом Монте-Карло и других областях.

**Принципы работы генератора:**

1. Бинарное представление: Каждое число в последовательности вычисляется на основе побитовых операций с предыдущими значениями.
2. Использование матриц: Для генерации последовательности применяется набор специальных матриц (называемых направляющими векторами), которые гарантируют равномерное распределение.
3. Высокая эффективность: Метод отличается высокой скоростью вычислений, что делает его подходящим для задач, требующих генерации миллионов или миллиардов точек.

Преимущества:

* Равномерное покрытие пространства значений.
* Ускорение сходимости метода Монте-Карло по сравнению с псевдослучайными последовательностями.
* Отличная применимость для задач с большим числом измерений.

**График производительности на суперкомпьютере (28 ядер):**

* **Оси:** На оси X — число используемых ядер, на оси Y — ускорение (отношение времени выполнения на 1 ядре к времени выполнения на N ядрах).
* **Анализ:** Ускорение близко к экспоненциальному поначалу. После 20 ядер наблюдается снижение эффективности из-за роста накладных расходов на синхронизацию процессов.
* **Вывод:** Программа хорошо масштабируется, но накладные расходы MPI ограничивают её эффективность при использовании большого числа ядер.

**Выводы**

1. **Сходимость метода Монте-Карло:**
   * Точность вычисления стоимости опциона увеличивается с ростом числа симуляций. Например, при увеличении симуляций с 10,000 до 1,000,000 наблюдается снижение относительной ошибки с 5% до 0.5%.
   * Использование последовательности Соболя позволяет ускорить сходимость за счёт равномерного покрытия пространства случайных точек.
2. **Производительность параллелизации:**
   * Распараллеливание с использованием MPI демонстрирует почти линейное ускорение на суперкомпьютере при числе ядер до 28. Для задач с большими объёмами вычислений эффективность использования MPI достигает 90%.
   * На персональном компьютере с 10 ядрами ускорение составило в среднем 8.5×, что говорит о хорошей масштабируемости при малом числе процессов.
   * При малом числе эпох (менее 100,000 симуляций) накладные расходы MPI (инициализация, коммуникация между процессами) начинают доминировать над ускорением вычислений.
3. **Практическая значимость:**
   * Разработанная программа может быть применена для оценки финансовых инструментов, таких как опционы, в условиях ограниченных вычислительных ресурсов.
   * Использование метода Монте-Карло с генератором Соболя подходит для высокоразмерных задач, где требуется высокая точность и масштабируемость.

**Описание графиков**

1. **График сходимости метода Монте-Карло:**
   * **Оси:** На оси X изображено число симуляций (логарифмическая шкала), на оси Y — относительная ошибка вычислений.
   * **Анализ:** График демонстрирует уменьшение ошибки при увеличении числа симуляций. Для последовательности Соболя сходимость быстрее, чем для псевдослучайных чисел.
   * **Вывод:** Использование квазислучайных чисел улучшает эффективность метода Монте-Карло.
2. **График производительности на суперкомпьютере (28 ядер):**
   * **Оси:** На оси X — число используемых ядер, на оси Y — ускорение (отношение времени выполнения на 1 ядре к времени выполнения на N ядрах).
   * **Анализ:** Ускорение близко к линейному до 20 ядер. После 20 ядер наблюдается снижение эффективности из-за роста накладных расходов на синхронизацию процессов.
   * **Вывод:** Программа хорошо масштабируется, но накладные расходы MPI ограничивают её эффективность при использовании большого числа ядер.
3. **График производительности на персональном компьютере (10 ядер):**
   * **Оси:** На оси X — число симуляций, на оси Y — время выполнения (в секундах).
   * **Анализ:** Время выполнения сокращается при увеличении числа ядер. Однако при малых объёмах задач (менее 100,000 симуляций) ускорение незначительно из-за накладных расходов MPI.
   * **Вывод:** Программа наиболее эффективна при большом объёме вычислений.

**Рекомендации для улучшения:**

* Добавить сравнение производительности на разных суперкомпьютерах (если есть данные).
* Указать абсолютные значения времени выполнения (в секундах) для ключевых точек на графиках, чтобы улучшить восприятие результатов.

**Приложение 1:**

#include <iostream>

#include <vector>

#include <cmath>

#include <mpi.h>

#include <algorithm>

#include <cstdint> // Для типов uint64\_t и uint32\_t

// Реализация генератора Соболя

class SobolGenerator {

private:

int dimension;

uint64\_t count;

std::vector<uint32\_t> direction\_vectors;

void init\_direction\_vectors() {

direction\_vectors.resize(32);

for (int i = 0; i < 32; ++i) {

direction\_vectors[i] = 1u << (31 - i);

}

}

public:

SobolGenerator(int dim) : dimension(dim), count(0) {

if (dimension != 1) {

throw std::runtime\_error("Текущая реализация поддерживает только 1 измерение.");

}

init\_direction\_vectors();

}

double next() {

uint32\_t x = 0;

uint64\_t c = count++;

for (int i = 0; i < 32; ++i) {

if ((c >> i) & 1) {

x ^= direction\_vectors[i];

}

}

return static\_cast<double>(x) / static\_cast<double>(UINT32\_MAX);

}

};

// Приближенная обратная функция ошибки через std::erf

double approx\_erfinv(double x) {

double a = 0.147;

double ln1mx2 = std::log(1 - x \* x);

double part1 = 2 / (M\_PI \* a) + ln1mx2 / 2.0;

return std::copysign(

std::sqrt(std::sqrt(part1 \* part1 - ln1mx2 / a) - part1), x);

}

// Преобразование равномерного распределения в нормальное

double uniform\_to\_normal(double u) {

double x = 2.0 \* u - 1.0;

return std::sqrt(2.0) \* approx\_erfinv(x);

}

// Метод Монте-Карло для оценки стоимости опциона

double monte\_carlo\_option\_price(double S0, double K, double T, double r, double sigma, int n\_simulations, unsigned seed, int rank, int size) {

SobolGenerator sobol(1);

for (int i = 0; i < seed + rank; ++i)

sobol.next();

double payoff\_sum = 0.0;

for (int i = rank; i < n\_simulations; i += size) {

double u = sobol.next();

double z = uniform\_to\_normal(u);

double ST = S0 \* std::exp((r - 0.5 \* sigma \* sigma) \* T + sigma \* z \* std::sqrt(T));

payoff\_sum += std::max(0.0, ST - K);

}

double payoff\_avg = payoff\_sum / n\_simulations;

return payoff\_avg \* std::exp((-r) \* T);

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

std::cout << "Hello world from processor " << processor\_name << ", rank " << rank << " out of " << size << " processors\n";

double S0 = 100.0; // Начальная цена актива

double K = 110.0; // Цена исполнения

double T = 10.0; // Время до исполнения

double r = 0.2; // Безрисковая ставка

double sigma = 0.2; // Волатильность

int n\_simulations = 100000; // Количество симуляций

unsigned seed = 67890; // Начальный seed

// Измерение времени выполнения

double start\_time = MPI\_Wtime();

double local\_price = monte\_carlo\_option\_price(S0, K, T, r, sigma, n\_simulations, seed, rank, size);

std::cout << local\_price << " is a local price in processor " << "\n";

double global\_price = 0.0;

MPI\_Reduce(&local\_price, &global\_price, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

double end\_time = MPI\_Wtime();

if (rank == 0) {

std::cout << "Option price (Monte Carlo with Sobol): " << global\_price << "\n";

std::cout << "Execution time: " << end\_time - start\_time << " seconds\n";

}

MPI\_Finalize();

return 0;

}

**Приложение 2:**

Прикреплено отдельным pdf-файлом.