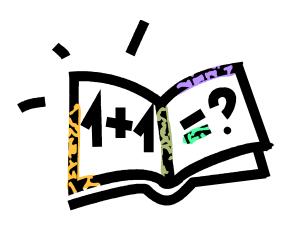


Параллельные численные методы

Лабораторная работа 4: Параллельные методы Монте-Карло

При поддержке компании Intel

Мееров И.Б., Сысоев А.В. Кафедра математического обеспечения ЭВМ





Если люди не полагают, что математика проста, то только потому, что они не понимают, как на самом деле сложна жизнь.

Джон фон Нейман

Не все так просто, как кажется...

Народная мудрость



Содержание

- □ Введение
- □ Цель работы
- □ Тестовая инфраструктура
- □ Оценивание опционов Европейского типа
- □ Интегрирование по методу Монте-Карло
- □ Программная реализация
- □ Задания для самостоятельной работы
- □ Литература

Авторы выражают благодарность Андрею Николаеву за полезные комментарии и обсуждения



1. Приступаем к работе



Введение. О методе Монте-Карло и задаче интегрирования

- Метод Монте-Карло (метод статистических испытаний) применяется в т.ч. для многомерного интегрирования.
- Описание метода Монте-Карло, его сравнение с квадратурными формулами в задаче численного интегрирования приводится в учебном пособии и изучено ранее. Попробуем ответить на следующие вопросы:
- □ В чем идеология метода Монте-Карло? Какова схема вычислений метода? Какие примеры задач, решаемые этим методом, вам известны? На какие теоретические результаты он опирается? Какова сходимость метода? Что требуется для программной реализации метода?
- Многомерное интегрирование: какие методы вам известны? Какова их сходимость? Что требуется для их программной реализации?
- Сравните работу сходимость методов (включая метод Монте-Карло) в задаче многомерного интегрирования.



Введение. Метод Монте-Карло и генераторы СЧ

Некоторые замечания.

- □ Метод Монте-Карло применяется для решения многих задач, как имеющих, таки не имеющих вероятностную природу (задачи финансовой математики, статистической физики, многомерное интегрирование и др.).
- □ Важные достоинства метода простота реализации, как правило, схема вычислений не зависит от размерности задачи.
- □ Сходимость метода 1/ __ требуется много СЧ.
- □ *Случайные числа*: случайные, псевдослучайные (ПСЧ), квазислучайные (КСЧ). На практике используются ПСЧ и КСЧ. Генерируются специально разработанными алгоритмами генераторами СЧ.
- □ Качество генератора критично для сходимости метода.
- Время работы генератора существенно относительно времени работы метода.
- Существуют специальные техники повышения скорости сходимости.



Введение. Метод Монте-Карло в задаче многомерного интегрирования

■ Метод Монте-Карло (метод статистических испытаний) – применяется в т.ч. для многомерного интегрирования.

Некоторые замечания.

- Метод Монте-Карло применяется для многомерного интегрирования, хотя, казалось бы, задача не имеет вероятностную природу.
 - В сравнении с квадратурными формулами:
- □ Важные достоинства метода простота реализации, схема вычислений не зависит от размерности области интегрирования!
- □ *Сходимость метода* 1/ \sqrt{N} требуется много СЧ (здесь N количество СЧ). Сходимость не зависит от размерности области!
- □ Случайные числа: случайные, псевдослучайные (ПСЧ), квазислучайные (КСЧ). На практике используются ПСЧ и КСЧ. В данной задаче используются как ПСЧ, так и КСЧ, однако КСЧ в ряде случаев дают большую скорость сходимости.



Введение. Замечания по задаче

- **В данной работе** будет рассмотрено применение ПСЧ и КСЧ в решении задачи численного интегрирования.
- □ Изложение ведется на примере задачи финансовой математики вычисление цены опционов европейского типа на акцию.
- Указанная задача при рассматриваемых параметрах имеет аналитическое решение, мы его посчитаем, но будем решать численно.
 Идея – понять, как ведет себя погрешность метода. Это легко сделать, зная точное решение.
- Мы будем рассматривать одномерную задачу в основном для упрощения выкладок, хотя и квадратурные формулы в этом случае работают лучше. В многомерном случае в методе Монте-Карло практически ничего не меняется (изменяется определение опциона, неизвестно аналитическое решение, с точки зрения метода необходимо использовать многомерное нормальное распределение вместо одномерного 2-3 «лишних » строки кода с использованием Intel MKL).



Цель работы

Изучение принципов построения корректных параллельных реализаций методов Монте-Карло с использованием математических библиотек.

- Изучение общей схемы метода Монте-Карло для решения задачи вычисления определенного интеграла.
- Освоение способов использования математической библиотеки Intel MKL для генерации случайных чисел в последовательном и параллельном случае.
- □ Выполнение корректной программной реализации метода Монте-Карло для вычисления определенного интеграла в последовательном и параллельном случае.
- Изучение принципов постановки вычислительного эксперимента для оценки корректности и производительности выполненной программной реализации.

Тестовая инфраструктура

| Процессор | 2 четырехъядерных процессора Intel Xeon E5520 (2.27 GHz) |
|---|---|
| Память | 16 Gb |
| Операционная система | Microsoft Windows 7 |
| Среда разработки | Microsoft Visual Studio 2008 |
| Компилятор, профилировщик, отладчик | Intel Parallel Studio SP1 |
| Математическая библиотека | Intel MKL v. 10.2.5.035 |



2. ОЦЕНИВАНИЕ ОПЦИОНОВ ЕВРОПЕЙСКОГО ТИПА



Модель финансового рынка

□ Модель Блэка-Шоулса

- $\Box dB_t = rB_t dt, \quad B_0 > 0 \tag{1}$
- $\Box dS_t = S_t ((r \delta)dt + \sigma dW_t), \quad S_0 > 0$ (2)
- \square S_t цена акции в момент времени t
- \square B_t цена облигации в момент времени t
- \Box σ волатильность (считаем константой)
- $luepsilon \delta$ ставка дивиденда (считаем равной нулю)
- \square $W = (W_t)_{\{t \ge 0\}}$ Винеровский случайный процесс (E=0 c P=1, независимые приращения, $W_t W_s \sim N(0, t\text{-}s)$, где s < t), траектории процесса $W_t(\omega)$ непрерывные функции времени с P=1.
 - S_0, B_0 заданы.

Модель финансового рынка

□ Модель Блэка-Шоулса

$$dB_{t} = rB_{t}dt, \quad B_{0} > 0$$

$$dS_{t} = S_{t}(rdt + \sigma dW_{t}), \quad S_{0} > 0$$

- □ Система стохастических дифференциальных уравнений.
- □ Вообще говоря, S вектор (цен акций). Для упрощения считаем, что S – скаляр (рассматривается одна акция).
- □ В одномерном случае несколько упрощаются выкладки и код, схема вычислений остается той же.



Модель финансового рынка

□ Модель Блэка-Шоулса, уравнение цен акций

$$dS_t = S_t (rdt + \sigma dW_t), \quad S_0 > 0$$

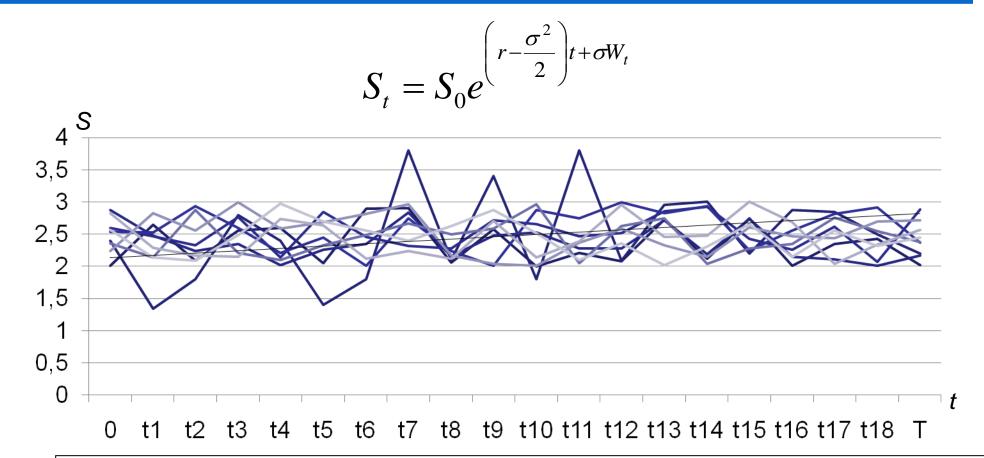
При постоянных параметрах система имеет аналитическое решение:

$$S_{t} = S_{0}e^{\left(r - \frac{\sigma^{2}}{2}\right)t + \sigma W_{t}} \tag{3}$$

□ В противном случае уравнение (система в многомерном случае) решается одним из стандартных методов (например, Рунге-Кутты), W_t получается из N(0, t).



Как работает формула?



- \square Выполняется моделирование поведения цены акции на рынке, значения W_t выход генератора случайных чисел.
- □ Шаг 1 повторяется много раз, далее результаты усредняются.



Опцион Европейского типа на акцию...

- □ Опцион производный финансовый инструмент контракт между сторонами P₁ и P₂, который дает право стороне P₂ в некоторый момент времени t в будущем купить у стороны P₁ или продать стороне P₁ акции по цене K, зафиксированной в контракте.
- \square За это право сторона P_2 выплачивает фиксированную сумму (премию) C стороне P_1 .
- □ *К* называется ценой исполнения опциона (*страйк*, strike price), а *C ценой опциона*.



Опцион Европейского типа на акцию...

- □ Колл-опцион европейского типа на акцию. Основная идея заключения контракта состоит в игре двух лиц P_1 и P_2 .
- □ Вторая сторона выплачивает некоторую сумму *C* и в некоторый момент времени *T* (*срок выплаты*, maturity, зафиксирован в контракте) принимает решение: покупать акции по цене *K* у первой стороны или нет. Решение принимается в зависимости от соотношения цены *S_T* и *K*.
 - Если $S_T < K$, покупать акции не выгодно, первая сторона получила прибыль C, а вторая убыток C.
 - Если $S_T > K$, вторая сторона покупает у первой акции по цене K, в ряде случаев получая прибыль (в зависимости от соотношения между C и $S_T K$).



Опцион Европейского типа на акцию

- □ Справедливая цена такого опционного контракта цена, при которой наблюдается баланс выигрыша/проигрыша каждой из сторон.
- □ Логично определить такую цену как средний выигрыш стороны P₂:

$$C = E\left(e^{-rT}\left(S_T - K\right)^+\right) \tag{4}$$

Математическое ожидание Разность между ценой акции в момент исполнения Т и страйком, если она положительна

Дисконтирование (1р. в момент t = T приводится к 1р. при t = 0)



Вычисление справедливой цены опциона

- Для Европейского колл-опциона при сделанных ранее предположениях известно аналитическое решение.
- □ Аналитическое решение описывается формулой Блэка Шоулса для вычисления цены опциона в момент времени t = 0
 (F функция стандартного нормального распределения):

$$C = S_0 F(d_1) - K e^{-rT} F(d_2)$$

$$d_1 = \ln \frac{S_0}{K} + \frac{\left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$
(5)



3. Интегрирование по методу Монте-Карло



Замечания по вычисления цены опциона

- □ При сделанных предположениях цена опциона может быть вычислена по формулам (5) путем выполнения нескольких несложных операций.
- □ Однако в ряде случаев (например, при переменных процентных ставках) формула (5) работать не будет, что потребует численного моделирования.
- □ В данной работе мы выполним вычисление опциона по определению (4), рассматривая вычисленное по формулам (5) решение как «точное»; именно от него мы будем отсчитывать погрешность численного моделирования.



Переход к задаче интегрирования

$$C = E\left(e^{-rT}\left(S_{T} - K\right)^{+}\right) \qquad \qquad S_{t} = S_{0}e^{\left(r - \frac{\sigma^{2}}{2}\right)t + \sigma W_{t}}$$

$$C = e^{-rT} \int_{-\infty}^{+\infty} f(z)\varphi(z)dz$$

$$f(z) = \left(S_{0}e^{\left(r - \frac{\sigma^{2}}{2}\right)T + \sigma\sqrt{T}z} - K\right)^{+} \qquad (6)$$

 $\varphi(z)$ – плотность стандартного нормального распределения



Метод Монте-Карло для вычисления интеграла...

- \square Рассмотрим случайную точку P с плотностью распределения $\varphi(z)$.
- □ Рассмотрим случайную величину Q = f(z), где f определяется формулами (6).
- □ Тогда математическое ожидание этой случайной величины E(Q) совпадает с первой формулой (6) с точностью до коэффициента дисконтирования.



Метод Монте-Карло для вычисления интеграла

□ Для вычисления математического ожидания случайной величины *Q с учетом коэффициента*, а, следовательно, и искомого интеграла *C*, можно использовать следующую оценку:

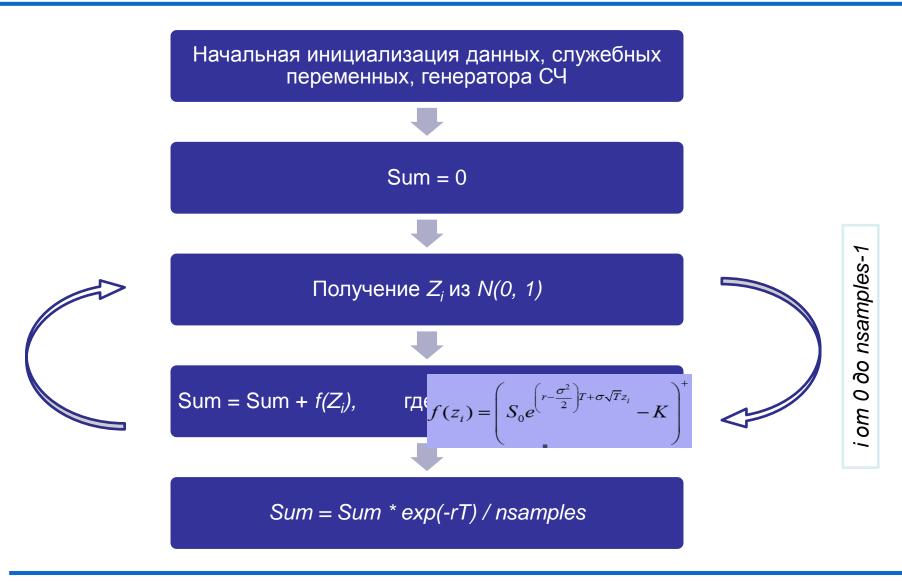
$$\widehat{C} = e^{-rT} \frac{1}{N} \sum_{i=1}^{N} f(z_i),$$

где $f(z_i) = \left(S_0 e^{\left(r - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}z_i} - K\right)^+,$
(7)

 \Box {*z_i*} выбирается из *N*(0, 1).



Алгоритм расчета

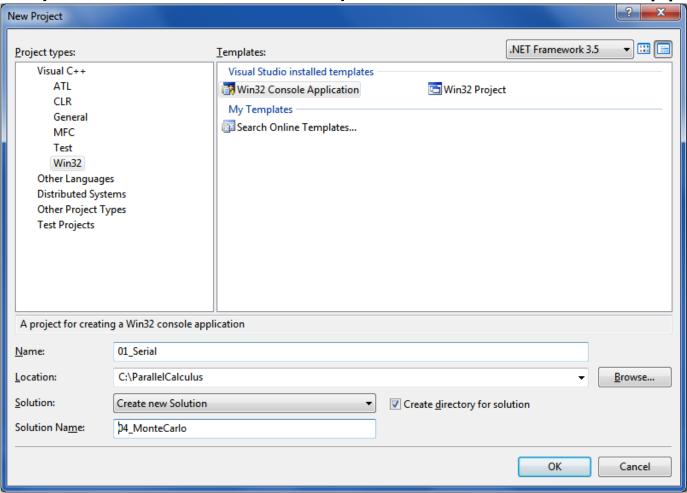




4. Программная реализация



□ Новый проект, консольное приложение, main_c.cpp





□ Новый проект, консольное приложение, main_c.cpp. □ Функция main(), nsamples – количество повторений (понадобится далее) int main(int argc, char *argv[]) { if (argc == 2)unsigned int nsamples = atoi(argv[1]); // Здесь будут вызовы функций для разных способов расчета return 0; else return 1;



- □ Новый проект, консольное приложение, main_c.cpp.
- □ Функция **GetOptionPrice()** для расчета цены опциона по аналитической формуле.
- □ Используем double. Проц. ставки проценты годовых, время в годах.

```
const double sig = 0.2; // Волатильность (20%) const double r = 0.05; // Проц. ставка (5%) const double T = 3.0; // Срок исполнения (в годах) const double s0 = 100.0;// Цена акции при t=0 (у.е.) const double K = 100.0;// Цена исполнения (у.е.)
```



- □ Новый проект, консольное приложение, main_c.cpp.
- □ Функция GetOptionPrice() для расчета цены опциона по аналитической формуле.
- □ vdCdfNorm() функция ст. норм. распределения из Intel MKL double GetOptionPrice() {
 double C, p1[2], p2[2];
 p1[0] = (log(s0 / K) + (r + sig * sig * 0.5) * T) /
 (sig * sqrt(T));
 p1[1] = p1[0] sig * sqrt(T);
 vdCdfNorm(2, p1, p2);
 C = s0 * p2[0] K * exp((-1.0) * r * T) * p2[1];
 return C;



```
□ Печать результата в функции main()
int main(int argc, char *argv[]) {
  if (argc == 2)
    unsigned int nsamples = atoi(argv[1]);
    printf("%d;%.15f;", nsamples, GetOptionPrice());
    return 0;
  else return 1;
  Сборка: подключаем Intel C++ Compiler:

    проект 01_Serial: Intel Parallel Composer→Use Intel C++

  Сборка: подключаем Intel MKL к проекту (последовательный вариант).

    mkl_core.lib mkl_intel_c.lib mkl_sequential.lib в Linker→input
```



```
Aдминистратор: C:\Windows\System32\cmd.exe

Microsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

c:\ParallelCalculus\04_MonteCarlo\Release>01_Serial.exe 10000
10000;20.924360952895213;
c:\ParallelCalculus\04_MonteCarlo\Release>
```



- □ Генератор ПСЧ алгоритм. Используем реализации алгоритмов из библиотеки Intel MKL (корректность + производительность).
- □ Схема получения N(0, 1):
 - Для получения равномерного распределения используется один из базовых генераторов (мы будем использовать МСС59).
 - Далее одним из методов из равномерно распределенных ПСЧ получают нормально распределенные ПСЧ (мы будем использовать метод обратной функции).

Подробная информация: mklman.pdf, vslnotes.pdf



- □ Служебная переменная: VSLStreamStatePtr stream;
- □ Инициализация базового генератора:
 vslNewStream(&stream, VSL_BRNG_MCG59, seed);
- □ Константа VSL_BRNG_MCG59 задает тип базового генератора (алгоритм получения ПСЧ МСG59), а параметр seed начальное значение, которым инициализируется генератор.
- □ Алгоритм получения ПСЧ является детерминированным, последовательность чисел определяется начальным значением seed.
- При достаточно большом числе повторений в методе Монте-Карло seed не должен оказывать большого влияния на результат.
- Чаще всего seed фиксируют для отладки (воспроизводимость результатов) и повышения точности (последовательности ПСЧ, задаваемые разными seed, обладают разными свойствами).



- □ Генератор MCG59: первое число, возвращаемое генератором, равно **seed** / (2^59 1). При любых «обычных» 32-битных **seed** это число близко к нулю.
- □ Для ряда ситуаций это обстоятельство является нежелательным (в том числе для нашей, о чем будет рассказано далее). В таких случаях можно использовать инициализацию генератора 64-битным числом при помощи функции vslNewStreamEx().
- □ Эта функция из двух 32-битных целых чисел делает одно 64-битное и уже им инициализирует генератор.
 - vslNewStreamEx(&stream, VSL_BRNG_MCG59, 2, seed);
- □ Здесь **seed** массив из двух элементов целого типа, например: **const unsigned int seed[2] = {2000000000, 2000000000}**;
- В этом случае 64-битный seed достаточно близок к модулю генератора, что устраняет эффект с близким к нулю первым значением.



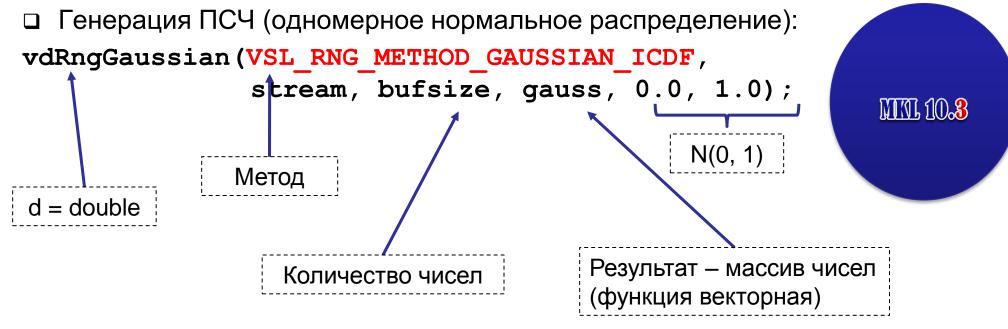
□ Используем метод обратной функции для получения нормального распределения – достаточно хорошая производительность. Важно инициализировать базовый генератор «большим» числом, иначе в последовательности нормальных ПСЧ первое число будет далеким от наиболее вероятного интервала.





Последовательная версия. Использование генератора ПСЧ....

□ Используем метод обратной функции для получения нормального распределения – достаточно хорошая производительность. Важно инициализировать базовый генератор «большим» числом, иначе в последовательности нормальных ПСЧ первое число будет далеким от наиболее вероятного интервала.





Последовательная версия. Использование генератора ПСЧ...

```
double GetMCPrice(unsigned int nsamples) {
 double sum = 0.0, s, tmp1 = (r - sig * sig * 0.5) * T, tmp2 = sig * sgrt(T);
 const unsigned int bufsize = 1000; // Размер буфера для генерации чисел
 const unsigned int seed[2] = {2000000000, 2000000000};
 assert (nsamples % bufsize == 0); // Считаем, что nsamples делится нацело на bufsize
  start = clock();
 double *gauss = new double[bufsize]; // Буфер для случайных чисел
 VSLStreamStatePtr stream;
                                      // Поток для MKL генератора
 vslNewStreamEx (&stream, VSL BRNG MCG59, 2, seed); // Инициализируем MCG59
   for (unsigned int portion = 0; portion < nsamples / bufsize; portion++) {</pre>
     vdRngGaussian (VSL METHOD DGAUSSIAN ICDF, stream, bufsize, gauss, 0.0, 1.0);
     for (int i = 0; i < bufsize; i++) {
     double payoff;
     s = s0 * exp(tmp1 + tmp2 * gauss[i]);
     payoff = s - K;
     if (payoff > 0.0) sum = sum + payoff;
  sum = sum / nsamples * exp(-r * T);
 vslDeleteStream(&stream);
 delete [] gauss;
  finish = clock();
  t = (double) (finish - start) / CLOCKS PER SEC;
  return sum;
```

Последовательная версия. Использование генератора ПСЧ....

- Некоторые замечания по коду:
 - Число повторений nsamples может быть достаточно большим.
 Разобьем общее число повторений на порции по размеру (bufsize) выделенного для хранения случайных чисел буфера (gauss). Отсюда же вытекает и двойной цикл: внутренний по размеру буфера, внешний по числу nsamples/bufsize. Оптимальный с точки зрения производительности размер буфера зависит от архитектуры вычислительной системы и может быть определен экспериментально.
 - Из формулы (7) в двойном цикле вычисляется только сумма.
 Остальные действия (деление на общее число повторений и умножение на экспоненту) вынесены за цикл в целях оптимизации.
 - В код функции вставлены замеры времени с использованием стандартной функции clock() – удовлетворительная точность при существенном времени вычислений.



Последовательная версия. Использование генератора ПСЧ....

□ Некоторые замечания по коду (основная вычислительная часть): for (unsigned int portion = 0; portion < nsamples/ bufsize; portion++) {</pre> vdRngGaussian (VSL METHOD DGAUSSIAN ICDF, stream, bufsize, gauss, 0.0, 1.0); for (int i = 0; i < bufsize; i++) { double payoff; s = s0 * exp(tmp1 + tmp2 * gauss[i]);payoff = s - K; if (payoff > 0.0) sum = sum + payoff; }} sum = sum / nsamples * exp(-r * T);□ Вместо вычисления tmp1 + tmp2 * gauss[i] можно генерировать случайные числа из N(tmp1, tmp2²). Если $Z \sim N(0, 1)$, $X = a + b * Z \sim N(a, b^2)$. □ Более того, можно избежать самостоятельного вычисления экспоненты,



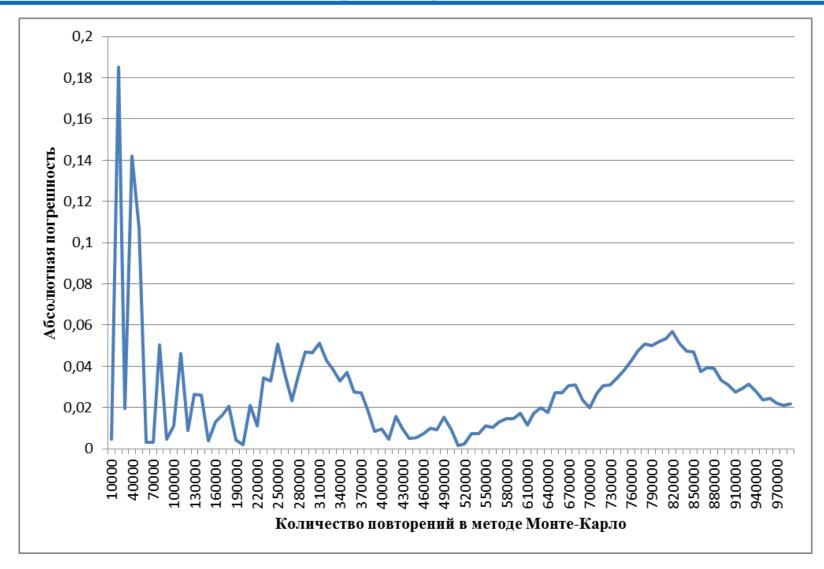
получая числа из логнормального распределения.

Последовательная версия. Использование генератора ПСЧ...

```
□ Модифицируем функцию mian():
int main(int argc, char *argv[])
  if (argc == 2)
    unsigned int nsamples = atoi(argv[1]);
    printf("%d;%.15f;", nsamples, GetOptionPrice());
    printf("%.15f;%f;", GetMCPrice(nsamples), t);
    return 0;
  else
    return 1;
```



Последовательная версия. Использование генератора ПСЧ





Последовательная версия. Использование генератора КСЧ...

- □ Принимая во внимание достоинства метода (например, простоту реализации, отсутствие «проклятия размерности»), необходимо помнить о достаточно медленной сходимости.
- Ускорение сходимости метода является темой многочисленных исследований. В частности, интерес представляют методы понижения дисперсии получаемых результатов (см. учебное пособие, работы [1, 2] и задания для дополнительной проработки).
- □ В лабораторной работе мы рассмотрим применение другого типа алгоритмов для получения случайных чисел. Идея этих алгоритмов [16, 19] состоит в том, чтобы при генерации равномерного распределения обеспечить более равномерное покрытие области значений, частично пожертвовав «случайностью».
- Числа, полученные в результате таких действий, называются квазислучайными (КСЧ). В ряде задач (например, в задаче численного интегрирования) применение последовательностей КСЧ обеспечивает существенно лучшую сходимость.

Последовательная версия. Использование генератора КСЧ...

- □ Используем генератор КСЧ из библиотеки Intel MKL для получения требуемой точности при меньшем числе повторений.
- □ Как и ранее нам потребуется создать и инициализировать служебную переменную для МКL генератора (используем базовый генератор И.М. Соболя).

```
VSLStreamStatePtr stream;
vslNewStream(&stream, VSL_BRNG_SOBOL, 1);
```

- □ Заметим, что в качестве **seed** для генератора необходимо использовать размерность случайной области (в данном случае, 1), а проблемы с появлением «странного» первого числа в выходе генератора здесь не возникает в силу особенностей алгоритма получения КСЧ.
- □ Поправка в код (схема вычислений не меняется):

```
if (method==0) vslNewStreamEx(&stream, VSL_BRNG_MCG59, 2, seed);
if (method==1) vslNewStream(&stream, VSL_BRNG_SOBOL, 1);
```

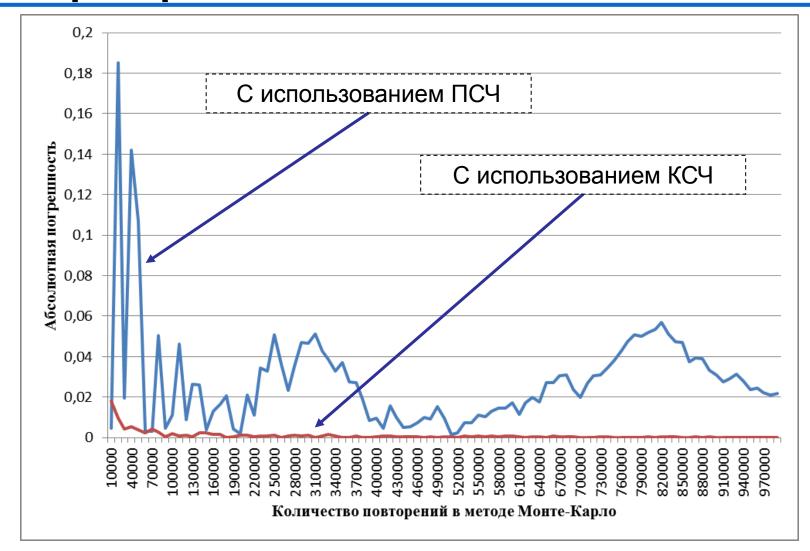


Последовательная версия. Использование генератора КСЧ...

```
int main(int argc, char *argv[])
  if (argc == 2)
   unsigned int nsamples = atoi(argv[1]);
   printf("%d;%.15f;", nsamples, GetOptionPrice());
   printf("%.15f;%f", GetMCPrice(nsamples, 0), t);
   printf("%.15f;%f\n", GetMCPrice(nsamples, 1), t);
    return 0;
 else return 1;
```



Последовательная версия. Использование генератора КСЧ





- □ Новый проект (02_Parallel).
- □ Копируем предыдущий код.
- □ Затем подключим в созданном проекте библиотеку **MKL** и выполним переход к использованию компилятора **Intel C++**.
- □ Наконец, настроим в свойствах проекта использование OpenMP. В дереве Configuration Properties перейдите к разделу C/C++→Language и в поле OpenMP Support справа выберите вариант: Generate Parallel Code (/openmp, equiv. to /Qopenmp).



- □ Основное содержимое функции **GetMCPrice()** цикл, в котором на каждой итерации заполняется случайными числами буфер на 1000 элементов и над ними выполняются некоторые действия.
- □ Очевидно, распараллеливать имеет смысл именно этот цикл.
 Зависимостей между итерациями нет, то есть все, что необходимо, разделить выполнение цикла между потоками.
- □ Применим OpenMP директиву **omp parallel for**, позволяющую и создать потоки, и распределить итерации цикла между ними. Заметим также, что в переменной **sum** накапливается общая сумма результат работы цикла. Воспользуемся редукцией.

```
#pragma omp parallel for reduction(+:sum)
for (unsigned int portion = 0;
  portion < nsamples / bufsize; portion++)</pre>
```



```
Містоsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

с:\ParallelCalculus\04_MonteCarlo\Release>02_Parallel.exe 10000
10000;20.924360952895213;68.716257143969699;0.001000;68.119633066946292;0.001000

c:\ParallelCalculus\04_MonteCarlo\Release>02_Parallel.exe 10000
10000;20.924360952895213;65.662456821814644;0.001000;99.449779185717844;0.001000

c:\ParallelCalculus\04_MonteCarlo\Release>02_Parallel.exe 10000
10000;20.924360952895213;68.996517280713320;0.001000;85.169866750954981;0.0000000
```

- □ Соберите проект **02_Parallel** и запустите на исполнение с числом повторений равным, например, 10 000 (напоминаем, что во всех экспериментах необходимо использовать конфигурацию **Release**).
- □ Убедитесь, что на многоядерной/многопроцессорной системе результат работы функции GetMCPrice() существенно отличается от последовательного и, кроме того, меняется от запуска к запуску.
- ⊐ Гонки данных!



- □ Во-первых, здесь нам на помощь должен прийти инструмент. Используем Intel Parallel Inspector для анализа программы на наличие «параллельных» ошибок.
- □ Для этого вернемся к конфигурации **Debug**, в панели инструментов **Intel Parallel Inspector** выберем тип ошибок **Threading Error** и нажмем кнопку **Inspect** (не забыв передать число повторений в командной строке в настройках проекта).
- □ По завершении работы нашей программы появятся результаты анализа, содержащие информацию о наличии двух ошибок типа «гонка данных» в функции **GetMCPrice()**





Threading Errors (level ti3)

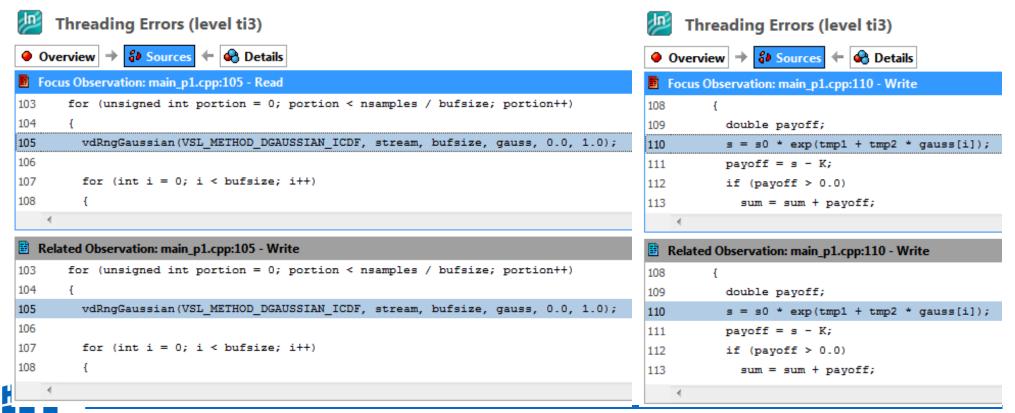
| • 0 | ven | view 👉 🔞 Sources 🖛 😘 | V > Sources Details | | | | |
|------|--------------|----------------------|----------------------|---------|--------------|--|--|
| Prob | Problem Sets | | | | | | |
| ID 📥 | © | Problem | Sources | Modules | State | | |
| | | | | | •••••••••••• | | |

| LT. | | Data face | main-bricbb | UZ_Parallel.exe | re inochixed |
|-----|----------|-----------|-------------|-----------------|--------------|
| P2 | @ | Data race | main_p1.cpp | 02_Parallel.exe | Not fixed |

| Data race: Observations in Problem Set | | | | | | | | | | |
|--|-----------------|-----------------|---------------------|-----------------|-------------|--|--|--|--|--|
| ID | Description 🔺 | Source | Function | Module | State | | | | | |
| ± X14 | Allocation site | main_p1.cpp:93 | ${\sf GetPMCPrice}$ | 02_Parallel.exe | Information | | | | | |
| ± X16 | Allocation site | main_p1.cpp:98 | ${\sf GetPMCPrice}$ | 02_Parallel.exe | Information | | | | | |
| ± X25 | Allocation site | main_p1.cpp:100 | ${\sf GetPMCPrice}$ | 02_Parallel.exe | Information | | | | | |
| ⊕ X13 | Read | main_p1.cpp:105 | GetPMCPrice | 02_Parallel.exe | Not fixed № | | | | | |
| ± X18 | Read | main_p1.cpp:110 | ${\sf GetPMCPrice}$ | 02_Parallel.exe | Not fixed № | | | | | |
| ⊕ X12 | Write | main_p1.cpp:105 | ${\sf GetPMCPrice}$ | 02_Parallel.exe | Not fixed № | | | | | |



□ Двойной щелчок по любой из позиций приведет к строкам исходного кода, где может иметь место ошибка. В данном случае это будут вызов функци vdRngGaussian(), а также оператор, изменяющий значение переменной s во внутреннем цикле.



- Проведите анализ кода на предмет наличия ошибок при распараллеливании.
- Определите источники проблем.
- □ Наметьте пути решения проблем.



- □ Какие переменные изменяются внутри цикла?
 - Счетчик цикла portion, но он локализован в потоках, поскольку объявлен непосредственно в операторе цикла, и не может являться причиной ошибки.
 - Счетчик внутреннего цикла і, с которым ситуация обстоит аналогично.
 - Переменная sum ее локализация выполняется параметром reduction.
 - Переменная s вот это уже первая ошибка, ее нужно либо локализовать, указав в параметре private, либо избавиться от нее, сразу подсчитывая значение во внутренней переменной payoff.
- □ Исправьте ошибку с переменной s, проведите эксперимент и убедитесь, что гонка данных не исчезла (собственно мы уже знаем, что еще одна проблема связана с массивом gauss).



- □ Итак, не рассмотренным остался вызов функции vdRngGaussian().
- В этом вызове проблема в том, что функция заполняет объявленный вне цикла динамически выделяемый массив gauss. Массив является в текущем варианте кода общим для всех потоков и при записи/чтении его элементов возникают конфликты.
- □ Необходимо локализовать его, перенеся в каждый поток (вместе с выделением памяти, конечно).
- □ Это изменение уже нельзя сделать в текущем варианте, поскольку после директивы **omp parallel for** может располагаться только цикл и ничего больше. Следовательно, придется разделить создание потоков и распределение работы в цикле.



- □ Новый проект (03_Parallel).
- □ Intel C++ Compiler, MKL, OpneMP.
- □ Будем создавать параллельную секция сразу после замера времени, включив в нее и объявление массива gauss, и поток для генератора stream.
- □ Внешний цикл **for** предварим OpenMP-директивой **omp for**, освобождение памяти, выделенной под переменные **gauss** и **stream**, внесем в параллельную секцию, а итоговую операция над переменной **sum**, напротив, перенесем ниже, после завершения параллельной работы потоков.
- □ Соберите проект **03_Parallel** и запустите на исполнение с числом повторений, равным, как и для предыдущего варианта, 10 000. Убедитесь, что от запуска к запуску результаты не меняются.



```
#pragma omp parallel private(s) {
 double *gauss = new double[bufsize];
 VSLStreamStatePtr stream:
  if (method == 0) vslNewStreamEx(&stream, VSL BRNG MCG59, 2, seed);
  if (method == 1) vslNewStream(&stream, VSL BRNG SOBOL, 1);
#pragma omp for reduction(+:sum)
  for (unsigned int portion = 0; portion < nsamples / bufsize; portion++) {
   vdRngGaussian (VSL METHOD DGAUSSIAN ICDF, stream, bufsize, gauss, 0.0, 1.0);
    for (int i = 0; i < bufsize; i++) {
     double payoff; s = s0 * exp(tmp1 + tmp2 * gauss[i]);
     payoff = s - K; if (payoff > 0.0) sum = sum + payoff;
 vslDeleteStream(&stream);
 delete [] gauss;
sum = sum / nsamples * exp(-r * T);
```

```
Містоsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

с:\ParallelCalculus\04_MonteCarlo\Release\01_Serial.exe 10000
10000;20.924360952895213;20.929066944139056;0.000000;20.906396070566934;0.000000

c:\ParallelCalculus\04_MonteCarlo\Release\03_Parallel.exe 10000
10000;20.924360952895213;21.054072306091868;0.006000;20.912345732403786;0.000000
```

- Результаты последовательной и параллельной версии не совпали. Всегда ли это говорит об ошибке?
- Результаты последовательной и параллельной версии совпали. Всегда ли это говорит об отсутствии ошибок?
- Инструмент показывает наличие ошибок. Всегда ли это правда?
- □ Инструмент показывает отсутствие ошибок. Всегда ли это правда?
- □ Алгоритмически порядок вычислений в посл. и паралл. версии одинаков. Всегда ли это будет так в машинной реализации?



```
Містоsoft Windows [Version 6.1.7600]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

c:\ParallelCalculus\04_MonteCarlo\Release\01_Serial.exe 10000
10000;20.924360952895213;20.929066944139056;0.000000;20.906396070566934;0.000000

c:\ParallelCalculus\04_MonteCarlo\Release\03_Parallel.exe 10000
10000;20.924360952895213;21.054072306091868;0.006000;20.912345732403786;0.000000
```

- □ Знаете ли вы, что, вообще говоря, (a+b)+c <> a + (b+c) в машинной арифметике?
- □ Итак, расхождение результатов достаточно естественно. Как определить, является ли оно допустимым, есть ли ошибка? В каждом случае приходится подходить индивидуально (пример: вычисление стандартной ошибки и т.д.). Ситуация в реальности еще сложнее точное решение на практике неизвестно! (Иначе зачем его считать).



- □ Внимательно изучите код. Анализ переменных внутри параллельной секции показывает, что гонок данных быть не должно.
- □ Проверим еще одно «узкое место» использование генератора СЧ в параллельных вычислениях.
- □ Запустим программу **01_Serial.exe** с числом повторений в два раза меньшим, чем при запуске **03_Parallel.exe** на 2 потока.
 - Результаты практически совпадают.
 - Таким образом, чтобы добиться от параллельной программы идентичных с последовательной результатов, фактически, нужно взять в р раз больше случайных чисел. Конечно, ни о каком ускорении в этом случае говорить не приходится.
- □ **Причина**: мы получаем одни и те же числа в каждом потоке неправильное использование генератора в параллельных вычислениях.



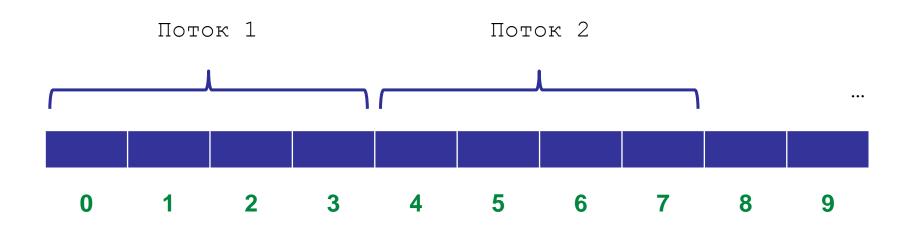
Корректная параллельная версия...

- □ Чтобы исправить положение, требуется, чтобы потоки, как и было задумано изначально, работали с одной последовательностью случайных чисел, «выбирая» из нее нужные для обработки.
- □ Известно несколько подходов
 - Техника skip-ahead
 - Техника leap-frog
 - Специальные генераторы для параллельных вычислений (МТ2203)
 - **—** ...
- □ Будем использовать технику skip-ahead. См. описание других методов в vslnotes.pdf и доп. заданиях к Л.Р.



Корректная параллельная версия...

□ Техника skip-ahead:



□ Поддерживается не всеми базовыми генераторами, **MCG59** и **Sobol** поддерживают (см. vslnotes.pdf).



Корректная параллельная версия...

```
#pragma omp parallel private(s) {
    int count = omp get num threads();
                                                  продолжение
    int num = omp get thread num();
    double *gauss = new double[bufsize];
    VSLStreamStatePtr stream;
    // Инициализируем базовый генератор MCG59 или SOBOL
    if (method == 0)
      vslNewStreamEx(&stream, VSL BRNG MCG59, 2, seed);
    if (method == 1)
      vslNewStream(&stream, VSL BRNG SOBOL, 1);
     // техника "skip-ahead" для корректного использования
    // последовательности случайных чисел
   vslSkipAheadStream(stream, nsamples / count * num);
```



Корректная параллельная версия

```
#pragma omp for reduction(+:sum)
    for (unsigned int portion = 0;
         portion < nsamples / bufsize; portion++) {</pre>
      vdRngGaussian (VSL METHOD DGAUSSIAN ICDF, stream,
                    bufsize, gauss, 0.0, 1.0);
       for (int i = 0; i < bufsize; i++) {
        double payoff;
        s = s0 * exp(tmp1 + tmp2 * gauss[i]);
        payoff = s - K;
        if (payoff > 0.0) sum = sum + payoff;
      } }
    vslDeleteStream(&stream); delete [] gauss;
                                                       начало
  sum = sum / nsamples * exp(-r * T);
```

Результаты

- Соберите результаты последовательной и параллельной версии.
- Определите ускорение, постройте график масштабируемости параллельной версии.
- □ Замечание:
 - В качестве отправной точки для расчетов необходимо принять время работы параллельной версии, запущенной в один поток.
 - Это время иногда оказывается больше времени работы последовательной версии
 - Компилятору труднее оптимизировать код.
 - Эвристики не рассчитаны на 1 поток.



- □ Дополнительные задания имеют разный уровень сложности. Некоторые из них являются достаточно трудоемкими и подходят в качестве тем зачетных тем для студентов, изучающих параллельные численные методы.
- □ Все задания предполагают выполнение параллельных реализаций для систем с общей памятью. Сравнение производительности и точности выполняется для параллельных версий.



- □ Реализовать вычисление определенных интегралов при помощи классических квадратурных формул. Провести сравнение погрешности вычислений с методом Монте-Карло. Сравнить время работы обоих подходов при достижении одинаковой точности.
 - Использовать метод прямоугольников.
 - Использовать метод трапеций.
 - Использовать метод Симпсона.
- □ Выполнить задание для многомерного случая. Изучить, как влияет увеличение размерности (кратности интеграла) на сходимость методов.



- □ Сравнить влияние размерности на соотношение погрешностей в методах Монте-Карло, использующих генераторы псевдо-случайных и квази-случайных чисел.
- Определить оптимальный размер буфера для генерации псевдо-случайных чисел для имеющейся в распоряжении тестовой инфраструктуры.
- □ Определить оптимальный размер буфера для генерации квази-случайных чисел для имеющейся в распоряжении тестовой инфраструктуры.
- □ Реализовать технику понижения дисперсии для метода Монте-Карло – метод антитетических переменных (antithetic variates). Провести эксперименты, подтверждающие эффективность реализации.

- □ Реализовать технику понижения дисперсии для метода Монте-Карло – метод выборки по значимости (importance sampling). Провести эксперименты, подтверждающие эффективность реализации.
- □ Реализовать технику понижения дисперсии для метода Монте-Карло – метод контрольных переменных (control variate). Провести эксперименты, подтверждающие эффективность реализации.
- □ Реализовать генератор равномерного распределения.
 Провести эксперименты оценить корректность и производительность генератора.



- □ Реализовать генератор нормального распределения.
 Провести эксперименты оценить корректность и производительность генератора.
- □ Заменить использование базового генератора MCG59 генератором MT2203. Обеспечить корректность при генерации и использовании случайных чисел.



Использованные источники информации

- 1. Кнут Д. Искусство программирования, том 2. Получисленные методы. 3-е изд. М.: Вильямс, 2007. С. 832.
- 2. Соболь И.М. Численные методы Монте-Карло. М.:Наука, 1973. 312с.
- Соболь И.М. Метод Монте-Карло. М.:Наука, 1968. 64с.
- 4. Ширяев А.Н. Основы стохастической финансовой математики. Фазис, 2004. 1076с.
- 5. Glasserman P. Monte Carlo methods in financial engineering. Springer-Verlag, 2003. 602p.



Дополнительная литература

- 6. Горбунова А.С., Козинов Е.А., Мееров И.Б., Шишков А.В. Применение Монте-Карло интегрирования для нахождения цены многомерного европейского опциона с нулевой корреляцией между акциями // Технологии Microsoft в теории и практике программирования. Материалы конференции / Под ред. проф. Р.Г. Стронгина. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2006. С. 70-72.
- 7. Ермаков С.М. Метод Монте-Карло и смежные вопросы.–М.: Наука, 1975. 472с.
- 8. Ермаков С.М., Михайлов Г.А. Статистическое моделирование. М: Наука, 1982. 296с.
- 9. Соболь И.М. О распределении точек в кубе и сетках интегрирования // УМН, 21:5(131), 1966. С. 271–272.
- 10. Boyle P., Broadie M., Glasserman P. Monte Carlo Methods for Security Pricing // Journal of Economic Dynamics and Control, Volume 21, Issues 8-9. Pp. 1267-1321.
- 11. Metropolis N. Ulam. S. The Monte Carlo Method // Journal of the American Statistical Association, № 247, 1949. pp. 335–341.
- 12. Niederreiter H. Random Number Generation and Quasi-Monte Carlo Methods. SIAM, 1992. 247 p.



Ресурсы сети Интернет

- 13. Weinzierl S. Introduction to Monte Carlo methods. [http://arxiv.org/PS_cache/hep-ph/pdf/0006/0006269v1.pdf].
- 14. Intel Math Kernel Library Reference Manual.
 [http://software.intel.com/sites/products/documentation/hpc/mkl/mklman.pdf].
- 15. Intel Vector Statistical Library Notes.

 [http://software.intel.com/sites/products/documentation/hpc/mkl/vsl/vslnotes.pdf]



Авторский коллектив

- Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. кафедры Математического обеспечения ЭВМ факультета ВМК ННГУ meerov@vmk.unn.ru
- □ Сысоев Александр Владимирович, ассистент кафедры Математического обеспечения ЭВМ факультета ВМК ННГУ sysoyev@vmk.unn.ru

