Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ПРИКЛАДНАЯ МАТЕМАТИКА

**Отчет по производственной практике**

Язык программирования Julia. Апробирование скорости и методов распараллеливания.

Преподаватель Царев С. П.

Студент гр. КИ 13-19Б Тетерятников А.С

Красноярск 2016

**СОДЕРЖАНИЕ**

Введение …………………….….………….……..……………………………...3

Постановка задачи………………….……….………..………...……………......5

1. Основы работы с Julia ……………….……….………..……….....………….6

1.1 Динамическая типизация и проверка принадлежности переменной к конкретному типу данных .………………………………………………6

1.2 Конвертирование типов данных …………………………………….7

1.3 Строки …………………………………………………………………7

1.4 Представление числа с плавающей запятой, работа с числами произвольной точности в Julia ……..…………………………………….8

1.5 Массивы ………...…………………………………………………….10

1.6 Циклы …………………………………………………………………11

1.7 Работа с файлами …………………………………………………….11

1.8 Распараллеливание …………………………………………………..12

2. Метод сопряженных градиентов ……………………………………………13

3. Реализация метода сопряженных градиентов ……………………………...14

3.1 Реализация метода сопряженных градиентов на Julia …………………....14

3.2 Тестирование программы godunovMatrix.jl………………………………..18

Список использованных источников…………………………………………...23

**Введение**

Julia — высокоуровневый высокопроизводительный свободный язык программирования с динамической типизацией, созданный для математических вычислений, появившийся в открытом доступе в феврале 2012 года. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков (например, MATLAB), однако имеет некоторые существенные отличия. Julia написана на Си, C++ и Scheme. В стандартный комплект входит JIT-компилятор на основе LLVM, благодаря чему, по утверждению авторов языка, приложения, полностью написанные на языке, практически не уступают в производительности приложениям, написанным на статически компилируемых языках вроде Си или C++. Большая часть стандартной библиотеки языка написана на нём же. Также язык имеет встроенную поддержку большого числа команд для распределенных вычислений.

В качестве алгоритма для изучения и тестирования Julia был выбран «метод сопряженных градиентов».

Метод сопряженных градиентов предназначен для решения систем линейных алгебраических уравнений вида:

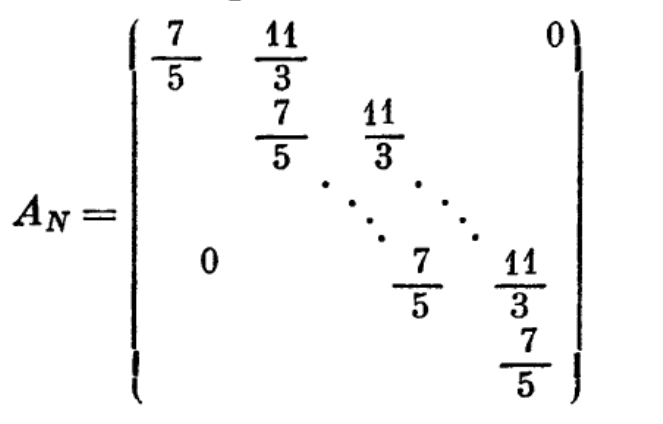
Для получения результата в методе сопряженных градиентов в Julia потребуется использовать числа высокой точности.

Для тестирования алгоритма была выбрана линейная система уравнений Годунова.

**Матрица Годунова**

Система линейных уравнений Годунова представляет собой систему N уравнений вида:

Где A - двухдиагональная матрица:



Вектор



Система уравнений имеет решение: , , … , , … , , .

В дальнейшем такую систему будем называть матрицей Годунова.

Источник: Годунов С.К. Решение систем линейных уравнений - Новосибирск: Наука, 1980.

**Постановка задачи**

* Изучить язык программирования Julia
* Реализовать алгоритм «Метод сопряженных градиентов» на языке программирования Julia.
* Апробировать скорость работы и методы распараллеливания Julia

**1.Основы работы с Julia**

**1.1 Динамическая типизация и проверка принадлежности переменной к конкретному типу данных**

В Julia используется динамическая типизация, то есть переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

В документации есть примечание, что в целях улучшения быстродействия программы необходимо контролировать, чтобы тип уже заданной переменной не изменялся.

Для того чтобы узнать, какой тип имеет существующая переменная x, используется функция возвращающая тип данных:

typeof(x)

Для того чтобы проверить принадлежность переменной к какому либо типу данных можно использовать команду:

x::Type

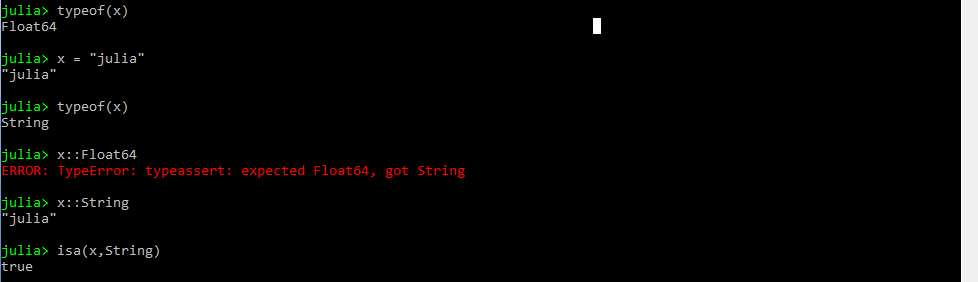
где x - переменная, Type – тип данных. Эту команду удобно использовать при отладке программы, так как в случае не принадлежности к типу, Julia выведет в консоли ошибку и остановит выполнение программы. Команда ничего не возвращает, если x принадлежит типу Type.

Ещё один метод проверки принадлежности к типу осуществляет функция:

isa(x,Type)

где x – переменная, Type – тип данных. Функция возвращает true в случае принадлежности, и возвращает false - в противном случае.

Пример 1.1:Рис. 1.1 – Пример динамической типизации в Julia, и методы проверки на принадлежность переменной к типам данных.



**1.2 Конвертирование типов данных**

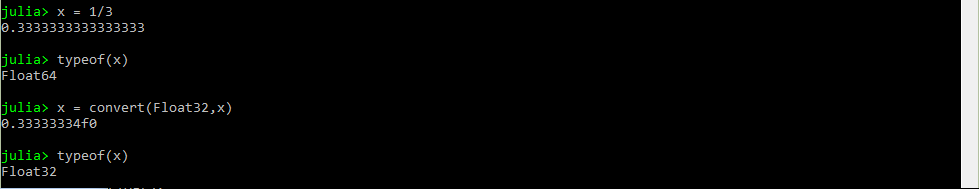
Конвертирование численных типов данных в другой численный тип производится функцией:

convert(Type,x)

где x – переменная, Type – тип данных. Функция возвращает переменную имеющую тип Type.

Пример 1.2:

Рис 1.2 – Пример использования функции convert(Type,x).



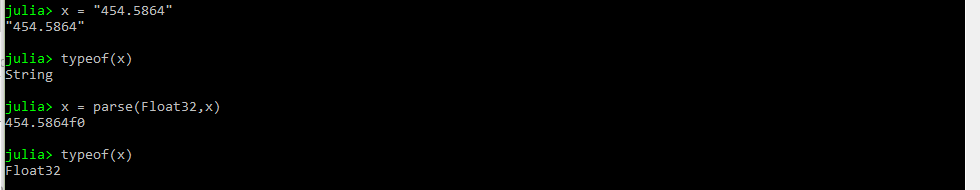
Конвертирование строки, содержащей только число и имеющей тип String, в численный тип производится функцией:

parse(Type,x)

где x – переменная, Type – численный тип данных. Функция возвращает переменную имеющую тип Type.

Пример 1.3:

Рис. 1.3 - Пример использования функции parse(Type,x).



**1.3 Строки**

В Julia строка задается двойными кавычками и имеет тип String:

str = “Hello, world.\n”

Чтобы получить символ строки используются квадратные скобки:

str[2]

в этом случае на выходе будет символ “e”. Нумерация символов строки начинается с единицы.

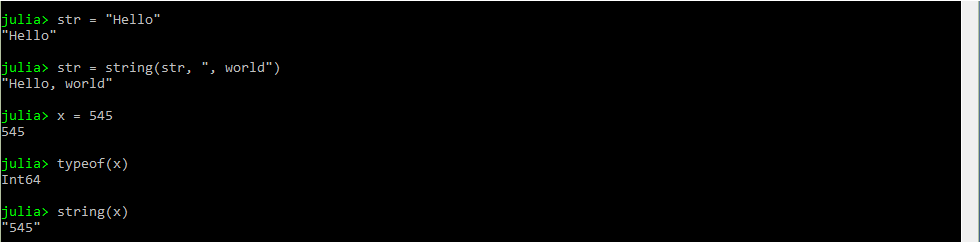
Чтобы получить подстроку строки str с 1 по 5 символ:

str[1:5]

на выходе получится строка “Hello”.

Функция string() используется для преобразования значения переменных в строку или для конкатенации строк.

Пример 1.4Рис. 1.4 – Пример использования функции string. Конкатенация и преобразования числа в строку.



search(str, substr) – поиск в строке str подстроку substr. Возвращает позиции начала и конца подстроки substr в строке str разделенные двоеточием.

split(str, substr, limit=0) – разделение строки str, на массив из строк, разделителем служит подстрока substr. limit означает количество таких разделений. По умолчанию limit=0 – неограниченно.

**1.4 Представление числа с плавающей запятой, работа с числами произвольной точности в Julia.**

Действительные числа обычно представляются в виде чисел с плавающей запятой. При этом лишь некоторые из действительных чисел могут быть представлены в памяти компьютера точным значением, в то время как остальные числа представляются приближёнными значениями. В наиболее распространённом формате число с плавающей запятой представляется в виде последовательности битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа.

Число с плавающей запятой состоит из следующих частей:

* знак мантиссы (указывает на отрицательность или положительность числа),
* мантисса (выражает значение числа без учёта порядка),
* знак порядка,
* порядок (выражает степень основания числа, на которое умножается мантисса).

Пример 1.5: Представление типа Float32



Рис. 1.5 – Побитовое представление числа типа Float32.

Источник: <https://ru.wikipedia.org/wiki/Число_одинарной_точности> .

Помимо типов данных унаследованных из C++, в Julia изначально имеются типы данных для работы с произвольной точностью, мантисса которых задается пользователем:

* BigFloat
* BigInt

Работа с произвольной точностью реализована в julia с помощью библиотек GNU Multiple Precision Arithmetic Library (GMP) и GNU MPFR Library.

Количество бит в мантиссе устанавливается при помощи команды:

setprecision(value)

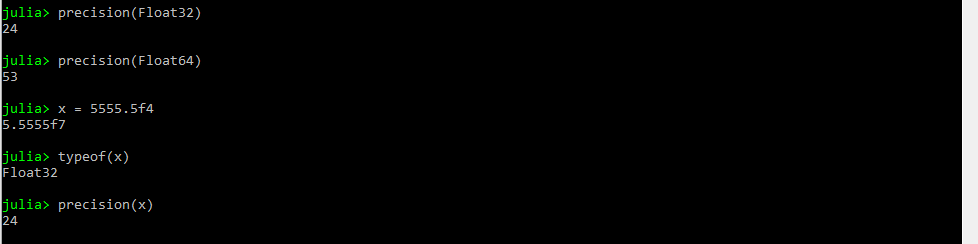
где value – целое число.

Чтобы узнать количество бит в мантиссе численного типа Type используется команда

precision(Type).

вместо Type может быть указана переменная численного типа.

Пример 1.6:

рис 1.6 – Пример определения мантиссы числа или численного типа.

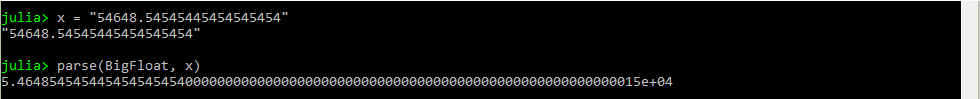
Чтобы преобразовать переменную x имеющую численный тип данных в BigInt или BigFloat используется команда:

big(x)

Для преобразования строки x, содержащей только число, в BigFloat поступают следующим образом:

parse(BigFloat, x)

Пример 1.7:рис. 1.7 – Пример преобразования строки в BigFloat.



**1.5 Массивы**

В Julia массивы можно задавать различными командами:

* Перечислением: arr = [1,2,3,4] – для одномерного массива;

arr = [1 5 6 4; 1 4 6 8] – для двумерного.

* Функцией fill(a, n, m) создает матрицу размерности заполненную значением a;
* Функцией eye(n) – создает квадратную единичную матрицу размерности n;
* Функцией Array(Type, n, m) - создает матрицу размерности заполненную случайными числами типа Type.

Матричные операции:

* A\*B - матричное умножение;
* A.\*B - поэлементное умножение матриц;
* A’ - транспонирование матрицы;
* inv(A) - обратная матрица;
* x.^2 - возведение элементов матрицы в квадрат;
* sum(A) - суммирование всех элементов матрицы

Некоторые полезные функции при работе с массивами:

* length(A) – количество элементов матрицы A;
* ndims(A) – размерность матрицы A;

**1.6 Циклы**

Цикл for:

for i in a:b:c

…

end

цикл перебирает все значения i начиная с a, с шагом b, пока .

Цикл while:

while (условие)

…

end

выполняется пока не выполнится условие.

**1.7 Работа с файлами**

include(path) служит для исполнения кода содержащегося в файле path, где path – путь до исполняемого файла формата \*.jl.

@\_\_FILE\_\_ - возвращает абсолютный путь до исполняемого файла. Не возвращает ничего, если команда выполняется через консоль. Под абсолютным путем понимается путь начинающийся с корневого каталога.

dirname(path) - возвращает путь к директории выше.

isdir(path) – возвращает true в случае если директория path существует, в противном случае – false.

mkdir(path)- создание новой директории с именем path.

rm(path, force=false, recursive=false) – удаление файла или каталога path. force = true – удалении несуществующих файлов не будет рассматриваться как ошибка. recursive = true – рекурсивное удаление.

open(path, mode) - открывает файл, определяемый по path, и подготавливает его к последующему чтению или записи, что определяется посредством mode. Mode может принимать значение:

* “r” – чтение;
* “r+” – чтение, запись;
* “w” – запись, создание, перезапись;
* “w+” – чтение, запись, создание, перезапись;

readlines(file) – считывание строки из file. Возвращает массив строк – всё содержимое файла file.

write(file, str) – запись строки str в файл file.

write(file, A,delim='\t'; opts) – запись матрицы A в файл file в форме текста с разделителем delim=’\t’ по умолчанию.

Текст будет записан в файл только после его закрытия.

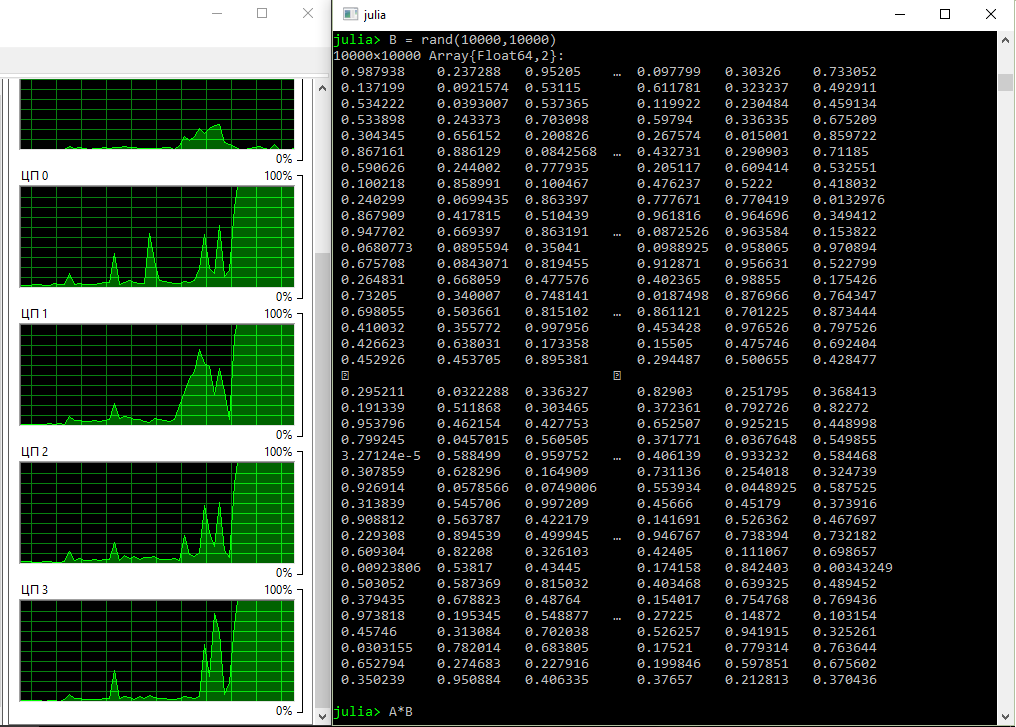
close(file) – закрытие файла.

**1.8 Распараллеливание**

Во время работы с Julia было замечено, что Julia автоматически управляет распараллеливанием на потоки.

Было проведено умножение двух случайных матриц A и B, заданных командой rand(10000,10000), размером 10000\*10000 и во время нагрузки на процессор, отслеживалось использование ядер:

рис 1.8. Отслеживание использования ядер процессора при умножении двух матриц размерности 10000\*10000.



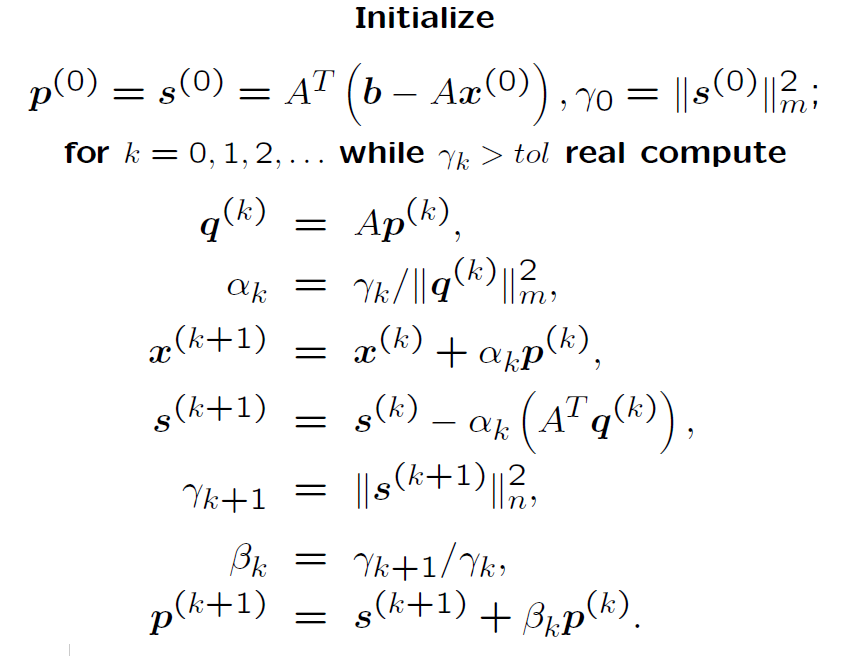
На рисунке 1.8, слева индикаторы нагрузки на ядра указывают, что задействованы все 4 ядра. Таким образом, можно отметить, что при выполнении матричных операций, Julia автоматически распределяет нагрузку на ядра процессора.

**2. Метод сопряженных градиентов**

Метод сопряженных градиентов предназначен для решения систем линейных алгебраических уравнений вида:

На рисунке 2.1 представлен алгоритм метода сопряженных градиентов. Алгоритм был взят из презентации Киреева И.В. «Метод сопряженных градиентов» - Красноярск, 2011.

рис. 2.1 – Алгоритм метода сопряженных градиентов.



**3. Реализация метода сопряженных градиентов**

**3.1 Реализация метода сопряженных градиентов на Julia**

Программа состоит из 3 файлов:

* godunovMatrix.jl
* readFile.jl
* config.txt

godunovMatrix.jl - файл запуска программы. В нем реализованы: таймер; подключение модуля readFile.jl; заполнение массива A и вектора B; алгоритм метода сопряженных градиентов; работа с файлами и папками.

Листинг программы godunovMatrix.jl:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93 | tic() #запуск таймера  ####################################### работа с путями и параметрами  path = @\_\_FILE\_\_ # @\_\_FILE\_\_ - возвращает путь до исполняемого  # файла .jl.  path = dirname(path) # dirname() возвращает путь до дириктории в  # которой находится файл указываемый в path.  include(string(path,"/readFile.jl")) # Выполнение модуля readfile.jl.  setprecision(mantissa) # Установка величины мантиссы BigFloat.  **if** (isdir(string(path,"/result"))) # Проверка на существование  # директории result.  rm(string(path,"/result"),recursive=true) # Удаление каталога result  # в случае его существования.  **end**  ####################################### Функция подсчета нормы  **function norma**(x)  result = big(**0.0**) # Переменной result присваивается  # тип BigFloat.  x = x.^**2** # Все элементы вектора возводятся в квадрат.  result = sum(x) # Все элементы вектора суммируются.  **return** result # функция возвращает переменную result.  **end**  ####################################### заполнение матрицы A и вектора B  n = Godunov\_matrix\_dim  A = fill(big(**0e0**),n,n) # Матрица A размерности n\*n типа BigFloat задается  # функцией fill() и заполняется нулевыми элементами.  B = Array(BigFloat, n) # Вектор B – массив размерности n типа BigFloat.  **for** i **in** **1**:**1**:n  A[i,i] = big"7"/**5** # При заполнении элементов массива, необходимо сначала  # преобразовать целое число в тип BigFloat, и только  # после этого, делить его на другое число.  **end**  **for** i **in** **1**:**1**:(n-**1**)  A[i,i+**1**] = big"11"/**3**  **end**  **for** i **in** **1**:**1**:(n-**2**)  B[i] = (**152**\*i+**118**)  B[i] = B[i]/(**15**\*(**2**\*i+**1**)\*(**2**\*i+**3**))  **end**  B[n] = **7**  B[n] = B[n]/(**5**\*(**2**\*n+**1**))  B[n-**1**] = (**152**\*n-**34**)  B[n-**1**] = B[n-**1**]/(**15**\*(**2**\*n-**1**)\*(**2**\*n+**1**))  ####################################### Начало МСГ  x = fill(big(**0.0**e),n)  p = A'\*(B-A\*x)  s=p  y=norma(s)  ####################################### работа цикла МСГ  k = **0**  **while**(y> norm\_residual && k<limit)  q = A\*p  a = y/norma(q)  x = x + a\*p  s = s - a\*A'\*q  y1 = norma(s)  b = y1/y  p = s + b\*p  k=k+**1**  y=y1  **end**  display(convert(Array{**Float64**},x)) # Отображение в консоли массива x в  #формате Float64  ####################################### запись результатов в файлы  **if** !(isdir(string(path,"/result")))  mkdir(string(path,"/result")) # Создание директории result в случае  # её отсутствия.  **end**  file = open(string(path, "/result/resultX.txt"), "w")  writedlm(file, convert(Array{**Float64**},x)) # Вектор x записывается в файл  # result.txt в формате Float64.  close(file)  # Запись параметров в файл param.txt  file = open(string(path, "/result/param.txt"), "w")  write(file, string("y = ",y , "**\n**", "norm\_residual = ", norm\_residual, "**\n**", "mantissa = ", mantissa))  close(file)  file = open(string(path, "/result/resultTime.txt"),"w")  timer = toc() # Завершение работы таймера  writedlm(file, timer) # Запись времени работы программы в  # секундах в файл resultTimer.txt  close(file) |

config.txt – конфигурационный файл, содержит параметры необходимые для работы программы. Во время работы программы, файл считывается и происходит поиск параметров по ключевым словам, а именно: limit, Godunov\_matrix\_dim, norm\_residual, mantissa.

Cодержимое конфигурационного файла config.txt:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # ограничение на количество итераций  limit = **1000**  # размерность матрицы Годунова  Godunov\_matrix\_dim = **10**  # максимально допустимое отклонение нормы градиента в точке  norm\_residual = **1e-100**  # мантисса  mantissa = **1000** |

readFile.jl – модуль программы godunovMatrix.jl. В нем происходит считывание параметров: limit, Godunov\_matrix\_dim, norm\_residual, mantissa.

Листинг модуля readFile.jl:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | file = open(string(path, "/config.txt"),"r") # открытие файла config.txt  strMas = readlines(file) # в strMas записывается всё содержимое файла  # config.txt  **global** norm\_residual  **global** limit  **global** Godunov\_matrix\_dim  **global** mantissa  **for** i **in** **1**:**1**:length(strMas) # считывается значение переменной limit  **if** search(strMas[i], "limit")!=**0**:-**1** # Если подстрока “limit” найдена  limit = split(strMas[i],"= ")[**2**] # строка i разделяется на массив  # строк через символ “= ” и переменной limit  # присваивается только 2 элемент этого массива.  limit = parse(**Int64**,limit) # Строка limit конвертируется в Int64.  Break # цикл for прерывается при нахождении ключевого слова.  **end**  **end**  **for** i **in** **1**:**1**:length(strMas) # Считывается значение Godunov\_matrix\_dim  # Аналогично считыванию значения limit.  **if** search(strMas[i], "Godunov\_matrix\_dim")!=**0**:-**1**  Godunov\_matrix\_dim = split(strMas[i],"= ")[**2**]  Godunov\_matrix\_dim = parse(**Int64**,Godunov\_matrix\_dim)  **break**  **end**  **end**  **for** i **in** **1**:**1**:length(strMas) # максимально допустимое отклонение нормы  # градиента.  # Аналогично считыванию значения limit.  **if** search(strMas[i], "norm\_residual")!=**0**:-**1**  norm\_residual = split(strMas[i],"= ")[**2**]  norm\_residual = parse(**Float64**,norm\_residual)  **break**  **end**  **end**  **for** i **in** **1**:**1**:length(strMas) # Мантисса  # Аналогично считыванию значения limit.  **if** search(strMas[i], "mantissa")!=**0**:-**1**  mantissa = split(strMas[i],"= ")[**2**]  mantissa = parse(**Int64**,mantissa)  **break**  **end**  **end**  close(file) #закрытие файла config.txt |

В результате работы программы в каталоге с исполняющимся файлом создается папка result, в которой создаются файлы param.txt, resultTime.txt, resultX.txt.

Param.txt содержит параметры с которыми запускалась программа: мантисса, допустимое отклонение нормы градиента и содержит конечное, полученное в результате работы программы, отклонение нормы градиента.

result.txt – содержит вектор X.

resultTime.txt – время работы программы в секундах.

**3.2 Тестирование программы godunovMatrix.jl**

Тестирование программы проводилось при фиксированных параметрах:

* Максимально допустимая норма градиента = 1e-100;
* Размерность матрицы Годунова = 50;
* Ограничении на количество итераций = 1000.

В тестах далее будет рассмотрена зависимость между величиной мантиссы и временем работы программы.

Первый тест производился при значении мантиссы равном 1000.

На рисунке 3.1 представлен результат работы программы в консоли. Время работы программы заняло 1.1814261 секунд.

рис 3.1 Результат работы программы при мантиссе = 1000.

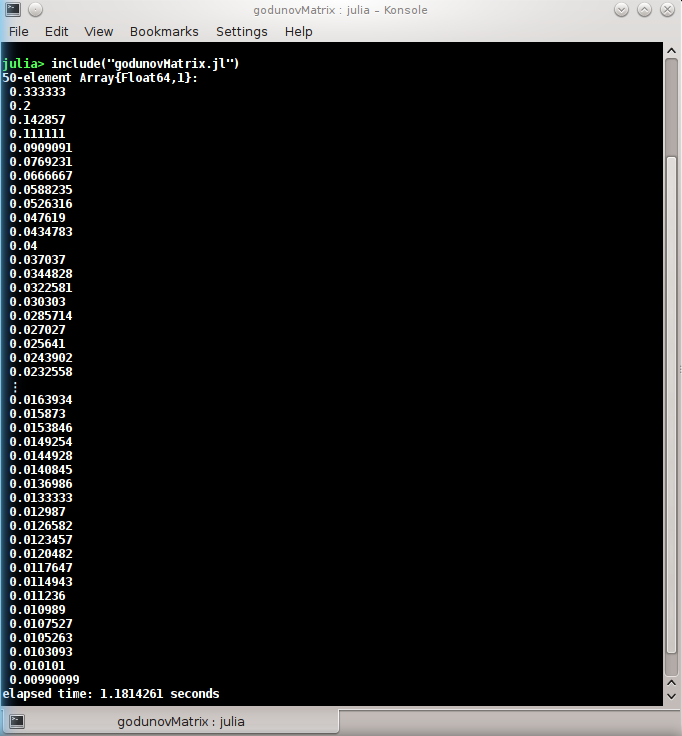
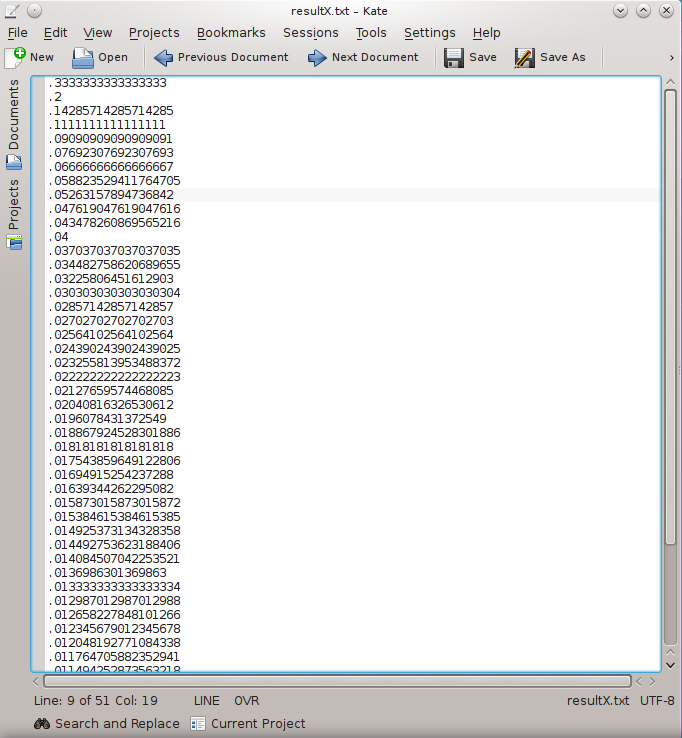


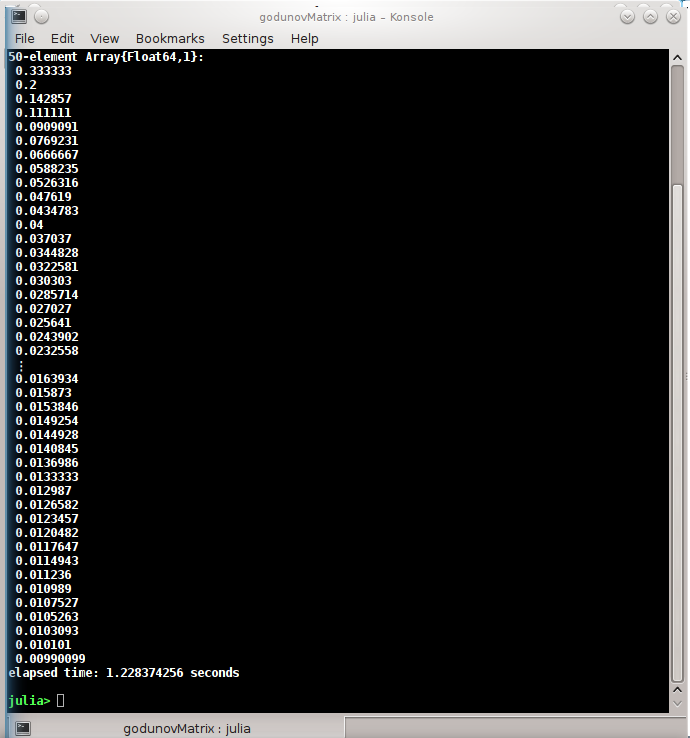
рис 3.2 содержимое файла result.txt



Второй тест производился при значении мантиссы равном 2000.

На рисунке 3.3 представлен результат работы программы в консоли. Время работы программы заняло 1.228374256 секунды.

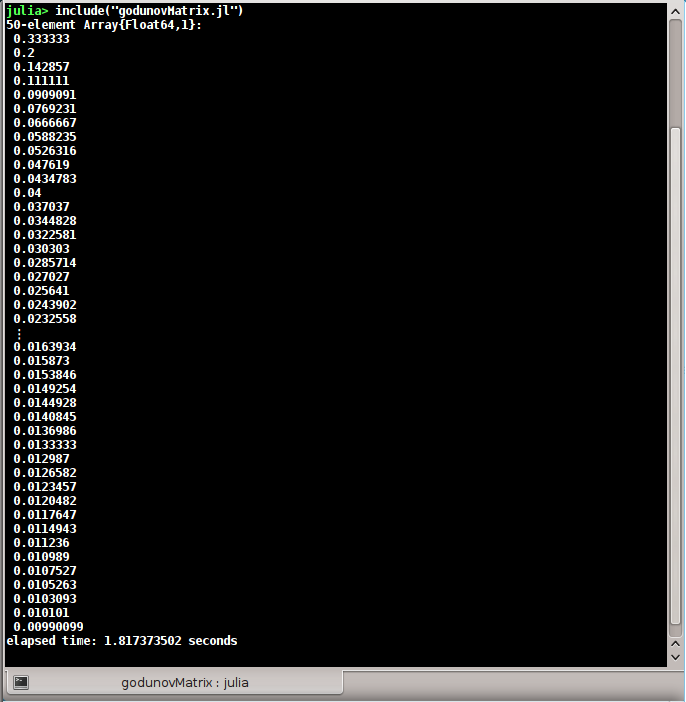
рис 3.3 Результат работы программы при мантиссе = 2000.



Третий тест производился при значении мантиссы равном 16000.

На рисунке 3.4 представлен результат работы программы в консоли. Время работы программы заняло 1.817373502 секунды.

рис 3.4 Результат работы программы при мантиссе = 16000.



**Список использованных источников:**

1. Ivo Balbaert «Getting Started with Julia».

2. Jonathan Richard Shewchuk «An Introduction to the Conjugate Gradient Method Without the Agonizing Pain» August 4, 1994.

3. <https://ru.wikipedia.org/wiki/Julia>

4. Киреев И.В. «Метод сопряженных градиентов» - Красноярск, 2011

5. Годунов С.К. Решение систем линейных уравнений - Новосибирск: Наука, 1980.

6. <http://docs.julialang.org/en/release-0.5>

7. https://ru.wikipedia.org/wiki/Число\_одинарной\_точности