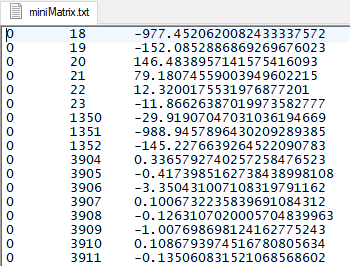
Основная задача – решение больших систем линейных уравнений в числах высокой точности.

Для отладки программы взята система, полученная Ю.Ю. Ушаковым.

В исходном виде матрица записана в разряженном виде:



1 столбец – номера строк; 2 столбец – номера столбцов; 3 столбец - значения.

Будем предполагать, что точности Float64 для записи значения не хватает. Поэтому для хранения придется использовать повышенную точность.

В Julia числа высокой точности представлены в виде типа данных BigFloat.

**Найден метод подкачки массива** [**Base.Mmap.mmap**](https://docs.julialang.org/en/stable/stdlib/io-network/#Base.Mmap.mmap-Tuple{Any,Type,Any,Any})

Whos() – отображает содержащиеся в памяти элементы.

Записываем матрицу командой write.

Командой Mmap.mmap можно считывать первые m строк и n столбцов из бинарного файла. Следовательно, реализовав особую функцию матричного умножения, можно решать СЛАУ с матрицами любого размера.

Пример использования:

Mmap.mmap(file, Matrix{Float64}, (m,n))

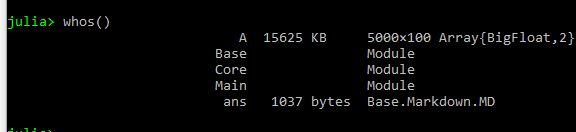
Где file – открытый файл на чтение, m – количество строк, n – количество столбцов в матрице.

Не поддерживает BigFloat!

**Проба JLD**

Создаем массив типа BigFloat размерности 5000\*100 элементов.

A = Array{BigFloat}(rand(5000,100))



Созданный массив занимает 15 Мбайт оперативной памяти.

Пробуем записать на диск при помощи JLD.

Создается файл saveA.jld, в который записывается массив A. Ключ «Ar» указывает доступ к массиву, записывающемуся в файле.

save("saveA.jld", "Ar", A)

Для открытия записанного файла используется команда:

file = jldopen("saveA.jld","r", mmaparrays=true)

Параметр mmaparrays указывает на использование подкачки. То есть, файл не будет сразу закачан в оперативную память, а будет подгружать указанные пользователем элементы в оперативную память при обращении к элементам открытого файла.

Теперь можно брать только желаемую часть массива следующим образом:

file["Ar"][1:10,1:5]

“Ar” служит указателем на массив A, хранящийся в «saveA.jld». Вторые квадратные скобки указывают на границу массива, которую хотим взять.

Следует обратить внимание, что в примере выше записывалась матрица, которая уже существовала в оперативной памяти. Для более больших матриц, которые не будут умещаться в оперативной памяти, потребуется записывать jld файл частями.

**Создание структуры хранения разряженной матрицы**

Для хранения значений используется тип BigFloat. Поэтому потребуется использовать JLD для записи массива BigFloat.

Хранить значения типа Int в оперативной памяти не требует много места в сравнении с типом BigFloat.

Чтобы иметь быстрый доступ к строкам или ненулевым столбцам создадим структуру индексов. В Julia создадим тип данных содержащий 3 поля: число - номер ненулевой строки, массив - номера ненулевых элементов в строке, массив - номер

Запишем в 3 файла разряженную матрицу. 1 бинарный файл – mapped Array содержит все ненулевые значения разряженной матрицы. 2 и 3 бинарные файлы представляет собой структуры, позволяющие по номеру строки находить все ненулевые элементы в строке и определять номер, под которым в исходном файле существует запись об этом ненулевом элементе.

**Проба создания структуры хранения индексов.**

Формируем тип myType:

type myType{}

i::Int

jArray::Array{Int}

rowNumberArray::Array{Int}

end

i – номер ненулевой строки. Предполагается, что все строки ненулевые;

jArray – массив номеров столбцов j, таких что пересечение строки i со столбцом j дает ненулевой элемент.

rowNumberArray – массив номеров строк исходного файла в которых содержится запись об i, j элементе.

findin(a:Array, b:Array) – возвращает номера вхождений элементов из b в массиве a.