|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра параллельных вычислительных технологий | | |
| Лабораторная работа № 1 | | |
| по дисциплине «Численные методы» | | |
| **Машинная арифметика. Форматы хранения разреженных****квадратных матриц** | | |
|  | | |
|  |  |  |
| Группа ПМИ-91 | Жарков федор |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | Марков сегрей игоревич |
|  |  |
| Новосибирск, 2021 | | |

1. **Цель работы**

Изучить принципы работы с вещественной арифметикой, форматы хранения разреженных матриц.

1. **Задание**
2. Разработать подпрограмму чтения бинарных файлов матрицы в следующем разреженном строчно-столбцовом формате (CSlR)

//size.bin: размер матрицы (тип int)

N

//iptr.bin: индексный массив начала строк/столбцов (тип int)

I1 I2 … IN+1

//jptr.bin: индексный массив столбцов/строк (тип int)

I1 I2 … IM

//altr.bin: нижний треугольник (тип double)

A1 A2 … AM

//autr.bin: верхний треугольник (тип double)

A1 A2 … AM

//di.bin: диагональ (тип double)

A1 A2 … AN

Матрицы для лабораторной работы выдаются преподавателем. В массивах altr и autr могут присутствовать нулевые элементы.

1. Разработать подпрограмму перевода разреженной матрицы из формата CSlR в плотный формат.
2. Разработать подпрограммы умножения матриц, хранящихся в плотном и разреженном форматах, на произвольный вектор длины N.
3. Выполните замер времени умножения матрицы с учётом формата её хранения (плотный или разреженный) на произвольный вектор длины N.
4. Задайте вектор
   1. Реализуйте следующую итерационную процедуру с учётом хранения матрицы Aв формате CSlR

*Выполнять пока не произойдёт особая ситуация*

* 1. Реализуйте следующую итерационную процедуру с учётом хранения матрицы Aв формате CSlR

*Выполнять пока не произойдёт особая ситуация*

Евклидова норма имеет вид

В отчёте привести результаты итераций, на которых возникли особые ситуации. Дать объяснение, предложить модификацию итерационной процедуры для обработки особых ситуаций.

1. **Программа**

Ссылка на репозиторий: <https://github.com/SoBark/Machine_Arithmetic>

1. **Ход работы**
2. Подпрограмма чтения бинарных файлов матрицы в разреженном строчно-столбцовом формате (CSlR):

Sparse\_Matrix\_CSLR::Sparse\_Matrix\_CSLR(const std::string PATH)

{

//размер матрицы

std::ifstream Reader(PATH + "size.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File size.bin was not found...");

Reader.read((char\*)&N, sizeof(N));

Reader.close();

//di

Reader.open(PATH + "di.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File di.bin was not found...");

di.resize(N);

for (int i = 0; i < N; i++)

Reader.read((char\*)&di[i], sizeof(double));

Reader.close();

//iptr

Reader.open(PATH + "iptr.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File iptr.bin was not found...");

iptr.resize(N + 1);

for (int i = 0; i < N + 1; i++)

Reader.read((char\*)&iptr[i], sizeof(int));

Reader.close();

//jptr

Reader.open(PATH + "jptr.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File jptr.bin was not found...");

int JPTR\_SIZE = iptr[N] - 1;

jptr.resize(JPTR\_SIZE);

for (int i = 0; i < JPTR\_SIZE; i++)

Reader.read((char\*)&jptr[i], sizeof(int));

Reader.close();

//altr

Reader.open(PATH + "altr.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File altr.bin was not found...");

altr.resize(JPTR\_SIZE);

for (int i = 0; i < JPTR\_SIZE; i++)

Reader.read((char\*)&altr[i], sizeof(double));

Reader.close();

//autr

Reader.open(PATH + "autr.bin", std::ios::binary);

if (!Reader.is\_open())

throw std::exception("File autr.bin was not found...");

autr.resize(JPTR\_SIZE);

for (int i = 0; i < JPTR\_SIZE; i++)

Reader.read((char\*)&autr[i], sizeof(double));

Reader.close();

}

1. Подпрограмма перевода разреженной матрицы из формата CSlR в плотный формат

void Sparse\_Matrix\_CSLR::transfer\_to\_dense(std::vector<std::vector<double>>& Matrix)

{

Matrix.resize(N);

for (int i = 0; i < N; i++)

Matrix[i].resize(N, 0.0);

//di

for (int i = 0; i < N; i++)

Matrix[i][i] = di[i];

//Заполнение верхнего и нижнего треугольников

for (int i = 0; i < N; i++)

for (int j = iptr[i] - 1; j < iptr[i + 1] - 1; j++)

{

Matrix[i][jptr[j] - 1] = altr[j];

Matrix[jptr[j] - 1][i] = autr[j];

}

}

1. Подпрограмма умножения матрицы, хранящихся в **плотном** формате, на произвольный вектор длины N:

void Dense\_Matrix::matrix\_mult\_vector(std::vector<double>& X, std::vector<double>& Res)

{

for (int i = 0; i < Res.size(); i++) Res[i] = 0;

for (int i = 0; i < Matrix.size(); i++)

for (int j = 0; j < Matrix.size(); j++)

Res[i] += Matrix[i][j] \* X[j];

}

Подпрограмма умножения матрицы, хранящихся в **разреженном** формате, на произвольный вектор длины N:

void Sparse\_Matrix\_CSLR::matrix\_mult\_vector(std::vector<double> & X, std::vector<double>& Res)

{

//инициализация результата через умножения вектора на диагональ

for (int i = 0; i < N; i++) Res[i] = X[i] \* di[i];

//проход по всем строкам и столбцам с учётом формата

for (int i = 0; i < N; i++)

for (int j = iptr[i] - 1; j < iptr[i + 1] - 1; j++)

{

Res[i] += X[jptr[j] - 1] \* altr[j];

Res[jptr[j] - 1] += X[i] \* autr[j];

}

}

1. Замеры времени работы написанных подпрограмм произведения матрицы на вектор, на примере матриц вида:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Время для матрицы в формате CSLR | | Время для матрицы в плотном формате | |
| Замеры | Среднее | Замеры | Среднее |
| 7 | 8e-07 | 9e-07 | 1.9e-06 | 12.7e-07 |
| 1.1e-06 | 1e-06 |
| 8e-07 | 9e-07 |
| 8 | 6e-07 | 8.3e-07 | 9e-07 | 11.7e-07 |
| 7e-07 | 1.2e-06 |
| 1.2e-06 | 1.4e-06 |
| 9 | 9e-07 | 8e-07 | 1e-06 | 9.3e-07 |
| 7e-07 | 8e-07 |
| 8e-07 | 1e-06 |

1. Были реализованы следующие процедуры

double EuclidianNorm(std::vector<double>& Y)

{

double Norm = 0.0;

for (int i = 0; i < Y.size(); i++)

Norm += Y[i] \* Y[i];

return sqrt(Norm);

}

void Special\_case\_mult(Sparse\_Matrix\_CSLR& Matrix)

{

int steps = STEPS;

std::vector<double> X(Matrix.N, 0.0), Y(Matrix.N);

X[0] = 1.0;

std::cout << "multiplication\n||Y||\tx1" << std::endl;

for (int i = 0; i < steps; i++)

{

Matrix.matrix\_mult\_vector(X, Y);

std::cout << EuclidianNorm(Y) << "\t" << X[0] << std::endl;

X[0] \*= 10;

}

std::cout << std::endl;

}

void Special\_case\_div(Sparse\_Matrix\_CSLR& Matrix)

{

int steps = STEPS;

std::vector<double> X(Matrix.N, 0.0), Y(Matrix.N);

X[0] = 1.0;

std::cout << "division\n||Y||\tx1" << std::endl;

for (int i = 0; i < steps; i++)

{

Matrix.matrix\_mult\_vector(X, Y);

std::cout << EuclidianNorm(Y) << "\t" << X[0] << std::endl;

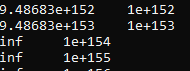
X[0] /= 10;

}

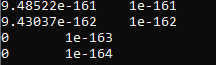
std::cout << std::endl;

}

В случае когда x1 растет, особая ситуация появляется после 153 итерации:



В случае когда x1 уменьшается, особая ситуация появляется после 162 итерации:



Данные результаты объясняются особенностями представления вещественных чисел в оперативной памяти компьютера. В современных системах чаще всего используется стандарт **IEEE 754**, описывающий формат представления чисел с плавающей точкой. В нем описано 4 формата представления чисел с плавающей точкой, в том числе и используемый в данной программе формат двойной точностью (вещественный тип double). В этом формате **максимальное** положительное допустимое значение равно +1,797693134862315708e+308, а **минимальное** положительное допустимое значение равно +4,940656458412465441e-324. В процессе вычисления евклидовой нормы, получаемое значение начинает в первом случае превышать (а во втором принижать) допустимые значения, так как при возведении в квадрат одно из значений становиться слишком большим (все значения становиться слишком маленькими) и не помешаются в данный формат. Данную проблему можно попытаться решить вынося общий множитель при вычислении евклидовой нормы, тем самым уменьшая показатель степени.

Улучшенные подпрограммы:

void Special\_case\_mult\_improved(Sparse\_Matrix\_CSLR& Matrix)

{

int steps = STEPS; //количество итераций

std::vector<double> X(Matrix.N, 0.0), Y(Matrix.N);

X[0] = 1.0;

std::cout << "multiplication\n||Y||\tx1" << std::endl;

for (int i = 0; i < steps; i++)

{

Matrix.matrix\_mult\_vector(X, Y);

double Norm = 0.0, nextNorm = 0.0, factor = 1.0;

for (int j = 0; j < Y.size(); j++) {

nextNorm = Norm + std::pow(Y[j]\*factor, 2.0);

if (nextNorm >= std::numeric\_limits<double>::max())

{

//подбор множителя

while (nextNorm >= std::numeric\_limits<double>::max() && factor >= std::numeric\_limits<double>::min()) {

factor /= 10;

Norm /= 100; //вынесение множителя

nextNorm = Norm + std::pow(factor \* Y[j], 2.0);

}

}

Norm = nextNorm;

}

std::cout << sqrt(Norm)/factor << "\t" << X[0] << std::endl;

X[0] \*= 10;

}

std::cout << std::endl;

}

void Special\_case\_div\_improved(Sparse\_Matrix\_CSLR& Matrix)

{

int steps = STEPS; //количество итераций

std::vector<double> X(Matrix.N, 0.0), Y(Matrix.N);

X[0] = 1.0;

std::cout << "multiplication\n||Y||\tx1" << std::endl;

for (int i = 0; i < steps; i++)

{

Matrix.matrix\_mult\_vector(X, Y);

double Norm = 0.0, nextNorm = 0.0, factor = 1.0;

for (int j = 0; j < Y.size(); j++) {

nextNorm = Norm + std::pow(Y[j] \* factor, 2.0);

if (nextNorm <= std::numeric\_limits<double>::min())

{

//подбор множителя

while (nextNorm <= std::numeric\_limits<double>::min() && factor <= std::numeric\_limits<double>::max()) {

factor \*= 10;

Norm \*= 100; //вынесение множителя

nextNorm = Norm + std::pow(factor \* Y[j], 2.0);

}

}

Norm = nextNorm;

}

std::cout << sqrt(Norm) / factor << "\t" << X[0] << std::endl;

X[0] /= 10;

}

std::cout << std::endl;

}

Особые ситуации для улучшенных подпрограмм:

