ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий
Высшая школа программной инженерии

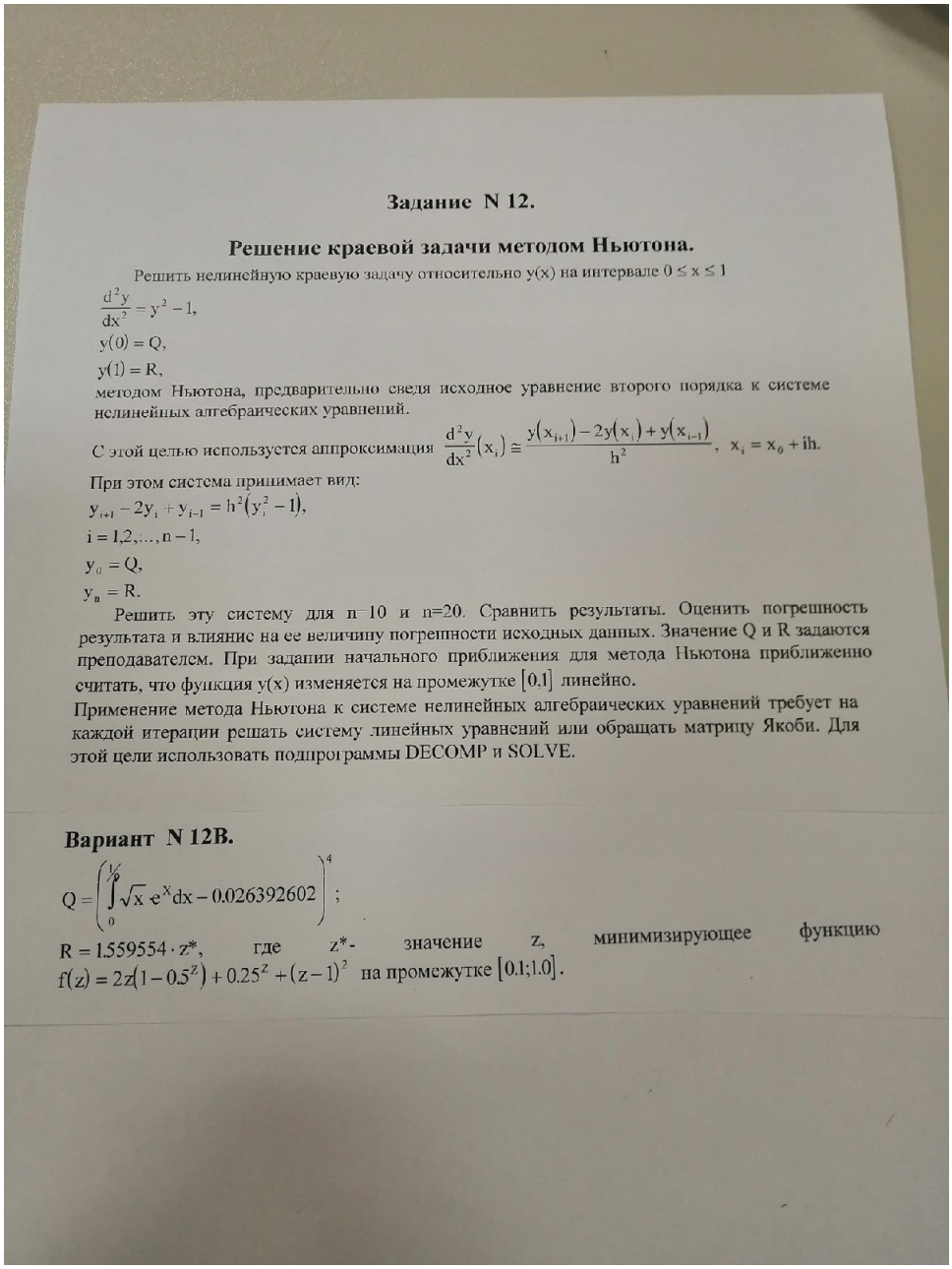
Отчет по курсовой работе по дисциплине «Вычислительная математика»

Выполнила студентка гр. 3530904/80001 Прохорова А. И.

Руководитель Устинов С. М.

Санкт-Петербург
2020

1 Задание



Задание N 12.

Решение краевой задачи методом Ньютона.

Решить нелинейную краевую задачу относительно y(x) на интервале $0 \le x \le 1$

$$\frac{d^2y}{dx^2} = y^2 - 1,$$

$$y(0) = Q,$$

$$y(1) = R,$$

методом Ньютона, предварительно сведя исходное уравнение второго порядка к системе нелинейных алгебраических уравнений.

С этой целью используется аппроксимация $\frac{d^2y}{dx^2}(x_i) \cong \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2}$, $x_i = x_0 + ih$.

При этом система принимает вид:

$$y_{i+1} - 2y_i + y_{i-1} = h^2(y_i^2 - 1),$$

$$i = 1,2,\ldots,n-1,$$

$$y_0 = Q,$$

$$y_n = R.$$

Решить эту систему для n=10 и n=20. Сравнить результаты. Оценить погрешность результата и влияние на ее величину погрешности исходных данных. Значение Q и R задаются преподавателем. При задании начального приближения для метода Ньютона приближенно считать, что функция y(x) изменяется на промежутке $[0.1]$ линейно.

Применение метода Ньютона к системе нелинейных алгебраических уравнений требует на каждой итерации решать систему линейных уравнений или обращать матрицу Якоби. Для этой цели использовать подпрограммы DECOMP и SOLVE.

Вариант N 12B.

$$Q = \left(\int_0^{1/6} \sqrt{x} \cdot e^x dx - 0.026392602 \right)^4;$$

$R = 1.559554 \cdot z^*$, где z^*- значение z, минимизирующее функцию $f(z) = 2z(1 - 0.5^z) + 0.25^z + (z-1)^2$ на промежутке $[0.1;1.0]$.

2 Результаты работы программы

```
For n = 10:

Cond: 36.6667
answers: 0.134443 ; 0.259066 ; 0.37436 ; 0.481056 ; 0.580066 ; 0.672441 ; 0.759337 ;
    0.842 ; 0.921752 ;
epsilon: 3.43941e-07 ; 6.80172e-07 ; 9.87026e-07 ; 1.23227e-06 ; 1.37868e-06 ;
    1.39173e-06 ; 1.2487e-06 ; 9.47506e-07 ; 5.13217e-07 ;
For n = 20:

Cond: 125.473
answers: 0.0684414 ; 0.134394 ; 0.197893 ; 0.258989 ; 0.317753 ; 0.374269 ; 0.428635
    ; 0.480961 ; 0.531365 ; 0.579975 ; 0.626926 ; 0.672359 ; 0.716423 ; 0.75927 ;
    0.801058 ; 0.84195 ; 0.882114 ; 0.921724 ; 0.960958 ;
epsilon: 2.03629e-07 ; 4.05589e-07 ; 6.01025e-07 ; 7.82217e-07 ; 9.39133e-07 ;
    1.06025e-06 ; 1.13366e-06 ; 1.14849e-06 ; 1.09642e-06 ; 9.7325e-07 ; 7.8022e-07 ;
     6.28188e-07 ; 5.06371e-07 ; 4.06562e-07 ; 3.22425e-07 ; 2.49034e-07 ; 1.82581e
    -07 ; 1.20185e-07 ; 5.97717e-08 ;
```

3 Выводы

Для n = 10 и n = 20 через один результаты очень близки и расположены в интервале от 0 до 1 в порядке возрастания, следовательно делаем вывод, что полученным значениям можно доверять.

4 Приложение

4.1 Код solveSystem.cpp

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>
#include "../cmath.h"
#include "matrix.hpp"
#include <cmath>

void fillRightPart(double h, std::vector<double>& y,
  std::vector<double>& b, double Q, double R)
{
  b[0] = - (y[1] - 2 * y[0] + Q - h * h * y[0] * y[0]  + h*h);
  int n = b.size();
  for (int i = 1; i < n; ++i)
  {
    b[i] = - (y[i+1] - 2 * y[i] + y[i-1] - h*h* y[i] * y[i] + h*h);
  }
  b[n-1] = - (R - 2 * y[n - 1] + y[n-2] - h*h* y[n -1] * y[n-1] + h*h);
}
void Jacoby(int n, double h, double* x, Matrix<double>& matrix)
{
  for (int i = 0; i < n; ++i)
  {
    matrix[i][i] = -2 - 2 * h * h * x[i];
    if (i > 0)
    {
      matrix[i][i-1] = 1;
    }
    if (i < n-1)
    {
```

```
      matrix[i][i+1] = 1;
    }
  }
}
bool checkEpsilon(int n, double* epsilon)
{
  for (int i = 0; i < n; ++i)
  {
    if (std::abs(epsilon[i]) > 0.00001)
    {
      return true;
    }
  }
  return false;
}
void solveSystem(int n, double h)
{
  Matrix<double> matrix(n);
  for (int i = 0; i < n; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
      matrix[i][j] = 0;
    }
  }
  std::vector<int> pivot(n);
  std::vector<double> b(n);
  std::vector<double> y(n);
  std::vector<double> y0(n);
  double Q = 0;
  double R = 1;
  for (int i = 0; i < n; ++i)
  {
    y0[i] = i / 10;
    y[i] = y0[i];
  }
  Jacoby(n, h, y0.data(), matrix);
  double cond = 0;
  int flag = 0;


  decomp(n, n, matrix.data(), &cond, pivot.data(), &flag);

  if (flag != 0)
  {
    throw std::runtime_error("Error occured in program decomp.\n");
  }

  std::cout << "\nCond: " << cond << "\n";

  do {
    fillRightPart(h, y, b, Q, R);

    solve(n, n, matrix.data(), b.data(), pivot.data());

    for (int i = 0; i < n; ++i)
    {
      y[i] += b[i];
    }
  }
  while(checkEpsilon(n, b.data()));
  std::cout << "answers: ";
  std::copy(y.begin(), y.end(), std::ostream_iterator<double>(std::cout, " ; "));
```

```
  std::cout << "\nepsilon: ";
  std::copy(b.begin(), b.end(), std::ostream_iterator<double>(std::cout, " ; "));
  std::cout << "\n";
}
int main()
{
  int n = 9;
  double h = 0.1;
  std::cout << "For n = 10:\n";
  solveSystem(n, h);
  n = 19;
  h = 0.05;
  std::cout << "For n = 20:\n";
  solveSystem(n, h);
  return 0;
}
```

4.2 Код matrix.hpp

```
#ifndef COMPUTATIONAL_MATH_LAB2_MATRIX
#define COMPUTATIONAL_MATH_LAB2_MATRIX

#include <vector>
#include <iosfwd>
#include <cmath>
#include <stdexcept>
#include <iterator>

template<typename T>
class Matrix
{
public:
  using iterator = typename std::vector<T>::iterator;
  using const_iterator = typename std::vector<T>::const_iterator;
  class Row
  {
  public:
    Row(T* begin, size_t size);
    T& operator[](size_t index);
  private:
    T* begin_;
    size_t size_;
  };

  Matrix(size_t size);
  Row operator[](size_t index);
  iterator begin();
  iterator end();
  const_iterator begin() const;
  const_iterator end() const;
  size_t size() const;
  void resize(size_t size);
  T* data();

private:
  std::vector<T> data_;
};

template<typename T>
Matrix<T>::Row::Row(T* begin, size_t size):
  begin_(begin),
  size_(size)
```

```
{
}
template<typename T>
T& Matrix<T>::Row::operator[](size_t index)
{
  if (index > size_)
  {
    throw std::invalid_argument("Index is out of bounds.");
  }
  return begin_[index];
}
template<typename T>
typename Matrix<T>::Row Matrix<T>::operator[](size_t index)
{
  return Row(&data_[0] + index * size(), size());
}
template<typename T>
Matrix<T>::Matrix(size_t size):
  data_(size * size)
{
}
template<typename T>
typename Matrix<T>::const_iterator Matrix<T>::begin() const
{
  return data_.begin();
}
template<typename T>
typename Matrix<T>::const_iterator Matrix<T>::end() const
{
  return data_.end();
}
template<typename T>
typename Matrix<T>::iterator Matrix<T>::begin()
{
  return data_.begin();
}
template<typename T>
typename Matrix<T>::iterator Matrix<T>::end()
{
  return data_.end();
}
template<typename T>
size_t Matrix<T>::size() const
{
  return sqrt(data_.size());
}
template<typename T>
void Matrix<T>::resize(size_t size)
{
  data_.resize(size * size);
}
template<typename T>
T* Matrix<T>::data()
{
  return &data_[0];
}

template <typename T>
std::ostream& operator<<(std::ostream& out, Matrix<T>& matrix)
{
  for (auto it = matrix.begin(); it != matrix.end(); it += matrix.size())
  {
    std::copy(it, it + matrix.size(), std::ostream_iterator<T>(out, " ; "));
```

```
107       out << "\n";
108     }
109     return out;
110 }
111
112 #endif // COMPUTATIONAL_MATH_LAB2_MATRIX
```