Metody numeryczne

Zadanie 1

Piotr Drabik

Treść zadania

2. Solve numerically Eq.(F) by Gauss elimination without pivoting

   Equation F:

A	x		b
2 3 5	x1		0
3 1 -2	x2	=	-2
1 3 4	x2		-3

Solution :

x0 = 1.5
x1 = -3.5
x2 = 1.5

Metoda eliminacji Gaussa

Metoda eliminacji Gaussa służy do rozwiązywania układów równań pierwszego stopnia, polega na sprowadzeniu macierzy powstałej z równań do postaci macierzy trójkątnej, czyli o uzyskanie zera pod przekątną (przyjęło się, że pod przekątną jednak można też nad przekątną) macierzy.

Implementacja

Potrzebne biblioteki

```
#include <array>
#include <iostream>
#include <vector>
```

Pomocnicza klasa „macierz”

```
template <size_t H, size_t W> class Mat {
public:
  Mat() {
    for (auto &i : data_)
      i.fill(0.0);
  }

  Mat(const Mat<W, H> &other) {
    unsigned height = w_ < other.w_ ? w_ : other.w_;
    unsigned width = w_ < other.w_ ? w_ : other.w_;

    for (size_t x = 0; x < height; x++)
      for (size_t y = 0; y < width; y++)
        data_[x][y] = other.data_[x][y];

  }

  Mat &operator=(const Mat<H, W> &other) {
    if (this == &other)
      return *this;
    data_ = other.data_;
    return *this;
  }

  void FromVec(std::vector<std::vector<double>> in_data) {
   for (size_t x = 0; x < h_; x++)
      for (size_t y = 0; y < w_; y++)
        data_[x][y] = in_data[x][y];
  }

  template <size_t N, size_t M>
  void FromMat(const Mat<N, M> &other) {
    unsigned height = w_ < other.w_ ? w_ : other.w_;
    unsigned width = w_ < other.w_ ? w_ : other.w_;

     for (int i = 0; i < height; i++)
      for (int j = 0; j < width; j++)
        data_[i][j] = other.data_[i][j];

  }

  double &At(size_t x, size_t y) { return data_[x][y]; }

  friend std::ostream &operator<<(std::ostream &out, const Mat<H, W> &ref) {
    for (size_t x = 0; x < ref.h_; x++) {
      for (size_t y = 0; y < ref.w_; y++)
        out << ref.data_[x][y] << '\t';
      out << '\n';
    }
    return out;
  }

  const size_t w_ = W;
  const size_t h_ = H;

  std::array<std::array<double, W>, H> data_;

private:
};
```

Deklaracja funkcji

Funkcja „GaussianElimination” jest w stanie rozwiązywać równania dowolnych rozmiarów,
Tak długo macierz współczynników stojących przy niewiadomych x jest macierzą kwadratową N x N
a wektor wyrazów wolnych b (tutaj array) jest długości N.

Wynikiem działania algorytmu będzie wektor długości N.

```
template <size_t N>
std::array<double, N> GaussianElimination(Mat<N, N> &matrix, std::array<double, N> &array);
```

Funkcja main

Domyślnie do funkcji „GaussianElimination” przekazywane są wartości równania z zadania.

```
int main() {

  Mat<3, 3> matrix;

  matrix.FromVec({{2, 3, 5}, {3, 1, -2}, {1, 3, 4}});

  std::array<double, 3> b = {0, -2, -3};

  auto solution = GaussianElimination(matrix, b);

  std::cout << "result:";
  for (int i = 0; i < solution.size(); i++)
    std::cout << "\nx" << i + 1 << " = " << solution[i];

  return 0;
}
```

Definicja funkcji

```
template <size_t N>
std::array<double, N> GaussianElimination(Mat<N, N> &matrix, std::array<double, N> &array) {
```

Zmieniamy rozmiar macierzy kwadratowej na prostokątną, by „zrobić miejsce” na kolumnę wyrazów wolnych, tym samym tworząc macierz rozszerzoną.

```
  Mat<N, N + 1> mat;
  mat.FromMat(matrix);
```

Dodajemy do nowej macierzy kolumnę wyrazów wolnych.

```
 for (int i = 0; i < N; i++)
    mat.At(i, N) = array[i];
```

Rezerwujemy miejsce na wektor wartości wynikowych.

```
  std::array<double, N> solution{};
```

Wykonujemy eliminację Gaussa.

Przekształcać będziemy powstałą macierz w macierz trójkątną górną,
aby tego dokonać redukować będziemy kolejne elementy macierzy zaczynając od a(2,1) przez a(3,1) aż do a(N,1)
następnie wyeliminujemy a(3,2) do a(N,2) itd...

```
for (int n = 0; n < N - 1; n++) {

    for (int j = n + 1; j < N; j++) {
      double f = mat.At(j, n) / mat.At(n, n);

      for (int k = 0; k < N + 1; k++)
        mat.At(j, k) -= f * mat.At(n, k);
    }
  }
```

Obliczamy wartości wynikowe na podstawie wyliczonej macierzy.

```
for (int n = N - 1; n >= 0; n--) {
    solution[n] = mat.At(n, N);

    for (int j = n + 1; j < N; j++)
      if (n != j)
        solution[n] -= mat.At(n, j) * solution[j];

    solution[n] /= mat.At(n, n);
  }
```

Zwracamy otrzymane wartości.

```
  return solution;
}
```

Wynik działania programu

```
result:
x1 = 1.5
x2 = -3.5
x3 = 1.5
```