Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«Введение в численные методы
Задание 1»

ОТЧЕТ

о выполненном задании

студента 203 учебной группы факультета ВМК МГУ
Травниковой Арины Сергеевны

гор. Москва
2018 г.

Содержание

Цели

- Часть 1

  Изучить классическую метод Гаусса (а также модифицированный метод Гаусса), применяемый для решения сислетмы линейных алгебраических уравнений:

  - Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента
  - Вычислить определителю матрицы $det(A)$
  - Вычислить обратную матрицу A^{-1}
  - Определить число обусловленности $M_A = ||A|| * ||A^{-1}||$
  - Исследовать вопрос вычислительной устойчивости метода Гаусса
  - Проверить правильность решения СЛАУ на различных тестах, используя wolframalpha.com

- Часть 2

  Изучить классические итерационные методы (Зейделя и верхней реалксации), применяемые для решения сислетмы линейных алгебраических уравнений:

  - Решить заданную СЛАУ итерационным методом Зейделя или методом верхней реалксации
  - Разработать критерий остановки итерационного процесса для гарантированного получения приближенного решения исходной СЛАУ с заданной точностью
  - Изучить скорость сходимости итераций к точному решению задачи при различных итреационных параметрах ω
  - Проверить правильность решения СЛАУ на различных тестах, используя wolframalpha.com

-

Постановка задачи

- Часть 1

  Дана система линейных уравнений $A\overline{x} = \overline{f}$ порядка $n * n$ с невырожденной матрицей A. Написать программу, решающую СЛАУ заданного пользователем размера методом Гаусса и методом Гаусса с выбором главного элемента.
  Предусмотреть возможность задания элементов матрицы системы и ее правой части как на входной файле данных, так и задания специальных формул.

- Часть 2

  Дана система линейных уравнений $A\overline{x} = \overline{f}$ порядка $n * n$ с невырожденной матрицей A. Написать программу численного решения СЛАУ заданного пользователем размера, использующую численный алгоритм итерационного метода Зейделя:

  $$(D + A^{(-)})(x^{k+1} - x^k) + Ax^k = f),$$

  где $D, A^{(-)}$ - соответственно диагональная и нижняя треугольные матрицы, k - номер текущей итерации;
  в случае использования итерационного метода верхней релакцсации итерационный процесс имеет вид:

  $$(D + \omega A^{(-)})\frac{x^{k+1} - x^k}{\omega} + Ax^k = f),$$

  где ω - итерационный параметр (при $\omega = 1$ метод переходит в метод Зейделя).
  Предусмотреть возможность задания элементов матрицы системы и ее правой части как на входной файле данных, так и задания специальных формул.

Описание алгоритмов

- Часть 1

1. Стандартный метод Гаусса.

   Первый шаг:
   Прямой ход метода Гаусса состоит из n последовательных итераций цикла на каждой из которых происходит изменений как матрицы-левой части ($A^{i-1} \Rightarrow A^i; A^0 = A$), так и вектора-правой части ($f^{i-1} \Rightarrow f^i, f^0 = f$). На каждой итерации рассматриваем i строку матрицы A^i. Найдём первый ненулевой элемент в данной строке и поменяем местами i столбец со столбцом, содержащим первый ненулевой элемент (при этом запомним, данную перестановку столбцов). Теперь обратим в 0 все элементы полученной матрицы, стоящие в i столбце, начиная с $i+1$ строки и вплоть до n. Этого можно добиться путём вычитания i строки, умноженной на $\frac{a^i_{j,i}}{a^i_{i,i}}$ из всех последующих строк $i < j \leq n$. При этом производим такие же преобразования в векторе-столбце f^i.

   Второй шаг:
   Обратный метод Гаусса также состоит из n итераций цикла. На i итерации вычисляется x^{n-i} – $n-i$ компонента вектора-ответа x. На i шаге рассмотрим $n-i$ строку. К этому моменту все $x^j, i < j \leq n$ уже вычислены. Тогда положим: $x^i = \frac{f^n_i - \sum\limits_{k=i+1}^{n} a^n_{i,k} x^k}{a^n_{i,i}}$. Таким образом, проведя все n итераций цикла, найдем искомый вектор-решение x.

2. Метод Гаусса с выбором ведущего элемента.

   Данный метод аналогичен стандартному методу Гаусса, за исключением того, что на каждом шаге прямого хода выбирается не первый нулевой элемент, а наибольший по модулю элемент в строке. Также для получения вектора-решения необходимо учесть перестановки столбцов, производимые в прямом методе.

3. Вычисление определителя матрицы.

   Проведём прямой ход метода Гаусса. При этом, заведём переменную `det`:
   - Начальное значение: $\texttt{det} = 1$
   - При перестановке столбцов: $\texttt{det} = -\texttt{det}$
   - На i итерации: $\texttt{det} = \texttt{det} * a^i_{i,i}$

   Тогда определитель A: $\texttt{det}(A) = \texttt{det} * \prod\limits_{k=1}^{n} a^n_{i,i}$

4. Вычисление обратной матрицы. Метод Гаусса-Жордана

   Рассмотрим расширенную матрицу $A|E$, где E – единичная матрица размера $n * n$. Модифицируем прямой и обратные ходы стандартного метода Гаусса.
   - Все операции над строками происходят одновременно в обеих матрицах.
   - Вместо выбора первого ненулевого элемента в строке на i шаге будем выбирать первый не нулевой элемент среди $a^i_{j,i}, i < j \leq n$ и переставлять соответствующие строки в расширенной матрице.
   - В обратном ходе метода Гаусса не будем производить вычисление x, а будем, подобно прямому ходу метода Гаусса обращать в 0 все элементы, стоящие над элементом с индексом i, i в том же столбце, а затем, разделив i строку расширенной матрицы на $a^{n+i}_{i,i}$, превратим $a^{n+i}_{i,i}$ в единицу.

Проведя данный алгоритм, получим, что расширенная матрица $A|E \Rightarrow E|A^{-1}$. Тогда взяв правую часть полученной расширенной матрицы получим искомую, обратную матрицу.

- Часть 2

1. Метод верхней релаксации
   Для написания алгоритма воспользуемся явной формулой для пересчёта вектора приближённого решения $x_i^{k+1} = x_i^k + \frac{w}{a_{i,i}}(f_i - \sum\limits_{j=1}^{i-1} a_{i,j}x_j^{k+1} - \sum\limits_{j=i}^{n} a_{i,j}x_j^k); 1 \leq i \leq n.$
   В качестве критерия остановки процесса рассмотрим следующее правило: зададим некое $\varepsilon > 0$ - точность решения. Будем на каждом щаге итерации вычислять следующую величину: $\rho(x^{k+1}, x^k) = \sqrt{\sum\limits_{i=1}^{n}(x_i^{k+1} - x_i^k)^2}$. Будем останавливать алгоритм как только $\rho(x^{k+1}, x^k) < \varepsilon$. Такой критерий гарантирует, что $|x_i^{k+1} - x_i^k| < \varepsilon$. Т.е. изменение координат вектора-решения стало достаточно мало. Помимо этого критерием останова будет служить достижение максимального числа итераций.

Описание программы

Рассмотрим ключевые функции программы:

- `Matrix.hpp/Matrix.cpp`

  Вычисление определителя:

```
1
2  int triangle(double **a, int size) //приведение матрицы к верхнетругольной форме
3  {
4    int j = 0;
5    int sign = 1;
6    for (int i = 0; i < size; i++){
7      // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
8      for (j = i; j < size; j++){
9        if (a[j][i]){
10         break;
11       }
12     }
13     if (j == size || fabs(a[j][i]) < EPS){
14       fprintf(stderr,"det 0\n");
15       exit(1);
16     }
17     if (i != j ){
18       sign *= change_str(a, i, j); // фя- меняет местами строки и возвращает -1
19     }
20     for (int k = i + 1; k < size; k++){
21       double k1 = a[i][i];
22       double k2 = a[k][i];
23       if (fabs(k1) < EPS){
24         exit(1);
25       }
26       if (fabs(k2) < EPS){
27         continue;
28       }
29       for (int l = 0; l < size ; l++){
30         a[k][l] -= a[i][l] * k2 / k1;
31       }
32     }
33   }
34   return sign;
35 }
36
37 double det(double **a1, int size)
38 {
39   double **a = copy_matrix(a1, size);
40   int sign = triangle(a, size);
41   double ans = 1;
42   for (int i = 0; i < size; i++){
43     ans *= a[i][i];
44   }
45   return ans * sign;
46 }
```

Вычисление обратной матрицы:

```
double **inverse(double **a1, int size)
{
  double **a = copy_matrix(a1, size);
  double **res = calloc(size, sizeof(*res));
  for (int i = 0; i < size; i++){
    res[i] = calloc(size, sizeof(*res[i]));
  }
  for (int i = 0; i < size; i++){
    res[i][i] = 1;
  }
  int j = 0;
  int sign = 1;
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    for (j = i; j < size; j++){
      if (a[j][i]){
        break;
      }
    }
    if (j == size || fabs(a[j][i]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    if (i != j){
      sign *= change_str(a, i, j);
      change_str(res, i, j);
    }
    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      if (fabs(k2) < EPS){
        continue;
      }
      for (int l = 0; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
        res[k][l] -= res[i][l] * k2 / k1;
      }
    }
  }
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      double k1 = a[j][i] / a[i][i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
        res[j][k] -= k1 * res[i][k];
      }
    }
  }
  for (int i = 0; i < size; i++){
    for (int j = 0; j < size; j++){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      res[i][j] /= a[i][i];
    }
  }
  return res;
}
```

Стандартный метод Гаусса

```
double *gauss(double **a1, double *f1, int size)
{
  double **a = copy_matrix(a1, size);
  double *f = calloc(size, sizeof(*f));
  for (int i = 0; i < size; i++){
    f[i] =f1[i];
  }
  //прямой ход
  int j = 0;
  int sign = 1;
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    for (j = i; j < size; j++){
      if (a[j][i]){
        break;
      }
    }
    if (j == size || fabs(a[j][i]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    if (i != j){
      sign *= change_str(a, i, j);
      double tmp = f[i];
      f[i] = f[j];
      f[j] = tmp;
    }
    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      if (fabs(k2) < EPS){
        continue;
      }
      f[k] -= f[i] * k2 / k1;
      for (int l = 0; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
      }
    }

  }
  //обратный ход
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      if (fabs(a[i][i]) < EPS){
        fprintf(stderr,"det 0\n");
        exit(1);
      }
      double k1 = a[j][i] / a[i][i];
      f[j] -= k1 * f[i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
      }
    }
  }
  for (int j = 0; j < size; j++){
    if (fabs(a[j][j]) < EPS){
      fprintf(stderr,"det 0\n");
      exit(1);
    }
    f[j] /= a[j][j];
  }
  return f;
}
```

Метод Гаусса с выбором ведущего элемента.

```
double *gauss_main(double **a1, double *f1, int size)
{
  double **a = copy_matrix(a1, size);
  double *f = calloc(size, sizeof(*f));
  //хранит перестановки столбцов
  int *vec = calloc(size, sizeof(*vec));

  for (int i = 0; i < size; i++){
    f[i] =f1[i];
  }
  //прямой ход
  int j = 0;
  for (int i = 0; i < size; i++){
    vec[i] = i;
  }
  for (int i = 0; i < size; i++){
    // Поиск первого ненулевого элемента в i столбце начиная с i+1 столбца
    double max = 0;
    int idx = 0;
    for (j = i; j < size; j++){
      if (fabs(a[i][j]) > max){
        max = fabs(a[i][j]);
        idx = j;
      }
    }
    double tmp = vec[i];
    vec[i] = vec[idx];
    vec[idx] = tmp;
    for (int l = 0; l < size; l++){
      double tmp = a[l][i];
      a[l][i] = a[l][idx];
      a[l][idx] = tmp;
    }

    for (int k = i + 1; k < size; k++){
      double k1 = a[i][i];
      double k2 = a[k][i];
      f[k] -= f[i] * k2 / k1;
      for (int l = i; l < size ; l++){
        a[k][l] -= a[i][l] * k2 / k1;
      }
    }

  }
  //обратный ход
  for (int i = size - 1; i > 0; i--){
    for (int j = i - 1; j >= 0; j--){
      double k1 = a[j][i] / a[i][i];
      f[j] -= k1 * f[i];
      for (int k = size - 1; k >=0; k--){
        a[j][k] -= k1 * a[i][k];
      }
    }
  }
  for (int j = 0; j < size; j++){
    f[j] /= a[j][j];
  }
  double *ans = calloc(size, sizeof(*ans));
  for (int i = 0; i < size; i++){
    ans[vec[i]] = f[i];
  }
  return ans;
}
```

Метод верхней релаксации

```
double *relax(double **a, double *f, double w, int size, int max_iter, double
    solution_eps)
{
  //критерий остановки - максимальное числло итераций
  double *x = calloc(size, sizeof(*x));
  double *tmp = calloc(size, sizeof(*tmp));
  for (int k = 0; k < max_iter; k++){
    for (int i = 0; i < size; i++){
      double sub1 = 0, sub2 = 0;
      for (int j = 0; j < i; j++){
        sub1 += a[i][j] * tmp[j];
      }
      for (int j = i; j < size; j++){
        sub2 += a[i][j] * x[j];
      }
      tmp[i] = x[i] + w / a[i][i] * (f[i] - sub1 - sub2);

      //Проверяем 2 критерий остановки алгоритма - расстояние между векторами решениями
      на 2 последовательных шагах становится меньше заданного значения - solution_eps
    }
    double dist = 0;
    for (int i = 0; i < size; i++){
      dist += (x[i] -tmp[i]) * (x[i] -tmp[i]);
    }
    if (sqrt(dist) < solution_eps){
      return tmp;
    }
    for (int i = 0; i < size; i++){
      x[i] = tmp[i];
    }
  }
  return x;
}
```

Число обусловленности

```
double cond_num(double **a, int n)
{
  double res1 = 0;
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      if(fabs(a[i][j]) > res1){
        res1 = fabs(a[i][j]);
      }
    }
  }
  double res2 = 0;
  double **b = inverse(a, n);
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
      if(fabs(b[i][j]) > res2){
        res2 = fabs(b[i][j]);
      }
    }
  }
  return res1 * res2;
}
```

В данном разделе содержится реализация дополнительных функций, использующихся в программе.
В частности, реализованы классы:

- Вывод на стандратный поток матрицы

```
1 void print_matrix(double **a, int size);
```

- Ввод матрицы со стандратного потока

```
1 void scan_matrix(double **a, int size);
```

- Вывод на стандратный поток матрицы-столбца

```
1 void print_vect(double *a, int size);
```

- Ввод матрицы-столбца со стандратного потока

```
1 void scan_vect(double *a, int size);
```

Тестирование

- Часть 1
  Тестирование проводится на наборах СЛАУ из приложения 1 и примера 1 приложения 2.
  Для матриц, имеющих определитель 0, программа выводит сообщение об этом на стандартный поток ошибок и завершается с кодом 1, кроме того метод верхней реалксации применим только для положительно определенных матриц.
  Результат работы на невырожденных матрицах сравниваем с точным ответом, полученном на сайте wolframalpha.com.
  Для генерации теста из приложения 2 использована программа:

```
int main(int argc, char**argv)
{
  if (!strcmp(argv[1], "--frm")){
    double n = 20, m = 8;
    printf("%d\n", (int)n);
    for (int i = 1; i <= n; i++){
      for (int j = 1; j <= n; j++){
        if (i == j){
          printf("%f ", n + m*m + j/m + i/n);
        } else {
          printf("%f ", (i + j) / (m + n));
        }
      }
      printf("\n");
    }
    for (int i = 1; i <= n; i++){
      printf("%f ", 200 + 50 * i);
    }

  }
  printf("\n");
  return 0;
}
```

  Для тестирования запускаем программу с ключом –gauss или –gauss+, в качетсве результата получаем определитель матрицы из левой части, обратную матрицу, число обусловленности и вектор - решение СЛАУ.

  Например, для варианта 3 - 1 создадим файл 1.txt:

2	5	4	1
1	3	2	1
2	10	9	7
3	8	9	2
20	11	40	37

  Запустив программу с ключом - -gauss+ с перенаправлением стандартного ввода-вывода, получим в качестве результата:
  Определитель: -3.000

  Обратная матрица:

15.000	-21.000	2.000	-4.000
-6.667	10.333	-1.000	1.667
-0.333	-0.333	0	0.333
5.667	-8.333	1	1.667

  Число обусловленности: 210.000

  Решение:

15.000
-6.667
-0.333
5.667

  Для варианта 3 - 2 программа выдает сообщение о том, что матрица вырождена.

- Часть 2

Для варианта 3 - 1 получаем те же ответы, что и при использовании метода Гаусса.
В варианте 3 - 2 программа выдает сообщение о том, что матрица вырождена.
В варианте 3 - 3 матрица не является положительно определенной

Исследуем скорость сходимости метода верхней релаксации.

Заметим, что для сходимости метода верхней релаксации необходима симметричность матрицы-левой части. Поэтому для тестирования будем использовать формулы из приложения 2, пример 1 с параметром max-iter = 500, epsilon = 0.000001. Запускаем тесты при различных значениях *test*. Оценим скорость сходимости при различных w:

w	тест 1	тест 2	тест 3	тест 4	тест 5	тест 6
0.2	47	77	76	96	70	75
0.4	23	36	34	44	32	35
0.6	14	21	19	25	19	20
0.8	8	12	12	15	11	12
1.0	5	7	7	7	5	6
1.2	10	14	14	16	12	13
1.4	17	24	24	28	21	23
1.6	29	43	43	50	37	40
1.8	66	97	97	113	84	91

По полученным измерениям можно сказать, что максимальная скорость сходимости достигается при $w = 1$ Дополнительно можно уедится, что для не симметричных матриц метод быстро расходится и все координаты обращаются в бесконечность.

Выводы

- Часть 1
  Можно заметить, что на всех тестах оба метода Гаусса или сходятся одновременно, и причём с почти одинаковой точностью, или расходятся одновременно, из чего можно сделать вывод, что значительной разницы между методами нет.

- Часть 2
  Можно сделать однозначный вывод, что в случае симметрической матрицы метод верхней релаксации будет сходится крайне быстро, особенно если выбрать $w = 1$. В этом случае метод верхней релаксации будет давать решение с высокой точностью уже всего через $10 - 20$ итераций, что будет давать значительную выгоду по сравнению с методами Гаусса. Однако множество применимости этого метода сильно уже, чем у методов Гаусса.