|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <math.h> |
|  |  |
|  | #define SIZE 3 |
|  |  |
|  | float matrA[SIZE][SIZE] = {{1.14, -2.15, -5.11}, |
|  | {-0.71, 0.81, -0.02}, |
|  | {0.42, -1.13, 7.05}}; |
|  |  |
|  | float matrB[SIZE] = {-4.16, -0.17, 6.15}; |
|  |  |
|  | float matrX[SIZE], matrRec[SIZE]; |
|  | float beta, e; |
|  |  |
|  | int CheckState() |
|  | { |
|  | float elementsSum = 0; |
|  | int successfulCount = 0; |
|  |  |
|  | for (int i = 0; i < SIZE; i++) |
|  | { |
|  | for (int j = 0; i < SIZE; j++) |
|  | if (i != j) |
|  | elementsSum += fabs(matrA[i][j]); |
|  |  |
|  | if (fabs(matrA[i][i]) > elementsSum) |
|  | successfulCount++; |
|  | elementsSum = 0; |
|  | } |
|  | if (successfulCount == 3) |
|  | return 1; |
|  | else |
|  | return 0; |
|  | } |
|  |  |
|  | void RowSum(int row1, int row2, float coeff, const int columnsCount) |
|  | { |
|  | for (int i = 0; i < columnsCount; i++) |
|  | matrA[row2][i] += matrA[row1][i] \* coeff; |
|  | matrB[row2] += matrB[row1] \* coeff; |
|  | } |
|  |  |
|  | void IteratiomMethod() |
|  | { |
|  | float elementsSum = 0; |
|  | static int iteration; |
|  |  |
|  | if (iteration != 0) |
|  | for (int i = 0; i < SIZE; i++) |
|  | matrX[i] = matrRec[i]; |
|  | else |
|  | for (int i = 0; i < SIZE; i++) |
|  | matrX[i] = matrB[i]; |
|  |  |
|  | for (int i = 0; i < SIZE; i++) |
|  | { |
|  | for (int j = 0; j < SIZE; j++) |
|  | if (i != j) |
|  | elementsSum += matrA[i][j] \* matrX[j]; |
|  |  |
|  | matrRec[i] = (matrB[i] - elementsSum) / matrA[i][i]; |
|  | elementsSum = 0; |
|  | } |
|  | iteration++; |
|  | } |
|  |  |
|  | void SeidelMethod() |
|  | { |
|  | float elementsSum = 0; |
|  | static int iteration; |
|  |  |
|  | if (iteration != 0) |
|  | for (int i = 0; i < SIZE; i++) |
|  | matrX[i] = matrRec[i]; |
|  | else |
|  | for (int i = 0; i < SIZE; i++) |
|  | matrX[i] = matrB[i]; |
|  |  |
|  | for (int i = 0; i < SIZE; i++) |
|  | { |
|  | for (int j = i + 1; j < SIZE; j++) |
|  | if (i != 1) |
|  | elementsSum += matrA[i][j] \* matrX[j]; |
|  |  |
|  | matrRec[i] = (matrB[i] - elementsSum) / matrA[i][i]; |
|  | elementsSum = 0; |
|  |  |
|  | for (int j = 0; j < i; j++) |
|  | if (i != j) |
|  | elementsSum += matrA[i][j] \* matrRec[j]; |
|  | elementsSum /= -matrA[i][i]; |
|  |  |
|  | matrRec[i] += elementsSum; |
|  | elementsSum = 0; |
|  | } |
|  | iteration++; |
|  | } |
|  |  |
|  | int CheckResult() |
|  | { |
|  | float elementsSum = 0; |
|  | for (int i = 0; i < SIZE; i++) |
|  | elementsSum += fabs(matrRec[i] - matrX[i]); |
|  |  |
|  | beta = elementsSum / SIZE; |
|  |  |
|  | if (beta <= e) |
|  | return 1; |
|  | else |
|  | return 0; |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | int choice; |
|  |  |
|  | printf("Введите необходимую точность вычислений: "); |
|  | scanf("%f", &e); |
|  |  |
|  | RowSum(2, 0, 1, SIZE); |
|  | RowSum(1, 0, 9, SIZE); |
|  | RowSum(2, 1, 0.05, SIZE); |
|  | RowSum(1, 0, -2.7, SIZE); |
|  | RowSum(0, 1, -0.2, SIZE); |
|  |  |
|  | printf("%s\n\n%s\n%s\n", "Выберите метод решения СЛАУ:", "(1) Метод простых итераций", "(2) Метод Зейделя"); |
|  |  |
|  | do |
|  | { |
|  | scanf("%d", &choice); |
|  | } while (choice < 1 && choice > 2); |
|  |  |
|  | switch (choice) |
|  | { |
|  | case 1: |
|  | do |
|  | { |
|  | IteratiomMethod(); |
|  | } while (!CheckResult()); |
|  | break; |
|  | case 2: |
|  | do |
|  | { |
|  | SeidelMethod(); |
|  | } while (!CheckResult()); |
|  | break; |
|  |  |
|  | default: |
|  | break; |
|  | } |
|  | for (int i = 0; i < SIZE; i++) |
|  | printf("x%d = %2.2f\n", i + 1, matrRec[i]); |
|  |  |
|  | return 0; |
|  | } |