**Приложение**

Метод разностной прогонки

Часто при численное решение задач математической физики приводит к решению СЛАУ, которые имеют особую структуру – трехдиагональную. Это привело к появлению одного их точных методов решения таких СЛАУ – методу разностной прогонки.

Пусть у нас имеется система вида:

Суть метода состоит в последовательном исключении переменных (как и в методе Гаусса), однако в более эффективном исключении: переменная исключается не из всех строк, а только из следующей за текущей строкой. Этот этап исключения переменных называется **прямой прогонкой**. Данный этап приводит систему к виду:

Прогоночные коэффициенты вычисляются по следующим формулам:

А затем, на основе прогоночных коэффициентов, вычисляются значения неизвестных по следующим формулам (этот этап называется **обратной прогонкой**):

При программной реализации можно немного оптимизировать алгоритм, если заметить, что для реализации метода не обязательно хранить всю матрицу системы, а достаточно хранить массивы поддиагональных, диагональных и наддиагональных элементов ( соответственно), а также правый столбец системы .

**Листинг**

def tridiagonal\_method(A, B, C, F, N):

alpha = [.0] \* N

beta = [.0] \* N

# этап прямой прогонки (вычисление прогоночных коэффициентов)

alpha[0] = -C[0] / B[0]

beta[0] = F[0] / B[0]

for i in range(1, N):

alpha[i] = -C[i] / (B[i] + A[i] \* alpha[i - 1])

beta[i] = (F[i] - A[i] \* beta[i - 1]) / (B[i] + A[i] \* alpha[i - 1])

y = [.0] \* (N + 1)

# этап обратной прогонки (вычисление неизвестных)

y[N] = (F[N] - A[N] \* beta[N - 1]) / (B[N] + A[N] \* alpha[N - 1])

for i in range(N - 1, -1, -1):

y[i] = alpha[i] \* y[i + 1] + beta[i]

return y