Отчёт о выполнении индивидуального задания "Решение систем линейных уравнений с разреженными матрицами специального вида"

Постановка задачи

Портреты матриц систем во всех вариантах задания различны и определяются номером варианта.

Однако по структуре они похожи – имеют по три диагонали и «испорчены» либо двумя столбцами, либо двумя строками, либо одним столбцом и одной строкой.

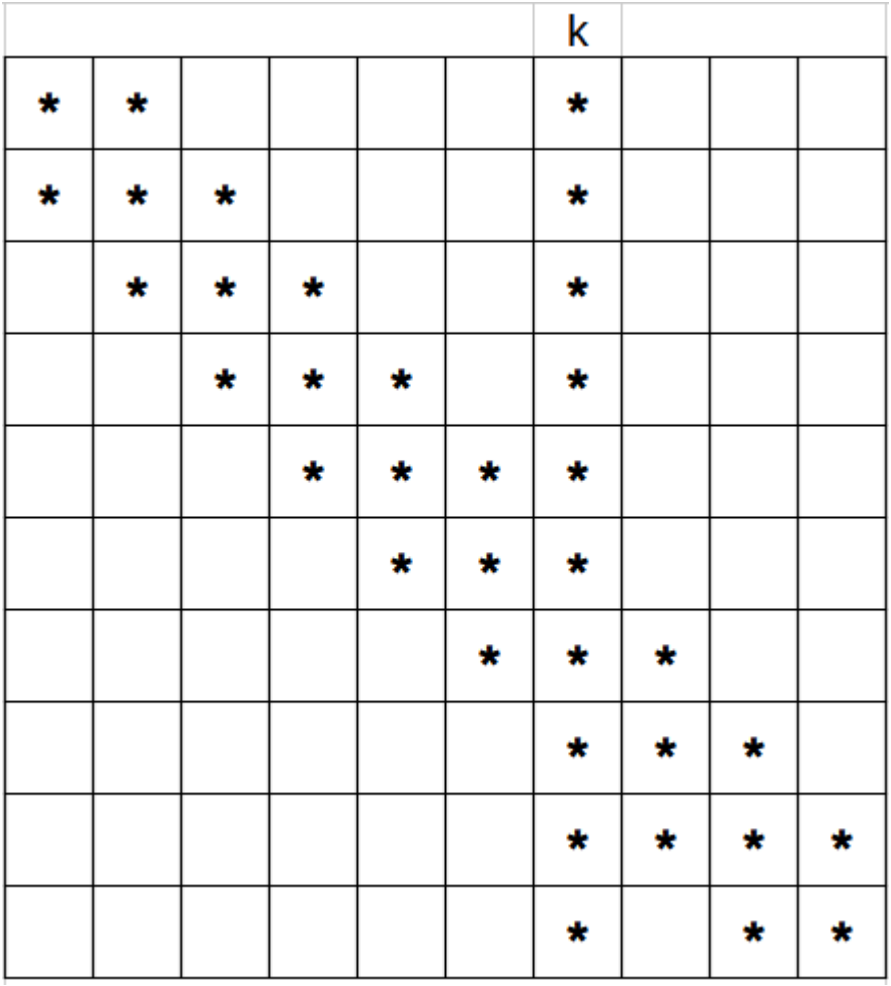
Во всех вариантах исходные системы уравнений задаются шестью векторами: **a, b, c, f, p, q**. Векторы **a, b, c** содержат значения трех диагоналей матрицы, **f** – вектор правой части системы, **p, q** – векторы для строк или столбцов, которые «портят» матрицы системы.

Теоретическая часть

Вариант \ 14

Система (1)

$$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq \left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$$



В данном варианте мы имеем диагональные векторы **a, b, c** и испорченный вектор **p** по индексу `0 < k < size - 1`

Алгоритм

Для описания решения системы (1) воспользуемся символическим изображением алгоритма:

Шаг первый

						k			
1	*					*			
0	1	*				*			
	0	1	*			*			
		0	1	*		*			
			0	1	*	*			
				0	1	*			
					0	*	*		
						*	*	*	
						*	*	*	*
						*		*	*

Шаг второй

						k			
1	*					*			
0	1	*				*			
	0	1	*			*			
		0	1	*		*			
			0	1	*	*			
				0	1	*			
					0	*	0		
						*	1	0	
						*	*	1	0
						*		*	1

Шаг третий

						k			
1	*					0			
0	1	*				0			
	0	1	*			0			
		0	1	*		0			
			0	1	*	0			
				0	1	0			
					0	1	0		
						0	1	0	
						0	*	1	0
						0		*	1

Шаг четвертый, обратный ход

						k			
1	*					0			
0	1	*				0			
	0	1	*			0			
		0	1	*		0			
			0	1	*	0			
				0	1	0			
					0	1	0		
						0	1	0	
						0	0	1	0
						0		0	1

Шаг пятый, обратный ход

						k			
1	0					0			
0	1	0				0			
	0	1	0			0			
		0	1	0		0			
			0	1	0	0			
				0	1	0			
					0	1	0		
						0	1	0	
						0	0	1	0
						0		0	1

После проделанных шагов полученный вектор **f'** и будет решением системы (1)

Основные процедуры

Деление строки на число:

```
    def div_line_by(self, line_idx, divisor):
        if divisor == 0:
            return None

        if line_idx > 0:
            self.a[line_idx - 1] /= divisor

        if line_idx < self.size - 1:
            self.c[line_idx] /= divisor

        self.b[line_idx] /= divisor
        self.p[line_idx] /= divisor
        self.f[line_idx] /= divisor
```

Вычитание строк

```
    def subtract_from(self, from_idx, source_idx):
        if from_idx < source_idx:
            value = self.c[from_idx]

            if source_idx == self.k + 2:
                self.a[from_idx - 1] -= self.p[source_idx] * value

            self.c[from_idx] -= value
            self.b[from_idx] -= self.a[source_idx - 1] * value
            self.p[from_idx] -= self.p[source_idx] * value
            self.f[from_idx] -= self.f[source_idx] * value
        else:
            value = self.a[from_idx - 1]

            self.a[from_idx - 1] -= value

            if source_idx == self.k - 2:
                self.c[from_idx] -= self.p[source_idx] * value

            self.b[from_idx] -= self.c[source_idx] * value
            self.p[from_idx] -= self.p[source_idx] * value
            self.f[from_idx] -= self.f[source_idx] * value
```

Процедура соответствующая первому проходу алгоритма

```
    def gauss_until_k(self):
        for i in range(self.k):
```

```
            self.div_line_by(i, self.b[i])
            self.subtract_from(i + 1, i)
```

Процедура соответствующая второму проходу алгоритма

```
    def gauss_past_k(self):
        for i in range(self.size - 1, self.k, -1):
            self.div_line_by(i, self.b[i])
            self.subtract_from(i - 1, i)
```

Процедура соответствующая третьему проходу алгоритма

```
    def gauss_line_k(self):
        self.div_line_by(self.k, self.b[self.k])
        f_k = self.f[self.k]

        self.a[self.k] = 0
        self.c[self.k - 1] = 0
        for i in range(self.k):
            value = self.p[i]
            self.p[i] -= value
            self.f[i] -= value * f_k

        for i in range(self.size - 1, self.k, -1):
            value = self.p[i]
            self.p[i] -= value
            self.f[i] -= value * f_k
```

Процедура соответствующая четвертому проходу алгоритма

```
    def reverse_gauss_past_k(self):
        for i in range(self.k, self.size - 1):
            self.subtract_from(i + 1, i)
```

Процедура соответствующая пятому проходу алгоритма

```
    def reverse_gauss_until_k(self):
        for i in range(self.k - 1, 0, -1):
            self.subtract_from(i - 1, i)
```

Основные процедуры решения

```

```

```
    def input_matrix_vec(self, a, b, c, p, f, k):
        self.vec_form = True
        self.k = k
        self.size = b.shape[0]
        self.shape = (self.size, self.size)

        self.a = np.copy(a)
        self.b = np.copy(b)
        self.c = np.copy(c)
        self.p = np.copy(p)
        self.set_f(f)

    def solve(self):
        if not self.vec_form:
            self.build_vectors()

        self.gauss_until_k()

        self.gauss_past_k()

        self.gauss_line_k()

        self.reverse_gauss_until_k()

        self.reverse_gauss_past_k()
        return self.f
```

Результаты тестирования

Номер теста	Размерность системы	Диапазон значений	Средняя точность	Средняя погрешность
1	10	(-10, 10)	1.17E-13	5.69E-13
2	10	(-100, 100)	4.20E-14	1.27E-12
3	10	(-1000, 1000)	5.18E-11	4.11E-11
4	100	(-10, 10)	4.02E-12	8.61E-12
5	100	(-100, 100)	6.78E-12	8.95E-12
6	100	(-1000, 1000)	1.36E-13	3,53E-12
7	1000	(-10, 10)	3.61E-11	1.27E-10
8	1000	(-100, 100)	1.17E-12	5.46E-11
9	1000	(-1000, 1000)	3.83E-11	2.46E-10