

Решение систем линейных алгебраических уравнений

Губкин А.С.

Тюменский филиал Института теоретической и прикладной механики
им. С. А. Христиановича СО РАН, г. Тюмень

23 октября 2020 г.



Задачи, приводящие к системам линейных алгебраических уравнений (СЛАУ):

Задачи, приводящие к системам линейных алгебраических уравнений (СЛАУ):

- ▶ статический анализ конструкций (мост, здание, несущая конструкция самолета);

Задачи, приводящие к системам линейных алгебраических уравнений (СЛАУ):

- ▶ статический анализ конструкций (мост, здание, несущая конструкция самолета);
- ▶ проектирование электрических цепей;

Задачи, приводящие к системам линейных алгебраических уравнений (СЛАУ):

- ▶ статический анализ конструкций (мост, здание, несущая конструкция самолета);
- ▶ проектирование электрических цепей;
- ▶ численное решение дифференциальных уравнений.

Система линейных алгебраических уравнений

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \quad (1)$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n = b_1 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots a_{nn}x_n = b_n \end{cases} \quad (2)$$

Существование и единственность решения этой системы гарантируется при $\det \mathbf{A} \neq 0$.

Задачи вычислительной линейной алгебры

Задачи вычислительной линейной алгебры

- ▶ Решение систем линейных алгебраических уравнений:

$$A \cdot x = b.$$

Задачи вычислительной линейной алгебры

- ▶ Решение систем линейных алгебраических уравнений:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}.$$

- ▶ Вычисление определителей и обратных матриц:

$$\det \mathbf{A}, \mathbf{B} = \mathbf{A}^{-1}.$$

Задачи вычислительной линейной алгебры

- ▶ Решение систем линейных алгебраических уравнений:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}.$$

- ▶ Вычисление определителей и обратных матриц:

$$\det \mathbf{A}, \mathbf{B} = \mathbf{A}^{-1}.$$

- ▶ Вычисление собственных чисел и векторов:

$$\mathbf{A} \cdot \mathbf{z}_n = \lambda_n \mathbf{z}_n.$$

Прямые и итерационные методы решения СЛАУ

Методы решения СЛАУ можно разделить на два класса:

Прямые и итерационные методы решения СЛАУ

Методы решения СЛАУ можно разделить на два класса:

- ▶ **Прямые методы.** Данные методы позволяют получить точное решение задачи (без учета ошибок округления) за конечное число арифметических действий.

Прямые и итерационные методы решения СЛАУ

Методы решения СЛАУ можно разделить на два класса:

- ▶ **Прямые методы.** Данные методы позволяют получить точное решение задачи (без учета ошибок округления) за конечное число арифметических действий.
- ▶ **Итерационные методы** или **методы последовательных приближений.** Позволяют вычислять последовательность векторов x_k , которая при $k \rightarrow \infty$ сходится к решению задачи. На практике используют некоторое конечное приближение в зависимости от допустимого уровня погрешности.

Метод Крамера

Известен явный способ получения решения СЛАУ - это метод Крамера:

$$x_i = \frac{\Delta_i}{\Delta},$$

где Δ – определитель матрицы \mathbf{A} , а Δ_i – определитель матрицы, полученной из \mathbf{A} заменой i -го столбца на вектор \mathbf{b} . Однако, вычислять определитель по формуле:

$$\Delta = \sum_{i_1, i_2, \dots, i_n} (-1)^{P(i_1, i_2, \dots, i_n)} a_{1i_1} a_{2i_2} \dots a_{ni_n}$$

оказывается весьма затратно. Сложность вычисления методом Крамера составляет $\mathcal{O}(n^2 \cdot n!)$. Практически этот метод применим лишь при небольших размерностях системы $n \lesssim 10$.

Итерационные методы

Итерационные методы обычно основываются на следующей эквивалентной форме системы (1):

$$\mathbf{x} = \mathbf{B} \cdot \mathbf{x} + \mathbf{g}, \quad (3)$$

где \mathbf{B} называется итерационной матрицей. Итерационный процесс формулируется на основе эквивалентной формы (3) в следующем виде:

- ▶ зададим начальное приближение (вектор) \mathbf{x}_0 ;
- ▶ для каждого значения k будем вычислять последующие приближения как

$$\mathbf{x}_{k+1} = \mathbf{B} \cdot \mathbf{x}_k + \mathbf{g};$$

- ▶ если $\frac{\|\mathbf{x}_{k+1} - \mathbf{x}_k\|}{\|\mathbf{x}_{k+1}\|} \leq \varepsilon$ или $\frac{\|\mathbf{r}_{k+1}\|}{\|\mathbf{x}_{k+1}\|} \leq \varepsilon$, тогда итерационный процесс завершен и мы полагаем, что $\mathbf{x}_* = \mathbf{x}_{k+1}$.

Классические итерационные методы и релаксация

Определение

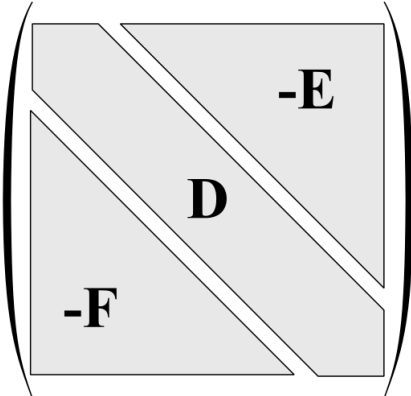
Невязкой системы (1), соответствующей вектору x , будем называть вектор $r = b - A \cdot x$.

Исторически первые итерационные методы основывались на циклическом покомпонентном изменении вектора решения, осуществляемом таким образом, чтобы обнулить соответствующий коэффициент вектора невязки и тем самым уменьшить его норму. Подобная методика уточнения решения получила название релаксации.

Методы Якоби и Гаусса-Зейделя

Представим матрицу \mathbf{A} системы (1) в виде разности трех матриц:

$$\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F}.$$

$$\mathbf{A} = \begin{pmatrix} & & \\ & \mathbf{D} & \\ & & \end{pmatrix} - \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} - \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$


Методы Якоби и Гаусса-Зейделя

Если имеется некоторое приближение x_k к точному решению СЛАУ x_* , то при $x_k = x_*$ это соотношение не выполняется. Однако, если в выражении

$$A \cdot x_k = D \cdot x_k - E \cdot x_k - F \cdot x_k = b \quad (4)$$

одно или два из вхождений вектора x_k заменить на x_{k+1} и потребовать, чтобы равенство имело место, можно получить некоторую вычислительную схему для уточнения решения.

Метод Якоби (векторная форма)

Наиболее простой с точки зрения объема вычислительной работы вариант получается при замене в (4) $\mathbf{D} \cdot \mathbf{x}_k$ на $\mathbf{D} \cdot \mathbf{x}_{k+1}$. При этом получается схема:

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1} \cdot (\mathbf{E} + \mathbf{F}) \cdot \mathbf{x}_k + \mathbf{D}^{-1} \cdot \mathbf{b}, \quad (5)$$

известная как метод Якоби.

Метод Якоби (скалярная форма)

Выражение (5) в скалярной форме имеет вид:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right), \quad i = 1, n. \quad (6)$$

Метод Якоби (исходный код)

```
1 do{
2   for(i = 0 ; i < N ; i++)
3   {
4     sum = 0.0;
5     for(j = 0 ; j < N ; j++)
6       sum = sum + matrix[i][j] * x_prev[j];
7     x[i] = rhs[i] + sum;
8   }
9   sum = 0;
10  sum_err = 0;
11  for(i = 0 ; i < N ; i++)
12  {
13    sum = sum + x[i] * x[i];
14    sum_err = sum_err + (x[i] - x_prev[i]) * (x[i] - x_prev[i]);
15    x_prev[i] = x[i];
16  }
17  solution_log << "iteration #" << ++k << "\t eps = " << sqrt(sum_err / sum) <<
    endl;
18 }while(sqrt(sum_err / sum) > eps);
```

Метод Гаусса-Зейделя (скалярная форма)

Очевидным недостатком схемы (5) – (6) является то, что при нахождении $x_i^{(k+1)}$ никак не используется информация о уже пересчитанных компонентах $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$. Исправить этот недостаток можно, переписав (6) в виде:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right), i = 1, n.$$

Метод Гаусса-Зейделя (исходный код)

```
1 do{
2   for(i = 0 ; i < N ; i++)
3   {
4     sum = 0;
5     x_prev[i] = x[i];
6     for(j = 0 ; j < N ; j++)
7       sum = sum + matrix[i][j]*x[j];
8     x[i] = rhs[i] + sum;
9   }
10  sum = 0;
11  sum_err = 0;
12  for(i = 0 ; i < N ; i++)
13  {
14    sum = sum + x[i] * x[i];
15    sum_err = sum_err + (x[i] - x_prev[i]) * (x[i] - x_prev[i]);
16  }
17  solution_log << "iteration #" << ++k << "\t eps = " << sqrt(sum_err / sum) <<
    endl;
18 }while(sqrt(sum_err / sum) > eps);
```

Метод Гаусса - Зейделя (векторная форма)

Векторную форму метода Гаусса-Зейделя можно получить из (4) заменой \mathbf{x}_k на \mathbf{x}_{k+1} при матрицах \mathbf{D} и \mathbf{E} , т.е.:

$$\mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{E})^{-1} \cdot \mathbf{F} \cdot \mathbf{x}_k + (\mathbf{D} - \mathbf{E})^{-1} \cdot \mathbf{b}.$$

Если вместо этой пары матриц взять \mathbf{D} и \mathbf{F} , то получится похожая схема:

$$\mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{F})^{-1} \cdot \mathbf{E} \cdot \mathbf{x}_k + (\mathbf{D} - \mathbf{F})^{-1} \cdot \mathbf{b},$$

которая называется **обратным методом Гаусса - Зейделя**

Обратный метод Гаусса - Зейделя (исходный код)

```
1 do{
2   for(i = N - 1 ; i >= 0 ; i--)
3   {
4     sum = 0;
5     x_prev[i] = x[i];
6     for(j = 0 ; j < N ; j++)
7       sum = sum + matrix[i][j]*x[j];
8     x[i] = rhs[i] + sum;
9   }
10  sum = 0;
11  sum_err = 0;
12  for(i = 0 ; i < N ; i++)
13  {
14    sum = sum + x[i] * x[i];
15    sum_err = sum_err + (x[i] - x_prev[i]) * (x[i] - x_prev[i]);
16  }
17  solution_log << "iteration #" << ++k << "\t eps = " << sqrt(sum_err / sum) <<
    endl;
18 }while(sqrt(sum_err / sum) > eps);
```

Симметричный метод Гаусса-Зейделя

Еще одной модификацией является симметричный метод Гаусса-Зейделя, который заключается в циклическом чередовании **прямого** и **обратного** метода Гаусса - Зейделя на соседних итерациях:

$$\begin{cases} \mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{E})^{-1} \cdot \mathbf{F} \cdot \mathbf{x}_k + (\mathbf{D} - \mathbf{E})^{-1} \cdot \mathbf{b} \\ \mathbf{x}_{k+1} = (\mathbf{D} - \mathbf{F})^{-1} \cdot \mathbf{E} \cdot \mathbf{x}_k + (\mathbf{D} - \mathbf{F})^{-1} \cdot \mathbf{b} \end{cases}$$

Расщепление

Все предыдущие методы можно записать в виде:

$$\mathbf{K} \cdot \mathbf{x}_{k+1} = \mathbf{R} \cdot \mathbf{x}_k + \mathbf{b},$$

где матрицы \mathbf{K} и \mathbf{R} связаны соотношением:

$$\mathbf{A} = \mathbf{K} - \mathbf{R}.$$

Подобное представление матрицы \mathbf{A} называется **расщеплением**, а методы такого вида – **методами, основанными на расщеплении**. Очевидно, матрица \mathbf{K} должна быть невырожденной и легко обратимой.

Нормы векторов и матриц

Для исследования сходимости численных методов решения задач линейной алгебры вводятся понятия нормы векторов и матриц.

Определение

Нормой вектора \mathbf{x} (обозначают $\|\mathbf{x}\|$) называют неотрицательное число, вычисляемое с помощью компонент вектора и обладающее следующими свойствами:

- ▶ $\|\mathbf{x}\| \geq 0$, $\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$.
- ▶ $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$, $\forall \alpha$.
- ▶ $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

Определение

Нормой матрицы $\mathbf{A}_{n \times n}$ (обозначается $\|\mathbf{A}\|$) с вещественными элементами называют неотрицательное число, вычисляемое с помощью элементов матрицы и обладающее следующими свойствами:

- ▶ $\|\mathbf{A}\| \geq 0$, $\|\mathbf{A}\| = 0 \Leftrightarrow \mathbf{A} = \mathbf{0}$.
- ▶ $\|\alpha \mathbf{A}\| = |\alpha| \|\mathbf{A}\|$, $\forall \alpha$.
- ▶ $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$.
- ▶ $\|\mathbf{A} \cdot \mathbf{B}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$.

Нормы векторов и матриц

Норма матриц должна быть согласована с нормой векторов. Это согласование осуществляется связью:

$$\|\mathbf{A} \cdot \mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\|.$$

Наиболее употребительными являются следующие нормы векторов:

$$\|\mathbf{x}\|_1 = \max_i |x_i|,$$

$$\|\mathbf{x}\|_2 = \sum_{i=1}^n |x_i|,$$

$$\|\mathbf{x}\|_3 = \sqrt{\sum_{i=1}^n x_i^2} = (\mathbf{x}, \mathbf{x}).$$

Нормы векторов и матриц

Нормы матриц, согласованные с нормами векторов, будут соответственно:

$$\|\mathbf{A}\|_1 = \max_i \sum_{j=1}^n |a_{ij}|,$$

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^n |a_{ij}|,$$

$$\|\mathbf{A}\|_3 = \sqrt{\max_i |\lambda_i| \mathbf{A}^T \cdot \mathbf{A}}.$$

Под знаком квадратного корня в норме матрицы $\|\mathbf{A}\|_3$ находится спектральный радиус симметрической матрицы $\mathbf{A}^T \cdot \mathbf{A}$, для которой все собственные значения являются действительными.

Нормы векторов и матриц

Из линейной алгебры известно, что собственные значения матриц не превышают их норм. Действительно, из равенства $\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x}$ и свойств норм векторов и матриц следует: $\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\| = \|\mathbf{A} \cdot \mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \Rightarrow |\lambda| \leq \|\mathbf{A}\|$, или $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$, где $\rho(\mathbf{A}) = \max_i |\lambda_i|$ – максимальное по модулю собственное значение или спектральный радиус матрицы \mathbf{A} . Таким образом, за норму матрицы можно принять ее спектральный радиус.

Теорема о сходимости итерационного метода

Условия сходимости изложенных методов устанавливает следующая теорема:

Теорема

Вычислительная схема сходится при любом начальном приближении x_0 тогда и только тогда, когда матрицы K и R удовлетворяют условию

$$\max_j |\lambda_j(K^{-1} \cdot R)| < 1.$$

Теорема о сходимости итерационного метода

Доказательство

Пусть известно точное решение СЛАУ \mathbf{x}_* , тогда:

$$\begin{cases} \mathbf{K} \cdot \mathbf{x}_{k+1} = \mathbf{R} \cdot \mathbf{x}_k + \mathbf{b} \\ \mathbf{K} \cdot \mathbf{x}_* = \mathbf{R} \cdot \mathbf{x}_* + \mathbf{b} \end{cases} \Rightarrow \mathbf{K} \cdot (\mathbf{x}_* - \mathbf{x}_{k+1}) = \mathbf{R} \cdot (\mathbf{x}_* - \mathbf{x}_k) \Rightarrow$$

$$\mathbf{K} \cdot \varepsilon_{k+1} = \mathbf{R} \cdot \varepsilon_k \Rightarrow \varepsilon_{k+1} = \mathbf{K}^{-1} \cdot \mathbf{R} \cdot \varepsilon_k = (\mathbf{K}^{-1} \cdot \mathbf{R})^{k+1} \cdot \varepsilon_0 \Rightarrow$$

$$\|\varepsilon_{k+1}\| = \|(\mathbf{K}^{-1} \cdot \mathbf{R})^{k+1}\| \cdot \|\varepsilon_0\| = \left(\sqrt{\max_i |\lambda_i(\mathbf{K}^{-1} \cdot \mathbf{R})|} \right)^{k+1} \|\varepsilon_0\|$$

Следовательно, для сходимости необходимо и достаточно выполнения условия $\|\mathbf{K}^{-1} \cdot \mathbf{R}\|^k \rightarrow 0$ при $k \rightarrow \infty$.

Ускорение сходимости релаксационных методов. Методы SOR и SSOR

Следствие из теоремы

Чем меньше величина $\max_j |\lambda_j(\mathbf{K}^{-1} \cdot \mathbf{R})|$, тем быстрее сходимость метода.

Одним из распространенных способов улучшения сходимости является введение параметра.

$$\omega \mathbf{A} \cdot \mathbf{x} = \omega \mathbf{b}.$$

Представим матрицу $\omega \mathbf{A}$ в виде:

$$\omega \mathbf{A} = (\mathbf{D} - \omega \mathbf{E}) - (\omega \mathbf{F} + (1 - \omega) \mathbf{D}),$$

тогда можно построить итерационную схему, похожую на метод Гаусса-Зейделя,

$$(\mathbf{D} - \omega \mathbf{E}) \cdot \mathbf{x}_{k+1} = (\omega \mathbf{F} + (1 - \omega) \mathbf{D}) \cdot \mathbf{x}_k + \omega \mathbf{b}.$$

Такая схема называется методом последовательной верхней релаксации (**Successive Over Relaxation** или сокращенно **SOR**). Для нее

$$\mathbf{K}_{\text{SOR}}(\omega) = \mathbf{D} - \omega \mathbf{E},$$

$$\mathbf{R}_{\text{SOR}}(\omega) = \omega \mathbf{F} + (1 - \omega) \mathbf{D}.$$

На практике было замечено, что такая итерационная схема сходится медленнее при $\omega < 1$ (нижняя релаксация), чем при $\omega \geq 1$ (верхняя релаксация).

Метод SOR (скалярная форма)

В методе **SOR** решение удовлетворяет соотношению:

$$x_i^{(k+1)} = \omega x_i^{GS} + (1 - \omega)x_i^{(k)}, \quad i = 1, n,$$

где x_i^{GS} итерация метода Гаусса-Зейделя.

Метод SOR (исходный код)

```
1 do{
2   for(i = 0 ; i < N ; i++)
3   {
4     sum = 0;
5     for(j = 0 ; j < N ; j++)
6       sum = sum + matrix[i][j] * x[j];
7     x[i] = omega * (rhs[i] + sum) + (1 - omega) * x_prev[i];
8   }
9   sum = 0;
10  sum_err = 0;
11  for(i = 0 ; i < N ; i++)
12  {
13    sum = sum + x[i] * x[i];
14    sum_err = sum_err + (x[i] - x_prev[i]) * (x[i] - x_prev[i]);
15    x_prev[i] = x[i];
16  }
17  solution_log << "iteration #" << ++k << "\t eps = " << sqrt(sum_err / sum) <<
    endl;
18 }while(sqrt(sum_err / sum) > eps);
```

Обратный метод SOR

Если в выражении

$$\omega \mathbf{A} = (\mathbf{D} - \omega \mathbf{E}) - (\omega \mathbf{F} + (1 - \omega) \mathbf{D})$$

поменять местами матрицы \mathbf{E} и \mathbf{F} , то такая перестановка дает **обратный метод последовательной верхней релаксации**:

$$(\mathbf{D} - \omega \mathbf{F}) \cdot \mathbf{x}_{k+1} = (\omega \mathbf{E} + (1 - \omega) \mathbf{D}) \cdot \mathbf{x}_k + \omega \mathbf{b}.$$

Метод RSOR (исходный код)

```
1 do{
2   for(i = N - 1 ; i >= 0 ; i--)
3   {
4     sum = 0;
5     for(j = 0 ; j < N ; j++)
6       sum = sum + matrix[i][j] * x[j];
7     x[i] = omega * (rhs[i] + sum) + (1 - omega) * x_prev[i];
8   }
9   sum = 0;
10  sum_err = 0;
11  for(i = 0 ; i < N ; i++)
12  {
13    sum = sum + x[i] * x[i];
14    sum_err = sum_err + (x[i] - x_prev[i]) * (x[i] - x_prev[i]);
15    x_prev[i] = x[i];
16  }
17  solution_log << "iteration #" << ++k << "\t eps = " << sqrt(sum_err / sum) <<
    endl;
18 }while(sqrt(sum_err / sum) > eps);
```


Метод SSOR

Последовательное применение прямого и обратного методов SOR дает симметричный метод последовательной верхней релаксации (**Symmetric Successive Over Relaxation** или сокращенно **SSOR**):

$$\begin{cases} (\mathbf{D} - \omega \mathbf{E}) \cdot \mathbf{x}_{k+1} = (\omega \mathbf{F} + (1 - \omega) \mathbf{D}) \cdot \mathbf{x}_k + \omega \mathbf{b} \\ (\mathbf{D} - \omega \mathbf{F}) \cdot \mathbf{x}_{k+1} = (\omega \mathbf{E} + (1 - \omega) \mathbf{D}) \cdot \mathbf{x}_k + \omega \mathbf{b} \end{cases}$$

Пример

Рассмотрим предыдущие итерационные методы на примере СЛАУ с матрицей:

$$\mathbf{A} = \begin{pmatrix} n & 1 & \dots & 1 \\ 1 & n & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & n \end{pmatrix}, \quad (7)$$

и вектором свободных членов:

$$\mathbf{f} = \begin{pmatrix} 2n - 1 \\ 2n - 1 \\ \vdots \\ 2n - 1 \end{pmatrix}. \quad (8)$$

Пример

Решение СЛАУ (1) с матрицей (7) и с вектором свободных членов (8) известно и равно:

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}. \quad (9)$$

Положим $n = 100, \omega = 0.9, \varepsilon = 1e - 10$.

Пример

Метод Якоби

iteration #1 $\text{eps} = 2.26422$

iteration #2 $\text{eps} = 1.80543$

iteration #3 $\text{eps} = 2.27005$

...

iteration #2580 $\text{eps} = 9.91072\text{e-}11$

Метод Гаусса-Зейделя

iteration #1 $\text{eps} = 3.3409$

iteration #2 $\text{eps} = 3.57371$

iteration #3 $\text{eps} = 0.672445$

...

iteration #18 $\text{eps} = 5.85572\text{e-}11$

Пример

Метод SOR

iteration #1 $\text{eps} = 3.87681$

iteration #2 $\text{eps} = 3.45209$

iteration #3 $\text{eps} = 0.280227$

...

iteration #16 $\text{eps} = 5.07988\text{e-}11$

Метод RSOR

iteration #1 $\text{eps} = 3.87681$

iteration #2 $\text{eps} = 3.45209$

iteration #3 $\text{eps} = 0.280227$

...

iteration #16 $\text{eps} = 5.07989\text{e-}11$

Предобусловливание (левое)

Пусть \mathbf{M} – некоторая невырожденная матрица размерности n , тогда система

$$\mathbf{M}^{-1}\mathbf{A} \cdot \mathbf{x} = \mathbf{M}^{-1} \cdot \mathbf{b},$$

имеет тоже самое точное решение \mathbf{x}_* , но матрица $\mathbf{M}^{-1}\mathbf{A}$ будет иметь иные спектральные характеристики, что ведет к изменению скорости сходимости.

Такой переход с целью улучшения характеристик матрицы называют **предобусловливанием** (переобусловливанием). \mathbf{M} – матрица предобусловливателя.

Требования

- ▶ M должна быть близка к матрице A ;
- ▶ M должна быть легко вычисляемой;
- ▶ M должна быть легко обратимой.

Такое предобусловливание называется **левым** поскольку мы умножаем матрицу СЛАУ на матрицу предобусловливателя слева.

Предобусловливание (правое)

Другой возможный подход основан на переходе от исходной системы к системе

$$AM^{-1} \cdot y = b,$$

у которой точное решение y_* связано с точным решением x_* исходной СЛАУ соотношением

$$x_* = M^{-1} \cdot y_*,$$

Такое предобусловливание называется **правым** поскольку мы умножаем матрицу СЛАУ на матрицу предобусловливателя справа.

Связь с предобуславливанием

Методы релаксации могут быть представлены в виде:

$$\mathbf{K} \cdot \mathbf{x}_{k+1} = \mathbf{R} \cdot \mathbf{x}_k + \mathbf{b} \text{ или } \mathbf{x}_{k+1} = \mathbf{G} \cdot \mathbf{x}_k + \mathbf{f}$$

это соотношение можно рассматривать как итерационную схему для СЛАУ

$$(\mathbf{I} - \mathbf{G}) \cdot \mathbf{x} = \mathbf{f},$$

в которой

$$\mathbf{f} = \mathbf{M}^{-1} \cdot \mathbf{b};$$

$$\mathbf{G} = \mathbf{K}^{-1}\mathbf{R} = \mathbf{K}^{-1}(\mathbf{K} - \mathbf{A}) = \mathbf{I} - \mathbf{K}^{-1}\mathbf{A}.$$

Связь методов релаксации с предобуславливанием

Таким образом, данная система эквивалентна системе

$$\mathbf{K}^{-1} \mathbf{A} \cdot \mathbf{x} = \mathbf{K}^{-1} \cdot \mathbf{b},$$

т.е. предобусловленной СЛАУ с матрицей предобуславливания \mathbf{K} .

Матрицы предобусловливателя

Для уже изученных методов

$$\mathbf{K}_J = \mathbf{D};$$

$$\mathbf{K}_{GS} = \mathbf{D} - \mathbf{E};$$

$$\mathbf{K}_{SOR} = \frac{1}{\omega} (\mathbf{D} - \omega \mathbf{E});$$

$$\mathbf{K}_{SSOR} = \frac{1}{\omega(2 - \omega)} (\mathbf{D} - \omega \mathbf{E}) \mathbf{D}^{-1} (\mathbf{D} - \omega \mathbf{F}).$$

Проекционные методы. Подпространства Крылова

Определение

Проекционные методы решения СЛАУ – класс итерационных методов, в которых решается задача проектирования неизвестного вектора на некоторое пространство оптимально относительно другого некоторого пространства.

Условие Петрова-Галёркина

Рассмотрим СЛАУ $\mathbf{A} \cdot \mathbf{x} = \mathbf{f}$, где \mathbf{A} – квадратная матрица размерности \mathbf{n} . Пусть \mathcal{K} и \mathcal{L} – два m -мерных подпространства пространства \mathcal{R}^n . Необходимо найти такой вектор $\mathbf{x} \in \mathcal{K}$, чтобы $\mathbf{r}_x = \mathbf{f} - \mathbf{A} \cdot \mathbf{x} \perp \mathcal{L}$, т.е. выполнялось условие:

$$\forall \mathbf{l} \in \mathcal{L} : (\mathbf{A} \cdot \mathbf{x}, \mathbf{l}) = (\mathbf{f}, \mathbf{l}), \quad (10)$$

или

$$\forall \mathbf{l} \in \mathcal{L} : (\mathbf{r}_x, \mathbf{l}) = 0, \quad (11)$$

называемое **условием Петрова-Галёркина**.

Такая задача называется задачей проектирования решения \mathbf{x} на подпространство \mathcal{K} ортогонально к подпространству \mathcal{L} .



Условие Петрова-Галёркина

Пусть для исходной системы (1) известно некоторое приближение \mathbf{x}_0 к решению \mathbf{x}_* . Требуется уточнить его поправкой $\delta\mathbf{x} \in \mathcal{K}$ таким образом, чтобы $\mathbf{f} - \mathbf{A} \cdot (\mathbf{x}_0 + \delta\mathbf{x}) \perp \mathcal{L}$. Условие Петрова - Галеркина в этом случае можно записать в виде

$$\forall \mathbf{l} \in \mathcal{L} : (\mathbf{r}_{\mathbf{x}+\delta\mathbf{x}}, \mathbf{l}) = (\mathbf{f} - \mathbf{A} \cdot \mathbf{x} - \mathbf{A} \cdot \delta\mathbf{x}, \mathbf{l}) = (\mathbf{r}_0 - \mathbf{A} \cdot \delta\mathbf{x}, \mathbf{l}) = 0. \quad (12)$$

Общая постановка задачи

Пусть $\dim \mathcal{K} = \dim \mathcal{L} = m$. Введем в подпространствах \mathcal{K} и \mathcal{L} базисы $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$ и $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]$ соответственно. Нетрудно видеть, что (12) выполняется тогда и только тогда, когда:

$$\forall j (1 \leq j \leq m) : (\mathbf{r}_0 - \mathbf{A} \cdot \delta \mathbf{x}, \mathbf{w}_j) = 0. \quad (13)$$

можно записать $\mathbf{x}_* - \mathbf{x}_0 = \delta \mathbf{x} = \mathbf{V} \cdot \mathbf{y}$ где $\mathbf{y} \in \mathcal{R}^m$ – вектор коэффициентов. Тогда (13) может быть записано в виде:

$$\mathbf{W}^T \cdot (\mathbf{r}_0 - \mathbf{A} \cdot \mathbf{V} \cdot \mathbf{y}) = 0,$$

откуда

$$\mathbf{y} = (\mathbf{W}^T \cdot \mathbf{A} \cdot \mathbf{V})^{-1} \cdot \mathbf{W}^T \cdot \mathbf{r}_0.$$

Общая постановка задачи

Таким образом, решение должно уточняться в соответствии с формулой:

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{V} \cdot \mathbf{y} = \mathbf{V} \cdot (\mathbf{W}^T \cdot \mathbf{A} \cdot \mathbf{V})^{-1} \cdot \mathbf{W}^T \cdot \mathbf{r}_0.$$

из которой сразу вытекает важное требование: в практических реализациях проекционных методов подпространства \mathcal{K} и \mathcal{L} и их базисы должны выбираться так, чтобы матрица $(\mathbf{W}^T \cdot \mathbf{A} \cdot \mathbf{V})$ либо была малой размерности, либо имела простую структуру, удобную для обращения.

Алгоритм проекционного метода

Алгоритм

В общем виде алгоритм любого метода проекционного класса может быть записан следующим образом:

- ▶ Выбираем пару подпространств \mathcal{K} и \mathcal{L} .
- ▶ Построение для \mathcal{K} и \mathcal{L} базисов $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m]$ и $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]$.
- ▶ $\mathbf{r}_0 = \mathbf{f} - \mathbf{A} \cdot \mathbf{x}_0$.
- ▶ $\mathbf{y} = (\mathbf{W}^T \cdot \mathbf{A} \cdot \mathbf{V})^{-1} \cdot \mathbf{W}^T \cdot \mathbf{r}_0$.
- ▶ $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{V} \cdot \mathbf{y}$.

Случай одномерных подпространств \mathcal{K} и \mathcal{L}

Наиболее простой ситуацией является случай, когда пространства \mathcal{K} и \mathcal{L} одномерны. В этом случае их матричные базисы являются векторами: $\mathbf{V} = [\mathbf{v}]$ и $\mathbf{W} = [\mathbf{w}]$ и выражение $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{V} \cdot \mathbf{y}$, можно переписать как:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \gamma_k \mathbf{v}_k, \quad (14)$$

где γ_k – неизвестный коэффициент, который легко находится из условия ортогональности $\mathbf{r}_k - \mathbf{A}(\gamma_k \mathbf{v}_k) \perp \mathbf{w}_k$:

$$(\mathbf{r}_k - \gamma_k \mathbf{A} \cdot \mathbf{v}_k, \mathbf{w}_k) = (\mathbf{r}_k, \mathbf{w}_k) - \gamma_k (\mathbf{A} \cdot \mathbf{v}_k, \mathbf{w}_k) = 0,$$

откуда

$$\gamma_k = \frac{(\mathbf{r}_k, \mathbf{w}_k)}{(\mathbf{A} \cdot \mathbf{v}_k, \mathbf{w}_k)}.$$

Метод наискорейшего спуска

Выберем $\mathbf{v}_k = \mathbf{w}_k = \mathbf{r}_k$. Тогда (14) примет вид:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{(\mathbf{r}_k, \mathbf{r}_k)}{(\mathbf{A} \cdot \mathbf{r}_k, \mathbf{r}_k)} \mathbf{r}_k.$$

Данный метод есть **метод наискорейшего спуска** (**Steepest Descent Method** или сокращенно **SDM**).

Метод наискорейшего спуска

Выберем $\mathbf{v}_k = \mathbf{w}_k = \mathbf{r}_k$. Тогда (14) примет вид:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{(\mathbf{r}_k, \mathbf{r}_k)}{(\mathbf{A} \cdot \mathbf{r}_k, \mathbf{r}_k)} \mathbf{r}_k.$$

Данный метод есть **метод наискорейшего спуска** (**Steepest Descent Method** или сокращенно **SDM**).

Замечание

В практических задачах метод наискорейшего спуска обладает достаточно медленной сходимостью!

Метод наискорейшего уменьшения невязки

Выберем $\mathbf{v}_k = \mathbf{A}^T \cdot \mathbf{r}_k$ и $\mathbf{w}_k = \mathbf{A} \cdot \mathbf{v}_k$. Тогда (14) примет вид:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{(\mathbf{r}_k, \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{r}_k)}{(\mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{r}_k, \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{r}_k)} \mathbf{A}^T \cdot \mathbf{r}_k.$$

Данный метод есть **метод наискорейшего уменьшения невязки** (**Residual norm Steepest Descent** или сокращенно **RnSD**).

Подпространства Крылова

При построении и реализации проекционных методов важную роль играют так называемые подпространства Крылова, часто выбираемые в качестве \mathcal{K} .

Определение

Подпространством Крылова размерности m , порожденным вектором \mathbf{v} и матрицей \mathbf{A} называется линейное пространство

$$\mathcal{K}_m(\mathbf{v}, \mathbf{A}) = \text{span} \{ \mathbf{v}, \mathbf{A} \cdot \mathbf{v}, \mathbf{A}^2 \cdot \mathbf{v}, \dots, \mathbf{A}^{m-1} \cdot \mathbf{v} \}.$$

В качестве вектора \mathbf{v} обычно выбирается невязка начального приближения \mathbf{r}_0 ; тогда выбор подпространства \mathcal{L} и способ построения базисов подпространств полностью определяет вычислительную схему метода.

Подпространства Крылова?

Было показано, что методы Якоби и Гаусса - Зейделя являются частными случаями класса методов, основанного на расщеплении \mathbf{A} в виде разности $\mathbf{A} = \mathbf{K} - \mathbf{R}$. Тогда исходная система (1) может быть записана в виде:

$$\mathbf{K} \cdot \mathbf{x} = \mathbf{f} + \mathbf{R} \cdot \mathbf{x} = \mathbf{f} + (\mathbf{K} - \mathbf{A}) \cdot \mathbf{x},$$

что позволяет построить итерационный процесс:

$$\mathbf{K} \cdot \mathbf{x}_{k+1} = \mathbf{K} \cdot \mathbf{x}_k + (\mathbf{f} - \mathbf{A} \cdot \mathbf{x}_k) \Rightarrow \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{K}^{-1} \cdot \mathbf{r}_k.$$

Подпространства Крылова?

Выберем $\mathbf{K} = \mathbf{I}$ и $\mathbf{R} = \mathbf{I} - \mathbf{A}$, тогда предыдущий итерационный процесс будет сведен к виду:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{r}_k \Rightarrow \mathbf{x}_k = \mathbf{x}_0 + \mathbf{r}_0 + \mathbf{r}_1 + \dots + \mathbf{r}_{k-1}.$$

Умножив обе части слева на $-\mathbf{A}$ и прибавив к ним \mathbf{f} , получим

$$\mathbf{r}_{k+1} = \mathbf{f} - \mathbf{A} \cdot \mathbf{x}_{k+1} = \mathbf{f} - \mathbf{A} \cdot \mathbf{x}_k - \mathbf{A} \cdot \mathbf{r}_k = \mathbf{r}_k - \mathbf{A} \cdot \mathbf{r}_k.$$

что позволяет найти выражение для невязки на k -ой итерации через невязку начального приближения:

$$\mathbf{r}_k = (\mathbf{I} - \mathbf{A}) \cdot \mathbf{r}_{k-1} = (\mathbf{I} - \mathbf{A})^k \cdot \mathbf{r}_0.$$

Подпространства Крылова?

Таким образом:

$$\mathbf{x}_k = \mathbf{x}_0 + \left[\sum_{j=0}^{k-1} (\mathbf{I} - \mathbf{A})^j \right] \cdot \mathbf{r}_0,$$

т.е. $\delta \mathbf{x} \in \text{span} \{ \mathbf{r}_0, \mathbf{A} \cdot \mathbf{r}_0, \mathbf{A}^2 \cdot \mathbf{r}_0, \dots, \mathbf{A}^{k-1} \cdot \mathbf{r}_0 \} = \mathcal{K}_k(\mathbf{r}_0, \mathbf{A})$.

Подпространства Крылова?

Таким образом:

$$\mathbf{x}_k = \mathbf{x}_0 + \left[\sum_{j=0}^{k-1} (\mathbf{I} - \mathbf{A})^j \right] \cdot \mathbf{r}_0,$$

т.е. $\delta \mathbf{x} \in \text{span} \{ \mathbf{r}_0, \mathbf{A} \cdot \mathbf{r}_0, \mathbf{A}^2 \cdot \mathbf{r}_0, \dots, \mathbf{A}^{k-1} \cdot \mathbf{r}_0 \} = \mathcal{K}_k(\mathbf{r}_0, \mathbf{A})$.

Следствие

В методах, использующих подпространства Крылова, невязка на k -ой итерации выражается через начальную невязку некоторым матричным полиномом.

Ортогонализация

Ортогонализация

Это алгоритм, в которых на основе счётного множества линейно независимых векторов $\mathbf{a}_1, \dots, \mathbf{a}_N$ строится множество ортогональных векторов $\mathbf{b}_1, \dots, \mathbf{b}_N$ или ортонормированных векторов $\mathbf{e}_1, \dots, \mathbf{e}_N$, причём так, что каждый вектор \mathbf{b}_j или \mathbf{e}_j может быть выражен линейной комбинацией векторов $\mathbf{a}_1, \dots, \mathbf{a}_j$.

Оператор проекции

Пусть имеются линейно независимые векторы $\mathbf{a}_1, \dots, \mathbf{a}_N$.
Определим оператор проекции следующим образом:

$$\text{proj}_{\mathbf{b}} \mathbf{a} = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\langle \mathbf{b}, \mathbf{b} \rangle} \mathbf{b},$$

где $\langle \mathbf{a}, \mathbf{b} \rangle$ – скалярное произведение векторов \mathbf{a} и \mathbf{b} . Этот оператор проецирует вектор \mathbf{a} коллинеарно вектору \mathbf{b} .

Ортогонализация Грама - Шмидта

Классический процесс Грама - Шмидта выполняется следующим образом:

$$\mathbf{b}_1 = \mathbf{a}_1$$

$$\mathbf{b}_2 = \mathbf{a}_2 - \text{proj}_{\mathbf{b}_1} \mathbf{a}_2$$

$$\mathbf{b}_3 = \mathbf{a}_3 - \text{proj}_{\mathbf{b}_1} \mathbf{a}_3 - \text{proj}_{\mathbf{b}_2} \mathbf{a}_3$$

$$\mathbf{b}_4 = \mathbf{a}_4 - \text{proj}_{\mathbf{b}_1} \mathbf{a}_4 - \text{proj}_{\mathbf{b}_2} \mathbf{a}_4 - \text{proj}_{\mathbf{b}_3} \mathbf{a}_4$$

$$\vdots$$

$$\mathbf{b}_N = \mathbf{a}_N - \sum_{j=1}^{N-1} \text{proj}_{\mathbf{b}_j} \mathbf{a}_N$$

Нормированный базис

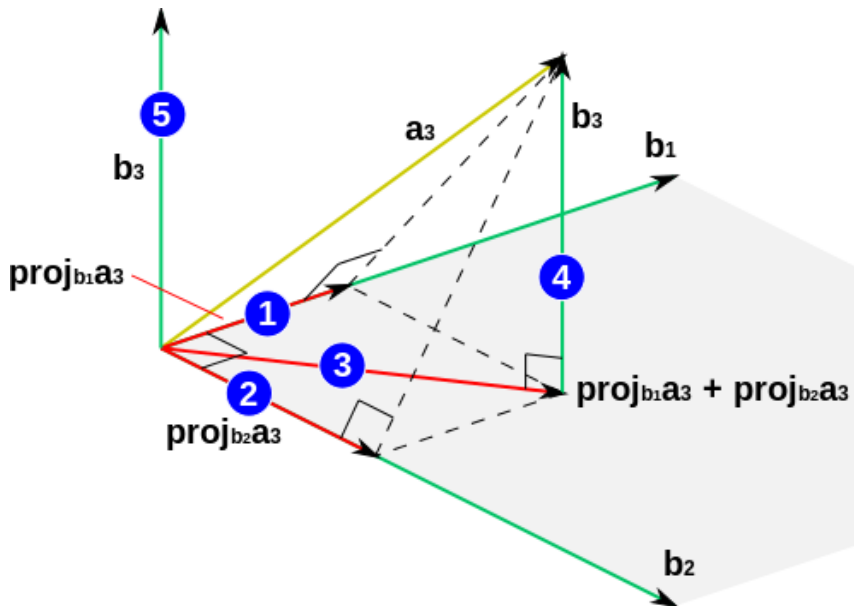
$$\mathbf{b}_j \Rightarrow \mathbf{e}_j = \frac{\mathbf{b}_j}{\|\mathbf{b}_j\|}$$

Результаты процесса Грама - Шмидта:

- ▶ $\mathbf{b}_1, \dots, \mathbf{b}_N$ – система ортогональных векторов либо
- ▶ $\mathbf{e}_1, \dots, \mathbf{e}_N$ – система ортонормированных векторов.

Вычисление $\mathbf{b}_1, \dots, \mathbf{b}_N$ носит название **ортogonalизации Грама - Шмидта**, а $\mathbf{e}_1, \dots, \mathbf{e}_N$ – **ортонормализации Грама - Шмидта**.

Геометрическая интерпретация



Базис подпространства Крылова. Ортогонализация Арнольди

Для построения базиса в пространстве Крылова $\mathcal{K}_m(\mathbf{v}_1, \mathbf{A})$ можно применить следующий подход.