2020.04.05

Требования и рекомендации по заданию 2 (1 курс, 2 семестр)

Второе задание — работа с матрицами. Преобразования матриц на основе исключения Гаусса (приведения к ступенчатому виду). Четыре основные задачи: вычисление ранга, вычисление определителя, решение системы линейных уравнений, вычисление обратной матрицы.

Описание математических вычислительных алгоритмов

Вспомнить линейную алгебру и соответствующие свойства матриц при перестановке строк или их линейной комбинации.

Метод Гаусса состоит в приведении матрицы к ступенчатому (верхне-треугольному) виду путем выполнения линейной комбинации строк. Будем далее нумеровать элементы матрицы в стиле C, т.е. от 0 до $n-1$. Если есть строки $\{a_{ij}\}_{j=0}^{n-1}$ и $\{a_{kj}\}_{j=0}^{n-1}$, то, умножая i-ю строку на $\alpha = -a_{k0}/a_{i0}$ и складывая ее с k-й строкой, получим строку $0, a'_{k1}, \ldots, a'_{k,n-1}$, где $a'_{kj} = a_{kj} + \alpha * a_{ij}$. То есть мы обнулили первый коэффициент k-й строки.

Взяв первую строку (ведущий элемент a_{00}), мы можем таким образом обнулить все элементы первого столбца для индексов $i = 1, \ldots, n-1$. В полученной подматрице $i = 1, \ldots, n-1$, $j = 1, \ldots, n-1$ можно таким же образом обнулить столбец для индесов $j = 1, i = 2, \ldots, n-1$, взяв в качестве ведущего элемент a'_{11}, и т.д. до последней строки. В результате будет обнулена нижняя треугольная часть матрицы, т.е. мы получили верхне-треугольний вид.

На практике не все так просто. Во-первых, деление a_{kj}/a_{ij} невозможно, если a_{ij} есть нуль. Во-вторых, умножение на α пропорционально увеличивает погрешность, присутствующую ранее в элементах матрицы. Поэтому при проведении исключения нужно сделать так, чтобы все операции были осуществимы и значение $|\alpha|$ было по возможности наименьшим. Для этого на каждом шаге метода Гаусса в матрице перествляются строки так, чтобы после перестновки очередное значение $|\alpha|$ было минимальным. При исключении элементов очередного столбца $j = s, i = s+1, \ldots, n-1$ мы предварительно ищем максимальное значение модуля $|a_{is}|$, $i = s+1, \ldots, n-1$, по текущему столбцу. Пусть этот максимум достигается при $i = r$, Переставляем местами s-ю и r-ю строки. Теперь коэффициент α по модулю не превосходит 1, и все последующие вычисления не будут давать экспоненциальное накопление погрешности. Если вдруг этот максимум оказался равен нулю, то все элементы уже обнулены, и на этом шаге ничего делать не надо.

После получения треугольного вида можно решать задачи, учитывая как изменяются ранг, определитель и т.д. при перестановке строк исходной матрицы.

Ранг. Решение очевидно — нужно подсчитать количество “ненулевых” диагональных элементов.

Определитель. Перемножаем диагональные элементы. Принимаем во внимание влияние перестановок и линейных комбинаций строк на значение определителя.

Система уравнений. При исключении Гаусса те же самые преобразования применяются к вектору правой части системы. Потом выполняется обратная постановка. Из последнего уравнения находится x_{n-1}, потом из предпоследнего уравнения находится x_{n-2} и т.д. При решении системы достаточно рассматривать случаи невырожденной и вырожденной матриц, т.е. существует ли единственное решение или нет. Не надо искать фундаментальную систему решений в случае вырожденной матрицы.

Обратная матрица. При исключении Гаусса те же самые преобразования применяются к отдельной дополнительной единичной матрице. После приведения к треугольному виду, далее по аналогии, снизу-вверх выполняется последовательное обнуление верхней треугольнной части матрицы с помощью обнуления столбцов для $j = n-1, \ldots, 1$ (с одновременным применением тех же преобразований к дополнительной матрице). Исходная матрица теперь преобразовалась к диагональной форме. Делим строки на диагональные элементы и получаем единичную матрицу, а на месте бывшей дополнительной единичной матрицы — обратную матрицу.

Более подробно эти алгоритмы можно посмотреть в других источниках, они много где описаны.

При решении всех задач важное значение имеет оценка того является ли данный диагональный элемент матрицы нулевым или нет. При наличии вычислительной погрешности это оказывается совсем неочевидным. Является ли нулем значение 10^{-6}? А 10^{-9}, 10^{-12}? Все зависит от специфики задачи. Если в одном случае элемент равен 10^{-6} и мы решили, что это не 0, то, умножив все коэффициенты матрицы на 10^{-8}, мы получим тот же коэффиицент равным 10^{-14}, и можем решить, что это уже 0, но по сути в задаче ничего не изменилось, умножение матрицы на число не меняет ранг, определитель просто умножается на число, решение системы не изменяется при умножении уравнений на число, отличное от нуля, и т.д. Таким образом, для проверки диагонального элемента на нуль/не нуль мы должны как-то использовать информацию о масштабе данных задачи. Это не так просто. Поэтому в данном случае мы примем два подхода к решению этой проблемы.

Первый путь — оценка масштабов исходных данных. Строгое математическое исследование этого вопроса опирается на понятия нормы матрицы, числа обусловленности и пр. Мы пока не будем вникать в эти вопросы и просто проведем анализ масштаба коэффициентов матрицы. Мы можем привести элементы матрицы к некоторому "стандартному" масштабу и потом выполнять вычисления. В качестве нормирующего коээффициента можно выбрать некоторые "нормы" матриц. Можно использовать разные нормы, например, вот такие

1. Норма $\|A\|_1 = \max\limits_{0\le j\le n-1} \sum\limits_{i=0}^{n-1} |a_{ij}|$
2. Норма $\|A\|_\infty = \max\limits_{0\le i\le n-1} \sum\limits_{j=0}^{n-1} |a_{ij}|$

Вычисляем норму исходной матрицы. Делим все коэффициенты на эту норму (т.е. приводим норму матрицы к единице). Далее считаем, что коэффициент равен нулю, если его значение по модулю меньше, чем δn, где n — размерность матрицы, а $\delta \sim 10^{-14} - 10^{-15}$. В действительности этот метод не совсем корректен, но во многих случаях он дает более-менее разумную оценку. Теперь масшабирование исходных данных задачи (умножение на константу) не будет оказывать вляния на результат.

Другой вариант состоит в непосредственном вычислении оценки погрешности каждого отдельного коэффициента. Но об этом будет отдельный разговор.

Реализация.

Алгорим должен быть реализован в виде отдельной функции, получающей на вход матрицу и другие необходмые данные. На выходе как обычно должен выдаваться требуемый ответ и код успешности вычислений (в зависимости от специфики задачи).

Кроме этого полезно и удобно реализовать несколько вспомогательныйх функций: ввод матрицы из файла, генерация матрицы данной размерности по заданным формулам, распечатка матрицы на экран или в файл, выделение памяти под матрицу, освобождение памяти матрицы и т.п.

Матрицу можно представлять разными способами. Можно с помощью двойного указателя `double**` — т.е. как массив указателей на массивы строк матрицы. Можно через одинарный указатель `double*`, записывая элементы матрицы в одномерный массив "по строкам". Второй способ является более предпочтительным с точки зрения быстродействия так как работа с элементом матрицы требует только одного обращения к памяти, хранящей матрицу, в отличие от первого способа, где таких обращений будет два. Для упрощения кода при записи элементов матрицы можно использовать конструкцию define чтобы скрыть арифметические операции вычисления одномерной позиции элемента по двум индексам. Подобные идеи описывались ранее в лекциях, поэтому здесь приведем только пример двух функций: выделения памяти под матрицу и заполнения матрицы по формуле (матрица Гильберта).

```
// создание прямоугольной матрицы на m строк и n столбцов
```

```
double * CreateMatr(int m, int n)
{
    double * p = (double*)malloc(m*n*sizeof(double));
    return p;
}

// заполнение матрицы по формуле (матрица Гильберта)
void FillMatr(double *matr, int m, int n)
{
#define A(i,j) matr[(i)*n + (j)]
    int i, j;
    for (i=0; i<m; i++) {
      for (j=0; j<n; j++) {
            A(i,j) = 1.0/(1.0 + i + j);
        }
    }
#undef A
}
```

Соответственно, функции, решающие данные задачи, могут иметь примерно такие заголовки

```
int  Rank(double *matr, int m, int n);
double Determinant(double *matr, int n);
double Solver(double *matr, int n, double *f, double *x, int *success);
int Inverse((double *matr, int n, double *inverse);
// success - код успешности решения системы уравнений
// inverse - обратная матрица
// return Inverse - код успешности вычисления обратной матрицы
```

Не забывайте освобождать память после работы с матрицей.

Тесты и проверка.

Для тестов можно взять сначала простейшие примеры из задачника по линейной алгебре. Матрицы порядка 3-4-5 и ответы из задачника позволят проверить программу на простейшие ошибки. В данном случае матрицу можно задать в файле. Соответственно тесты будут включать несколько таких файлов с известными ответами.

Вторая проверка состоит в обработке матриц большой размерности. Известно, что трудомкость метода Гаусса есть $O(n^3)$ арифметических операций. Таким образом, обработка матрицы размерностью $n \sim 1000$ должна занимать несколько секунд или меньше. Нужно предусмотреть формирование матрицы для заданной размерности n прямо в программе, вычисляя элементы по заданной формуле.

Для ранга можно провести такой тест: взять матрицу размерности n, состоящую из одинаковых строк. Ее ранг будет 1. Потом изменить в ней один элемент (чтобы ранг стал 2). Изменить еще один элемент, чтобы ранг стал 3 и т.д. Можно придумать или поискать матрицы большой размерности с известным рангом. Другой тест — взять матрицу, строки которой есть “почти параллельные” векторы. Насколько чувствительным будет вычисление ранга к этому “почти”. Например, можно взять ту же матрицу из одинаковых строк (ранг 1) и прибавить к ней диагональную матрицу с очень маленькими элементами на диагонали. Векторы-строки станут формально линейно независимыми (ранг n), но в пределах малых добавок по диагонали. Меняя масштабы этих добавок, можно проверять надежность работы программы.

Такие же матрицы можно взять для вычисления определителя. Кроме этого известно много матриц, задаваемых формулами, для которых известна формула для вычисления определителя. Поищите такие матрицы в литературе и присылайте. Мы составим базу подобных примеров.

Для системы линейных уравнений, кроме простых тестов из задачников по алгебре, можно просто сгенерировать систему с известным решением. Генерируем матрицу любым образом (по формуле или случайную). Берем известное решение, например $x = (1, 1, \ldots, 1)$ или $x = (1, 2, \ldots, n)$. Умножаем сгенерированную матрицу на этот вектор (отдельной процедурой или может можно вычислить сразу, если матрица задана формулами) и получаем правую часть системы. Решаем систему вашей процедурой, и проверяем какой ответ получился.

Для обратной матрицы также существуют примеры с явными формулами для прямой и обратной матриц, и также можно написать процедуру умножения матриц и проверять, что произведение прямой на обратную дает единичную матрицу.

Вот несколько примеров тестовых матриц, которые можно использовать для проверки решений.

Матрицы небольшого размера.

1.

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix},$$

2.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} 4 & -6 & 4 & -1 \\ -6 & 14 & -11 & 3 \\ 4 & -11 & 10 & -3 \\ -1 & 3 & -3 & 1 \end{pmatrix},$$

3.

$$A = \begin{pmatrix} 99 & 99 & 99 & 98 \\ 99 & 99 & 98 & 99 \\ 99 & 98 & 97 & 99 \\ 98 & 99 & 99 & 97 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} -97 & -99 & 99 & 98 \\ -99 & -97 & 98 & 99 \\ 99 & 98 & -99 & -99 \\ 98 & 99 & -99 & -99 \end{pmatrix},$$

4.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 9.9 & 99.8 & 999.7 \\ 1 & 99.8 & 9999.9 & 999999.8 \\ 1 & 999.7 & 999999.8 & 999999999.9 \end{pmatrix}, \qquad b = \begin{pmatrix} 10 \\ 4319 \\ 4030199.5 \\ 4003001999.4 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix},$$

5.

$$A = \begin{pmatrix} 7777 & 7777 & 7778 \\ 7777 & 7776 & 7776 \\ 7776 & 7776 & 7775 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} -7776 & 1 & 7776 \\ 1 & -1 & 0 \\ 7776 & 0 & -7777 \end{pmatrix},$$

6.

$$A = \begin{pmatrix} 99999 & 99999 & 99998 \\ 99999 & 99998 & 99998 \\ 99998 & 99998 & 99997 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} -99998 & 1 & 99998 \\ 1 & -1 & 0 \\ 99998 & 0 & -99999 \end{pmatrix},$$

Матрицы произвольного размера

1.

$$a_{ij} = \begin{cases} c, & i = j = 0, \ldots, n-2; \\ d, & i \neq j \;||\; i = j = n-1; \end{cases} \qquad \det A = d(c-d)^{n-1},$$

$$a_{ij}^{(-1)} = \begin{cases} \dfrac{1}{c-d}, & i = j = 0, \ldots, n-2; \\ \dfrac{c + (n-2)d}{d(c-d)}, & i = j = n-1; \\ -\dfrac{1}{c-d}, & i = 0, 0 < j < n-1 \;||\; j = 0, 0 < i < n-1 \\ 0 & \text{остальные } i, j \end{cases}$$

2.

$$a_{ij} = \begin{cases} 1, & i = j = 0, \ldots, n-2; \\ j+1, & i = n-1, j = 0, \ldots, n-1; \\ i+1, & i = 0, \ldots, n-1, j = 0; \\ 0 & \text{остальные } i, j \end{cases} \qquad \det A = -\frac{n(n+1)(2n-5)}{6}.$$

$$a_{ij}^{(-1)} = \begin{cases} 1 - \alpha i^2, & i = j = 0, \ldots, n-2; \\ -\alpha, & i = j = n-1; \\ -\alpha ij, & i \neq j, i, j < n-1; \\ \alpha i & i = 0, \ldots, n-2, j = n-1; \\ \alpha j & i = n-1, j = 0, \ldots, n-2; \\ 0 & \text{остальные } i, j \end{cases} \qquad \text{где } \alpha = \frac{6}{n(n+1)(2n-5)}.$$

3.

$$a_{ij} = \begin{cases} -1, & i + j = n-2; \\ 1, & i + j = n-1; \\ 0 & \text{остальные } i, j \end{cases} \qquad a_{ij}^{(-1)} = \begin{cases} 1, & i + j \geq n-1; \\ 0 & \text{остальные } i, j \end{cases}$$

4.

$$a_{ij} = \begin{cases} 2, & i = j; \\ -1, & i = j+1 \mid\mid i = j-1; \\ 0 & \text{остальные } i, j \end{cases} \qquad \det A = n + 1.$$

5. Матрица Гильберта

$$a_{ij} = \frac{1}{1+i+j}, \quad i, j = 0, 1, \ldots, n-1$$

Решение задач с такой матрицей будет “ломаться” на размерностях порядка $n \sim 14 \ldots 16$.

Эти матрицы можно использовать также и для тестов решения линейных систем.

Формальные требования в выполнению задания с матрицами.

1. Решение должно быть реализовано в нескольких файлах, один из которых содержит только функции обработки матрицы, а прототипы этих функций должны быть собраны в отдельном h файле (т.е. должна быть создана как-бы независимая “библиотека” для работы с матрицами). Запись кода должна удовлетворять правилам хорошего стиля. Из кода теста должно быть понятно как отключить (закомментировать) отдельные тесты или изменить параметры вызовов тестирующих процедур.

2. Тест должен последовательно запускать варианты тестов отдельных задач и выдавать результаты так, чтобы можно было понять результаты каждого теста (писать диагностику на экран, записывать ответы в файл и т.п.).

3. Тесты должны содержать разнообразный набор данных, матрицы небольшого размера, матрицы произвольного размера (генерируемые по формулам) для разных масштабов размеров ($n = 10, 500, 1000, \ldots$). Отчет по тесту должен описывать выводы по тесту — достигнутая точность, погрешность определения диагонального нуля и т.д. Проверка теста начинается с чтения отчета, т.е. буз отчета тест не рассматривается.

4. Тесты дожны включать (и проходить) примеры масштабирования исходных данных, т.е. результат не должен зависеть от умножения матрицы на число.

5. После сдачи первого этапа (прямая реализация алгоритма) можно приступить к реализации с одновременным вычислением решения и погрешности.

Про обработку матрицы с одновременным вычислением погрешности элементов будет отдельное описание ...