Нижегородский государственный университет им. Н.И. Лобачевского

Параллельные численные методы

Лабораторная работа
Решение невырожденных СЛАУ методом бисопряженных градиентов с предобуславливанием

При поддержке компании Intel

Козинов Е.А.,
кафедра математического обеспечения ЭВМ

Содержание

- Задача поиска решения системы линейных уравнений итерационным методом бисопряженных градиентов
- Последовательная реализация алгоритма бисопряженных градиентов
- Анализ сходимости метода бисопряженных градиентов
- Метод бисопряженных градиентов с предобуславливанием
- Об возможности параллельной реализации алгоритма

Введение (1)

- Рассмотрим систему из n линейных алгебраических уравнений вида

$$
\begin{aligned}
a_{11}x_1 &+ a_{12}x_2 &+ \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 &+ a_{22}x_2 &+ \ldots + a_{2n}x_n &= b_2 \\
\ldots \\
a_{n1}x_1 &+ a_{n2}x_2 &+ \ldots + a_{nn}x_n &= b_n
\end{aligned}
$$

- В матричном виде система может быть представлена как

$$Ax=b$$

- $A=(a_{ij})$ есть вещественная матрица размера $n\times n$; A – разреженная матрица; b и x – вектора из n элементов.

Введение (2)

- Методы решения СЛАУ можно разделить на *прямые* и *итерационные*.
- Оба подхода имеют свои достоинства и недостатки.
  - При использовании прямых методов возникает *проблема заполнения матрицы системы* при выполнении разложения, что может приводить к неудовлетворительным затратам памяти.
  - При использовании итерационных методов, не приводящих к заполнению матрицы, можно в ряде случаев наблюдать достаточно *медленную сходимость к решению*.
- В данной лабораторной работе рассмотрим итерационные методы решения СЛАУ.

Введение (3)

- *Итерационный метод* генерирует последовательность векторов $x^{(s)} \in R^m$, $s=0,1,2,\ldots$, где $x^{(s)}$ – приближенное решение системы.
- *Сходимостью метода* называют сходимость последовательности $x^{(s)}$ к точному решению системы из любого начального приближения .
- *Скорость сходимости* метода определяется количеством приближений, которые построены методом до удовлетворения критерию остановки.
- На практике сходимости недостаточно. В процессе вычислений неизбежно появление вычислительной погрешности. Метод называется *вычислительно устойчивым*, если вычислительная погрешность стремится к нулю при уменьшении погрешности вычислений.
- *Проблемы сходимости и вычислительной устойчивости итерационных методов – это основные вопросы, которые решаются при исследовании качества итерационных методов.*

Введение (4)

- Большинство методов быстро сходятся, если матрица хорошо обусловлена или имеет небольшое число собственных значений.
- В противном случае из-за накопления вычислительной погрешности метод, который сходится в теории, на практике может разойтись.
- Для борьбы с плохой обусловленостью матрицы используется *предобуславливание* системы – переход к СЛАУ с тем же решением и матрицей, обладающей лучшими качествами, с помощью умножения системы на матрицу специального вида.

Цели работы

- **Цель лабораторной работы** – продемонстрировать практическую реализацию метода бисопряженных градиентов для невыражденных разреженных матриц систем линейных уравнений

Задачи работы

- Изучение метода бисопряженных градиентов для плотных матриц.
- Модификация метода бисопряженных градиентов для разреженных матриц.
- Разработка последовательной реализации метода бисопряженных градиентов для разреженных матриц.
- Анализ сходимости разработанного метода.
- Разработка последовательной реализации метода бисопряженных градиентов с применением предобуславливателя.
- Анализ сходимости разработанного метода.

Тестовая инфраструктура

Процессор	2 четырехъядерных процессора Intel® Xeon E5520 (2.27 GHz)
Память	16 Gb
Операционная система	Microsoft Windows 7
Среда разработки	Microsoft Visual Studio 2008
Компилятор, профилировщик, отладчик	Intel® Parallel Studio XE 2011
Библиотеки	Intel® Math Kernel Library (в составе Intel® Parallel Studio XE 2011)

Метод бисопряженных градиентов (1)

- Метод бисопряженных градиентов является обобщением метода сопряженных градиентов на случай СЛАУ с *произвольной невырожденной* матрицей системы уравнений.
- Известно, что матрица $A^t * A$ – симметричная и положительно определенная.
- Следовательно, можно перейти к решению новой системы эквивалентной исходной:

$$A^t * A\, x = A^t * b$$

- Данную систему линейных уравнений можно решать итерационным методом *сопряженных градиентов*, но данный подход плохо применим на практике, так как произведение $A^t * A$ существенной повышает обусловленность матрицы.
- Основываясь на соотношении можно получить алгоритм не обладающим недостатком решения системы $A^t * A\, x$.
  - Для этого применяются последовательность невязок и направлений из метода сопряженных градиентов и бисопряженные к ним.

Метод бисопряженных градиентов (2)

- Алгоритм бисопряженых градиентов

1. Вычислить $r_0 = b - Ax_0$; выбрать $\bar{r}_0$ так, чтобы $(r_0, \bar{r}_0) \neq 0$ (например, выбрать $\bar{r}_0 = r_0$).
2. Положить $p_0 = r_0$, $\bar{p}_0 = \bar{r}_0$
3. for j=0,1,… do
4. $\quad \alpha_j = (r_j, \bar{r}_j) / (Ap_j, \bar{p}_j)$
5. $\quad x_{j+1} = x_j + \alpha_j p_j$
6. $\quad r_{j+1} = r_j - \alpha_j A p_j$
7. $\quad \bar{r}_{j+1} = \bar{r}_j - \alpha_j A^T \bar{p}_j$
8. $\quad \beta_j = (r_{j+1}, \bar{r}_{j+1}) / (r_j, \bar{r}_j)$. Если $\beta_j = 0$ или $\| r_{j+1} \| < \varepsilon$ то Стоп.
9. $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
10. $\quad \bar{p}_{j+1} = \bar{r}_{j+1} + \beta_j \bar{p}_j$
11. end j

Последовательная реализация метода бисопряженных градиентов

Создание проектов (1)

- Для удобства разделим реализацию метода бисопряженных градиентов на несколько проектов. Всего необходимо создать три проекта:
  - **parser** – проект содержащий реализацию функциональности связанной с чтением систем уравнений из файлов и часть необходимых операций используемых для выделения памяти и инициализации данных.
  - **routine** – проект содержащий часть математических операций таких как умножение матриц и векторов, а также проверку корректности решения.
  - **BiCG** – проект содержащий реализацию метода бисопряженных градиентов, а также главную функцию программы.

Создание проектов (2)

- В проекте **parser** необходимо создать следующий набор файлов:
  - **readMTX.h, readMTX.cpp** – файлы для объявления и реализации функций необходимых для чтения матриц из файла
  - **routines.h, routines.cpp** – файлы для объявления и реализации вспомогательных функций необходимых для чтения матриц из файла и вывода прочитанных данных из файла.
  - **type.h** – файл содержащий объявление используемых структур данных и констант
  - **util.h, util.cpp** – файлы для объявления и реализации набора функций выделения, инициализации и удаления данных для матриц, представленных в формате CRS.

Создание проектов (3)

- В проекте **routine** необходимо создать следующий набор файлов:
  - **sparseMatrixOperation.h, sparseMatrixOperation.cpp** – файлы для объявления и реализации разреженных операций работы с матрицами в формате CRS.
  - **timer.hpp, timer.cpp** – файлы для объявления и реализации функций замера времени.
  - **validation.h, validation.cpp** – файлы для объявления и реализации функций проверки корректности полученного решения.

Создание проектов (4)

- Проект **BiCG** должен содержать следующий набор файлов:
  - **BiCG.h, BiCG.cpp** – различные реализации метода бисопряженных градиентов.
  - **main.cpp** – файл должен содержать реализацию главной функции программы.

Создание проектов (5)

- Между проектами необходимо установить зависимости.
- Проект **routine** зависит от **parser**, а **BiCG** зависит от **routine** и **parser**.

Используемые структуры данных

- Для представления разреженной матрицы используем формат CRS (Column Row Storage).
- Объявим в файле **type.h, проекта parser** структуру **CrsMatrix**, описывающую матрицу в формате CRS:

```
typedef struct CrsMatrix
{
  int N;                // Размер матрицы (N x N)
  int NZ;               // Кол-во ненулевых элементов
  FLOAT_TYPE* Value;    // Массив значений (размер NZ)
  int* Col;             // Массив номеров столбцов
                        //   (размер NZ)
  int* RowIndex;        // Массив индексов строк
                        //   (размер N + 1)
} crsMatrix;
```

Функция main() (1)

- Функцию **main()**построим по следующей схеме:
  - Чтение аргументов командной строки
  - Чтение матрицы системы из файла
  - Инициализация переменных
    - Выделение памяти
    - Задание вектора правой части
  - Решение СЛАУ методом бисопряженных градиентов
  - Подсчет нормы невязки системы с полученным решением
  - Вывод информации о работе метода
  - Освобождение памяти

Функция main() (2)

- Прочитаем аргументы командной строки и объявим переменные

```
int main(int argc, char ** argv)
{
  // 1. Чтение аргументов командной строки
  char *matrixName;
  ParseArgv(argc, argv, matrixName);

  // объявление необходимых переменных
  crsMatrix readA;   // прочтеная матрица
  crsMatrix *matA;   // указатель на матрицу
                     // используюмую в вычислениях
  int typeOfMatrix;  // тип прочтенной матрицы
  int error;         // код ошибки возвращаемый   функцями
  double diff;       // ошибка вычисления
  int iter;          // количество произведенных итераций
  // таймер используемый для замера
  // времени работы частей алгоритма
  Stopwatch *time = createStopwatch();
  int i;
  double *b;         // вектор правой части
  double *x;         // искомое решение СЛАУ
```

Функция main() (3)

- Прочитаем матрицу

```
// 2. Чтение матрицы из файла
 printf("read matrix (%s) \n", matrixName);
 time->start();
 error = ReadMatrixFromFile(matrixName,
   &(readA.N), &(readA.NZ),
   &(readA.Col), &(readA.RowIndex), &(readA.Value),
   &(typeOfMatrix));

 if(error != BICG_OK)
 {
 printf("error read matrix %d\n", error);
 return error;
 }
```

Функция main() (4)

```
// если матрица симметричная и
// задана только верхним треугольником,
// то дополняем ее до полной
if(typeOfMatrix == UPPER_TRIANGULAR)
{
  matA = UpTriangleMatrixToFullSymmetricMatrix(&readA);
  FreeMatrix(readA);
}
else
{
  matA = &readA;
}
time->stop();

printf("read matrix from file time: %f\n",
  time->getElapsed());
```

Функция main() (5)

- Проинициализируем значения переменных

```
// 3. Инициализация переменных
// Выделение памяти под вектор правой части и
// решение СЛАУ
x = new double [matA->N];
b = new double [matA->N];

// инициализация правой части
for(i = 0; i < matA->N; i++)
{
  b[i] = 1.0;
}
```

Функция main() (6)

- На месте выделенного комментария «вызов функции решения СЛАУ методом BiCG» поместим в дальнейшем вызов функции с нашей реализацией метода.

```
time->reset();
time->start();
// 4. вызов функции решения СЛАУ методом BiCG
time->stop();
```

Функция main() (7)

- В финале необходимо проверить корректность решения и освободить выделенную память

```
// 5. Проверка корректности BiCG
 diff = diffSolution(*matA, x, b);

 // 6. Вывод информации о работе метода
 printf("BiCG time: %f\n", time->getElapsed());
 printf("count of iteration: %d\n", iter);
 printf("calc error: %f\n", diff);
 // 7.  Освобождение динамической памяти
 FreeMatrix(*matA);
 delete [] b;
 delete [] x;
```

Вспомогательные функции (1)

- Рассмотрим вспомогательные функции, которые необходимы для реализации метода.
- Функции выделения и освобождения памяти для используемой структуры хранения матрицы поместим в файлах **util.h** и **util.cpp**.
  - **InitializeMatrix()** – выделение памяти для хранения матрицы в формате CRS. На вход функция принимает размер матрицы **N**, число ненулевых элементов **NZ**. Выходом функции являются ссылка на структуру матрицы в формате CRS с проинициализированными полями и выделенной необходимой памятью. Функция возвращает код ошибки.

```
int InitializeMatrix(int N, int NZ, crsMatrix &mtx);
```

  - **FreeMatrix()** – освобождение памяти из-под матрицы в формате CRS. Выходом функции являются ссылка на структуру содержащую матрицу. Функция возвращает код ошибки.

```
int FreeMatrix(crsMatrix &mtx);
```

Вспомогательные функции (2)

- В файл **readMTX.h** проекта **parser** поместим объявление, а в файл **readMatrix.cpp** – реализацию функции **ReadMatrixFromFile()**, выполняющей чтение матрицы из файла в формате mtx и сохранение ее в формат CRS. На вход функция будет принимать имя файла **matrixName**, содержащего матрицу. Выходом функции являются размер матрицы **n**, указатели на инициализированные массивы **column**, **row**, **val**, описывающие матрицу. Функция возвращает код ошибки.

```
int ReadMatrixFromFile(char* matrixName,
  int* n, int** column, int** row,
  FLOAT_TYPE** val);
```

Формат mtx

- Файл в формате mtx представляет собой запись матрицы в координатном формате.
  - В файле записывается строка с параметрами матрицы – числом строк, столбцов и ненулевых элементов.
  - Затем построчно записаны характеристики ненулевых элементов матрицы: строка, столбец и значение каждого.
  - Нумерация строк и столбцов начинается с единицы.
  - Если матрица симметричная, файл может содержать только ее верхний или нижний треугольник.
  - Поля комментариев начинаются символом «%»

Задание №1

- Реализуйте рассмотренные вспомогательные функции работы с матрицами в формате CRS, а также их чтение из файла.

Вспомогательные функции (3)

- Выполним реализацию матрично-векторных операций, используемых в методе бисопряженных градиентов.
- В файле **sparseMatrixOperation.h** проекта **routine** поместим объявление, а в файле **sparseMatrixOperation.cpp** – реализацию соответствующих функций.

Вспомогательные функции (4)

- **MatrixVectorMult()** – вычисление произведения матрицы на вектор. На вход функция принимает указатели на матрицу **A** в формате CRS и вектор **b**. Выходом функции является указатель на их произведение **x**. В качестве результата функция возвращает код ошибки.

```
int MatrixVectorMult(crsMatrix A, double * b,  double *x)
{
  int i, j;
  int s, f;
  for(i = 0; i < A.N; i++)
  {
    s = A.RowIndex[i];
    f = A.RowIndex[i + 1];
    x[i] = 0.0;
    for(j = s; j < f; j++)
      x[i] += A.Value[j] * b[ A.Col[j] ];
  }
  return BICG_OK;
}
```

Вспомогательные функции (5)

- **scalarProduct()** – вычисление скалярного произведения векторов. На вход функция принимает указатели на вектора **a**, **b** и их размер **n**. Выходом функции является скалярное произведение .

```
double scalarProduct(int n, double *a,  double *b)
{
  double sum = 0.0;
  int i;
  for(i = 0; i < n; i++)
  {
    sum += a[i] * b[i];
  }
  return sum;
}
```

Программная реализация метода бисопряженных градиентов (1)

- Приступим к программной реализации метода решения системы линейных уравнений методом бисопряженных градиентов.
- В файле **BiCG.h** объявим и в файле **BiCG.cpp** реализуем функцию **BiCG()**, выполняющую итерационное решение СЛАУ рассматриваемым методом.
- На вход функция будет принимать матрицу системы **A** в формате CRS, вектор правой части **b** и максимально допустимое количесво итераций **CountIteration**. Выходом функции будет указатель на вычисленное приближенное решения **x** и число выполненных итераций **iter**.

Программная реализация метода бисопряженных градиентов (2)

- В теле функции будем вычислять приближения к решению системы в массиве **x**.
- В качестве критерия останова метода возьмем достижение максимально допустимого числа итераций **CountIteration** или достижение необходимой точности решения.
- Достигнутую точность будем вычислять в переменной **check**, как относительную норму невязки $\|r^{(k)}\| / \|b\|$
  - Необходимая точность задана константой **EPSILON** в файле **type.h** проекта **parser**.

Программная реализация метода бисопряженных градиентов (3)

```
int BiCG(crsMatrix A, double * b,  double *x,
  int CountIteration, int &iter)
{
  // Для ускорения вычислений вычислим
  // транспонированную матрицу A
  crsMatrix At;

  At.N  = A.N;
  At.NZ = A.NZ;

  Transpose(A.N, A.Col, A.RowIndex, A.Value,
    &(At.Col), &(At.RowIndex), &(At.Value));
```

Программная реализация метода бисопряженных градиентов (4)

```
// массивы для хранения невязки
// текущего и следующего приближения
double * R, * biR;
double * nR, * nbiR;

R    = new double [A.N];
biR  = new double [A.N];
nR   = new double [A.N];
nbiR = new double [A.N];

// массивы для хранения текущего и следующего вектора
// направления шага метода
double * P, * biP;
double * nP, * nbiP;

P    = new double [A.N];
biP  = new double [A.N];
nP   = new double [A.N];
nbiP = new double [A.N];
```

Программная реализация метода бисопряженных градиентов (5)

```
// указатель, для смены указателей на вектора текущего
// и следующего шага метода
double * tmp;

// массивы для хранения произведения матрицы на вектор
//направления и бисопряженный к нему
double * multAP, * multAtbiP;
multAP    = new double [A.N];
multAtbiP = new double [A.N];

// beta и alfa - коэффициенты расчетных формул
double alfa, beta;
// числитель и знаменатель коэффициентов beta и alfa
double numerator, denominator;

// переменные для вычисления
// точности текущего приближения
double check, norm;
norm = sqrt(scalarProduct(A.N, b, b));
```

Программная реализация метода бисопряженных градиентов (6)

- В качестве начального приближения возьмем единичный вектор

```
//задание начального приближения
int i;
int n = A.N;

for(i = 0; i < n; i++)
{
  x[i] = 1.0;
}

//инициализация метода
MatrixVectorMult(A, x, multAP);
for(i = 0; i < n; i++)
{
  R[i] = biR[i] = P[i] = biP[i] = b[i] - multAP[i];
}
```

Программная реализация метода бисопряженных градиентов (7)

```
// реализация метода
for(iter = 0; iter < CountIteration; iter++)
{
  MatrixVectorMult(A, P, multAP);
  MatrixVectorMult(At, biP, multAtbiP);

  numerator   = scalarProduct(A.N, biR, R);
  denominator = scalarProduct(A.N, biP, multAP);
  alfa = numerator / denominator;

  for(i = 0; i < n; i++)
  {
    nR[i] = R[i] - alfa * multAP[i];
  }

  for(i = 0; i < n; i++)
  {
    nbiR[i] = biR[i] - alfa * multAtbiP[i];
  }

  denominator = numerator;
  numerator   = scalarProduct(A.N, nbiR, nR);
  beta = numerator / denominator;
```

Программная реализация метода бисопряженных градиентов (8)

```
    for(i = 0; i < n; i++)
    {
      nP[i] = nR[i] + beta * P[i];
    }
    for(i = 0; i < n; i++)
    {
      nbiP[i] = nbiR[i] + beta * biP[i];
    }

    // контроль достежения необходимой точности
    check = sqrt(scalarProduct(n, R, R)) / norm;
    if (check < EPSILON)
      break;
    for(i = 0; i < n; i++)
    {
      x[i] += alfa * P[i];
    }

    // меняем массивы текущего и следующего шага местами
    tmp = R; R = nR; nR = tmp;
    tmp = P; P = nP; nP = tmp;
    tmp = biR; biR = nbiR; nbiR = tmp;
    tmp = biP; biP = nbiP; nbiP = tmp;
  }
```

Программная реализация метода бисопряженных градиентов (9)

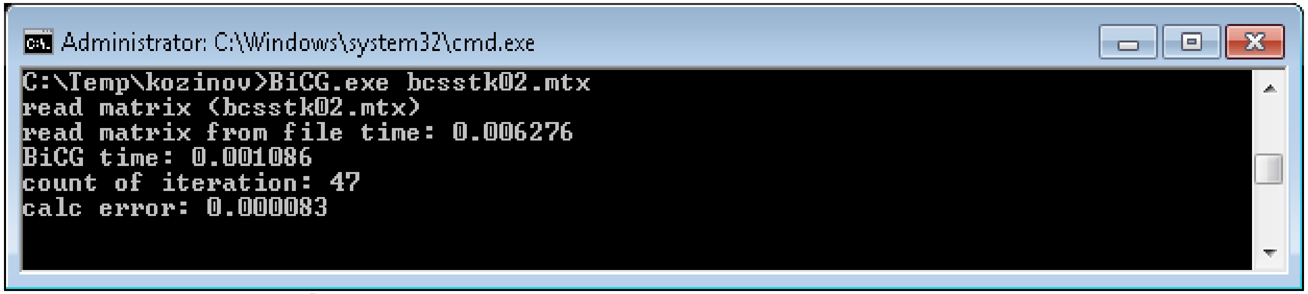
```
    // освобождение памяти
    FreeMatrix(At);
    delete [] R;
    delete [] biR;
    delete [] nR;
    delete [] nbiR;

    delete [] P;
    delete [] biP;
    delete [] nP;
    delete [] nbiP;

    delete [] multAP;
    delete [] multAtbiP;

    return BICG_OK;
}
```

Анализ сходимости метода бисопряженных градиентов (1)

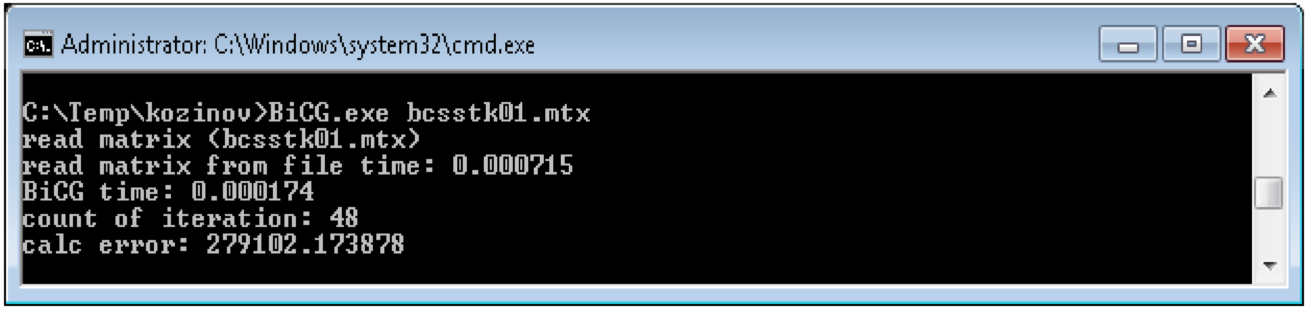
- В тело функции **main()** вставим вызов функции **BiCG()**.
- Теперь можно скомпилировать проект командой **Build→Rebuild**, и убедиться в корректности его работы.

Анализ сходимости метода бисопряженных градиентов (2)

- Пример работы программы реализующей метод бисопряженых градиентов
  на хорошо обусловленной матрице



- Пример работы программы реализующей метод бисопряженых градиентов
  на плохо обусловленной матрице



Анализ сходимости метода бисопряженных градиентов (3)

- На второй матрице решение не было получено из-за двух факторов.
  – Во-первых, сама матрица **bcsstk01.mtx** является плохо обусловленной.
  – Во-вторых, при проведении экспериментов в качестве ограничения на количество итераций алгоритма была взята теоретическая оценка – размер матрицы.

Анализ сходимости метода бисопряженных градиентов (4)

- Результаты запуска программной реализации метода бисопряженных градиентов на симметричных матрицах (необходимая точность вычисления 0,0001).

матрица	размер матрицы	достигнутая точность метода	количество итераций	время работы алгоритма
bcsstk01	48	116213,2199	48	0,000
bcsstk05	153	27,2275	153	0,002
bcsstk10	1 086	162,2504	1 086	0,092
bcsstk12	1 473	8576,2735	1 473	0,187
parabolic_fem	525 825	0,0012	717	25,089
tmt_sym	726 713	0,0062	2 487	122,543

Анализ сходимости метода бисопряженных градиентов (5)

- Результаты запуска программной реализации метода бисопряженных градиентов на не симметричных матрицах (необходимая точность вычисления 0,0001).

матрица	размер матрицы	достигнутая точность метода	количество итераций	время работы алгоритма
fs_541_1	541	0,00030	6	0,000
ex22	839	1,63999	839	0,068
sherman2	1080	5075084918,6	1080	0,103
cage10	11397	0,00213	10	0,011

Анализ сходимости метода бисопряженных градиентов (6)

- Для улучшения скорости сходимости метода используются предобуславливание.
- Перейдем к реализации метода бисопряженных градиентом с предобуславливателем.

Программная реализация метода бисопряженных градиентов с предобуславливанием

Метод бисопряженных градиентов с предобуславливанием

- Псевдокод алгоритма:

1. Вычислить $r_0 = b - Ax_0$; выбрать $\bar{r}_0$ так, чтобы $(r_0, \bar{r}_0) \neq 0$ (например, выбрать $\bar{r}_0 = r_0$).
2. Вычислить $z_0 = M^{-1} r_0$, при известном факторе ($sol = L^{-1} r_0$, $z_0 = U^{-1} sol$)
3. Вычислить $\overline{z_0} = M^{-1} \overline{r_0}$, при известном факторе ($sol = L^{-1} \overline{r_0}$, $\overline{z_0} = U^{-T} sol$)
4. Положить $p_0 = z_0$, $\overline{p_0} = \overline{z_0}$
5. for j=0,1,… do
6. $\alpha_j = \frac{(z_j, \overline{r_j})}{(Ap_j, \overline{p_j})}$
7. $x_{j+1} = x_j + \alpha_j p_j$
8. $r_{j+1} = r_j - \alpha_j A p_j$
9. $\bar{r}_{j+1} = \bar{r}_j - \alpha_j A^T \bar{p}_j$
10. Вычислить $z_{j+1} = M^{-1} r_{j+1}$, при известном факторе ($sol = L^{-1} r_{j+1}$, $z_{j+1} = U^{-1} sol$)
11. Вычислить $\overline{z_{j+1}} = M^{-T} \overline{r_{j+1}}$, при известном факторе ($sol = L^{-1} \overline{r_{j+1}}$, $\overline{z_{j+1}} = U^{-1} sol$)
10.
8. $\beta_j = \frac{(z_{j+1}, \overline{r_{j+1}})}{(z_j, \overline{r_j})}$. Если $\beta_j = 0$ или $\| r_{j+1} \| < \varepsilon$ то Стоп.
9. $p_{j+1} = z_{j+1} + \beta_j p_j$
10. $\bar{p}_{j+1} = \bar{z}_{j+1} + \beta_j \bar{p}_j$
11. end j

Создание проекта

- Реализацию предобуславливателя оставим за рамками рассмотрения данной лабораторной работы. Воспользуемся *ILU*-предобуславливателем из соответствующей лабораторной работы курса.
- Добавим в решение проект содержащий реализацию *ILU*(p)-предобуславливателя. Проект содержит следующие файлы:
  - **ilup.h** и **ilup.cpp** – файлы содержат объявление и программную реализацию символьной и численной части алгоритма *ILU*(p)
  - **validation.h** и **validation.cpp** – файлы содержат объявление и программную реализацию проверки корректности получаемого разложения, а также функцию разделения матрицы содержащей одновременно *L* и *U* на две отдельные матрицы.

Вспомогательные функции (1)

- Чтобы применить ILU-предобуславливатель реализуем в файлах **sparseMatrixOperation.cpp** и **sparseMatrixOperation.h** вспомогательную функцию решения треугольных систем **GaussSolve()**.
- Функция **GaussSolve()** решает систему линейных уравнений с треугольной матрицей. На вход функция получает структуру матрицы системы **A**, вектор правой части **b** и символ обозначающий вид системы **uplo** – символ «*L*» соответствует нижне треугольной системе, а «*U*» верхнетреугольной. Выходом функции является решение системы **x**.
- Для реализации функции воспользуемся готовой функнкцианальностью представленной в MKL.
  – Для того что бы воспользоваться функцией решения треугольных систем **mkl_dcsrtrsv()**, необходимо перевести матрицу в предстакление с индексани начинающимеся с 1.
  – После решения системы необходимо вернутся к нумерации принятой в C/C++ с 0.

Вспомогательные функции (2)

```
void GaussSolve(crsMatrix* A, char uplo,
  double* b, double* x)
{
  char transa = 'N';
  char diag   = 'N';
  int i;
  for(i = 0; i < A->N + 1; i++)
    A->RowIndex[i] ++;
  for(i = 0; i < A->NZ; i++)
    A->Col[i] ++;
  mkl_dcsrtrsv(&uplo, &transa, &diag, &(A->N), A->Value,
    A->RowIndex, A->Col, b, x);
  for(i = 0; i < A->N + 1; i++)
    A->RowIndex[i] --;
  for(i = 0; i < A->NZ; i++)
    A->Col[i] --;
}
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (1)

- Разработаем новую функцию решения системы линейных уравнений методом бисопряженных градиентов с предобуславливателем **BiCG_M()**.
- Основное отличие функции
  1. Функция принимает предобуславливатель в виде двух матриц *L* и *U*.
  2. Для вычислений необходимо хранить помимо транспонированной матрицы транспонированные матрицу фактора предобуславливателя
  3. В методе бисопряженных градиентов появляются дополнительные шаги связанные с применением предобуславливания к СЛАУ.

Программная реализация метода бисопряженных градиентов с предобуславливанием (2)

```
int BiCG_M(crsMatrix A, double * b,  double *x, crsMatrix L, crsMatrix U,
           int CountIteration, int &iter)
{
  // Для ускорения вычислений вычислим транспонированную матрицу A
  crsMatrix At;
  ...
  // Для вычисления обратной матрицы к транспонированной
  // матрицы предобуславливателя вычислим транспонированные
  // матрицы L и U
  crsMatrix Lt;

  Lt.N  = L.N;
  Lt.NZ = L.NZ;

  Transpose(L.N, L.Col, L.RowIndex, L.Value,
    &(Lt.Col), &(Lt.RowIndex), &(Lt.Value));

  crsMatrix Ut;

  Ut.N  = U.N;
  Ut.NZ = U.NZ;

  Transpose(U.N, U.Col, U.RowIndex, U.Value,
    &(Ut.Col), &(Ut.RowIndex), &(Ut.Value));
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (3)

```
// массивы для хранения невязки текущего и
// следующего приближения
double * R, * biR;
double * nR, * nbiR;
...
// вспомогательный вектор и бисоряженный к нему
// для применения предобуславливателя
double * Z, * biZ;
double * nZ, * nbiZ;
double * sol;

Z    = new double [A.N];
biZ  = new double [A.N];
nZ   = new double [A.N];
nbiZ = new double [A.N];
sol  = new double [A.N];

// массивы для хнанения текущего и следующего вектора
// направления шага метода
double * P, * biP;
double * nP, * nbiP;
...
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (4)

```
//инициализация метода
MatrixVectorMult(A, x, multAP);
for(i = 0; i < n; i++)
{
  R[i] = biR[i] = b[i] - multAP[i];
}

GaussSolve(&L, 'L', R  , sol);
GaussSolve(&U, 'U', sol, Z);
GaussSolve(&Ut, 'L', biR, sol);
GaussSolve(&Lt, 'U', sol, biZ);

for(i = 0; i < n; i++)
{
  P[i] = Z[i];
  biP[i] = biZ[i];
}
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (5)

```
// реализация метода
for(iter = 0; iter < CountIteration; iter++)
{
  MatrixVectorMult(A, P, multAP);
  MatrixVectorMult(At, biP, multAtbiP);
  numerator   = scalarProduct(A.N, biR, Z);
  denominator = scalarProduct(A.N, biP, multAP);
  alfa = numerator / denominator;
  for(i = 0; i < n; i++)
  {
    nR[i] = R[i] - alfa * multAP[i];
  }
  for(i = 0; i < n; i++)
  {
    nbiR[i] = biR[i] - alfa * multAtbiP[i];
  }
  GaussSolve(&L, 'L', nR , sol);
  GaussSolve(&U, 'U', sol, nZ);
  GaussSolve(&Ut, 'L', nbiR, sol);
  GaussSolve(&Lt, 'U', sol , nbiZ);

  denominator = numerator;
  numerator   = scalarProduct(A.N, nbiR, nZ);
  beta = numerator / denominator;
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (6)

```
  for(i = 0; i < n; i++)
  {
    nP[i] = nZ[i] + beta * P[i];
  }
  for(i = 0; i < n; i++)
  {
    nbiP[i] = nbiZ[i] + beta * biP[i];
  }
  check = sqrt(scalarProduct(n, R, R)) / norm;
 if (check < EPSILON)
      break;
  for(i = 0; i < n; i++)
  {
    x[i] += alfa * P[i];
  }
  // меняем массивы местами
  tmp = R; R = nR; nR = tmp;
  tmp = P; P = nP; nP = tmp;
  tmp = biR; biR = nbiR; nbiR = tmp;
  tmp = biP; biP = nbiP; nbiP = tmp;

  tmp = Z; Z = nZ; nZ = tmp;
  tmp = biZ; biZ = nbiZ; nbiZ = tmp;
}
```

Программная реализация метода бисопряженных градиентов с предобуславливанием (7)

```
  // освобождение памяти
   ...
   FreeMatrix(Lt);
   FreeMatrix(Ut);

   delete [] Z;
   delete [] biZ;
   delete [] nZ;
   delete [] nbiZ;
   delete [] sol;

   ...

   return BICG_OK;
}
```

Задание №2

- Модифицируйте функцию **main()** для вызова метода бисопряженных градиентов.
  - Для этого подключите заголовочный файл **ilup.h**.
  - Вызовите функцию вычисления *ILU*(p).
  - Разделите матрицы на *L* и *U.*
  - Вызовите реализованную функцию *BiCG_M*().
- Оцените влияния качества предобуславливателя на качество работы метода.
  - Для этого добавьте вызов решения системы методом бисопряженных градиентов с предобуславливателем вычисленным с разным уровнем *p*.

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (1)

- Запустим реализацию алгоритма бисопряженных градиентов на матрице **bcsstk01.mtx**.

```
Administrator: C:\Windows\system32\cmd.exe
C:\Temp\kozinov>BiCG.exe bcsstk01.mtx
read matrix (bcsstk01.mtx)
read matrix from file time: 0.000474
BiCG time: 0.000121
count of iteration: 48
calc error: 279102.173878


############### ILU(0) ################
Count elements: maxL NZ(1176) L NZ(224) U NZ(224)
ILU symbolic factorization time: 0.000028
ILU factorization time: 0.000013
BiCG with precondition time: 0.022136
count of iteration: 21
calc error: 0.000055


############### ILU(1) ################
Count elements: maxL NZ(1176) L NZ(382) U NZ(224)
ILU symbolic factorization time: 0.000070
ILU factorization time: 0.000023
BiCG with precondition time: 0.000271
count of iteration: 18
calc error: 0.000051


############### ILU(2) ################
Count elements: maxL NZ(1176) L NZ(545) U NZ(224)
ILU symbolic factorization time: 0.000130
ILU factorization time: 0.000031
BiCG with precondition time: 0.000297
count of iteration: 18
calc error: 0.000095


############### ILU(3) ################
Count elements: maxL NZ(1176) L NZ(712) U NZ(224)
ILU symbolic factorization time: 0.000184
ILU factorization time: 0.000035
BiCG with precondition time: 0.000308
count of iteration: 17
calc error: 0.000034
```

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (2)

- Матрица **bcsstk01.mtx** является плохо обусловленной.
  - Как следствие, на данной матрице наблюдалась большая погрешность итерационного метода при ограничении на количество итераций.
- Применение предобуславливателя позволило решить систему уравнений с высокой точностью.
- С повышением параметра уровня алгоритма ***p***, количество итераций затраченных для поиска решения с заданной точностью уменьшается.
  - Повышения уровня для данной матрицы позволяет улучшить качество предобуславливателя.

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (3)

- Результаты запуска программной реализации метода бисопряженных градиентов с предобуславливателем на примере матрицы bcsstk10.

матрица:	bcsstk10		**размер:**	1 086		
p	количество итераций	достигнутая точность метода	время символьной части ILU(p)	время численной части ILU(p)	время BiCG	Общее время
без ILU	*1086*	*162,250*			*0,0922*	*0,0922*
0	211	0,00030	0,0010	0,0007	0,0910	0,0927
1	83	0,00059	0,0030	0,0009	0,0335	0,0374
2	70	0,00032	0,0059	0,0010	0,0303	0,0373
3	92	0,00012	0,0093	0,0011	0,0407	0,0511

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (4)

- Результаты запуска программной реализации метода бисопряженных градиентов с предобуславливателем на примере матрицы tmt_sym.

матрица:	tmt_sym		**размер:**	726 713		
P	количество итераций	достигнутая точность метода	время символьной части ILU(p)	время численной части ILU(p)	время BiCG	Общее время
без ILU	*2487*	*0,006*			*122,5435*	*122,5435*
0	894	0,00933	0,2886	0,0906	140,6882	141,0673
1	896	0,00813	0,6497	0,1218	151,9161	152,6875
2	894	0,00870	1,3676	0,1608	163,7241	165,2525
3	895	0,01176	2,4250	0,2079	183,1238	185,7567

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (5)

- Из представленных таблиц видно, что применение предобуславливателя позволяет существенно уменьшить количество итераций затраченных методом на решение системы линейных уравнений.
- Сокращение количества итераций метода не всегда положительно влияет на время решения в целом (важен не уровень а качество предобуславливателя).
  - В матрице **tmt_sym** количество итераций сократилось более чем в два раза, но при этом время решения увеличилось.
  - В тоже время на матрице **bcsstk10** наблюдается ускорение вычислений.

Анализ сходимости метода бисопряженных градиентов с предобуславливанием (5)

- Качество предобуславливателя в алгоритме не всегда зависит от уровня.
  – Для матрицы **bcsstk10** оптимально выбрать параметр уровня равный либо 1, либо 2.
  – Для матрицы **tmt_sym** оптимально выбрать уровень 0.
- Для получения более качественных предобуславливателей, в меньшей степени зависящих от параметров, необходимо применять другие алгоритмы поиска предобуславливателей

Параллельная реализация алгоритма

- Основной вычислительной операцией в алгоритме бисопряженных градиентов является скалярное произведение.
  - Скалярное произведение имеет малую вычислительную сложность, но в алгоритме применяется большое количество раз. Распараллеливать скалярное произведение будет заведомо не эффективно.
  - Низкая эффективность связана с большим количеством накладных расходов на организацию параллелизма.
- С бОльшой эффективностью можно заменить разработанную программную реализацию скалярного произведения на вызов функции из библиотеки.
  - В качестве библиотеки можно использовать, например, Intel MKL.
- Для распараллеливания алгоритма бисоряженных градиентов с более высокой степенью эффективности можно параллельно вычислять независимые скалярные произведения.

Дополнительные задания

1. Выполнить анализ скорости сходимости метода в зависимости от точности вещественной арифметики.
2. Реализовать метод бисопряженных градиентов с предобуславливателем найденным алгоритмом *ILUT*.
3. Выполнить анализ эффективности и масштабируемости параллельной модификации алгоритма бисопряженных градиентов с распараллеливанием на уровне скалярных произведений
4. Оценить эффективность применения библиотечных реализации математических операций предлагаемых библиотекой Intel MKL в методе бисопряденных градиентов.
5. Выполнить анализ эффективности и масштабируемости параллельной модификации алгоритма бисопряженных градиентов с распараллеливанием, методом параллельного вычисления независимых скалярных произведений.

Литература

1. Saad Y. Iterative methods for sparse linear systems. – SIAM, 2003.
2. Решение симметричных разреженных СЛАУ методом сопряженных градиентов с предобуславливанием
3. Лабораторная работа умножения матриц
4. Лабораторная работа ILU(p)
5. Белов С.А., Золотых Н.Ю. Численные методы линейной алгебры. – Н.Новгород, Изд-во ННГУ, 2005.

Вопросы

- ???

Авторский коллектив

- Козинов Евгений Александрович,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ.
  Evgeniy.Kozinov@gmail.com