程序作业 1

- **提交截止时间**: 2024.10.30

本次作业主要讨论针对大规模稀疏离散系统的迭代求解。我们将求解如下线性方程组:

$$Ax = b, \tag{1}$$

其中，$A \in \mathbb{R}^{n\times n}$, $h = \frac{1}{n}$ 以及

$$A = \frac{1}{h}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}, b = h\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}.$$

我们将考虑 n 特别大的情况，比如 $n = 10^8$。此时，直接利用二维数组存储三对角矩阵 A 将浪费巨大的的存储资源。所以我们介绍一种存储稀疏矩阵的的数据结构，并基于此实现迭代算法。

1. 行压缩结构（Compressed Sparse Row）

行压缩结构，或者叫做 CSR 结构，可以有效存储稀疏矩阵中的非零元素，而不需要存储其他的零元素，并可以高效实现矩阵加法，矩阵与向量之间的乘法。该数据结构的具体介绍可以详见

`https://www.netlib.org/utk/people/JackDongarra/etemplates/node373.html`

CSR 结构含有三个数组，在程序实现中可以使用结构类语句表示。比如在 `Matlab` 中我们可以将矩阵 A 表示为如下结构:

```
A.val = [];
A.col_ind = [];
A.row_ptr = zeros(n+1, 1);
```

其中数组 `val` 用于存储稀疏矩阵中的所有非零元素（可以存储有限量的零元素），我们规定从第 1 行之第 n 行依次存储，同时为了方便我们之后的计算，在存储每一行时，我们首先存储对角元素（当然假设对角元素非零）。数组 `col_ind` 存储的是 `val` 中元素所在位置中列序号。所以，`col_ind` 和 `val` 长度相同。而数组 `row_ptr` 的长度为 $n+1$，对于 $i = 1,\ldots,n$，`row_ptr`(i) 表示的是第 i 中的第一个非零元素在数组 `val` 中的位置。最后我们规定 `col_ind`$(n+1)$ = 非零元素个数 +1。比如当 $n = 5$ 是，(1)中的矩阵 A 可以表示为如下形式:

```
A.val = [2,−1,2,−1,−1,2,−1,−1,2,−1,−1,2,−1];
A.col_ind = [1,2,2,1,3,3,2,4,4,3,5,5,4];
A.row_ptr = [1,3,6,9,12,14];
```

任务 1 建立一个函数 `[A] = csr_tri_diag_matrix(n)`, 使得根据不同大小的 n 可以输出 CSR 结构下的稀疏矩阵 A。

需要指出的是，以上介绍的 CSR 结构与一般 CSR 结构稍有修改，即我们将对角元素存储在每一行的第一个位置，这方面我们提取对角元素，即对 i 行而言，矩阵 A 的对角元素就是 `A.val(A.row_ptr(i))`。这方便我们建立 `Jacobi` 迭代。

2. 矩阵向量乘法

利用 CSR 结构，我们可以简单实现稀疏矩阵与向量之间的乘法，具体的 `Matlab` 代码可以如下编写:

```
function dst = csr_vmult(A, src)

n = length(src);
dst = zeros(n,1);

for i = 1:n
    dst(i) = 0;
    for j = A.row_ptr(i):A.row_ptr(i+1) -1
        dst(i) = dst(i) + A.val(j) * src(A.col_ind(j));
    end
end

end
```

任务 2 建立函数 `csr_vmult`，利用任务 1 中的结果计算当 $n = 16$ 时，矩阵 A 与等式(1)中的向量 b 的乘法，并输出该结果。

3. Jacobi 与 Gauss-Seidel 迭代

以上代码可以稍作修改，实现矩阵 A 的 Jacobi 迭代，具体代码如下:

```
function dst = csr_jacobi_iteration(A, b, src)

n = length(src);
dst = b;

for i = 1:n
    for j = A.row_ptr(i)+1:A.row_ptr(i+1) -1
            dst(i) = dst(i) - A.val(j) * src(A.col_ind(j));
        end
    end
    dst(i) = dst(i) / A.val(A.row_ptr(i));
end

end
```

任务 3 建立 Jacobi 迭代函数 `csr_jacobi_iteration`，利用任务 1 中的结果计算当 $n = 16$ 时，输出 `csr_jacobi_iteration(A, b, b)` 的结果。

任务 4 根据课堂笔记以及课本中的思路，建立 Gauss-Seidel 迭代函数 `csr_gs_iteration`，利用任务 1 中的结果计算当 $n = 16$ 时，输出 `csr_gs_iteration(A, b, b)` 的结果。

4. 利用迭代求解方程

利用上面所定义的函数，我们将求解问题(1)。这里用 Jacobi 迭代举例。首先，我们建立基于 Jacobi 迭代的求解器，将此函数命名为 `jacobi_solver`。该函数不仅需要矩阵 A 和向量 b，我们还需设置初始向量 `x0` 以及停止误差 `tol`。需要指出的是，我们的停止策略是：相邻两个迭代向量之间的 l^2 范数小于停止误差。在该方法收敛且迭代矩阵范数不接近 1 的情况下，该停止策略可以保证真实误差也小于我们设定的误差（详见讲义）。程序如下:

```
function x = jacobi_solver(A, b, x0, max_iteration, tol)
  x = x0;
  for k = 1:max_iteration
    p = csr_jacobi_iteration(A, b, x);
    if (norm(x-p) < tol)
      fprintf('Iteration stops at step %d.\n', k);
      x = p;
      return;
    endif
    x = p;
  endfor
  fprintf('Max iteration step reached.\n');
end
```

最后，我们编写测试脚本验证以上程序:

```
clear all;
format long;

n = 64;
A = csr_tri_diag_matrix(n);
b = ones(n, 1) / n;

max_iteration = 1000000;
tol = 1e-7;
x0 = zeros(n, 1);

x = jacobi_solver(A, b, x0, max_iteration, tol);

plot(linspace(0,1,n+2), [0, x', 0]);
```

这里我们设置的较大的 `max_iteration` 保证我们可以用足够多的迭代次数以达到设定的误差 $1e-7$。最后一行可以输出未知向量所对应的图像，它表示的是关于方程

$$-u''(x) = 1, \quad x \in (0,1),$$
$$u(0) = u(1) = 0$$

的数值解。

任务 5 实现以上 Jacobi 迭代求解方程(1)，输出当 $n = 8, 16, 32, 64, 128$ 时的迭代次数，以此列表并说明迭代次数与 n 的关系。

任务 6 类似地，实现 Gauss-Seidel 迭代求解方程(1)，输出当 $n = 8, 16, 32, 64, 128$ 时的迭代次数，以此列表并说明迭代次数与 n 的关系。

任务 7 沿用上面的思路，编写梯度下降法程序，并求解方程(1)，输出当 $n = 8, 16, 32, 64, 128$ 时的迭代次数，以此列表并说明迭代次数与 n 的关系。

任务 8(选做) 沿用上面的思路，编写共轭梯度法程序，并求解方程(1)，输出当 $n = 8, 16, 32, 64, 128$ 时的迭代次数，以此列表并说明迭代次数与 n 的关系。