Otto-von-Guericke-University Magdeburg
Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods in Systems and Control Theory

Dr. Jens Saak, Dipl.-Math. Martin Köhler
Website: http://www.mpi-magdeburg.mpg.de/csc/teaching/18ws/sc1/

Winterterm 2018/2019

Scientific Computing 1
12th Homework

Handout: 01/11/2019 **Return:** 01/18/2019

Exercise 1: (4 Points)

The LU decomposition is the main technique to solve moderate size linear systems and has an asymptotic flop count of $\frac{2}{3}n^3$ flops. Beside the classical scalar implementation nowadays panel implementations with a panel width r are used to exploit modern computer architectures.

a.) Show that the asymptotic flop count for the panel LU decomposition is invariant under the chosen panel width r.

b.) Compare the exact flop count (including the lower order terms) of the standard and the panel implementation.

Exercise 2: (12 Points)

The outer product Gaussian elimination defines one possible way to compute an LU decomposition of a matrix $A \in \mathbb{R}^{m\times n}$.

a.) Implement the rank-1 version of the algorithm as a C function with the following signature:

```
int LU2(int m, int n, double * A, int lda);
```

The input `A` with leading dimension `lda` should be overwritten by its LU decomposition as shown in the lecture. The integer return value should be 0 if the matrix was decomposed successfully or a non zero value otherwise.

Hint: The appearing rank-1 updates can be performed using your `gemm` implementation from the previous exercises.

b.) Implement the rank r version of this algorithm as a C function with the following signature:

```
int LU3(int m, int n, double * A, int lda, int r);
```

The input `A` with leading dimension `lda` should be overwritten by its LU decomposition as shown in the lecture. The panel width parameter `r` should be freely selectable by the caller. The integer return value should be 0 if the matrix was decomposed successfully or a non zero value otherwise. The the LU decomposition function from a.) to decompose the small panels.

Hint: The appearing rank-r updates can be performed using your `gemm` implementation from the previous exercises.

c.) Write a solver function that takes a square LU decomposed matrix from a.) or b.), $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$, and a right hand side $b \in \mathbb{R}^n$ as inputs and overwrites b with the solution of $LUx = b$. Use the BLAS function `DTRSV` or `DTRSM` for this purpose. The function header should be

```
void LU_solve(int n, double *LU, int ldlu, double *b);
```

where `ldlu` is the leading dimension of the matrix containing the LU decomposition.

d.) Solve the following linear system to check your code:

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{pmatrix} x = \begin{pmatrix} \frac{49}{20} \\ \frac{223}{140} \\ \frac{341}{280} \\ \frac{2509}{2520} \\ \frac{2131}{2520} \\ \frac{20417}{27720} \end{pmatrix}$$

What can you recognize? Is the solution sufficiently accurate when you think about the possibilities of double precision floating point numbers?

Hint: A skeleton code with the data for d.) is provided on the lecture's website.

Exercise 3: (10 Points)

One efficient storage scheme for sparse matrices is the Compressed Sparse Row (CSR) format introduced in the lecture. In this scheme the matrix is stored using two integer arrays, `rowptr` and `colptr`, and a `values` array containing the entry values. This storage scheme can be realized using a C structure:

```
struct sparse_matrix_st {
        int cols;
        int rows;
        int nnz;
        int *rowptr;
        int *colptr;
        double *values;
};
```

The skeleton implementation available from the lecture homepage contains a function to read the matrix from a file into CSR format called `sparse_matrix_read` . Additionally the `sparse_matrix_print` function prints a sparse matrix to the screen.

Based on the Compressed Row Storage implement the following functions:

a.) The sparse matrix vector product

$$y = Ax$$

as a C function:

```
void sparse_mvp(struct sparse_matrix_st *A, double *x, double *y);
```

b.) The Conjugate Gradient Algorithm to solve $Ax = b$ as a C function

```
int cg(struct sparse_matrix_st *A, double *x, double * b, int maxit,
                                   double tol);
```

The return value should be used to indicate if the algorithm has converged or not. Use the matrix vector product implementation from a.) and BLAS for all other linear algebra operations. The parameter `maxit` gives the maximum number of iterations allowed and the parameter `tol` stops the iteration if

$$||r_i||_2 < tol \cdot ||b||_2$$

is fulfilled. The parameter `x` is used as x_0 on input and contains the solution x_* on output.

c.) Demonstrate both functions in a main program. Use $x_0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ and $b = A \cdot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ to check if the algorithm works properly.

A skeleton code with some symmetric positive definite matrices is available on the website.

Overall Points: 26