Algorithms for symmetric, banded and sparse matrices

Numerical Linear Algebra

As a rule, the structure of the matrix should be exploited when solving a linear system $Ax = b$. There are specific methods for symmetric (definite/indefinite matrices), banded systems and sparse systems. These methods are usually faster, more accurate and require less memory (hence can be applied to larger systems).

For the symmetric case the LU factorization can be recasted as the LDL^t factorization which, for symmetric positive definite systems, reduces to the Cholesky factorization GG^t. This last turns out to be a more stable factorization and can be implement with a few memory cost.

An important aspect is the role of pivoting strategies and how they alter the structure of the matrices. We start by considering banded systems and by investigating the LU factorization for this class.

Banded systems

1. LU for banded systems: Write a routine that, for a given matrix $A \in \mathbb{R}^{n\times n}$ and positive integers $0 \leq p, q \leq n$, checks if A is p, q-banded. Then write a program that generates random matrices (e.g. with uniformly distributed coefficients in $[0, 1]$) and, for each matrix,

   - compute the $A = LU$ factorization,
   - compute the $PA = LU$ factorization.

   Use the routine to investigate the banded structure of $L + U$. What can be said about the banded structure in the $A = LU$ case? And in the $PA = LU$ case? Discuss about the implications concerning the memory requirements.

2. Multiplication of banded matrices: Banded matrices are typically stored by diagonals. If matrix $A \in \mathbb{R}^{n\times n}$ has left/lower and right/upper bandwidth p_A, q_A, and matrix B has p_B, q_B, write a routine that computes $C \leftarrow AB$ by diagonals (assume that an array of sufficient size has been allocated for C but take into account the proper bandwidth of C).

Sparse systems

Sparse matrices appear in many applications (e.g. PageRank, transition matrices,). There are several storage schemes. Available in Python there are the CSR, CSC, BSR, LIL, DOK, COO and DIA schemes (see `http://www.scipy-lectures.org/advanced/scipy_sparse/storage_schemes.html#common-methods`).

3. Consider sparse matrices stored in compressed sparse row format (CSR). Concretely,

   (a) Use `scipy.sparse.csr_matrix` to create the matrix

$$\begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

(b) Write down a routine to perform the saxpy product using the (data,indices,indptr) storage scheme.[1]

(c) Write out the code for the transpose product $y = A^t x$ where A is stored in CSR format. Write a simple test program and confirm that your code computes the right thing.

One of the largest database of sparse matrices accessible online is the University of Florida Sparse Matrix Collection (`http://www.cise.ufl.edu/research/sparse/matrices/`).

4. Take any matrix (e.g. the real matrix `CurlCurl_0.mtx` or any other square matrix of full rank) and write a routine that reads it. Then, take a random vector b and use `scipy.sparse` functions to solve the corresponding linear system.

[1] Most of the iterative methods to compute eigenvalues are based on saxpy operations.