Numerik partieller Differentialgleichungen

Sparse matrices

Since the basis functions $\{\varphi_i\}_{i=1}^S$ have a local support, the stiffness matrix A is sparse[1]. In order to handle these matrices efficiently, there are several formats, as explained for

$$A = \begin{pmatrix} 1 & \cdot & \cdot & 2 & \cdot \\ 3 & 4 & \cdot & 5 & \cdot \\ 6 & \cdot & 7 & 8 & 9 \\ \cdot & \cdot & 10 & 11 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 12 \end{pmatrix}.$$

In principle, only the nonvanishing entries of the matrix and their positions are saved.

Compressed **S**parse **C**olumn (CSC)[2]:

The nonvanishing entries are stored (columnwise) in a vector a, the vector i contains the associated row indices and j contains the indicis in a at which a new column begins.

a	$\boxed{1}$	3	6	$\boxed{4}$	$\boxed{7}$	10	$\boxed{2}$	5	8	11	$\boxed{9}$	12	entries
i	1	2	3	2	3	4	1	2	3	4	3	5	rows
j	1	4	5	7	11	13							index in a

Compressed **S**parse **R**ow (CSR):

This format is analogous to CSC, but works rowwise.

a	$\boxed{1}$	2	$\boxed{3}$	4	5	$\boxed{6}$	7	8	9	$\boxed{10}$	11	$\boxed{12}$	entries
j	1	4	1	2	4	1	3	4	5	3	4	5	columns
i	1	3	6	10	12	13							index in a

Modified **S**parse **R**ow (MSR):

First, the vector a contains the diagonal elements of the matrix (these are typically nonzero) and afterwards, the remaining nonvanishing entries rowwise. The first indices in i encode the start of new rows in the vector a (similar to i in CSR). The later indices are the column indices of the associated entries (similar to j in CSR).

a	1	4	7	11	12	$*$	$\boxed{2}$	$\boxed{3}$	5	$\boxed{6}$	8	9	$\boxed{10}$	Einträge
i	7	8	10	13	14	14	4	1	4	1	4	5	3	Index in a

[1] The MATLAB functions for using sparse matrices can be shown with `doc sparfun`.
[2] This format is used by MATLAB.

Creation and modification of sparse matrices

While some operations are implemented efficiently for sparse matrices (e.g., matrix-matrix products and matrix-vector products), some other operations are inherently slow. In particular, this applies to the indexing of arbitrary entries or the successive filling of the matrix A with entries. The following will be very slow and inefficient.

```
% Create matrix
A = sparse(n,n);

for idx = 1:m
  % Compute some updates for entries of A
  ...

  % Now, we add these entries to A
  for k = 1:length(i)
    A(i(k), j(k)) = A(i(k), j(k)) + s(k);
  end
end
```

The reason is that in each update, MATLAB has to check whether the index already exists. Otherwise, it has to create the value and (almost) the entire structure of the sparse format has to rebuilt.

This should be better implemented via

```
% Create empty vectors
I = [];
J = [];
S = [];

for idx = 1:m
  % Compute some updates for entries of A
  ...

  % Now, we store these updates
  I = [I; i];
  J = [J; j];
  S = [S; s];
end
% Build the matrix A
A = sparse(I, J, S, n, n);
```

An even better solution would preallocate the vectors `I`, `J`, `S`.