Storing of sparse matrices

- variables:
  - `npoin` = number of matrix rows (and columns)
  - `nzero` = number of nonzero matrix entries
- arrays storring the matrix:
  - `grid%irow(1:grid%npoin+1)` – integer array, storing the starts of row in the sequence storing of entries,
  - `grid%icol(1:grid%nzero)` – integer array, storing the column indexes
  - `grid%sparse(1:grid%nzero)` – real array, storing the values of matrix entries
- matrix-vector multiplication

```
npoin = grid%npoin
ip => grid%ip(1:npoin, 1:2)
v2(:) = 0.
do is = 1, npoin
   if(ip(is, 1) >=0 ) then
      i = ip(is, 2)

      do ks = grid%irow(is), grid%irow(is+1) -1
         js = grid%icol(ks)
         if(ip(js, 1) >=0) then
            j = ip(js, 2)
            v2(i) = v2(i) + grid%sparse(ks) * v1(j)
         endif
      enddo

   endif
enddo
```

Write a simple code for the solution of problem: find $u : \Omega \to \mathbb{R}$ such that

$$-\Delta u(x) = g(x), \quad x \in \Omega, \tag{0.1}$$

$$u(x) = u_D(x), \quad x \in \partial\Omega_D, \tag{0.2}$$

$$\nabla u(x) \cdot \boldsymbol{n} = g_N(x), \quad x \in \partial\Omega_N, \tag{0.3}$$

where Ω is a domain in $\mathbb{R}^2$ with a boundary $\partial\Omega$ consisting of two disjoint parts $\partial\Omega_D$ and $\partial\Omega_N$, $g \in L^2(\partial\Omega_D)$, $g_N \in L^2(\partial\Omega)$ and u_D is a trace of some $u^* \in H^1(\Omega)$.

- use P_1-conforming FE
- the arising linear system solve by the Jacobi and BiCG method
- the stiffness matrix is computed and stored as sparse
- compare with the dense variant of implementation for various sizes of matrices (see tutorial7)

Use the code: `http://msekce.karlin.mff.cuni.cz/~dolejsi/Vyuka/NS_source/FEM/FEM-code3.tgz` link

- `mesh.f90` – reading the mesh from the file `triang`
- `matrix.f90` – create the stiffness matrix, solution of $\mathbb{A}x = b$
- `sol.f90` – setting of RHS and BC (input of data)
- `femP1.f90` – main code

- type of boundary set in `subroutine Read_mesh`, file `mesh.f90`
- array `ip(:, 1)` – type of mesh vertices: `> 0` – interior, `= 0` – Neumann, `< 0` – Dirichlet,
- array `ip(:, 2)` – index of vertex after removing Dirichlet nodes

Syntax of the code:

```
> ./femP1 <J/B> <S/D>
  <J> .. Jacobi
  <B> .. BiCG

  <S> .. sparse variant
  <D> .. dense variant
```

Basic tasks

1. study the code line by line, if something is unclear ask the teacher
2. compare the **dense** and **sparse** variants of the codes:

   `http://msekce.karlin.mff.cuni.cz/~dolejsi/Vyuka/NS_source/FEM/FEM-code.tgz` dense

   `http://msekce.karlin.mff.cuni.cz/~dolejsi/Vyuka/NS_source/FEM/FEM-code3.tgz` sparse
3. use Linux codes:
   - `diff file1 file1`
   - `meld file1 file1 &` – has to be installed
4. perform numerical experiments for increasing number of elements using
   (a) both solvers (Jacobi, BiCG)
   (b) dense and sparse variants

Further tasks

1. write a subroutine computing the error in the L^2-norm and H^1-seminorm (provided that the exact solution is known)
2. using computation on a sequence of meshes set the *experimental order of convergence*

More advanced tasks

1. develop an algorithm for the spase multiplication of the transpose matrix in the BiCG method (note that the actual implementation of BiCG works only for symmetric matrices!!!!)
2. test this subroutine with the code
3. in order to have a real nonsymmetric problem, solve the convection-diffusion equation

$$\begin{aligned} -\Delta u + \boldsymbol{b} \cdot \nabla u &= f \qquad \text{in } \Omega, \\ u_h &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{0.4}$$

where, e.g., $\boldsymbol{b} = (1, 0)^{\mathrm{T}}$.

The approximate solution is given by

$$\int_\Omega \nabla u_h \cdot \nabla \varphi_h \,\mathrm{d}x + \int_\Omega (\boldsymbol{b} \cdot \nabla u_h)\varphi_h \,\mathrm{d}x = \int_\Omega f \varphi_h \,\mathrm{d}x. \tag{0.5}$$

Let the approximate solution is $u_h = \sum_{j=1}^N u_j \varphi_j$ then we have

$$\sum_{j=1}^N u_j \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \,\mathrm{d}x + \sum_{j=1}^N u_j \int_\Omega (\boldsymbol{b} \cdot \nabla \varphi_j)\varphi_i \,\mathrm{d}x = \int_\Omega f \varphi_i \,\mathrm{d}x, \quad i = 1, \ldots, N. \tag{0.6}$$

If $\boldsymbol{b} = (1, 0)^{\mathrm{T}}$ then $(\boldsymbol{b} \cdot \nabla \varphi_j) = \frac{\partial \varphi_j}{\partial x_1}$.

BiCG algorithms for the solution of

$$\mathbb{A}\mathbf{x} = \boldsymbol{b}, \qquad \mathbb{A}^{\mathsf{T}}\mathbf{y} = \boldsymbol{c},$$

where $\mathbb{A}$ is given matrix, $\boldsymbol{b}$ and $\boldsymbol{c}$ are the given-right-hand sides, $\mathbf{x}_0$ and $\mathbf{y}_0$ are initial guess.

- One can put $\boldsymbol{c} := \boldsymbol{b}$ if any other better choice does not exist.
- If $\mathbb{A}$ is symmetric and $\boldsymbol{c} := \boldsymbol{b}$ then BiCG is equivalent to CG, only BiCG requires two times larger number of arithmetic operations.
- $\mathbb{P}$ is the preconditioned matrix in the following algorithm.

input $\mathbb{A}$, $\mathbf{x}_0$, $\mathbf{y}_0$, $\mathbb{P}$
$\boldsymbol{r}_0 = \boldsymbol{b} - \mathbb{A}\mathbf{x}_0$, $\boldsymbol{s}_0 = \boldsymbol{c} - \mathbb{A}^{\mathsf{T}}\mathbf{y}_0$,
$\boldsymbol{p}_0 = \mathbb{P}^{-1}\boldsymbol{r}_0$
$\boldsymbol{q}_0 = \mathbb{P}^{-\mathsf{T}}\boldsymbol{s}_0$
$\tilde{\boldsymbol{r}}_0 = \boldsymbol{p}_0$
for $k = 0, 1, \ldots$ **do**
  $\alpha_k = \frac{\boldsymbol{s}_k^{\mathsf{T}}\tilde{\boldsymbol{r}}_k}{\boldsymbol{q}_k^{\mathsf{T}}\mathbb{A}\boldsymbol{p}_k}$
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\boldsymbol{p}_k$, $\qquad \mathbf{y}_{k+1} = \mathbf{y}_k + \alpha_k\boldsymbol{q}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\mathbb{A}\boldsymbol{p}_k$, $\qquad \boldsymbol{s}_{k+1} = \boldsymbol{s}_k - \alpha_k\mathbb{A}^{\mathsf{T}}\boldsymbol{q}_k$
  $\tilde{\boldsymbol{r}}_{k+1} = \mathbb{P}^{-1}\boldsymbol{r}_{k+1}$, $\qquad \tilde{\boldsymbol{s}}_{k+1} = \mathbb{P}^{-\mathsf{T}}\boldsymbol{s}_{k+1}$
  $\beta_{k+1} = \frac{\boldsymbol{s}_{k+1}^{\mathsf{T}}\tilde{\boldsymbol{r}}_{k+1}}{\boldsymbol{s}_k^{\mathsf{T}}\tilde{\boldsymbol{r}}_k}$
  $\boldsymbol{p}_{k+1} = \tilde{\boldsymbol{r}}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$, $\qquad \boldsymbol{q}_{k+1} = \tilde{\boldsymbol{s}}_{k+1} + \beta_{k+1}\boldsymbol{q}_k$
end for