Scilab tools for PDE's : Application to time-reversal

Bruno Pinçon and Karim Ramdani
Corida team INRIA and Institut Elie Cartan
University Henri Poincaré
Nancy, France
bruno.pincon@iecn.u-nancy.fr
karim.ramdani@loria.fr

Outlines

1. introduction
2. sparse linear algebra tools
3. finite element tools
4. time-reversal in a 2D acoustic waveguide
5. conclusion

Introduction

While Scilab is rich in tools to deal with ordinary differential equations (ode's) it is less practical for partial differential equations (pde's) numerical experiments.

some good points:

- almost complete sparse matrix algebra support but:
  - the sparse solver (Sparse 1.3) is outperformed by more modern ones (umfpack, superlu,...);
  - some sparse operations (extraction/insertion) are slow.
- a routine for solving 1D pde's system (`bvode`) but nothing for 2D.

our goals:

- design a pde's toolbox for 2D (space) simulations such that:
  - it may be easy to use;
  - it may be adaptable;
  - it may be well integrated in scilab.
- improve some scilab sparse matrix operations (this is necessary to deal with high discretisations).

sparse matrix algebra 1: an interface onto a good sparse solver

We have developped a Scilab interface onto the umfpack v4-x sparse linear solver (Tim Davis). Freely available (SCISPT toolbox) at the url:

`http://www.iecn.u-nancy.fr/~pincon/scilab/scilab.html`.

main features:

```
  x   =   umfpack(A,"\",b)
LUp   =   umf_lufact(A)
  x   =   umf_lusolve(LUp, b [,A])
          umf_ludel([LUp])
```

A small benchmark (done on a Latitude D400 Dell laptop, umfpack using an ATLAS BLAS library)

matrix name	order	nnz	t_1	t_2	t_1/t_2
bcsstk24	3562	159910	22.59	0.26	87
bcsstk38	8032	355460	160.31	0.75	160
ex14	3251	65875	87.61	0.25	350
epb2	25268	175027	571.60	1.07	534
utm5940	5940	83842	170.43	0.29	594

t_1 current sparse solver
t_2 umfpack solver

Clearly these results show that it is time for Scilab to change its sparse solver !

sparse matrix algebra 2: improvement of some sparse operations

insertion of a full matrix in a sparse matrix: $A(ii, jj) = B$

- When B is small comparing to A it is interesting to try in place insertion:

```
n = 3000;
A = sprand(n,n,6/n);
// get the indices of nnz elements
[ij,v,mn]=spget(A);
timer();
for k = 1:size(ij,1), A(ij(k,1),ij(k,2)) = 1; end
t = timer()
```

old insertion way	66 s
in place insertion	0.48 s

- But when in place insertion is not possible (or not interesting), the insertion algorithm used by Scilab is too slow. We have written a new insertion code based on (scilab sparse format being row oriented):
  - fast copy of sparse rows;
  - carrefully assembling of a row built from a sparse and a full row.

a small benchmark:

```
A = sprand(10000,10000,0.001);
B = rand(500,500);
timer(); A(101:600,201:700) = B; t=timer();
```

old insertion code	9.2 s
new insertion code	0.1 s
Matlab insertion code	350 s
Matlab in place insertion code	0.42 s

sparse matrix extraction: $B = A(ii, jj)$

• When a row has many elements it is interesting to use a dichtomic search onto the column indices in place of a linear search (feature added in the scilab core);

• When the indexing domain is very large (for instance A is 20000×20000 and $ii = 1 : 10000$, $jj = 10001 : 20000$) and A have a low element density (that is A is a normal sparse matrix !) then the usual algorithm don't perform well. But it is possible to invert the search that is to do a loop on the sparse matrix elements and test if they are in the extraction region. We have written a new insertion code based on this idea.

Here is a small benchmark:

```
n = 20000;
A = sprand(n, n, 6/n);
timer(); C = A(1:n/2,n/2+1:n); t=timer();
```

old extraction code	2.33 s
new extraction code	0.04 s
Matlab extraction code	0.04 s

finite elements tools 1: boundary description and domain discretisation

Some remarks:

• For numerical experiments with pde's control problems it is important to be able to locate some regions (2D subdomain or 1D curve or even points) where a specific action (like a feedback) will be applied;

• From a first simulation it may be useful to be able to refine the mesh where needed.

We have chosen to use:

• triangle (J.R. Shewchuk) for meshing purpose, triangle is:
- fast;
- may be called through a C interface;
- can refine a previous mesh;
- may manage points / edges / triangles (regional) attributes;
- have many options.

• Scilab for the description (tlist) and the discretisation of the boundaries (and also "internal boundaries")

boundary description and domain discretisation: how it works

- A boundary is described from elementary border pieces:

```
S  = Seg(P1,P2, bdy_marker)
A  = Arc(C, theta1, theta2, orientation, bdy_marker)
Ci = Circle(C, r, orientation, bdy_marker)
U  = Ufunc(t1, t2, x_expr, y_expr, bdy_marker)
Sp = Spline(Pts, bdy_marker)
CSp= CSpline(Pts, bdy_marker)
```

- In fact the boundary is formed from one exterior closed contour and eventually with one or several interior closed contour(s). The function `MakeContour` take several (or only one) element border pieces and build a closed contour.

```
CC = MakeContour(S1, S2, S3, S4)
```

- The interior constrained curves (which are not part of the boundary) must be described only from elementary border pieces.

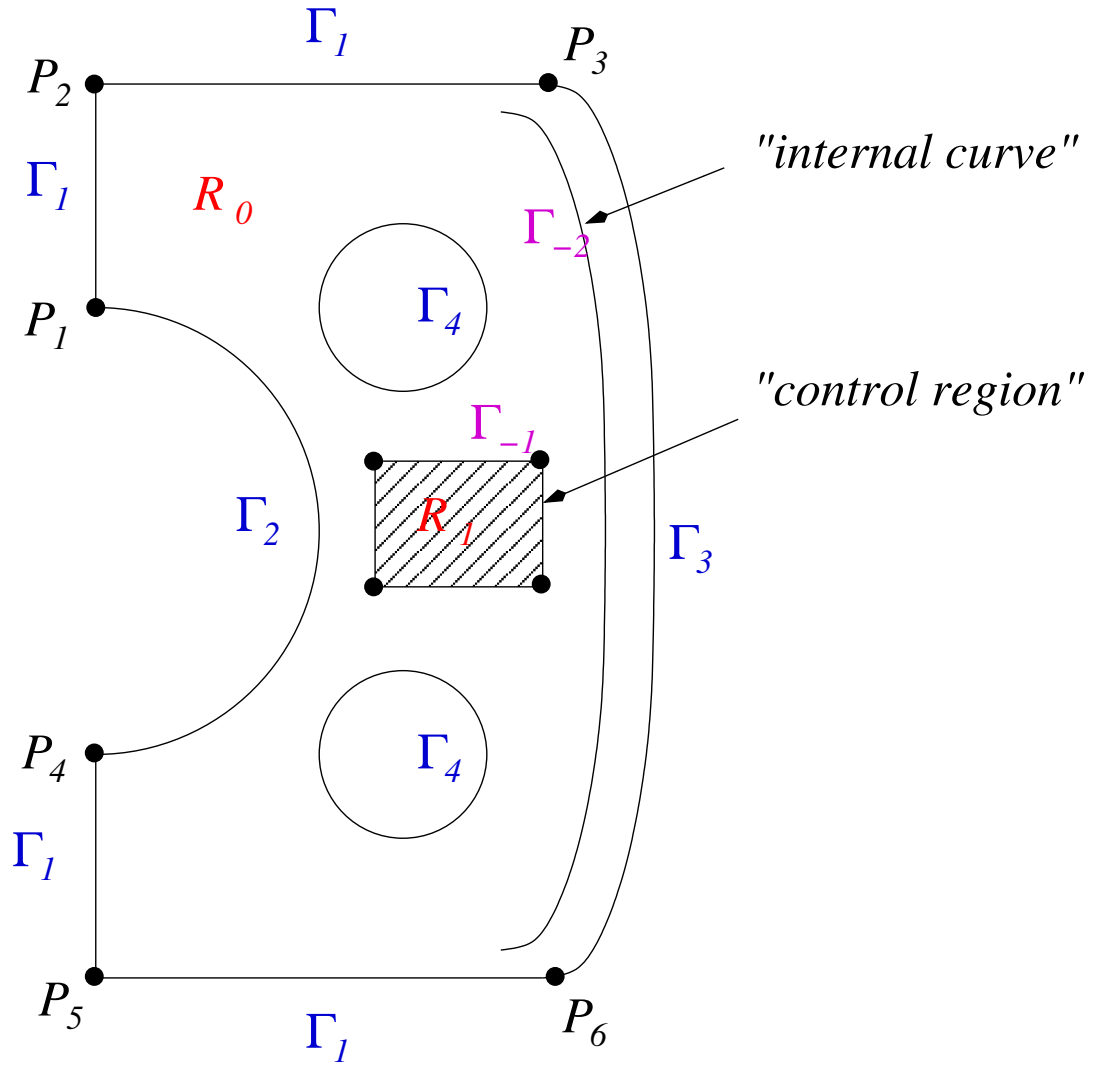
- The boundary (with the eventually interior constrained curves) is discretised with:

```
[PtB, EdB, MarkEdB, ne] = BoundaryDiscretisation(h, CCext, CCint1,.., Ebp1, Ebp2, ...)
```

- From the boundary edges (and also internal edges), the triangulation is got with:

```
[P, T, markT, E, T2T] = triangulate(PtB, EdB [, ne, Region]);
```

boundary description and domain discretisation: an example



the scilab script

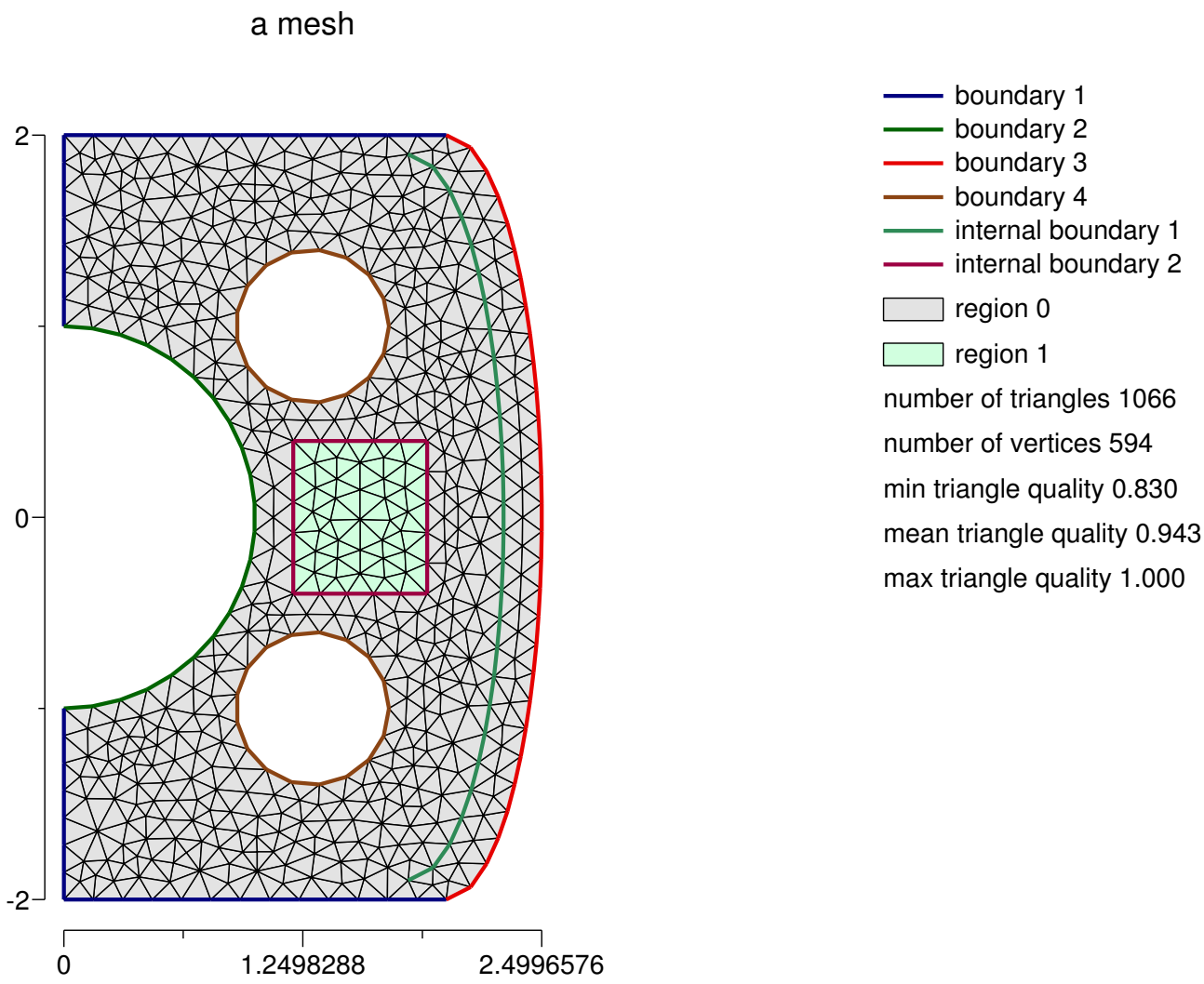
```
P1 = [0;1]; P2 = [0;2]; P3 = [2;2]; P4 = -P1;   P5 = -P2;   P6 = [2;-2];
C = [0;0]; C1 = [1.3;1]; C2 = [1.3;-1]; h = 0.15; r = 0.4;

S1 = Seg(P3,P2,1);
S2 = Seg(P2,P1,1);
A1 = Arc(C,1, -%pi/2, %pi/2, -1, 2);
S3 = Seg(P4,P5,1);
S4 = Seg(P5,P6,1);
T = Ufunc(-%pi/2,%pi/2, "x=2+0.5*cos(t)", "y=2*sin(t)", 3);
Ce = MakeContour(S1,S2,A1,S3,S4,T);
Ci1 = MakeContour(Circle(C1,r,-1,4));
Ci2 = MakeContour(Circle(C2,r,-1,4));

// internals constrained curves:
Ti = Ufunc(-%pi/2,%pi/2, "x=1.8+0.5*cos(t)", "y=1.9*sin(t)", -1);
P7 = [1.2;0.4]; P8 = [1.9;0.4];
P10 = [1.2;-0.4]; P9 = [1.9;-0.4];
S5 = Seg(P7,P8,-2); S6 = Seg(P8,P9,-2);
S7 = Seg(P9,P10,-2); S8 = Seg(P10,P7,-2);

[PtB, EdB, MarkEdB, ne] = BoundaryDiscretisation(h,Ce,Ci1,Ci2,Ti,S5,S6,S7,S8);
Region = [1.1  1.3 ;...    // x coordinate of a point in each region
          0    0   ;...    // y coordinate of a point in each region
          0    1   ;...    // region markers identifiers
          %nan %nan];      // wanted max size of the triangles in each region
[P, T, markT, E, T2T] = triangulate(PtB, EdB, ne, Region);
```

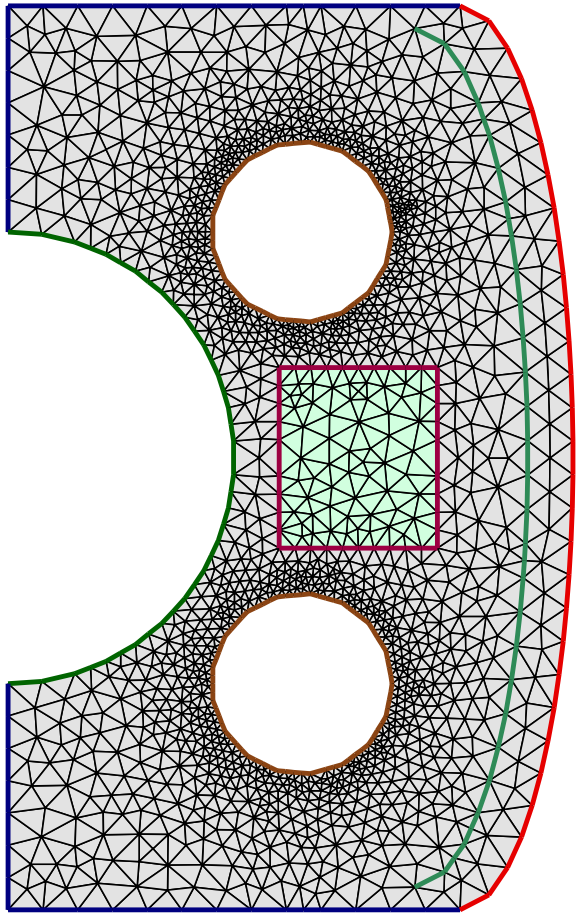
the resulting mesh

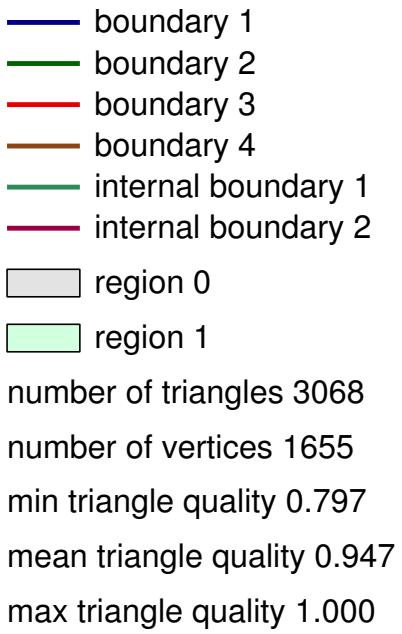


finite elements tools 2: mesh refinement

To refine the mesh one have to give an area constraint for each triangle:

```
[Pn, Tn, markTn, EdBn, MarkEdBn, En, T2Tn] = refine_mesh(P, T, markT, area_cstr,  EdB, MarkEdB);
```





finite elements tools 3: f.e.m. matrices and vector

Considering a problem of the form:

$$\begin{cases} -\nabla \cdot (K(x,y) \times \nabla u) + V(x,y) \cdot \nabla u + c(x,y)u &= f(x,y) & \text{in } \Omega \\ (K(x,y) \times \nabla u).n &= g(x,y) & \text{on } \Gamma_N \\ u &= h(x,y) & \text{on } \Gamma_D \end{cases}$$

discretized with the classic f.e.m.. Each matrix can be computed with:

```
sp = SparsePattern(E, np)
.......................
cM =  build_values_per_triangle(P, T, MarkT, dimM, ...
                                mkr1, fM1, mkr2, fM2, ...)
M  = assemb_fem_matrix( matrix_type, P, T, sp, cM);
.......................
A = M + others matrices
```

with matrix_type:
"stiffness"
"tensor_stiffness"
"mass"
"advection"

`build_values_per_triangle` is quite flexible, each "function" `fMx` may be a scalar or a scilab function or even a scilab list with a function and its parameters.

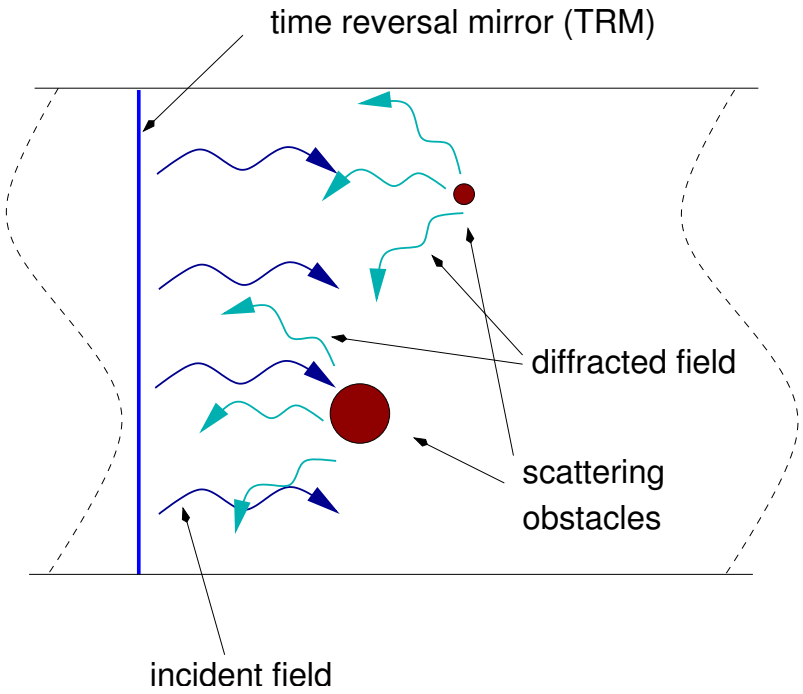
the right hand side vector is built with:

```
coefb =  build_values_per_triangle(P, T, MarkT, 1, mkr1, f1, mkr2, f2, ...)
b  = assemb_fem_rhs(P, T, coefb);
```

and the boundary conditions are introduced by:

```
b = put_neumann_conditions(b, P, EdB, MarkEdB, mkbn1, g1, mkbn2, g2, ...);
[J,valJ] = dirichlet_cond(P, EdB, MarkEdB, mkbd1, h1, mkbd2, h2, ...);
[A,b] = put_dirichlet_conditions(J, valJ, A, b);
```

an application: time-reversal in a 2D acoustic waveguide



a time-reversal mirror can:

- emit
- measure
- record
- achieve specific treatments of

acoustic waves.

(These mirrors have been developped in the team of M. Fink at LOA (ESPCI, France))

The main idea is to use the reversibility of the wave equation in a non dissipative medium to back-propagate signals to the sources that emitted them.

In particular time-reversal can be used to focus acoustic waves on scattering obstacles whose position is unknown.

Here we want to experiment the D.O.R.T. ("Decomposition of the Time-Reversal Operator") method in the case of an (infinite) 2D acoustic waveguide.

the D.O.R.T. method

1/ the TRM first emits an acoustic wave and measures the diffracted field.
2/ the measured field is then reversed and reemitted by the TRM which also measures the new diffracted field.

The Time-Reversal Operator T is the operator obtained by this 2 steps procedure.

If one works with time-harmonic waves ($U_T(x,t) = \mathcal{R}e(u_T(x)e^{-i\omega t})$) the reversing operation corresponds to a conjugation, that is, if S is the operator corresponding to the first step then:

$$T = \bar{S}S$$

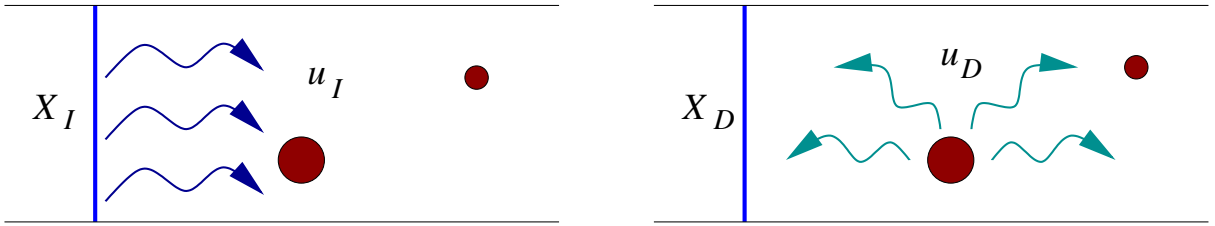
Under some assumptions (small scatterers, ...), it has been shown experimentally and mathematically proved (not currently for waveguides but only for free space) that:

- the number of non zero (or significant) eigenvalues of T is exactly the number of obstacles (eigenvalues modules being related to the size of the obstacles);
- the corresponding eigenvectors generate waves that focus selectively on the obstacles.

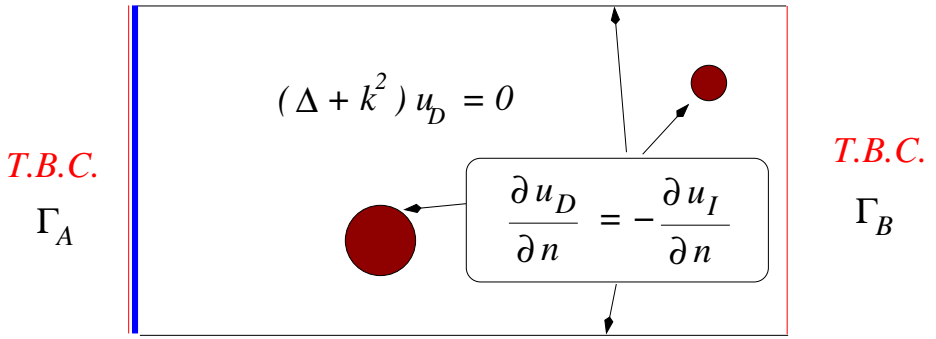
We want to test and (numerically) experiment the case of 2D waveguides with time-harmonic waves.

the mathematical model

The scattering matrix satisfies: $X_D = SX_I$



To obtain S, we solve a family of boundary value problems of the form: (each b.v.p. corresponding to an incident field u_I associated to one of the propagative modes of the waveguide)



which yields through a finite element discretisation to a matrix of the form:

$$A = \begin{array}{|c|c|c|l} \hline A_{11} & A_{12} & A_{13} & \text{others nodes} \\ \hline A_{21} & A_{22} & 0 & \text{nodes of } \Gamma_A \\ \hline A_{31} & 0 & A_{33} & \text{nodes of } \Gamma_B \\ \hline \end{array}$$

SPARSE FULL

some details on the implementation

Final steps:

- $T = \bar{S}S$;
- then compute the eigenelements of T
- check the expected focusing properties.

Remarks:

1/ the experimental setting imposes: $a << \lambda = \frac{2\pi}{k} << d$. Since the Helmholtz fem approximation requires 10 points per wavelength, the mesh size must be very small;

2/ to compute the matrix S the use of a fast solver is crucial;

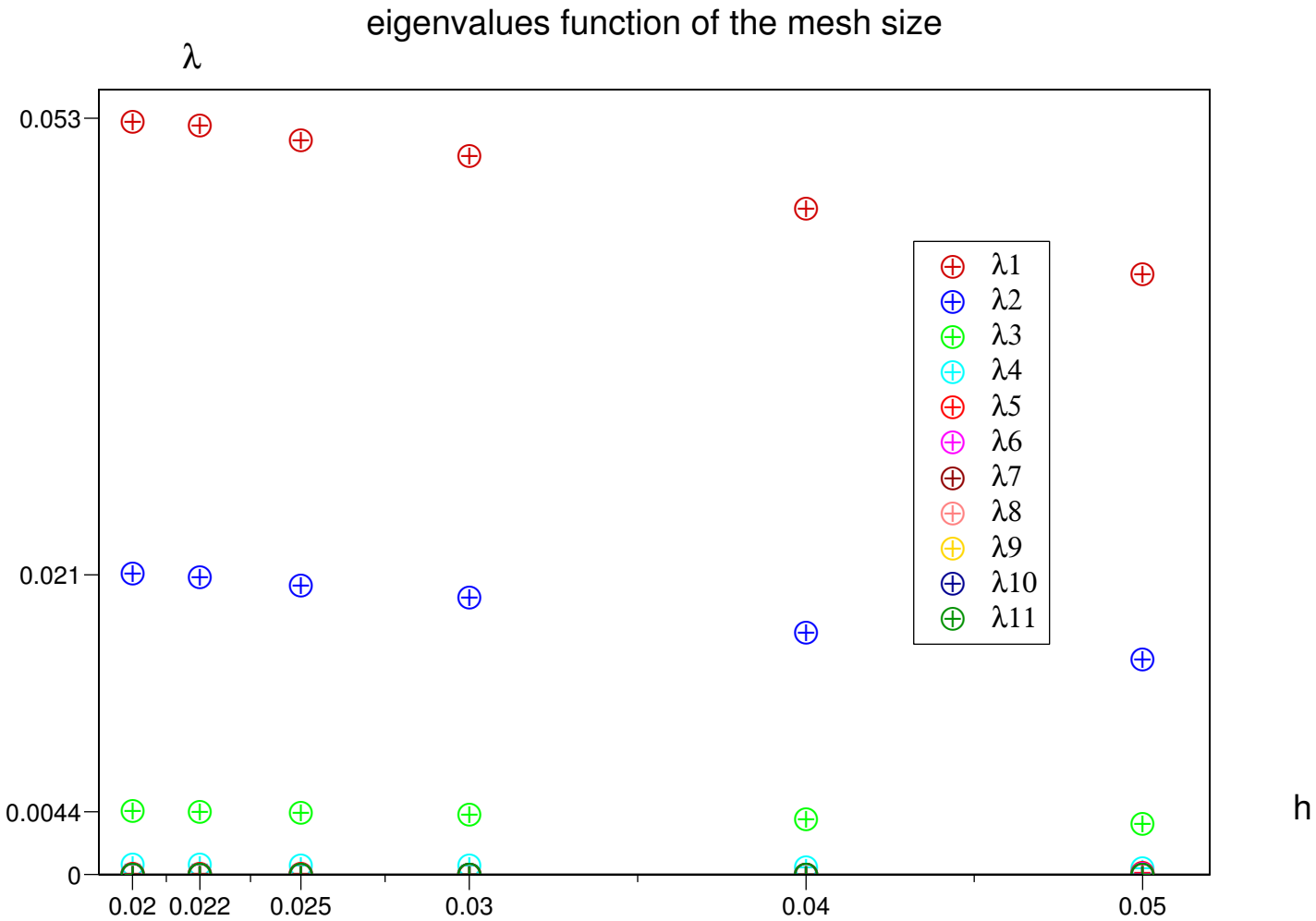
3/ our fast insertion algorithm (full inside sparse) is very useful here to take into account the TBC.

Our numerical test: uses the values, $a_1 = 0.13$, $a_2 = 0.1$, $\lambda = 1$, $d = 5.2$, $L = 5$. This gives 11 propagative modes for the wave guide. With $h = 0.02$, we got 229114 triangles, 115084 vertices, the different timings being:

0.6 s (bdy description)	1.56 s (triangulation)
3.6 s (mesh optim)	3.7 s (matrix build)
1.9 s (TBC)	16.8 s (factorisation)
12 s (computing of S)	

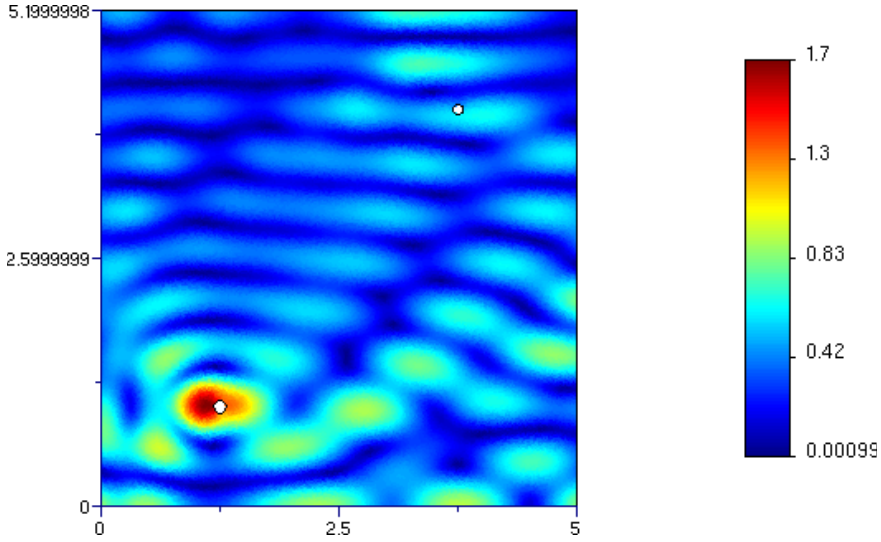
eigenvalues of the time reversal operator

We must found 2 significant eigenvalues ! (is λ_3 negligible ?)

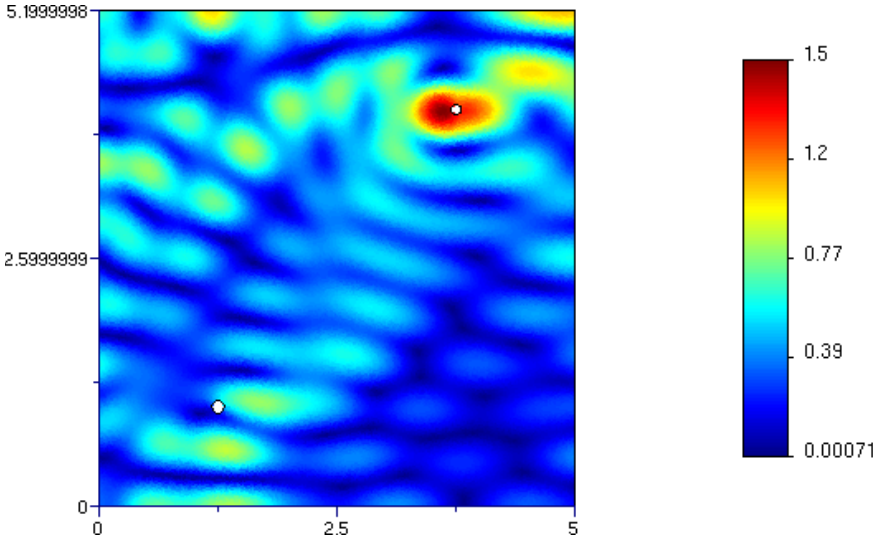


acoustic fields associated to the 4 first eigenvectors

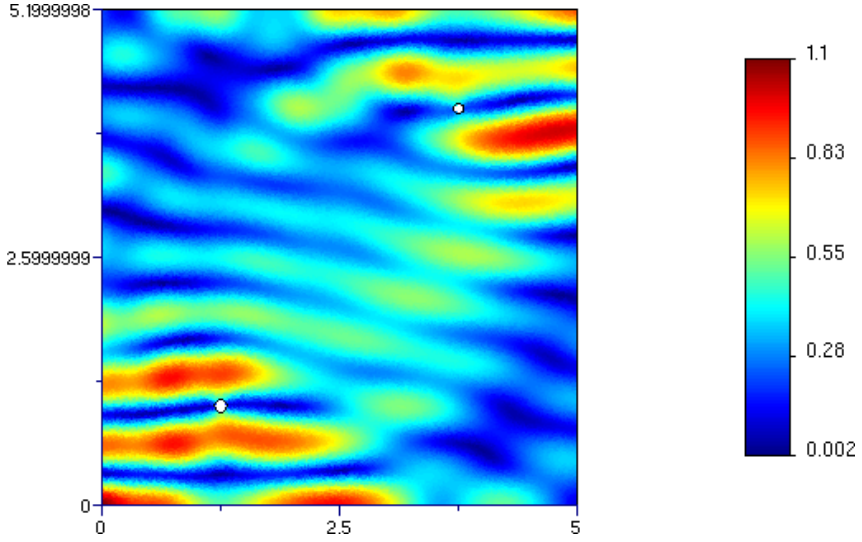
module of the total field associated with vp1



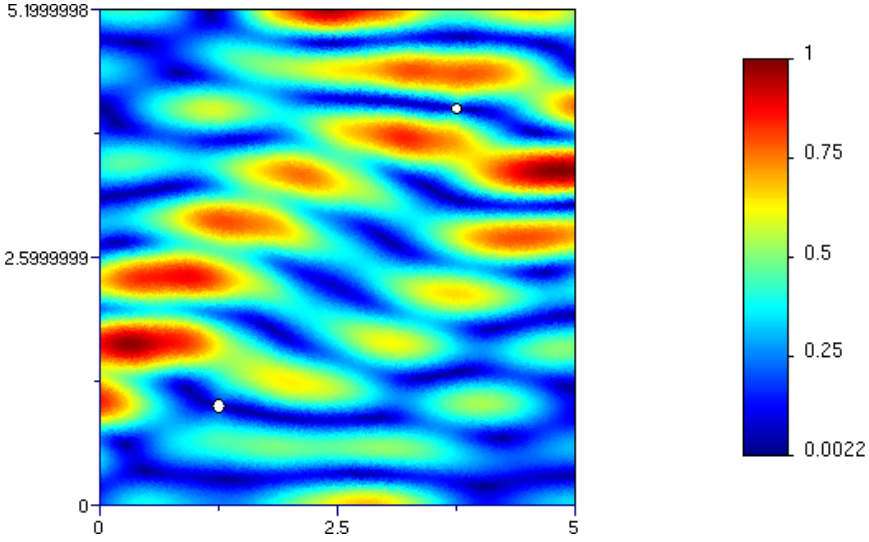
module of the total field associated with vp2



module of the total field associated with vp3



module of the total field associated with vp4



Conclusion

Having a (free) interpreted environment (which is not too slow !) to do pde's experiment is something useful !

For instance, at the beginning (and in the paper !) we have used 2 TRM but the focalisation results are not clear . . . and it stays many things to experience with one TRM !

Remarks:

1/ We think to complete progressively our pde's toolbox and make a first public release at the end of this year. You will find it at the url:

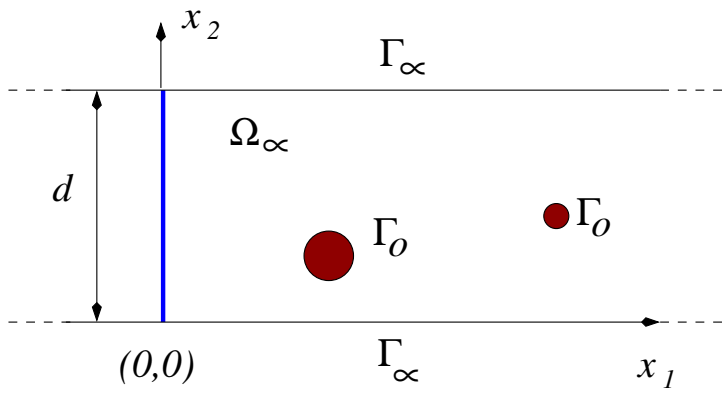
`http://www.iecn.u-nancy.fr/~pincon/scilab/scilab.html`

2/ Our code for sparse insertion and extraction will be soon integrated in Scilab.

3/ We hope that the umfpack solver will replace the scilab 's one.

the mathematical model

• the use of time-harmonic waves $u_T(x,t) = \mathcal{R}e(u(x)e^{-i\omega t})$ simplifies the wave equation into the Helmholtz equation: $\Delta u_T + k^2 u_T = 0, \quad k = \omega/c$ ($u_T = u_I + u_D$ being the total field);



• we use Neumann boundary conditions on the guide boundaries (Γ_∞) and on the obstacles (Γ_o): $\partial u_T/\partial n = 0$;

• our (abstract) mirror will generate an incident field (u_I) which is supposed to be a superposition of the propagative (toward the obstacles) modes of the wave guide, that is:

$$u_I = \sum_{n=0}^{N} a_I^n c_n(x_2) e^{i\beta_n x_1}, \quad \text{with} \quad \left\{ \begin{array}{l} c_n(x) = \sqrt{2/d}\cos(\lambda_n x), \quad (c_0(x) = \sqrt{1/d}) \\ \lambda_n = n\pi/d, \quad \beta_n = \sqrt{k^2 - \lambda_n^2} \end{array} \right.$$

• the diffracted field u_D is the outgoing solution of the problem:

$$\left\{ \begin{array}{rcll} \Delta u_D + k^2 u_D & = & 0 & \text{in } \Omega_\infty \setminus \bar{\mathcal{O}} \\ \partial u_D/\partial n & = & -\partial u_I/\partial n & \text{on } \Gamma_o \cup \Gamma_\infty \\ u_D & = & 0 & \text{on } \partial\Omega_\infty. \end{array} \right.$$

the mathematical model (continued)

• the fact that u_D is outgoing implies that $(x = (x_1, x_2))$:

$$u_D(x) = \begin{cases} \sum_{n\in\mathbb{N}} a_n^D e^{i\beta_n x_1} c_n(x_2) & \text{for } x_1 < 0 \quad (a_n^D = \int_0^d u_D(0,x_2)c_n(x_2)dx_2) \\ \sum_{n\in\mathbb{N}} b_n^D e^{-i\beta_n (x_1 - L)} c_n(x_2) & \text{for } x_1 > L \quad (b_n^D = \int_0^d u_D(L,x_2)c_n(x_2)dx_2) \end{cases}$$

• our TRM "prescribes" the coefficients $X_I = (a_I^0, a_I^1, \ldots, a_I^N)$ of the incident field and "measures" the coefficients $X_D = (a_D^0, a_D^1, \ldots, a_I^D)$ of the diffracted field. The scattering matrix S is the complex $(N+1)\times(N+1)$ matrix such that: $X_D = SX_I$ and the time reversal matrix is: $T = \bar{S}S$.

• to solve numerically the problem we must reduce it to problem posed on a bounded domain (Ω). We use on Γ_A and Γ_b transparent boundary conditions. They may be obtained by using the Dirichlet to Neumann operator (explicitely known here).