Introduction à l'implémentation de la méthode des éléments finis virtuels – Application au calcul des modes de résonance en acoustique

Marc Bakry avec Amadou Diallo

Merci à M. Aussal pour le financement
et à S. Pernet pour les discussions fructueuses

Séminaire d'Au Revoir

9 octobre 2020

Table des matières

Problème général

Soit le problème variationnel discret : trouver $w_h \in \mathcal{V}_h$, t.q. pour tout $v_h \in \mathcal{V}_h$

$$a(w_h, v_h) = b(v_h)$$

- $V_h \equiv V_h(\mathcal{T}_h)$
- $\mathcal{T}_h$ une discrétisation d'un domaine Ω.

Eléments finis "classiques" (2D)

- maillage triangulaire ou quadrangulaire
- fonctions de base explicites : $\mathcal{P}^0, \mathcal{P}^1, \mathrm{RT}_0$, etc.
  - $\hookrightarrow$ les matrices élémentaire se calculent facilement par intégration avec des règles de quadrature

Equation de Helmholtz

On considère l'équation de Helmholtz homogène dans $\Omega \subset \mathbb{R}^2$

$$
\begin{aligned}
& -\Delta p - \frac{\rho \cdot \omega^2}{\rho \cdot c^2} p = 0, \\
& \frac{\partial p}{\partial n} = 0 \qquad\qquad \text{sur } \partial\Omega
\end{aligned}
$$

qu'on réécrit sous la forme

$$
\begin{cases}
-\omega^2 \cdot \rho \cdot \mathbf{w} = -\nabla p & \text{dans } \Omega, \\
p = -\rho \cdot c^2 \cdot \operatorname{div} \mathbf{w} & \text{dans } \Omega, \\
\mathbf{w} \cdot n = 0 & \text{sur } \partial\Omega
\end{cases}
$$

et

$$
\mathbf{w} \in \mathcal{V} := \{\mathbf{v} \in H_{\mathrm{div}}(\Omega), \mathbf{v} \cdot n = 0 \text{ sur } \partial\Omega\}
$$

Problème aux valeurs propres

Mult. par fonction test $\mathbf{v}$ + Intégration par parties, etc.

$$\int_\Omega \operatorname{div}\mathbf{w}\cdot\operatorname{div}\mathbf{v} = \lambda \int_\Omega \mathbf{v}\cdot\mathbf{w}$$

reformulé en

$$\begin{aligned}\int_\Omega \operatorname{div}\mathbf{w}\cdot\operatorname{div}\mathbf{v} + \int_\Omega \mathbf{v}\cdot\mathbf{w} &= (\lambda+1)\int_\Omega \mathbf{v}\cdot\mathbf{w},\\ a(\mathbf{w},\mathbf{v}) + b(\mathbf{w},\mathbf{v}) &= (\lambda+1)\, b(\mathbf{w},\mathbf{v})\end{aligned}$$

- moins de modes parasites
- H_{div}– elliptique

On pose encore, pour $E \in \mathcal{T}_h$,

$$a^E(.,.) \quad \text{et} \quad b^E(.,.)$$

les formes bilinéaires locales.

Discrétisation "éléments finis virtuels"

- discrétisation en **polygones étoilés quelconques**
- si discrétisation triangulaire ou quadrangulaire : éléments de Raviart-Thomas …
- …mais sur un polygone quelconque ?

$\Rightarrow$ éléments finis "virtuels"

- fonctions de base pas connues explicitement …
- …mais on **suppose** qu'elles ont des propriétés sympathiques.

Autrement dit : on fait un choix *éclairé* d'approximation, puis on prouve que c'est le bon (mais ce n'est pas l'objet ici).

Recette

Soit $E \in \mathcal{T}_h$ un élément du maillage, on choisit

$$\mathcal{V}_h^E := \{\mathbf{v}_h \in H_{\mathrm{div}}(E), (\mathbf{v}_h \cdot n) \in \mathbb{P}_k(e), \forall e \subset \partial E, \\ \operatorname{div} \mathbf{v}_h \in \mathbb{P}_k(E), \operatorname{rot} \mathbf{v}_h = 0 \text{ dans } E)\}$$

un espace d'approximation avec

- $\mathbb{P}_k(e)$ polynôme d'ordre k sur les arêtes de E,
- $\mathbb{P}_k(E)$ polynôme d'ordre k sur E.

L'idée : *travailler autant que possible avec des polynômes*

- facile à exprimer à n'importe quel ordre
- intégration exacte par quadrature sur des polygones.

Degrés de liberté

Deux "classes" de degrés de liberté

- "d'arête"

$$\int_e (\mathbf{v}_h \cdot n) \cdot q \cdot ds, \quad \forall q \in \mathbb{P}_k(e), \quad \forall e \subset \partial E$$

- "intérieurs"

$$\int_E \mathbf{v}_h \cdot \nabla p, \quad \forall p \in \mathbb{P}_k(E) \backslash \mathbb{P}_0(E)$$

Remarques

- pour $k = 0$, pas de dofs intérieurs !
- nb. dofs entièrement déterminé par k
  - $\hookrightarrow$ pour $k = 1$: 2 dofs intérieurs (il faut enlever le polynôme constant de la base), 2 dofs par arête

Calcul des matrices élémentaires 1/

- $a^E(\mathbf{w}_h, \mathbf{v}_h)$ se calcule facilement car $\operatorname{div} \mathbf{w}_h$ est un $\mathbb{P}_k(E)$
- comment faire pour $b(\mathbf{w}_h, \mathbf{v}_h)$?

  $\hookrightarrow$ on projette $b^E(.,.)$ sur un sous-espace local *explicite* $\widehat{\mathcal{V}_h^E}$.

Soient l'espace local de projection

$$\widehat{\mathcal{V}_h^E} := \nabla(\mathbb{P}_{k+1}(E)) \subset \mathcal{V}_h^E, \qquad E \in \mathcal{T}_h$$

et le projecteur

$$\Pi^E : \mathcal{V}_h^E \to \widehat{\mathcal{V}_h^E},$$
$$b^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{m}) = 0, \qquad \mathbf{w}_h \in \mathcal{V}_h^E, \forall \mathbf{m} \in \widehat{\mathcal{V}_h^E}$$

Calcul des matrices élémentaires 2/

On a donc

$$\begin{aligned} b^E(\mathbf{w}_h, \mathbf{v}_h) &= b^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h + \Pi^E \mathbf{w}_h, \mathbf{v}_h - \Pi^E \mathbf{v}_h + \Pi^E \mathbf{v}_h), \\ &= b^E(\Pi^E \mathbf{w}_h, \mathbf{v}_h) + b^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{v}_h), \\ &= b^E(\Pi^E \mathbf{w}_h, \Pi^E \mathbf{v}_h) + b^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{v}_h - \Pi^E \mathbf{v}_h). \end{aligned}$$

$\hookrightarrow$ on sait calculer le 1^er terme ci-dessus !

On va approximer le second terme !

On introduit $S^E(.,.)$ et $\{c, C\}$ tels que

$$c \cdot b^E(\mathbf{w}_h, \mathbf{w}_h) \leq S^E(\mathbf{w}_h, \mathbf{w}_h) \leq C \cdot b^E(\mathbf{w}_h, \mathbf{w}_h),$$

On a finalement

$$b^E(\mathbf{w}_h, \mathbf{v}_h) \approx b^E(\Pi^E \mathbf{w}_h, \Pi^E \mathbf{v}_h) + S^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{v}_h - \Pi^E \mathbf{v}_h)$$

Calcul des matrices élémentaires 3/

On a donc

- un terme de ***consistance*** : $b^E(\Pi^E \mathbf{w}_h, \Pi^E \mathbf{v}_h)$
  - $\hookrightarrow$ il permet d'assurer qu'on est consistant avec $b^E(\mathbf{w}_h, \mathbf{v}_h)$
- une terme de ***stabilité*** : $S^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{v}_h - \Pi^E \mathbf{v}_h)$
  - $\hookrightarrow$ il compense la perte d'information liée à la projection

Par un *choix* éclairé,

$$S^E(\mathbf{w}_h - \Pi^E \mathbf{w}_h, \mathbf{v}_h - \Pi^E \mathbf{v}_h) = \sigma_E \sum_k \mathrm{dof}_k(\mathbf{w}_h - \Pi \mathbf{w}_h) \cdot \mathrm{dof}_k(\mathbf{v}_h - \Pi \mathbf{v}_h)$$

- $\sigma_E \in \mathbb{R}^+$,
- $\mathrm{dof}_k(\mathbf{w}_h)$: évaluation du k-ième dof local pour $\mathbf{w}_h$.

Table des matières

Choix des bases polynômiales *locales*

Un bon choix permet de s'épargner *beaucoup* de prise de tête

- base pour $\mathbb{P}_{k+1}(E)$

$$
\begin{gathered}
p_0(x,y) = 1,\\
p_\alpha(x,y) = \frac{(x-x_E)^{\alpha_1}\cdot(y-y_E)^{\alpha_2}}{h_E^2} + C_\alpha,\\
0 < \alpha_1+\alpha_2 \leq k+1
\end{gathered}
$$

avec $C_\alpha \in \mathbb{R}$ tel que $\int_E p_\alpha = 0$, (x_E, y_E) le *centroïde*, h_E le diamètre et $|E|$ l'aire de E.

- base pour $\mathbb{P}_k(e), \forall e \subset \partial E$: les **polynômes de Legendre**

$$
q_e^0(x) = 1, \qquad \|q_e^i\|_{\infty,e} = 1, \qquad \int_e q_e^i \cdot q_e^j = \delta_{ij}.
$$

Propriétés des fonctions de base

- fonctions de base "d'arête" φ_e^i

$$\varphi_e^i \in \mathcal{V}_h^E,$$
$$\int_{e_m} (\varphi_e^i \cdot n) \cdot q_m^j = \delta_{em}\delta_{ij},$$
$$\int_E \operatorname{div} \varphi_e^i \cdot p_\alpha = 0, \qquad \alpha \geq 1$$

- fonctions de base "interieures"

$$\tilde{\varphi}^s \in \mathcal{V}_h^E,$$
$$\tilde{\varphi}^s|_{\partial E} \cdot n = 0,$$
$$\int_E (\operatorname{div} \tilde{\varphi}^s) \cdot p^r = \delta_{sr}, \qquad r \geq 1$$

Calcul de $\operatorname{div} \varphi_e^i$

On a

$$\boxed{\operatorname{div} \varphi_e^i = \frac{\delta_{i0}}{|E|}}.$$

Preuve :

- la projection de $\operatorname{div} \varphi_e^i$ sur les p^s est nulle pour $s \geq 1$,
- donc $\operatorname{div} \varphi_e^i$ est une constante.
- De plus

$$\int_E \operatorname{div} \varphi_e^i = \operatorname{div} \varphi_e^i \int_E 1 = \sum_{m=1}^{N_E} \int_{e_m} (\varphi_e^i \cdot n) \cdot q_m^0 = \delta_{i0}.$$

- Et conclusion.

Calcul de div $\tilde{\varphi}^s$

Par hypothèse, div $\tilde{\varphi}^s \in \mathbb{P}_k(E)$ donc on peut calculer sa projection sur la base des p_α

$$\operatorname{div} \tilde{\varphi}^s = \sum_{\alpha \geq 1} f_\alpha \cdot p_\alpha$$

On montre facilement qu'il suffit de résoudre le système

$$\begin{pmatrix} \ddots & \cdots & \iddots \\ \vdots & \{\int_E p_\alpha p_\beta\} & \vdots \\ \iddots & \cdots & \ddots \end{pmatrix} \cdot \begin{pmatrix} f_1^s \\ \vdots \\ f_{\tilde{N}}^s \end{pmatrix} = \begin{pmatrix} \vdots \\ \delta_{\beta s} \\ \vdots \end{pmatrix}$$

grâce aux propriétés de $\tilde{\varphi}^s$.

Un mot sur S^E et Π^E

- les dofs sont lagrangiens donc pour une fonction de base φ^i

$$\mathrm{dof}_k(\varphi^i) = \delta_{ki}$$

- la matrice élémentaire $\mathbf{S}^E$ pour S^E est donc

$$\mathbf{S}^E = \sigma_E \cdot (\mathbf{F}^T \cdot \mathbf{F})$$

avec

$$\mathbf{F} = \mathbf{I}_3 - \mathbf{D} \cdot \Pi, \qquad \mathbf{D}_{i\alpha} = \mathrm{dof}_i(\nabla p_\alpha)$$

et Π est la matrice de la projection sur $\widehat{\mathcal{V}_h^E}$ telle que

$$\mathbf{P} \cdot \Pi = \mathbf{B}$$

avec

$$\mathbf{P}_{\alpha\beta} = b^E(\nabla p_\alpha, \nabla p_\beta), \qquad \mathbf{B}_{\alpha i} = b^E(\varphi^i, \nabla p_\alpha).$$

Calcul de la projection Π sur la base locale des ∇p_α

Matrice du projecteur (classique)

$$\mathbf{P}_{\alpha\beta} = \int_E \nabla p_\alpha \cdot \nabla p_\beta$$

Second membre (par intégration par parties)

- Cas "arête" (on se souvient que $\varphi_e^i \cdot n \in \mathbb{P}_k(\partial E)$)

$$-\int_E \operatorname{div} \varphi_e^i \cdot p_\alpha + \int_e (\varphi_e^i \cdot n) \cdot p_\alpha = 0 + \int_e q_e^i \cdot p_\alpha$$

car $\varphi_e^i \cdot n = \sum_\alpha g^\alpha \cdot q_e^\alpha$ avec $g_\alpha = \delta_{i\alpha}$

- Cas "intérieur" (on se souvient que $\operatorname{div} \varphi \in \mathbb{P}_k(E)$)

$$\ldots = -\int_E \operatorname{div} \varphi^s \cdot p_\alpha = -\sum_\beta f_\beta \int_E p_\beta \cdot p_\alpha$$

Calcul de la matrice élémentaire pour $a(\mathbf{w}_h, \mathbf{v}_h)$

Trois combinaisons de fonctions de base

- Cas "arête – arête" : trivialement,

$$\int_E \operatorname{div} \varphi_{e_l}^i \cdot \operatorname{div} \varphi_{e_m}^j = \frac{\delta_{i0}\delta_{j0}}{|E|}$$

- Cas "arête – intérieur" :

$$\int_E \operatorname{div} \varphi_{e_l}^i \cdot \operatorname{div} \varphi^s = \frac{1}{|E|} \int_E \operatorname{div} \varphi^s = \int_{\partial E} \tilde{\varphi}^s \cdot n = 0$$

- Cas "intérieur – intérieur" : projection sur les $\{p_\alpha\}$, puis intégration par quadrature !

  $\hookrightarrow$ en fait on peut réutiliser la matrice des $\left\{ \int_E p_\alpha p_\beta \right\}$ vue précédemment !

<u>**Moralité :**</u> pas de difficulté ici !

Calcul de la matrice élémentaire pour $b(w_h, v_h)$

Rappel des étapes principales :

1. Calcul de la projection Π sur $\widehat{\mathcal{V}_h^E}$
2. Calcul de $\displaystyle\int_E \Pi\mathbf{w}_h \cdot \Pi\mathbf{v}_h$
   - $\hookrightarrow$ calcul de la matrice du projecteur
   - $\hookrightarrow$ un second membre par fonctions de base
   - $\hookrightarrow$ on obtient la *matrice de projection* $\Pi \Leftrightarrow$ matrice de passage "base globale" $\rightarrow$ "locale"
3. Calcul de $\displaystyle\sigma_E \sum_k \mathrm{dof}_k(\mathbf{w}_h - \Pi\mathbf{w}_h) \cdot \mathrm{dof}_k(\mathbf{v}_h - \Pi\mathbf{v}_h)$
   - $\hookrightarrow$ $\mathrm{dof}_k(\varphi_l) = \delta_{kl}$ (dof lagrangien)
   - $\hookrightarrow$ $\mathrm{dof}_k(\Pi\varphi_l)$: d'abord une projection sur la base locale, puis "relèvement" sur la base globale (évaluation du degré de liberté pour les éléments de la base locale)

Implémentation !

- matrice élémentaire pour $k = 0$
- en `Julia` (`https://julialang.org`)
  - ↪ langage récent (8 août 2018 pour la v1.0.0)
  - ↪ compatible UTF-8, on peut écrire "σ" au lieu de "`sigma`"
  - ↪ bibliothèque standard de haut niveau
  - ↪ pensé pour remplacer le `Fortran`
  - ↪ langage compilé (mais c'est un peu caché à l'utilisateur)
  - ↪ *très* performant (comme du `C` en fait)

Calcul numérique des intégrales

Pour l'intégrale sur le polygone

- polygone *étoilé* $\Rightarrow$ subdivision en triangles tq $E = \bigcup_i T_i^E$
- $\displaystyle\int_E \square = \sum_i \int_{T_i^E} \square$

Intégrale sur chaque triangle

- on manipule des polynômes de degré au plus 1 !
- évaluation au centre, puis multiplication par l'aire

Intégrale sur les arêtes

- comme pour les triangles !

(Exemple de) Code source – Matrice élémentaire

- calcul des données caractéristiques du polygone
- calcul de la matrice div–div

```
# verts : coordonnees des noeuds rangees en colonnes
function melem(verts::Array{Float64,2})
    # nombre de noeuds
    nvtx = size(verts,2)

    # 0. Preliminaires
    E  = polygonArea(verts)     # aire
    xE = polygonCentroid(verts) # centroide
    hE = polygonDiameter(verts) # diametre

    # 1. Matrice de divergence
    DIV = ones(Float64, nvtx, nvtx) / E
```

Calcul de la projection

- terme de gauche pour le calcul de la projection

```julia
# 2. membre de gauche pour calcul matrice de projection
gma   = getGradientMonomialsP1(xE, hE)
ref2d = quadGL2d(1) # quadrature sur triangle a un point

Pab = zeros(Float64, 2, 2)
for a=1:2
    for b=1:2
        # renvoie un vecteur de fonctions permettant
        # de calculer le gradient des 3 fonctions
        # de base. Comme on est en P1, la premiere
        # renvoie le vecteur nul.
        Pab[a,b] = integratePolygon(verts, xE, ref2d,
            x -> dot(gma[a+1](x),gma[b+1](x)))
    end
end
```

Calcul de la projection

- second membre pour le calcul de la projection

```
# vecteur de fonctions permettant le calcul des
# monomes de degre 1
ma = getMonomialsP1(xE, hE)
Cs = (-1/E)*integratePolygon(verts, xE, ref2d, ma)
ma = getMonomialsP1(xE, hE, Cs) # de moyenne nulle
#
rhs   = zeros(Float64, 2, nvtx)
for i=1:nvtx
    ip = mod(i, nvtx) + 1          # noeud suivant
    ev = verts[:,ip] - verts[:,i] # tangente
    le = norm(ev)                  # longueur de l'arete
    for a=1:2
        rhs[a,i] = ma[a+1]((verts[:,ip] + verts[:,i])/2)
    end
end
# calcul de la matrice de projection
P = Pab \ rhs
```

Matrice des dofs

- on va quand même utiliser une quadrature d'arête, mais elle est inutile ici !

```
# 3. matrice d'evaluation des dofs
D = zeros(Float64, nvtx, 2)
# quadrature 1d a 2 points sur [0,1]
ref1d = quadGL1d(2, 0., 1.)
# Di,a = int{(\nabla ma . n)*q^i}
for i=1:nvtx
    ip = mod(i, nvtx) + 1          # noeud suivant
    ev = verts[:,ip] - verts[:,i] # tangente
    le = norm(ev)                  # longueur de l'arete
    nv = e_v/le
    nv = [nv[2], -nv[1]]        # normale a l'arete
    for a=1:2
        D[i,a] = integrate(ref1d,
            t -> dot(gma[a+1](verts[:,i] + t*ev,nv))*le
    end
end
```

Fin de l'assemblage

- assemblage de la matrice de masse

  ↪ on prend $\sigma = 1$, ça suffit ici

```
  # 4. Assemblage du terme de masse
  sigma     = 1.
  Id        = Matrix(1.0I, nvtx, nvtx) # matrice identite
  F         = Id - D*P
  MASS_cons = P' * (Pab*P)         # consistance
  MASS_stab = sigma*(F' * F)       # stabilite
  MASS      = MASS_cons + MASS_stab

  # 5. On retourne les deux matrices elementaires
  return DIV, MASS
end
```

- assemblage final comme en FEM !
- conditions aux limites comme en FEM !

Table des matières

Calcul des valeurs propres acoustiques

Rappel : on résout le problème

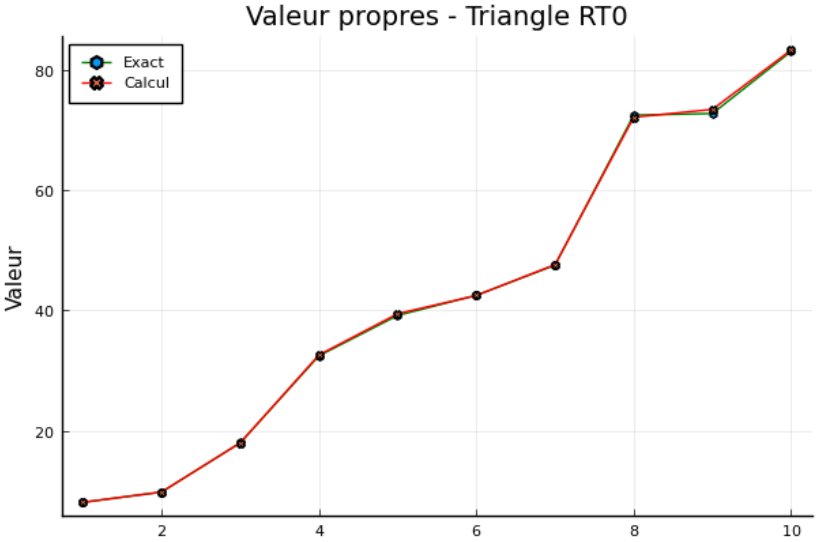
$$(\mathbf{A} + \mathbf{B}) \cdot V = (\lambda + 1) \cdot \mathbf{B} \cdot V$$

pour tous les couples valeur/vecteur propre $\{\lambda, V\}$ possibles.

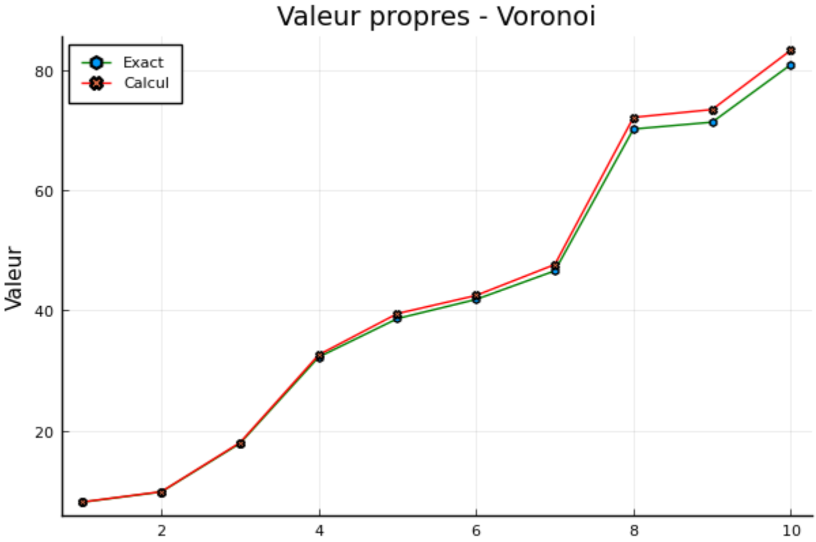
- solveur exact
- deux types de maillages
  - ↪ éléments convexes mixtes (sur une base d'hexaèdres)
  - ↪ éléments non convexes
- plaque de dimension $(a = 1, b = 1.1)$

$$\lambda_{mn} = \pi^2 \left(\left(\frac{m}{a} \right)^2 + \left(\frac{n}{b} \right)^2 \right), \qquad m + n \neq 0$$

Comparaison avec solution analytique 1/



Comparaison avec solution analytique 2/



Comparaison avec solution analytique 3/

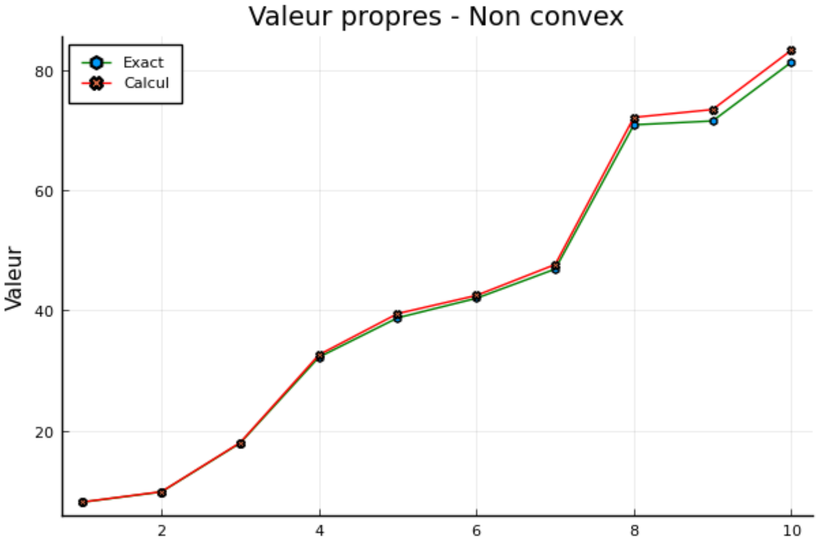


Illustration de quelques vecteurs propres 0/

- affichage du champ de pression

$$p = -\operatorname{div} \mathbf{w}$$

Illustration de quelques vecteurs propres 1/

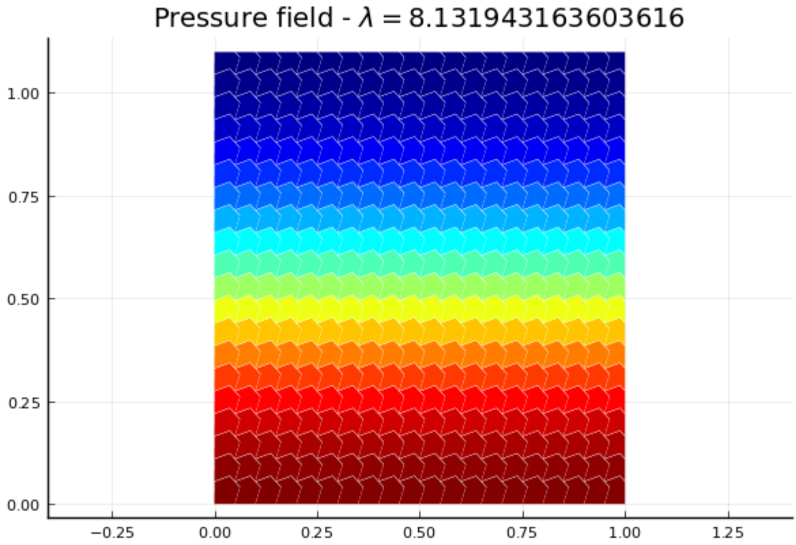


Illustration de quelques vecteurs propres 2/

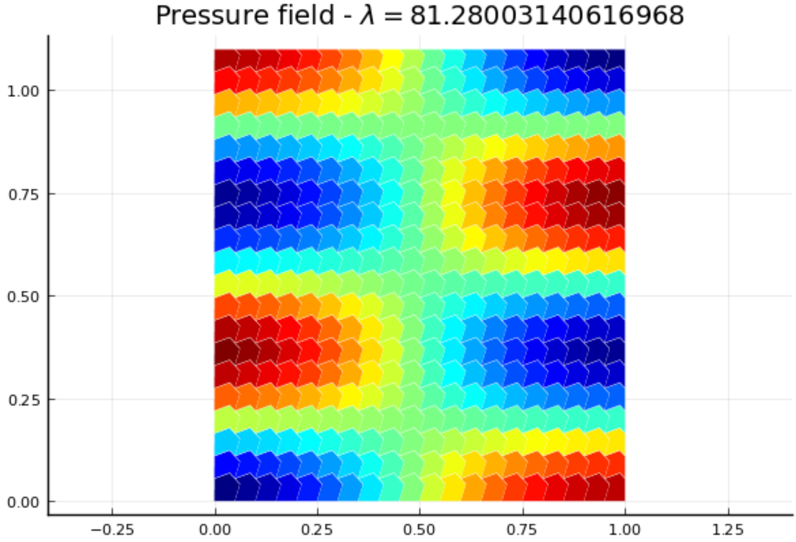
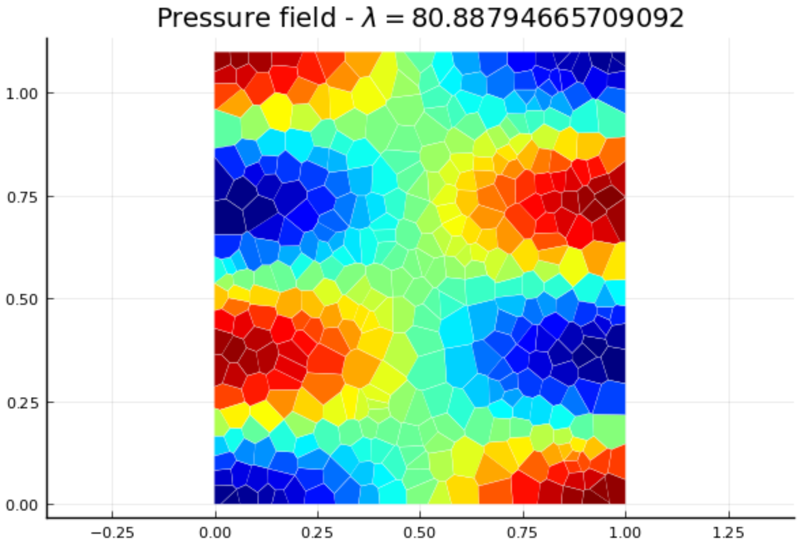


Illustration de quelques vecteurs propres 3/



Quelques mots sur l'implémentation

- le "code exemple" ci-avant peut être encore "compressé"
- matrice élémentaire : pas une simple intégration des fonctions de base ...
- ...mais pas très compliqué non plus.
- en `Julia` : on peut écrire des boucles, créer des matrices, les inverser avec `\`, etc.
- une fois la matrice élémentaire calculée, pas de différence avec un code FEM "classique"
  - ↪ à vous de choisir/construire votre connectique.
  - ↪ pour ceux qui ont déjà implémenté une EFIE Maxwell en BEM, c'est la même connectique.

Faire fonctionner l'exemple

1. code ici : `git clone https://github.com/marcbakry/vem_acoustics.git`
2. récupérer Julia : `https://julialang.org/downloads/`
3. ouvrir l'interpréteur et installer les dépendances
   ```
   Appuyer sur la touche "]" (fait apparaître "pkg>"), puis
   pkg> add Plots Arpack
   ```
4. naviguer dans le répertoire où est le `main`, et exécuter
   ```
   Appuyer sur la touche ";" (fait apparaître "shell>"), puis
   shell> cd versLeDossierMain
   julia> include("exeEigs.jl") # lancer l'exécution
   ```

Remarque : la première exécution peu être *très* lente

- compilation des dépendances et du code
- et ce à chaque fermeture/réouverture de l'interpréteur

⇒ C'est un des gros défaut de Julia (pour le moment ?).

Conclusion sur la VEM

- on n'a *jamais* explicité les fonctions de base
  - $\hookrightarrow$ cool, non ?
- on n'a fait que manipuler des *polynômes*
- ça fonctionne *comme la FEM classique* !
  - $\hookrightarrow$ même s'il y a un terme de stabilité
- on aurait pu projeter sur $\nabla\mathbb{P}_{k+2}(E)$
  - $\hookrightarrow$ ça donne implicitement les RT_k (sur triangle et quadrangle)
- monter en ordre est (relativement) trivial !
- on peut traiter facilement des maillages non-conformes
  - $\hookrightarrow$ tout est polygone
- le schéma pour la VEM présenté ici est très général
- ça fonctionne de la même manière en 3D, mais on ne l'a pas fait

Merci pour votre attention !

Pour cette présentation

- L. Beirão da Veiga, D. Mora, G. Rivera & R. Rodríguez, *A virtual element method for the acoustic vibration problem*, `arXiv:1601.04316`

Autres

- L. Beirão da Veiga, F. Brezzi, L. D. Marini & A. Russo, *The Hitchhiker's Guide to the Virtual Element Method*, Math. Models Methods Appl. Sci. **24** (8), 2014, pp. 1541–1573
- O. J. Sutton, *The virtual element method in 50 lines of* MATLAB, Numer. Algor. **75**, 2017, pp. 1141–1159

Calcul de l'aire

```
function polygonArea(vtx)
    area = 0.
    nvtx = size(vtx,2)
    for i=1:nvtx
        ip = mod(i,nvtx) + 1
        area += vtx[1,i]*vtx[2,ip] - vtx[2,i]*vtx[1,ip]
    end
    return 0.5*abs(area)
end
```

Calcul du diamètre

```
function polygonDiameter(vtx)
    diameter = 0.
    nvtx = size(vtx,2)
    for i=1:(nvtx-1)
        for j=(i+1):nvtx
            diameter = max(diameter, norm(vtx[:,i]-vtx[:,j]))
        end
    end
    return diameter
end
```

Centroïde

```
function polygonCentroid(vtx)
    nvtx     = size(vtx,2)
    centroid = zeros(Float64,2)
    #
    den = 0.
    for i=1:nvtx
        ip = mod(i,nvtx) + 1
        w  = vtx[1, i]*vtx[2, j] - vtx[1, j]*vtx[2, i]
        centroid[1] += (vtx[1, i] + vtx[1, j])*w
        centroid[2] += (vtx[2, i] + vtx[2, j])*w
        den += w
    end
    return centroid/(3*den)
end
```

Monômes et gradients

```
function getMonomialsP1(xE::Array{Float64,1}, hE::Float64)
    #= Compute single P1 monomials =#
    return [x->1., x->(x[1]-xE[1])/hE, x->(x[2]-xE[2])/hE]
end

function getMonomialsP1(xE::Array{Float64,1}, hE::Float64,
    Cs::Array{Float64,1})
    #= Compute single P1 monomials with 0 average =#
    return [x->1., x->(x[1]-xE[1])/hE+Cs[2],
        x->(x[2]-xE[2])/hE+Cs[3]]
end

function getGradientMonomialsP1(xE::Array{Float64,1}, hE::Float64)
    #= Compute the gradient of the single P1 monomials =#
    return [x->[0.,0.], x->[1/hE,0.], x->[0.,1/hE]]
end
```