Quelques Idées d'utilisation du C++ en Calcul Scientifique

Frédéric Hecht, Cours Informatique Scientifique 2010-2011, Master 1
Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie

4 février 2011

Remarque typographique, tous les changement par rapport à la version de 01 janvier 2011 sont écrits en bleu.

Table des matières

1 Element et Différence finis en dimension 1

1.1 Notation

d est la dimension de l'espace, $d = 1$ ou 2 dans ce cours, et Ω est un ouvert connexe de $\mathbb{R}^d$ borné.

$$x \in \mathbb{R}^d \quad \Leftrightarrow \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Remarque, les vecteurs $\boldsymbol{u}$ seront notés généralement en police grasse, on a donc :

$$\boldsymbol{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_d \end{pmatrix}, \qquad \text{le gradient :} \qquad \nabla u = \mathbf{grad}\, u = \begin{pmatrix} \frac{\partial u}{\partial x_1} \\ \vdots \\ \frac{\partial u}{\partial x_d} \end{pmatrix}$$

$$\text{la divergence :} \qquad \nabla . \boldsymbol{u} = \operatorname{div} \boldsymbol{u} = \sum_{i=1}^{d} \frac{\partial u_i}{\partial x_i}$$

Le produit scalaire est noté avec un .

$$\boldsymbol{u}.\boldsymbol{v} = \sum_{i=1}^{d} u_i v_i$$

1.2 Rappels

Définition 1.1. *Dans un espace affine réel $\mathcal{A}$, le barycentre de $(\lambda_i, P_i)_{i=1,n} \in (\mathbb{R} \times A)^n$ telle que $\sum_{i=1}^n \lambda_i \neq 0$ est l'unique point $G = \operatorname{bar}((\lambda_i, P_i)_{i=1,n}) \in \mathcal{A}$ telle que*

$$\forall O \in \mathcal{A}, \qquad \sum_{i=1}^{n} \lambda_i \overrightarrow{OP}_i = (\sum_{i=0}^{n} \lambda_i) \overrightarrow{OG}.$$

Et si cette espace affine A est plongé dans un espace vectoriel, alors

$$G = \frac{\sum_{i=0}^n \lambda_i P_i}{\sum_{i=0}^n \lambda_i}$$

Définition 1.2. *Par définition, si f est une fonction affine de $\mathcal{A} \mapsto \mathcal{B}$, alors cette fonction commute avec les barycentres, c'est-à-dire $f(\operatorname{bar}((\lambda_i, P_i)_{i=1,n})) = \operatorname{bar}((\lambda_i, f(P_i))_{i=1,n})$, ou si l'espace affine est plongé dans un espace vectoriel, on a*

$$si \qquad \sum_{i=1}^{n} \lambda_i = 1, \qquad alors \quad f(\sum_{i=1}^{n} \lambda_i x_i) = \sum_{i=1}^{n} \lambda_i f(x_i) \tag{1}$$

De plus elle peut s'écrire comme $f(x) = A(x) + b$ où A est une application linéaire et b une constante.

– Forme bilinéaire et matrice. Soient E et F deux espaces vectoriels réels de dimension finie, soit $\{e_i/i \in I = \{1, ..., n\}\}$ une base de E, et $\{f_j/j \in J = \{1, ..., m\}\}$ une base de F, alors à toute forme bilinéaire a de $E \times F \mapsto \mathbb{R}$, on associe la matrice $A = (a_{ij})_{(i,j)\in I\times J}$ telle que $a_{ij} = a(e_i, f_j)$. Soit $l \in F'$ une forme linéaire de F.
Et Le problème : trouver $u \in E$ tel que

$$\forall v \in F, \quad a(u, v) = l(v) \tag{2}$$

est équivalent trouver $U \in \mathbb{R}^I$ en résolvant le système linéaire

$$^tA\,U = F \quad \text{où} \quad U = (u_i)_{i\in I}, \qquad u = \sum_{i\in I} u_i e_i, \quad \text{et où} \quad F = (l(f_j))_{j\in J}.$$

– Norme des espace $L^1(\Omega)$, $L^2(\Omega)$, $L^\infty(\Omega)$

$$||u||_{L^1(\Omega)} = \int_\Omega |u| dx; \quad ||u||_{L^2(\Omega)} = (\int_\Omega |u|^2 dx)^{1/2}; \quad ||u||_{L^\infty(\Omega)} = \sup_{x\in\Omega} \operatorname{ess} |u(x)| \tag{3}$$

– Produit scalaire et norme de espace $H^1(\Omega)$

$$(u, v)_{H^1(\Omega)} = \int_\Omega uv \; dx + \int_\Omega \mathbf{grad}\, u. \mathbf{grad}\, v \; dx \tag{4}$$

$$||u||_{H^1(\Omega)} = ||u||_{L^2(\Omega)} + ||(\mathbf{grad}\, u. \mathbf{grad}\, u)^{1/2}||_{L^2(\Omega)} \tag{5}$$

1.3 Différence finis mono dimensionnel

Trouver $u(x)$ fonction de $]\ell_0, \ell_1[$ tel que

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} + au &= f \\ u(\ell_0) = g_0, \quad \frac{\partial u}{\partial x}(\ell_1) &= g_1. \end{aligned} \tag{6}$$

où $\ell_0, \ell_1, g_0, g_1, a, f$ sont les données du problèmes.

1.3.1 Résolution du problème stationnaire par différence fini

Construction d'un schéma au différence fini :

Soit, M le nombre de points en espace, Notons x_i les points de calcul qui une suite strictement croissante tel que $x_0 = \ell_0$ et $x_M = \ell_1$, par exemple $x_i = \ell_0 + i\frac{\ell_1-\ell_0}{M}$. Notons $h_i = x_{i+1} - x_i$.

Soit u_i la valeur de la solution en x_i. si l'on écrit les formules de Taylor en x_i pour calculer u_{i+1} et u_{i-1}, on obtient :

$$u_{i+1} = u_i + u'_i h_i + \frac{1}{2}u''_i h_i^2 + \frac{1}{6}u'''_i h_i^3 + O(h_i^4) \tag{7}$$

$$u_{i-1} = u_i - u'_i h_{i-1} + \frac{1}{2}u''_i h_{i-1}^2 - \frac{1}{6}u'''_i h_{i-1}^3 - O(h_{i-1}^4) \tag{8}$$

Si $h_i = h$ est une constante, (7)+(8) donne :

$$u_i'' = \frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} + O(h^2) \tag{9}$$

et (7)-(8) donne :

$$u_i' = \frac{u_{i+1} - u_{i-1}}{2h} + O(h^2) \tag{10}$$

où des expression décentrées à gauche (resp.) à droite :

$$u_i' = \frac{u_{i+1} - u_i}{h} + O(h), \quad \text{(resp.) } u_i' = \frac{u_i - u_{i-1}}{h} + O(h) \tag{11}$$

Exercice 1. *Trouver, si h_i n'est pas constant, Construire les deux formules*

$$u_i'' = D_+^2 u_{i+1} + D_-^2 u_{i-1} + D_\cdot^2 u_i + O(h^1)$$

$$u_i' = D_+^1 u_{i+1} + D_-^1 u_{i-1} + D_\cdot^1 u_i + O(h^2)$$

Remarque : l'on supprime les dérives troisième et on est en $O(h^3)$ où $h = max(h_i, h_{i-1})$ et l'on résout un système linéaire pour calculer les 6 coefficient $D_+^2, D_-^2, D_\cdot^2, D_\cdot^1, D_-^1, D_\cdot^1$ en fonction de h_{i-1} et h_i.

(Ajoute 01/02/2010) Pour étudier la convergence de ce type de schéma, il faut montrer

Pour une bonne norme $||.||$ qui est telles que la norme du vecteur constant $v_i = 1$ $(i = 0, .., M)$ ne doit pas dépendre de M comme par exemples la norme l^∞. L'erreur $E = ||U - u^*||$ est petite (par exemple en $O(1./M)$) où le vecteurs u^* est le vecteur des valeurs de la solution exact en x_i et $U = (U_i)_{0,..,M}$ est la solution du schéma, solution du système linéaire $\mathcal{H}U = b$ avec l'opérateur linéaire $\mathcal{H}$ correspond au schéma numérique, et avec b est la partie dû au second membre et aux conditions au limites.

L'erreur de consistance du schéma est donné par $E_c = ||\mathcal{H}u^* - b||$ qui sera contrôlé avec les formules de Taylor.

L'erreur E ce calcul comme suit :

$$\begin{aligned} E &= ||U - u^*|| \\ &\leq ||\mathcal{H}^{-1}||\,||H(U - u^*)|| = ||\mathcal{H}^{-1}||\,||b - u^*|| = ||\mathcal{H}^{-1}||E_c \end{aligned}$$

Pour que l'erreur tend vers 0 quand M tend vers ∞, il suffit que le norme matriciel de $||\mathcal{H}^{-1}||$ soit borné indépendamment de M et que l'erreur de consistance E_c tend vers 0 quand M tend vers ∞.

Il n'est pas facile généralement de montre que la norme matriciel de $||\mathcal{H}^{-1}||$ est borné borné indépendamment M, mais l'on peut toujours le faire numériquement avec les logiciels comme Scilal, MathLab, ...

1.3.2 Résolution du problème instationnaire par différence fini

Trouver $u(x,t)$ fonction de $]\ell_0,\ell_1[\times[0,T]$ tel que

$$\begin{aligned} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} + au &= f \\ u(.,\ell_0) = u_0, \quad ,u(0,.) &= g_0, \quad ,\frac{\partial u}{\partial x}(\ell_1,.) = g_1. \end{aligned} \tag{12}$$

où $T, \ell_0, \ell_1, g_0, g_1, a, f$ sont les données du problèmes.

avec une discrétisation régulière en M pas en espace et N pas en temps : on cherche a approcher u en (x_i,t_j) par U_i^j avec $x_i = ih$ et $t_j = j\delta t$ et où $h = L/M$ et $\delta t = T/M$, et definition $u_i^j = u(x_i,t_j)$.

Comme précédemment :

$$\frac{\partial^2 u}{\partial x^2}(x_i,t_{n'}) \simeq \frac{U_{i+1}^{n'} - 2U_i^{n'} + U_{i-1}^{n'}}{h^2} \tag{13}$$

$$\frac{\partial u}{\partial t}(x_i,t_n) \simeq \frac{U_i^n - U_i^{n-1}}{\delta t} \tag{14}$$

Donc pour approcher le problème (18), il suffit d'utiliser (13) et (14) pour obtenir le problème approcher suivant

$$\frac{U_i^n - U_i^{n-1}}{\delta t} - \frac{U_{i+1}^{n'} - 2U_i^{n'} + U_{i-1}^{n'}}{h^2} + a_i U_i^n = f_i, \tag{15}$$

pour $i = 1, ..., M-1$ avec les conditions aux limites suivantes $U_i^0 = u_0(ih)$, et $U_0^n = g_0$ et $\frac{U_M^n - U_{M-1}^n}{h} = g_1$.

Ici nous donne respectivement la version implicite ($n' = n$) ou explicite ($n' = n-1$) du schéma d'Euler.

Définition 1.3. *Le schéma est dit explicite quand, il ne nécessite pas de résolution de système (linéaire ou non-linéaire).*

Pour étudier la convergence de ce type de schéma, il faut montrer que l'erreur $E_n = ||U^n - u^n||$ est petite (par exemple $O(\delta t + h)$ où $U^n = (U_i^n)_{i=0,...,M}$ et $u^n = (u_i^n)_{i=0,...,M}$ pour une bonne norme.

Pour cela nous sommes amené à étudier l'opérateur linéaire $\mathcal{H}$ qui correspond au schéma numérique, tel que l'on ait $U^n = \mathcal{H}(U^{n-1}) + b^n$ où b^n est la partie constante dû à f et aux conditions au limites.

L'erreur E_n ce décompose naturellement de deux erreurs :

$$\begin{aligned} E_n &= ||U^n - u^n|| \\ &= ||\overbrace{\mathcal{H}(U^{n-1}) + b^n}^{U^n} - u^n + \overbrace{\mathcal{H}u^{n-1} - \mathcal{H}u^{n-1}}^{0}|| = ||\overbrace{\mathcal{H}U^{n-1} - \mathcal{H}u^{n-1}}^{\text{Stabilite}} + \overbrace{\mathcal{H}u^{n-1} + b^n - u^n}^{\text{Consistance}}|| \\ &\leq ||\mathcal{H}U^{n-1} - \mathcal{H}u^{n-1}|| + ||\mathcal{H}u^{n-1} + b^n - u^n|| \end{aligned}$$

Définition 1.4. *L'erreur de consistance est* $||\mathcal{H}u^{n-1} - u^n + b^n||$ *et l' erreur de stabilité est* $||\mathcal{H}U^{n-1} - \mathcal{H}u^{n-1}||$.

Définition 1.5. *Un schéma sera dit consistant à l'ordre* p *en espace et* q *en temps si*

$$||\mathcal{H}u^{n-1} - u^n + b^n|| = O(h^p) + O(\delta t^q)$$

Définition 1.6. *Un schéma sera dit stable au sens de Lax et Richtmeyer si* $||\mathcal{H}^n||$ *reste borné* $(\mathcal{H}^n = \mathcal{H}\mathcal{H}^{n-1})$*, c'est à dire*

$$\forall n < \frac{T}{\delta t}, \quad ||\mathcal{H}^n|| \leq C_s$$

Théorème 1.7. *Si un Schéma est consistant et stable au sens de Lax et Richtmeyer alors le schéma est convergent, si la solution* u *du problème continue est suffisamment régulière.*

Démonstration en exercice

Etude stabilité au sens de Von Neuman

(Version corriger du 31/01/2011) Dans cette étude nous allons étudier le schéma via une transformée de Fourier discrète. Il faut faire les hypothèses suivantes : des conditions aux limites seront périodiques, un maillage régulier $x_j = \ell_0 + hj$ avec $h = (\ell_1 - \ell_0)/M$ en M segments, et les coefficients seront constant et le second membre nulle.

Nous allons recherche des solutions du schéma sont la forme

$$U_j^n = \sum_{k=0}^{M-1} c_k^n e^{i\pi kj/M}, \text{ avec} \qquad c_k^n = \sum_{j=0}^{M-1} \frac{U_j^n}{M} e^{i\pi - kj/M},$$

où k est la fréquence, et est le paramètre de Fourier spatial

La stabilité de Von Neumann se réduit simplement à verifier l'inégalité $c_k^n \leq c_k^{n-1}$, car les vecteurs $\omega^k = (\omega_j^k)_{j=O}^{M-1}$ définie par $\omega_j^k = e^{i\pi kj/M}$ forme une base pour $k = 0, M-1$ de $\mathbb{R}^M$ et de plus ces vecteurs sont orthogonaux pour le produit scalaire Hermitien de $\mathbb{R}^M$, on a :

$$(\omega^k, \overline{\omega^{k'}}) = \sum_{j=0}^{M-1} \omega_j^k \overline{\omega_j^{k'}} = \frac{\delta_{kk'}}{M}$$

où $\delta_{kk'}$ est le symbole de Kronecker (si $k = k'$ alors $\delta_{kk'} = 1$ sinon $\delta_{kk'} = 0$).

Donc l'étude des schémas d'Euler (15) page 10 donne donc :

$$\frac{U_j^n - U_j^{n-1}}{\delta t} - \frac{U_{j+1}^{n'} - 2U_j^{n'} + U_{j-1}^{n'}}{h^2} + aU_j^n = 0.$$

Réécrit dans la dans la base de Fourier avec le terme en n est mis a gauche de l'égalité , on a :

$$c_k^n e^{i\pi kj/M} = c_k^{n-1} e^{i\pi kj/M} + \frac{c_k^{n'}\delta t}{h^2}(e^{i\pi k(j-1)h} - 2e^{i\pi kj/M} + e^{i\pi k(j+1)h}) - \delta t a c_k^n e^{i\pi kj/M}$$

après simplification par $e^{i\pi kj/M}$, il reste

$$c_k^n(1+a\delta t) = c_k^{n-1} + c_k^{n'}\frac{\delta t}{h^2}(e^{i\pi k/M} - 2 + e^{-i\pi k/M})) \tag{16}$$

comme $\cos(x) = \frac{1}{2}(e^{ix} + e^{-ix})$ et que $\cos 2x = \cos^2 x - sin^2 x = 1 - 2sin^2 x$, on a :

$$c_k^n(1+a\delta t) = c_k^{n-1} + c_k^{n'}\frac{2\delta t}{h^2}(cos(\pi k/M) - 1)) = c_k^{n-1} - c_k^{n'}\frac{4\delta t}{h^2}sin^2(\pi k/M/2) \tag{17}$$

Ce schéma d'Euler implicite ($n' = n$) est inconditionnellement stable pour $a = 0$ au sens de Von Neumann, car il suffit de remarquer que (17) implique que

$$c_k^n \leq \frac{1}{1 + a\delta t - \frac{4\delta t}{h^2}sin^2(\pi k/M/2)}c_k^{n-1}$$

et que $(1 - \frac{4\delta t}{h^2}sin^2(\pi k/M/2)) < 1$.

Etude de la stabilité au sens de Von Neumann dans le cas du schéma d'Euler explicite ($n' = n-1$).

$$c_k^n(1+a\delta t) = c_k^n(1 - \frac{4\delta t}{h^2}sin^2(\pi k/M/2))$$

il est stable si

$$-1 \leq \frac{1 - \frac{4\delta t}{h^2}sin^2(\pi k/M/2)}{(1+a\delta t)} \leq 1$$

comme l'inégalité droite est toujours vérifié, il suffit d'avoir $1 + a\delta t \geq -1 + \frac{4\delta t}{h^2}sin^2(\pi k/M/2))$ donc finalement ce schéma d'Euler explicite est stable si $\delta t \leq \frac{(2+(a\delta t))h^2}{4}$.

Exercice 2. *Faire la même étude avec l'équation convection diffusion suivante : trouver $u(x,t)$ fonction de $]\ell_0, \ell_1[\times[0,T]$ tel que*

$$\begin{array}{rcl} \frac{\partial u}{\partial t} - \kappa\frac{\partial^2 u}{\partial x^2} + b\frac{\partial u}{\partial x} & = & 0 \\ u(.,0) = u^0, \quad , u(\ell_0,.) & = & u(\ell_1,.) \end{array} \tag{18}$$

où $T, \ell_0, \ell_1, \kappa > 0, a, b, f$ sont les données du problèmes réelles constantes et u^0 est la fonction donnée initial.

où le $\frac{\partial u}{\partial x}$ sera approché par l'une des formules (10) où (11). Parmi les six schémas implicite ou explicite, décrire lesquels sont stables en fonction du signe de b.

Etude pour pour les schémas décentre à gauche

$$\frac{U_j^n - U_j^{n-1}}{\delta t} - \kappa\frac{U_{j+1}^{n'} - 2U_j^{n'} + U_{j-1}^{n'}}{h^2} + b\frac{U_j^{n''} - U_{j-1}^{n''}}{h} = 0$$

On a après simplification par $e^{i\pi kjh}$, et en faisant les mêmes calculs avec n' , n'' prenant les valeurs n ou $n-1$ suivant les cas.

$$c_k^n = c_k^{n-1} - \kappa c_k^{n'}\frac{4\delta t}{h^2}sin^2(\pi k/M/2) - c_k^{n''}\frac{b\delta t}{h}(1 - e^{-i\pi k/M})$$

Par exemple pour le cas implicite / explicite , c'est-à-dire $n' = n$ et $n'' = n-1$ on a

$$\frac{c_k^n}{c_k^{n-1}} = \frac{1 - \frac{b\delta t}{h}(1 - e^{-i\pi k/M})}{1 + \frac{4\delta t}{h^2} sin^2(\pi k/M/2)} = \frac{1 - \frac{b\delta t}{h} + \frac{b\delta t}{h} e^{-i\pi k/M}}{1 + \frac{4\delta t}{h^2} sin^2(\pi k/M/2)}$$

pour que le module de $|\frac{c_k^n}{c_k^{n-1}}| \leq 1$ il suffit que

$$0 \leq b\delta t/h \leq 1, \tag{19}$$

cette condition (19) est appelé la condition CFL de Courant-Friedrick-Levy (1928).

Preuve : le numérateur décrit le cercle dans le plan complexe de centre $1 - \frac{b\delta t}{h}$ et de rayon $\frac{b\delta t}{h}$ qui est clairement inclus dans le cercle unité si $b > 0$. Ceci implique que le module du numérateur est plus petit ou égal à 1 et comme le dénominateur est plus grand où égal à 1, on a fini.

De plus,
- cette condition est optimale à la limite quand $M \to \infty$, Il suffit de prendre le dernier mode $k = M - 1$ et de remarquer : $\lim_{M\to\infty} e^{-i\pi(M-1)/M} = -1$;
- et si $b < 0$ alors le schéma est inconditionnellement instable , il suffit de prendre $k = 0$ et de remarquer que $\frac{c_0^n}{c_0^{n-1}} = 1 - 2\frac{b\delta t}{h} > 1$.

Donc si b est négatif, il faut faire un décentrage à droite et l'on peut donc écrire le schéma comme suit

$$\frac{U_j^n - U_j^{n-1}}{\delta t} - \kappa \frac{U_{j+1}^n - 2U_j^{n-1} + U_{j-1}^n}{h^2} + b^+ \frac{U_j^{n-1} - U_{j-1}^{n-1}}{h} - b^- \frac{U_{j+1}^{n-1} - U_j^{n-1}}{h} = 0$$

où par définition $b = b^+ - b^-$ avec $b^+ = max(b, 0)$, et $b^- = -max(-b, 0)$.

1.4 Le problème modèle

Trouver $u(t, x)$ une fonction de l'ouvert $[0, T] \times \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ telle que

$$\frac{\partial u}{\partial t} - \text{ div }(\ a\ \mathbf{grad}\,(u)) + (\boldsymbol{b}.\mathbf{grad}\, u) + c\, u \quad = f, \quad \text{dans} \quad \Omega \tag{20}$$

$$u \quad = u_d, \quad \text{sur} \quad \Gamma_d \tag{21}$$

$$(a\,\mathbf{grad}\, u).\boldsymbol{n} + \alpha_r u \quad = \beta_r, \quad \text{sur} \quad \Gamma_r$$

Où $u(0, x) = u_0(x)$ est donné au temps $t = 0$, et où a,$\boldsymbol{b}$, c, u_0, u_d, α_r, β_r sont des fonctions données qui peuvent dépendre de t et de x suivante la modélisation. Et où $\boldsymbol{n}$ est le vecteur normal au bord de Ω, dirigé vers l'extérieur du domaine.

La fonction a peut même être un champ de matrices $d \times d$.

1.5 Le problème modèle 1D stationnaire

Trouver u une fonction de l'ouvert $\Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ tel que

$$-\text{ div }(\ a\ \mathbf{grad}\,(u)) + (\boldsymbol{b}.\mathbf{grad}\, u) + c\, u \quad = f, \quad \text{dans} \quad \Omega \tag{22}$$

$$u \quad = u_d, \quad \text{sur} \quad \Gamma_d \tag{23}$$

$$a\,\mathbf{grad}\, u.\boldsymbol{n} + \alpha_r u \quad = \beta_r, \quad \text{sur} \quad \Gamma_r \tag{24}$$

Les données sont :
- $d = 1$, l'ouvert $\Omega =]\ell_0, \ell_1[$, avec $\ell_0 < \ell_1$ deux nombres réels donnés, avec $\Gamma_d = \{\ell_0\}$ et $\Gamma_r = \{\ell_1\}$.
- a,$\boldsymbol{b}$, c, sont des fonctions de x connues , et les termes u_d, α_r, β_r sont des constantes dans ce cas.

$\boldsymbol{n}$ est le vecteur normal exterieur au bord de Ω, c'est-à-dire que $\boldsymbol{n} = -1$ en ℓ_0 et $\boldsymbol{n} = +1$ en ℓ_1.

La formulation variationnelle est obtenue en multipliant (22) par une fonction régulière v et en intégrant.

$$-\int_\Omega \operatorname{div}(\ a\ \mathbf{grad}\,(u))v + (\boldsymbol{b}.\mathbf{grad}\,u)v + c\ uv\ dx = \int_\Omega fv\ dx, \tag{25}$$

On utilise la formule d'intégration par partie suivante

$$-\int_\Omega (\ a\ (u)')'v dx = \int_\Omega a\ u'v' dx - [a\ u'v]_{\ell_0}^{\ell_1}. \tag{26}$$

Comme u est connue en ℓ_0, on va prendre $v = 0$ en ℓ_0. Le terme $[a\ u'v]_{\ell_0}^{\ell_1}$ devient donc $a\ u'(\ell_1)v(\ell_1)$, qui après utilisation de la condition de Robin (ou Fourier suivant les auteurs) (24) et avec $\boldsymbol{n} = 1$ en ℓ_1, on obtient $a\ u'(\ell_1) = \beta_r - \alpha_r u(\ell_1)$.

Il faut dériver v et u, donc introduisons naturellement l'espace $X = H^1(\Omega)$ des fonctions de $L^2(\Omega)$ à dérivée au sens des distributions dans $L^2(\Omega)$ et l'espace X_0 l'espace des fonctions test v nulles sur Γ_d, c'est-à-dire

$$X_0 = \{v \in X / v(\ell_0) = 0\}. \tag{27}$$

Le problème se réécrit donc

Trouver $u \in X_{u_d} = \{v \in X / v(\ell_0) = u_d\}$ tel que $\forall v \in X_0$ on a

$$\int_\Omega a\ \mathbf{grad}\,u.\mathbf{grad}\,v + (\boldsymbol{b}.\mathbf{grad}\,u)v + c\ uv\,dx + \alpha_r u(\ell_1)v(\ell_1) = \int_\Omega fv\,dx + \beta_r v(\ell_1) \tag{28}$$

Notons par $\boldsymbol{a}$ la forme bilinéaire

$$\boldsymbol{a}(u,v) = \int_\Omega a\ \mathbf{grad}\,u.\mathbf{grad}\,v + (\boldsymbol{b}.\mathbf{grad}\,u)v + c\ uv\ dx\ + \alpha_r u(\ell_1)v(\ell_1), \tag{29}$$

et par $\boldsymbol{l}$ la forme linéaire

$$\boldsymbol{l}(v) = \int_\Omega fv\ dx\ + \beta_r v(\ell_1). \tag{30}$$

Le problème ce réduit formellement à

Trouver $u \in X_{u_d}$ tel que

$$\forall v \in X_0; \qquad \boldsymbol{a}(u,v) = \boldsymbol{l}(v). \tag{31}$$

Ce problème n'est pas linéaire, il est affine. Pour le rendre linéaire, il suffit de choisir une fonction $\tilde{u}_d$ qui vérifie la condition de Dirichlet (c'est-à-dire $\tilde{u}(x) = u_d(x)$ pour $x \in \Gamma_d$).

Notons $\tilde{u} = u - \tilde{u}_d$, on a $\tilde{u} \in X_0$ et $u = \tilde{u} + \tilde{u}_d$. Le problème précédent est équivalent au problème linéaire suivant :

Trouver, $\tilde{u} \in X_0$ tel que

$$\forall v \in X_0; \qquad \boldsymbol{a}(\tilde{u}, v) = \boldsymbol{l}(v) - \boldsymbol{a}(\tilde{u}_d, v) \tag{32}$$

Le système linéaire associé est clairement carré, les inconnues et les fonctions test sont dans le même espace X_0.

Si l'espace X_0 était de dimension finie, on saurait écrire le système linéaire associé. Mais bien sûr X_0 n'est pas de dimension finie et on va approcher X_0 par une famille d'espaces X_{0h} de dimension finie. Le paramètre h est fait pour tendre vers 0. Cette famille d'espaces X_h de dimension finie dans X vont approcher X. C'est à dire que, par exemple, on a l'estimation suivante

$$\inf_{v_h \in X_h} ||u - v_h||_X \leq Ch^s ||v||,$$

où C et s sont deux constantes strictement positives données. Notons $X_{h0} = X_h \cap X_0$.

Alors, on peut résoudre le problème approché dans X_{h0} :

Trouver $\tilde{u}_h \in X_{0h}$ tel que :

$$\forall v_h \in X_{h0}; \qquad \boldsymbol{a}(\tilde{u}_h, v_h) = \boldsymbol{l}(v_h) - \boldsymbol{a}(\tilde{u}_{0h}, v_h) \tag{33}$$

où $\tilde{u}_{0h}$ est une fonction de X_h qui relève les conditions aux limites de Dirichlet, c'est-à-dire telle que $\tilde{u}_{0h}(\ell_0) = u_d$.

1.6 Elément fini 1D

Soit $q^i, i = 0, ..., N$ une suite strictement croissante de $N+1$ réels telle que $q^0 = \ell_0$ et $q^N = \ell_1$. Nous noterons par K_k l'intervalle $]q^{k-1}, q^k[$, et $|K_k| = q^k - q^{k-1}$ la mesure de K_k. Nous appelons $h = \sup_{k=1}^N |K_k|$, et par abus de langage nous noterons $\mathcal{T}_{d,h} = \{K_k, k = 1, ..., N\}$ un maillage de Ω de taille maximale h. Les intervalles K_k seront appelés les élements de $\mathcal{T}_{d,h}$. Soit K un élément de $\mathcal{T}_{d,h}$, nous notons i_0^K, i_1^K les deux sommets de K tels que $i_0^K < i_1^K$ et notons k^K le numéro de l'élément K, c'est à dire :

$$K =]q^{i_0^K}, q^{i_1^K}[, \qquad k^K - 1 = i_0^K \qquad k^K = i_1^K, \qquad k = k^{K_k}.$$

Associé à ce maillage mono-dimensionnel, nous définissons l'espace des éléments finis P_1 Lagrange comme suit :

$$X_h = \{v \in \mathcal{C}^0(\Omega) / \ \forall K \in \mathcal{T}_{d,h}, \quad v_{|K} \in P_1(K)\} \quad \subset H^1(\Omega) \tag{34}$$

– $v_{|K}$ est la fonction restriction de v sur l'intervalle K qui est définie par

$$v_{|K} : \quad x \in K \mapsto v(x).$$

– $P_1(K)$ est l'espace des fonctions polynômes de degré inférieur ou égal à 1 de l'intervalle K. C'est-à-dire que $v_{|K}(x) = a_K x + b_K$.

On remarque que les fonctions de $H^1(\Omega)$ sont des fonctions continues sur Ω dans le cas mono-dimensionnel, ce qui est malheureusement faux si $d > 1$. Et les fonctions de $\mathcal{C}^0(\Omega)$ à dérivée bornée sont toujours dans $H^1(\Omega)$ si l'ouvert Ω est borné. C'est pour cette raison que l'on a une approximation interne.

Lemme 1.8. *Les fonctions de X_h sont uniquement définies par leurs valeurs aux points q^i, pour $i = 0, ..., N$.*

Effectivement, notons le deux fonctions de base λ_0^K et λ_1^K de $P_1(K_k)$ suivante :

$$\lambda_0^K(x) = \frac{x - q^{i_1^K}}{q^{i_0^K} - q^{i_1^K}}, \qquad \text{et} \qquad \lambda_1^K(x) = \frac{q^{i_0^K} - x}{q^{i_0^K} - q^{i_1^K}} \tag{35}$$

Remarquons que $\lambda_0^K(q^{i_0^K}) = 1$ et $\lambda_0^K(q^{i_1^K}) = 0$ et inversement $\lambda_1^K(q^{i_0^K}) = 0$ et $\lambda_0^K(q^{i_1^K}) = 1$, ce que l'on peut résumer par $\lambda_l^K(q^{i_m^K}) = \delta_{lm}$ pour $l, m = 0$ ou 1 (où δ_{lm} est le symbole de Kroneker). Ces 2 fonctions sont appelées les coordonnées barycentrique de K, car on a les deux égalités triviales fondamentales suivantes

$$x = \lambda_0^K(x) q^{i_0^K} + \lambda_1^K(x) q^{i_0^K} \quad \text{et} \quad \lambda_0^K(x) + \lambda_1^K(x) = 1 \tag{36}$$

Donc, sur tout intervalle K de $\mathcal{T}_{d,h}$, on a

$$\forall v_h \in X_h; \quad v_{h|K}(x) = v(q^{i_0^K}) \lambda_0^K(x) + v(q^{i_1^K}) \lambda_1^K(x) \tag{37}$$

Définition 1.9. *Notons $\{w_i, i = 0, ..., N\}$ l'ensemble des fonctions de X_h telles que*

$$w_i(q^j) = \delta_{ij}.$$

Alors cet ensemble est une base de X_h, et l'ensemble $\{w_i, i = 1, ..., N\}$ est une base de $X_{0h} = X_h \cap X_0 = \{v_h \in X_h / v_h(\ell_0) = 0\}$. De plus on a

$$\forall v \in X_h \qquad v = \sum_{i=0}^{N} v(q^i)\, w_i \tag{38}$$

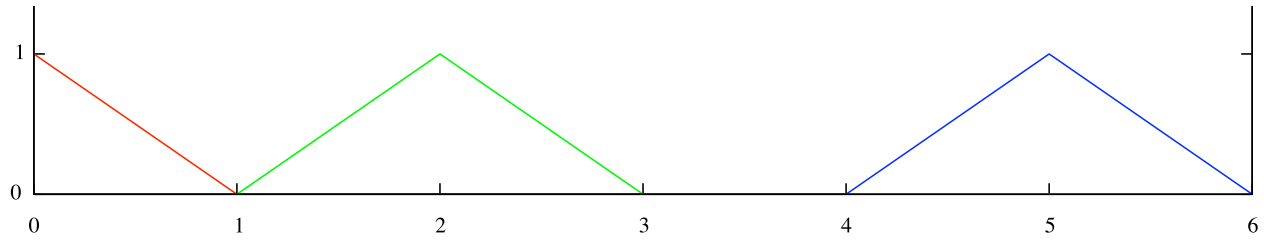

FIGURE 1 – Les trois fonctions de base respectivement w_0, w_2 et w_5.

1.7 Construction du système linéaire

Pour simplifier, toutes le données sont supposées dans un premier temps constantes, ce sont donc des nombres réels.

Lemme 1.10. *Le système linéaire $N \times N$ à résoudre $A_0 U_0 = F_0$ est défini par*

$$A_0 = (a_{ij})_{i,j=1,\ldots,N} \quad \text{avec} \quad a_{ij} = \boldsymbol{a}(w_j, w_i)$$

Attention à la transposition des indices i **et** j, *car les lignes correspondent aux fonctions test v par construction.*

et

$$F_0 = (f_i)_{i=1,..,N} \quad \text{avec} \quad f_i = \boldsymbol{l}(w_i) - a(\tilde{u}_{dh}, w_i)$$

et où l'on a défini $\tilde{u}_{dh} = u_d\, w_0$.

Pour finir la solution du problème est la fonction suivante

$$u = \sum_{i=0}^{N} u_i\, w_i \qquad \text{avec} \quad u_0 = u_d \quad \text{et avec} \quad U_0 = (u_i)_{i=1}^{N}$$

Ce système peut être réécrit plus simplement pour se rapprocher de l'informatique comme le système $(N+1) \times (N+1)$ suivant :

$$AU = F, \quad A = (a_{ij})_{i,j=0,\ldots,N}, \quad F = (f_i)_{i=0}^{N}, \quad U = (u_i)_{i=0}^{N} \tag{39}$$

où $a_{0,j} = \delta_{0j}$, $\quad a_{ij} = \boldsymbol{a}(w_j, w_i)$, si $i > 0$, $\quad f_0 = u_d$, $\quad$ et $\quad f_i = \boldsymbol{l}(w_i)$.

La construction de la matrice A et du second membre F :

Notons $\boldsymbol{a}_K$ la forme bilinéaire suivante :

$$\boldsymbol{a}_K(u, v) = \int_K a\ \mathbf{grad}\, u.\, \mathbf{grad}\, v + (\boldsymbol{b}.\, \mathbf{grad}\, u)v + c\, uvdx, \tag{40}$$

et $\boldsymbol{l}_K$ la forme linéaire

$$\boldsymbol{l}_K(v) = \int_K f v\, dx. \tag{41}$$

On a clairement

$$\boldsymbol{a}(u, v) = \sum_{K \in \mathcal{T}_{d,h}} \boldsymbol{a}_K(u, v) + \alpha_r u(\ell_1) v(\ell_1)$$

et

$$\boldsymbol{l}(v) = \sum_{K \in \mathcal{T}_{d,h}} \boldsymbol{l}_K(u, v) + \beta_r v(\ell_1)$$

Remarque 1. $\boldsymbol{a}_K(w_i, w_j) = 0$ *si i, j ne sont pas des sommets $\{i_0^K, i_1^K\}$ de K alors les seuls éléments non nuls sont pour (i, j) dans $\{i_0^K, i_1^K\}^2$, soit quatre valeurs. Donc, la matrice A_K associé à $\boldsymbol{a}_K$ est une matrice de la forme*

$$A_K = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \check{a}_{00}^K & \check{a}_{01}^K & 0 & 0 \\ 0 & 0 & \check{a}_{10}^K & \check{a}_{11}^K & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{42}$$

pour l'élément K_3.

$$\check{a}_{lm}^K = \int_K a \ \mathbf{grad}\,\lambda_m^K . \mathbf{grad}\,\lambda_l^K + (\boldsymbol{b}.\mathbf{grad}\,\lambda_m^K)\lambda_l^K + c\lambda_m^K\lambda_l^K dx$$

ce terme $\check{a}_{lm}^K$ se place en $a^K_{i_l^K i_m^K}$ avec i_0^K, i_1^K les deux numéros des sommets de K, c'est-à-dire que $K =]q^{i_0^K}, q^{i_1^K}[$.

Lemme 1.11. *La matrice A est tridiagonale.*

Calcul de la matrice $\check{A}^K$

Soit K l'élement K de sommets i_0^K, i_1^K, avec $i_0^K < i_1^K$, notons $h_K = (q^{i_1^K} - q^{i_0^K})$, on a

- $\mathbf{grad}\,\lambda_m^K = -1^{\delta_{0m}}/h_K$
- $\int_K \lambda_m^K dx = h_K/2$
- $\int_K \lambda_m^K \lambda_l^K dx = h_K(1+\delta_{lm})/6$

Donc si a, $\boldsymbol{b}$, c sont des fonctions constantes, la matrice $\check{A}^K$ (2×2) est définie par :

$$\forall (l,m) \in \{0,1\}^2, \qquad \check{a}_{lm}^K = (-1)^{1+\delta_{lm}}\frac{a}{h_k} + (-1)^{\delta_{m0}}\frac{\boldsymbol{b}}{2} + h_K c\frac{1+\delta_{lm}}{6}$$

Exercice 3.

Ecrire une fonction C++ qui ajoute à une matrice tri-diagonale A, la matrice A^K, dans le cas où les fonctions a, $\boldsymbol{b}$, c, f sont constante.
Puis, écrire une fonction qui construit la matrice tri-diagonale A complète définie en (39), et une autre fonction pour construire le seconde membre associé F.
La matrice tri-diagonale pourra être modélisée avec la classe C++ suivante :

```
class MatTriDia { public:
  int n;
  vector<double> a,b,c;
  MatTriDia(int nn) : n(nn),a(n),b(n),c(n) {MiseAZero();}
  void MiseAZero() { for (inti=0;i<n;++i) a[i]=b[i]=c[i]=0.;}
  double & operator()(int i,int j) {
     if(i==j-1) return a[i];
     else if (i==j) return b[i];
     else if (i==j+1) return c[i];
     else {cerr << "..\n"; abort();} }  //    #include <cstdlib>
 };
```

Pour un objet défini par `MatTriDia A(n);` on aura les correspondances suivantes :
`A.a(i)` $\equiv a_{i,i-1}$, `A.b(i)` $\equiv a_{i,i}$, `A.c(i)` $\equiv a_{i,i+1}$.
La matrice `A` de taille n est donc de la forme :

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & . & . & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & a_{n-2} & b_{n-2} & c_{n-2} \\ 0 & 0 & 0 & 0 & . & . & . & a_{n-1} & b_{n-1} \end{pmatrix} \tag{43}$$

Attention à la correspondance entre N et n.

Exercice 4.

Voila une classe `MatElement` qui représente une matrice A_K.

```
class MatElement { public:
  int iK[2];      //    les indices où l'on met la matrice 2x2
  double  a[2][2];                          //    les 2x2 coef
  MatElement(int i1,int i2,double a00,double a01,
                           double a10,double a11)
   {
     ii[0]=i1; ii[1]=i2;
     a[0][0]=a00;
     a[1][0]=a10;
     a[0][1]=a01;
     a[1][1]=a11;
  };
  };
```

1. Ajouter à la classe `MatTriDia`, l'opérateur
   `MatTriDia & MatTriDia::operator+= (MatElem &Ak);`
2. Construire une fonction `AK` qui retourne la matrice élémentaire A_K de prototype :
   `MatElement AK(double qk1,double qk,int k,double a, double b, double c);`

1.8 Méthodes directes de résolution du système linéaire

Exercice 5.

Ajouter à la classe `MatTriDia`, l'opérateur
`MatTriDia & MatTriDia::operator+= (MatElem &Ak);`
où la classe `MatElem` représente une matrice élémentaire.

Les matrices tri-diagonales sont très faciles à factoriser par la méthode de Gauss. Si le système $AX = F$ est écrit sous la forme générale ($a_0 = c_{n-1} = 0$ par convention)

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & . & . & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & a_{n-2} & b_{n-2} & c_{n-2} \\ 0 & 0 & 0 & 0 & . & . & . & a_{n-1} & b_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ . \\ . \\ x_{n-2} \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ . \\ . \\ f_{n-2} \\ f_{n-1} \end{pmatrix} \tag{44}$$

la matrice A peut se décomposer sous la forme $A = LU$, avec les matrices

$$L = \begin{pmatrix} b_0^* & 0 & 0 & . & . & 0 & 0 \\ a_1 & b_1^* & 0 & 0 & . & 0 & 0 \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & a_{n-2} & b_{n-2}^* & 0 \\ 0 & 0 & 0 & 0 & . & a_{n-1} & b_{n-1}^* \end{pmatrix}, U = \begin{pmatrix} 1 & c_0^* & 0 & . & . & 0 & 0 \\ 0 & 1 & c_1^* & 0 & . & 0 & 0 \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & 1 & c_{n-2}^* \\ 0 & 0 & 0 & 0 & . & 0 & 1 \end{pmatrix}.$$

Exercice 6. Ajouter à la classe `MatTriDia` la méthode `void LU();` qui remplace les vecteurs a,b,c par leurs correspondants a^*,b^*,c^*.

En identifiant les coefficients, nous obtenons les récurrences suivantes pour le calcul des coefficients b^* et c^* :

$$\left\{\begin{array}{l} b_0^* = b_0 \\ c_0^* = c_0/b_0 \end{array}\right. \qquad \left\{\begin{array}{l} b_k^* = b_k - a_k \cdot c_{k-1}^* \\ c_k^* = c_k/b_k^* \end{array}\right. \qquad k = 1 \ldots n-1. \tag{45}$$

Remarque 2. *Observons que la factorisation LU est possible si les coefficients b_k^* sont non-nuls, donc la matrice A doit être inversible.*

Le système peut être résolu maintenant en deux étapes $AX = F \Longrightarrow L\underbrace{UX}_{Y} = F$.

$$LY = f \Longrightarrow \left\{\begin{array}{l} y_0 = f_0/b_0^* \\ y_k = \left(f_k - a_k\, y_{k-1}\right)/b_k^*, \quad k = 1 \ldots n-1 \end{array}\right. \tag{46}$$

$$UX = Y \Longrightarrow \left\{\begin{array}{l} x_{n-1} = Y_{n-1} \\ x_k = y_k - c_k^*\, x_{k+1}, \quad k = (n-2) \ldots 0 \end{array}\right. \tag{47}$$

Remarque 3. *Pour la programmation, nous utiliserons seulement quatre vecteurs a, b, c, f contenant initialement les coefficients a_k, b_k, c_k, f_k du système. Les récurrences (45) seront groupées dans la même boucle, ce qui permet de stocker b^* dans b et c^* dans c. Pour la boucle ascendante (46) Y peut être stocké dans F et, finalement, dans la boucle descendante (47), la solution X du système sera être stockée également dans F.*

Exercice 7. Ajouter la méthode `void Solve(vector<double> &F);` à la classe `MatTriDia` qui résout le système $AX = F$, si la matrice A est factorisée sous la forme LU. La fonction retournera la solution dans le vecteur F.

1.9 Formule d'intégration

Comme vous avez pu le remarquer, nous avons besoin de calculer des intégrales sur des segments. Voilà quelques formules pour approcher ces intégrales.

Soit $K =]q^i, q^j[$ un intervalle ($i = i_0^K$ et $j = i_1^K$), notons $h_K = |q^i - q^j|$ la longueur de l'intervalle.

Une formule d'intégration s'écrit

$$\int_K f dx \sim h_K \sum_{\ell=1}^{N} \omega_k f(\xi_\ell^K) \tag{48}$$

Les points integrations ξ_ℓ^K sont donnés dans un élément de référence $\hat{K}$ ici l'élément $\hat{K} =]0, 1[$. Ces points dans l'élément de référence $\hat{K}$ sont notés $\hat{\xi}_\ell$ et on a

$$\xi_\ell^K = (1 - \hat{\xi}_\ell) q^i + \hat{\xi}_\ell q^j \tag{49}$$

De plus, on remarquera que l'on a :

$$\lambda_0^K(\xi_\ell^K) = 1 - \hat{\xi} \qquad \lambda_1^K(\xi_\ell^K) = \hat{\xi} \tag{50}$$

donc pour on a

$$w_i(\xi_\ell^K) = 1 - \hat{\xi}_\ell \quad \text{et} \quad w_j(\xi_\ell^K) = \hat{\xi}_\ell \tag{51}$$

ce qui simplifie grandement, utilisation des formule d'intégration avec les fonctions de base de l'élément K.

Les paramètres des formules sont N le nombre points d'intégrations, les points $\hat{\xi}_\ell$ et poids ω_ℓ d'intégration, l'ordre de la formule, et pour quelle type de polynôme la formule est exact.

Voilà une petite liste de formule d'intégration bien utile :

N	`ordre`	$\hat{\xi}_\ell$	ω_ℓ	exact sur $P_k, k =$
1	2	$1/2$	h_K	1
2	4	$(1 \pm \sqrt{1/3}\ )/2$	$h_K/2$	3
3	6	$(1 \pm \sqrt{3/5})/2$	$(5/18)h_K$	5
		$1/2$	$(8/18)h_K$	
4	8	$(1 \pm \frac{\sqrt{525+70\sqrt{30}}}{35})/2$	$\frac{18-\sqrt{30}}{72}\ h_K$	7
		$(1 \pm \frac{\sqrt{525-70\sqrt{30}}}{35})/2$	$\frac{18+\sqrt{30}}{72}h_K$	
5	10	$(1 \pm \frac{\sqrt{245+14\sqrt{70}}}{21})/2$	$\frac{322-13\sqrt{70}}{1800}h_K$	9
		$1/2$	$\frac{64}{225}h$	
		$(1 \pm \frac{\sqrt{245-14\sqrt{70}}}{21})/2$	$\frac{322+13\sqrt{70}}{1800}h_K$	
2	2	$1/2 \pm 1/2$	$h_K/2$	1

1.10 Analyse de la méthode

L'un point de vue théorique notre problème est de la forme suivante :

Définition 1.12 (Le problème abstrait). *Soit H un espace de Hilbert. soit $\boldsymbol{a}$ une forme bilinéaire sur H et $\boldsymbol{l}$ un forme linéaire sur H.*

Le problème formelle est : trouvez $u \in H$, telle que

$$\forall v \in H; \qquad \boldsymbol{a}(u, v) = l(v). \tag{52}$$

Définition 1.13. *Un problème est dit bien posé s'il existe une silution unique et qu'elle dépend continûment des données.*

Le théorème suivant nous donne les hypothèses pour que le problème soit bien posé.

Théorème 1.14 (Banach-Necas-Babuska)**.** *Si la forme bilinéaire* $\boldsymbol{a}$ *est continue et la forme linéaire* $\boldsymbol{l}$ *sont continue sur* H*, c'est à dire*

$$\boldsymbol{a}(u,v) \leq c_M \, ||u|| \, ||v||, \qquad \boldsymbol{l}(v) \leq c_l ||v||$$

le problème (52) est bien pose si et seulement si la condition inf-sup suivante est vérifé

$$\inf_{u \in H} \sup_{v \in H} \boldsymbol{a}(u,v) \geq \alpha ||u|| \, ||v||, \qquad \alpha > 0$$

et la condition : $(\forall u \in H; \quad \boldsymbol{a}(u,v) = 0) \Rightarrow v = 0$.

Démonstration. Notons $A : u \in H \mapsto a(u,.) \in H'$, où H' est l'ensemble des formes linéaires continue de H, la norme pour $f \in H'$ est

$$||f|| = \sup_{v \in H, v \neq 0} \frac{f(v)}{||v||}.$$

Donc, pour tout $v \in H$, on a $\alpha ||u|| \leq ||A(u)|| \leq c_M ||u||$, donc A est injective.

Maintenant, montrons que A est sujective, pour cela démontrons que $A(H)$ est fermé. Il suffit de montrer que toute suite y_n un suite de Cauchy de $A(H)$ converge. Notons la suite x_n tel que $A(x_n) = y_n$ (A est injective), la condition inf-sup nous donne

$$||y_n - y_m|| = A(x_m - y_m) = \sup_{v \in H, v \neq 0} \frac{a(x_n - x_m, v)}{||v||} \leq \alpha ||x_n - x_m||$$

La suite x_n est de cauchy, et donc converge vers x (H est complet) et par continuité la suite de Cauchy y_n converge vers $A(x)$ donc $A(H)$ est bien fermé. La second condition nous montre que l'orthognonal de $A(H)$ est réduit à 0, comme $A(H)$ est ferme, on a $A(H) = H'$, d'où A est inversible et continue de H dans H', ce qui termine la première implication.

La réciproque est trivial, si l'on remarque qu'un problème est bien posé implique que A à un inverse continue. □

Pour utiliser ce théoreme, si l'on construit un $v_u \in H$ telle que $a(u, v_u) \geq \alpha_1 ||u||^2$ et $||v_u|| \leq \alpha_2 ||u||$, alors la condition inf-sup est vérifie pour $\alpha = \frac{\alpha_1}{\alpha_2}$.

Définition 1.15 (Coercivité). *On dit qu'une forme bilinéaire* $\boldsymbol{a}$ *de* $H \times H$*, est coersive, si il existe une constante réelle* $c_m > 0$ *telle que*

$$\forall v \in H; \qquad c_m ||v||^2 \leq \boldsymbol{a}(v,v) \tag{53}$$

Lemme 1.16 (Lax-Milgram)**.** *Si la forme bilinéaire* $\boldsymbol{a}$ *de* $H \times H$*, est coercive et continue, et si forme linéaire* $\boldsymbol{l}$ *est continue, alors le problème (52) est bien posé.*

Démonstration. Il suffit d'utiliser le théoréme précédent, en choisissant $v_u = u$ pour la condition inf-sup. Pour la second condition, il suffit de remarquer que si $\boldsymbol{a}(v,v) = 0$ alors $c_m||v||^2 \leq \boldsymbol{a}(v,v) = 0$, et donc $v = 0$. □

Lemme 1.17 (Inégalite de Poincaré)**.** *Soit Ω un ouvert connexe borné de $\mathbb{R}^d$ soit Γ^d u, partie de du bord de Ω de mesure positive réguliere, alors sur le sous espace X_0 des fonctions de $H^1(\Omega)$ nulle sur Γ_d, il existe une constant C telle que*

$$\forall v \in V_0; \qquad \int_\Omega u^2 dx \leq C \int_\Omega ||\,\mathbf{grad}\, u||^2 dx. \tag{54}$$

Cette inégalité, implique que la semi-norme suivante

$$|u|_{H_1(\Omega)} = \left(\int_\Omega ||\,\mathbf{grad}\, u||^2 dx \right)^{\frac{1}{2}}$$

est une norme équivalente à la norme $H^1(\Omega)$ sur l'espace X_0.

Démonstration. Premièrement, Si $X_0 = H_0^1(\Omega)$ alors, Comme $\mathcal{D}(\Omega)$ est dense dans $H_0^1(\Omega)$ il suffit d'écrire une majoration pour les fonctions de $\mathcal{D}(\Omega)$.

En intégrant sur l'axe de x on a

$$u(x,..) = \int_{x_\Gamma(...)}^{x} \partial_x u(\xi,...) d\xi, \text{ avec } x_\Gamma(...) \in \partial\Omega$$

Puis en utilisant l'inégalite de Cauchy-Schawrz on a

$$u(x,...)^2 = (\int_{x_\Gamma(...)}^{x} \partial_x u(\xi,...) d\xi)^2 \leq (\int_{x_\Gamma(...)}^{x} (\partial_x u(\xi,...))^2 d\xi) \int_{x_\Gamma(...)}^{x} 1 d\xi$$

Donc, on en utilisant le théorème de Fubini ;

$$||u||_{L^2}^2 \leq ||\partial_x u||_{L^2}^2 (\int_\Omega (x - x_\Gamma(...))^2$$

Pour finir,il suffit de remarquer que $(\int_\Omega (x - x_\Gamma(...))^2$ est borné.

Sinon, comme l'ouvert est borné et regulier et connexe , soit x_0 sur le bord de l'ouvert, pour tout point x de l'ouvert il existe une courbe γ_x C^∞ par morceaux qui va de x_0 à x de paramètre par l'abscisse curviligne de plus les longueurs des γ_x est L_x borné, par L suffissament grand.

Il suffit de refaire la même preuve avec les paramétrisations γ_x.

□

Montrons que notre problème mono-dimensionnel est bien posé sous les hypothèses suivantes.

Théorème 1.18. *Si la mesure de Γ_d est strictement positive, si les fonctions sont telles que $a \in L^\infty(\Omega)$, $\boldsymbol{b} \in W^{1,\infty}(\Omega)$, $c \in L^\infty(\Omega)$, si il existe une constant réel c_a, telle que $0 < c_a \leq a$, $c - \frac{\boldsymbol{b}'}{2} \geq 0$, $0 \leq \alpha_r$, $\boldsymbol{b}.\boldsymbol{n} \geq 0$ sur Γ_r alors le problème de bien posé.*

Démonstration. Nous allons utiliser le lemme de Lax-Milgram.

Premièrement, il est évident $\boldsymbol{a}$ est bien continue.

On a

$$a(v,v) = \int_\Omega a||\,\mathbf{grad}\,v||^2 dx + \int_\Omega (\boldsymbol{b}.\,\mathbf{grad}\,v)v dx + \int_\Omega cv^2 dx + \underbrace{\alpha_r v(\ell_1)^2}_{\geq 0}.$$

En notant $(\boldsymbol{b}v^2)' = \boldsymbol{b}'v^2 + 2\boldsymbol{b}v'v$, on obtient

$$\begin{aligned}\int_{\ell_0}^{\ell_1} ((\boldsymbol{b}.v')v \;+\; cv^2\; dx &= \int_{\ell_0}^{\ell_1} \frac{1}{2}(\boldsymbol{b}v^2)' + (c - \frac{1}{2}\boldsymbol{b}')v^2\; dx \\ &= \underbrace{-(\boldsymbol{b}v^2)(\ell_0)}_{=0} + \underbrace{(\boldsymbol{b}v^2)(\ell_1)}_{\geq 0} + \int_{\ell_0}^{\ell_1} \underbrace{(c - \frac{1}{2}\boldsymbol{b}')}_{\geq 0} v^2\; dx \quad \geq 0\end{aligned}$$

D'où

$$a(v,v) \geq \int_\Omega a||\,\mathbf{grad}\,v||^2 dx \geq c_a \int_\Omega ||\,\mathbf{grad}\,v||^2 dx = c_a |v|_{H_1(\Omega)}.$$

Comme la mesure de Γ_d est strictement positive, l'inégalité de Poincaré, nous dit que la semi-norme $|.|_{H_1(\Omega)}$ équivalente u sur X_0.

□

Maintenant nous allons faire l'analyse de l'erreur.

1.11 Principe de la méthode de Galerkin

Soit H_h un sous espace de dimension fini de H,

Définition 1.19. *Notre problème discret est : trouvez $u_h \in H_h$, telle que*

$$\forall v_h \in H_h; \qquad \boldsymbol{a}(u_h, v_h) = l(v_h). \tag{55}$$

Lemme 1.20 (de Cea)**.** *Soit une la forme bilinxéaire $\boldsymbol{a}$ de $H \times H$ (H espace de Hilbert) coercive et continue, c'est a dire qu'il existe deux contantes rééls strictement positives c_m et c_M telles que*

$$\forall (v,w) \in H^2, \qquad ||v||^2 c_m \leq \boldsymbol{a}(v,v) \quad et \quad \boldsymbol{a}(v,w) \leq c_M ||v||\;||w||$$

et si forme linéaire $\boldsymbol{l}$ est continue, alors nous avons l'estimation d'erreur entre u solution de (52) et u_h solution de (55) suivante :

$$\forall v_h \in H_h, \qquad ||u - u_h||_H \leq \frac{c_M}{c_m} ||u - v_h||_H \tag{56}$$

Démonstration. Remarquons la relation orthogonalité suivant

$$\forall w_h \in H_h; \qquad a(u-u_h, w_h) = l(v_h) - l(v_h) = 0 \tag{57}$$

On a pour tout v_h dans H_h

$$\begin{aligned} c_m||u-u_h||_H^2 \leq a(u-u_h, u-u_h) &= a(u-u_h, u-u_h + \underbrace{u_h - v_h}_{w_h \in H_h}) \\ &= a(u-u_h, u-v_h) \\ &\leq c_M||u-u_h||_H \, ||u-v_h||_H \end{aligned}$$

En divisant par $c_m||u-u_h||_H$, on obtient le résultat. □

Ce résultat est fondamental, car il montre que si u est approchable à ε près dans H_h, alors $||u-u_h||$ est de l'ordre de ε.

Dans le cas plus général où on a juste une condition inf-sup, il faut utiliser le premier lemme de Strang que voici :

Lemme 1.21 (Strang)**.** *Sous les mêmes hypothèse que le théorème Banach-Necas-Babuska 1.14 et sous l'Hypothèse de la condition inf-sub discrète :*

$$\inf_{u_h \in H_h} \sup_{v_h \in H_h} \frac{\boldsymbol{a}(u_h, v_h)}{||u_h||_H \, ||v_h||_H} \geq \alpha^* > 0$$

indépendant du paramètre de discrétisation h.

alors on a

$$||u-u_h||_H \leq (1 + \frac{C_M}{\alpha^*})||u-v_h||_H$$

Démonstration. Notre condition inf-sup discret :

$$\alpha^*||u_h - v_h||_H \leq \sup_{w_h \in H_h} \frac{\boldsymbol{a}(u_h - v_h, w_h)}{||w_h||}$$

et La relation d'orthogonalité (57) nous donne

$$\boldsymbol{a}(u_h - v_h, w_h) = \boldsymbol{a}(u_h - u + u - v_h, w_h) = \boldsymbol{a}(u - v_h, w_h)$$

implique

$$\alpha^*||u_h - v_h||_H \leq \sup_{w_h \in H_h} \frac{\boldsymbol{a}(u - v_h, w_h)}{||w_h||} \leq C_M||u - v_h||_H$$

Et donc l'inégalité précédente , plus inégalité triangulaire donne :

$$||u-u_h||_H \leq ||u_h - v_h||_H + ||v_h - u||_H \leq (1 + \frac{C_M}{\alpha^*})||v_h - u||_H$$

□

1.11.1 Estimation a priori

Si la solution est continue, et si on utilise la méthode des éléments finis.

Notons $\mathcal{I}_h(v)$ l'interpolé de v, la fonction de H_h telle que $\mathcal{I}_h(v)(q) = v(q)$ aux $N+1$ points q du maillage, c'est à dire que

$$\mathcal{I}_h(v) = \sum_{i=0}^{N} v(q^i) w_i \tag{58}$$

où les w_i sont les fonctions de base de H_h de la définition 1.9.

Maintenant, nous allons estimé $||u - \mathcal{I}_h u||_{H^1(\Omega)}$, sur un élément $K =]a, b[$, de longueur $h_K = b - a$ (*Attention, a et b non rien avoir avec les coefficients du problème original*).

Notons $W^{2,\infty}(K)$, espace de Sobolev : l'ensemble des fonctions réelles de l'intervalle K à dérivées première et secondes bornées presque partout (ces fonctions sont continues).

Lemme 1.22. *Soit $u, f, g \in W^{2,\infty}(]a, b[)^3$ telles que $f(a) \leq u(a) \leq g(a)$, $f(b) \leq u(b) \leq g(b)$ et telles que $f'' \geq u'' \geq g''$, alors nous avons $f \leq u \leq g$.*

Démonstration. La preuve est basée sur la remarque qu'une fonction convexe sur $[a, b]$ et négative en a et b est négative sur $[a, b]$ entier. Pour finir, utilisons cette remarque avec $f - u$ et $u - g$. □

Soit $[a, b]$ un intervalle de $\mathbb{R}$, et soit $u \in W^{2,\infty}(]a, b[)$ une fonction continue, soit $\mathcal{I}_1(u)$ l'interpolation P_1 de u , l'interpolé est défini par $\mathcal{I}_1(u) : t \mapsto (1-t)u(a) + tu(b)$.

Pour obtenir les majorations d'erreurs classiques, il suffit utiliser comme fonction majorante (resp. minorante) la fonction $f(x) = \frac{1}{2}\inf_K(u'')(x-a)(x-b)$ (resp. $g(x) = \frac{1}{2}\sup_K(u'')(x-a)(x-b)$). Le lemme 1.22 nous donne l'encadrement suivant :

$$\frac{1}{2}\sup_K(u'')(x-a)(x-b) \quad \leq \quad (u - \mathcal{I}_1 u) \quad \leq \quad \frac{1}{2}\inf_K(u'')(x-a)(x-b) \tag{59}$$

si $u \in W^{2,\infty}(]a, b[)$.

On en déduit, l' erreur d'interpolations P_1 dans trois normes classique sur l'élément K :

$$\|u - \mathcal{I}_1 u\|_{L^\infty} \leq \frac{h_K^2}{8}||u''||_\infty \tag{60}$$

$$\|u - \mathcal{I}_1 u\|_{L^1} \leq \frac{h_K^3}{12}||u''||_\infty \tag{61}$$

$$\|u - \mathcal{I}_1 u\|_{L^2} \leq \frac{h_K^{\frac{5}{2}}}{2\sqrt{30}}||u''||_\infty \tag{62}$$

Pour calculer l'estimation d'erreur en semi norme $H^1(K)$, remarquons que $f = u - \mathcal{I}_1 u$ est nulle en a, b, et donc en intégrant par partie on a $\int_a^b f'^2 = -\int_a^b f f''$, d'où l'inégalité $\int_a^b f'^2 \leq ||f''||_{L^\infty}\int_a^b |f|$.

ce qui nous donne :

$$|u - \mathcal{I}_1 u|_{H^1} = \|(u - \mathcal{I}_1 u)'\|_{L^2} \leq \frac{h_K^{\frac{3}{2}}}{2\sqrt{3}} |u''|_{L^\infty} \tag{63}$$

Théorème 1.23. *Pour une famille d'espace d'éléments finis X_h définie en (34) au paragraphe 1.6, nous avons les estimations d'erreur suivante :*

$$\forall u \in W^{2,\infty}(\Omega), \qquad |u - \mathcal{I}_h u|_{H^1(\Omega)} \leq \frac{\sqrt{|\Omega|}}{2\sqrt{3}}\, h\, |u''|_{L^\infty(\Omega)}. \tag{64}$$

$$||u - \mathcal{I}_h u||_{L^2(\Omega)} \leq \frac{\sqrt{|\Omega|}}{2\sqrt{30}}\, h^2\, |u''|_{L^\infty(\Omega)}. \tag{65}$$

Démonstration. Il suffit utiliser l'estimation de l'inéquation (63)

$$|u - \mathcal{I}_h u|^2_{H^1(\Omega)} = \sum_{K \in \mathcal{T}_{d,h}} \int_K |(u - \mathcal{I}_h u)'|^2 dx \leq \sum_{K \in \mathcal{T}_{d,h}} \Big(\frac{h_K^{\frac{3}{2}}}{2\sqrt{3}} |u''|_{L^\infty(K)} \Big)^2$$

Donc

$$|u - \mathcal{I}_h u|^2_{H^1(\Omega)} \leq (\sum_{K \in \mathcal{T}_{d,h}} h_K) \Big(\frac{h}{2\sqrt{3}} |u''|_{L^\infty(\Omega)} \Big)^2 \leq (\sqrt{|\Omega|} \frac{h}{2\sqrt{3}} |u''|_{L^\infty(\Omega)})^2$$

On démontre, la seconde inégalité en utilisant la même technique. □

2 C++, quelques éléments de syntaxe

Il y a tellement de livres sur la syntaxe du C++ qu'il me paraît déraisonnable de réécrire un chapitre sur ce sujet, je vous propose le livre de Thomas Lachand-Robert qui est disponible sur la toile à l'adresse suivante http://www.ann.jussieu.fr/courscpp/, ou le cours C,C++ plus moderne aussi disponible sur la toile http://casteyde.christian.free.fr/cpp/cours. Pour avoir une liste à jour lire la page http://www.developpez.com/c/cours/. Bien sur, vous pouvez utiliser le livre The C++ , programming language [Stroustrup-1997]

Je veux décrire seulement quelques trucs et astuces qui sont généralement utiles comme les déclarations des types de bases et l'algèbre de typage.

Donc à partir de ce moment je suppose que vous connaissez, quelques rudiments de la syntaxe C++ . Ces rudiments que je sais difficile, sont (pour les connaître, il suffit de comprendre ce qui est écrit après) :

– le lexique infomatique à connaitre par coeur :

Mémoire
Octet
Type
Adresse
Ecriture mémoire
Lecture mémoire
Affectation
Fonction
Paramètre
Pointeur
Référence
Argument
Variable
Polymorphisme
Surcharge d'opérateur
Méthode
Objet
Classe
Instance
Opérateur
Conversion
Sémantique
Mémoire cache
Mémoire secondaire
Terminal
Bibliothèque
Edition de lien
Compilation
Répertoire
Make
Editeur de texte
Operateur de copie
Affectation par copie
Conversion
Evaluation
Execution
Executable
Shell script
Processus
Entre-Sortie
Fichier
Dynamique
Statique
Iteration
Boucle
Recursivité
Implementation
Généricité
Hérédité
Algorithme
Pile
Debugger
Compilateur
Préprocesseur
Complexité
url

Exemple de phase à comprendre : Une variable locale et dynamique d'une fonction n'existe en mémoire que pendant l'exécution la fonction, si la fonction est récursive cette fonction peux être plusieurs fois en mémoire comme cette variable, lorsqu'une variable locale statique d'une fonction existe toujours pendant le temps d'exécution et est unique.

- Les types de base, les définitions de pointeur et référence (je vous rappelle qu'une référence est défini comme une variable dont l'adresse mémoire est connue et cet adresse n'est pas modifiable, donc une référence peut être vue comme une pointeur constant automatiquement déférencé, ou encore comme « donné un autre nom à une zone mémoire de la machine »).
- L'écriture d'un fonction, d'un prototypage,
- Les structures de contrôle associée aux mots clefs suivants : `if, else, switch, case, default, while, for, repeat, continue, break`.
- L'écriture d'une classe avec constructeur et destructeur, et des méthodes membres.
- Les passages d'arguments
  - par valeur (type de l'argument sans `&`), donc une copie de l'argument est passée à la fonction. Cette copie est créée avec le constructeur par copie, puis est détruite avec le destructeur. L'argument ne peut être modifié dans ce cas.
  - par référence (type de l'argument avec `&`) donc l'utilisation du constructeur par copie.
  - par pointeur (le pointeur est passé par valeur), l'argument peut-être modifié.
  - paramètre non modifiable (cf. mot clef `const`).
  - La valeur retournée par copie (type de retour sans `&`) ou par référence (type de retour avec `&`)
- Polymorphisme et surcharge des opérateurs. L'appel d'une fonction est déterminée par son nom et par le type de ses arguments, il est donc possible de créer des fonctions de même nom pour des type différents. Les opérateurs n-naire (unaire `n=1` ou binaire `n=2`) sont des fonctions à `n` argument de nom `operator` ♣ `(n-args )` où ♣ est l'un des opérateurs du `C++` :

```
+    -    *    /    %    ^    & |    ~    !    =    <       >       +=
-=   *=   /=   %=   ^=   &=   |= <<   >>   <<=  >>=  ==      !=      <=
>=   &&   ||   ++   --   ->*  , ->   []   ()   new  new[]   delete  delete[]
(T)
```

  où (T est une expression de type), et où `(n-args )` est la déclaration classique des `n` arguments. Remarque si opérateur est défini dans une classe alors le premier argument est la classe elle même et donc le nombre d'arguments est $n-1$.
- Les règles de conversion (« cast » en Anglais) d'un type T en A par défaut qui sont génèrées à partir d'un constructeur `A(T)` dans la classe A ou avec l'opérateur de conversion `operator (A) ()` dans la classe T, `operator (A) (T)` hors d'une classe. De plus il ne faut pas oublier que `C++` fait automatiquement un au plus un niveau de conversion pour trouver la bonne fonction ou le bon opérateurs.
- Programmation générique de base (c.f. `template`). Exemple d'écriture de la fonction `min` générique suivante `template<class T> T & min(T & a,T & b){return a<b? a :b;}`
- Pour finir, connaître seulement l'existence du macro générateur et ne pas l'utiliser.

2.1 Les déclarations du C++

Les types de base du `C++` sont respectivement : `char`, `short`, `int`, `long`, `long long`, `float`, `double`, plus des pointeurs, ou des références sur ces types, des tableaux, des fonctions

sur ces types et les constantes (objet non modifiable). Le tout nous donne une algèbre de type qui n'est pas triviale.

Voilà les principaux types généralement utilisé pour des types `T`,`U` :

déclaration	Prototypage	description du type en français
`T * a`	`T *`	un pointeur sur `T`
`T a[10]`	`T[10]`	un tableau de `T` composé de 10 variable de type T
`T a(U)`	`T a(U)`	une fonction qui a `U` retourne un `T`
`T &a`	`T &a`	une référence sur un objet de type `T`
`const T a`	`const T`	un objet constant de type `T`
`T const * a`	`T const *`	un pointeur sur objet constant de type `T`
`T * const a`	`T * const`	un pointeur constant sur objet de type `T`
`T const * const a`	`T const * const`	un pointeur constant sur objet constant
`T * & a`	`T * &`	une référence sur un pointeur sur `T`
`T ** a`	`T **`	un pointeur sur un pointeur sur `T`
`T * a[10]`	`T *[10]`	un tableau de 10 pointeurs sur `T`
`T (* a)[10]`	`T (*)[10]`	un pointeur sur tableau de 10 `T`
`T (* a)(U)`	`T (*)(U)`	un pointeur sur une fonction `U` $\rightarrow$`T`
`T (* a[])(U)`	`T (*[])(U)`	un tableau de pointeur sur des fonctions `U`$\rightarrow$`T`
`T C:: *a`	`T C:: *`	pointeur sur une membre de type `T` dans la classe `C`
`T (C:: *a)(U)`	`T (C:: *)(U)`	pointeur sur une méthode membre `U` $\rightarrow$`T` dans la classe `C`

...

Remarque il n'est pas possible de construire un tableau de référence car il sera impossible à initialiser.

Exemple d'allocation d'un tableau data de *ldata* pointeurs de fonctions de R à valeur dans R :

```
R (**data)(R) = new (R (*[ldata])(R));
```

ou encore avec déclaration et puis allocation :

```
R (**data)(R); data = new (R (*[ldata])(R));
```

Pour l'utilisation des pointeurs sur des membres de classe `C`, (voir le point [Stroustrup-1997, C.12, page 853,] pour plus de details).

```
T (C:: *a)(U) = & C::func_T2U;
T C:: *a = & C:val_T;
```

Où `C` est une classe avec les deux membres `func_T2U` et `data_T`.

2.2 Comment Compile et éditer de liens

Comme en `C` ; dans les fichiers `.cpp`, il faut mettre les corps des fonctions et de les fichiers `.hpp`, il faut mettre les prototype, et la définition des classes, ainsi que les fonctions `inline` et les fonctions `template`, Voilà, un exemple complète avec trois fichiers `a.hpp`,`a.cpp`,`tt.hpp`, et un Makefile dans `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/l1/a.tar.gz` .

Remarque, pour déarchiver un fichier `xxx.tar.gz`, il suffit d'entrer dans une fenêtre shell `tar zxvf xxx.tar.gz`.

Listing 1: *(a.hpp)*

```
class A { public:
  A();                                          //    constructeur de la class A
};
```

Listing 2: *(a.cpp)*

```
#include <iostream>
#include "a.hpp"
using namespace std;

A::A()
{
  cout << " Constructeur A par défaut " << this << endl;
}
```

Listing 3: *(tt.cpp)*

```
#include <iostream>
#include "a.hpp"
using namespace std;
int main(int argc,char ** argv)
{
  for(int i=0;i<argc;++i)
    cout << " arg " << i << " = " << argv[i] << endl;
  A a[10];                                                    //    un tableau de 10 A
  return 0;                                                                  //    ok
}
```

les deux compilations et l'édition de liens qui génère un executable `tt` dans une fenêtre `Terminal`, avec des commandes de type `shell; sh, bash tcsh, ksh, zsh, ...` , sont obtenues avec les trois lignes :

```
[brochet:P6/DEA/sfemGC] hecht% g++ -c a.cpp
[brochet:P6/DEA/sfemGC] hecht% g++ -c tt.cpp
[brochet:P6/DEA/sfemGC] hecht% g++ a.o tt.o -o tt
```

Pour faire les trois choses en même temps, entrez :

```
[brochet:P6/DEA/sfemGC] hecht% g++  a.cpp tt.cpp -o tt
```

Puis, pour executer la commande `tt`, entrez par exemple :

```
[brochet:P6/DEA/sfemGC] hecht% ./tt aaa bb cc dd qsklhsqkfhsd " ----------
 arg 0 = ./tt
 arg 1 = aaa
 arg 2 = bb
 arg 3 = cc
 arg 4 = dd
 arg 5 = qsklhsqkfhsd
 arg 6 =  ----------
 Constructeur A par défaut 0xbfffeef0
 Constructeur A par défaut 0xbfffeef1
 Constructeur A par défaut 0xbfffeef2
 Constructeur A par défaut 0xbfffeef3
 Constructeur A par défaut 0xbfffeef4
 Constructeur A par défaut 0xbfffeef5
 Constructeur A par défaut 0xbfffeef6
 Constructeur A par défaut 0xbfffeef7
 Constructeur A par défaut 0xbfffeef8
 Constructeur A par défaut 0xbfffeef9
```

remarque : les `aaa bb cc dd qsklhsqkfhsd " ---------- "` après la commande `./tt` sont les paramètres de la commande est sont bien sur facultatif.

remarque, il est aussi possible de faire un `Makefile`, c'est à dire de créé le fichier :

Listing 4: *(Makefile)*

```
CXX=g++
CXXFLAGS= -g
LIB=
%.o:%.cpp
(caractere de tabulation  -->/)  $(CXX) -c $(CXXFLAGS) $^
tt: a.o tt.o
(caractere de tabulation  -->/)  $(CXX) tt.o  a.o -o tt
clean:
(caractere de tabulation  -->/) rm *.o tt

# les dependences
#
a.o: a.hpp  # il faut recompilé a.o si a.hpp change
tt.o: a.hpp  # il faut recompilé tt.o si a.hpp change
```

Pour l'utilisation :
– pour juste voir les commandes exécutées sans rien faire :

```
[brochet:P6/DEA/sfemGC] hecht% make -n tt
g++  -c  tt.cpp
g++  -c  a.cpp
g++  -o tt tt.o a.o
```

– pour vraiment compiler

```
[brochet:P6/DEA/sfemGC] hecht% make  tt
g++  -c  tt.cpp
g++  -c  a.cpp
g++  -o tt tt.o a.o
```

– pour recompiler avec une modification du fichier `a.hpp` via la command `touch` qui change la date de modification du fichier.

```
[brochet:P6/DEA/sfemGC] hecht% touch a.hpp
[brochet:P6/DEA/sfemGC] hecht% make tt
g++  -c  tt.cpp
g++  -c  a.cpp
g++  -o tt tt.o a.o
```

remarque : les deux fichiers sont bien à recompiler car il font un *include* du fichier `a.hpp`.

– pour nettoyer :

```
[brochet:P6/DEA/sfemGC] hecht% touch a.hpp
[brochet:P6/DEA/sfemGC] hecht% make clean
rm *.o tt
```

Remarque : Je vous conseille très vivement d'utiliser un `Makefile` pour compiler vous programme.

Exercice 8. Ecrire un Makefile pour compile et tester tous les programmes `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/l1/exemple_de_base`

2.3 Compréhension des constructeurs, destructeurs et des passages d'arguments

Exercice 9.

Faire une classe `T` avec un constructeur par copie et le destructeur, et la copie par affectation : qui imprime quelque chose comme par exemple :

```
class T { public:
  T() { cout <<  "Constructeur par défaut " << this << "\n"}
  T(const & T a) { cout <<"Constructeur par copie "
                        << this << "\n"}
  ~T() { cout <<  "destructeur "<< this << "\n"}
  T & operator=(T & a) {cout << " copie par affectation :"
                        << this << " = " << &a << endl;}
  };
```

Puis tester, cette classe faisant un programme qui appelle les 4 fonctions suivantes, et qui contient une variable globale de type `T`.

```
T f1(T a){ return a;}
T f2(T &a){ return a;}
T &f3(T a){ return a;}            //    il y a un bug, le quel?
T &f4(T &a){ return a;}
```

Analysé et discuté très finement les résultat obtenus, et comprendre pourquoi, la classe suivant ne fonctionne pas, ou les opérateur programme font la même chose que les opérateurs par défaut.

```
class T { public:
  int * p;                                        //    un pointeur
  T() {p=new int;
       cout <<  "Constructeur par defaut " << this
            << " p=" << p << "\n"}
  T(const & T a) {
       p=a.p;
       cout <<  "Constructeur par copie "<< this
            << " p=" << p << "\n"}
  ~T() {
       cout <<  "destructeur "<< this
            << " p=" << p << "\n";
       delete p;}
  T & operator=(T & a) {
     cout << "copie par affectation  "<< this
          << "old  p=" << p
          <<  "new p="<< a.p << "\n";
     p=a.p; }
  };
```

remarque : bien sur vous pouvez et même vous devez tester ses deux classes, avec le vérificateur d'allocation dynamique.

2.4 Quelques règles de programmation

Malheureusement, il est très facile de faire des erreurs de programmation, la syntaxe du C++ n'est pas toujours simple à comprendre et comme l'expressibilité du langage est très grande, les possibilités d'erreur sont innombrables. Mais avec un peu de rigueur, il est possible d'en éviter un grand nombre.

La plupart des erreurs sont dû à des problèmes des pointeurs (débordement de tableau, destruction multiple, oublie de destruction), retour de pointeur sur des variable locales.

Voilà quelques règles à respecté.

Règle 1. *absolue* Dans une classe avec des pointeurs et avec un destructeur, il faut que les deux opérateurs de copie (création et affection) soient définis. Si vous considérez que ces deux opérateurs ne doivent pas exister alors les déclarez en privé sans les définir.

```
class sans_copie { public:
   long * p;                                                    //    un pointeur
   . . .
   sans_copie();
   ~sans_copie() { delete p;}
   private:
   sans_copie(const sans_copie &);              //    pas de constructeur par copie
   void operator=(const sans_copie &);             //    pas d'affection par copie
};
```

Dans ce cas les deux opérateurs de copies ne sont pas programmer pour qu'une erreur à l'édition des liens soit généré.

```
class avec_copie { public:
   long * p;                                                    //    un pointeur
   ~avec_copie() { delete p;}
   . . .
   avec_copie();
   avec_copie(const avec_copie &);           //    construction par copie possible
   void operator=(const avec_copie &);          //    affection par copie possible
};
```

Par contre dans ce cas, il faut programmer les deux opérateurs construction et affectation par copie.

Effectivement, si vous ne définissez ses opérateurs, il suffit d'oublier une esperluette (&) dans un passage argument pour que plus rien ne marche, comme dans l'exemple suivante :

```
  class Bug{ public:
   long * p;                                                    //    un pointeur
   Bug() p(new long[10]);
   ~Bug() { delete p;}
  };
  long & GetPb(Bug   a,int i){ return a.p[i];}                    //    copie puis
                                                          //    destruction de la copie
```

```
long & GetOk(Bug & a,int i){ return a.p[i];}                    //    ok

int main(int argc,char ** argv) {
 bug a;
 GetPb(a,1) = 1;                          //    bug le pointeur a.p est détruit ici
                                            //    l'argument est copie puis détruit
 cout << GetOk(a,1) << "\n";          //    bug on utilise un zone mémoire libérée

 return 0;                                //    le pointeur a.p est encore détruit ici
}
```

Le pire est que ce programme marche sur la plupart des ordinateurs et donne le résultat jusqu'au jour où l'on ajoute du code entre les 2 get (2 ou 3 ans après), c'est terrible mais ça marchait !...

Règle 2. Dans une fonction, ne jamais retournez de référence ou le pointeur sur une variable locale

Effectivement, retourner une référence sur une variable local implique que l'on retourne l'adresse mémoire de la pile, qui est libéré automatique en sortie de fonction, et qui est donc invalide hors de la fonction. mais bien sur le programme écrire peut marche avec de la chance.

Il ne faut jamais faire ceci :

```
int & Add(int i,int j)
{ int l=i+j;
  return l; }                              //    bug return d'une variable local l
```

Mais vous pouvez retourner une référence définie à partir des arguments, ou à partir de variables static ou global qui sont rémanentes.

Règle 3. Si, dans un programme, vous savez qu'un expression logique doit être vraie, alors vous devez mettre une assertion de cette expression logique.

Ne pas penser au problème du temps calcul dans un premier temps, il est possible de retirer toutes les assertions en compilant avec l'option `-DNDEBUG`, ou en définissant la macro du preprocesseur `#define NDEBUG`, si vous voulez faire du filtrage avec des assertions, il suffit de définir les macros suivante dans un fichier `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/assertion.hpp` qui active les assertions

```
#ifndef ASSERTION_HPP_
#define ASSERTION_HPP_
//    to compile all assertion
//   #define ASSERTION
//    to remove all the assert
//   #define NDEBUG
#ifndef ASSERTION
#define ASSERTION(i)  0
#else
#include <cassert>
#undef ASSERTION
#define ASSERTION(i) assert(i)
#endif
#endif
```

comme cela il est possible de garder un niveau d'assertion avec `assert`. Pour des cas plus fondamentaux et qui sont négligeables en temps calcul. Il suffit de définir la macro `ASSERTION` pour que les testes soient effectués sinon le code n'est pas compilé et est remplacé par `0`.
Il est fondamental de vérifier les bornes de tableaux, ainsi que les autres bornes connues. Aujourd'hui je viens de trouver une erreur stupide, un déplacement de tableau dû à l'échange de 2 indices dans un tableau qui ralentissait très sensiblement mon logiciel (je n'avais respecté cette règle).
Exemple d'une petite classe qui modélise un tableau d'entier

```
class Itab{ public:
  int n;
  int *p;
  Itab(int nn)
   { n=nn);
     p=new int[n];
     assert(p);}                                    //    vérification du pointeur
  ~Itab()
   { assert(p);                                     //    vérification du pointeur
     delete p;
     p=0;}                                  //    pour éviter les doubles destruction
  int & operator[](int i) { assert( i >=0 && i < n && p ); return p[i];}

private:       //    la règle 1 : pas de copie par défaut il y a un destructeur
  Itab(const Itab &);                           //    pas de constructeur par copie
  void operator=(const Itab &);                 //    pas d'affection par copie
}
```

Règle 4. N'utilisez le macro générateur que si vous ne pouvez pas faire autrement, ou pour ajoutez du code de vérification ou test qui sera très utile lors de la mise au point.

Règle 5. Une fois toutes les erreurs de compilation et d'édition des liens corrigées, il faut éditer les liens en ajoutant `CheckPtr.o` (le purify du pauvre) à la liste des objets à éditer les liens, afin de faire les vérifications des allocations.

Corriger tous les erreurs de pointeurs bien sûr, et les erreurs assertions avec le débogueur.

2.5 Vérificateur d'allocation

L'idée est très simple, il suffit de surcharger les opérateurs **new** et **delete**, de stocker en mémoire tous les pointeurs alloués et de vérifier avant chaque déallacation s'il fut bien alloué (cf. `AllocExtern::MyNewOperator(size_t )` et `AllocExternData.MyDeleteOperator(void *)`.
Le tout est d'encapsuler dans une classe `AllocExtern` pour qu'il n'y est pas de conflit de nom. De plus, on utilise `malloc` et `free` du C, pour éviter des problèmes de récurrence infinie dans l'allocateur. Pour chaque allocation, avant et après le tableau, deux petites zones mémoire de 8 octets sont utilisées pour retrouver des débordement amont et aval.
Et le tout est initialisé et terminé sans modification du code source en utilisant la variable `AllocExternData` globale qui est construite puis détruite. À la destruction la liste des pointeurs non détruits est écrite dans un fichier, qui est relue à la construction, ce qui permet de déboguer les oublis de déallocation de pointeurs.

Remarque 4. *Ce code marche bien si l'on ne fait pas trop d'allocations, destructions dans le programme, car le nombre d'opérations pour vérifier la destruction d'un pointeur est en nombre de pointeurs alloués. L'algorithme est donc proportionnel au carré du nombre de pointeurs alloués par le programme. Il est possible d'améliorer l'algorithme en triant les pointeurs par adresse et en faisant une recherche dichotomique pour la destruction.*

Le source de ce vérificateur `CheckPtr.cpp` est disponible à l'adresse suivante `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/CheckPtr.cpp`. Pour l'utiliser, il suffit de compiler et d'éditer les liens avec les autres parties du programme.
Il est aussi possible de retrouver les pointeurs non désalloués, en utilisant votre débogueur favori (par exemple gdb).
Dans `CheckPtr.cpp`, il y a une macro du préprocesseur `DEBUGUNALLOC` qu'il faut définir, et qui active l'appel de la fonction `debugunalloc()` à la création de chaque pointeur non détruit. La liste des pointeurs non détruits est stockée dans le fichier `ListOfUnAllocPtr.bin`, et ce fichier est génèré par l'exécution de votre programme.
Donc pour déboguer votre programme, il suffit de faire :

1. Compilez `CheckPtr.cpp`.
2. Editez les liens de votre programme `C++` avec `CheckPtr.o`.
3. Exécutez votre programme avec un jeu de donné.
4. Réexécutez votre programme sous le debogueur et mettez un point d'arrêt dans la fonction `debugunalloc()` (deuxième ligne `CheckPtr.cpp` .
5. remontez dans la pile des fonctions appelées pour voir quel pointeur n'est pas désalloué.
6. etc...

Exercice 10. un Makefile pour compiler et tester tous les programmes du `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/l1/exemple_de_base` avec `CheckPtr`

2.6 Le déboggeur en 5 minutes

Premièrement, le déboggeur est un programme `gdb` ou `ddd`, qui permet d'ausculter votre programme, il vous permettra peut être de retrouver des erreurs de programmations qui génèrent des erreurs à l'exécution.
Pour utiliser le déboggeur il faut premièrement compiler les programmes avec l'option de compilation `-g`, pour cela il suffit de l'ajouter à la variable `CXXFLAGS` dans le `Makefile`.

Voilà la liste des commandes les plus utiles à mon avis :

`run` lance ou relance le programme à débogger, (ajouter les paramètres de la commande après `run`.

`c` continue l'exécution

`s` fait un pas d'une instruction (en entant dans les fonctions)

`n` fait un pas d'une instruction (sans entrer dans les fonctions)

`finish` continue l'exécution jusqu'à la sortie de la fonction (`return`)

u sort de la boucle courante

p print la valeur d'une variable, expression ou tableau : `p x` ou `p *v@100` (affiche les 100 valeur du tableau défini par le pointeur `v`).

where montre la pile des appels

up monte dans la pile des appels

down descend dans la pile des appels

l listing de la fonction courante

l 10 listing à partir de la ligne 10.

info functions affiche toutes les fonctions connues, et `info functions tyty` n'affiche que les fonctions dont le nom qui contient la chaîne `tyty`,

info variables même chose mais pour les variables.

b main.cpp:100 défini un point d'arrêt en ligne 100 du fichier `main.cpp`

b zzz défini un point d'arrêt à l'entrée de la fonction `zzz`

b A::A() défini un point d'arrêt à l'entrée du constructeur par défaut de la classe A.

d 5 détruit le 5^{e} point d'arrêt.

watch défini une variable à tracer, le programme va s'arrêter quand cette variable va changer.

help pour avoir plus d'information en anglais.

Une petite ruse bien pratique pour débogger, un programme avant la fin (`exit`). Pour cela ajouter une fonction `zzzz` et utiliser la fonction `atexit` du système. C'est à dire qui votre programme doit ressembler à :

```
#include <cstdlib>
void zzzz(){}                          //    fonction vide qui ne sert qu'à débogger
int main(int argc,char **argv)
{
  atexit(zzzz);                        //    pour que la fonction zzzz soit exécutée
                                       //    avant la sortie en exit
```

sous gdb ou ddd entrez

```
  b zzzz
```

pour ajouter un point d'arrêt à l'appel de la fonction `zzzz`.

Pour finir, voilà, un petit exemple d'utilisation :

```
brochet:~/work/Cours/InfoBase/l4 hecht$ gdb mainA2
```

```
GNU gdb 6.3.50-20050815 (Apple version gdb-563)
....  bla bla ...
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-apple-darwin"...Reading symbols
for shared libraries .... done
```
(gdb) b main
```
Breakpoint 1 at 0x2c79: file Amain.cpp, line 16.
```
(gdb) b zzzz()
```
Breakpoint 2 at 0x2c36: file Amain.cpp, line 11.
```
(gdb) b Amain.cpp:19
```
Breakpoint 3 at 0x2c8e: file Amain.cpp, line 19.
```
(gdb) run
```
Starting program: /Users/hecht/work/Cours/InfoBase/l4/mainA2
Reading symbols for shared libraries . done
Breakpoint 1, main () at Amain.cpp:16
16        atexit(zzzz);
```
(gdb) c
```
Continuing.
Breakpoint 3, main () at Amain.cpp:19
19        A a(n),b(n),c(n),d(n);
```
(gdb) s *on fait un step: on entre dans le constructeur*
```
A::A (this=0xbffff540, i=100000) at A2.hpp:11
11        A(int i) : n(i),v(new K[i]) { assert(v);} // constructeur
```
(gdb) finish *on sort du constructeur*
```
Run till exit from #0  A::A (this=0xbffff540, i=100000) at A2.hpp:11
0x00002ca0 in main () at Amain.cpp:19
19        A a(n),b(n),c(n),d(n);
```
(gdb) n *on fait un pas sans entrer dans les fonctions*
```
21        for(int i=0;i<n;++i)
```
(gdb) l *Affichage du source de la fonction autour du point courant*
```
16        atexit(zzzz);
17
18        int n=100000;
19        A a(n),b(n),c(n),d(n);
20        //  initialisation des tableaux a,b,c
21        for(int i=0;i<n;++i)
22          {
23            a[i]=cos(i);
24            b[i]=log(i);
25            c[i]=exp(i);
```
(gdb) *suite de l'Affichage du source*
```
26          }
27          d = (a+2.*b)+c*2.0;
28      }
```
(gdb) b 27 *defini un point d'arrêt en ligne 27 après la fin de boucle*
```
Breakpoint 4 at 0x2d50: file Amain.cpp, line 27.
```
(gdb) c *continue*
```
Continuing.
Breakpoint 4, main () at Amain.cpp:27
27          d = (a+2.*b)+c*2.0;
```

(gdb) d 4 *supprime le point d'arrêt n°4*
(gdb) p d *affiche la valeur de d*

```
$1 = {
  n = 100000,
  v = 0x4c9008
}
```

(gdb) p d.v *affiche la valeur de d.v*

```
$2 = (double *) 0x4c9008
```

(gdb) p *d.v@10 *affiche les 10 valeurs pointées par d.v*

```
$4 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

(gdb) p *a.v@10 *affiche les 10 valeurs pointées par a.v*

```
$5 = {1, 0.54030230586813977, -0.41614683654714241, -0.98999249660044542,
 -0.65364362086361194, 0.28366218546322625, 0.96017028665036597,
  0.7539022543433046, -0.14550003380861354,
  -0.91113026188467694}
```

(gdb) p *b.v@10 *affiche les 10 valeurs pointées par b.v*

```
$6 = {-inf, 0, 0.69314718055994529, 1.0986122886681098, 1.3862943611198906,
1.6094379124341003, 1.791759469228055, 1.9459101490553132, 2.0794415416798357,
2.1972245773362196}
```

on remarquera que log(0) == -inf et ne génère par d'erreur

(gdb) p *c.v@10 *affiche les 10 valeurs pointées par c.v*

```
$7 = {1, 2.7182818284590451, 7.3890560989306504, 20.085536923187668,
54.598150033144236, 148.4131591025766, 403.42879349273511,
1096.6331584284585, 2980.9579870417283,  8103.0839275753842}
```

(gdb) c

```
Continuing.
Breakpoint 2, zzzz () at Amain.cpp:11
11        cout << " At exit " << endl;
```

(gdb) c

```
Continuing.
 At exit
                CheckPtr:Max Memory used   6250.000 kbytes  Memory undelete
0
Program exited normally.
```

(gdb) quit *On sort de gdb*

3 Exemples

3.1 Le Plan $\mathbb{R}^2$

Voici une modélisation de $\mathbb{R}^2$ disponible à `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/l1/R2.hpp` qui permet de faire des opérations vectorielles et qui définit le produit scalaire de deux points A et B, le produit scalaire sera défini par (A, B).

3.1.1 La classe R2

```
//    Définition de la class R2
//    sans compilation sépare toute les fonctions
//    sous défini dans ce R2.hpp avec des inline
                                                                          //
//    remarque la fonction abort est déclaré dans #include <cstdlib>

//    definition R (les nombres reals)
typedef double R;                                  //    définition de R (les réel)

//    The class R2
class R2 {

public:
  R x,y;                                                 //    declaration des membre
                                                        //    les 3 constructeurs ---
  R2 () :x(0.),y(0.) {}                 //    rappel : x(0), y(0) sont initialiser
   R2 (R a,R b):x(a),y(b)  {}                    //    via le constructeur de double
  R2 (const R2 & a,const R2 & b):x(b.x-a.x),y(b.y-a.y)  {}

  //    le constucteur par defaut est inutile
                                        //    R2 ( const R2 & a) :x(a.x),y(a.y)

  //    rappel les operator definis dans une class on un parametre
  //    caché qui est la class elle meme (*this)

  //    les opérateurs affectations
  //    operateur affection = est inutil car par défaut,il est correct
          //    R2 & operator=( const R2 & P) x = P.x;y = P.y;return *this;
                                        //    les autre opérateur d'affectations
  R2 &  operator+=(const R2 & P)  {x += P.x;y += P.y;return *this;}
  R2 &  operator-=(const R2 & P) {x -= P.x;y -= P.y;return *this;}
  R2 &  operator*=(R a)  {x *= a;y *= a;return *this;}
  R2 &  operator/=(R a)  {x /= a;y /= a;return *this;}
  //    les operateur binaire + - * , ^
  R2   operator+(const R2 & P)const  {return R2(x+P.x,y+P.y);}
  R2   operator-(const R2 & P)const {return R2(x-P.x,y-P.y);}
  R    operator,(const R2 & P)const {return  x*P.x+y*P.y;} //    produit scalaire
  R    operator^(const R2 & P)const {return  x*P.y-y*P.x;}      //    produit det
  R2   operator*(R c)const {return R2(x*c,y*c);}
  R2   operator/(R c)const {return R2(x/c,y/c);}
                                                        //    operateur unaire
  R2   operator-()const  {return R2(-x,-y);}
  R2   operator+()const  {return *this;}
                                                              //    un methode
  R2   perp() const {return R2(-y,x);}                  //    la perpendiculaire
```

```
  //    les operators tableau
  //    version qui peut modifie la class via l'adresse de x ou y
  R & operator[](int i) { if(i==0) return x;                  //    donc pas const
                          else if (i==1) return y;
                          else {assert(0);exit(1);};}
  //    version qui retourne une reference const qui ne modifie pas la class
  const R & operator[](int i) const { if(i==0) return x;       //    donc  const
                                      else if (i==1) return y;
                                      else {assert(0);exit(1);};}

  //    les operateurs fonction
                  //    version qui peut modifie la class via l'adresse de x ou y
  R & operator()(int i) { if(i==1) return x;                  //    donc pas const
                          else if (i==2) return y;
                          else {assert(0);abort();};}
      //    version qui retourne une reference const qui ne modifie pas la class
  const R & operator()(int i) const { if(i==1) return x;       //    donc  const
                                      else if (i==2) return y;
                                      else {assert(0);abort();};}

};                                                            //    fin de la class R2

//    la class dans la class
inline std::ostream& operator <<(std::ostream& f, const R2 & P )
       { f << P.x << ' ' << P.y  ; return f; }
inline std::istream& operator >>(std::istream& f,  R2 & P)
       { f >>  P.x >>  P.y ; return f; }

inline R2 operator*(R c,const R2 & P) {return P*c;}
inline R2 perp(const R2 & P) { return R2(-P.y,P.x); }
inline R2 Perp(const R2 & P) { return P.perp(); }    //    un autre fonction perp
```

Quelques remarques sur la syntaxe

- Dans une classe (`class`) par défaut, toutes les définitions sont dans la sphère privé, c'est-à-dire quelle ne sont utilisable que par les méthodes de la classe ou les amis (c.f. `friend`) . L'ajout de `public:` permet de dire que tout ce qui suit est dans la sphère publique donc utilisable par tous. Et la seule différence entre un objet défini par `struct` et par `class` est la sphère d'utilisation par défaut qui est respectivement publique (`pubic`) ou privée (`private`) .
- Les membres de la classe (les données) ici sous les champs `x` et `y`. Ces 2 champs seront construction à la création de la classe par les constructeurs du la classe. Les constructeurs sont des méthodes qui ont pour nom, le nom de la classe, ici `R2`. Dans le constructeur `R2 (R a,R b):x(a),y(b) {}` les objets sont initialise par les constructeurs des membres entre le `:` et `{` des membres de la classe qu'il faut mettre dans le même ordre que l'ordre d'apparition dans la classe. . Le constructeur ici ne fait initialiser les deux champs `x` et `y`, il n'y a pas de code dans le corps (zone entre les `{}`) du constructeur.
- Les méthodes (fonctions membres) dans une classe ou structure ont un paramètre caché qui est le pointeur sur l'objet de nom `this`. Donc, les opérateurs binaires dans une classe n'ont seulement qu'un paramètre. Le premier paramètre étant la classe et le second étant le paramètre fourni.

– Si un opérateur ou une methode d'une classe ne modifie pas la classe alors il est conseillé de dire au compilateur que cette fonction est « constante »en ajoutant le mots clef **const** après la définition des paramètres.
– Dans le cas d'un opérateur défini hors d'une classe le nombre de paramètres est donné par le type de l'opérateur uniare (`+ - * ! [] etc..` : 1 paramètre), binaire (`+ - * / | & || &&^ == <= >= < > etc..` 2 paramètres), n-aire (`()` : n paramètres).
– `ostream`, `istream` sont les deux types classique pour respectivement écrire et lire dans un fichier ou sur les entrées sorties standard. Ces types sont définis dans le fichier « iostream »inclue avec l'ordre `#include<iostream>` qui est mis en tête de fichier ;
– les deux opérateurs `<<` et `>>` sont les deux opérateurs qui généralement et respectivement écrivent ou lisent dans un type `ostream`, `istream`, ou `iostream`.
– Il y a bien autre façon d'écrire ces classes. Dans l'archive `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/R2.tgz` , vous trouverez trois autres méthodes d'écriture de cette petite classe.

v1) Avec compilation sépare où toutes les mèthodes et fonctions sont prototypées dans le fichier `R2-v1.hpp` et elles sont défini dans le fichier `R2-v1.cpp`

v2) Sans compilation sépare où toutes les méthodes et fonctions sont définies dans le fichier `R2-v2.hpp`.

v3) Version avec des champs privés qui interdisent utilisation direct de `x` et `y` hors de la classe, il faut passer par les méthodes `X()` et `Y()`

3.1.2 Utilisation de la classe R2

Cette classe modélise le plan $\mathbb{R}^2$, pour que les opérateurs classiques fonctionnent, c'est-à-dire : Un point P du plan définit via modélisé par ces 2 coordonnées x, y, et nous pouvons écrire des lignes suivantes par exemple :

```
R2 P(1.0,-0.5),Q(0,10);
R2 O,PQ(P,Q);                          //    le point O est initialiser à 0,0
R2 M=(P+Q)/2;                           //    espace vectoriel à droite
R2 A = 0.5*(P+Q);                       //    espace vectoriel à gauche
R  ps = (A,M);                          //    le produit scalaire de R2
R  pm = A^M;                                  //    le determinant de A,M
                                  //    l'aire du parallélogramme formé par A,M
R2 B = A.perp();                        //    B est la rotation de A par π/2
R  a= A.x + B.y;
A  = -B;
A += M;                                                 //    ie. A = A + M;
A -= B;                                                    //    ie. A = -A;
double abscisse= A.x;                                 //    la composante x de A
double ordonne = A.y;                                 //    la composante y de A
A.y= 0.5;                                          //    change la composante de A
A(1) =5;                                     //    change la 1 composante (x) de A
A(2) =2;                                     //    change la 2 composante (y) de A
A[0] =10;                                    //    change la 1 composante (x) de A
A[1] =100;                                   //    change la 2 composante (y) de A
cout <<  A;                                 //    imprime sur la console A.x et A.x
cint >> A;                          //    vous devez entrer 2 double à la console
```

Exercice 11.

Le but de cette exercice de d'affiche graphiquement les bassins d'attraction de la méthode de Newton, pour la résolution du problème $z^p - 1 = 0$, où $p = 3, 4, ...$
Rappel : la méthode de Newton s'écrit :

$$\text{Soit } z_0 \in \mathbb{C}, \quad z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)};$$

Q 1 Faire une classe `C` qui modélise le corps $\mathbb{C}$ des nombres complexes ($z = a + i * b$), avec toutes les opérations algébriques.

Q 2 Ecrire une fonction `C Newton(C z0,C (*f)(C),C (*df)(C) )` pour qui retourne la racines de l'équation $f(z) = 0$, limite des itérations de Newton : $z_{n+1} = z_n - \frac{f(z_n)}{df(z_n)}$ avec par exemple $f(z) = z^p - 1$ et $df(z) = f'(z) = p * z^{p-1}$, partant de $z0$. Cette fonction retournera le nombre complexe nul en cas de non convergence.

Q 3 Faire un programme, qui appelle la fonction Newton pour les points de la grille $z_0 + dn + dmi$ pour $(n, m) \in \{0, \ldots, N-1\} \times \{0, \ldots, M-1\}$ où les variables z_0, d, N, M sont données par l'utilisateur.

Afficher un tableau $N \times M$ résultat, qui nous dit vers quel racine la méthode de Newton a convergé en chaque point de la grille.

Q4 modifier les sources `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/mini-pj-fractale.tgz` afin d'affiché graphique, il suffit de modifier la fonction `void Draw()`, de plus les bornes de l'écran sont entières et sont stockées dans les variables global `int Width , Height;`.

Sous linux pour compiler et éditer de lien, il faut entrer les commandes `shell` suivantes :

```
g++ ex1.cpp -L/usr/X11R6/lib  -lGL -lGLUT -lX11 -o ex1
 # pour afficher le dessin dans une fenêtre X11.
```

ou mieux utiliser la commande unix `make` ou `gmake` en créant le fichier `Makefile` contenant :

```
CPP=g++
CPPFLAGS= -I/usr/X11R6/include
LIB= -L/usr/X11R6/lib -lglut -lGLU -lGL -lX11 -lm
%.o:%.cpp
(caractere de tabulation  -->/)$(CPP) -c $(CPPFLAGS) $^
cercle: cercle.o
(caractere de tabulation  -->/)g++ ex1.o  $(LIB) -o ex1
```

3.2 Les classes tableaux

Nous commencerons sur une version didactique, nous verrons la version complète qui est dans le fichier « tar compressé » `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/RNM-v3.tar.gz`.

3.2.1 Version simple d'une classe tableau

Mais avant toute chose, me paraît clair qu'un vecteur sera un classe qui contient au minimum la taille n, et un pointeur sur les valeurs. Que faut'il dans cette classe minimale (noté `A`).

```
typedef double K;                                          //    définition du corps
class A { public:                                            //    version 1 -------

  int n;                                                  //    la taille du vecteur
  K *v;                                                //    pointeur sur les n valeurs
  A() { cerr <<" Pas de constructeur pas défaut " << endl; exit(1);}
  A(int i) : n(i),v(new K[i]) { assert(v);}                  //    constructeur
  ~A() {delete [] v;}                                       //    destructeur
   K & operator[](int i) const  {assert(i>=0 && i <n);return v[i];}
};
```

Cette classe ne fonctionne pas car le constructeur par copie par défaut fait une copie bit à bit et donc le pointeur `v` est perdu, il faut donc écrire :

```
class A { public:                                            //    version 2 -------
  int n;                                                  //    la taille du vecteur
  K *v;                                                //    pointeur sur les n valeurs
  A() { cerr <<" Pas de constructeur pas défaut " << endl; exit(1);}
  A(const A& a) :n(a.n),v(new K[a.n])                 //    constructeur par copie
       { operator=(a);}
  A(int i) : n(i),v(new K[i]) { assert(v);}                  //    constructeur
  A& operator=(A &a) {                                      //    copie
        assert(n==a.n);
        for(int i=0;i<n;i++) v[i]=a.v[i];
        return *this;}
  ~A() {delete [] v;}                                       //    destructeur
  K & operator[](int i) const  {assert(i>=0 && i <n);return v[i];}
};
```

Maintenant nous voulons ajouter les opérations vectorielles $+, -, *, ...$

```
class A { public:                                            //    version 3 -------
  int n;                                                  //    la taille du vecteur
  K *v;                                                //    pointeur sur les n valeurs
  A() { cerr <<" Pas de constructeur pas défaut " << endl; exit(1);}
  A(const A& a) :n(a.n),v(new K[a.n]) { operator=(a);}
  A(int i) : n(i),v(new K[i]) { assert(v);}                  //    constructeur
  A& operator=(A &a) {assert(n==a.n);
                      for(int i=0;i<n;i++) v[i]=a.v[i];
                      return *this}
  ~A() {delete [] v;}                                       //    destructeur
  K & operator[](int i) const  {assert(i>=0 && i <n);return v[i];}
```

```
  A operator+(const &a) const;                                    //    addition
  A operator*(K &a) const;                          //    espace vectoriel à droite

 private:                    //    constructeur privé pour faire des optimisations
  A(int i, K* p) : n(i),v(p){assert(v);}
  friend A operator*(const K& a,const A& );       //    multiplication à gauche

 };
```

Il faut faire attention dans les paramètres d'entrée et de sortie des opérateurs. Il est clair que l'on ne veut pas travailler sur des copies, mais la sortie est forcément un objet et non une référence sur un objet car il faut allouer de la mémoire dans l'opérateur et si l'on retourne une référence aucun destructeur ne sera appelé et donc cette mémoire ne sera jamais libérée.

```
                  //   version avec avec une copie du tableau au niveau du return
A A::operator+(const A & a) const {
 A b(n); assert(n == a.n);
 for (int i=0;i<n;i++) b.v[i]= v[i]+a.v[i];
 return b;                          //    ici l'opérateur A(const A& a) est appeler
}
```

Pour des raisons optimisation nous ajoutons un nouveau constructeur `A(int,K*)` qui évitera de faire une copie du tableau.

```
                                    //    --- version optimisée sans copie----
A A::operator+(const A & a) const {
 K  *b(new K[n]); assert(n == a.n);
 for (int i=0;i<n;i++) b[i]= v[i]+a.[i];
 return A(n,b);                //    ici l'opérateur A(n,K*) ne fait pas de copie
}
```

Pour la multiplication par un scalaire à droite on a :

```
                                                //    --- version optimisée ----
A A::operator*(const K & a) const {
 K  *b(new K[n]); assert(n == a.n);
 for (int i=0;i<n;i++) b[i]= v[i]*a;
 return A(n,b);                //    ici l'opérateur A(n,K*) ne fait pas de copie
}
```

Pour la version à gauche, il faut définir `operator*` extérieurement à la classe car le terme de gauche n'est pas un vecteur.

```
A operator*(const K & a,const A & c) {
 K  *b(new K[c.n]);
 for (int i=0;i<c.n;i++) b[i]= c[i]*a;
 return A(n,b);                    //    attention cette opérateur est privé donc
                        //    cette fonction doit est ami ( friend) de la classe
}
```

Maintenant regardons ce qui est exécuté dans le cas d'une expression vectorielle.

```
int n=100000;
A a(n),b(n),c(n),d(n);
...                                          //    initialisation des tableaux a,b,c
d = (a+2.*b)+c*2.0;
```

voilà le pseudo code généré avec les 3 fonctions suivantes : `add(a,b,ab)`, `mulg(s,b,sb)`, `muld(a,s,as)`, `copy(a,b)` où le dernier argument retourne le résultat.

```
A a(n),b(n),c(n),d(n);
A t1(n),t2(n),t3(n),t4(n);
muld(2.,b),t1);                                                  //    t1 = 2.*b
add(a,t1,t2);                                                   //    t2 = a+2.*b
mulg(c,2.,t3);                                                   //    t3 = c*2.
add(t2,t3,t4);                                          //    t4 = (a+2.*b)+c*2.0;
copy(t4,d);                                              //    d = (a+2.*b)+c*2.0;
```

Nous voyons que quatre tableaux intermédiaires sont créés ce qui est excessif. D'où l'idée de ne pas utiliser toutes ces possibilités car le code généré sera trop lent.

Remarque 5. *Il est toujours possible de créer des classes intermédiaires pour des opérations prédéfinies afin obtenir se code générer :*

```
A a(n),b(n),c(n),d(n);
for (int i;i<n;i++)
  d[i] = (a[i]+2.*b[i])+c[i]*2.0;
```

Ce cas me paraît trop compliquer, mais nous pouvons optimiser raisonnablement toutes les combinaisons linéaires à 2 termes.

Mais, il est toujours possible d'éviter ces problèmes en utilisant les opérateurs `+=`, `-=`, `*=`, `/=` ce que donnerai de ce cas

```
A a(n),b(n),c(n),d(n);
for (int i;i<n;i++)
   d[i] = (a[i]+2.*b[i])+c[i]*2.0;
```

D'où l'idée de découper les classes vecteurs en deux types de classe les classes de terminer par un _ sont des classes sans allocation (cf. new), et les autres font appel à l'allocateur `new` et au déallocateur `delete`. De plus comme en fortran 90, il est souvent utile de voir une matrice comme un vecteur ou d'extraire une ligne ou une colonne ou même une sous-matrice. Pour pouvoir faire tous cela comme en fortran 90, nous allons considèrer un tableau comme un nombre d'élément n, un incrément s et un pointeur v et du tableau suivant de même type afin de extraire des sous-tableau.

3.2.2 les classes RNM

Nous définissons des classes tableaux à un, deux ou trois indices avec ou sans allocation. Ces tableaux est défini par un pointeur sur les valeurs et par la forme de l'indice (class `ShapeOfArray`) qui donne la taille, le pas entre de valeur et le tableau suivant de même type (ces deux données supplémentaire permettent extraire des colonnes ou des lignes de matrice, ...).

La version est dans le fichier « tar compressé » `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/RNM-v3.tar.gz`.
Nous voulons faire les opérations vectorielles classiques sur `A, C,D` tableaux de type `KN<R>` suivante par exemples :

```
A = B; A += B; A -= B; A = 1.0; A = 2*.C;
A = A+B; A = 2.0*C+ B; C = 3.*A-5*B;
 A /= C;  A *=C;                                          //    .. non standard ..
R c = A[i];
A[j] = c;
```

Pour des raisons évidentes nous ne sommes pas allés plus loin que des combinaisons linéaires à plus de deux termes. Toutes ces opérations sont faites sans aucune allocation et avec une seule boucle.
De plus nous avons défini, les tableaux 1,2 ou 3 indices, il est possible extraire une partie d'un tableau, une ligne ou une colonne.
Attention, il y a deux types de classe les classes avec un _ en fin de nom, et les autre sans. Les classes avec un _ en fin de nom, ne font aucune opération d'allocation ou destruction de mémoire dynamique, c'est-à-dire qui n'y a aucun opérateur `new` ou `delete` dans le code associé. Ces classes tableaux peuvent être vue comme une généralisation de la notion de références sur un tableau. Elle permet de donner un nom a une tranche d'un tableau, comme par exemple :

```
KN<double> a(10);                  //    un tableau de 10 valeurs qui est alloué
a=100.;                                 //    met tous les 10 valeurs de a à 100.
KN_<double> b(a);          //    le tableau de 10 valeurs avec le même pointeur
KN<double>  c(a);                  //    un autre tableau qui est la copie de a.
                                                                  //    la preuve
a[1]=1.;                                     //    => b[1] == 1. car &a[i] == &b[i]
                                  //    mais bien sur c[1] == 100. car &c[i] != &a[i]
```

Voilà le début du code de la classe `KN_` :

```
template<class R>
class KN_: public  ShapeOfArray {
protected:
  R *v;                                  //    le pointeur du tableau v[i × s], i ∈ [0..N()[
public:
  typedef R K;                                                    //    type of data
                                       //    les constructeurs a partir d'un pointeur
  KN_(R *u,long nn);                                              //    pointeur + n
  KN_(R *u,long nn,long s);          //    pointeur + n + valeurs avec un pas de s
  KN_(const KN_<R> & u);                             //    le constructeur par copy

                                                                  //    des méthodes
  long N() const {return n;}                              //    indice dans ı ∈ [0..N()[
  bool unset() const { return !v;}
  void set(R * vv,int nn,int st=1,int nx=-1) {v=vv;n=nn;step=st;next=nx;}
  long size() const{return step?n*step:n;}                        //    taille alloué

  //    remarque s est le pas (step) du tableau (généralement s = 1)
  R min() const;                                    //    min_{i∈[0..N()[} v[i s]
```

```
  R max() const;                                  //      max_{i∈[0..N()[} v[i s]
  R sum() const;                                  //      Σ_{i∈[0..N()[} v[i s]
  double norm() const;                            //      Σ_{i∈[0..N()[} |v[i s]|^2
  double l2() const;                              //      ( Σ_{i∈[0..N()[} v[i s]^2 )^{1/2}
  double l1() const;                              //      Σ_{i∈[0..N()[} |v[i s]|
  double linfty() const;                          //      max_{i∈[0..N()[} |v[i s]|
  double lp(double p) const;                      //      ( Σ_{i∈[0..N()[} v[i s]^p )^{1/2}

  //    pour conversion en pointeur
  operator R *() const {return v;}                //    transforme un KN_ en pointeur

 .
  KN_ operator()(const SubArray & sa) const     //    la fonction pour prendre un
       { return KN_(*this,sa);}     //   sous tableau (SubArray(N,debut=0,pas=1))

...  + tous les opérateurs  =, +=, -=, *=,  /=

 //    Operation avec des matrices VirtualMatrice
 KN_& operator =(const typename VirtualMatrice<R>::plusAx & Ax)
    {*this=R();             //    mise à 0 (le constructeur R() rend 0 de type R)
     Ax.A->addMatMul(Ax.x,*this);return *this;}

 KN_& operator +=(const typename VirtualMatrice<R>::plusAx & Ax)
    {  Ax.A->addMatMul(Ax.x,*this);return *this;}

.... etc
};
```

Maintenant, Voilà les constructeurs et le destructeur, les fonctions d'allocations de la classe KN qui dérive de KN_ :

```
template<class R>
class KN :public KN_<R> { public:
  typedef R K;
  KN() : KN_<R>(0,0) {}   //    constructeur par défaut tableau le longueur nulle
  KN(long nn);                                  //    construit un KN de longueur nn
  KN(long nn, R * p); //    construit un KN de longueur nn initiales le tableau p
  KN(long nn,R (*f)(long i) );                      //    construit KN (f(i))_{i=[0..nn[}
  KN(long nn,const  R & a);                        //    construit un KN (a)_{i=[0..nn[}
  template<class S>   KN(const KN_<S> & s);            //    construit par copie
  template<class S>
       KN(const KN_<S> & s,R (*f)(S ));             //    construit (f(s[i]))_{i=[0..s.N()[}
  KN(const KN<R> & u);                                //    construit par copie
   void resize(long nn);                       //    pour redimensionner le tableau
  ~KN(){delete [] this->v;}                                  //    le destructeur
  ...
  };
```

Et voilà de la classe `VirtualMatrice` qui modélise, une matrice. Une matrice sera une classe dérive de `VirtualMatrice`, qui aura définit la méthode `void addMatMul( const KN_<R> & x, K` qui ajoute à `y` le produit matrice vecteur.

```
template<class R>
struct  VirtualMatrice { public:
  virtual void addMatMul(const KN_<R> &  x, KN_<R> & y) const =0;
   ...
                              //    pour stocker les données de l'opération += A*x
   struct  plusAx { const VirtualMatrice * A; const KN_<R> &  x;
   plusAx( const VirtualMatrice * B,const KN_<R> &  y) :A(B),x(y) {} };

  virtual ~VirtualMatrice(){}
};
```

Voilà, un example très complet des possibilité de classes.

```
#include<RNM.hpp>

 ....

  typedef double R;
  KNM<R> A(10,20);                                                    //    un matrice
  . . .
  KN_<R> L1(A(1,'.');                                 //    la ligne 1 de la matrice A;
  KN<R> cL1(A(1,'.');                       //    copie de la ligne 1 de la matrice A;
  KN_<R> C2(A('.',2);                                 //    la colonne 2 de la matrice A;
  KN<R> cC2(A('.',2);                     //    copie de la colonne 2 de la matrice A;
  KNM_<R> pA(FromTo(2,5),FromTo(3,7));      //    partie de la matrice A(2:5,3:7)
                                                      //    vue comme un matrice 4x5

  KNM B(n,n);
  B(SubArray(n,0,n+1))                                //    le vecteur diagonal de B;
  KNM_ Bt(B.t());                                   //    la matrice transpose sans copie
```

Pour l'utilisation, utiliser l'ordre `#include "RNM.hpp"`, et les flags de compilation `-DCHECK_KN` ou en définisant la variable du preprocesseur `cpp` du `C++` avec l'ordre `#defined CHECK_KN`, avant la ligne include.
Les définitions des classes sont faites dans 4 fichiers `RNM.hpp`, `RNM_tpl.hpp`, `RNM_op.hpp`, `RNM_op.hpp`.
Pour plus de détails voici un exemple d'utilisation assez complet.

3.2.3 Exemple d'utilisation

```
namespace std

#define CHECK_KN
#include "RNM.hpp"
#include "assert.h"
```

```
using namespace std;
          //    definition des 6 types de base des tableaux a 1,2 et 3 parametres
typedef double R;
typedef KN<R> Rn;
typedef KN_<R> Rn_;
typedef KNM<R> Rnm;
typedef KNM_<R> Rnm_;
typedef KNMK<R> Rnmk;
typedef KNMK_<R> Rnmk_;
R f(int i){return i;}
R g(int i){return -i;}
int main()
{
  const int n= 8;
  cout << "Hello World, this is RNM use!" << endl << endl;
  Rn a(n,f),b(n),c(n);
  b =a;
  c=5;
  b *= c;

  cout << " a = " << (KN_<const_R>) a << endl;
  cout << " b = " <<  b << endl;

                                                  //    les operations vectorielles
  c = a + b;
  c = 5. *b + a;
  c = a + 5. *b;
  c = a - 5. *b;
  c = 10.*a - 5. *b;
  c = 10.*a + 5. *b;
  c += a + b;
  c += 5. *b + a;
  c += a + 5. *b;
  c += a - 5. *b;
  c += 10.*a - 5. *b;
  c += 10.*a + 5. *b;
  c -= a + b;
  c -= 5. *b + a;
  c -= a + 5. *b;
  c -= a - 5. *b;
  c -= 10.*a - 5. *b;
  c -= 10.*a + 5. *b;


  cout <<" c = " << c << endl;
  Rn u(20,f),v(20,g);                                  //    2 tableaux u,v de 20
                                                //   initialiser avec u_i = f(i), v_j = g(i)
  Rnm A(n+2,n);                                        //    Une matrice n+2 x n

  for (int i=0;i<A.N();i++)                                                 //    ligne
     for (int j=0;j<A.M();j++)                                             //    colonne
       A(i,j) = 10*i+j;

  cout << "A=" << A << endl;
  cout << "Ai3=A('.', 3 ) = " << A('.', 3 ) << endl;          //    la colonne 3
  cout << "A1j=A( 1 ,'.') = " << A( 1 ,'.') << endl;            //    la ligne 1
  Rn CopyAi3(A('.', 3 ));                            //    une copie de la colonne 3
```

```
cout << "CopyAi3 = " << CopyAi3;

Rn_ Ai3(A('.', 3 ));                          //    la colonne 3 de la matrice
CopyAi3[3]=100;
cout << CopyAi3 << endl;
cout << Ai3 << endl;

assert( & A(0,3) == & Ai3(0));                //    verification des adresses


Rnm S(A(SubArray(3),SubArray(3)));                      //    sous matrice 3x3

Rn_ Sii(S,SubArray(3,0,3+1));         //    la diagonal de la matrice sans copy

cout << "S=  A(SubArray(3),SubArray(3)  = " << S <<endl;
cout << "Sii     = " << Sii <<endl;
b = 1;
                                                //    Rn Ab(n+2) = A*b; error
Rn Ab(n+2);
Ab = A*b;
cout << " Ab = A*b  =" << Ab << endl;

Rn_ u10(u,SubArray(10,5));                //    la partie [5,5+10[ du tableau u
cout << "u10 " << u10 << endl;
v(SubArray(10,5)) += u10;
cout << " v = " << v << endl;
cout << " u(SubArray(10))    "  << u(SubArray(10)) << endl;
cout << " u(SubArray(10,5))  "  << u(SubArray(10,5)) << endl;
cout << " u(SubArray(8,5,2))"  << u(SubArray(8,5,2))
     << endl;


cout << " A(5,'.')[1] " << A(5,'.')[1] << " " << " A(5,1) = "
     << A(5,1) << endl;
cout << " A('.',5)(1) = "<< A('.',5)(1) <<  endl;
cout << " A(SubArray(3,2),SubArray(2,4)) = " << endl;
cout <<  A(SubArray(3,2),SubArray(2,4)) << endl;
A(SubArray(3,2),SubArray(2,4)) = -1;
A(SubArray(3,2),SubArray(2,0)) = -2;
cout << A << endl;

Rnmk B(3,4,5);
for (int i=0;i<B.N();i++)                                    //    ligne
   for (int j=0;j<B.M();j++)                                //    colonne
       for (int k=0;k<B.K();k++)                              //    ....
          B(i,j,k) = 100*i+10*j+k;
cout << " B      = " << B << endl;
cout << " B(1  ,2  ,'.') " << B(1  ,2  ,'.') << endl;
cout << " B(1  ,'.',3  ) " << B(1  ,'.',3  ) << endl;
cout << " B('.',2  ,3  ) " << B('.',2  ,3  ) << endl;
cout << " B(1  ,'.','.') " << B(1,'.'  ,'.') << endl;
cout << " B('.',2  ,'.') " << B('.',2  ,'.') << endl;
cout << " B('.','.',3  ) " << B('.','.',3  ) << endl;

cout << " B(1:2,1:3,0:3)   = "
     << B(SubArray(2,1),SubArray(3,1),SubArray(4,0)) << endl;
```

```
                                                //    copie du sous tableaux

  Rnmk Bsub(B(FromTo(1,2),FromTo(1,3),FromTo(0,3)));
  B(SubArray(2,1),SubArray(3,1),SubArray(4,0))  = -1;
  B(SubArray(2,1),SubArray(3,1),SubArray(4,0)) += -1;
  cout << " B      = " << B << endl;
  cout << Bsub << endl;

  return 0;
}
```

3.2.4 Un resolution de systéme linéaire avec le gradient conjugué

L'algorithme du gradient conjugué présenté dans cette section est utilisé pour résoudre le système
linéaire $Ax = b$, où A est une matrice symétrique positive $n \times n$.
Cet algorithme est basé sur la minimisation de la fonctionnelle quadratique $E : \mathbb{R}^n \to \mathbb{R}$
suivante :

$$E(x) = \frac{1}{2}(Ax, x) - (b, x),$$

avec un changement de variable $y = Cx$ où $(.,.)_C$ est le produit scalaire associer à une une matrice C, symétrique définie positive
de $\mathbb{R}^n$.
C'est aussi la minimisation de la fonctionnelle :

$$E_c(y) = \frac{1}{2}(ACy, y)_C - (b, y)_C,$$

car la matrice CAC est symétrique positive.

Algorithme 1. **Le gradient conjugué preconditionné**

soient $x^0 \in \mathbb{R}^n$, ε, C donnés
$G^0 = Ax^0 - b$
$H^0 = -CG^0$
– *pour* $i = 0$ *à* n
$\rho = -\dfrac{(G^i, H^i)}{(H^i, AH^i)}$
$x^{i+1} = x^i + \rho H^i$
$G^{i+1} = G^i + \rho AH^i$
$\gamma = \dfrac{(G^{i+1}, G^{i+1})_C}{(G^i, G^i)_C}$
$H^{i+1} = -CG^{i+1} + \gamma H^i$
si $(G^{i+1}, G^{i+1})_C < \varepsilon$ *stop*

Théorème 3.1. *Notons* $\mathcal{E}_C(x) = \sqrt{(Ax-\overline{x}, x-\overline{x})_C}$, *l'erreur dans la norme de* A *preconditionné, où* $\overline{x}$ *est la solution du problème. Alors l'erreur à l'intération* k *un gradient conjugué est majoré :*

$$\mathcal{E}_C(x^k) \leq 2\left(\frac{\sqrt{K_C(A)-1}}{\sqrt{K_C(A)+1}}\right)^k \mathcal{E}_C(x^0)$$

où $K_C(A)$ *est le conditionnement de la matrice* CA, *c'est à dire* $K_C(A) = \frac{\lambda_1^C}{\lambda_n^C}$ *où* λ_1^C *(resp.* λ_n^C*) est la plus petite (resp. grande) valeur propre de matrice* CA .

Le démonstration est technique et est faite dans [Lascaux et Théodor, chap 8.3]. Voila comment écrire un gradient conjugué avec ces classes.

3.2.5 Gradient conjugué préconditionné

Listing 5: *(GC.hpp)*

```
                //    exemple de programmation du gradient conjugue precondiconnée
template<class R,class M,class P>
int GradienConjugue(const M & A,const P & C,const KN_<R> &b,KN_<R> &x,
                    int nbitermax,double eps)
{
   int n=b.N();
   assert(n==x.N());
   KN<R> g(n), h(n), Ah(n), & Cg(Ah);          //    on utilise Ah pour stocke Cg
   g = A*x;
   g -= b;                                                    //    g = Ax-b
   Cg = C*g;                                          //    gradient preconditionne
   h =-Cg;
   R g2 = (Cg,g);
   R reps2 = eps*eps*g2;                                    //    epsilon relatif
   for (int iter=0;iter<=nbitermax;iter++)
     {
       Ah = A*h;
       R ro =  - (g,h)/ (h,Ah);          //    ro optimal (produit scalaire usuel)
       x += ro *h;
       g += ro *Ah;      //    plus besoin de Ah, on utilise avec Cg optimisation
       Cg = C*g;
       R g2p=g2;
       g2 = (Cg,g);
       cout << iter <<  "  ro = " << ro << " ||g||^2 = " << g2 << endl;
       if (g2 < reps2) {
          cout << iter <<  "  ro = " << ro << " ||g||^2 = " << g2 << endl;
          return 1;                                                    //    ok
          }
       R gamma = g2/g2p;
       h *= gamma;
       h -= Cg;                                          //    h = -Cg * gamma* h
     }
   cout << " Non convergence de la méthode du gradient conjugue " <<endl;
   return 0;
}
//    la matrice Identite --------------------
template <class R>
```

```
class MatriceIdentite: VirtualMatrice<R> { public:
 typedef  VirtualMatrice<R>::plusAx plusAx;
 MatriceIdentite() {};
 void addMatMul(const  KN_<R>  & x, KN_<R> & Ax) const {   Ax+=x; }
 plusAx operator*(const KN<R> &  x) const {return plusAx(this,x);}
};
```

3.2.6 Test du gradient conjugué

Pour finir voilà, un petit programme pour tester le GC sur cas différent. Le troisième cas étant la résolution de l'équation différencielle : $-u'' = 1$ sur $[0, 1]$ avec comme conditions aux limites $u(0) = u(1) = 0$, par la méthode de l'élément fini. La solution exact est $f(x) = x(1 - x)/2$, nous verifions donc l'erreur sur le resultat.

Listing 6: *(GradConjugue.cpp)*

```
#include <fstream>
#include <cassert>
#include <algorithm>

using namespace std;

#define KN_CHECK
#include "RNM.hpp"
#include "GC.hpp"

typedef double R;
class MatriceLaplacien1D: VirtualMatrice<R> { public:
 MatriceLaplacien1D() {};
 void addMatMul(const KN_<R> & x, KN_<R> & Ax) const;
 plusAx operator*(const KN<R> &  x) const {return plusAx(*this,x);}
};

 void MatriceLaplacien1D::addMatMul(const KN_<R> & x, KN_<R> & Ax) const {
  int n= x.N(),n_1=n-1;
  double h=1./(n_1), h2= h*h, d = 2/h, d1 = -1/h;
  R Ax0=Ax[0], Axn_1=Ax[n_1];
  Ax=0;
  for (int i=1;i< n_1; i++)
    Ax[i] = (x[i-1] +x[i+1]) * d1 + x[i]*d ;

                                                                    //     CL
   Ax[0]=x[0];
   Ax[n_1]=x[n_1];
  }


int main(int argc,char ** argv)
{
 typedef KN<double> Rn;
 typedef KN_<double> Rn_;
 typedef KNM<double> Rnm;
 typedef KNM_<double> Rnm_;
```

```
{
 int n=10;
 Rnm A(n,n),C(n,n),Id(n,n);
 A=-1;
 C=0;
 Id=0;
 Rn_ Aii(A,SubArray(n,0,n+1));        //   la diagonal de la matrice A sans copy
 Rn_ Cii(C,SubArray(n,0,n+1));        //   la diagonal de la matrice C sans copy
 Rn_ Idii(Id,SubArray(n,0,n+1));     //   la diagonal de la matrice Id sans copy
 for (int i=0;i<n;i++)
    Cii[i]= 1/(Aii[i]=n+i*i*i);
 Idii=1;
 cout << A;
 Rn x(n),b(n),s(n);
 for (int i=0;i<n;i++) b[i]=i;
 cout << "GradienConjugue preconditionne par la diagonale " <<  endl;
 x=0;
 GradienConjugue(A,C,b,x,n,1e-10);
 s = A*x;
 cout << " solution : A*x= "  << s << endl;
 cout << "GradienConjugue preconditionnee par la identity  " << endl;
 x=0;
 GradienConjugue(A,MatriceIdentite<R>(),b,x,n,1e-6);
 s = A*x;
 cout << s << endl;
}
{
 cout << "GradienConjugue laplacien 1D  par la identity  " << endl;
 int N=100;
 Rn b(N),x(N);
 R h= 1./(N-1);
 b= h;
 b[0]=0;
 b[N-1]=0;
 x=0;
 R t0=CPUtime();
 GradienConjugue(MatriceLaplacien1D(),MatriceIdentite<R>() ,b,x,N,1e-5);
 cout << " Temps cpu = " << CPUtime() - t0<< "s" << endl;
 R err=0;
 for (int i=0;i<N;i++)
   {
   R xx=i*h;
   err= max(fabs(x[i]- (xx*(1-xx)/2)),err);
   }
 cout << "Fin err=" << err << endl;
}
 return 0;
}
```

3.2.7 Sortie du test

```
10x10   :
          10  -1  -1  -1  -1  -1  -1  -1  -1  -1
          -1  11  -1  -1  -1  -1  -1  -1  -1  -1
          -1  -1  18  -1  -1  -1  -1  -1  -1  -1
          -1  -1  -1  37  -1  -1  -1  -1  -1  -1
          -1  -1  -1  -1  74  -1  -1  -1  -1  -1
          -1  -1  -1  -1  -1 135  -1  -1  -1  -1
          -1  -1  -1  -1  -1  -1 226  -1  -1  -1
          -1  -1  -1  -1  -1  -1  -1 353  -1  -1
          -1  -1  -1  -1  -1  -1  -1  -1 522  -1
          -1  -1  -1  -1  -1  -1  -1  -1  -1 739
        GradienConjugue preconditionne par la diagonale
6  ro = 0.990712 ||g||^2 = 1.4253e-24
 solution : A*x= 10     :    1.60635e-15  1 2 3 4 5 6 7 8 9

GradienConjugue preconditionnee par la identity
9  ro = 0.0889083 ||g||^2 = 2.28121e-15
10      :       6.50655e-11    1 2 3 4 5 6 7 8 9

GradienConjugue laplacien 1D  preconditionnee par la identity
48  ro = 0.00505051 ||g||^2 = 1.55006e-32
 Temps cpu = 0.02s
Fin err=5.55112e-17
```

Exercice 12.

Modifier, l'exemple `GradConjugue.cpp`, pour résoudre le problème suivant, trouver $u(x,t)$ une solution

$$\frac{\partial u}{\partial t} - \Delta u = f; \quad \text{dans }]0,L[$$

$$\text{pour } t=0, u(x,0) = u_0(x) \quad \text{et } u(0,t) = u(L,t) = 0$$

en utilisant un θ schéma pour discrétiser en temps, c'est à dire que

$$\frac{u^{n+1}-u^n}{\Delta t} - \Delta\big(\theta u^{n+1} + (1-\theta)u^n\big) = f; \quad \text{dans }]0,L[$$

où u^n est une approximation de $u(x, n\Delta t)$, faite avec des éléménts finis P_1, avec un maillage régulier de $]0,L|$ en M éléments. les fonctions élémentaires seront noté w^i, avec pour $i=0,...,M$, avec $x_i = \frac{i/N}{L}$

$$w^i|_{]x_{i-1},x_i[\cup]0,L[} = \frac{x-x_i}{x_{i-1}-x_i}, \quad w^i|_{]x_i,x_{i+1}[\cup]0,L[} = \frac{x-x_i}{x_{i+1}-x_i}, \quad w^i|_{]0,L[\setminus]x_{i-1},x_{i+1}[} = 0$$

C'est a dire qu'il faut commencer par construire du classe qui modélise la matrice

$$\mathcal{M}_{\alpha\beta} = \left(\int_{]O,L[} \alpha w^i w^j + \beta (w^i)'(w^j)' \right)_{i=1...M, j=1...M}$$

en copiant la classe `MatriceLaplacien1D`.
Puis, il suffit, d'approcher le vecteur $F = (f_i)_{i=0..M} = \left(\int_{]O,L[} f w^i \right)_{i=0..M}$ par le produit $F_h = \mathcal{M}_{1,0} \left(f(x_i) \right)_{i=1,M}$, ce qui revient à remplacer f par

$$f_h = \sum_i f(x_i) w^i$$

.

Q1 Ecrire l'algorithme mathematiquement, avec des matrices

Q2 Trancrire l'algorithme

Q3 Visualiser les résultat avec gnuplot, pour cela il suffit de crée un fichier par pas de temps contenant la solution, stocker dans un fichier, en inspirant de

```
#include<sstream>
#include<ofstream>
#include<iostream>
  ....
  stringstream ff;
  ff << "sol-"<< temps << ends;
  cout << " ouverture du fichier" <<ff.str.c_str()<< endl;
  {
     ofstream file(ff.str.c_str());
     for (int i=0;i<=M;++i)
       file <<x[i]<< endl;
   }          //    fin bloque => destruction de la variable file
                                  //    => fermeture du fichier
  ...
```

4 Méthodes d'éléments finis P_1 Lagrange

4.1 Formules de Green

Grâce à la densité de $\mathcal{D}(\overline{\Omega})$ dans $H^1(\Omega)$, on peut démontrer la formule de Green pour les fonctions de $H^1(\Omega)$:

$$\forall u \in H^1(\Omega)\ ,\ \forall v \in H^1(\Omega)\ ,\quad \int_\Omega u\,\frac{\partial v}{\partial x_i}\,d\boldsymbol{x} = -\int_\Omega v\,\frac{\partial u}{\partial x_i}\,d\boldsymbol{x} + \int_{\partial\Omega} u\,v\,n_i\,d\sigma\,, \tag{66}$$

où n_i est la $i$ème composante de $\boldsymbol{n}$ la normale extérieur unitaire, et σ est la mesure du bord.
En utilisant les notations du points 1.1 page 6, et toujours grâce à la densité, et en sommant sur les composantes, pour les fonctions vectorielles de $H(\mathrm{div},\Omega) = \{\boldsymbol{v} \in L^2(\Omega)^d / \nabla.\boldsymbol{v} \in L^2(\Omega)\}$, on peut écrire la formule de Stokes avec le gradient ∇ et la divergence $\nabla.$:

$$\forall u \in H^1(\Omega)\ ,\ \forall \boldsymbol{v} \in H(\mathrm{div},\Omega)\ ,\quad \int_\Omega u\,\nabla.\boldsymbol{v}\,d\boldsymbol{x} = -\int_\Omega (\nabla u).\boldsymbol{v}\,d\boldsymbol{x} + \int_{\partial\Omega} u(\boldsymbol{v}.\boldsymbol{n})\,d\sigma\,, \tag{67}$$

La formule précédente avec v et ∇u, nous donne donc :

$$\forall u \in H^2(\Omega)\ ,\ \forall v \in H^1(\Omega)\ ,\quad \int_\Omega (\Delta u)\,v\,d\boldsymbol{x} = -\int_\Omega \nabla u.\nabla v\,d\boldsymbol{x} + \int_{\partial\Omega} v\frac{\partial u}{\partial \boldsymbol{n}}\,d\sigma\,. \tag{68}$$

4.2 Rappel : Forme linéaire, bilinéaire, vecteur , matrice

Soit V un espace vectoriel de dimension finie réel munie d'une base $\{w^i, i = 1, ..., n\}$.
Soit a une forme bilinéaire de V (application de $V \times V \mapsto \mathbb{R}$) et $\ell \in V'$ une forme linéaire de (application de $V \mapsto \mathbb{R}$).
Alors la résolution du problème : trouver $u \in V$ tel

$$\forall v \in V, \qquad a(u,v) = l(v), \tag{69}$$

est équivalent à la résolution du système linéaire matricielle $n \times n$ suivant : trouver $U = (u_i)_{i=1,n} \in \mathbb{R}^n$ telle que

$$\boldsymbol{A}U = B, \tag{70}$$

Où la matrice A est définie par

$$A = (a_{ij}) \quad \text{avec} \quad \forall (i,j) \in \{1, ..., n\}^2 \quad a_{ij} = a(w^j, w^j) \tag{71}$$

et où le second membre B est donné par

$$B = (b_i) \in \mathbb{R}^d, \qquad \text{avec} \quad \forall i \in \{1, ..., n\} \quad b_i = \ell(w^i) \tag{72}$$

Pour finir la solution u est égale à

$$u = \sum_{i=1}^{n} u_i w^i. \tag{73}$$

4.3 Espace affine, convexifié, et simplexe

Définition 4.1. *Dans un espace affine réel $\mathcal{A}$, le barycentre de $(\lambda_i, P_i)_{i=1,n} \in (\mathbb{R} \times A)^n$ telle que $\sum_{i=1}^n \lambda_i \neq 0$ est l'unique point $G = \text{bar}((\lambda_i, P_i)_{i=1,n}) \in \mathcal{A}$ telle que*

$$\forall O \in \mathcal{A}, \qquad \sum_{i=1}^{n} \lambda_i \overrightarrow{OP}_i = (\sum_{i=0}^{n} \lambda_i) \overrightarrow{OG}.$$

Et si cette espace affine A est plongé dans un espace vectoriel, alors

$$G = \frac{\sum_{i=0}^n \lambda_i P_i}{\sum_{i=0}^n \lambda_i}$$

Définition 4.2. *Par définition, si f est une fonction affine de $\mathcal{A} \mapsto \mathcal{B}$, alors cette fonction commute avec les barycentres, c'est-à-dire $f(\text{bar}((\lambda_i, P_i)_{i=1,n})) = \text{bar}((\lambda_i, f(P_i))_{i=1,n})$, ou si l'espace affine est plongé dans un espace vectoriel, on a*

$$si \qquad \sum_{i=1}^{n} \lambda_i = 1, \qquad alors \quad f(\sum_{i=1}^{n} \lambda_i x_i) = \sum_{i=1}^{n} \lambda_i f(x_i) \tag{74}$$

Définition 4.3. *Le convexifié d'un ensemble S de points de $\mathbb{R}^d$ est noté $\mathcal{C}(S)$ est le plus petit convexe contenant S et si l'ensemble est fini (*i.e. $S = \{x^i, i = 1, .., n\}$*) alors nous avons :*

$$\mathcal{C}(S) = \left\{ \sum_{i=1}^{n} \lambda_i x^i : \quad \forall (\lambda_i)_{i=1,..,n} \in \mathbb{R}_+^n, \text{ tel que } \sum_{i=1}^{n} \lambda_i = 1 \right\}$$

.

Définition 4.4. *Un k-simplexe $(P^0, ..., P^k)$ est le convexifié des $k+1$ points de $\mathbb{R}^d$ affine independant (donc $k \leq d$)*

– un sommet est un 0-simplexe,
– une arête ou un segment est un 1-simplexe,
– un triangle est un 2-simplexe,
– un tétraèdre est un 3-simplexe ;
La mesure signée d'un d-simplexe $K = (P^0, ..., P^d)$ en dimension d est donnée par

$$mes(P^0, ..., P^d) \quad = \quad \frac{1}{d!} det \left| \overrightarrow{P^0 P^1} \; ... \; \overrightarrow{P^0 P^d} \right| \quad = \quad \frac{-1^d}{d!} det \begin{vmatrix} P_1^0 & \dots & P_1^d \\ \vdots & \dots & \vdots \\ P_d^0 & \dots & P_d^d \\ 1 & \dots & 1 \end{vmatrix}$$

et le d-simplexe sera dit positif si sa mesure est positive.
les sous p-simplexe d'un d-simplexe sont formés avec $p+1$ points distinctes parmi $(P^0, ..., P^d)$.
Les ensembles de k-simplexe
– des sommets seront l'ensemble des $d+1$ points (0-simplexe),
– des arêtes seront l'ensemble des $\frac{(d+1)\times d}{2}$ 1-simplexe ,
– des triangles seront l'ensemble des $\frac{(d+1)\times d\times (d-1)}{6}$ 2-simplexe ,

– des hyperfaces seront l'ensemble des $(d+1)$ hyperfaces ($(d-1)$-simplexe) ,

Définition 4.5. *Les coordonnées barycentriques d'un d-simplex $K = (P^0, ..., P^d)$ sont des $d+1$ fonctions affines de $\mathbb{R}^d$ dans $\mathbb{R}$ noté $\lambda_i^K, j = 0, ..., d$ et sont défini par*

$$\forall x \in \mathbb{R}^d; \qquad x = \sum_{i=0}^{d} \lambda_i^K(x) P^i; \qquad et \qquad \sum_{i=0}^{d} \lambda_i^K(x) = 1 \tag{75}$$

on a $\lambda_i^K(P^j) = \delta_{ij}$ où δ_{ij} est le symbole de Kroneker. Et les formules de Cramers nous donnent :

$$\lambda_i^K(x) = \frac{mes(P^0, ..., P^{i-1}, x, P^{i+1}, ..., P^d)}{mes(P^0, ..., P^{i-1}, P^i, P^{i+1}, ..., P^d)}$$

Le gradient de coordonnée barycentrique est donnée par la formule suivante

$$\nabla \lambda_i^K(x) = \frac{-1^{d+i}}{|K|d!} \overrightarrow{P^{n_{i,1}} P^{n_{i,2}}} \wedge ... \wedge \overrightarrow{P^{n_{i,1}} P^{n_{i,d}}}$$

où $|K|$ est la mesure signée de K, les $(n_{i,j})_{j=1}^d$ pour i fixé est la suite croissante des nombres sans l'élément i, c'est-à-dire $(n_{i,1}, ..., n_{i,d}) = (0, ..., i-1, i+1, ..., d)$, où le produit vectoriel est l'application $(d-1)$-linéaire alternée $(R^d)^{d-1} \mapsto R^d$ qui est la généralisation du produit vectoriel dans $\mathbb{R}^d$, et qui est définie par la formule suivante :

$$\forall y \in \mathbb{R}^d; \quad x^1 \wedge ... \wedge x^{d-1}.y = det|x^1, ..., x^{d-1}, y|$$

Démonstration.

$$\begin{aligned}
D\lambda_i^K(x)y &= \frac{-1^d}{|K|d!} det \begin{vmatrix} P_1^0 & \dots & P_1^{i-1} & y_1 & P_1^{i+1} & \dots & P_1^d \\ \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ P_d^0 & \dots & P_d^{i-1} & y_d & P_d^{i+1} & \dots & P_d^d \\ 1 & \dots & 1 & 0 & 1 & \dots & 1 \end{vmatrix} \\
&= \frac{-1^{d-d+i}}{|K|d!} det \begin{vmatrix} P_1^0 & \dots & P_1^{i-1} & P_1^{i+1} & \dots & P_1^d & y_1 \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots \\ P_d^0 & \dots & P_d^{i-1} & P_d^{i+1} & \dots & P_d^d & y_d \\ 1 & \dots & 1 & 1 & \dots & 1 & 0 \end{vmatrix} \\
&= \frac{-1^{d-d+i}}{|K|d!} det \begin{vmatrix} P_1^{n_{i,1}} & \dots & P_1^{n_{i,d}} & y_1 \\ \vdots & \dots & \vdots & \vdots \\ P_d^{n_{i,1}} & \dots & P_d^{n_{i,d}} & y_d \\ 1 & \dots & 1 & 0 \end{vmatrix} \\
&= \frac{-1^{i}}{|K|d!} det \begin{vmatrix} P_1^{n_{i,1}} & \overrightarrow{P_1^{n_{i,1}} P_1^{n_{i,2}}} & \dots & \overrightarrow{P_1^{n_{i,1}} P_1^{n_{i,d}}} & y_1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ P_d^{n_{i,1}} & \overrightarrow{P_d^{n_{i,1}} P_d^{n_{i,2}}} & \dots & \overrightarrow{P_d^{n_{i,1}} P_d^{n_{i,d}}} & y_d \\ 1 & 0 & \dots & 0 & 0 \end{vmatrix} \\
&= \frac{-1^{i+d}}{|K|d!} \overrightarrow{P^{n_{i,1}} P^{n_{i,2}}} \wedge ... \wedge \overrightarrow{P^{n_{i,1}} P^{n_{i,d}}}.y
\end{aligned}$$

□

Donc si $d = 2$ nous avons donc

$$\nabla\lambda_i^K(x) = \frac{-1^i}{2|K|}\overrightarrow{P^{n_{i,1}}P^{n_{i,2}}}^{\perp}$$

où $\overrightarrow{PQ}^{\perp}$ est la rotation de $\pi/2$ du vecteur $\overrightarrow{PQ}$. Ce qui donne

$$\nabla\lambda_0^K(x)\frac{+1}{2|K|} = \overrightarrow{P^1P^2}^{\perp}, \qquad \nabla\lambda_1^K(x)\frac{-1}{2|K|} = \overrightarrow{P^0P^2}^{\perp}, \qquad \nabla\lambda_2^K(x)\frac{+1}{2|K|} = \overrightarrow{P^0P^1}^{\perp}$$

ce que l'on peut réécrire comme avec la convention $P^3 = P^0$ et $P^4 = P^1$:

$$\nabla\lambda_i^K(x) = \frac{+1}{2|K|}(\overrightarrow{P^{i+1}P^{i+2}})^{\perp}, \tag{76}$$

Et si $d = 3$ alors on a

$$\nabla\lambda_i^K(x) = \frac{-1^{i+1}}{6|K|}\ \overrightarrow{P^{n_{i,1}}P^{n_{i,2}}} \wedge \overrightarrow{P^{n_{i,1}}P^{n_{i,3}}}$$

avec la numérotation des sommets des faces suivante : $(n_{0,j})_{j=1,2,3} = (1,2,3)$, $(n_{1,j})_{j=1,2,3} = (0,2,3)$, $(n_{2,j})_{j=1,2,3} = (0,1,3)$, et $(n_{3,j})_{j=1,2,3} = (0,1,2)$.
Pour supprimer le -1^{i+1} il suffit d'utiliser la numérotation des sommets des faces suivante :

$$(\tilde{n}_{0,j})_{j=1,2,3} = (2,1,3), \qquad (\tilde{n}_{1,j})_{j=2,1,3} = (0,2,3), \tag{77}$$
$$(\tilde{n}_{2,j})_{j=1,2,3} = (1,0,3), \qquad (\tilde{n}_{3,j})_{j=1,2,3} = (0,1,2). \tag{78}$$

Ce qui donne :

$$\nabla\lambda_i^K(x) = \frac{1}{6|K|}\ \overrightarrow{P^{\tilde{n}_{i,1}}P^{\tilde{n}_{i,2}}} \wedge \overrightarrow{P^{\tilde{n}_{i,1}}P^{\tilde{n}_{i,3}}} \tag{79}$$

Remarque 6. *Sur le d-simplex référence note $\hat{K} = (0, \boldsymbol{e}_1, \cdots, \boldsymbol{e}_d)$ où les $\boldsymbol{e}_1, \cdots, \boldsymbol{e}_d)$ forment la base canonique de $\mathbb{R}^d$ alors les coordonnes barycentrique de $\lambda_i^{\hat{K}}$ sont*

$$\lambda_0^{\hat{K}}(\hat{\boldsymbol{x}}) = 1 - \sum_{i=1}^{d}\hat{x}_i, \qquad et \quad \lambda_j^{\hat{K}}(\hat{\boldsymbol{x}}) = \hat{x}_j \quad pour\ j = 1, \cdots, d \tag{80}$$

où $\boldsymbol{x} = (x_i)_{i=0,\cdot,d}$.

Les coordonnées barycentriques permettent de construire une fonction affine à partir des valeurs aux sommets de simplexe K via le lemme suivant :

Lemme 4.6. ?? *Soit f une fonction affine d'un espace affine $\mathcal{A}$ de dimension d à valeur dans espace vectoriel réel et un d-simplexe $K = (P^0, ..., P^d)$ de $\mathcal{A}$ alors on à l'égalité suivante :*

$$f(x) = \sum_{i=0}^{d} f(P^i)\,\lambda_i^K(x). \tag{81}$$

Démonstration. Il suffit juste de remarquer que comme f est affine, elle commute avec les barycentres et que x est le barycentres de $(\lambda_i^K(x), P^i), i = 0, \cdots, d$ car les $(P^0, ..., P^d$ forme une base affine de $\mathcal{A}$ □

4.4 Formule d'intégration

Let D be a N-dimensional bounded domain. For an arbitrary polynomials f of degree r, if we can find particular points $\vec{\xi_j}$, $j = 1, \cdots, J$ in D and constants ω_j such that

$$\int_D f(\vec{x}) = |D| \sum_{\ell=1}^{L} \omega_\ell f(\vec{\xi_\ell}) \tag{82}$$

then we have the error estimation (see Crouzeix-Mignot (1984)), and then there exists a constant $C > 0$ such that,

$$\left| \int_D f(\vec{x}) - |D| \sum_{\ell=1}^{L} \omega_\ell f(\vec{\xi_\ell}) \right| \leq C|D|h^{r+1} \tag{83}$$

4.4.1 Formule d'intégration sur le triangle

Soit le triangle $K = [q^{i_0} q^{i_1} q^{i_2}]$ ($d = 2$, le point intégration est défini via le point $\hat{\xi}^\ell$ sur le triangle référence $\hat{K} = \{(0,0),(1,0),(0,1)\}$, comme suit

$$\xi_l = (1 - \sum_{k=0}^{d} \hat{\xi}_k) q^{i_0} + \sum_{k=1}^{d} \hat{\xi}_k q^{i_k}$$

L	points $\hat{\xi}^\ell$ in $\hat{K}$	ω_ℓ	degree of exact
1	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$\lvert T_k \rvert$	1
3	$\left(\frac{1}{2}, \frac{1}{2}\right)$	$\lvert T_k \rvert/3$	2
	$\left(\frac{1}{2}, 0\right)$	$\lvert T_k \rvert/3$	
	$\left(0, \frac{1}{2}\right)$	$\lvert T_k \rvert/3$	
7	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$0.225\lvert T_k \rvert$	5
	$\left(\frac{6-\sqrt{15}}{21}, \frac{6-\sqrt{15}}{21}\right)$	$\frac{(155-\sqrt{15})\lvert T_k \rvert}{1200}$	
	$\left(\frac{6-\sqrt{15}}{21}, \frac{9+2\sqrt{15}}{21}\right)$	$\frac{(155-\sqrt{15})\lvert T_k \rvert}{1200}$	
	$\left(\frac{9+2\sqrt{15}}{21}, \frac{6-\sqrt{15}}{21}\right)$	$\frac{(155-\sqrt{15})\lvert T_k \rvert}{1200}$	
	$\left(\frac{6+\sqrt{15}}{21}, \frac{6+\sqrt{15}}{21}\right)$	$\frac{(155+\sqrt{15})\lvert T_k \rvert}{1200}$	
	$\left(\frac{6+\sqrt{15}}{21}, \frac{9-2\sqrt{15}}{21}\right)$	$\frac{(155+\sqrt{15})\lvert T_k \rvert}{1200}$	
	$\left(\frac{9-2\sqrt{15}}{21}, \frac{6+\sqrt{15}}{21}\right)$	$\frac{(155+\sqrt{15})\lvert T_k \rvert}{1200}$	
3	$(0,0)$	$\lvert T_k \rvert/3$	1
	$(1,0)$	$\lvert T_k \rvert/3$	
	$(0,1)$	$\lvert T_k \rvert/3$	

Voilà une formule bien utile d'ou son nom :

Proposition 4.7. *(formule magique) Pour calculer, l'integral sur un d-simplex K du produit de puissance entiere $n_i >= 0$ des coordonnées barycentrique λ_i de ce simplex, il suffit utiliser la formule suivante :*

$$\int_K \prod_{i=0}^{d} \lambda_i^{n_i} = |K| \frac{d! \prod_{i=0}^{d} n_i!}{(d + \sum_{i=0}^{d} n_i)!} \tag{84}$$

Et donc

$$\int_K \lambda_i = |K|\frac{1}{(d+1)}, \quad et \quad \int_K \lambda_i\lambda_j = |K|\frac{1+\delta_{ij}}{(d+1)(d+2)} \tag{85}$$

Démonstration. Le démonstration ce fait par récurrence sur la dimension de l'espace d sur le simplex de référence $\hat{K}_d = (0, \boldsymbol{e}_1, \cdots, \boldsymbol{e}_d)$ où les $\boldsymbol{e}_i$ sont les vecteurs de base canonique de $\mathbb{R}^d$.

– Pour $d = 1$, il faut montrer que

$$\int_0^1 x^{n_1}(1-x)^{n_0} = \frac{n_0!n_1!}{(1+n_0+n_1)!} \tag{86}$$

si $n_1 = 0$ c'est vrai, puis il suffit de faire une récurrence sur n_0,pour tout n_1. L'héritage, quelque soit $n_1 >= 0$, on suppose la propriété vraie pour n_0 quelque soit $n_1 >= 0$ en intégrant par partie et en utilisant l'héritage pour $n_1 + 1$ et n_0 on a :

$$\int_0^1 x^{n_1}(1-x)^{n_0+1} = \frac{n_0+1}{n_1+1}\int_0^1 x^{n_1+1}(1-x)^{n_0} = \frac{n_0+1}{n_1+1}\frac{(n_1+1)!n_0!}{(2+n_0+n_1)!} = \frac{n_1!(n_0+1)!}{(2+n_0+n_1)!}$$

ce qui fini le cas $d = 1$.

– Supposons la formule vraie pour $d-1$, et montrons la formule pour d. Nous avons ajouter une dimension en nous avons en utilisants Fubini

$$\int_{\hat{K}_d}\prod_{i=0}^{d}\lambda_i^{n_i} = \int_0^1 \lambda_d^{n_d}\left(\int_{(1-x_d)\hat{K}}\prod_{i=0}^{d-1}\lambda_i^{n_i}dx_1\cdots dx_{d-1}\right)dx_d \tag{87}$$

Où, $(1-x_d)K_{d-1}$ est intersection de K_d avec l'hyperlan de dernière composante égala à x_d, et il suffit de remarquer que la mesure de $(1-x_d)\hat{K}_{d-1}$ est égale à $(1-x_d)^{d-1}|\hat{K}_{d-1}|$ et que les $\lambda_i^{\hat{K}_d}{}_{|(1-x_d)K_{d-1}} = (1-x_d)\lambda_i^{(1-x_d)K_{d-1}}$ pour $i \leq d$.
Ceci nous donne donc :

$$\int_{\hat{K}_d}\prod_{i=0}^{d}\lambda_i^{n_i} = \int_0^1 \lambda_d^{n_d}(1-x^d)^{d-1+\sum_{i=0}^{d-1}n_i}\left(\int_{\hat{K}_{d-1}}\prod_{i=0}^{d-1}\lambda_i^{n_i}dx_1\cdots dx_{d-1}\right)dx_d \tag{88}$$

En appliquant l'hypothèse de récurrence et en remarquons que $\lambda_d^{\hat{K}_d} = x_d$ on a :

$$\int_{\hat{K}_d}\prod_{i=0}^{d}\lambda_i^{n_i} = |\hat{K}_{d-1}|\frac{(d-1)!\prod_{i=0}^{d-1}n_i!}{(d-1+\sum_{i=0}^{d-1}n_i)!}\int_0^1 x_d^{n_d}(1-x^d)^{d-1+\sum_{i=0}^{d-1}n_i}dx_d \tag{89}$$

Il suffit appliquer la formule dans le cas $d = 1$, et $|K_d| = |K_{d-1}|/d$ pour avoir

$$\int_{\hat{K}_d}\prod_{i=0}^{d}\lambda_i^{n_i} = |\hat{K}_d|\frac{d(d-1)!\prod_{i=0}^{d-1}n_i!}{(d-1+\sum_{i=0}^{d-1}n_i)!}\frac{n_d!(d-1+\sum_{i=0}^{d-1}n_i)!}{(d+\sum_{i=0}^{d}n_i)!} \tag{90}$$

En regroupe les termes et simplifiant les termes on obtient les résultats.

□

4.5 Le maillage

Commençons par définir la notion de maillage simplexial.

Définition 4.8. *Un maillage simplexial $\mathcal{T}_{d,h}$ d'un ouvert polygonal $\mathcal{O}_h$ de $\mathbb{R}^d$ est un ensemble de $d-$ simplex K^k de $\mathbb{R}^d$ pour $k = 1, N_t$ (triangle si $d = 2$ et tétraèdre si $d = 3$), tel que l'intersection de deux d-simplex distincts $\overline{K}^i, \overline{K}^j$ de $\mathcal{T}_{d,h}$ soit :*
– l'ensemble vide,
– ou p-simplex commun à K et K' avec $p \leq d$

Le maillage $\mathcal{T}_{d,h}$ couvre l'ouvert défini par :

$$\mathcal{O}_h \stackrel{def}{=} \overline{\overset{\circ}{\bigcup_{K\in\mathcal{T}_{d,h}} K}} \tag{91}$$

De plus, $\mathcal{T}_{0,h}$ désignera l'ensemble des sommets de $\mathcal{T}_{d,h}$ et $\mathcal{T}_{1,h}$ l'ensemble des arêtes de $\mathcal{T}_{d,h}$ et l'ensemble de faces serat $\mathcal{T}_{d-1,h}$. Le bord $\partial\mathcal{T}_{d,h}$ du maillage $\mathcal{T}_{d,h}$ est défini comme l'ensemble des faces qui ont la propriété d'appartenir à un unique d-simplex de $\mathcal{T}_{d,h}$. Par conséquent, $\partial\mathcal{T}_{d,h}$ est un maillage du bord $\partial\mathcal{O}_h$ de $\mathcal{O}_h$. Par abus de langage, nous confondrons une arête d'extrémités (a, b) et le segment ouvert $]a, b[$, ou fermé $[a, b]$.

4.6 Le problème et l'algorithme

Le problème est de résoudre numériquement l'équation de la chaleur dans un domaine Ω de $\mathbb{R}^2$.

$$-\Delta u = f, \quad \text{dans } \Omega, \quad u = g \quad \text{sur } \partial\Omega. \tag{92}$$

Nous utiliserons la discrétisation par des éléments finis P_1 Lagrange construite sur un maillage $\mathcal{T}_{d,h}$ de Ω. Notons V_h l'espace des fonctions éléments finis et V_{0h} les fonctions de V_h nulle sur bord de Ω_h (ouvert obtenu comme l'intérieur de l'union des triangles fermés de $\mathcal{T}_{d,h}$).

$$V_h = \{v \in \mathcal{C}^0(\Omega_h)/\forall K \in \mathcal{T}_{d,h}, v|_K \in P_1(K)\} \tag{93}$$

Après utilisation de la formule de Green, et en multipliant par v, le problème peut alors s'écrire :

Calculer $u_h^{n+1} \in V_h$ à partir de u_h^n , où la donnée initiale u_h^0 est interpolé P_1 de u_0.

$$\int_\Omega \nabla u_h \nabla v_h = \int_\Omega f v_h, \quad \forall v_h \in V_{0h}, \tag{94}$$

$$u_h = g \quad \text{sur les sommets de } \mathcal{T}_{d,h} \text{ dans } \partial\Omega$$

Nous nous proposons de programmer cette méthode l'algorithme du gradient conjugue pour résoudre le problème linéaire car nous avons pas besoin de connaître explicitement la matrice, mais seulement, le produit matrice vecteur.
Puis, notons, $(w^i)_{i=1,N_s}$ les fonctions de base de V_h associées aux N_s sommets de $\mathcal{T}_{d,h}$ de coordonnées $(q^i)_{i=1,N_s}$, tel que $w^i(q_j) = \delta_{ij}$. Notons, U_i le vecteur de $\mathbb{R}^{N_s}$ associé à u et tel que $u = \sum_{i=1,N_s} U_i w^i$.

Sur un triangle K formé de sommets i, j, k tournants dans le sens trigonométrique. Notons, H_K^i le vecteur hauteur pointant sur le sommet i du triangle K et de longueur l'inverse de la hauteur, alors on a comme dans (76) :

$$\nabla w^i|_K = H_K^i = \frac{\overrightarrow{(q^j q^k)}^\perp}{2\, aire_K} \tag{95}$$

où l'opérateur $\perp$ de $\mathbb{R}^2$ est défini comme la rotation de $\pi/2$, ie. $(a,b)^\perp = (-b,a)$.

Algorithme 2. **Le programme d'éléments finis sans matrice**

1. *Soit la matrice $\boldsymbol{M}_{\alpha,\beta}$ associée la forme bilinéaire*

$$\boldsymbol{a}_{\alpha,\beta}(u,v) = \int_\Omega \alpha uv + \int_\Omega \beta \nabla u.\nabla v$$

et la matrice élémentaire 3×3 $\boldsymbol{M}^K$ associée la forme bilinéaire

$$\boldsymbol{a}^K_{\alpha,\beta}(u,v) = \int_K \alpha uv + \int_K \beta \nabla u.\nabla v$$

sur l'espace des fonctions $P_1(K)$ dans la base $\lambda_i^K, i = 0,1,2$.

2. *Pour la méthde du gradient conjugue nous avons juste besoin de la méthode* `addMatMul` *qui ajoute à y le produit matrice vecteur Ax, c'est à dire :*

$$y+ = \boldsymbol{M}_{\alpha,\beta} x$$

Ce cas s'écrit mathématiquement :
Pour $K \in \mathcal{T}_{d,h}$, pour $i = 0,1,2$ et pour $j = 0,1,2$ faire :

$$y_{i_i^K} + = \begin{cases} \boldsymbol{M}_{ij}^K \; x_{i_j^K} & si\ i_i^K \notin \Gamma \\ 0 & sinon \end{cases}$$

3. *On peut calculer $\boldsymbol{b} = (\int_\Omega w^i f_h)_i = \boldsymbol{M}_{1,0}\boldsymbol{f}_h$ en utilisant la formule où $\boldsymbol{f}_h$ est le vecteur de la fonction f_h qui est l'interpolé de f , c'est a dire $f_h = \sum_i f(x_i)w^i$ et $\boldsymbol{f}_h = (f(q_i))_i$.*
4. *La matrice A est simplement $M_{0,1}$.*

Pour finir, on initialiserons le gradient conjugué avec l'initilisation suivante :

$$\boldsymbol{x} = (x_i)_i = \begin{cases} 0 & si\ i\ n'est\ pas\ sur\ le\ bord \\ g(q_i) & si\ i\ est\ sur\ le\ bord \end{cases}$$

pour resoudre de système linéaire

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

avec la matrice $\boldsymbol{A} = \boldsymbol{M}_{0,1}$

4.7 Maillage et Triangulation 2D

4.8 Les classes de Maillages

Nous allons définir les outils informatiques en C++ pour utiliser des maillages, pour cela nous utilisons la classe R2 définie au paragraphe 3.1 page 54. Le maillage est formé de triangles qui sont définis par leurs trois sommets. Mais attention, il faut numéroter les sommets. La question classique est donc de définir un triangle : soit comme trois numéro de sommets, ou soit comme trois pointeurs sur des sommets (nous ne pouvons pas définir un triangle comme trois références sur des sommets car il est impossible initialiser des références par défaut). Les deux sont possibles, mais les pointeurs sont plus efficaces pour faire des calculs, d'où le choix de trois pointeurs pour définir un triangle. Maintenant les sommets seront stockés dans un tableau donc il est inutile de stocker le numéro du sommet dans la classe qui définit un sommet, nous ferons une différence de pointeur pour retrouver le numéro d'un sommet, ou du triangle du maillage.
Un maillage (classe de type Mesh) contiendra donc un tableau de triangles (classe de type Triangle) et un tableau de sommets (classe de type Vertex2), bien sur le nombre de triangles (nt) , le nombre de sommets (nv), de plus il me paraît naturel de voir un maillage comme un tableau de triangles et un triangle comme un tableau de 3 sommets.
Remarque : Les sources de ces classes et méthodes sont disponible dans l'archive
http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/EF.tar.bz2
avec un petit exemple.

4.8.1 La classe **Label**

Nous avons vu que le moyen le plus simple de distinguer les sommets appartenant à une frontière était de leur attribuer un numéro logique ou étiquette (*label* en anglais). Rappelons que dans le format FreeFem++ les sommets intérieurs sont identifiés par un numéro logique nul.
De manière similaire, les numéros logiques des triangles nous permettront de séparer des sous-domaines Ω_i, correspondant, par exemple, à des types de matériaux avec des propriétés physiques différentes.
La classe Label va contenir une seule donnée (lab), de type entier, qui sera le numéro logique.

Listing 7: *(label.hpp - la classe Label)*

```
 class Label {
  friend ostream& operator <<(ostream& f,const Label & r  )
  { f <<  r.lab; return f; }
  friend istream& operator >>(istream& f, Label & r  )
  { f >>  r.lab; return f; }
public:
  int lab;
  Label(int r=0):lab(r){}
  int onGamma() const { return lab;}
};
```

Cette classe n'est pas utilisée directement, mais elle servira dans la construction des classes pour les sommets et les triangles. Il est juste possible de lire, écrire un label, et tester si elle est nulle.

Listing 8: *(utilisation de la classe Label)*

```
Label r;
cout << r;                                          //    écrit r.lab
cin  >> r;                                          //    lit r.lab
if(r.onGamma()) { ..... }                    //    à faire si la r.lab != 0
```

4.8.2 La classe `Vertex2` (modélisation des sommets 2d)

Il est maintenant naturel de définir un sommet comme un point de $\mathbb{R}^2$ et un numéro logique `Label`. Par conséquent, la classe `Vertex2` va dériver des classes `R2` et `Label` tout en héritant leurs données membres et méthodes :

Listing 9: *(Mesh2d.hpp - la classe Vertex2)*

```
class Vertex2 : public R2,public Label {
  friend inline ostream& operator <<(ostream& f, const Vertex2 & v )
    { f << (R2) v << ' ' << (Label &) v  ; return f; }
  friend inline istream& operator >> (istream& f,  Vertex2 & v )
      { f >> (R2 &) v >> (Label &) v; return f; }
   public:
   Vertex2() : R2(),Label(){};
   Vertex2(R2 P,int r=0): R2(P),Label(r){}
   private:                          //    pas de copie pour ne pas prendre l'adresse
     Vertex2(const Vertex2 &);
     void operator=(const Vertex2 &);
};
```

Nous pouvons utiliser la classe `Vertex2` pour effectuer les opérations suivantes :

Listing 10: *(utilisation de la classe Vertex2)*

```
Vertex2 V,W;                                  //    construction des sommets V et W
cout << V ;                                          //    écrit V.x, V.y , V.lab
cin  >> V;                                           //    lit V.x, V.y , V.lab
R2  O = V;                                           //    copie d'un sommet
R2  M = ( (R2) V +  (R2) W ) *0.5;        //    un sommet vu comme un point de R2
(Label) V                    //    la partie label d'un sommet (opérateur de cast)
if (!V.onGamma()) { .... }     //    si V.lab = 0, pas de conditions aux limites
```

Remarque 7. *Les trois champs* `(x,y,lab)` *d'un sommet sont initialisés par* `(0.,0.,0)` *par défaut, car les constructeurs sans paramètres des classes de base sont appelés dans ce cas.*

4.8.3 La classe `Triangle` (modélisation des triangles)

Un triangle sera construit comme un tableau de trois pointeurs sur des sommets, plus un numéro logique (*label*). Nous rappelons que l'ordre des sommets dans la numérotation locale ($\{0,1,2\}$) suit le sens trigonométrique. La classe `Triangle` contiendra également une donnée supplémentaire, l'aire du triangle (`area`), et plusieurs fonctions utiles :
- `Edge(i)` qui calcule le vecteur «arête du triangle» opposée au sommet local i ;
- `H(i)` qui calcule directement le gradient de la i coordonnée barycentrique λ^i par la formule :

$$\nabla\lambda^i|_K = H_K^i = \frac{(q^j - q^k)^\perp}{2\,aire_K} \tag{96}$$

où l'opérateur $\perp$ de $\mathbb{R}^2$ est défini comme la rotation de $\pi/2$, ie. $(a,b)^\perp = (-b,a)$, et où les q^i, q^j, q^k sont les coordonnées des 3 sommets du triangle.

Listing 11: *(Mesh2d.hpp - la classe Triangle)*

```
class Triangle: public Label {
                                                          //    variable prive
  Vertex2 *vertices[3];                          //    an array of 3 pointer to vertex
  public:
  R area;
  Triangle() :area() {};                              //    constructor empty for array
  Vertex2 & operator[](int i) const {
     ASSERTION(i>=0 && i <3);
     return *vertices[i];}                  //    to see triangle as a array of vertex
  void init(Vertex2 * v0,int i0,int i1,int i2,int r)
    { vertices[0]=v0+i0; vertices[1]=v0+i1; vertices[2]=v0+i2;
      R2 AB(*vertices[0],*vertices[1]);
      R2 AC(*vertices[0],*vertices[2]);
      area = (AB^AC)*0.5;
      lab=r;
      ASSERTION(area>0);
    }
  R2 Edge(int i) const {ASSERTION(i>=0 && i <3);
       return R2(*vertices[(i+1)%3],*vertices[(i+2)%3]);}      //    opposite edge
vertex i
  R2 H(int i) const { ASSERTION(i>=0 && i <3);
       R2 E=Edge(i);return E.perp()/(2*area);}                          //    heigth

  void Gradlambda(R2 * GradL) const
  {
    GradL[1]= H(1);
    GradL[2]= H(2);
    GradL[0]=-GradL[1]-GradL[2];
```

```
  }

  R lenEdge(int i) const {ASSERTION(i>=0 && i <3);
      R2 E=Edge(i);return sqrt((E,E));}

                                                    //    Transformation: K̂ ↦ K
  R2 operator()(const R2 & Phat) const    {
    const R2 &A =*vertices[0];
    const R2 &B =*vertices[1];
    const R2 &C =*vertices[2];
    return (1-Phat.x- Phat.y)* A + Phat.x *B  +Phat.y*C ;}

 private:
   Triangle(const Triangle &);                //    pas de construction par copie
   void operator=(const Triangle &);          //    pas affectation par copy
 public:                  //   -- Ajoute 2007 pour un ecriture generique 2d 3d --
  R  mesure() const {return area;}
  static const int nv=3;
};
```

Remarque 8. *La construction effective du triangle n'est pas réalisée par un constructeur, mais par la fonction `init`. Cette fonction est appelée une seule fois pour chaque triangle, au moment de la lecture du maillage (voir plus bas la classe `Mesh`).*

Remarque 9. *Les opérateurs d'entrée-sortie ne sont pas définis, car il y a des pointeurs dans la classe qui ne sont pas alloués dans cette classe, et qui ne sont que des liens sur les trois sommets du triangle (voir également la classe `Mesh`).*

Regardons maintenant comment utiliser cette classe :

Listing 12: *(utilisation de la classe Triangle)*

```
                                  //    soit T un triangle de sommets A,B,C ∈ IR²
                                     //    ----------------------------------
  Triangle::nv;                    //    nombre de sommets d'un triangle (ici 3)
  const Vertex2 & V =  T[i];                  //    le sommet i de T (i ∈ {0,1,2}
  double a = T.area;                                    //    l'aire de T
  R2 AB = T.Edge(2);                        //    "vecteur arête" opposé au sommet 2
  R2 hC = T.H(2);       //    gradient de la fonction de base associé au sommet 2
  R l = T.lenEdge(i);               //    longueur de l'arête opposée au sommet i
  (Label) T  ;                                  //    la référence du triangle T
  R2 G(T(R2(1./3,1./3)));                         //    le barycentre de T
  Triangle T;
  T.init(v,ia,ib,ic,lab);     //    initialisation du triangle T avec les sommets
                                     //    v[ia],v[ib],v[ic] et l'étiquette lab
                                           //    (v est le tableau des sommets)
```

4.8.4 La classe `Seg` (modélisation des segments de bord)

On fait la même type de classe que les triangles mais juste avec deux sommets, on utilisera cette classe pour calculer les intégrales pour les conditions aux limites de type Neumann,

Listing 13: *(Mesh2d.hpp - la classe Seg)*

```
class Seg: public Label {
                                                          //    variable prive
  Vertex2 *vertices[2];                          //    an array of 3 pointer to vertex
public:
  R l;                                                      //    longueur du segment
  Seg() :l() {};                                  //    constructor empty for array
  Vertex2 & operator[](int i) const {
    ASSERTION(i>=0 && i <2);
    return *vertices[i];}                    //    to see traingle as a array of vertex
  void init(Vertex2 * v0,int * iv,int r)
  {
    vertices[0]=v0+iv[0];
    vertices[1]=v0+iv[1];
    R2 AB(*vertices[0],*vertices[1]);
    l=  AB.norme();
    lab=r;
    ASSERTION(l>0);
  }


                                                        //    Transformation: [0,1] ↦ K
  R2 operator()(const R & Phat) const    {
    const R2 &A =*vertices[0];
    const R2 &B =*vertices[1];
    return (1-Phat)* A + Phat *B ;}

private:
  Seg(const Seg &);                              //    pas de construction par copie
  void operator=(const Seg &);                   //    pas affectation par copy
public:
  R  mesure() const {return l;}
  static const int nv=2;
};
```

Regardons maintenant comment utiliser cette classe :

Listing 14: *(utilisation de la classe Seg)*

```
                                                  //    soit K un Seg de sommets A, B ∈ ℝ²
                                                //    ----------------------------------
  Seg::nv;                                      //    nombre de sommets d'un Seg (ici 3)
  const Vertex2 & V =  K[i];                          //    le sommet i de T (i ∈ {0,1}
```

```
  double a = K.l;                                          //    la longueur de K
  (Label) K  ;                                          //    le label du triangle Seg
  R2 G(T(0.5)));                                           //    le barycentre de K
  Seg K;
  K.init(v,ia,ib,lab);              //    initialisation d'un Seg avec les sommets
                                        //    v[ia],v[ib] et l'étiquette lab
                                        //    (v est le tableau des sommets)
```

4.8.5 La classe `Mesh2` (modélisation d'un maillage 2d)

Nous présentons, pour finir, la classe maillage (`Mesh`) qui contient donc :

- le nombre de sommets (`nv`), le nombre de triangles, le nombre de arêtes de bord (`nbe`) (`nt`) ;
- le tableau des sommets ;
- le tableau des triangles ;
- le tableau des arêtes du bord ;

Listing 15: *(Mesh2d.hpp - la classe Mesh)*

```
class Mesh2
{
public:
  typedef Triangle Element;
  typedef Seg  BorderElement;
  typedef R2 Rd;
  typedef Vertex2  Vertex;
  int nv,nt, nbe;
  R area,peri;
  Vertex2 *vertices;
  Triangle *triangles;
  Seg * borderelements;
  Triangle & operator[](int i) const {return triangles[CheckT(i)];}
  Vertex2 & operator()(int i) const {return vertices[CheckV(i)];}
  Seg & be(int i) const {return borderelements[CheckBE(i)];}
  Mesh2(const char * filename);                              //    read on a file
                                                          //    to get numbering:
  int operator()(const Triangle & t) const {return CheckT(&t - triangles);}
  int operator()(const Triangle * t) const {return CheckT(t - triangles);}
  int operator()(const Vertex2 & v) const {return CheckV(&v - vertices);}
  int operator()(const Vertex2  * v) const{return CheckV(v - vertices);}
  int operator()(const Seg & v) const {return CheckBE(&v - borderelements);}
  int operator()(const Seg  * v) const{return CheckBE(v - borderelements);}

  //    the global Number of vertex j of triangle it (j=0,1,2, it=0, .., nt-1)
  int operator()(int it,int j) const {return (*this)(triangles[it][j]);}
                                                          //    to check the bound
  int CheckV(int i) const { ASSERTION(i>=0 && i < nv); return i;}
  int CheckT(int i) const { ASSERTION(i>=0 && i < nt); return i;}
  int CheckBE(int i) const { ASSERTION(i>=0 && i < nbe); return i;}
  ~Mesh2() { delete [] vertices; delete [] triangles;delete [] borderelements; }
 private:
   Mesh2(const Mesh2 &);                        //    pas de construction par copie
```

```
   void operator=(const Mesh2 &);                    //    pas affectation par copy
};
```

Avant de voir comment utiliser cette classe, quelques détails techniques nécessitent plus d'explications :

- Pour utiliser les opérateurs qui retournent un numéro, il est fondamental que leur argument soit un pointeur ou une référence ; sinon, les adresses des objets seront perdues et il ne sera plus possible de retrouver le numéro du sommet qui est donné par l'adresse mémoire.
- Les tableaux d'une classe sont initialisés par le constructeur par défaut qui est le constructeur sans paramètres. Ici, le constructeur par défaut d'un triangle ne fait rien, mais les constructeurs par défaut des classes de base (ici les classes `Label, Vertex2`) sont appelés. Par conséquent, les labels de tous les triangles sont initialisées et les trois champs `(x,y,lab)` des sommets sont initialisés par `(0.,0.,0)`. Par contre, les pointeurs sur sommets sont indéfinis (tout comme l'aire du triangle).

Tous les problèmes d'initialisation sont résolus une fois que le constructeur avec arguments est appelé. Ce constructeur va lire le fichier `.msh` contenant la triangulation.

Listing 16: *(Mesh2d.cpp - constructeur de la classe Mesh)*

```
#include <cassert>
#include <fstream>
#include <iostream>
#include "ufunction.hpp"
#include "Mesh2d.hpp"

Mesh2::Mesh2(const char * filename)
  : nv(0),nt(0),nbe(0),
    area(0),peri(0),
    vertices(0),triangles(0),borderelements(0)
{                                                          //    read the mesh
  int i,iv[3],ir;
  ifstream f(filename);
  if(!f) {cerr << "Mesh2::Mesh2 Erreur openning " << filename<<endl;exit(1);}
  cout << " Read On file \\"" <<filename<<"\\""<<  endl;
  f >> nv >> nt >> nbe;
  cout << " Nb of Vertex " << nv << " Nb of Triangles " << nt
       << " Nb of Border Seg : " << nbe << endl;
  assert(f.good() && nt && nv );
  triangles = new Triangle [nt];
  vertices = new Vertex[nv];
  borderelements = new Seg[nbe];
  area=0;
  assert(triangles && vertices);
  for (i=0;i<nv;i++)
    {
      f >> vertices[i];
      assert(f.good());
    }
```

```
  for (i=0;i<nt;i++)
    {
      f >> iv[0] >> iv[1] >> iv[2] >> ir;
      assert(f.good() && iv[0]>0 && iv[0]<=nv && iv[1]>0
             && iv[1]<=nv && iv[2]>0 && iv[2]<=nv);
      for (int v=0;v<3;++v) iv[v]--;
      triangles[i].init(vertices,iv,ir);
      area += triangles[i].area;
    }
  for (i=0;i<nbe;i++)
    {
      f >> iv[0] >> iv[1]  >> ir;
      assert(f.good() && iv[0]>0 && iv[0]<=nv && iv[1]>0 && iv[1]<=nv);
      for (int v=0;v<2;++v) iv[v]--;
      borderelements[i].init(vertices,iv,ir);
      peri += borderelements[i].l;
    }

  cout << " End of read: area = " << area << "  perimeter: " << peri << endl;
}
```

L'utilisation de la classe `Mesh` pour gérer les sommets, les triangles devient maintenant très simple et intuitive.

Listing 17: *(utilisation de la classe Mesh2)*

```
  Mesh2 Th("filename");            //    lit le maillage Th du fichier "filename"
  Th.nt;                                                //    nombre de triangles
  Th.nv;                                                  //    nombre de sommets
  Th.neb;                                              //    nombre de éléments de bord
  Th.area;                                              //    aire du domaine de calcul
  Th.peri;                                          //    perimetre du domaine de calcul

  Triangle & K = Th[i];                                 //    triangle i , int i∈ [0,nt[
  R2 A=K[0];                                 //    coordonnée du sommet 0 sur triangle K
  R2 G=K(R2(1./3,1./3)):                                //    le barycentre de K.
  R2 DLambda[3];
  K.Gradlambda(DLambda);                        //    calcul des trois ∇λ_i^K pour i = 0,1,2
  Vertex2   & V = Th(j);                                //    sommet j , int j∈ [0,nv[
  Seg &   BE=th.be(l);                                  //    Seg du bord, int l∈ [0,nbe[
  R2 B=BE[1];                                //    coordonnée du sommet 1 sur Seg BE
  R2 M=BE(0.5);                                         //    le milieu de BE.
  int j = Th(i,k);       //    numéro global du sommet k∈ [0,3[ du triangle i∈ [0,nt[
  Vertex2 & W=Th[i][k];      //    référence du sommet k∈ [0,3[ du triangle i∈ [0,nt[

  int ii = Th(K);                                       //    numéro du triangle K
  int jj = Th(V);                                       //    numéro du sommet V
  int ll = Th(BE);                                   //    numéro de Seg de bord BE

  assert( i == ii && j == jj);                                  //    vérification
```

4.9 Le programme quasi générique

les sources sont dans `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/EF2D.tgz`

```
#include <cassert>
#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <map>
#include "ufunction.hpp"
#include "Mesh2d.hpp"
#include "RNM.hpp"
#include "GC.hpp"

#include "cputime.h"

              //    les type elements finis ... pour un passage on trois trivial.
typedef Mesh2 Mesh;
typedef Mesh::Rd Rd;
typedef Mesh::Element Element;
typedef Mesh::Vertex Vertex;
typedef double R;

R f(const R2 & ){return -6.;}                                    //    right hand side
R g(const R2 & P){return P.x*P.x+2*P.y*P.y;}                //    boundary condition
R u0(const R2 & ){return 0;}                                      //    initialization

                                                          //    calcul de la matrice M^K
void MatElement(const Element & K,R alpha,R beta,R matK[Element::nv][Element::nv])
{
      const int nve = Element::nv;
      const int d = Rd::d;
      double  clilj = 1./((d+2)*(d+1));                          //    formule magique
      Rd Gradw[nve];
      K.Gradlambda(Gradw);
      R cgg=K.mesure()*beta, cuu=K.mesure()*alpha*clilj;
      for(int i=0;i<nve;++i)
        for(int j=0;j<=i;++j)
          matK[j][i] = matK[i][j] = cgg*(Gradw[i],Gradw[j])+cuu* ( 1. + (i==j) );
}

                                                  //    assemblage de de la matrice M^K_αβ
class MatLap: VirtualMatrice<R> {
public:
                                                                //    Matrice ∫ αuv + β∇u∇v
  const  Mesh & Th;
  const R alpha,beta;
  typedef    VirtualMatrice<R>::plusAx plusAx;
```

```
  MatLap(const  Mesh & T,R a,R b) : Th(T),alpha(a),beta(b) {};
  void addMatMul(const  KN_<R>  & x, KN_<R> & Ax) const
  {
    const int nve = Element::nv;

    for (int k=0;k<Th.nt;k++)
      {
        const Element & K(Th[k]);
        int iK[nve];
        R xK[nve],matKx[nve],matK[nve][nve];
        MatElement(K,alpha,beta,matK);
        for (int i=0;i<nve;++i)
          {
            iK[i]=Th(K[i]);
            xK[i]=x[iK[i]];
            matKx[i]=0.;
          }
        for (int i=0;i<nve;++i)
          for (int j=0;j<nve;++j)
            matKx[j] += matK[j][i]*xK[i];

        for(int i=0;i<nve;++i)
          if ( ! K[i].onGamma() ) Ax[iK[i]] +=  matKx[i];
      }
  }
  plusAx operator*(const KN<R> &  x) const {return plusAx(this,x);}
};


int Lap(int argc, char** argv )
{

  assert(argc>1);
  Mesh Th(argv[1]);

  const  int nve = Element::nv;
  KN<R> fh(Th.nv);                                                    //     fh[i] = Π_h f
  for (int k=0;k<Th.nt;k++)
    {
      const Element & K(Th[k]);
      for (int i=0;i<nve;i++)
        fh[Th(k,i)] = f(K[i]);
    }
  KN<R> b(Th.nv);                                                     //     b[i] = ∫_Ω f_h w_i
  KN<R> e(Th.nv);                                                     //     e[i]
  MatLap M(Th,1.,0.);                                          //    Matrice de Masse
  MatLap A(Th,0.,1);                                        //    Matrice du Laplacien

  MatriceIdentite<R> Id;                                       //    Matrice Identite
  b = M*fh;

  KN<R> x(Th.nv);
  x=0.;                          //    donne initial avec les conditions aux limites.
  int ii=0;
  for (int k=0;k<Th.nv;k++)
    if(Th(k).onGamma())  ii++;
  for (int k=0;k<Th.nv;k++)
```

```
    e[k]=g(Th(k));
  cout << " nb sommet on gamma = " << ii << endl;

  for (int k=0;k<Th.nt;k++)
    {
      Element & K(Th[k]);
      for (int i=0;i<Element::nv;i++)
        if(K[i].onGamma())  x[Th(K[i])] = g(K[i]);
        else                      x[Th(K[i])] = u0(K[i]);
    }

  R cpu0= CPUtime();
  R eps;
  int res:
  res=GradienConjugue(A,Id, b,x,Th.nv,eps=1e-10);
  R cpu1=CPUtime()-cpu0;
  cout << " CPU = " << cpu1 << " second " << endl;

  e -= x;
  cout << "err= " << e.min() << " " << e.max() << endl;

  {
    ofstream f("x.sol");
    f << x << endl;
  }

  return 0;
}

int main(int argc, char** argv )
{
  return Lap(argc,argv);
}
```

Si l'on veut utiliser la méthode GMRES pour la résolution, il suffit de remplace la ligne

```
  GradienConjugue(A,Id, b,x,Th.nv,eps=1e-10);
```

par

```
      KNM<R> H(nbkrylov+1,nbkrylov+1);     //    taille max de l'espace de krylov
      res=GMRES(A,x,b,Id,H,nbkrylov,nitermax,eps);
```

et bien sur il faut ajouter l'include à `"gmres.hpp"` en tête de fichier.

4.10 Construction avec des matrices creuse

L'idée est trés simple, si l'on réfléchi un peu, une matrice creuse n'est qu'une structure qui a une paire d'entier i, j (la clef) associe une valeur a_{ij}. Ceci ressemble formellement à un dictionnaire, hors en `c++`, dans la `STL` (voir chapitre 6 page 97), il y a un container `map` qui correspond à une structure de dictionnaire qui a une clef associe une valeur. Cette objet est automatiquement trié par rapport aux clefs. De plus dans la `STL`, il y a une classe `pair` qui permet de définir des

paire d'objets quelconque avec de plus l'ordre lexicographique défini par défaut, et la fonction `make_pair(i,j)` que crée des paires de i, j.

D'où le code suivant pour définir des matrices creuse de type `map` après avoir ajoute `#include <map>` en tête de programme.

```
typedef map<pair<int,int>,R>  MatriceCreuse;

class MatriceMap: VirtualMatrice<R> { public:
    typedef    VirtualMatrice<R>::plusAx plusAx;
    typedef map<pair<int,int>,R> Map;

    const Map &m;
    MatriceMap(const Map & mm) :m(mm) {}
    void addMatMul(const KN_<R> & x, KN_<R> & Ax) const;
    plusAx operator*(const KN<R> &  x) const {return plusAx(this,x);}
};

void MatriceMap::addMatMul(const KN_<R> & x, KN_<R> & Ax) const {
   //    parcours des éléments d'un contenaire de la STL voir section 6.1 page 97
    for ( Map::const_iterator k=this->m.begin(); k != this->m.end(); ++k)
    {
  int i= k->first.first;
  int j= k->first.second;
  R  aij= k->second;
  Ax[i] += aij*x[j];
    }
}
```

Maintenant pour générer la matrice creuse, il suffit d'écrire :

```
void BuildMatMap(MatriceCreuse &M, const  Mesh & Th, const R alpha,const R beta)
{
  typedef   Mesh::Element Element;
  const int nve=Element::nv;
  int iK[nve];
  R matK[nve][nve];
  for (int k=0;k<Th.nt;k++)
    {
      const Element & K(Th[k]);

      for(int i=0;i<nve;i++)
        iK[i]=Th(K[i]);

      MatElement(K,alpha,beta,matK);
                                                                    //    assemblage
      for (int i=0;i<nve;++i)
       for (int j=0;j<nve;++j)
         if(fabs(matK[i][j])>1e-30)
          M[make_pair(iK[i],iK[j])] += matK[i][j];
    }
}
```

L'exemple complet est dans le programme `EF2dMat.cpp` de l'archive

http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/EF2D.tgz.
Mais, malheuresement, ce code n'est pas très éfficace car le parcours d'une map est assez cher $nlog(n)$ est donc, nous allons optimiser le programme, en construisant une vrai classe matrice creuse de nom SparseMatrix.
Nous allons faire une version simple mais très efficace, nous allons simplement stockes trois tableaux i, j, a de taille le nombre de coefficients non nulle note nbcoef. On a donc pour une matrice a_{ij} simplement $a_{i[k],j[k]} = a[k]$.
les membres et méthode de la classe SparseMatrix sont :

```
template<class R>
class SparseMatrix: public VirtualMatrice<R>
{                                                    //    A sparse matrice n × m
public:
  int n,m;
  int nbcoef;                                        //    nombre de trem non nulle
  int *i;                                          //    tableau des indices des lignes
  int *j;                                        //    tableau des indices des colonnes
  R *a;                                            //    tableau dev valeurs des coef

     //    a_ij : pour k=0,..., nbcoef-1, this->a[k], j= this->j[j], i= this->i[k]


  template<class Mesh>
  SparseMatrix(Mesh & Th);           //    un constructeur a partir d'un maillage

  SparseMatrix(int nn,int mm, map<pair<int,int>,R> M);        //    constrution à
                                                             //   partir d'une map
  R *pcoef(int ii,int jj);          //    return si il existe le pointeur sur a_ii,jj

  static R & check(R * p){assert(p);return *p;} //   vérife si le pointeur existe
  static const R & check(const R * p){assert(p);return *p;}            //    même


  const R & operator()(int i,int j) const  {return check(pcoef(i,j);}
        R & operator()(int i,int j)        {return check(pcoef(i,j);}

  //    pour avoir le même code qu'avec une map.
   const  R & operator[](pair<int,int> ij) const
                              {return check(pcoef(ij.first,ij.second);}
          R & operator[](pair<int,int> ij)
                              {return check(pcoef(ij.first,ij.second);}

  int size() const { return nbcoef;}
  void clear() { for(int i=0;i<nbcoef;++i) a[i]=R(); }
  //    fin des methodes pour ressembler à une map.

                                     //    les méthodes pour les matrices virtual.
  typedef  typename VirtualMatrice<R>::plusAx plusAx;
  void addMatMul(const  KN_<R>  & x, KN_<R> & Ax) const
  {
    for(int k=0;k<nbcoef;++k)
      Ax[i[k]] += a[k]*x[j[k]];
  }
  plusAx operator*(const KN<R> &  x) const {return plusAx(this,x);}
```

```
  ~SparseMatrix() { delete [] i; delete [] j; delete [] a;}
private:
  SparseMatrix(const SparseMatrix &);
  void operator=(const SparseMatrix &);
};
```

Voilà le code pour construire la matrice à partir d'une map.

```
template<class R>
SparseMatrix<R>::SparseMatrix(int nn,int mm, map<pair<int,int>,R> M)
  :
  VirtualMatrice<R>(nn,mm),
  n(nn),
  m(mm),
  nbcoef(M.size()),
  i(new int[nbcoef]),
  j(new int[nbcoef]),
  a(new R[nbcoef])
{
  R cmm=0;

  int k=0;
  for (typename map<pair<int,int>,R>::const_iterator p=M.begin();
       p != M.end();
       ++p, ++k)
    {
      this->i[k]=p->first.first;
      this->j[k]=p->first.second;
      this->a[k]=p->second;
                              //   assert(i[k]<n && j[k] <m && i[k]>=0 && j[k]>=0);
      cmm=max(cmm,this->a[k]);
    }
  cout <<  " nb coef = " << nbcoef << " c max = "<< cmm << endl;
  assert(k==this->nbcoef);
}
```

Et voilà la fonction qui recherche le pointeur sur coefficient ii, jj par dichotomie si il existe. Mais bien sur il est supposé que les éléments sont triés en i, j de manière lexicographique, et donc cette algorithme est en $o(log_2(nbcoef))$.

```
template<class R>
R * SparseMatrix<R>::pcoef(int ii,int jj)
{
                                          //    pas de probleme avec les bornes car
                                                //    on supprime le milieu.
  int k0=0,k1=nbcoef-1;
  while (k0<=k1)
    {
      int km=(k0+k1)/2;
      int aa=0;
      if ( i[km] > ii)
        aa=-1;
      else if (i[km]<ii)
        aa=1;
```

```
      else  if ( j[km] > jj)
        aa=-1;
      else if (j[km]<jj)
        aa=1;
      if(aa<0) k1=km-1;
      else if (aa>0) k0=km+1;
      else { return a+km; }
    }
  return 0;
}
```

La construction avec un maillage sera faite dans le chapitre suivant voir la section 5.8.3 page 124.

5 Algorithmique

5.1 Introduction

Dans ce chapitre, nous allons décrire les notions d'algorithmique élémentaire, La notions de complexité.
Puis nous présenterons les chaînes et de chaînages ou liste d'abord d'un point de vue mathématique, puis nous montrerons par des exemples comment utiliser cette technique pour écrire des programmes très efficaces et simple.
Rappelons qu'une chaîne est un objet informatique composée d'une suite de maillons. Un maillon, quand il n'est pas le dernier de la chaîne, contient l'information permettant de trouver le maillon suivant. Comme application fondamentale de la notion de chaîne, nous commencerons par donner une méthode efficace de construction de l'image réciproque d'une fonction.
Ensuite, nous utiliserons cette technique pour construire l'ensemble des arêtes d'un maillage, pour trouver l'ensemble des triangles contenant un sommet donné, et enfin pour construire la structure creuse d'une matrice d'éléments finis.

5.2 Complexité algorithmique

Copié de http ://fr.wikipedia.org/wiki/Complexité algorithmique
La théorie de la complexité algorithmique s'intéresse à l'estimation de l'efficacité des algorithmes. Elle s'attache à la question : entre différents algorithmes réalisant une même tâche, quel est le plus rapide et dans quelles conditions ?
Dans les années 1960 et au début des années 1970, alors qu'on en était à découvrir des algorithmes fondamentaux (tris tels que quicksort, arbres couvrants tels que les algorithmes de Kruskal ou de Prim), on ne mesurait pas leur efficacité. On se contentait de dire : « cet algorithme (de tri) se déroule en 6 secondes avec un tableau de 50 000 entiers choisis au hasard en entrée, sur un ordinateur IBM 360/91. Le langage de programmation PL/I a été utilisé avec les optimisations standard ». (cet exemple est imaginaire)
Une telle démarche rendait difficile la comparaison des algorithmes entre eux. La mesure publiée était dépendante du processeur utilisé, des temps d'accès à la mémoire vive et de masse, du langage de programmation et du compilateur utilisé, etc.
Une approche indépendante des facteurs matériels était nécessaire pour évaluer l'efficacité des algorithmes. Donald Knuth fut un des premiers à l'appliquer systématiquement dès les premiers volumes de sa série The Art of Computer Programming. Il complétait cette analyse de considérations propres à la théorie de l'information : celle-ci par exemple, combinée à la formule de Stirling, montre qu'il ne sera pas possible d'effectuer un tri général (c'est-à-dire uniquement par comparaisons) de N élements en un temps croissant moins rapidement avec N que N ln N sur une machine algorithmique (à la différence peut-être d'un ordinateur quantique).

Pour qu'une analyse ne dépende pas de la vitesse d'exécution de la machine ni de la qualité du code produit par le compilateur, il faut utiliser comme unité de comparaison des « opérations élémentaires » en fonction de la taille des données en entrée. Exemples d'opérations élémentaires : accès à une cellule mémoire, comparaison de valeurs, opérations arithmétiques (sur valeurs à codage de taille fixe), opérations sur des pointeurs. Il faut souvent préciser quelles sont les opérations élémentaires pertinentes pour le problème étudié : si les nombre manipulés restent de taille raisonnable, on considérera que l'addition de deux entiers prend un temps

constant, quels que soient les entiers considérés (ils seront en effet codés sur 32 bits). En revanche, lorsque l'on étudie des problèmes de calcul formel où la taille des nombres manipulés n'est pas bornée, le temps de calcul du produit de deux nombre dépendra de la taille de ces deux nombres.
On définit alors la taille de la donnée sur laquelle s'applique chaque problème par un entier lié au nombre d'éléments de la donnée. Par exemple, le nombre d'éléments dans un algorithme de tri, le nombre de sommets et d'arcs dans un graphe.
On évalue le nombre d'opérations élémentaires en fonction de la taille de la donnée : si 'n' est la taille, on calcule une fonction t(n).
Les critères d'analyse : le nombre d'opérations élémentaires peut varier substantiellement pour deux données de même taille. On retiendra deux critères :
– analyse au sens du plus mauvais cas : t(n) est le temps d'exécution du plus mauvais cas et le maximum sur toutes les données de taille n. Par exemple, le tri par insertion simple avec des entiers présents en ordre décroissants.
– analyse au sens de la moyenne : comme le « plus mauvais cas » peut en pratique n'apparaître que très rarement, on étudie tm(n), l'espérance sur l'ensemble des temps d'exécution, où chaque entrée a une certaine probabilité de se présenter. L'analyse mathématique de la complexité moyenne est souvent délicate. De plus, la signification de la distribution des probabilités par rapport à l'exécution réelle (sur un problème réel) est à considérer.

On étudie systématiquemnt la complexité asymptotique, noté grâce aux notations de Landau.

idée 1 : évaluer l'algorithme sur des données de grande taille. Par exemple, lorsque n est 'grand', $3n^3 + 2n^2$ est essentiellement $3n^3$.

idée 2 : on élimine les constantes multiplicatrices, car deux ordinateurs de puissances différentes diffèrent en temps d'exécution par une constante multiplicatrice. De $3 * n^3$, on ne retient que n^3

L'algorithme est dit en $0(n^3)$.
L'idée de base est donc qu'un algorithme en $0(n^a)$ est « meilleur » qu'un algorithme en $0(n^b)$ si $a < b$.
Les limites de cette théorie :
le coefficient multiplicateur est oublié : est-ce qu'en pratique $100 * n^2$ est « meilleur » que $5*n^3$? l'occupation mémoire, les problèmes d'entrées/sorties sont occultés, dans les algorithmes numériques, la précision et la stabilité sont primordiaux. Point fort : c'est une notion indispensable pour le développement d'algorithmes efficaces.

Les principales classes de complexité :
– logarithmique : $\log_b n$
– linéaire : $an + b$
– polynomiale : $\sum_{i=0}^{n} a_i n^i$
– exponentielle : a^n
– factorielle : $n!$

Pour finir, il est aussi possible de parlé de la complexité mémoire, d'un algorithme.

5.3 Base, tableau, couleur

Recherche de la valeur minimal d'un tableau de `double u` de taille `N`.

```
double umax=u[0];
for (int i=1;i<N;++i)
  umax=max(umax,u[i]);
```

Si l'on veut connaitre l'indice `imax` associé à la plus grand valeur

```
int imax=0;
for (int i=1;i<N;++i)
  if(u[imax] < u[i])
    imax=i;
```

Exercice 13. Ecrire un programme trouve les 5 plus grande valeurs du tableau `u` avec une complexité de $O(N)$ en temps calcul et $O(1)$ en mémoire additionnelle.

5.3.1 Décalage d'un tableau

Nous voulons décaler le tableau de 1 comme suit formellement $u^{new}[i-1] = u^{old}[i], \forall i \in \{1, .., n-1\}$.

```
for (int i=1;i<N;++)
  u[i-1]=u[i];
```

Remarque, tout ce passe bien car la valeur de $u[i-1]$ n'est plus utilisé dans la boucle, pour les i suivants mais dans le cas contraire $u^{new}[i+1] = u^{old}[i], \forall i \in \{0, .., n-2\}$, il suffit de faire la boucle à l'envers pour éviter le problème d'écrasement.

```
for (int i=N-1; i>1 ;--i )
  u[i]=u[i-1];
```

Ou si l'on ne veut pas changer le sens de parcours, alors il suffit de stocker les dépendances dans une deux mémoires auxiliaires en $u[i-1]$ et $u[i]$, ce qui donne

```
u0=u[0];                              //    valeur précédente avant modification
for (int i=1;i<N;++)
  {
  u1=u[i];                                  //    valeur avant modification
  u[i]=u0;
  u0=u1;                              //    valeur précédente avant modification
  }
```

5.3.2 Renumérote un tableau

Soit `sigma` une permutation (bijection) de $\{0, ..., n-1\}$, le but est de changer l'ordre du tableau par la permutation σ ou par la permutation inverse $\sigma^{-}1$.
Il est difficile de faire ce type algorithme sur place, mais avec une copie, c'est trivail :

```
                                                    //    remumérotation
for (int i=0; i<n;++i )
  v[i]=u[sigma[i]];


                                                  //    remumérotation inverse
for (int i=0; i<n;++i )
  v[sigma[i]]=u[i];                          //    remarquons v[sigma[i]]=u[i];
```

Et donc pour construction la permutation inverse stocker dans le tableau `sigma1` il suffit d'ecrire :

```
                                                    //    permutation inverse
for (int i=0; i<n;++i )
  sigma1[sigma[i]]=i;
```

5.4 Construction de l'image réciproque d'une fonction

On va montrer comment construire l'image réciproque d'une fonction F. Pour simplifier l'exposé, nous supposerons que F est une fonction entière de $I = \{0, ..., n-1\}$ dans $J = \{0, ..., m-1\}$ et que ses valeurs sont stockées dans un tableau. Le lecteur pourra changer les bornes du domaine de définition ou de l'image sans grand problème.
Voici une méthode simple et efficace pour construire $F^{-1}(j)$ pour de nombreux j dans $\{0, ..., m-1\}$, quand n et m sont des entiers raisonnables. Pour chaque valeur $j \in Im\, F \subset \{0, ..., m-1\}$, nous allons construire la liste de ses antécédents. Pour cela nous utiliserons deux tableaux : `int head_F[m]` contenant les "têtes de listes" et `int next_F[n]` contenant la liste des éléments des $F^{-1}(i)$. Plus précisément, si i_1, i_2,...,$i_p \in [0, n]$, avec $p \geq 1$, sont les antécédents de j, `head_F[j]`=i_p, `next_F[`i_p`]`=i_{p-1}, `next_F[`i_{p-1}`]`=i_{p-2}, ..., `next_F[`i_2`]`=i_1 et `next_F[`i_1`]`= -1 (pour terminer la chaîne).
L'algorithme est découpé en deux parties : l'une décrivant la construction des tableaux `next_F` et `head_F`, l'autre décrivant la manière de parcourir la liste des antécédents.

Algorithme 3.

Construction de l'image réciproque d'un tableau

1. Construction :

```
int Not_In_I = -1;
 for (int j=0;j<m;j++)
  head_F[j]= Not_In_I;                  //    initialement, les listes
                                      //    des antécédents sont vides
 for (int i=0;i<n;i++)
   {j=F[i];next_F[i]=head_F[j];head_F[j]=i;}  //   chaînage amont
```

2. Parcours de l'image réciproque de j dans $[0, n]$:

```
 for (int i=head_F[j];i!= Not_In_I;i=next_F[i])
  { assert(F[i]==j);              //    j doit être dans l'image de i
                                                //    ... votre code
  }
```

Exercice 14. *Le pourquoi est laissé en exercice.*

5.5 Tri par tas (heap sort)

La fonction `HeapSort` voir `http://fr.wikipedia.org/wiki/Tri_par_tas` pour les explications du trie par tas. La complexité de ce trie est $O(nlog_2(n))$.

```
template<class T>
void  HeapSort(T *c,int n) {
  c--;              //    because fortran version array begin at 1 in the routine
  int m,j,r,i;
  T crit;
  if( n <= 1) return;
  m = n/2 + 1;
  r = n;
  while (1) {
    if(m <= 1 ) {
      crit = c[r];
      c[r--] = c[1];
    if ( r == 1 ) { c[1]=crit; return;}
    } else  crit = c[--m];
    j=m;
    while (1) {
      i=j;
      j=2*j;
      if  (j>r) {c[i]=crit;break;}
      if ((j<r) && c[j] < c[j+1])) j++;
      if (crit < c[j]) c[i]=c[j];
      else {c[i]=crit;break;}
    }
  }
}
```

5.6 Construction des arêtes d'un maillage

Rappelons qu'un maillage est défini par la donnée d'une liste de points et d'une liste d'éléments (des triangles par exemple). Dans notre cas, le maillage triangulaire est implémenté dans la classe `Mesh` qui a deux membres `nv` et `nt` respectivement le nombre de sommets et le nombre de triangle, et qui a l'opérateur fonction `(int j,int i)` qui retourne le numero de du sommet `i` du triangle `j`. Cette classe `Mesh` pourrait être par exemple :

```
  class Mesh { public:
   int nv,nt;                                  //    nb de sommet, nb de triangle
   int (* nu)[3];                                          //    connectivité
   double (* c)[2];                                 //    coordonnées de sommet
   int operator()(int i,int j) const { return nu[i][j];}
   Mesh(const char * filename);                     //    lecture d'un maillage
 }
```

Ou bien sur la classe défini en 4.8.5.

Dans certaines applications, il peut être utile de construire la liste des *arêtes du maillage*, c'est-à-dire l'ensemble des arêtes de tous les éléments. La difficulté dans ce type de construction réside dans la manière d'éviter – ou d'éliminer – les doublons (le plus souvent une arête appartient à deux triangles).
Nous allons proposer deux algorithmes pour déterminer la liste des arêtes. Dans les deux cas, nous utiliserons le fait que les arêtes sont des segments de droite et sont donc définies complètement par la donnée des numéros de leurs deux sommets. On stockera donc les arêtes dans un tableau `arete[nbex][2]` où `nbex` est un majorant du nombre total d'arêtes. On pourra prendre grossièrement `nbex = 3*nt` ou bien utiliser la formule d'Euler en 2D

$$\texttt{nbe} = \texttt{nt} + \texttt{nv} + \texttt{nb\_de\_trous} - \texttt{nb\_composantes\_connexes}, \tag{97}$$

où `nbe` est le nombre d'arêtes (*edges* en anglais), nt le nombre de triangles et nv le nombre de sommets (*vertices* en anglais).
La première méthode est la plus simple : on compare les arêtes de chaque élément du maillage avec la liste de *toutes* les arêtes déjà répertoriées. Si l'arête était déjà connue on l'ignore, sinon on l'ajoute à la liste. Le nombre d'opérations est $nbe * (nbe + 1)/2$.
Avant de donner le première algorithme, indiquons qu'on utilisera souvent une petite routine qui échange deux paramètres :

```
template<class T> inline void Exchange (T& a,T& b) {T c=a;a=b;b=c;}
```

Algorithme 4. **Construction lente des arêtes d'un maillage $\mathcal{T}_{d,h}$**

```
int ConstructionArete(const Mesh & Th,int  (* arete)[2])
{
  int SommetDesAretes[3][2] = { {0,1},{1,2},{2,0}};
  nbe = 0;                                        //    nombre d'arête;
  for(int t=0;t<Th.nt;t++)
   for(int et=0;et<3;et++) {
      int  i= Th(t,SommetDesAretes[et][0]);
      int  j= Th(t,SommetDesAretes[et][1]);
      if (j < i) Exchange(i,j)             //    on oriente l'arête
      bool existe =false;      //    l'arête n'existe pas a priori
       for (int e=0;e<nbe;e++)    //   pour les arêtes existantes
        if (arete[e][0] == i && arete[e][1] == j)
          {existe=true;break;}  //    l'arête est déjà construite
      if (!existe)                                 //    nouvelle arête
        arete[nbe][0]=i,arete[nbe++][1]=j;}
   return nbe;
}
```

Cet algorithme trivial est bien trop cher (en $O(9\,n^2)$) dès que le maillage a plus de 10^3 sommets (plus de 9 10^6 opérations). Pour le rendre de l'ordre du nombre d'arêtes, on va remplacer la boucle sur l'ensemble des arêtes construites par une boucle sur l'ensemble des arêtes ayant le même plus petit numéro de sommet. Dans un maillage raisonnable, le nombre d'arêtes incidentes sur un sommet est petit, disons de l'ordre de six, le gain est donc important : nous obtiendrons ainsi un algorithme en $9 \times nt$.

Pour mettre cette idée en oeuvre, nous allons utiliser l'algorithme de parcours de l'image réciproque de la fonction qui à une arête associe le plus petit numéro de ses sommets. Autrement dit, avec les notations de la section précédente, l'image par l'application F d'une arête sera le minimum des numéros de ses deux sommets. De plus, la construction et l'utilisation des listes, c'est-à-dire les étapes 1 et 2 de l'algorithme 3 seront simultanées.

Algorithme 5. **Construction rapide des arêtes d'un maillage $\mathcal{T}_{d,h}$**

```
int ConstructionArete(const Mesh & Th,int  (* arete)[2],
                      int nbex) {
   int SommetDesAretes[3][2] = { {1,2},{2,0},{0,1}};
   int end_list=-1;
   int * head_minv = new int [Th.nv];
   int * next_edge = new int [nbex];

   for ( int i =0;i<Th.nv;i++)
     head_minv[i]=end_list;                          //   liste vide

   nbe = 0;                                        //    nombre d'arête;

   for(int t=0;t<Th.nt;t++)
    for(int et=0;et<3;et++) {
      int  i= Th(t,SommetDesAretes[et][0]); //    premier sommet;
      int  j= Th(t,SommetDesAretes[et][1]);  //    second sommet;
      if (j < i) Exchange(i,j)              //    on oriente l'arête
      bool existe =false;      //    l'arête n'existe pas a priori
       for (int e=head_minv[i];e!=end_list;e = next_edge[e] )
                        //    on parcourt les arêtes déjà construites
        if ( arete[e][1] == j)  //    l'arête est déjà construite
          {existe=true;break;}                          //    stop
      if (!existe) {                          //    nouvelle arête
        assert(nbe < nbex);
        arete[nbe][0]=i,arete[nbe][1]=j;
                                      //    génération des chaînages
        next_edge[nbe]=head_minv[i],head_minv[i]=nbe++;}
      }
  delete []  head_minv;
  delete []  next_edge;
  return nbe;
}
```

Preuve : la boucle `for(int e=head_minv[i];e!=end_list;e=next_edge[e])` permet de parcourir toutes des arêtes (i, j) orientées $(i < j)$ ayant même i, et la ligne :

```
next_edge[nbe]=head_minv[i],head_minv[i]=nbe++;
```

permet de chaîner en tête de liste des nouvelles arêtes. Le `nbe++` incrémente pour finir le nombre d'arêtes. ∎

Remarque : les sources sont disponible : `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/mesh-liste/BuildEdges.cpp`.

Exercice 15. *Il est possible de modifier l'algorithme précédent en supprimant le tableau* `next_edge` *et en stockant les chaînages dans* `arete[i][0]`*, mais à la fin, il faut faire une boucle de plus sur les sommets pour reconstruire* `arete[.][0]`*.*

Exercice 16. *Construire le tableau* `adj` *d'entier de taille* `3 nt` *qui donne pour l'arête* `p=i+3k` *(arête* `i` *du triangle* `k`*) , qui donne l'arête correspondante* `p'=i'+3k'` *.*

- *si cette arete est interne alors* `adj[i+3k]=i'+3k'` *où est l'arete* `i'` *du triangle* `k'`*, remarquons :* `i'=adj[i+3k]%3, k'=adj[i+3k]/3.`
- *sinon* `adj[i+3k]=-1`*.*

5.7 Construction des triangles contenant un sommet donné

La structure de données classique d'un maillage permet de connaître directement tous les sommets d'un triangle. En revanche, déterminer tous les triangles contenant un sommet n'est pas immédiat. Nous allons pour cela proposer un algorithme qui exploite à nouveau la notion de liste chaînée.

Rappelons que si `Th` est une instance de la class `Mesh` (voir 5.6), `i=Th(k,j)` est le numéro global du sommet `j`$\in [0,3[$ de l'élément `k`. L'application F qu'on va considérer associe à un couple (k,j) la valeur `i=Th(k,j)`. Ainsi, l'ensemble des numéros des triangles contenant un sommet `i` sera donné par les premières composantes des antécédents de `i`.

On va utiliser à nouveau l'algorithme 3, mais il y a une petite difficulté par rapport à la section précédente : les éléments du domaine de définition de F sont des *couples* et non plus simplement des entiers. Pour résoudre ce problème, remarquons qu'on peut associer de manière unique au couple `(k,j)`, où $j \in [0,m[$, l'entier `p(k,j)=k*m+j` [1]. Pour retrouver le couple `(k,j)` à partir de l'entier `p`, il suffit d'écrire que `k` et `j` sont respectivement le quotient et le reste de la division euclidienne de `p` par `m`, autrement dit :

$$\mathtt{p} \longrightarrow (\mathtt{k},\mathtt{j}) = (\mathtt{k} = \mathtt{p}/\mathtt{m}, \mathtt{j} = \mathtt{p}\%\mathtt{m}). \tag{98}$$

Voici donc l'algorithme pour construire l'ensemble des triangles ayant un sommet en commun :

1. Noter au passage que c'est ainsi que `C++` traite les tableaux à double entrée : un tableau `T[n][m]` est stocké comme un tableau à simple entrée de taille `n*m` dans lequel l'élément `T[k][j]` est repéré par l'indice `p(k,j) = k*m+j`.

Algorithme 6. **Construction de l'ensemble des triangles ayant un sommet commun**

Préparation :

```
int end_list=-1,
int *head_s = new int[Th.nv];
int *next_p = new int[Th.nt*3];
int i,j,k,p;
for (i=0;i<Th.nv;i++)
  head_s[i] = end_list;
for (k=0;k<Th.nt;k++)                          //    forall triangles
  for(j=0;j<3;j++) {
    p = 3*k+j;
    i = Th(k,j);
    next_p[p]=head_s[i];
    head_s[i]= p;}
```

Utilisation : parcours de tous les triangles ayant le sommet numéro `i`

```
   for (int p=head_s[i]; p!=end_list; p=next_p[p],k=p/3,j = p %
3)
   { assert( i == Th(k,j));
                                                    //    votre code
   }
```

Exercice 17. *Optimiser le code en initialisant* `p =-1` *et en remplaçant* `p = 3*j+k` *par* `p++`.

Remarque : les sources sont disponible : `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/mesh-liste/BuildListeTrianglesVertex.cpp`.

Remarque : en utilisant la STL, il est facile de construire un programme répondant a la question. La Construction en $O(ntlog(nt))$ opération

```
vector<vector<int> > lst(Th.nv);
for (int k=0;k<Th.nt;++k)
  for(int j=0;j<3:++j)
    lst[Th(k,j)].push_back(k);
```

Utilisation pour parcours optimal de tous les triangles ayant le sommet numéro `i`

```
for (int l=0;l<lst[i].size();++l)
  {
   int k= lst[i][l];
   assert( Th(k,0) == i || Th(k,1) == i || Th(k,2) == i );
   ...
  }
```

5.8 Construction de la structure d'une matrice morse

Il est bien connu que la méthode des éléments finis conduit à des systèmes linéaires associés à des matrices très *creuses*, c'est-à-dire contenant un grand nombre de termes nuls. Dès que le

maillage est donné, on peut construire le graphe des coefficients *a priori* non nuls de la matrice. En ne stockant que ces termes, on pourra réduire au maximum l'occupation en mémoire et optimiser les produits matrices/vecteurs.

5.8.1 Description de la structure morse

La structure de données que nous allons utiliser pour décrire la matrice creuse est souvent appelée "matrice morse" (en particulier dans la bibliothèque MODULEF), dans la littérature anglo-saxonne on trouve parfois l'expression "Compressed Row Sparse matrix" (cf. SIAM book...). Notons `n` le nombre de lignes et de colonnes de la matrice, et `nbcoef` le nombre de coefficients non nuls *a priori*. Trois tableaux sont utilisés : `a[k]` qui contient la valeur du k-ième coefficient non nul avec $k \in [0, nbcoef[$, `ligne[i]` qui contient l'indice dans `a` du premier terme de la ligne `i+1` de la matrice avec $i \in [-1, n[$ et enfin `colonne[k]` qui contient l'indice de la colonne du coefficient `k`$\in$`[0:nbcoef[`. On va de plus supposer ici que la matrice est symétrique, on ne stockera donc que sa partie triangulaire inférieure. En résumé, on a :

$$\mathtt{a[k]} = a_{\mathtt{ij}} \quad \text{pour } \mathtt{k} \in [\mathtt{ligne[i-1]}+1, \mathtt{ligne[i]}] \quad \text{et } \mathtt{j} = \mathtt{colonne[k]} \quad \text{si } \mathtt{i} \leq \mathtt{j}$$

et s'il n'existe pas de k pour un couple `(i,j)` ou si `i` $\leq$ `j` alors $a_{\mathtt{ij}} = 0$.
La classe décrivant une telle structure est :

```
class MatriceMorseSymetrique {
 int n,nbcoef;  //    dimension de la matrice et nombre de coefficients non nuls
 int *ligne,* colonne;
 double *a;
 MatriceMorseSymetrique(Maillage & Th);                    //    constructeur
 double*  pij(int i,int j) const;     //    retourne le pointeur sur le coef i,j
                                          //    de la matrice si il existe
}
```

Exemple : on considère la partie triangulaire inférieure de la matrice d'ordre 10 suivante (les valeurs sont les rangs dans le stockage et non les coefficients de la matrice) :

$$\begin{pmatrix}
0 & . & . & . & . & . & . & . & . & . \\
. & 1 & . & . & . & . & . & . & . & . \\
. & 2 & 3 & . & . & . & . & . & . & . \\
. & 4 & 5 & 6 & . & . & . & . & . & . \\
. & . & 7 & . & 8 & . & . & . & . & . \\
. & . & . & 9 & 10 & 11 & . & . & . & . \\
. & . & 12 & . & 13 & . & 14 & . & . & . \\
. & . & . & . & . & 15 & 16 & 17 & . & . \\
. & . & . & . & 18 & . & . & . & 19 & . \\
. & . & . & . & . & . & . & . & . & 20
\end{pmatrix}$$

On numérote les lignes et les colonnes de `[0..9]`. On a alors :

```
n=10,nbcoef=20,
ligne[-1:9] = {-1,0,1,3,6,8,11,14,17,19,20};
colonne[21] ={0, 1,  1,2,  1,2,3,  2,4,  3,4,5,  2,4,6,
              5,6,7,  4,8,  9};
a[21]                          //    ... valeurs des 21 coefficients de la matrice
```

5.8.2 Construction de la structure morse par coloriage

Nous allons maintenant construire la structure morse d'une matrice symétrique à partir de la donnée d'un maillage d'éléments finis P_1.
Les coefficient a_{ij} non nulles de la matrice sont tels qu'il existe un triangle K contenant les sommets i et j.
Donc, pour construire la ligne i de la matrice, il faut trouver tous les sommets j tels que i, j appartiennent à un même triangle. Ainsi, pour un noeud donné i, il s'agit de lister les sommets appartenant aux triangles contenant i. Le premier ingrédient de la méthode sera donc d'utiliser l'algorithme 6 pour parcourir l'ensemble des triangles contenant i. Mais il reste une difficulté : il faut éviter les doublons. Nous allons pour cela utiliser une autre technique classique de programmation qui consiste à « colorier »les coefficients déjà répertoriés : pour chaque sommet i (boucle externe), nous effectuons une boucle interne sur les triangles contenant i puis une balayage des sommets j de ces triangles en les coloriant pour éviter de compter plusieurs fois les coefficients a_{ij} correspondant. Si on n'utilise qu'une couleur, nous devons remettre la couleur du initial les sommets avant de passer à un autre i. Pour éviter cela, nous allons utiliser plusieurs couleurs, et on changera simplement de couleur de marquage à chaque fois qu'on changera de sommet i dans la boucle externe car toutes couleurs utilisées ont un numéro strictement inférieur.

Algorithme 7. **Construction de la structure d'une matrice morse**

```
MatriceMorseSymetrique::MatriceMorseSymetrique(const Mesh & Th){
  int i,j,jt,k,p,t;
  n = Th.nv;                                    //    nombre de ligne
  int color=0, * mark;
  mark = new int [n];    //    pour stocker la couleur d'un sommet
                         //    initialisation du tableau de couleur
  for(j=0;j<Th.nv;j++)  mark[j]=color;
  color++;                                     //    on change la couleur
     //   construction optimisee de l'image réciproque: i=Th(k,j)
  int end_list=-1,*head_s,*next_t;
  head_s = new int [Th.nv];
  next_p = new int [Th.nt*3];
  int i,j,k,p=0;
  for (i=0;i<Th.nv;i++)
    head_s[i] = end_list;
  for (k=0;k<Th.nt;k++)
   for(j=0;j<3;j++)
     { next_p[p]=head_s[i=Th(k,j)]; head_s[i]=p++;}
//    1) calcul du nombre de coefficients non nuls
//    a priori de la matrice pour pourvoir faire les allocations
  nbcoef = 0;
  for(i=0; i<n; i++,color++,nbcoef++)
    for (p=head_s[i],t=p/3; p!=end_list; t=(p=next_p[p])/3)
       for(jt=0; jt< 3; jt++ )
        if(i <= (j=Th(t,jt)) && mark[j] != color))  //   nouveau j
        { mark[j]=color; nbcoef++;}     //    => marquage + ajout
//    2) allocations mémoires
  ligne = new int [n+1];
  ligne++;                     //    car le tableau commence en -1;
  colonne = new int [ nbcoef];
  a = new double [nbcoef];
//    3) constructions des deux tableaux ligne et colonne
  ligne[-1] = -1;
  nbcoef =0;
  for(i=0; i<n; ligne[i++]=nbcoef, color++)
    for (p=head_s[i],t=p/3; p!=end_list; t=(p=next_p[p])/3)
       for(jt=0; jt< 3; jt++ )
        if ( i <= (j=Th(t,jt)))  && mark[j] != color))
                     //    nouveau coefficient => marquage + ajout
         mark[j]=color, colonne[nbcoef++]=j;
//    4) tri des lignes par index de colonne
  for(i=0; i<n; i++)
    HeapSort(colonne + ligne[i-1] + 1 ,ligne[i] - ligne[i-1]);
//    nettoyage memoire
  delete  [] head_s;
  delete  [] next_p;
  delete  [] mark;
}
```

Au passage, nous avons utilisé la fonction `HeapSort` qui implémente un petit algorithme de tri, présenté dans [Knuth-1975], qui a la propriété d'être toujours en $nlog_2n$ (*cf.* 5.5 page 114). Noter que l'étape de tri n'est pas absolument nécessaire, mais le fait d'avoir des lignes triées par indice de colonne permet d'optimiser l'accés à un coefficient de la matrice dans la structure

creuse, en utilisant une recherche dichotomique comme suit :

```
inline double*  MatriceMorseSymetrique::pij(int i,int j) const
 {
   assert(i<this->n && j< this->n);
   if (j > i ) {  int k=i; i=j;  j=k;}                    //    echange i et j
   int i0= ligne[i];
   int i1= ligne[i+1]-1;
   while (i0<=i1)                                              //    dichotomie
    {
      int im=(i0+i1)/2;
      if (j< colonne[im]) i1=im-1;
      else if (j> colonne[im]) i0=im+1;
      else return a+im;
    }
   return 0;                                              //    le coef n'existe pas
 }
```

Remarque : Si vous avez tout compris dans ces algorithmes, vous pouvez vous attaquer à la plupart des problèmes de programmation.

5.8.3 Le constructeur de la classe `SparseMatrix`

En modifiant légèrement l'algorithme précédent on obtient le constructeur de la classe `SparseMatrix` suivant :

```
template<class R>
template<class Mesh>
SparseMatrix<R>::SparseMatrix(Mesh & Th)
  :
  VirtualMatrice<R>(Th.nv),n(Th.nv),m(Th.nv), nbcoef(0),
  i(0),j(0),a(0)
  {
    const int nve = Mesh::Element::nv;
    int end=-1;
    int nn= Th.nt*nve;
    int mm= Th.nv;
    KN<int> head(mm);
    KN<int> next(nn);
    KN<int> mark(mm);
    int color=0;
    mark=color;

    head = end;
    for (int p=0;p<nn;++p)
      {
        int s= Th(p/nve,p%nve);
        next[p]=head[s];
        head[s]=p;
      }
    nbcoef =0;
    n=mm;
    m=mm;
    int kk=0;
```

```
  for (int step=0;step<2;++step)                                    //    2 etapes
                                          //    une pour le calcul du nombre de coef
                                                //    l'autre pour la construction
    {
      for (int ii=0;ii<mm;++ii)
        {
          color++;
          int ki=nbcoef;
          for (int p=head[ii];p!=end;p=next[p])
            {
              int k=p/nve;
              for (int l=0;l<nve;++l)
                {
                  int jj= Th(k,l);
                  if( mark[jj] != color)   //    un nouveau sommet de l'ensemble
                    if (step==1)
                      {
                        i[nbcoef]=ii;
                        j[nbcoef]=jj;
                        a[nbcoef++]=R();
                      }
                    else
                      nbcoef++;
                  mark[jj]=color;                 //    on colorie le sommet j;
                }
            }
          int ki1=nbcoef;
          if(step==1)
            HeapSort(j+ki,ki1-ki);
        }

      if(step==0)
        {
          cout << " Allocation des tableaux  " << nbcoef << endl;
          i= new int[nbcoef];
          j= new int[nbcoef];
          a= new R[nbcoef];
          kk=nbcoef;
          nbcoef=0;
        }


    }
}
```

6 Utilisation de la STL

6.1 Introduction

La STL un moyen pour unifier utilisation des différents type de stockage informatiques que sont les, tableau, liste, arbre binaire, etc....
Ces moyens de stockage sont appelés des conteneurs. A chaque type de conteneur est associé un `iterator` et `const_iterator` , qui une formalisation des pointeurs de parcours associés au conteneur.
Si T est un type conteneur de la STL , alors pour parcourir sans modification les éléments du conteneur l de type T, il suffit d'écrire

```
for (typename T::const_iterator i=l.begin(); i != l.end(); ++i)
  {
   *i;                          //    sera la valeur associe a vos données stoker.
cout << * i << endl;       //    imprime une valeur par ligne de votre conteneur.
  }
```

Pour parcourir avec modification possible les éléments du conteneur l de type T, il suffit d'écrire

```
for (typename T::iterator i=l.begin(); i != l.end(); ++i)
  {
   *i;                          //    sera la valeur associe a vos données stoker.
cout << * i << endl;       //    imprime une valeur par ligne de votre conteneur.
  }
```

Les conteneurs utiles pour stoker un jeu de valeur de type `V`.

vector Pour stocker, un tableau `v` de valeurs `{vector<V> v;`, et la ligne d'include associé est `#include <vector>`.

list Pour stoker une liste doublement chainées de valeur, `list<V> l;`, et la ligne d'include associé est `#include <list>`.

map Pour stoker des pairs de (clef de type `K`, valeur) automatiquement trié par rapport aux clefs, avec unicité des clefs dans le conteneurs. Attentions ici les valeurs stokes seront donc des `pair<K,V>`.

set Pour stoker un ensemble de valeurs ordonnées, c'est un `map` sans valeur.

6.2 exemple

Le directory des sources : `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/stl2`

```
                //    voila un exemple assez complet des possibilites de la STL
                 //    ---------------------------------------------------------
#include <list>
#include <vector>
#include <map>
#include <set>
```

```
#include <queue>
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include <algorithm>
#include <iterator>
#include <complex>

using namespace std;


                                                    //    pour afficher une pair
template<typename A,typename B>
ostream & operator<<(ostream & f, pair<A,B> ab)
{
    f << ab.first << " : " <<  ab.second;
    return f;
}

//    ------------------------------------//
//    pour afficher un container de la stl //
//    ------------------------------------//
template<typename T>
void  show(const char * s,const T & l,const char * separateur="\n")
{
  cout << s <<  separateur;
  for (typename T::const_iterator i=l.begin(); i != l.end(); ++i)
    cout << '\t' << * i <<  separateur;
  cout << endl;
}

//    ------------------------------------------------------ //
//    pour afficher un container de la stl dans l'autre sens //
//    ------------------------------------------------------ //
template<typename T>
void  showr(const char * s,const T & l,const char * separateur="\n")
{
  cout << s <<  separateur;
  for (typename T::const_reverse_iterator i=l.rbegin(); i != l.rend(); ++i)
    cout << '\t' << * i << * separateur;
  cout << endl;
}
//    ---------------------------------------
//   / ordre lexicographie sur les pair est deja dans la stl
//    ------------------------------------


int main() {
  //    ------------------------------------
  //    les vecteurs sont des tableaux ----
  //   ------------------------------------
  {
    vector<double> v;                                   //    un vector vide
    v.push_back(1.);
    v.push_back(-1.);
    v.push_back(-5.);
```

```
    v.push_back(4.);

                                          //    v.sort(); // n'est pas implemente
    v[1]++;
    vector<double>::iterator k=v.begin()+3;
    v.erase(v.begin()+3);
           //    v.erase(find(v.begin(),v.end(),-5.)); // ok retire la valeur -5
    show("vector      :",v,"  ");
    showr("vector (reverse)       :",v,"  ");
    sort(v.begin(),v.end());                                    //    trie le vector
    show("vector trie :",v,"  ");
    vector<double> v10(10);                        //    un vecteur de 10 elements
    v10[1]= 3;
      //    remarque dans la STL, le programe suivant n'est pas en o(n^2) operation
    int n = 1000000;
    for (int i=0;i<n;++i)
      v.push_back(i);
     //    mais en logno(n) car la STL reserve de la place en plus dans le tableau
                                           //    de l'ordre d'une fraction de n
}



//    ----------------------------------------------------
//    ----- Test des listes doublement chainees----------
//   ----------------------------------------------------
{
    list<double> l;
    l.push_back(1.);
    l.push_back(5.);
    l.push_back(3.);
    l.push_back(6.);
    l.push_back(3.);
    list<double>::iterator l5=find(l.begin(),l.end(),5.);
    cout << " taille de la liste : o(n) operation " << l.size() << endl;
    cout << " liste vide? " << l.empty() << endl;
                                                                  //    remarque:
                          //    list< double>::iterator l3=l.begin()+3; interdit
                     //    car un iterator de list n'est pas aleatoire (random)

    assert( l5 != l.end());
    l.insert(l5,1000.);                                 //    insert 1000. avant l5
    show("list    :",l,"  ");
    list<double> ll(l);                                  //    copie la liste
    show("list ll : ",l,"  ");
    l.splice(l5,ll);  //    deplace la list ll dans l avant l5 (sans copie =>vide
ll)
    cout << " ll.size = " << ll.size() << endl;
    assert( ll.empty());                                       //    la liste est vide
                                                             //    remarque utile:
                    //    empty pour savoir si un liste de vide O(1) operations
                                  //    car size() est en O(size()) operations.
    ll.push_front(-3.);                          //    ajoute un element a la liste ll
    l.insert(l5,ll.begin(),ll.end());                 //    insert en copiant avant
    l.erase(l5);                               //    supprime l'element 5 de la liste
             //    remarque l5 est valide tant que l'element n'est pas supprimer
    {
```

```
                                                                        //
    const  list<double> & ll(l);    //    un autre list ll (bloque different)
                            //    recherche de la valeur 5 dans l'autre liste ll
    list<double>::const_iterator l5=find(ll.begin(),ll.end(),5.);
  }


  cout << " impression with copy : ";
  copy(l.begin(),l.end(),ostream_iterator<double>(cout, " "));
  cout << endl;

                        //    sort(l.begin(),l.end());// Erreur car un iterator
                                                //   de liste n'est pas ramdon
  l.sort();                                     //    trie la liste en n log(n)
  show("list trie : ",l,"  ");
  l.clear();                                                //    vide la liste
}


//    ---------------------------------- //
//    exemple d'utilisation des map //
//    ---------------------------------- //

{
  map<string,string> m;
  m["hecht"]="+33 1 44 27 44 11";
  m["toto"]="inconnue";
  m["aaaa"]="inconnueaaa";

  map<string,string>::iterator  ff=m.find("toto");
  if (ff!= m.end())
    cout << " find " <<  *ff<< endl;
  else
    cout << " pas trouver " << endl;
  {
    pair<map<string,string>::iterator,bool>
        pbi=m.insert(make_pair<string,string>("toto","inconnuetoto"));
    cout << "insert map : " << *pbi.first << " " << pbi.second << endl;
  }
  {
    pair<map<string,string>::iterator,bool>
       pbi=m.insert(make_pair<string,string>("toto1","inconnuetoto"));
    cout << "insert map : " << *pbi.first << " " << pbi.second << endl;
  }
  show("map",m);
  cout << " -------------------------- \n";
}

//    -----------------
//    un ensemble .
//    ------------------
{
                          //    ------------------------------------------------
                                      //    un petit exemple d'ensemble d'entier
            //    les elements doivent etre comparable ( operateur "<" existe)
  set<int>   S;
  S.insert(1);
```

```
  S.insert(2);
  S.insert(2);
                        //    le test apartenence est i ∈ S (S.find(i) != S.end())
  show("set S :",  S, "  ");
  for (int i=0;i<5;i++)
    if ( S.find(i) != S.end())
      cout << i << " est dans l'ensemble S\n";
    else
      cout << i << " n'est pas dans l'ensemble S\n";
  cout << " -------------------------- \n";
}


//   ------------------------------------------------
//    ici modelisation d'un ensemble de arete defini
//    par les numeros d'extermités (first et second)
//    de plus a chaque arete on associe un entier
//    ------------------------------------------------
{
  map<complex<double>,int >  cc;                              //    pas de bug ici
  complex<double> cca=1;
              //    cc[cca]=2; // bug car pas de comparasion < sur des complex
}



                                                                                //    /*
                          //    Attention dans une map tout depend de l'ordre.
      //    Voilà une exemple qui ne fait pas ce qui est attendue generalement
                                                                                //    */
{
  map<const char *,const char *>  dicofaux;       //    ICI L'ORDRE UTILISE EST
                                                            //    l'adresse memoire.
  const char * b="b";
  const char * c="c";
  char  a[2];
  a[0]='a'; a[1]=0;
  const char * d="d";
  dicofaux[a]="1";
  dicofaux[c]="4";
  dicofaux[d]="3";
  dicofaux[b]="2";
  dicofaux["a"]="22";
  show(" dico faux: ",dicofaux,"\n");
}



//    ------------------------------------------------
//    une exemple d'ensemble d'arete (pair d'entier)
//    numerote.
//    ------------------------------------------------
{
  map<pair<int,int>,int > edges;
  int nbe =0, nbedup=0;
  for (int i=0;i<10;i++)
    {
```

```
        pair<int,int> e(i%5,(i+1)%5);
                                    //    insertion de l'arete e avec la valeur nbe;

                          //    macro generation pour le choisir la méthode (bofbof)
#if 1
                                                                  //    methode optimiser
                        //    remarque insert retourne une pair de (iterator, bool)
                                                              //    ( true => n'existe pas)
        if( edges.insert(make_pair(e,i)).second)
          nbe++;                                                  //    ici nouvelle item
        else
          nbedup++;                                          //    ici item existe deja
#else
                                  //    Autre methode plus simple mais moins efficace
        if( edges.find(e) == edges.end() )
          {
            nbe++;                                            //    ici nouvelle item
            edges[e]=i;
          }
        else
          nbedup++;                                          //    ici item existe deja
                                //    remarque edges[e] peut etre un nouvel élément dans
             //    dans ce cas, il est initialisé avec T() qui est la valeur par
defaut
                    //    de toute classe / type T, ici on a T == int et int() == 0
#endif
      }
    cout << " nbe = " << nbe <<  " nbedup = " << nbedup << endl;
                     //    du code pour voir les items qui sont ou non dans la map
    for (int i=0;i<10;i++)
        {
          pair<int,int> e(i,(i+1));
          if (edges.find(e)!=edges.end() )
            cout << "     trouver   arete  (" << e << ")  \n";
          else
            cout << " non trouver   arete  (" << e << ") \n";
        }
    show(" les aretes ", edges);
  }

                                                  //    ---------------------------- //
                                                   //    exemple de queue prioritaire //
                                                  //    ---------------------------- //
  {

    priority_queue<int> pq;
                                                                  //    pour ajoute des valeurs
    pq.push(10);
    pq.push(1);
      pq.push(5);
                                                             //    pour voir la haut de la queue
      while(pq.empty())                                           //    la queue est telle vide
        {
            int t=pq.top();
                                                //    pour supprimer le haut de la queue
            pq.pop();
```

```
            if (t== 5) pq.push(6);                  //    je mets 6 dans la queue
            cout << t << endl;

        }
                    //    il est impossible de parcourir les elements d'une queue
                                                              //    ils sont caches
  }

    return  0;
}
```

6.3 Chaîne de caractères

A FAIRE

6.4 Entrée Sortie en mémoire

Voilà, un dernier truc, il est souvent bien utile de générer de nom de fichier qui dépend de valeur de variable, afin par exemple de numeroter les fichier. Les classes `istringstream` et `ostringstream` permet de faire de entrée sortir en mémoire permette de résoudre se type de problème.

```
#include <sstream>
#include <iostream>
using namespace std;
int main(int argc,char ** argv)
{
  string str;
  char * chaine = "1.2+ dqsdqsd q ";
                                                  //    pour construire la chaine str
  for (char * c=chaine; *c; ++c)
    str+= *c;                                                          //    ajoute

  cout << str << endl;
  istringstream f(str);              //    pour lire dans une chaine de caractere
  double a;
  char c;
  f >> a;
  c=f.get();
  cout << " a= " <<a << " " << c <<  endl;

          //    util pour generer des noms de fichier qui dependent de variables
  ostreamstream ff;                 //    pour ecrire dans une chaine de caracteres
  ff << "toto"<< i << ".txt" << ends;
  cout << " la string associer : " << ff.str() << endl << endl;
  cout << " la C chaine (char *) " << ff.str().c_str() << endl;

  return 0;
}
```

7 Différentiation automatique

Les dérivées d'une fonction décrite par son implémentation numérique peuvent être calculées automatiquement et exactement par ordinateur, en utilisant la différentiation automatique (DA). La fonctionnalité de DA est très utile dans les programmes qui calculent la sensibilité par rapport aux paramètres et, en particulier, dans les programmes d'optimisation et de design.

7.1 Le mode direct

L'idée de la méthode directe est de différencier chaque ligne du code qui définit la fonction. Les différentes méthodes de DA se distinguent essentiellement par l'implémentation de ce principe de base (cf. [**?**]).
L'exemple suivant illustre la mise en œuvre de cette idée simple.

Exemple : Considérons la fonction $J(u)$, donnée par l'expression analytique suivante :

$$J(u) \quad \text{avec} \quad \begin{array}{l} x = u - 1/u \\ y = x + \log(u) \\ J = x + y. \end{array}$$

Nous nous proposons de calculer automatiquement sa dérivée $J'(u)$ par rapport à la variable u, au point $u = 2.3$.

Méthode : Dans le programme de calcul de $J(u)$, chaque ligne de code sera précédée par son expression différentiée (avant et non après à cause des instructions du type $x = 2 * x + 1$) :

$$\begin{array}{l} dx = du + du/(u * u) \\ \mathbf{x = u - 1/u} \\ dy = dx + du/u \\ \mathbf{y = x + \log(u)} \\ dJ = dx + dy \\ \mathbf{J = x + y} \end{array}$$

Ainsi avons nous associé à toute variable (par exemple x) une variable supplémentaire, sa différentielle (dx). La différentielle devient la dérivée seulement une fois qu'on a spécifié la variable de dérivation. La dérivée (dx/du) est obtenue en initialisant toutes les différentielles à zéro en début de programme sauf la différentielle de la variable de dérivation (ex du) que l'on initialise à 1.
La valeur de la dérivée $J'(u)|_{u=2.3}$ est donc obtenue en exécutant le programme ci-dessus avec les valeurs initiales suivantes : $u = 2.3, du = 1$ et $dx = dy = 0$.

Remarque 10. *Les langages C,C++ , FORTRAN... ont la notion de* constante. *Donc si l'on sait que, par exemple, $a = 2$ dans tous le programme et que a ne changera pas, on n'est pas obligé de lui associer une différentielle. Par exemple, la fonction C*

```
float mul(const int a, float u)
{  float x; x=a*u; return x;}
```

se dérive comme suit :

```
float dmul(const int a, float u, float du)
{  float x,dx; dx = a*du; x=a*u; return dx;}
```

Les structures de contrôle (boucles et tests) présentes dans le code de définition de la fonction seront traitées de manière similaire. En effet, une instruction de test de type *if* où a est prédéfini,

```
y = a;
if ( u>0) x = u;
  else    x = 2*u;
J=x+y;
```

peut être vue comme deux programmes distincts :
• le premier calcule

```
y=a; x=u; J=x+y;
```

et avec la DA, il doit retourner

```
dy=0; y=a; dx=du; x=u; dJ=dx+dy; J=x+y;
```

• le deuxième calcule

```
y=a; x=2*u; J=x+y;
```

et avec la DA, il doit retourner

```
dy=0; y=a; dx=2*du; x=2*u; dJ=dx+dy; J=x+y;
```

Les deux programmes sont réunis naturellement sous la forme d'un unique programme

```
dy=0; y=a;
if (u>0) {dx=du; x=u;}
    else {dx=2*du; x=2*u;}
dJ=dx+dy; J=x+y;
```

Le même traitement est appliqué à une structure de type boucle :

```
x=0;
for( int i=1; i<= 3; i++) x=x+i/u;
cout << x << endl;
```

qui, en fait, calcule

```
x=0; x=x+1/u; x=x+2/u; x=x+3/u; cout << x << endl;
```

Pour la DA, il va falloir calculer

```
dx=0;x=0;
dx=dx-du/(u*u); x=x+1/u;
dx=dx-2*du/(u*u); x=x+2/u;
dx=dx-3*du/(u*u); x=x+3/u;
cout << x <<'\t'<< dx << endl;
```

ce qui est réalisé simplement par l'instruction :

```
dx=0;x=0;
for( int i=1; i<= 3; i++)
    { dx=dx-i*du/(u*u); x=x+i/u;}
cout << x <<'\t'<< dx << endl;
```

Limitations :
• Si dans les exemples précédents la variable booléene qui sert de test dans l'instruction `if` et/ou les limites de variation du compteur de la boucle `for` dépendent de u, l'implémentation décrite plus haut n'est plus adaptée. Il faut remarquer que dans ces cas, la fonction définie par ce type de programme n'est plus différentiable par rapport à la variable u, mais est différentiable à droite et à gauche et les dérivées calculées comme ci-dessus sont justes.

Exercice 18. ‖ Ecrire le programme qui dérive une boucle `while`.

• Il existe des fonctions non-différentiables pour des valeurs particulières de la variable (par exemple, $\sqrt{u}$ pour $u = 0$). Dans ce cas, toute tentative de différentiation automatique pour ces valeurs conduit à des erreurs d'exécution du type *overflow* ou *NaN* (not a number).

7.2 Fonctions de plusieurs variables

La méthode de DA reste essentiellement la même quand la fonction dépend de plusieurs variables. Considérons l'application $(u_1, u_2) \to J(u_1, u_2)$ définie par le programme suivant :

$$y_1 = l_1(u_1, u_2) \quad y_2 = l_2(u_1, u_2, y_1) \quad J = l_3(u_1, u_2, y_1, y_2) \tag{99}$$

En utilisant la méthode de DA décrite précédemment, nous obtenons :

$$\begin{aligned}
&dy_1 = \partial_{u_1} l_1(u_1, u_2)dx_1 + \partial_{u_2} l_1(u_1, u_2)dx_2\\
&\mathbf{y_1} = \mathbf{l_1}(\mathbf{u_1}, \mathbf{u_2})\\
&dy_2 = \partial_{u_1} l_2 dx_1 + \partial_{u_2} l_2 dx_2 + \partial_{y_1} l_2 dy_1\\
&\mathbf{y_2} = \mathbf{l_2}(\mathbf{u_1}, \mathbf{u_2}, \mathbf{y_1})\\
&dJ = \partial_{u_1} l_3 dx_1 + \partial_{u_2} l_2 dx_2 + \partial_{y_1} l_3 dy_1 + \partial_{y_2} l_3 dy_2\\
&\mathbf{J} = \mathbf{l_3}(\mathbf{u_1}, \mathbf{u_2}, \mathbf{y_1}, \mathbf{y_2}).
\end{aligned}$$

Pour obtenir dJ pour (u_1, u_2) donnés, il faut exécuter le programme deux fois : une première fois avec $dx_1 = 1, dx_2 = 0$, ensuite, une deuxième fois, avec $dx_1 = 0, dx_2 = 1$.
Une meilleure solution est de dupliquer les lignes $dy_i = \cdots$ et les évaluer successivement pour $dx_i = \delta_{ij}$. Le programme correspondant :

$$
\begin{aligned}
d1y_1 &= \partial_{u_1} l_1(u_1, u_2) d1x_1 + \partial_{u_2} l_1(u_1, u_2) d1x_2 \\
d2y_1 &= \partial_{u_1} l_1(u_1, u_2) d2x_1 + \partial_{u_2} l_1(u_1, u_2) d2x_2 \\
\mathbf{y_1} &= \mathbf{l_1}(\mathbf{u_1}, \mathbf{u_2}) \\
d1y_2 &= \partial_{u_1} l_2 d1x_1 + \partial_{u_2} l_2 d1x_2 + \partial_{y_1} l_2 d1y_1 \\
d2y_2 &= \partial_{u_1} l_2 d2x_1 + \partial_{u_2} l_2 d2x_2 + \partial_{y_1} l_2 d2y_1 \\
\mathbf{y_2} &= \mathbf{l_2}(\mathbf{u_1}, \mathbf{u_2}, \mathbf{y_1}) \\
d1J &= \partial_{u_1} l_3 d1x_1 + \partial_{u_2} l_2 d1x_2 + \partial_{y_1} l_3 d1y_1 + \partial_{y_2} l_3 d1y_2 \\
d2J &= \partial_{u_1} l_3 d2x_1 + \partial_{u_2} l_2 d2x_2 + \partial_{y_1} l_3 d2y_1 + \partial_{y_2} l_3 d2y_2 \\
\mathbf{J} &= \mathbf{l_3}(\mathbf{u_1}, \mathbf{u_2}, \mathbf{y_1}, \mathbf{y_2})
\end{aligned}
$$

sera exécuté pour les valeurs initiales : $d1x_1 = 1,\ d1x2 = 0,\ d2x_1 = 0,\ d2x_2 = 1$.

7.3 Une bibliothèque de classes pour le mode direct

Il existe plusieurs implémentations de la différentiation automatique, les plus connues étant **Adol-C, ADIFOR** et **Odyssee**. Leur utilisation implique une période d'apprentissage importante ce qui les rend peu accessibles aux programmeurs débutants. C'est la raison pour laquelle nous présentons ici une implémentation utilisant le mode direct qui est très simple d'utilisation, quoique moins performante que le mode inverse.
On utilise la technique de surcharge d'opérateurs (voir chapitre **??**). Toutefois, cette technique n'est efficace pour le calcul de dérivées partielles que si le nombre de variables de dérivation est inférieur à la cinquantaine. Pour une implémentation similaire en FORTRAN 90, voir Makinen [**?**].

7.4 Principe de programmation

Considérons la même fonction

$$
J(u) \quad \text{avec} \quad \begin{aligned} x &= 2u(u+1) \\ y &= x + \sin(u) \\ J &= x * y. \end{aligned}
$$

Par rapport à la méthode décrite précédemment, nous allons remplacer chaque variable par un tableau de dimension deux, qui va stocker la valeur de la variable et la valeur de sa différentielle. Le programme modifié s'écrit :

```
float y[2],x[2],u[2];
                                                    //    dx = 2 u du + 2 du (u+1)
      x[1] = 2 * u[0] * u[1] + 2 * u[1] * (u[0] + 1);
                                                          //    x = 2 u (u+1)
```

```
x[0] = 2 * u[0] * (u[0] + 1);
y[1] = x[1] + cos(u[0])*u[1];
y[0] = x[0] + sin(u[0]);
J[1] = x[1] * y[0] + x[0] * y[1];
                                                    //     J = x * y
J[0] = x[0] * y[0];
```

L'étape suivante de l'implémentation (voir [**?**], chapitre 4) consiste à créer une classe C++ qui contient comme données membres les tableaux introduits plus haut. Les opérateurs d'algèbre linéaire classiques seront redéfinis à l'intérieur de la classe pour prendre en compte la structure particulière des données membres. Par exemple, les opérateurs d'addition et multiplication sont définis comme suit :

7.5 Implémentation comme bibliothèque C++

Tous les fichiers sont dans l'archive `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/FH/autodiff.tar.bz`.
Afin de rendre possible le calcul des dérivées partielles (N variables), l'implémentation C++ de la DA va utiliser la classe suivante :

Listing 18: *(la classe `ddouble`)*

```
  #include <iostream>
using namespace std;

struct ddouble {
 double val,dval;
 ddouble(double x,double dx): val(x),dval(dx) {}
 ddouble(double x): val(x),dval(0) {}

};

inline ostream &  operator<<(ostream & f,const  ddouble & a)
{ f << a.val << " ( d = "<< a.dval << ") "; return f;}
inline ddouble operator+(const ddouble & a,const ddouble & b)
   { return ddouble(a.val+b.val,a.dval+b.dval);}
inline ddouble operator+(const double & a,const ddouble & b)
   { return ddouble(a+b.val,b.dval);}

inline ddouble operator*(const ddouble & a,const ddouble & b)
   { return ddouble(a.val*b.val,a.dval*b.val+a.val*b.dval);}
inline ddouble operator*(const double & a,const ddouble & b)
   { return ddouble(a*b.val,a*b.dval);}
inline ddouble operator/(const ddouble & a,const ddouble & b)
   { return ddouble(a.val/b.val,(a.dval*b.val-a.val*b.dval)/(b.val*b.val));}
inline ddouble operator/(const double & a,const ddouble & b)
   { return ddouble(a/b.val,(-a*b.dval)/(b.val*b.val));}
inline ddouble operator-(const ddouble & a,const ddouble & b)
   { return ddouble(a.val-b.val,a.dval-b.dval);}

inline ddouble operator-(const ddouble & a)
   { return ddouble(-a.val,-a.dval);}
inline ddouble sin(const ddouble & a)
```

```
   { return ddouble(sin(a.val),a.dval*cos(a.val));}
inline ddouble cos(const ddouble & a)
   { return ddouble(cos(a.val),-a.dval*sin(a.val));}
inline ddouble exp(const ddouble & a)
   { return ddouble(exp(a.val),a.dval*exp(a.val));}
inline ddouble fabs(const ddouble & a)
   { return (a.val > 0)? a : -a;}
inline  bool operator<(const ddouble & a ,const ddouble & b)
   { return a.val < b.val;}
```

---

voila un petit exemple d'utilisation de ces classes

```
template<typename R> R f(R x)
{
   R y=(x*x+1.);
   return y*y;
}

int main()
{
        ddouble x(2,1);
        cout << f(2.0) << " x = 2.0,  (x*x+1)^2 " << endl;
        cout << f(ddouble(2,1)) << "2 (2x) (x*x+1)   " << endl;
        return 0;
}
```

Mais de faite l'utilisation des (templates) permet de faire une utilisation recursive.

```
#include <iostream>
using namespace std;

template<class R> struct Diff {
 R val,dval;
 Diff(R x,R dx): val(x),dval(dx) {}
 Diff(R x): val(x),dval(0) {}
};

template<class R>  ostream &  operator<<(ostream & f,const  Diff<R> & a)
{ f << a.val << " ( d = "<< a.dval << ") "; return f;}
template<class R> Diff<R> operator+(const Diff<R> & a,const Diff<R> & b)
   { return Diff<R>(a.val+b.val,a.dval+b.dval);}
template<class R> Diff<R> operator+(const R & a,const Diff<R> & b)
   { return Diff<R>(a+b.val,b.dval);}

template<class R> Diff<R> operator*(const Diff<R> & a,const Diff<R> & b)
   { return Diff<R>(a.val*b.val,a.dval*b.val+a.val*b.dval);}
template<class R> Diff<R> operator*(const R & a,const Diff<R> & b)
   { return Diff<R>(a*b.val,a*b.dval);}
template<class R> Diff<R> operator/(const Diff<R> & a,const Diff<R> & b)
   { return Diff<R>(a.val/b.val,(a.dval*b.val-a.val*b.dval)/(b.val*b.val));}
template<class R> Diff<R> operator/(const R & a,const Diff<R> & b)
   { return Diff<R>(a/b.val,(-a*b.dval)/(b.val*b.val));}
```

```
template<class R> Diff<R> operator-(const Diff<R> & a,const Diff<R> & b)
   { return Diff<R>(a.val-b.val,a.dval-b.dval);}

template<class R> Diff<R> operator-(const Diff<R> & a)
   { return Diff<R>(-a.val,-a.dval);}
template<class R>Diff<R> sin(const Diff<R> & a)
   { return Diff<R>(sin(a.val),a.dval*cos(a.val));}
template<class R> Diff<R> cos(const Diff<R> & a)
   { return Diff<R>(cos(a.val),-a.dval*sin(a.val));}
template<class R> Diff<R> exp(const Diff<R> & a)
   { return Diff<R>(exp(a.val),a.dval*exp(a.val));}
template<class R> Diff<R> fabs(const Diff<R> & a)
   { return (a.val > 0)? a : -a;}
template<class R> bool operator<(const Diff<R> & a ,const Diff<R> & b)
   { return a.val < b.val;}
template<class R> bool operator<(const Diff<R> & a ,const R & b)
   { return a.val < b;}
```

Si l'on veut calculer l'a racine d'une equation $f(x) = y$

```
template<typename R> R f(R x)
{
   return x*x;
}
template<typename R> R Newtow(R y,R x)
{
                                                    //    solve: f(x) -y = 0
                                                //    x -= (f(x)-y) / df(x);
   while (1) {
   Diff<R> fff=f(Diff<R>(x,1));
   R ff=fff.val;
   R dff=fff.dval;
   cout << ff << endl;
   x = x - (ff-y)/dff;
   if (fabs(ff-y) < 1e-10) break;
   }
   return x;
}
```

Maintenant, il est aussi possible de re-différencier automatiquement l'algorithme de Newtow. Pour cela ; il suffit d'ecrit

```
int main()
{
    typedef double R;
    cout << " -- newtow (2)  = " << Newtow(2.0,1.) << endl;
    Diff<R> y(2.,1) ,  x0(1.,0.);                                    //    donne
    Diff<R> xe(sqrt(2.), 0.5/sqrt(2.));                          //    solution exact
    cout << "\n -- x = Newton " << y << " ," << x0 << ")"<<  endl;
    Diff<R> x= Newtow(y,x0) ;
    cout << " x = " << x << " ==  " <<  xe << " = xe    "  << endl;
    return 0;
}
```

Les resultats sont

```
[guest-rocq-135177:~/work/coursCEA/autodiff] hecht% ./a.out
 -- newtow (2)  = 1
2.25
2.00694
2.00001
2
1.41421
 -- x = Newton 2 ( d = 1)  ,1 ( d = 0) )
1 ( d = 0)
2.25 ( d = 1.5)
2.00694 ( d = 1.02315)
2.00001 ( d = 1.00004)
2 ( d = 1)
 x = 1.41421 ( d = 0.353553)  ==  1.41421 ( d = 0.353553)  = xe
```

8 Interpréteur de formules

Le but de ce chapitre est de donner les outils informatiques pour manipuler des formules.

8.1 Grammaire LL(1)

Dans cette section, nous allons étudier des grammaires telles qu'il nous sera facile de faire un programme qui nous dira si un texte fait partie d'un langage. Le programme sera écrit en C++ , s'arrêtera *via* la fonction `exit(1)` du système si le texte ne fait pas partie du langage.
Dans un premier temps, nous allons prendre comme exemple la grammaire des expressions

$$\begin{array}{rcl} E & = & T \quad | \quad T+E \\ T & = & F \quad | \quad F*T \\ F & = & c \quad | \quad '('\ E\ ')' \end{array}$$

Où les symbole non-terminaux de la grammaire sont E qui est une expression, T qui est un term, F qui est un facteur. Ces trois non terminaux qui sont définis par les trois règles. Et où les trois symboles terminaux de la gramaire sont c qui représente un nombre et le symbole $'('$, resp. $')'$ qui représente le charactère (, resp.) .
L'idée est très simple : nous allons supposer que nous disposons d'un analyseur lexical qui découpe l'entrée standard (le texte) en symboles terminaux, qui lit le symbole terminal suivant si le terminal est reconnu. Les symboles terminaux sont numérotés *via* un entier, les caractères ont leur code ASCII et les nombres sont définis par le numéro `c=257` et la fin de fichier par le numéro `EOF = - 1`.
Cet analyseur lexical est modélisé par une classe `Lexical` composée de :

- `sym` le symbole courant ;
- `void Nextsym()` fonction qui lit le symbole terminal suivant qui peut être ici `'+' '*' '(' ')' c EOF` ;
- `void Match(int c)` : si le symbole courant est `c`, la fonction lit le symbole suivant, sinon elle génére une erreur ;
- `bool IFSym(int c)` retourne faux si le symbole courant n'est pas `c`, sinon elle lit le symbole suivant et retourne vrai.

Listing 19: *(la classe Lexical)*

```
class Lexical {                                                    //
  typedef double R;
  int sym;
  static const int c=257;                          //    numéro du terminal nombre
  static const int EOF=-1;                            //    numéro du terminal EOF
  istream & cin;                                        //    flot de lecture
  R valeur;                                            //    Valeur du nombre
 public:
   Lexical(istream & s) :cin(s)          //    le constructeur définit le istream
     {NextSym();}                                       //    et lit un symbole

   void Error(const char * message)
    { cerr <<  message  << endl;
      exit(1);}
```

```
   void Nextsym() {                                          //    analyse lexicale
    int s=cin.peek();
    if (s==EOF) { sym=EOF; }
    else if (isdigit(s) || s == '.')
            { sym=s; cin >> valeur; assert(cin.good());}
    else if (s=='+' || s=='*' || s=='(' || s==')'  || s==EOF)
            { sym=cin.get(); }
    else if (isspace(s))
           { sym=' '; cin.get(); }
    else
           { cerr << " caract{è}re invalide " << char(s) << " " << s << endl;
             exit(1);}
  };

  bool Match(int s)                        //    assure que le symbole courant est s
   { if(s!=sym) { cerr << " On attendait le symbole ";
                  if ( s == c) cerr << " un nombre \n" ;
                  if (s == EOF) cerr << "EOF\n";
                  else cerr <<  char(s) << endl;
                  exit(1);
                   }
    else NextSym();
    return true;}

  bool IFSym(int s)                          //     est si le symbole courant est s
    { if (s==sym) { NextSym();return true;} else return false;}

  bool Match(bool b,const char *s)                                          //
    { if (!b) { cerr << " Erreur " << s << " is false " << endl;
                exit(1);}
      return b;
    }
};
```

Une fois le problème de l'analyse lexicale résolu, nous allons nous occuper de la grammaire. À chaque non-terminal nous allons associer une fonction booléenne qui retourne vrai si l'on a trouvé le non-terminal, faux sinon. Nous utiliserons la fonction `IFSym(c)` pour tester les terminaux. De plus, nous ne voulons pas faire de retour en arrière (un symbole d'avance, le 1 de `LL(1)`) donc dans une concaténation AB si A est vrai et B est faux, il faut remettre l'état du système avant l'appel de A, ce que l'on ne sait pas faire généralement. Donc, dans ce cas, nous générons une erreur.
Il suffit de programmer chaque fonction non-terminale, en factorisant à gauche par non-terminaux, de façon à factoriser tous les termes en commun dans les opération de concaténation.
Les concaténations $E = AB$, $F = Ac$, $G = cA$ sont programmées comme suit :

```
   bool E(){
      if (  A() )
       return Match(B(),'B()');}                    //    Erreur si B() est faux
   bool F(){
      if (  A() )
       return Match(c); }                             //    Erreur si c'est faux
```

```
   bool G(){
      if (IFSym(c))
          return Match(A(),"A()");}                    //    Erreur si A() est faux
```

L'opérateur ou dans $P = C \quad | \quad c \quad | \quad '(' \quad$ donne

```
  bool  P() {
   if (C()) return true;
   else if(IFsym(c)) return true;
   else if(IFsym('(')) return true;
   else return false;}
```

l'opérateur ou dans $Q = C|c|\ '('\ |\varepsilon$ où ε la chaine vide, donne

```
  bool Q() {
   if (C()) return true;
   else if(IFsym(c)) return true;
   else if(IFsym('(')) return true;
   else return true; }                                 //    la chaîne vide rend vrai
```

La règle $E = T|T\ '+'\ E$ se réécrit comme suit $E = T('+'E|\varepsilon)$, où les parenthèses () changent la priorité entre l'opérateur ou et la concaténation. Attention, il ne faut pas confondre avec '(' et ')' qui sont les deux symboles terminaux parenthèses ouvrante et fermante,

$$\begin{array}{rcl} E & = & T|T\ '+'\ E \\ T & = & F|F\ '*'\ T \\ F & = & c|\ '('\ E\ ')' \end{array} \quad \text{en} \quad \begin{array}{rcl} F & = & c|\ '('E\ ')' \\ E & = & T(\ '+'\ E|\varepsilon) \\ F & = & c|\ '('E\ ')' \end{array}$$

Après cette modification élémentaire, il est trivial d'écrire le programme C++ suivant :

Listing 20: *(la classe Grammaire)*

```
class Grammaire :public Lexical {
Grammaire(istream & cc) : Lexical(cc) {}
bool E() { if (T())
              if ( IFSym('+') return Match(E(),"E()");
              else return true;
           else return false; }

bool T() { if(F())
              if ( IFSym('*')  return Match(T(),"T()");
              else return true;
           else return false; }

bool F() {  if( IFSym(c) return true;
            else if ( IFSym('(') { Match(c);return Match(')');}
            else return false; }`
};
```

```
int main(int argc,char ** argv) {
 Grammaire exp(cin);
 bool r=exp.E();
 exp.Match(Exp:EOF);                          //    vérification de fin de text Ô
 cout << " Bien dans le langague " << endl;
 return 0;
};
```

Malheureusement, cette technique ne marche pas dans tous les cas, il faut donc encore travailler.

Remarque 11. *La remarque fondamentale, qui donne le principe de fonctionnement et qui définit les grammaires* `LL(1)` *est : si une règle de production commence par un symbole s, alors la fonction associée retourne vrai ou génère une erreur si le symbole courant est s.*

Nous en déduisons la règle suivante :

Règle 6. Si deux expressions grammaticales commençent par un terminal commun dans une opération logique du type | (ou), alors la grammaire n'est pas `LL(1)`.

Effectivement, la première expression retourne vrai, ou génère une erreur, donc la seconde expression n'est jamais utilisée.
L'autre règle est simplement sur l'utilisation de la règle vide (ε).

Règle 7. La règle vide (ε) doit être mise en fin de chaîne de l'opérateur logique | (ou), car l'expression associée à la règle vide est toujours vraie. Pratiquement, si l'on teste les autres branches du |, il faut que cela soit la dernière régle.

Maintenant nous pouvons définir une grammaire `LL(1)`.
Une grammaire est dite `LL(1)` si le langage reconnu par le programme est le même que celui défini par la grammaire.

Exemple de grammaire non-$LL(1)$

$$\begin{array}{rcl} E & = & T|T+E \\ T & = & F|F*T \\ F & = & P|\ '('\ E\ ')' \\ P & = & c|\ '('\ c\ ','\ c\ ')' \end{array}$$

où c et respectivement $'('\ c\ ','\ c\ ')'$ représentent un `double`, respectivement `complex<double>`.

Exercice 19. Écrire une autre grammaire équivalente et qui est $LL(1)$.

Mais une grammaire sans sémantique n'est pas vraiment utile.
Voici deux versions de la même grammaire avec l'évaluation des calculs. Le principe est d'effectuer les calculs seulement dans le cas où la fonction retourne vrai. Mais attention, deux grammaires qui définissent le même langage peuvent avoir deux sémantiques différentes :

$$\begin{array}{rcl} E & = & T|T+E \\ T & = & F|F*T \\ F & = & c|(E) \end{array} \quad \text{et} \quad \begin{array}{rcl} E & = & T|T+E \\ T & = & F|F*E \\ F & = & c|(E) \end{array}$$

Effectivement, la phrase $2*1+1$ donnera $\{2*1\}+1=3$ pour la grammaire de gauche et $2*\{1+1\}=4$ pour la grammaire de droite.

8.1.1 Une calculette complete

Afin d'écrire une calculette complète qui calcule la valeur associé, Nous remarquons le problème suivant lié à la sémentique et non à la syntaxe.
La grammaire de la calculette devrait etre :

$$\begin{array}{rcl} E & = & T|T\ '+'\ E|T\ '-'\ E \\ T & = & F|F\ '*'\ T|F\ '/'\ T \\ F & = & c|(E) \end{array}$$

Mais cette grammaire faite une associativité à gauche, c'est dire que l'évaluation de $1-1+2$ est donné pas $1-(1+2)$. donc pour corrigé ce problème il suffit d'ajouter deux règles E^- et $T^/$ qui correspondent aux expressions commençant par $-$ ou aux termes commençant par $/$.

$$\begin{array}{rcl} E & = & T|T\ '+'\ E|T\ '-'\ E^- \\ T & = & F|F\ '*'\ T|F\ '/'\ T^/ \\ E^- & = & T|T\ '+'\ E^-|T\ '-'\ E \\ T^/ & = & F|F\ '*'\ T^/|F\ '/'\ T \\ F & = & c|(E) \end{array}$$

Cette écriture est lourde et toutes boucles se font par récurrence, mais si l'on introduit un nouvelle opérateur $*$ dans la description des grammaires qui dit expression suivante doit être là n fois ($n \in \mathbb{N}$) comme dans les expressions régulières, alors nous pouvons réécrit la grammaire comme :

$$\begin{array}{rcl} E & = & T\ *('+'\ T|'-'\ T) \\ T & = & F\ *('*'\ F|'/'\ F) \\ F & = & c|(E) \end{array}$$

Voila un exemple complète de calculette, avec la fonction cos, la constante π et trois variables x, y, r qui sont utiliser via les pointeurs `xxxx`,`yyyy`,`rrrrr`.
Le source de l'exemple suivante est disponible à : `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/Calculette.cpp`

Listing 21: *(Calculette.cpp)*

```
#include <fstream>
#include <iostream>
#include <cstdio>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <cstring>

                          //   #define NDEBUG // uncomment when debugging is over
using namespace std;

//    la grammaire des expression:
//    E = T * ( '+' T | '-' T)
//    T = F * ( '*' F | '/' F )
```

```
//    F = c | '(' E ')' | FUNC1 '(' E ')' | GVAR | '-' F | '+' F;
//    Les fonctions FUNC1 implementées: cos,


double *xxxx,*yyyy,*rrrr;
const double PI=4*atan(1.0);

class Exp {
   public:
                typedef double R;
   private:
            int sym;                              //    la valeur de symbole courant
            static const int Number=257;
            static const int FUNC1=258;
            static const int GVAR=259;
            istringstream  cin;          //    contient la chaine de l'expression

  R valeur, valeuro;                                          //    si sym == Number
  R (*f1)(R), (*f1o)(R);                                       //    si sym == FUNC1
  R *pv, *pvo;                                                  //    Si sym == GVAR

  public:

  Exp(char* f) : cin(f) { cin.seekg(0);                       //    initialisation:
                          NextSym();}                  //    lit le premier symbol
                             //    car l'analyseur a toujours un symbole d'avance

  void NextSym() {                                                //    ReadSymbole
    pvo=pv;                           //    on sauve les valeurs associer au symbol
    f1o=f1;
    valeuro=valeur;

  int c=cin.peek();
   if (isdigit(c) || c == '.') { sym=Number; cin >> valeur;}
   else if (c=='+' || c=='*' || c=='(' || c==')'  || c==EOF) {sym=c;cin.get();}
   else if (c=='-' || c=='/' || c=='^' ) {sym=c;cin.get();}
   else if (isspace(c)) {sym=' '; cin.get();}
   else if (c==EOF ) sym=EOF;
   else if (isalpha(c)) {
     string buf;
     buf += cin.get();
     while (isalnum(cin.peek()))
       buf += cin.get();
     if(buf=="cos") { sym=FUNC1; f1=cos;}
     else if (buf=="x") { sym=GVAR; pv=xxxx;}
     else if (buf=="y") { sym=GVAR; pv=yyyy;}
     else if (buf=="r") { sym=GVAR; pv=rrrr;}
     else if (buf=="pi") { sym=Number; valeur=PI;}
     else { cerr << " mot inconnu '" << buf <<"'" << endl; exit(1);}
       cout << "Id " << buf << " " << sym << endl;}
   else {  cerr << "caractere invalide " << char(c) << " " << c << endl; exit(1);}
  }

 //    ---------------------------------------------------------------------
  void Match(int c) {                          //    le symbole courant doit etre c
       if(c!=sym) { cerr << " On attendait le symbole ";
       if ( c == Number) cerr << " un nombre " ;
```

```
      else cerr << "'"<<char(c)<<"'"; exit(1);}
    else NextSym();
  }

  //    ---------------------------------------------------------------------

  bool IFSym(int c)              //    retourne vraie si le symbole courant est c
    { if (c==sym) { NextSym();return true;}
      else return false;}
 //    ---------------------------------------------------------------------

                       //    les fonctions boolean de la grammaire qui retournent
                            //    la valeur de l'expression dans le paramètre v
bool F(R & v) {
  if(IFSym(Number)) {v=valeuro;    cout << " F " << v <<endl; return true;}
  if(IFSym('-')) {
      if (!F(v) ) { cerr << " -F On attendait un facteur " << endl;exit(1);}
        v = -v;
        return true;
      }
  if(IFSym('+')) {
      if (!F(v) ) { cerr << " +F On attendait un facteur " << endl;exit(1);}
        return true;
      } else if (IFSym('(')) {
          if (!E(v) ) { cerr << " (E) : On attendait un expression"<< endl;exit(1);}
          Match(')');    cout << " (E) " << v <<endl;
          return true;}
  else if (IFSym(GVAR)) {
        v = *pvo;
      return true;
    }
  else if (IFSym(FUNC1))
    {
        R (*ff)(R) = f1o;
      Match('(');
        if (!E(v) ) { cerr << " On attendait un expression " << endl;exit(1);}
      v=ff(v);
        Match(')');    cout << " E : f( exp) =  " << v <<endl;
      return true;
    }
    else return false;
  }

 //    ---------------------------------------------------------------------

bool T(R & v) {
      if (! F(v) ) return false;
      while (1)                      //    correction 1 (operateur gramamtical *)
      {
        if (IFSym('*')) { R vv;
           if (!F(vv) ) { cerr << " On attendait un facteur " << endl;exit(1);}
             v*=vv;}
        else if (IFSym('/')) { R vv;
           if (!F(vv) ) { cerr << " On attendait un facteur " << endl;exit(1);}
             v/=vv;}
        else break;
      }
```

```
       cout << " T " << v <<endl;
      return true;
}

 //    ------------------------------------------------------------------------

bool E(R & v) {
    if (!T(v) ) {  cout << " E false " << endl;  return false;}
      while(1)                        //    correction 2 (operateur gramamtical *)
        if (IFSym('+')) { R vv;
           if (!T(vv) ) { cerr << " On attendait un term " << endl;exit(1);}
           v+=vv;}
           else if (IFSym('-')) { R vv;
             if (!T(vv) ) { cerr << " On attendait un term " << endl;exit(1);}
               v-=vv;}
             else break;
     cout << " E " << v <<endl;
    return true;
}


};

 //    ------------------------------------------------------------------------

int main(int argc,char **argv)
{
   if(argc <2) { cout << " usage : "<< argv[0] << " '1+5*cos(pi) '" <<endl;
   return 1;}

  assert(argc==2);
  Exp exp(argv[1]);
  Exp::R v;         //    pour declare une variable de type real de la calculette
  exp.E(v);
  cout << argv[1]<<  "= "<< v << endl;
  assert(exp.IFSym(EOF));
  return 0;
}
```

8.2 Algèbre de fonctions

Bien souvent, nous aimerions pouvoir considérer des fonctions comme des données classiques, afin de construire une nouvelle fonction en utilisant les opérations classiques `+, -, *, /` ... Par exemple, la fonction $f(x, y) = cos(x) + sin(y)$, pour que la calculette généré une peusdo fonction que l'on pourra évaluer plusieurs fois sans réinterpréter la chaîne de caractère définissant l'expression.
Dans un premier temps, nous allons voir comment l'on peut construire et manipuler en `C++` l'algèbre des fonctions $\mathcal{C}^\infty$, définies sur $\mathbb{R}$ à valeurs dans $\mathbb{R}$.

8.2.1 Version de base

Une fonction est modélisée par une classe (`Cvirt`) qui a juste l'opérateur `() ()` virtuel.

Pour chaque type de fonction, on construira une classe qui dérivera de la classe `Cvirt`. Comme tous les opérateurs doivent être définis dans une classe, une fonction sera définie par une classe `Cfonc` qui contient un pointeur sur une fonction virtuelle `Cvirt`. Cette classe contiendra l'opérateur « fonction » d'évaluation, ainsi qu'une classe pour construire les opérateurs binaires classiques. Cette dernière classe contiendra deux pointeurs sur des `Cvirt` qui correspondent aux membres de droite et gauche de l'opérateur et la fonction de $\mathbb{R}^2$ à valeur de $\mathbb{R}$ pour définir l'opérateur.
Les programmes sources sont accessibles à l'adresse :
`http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/lg/fonctionsimple.cpp`.

Listing 22: *(fonctionsimple.cpp)*

```
 #include <iostream>
                                                    //    pour la fonction pow
#include <math.h>
 typedef double R;
 class Cvirt { public: virtual R operator()(R ) const =0;};

 class Cfonc : public Cvirt { public:
   R (*f)(R);                                      //    pointeur sur la fonction Cpp
   R operator()(R x) const { return (*f)(x);}
   Cfonc( R (*ff)(R)) : f(ff) {} };

 class Cconst : public Cvirt { public:
   R a;
   R operator()(R ) const { return a;}
   Cconst( R aa) : a(aa) {}          };

 class Coper : public Cvirt { public:
   const  Cvirt *g, *d;
   R (*op)(R,R);
   R operator()(R x) const { return (*op)((*g)(x),(*d)(x));}
   Coper( R (*opp)(R,R), const Cvirt *gg, const Cvirt *dd)
         : op(opp),g(gg),d(dd) {}
   ~Coper(){delete g,delete d;}    };

                                                    //    les cinq opérateurs binaires
 static  R Add(R a,R b) {return a+b;}
 static  R Sub(R a,R b) {return a-b;}
 static  R Mul(R a,R b) {return a*b;}
 static  R Div(R a,R b) {return a/b;}
 static  R Pow(R a,R b) {return pow(a,b);}

 class Fonction { public:
   const Cvirt *f;
   R operator()(const R x){ return (*f)(x);}
   Fonction(const Cvirt *ff) : f(ff) {}
   Fonction(R (*ff)(R)) : f(new Cfonc(ff)) {}
   Fonction(R a) : f(new Cconst(a)) {}
   operator const  Cvirt * () const {return f;}
   Fonction operator+(const Fonction & dd) const
        {return new Coper(Add,f,dd.f);}
   Fonction operator-(const Fonction & dd) const
```

```
        {return new Coper(Sub,f,dd.f);}
   Fonction operator*(const Fonction & dd) const
        {return new Coper(Mul,f,dd.f);}
   Fonction operator/(const Fonction & dd) const
        {return new Coper(Div,f,dd.f);}
   Fonction operator^(const Fonction & dd) const
        {return new Coper(Pow,f,dd.f);}
 };

using namespace std;                                  //   introduces namespace std

R Identite(R x){ return x;}
int main()
{
  Fonction Cos(cos),Sin(sin),x(Identite);
  Fonction f((Cos^2)+Sin*Sin+(x*4));          //    attention ^ n'est prioritaire
  cout << f(2) << endl;
  cout << (Cos^2)+Sin*Sin+(x*4))(1) << endl;
  return 0;
}
```

Dans cet exemple, la fonction $f = cos^2 + (sin * sin + x^4)$ sera définie par un arbre de classes qui peut être représenté par :

```
f.f= Coper (Add,t1,t2);                 // t1=(cos^2)     +  t2=(sin*sin+x^4)
    t1.f= Coper (Pow,t3,t4);            // t3=(cos)       ^  t4=(2)
    t2.f= Coper (Add,t5,t6);            // t5=(sin*sin)   +  t6=x^4
        t3.f= Ffonc (cos);
        t4.f= Fconst (2.0);
        t5.f= Coper (Mul,t7,t8);        // t7=(sin)       *  t8=(sin)
            t6.f= Coper (Pow,t9,t10);   // t9=(x)         ^  t10=(4)
                t7.f= Ffonc (sin);
                t8.f= Ffonc (sin);
                t9.f= Ffonc (Identite);
                t10.f= Fconst (4.0);
```

8.2.2 Les fonctions C^∞

Maintenant, nous voulons aussi pouvoir dériver les fonctions. Il faut faire attention, car si la classe contient la dérivée de la fonction, il va y avoir implicitement une récursivité infinie, les fonctions étant indéfiniment dérivables. Donc, pour que la classe fonctionne, il faut prévoir un processus à deux niveaux : l'un qui peut construire la fonction dérivée, et l'autre qui évalue la fonction dérivée et qui la construira si nécessaire.
Donc, la classe `CVirt` contient deux fonctions virtuelles :

```
virtual float operator () (float) = 0;
virtual CVirt *de () {return zero;}
```

La première est la fonction d'évaluation et la seconde calcule la fonction dérivée et retourne la fonction nulle par défaut. Bien entendu, nous stockons la fonction dérivée (`CVirt`) qui ne sera construite que si l'on utilise la dérivée de la fonction.
Nous introduisons aussi les fonctions de $\mathbb{R}^2$ à valeurs dans $\mathbb{R}$, qui seront modélisées dans la classe `Fonction2`.
Les programmes sources sont dans les 2 URL

- `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/lg/fonction.cpp`
- `http://www.ljll.math.upmc.fr/~hecht/ftp/InfoSci/FH/cpp/lg/fonction.hpp`

Toutes ces classe sont définies dans le fichier d'en-tête suivant :

Listing 23: *(fonction.hpp)*

```
#ifndef __FONCTION__
#define __FONCTION__

struct CVirt {
  mutable CVirt *md;                    //    le pointeur sur la fonction dérivée
  CVirt () : md (0) {}
  virtual R operator () (R) = 0;
  virtual CVirt *de () {return zero;}
  CVirt *d () {if (md == 0) md = de (); return md;}
  static CVirt *zero;
};

class Fonction {                                    //    Fonction d'une variable
  CVirt *p;
public:
  operator CVirt *() const {return p;}
  Fonction (CVirt *pp) : p(pp) {}
  Fonction (R (*) (R));                    //    Création à partir d'une fonction C
  Fonction (R x);                                   //    Fonction constante
  Fonction (const Fonction& f) : p(f.p) {}                //   Constructeur par
copie
  Fonction d () {return p->d ();}                                   //    Dérivée
  void setd (Fonction f) {p->md = f;}
  R operator() (R x) {return (*p)(x);}                  //    Valeur en un point
  Fonction operator() (Fonction);                  //    Composition de fonctions
  friend class Fonction2;
  Fonction2 operator() (Fonction2);
  static Fonction monome (R, int);
};

struct CVirt2 {
  CVirt2 ():  md1 (0), md2 (0) {}
  virtual R operator () (R, R) = 0;
  virtual CVirt2 *de1 () {return zero2;}
  virtual CVirt2 *de2 () {return zero2;}
  CVirt2 *md1, *md2;
  CVirt2 *d1 () {if (md1 == 0) md1 = de1 (); return md1;}
  CVirt2 *d2 () {if (md2 == 0) md2 = de2 (); return md2;}
```

```
  static CVirt2 *zero2;
};

class Fonction2 {                                    //    Fonction de deux variables
  CVirt2 *p;
public:
  operator CVirt2 *() const {return p;}
  Fonction2 (CVirt2 *pp) : p(pp) {}
  Fonction2 (R (*) (R, R));                 //    Création à partir d'une fonction C
  Fonction2 (R x);                                                //    Fonction constante
  Fonction2 (const Fonction2& f) : p(f.p) {}          //   Constructeur par copie
  Fonction2 d1 () {return p->d1 ();}
  Fonction2 d2 () {return p->d2 ();}
  void setd (Fonction2 f1, Fonction2 f2) {p->md1 = f1; p->md2 = f2;}
  R operator() (R x, R y) {return (*p)(x, y);}
  friend class Fonction;
  Fonction operator() (Fonction, Fonction);         //    Composition de fonctions
  Fonction2 operator() (Fonction2, Fonction2);
  static Fonction2 monome (R, int, int);
};

extern Fonction Chs,Identity;
extern Fonction2 Add, Sub, Mul, Div, Abscisse, Ordonnee;

inline Fonction operator+ (Fonction f, Fonction g) {return Add(f, g);}
inline Fonction operator- (Fonction f, Fonction g) {return Sub(f, g);}
inline Fonction operator* (Fonction f, Fonction g) {return Mul(f, g);}
inline Fonction operator/ (Fonction f, Fonction g) {return Div(f, g);}
inline Fonction operator- (Fonction f) {return Chs(f);}

inline Fonction2 operator+ (Fonction2 f, Fonction2 g) {return Add(f, g);}
inline Fonction2 operator- (Fonction2, Fonction2 g) {return Sub(f, g);}
inline Fonction2 operator* (Fonction2 f, Fonction2 g) {return Mul(f, g);}
inline Fonction2 operator/ (Fonction2 f, Fonction2 g) {return Div(f, g);}
inline Fonction2 operator- (Fonction2 f) {return Chs(f);}

#endif
```

Maintenant, prenons l'exemple d'une fonction `Monome` qui sera utilisée pour construire les fonctions polynômes. Pour construire effectivement ces fonctions, nous définissons la classe `CMonome` pour modéliser $x \mapsto cx^n$ et la classe `CMonome2` pour modéliser $x \mapsto cx^{n_1}y^{n_2}$.

Listing 24: *(les classes CMonome et CMonome2)*

```
class CMonome : public CVirt {
R c; int n;
public:
CMonome (R cc = 0, int k = 0) : c (cc), n (k) {}
R operator () (R x);                                                    //    return cx^n
CVirt *de () {return n? new CMonome (c * n, n - 1) : zero;}
};
Function Function::monome (R x, int k) {return new CMonome (x, k);}
```

```
class CMonome2 : public CVirt2 {
R c; int n1, n2;
public:
CMonome2 (R cc = 0, int k1 = 0, int k2 = 0) : c (cc), n1 (k1), n2 (k2) {}
R operator () (R x, R y);                                    //    return cx^n1 y^n2
CVirt2 *dex () {return n1 ? new CMonome2 (c * n1, n1 - 1, n2) : zero;}
CVirt2 *dey () {return n2 ? new CMonome2 (c * n2, n1, n2 - 1) : zero;}
};
Function2 Function2::monome (R x, int k, int l)
{return new CMonome2 (x, k, l);}
```

En utilisant exactement la même technique, nous pouvons construire les classes suivantes :

CFunc une fonction C++ de prototype `R (*)(R)` .

CComp la composition de deux fonctions f, g comme $(x) \to f(g(x))$.

CComb la composition de trois fonctions f, g, h comme $(x) \to f(g(x), h(x))$.

CFunc2 une fonction C++ de prototype `R (*)(R,R)`.

CComp2 la composition de f, g comme $(x, y) \to f(g(x, y))$.

CComb2 la composition de trois fonctions f, g, h comme $(x, y) \to f(g(x, y), h(x, y))$

Pour finir, nous indiquons juste la définition des fonctions globales usuelles :

```
CVirt *CVirt::zero = new CMonome;
CVirt2 *CVirt2::zero = new CMonome2;

Function::Function (R (*f) (R)) : p(new CFunc(f)) {}
Function::Function (R x) : p(new CMonome(x)) {}
Function Function::operator() (Function f) {return new CComp (p, f.p);}
Function2 Function::operator() (Function2 f) {return new CComp2 (p, f.p);}

Function2::Function2 (R (*f) (R, R)) : p(new CFunc2(f)) {}
Function2::Function2 (R x) : p(new CMonome2(x)) {}
Function Function2::operator() (Function f, Function g)
{return new CComb (f.p, g.p, p);}
Function2 Function2::operator() (Function2 f, Function2 g)
{return new CComb2 (f.p, g.p, p);}

static R add (R x, R y) {return x + y;}
static R sub (R x, R y) {return x - y;}

Function Log = new CFunc1(log,new CMonome1(1,-1));
Function Chs = new CMonome (-1., 1);
Function Identity = new CMonome (-1., 1);

Function2 CoordinateX = new CMonome2 (1., 1, 0);                    //    x
Function2 CoordinateY = new CMonome2 (1., 0, 1);                    //    y
Function2 One2 = new CMonome2 (1., 0, 0);                           //    1
Function2 Add = new CFunc2 (add, Function2(1.), Function2(1.));
Function2 Sub = new CFunc2 (sub, Function2(1.), Function2(-1.));
Function2 Mul = new CMonome2 (1., 1, 1);
Function2 Div = new CMonome2 (1., 1, -1);
                                          //    pow(x,y) = x^y = e^{log(x)y}
```

```
Function2 Pow = new CFunc2 (pow, CoordinateY*Pow(CoordinateX,
CoordinateY-One2)),Log(CoordinateX)*Pow);
```

Avec ces définitions, la construction des fonctions classiques devient très simple ; par exemple, pour construire les fonctions sin, cos il suffit d'écrire :

```
Function Cos(cos),Sin(sin);
Cos.setd(-Sin);                                    //    définit la dérivée de cos
Sin.setd(Cos);                                     //    définit la dérivée de sin
Function d4thCos=Cos.d().d().d().d();        //    construit la dérivée quatrième
```

8.3 Un petit langage

Pour faire un langage, il faut une machine virtuelle car l'on ne connaît pas le langage machine. Un idée simple est d'utiliser l'algèbre de fonctions comme machine virtuelle. Mais dans notre cas les fonctions seront juste des fonctions sans argument.

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <map>
using namespace std;
typedef double R;

  static R Add(R a,R b) { return a+b;}
  static R Comma(R a,R b) { return a,b;}
  static R Mul(R a,R b) { return a*b;}
  static R Div(R a,R b) { return a/b;}
  static R Sub(R a,R b) { return a-b;}
  static R Lt(R a,R b) { return a<b;}
  static R Gt(R a,R b) { return a>b;}
  static R Le(R a,R b) { return a<=b;}
  static R Ge(R a,R b) { return a>=b;}
  static R Eq(R a,R b) { return a==b;}
  static R Ne(R a,R b) { return a!=b;}

  static R Neg(R a)     { return -a;}
  static R Not(R a)     { return!a;}
static R Print(R a)     { cout << a << " ";  return a;}

class Exp { public:

  class ExpBase{ public:    virtual R operator()() const = 0;
    virtual ExpBase * set(const ExpBase *) const {return 0;}
  };

  class ExpConst : public ExpBase {public:
    R a;
    ExpConst(R aa) : a(aa) {}
    R operator()() const {return a;}};
```

```
class ExpPString : public  ExpBase {public:
  const string *p;
  ExpPString(const string *pp) :p(pp){}
  R operator()() const {cout << *p; return 0;}
};

class ExpPtr : public  ExpBase {public:

  class SetExpPtr  : public  ExpBase  {public:
    R *a;
    const ExpBase *e;
    R operator()() const {return *a=(*e)();}                              //    set
    SetExpPtr(R * aa, const ExpBase *ee) : a(aa),e(ee) {}};
  R *a;
  ExpPtr(R *aa) : a(aa) {}
  R operator()() const {return *a;}
  virtual ExpBase * set(const ExpBase * e) const
                {return new SetExpPtr(a,e);}};

class ExpOp2 : public ExpBase {public:
  typedef  R (*func)(R,R);
  const ExpBase &a,&b; func op;
  ExpOp2(func o,const ExpBase &aa,const ExpBase &bb )
         : a(aa),b(bb),op(o) {assert(&a && &b);}
  R operator()() const {R aa=a();R bb=b(); return op(aa,bb);}};

class ExpOp2_ : public ExpBase {public:
  typedef  R (*func)(const ExpBase &,const ExpBase &);
  const ExpBase &a,&b;func op;

  ExpOp2_(func o,const ExpBase &aa,const ExpBase &bb )
         : a(aa),b(bb),op(o) {assert(&a && &b);}
  R operator()() const {return op(a,b);}};


class ExpOp : public ExpBase {public:
  const ExpBase &a; R (*op)(R);
  ExpOp(R (*o)(R),const ExpBase &aa) : a(aa),op(o) {assert(&a);}
  R operator()() const {return op(a());}};


const ExpBase * f;
Exp() : f(0) {}
Exp(const ExpBase *ff) : f(ff) {}
Exp(R a)  : f(  new ExpConst(a)) {}
Exp(R* a) :f( new ExpPtr(a)) {}
R operator()() const {return (*f)();}
operator const ExpBase * () const  {return f;}
Exp operator=(const Exp & a) const { return f->set(a.f);}
Exp operator,(const Exp & a) const { return new ExpOp2(Comma,*f,*a.f);}
Exp operator+(const Exp & a) const { return new ExpOp2(Add,*f,*a.f);}
Exp operator-(const Exp & a) const { return new ExpOp2(Sub,*f,*a.f);}
Exp operator-() const              { return new ExpOp(Neg,*f);}
Exp operator!() const              { return new ExpOp(Not,*f);}
Exp operator+() const              { return *this;}
Exp operator/(const Exp & a) const { return new ExpOp2(Div,*f,*a.f);}
Exp operator*(const Exp & a) const { return new ExpOp2(Mul,*f,*a.f);}
```

```
  Exp operator<(const Exp & a) const { return new ExpOp2(Lt,*f,*a.f);}
  Exp operator>(const Exp & a) const { return new ExpOp2(Gt,*f,*a.f);}
  Exp operator<=(const Exp & a) const { return new ExpOp2(Le,*f,*a.f);}
  Exp operator>=(const Exp & a) const { return new ExpOp2(Ge,*f,*a.f);}
  Exp operator==(const Exp & a) const { return new ExpOp2(Eq,*f,*a.f);}
  Exp operator!=(const Exp & a) const { return new ExpOp2(Ne,*f,*a.f);}

  Exp comp(R (*ff)(R )){  return new ExpOp(ff,*f);}
  Exp comp(R (*ff)(R,R ),const Exp & a){  return new ExpOp2(ff,*f,*a.f);}

};

Exp comp(R (*ff)(R ),const Exp & a){  return new Exp::ExpOp(ff,*a.f);}
Exp comp(R (*ff)(R,R ),const Exp & a,const Exp & b)
                                {  return new Exp::ExpOp2(ff,*a.f,*b.f);}
Exp comp(Exp::ExpOp2_::func ff,const Exp & a,const Exp & b)
                                {  return new Exp::ExpOp2_(ff,*a.f,*b.f);}
Exp print(const  Exp & a)    { return new Exp::ExpOp(Print,*a.f);}
Exp print(const string * p)  { return new Exp::ExpPString(p);}
R While(const  Exp::ExpBase & a,const  Exp::ExpBase & b) {
  R r=0;
  while( a() ) { r=b();}
  return r;
}

                                        //    ------ utilisation de la stl --------
class TableOfIdentifier {
public:
  typedef  pair<int,void  *> value;                          //    typeid + data
  typedef map<string,value> maptype;
  typedef maptype::iterator iterator;
  typedef maptype::const_iterator const_iterator;
  maptype m;

  value Find(const string & id) const;
  void * Add(const string & id,int i=0,void *p=0)
  { m.insert(make_pair<string,value>(id,value(i,p))); return p;}
  void * Add(const string & id, R (*f)(R));
  void * Add(const string & id,const  string & value);
  const Exp::ExpBase * Add(const string & id,const Exp::ExpBase * f);

} tableId;
```

Remarque dans le code précédent, la table des symboles est juste une `map` de la STL, ou les valeurs sont des paires de `int` le type du symblos et `void *` pointeur sur la valeur du symbole, ici les types des symboles peuvent être de

–

8.3.1 La grammaire en bison ou yacc

Il y a une très belle documentation de bison donner avec la distribution de paquet `bison`, je vous conseille vivement sa lecture, il y a de nombreux exemples. Mais, je pense que le code suivante doit vous permettre de comprendre, sauf pour la priorité des opérateurs qui est gérée par les obscur commandes `%left` et `%right`.

```
%{
#include <iostream>
#include  <complex>
#include <string>
#include <cassert>
using namespace std;
#ifdef __MWERKS__
#include "alloca.h"
#endif
#include "Expr.hpp"

                                                    //    to store le code generer
 const  Exp::ExpBase  * leprogram=0;

inline void yyerror(const char * msg)
{  cerr << msg << endl;
   exit(1);}

int yylex();
%}


%union{
 double dnum;
 string * str;
 const  Exp::ExpBase * exp;
 R (*f1)(R);
 void * p;
}

%token WHILE
%token PRINT
%token GE
%token LE
%token EQ
%token NE

%token <dnum> DNUM
%token <str> NEW_ID
%token <str> STRING
%token <exp> ID
%token <f1> FUNC1
%token ENDOFFILE
%type  <exp> code
%type  <exp> parg
%type  <exp> instr
%type  <exp> expr
%type  <exp> expr1
```

```
%type  <exp> start
%type  <exp> unary_expr
%type  <exp> pow_expr
%type  <exp> primary
%left  <exp> ','
%left  <exp>  '<' '>' EQ NE LE GE
%left  <exp>  '+' '-'
%left  <exp> '*' '/'  '%'
%right <exp>  '^'

%%
start:   code  ENDOFFILE { leprogram=$1; return 0; }
;

code : instr
      | code instr { $$=(Exp($1),$2);}
;

instr:  NEW_ID '=' expr ';' { $$=(Exp(tableId.Add(*$1,Exp(new R)))=$3);
                              cout << "new id = " << (*$$)() << endl;}
     | ID '=' expr ';' { $$ = (Exp($1)=$3);
                         cout << "id = " << (*$$)() << endl;}
     | PRINT parg ';' {$$=$2;}
     | WHILE '(' expr ')' '{' code '}' { $$ = comp(While,Exp($3),Exp($6));}
     | expr ';'
;

parg:   expr { $$=print($1);}
      | STRING { $$=print($1);}
      |  parg ',' parg {$$=(Exp($1),$3);}
;

expr: expr1
    | expr '<' expr {$$ = Exp($1)<Exp($3);}
    | expr '>' expr {$$ = Exp($1)>Exp($3);}
    | expr LE expr {$$ = Exp($1)<=Exp($3);}
    | expr GE expr {$$ = Exp($1)>=Exp($3);}
    | expr EQ expr {$$ = Exp($1)==Exp($3);}
    | expr NE expr {$$ = Exp($1)!=Exp($3);}
;

expr1: unary_expr
    | expr1 '+' expr1 {$$ = Exp($1)+$3;}
    | expr1 '-' expr1 {$$ = Exp($1)-Exp($3);}
    | expr1 '*' expr1 {$$ = Exp($1)*$3;}
    | expr1 '/' expr1 {$$ = Exp($1)/$3;}
;

unary_expr:   pow_expr
  | '-'  pow_expr {$$=-Exp($2);}
  | '!'  pow_expr {$$=!Exp($2);}
  | '+'  pow_expr {$$=$2;}
;

pow_expr: primary
     |      primary  '^' unary_expr   {$$=comp(pow,$1,$3);}
;
```

```
primary:   DNUM {$$=Exp($1);}
     | '(' expr ')' { $$=$2;}
     |  ID { $$=$1;}
     |  FUNC1 '('expr ')' { $$=comp($1,$3);}
;

%%

//    mettre ici les fonctions utilisateurs
//    et généralement le programme principal.
```

8.3.2 Analyseur lexical

Théoriquement, il eut fallu utiliser `flex` pour construire l'analyseur lexical, mais le code est simple donc je l'ai réécris (une page).

```
int yylex() {                    //    remarque yylex est le nom donne par bison.
  L1:
   if (cin.eof())
     return ENDOFFILE;
   int c=cin.peek();
   if (c<0) return
     ENDOFFILE;
   if (isdigit(c)) {                              //    un nombre le C++ va le lire
       cin >> yylval.dnum; assert(cin.eof() || cin.good()); return DNUM;}
   else if (c=='.') {              //    c''est peut etre un nombre ou un point .
             cin.get(); c=cin.peek(); cin.putback('.');
             if (isdigit(c)) {                                  //    c'est un nombre
                   cin >> yylval.dnum;
                   assert(cin.eof() || cin.good());
                   return DNUM;}
             return cin.get();}                                 //    c'est un point .
   else if (c=='"')                              //    une chaine de caractères
     {  yylval.str = new string();
        cin.get();
        while (cin.peek() != '"' && cin.peek() != EOF)
          *yylval.str += char(cin.get());
         assert(cin.peek() == '"' );
         cin.get();
        return STRING;     }
   else if (isalpha(c)) {                                   //    un identicateur
      yylval.str = new string();
      do {
        *yylval.str += char(cin.get());
       } while (isalnum(cin.peek()));
      pair<int,void*> p=tableId.Find(*yylval.str);
     if (p.first==0) return NEW_ID;                                 //    n'existe pas
      else {                                 //    existe on retourne type et valeur
         delete yylval.str;  yylval.p=p.second; return p.first;} }
   else if (isspace(c))                                     //    on saute les blancs
     {cin.get(); goto L1;}
   else {                                          //    caractère spéciaux du langage
     c = cin.get();                                        //    le caractère courant
     int cc = cin.peek();                                  //    le caractère suivant
```

```
    if (c=='+' || c=='*' || c=='(' || c==')' ) return c;
    else if (c=='-' || c=='/' || c=='{' || c=='}' ) return c;
    else if (c=='^' || c==';' || c==',' || c=='.' ) return c;
    else if (cc != '=' && (c=='<' || c=='>' || c=='='  || cc=='!' ))
              return c;
    else if (c=='<' && cc=='=') { cin.get(); return LE;}
    else if (c=='>' && cc=='=') { cin.get(); return GE;}
    else if (c=='=' && cc=='=') { cin.get(); return EQ;}
    else if (c=='!' && cc=='=') { cin.get(); return NE;}
    else {                                                          //    erreur
             cerr << " caractere invalide " << char(c) << " " << c << endl;
             exit(1);}  }
}
```

le programme principale :

```
int main (int  argc, char **argv)
{
                                                          //    yydebug = 1;
  R x,y;

  x=0;
  y=0;

  tableId.Add("x",Exp(&x));
  tableId.Add("y",Exp(&y));
  tableId.Add("cos",cos);
  tableId.Add("end",ENDOFFILE);
  tableId.Add("while",WHILE);
  tableId.Add("print",PRINT);
  tableId.Add("endl","\n");
  yyparse();
  cout << " Compile OK code: " << leprogram <<endl;
  if(leprogram)
    {
      R r= ( * leprogram)();                          //    execution du code
      cout << " return " << r <<endl;
    }

  return 0;
}
```

Les fonctions utiles pour la table des symboles : recherche un symbole `Find` qui retour les deux valeur associer, type et pointeur. Et trois méthodes pour ajouter des symboles de type différent, une chaîne de caractères, une fonction `C++ : double (*)(double)` , ou une expression du langage.

```
 pair<int,void*> TableOfIdentifier::Find(const string & id) const  {
    maptype::const_iterator i=m.find(id);
    if (i == m.end()) return make_pair<int,void*>(0,0);
    else return i->second;
  }

void * TableOfIdentifier::Add(const string & id,const string & value)
```

```
  { return Add(id,STRING ,(void*) new string(value));}

void * TableOfIdentifier::Add(const string & id, R (*f)(R))
  { return Add(id,FUNC1,(void*)f);}

const Exp::ExpBase * TableOfIdentifier::Add(const string & id,
                                            const Exp::ExpBase * f)
  {  Add(id,ID,(void*)f); return f;}
```

Voilà un petit programme `example-exp.txt` testé :

```
i=1;
while(i<10)
{
  i = i+1;
  a= 2^i+cos(i*3+2);
  print " i = ",  i,a, endl;
}
i;
end
```

et la sortie :3

```
Brochet:~/work/Cours/InfoBase/l3-lg hecht$ ./exp <example-exp.txt
new id = 1
id = 2
new id = 3.8545
 Compile OK code: 0x3007b0
 i = 2 3.8545
 i = 3 8.00443
 i = 4 16.1367
 i = 5 31.7248
 i = 6 64.4081
 i = 7 127.467
 i = 8 256.647
 i = 9 511.252
 i = 10 1024.83
 return 10
```

Exercice 20.

Ajouter un teste a la grammaire : **if** `(expr) instruction` **else** `instruction` . Pour cela, il faut ajouter un nouveau type `ExpOp3_` de `Exp` en s'aidant de `ExpOp2_` dans le fichier `Expr.hpp`, Puis ajouter les `token` `if` et `else` à bison, ajouter en plus à la définition de `instr` le code correspondant :

```
  | IF '(' expr ')' instr {$$= comp2(If2,Exp($3),Exp($5));}
  | IF '(' expr ')' instr  ELSE instr
                {$$=comp3(If3,Exp($3),Exp($5),Exp($7));}
  |  '{' code '}'  { $$ = $2;}
```

Pour finir, il suffit de créer dans le `main`, les deux nouveau mots clefs (`if`,`else`)comme `while`.

Références

[S. Bourne] S. Bourne le système unix, InterEdition, Paris.

[Kernighhan,Pike] B.W. Kernighan, R. Pike L'environnement de programmation UNIX, InterEdition, Paris.

[1] [Kernighhan, Richie]B.W. Kernighhan et D.M. Richie Le Langage C, Masson, Paris.

[J. Barton, Nackman-1994] J. Barton, L. Nackman *Scientific and Engineering, C++*, Addison-Wesley, 1994.

[Ciarlet-1978] P.G. Ciarlet , *The Finite Element Method*, North Holland. n and meshing. Applications to Finite Elements, Hermès, Paris, 1978.

[Ciarlet-1982] P. G. Ciarlet *Introduction à l'analyse numérique matricielle et à l'optimisation*, Masson ,Paris,1982.

[Ciarlet-1991] P.G. Ciarlet , Basic Error Estimates for Elliptic Problems, in Handbook of Numerical Analysis, vol II, Finite Element methods (Part 1), P.G. Ciarlet and J.L. Lions Eds, North Holland, 17-352, 1991.

[2] I. Danaila, F. hecht, O. Pironneau : *Simulation numérique en C++* Dunod, 2003.

[Dupin-1999] S. Dupin *Le langage C++*, Campus Press 1999.

[Frey, George-1999] P. J. Frey, P-L George *Maillages*, Hermes, Paris, 1999.

[George,Borouchaki-1997] P.L. George et H. Borouchaki , *Triangulation de Delaunay et maillage. Applications aux éléments finis*, Hermès, Paris, 1997. Also as
P.L. George and H. Borouchaki , *Delaunay triangulation and meshing. Applications to Finite Elements*, Hermès, Paris, 1998.

[FreeFem++] F. Hecht, O. Pironneau, K. Othsuka FreeFem++ : Manual http ://www.freefem.org/

[Hirsh-1988] C. Hirsch *Numerical computation of internal and external flows*, John Wiley & Sons, 1988.

[Koenig-1995] A. Koenig (ed.) : *Draft Proposed International Standard for Information Systems - Programming Language C++*, ATT report X3J16/95-087 (ark@research.att.com)., 1995

[Knuth-1975] D.E. Knuth , The Art of Computer Programming, 2nd ed., *Addison-Wesley*, Reading, Mass, 1975.

[Knuth-1998a] D.E. Knuth The Art of Computer Programming, Vol I : Fundamental algorithms, *Addison-Wesley*, Reading, Mass, 1998.

[Knuth-1998b] D.E. Knuth The Art of Computer Programming, Vol III : Sorting and Searching, *Addison-Wesley*, Reading, Mass, 1998.

[Lachand-Robert] T. Lachand-Robert, A. Perronnet (http://www.ann.jussieu.fr/courscpp/)

[Lascaux et Théodor] P. lascaux et R. Théodor Analyse numérique matricielle appliquée a l'art de l'ingénieur, Tome 2 Masson, 1987

[Löhner-2001] R. Löhner Applied CFD Techniques, Wiley, Chichester, England, 2001.

[Lucquin et Pironneau-1996] B. Lucquin, O. Pironneau *Introduction au calcul scientifique*, Masson 1996.

[Numerical Recipes-1992] W. H. Press, W. T. Vetterling, S. A. Teukolsky, B. P. Flannery : *Numerical Recipes : The Art of Scientific Computing*, Cambridge University Press, 1992.

[Raviart,Thomas-1983] P.A. Raviart et J.M. Thomas, *Introduction à l'analyse numérique des équations aux dérivées partielles*, Masson, Paris, 1983.

[Richtmyer et Morton-1667] R. D. Richtmyer, K. W. Morton : *Difference methods for initial-value problems*, John Wiley & Sons, 1967.

[Shapiro-1991] J. Shapiro *A C++ Toolkit*, Prentice Hall, 1991.

[Stroustrup-1997] B. Stroustrup *The C++ Programming Language*, Third Edition, Addison Wesley, 1997.

[Wirth-1975] N Wirth *Algorithms + Dat Structure = program*, Prentice-Hall, 1975.

[Aho et al -1975] A. V. Aho, R. Sethi, J. D. Ullman , Compilers, Principles, Techniques and Tools, Addison-Wesley, Hardcover, 1986.

[Lex Yacc-1992] J. R. Levine, T. Mason, D. Brown Lex & Yacc, O'Reilly & Associates, 1992.

[Campione et Walrath-1996] M. Campione and K. Walrath The Java Tutorial : Object-Oriented Programming for the Internet, Addison-Wesley, 1996.

Voir aussi Integrating Native Code and Java Programs. `http://java.sun.com/nav/read/Tutorial/native1.1/index.html`.

[Daconta-1996] C. Daconta Java for C/C++ Programmers, Wiley Computer Publishing, 1996.

[Casteyde-2003] Christian Casteyde Cours de C/C++ `http://casteyde.christian.free.fr/cpp/cours`

[3] C. Berge, *Théorie des graphes*, Dunod, 1970.