

# TRAVAUX PRATIQUES : METHODES NUMERIQUES

Programmation en fortran

## Table des matières

1	Quelques éléments de Fortran	2
2	Séance 1 : Introduction - Premières Exercices	8
3	Séance 2 : Zéros d'une fonction	8
4	Séance 3 : Intégration Numérique	8
5	Séance 4a : Equations Différentielles, application à un problème de thermique	9
6	Séance 4b : Résolution de systèmes linéaires, application à un problème de RdM	10

Vous devez rédiger un compte-rendu de TP pour la séance 4.

# 1 Quelques éléments de Fortran

## Objectif

L'objectif de ce petit fascicule est de fournir sous forme d'exemples, des différentes commandes de Fortran que vous serez amenées à utiliser au cours des séances de TP sur les méthodes numériques.

## Un petit programme comme exemple

Le programme `exemple.f`<sup>1</sup> exécute les actions suivantes :

- demande à l'utilisateur de définir un polynôme (degré et valeur des coefficients) de degré 10 au maximum.
- demande à l'utilisateur de choisir entre
  1. écrire dans le fichier `exemple.out` les valeurs du polynôme pour  $x \in [x_0, x_f]$  en  $N_{pt}$  points afin de le tracer avec un autre programme,
  2. calculer la valeur de la dérivée du polynôme en  $x$ ,
  3. sortir du programme

Le listing du programme est donné ci-dessous, ainsi que quelques explications.

```

Program Exemple
Implicit None
Real*8    x0,xf,coeff
Real*8    derivee,der
Integer   i,Npt,degre,choix

        Dimension coeff(11)

!      initialisation
Do i=1,11
    coeff(i)=0.d0
enddo

!      lecture au clavier du polynome
Write(*,*)'Entrer le degre du polynome (1 <= d <= 9) : '
Read(*,*)degre
If ((degre.LT.1).OR.(degre.GT.10)) Then
    Write(*,*)'Le degre du polynome doit etre compris entre 1 et 10'
    Stop
End If
degre=degre+1
Write(*,*)'Saisie des coefficientsdupolynome a_i.x^i,i=0 ..',degre
Do i = 1,degre
    Write(*,*)'coefficient devant x^',i-1
    Read(*,*)coeff(i)
End Do

Write(*,*)'  Que voulez vous faire?'

```

<sup>1</sup>que vous pouvez télécharger sur le site de l'IUP <http://www-meca.ujf-grenoble.fr/iup>

```

Write(*,*)' Tracer le polynome (1)'
Write(*,*)' Calculer une derivee (2)'
Write(*,*)' Finir (3)'

Read(*,*)choix
If ((choix.LT.1).OR.(choix.GT.3)) Then
  Write(*,*)'Vous devez choisir un numero entre 1 et 3'
  Stop
End If

If (choix.EQ.1) Then
  Write(*,*)'Nombre d''intervalles (entier) :'
  Read(*,*)Npt
  Write(*,*)'Valeur initiale :'
  Read(*,*)x0
  Write(*,*)'Valeur finale :'
  Read(*,*)xf
  Call filetracer(x0,xf,Npt,coeff,degre)
  Write(*,*)'Arret normal du programme'
  stop

ElseIf (choix.EQ.2) Then
  Write(*,*)'Abscisse de calcul de la derivee :'
  Read(*,*)x0
  der=derivee(x0,coeff,degre)
  Write(*,*)'la derivee en ',x0,' vaut ',der
  Write(*,*)'Arret normal du programme'
  stop

ElseIf (choix.EQ.3) Then
  Write(*,*)'Arret normal du programme'
  Stop
EndIf

End

```

!!!!!!!!!!!!!! Subroutine filetracer !!!!!!!!!!!!!!!

```

Subroutine filetracer(x1,x2,N,a,d)
Implicit None
Real*8 x1,x2,a,dx,x,y
Real*8 valpoly
Integer N,i,d
Dimension a(11)

Open(1,file='exemple.out')

dx=(x2-x1)/N
x=x1
Do i=1,(N+1)
  y=valpoly(x,a,d)
  Write(1,1000)x,y
  x=x+dx

```

```
End Do
Close(1)

1000 Format(e14.8,2x,e14.8)
Return
End

!!!!!!!!!!!!!! Fonction derivee !!!!!!!!!!!!!!!
Function derivee(x,a,d)
Implicit None
Real*8 x,a,ad,derivee
Real*8 valpoly
Integer d,i
Dimension a(11),ad(10)

Do i=1,d-1
    ad(i)=a(i+1)*i
End Do

derivee=valpoly(x,ad,d-1)
Return
End

!!!!!!!!!!!!!! Fonction valpoly !!!!!!!!!!!!!!!
Function valpoly(x,a,d)
Implicit None
Real*8 valpoly,x,a,poly
Integer d,i
Dimension a(*)

poly=a(d)

Do i=1,d-1
    poly=poly*x+a(d-i)
End Do
valpoly=poly
Return
End
```

### Quelques explications

Généralités sur la syntaxe d'un code en Fortran:

- MAJUSCULES et minuscules sont identiques,
- le nom d'un identificateur peut atteindre 31 caractères dont le blanc souligné " \_",
- le nom d'un identificateur ne peut pas commencer par un chiffre, mais peut en contenir (exemple 3xnf est mauvais mais gf65\_453 est bon).
- Program exemple : déclare le nom du *programme principal exemple* et marque le début du programme.

- **Subroutine NOMSUB** : déclare un *sous-programme* appelé NOMSUB. Pour le sous-programme tracer les *arguments* sont `x1`, `x2`, `N`, `a`, `d`. Seules les *variables* passées comme *arguments* sont connues à l'intérieur du sous-programme (par exemple la variable `choix` du programme principal n'est pas connue à l'intérieur des sous-programmes). Remarquez que les noms des *arguments* du sous-programme ne sont pas forcément identiques à ceux passés en *arguments* lors de l'appel.
- **Call NOMSUB** : pour appeler le sous-programme NOMSUB.
- **Function NOMFUNC** : déclare une *fonction* appelée NOMFUNC (*derivée*, *valpoly*). Une fonction est une *variable*, il faut donc définir son *type* (ici `Real*8`) et ne pas oublier d'affecter la valeur de la fonction avant le `Return`. Une fonction est appelée au sein d'une expression arithmétique. Il existe de nombreuses fonctions intrinsèques définies dans la librairie Fortran (exemple: la fonction *sinus* : `Sin(x)`).
- **End** : marque la fin du programme principal, d'un sous-programme ou d'une fonction (Obligatoire).
- **Implicit None** : indique que toutes les *variables* sont déclarées. C'est une sécurité indispensable pour éviter les erreurs de frappe. Les *variables* peuvent être de différents *types*:
  1. de type réel sur 4 octets (par défaut) ou 8 octets (`Real*8`),
  2. de type entier (`Integer`),
  3. ou aussi de type *complexe* (`Complex`) pour une paire ordonnée de nombres réels (parties réel et imaginaire), de type *logique* (`Logical`) qui ne peut prendre que deux valeurs `.TRUE.` ou `.FALSE.` et de type *caractère* `Character*n` pour une chaine de `n` caractères.
- **Dimension** : déclare la taille d'un tableau. Ici `a(11)` est un tableau de 11 réels. Il est possible de créer des tableaux à plusieurs *indices* (par exemple une matrice  $3 \times 3$  : `A(3,3)`).
- **!** indique que les caractères qui suivent sont des commentaires.
- **Boucle Do While (expression logique)**

```

      instructions exécutables
    End Do

```

exécute les instructions tant que l'expression logique est vrai. Pour exécuter au moins une fois la boucle il faut donc s'assurer que l'expression logique est vrai en entrant (c'est le rôle de `degre=11` ici). On dispose d'*opérateurs relationnels*: `.LT.` (plus petit), `.LE.` (plus petit ou égal), `.EQ.` (égal), `.NE.` (différent), `.GT.` (plus grand) et `.GE.` (plus grand ou égal). On a aussi des *opérateurs logiques* `.NOT.`, `.AND.`, `.OR.`, `.EQV.`, `.NEQV.` pour l'écriture de l'expression logique.
- **Test If (test1) Then**

```

      instruction1
    ElseIf (test2) Then
      instruction2
    Else
      instruction3
    End If

```

exécute l'`instruction1` si `test1` est vrai, l'`instruction2` si `test2` est vrai ou l'`instruction3` si `test1` et `test2` sont faux.

- Boucle `Do i=i1,i2,n`  
`instruction`  
`End Do`  
 exécute l'instruction pour `i` entier prenant les valeurs de `i1` à `i2` par pas de `n` (si le pas `n` n'est pas spécifier, il vaut 1 par défaut).
- `Write(iout,etiq) liste` : imprime à l'écran ou dans un fichier la liste. Ici `iout=* et etiq=* signifie que la sortie se fait à l'écran sans format. Pour écrire du texte, il faut le mettre entre ' . Attention, pour écrire Nombre d'intervalles, il faut taper 'Nombre d''intervalles'.`
- `Write(*,*)' texte ',degre` : écrit le texte suivi de la valeur de la *variable* `degre`.
- `Write(1,1000)x,y` : écrit dans le fichier 1 les *variables* `x` et `y` selon le *format* donné à l'*étiquette* 1000.  
`1000 Format(e14.8,2x,e14.8)` spécifie un format de lecture ou d'écriture. La syntaxe est la suivante :
- `eI.d` signifie qu'il s'agit d'un réel écrit sur `I` caractères (comprenant le signe, le point décimal, la lettre E) et ayant `d` chiffres après le point décimal. Par exemple 65.987 au format `e14.8` s'écrira `+.65987000E+02`.  
`nx` pour laisser `n` blancs.  
`An` spécifie qu'il s'agit de `n` caractères.  
`In` spécifie qu'il s'agit d'un entier de `n` chiffres significatifs (par exemple 12 au format `I5` s'écrira `00012`).
- `Read(iin,etiq) liste` : lit au clavier (`iin=*`) ou dans un fichier (`iin=numéro du fichier`) la *liste*. L'écriture d'un format se fait de façon identique à l'instruction `Write`.
- Gestion des fichiers `Open(ifich,file=nomfic)`  
`Close(ifich)`  
 pour ouvrir le fichier `nomfic`, lui affecter le numéro d'unité logique `ifich` et le fermer.

### et encore quelques unes ...

Quelques instructions utiles et non présentées dans le programme `exemple.f`:

- `Goto label` : dérouté l'exécution séquentielle d'un programme vers la ligne d'étiquette `label`.
- `Common` : permet d'attribuer la même valeur à des variables appartenant à des sous-programmes différents (sans passage par arguments). Exemple :  

```
! dans le programme principal
Real*8 fc28,feE,gamma_b,gamma_s
Common /CTE/ fc28,feE,gamma_b,gamma_s

!dans une fonction
Real*8 fcj,fe,g_b,g_s
Common /CTE/ fcj,fe,g_b,g_s
```

 où CTE est le nom du `Common` contenant quatre variables réelles (notez que le nom des variables peut changer puisque c'est l'ordre dans le `Common` qui compte).
- `Data`: assigne des valeurs initiales aux variables. Exemple:  

```
! à placer après la déclaration des types
Data fc28,feE,gamma_b,gamma_s /25, 500, 1.5,1.15 /
```

- Le tableau suivant donne quelques fonctions intrinsèques :

Fonction	Nom	Type paramètre	Type Fonction
Valeur absolue	Abs (DAbs)	Real*4 (Real*8)	Real*4 (Real*8)
Log naturel	ALog (DLog)	Real*4 (Real*8)	Real*4 (Real*8)
Log base 10	ALog10 (DLog10)	Real*4 (Real*8)	Real*4 (Real*8)
Exponentielle	Exp (DExp)	Real*4 (Real*8)	Real*4 (Real*8)
Racine carré	Sqrt (DSqrt)	Real*4 (Real*8)	Real*4 (Real*8)
Sinus	Sin (DSin)	Real*4 (Real*8)	Real*4 (Real*8)
Cosinus	Cos (DCos)	Real*4 (Real*8)	Real*4 (Real*8)
Tangente	Tan (DTan)	Real*4 (Real*8)	Real*4 (Real*8)
Arcsinus	ASin (DASin)	Real*4 (Real*8)	Real*4 (Real*8)
Arccosinus	ACos (DACos)	Real*4 (Real*8)	Real*4 (Real*8)
Arctangente	ATan (DATan)	Real*4 (Real*8)	Real*4 (Real*8)
sinus hyper.	Sinh (DSinh)	Real*4 (Real*8)	Real*4 (Real*8)
Cosinus hyper.	Cosh (DCosh)	Real*4 (Real*8)	Real*4 (Real*8)
Tangente hyper.	Tanh (DTanh)	Real*4 (Real*8)	Real*4 (Real*8)
Troncature	AINT (DINT)	Real*4 (Real*8)	Real*4 (Real*8)
Arrondi	ANINT (DNINT)	Real*4 (Real*8)	Real*4 (Real*8)
Arrondi entier	ININT (IDNINT)	Real*4 (Real*8)	Integer (Integer)
Maximum	AMax1 (DMax1)	Real*4 (Real*8)	Real*4 (Real*8)
Minimum	AMin1 (DMin1)	Real*4 (Real*8)	Real*4 (Real*8)
Partie imaginaire	AImag (DImag)	Comp*8 (Comp*16)	Real*4 (Real*8)
Partie réelle	AReal (DReal)	Comp*8 (Comp*16)	Real*4 (Real*8)
Conjugué	Conjg (CDConjg)	Comp*8 (Comp*16)	Comp*4 (Comp*8)

## 2 Séance 1 : Introduction - Premières Exercices

A1/ Calculer  $i * i$  ( $i = 0, 1, \dots, 100$ ) et afficher les résultats à l'écran.

A2/ Calculer la somme des carrés  $i * i$  ( $i = 0, 1, \dots, 100$ ) et afficher le résultat final à l'écran.

A3/ Calculer le  $\sin x$ ,  $x$  allant de 0 à  $2\pi$ . Ensuite tracer le  $\sin x$  avec OpenOffice.

A4/ Faire un programme avec lequel l'utilisateur peut choisir entre les programmes A1 (option 1), A2 (option 2) ou A3 (option 3).

## 3 Séance 2 : Zéros d'une fonction

Appliquer la méthode de bisection, la méthode linéaire et la méthode secante ( $N_{max} = 20$ ,  $Précision = 0.0001$ ,  $x_0 = 0.5$ ,  $x_1 = 0.6$ ) et la méthode de Newton ( $N_{max} = 20$ ,  $Précision = 0.0001$ ,  $x_0 = 0.5$ ) à la recherche de la racine de :  $f(x) = \exp(-x) - \sin(x)$ . Utilisez les résultats de ce test pour obtenir une idée plus précise de la vitesse de convergence.

**Remarque** La méthode de Newton est certainement la plus efficace pour rechercher les zéros d'une fonction :  $f(x) = 0$  (sauf évidemment dans quelques cas particuliers où elle peut alors échouer totalement). Partant d'une estimation  $x_0$  de la racine on a l'algorithme itératif,

$$(1) \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

## 4 Séance 3 : Intégration Numérique

Pour intégrer numériquement la fonction  $f(x)$  sur un intervalle fini, les différents algorithmes procèdent à ce calcul par discrétisation :

$$(2) \quad \int_a^b f(x) dx \simeq \sum_i h_i f(x_i)$$

Les abscisses  $x_i$  et les poids  $h_i$  sont caractéristiques de la méthode employée.

Nous étudierons uniquement deux méthodes de Newton-Cotes (trapèze, Simpson).

### Les méthodes de Newton-Cotes.

Ce sont des méthodes à pas constant  $x_i - x_{i-1} = h$  ; elles consistent à remplacer localement la fonction à intégrer par un polynôme de degré  $n$ . On notera  $f_i = f(x_i)$ . Ces algorithmes, bien que peu précis, sont néanmoins très utiles quand la fonction  $f(x)$  est connue seulement en des points  $x_i$  discrets (résultats d'expériences).

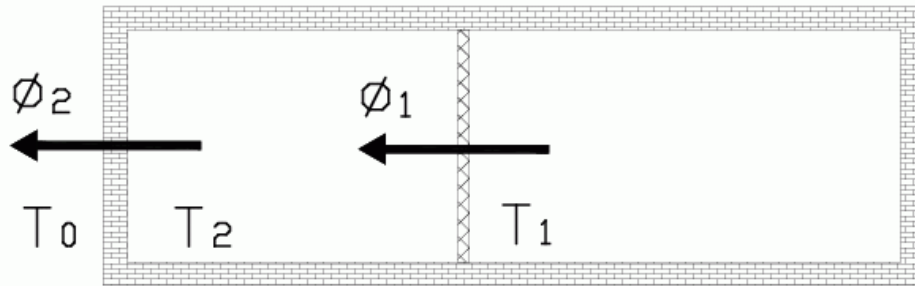
### Les trapèzes et Simpson

1. La méthode des trapèzes pour  $m + 1$  points s'écrit :

$$(3) \quad \int_{x_0}^{x_m} f(x) dx = h \left[ \frac{f_0}{2} + f_1 + \dots + f_{m-1} + \frac{f_m}{2} \right] - \frac{mh^3}{12} f''(\xi) \quad (\xi \in [x_0, x_m]).$$

2. La méthode de Simpson pour  $2n + 1$  points (nombre impair) donne :

$$(4) \quad \int_{x_0}^{x_{2n}} f(x) dx = \frac{h}{3} [f_0 + 4(f_1 + f_3 + \dots + f_{2n-1}) + 2(f_2 + f_4 + \dots + f_{2n-2}) + f_{2n}] - \frac{nh^5}{90} f^{(4)}(\xi).$$



**Application** Evaluer l'intégrale de  $f(x) = e^{-x} \sin(kx)$  pour  $x \in [0, 2\pi]$  et  $k = 2, 4, 6, \dots$

On étudiera en particulier, pour les deux méthodes précédentes, la convergence en fonction du nombre de points.

## 5 Séance 4a : Equations Différentielles, application à un problème de thermique

### Equations Différentielles Ordinaires du 1<sup>er</sup> ordre (E.D.O.)

#### Méthode d'Euler

Soit à résoudre une équation différentielle du type :

$$(5) \quad y' = f(x, y) \quad \text{pour } x \in [a, b] \quad \text{avec la condition } y(a) = y_0.$$

Voyons d'abord la méthode d'Euler, qui est la plus simple, mais aussi la plus mauvaise (il vaut mieux en règle générale utiliser une méthode de Runge-Kutta, cf. les sections suivantes).

On détermine les valeurs  $y_n = y(x_n)$  pour  $x_n = a + nh$ , ( $n = 0, 1, 2, \dots$ ,  $n$  numéro de l'intervalle et  $h$  le pas), par la formule dite d'Euler (on la dérive très simplement à partir du développement de Taylor).

$$(6) \quad y_{n+1} = y_n + hf(x_n, y_n) + O(h^2)$$

#### Méthodes de Runge-Kutta

Les schémas de Runge-Kutta assurent, en général, une meilleure stabilité de la solution que la méthode d'Euler. Il existe des schémas à 2, 3 ou 4 points, avec des termes d'erreurs en  $O(h^3)$ ,  $O(h^4)$  et  $O(h^5)$  respectivement.

L'un des deux schémas de Runge-Kutta d'ordre 2, s'écrit:

$$(7) \quad \begin{aligned} y_{n+1} &= y_n + (k_1 + k_2)/2 + O(h^3); \\ k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h, y_n + k_1) \end{aligned}$$

### Application à un problème de thermique

#### Contexte

On se propose de simuler numériquement l'échange thermique entre deux pièces (1) et (2) d'un bâtiment et l'extérieur. Les températures  $T_1(t)$  dans (1) et  $T_0(t)$  à l'extérieur sont imposées en fonction du temps  $t$ . La température  $T_2(t)$  dans (2) est inconnue. Les flux thermiques sont modélisés de la manière suivante :

On suppose que  $\Phi_2 = K_2(T_2(t) - T_0(t))$  et que  $\Phi_1 = K_1(T_1(t) - T_2(t))$ , o  $K_1 = 6W/^\circ C$  et  $K_2 = 3W/^\circ C$  sont les coefficients de transmission des parois entre (1)-(2) et (2)-extérieur.

Par ailleurs, on suppose que la capacité volumique de la pièce (2) reste constante et vaut  $C = 200Wh/^\circ C$ . En faisant un bilan des échanges thermiques de la pièce (2), on en déduit l'équation différentielle suivante pour la température  $T_2(t)$  :

$$(8) \quad C \frac{dT_2(t)}{dt} = K_1(T_1(t) - T_2(t)) - K_2(T_2(t) - T_0(t))$$

### Travail demandé

Après avoir mis l'équation précédente sous la forme  $y' = f(x, y)$ , écrire un programme permettant de résoudre les deux problèmes suivants :

1. **Incendie dans la pièce (1) :** Pour ce problème, on suppose que  $T_0 = 10^\circ C$  est constante au cours du temps et que la température dans la pièce (1) augmente linéairement de  $T_1(t = 0) = T_0 = 10^\circ C$  à  $T_1(t = 0.5h) = 910^\circ C$ , puis reste constante pour  $t > 0.5h$ . En prenant comme condition initiale  $T_2(t = 0) = T_0$ , déterminer le temps pour lequel l'alarme incendie dans la pièce (2) se déclenche, celle-ci étant réglée sur une température de  $70^\circ C$ . Vers quelle valeur tend la température  $T_2$  ?
2. **Variation journalière de la température extérieure :** On modélise la variation de température extérieure  $T_O(t)$  par une fonction cosinus de période  $24h$ , de valeur moyenne  $10^\circ C$  et d'amplitude  $10^\circ C$ , les températures minimale et maximale étant à  $0h$  et  $12h$ , respectivement. Par ailleurs, on suppose que la température dans la pièce (1) est constante et vaut  $T_1 = 18^\circ C$ . Etudier l'évolution de la température dans la pièce (2) sur un certain nombre de jours. Quelle est l'influence de la condition initiale sur  $T_2$  ? Quel est la valeur du déphasage entre la température extérieure et celle dans la pièce (2) ?

## 6 Séance 4b : Résolution de systèmes linéaires, application à un problème de RdM

### Contexte

On se propose de traiter un problème de mécanique qui conduit à la résolution d'un système d'équations linéaires  $A \cdot \vec{x} = \vec{b}$ . L'objectif du TP est de programmer la méthode des pivots de Gauss permettant le calcul de la solution  $\vec{x}$  du système.

### Poutre isostatique supportant une charge répartie $q(x)$

Soit une poutre isostatique de caractéristiques mécaniques  $E$ ,  $I_z$  et de longueur  $l_x$ , supportant une charge répartie  $q(x)$ . L'équation de la déformée est

$$(9) \quad EI_z y^{IV} = q(x),$$

o  $y^{IV}$  est la dérivée 4<sup>ième</sup> de la déformée  $y(x)$ .

La discrétisation selon  $x$  en  $N_{pt}$  points régulièrement répartis permet d'exprimer la dérivée 4<sup>ième</sup> au point  $x_i$  en fonction des valeurs de la déformée  $y_{i-2}$ ,  $y_{i-1}$ ,  $y_i$ ,  $y_{i+1}$  et  $y_{i+2}$ , soit:

$$(10) \quad y_i^{IV} = \frac{y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}}{h^4}.$$

1/ En appliquant les conditions aux limites en  $x = 0$  et  $x = l_x$ , réécrivez cette expression pour les points  $x_0$ ,  $x_1$ ,  $x_{N_{pt}-1}$  et  $x_{N_{pt}}$ . Montrez alors que la résolution de l'équation (9) revient

à résoudre un système d'équations linéaires d'inconnues  $y_i$ . Donnez la forme de la matrice  $\mathbf{A}$  et du second membre  $\vec{b}$ .

Une partie du programme principale permettant la résolution de ce problème vous sera donnée (`poutre.f`)<sup>2</sup>. Ce programme permet la saisie d'un chargement  $q(x) = a_k + b_k x + c_k x^2$  défini par morceaux sur des tronçons  $k$ .

2/ Vérifiez que la matrice  $\mathbf{A}$  et le second membre  $\vec{b}$  calculés avec le programme `poutre.f` sont identiques à ceux obtenus en 1/.

3/ Ecrivez le sous programme `PIVOT(A,B,y,N,Nmax)` (où  $N$  est la dimension du problème et  $N_{max}$  la taille maximale du problème) qui retourne la solution  $\mathbf{y}$  du système linéaire  $\mathbf{A} \mathbf{y} = \mathbf{B}$ . Dans un premier temps, il ne sera pas fait de modification sur l'ordre des lignes (les lignes sont prises dans l'ordre initial, avec comme conséquence le risque d'obtenir un pivot nul alors que la matrice n'est pas singulière). Dans un deuxième temps, le sous-programme sera modifié pour effectuer la recherche du plus grand pivot en valeur absolue<sup>3</sup>.

4/ Testez le programme pour les cas de charges suivants:

$$y(lx/2) = \frac{5ql_x^4}{384EI_z} \qquad y(0.519lx) = \frac{10ql_x^4}{1536EI_z} \qquad y(lx/2) = \frac{ql_x^4}{120EI_z}$$

<sup>2</sup>vous pouvez aussi le télécharger sur le site de l'IUP <http://www-meca.ujf-grenoble.fr/iup>

<sup>3</sup>La méthode la plus simple pour modifier l'ordre des lignes est de le faire au fur et à mesure dans un vecteur appelé *pointeur*.