Première partie

Représentation des nombres

Nomre IEEE754

1 byte = 1 octet = 8 bits
un entier est représenté par 4 byte =32 bits. La plage de représentation est de

$$-2^{-31} < N < 2^{31}$$

équiavament en puissance de 10 à

$$-10^{-9} < N < 10^{9}$$

Representation des decimaux

Un decimal s'ecrit sous la forme $m10^e$ avec m la mantisse et e l'exposant.
En machine, le décimal s'écrit $M2^E$.

31	30... 23	22... 0
signe	exposant biaisé	mantisse
1 bit	8 bits	23 bits

$(-1)^{bitSigne} * (1 + mantisse) * 2^{exposantbiaisé-127}$
maximum = $2^{2^7} = 2^{128} \simeq 2^{3^{40}} \simeq 10^{40}$
minimum = $2^{-2^7} \simeq 10^{-40}$

Zero machine et precision machine

Définition zéro machine : Le plus grand x tel que x soit represente par 0 en machine. (dépend de l'exposant)

Définition précision machine : Le plus grand x tel que 1+x soit represente par 1 en machine. (dépend de la mantisse)

exemple
$2^{23} \simeq 10^7$ précision machine : 10^{-7}
1 + 0.000 001 = 1.000 001
1 +0.000 000 01 = 1.000 000 0 = 1

Flottants représentés sur 8 byte

63	62... 52	51... 0
signe	exposant biaisé	mantisse
1 bit	11 bits	52 bits

$(-1)^{bitSigne} * (1 + mantisse) * 2^{exposantbiaisé-1023}$
zéro machine = 10^{-308}
précision machine = 10^{-14}

Arithmetique etendue Norme IEEE 754

La norme introduit Inf + ∞Overflow -Inf $-\infty$ Underflow NaN not a number, Invalid operation, operation illegale (ex $\sqrt{-1}$)

Ces quantites obeissent a des regles d'operation bien precises
Inf $\pm$ x = Inf

Inf $\times$ x = Inf
Inf - Inf = NaN
Inf * 0 = NaN
N'importe quelle operation sur un NaN donne un NaN

Deuxième partie

L'Éditeur VI

Cet éditeur peut fonctionner en 3 modes différents : un mode commande, un mode insertion et un mode ligne.

Pour l'appeler taper 'vi nom de fichier'. Le pointeur se trouve alors en 1ere ligne 1ere colonne du fichier en mode commande.

Comme tout programme UNIX vi distingue les majuscules, si vous avez des problèmes et que rien ne marche il est fort possible que vous soyez en mode majuscule (Caps. Lock). Habituez vous à utiliser Esc de façon préventive.

Le mode commande

Ce mode permet de visualiser le fichier, de se déplacer, la recherche de chaines de caractères... Pour s'assurer que l'on est en mode commande taper Esc (escape) , si on est en mode insertion on en sort.

Déplacements

- lents : utiliser les fleches, ou les touches 'h,j,k,l'
- rapides : Ctrl f (forward) , Ctrl b (backward) , Ctrl u(up) , Ctrl d (down)
- '0' positionne le pointeur en début de ligne courante, '$' en fin.
- 'G' positionne le pointeur en fin de fichier. Pour aller en début de fichier il faut utiliser le mode ligne et taper ' :1' suivi de Return.
- Pour voir ou on est dans le fichier : Ctrl g (affiche ou, dans quel fichier)
- Au cas ou l'affichage a été brouille, Crtl l le restaure
- Recherche de chaines : '/chaine/' suivi de enter (return) met le pointeur sur la première occurrence de chaine dans le texte, pour voir les suivantes taper n (next), pour les précédentes taper N. Attention aux caractères particuliers , ils doivent être précédés d'un / ainsi /3\.4\$/ recherche la chaine 3.4$ dans le texte.

Modification de texte

- 'x' détruit la lettre sur laquelle est le pointeur
- 'dw' détruit le mot (un mot est un groupe de lettres définis par 2 blancs ou deux caractères spéciaux . *..)
- 'dd' détruit la ligne courante
- 'D' détruit la ligne a partir du pointeur
- 'J' joint la ligne courante a sa suivante (mise en page)

Repetition et annulation de commande

- '.' répète la dernière commande effectuée (même en mode ligne!) Couplee avec n ou N se revele très utile pour des substitutions.
- 'u' annule la dernière opération effectuée.

Copie - coller

- Il existe 10 buffers temporaires sous vi reperes par les lettres a-j. Ces buffers restent actifs meme si on change de fichier, lors d'une session vi. Pour copier les 6 lignes suivant le pointeur dans le buffer a "a6yy Pour coller après avoir positioné le pointeur "ap
- '4yy' copie 4 lignes à partir du curseur
- Se placer en mode visuel : Esc puis taper 'v'. Se séplacer ensuite avec les lignes pour pour sélectionner les lignes à copier. Taper ensuite sur la touche y pour copier puis p pour coller.

Le mode insertion

Ce mode permet d'introduire du texte dans le fichier. Les deux principales commandes sont 'i' (insert) , 'a' (append). Lorsque l'on tape 'i' , rien ne s'affiche, on peut alors taper son texte avec les retours a la ligne. Ce texte sera place juste avant le pointeur. 'a' fonctionnne de la meme facon sauf que le texte est

place apres le pointeur. On sort du mode insertion en tapant Esc (escape). Pour placer du texte en debut de ligne utiliser 'I', en fin 'A' .

- Pour changer un mot utiliser 'cw', taper la correction, sortir avec Esc.
  - manip pratique pour changer rapidement des instructions qui se ressemblent dans un makefile par exemple.
  - /mot1
    cw mot2 [esc]
    n [enter]
    . [enter]
- Pour changer toute les occurences d'une chaine de caractères dans un fichier : ' :%s/anciennechaine/nouvellechaine/g'

Le mode ligne

Il est constitué par un éditeur ligne appele 'ex'. Pour l'appeler il suffit de faire précéder la commande par ' :'. Il permet la position du pointeur sur une ligne donnée, ce qui est très utile pour une message d'erreur lors d'une compilation par exemple. Il sert aussi pour substituer des chaines de caracteres, il sert à la sauvegarde du fichier et permet l'accès à unix.

position du pointeur

- ' :10' suivi de Return met le pointeur en ligne 10 ' :=' donne le numero de la ligne courante
- ' :set nu' affiche les numeros de ligne

sauvegarde du fichier

- ' :w ' ecrit le fichier courant sur disque
- ' :wq' sauve et sort
- ' :q!' sort sans sauver
- ' :w! file' force l'ecriture dans file

manipulation de blocs de lignes

detruction

- '23,45d' detruit les lignes 23 a 45
- '.,$d' detruit les lignes de la ligne courante a la fin du fichier.

ecriture dans un fichier

- ' :1,45w file' ecrit les lignes 1 a 45 dans file.

inclusion d'un fichier

- ' :r file' importe file dans le fichier courant a la position du pointeur. (cette commande couplee a la precedente permet facilement de faire du 'copie- coller')

changement de fichier courant

- ' :e file2' edite file2

Appel a UNIX

- ' : sh' permet de lancer un shell apartir de vi, sans sortir. On en sort avec Ctrl d
- ' : ! commande unix' permet de lancer une commande unix quelconque. Ainsi ' : !ls' fait une liste des fichiers dans le repertoire courant.
- ' :r !ls' inclue la liste des fichiers dans le repertoire courant dans le fichier en cours d'edition.
- '!xterm &' lance un xterm en tache de fond (pour ne pas bloquer la fenetre "lanceuse").

Troisième partie

Fortran

Syntaxe et règles operatoires

- langage utilise principalement pour le calcul scientifique standard
- Variables en minuscule, pas d'accents
- typer toutes les variables (implicit none) (a-h,o-z) real , (i-m) integer
- Format fixe : instructions entre colonnes 7 et 72 colonne 6 prolongation ligne precedente, colonne 1 commentaire (c)
- L'espace ne joue pas de role, utilisez le pour que votre code soit lisible : indentez les boucles, 'aerez' vos formules...
- fortran passe des adresses, toutes les variables sont locales pour un tableau fortran passe l'adresse du premier element C passe des valeurs
- pas de melanges de types de flottants, ou real ou double precision pas les deux

Évaluation des expressions

Le resultat d'operations depend du type de variables utilisées.

Lorsque l'on ecrit n3 =n1/n2 ou n1,n2 et n3 sont des entiers, le resultat est le quotient de la division euclidienne de n1 par n2.

Une operation entre deux variables donne un resultat qui est represente par la machine par le type le plus eleve des deux variables en respectant la hierarchie double precision > real > integer

exemples : $1./2 \rightarrow 0.5$ $1/2 \rightarrow 0.$

Les priorités des operateurs arithmetiques sont les suivantes $** > */ > +-$

Les priorités des operateurs logiques sont les suivantes

- comparateurs .gt. .ge. .eq. .ne. .le. .lt.
- .not.
- .or.
- .eqv. .neqv.

Les priorités des operateurs sont les suivantes

- operateurs arithmetiques
- concatenation
- comparaison
- operateurs logiques

Dans l'evaluation d'expressions complexes le compilateur suit l'ordre suivant

- appels de fonctions
- expressions entre parentheses en commmencant par les plus internes
- le reste est evalué en suivant les priorités des opérateurs de gauche à droite pour les opérateurs de priorités égales

Boucle a nombre fixe d'iterations

Repetition d'un grand nombre d'operations en variant des données notation recente (fortran 90 et 95) , plus claire

```
do  i=n1 , n2 , n3
          .
          .
          .
enddo
```

avec les regles suivantes :

- n1 = n2 la boucle est effectuee une fois
- n1 > n2 et n3 >0 , la boucle n'est pas effectuee
- n1 < n2 et n3 >0 , la boucle est effectuee pour i=n1,n1+n3 n1+2*n3,...n1+k*n3 avec k tel que n1+(k+1)*n3 > n2

Dans l'exemple donne plus haut n3=1 (la valeur par defaut)

4 Boucle conditionnelle

Repetition d'un grand nombre d'operations jusqu'a ce qu'un resultat de test change.

do while (tant que) obsolete remplace par boucle do avec l'instruction exit de sortie de boucle permet un meilleur controle de flux.

```
do
        if(condition) then
                exit
        endif
enddo
```

Exemple : iteration $x_{n+1} = f(x_n)$ avec comme criteres d'arret n < nmax ou $|x_{n+1} - x_n| <$ tol

— Version avec exit

```
x = x0
n = 0
res = abs(x0 - f(x0))
icon = 1
do n=1,nmax
        x1 = f(x)
        res = abs(x1 - x)
        if (res < tol) then
                icon = 0
                exit
        endif
        write(*,*) n,x1,res
        x = x1
enddo
```

L'indice icon indique quel est la raison pour l'arret icon = 0 convergence , sinon nmax iterations sans converger

— Version moins precise avec le do while (ne permet pas de preciser la raison de l'arret)

```
x = x0
n = 0
res = abs(x0 - f(x0))
do while(n < nmax) .and. (res > tol)
        n = n + 1
        x1 = f(x)
        res = abs(x1 - x)
        write(*,*) n,x1,res
        x = x1
enddo
```

Remarques : Apres la sortie de boucle la valeur de l'indice de boucle est perdue donc il ne faut jamais l'utiliser hors de la boucle. Un indice de boucle ne doit jamais etre modifie dans la boucle.

Manipulation des tableaux dynamique

— **allocate** (array, stat=err) permet de faire l'allocation mémoire du tableau. Si le mot clé stat= est spécifié, la variable err (de type integer) vaut 0 si l'allocation s'est déroulée sans problème et 1 sinon.

— **deallocate** (array, stat=err) permet de libérer l'espace mémoire réservée à un tableau.

— **allocated**(array) est une fonction intrinsèque renvoyant true ou false suivant que le tableau spécifié en argument est alloué ou non.

```
integer, dimension(:,:), allocatable :: a
integer :: n,m,err
```

```
read(*,*) n , m
if (.not(allocated(a)) then
        allocate(a(n,m),stat=err)
        if (err /= 0) then
                write(*,*) 'erreur allocation dynamique du tableau a'
                stop
        endif
endif
.
.
.
deallocate(a)
```

Fonctions information sur des tableaux

- **size** (ARRAY, dim) is a function which returns the number of elements in an array ARRAY , if DIM is not given, and the number of elements in the relevant dimension if DIM is included.
- **product** (array [,dim][,mask]) et sum (array [,dim][,mask]) retournent le produit / la somme des valeurs des éléments du tableau array (après un éventuel filtrage par dim et / ou mask).

Exemple : product(A) retourne 720 sum(A,dim=1,A > 2) vaut (/ 4, 5, 9 /)

- **lbound** (array [,dim]) et **ubound**(array [,dim]) retournent les bornes inférieures / supérieures de chacune des dimensions (ou seulement de la dimension dim) du tableau array.

Exemple : integer, dimension(21 :2, 45 :49) : : tab

lbound(tab) vaut (/ 21, 45 /) et ubound(tab) vaut (/ 2, 49 /) l

bound(tab, dim=2) vaut 45 et ubound(tab, dim=1) vaut 2

- **all** (mask [,dim]) applique un masque de type logique sur les éléments du tableau et renvoit vrai si pour tous les éléments le résultat du masque est vrai. Si dim est précisé, la fonction travaille sur cet indice pour chaque valeur des autres dimensions.
- **any** (mask [,dim]) fonctionne de la même façon que la fonction ALL à l'exception qu'elle renvoit vrai si l'un des résultats du masque est vrai.
- **count** (mask [,dim]) comptabilise le nombre d'éléments pour lesquels le résultat du masque est vrai.

```
program test

logical l
integer a(2,3),b(2,3)

write(*,*) all((/.false.,.true.,.true./))
write(*,*) any((/.false.,.true.,.true./))

a = 1
b = 1
b(2,2) = 2
b(2,3) = 2
write(*,*) a
write(*,*) b
write(*,*) all(a == b, 1)
write(*,*) all(a .eq. b,2)
write(*,*) any(a .eq. b,1)
write(*,*) any(a .eq. b,2)
count(a /= b)
end.
```

Retourne une fois exécuté

```
F
T
1  1  1  1  1  1
1  1  1  2  1  2
T  F  F
T  F
T  T  T
T  T
2
```

- **minval** (array [,dim][,mask]) et **maxval**(array [,dim][,mask]) retournent la plus petite / plus grande valeur du tableau array (après un éventuel filtrage par dim et / ou mask).

exemple minval(A,dim=2,A > 2) vaut (/ 3, 4 /) norme du max maxval(abs(A))

- **minloc** (array [,dim][,mask]) et **maxloc** retournent l'indice de la plus grande ou dela plus petite valeur. Attention : maxloc/minloc dans des sections de matrice, l'indice doit etre corrige et il faut mettre dim.

exemple recherche du pivot dans la colonne k sous diagonale de matrice D ipiv = maxloc(D(k :n,k),1)+k-1

Initialisation de tableaux

```
integer, parameter :: n = 4
integer, dimension(n) :: t1, t2, t3
integer i
t1 = (/6.5, 10, 1/)
t2 = (/(i+1, i=1,n)/)
t3 = (/t2(1), t1(3), 1, 9/)
```

Programme pour remplir une matrice triangle inférieur à partir d'une matrice

```
integer, parameter :: n = 4
integer, dimension(n,n) :: a, t
integer i

call random_number(a)
t = 0.

do i = 1,n
        t(i,1:i) = a(i,1:i)
enddo
```

Initialisation à partir d'un fichier de donnée

```
integer n
integer, dimension(:,:), allocatable :: a
integer, dimension(:), allocatable :: b

read(*,*) n
allocate(a(n,n))
allocate(b(n))
read(*,*)
do i = 1,n
        read(*,*) a(i,1:n)
enddo
read(*,*)
read(*,*) b
```

avec le bon fichier de données :

```
4

1  2  3  10
5  4  1  2
4  8  7  9
4  5  5  1

1  0  0  1
```

Sections de tableaux

Section régulière : les éléments forment une progression géométrique
tab(ini : fin : pas)
exemple

```
integer ,dimension(10) :: a=(/1,-2,3,-4,5,-6,7,-8,9,-10/)
integer :: i=3,j=7
a (:)
a(i:j)              (3, -4,5,-6,7)
a(i:j:i)            (3, -6)
a(i:)               (3,-4,5,-6,7,-8,9,-10)
a(:j)               (1,-2,3,-4,5,-6,7)
a(::i)              (1,-4,7,-10)
```

Possibilite de sections non regulieres , adressage indirect

Operations entre tableaux

- **transpose** (matrix), où matrix est un tableau de dimension 2, retourne la transposée de matrix.
- **dot_product** (a, b) retourne le produit scalaire de deux vecteurs, c'est à dire at.b si a et b sont de type entier ou réel, conjugué(a)t.b si a et b sont complexes, et Somme(ai .and. bi) si a et b sont de type logique.
- **matmul** (a, b) retourne le produit matriciel de a et b, sous réserve de compatibilité des dimensions. Soit a et b sont tous deux des matrices, soit a est un vecteur et b est une matrice, soit a est une matrice et b est un vecteur. a et b peuvent être de type quelconque.
- **norm2** (a) retourne la norme euclidienne du tableau a (vecteur ou matrice).

Action sur des tableaux

L'instruction **where** permet d'effectuer des opérations sur des éléments d'un tableau sélectionnés via un filtre de type logique.

```
real, dimension(10) :: a
where (a > 0.) a = sqrt(a)
where (a > 0.)
        a = log(a)
elsewhere
        a = 1
end where
end.
```

Pour plus d'informations https ://gcc.gnu.org/onlinedocs/gfortran/Intrinsic-Procedures.html#Intrinsic-Procedures

Variables statique ou dynamiques

Variable statique : mémoire reservé à la compilation

Variable dynamique : mémoire reservé à l'exécution
Variables et initialisations : data et save
Déclarer les variables par type et centre d'intérêt suivant leur fonction en haut du code
— déclaration dynamique
— déclaration et affectation statique
— save pour garder la valeur des variables entre chaque passage dans une subroutine.

Exemple : réalisation d'un compteur interne

```
 subroutine sub(..)
integer :: icom=0
real,save :: pi

if (icom .eq. 0) then
        pi = 4.*atan(1.)
endif
...
icom = icom + 1
return
end
```

Le save icom n'est pas nécessaire car icom étant initialisée devient statique. Par contre le save pi lui est nécessaire.

Procédure et fonction

Principal

```
program mm
implicit none
integer n,i
real y
double precision x
....
```

Sous–programme

```
subroutine trap(n,a,b,f,s)
implicit none
integer n,i
real a,b
double precision,external f  #on retype tout même ce qu'il y a en entré de la sub
double precision s
...
end
```

```
real function sinx(x)
implicit none
real x
sinx = sin(x)/x
return
end
```

Fonctions déjà inclues dans Fortran

Fonctions de conversion

- **int**(e) partie entière de e
- **real**(e) convertit e en réel
- **dble**(e) convertit e en double précision
- **cmplx**(e) convertit e en complexe

Exemples

int(3.14) = 3 c
mplx(3.14) = 3.14 + 0*i
cmplx(3.14,2.) = 3.14 + 2.*i 2

Fonctions incorporées arithmétiques

Le ou les arguments et le résultat sont de même type (entiers, réels ou double precision).

- **abs**(e) valeur absolue
- arrondi **nint**(e) entier le plus proche de e **anint**(e) entier le plus proche de e, ecrit sous forme reelle.
- troncature **aint**(e) : partie entiere de e
- reste d'une division **mod**(a,b) avec b non nul

mod(5,3)=2
mod(5.,3.)=2.

- Extrema de valeurs scalaires **min**(e1,e2...) = minimum des ei **max**(e1,e2...) = maximum des ei
- transfert de signe **sign**(q,s) avec s non nul = |q| si s $>$ 0 $-$|q| si s $<$ 0

Fonctions relatives aux complexes

declaration
complex z

- z = cmplx(x,y)
- **real**(z) donne x
- **aimag**(z) donne y
- **conjg**(z) donne x-i*y
- **abs**(z) donne $x^2 + y^2$

Fonctions mathematiques

Les arguments sont reels ou double précision parfois complexes mais jamais entiers.

- racine carrée **sqrt**(x)
- exponentielle **exp**(x)
- **log**(x), **log10**(x) (si $z = \rho e^{i\theta}$ complexe $log(z) = log(\rho) + i\theta' 0 < \theta' < \pi$)
- cos, sin,tan , cosh,sinh,tanh
- **acos**(x) , **asin**(x) pour $|x| < 1, 0 < acos < \pi, |asin| < \pi/2$
- **atan**(x) abs(atan) $< \frac{\pi}{2}$, **atan2**(y,x) = angle sur le demi-cercle trigo $z = x + i * y$ positif ou -angle sur le demi-cercle trigo négatif

Passage de fonction / subroutine comme argument

Pour passer une fonction ou une procédure en paramètre d'une subroutine, on déclare la fonction avec le mot clé **external.**

```
program main
implicit none
double precision, external MaFonction
double precision a,b,tol

read(*,*) a,b,tol
```

```
call bis(a,b,MaFonction,tol)

end.

subroutine bis(a,b,f,tol)
implicit none
real a,b
double precision tol,f,res

res = f(a) - f(b)
write(*,*) res
return
end

double precision fonction MaFonction(x)
implicit none
double precision x

MaFonction = sin(x)
end
```

Module

Lorsqu'une variable est utilisé par différents sous programme, on peut faire une fichier appelé module qui sera utilisable par toute les subroutine.

```
module modcch

double precision, save :: w

end module modcch
```

la variable w devient statique et peut être utiliser sans être redéclaré dans d'autre fonction à condition que la compilation soit correctement faite

Quatrième partie

Compilation

Les options du compilateur

Sous Unix, les compilateurs gfortran, gcc, g++ effectuent l'opération de compilation et de link à la suite.

syntaxe de la commande

gfortran file.f -o file options lib.a otherfile.o

Le fichier qui est compilé est file, on veut le linker avec une librairie (collection de fichiers objet) notée lib.a sous unix. Cette librairie peut avoir ete construite par l'utilisateur (commande 'ar') ou non.

Remarques importantes :

— On a adjoint un autre fichier otherfile.o qui lui est sous la forme objet, ceci montre les deux effets de gfortran : compilation et link.

— ATTENTION A l'ORDRE pour les bibliotheques. Si le programme a compiler a besoin de routines dans lib.a alors il faut placer lib.a **APRES** file.f
gfortran lib.a file.f -o file donne une erreur
alors que
gfortran file.f lib.a -o file compile et lie correctement.

— Le compilateur gfortran est aussi un compilateur C. Il est donc possible de melanger du C et du fortran.
Exemples :
gfortran -o exec file1.f file2.o file3.c lib.a
gfortran -c file1.f gfortran -c file3.c
gfortran -o exec file1.o file2.o file3.o file4.a

Les options de gfortran

Il y en a une multitude que l'on pourra consulter en faisant 'man gfortran' . Elles sont de la forme '-lettre' . Voici celles qui sont le plus utiles.

— '-c' Effectue une compilation seule, sans link. La pratique Unix est d'effectuer d'abord cette operation sur tous les fichiers puis de creer l'executable avec la commande 'gfortran file.o file2.o...'. C'est la base du makefile (voir plus bas). Cette option permet aussi de verifier la syntaxe d'un code rapidement.

— '-g' Etablit une table de symboles pour le 'debuggage' du code. Voir doc sur gdb le debuggeur de haut niveau sous unix. Attention cette option est en general incompatible avec '-O' qui est l'option d'optimisation.

— '-l' Suivie d'un nom cette option d'edition de lien cherche la bibliothèque libnom.a dans le repertoire indique par l'option -L puis dans /lib et /usr/lib. Cette option est utilisee en general pour les bibliotheques de l'utilisateur.

— '-o' Elle est indiquée ci-dessus et permet de nommer l'executable. Ainsi le resultat de la compilation ci dessus sera un executable nomme 'file'.

— '-O' Optimise le code. Cette option doit etre maniée avec precaution , et en particulier il est bon de comparer les resultats du code avec et sans optimisation. Sur les nouvelles machines -O1 -O2 .. Une des optimisations simple est le deroulement de boucles.

— '-p' Cette option avancée permet d'établir le pourcentage de temps passe dans chaque procédure. Voir amélioration du code.

— '-u' Exige une déclaration explicite pour chaque variable utilisée dans le code. Très pratique pour verifier la syntaxe avant compilation.

Un exécutable = gros fichier multiples informations utilisables après un arrêt erroné (core dump). strip exe

Possible d'utiliser debugger gdb pour obtenir des informations sur la cause de l'arret.

Le debuggeur

gdb (Gnu debugger) debugge gfortran, gcc, g++ (compilateurs GNU)

Procedure

- compiler avec OPT = -g
- lancer gdb avec gdb exe où exe est le nom donné au fichier executable. On se retrouve dans l'environnement gdb (gdb)
- on met un ou plusieurs points d'arret
  (gdb) list (ou (gdb) l) liste 10 lignes de code . (gdb) l 45 liste 10 lignes de code après la ligne 45
  (gdb) break 10 (ou (gdb) b 10) place un point d'arrêt a la ligne 10
  (gdb) break sub place un point d'arrêt dans la subroutine sub
- on lance exe (gdb) run < dexe
  (gdb) stopped at 10 5.
- pas a pas
  (gdb) step (entre dans fonctions, sous-programmes)
  (gdb) next (saute fonctions, sous-programmes)
- visualisation de variables (
  gdb) print a

Autres commandes

- liste des points d'arret (gdb) (gdb) info break
- (gdb) del 2 (detruit break No 2)
- visualisation tableau a(10,2)
  (gdb) print ∗(&a(1, 1)@4) affiche 4 premiers elements de a
- (gdb) cont saute d'un break au break suivant
- (gdb) where où on est dans le programme

Makefile

Le makefile permet de compiler rapidemet et simplement. Il se construit de la façon suivante

```
nomCompilation : fichier1.f fichier2.f #nom des dépendances
                 gfortran -g fichier1.f fichier2.f -o exe
```

Dans la pratique, on compile toute les sub et fonction jusqu'à l'objet, puis on compile les objets ensembles pour former l'executable.

Exemple

```
FC = gfortran
OPT = -g

#executable
main : Mafonction1.o Mafonction2.o Masub.o modMedDonnées.mod
        $(FC) $(OPT) Mafonction1.o Mafonction2.o Masub.o \
        modMedDonnées.mod -o MonExe

#objet
MaFonction1.o : Mafonction1.f modMesDonnées.mod
        $(FC) $(OPT) -c Mafonction1.f
MaFonction2.o : Mafonction2.f modMesDonnées.mod
        $(FC) $(OPT) -c Mafonction2.f
MaSub.o : Masub.f modMesDonnées.mod
        $(FC) $(OPT) -c Masub.f
#module
modMesDonnées.mod : modMesDonnées.f
        $(FC) $(OPT) -c modMesDonnées.f
```

Cinquième partie

Les Entrées/Sortie Linux

Le Grep

Grep est un programme de recherche d'occurence dans un fichier.

```
grep read jac.f
```

donneras toutes les lignes de codes où le mot read apparait.

Fichier de données

Lorsqu'un programme lit des données , on peut se passer de les lui donner directement sur le terminal mais plutôt à partir d'un fichier. Il faut faire attention à l'ordre de lecture. C'est pour cela que le grep read est interessant à utiliser. Pour envoyer les données dans le programme on procède de la façon suivante :

```
$ monExe < mesDonnéesEntrées
```

On peut faire de même en sortie si le programme écrit des données

```
$ monExe < mesDonnéesEntrées >mesDonnéesSorties
```

Si le fichier n'existe pas en sortie, celui ci est crée.

Le pipe

| sert de redirection (tunnel) entre diffèrents programmes