PLS4SNP
Guide de l'Utilisateur

22 mars 2013

Table des matières

Introduction

PLS4SNP implémente des fonctionnalités pour l'évaluation génomique utilisant sur les régressions PLS/Sparse-PLS décrites dans la thèse de **Carine Colombani** (*Modèle de prédiction pour l'évaluation génomique des bovins laitiers*). La régression PLS (*Partial Least Square regression* ou *Projection to Latente Structure*) repose sur la construction de variables latentes, qui sont des combinaisons des variables explicatives (SNP), associées à des poids appelés vecteurs *loadings*. Les analyses implémentées ne sont pas multivariées (Chaque phénotype est analysé indépendamment).
Cette application a été implémenté dans le cadre du projet ANR UtOpIGe : "Vers une Utilisation Optimale de l'information Génomique dans les schémas pyramidaux".

1 Installation

1.1 Pré-requis

- Les compilateurs de la GNU Compiler Collection : gfortran/gcc/g++
- La suite GNU autotools
- Librairie LAPACK (Linear Algebra PACKage) sous license BSD (licence libre)

1.2 Compilation

Si la librairie LAPACK n'est pas installée dans /usr/lib :

```
export LDFLAGS=-L<path-lapack>
```

Pour compiler et installer le programme dans le répertoire *<path-for-pls4snp>* :

```
./configure --prefix=<path-for-pls4snp>;make;make install
```

Pour obtenir un calcul en simple précision :

```
./configure --disable-double ;make;make install
```

Pour le mode debug :

```
./configure --enable-debug ;make;make install
```

1.3 Arborescence du package PLS4SNP

Description du contenu du répertoire d'installation de PLS4SNP :

- **bin/pls4snp** : L'exécutable.
- **lib/libpls4snp.a** : La librairie pour la programmation fortran ou C/C++.
- **include/pls4snp.hpp** : Le prototypage des fonctions pour la programmation C/C++.
- **share/pls4snp/doc/pls4snp.pdf** : Cette documentation.
- **share/pls4snp/README** : Les informations sur les exemples de programmation et le jeux de donnée (QTLMas 2011).

- **share/pls4snp/Fortran** : Le répertoire contenant un exemple de programme fortran.
- **share/pls4snp/C** : Le répertoire contenant un exemple de programme C.
- **share/pls4snp/typage_ap** : Le fichier de typage d'une population de référence (1500 individus).
- **share/pls4snp/typage_val** : Le fichier de typage d'une population de validation (500 individus).
- **share/pls4snp/pheno_ap.txt** : Le fichier de performance de la population de référence.
- **share/pls4snp/pheno_val.txt** : Le fichier de performance de la population de validation.
- **share/pls4snp/typage_pred** : Le fichier de typage d'une population à prédire (1000 individus).
- **share/pls4snp/tbv.txt** : La prédiction des individus à prédire pour comparer les résultats.

2 Format des fichiers utilisateurs

Le programme PLS4SNP prend comme donnée d'entrée deux types de fichiers : Les fichiers de génotype de phénotype. Les identifiants des animaux peuvent être en alpha-numérique. L'ordre de construction du fichier n'est pas imposé.

2.1 Le fichier des génotypes

Le fichier des données de génotype contient l'information au marqueur pour chacun des descendants. Le programme accepte plusieurs formats d'entrée. Ces formats renseignent l'identifiant de l'individu en première colonne (alpha-numérique). Par défaut, ces fichiers ne contiennent pas d'entête. Une option peut être donné en argument pour spécifier la présence d'une entête éventuelle.

2.1.1 Le format AIPL_SPACE

Ce format est codé avec 4 valeurs entières. Une valeur est associée à chaque marqueur :
- `0` pour le statut homozygote sur le premier allèle
- `1` le statut hétérozygote,
- `2` le statut homozygote sur le deuxième allèle.
- `5` le status "valeur manquante".

Les valeurs de chaque marqueurs sont séparées par des espaces.

```
ID1 1 1 2 1 1 2 1 1
ID2 0 1 1 1 2 1 0 2
```

2.1.2 Le format AIPL

Ce format utilise le même codage. Le génotype d'un individu est codé sur une chaîne de caractère.

```
ID1 11211211
ID2 01112102
```

2.1.3 Le format UGA

Le format UGA est similaire l'AIPL. Le nombre de marqueurs est ajouté en deuxième colonne.

```
ID1 8 11211211
ID2 8 01112102
```

2.1.4 Le format MPE

Le format MPE renseigne l'information de l'allèle 1 et l'allèle 2 pour chacun des marqueurs. Les valeurs possibles sont 1 ou 2. Les valeurs manquantes prennent la valeur 0.

```
ID1 1 1 1 2 1 1 2 2 1 1 1 1 2 2
ID2 1 1 1 2 1 2 1 1 1 1 1 1 1 2
```

2.2 Le fichier des performances

Le fichier de performance contient une entête et une suite d'enregistrement de deux ou trois colonnes pour chaque individu. L'entête est constituée du nom de l'EDC [1], si celui-ci est présent, et du phénotype considéré.

exemple sans EDC

```
PHENO1
ID1     30.5
ID2     24.2
ID3     32.5
ID4     36.7
```

exemple avec EDC

```
EDC1 PHENO1
ID1     23.2 30.5
ID2     22.5 24.2
ID3     25.3 32.5
ID4     21.8 36.7
```

2.3 La gestion des données manquantes

Le fichier de phénotype ne peut pas contenir de valeur manquante. Le fichier de génotype peut contenir des valeurs manquantes qui seront remplacées par la moyenne des valeurs des individus génotypés pour ce marqueur.

1. Effective Daughter Contribution

3 Sparse-PLS / PLS

3.1 Description

Cette analyse construit les *ncomp* variables latentes et enregistre le résultat dans un fichier `pls.bin` ou `spls-<taux>.bin`. Ce binaire peut être préfixé avec l'option `--bin-prefix`.

3.2 Arguments en entrée du programme

- `--pls` ou `--spls <real>` : Le choix de la régression réaliser. L'option `spls` prend en entrée la proportion (entre 0 et 1) de variables explicatives (SNP) à garder dans le modèle (intensité de sélection). Le nombre de variables à sélectionner est le même sur chacune des variables latentes.
- `--geno <pathfile>` : Le fichier contenant le statut des individus de la population de référence pour chaque SNP.
- `--format-geno <format>` : Le format du fichier de génotype : `aipl_space|aipl|mpe|uga`.
- `--pheno <pathfile>` : Le fichier contenant les performances des individus de la population de référence.
- `--ncomp <integer>` : Le nombre de variables latentes à construire.

3.2.1 Arguments optionnels

- `--bin-prefix` : Préfixe la génération du binaire contenant les résultats.
- `--edc` : Indique si l'information de l'EDC est disponible. Cet option implique la prise en compte de l'EDC dans le calcul.
- `--not-use-edc` : Annule la prise en compte de l'EDC dans le calcul si l'option `--edc` est présente.
- `--header-geno` : Indique si le fichier de génotype contient une entête.
- `--maxiter <integer>` : Donne le nombre d'itération maximum pour la convergence du calcul d'une variable latente.
- `--tol <real>` : Donne le seuil de tolérance pour la convergence du calcul d'une variable latente.
- `--remove-minmaf <real>` : Supprime de l'analyse les marqueurs dont la MAF est inférieure à la MAF minimale indique.

3.3 Exemple

Calcul de l'analyse PLS sur un fichier génotype au format AIPL et sur un fichier de phénotype contenant une colonne EDC :

```
>pls4snp --geno GENO1_AIPL.txt --format-geno aipl --pheno PHENO_TP_ap.txt --pls --ncomp 50 --edc
```

Calcul de l'analyse Sparse-PLS (selection de variable 5%) sans prise en compte des EDC, sur un fichier de genotype au format AIPL et sur un fichier de phénotype contenant une colonne EDC :

```
>pls4snp --geno GENO1_AIPL.txt --format-geno aipl --pheno PHENO_TP_ap.txt --spls 0.05 --ncomp 50 --edc --not-use-edc
```

4 Validation

4.1 Description

Cette étape calcul la prédiction des phénotypes pour chaque ajout d'une variable latente dans le modèle ($h <= ncomp$). Pour chaque h, une corrélation est calculée entre les phénotypes réels et prédits.

4.2 Données en entrée

- `--valid` : Exécute la validation.
- `--geno <pathfile>` : Le fichier contenant le statut des individus de la population de validation pour chaque SNP.
- `--format-geno <format>` : Le format du fichier de génotype : `aipl_space|aipl|mpe|uga`.
- `--pheno <pathfile>` : Le fichier contenant les performances des individus de la population de validation.
- `--binary <file.bin>` : Le fichier binaire résultat de l'analyse PLS ou Sparse-PLS.

4.2.1 Arguments optionnels

- `--edc` : Indique si les EDC sont dans le fichier de performance. Cet option implique la prise en compte de l'EDC dans le calcul.
- `--not-use-edc` : Annule la prise en compte de l'EDC dans le calcul si l'option `--edc` est présente.
- `--header-geno` : Indique si le fichier de génotype contient une entête.

4.3 Exemple

Exécute la validation sur un fichier de génotype au format AIPL et sur un fichier de phénotype contenant des EDC. Cette validation porte sur le résultat enregistré dans le fichier *pls.bin* résultant d'une analyse PLS/Sparse-PLS.

```
>pls4snp --geno GENO_CAND_AIPL.txt --format-geno aipl --pheno PHENO_TP_val.txt --valid --binary pls.bin --edc
```

5 Calcul des VIP (Variable Importance in the Projection)

5.1 Description

Un critère permettant de rendre compte des effets est le coefficient **VIP**. Ce coefficient permet de juger de l'importance d'une variable explicative dans la construction du modèle de régression PLS.

5.2 Données en entrée

- `--vip` : Exécute le calcul des VIP.

- `--binary <file.bin>` : Le fichier binaire résultat de l'analyse PLS/Sparse-PLS.
- `--H <integer>` : L'indice maximum des variables latentes utilisées pour le calcul des VIP.
- `--out <filepath>` : Un fichier résultat au format text contenant la valeur des VIP.

5.3 Exemple

Execute le calcul des VIP à partir du fichier binaire pls.bin généré par une analyse PLS.

```
>pls4snp --vip --binary pls.bin --H 49 --out vip.txt
```

5.4 Sortie

Le fichier résultat contient une valeur VIP pour chaque variable latente (en colonne) et chaque SNP (en ligne). Il est possible d'obtenir un diagramme en utilisant R :

```
R -e "png(file='vip.png',width=800,res=120);\
VIP<-read.table('vip.txt',header=F,na.strings="-9999");plot(abs(VIP[,49]))"
```

6 Prédiction des phénotypes

6.1 Description

6.2 Données en entrée

- `--predict` : Exécute la prédiction des phénotypes des individus génotypés.
- `--geno <pathfile>` : Le fichier contenant le statut des individus à prédire pour chaque SNP.
- `--format-geno <format>` : Le format du fichier de génotype : `aipl_space|aipl|mpe|uga`.
- `--binary <file.bin>` : Le fichier binaire résultat de l'analyse PLS/Sparse-PLS.
- `--H <integer>` : L'indice optimal des variables latentes utilisées pour le calcul des prédictions.
- `--out <filepath>` : Un fichier résultat au format text contenant la valeur des prédictions pour chaque individus.

6.2.1 Arguments optionnels

- `--header-geno` : Indique si le fichier de génotype contient une entête.

6.3 Exemple

Exécute une prédiction des phénotypes à partir d'un fichier de génotype au format AIPL et utilisant le résultat d'une analyse PLS enregistré dans le fichier pls.bin.

```
>pls4snp --geno GENO_CAND_AIPL.txt --format-geno aipl --predict --binary pls.bin --H 49 --out ebv.txt
```

6.4 Sortie

Le fichier résultat contient une première colonne contenant les ID des animaux et une deuxième colonne contenant la prédiction.

7 Analyse du jeux de donne QTLMas 2011

7.1 Description

Ce jeux de données a été généré avec logiciel LDSO [2] pour le workshop QTLMAS 2011(Rennes). Une population "outbreed" est simulée sur 1000 générations de 1000 individus suivies de 30 générations de 150 individus. 9990 marqueurs SNP sont distribués sur cinq chromosomes. Chaque chromosome a une longueur de 1 Morgan et porte 1998 marqueurs uniformément distribués (1 SNP tous les 0.05 cM). Les données correspondent à la dernière génération de ce pédigré, à savoir, 3000 individus issues de 20 pères accouplés à 10 dames chacun.
Ce jeux de donnée et les résultats (position des QTL, True Breeding Value) sont librement téléchargeables sur le site du workshop QTLMas 2011 et accessible dans le répertoire */share/pls4snp/examples/* .

- `typage_ap` : Le fichier de génotype de la population de référence (1500 individus) au format AIPL.
- `typage_val` : Le fichier de génotype de la population de validation (500 individus) au format AIPL.
- `pheno_ap.txt` : Le fichier de performance de la population de référence.
- `pheno_val.txt` : Le fichier de performance de la population de validation.
- `typage_pred` : Le fichier de génotype des animaux à prédire (1000 individus).
- `tbv.txt` : Les prédictions QTLMas 2011.

PLS/Sparse-PLS (10%) en prenant en compte seulement les marqueurs ayant une MAF supérieure à 3%

```
>pls4snp --pheno pheno_ap.txt --geno typage_ap --format-geno aipl --pls --ncomp 10 --remove-minmaf 0.03
>pls4snp --pheno pheno_ap.txt --geno typage_ap --format-geno aipl --spls 0.1 --ncomp 10 --remove-minmaf 0.03
```

Étape de Validation

```
>pls4snp --pheno pheno_val.txt --geno typage_val --format-geno aipl --valid --binary pls.bin

 == correlation between observed and predicted values of PHENO ==
h=1 c=0.384665
h=2 c=0.425269
h=3 c=0.422542
h=4 c=0.432261
h=5 c=0.433467
h=6 c=0.433488
h=7 c=0.449797
h=8 c=0.432463
h=9 c=0.423787
h=10 c=0.410914
hmax=7 corr=0.449797

>pls4snp --pheno pheno_val.txt --geno typage_val --format-geno aipl --valid --binary spls_0.1.bin

 == correlation between observed and predicted values of PHENO ==
h=1 c=0.479853
h=2 c=0.524215
h=3 c=0.48479
h=4 c=0.484273
```

2. Ytournel, F. (2008). Linkage disequilibrium and QTL fine mapping in a selected population. PhD thesis, Station de Génétique Quantitative et Appliquée, INRA.

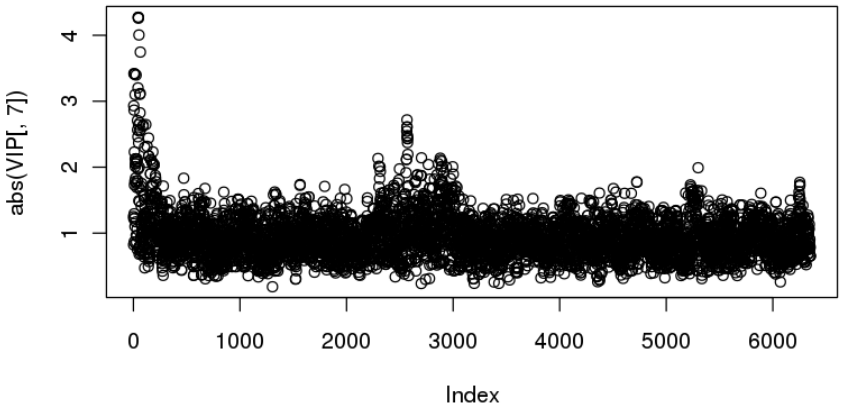

FIGURE 1 – VIP résultant d'une PLS sur le jeux de donnée QTLMas 2011

```
h=5 c=0.483326
h=6 c=0.473774
h=7 c=0.449812
h=8 c=0.435191
h=9 c=0.424595
h=10 c=0.421762
hmax=2 corr=0.524215
```

Calcul des VIP et génération des graphiques.

```
>pls4snp --vip --binary pls.bin --H 7 --out vip_pls.txt
>R -e "png(file='vip_pls.txt.png',width=800,res=120);\
VIP<-read.table('vip_pls.txt',header=F,na.strings="-9999");plot(abs(VIP[,7]))"

>pls4snp --vip --binary spls_0.1.bin --H 2 --out vip_spls.txt
>R -e "png(file='vip_spls.txt.png',width=800,res=120);\
VIP<-read.table('vip_spls.txt',header=F,na.strings="-9999");plot(abs(VIP[,2]))"
```

Calcul des prédictions sur les individus du fichier `typage_pred}`.

```
>pls4snp --geno typage_pred --format-geno aipl --predict --binary pls.bin --H 7 --out predicty_pls.txt
>pls4snp --geno typage_pred --format-geno aipl --predict --binary spls_0.1.bin --H 2 --out predicty_spls01.txt
```

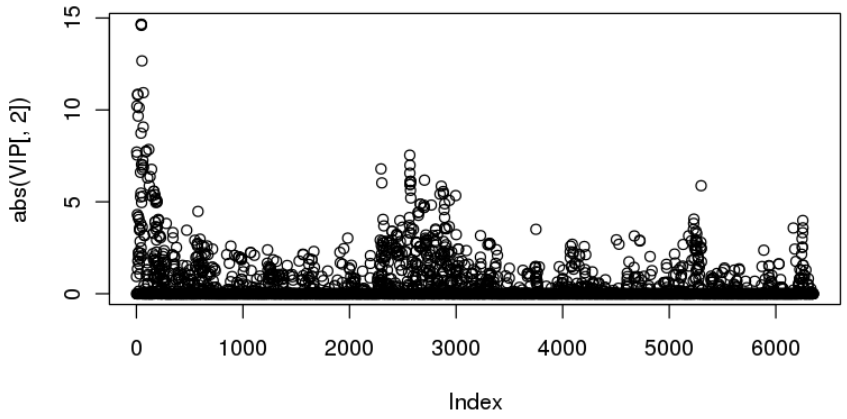

FIGURE 2 – VIP résultant d'une Sparse-PLS (10%) sur le jeux de donnée QTLMas 2011

8 Interface de programmation

Il est possible d'appeler les routines de construction des variables latentes, de prédiction et de calcul des VIP par programmation en langage Fortran et C++.

- `spls` : Construction des variables latentes et des coefficients associés par la PLS/Sparse-PLS.
- `predict` : Prédiction des phénotypes partir des informations aux marqueurs et des variables latentes et des coefficients associés générés par la PLS/Sparse-PLS.
- `calcul_vip` : Calcul des VIP partir des informations aux marqueurs et des variables latentes et des coefficients associés générés par la PLS/Sparse-PLS.

8.1 Interface C/C++/Fortran

Les prototypes des routines sont définis dans le fichier include *<path-pls4snp>/pls4snp.hpp*. La compilation nécessité la librairie *libpls4snp.a* généré à l'installation du package PLS4SNP. Pour compiler un programme fortran, il est nécessaire d'associer à l'édition de lien l'environnement C++.

Compilation du programme *pls.f90* La ligne de commande pour compiler le programme exemple *pls.f90* :

```
>gfortran -o exe pls.f90 -L<path-libpls4snp> -L<path-lib-lapack> -lpls4snp -lstdc++ -llapack
```

Compilation du programme *pls.cpp* La ligne de commande pour compiler le programme exemple *pls.cpp* :

```
>g++ -D DT=double -o exe pls.cpp -I<path-pls4snp>/include -L<path-pls4snp>/lib -lpls4snp -llapack
```

Le prototype des fonctions spls, predict et calcul_vip décrit dans *include/pls4snp.hpp*.

```
/*
 * Interface to call pls/spls from a programme C/C++/fortran
 *
 * Purpose:
 *
 * Parameters:
 *   [in]  n           : specifies number of row of Z and Y
 *   [in]  p           : specifies number of SNP (column) of Z
 *   [in]  Z[n,p]      : incidence matrix
 *   [in]  Y[n]        : performance vector
 *   [in]  edc_pres    : 0 if EDC values are available otherwise >0.
 *   [in]  EDC[n]      : EDC vector
 *   [in]  ncomp       : specifies number of latent variable to build
 *   [in]  maxiter     : maximum number of iteration for convergence
 *   [in]  tol         : criteria for convergence
 *   [in]  keepX       : number of SNP to keep (Sparse PLS option), -1 to perform a PLS.
 *   [out] meanY       : mean of Y
 *   [out] sigmY       : standard deviation of Y
 *   [out] meanZ[p]    : vector of mean of each SNP column
 *   [out] sigmZ[p]    : vector of standard deviation of each SNP column
 *   [out] t[n,ncomp]  : latent variable associated to X
 *   [out] u[n,ncomp]  : latent variable associated to Y
 *   [out] a[p,ncomp]  : loading vectors associated to X
 *   [out] b[ncomp]    : loading vectors associated to Y
 *   [out] c[p,ncomp]  : buffer useful for the pedict subroutine
 */

extern "C" void spls_(int * n, int * p, DT * Z, DT * Y, int * edc_pres, DT * EDC,
                      int * ncomp, int * maxiter, DT * tol,int * keepX,
             DT * meanY, DT * sigmY, DT * meanZ, DT * sigmZ,
                      DT * t, DT * u, DT * a, DT * b, DT * c )

/*
 * Interface to call calcul_vip from a programme C/C++/fortran
 *
 * Purpose:
 *
 * Parameters:
 *   [in]  ncomp          : specifies number of latent variable associated with t and a
 *   [in]  hcomp          : specifies number of latent variable to include
 *   [in]  n              : specifies number of row of Y
 *   [in]  p              : specifies number of SNP
 *   [in]  Y[n]           : performance vector
 *   [in]  t[n,ncomp]     : latent variable associated to X
 *   [in]  a[p,ncomp   ]  : loading vectors associated to X
 *   [out] VIP[p,ncomp]   : VIP results
 */

extern "C" void  calcul_vip_(int *ncomp,int * hcomp,int *n,int *p, DT* Y, DT * t,DT *a,DT *VIP);

/*
 * Interface to call predict from a programme C/C++/fortran
 *
 * Purpose:
 *
 * Parameters:
 *   [in]  ncomp               : specifies number of latent variable associated with t and a
 *   [in]  n                   : specifies number of row of Y
 *   [in]  p                   : specifies number of SNP
 *   [in]  a[p,ncomp]          : loading vectors associated to X
 *   [in]  c[p,ncomp]          : buffer result from spls subroutine
 *   [in]  b[ncomp]            : loading vectors associated to Y
 *   [in]  meanY               : mean of Y
 *   [in]  sigmY               : standard deviation of Y
 *   [in]  meanZ[p]            : vector of mean of each SNP column
 *   [in]  sigmZ[p]            : vector of standard deviation of each SNP column
 *   [in]  n_pred              : specifies number of row of Z_pred
 *   [in]  Z_pred[n_pred,p]    : incidence matrix to predict
 *   [in]  hcomp               : specifies the number of latente variate to compute yhat,tpred,bhat
 *   [out] yhat[n_pred,hcomp]  : Y prediction
 *   [out] tpred[n_pred,hcomp]:
 *   [out] bhat[p,hcomp]       :
 */

extern "C" void predict_(int *n, int *p, int *ncomp, DT * a, DT* c, DT * b,
                         DT * meanY, DT * sigmaY, DT * meanZ, DT* sigmaZ,
                         int * n_pred, DT * Z_pred,int * hcomp,DT * yhat,DT * tpred,DT * bhat);
```

L'interface de définition des routines PLS4SNP dans un programme fortran.

```
interface
   subroutine spls(n, p,Z,Y,edc_present,EDC,ncomp,maxiter,tol,keepX,meanY,sigmY,meanZ,sigmZ,t,u,a,b,c)
     integer ,intent(in)                          :: n
     integer ,intent(in)                          :: p
     double precision,dimension(n,p),intent(in) :: Z
     double precision,dimension(n),intent(in)   :: Y
     integer ,intent(in)                          :: edc_present
     double precision,dimension(n),intent(in)   :: EDC
     integer ,intent(in)                          :: ncomp
     integer ,intent(in)                          :: maxiter
     double precision ,intent(in)                 :: tol
     integer ,intent(in)                          :: keepX
     double precision ,intent(out)                :: meanY
     double precision ,intent(out)                :: sigmY
     double precision,dimension(p),intent(out)  :: meanZ
     double precision,dimension(p),intent(out)  :: sigmZ
     double precision,dimension(n,ncomp),intent(out)  :: t
     double precision,dimension(n,ncomp),intent(out)  :: u
     double precision,dimension(p,ncomp),intent(out)  :: a
     double precision,dimension(ncomp),intent(out)    :: b
     double precision,dimension(p,ncomp),intent(out)  :: c
   end subroutine spls

   subroutine calcul_vip(ncomp,hcomp,n, p,Y,t,a,VIP)
     integer ,intent(in)                              :: ncomp
     integer ,intent(in)                              :: hcomp
     integer ,intent(in)                              :: n
     integer ,intent(in)                              :: p
     double precision,dimension(n),intent(in)         :: Y
     double precision,dimension(n,ncomp),intent(in)   :: t
     double precision,dimension(p,ncomp),intent(in)   :: a
     double precision,dimension(p,hcomp),intent(out)  :: VIP
   end subroutine calcul_vip

   subroutine predict(n,p,ncomp,a,c,b,meanY,sigmaY,meanZ,sigmaZ,n_pred,Z_pred,hcomp,yhat,tpred,bhat)
     integer ,intent(in)                                  :: n
     integer ,intent(in)                                  :: p
     integer ,intent(in)                                  :: ncomp
     double precision,dimension(p,ncomp),intent(in)       :: a
     double precision,dimension(p,ncomp),intent(in)       :: c
     double precision,dimension(ncomp),intent(in)         :: b
     double precision             ,intent(in)             :: meanY
     double precision             ,intent(in)             :: sigmaY
     double precision,dimension(p)      ,intent(in)       :: meanZ
     double precision,dimension(p)      ,intent(in)       :: sigmaZ
     integer ,intent(in)                                  :: n_pred
     double precision,dimension(n_pred,p) ,intent(in)     :: Z_pred
     integer ,intent(in)                                  :: hcomp
     double precision,dimension(n_pred,hcomp),intent(out) :: yhat
     double precision,dimension(n_pred,hcomp),intent(out) :: tpred
     double precision,dimension(p,hcomp),intent(out)      :: bhat
    end subroutine predict

  end interface
```

8.2 Exemple de programme Fortran 90 et C++

Le programme applique une PLS sur la population de référence. Puis un calcul de corrélation entre les phénotypes prédits et réels de la population de validation est affiché pour chaque ajout de variable latente dans le modèle. En fin de programme, un fichier contenant les VIP est généré en utilisant l'indice H optimal (correspondant à la meilleure corrélation trouvée lors de l'étape de validation) ainsi que le calcul des prédictions des animaux génotypés dans le fichier `typage_pred`.

Le programme pls.f90.

```
program pls_prog
    implicit none

    interface
            subroutine spls(n, p,Z,Y,edc_present,EDC,ncomp,maxiter,tol,keepX,meanY,sigmY,meanZ,sigmZ,t,u,a,b,c)
               integer ,intent(in)                          :: n
               integer ,intent(in)                          :: p
               double precision,dimension(n,p),intent(in) :: Z
               double precision,dimension(n),intent(in)   :: Y
               integer ,intent(in)                          :: edc_present
               double precision,dimension(n),intent(in)   :: EDC
               integer ,intent(in)                          :: ncomp
               integer ,intent(in)                          :: maxiter
               double precision ,intent(in)                 :: tol
               integer ,intent(in)                          :: keepX
               double precision ,intent(out)                :: meanY
               double precision ,intent(out)                :: sigmY
               double precision,dimension(p),intent(out)  :: meanZ
               double precision,dimension(p),intent(out)  :: sigmZ
               double precision,dimension(n,ncomp),intent(out)  :: t
               double precision,dimension(n,ncomp),intent(out)  :: u
               double precision,dimension(p,ncomp),intent(out)  :: a
               double precision,dimension(ncomp),intent(out)    :: b
               double precision,dimension(p,ncomp),intent(out)  :: c
            end subroutine spls

           subroutine calcul_vip(ncomp,hcomp,n, p,Y,t,a,VIP)
              integer ,intent(in)                                 :: ncomp
              integer ,intent(in)                                 :: hcomp
              integer ,intent(in)                                 :: n
              integer ,intent(in)                                 :: p
              double precision,dimension(n),intent(in)          :: Y
              double precision,dimension(n,ncomp),intent(in)    :: t
              double precision,dimension(p,ncomp),intent(in)    :: a
              double precision,dimension(p,hcomp),intent(out)   :: VIP
           end subroutine calcul_vip

           subroutine predict(n,p,ncomp,a,c,b,meanY,sigmaY,meanZ,sigmaZ,n_pred,Z_pred,hcomp,yhat,tpred,bhat)
              integer ,intent(in)                                 :: n
              integer ,intent(in)                                 :: p
              integer ,intent(in)                                 :: ncomp
              double precision,dimension(p,ncomp),intent(in)    :: a
              double precision,dimension(p,ncomp),intent(in)    :: c
              double precision,dimension(ncomp),intent(in)      :: b
              double precision             ,intent(in)          :: meanY
              double precision             ,intent(in)          :: sigmaY
              double precision,dimension(p)       ,intent(in)   :: meanZ
              double precision,dimension(p)       ,intent(in)   :: sigmaZ
              integer ,intent(in)                                 :: n_pred
              double precision,dimension(n_pred,p) ,intent(in) :: Z_pred
              integer ,intent(in)                                 :: hcomp
              double precision,dimension(n_pred,hcomp),intent(out)  :: yhat
              double precision,dimension(n_pred,hcomp),intent(out)  :: tpred
              double precision,dimension(p,hcomp),intent(out)       :: bhat
           end subroutine predict

   end interface

   integer             :: hmax
   character(len=9990) :: typeid
   character(len=30)   :: file
   integer             :: i,j,h,ios,n_ap,n_val,n_pred,p,keepx,maxiter,ncomp,edc_present
   ! DATA
   double precision  ,dimension(:,:), allocatable  :: Z_ap,Z_val,Z_pred
   double precision  ,dimension(:)  , allocatable  :: Y_ap,Y_val,Y_tbv
   double precision                                :: val,tol,meanY,sigmY
   ! Results
   double precision  ,dimension(:,:), allocatable  :: t,u,a,c
   double precision  ,dimension(:)  , allocatable  :: b,meanZ,sigmZ,EDC
   double precision  ,dimension(:,:), allocatable  :: yhat,tpred,bhat,VIP
   double precision :: meanYVal,meanYpred,cor,cormax
```

```
72    double precision  ,dimension(:)  , allocatable  :: AA,BB
73
74    ncomp = 10      ! number of latent variate to build
75    maxiter = 20    ! maxiter criteria
76    tol = 1.d-5     ! tolerance for pls
77    keepx = -1      ! PLS so....
78    edc_present = 1 ! no edc
79    n_ap = 1500     ! size reference population
80    n_val = 500     ! size validation population
81    n_pred = 1000   ! size predict population
82    p = 9990        ! number of SNP
83
84    allocate (Z_ap(n_ap,p))
85    allocate (Y_ap(n_ap))
86    allocate (Z_val(n_val,p))
87    allocate (Y_val(n_val))
88
89    file='../typage_ap';call read_Z(file,n_ap,p,Z_ap)
90    file="../pheno_ap.txt";call read_Y(file,n_ap,Y_ap)
91    file='../typage_val';call read_Z(file,n_val,p,Z_val)
92    file="../pheno_val.txt";call read_Y(file,n_val,Y_val)
93
94    ! Allocate results structures
95    allocate(t(n_ap,ncomp),u(n_ap,ncomp),a(p,ncomp),c(p,ncomp),b(ncomp))
96    allocate (meanZ(p),sigmZ(p))
97
98    !Call lib PLS4SNP => spls
99    call spls(n_ap,p,Z_ap,Y_ap,edc_present,EDC,ncomp,maxiter,tol,keepX,meanY,sigmY,meanZ,sigmZ,t,u,a,b,c)
100
101   !Allocate prediction results
102   allocate(yhat(n_val,ncomp),tpred(n_val,ncomp),bhat(p,ncomp))
103   !Call lib PLS4SNP => predict
104   call predict(n_ap,p,ncomp,a,c,b,meanY,sigmY,meanZ,sigmZ,n_val,Z_val,ncomp,yhat,tpred,bhat)
105
106   allocate (AA(n_val),BB(n_val))
107   meanYVal = sum(Y_val) / n_val
108   AA = Y_val - meanYVal
109   cormax=0;hmax=0;
110   do h=1,ncomp
111     meanYpred = sum(yhat(:,h)) / n_val
112     print *,meanYpred
113     BB = yhat(:,h) - meanYpred
114     cor = sum(AA*BB) / sqrt(sum(AA*AA)*sum(BB*BB))
115     print *,"H=",h," correlation=",cor
116     if (cormax<abs(cor)) then
117      cormax = cor
118      hmax = h
119     end if
120   end do
121
122   deallocate (AA,BB)
123
124   print *,"hmax=",hmax
125   allocate (VIP(p,hmax))
126   !Call lib PLS4SNP => calcul_vip
127   call calcul_vip(ncomp,hmax,n_ap,p,Y_ap,t,a,VIP)
128
129   print *," == Print VIP in file : vip.txt =="
130   open(unit=20,file="vip.txt")
131   do i=1,p
132     write(20,fmt="(f7.5)") VIP(i,hmax)
133   end do
134   close(20)
135
136
137   deallocate(VIP)
138   deallocate(yhat,tpred,bhat)
139
140   allocate (Z_pred(n_pred,p))
141   allocate (Y_tbv(n_pred))
142
143   file='../typage_pred';call read_Z(file,n_pred,p,Z_pred)
144   file="../tbv.txt";call read_Y(file,n_pred,Y_tbv)
145
146   !Allocate prediction results
147   allocate(yhat(n_pred,hmax),tpred(n_pred,hmax),bhat(p,hmax))
148   !Call lib PLS4SNP => predict
149   call predict(n_ap,p,ncomp,a,c,b,meanY,sigmY,meanZ,sigmZ,n_pred,Z_pred,hmax,yhat,tpred,bhat)
150   !read true breading value and compute correlation
151   allocate (AA(n_pred),BB(n_pred))
152   meanYVal = sum(Y_tbv) / n_pred
153   AA = Y_tbv - meanYVal
154   meanYpred = sum(yhat(:,hmax)) / n_pred
155   BB = yhat(:,hmax) - meanYpred
156   cor = sum(AA*BB) / sqrt(sum(AA*AA)*sum(BB*BB))
157   print *,"Correlation between true breading value and prediction:",cor
158
159   deallocate (AA,BB)
160   deallocate (Z_ap,Y_ap,Z_val,Y_val,Z_pred)
161   deallocate(yhat,tpred,bhat)
162   deallocate (meanZ,sigmZ)
163   deallocate(t,u,a,c,b)
```

```
164
165     contains
166
167     subroutine read_Z(file,n,p,Z)
168        character(len=30) ,intent(in)    :: file
169        integer           ,intent(in)    :: n
170        integer           ,intent(in)    :: p
171        double precision,dimension(n,p),intent(inout) :: Z
172        integer :: ios,j,i,id
173
174        open(unit=19,file=file)
175        ios=0
176        j=0
177        do while( ios == 0 )
178          read (19,*,iostat=ios) id,typeid
179          if ( ios == 0 ) then
180          j=j+1
181          do i=1,len(typeid)
182           Z(j,i) = ichar(typeid(i:i)) - ichar('0')
183          end do
184          end if
185        end do
186        print *,"read ",j," genotypes"
187        close(19)
188
189     end subroutine read_Z
190
191     subroutine read_Y(file,n,Y)
192        character(len=30) ,intent(in)    :: file
193        integer           ,intent(in)    :: n
194        double precision,dimension(n),intent(inout) :: Y
195        integer :: ios,i,id
196
197        open(unit=19,file=file)
198        ios=0
199        i=0
200        read (19,*,iostat=ios) ! header....
201        do while( ios == 0 )
202         read (19,*,iostat=ios) id,val
203         if ( ios == 0 ) then
204           i=i+1
205           Y(i) = val
206         end if
207        end do
208
209        print *,"read ",i," phenotypes - validation "
210     close(19)
211     end subroutine read_Y
212
213   end program pls_prog
```

Le programme pls.cpp.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
using namespace std;

/* include pls4snp */
#include "pls4snp.hpp"

void read_Z(string file, int n,int p, double *Z) {
   string name;
   string snp;
   string line;
   ifstream zfile;
   zfile.open(file.c_str());
   int c=0;
   while ( ! zfile.eof() )
   {
      getline (zfile,line);
      if ( line == "" ) continue;
      istringstream stream1(line);
      stream1 >> name;
      stream1 >> snp;
      for (int i=0;i<p;i++) {
         Z[c+i*n] = snp[i] - '0';
      }
      c++;
   }

   zfile.close();
}

void read_Y(string file, double *Y) {
   string name;
   string snp;
   string line;
   ifstream yfile;
   yfile.open(file.c_str());
   int c=0;

   getline (yfile,line);/* header to remove */

   while ( ! yfile.eof() )
   {
      getline (yfile,line);
      if ( line == "" ) continue;
      istringstream stream1(line);
      stream1 >> name;
      stream1 >> Y[c];
      c++;
   }

   yfile.close();
}


int main( int argc, const char* argv[] )
{
   int ncomp = 10;        // number of latent variables to build
   int p=9990;            // number of marker
   int n_ap = 1500;       // number of reference population
   int n_val=500;         // number of validation population
   int n_pred=1000;       // number of animal to predict
   int maxiter=20;        // max iteration for convergence
   double tol = 0.00001; // criteria convergence
   int keepx = -1;        // if spls, set the number of snp to keep

   double *Z_ap = new double [n_ap*p];
   double *Z_val = new double [n_val*p];
   double *Y_ap = new double [n_ap];
   double *Y_val = new double [n_val];

   read_Z("../typage_ap",n_ap,p,Z_ap);
   read_Z("../typage_val",n_val,p,Z_val);
   read_Y("../pheno_ap.txt",Y_ap);
   read_Y("../pheno_val.txt",Y_val);

   /* init for latent variates and buf struct for pls */
   double * t = new double[n_ap*ncomp];
   double * u = new double[n_ap*ncomp];
   double * a = new double[p*ncomp];
   double * c = new double[p*ncomp];
   double * b = new double[ncomp];
   double * meanZ = new double[p];
   double * sigmZ = new double[p];
```

```
90
91      double meanY,sigmaY;
92      int edc_pres=1; // false
93      double * EDC = NULL ;
94
95      /* Lib PLS4SNP : build Latent Variables with PLS */
96
97      spls_(&n_ap,&p,Z_ap,Y_ap,&edc_pres,EDC,&ncomp,&maxiter,&tol,&keepx,
98            &meanY,&sigmaY,meanZ,sigmZ,t,u,a,b,c);
99
100     double * yhat = new double[n_val*ncomp];
101     double * tpred = new double[n_val*ncomp];
102     double * bhat = new double[p*ncomp];
103
104     /* Lib PLS4SNP : Predict Y for successive combination of latent variables */
105     predict_(&n_ap,&p,&ncomp,a,c,b,&meanY,&sigmaY,meanZ,sigmZ,&n_val,Z_val,&ncomp,yhat,tpred,bhat);
106
107     /** Correlation between YVAL and each YPRED **/
108     double cormax = 0;
109     int hmax = 0;
110
111     /* mean YVAL */
112     double meanYVAL = 0;
113     for (int i=0;i<n_val;i++)
114        meanYVAL += Y_val[i];
115     meanYVAL /= n_val;
116
117     /* Temporary variable for correlation computation */
118     double * aa = new double [n_val];
119     for (int i=0;i<n_val;i++)
120        aa[i] = Y_val[i] - meanYVAL ;
121
122     double * bb = new double [n_val];
123     for (int h=0;h<ncomp;h++) {
124        /* mean YPRED */
125        double meanYPRED = 0;
126        for (int i=0;i<n_val;i++)
127           meanYPRED += yhat[i+h*n_val]; /* CARREFULLY : Fortran access - Column Major Order */
128        meanYPRED /= n_val;
129
130        for (int i=0;i<n_val;i++)
131           bb[i] = yhat[i+h*n_val] - meanYPRED ;
132
133        double saa=0,sab=0,sbb=0;
134        for (int i=0;i<n_val;i++) {
135           saa += aa[i]*aa[i];
136           sab += aa[i]*bb[i];
137           sbb += bb[i]*bb[i];
138        }
139
140        double cor = sab / sqrt(saa*sbb);
141        if (cor > cormax) {
142           cormax = cor;
143           hmax = h;
144        }
145        cout << "H=" << (h+1) << " correlation="<< cor << endl;
146     }
147
148     cout << "Hmax="<< (hmax+1) << endl ;
149     delete[]aa;
150     delete[]bb;
151
152     int hmaxborn=hmax+1;
153     double * VIP = new double [hmaxborn*p];
154
155     /* Lib PLS4SNP : Information about SNP effect : VIP computation */
156     calcul_vip_(&ncomp,&hmaxborn,&n_ap,&p,Y_ap,t,a,VIP);
157
158     cout << " == Print VIP in file : vip.txt ==" << endl ;
159     ofstream vipfile;
160     vipfile.open("vip.txt");
161     vipfile.precision(5);
162
163     for (int i=0;i<p;i++) {
164        vipfile << VIP[i+hmax*p]<< endl ;
165     }
166
167     vipfile.close();
168     delete[] VIP;
169     delete[] yhat;
170     delete[] tpred;
171     delete[] bhat;
172
173     /* Predict Y with hmax */
174
175     double *Z_pred = new double [n_pred*p];
176     double *Y_pred = new double [n_pred];
177
178     read_Z("../typage_pred",n_pred,p,Z_pred);
179     read_Y("../tbv.txt",Y_pred);
180
181     yhat = new double[n_pred*hmaxborn];
```

```
182     tpred = new double[n_pred*hmaxborn];
183     bhat = new double[p*hmaxborn];
184 
185     /* Lib PLS4SNP : Predict Y for successive combination of latent variables */
186     predict_(&n_ap,&p,&ncomp,a,c,b,&meanY,&sigmaY,meanZ,sigmZ,&n_pred,Z_pred,&hmaxborn,yhat,tpred,bhat);
187 
188     /* Solution in the matrix yhat at hmax column */
189 
190     /* Correlation with TBV */
191     /* mean YVAL */
192     meanYVAL = 0;
193     for (int i=0;i<n_pred;i++)
194        meanYVAL += Y_pred[i];
195     meanYVAL /= n_pred;
196 
197     /* Temporary variable for correlation computation */
198     aa = new double [n_pred];
199     bb = new double [n_pred];
200     for (int i=0;i<n_pred;i++)
201        aa[i] = Y_pred[i] - meanYVAL ;
202     /* mean YPRED */
203     double meanYPRED = 0;
204     for (int i=0;i<n_pred;i++)
205        meanYPRED += yhat[i+hmax*n_pred]; /* CARREFULLY : Fortran access - Column Major Order */
206     meanYPRED /= n_pred;
207     for (int i=0;i<n_pred;i++)
208        bb[i] = yhat[i+hmax*n_pred] - meanYPRED ;
209     double saa=0,sab=0,sbb=0;
210     for (int i=0;i<n_pred;i++) {
211        saa += aa[i]*aa[i];
212        sab += aa[i]*bb[i];
213        sbb += bb[i]*bb[i];
214     }
215 
216     double cor = sab / sqrt(saa*sbb);
217 
218     cout << "Correlation between true breading value and prediction:"<< cor << endl ;
219 
220     delete[] yhat;
221     delete[] tpred;
222     delete[] bhat;
223 
224     delete[] meanZ;
225     delete[] sigmZ;
226 
227     delete[] t;
228     delete[] u;
229     delete[] a;
230     delete[] c;
231     delete[] b;
232 
233     delete[] Z_ap;
234     delete[] Z_val;
235     delete[] Y_ap;
236     delete[] Y_val;
237     delete[] Z_pred;
238     delete[] Y_pred;
239 }
```