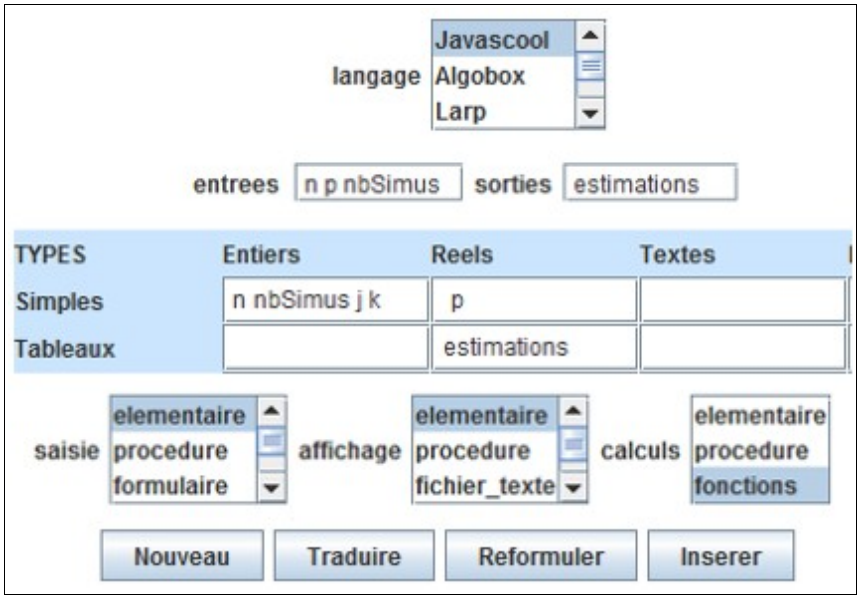
Loi binomiale en Javascool avec PluriAlgo

Ce document reprend en Javascool le problème de synthèse traité en Xcas dans l'article. Quelques adaptations sont nécessaires, notamment parce qu'il faut créer un programme principal.

Création du programme principal

Les entrées du problème sont les paramètres de la loi (n et p) et le nombre de simulations (nbSimus). La sortie est le tableau des estimations, ce qui est précisé dans l'onglet Principal :



La copie d'écran est en fait celle le l'étape 2 de l'article. Mais plutôt que de cliquer sur le bouton **Insérer**, nous cliquerons ici sur le bouton **Nouveau** afin de créer un programme principal :

```
double [] calculer_estimations(int n,double p,int nbSimus) {
      ...
      double [] estimations = new double[10];
      return estimations;
}
void main() {
      ...
      double [] estimations = new double[10];
      ...
      nbSimus = readInt( "nbSimus : " );
      estimations = calculer_estimations(n, p, nbSimus);
      for(i1=0; i1<10; i1++) {
            println( estimations[i1] );
      }
}
```

new double[n+1]

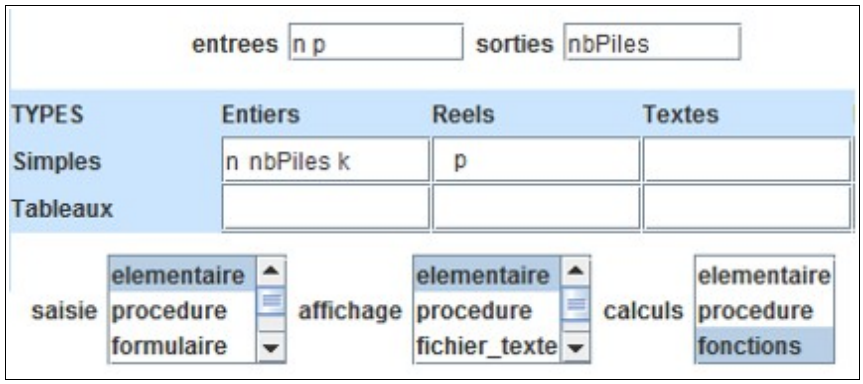
A supprimer (taille gérée dans la fonction)

i1<=n

Quelques corrections sont nécessaires pour gérer la taille du tableau des estimations.

Calcul du nombre de piles

Nous poursuivons par l'étape 1 de l'article. L'onglet Principal permet de créer une fonction comptant le nombre de piles (nbPiles) lors d'une simulation de n lancers d'une pièce ayant la probabilité p de tomber sur pile.



Le code obtenu (en cliquant sur le bouton **Insérer**) doit être ensuite complété (partie entourée en rouge) :

```
int calculer_nbPiles(int n, double p) {
      int k;
      int nbPiles;
      nbPiles = 0;
      for(k=1; k<=n; k=k+1) {
            if ( random()<=p ) {
                  nbPiles = nbPiles+1;
            }
      }
      return nbPiles;
}
```

Le code ajouté calcule le nombre de fois où le réel aléatoire (compris entre 0 et 1) est inférieur ou égal à p (la probabilité d'une pièce de tomber sur pile). Il peut être obtenu en complétant l'onglet Boucles :

Complétion de la fonction calculant les estimations

Il reste à compléter (instructions entourées en rouge) la fonction calculer_estimations, créée en même temps que le programme principal :

```
double [] calculer_estimations(int n, double p, int nbSimus) {
      int j;
      int k;
      double [] estimations = new double[n+1];
      for(k=0; k<=n; k=k+1) {
            estimations[k]=0;
      }
      for(j=1; j<=nbSimus; j=j+1) {
            k = calculer_nbPiles(n, p);
            estimations[k] = estimations[k] + 1.0/nbSimus;
      }
      return estimations;
}
```

PluriAlgo n'est ici d'aucun secours, sauf pour insérer les deux boucles.