Description des codes R développés pour les analyses des données d'échantillonage

Auteur: Laura Tremblay-Boyer, contact: lauratboyer@gmail.com
Nouméa le 25 février 2015

1 À compléter

vérifier format couleurs relecture finale

Contents

2 Survol: une session en exemple

Pour une illustration rapide de l'utilisation des codes, vous trouverez ci-dessous une série de `Commandes R` ainsi que les *explications* pour les arguments donnés. Ces fonctions et arguments sont expliqués en détails dans les sections suivantes.

```
> source(GS_MotherCode_TProj.r)
```
On commence par charger les données et les fonctions sous R.

```
> selection.projet()
```
Sélection du projet à analyser, KNS_KONIAMBO par défaut, voir ***nom.projets()*** *pour les autres options*

```
> facteurs.tempo
[1] "Période.BACI" "Saison"
```
Valeurs disponibles pour l'aggrégation temporelle, voir aussi ***facteurs.spatio*** *et* ***facteurs.taxo****.*

```
> obj1 <- INV.dens.gnrl(fspat="Cote",agtaxo="Famille", filt.camp="A", save=TRUE)
```
Densité moyenne (et ET) par Côte et Campagne (vu que l'argument ***ftemp*** *n'a pas été spécifié, on utilise l'aggrégation temporelle par défaut, voir objet* ***ftempo.defaut****). Le tableau a été sauvegardé dans le dossier KNS_KONIAMBO/Tableaux*

```
> obj2 <- INV.biodiv.gnrl(ftemp=c("Saison","Campagne"),unit.base="St")
```
Shannon, H et d (moyenne et ET), richesse taxonomique (totale, moyenne et ET) par saison et station, en utilisant la station comme unité de base (donc tous transects agrégés) (l'argument ***fspat*** *n'a pas été spécifié, donc l'aggrégation spatiale est sur les stations* ***fspat.defaut****). Mettre* ***ftemp=c('Saison','Campagne')*** *fera l'aggrégation par campagne et saison (mais vu que chaque campagne n'apparaît que dans une saison ça ne fait que rajouter une colonne et les densités ne changent pas),* ***fspat = c('Geomorpho','Cote','St')*** *fera l'aggrégation spatiale par station, côte et géomorphologie,* ***fspat = c('Geomorpho','N_Impact')*** *fera l'aggrégation par géomorphologie et zone d'impact, mais ici les densités seront différentes vu qu'il y a plus qu'une zone d'impact par géomorphologie.*

```
> obj3 <- POIS.dens.gnrl(fspat='Geomorpho', agtaxo='moblabel')
```
Densité, biomasse, taille moyenne, richesse spécifique (et ET) par géomorphologie et catégorie de mobilité. Voir facteurs.taxo$poissons pour les autres options d'aggrégation taxonomique.

```
> facteurs.taxo$LIT
[1] "General" "Forme" "Acroporidae" "Sensibilite" "Famille" "Genre"
```
Types de catégorie LIT disponible pour rafiner les analyses avec l'argument ***LIT.cat***

```
> obj4 <- LIT.couvrtr.gnrl(LIT.cat='Acroporidae')
```
*Couverture (moyenne et ET) des Acroporidae/non-Acroporidae sur les transects LIT par station et campagne (****fspat*** *et* ***ftemp*** *n'ayant pas été spécifiés, les valeurs pas défaut sont utilisées)*

```
> obj5 <- Quad.couvrtr.gnrl(fspat='Geomorpho')
```
Couverture (moyenne et ET) par catégorie LIT générale (valeur par défaut de ***LIT.cat****) sur les quadrats par géomorphologie et campagne*

3 Lancement initial

La dernière version des codes peut être téléchargée sous le lien suivant:

https://github.com/lauratboyer/cw-ginger-soproner/archive/master.zip

Repérez le dossier 'Codes-tous-projets' et transférez-le à l'emplacement d'où vous voulez lancer et sauvegarder vos analyses. Ouvrir R et définir le répertoire courant à cet emplacement avec la commande `setwd()`,

ou en allant dans Fichier → Changer le répertoire courant...

```
setwd('.../Codes-tous-projets') # remplacer ''...''  par la trajectoire appropriée.
```

Par exemple si 'Codes-tous-projets' est placé sur le bureau, le répertoire courant devrait être défini comme suit:
`setwd('C:/Users/gilbert/Desktop/Codes-tous-projets')`.

Pour confirmer que le changement a bien été fait, tapez `getwd()` et confirmez que le retour correspond bien au dossier voulu.

*** Si vous recevez un message d'erreur du genre: `'Erreur dans getwd('...'): impossible de changer le répertoire de travail'`, vous avez soit une erreur de frappe, soit 'Codes-tous-projets' n'est pas situé à l'emplacement spécifié.

Truc: si vous double-cliquez directement sur un des fichiers .r, e.g. `GS_MotherCode_TProj.r` , le fichier s'affichera dans R Studio et le répertoire de travail sera déjà mis à la valeur du dossier contenant le code ouvert.

4 Chargement des données

Avant de commencer les analyses, vérifiez dans le fichier `GS_Mother-code_TProj.r` la valeur des variables suivantes, soit:

1. `facteurs.spatio`: Cet objet contient les variables explicatives spatiales disponibles pour l'analyse (autre que `St`). Si vous rajoutez des variables dans le fichier `Facteurs_spatiaux.csv`, vous devrez rajouter le nom de la nouvelle colonne dans cet objet pour que la variable soit prise en compte. En tout temps dans la session R vous pouvez consultez la valeur de cet objet pour voir les options qui peuvent être passées à l'argument `fspat` des fonctions d'analyses.
2. `facteurs.tempo`: Comme pour les facteurs spatiaux, mais pour les variables explicatives temporelles (autre que 'Campagne') disponibles dans le fichier `Facteurs_temporels.csv`. Ces variables peuvent être données en argument sous `ftemp` dans les fonctions d'analyses.
3. `facteurs.taxo`: Cet objet contient les valeurs disponibles pour les aggrégations taxonomiques (argument `ftaxo`) selon le groupe invertébrés, poissons ou LIT. Il existe purement pour un but informatif, donc vous ne devriez pas à avoir le changer (mais si vous le faites cela n'affectera pas les analyses).
4. `fspat.defaut`, `ftempo.defaut` et `agtaxo.defaut`: Valeurs par défaut des facteurs spatiaux, temporels, et taxonomiques, utilisées lorsque les arguments `fspat`, `ftemp` ou `agtaxo` ne sont pas spécifiés en argument. Vous pouvez changez la valeur de ces objets selon vos préférences pour les analyses.
5. `filtre.annees`: Un filtre sur les années pour l'analyse peut être spécifié ici mais je vous conseille de garder cette valeur à toutes les années (donc, `2006:2014`) quand vous initialisez les codes, et la changer ensuite manuellement dans la console R en tapant: `filtre.annees <- 2013`
6. `filtre.famille`: Gardez cette valeur à `TRUE` pour appliquer automatiquement sur les invertébrés le filtre sur les familles spécifiées dans le ficher `Filtre-taxo_Famille.csv`, sinon changez la valeur à `FALSE`.
7. `filtre.sur.especes`: un filtre taxonomique peut être spécifié manuellement ici. Voir section 4.1 pour les autres options.

Une fois les valeurs des variables changées (au besoin), sauvegardez les changements et initialisez le chargement des fonctions R et des données avec la commande:
`source('GS_Mother-code_TProj.r')`

Ce code lance automatiquement la fonction `import.tableaux()` , qui importe les bases de données dans R, et la fonction `prep.analyse()` qui nettoie les données pour l'analyse.

4.0.1 Rajout de nouvelles données

Si vous avez de nouvelles données à incorporer dans les bases R, sauvegarder les tableaux sous format .csv et mettez les nouvelles versions dans le dossier indiqué sous `dossier.DB` (et non `dossier.donnees`). Dans une nouvelle session R, allez dans `GS_MotherCode_TProj.r` , changez la valeur de la variable `refaire.tableaux` à `TRUE` et lancer le script avec `source(...)`. Si le script tourne sans erreurs (vérifiez que de nouveaux objets ont été créés dans le dossier Tableaux-pour-analyses), re-changez la valeur de `refaire.tableaux` à `FALSE` pour sauver du temps les prochaines fois lorsque vous lancez les scripts (les objets seront déja formattés, donc le chargement sera beaucoup plus rapide).

4.1 Filtre taxonomique

Le filtre sur espèces est défini par trois objets:

1. `taxoF.incl` spécifie si les noms donnés devraient être exclus (`'exclure'`) ou inclus (`'inclure'`) dans l'analyse.
2. `taxoF.utaxo` contient le niveau taxonomique à laquelle l'inclusion (ou l'exclusion) est faite, e.g. G_Sp, Genre, Famille, S_Groupe ou Groupe.
3. `taxoF.nom` contient la liste des noms des membres du niveau `taxoF.utaxo` sur laquelle l'action `taxoF.incl` sera portée.

Une fois la session lancée, pour voir la valeur des filtres présentement enregistrée, tapez `voir.filtre.taxo()` dans la console R.

Vous avez plusieurs options pour changer la valeur du filtre:

- Pour un lancement des codes complètement automatique, mettez dans `GS_MotherCode_TProj.r` l'option `filtre.sur.especes = TRUE` et modifiez directement les valeurs des objets `taxoF.incl`, `taxoF.utaxo` et `taxoF.nom`. *** Ces valeurs peuvent aussi être modifiée manuellement dans la console R une fois que `GS_MotherCode_TProj.r` est lancé.***
- Si vous avez une liste de noms à importer d'un fichier .csv, utilisez la fonction `import.filtre.taxo()` et spécifiez en argument `niveau='...'` pour le niveau taxonomique désiré et `action=exclure'` si cette action est désirée (par défaut la fonction spécifie `taxoF.incl='inclure'`. Pour voir le format approprié au fichier .csv, référez-vous au fichier `Filtre-taxo_Famille.csv`
- Finalement vous pouvez aussi utiliser la fonction `def.filtre.especes()` pour définir intéractivement sous R les valeurs du filtre, comme suit:

```
> def.filtre.especes.def("Inclure")

Unité taxonomique?  (Groupe/Sous-Groupe/Famille/Genre/Espece)
Groupe

Nom?
```

```
Crustaces, Mollusques, Echinodermes
```

Pour éliminer le fitre sur espèces, faites `def.filtre.especes()` dans la console R, sans arguments.

4.2 Sélection du projet

→ Cette étape est essentielle aux lancements des codes, sinon vous aurez une erreur R.

Une fois les données chargées et nettoyées, vous devez sélectionner le projet à utiliser. Pour voir les noms de projet disponibles, faites:

```
nom.projets()
```

Vous pourrez sélectionner un de ces projets pour les analyses avec la fonction `selection.projet()` , par exemple:

```
selection.projet('ADECAL_TOUHO')
```

Si vous ne donnez pas d'arguments à cette fonction, la valeur définie par défaut est 'KNS_KONIAMBO'. Veillez à utiliser exactement le même format défini sous `nom.projets()`.

En tout temps vous pouvez voir la valeur du projet en cours en tapant la variable `projet` dans la console R.

À noter que lorsqu'un projet est sélectionné, un dossier est automatiquement créé (si non-existant) à l'intérieur du dossier `Codes-tous-projets` pour sauvegarder les sorties tableaux ou graphiques. Les bases de données sont filtrées pour ne conserver que les tableaux pertinents au projet en cours, et mises dans les tableaux `dbio` pour les invertébrés, `dpoissons` pour les poissons, `dLIT` pour les LIT et `dQuad` pour les quadrats.

5 Début des analyses

Les fonctions suivantes font le calcul de la densité/couverture moyenne et de l'écart type sur les aggrégations spatiales, temporelles et taxonomiques définies en argument:

1. hop
2. `INV.dens.gnrl()` : fspat, ftemp, ftaxo
3. `Pois.dens.gnrl()` : fspat, ftemp, ftaxo
4. `LIT.couvrt.gnrl()` : fspat, ftemp, LIT.cat
5. `Quad.couvrt.gnrl()` : fspat, ftemp, LIT.cat

Pour toutes les fonctions vous pouvez spécifiez 'A' ou 'S' à l'argument `filt.camp` pour ne conservez que les campagnes annuelles ou semestrielles, respectivement.

Pour voir les arguments disponibles pour n'importe quelle fonction (et leurs valeurs par défaut), utilisez la fonction `formals()` , e.g:

```
formals(INV.dens.gnrl)
```

Pour sauvegarder les calculs dans un tableau et les visualiser/utiliser sous R, assignez la fonction à un objet. Pour sauvegarder le tableau, spécifiez l'argument `'save=TRUE'`. Par exemple:

```
obj1 <- INV.dens.gnrl() # tableau contenu dans 'obj1'
INV.dens.gnrl(save=TRUE) # tableau sauvegardé mais non-visible dans R
obj1 <- INV.dens.gnrl(save=TRUE) # tableau contenu dans 'obj1'
```

6 Invertébrés

Fichier: `GS_CodesInvertebres_TProj.r`
Tableau formatté: `dbio`
Fonctions:
(1) `INV.dens.gnrl()`: mesures de densité
(2) `INV.biodiv.gnrl()`: métriques de biodiversité

Les arguments `fspat` et `ftemp` définisent les facteurs spatiaux et temporels d'aggrégation, respectivement, alors que `agtaxo` défini le niveau taxonomique sur lequel les densités sont calculées. Les valeurs par défaut pour ces arguments sont 'St', 'Campagne' et 'Groupe' (modifiables dans `GS_mother-code_TProj.r`). Donc, lancer `INV.dens.gnrl()` sans arguments spécifiques produira une moyenne (ET) de la densité par groupe par station par campagne, alors que `INV.dens.gnrl(fspat='Geomorpho', ftemp='Saison', agtaxo='Famille')` produira ces mêmes statistiques mais par famille, géomophologie et saison.

Pour avoir les valeurs par transect, donc sans aggrégation, utilisez l'argument `'par.transect = TRUE'`.

Abondances nulles: `wZeroT = TRUE` (valeur par défaut) conserve les abondances nulles sur tous les transects et campagnes où une espèce n'a pas été observée. `wZeroSt = TRUE` (mis à `FALSE` par défaut) garde les abondances nulles sur les stations même lorsque `wZeroT = FALSE`. Ça peut être utile si vous voulez calculez les densités moyennes observées en excluant les abondances nulles, mais quand même voir les stations où l'espèce n'a été observée sur aucun des transects. Changer la valeur de `wZeroSt` n'affectera le résultat seulement lorsque `wZeroT = FALSE`.

Les arguments sont les mêmes pour `INV.biodiv.gnrl()`, à part qu'il n'y pas d'aggrégation taxonomique `ftaxo`. Les métriques sont calculées à l'échelle de l'espèce en utilisant soit le transect (valeur par défaut, `unit.base = 'T'`) ou la station (`unit.base = 'St'`) comme unité de base pour le calcul. Les richesses spécifiques sont calculées à toutes les échelles taxonomiques.

7 Poissons

Fichier: `GS_CodesPoissons_TProj.r`
Tableau formatté: `dpoissons`
Fonctions:
`POIS.dens.gnrl()`: mesures de densité, biomasse, taille moyenne et richesse spécifique

La fonction `POIS.dens.gnrl()` produit les analyses pour les poissons. Les arguments principaux sont les mêmes que pour les statistiques sur les invertébrés, à part que l'argument `'agtaxo'` peut-être utilisé pour spécifier une échelle taxonomique ou une charactéristique écologique, e.g. `agtaxo='moblabel'` rendra les statisques par type de mobilité comme spécifié dans le tableau Bioeco. Pour conserver les abondances nulles

sur les transects vous utilisez, comme pour les invertébrés, `wZeroT = TRUE` (valeur par défaut). A noter que `wZeroSt` n'est pas disponible pour les poissons vu qu'il n'y a qu'un transect par station dans la grande majorité des cas.

8 LIT et Quadrats

Fichier: `GS_CodesLIT_TProj.r`
Tableaux formattés: `...`
Fonctions:
`LIT.couvrt.gnrl()` et `Quad.couvrt.gnrl()`: couverture moyenne par typologie LIT

Les statistiques LIT/Quadrat peuvent être obtenues avec les fonctions `LIT.couvrt.gnrl()` et `Quad.couvrt.gnrl()`. L'argument `LIT.cat` spécifie le type de catégorie LIT utilisé pour l'aggrégation (défini dans le tableau 'Type_LIT.csv'). Les arguments `fspat` et `ftemp` sont utilisés comme ci-dessus. Note: pour avoir les données brutes par transect/quadrat, sans aggrégation, vous pouvez utiliser les fonctions `LIT.tableau.brut()` et `Quad.tableau.brut()`.

9 Abondances nulles

A partir du moment où une espèce est observée sur un transect pour un projet, une abondance nulle est rajoutée sur tous les transects, stations et campagnes où l'espèce est absente.

10 Création de tableaux formattés

11 Trucs pour trouver la source de l'erreur

Avant tout travail de détective, commencez par vous assurez que vous n'avez faites aucune erreur de frappe (!) et/ou spécification du répertoire de travail.

S'il y a un bug lors du lancement des codes (initialement ou dans les analyses subséquentes), commencez par identifier la fonction où le bug apparaît. Typiquement ça sera indiqué dans le message d'erreur (e.g. `Erreur dans la fonction LIT.couvrt.gnrl()`). Sinon regardez bien le message pour identifier un des objets où l'erreur prend place, e.g.

```
Error in data.frame(t1[, colcl != "matrix"], t1.sub) (from con#756)
```

nous indique qu'il y a une commande, quelque part dans les codes, où la ligne suivante est utilisée: `data.frame(t1[, colcl != "matrix"], t1.sub)`, possiblement sur la ligne 756 d'un des fichiers de code.

La fonction `file.scan()` permet de retrouver le fichier R où une certaine ligne de code apparaît, comme suit:

```
> file.scan("data.frame(t1")
[1] "GS_ExtractionDonnees_TProj.r"
```

... et effectivement si je vais vers la ligne 756 de ce fichier, je retrouve cette commande, qui se trouve en fait dans une fonction s'appelant `aggr.multi()`. (Notez que la fonction `file.scan()` ne vient avec R par défaut mais est définie lors du lancement initial des codes.)

Alternativement, si vous connaissez déjà le nom de la fonction où le bug prend place, vous pouvez utilisez la fonction `debugonce(...)` :

```
> debugonce(prep.analyse)
```

... qui vous permettra de lancer les commandes ligne par ligne (vous devrez peser sur la touche retour pour que la ligne suivante soit lancée dans la console), et donc identifier la ligne exacte où le bug prend place.

Une fois la ligne identifiée, examinez les objets utilisés dans la commande. En continuant avec l'exemple précédent, vous pourriez faire:

```
> debugonce(aggr.multi)
```

et relancer la commande initiale qui avait produit le bug. Lorsque la fonction arrivera à l'étape d'utilisation de `aggr.multi()` elle devrait s'arrêter et vous faire passer par chaque ligne. Quand vous reconnaissez la ligne où le bug s'est produit, avant de peser sur la touche RETOUR, examinez en détail la commande... `t1.fn = data.frame(t1[,colcl!="matrix"], t1.sub)` Ici on sélectionne les colonnes `colcl` dans le tableau `t1`, et on le joint avec le tableau `t1.sub`. Vu que le message d'erreur disait:

```
arguments imply differing number of rows:  564, 0
```

on peut se douter que `t1` et `t1.sub` n'ont pas le même nombre de rangées... et effectivement lorsqu'on vérifie avec les commandes `nrow(t1); nrow(t1.sub)` c'est bien le cas. `t1.sub` est vide. L'étape suivante est de trouver où `t1.sub` est crée, et de vérifier pourquoi il est vide... et ainsi de suite. Tant que vous y allez étape par étape vous allez trouver la source du bug, et deviendrez plus efficace avec la pratique dans l'interprétation du message d'erreur. Ici, la commande initiale ayant causée le bug était:

```
> LIT.couvrt.gnrl(LIT.cat="hop")
```

et éventuellement vous auriez trouvez que `t1.sub` est vide parce qu'il n'y pas de catégorie s'appellant 'hop' dans les tableaux LIT.

Finalement, il y a plusieurs fonctionalités pour 'débugger' à explorer dans RStudio.