Rcpp

ANF R pour le calcul

Daphné Giorgi

24 septembre 2024

Villa Clythia, Fréjus

Motivations

Quand utiliser Rcpp

`R` est faible dans certains types d'opérations.

- Opérations en boucle imbriquées.
- Accès aux éléments d'un vecteur ou d'une matrice.
- Appels de fonctions dans les boucles.
- Modification dynamique de la taille des vecteurs.
- Opérations nécessitant des structures de données et des algorithmes avancés.

L'utilisation de `Rcpp` permet d'améliorer considérablement les performances.

R et C++

Les deux langages ont leurs pro et leurs contre :

	C++	R
Rapidité	✓	✗
Gestion mémoire	✓	✗
Prise en main	✗	✓
Collaboration	✗	✓
Joli rendu	✗	✓

Une bonne combinaison des deux serait l'idéal !

Paquet Rcpp

Paquet Rcpp

`Rcpp` est un outil écrit par *Dirk Eddelbuettel* et *Romain Francois* (avec des contributions clés de Doug Bates, John Chambers et JJ Allaire).

`Rcpp` est un peu différent de la plupart des paquets `R` standard. Il permet de **connecter très simplement** `C++` **à** `R`, en rendant accessibles des fonctions écrites en `C++` et en passant des scalaires, des vecteurs, des matrices, des listes ou des objets R entiers entre `R` et `C++` avec facilité.

En début janvier 2024, 2791 paquets `CRAN` contiennent du `Rcpp` (13.8% des paquets) et `Rcpp` est téléchargé 1 361 411 de fois par mois.

Installation

- Compilateur `C++`
  - Installer Rtools (Windows)
  - Installer Xcode (Mac)
  - `sudo apt-get install r-base-dev` (Linux)
- Variables d'environnement :
  - `.R/Makevars` (Mac, Linux)
  - `.R/Makevars.win` (Windows)
- Paquet `Rcpp`
  - `install.packages("Rcpp")`

```
> library("Rcpp")
> evalCpp("2+2")
#  [1] 4
```

Quelques concepts importants

Langages compilés et interprétés

Dans un **langage compilé** – ce qui est le cas de `C++` —, une fois passé les étapes de conception de l'algorithme et d'écriture du code, on procède comme il suit :

- Un *compilateur* compile le code pour produire le programme exécutable
- Le programme peut être exécuté autant de fois qu'on le souhaite

Un **langage interprété** – ce qui est le cas de `R` — mélange les deux dernières étapes:

- Un *interpréteur* est lancé sur le code non-compilé et transforme chaque ligne en instruction machine immédiatement exécutée.

Mémoire et typage

Mémoire ordonnée : Toute mémoire, vive ou morte, est organisée en une suite ordonnée de *bits* (valeurs 0 ou 1).

Adressage : On accède à toute séquence en donnant au processeur le numéro du bit de début et le nombre de bits à lire. Cet *adressage* de la mémoire est à la base des notions de *pointeur* et *référence* essentielles en C++.

Langage typé : La suite de bits `01101` peut coder aussi bien

- $(0, 1, 1, 0, 1)$ de cinq valeurs dans $\{0, 1\}$
- l'entier $0 \times 1 + 1 \times 2 + 1 \times 4 + 0 \times 8 + 1 \times 16 = 22$
- l'entier $1 \times 1 + 0 \times 2 + 1 \times 4 + 1 \times 8 + 0 \times 16 = 13$ (écrit dans l'autre sens)

Le *codage* de l'information a donc une grande importance : c'est à la base de la notion de *type* qu'il existe en C++.

Mémoire et typage

Déclaration d'une variable :

```
int x;
int z;
```

Affectation d'une variable :

```
z= 19;
x= z+3;
```

Déclaration et affectation d'une variable :

```
int z=19;
```

Mémoire et typage

Les entiers :

- `short int` : 2 octets (16 bits, un pour le signe et ensuite 15 bits pour le codage du nombre en base deux : on peut donc atteindre $2^{15} = 32\,768$, c'est peu)
- `int` : 4 octets (32 bits, un pour le signe et ensuite 31 bits pour le codage du nombre en base deux, valeur maximale 2 147 483 648)
- `long` : 4 ou 8 octets
- `long long int` : 8 octets depuis la norme de 1998
- Entiers positifs : il est possible de récupérer le bit de signe en déclarant les entiers par le type `unsigned`

Mémoire et typage

Les réels :

La notion la plus proche de nombres réels est celle de *flottants* qui correspond à l'écriture d'un nombre réel sous la forme

$$\begin{aligned} 254,36 &= 2,5436 \times 10^2 = \overline{11111110,010111}_2 \\ &= \overline{1.11111100010111}_2 \times 2^7 = \overline{1.11111100010111}_2 \times 2^{\overline{111}_2} \end{aligned}$$

où $\overline{\bullet}_2$ désigne l'écriture en base 2. Le nom de *flottant* provient du fait que l'écriture précédente s'appelle écriture en virgule flottante.

- `float`: 4 octets, i.e. 32 bits (1 bit de signe, 8 bits pour l'exposant, 23 bits pour la mantisse), 6 chiffres après la virgule et des exposants variant de -38 à 38.
- `double`: 8 octets, i.e. 64 bits (1 bit de signe, 11 bits d'exposants et 52 bits pour la mantisse), 15 chiffres après la virgule et des exposants variant de -308 à 308.

Les booléens :

Une variable de type `bool` ne prend que deux valeurs possibles, `true` (écrit aussi `1`) et `false` (écrit aussi `0`).

Opération logique	Opérateur en C++	exemple
ET	`&&`	`a && b ;`
OU	`||`	`a || b ;`
NON	`!`	`!a ;`

Fonctions et objets

La *programmation fonctionnelle* consiste à écrire des fonctions qui effectuent des opérations sur les données, tandis que la *programmation orientée objet* consiste à créer des objets qui contiennent à la fois des données et des fonctions.

`C++` **est un langage orienté objet**, qui prend en charge également la programmation fonctionnelle. Il offre des fonctionnalités telles que les classes, l'héritage et le polymorphisme pour la construction de programmes complexes.

`R` **est avant tout un langage de programmation fonctionnelle**, qui prend également en charge la programmation orientée objet grâce aux systèmes d'objets S3 et S4.

Fonctions en R et en C++

```r
NOM <-  function(a1, a2, ... , an) {
        ...#instructions
        return(RESULTAT)
}
```

R

```cpp
TYPE_SORTIE  NOM( TYPE1 a1, TYPE2 a2, ... , TYPEn an) {
        ...//instructions
        return RESULTAT;
};
```

C++

Objets en C++

```cpp
#include <string>
#include <vector>

class MyClass {
  public:
    int myNum;
    string myString;
    int get_myNum() const;
    void setInternalNum(const vector<int>& v);

  private :
    int internalNum;
};
```

C++

Les standards C++

Évolution du C++

- 1979 : Stroustrup ajoute les classes au C
- 1982 : C++
- 1998 : std C++98
- modern : Le langage a été mis à jour quatre fois en 2011, 2014, 2017 et 2020 pour devenir C++11, C++14, C++17, C++20.

Tutoriel

Ressources

`Rcpp` est particulièrement bien documenté.

Plusieurs ressources de qualité sont disponibles, parmi lesquelles :

- Rcpp for everyone
- High performance functions with Rcpp
- Rcpp Gallery
- Statistical Computing 2

Programme

Rappel des types en `R`

Correspondances avec les types de `Rcpp`

Différentes façons de compiler du code en utilisant `Rcpp`

Utilisation des différents types en `Rcpp`

Les librairies de la `STL`

`Rcpp Sugar`

`Rcpp Armadillo`

Aléatoire

Intégrer `Rcpp` dans un paquet