ribModel Code specifications

June 6, 2017

Contents

1 C++ Code

1.1 Bracket Format

- Brackets should begin on the line below a function, if statement, while clause, etc.
- One-line statements may have brackets omitted with just an indentation, or it may be put on the same line.
- Example code:

```
void Gene::cleanSeq()
{
    std::string valid = "ACGTN";
    for (unsigned i = 0; i < seq.length(); i++)
    {
        if (valid.find(seq[i]) == std::string::npos)
            seq.erase(i);
    }
}
```

1.2 Commenting

- There should be a documentation block above each function. The format of the documentation block is a long-form C++ comment with the name of the function and whether or not it is exposed to RCPP, followed by *-separated lines listing the arguments and any further description.
- Caveats to the function and any TODOs are listed in this documentation block at the end.
- If you copy code, mention the source so the code can be double checked-later.
- Example code:

```
/* Gene = operator (NOT EXPOSED)
 * Arguments: Gene object
 * Overloaded definition of the assignment operator ( = ). Function is
 * similar to that of the copy constructor.
*/
```

- Within the code itself, document/describe parts of code/functions that are not self-explanatory (Can you come back to it a month later and still know what the code does?).
- Major blocks of code separating divisions in many functions' usage or unit testing may be denoted by `//--------------------//` blocks.
- Example code:

```
    //--------------------------------------//
    //------ initParameterSet Function ------//
    //--------------------------------------//
```

1.3 Code Spacing

- Between different functions and their respective ends and comment blocks, there should be two lines of spacing between them.

- Before a major section of code (marked by `//--------------------//` blocks) there should be five lines of spacing at the end of the previous function.
- After that comment block, however, there may be only two lines of spacing like usual.

1.4 Function/Variable Naming Conventions

- Functions should be named as clearly as possible, in camelCase.
- Example: "Gene::getObservedSynthesisRateValues()"
- Variables that are meant to be temporary should be named as such.
- Example: "tmpGene"
- Use speaking variable names.

1.5 New Functions/Classes

- Create a test case for new functions in the class Testing.
- Ensure that constructors create a VALID object. That does not necessary mean all information has to be placed, but all members have to be properly initialized.
- Functions are only defined in headers, but never implemented.
- New classes should follow naming convention: [MODEL NAME]Parameter, [MODEL NAME]Model. That should be generally applied for inherited classes.
- Constructors can not have more than 6 arguments. That causes us to use setter functions in cases where more arguments are needed. Adjustment can be made once Rcpp can handle more arguments.

1.6 Implementation

- Avoid infinite loop structures (for(;;), while(true), ...) that break under various conditions.
- Avoid dynamically-allocated arrays and use vectors instead if possible. Dynamic allocated arrays seem to cause problems with openmp.
- All log output should use the wrapper functions "my_print", "my_printError", "my_printWarning".
- Rcpp modules are used to expose classes and member functions. Member functions and static functions are exposed using a different syntax (see code).

2 R Code

- Stick to the Google style guide for R.