Package ‘ast2ast’

March 14, 2022

Type Package

Title Translates a R Function into an External Pointer to a C++ Function

Version 0.1

Date 2022-02-28

Author Krämer Konrad [aut, cre]

Maintainer Krämer Konrad <konrad_kraemer@yahoo.de>

BugReports https://github.com/Konrad1991/ast2ast

URL https://github.com/Konrad1991/ast2ast

Description Enable translation of a tiny subset of R to C++. The user has to define a R function which gets translated. For a full list of possible functions check the documentation. After translation an external pointer to the C++ function is returned to the user. The intention of the package is to generate fast functions which can be used as ode-system or during optimization.

License GPL-2

Imports Rcpp (>= 1.0.4), purrr, R6, RcppXPtrUtils, RcppArmadillo

VignetteBuilder knitr

Suggests knitr, kableExtra, rmarkdown, tinytest, microbenchmark, r2sundials, paropt

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-14 09:00:05 UTC

R topics documented:

`translate`	*Translates a R function into a C++ function and returns an external pointer (XPtr) to this function. Further information can be found in the vignette: 'Detailed Documentation'.*

Description

Translates a R function into a C++ function and returns an external pointer (XPtr) to this function. Further information can be found in the vignette: 'Detailed Documentation'.

Usage

```
translate(f, verbose = FALSE, reference = FALSE)
```

Arguments

`f`	The function which should be translated from R to C++.
`verbose`	If set to TRUE the output of RcppXPtrUtils::cppXPtr is printed.
`reference`	If set to true the arguments are passed by reference.

Details

The following types are supported:

1. numeric vectors
2. numeric matrices

Variables can be either numeric vectors or matrices. Notably, it is possible that the variables change the type within the function. **It is possible to declare a variable of a scalar numeric data type. This is done by adding '_db' to the end of the variable. Each time '_db' is found the variable is declared as a scalar numeric data type. In this case the object cannot change its type!**

The following functions are supported:

1. assignment: = and <-
2. allocation: vector and matrix
3. information about objects: length and dim
4. Basic operations: +, -, *, /
5. Indices: [] and at
6. mathematical functions: sin, asin, sinh, cos, acos, cosh, tan, atan, tanh, log, ^ and exp
7. concatenate objects: c
8. comparison: ==, !=, >, <, >= and <=
9. printing: print
10. returning objects: return
11. catmull-rome spline: cmr

12. to get a range of numbers the ':' function can be used

Some details about the implemented functions

- allocation of memory works: Following forms are possible: vector(size_of_elements), vector(value, size_of_elements), vector(other_vec, size_of_other_vec), matrix(nrows, ncols), matrix(value, nrows, ncols) and matrix(vector, nrows, ncols). The latter fills the matrix or the vector with the specified 'value'.
- For indices squared brackets '[]' can be used as common in R. **Despite the results of calculations cannot be used!** Beyond that the function 'at' exists which accepts as first argument a variable and as the second argument you pass the desired index. The caveat of using 'at' is that only one entry can be accessed. The function '[]' can return more then one element. **The 'at'function returns a reference to the vector entry. Therefore variable[index] can behave differently then at(variable, index). The function has to be used carefully when 'at' is used. Especially if '[]' and 'at' are mixed the function behaviour is difficult to predict. Please test it before using it in a serious project.**
- For-loops can be written as common in R
  - Nr.1
    ```
    for(index in variable){
    # do whatever
    }
    ```
  - Nr.2
    ```
    for(index in 1:length(variable){
    # do whatever
    }
    ```
- Be aware that it is not possible to assign the result of a comparison to a variable.
- The print function accepts either a scalar, vector, matrix, string, bool or nothing (empty line).
- In order to return an object use the 'return' function (The last object is not returned automatically as in R).
- In order to interpolate values the 'cmr' function can be used. The function needs three arguments.
  1. the first argument is the point of the independent variable (x) for which the dependent variable should be calculated (y). This has to be a vector of length one.
  2. the second argument is a vector defining the points of the independent variable (x). This has to be a vector of at least length four.
  3. the third argument is a vector defining the points of the dependent variable (y). This has to be a vector of at least length four.

Be aware that the R code is translated to ETR an expression template library which tries to mimic R. However, it does not behave exactly like R! Please check your compiled function before using it in a serious project. If you want to see how ast2ast differs from R in detail check the vignette: 'Detailed Documentation'.

Value

The external pointer of the generated C++ function

Examples

```
#Further examples can be found in the vignette: 'Detailed Documentation'.
#Hello World
## Not run:
f <- function() { print("Hello World!")}
pointer_to_f_cpp <- ast2ast::translate(f)
Rcpp::sourceCpp(code = "
#include <Rcpp.h>
typedef void (*fp)();

// [[Rcpp::export]]
void call_fct(Rcpp::XPtr<fp> inp) {
  fp f = *inp;
  f();
}
")
call_fct(pointer_to_f_cpp)

## End(Not run)
```

Index