

nCompiler: generating C++ from R

Perry de Valpine

Environmental Science, Policy & Management
University of California, Berkeley

Contributors:

- Daniel Turek, Chris Paciorek, Nicholas Michaud (via contributions to nimble)
- James Duncan

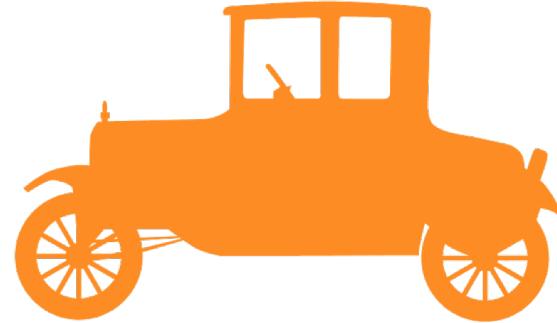
useR!2019
Toulouse

<https://github.com/nimble-dev/nCompiler>

Outline

- History
- Goals
- Main abstractions, features and current status
- We welcome ideas and contributions.

History



- nCompiler started as an internal tool for nimble.



r-nimble.org

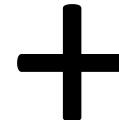
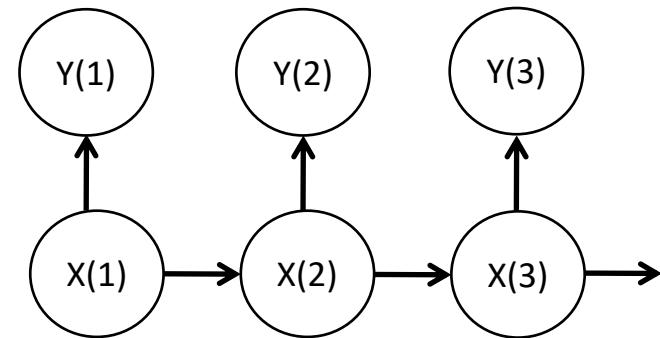
Numerical Inference for statistical Models using Bayesian and Likelihood Estimation

Funded in part by:



What is NIMBLE?

Statistical model language:
New dialect of BUGS/JAGS.

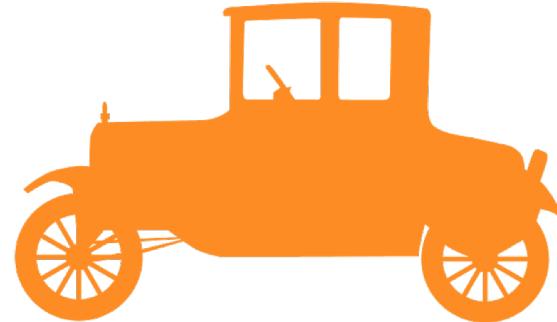


Algorithm language
embedded in R



“nimble compiler”:
Generates C++ for each model and algorithm (e.g. MCMC)

History



- nCompiler started as an internal tool for nimble.
- The “nimble compiler” works pretty well!
- Maybe it could be a more general tool:
 - Gain C++ speed-ups without coding C++ directly.
 - Automatically get derivatives, parallelization, and serialization.
- It has some design limitations and concepts particular to nimble.
- nCompiler is a complete re-write with heavy borrowing from nimble.

nFunction

```
1 library(nCompiler)
2
3 exp_vec <- nFunction(
4   fun = function(x) {
5     ans <- exp(x)
6     return(ans)
7   },
8   argTypes = list(x = 'numericVector'),
9   returnType = 'numericVector'
10 )
```

explicit “return”

Argument and return type-declarations.

```
> exp_vec(1:3)
[1] 2.718282 7.389056 20.085537
> Cexp_vec <- nCompile(exp_vec)
> Cexp_vec(1:3)
[1] 2.718282 7.389056 20.085537
```

Everything runs uncompiled and compiled.

Goals

Keep what worked well:

- Code generation from R mathematical and distribution functions
- Automatic type determination based on declared inputs
- Coding embedded in R via new types of “function” and “class”
- Linear algebra via Eigen
- Algorithmic differentiation (AD) via CppAD (not released)
- Calls to external libraries or to R
- Basic flow control



Goals

Keep what worked well:

- Code generation from R mathematical and distribution functions
- Automatic type determination based on declared inputs
- Coding embedded in R via new types of “function” and “class”
- Linear algebra via Eigen
- Algorithmic differentiation (AD) via CppAD (not released)
- Calls to external libraries or to R
- Basic flow control



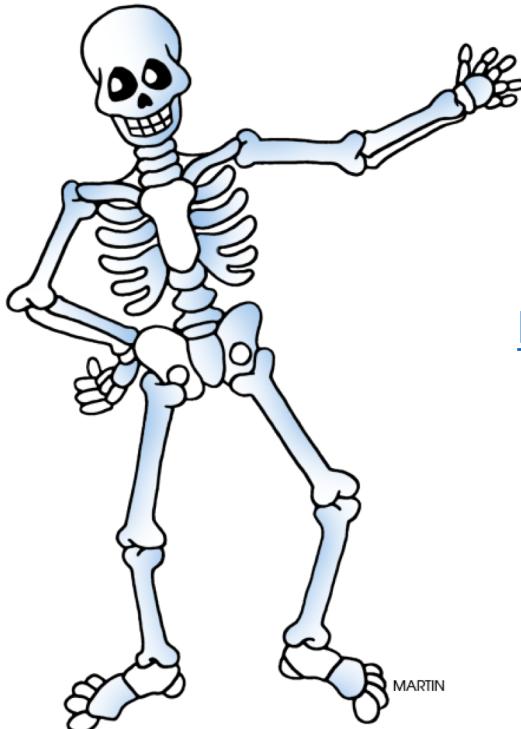
What to add or change:

- Clarify key abstractions: nFunction, nClass.
- Use Eigen more deeply and Eigen::Tensor for math with arbitrary arrays
- Ground-up support for:
 - Parallelization (Threading Building Blocks)
 - Serialization (saving and loading C++ objects) (Cereal).
 - Use in package development
- Easier integration with hand-written C++
- Better use/integration/compatibility with other tools (Rcpp family).
- Extensibility and developer tools



https://creazilla.com/nodes/2875-birds-on-a-wire-silhouette?tag_id=9
<https://creativecommons.org/licenses/by/4.0/>

Current status: A working skeleton of all major components.



<https://github.com/nimble-dev/nCompiler>

nFunction

```
1 library(nCompiler)
2
3 exp_vec <- nFunction(
4   fun = function(x) {
5     ans <- exp(x)
6     return(ans)
7   },
8   argTypes = list(x = 'numericVector'),
9   returnType = 'numericVector'
10 )
```

explicit “return”

Argument and return type-declarations.

```
> exp_vec(1:3)
[1] 2.718282 7.389056 20.085537
> Cexp_vec <- nCompile(exp_vec)
> Cexp_vec(1:3)
[1] 2.718282 7.389056 20.085537
```

Everything runs uncompiled and compiled.

```
// [[Rcpp::export]]
Eigen::Tensor<double, 1> nFun_2_NFID_2 ( Eigen::Tensor<double, 1> x ) {
Eigen::Tensor<double, 1> ans;
ans = (x).exp();
return(ans);
}
#endif
```

Harness Rcpp

```
// [[Rcpp::export]]  
Eigen::Tensor<double, 1> nFun_2_NFID_2 ( Eigen::Tensor<double, 1> x ) {  
Eigen::Tensor<double, 1> ans;  
ans = (x).exp();  
return(ans);  
}  
#endif
```

Extend as<> and wrap<> as needed

```
// [[Rcpp::export]]
Eigen::Tensor<double, 1> nFun_2_NFID_2 ( Eigen::Tensor<double, 1> x ) {
Eigen::Tensor<double, 1> ans;
ans = (x).exp();
return(ans);
}
#endif
```

nFun_2_NFID_2 (

ans;

Eigen::Tensor<double, 1> x) {

Use Eigen more deeply.
Use Eigen::Tensor



<http://eigen.tuxfamily.org>

```
// [[Rcpp::export]]
Eigen::Tensor<double, 1> nFun_2_NFID_2 ( Eigen::Tensor<double, 1> x ) {
Eigen::Tensor<double, 1> ans;
ans = (x).exp();
return(ans);
}
#endif
```

Annotate and transform abstract
syntax tree and symbol table(s) to
generate C++.

nClass

```
multClass <- nClass(  
  classname = "multClass",  
  Rpublic = list(),  
  Cpublic = list(  
    v = 'numericVector',  
    multV = nFunction(  
      fun = function(c = 'numericScalar') {  
        return(c*v)  
      },  
      returnType = 'numericVector'  
    )  
)
```

```
> CmultClass <- nCompile(multClass)  
> my_CmultClass <- CmultClass$new()  
> my_CmultClass$v <- 1:3  
> my_CmultClass$multV(2)  
[1] 2 4 6  
> |
```

nClass generates a custom R6 class.

Rpublic implemented in R.

Cpublic implemented in C++.

AD: Algorithmic (or Automatic) Derivatives

cppad-20190707: A C++ Algorithmic Differentiation Package



[releases](#) , [20190200.3](#) , [github](#) , [travis](#) , [appveyor](#) , [cppad.spec](#)

[install](#) , [get started](#) , [whats new](#) , [addon](#) , [research](#) , [project manager](#)

CppAD is distributed by [COIN-OR](#) with the Eclipse Public License [EPL-2.0](#) or the GNU General Public License [GPL-2.0](#) or later.

Also used by

- [TMB](#) (Kristensen, Bell, Skaug, Magnusson, Berg, Nielsen, Maechler, Michelot, Brooks, Forrence, Albertsen, & Monnahan). On CRAN.
- [RcppEigenAD](#) (Berridge, Crouchley & Grose). On CRAN.

AD: Algorithmic (or Automatic) Derivatives

```
set_nOption('automaticDerivatives', TRUE)
a_exp_v <- nClass(
  classname = "a_exp_v",
  Rpublic = list(),
  Cpublic = list(
    go = nFunction(
      fun = function(a = 'numericScalar',
                     v = 'numericVector(length = 3)') {
        return(a*exp(v))
      },
      returnType = 'numericVector(length = 3)')
  ),
  enableDerivs = 'go',
)
```

Call via nDerivs

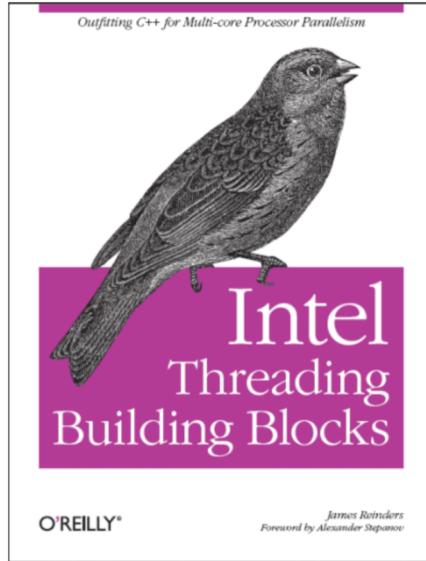
Jacobian:

Fixed-length inputs and output

enableDerivs argument

```
> C_a_exp_v <- nCompile(a_exp_v)
> my_C_a_exp_v <- C_a_exp_v$new()
> nDerivs(my_C_a_exp_v$go(2, 1:3))$gradient
     [,1]      [,2]      [,3]
[1,] 2.718282  7.389056 20.08554
[2,] 5.436564  0.000000  0.00000
[3,] 0.000000 14.778112  0.00000
[4,] 0.000000  0.000000 40.17107
```

Parallelization



Also used by



Allaire, Francois, Ushey, Vandenbrouck, Geelnard, RStudio, Intel, Microsoft
(On CRAN)

Parallelization

```
nc <- nClass(  
  Cpublic = list(  
    go = nFunction(  
      fun = function(x = 'numericVector') {  
        y <- x  
        parallel_for(i, 1:10,  
                     {y[i] <- 2 * x[i]},  
                     "x", ## copy for each thread  
                     "y") ## share across threads  
        return(y)  
      },  
      returnType = 'numericVector'  
    )))
```

parallel_for (final syntax TBD)

Variables to copy or share
across threads.

```
> Cnc <- nCompile(nc)  
> Cnc1 <- Cnc$new()  
> Cnc1$go(101:110)  
[1] 202 204 206 208 210 212 214 216 218 220
```

Argument passing

- By copy
- By reference
- By block reference

Mixing with other C++

```
nf <- nFunction(  
  fun = function(x = 'numericVector') {  
    z <- x + 10  
    cppLiteral(  
      'ans = Rcpp::List::create(  
        Rcpp::Named("x") = Rcpp::wrap(x),  
        Rcpp::Named("z") = Rcpp::wrap(z));',  
      types = list(ans = list())  
    )  
    return(ans)}  
  returnType = 'list')
```

Using nCompiler code in packages

- Generate necessary R and C++ into package src and inst directories.

Argument passing

- By copy
- By reference
- By block reference

Mixing with other C++

```
nf <- nFunction(  
  fun = function(x = 'numericVector') {  
    z <- x + 10  
    cppLiteral(  
      'ans = Rcpp::List::create(  
        Rcpp::Named("x") = Rcpp::wrap(x),  
        Rcpp::Named("z") = Rcpp::wrap(z));',  
      types = list(ans = list())  
    )  
    return(ans)}),  
  returnType = 'list')
```

Using nCompiler code in packages

- Generate necessary R and C++ into package src and inst directories.

Argument passing

- By copy
- By reference
- By block reference

Mixing with other C++

```
nf <- nFunction(  
  fun = function(x = 'numericVector') {  
    z <- x + 10  
    cppLiteral(  
      'ans = Rcpp::List::create(  
        Rcpp::Named("x") = Rcpp::wrap(x),  
        Rcpp::Named("z") = Rcpp::wrap(z));',  
      types = list(ans = list())  
    )  
    return(ans)  
  },  
  returnType = 'list')
```

Using nCompiler code in packages

- Generate necessary R and C++ into package src and inst directories.

Serialization for saving and loading compiled objects.



<https://github.com/USCI Lab/cereal>

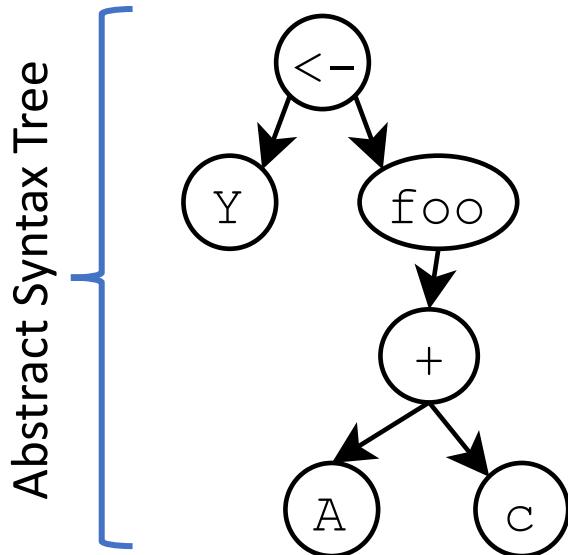
Also provided by

- **Rcereal** (Wu, Voorhees and Grant). On CRAN.

nCompiler generates Cereal code into nClass C++ code.

Extensibility

```
Y <- foo(A + c)
```

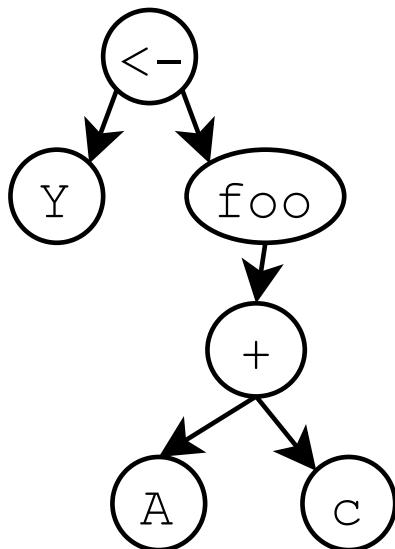


Compilation = clearly defined
traversals and transformations of
the tree.

Extensibility

```
Y <- foo(A + c)
```

Abstract Syntax Tree



Compilation = clearly defined
traversals and transformations of
the tree.

How to handle `<-`, `foo`, or `+`?

```
assignOperatorDef(  
  c('+', '-'),  
  list(  
    labelAbstractTypes = list(  
      handler = 'BinaryUnaryCwise',  
      returnTypeCode = returnTypeCo  
    eigenImpl = list(  
      handler = 'cWiseAddSub'),  
    cppOutput = list(  
      handler = 'BinaryOrUnary'),  
    testthat = list(  
      isBinary = TRUE,  
      testMath = TRUE,  
      testAD = TRUE)  
)
```

Questions?

<https://github.com/nimble-dev/nCompiler>

