Template Model Builder

Kasper Kristensen
kaskr@imm.dtu.dk

June 23, 2016

TMB Intro

- ADMB inspired R-package
- Combines external libraries: CppAD, Eigen, CHOLMOD
- Continuously developed since 2009, relatively few lines of code
- Implements Laplace approximation for random effects
- C++ Template based
- Automatic sparseness detection
- Parallelism through BLAS
- Parallel user templates
- Parallelism through `parallel` package

Example:

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() () {
  DATA_VECTOR(Y);
  PARAMETER_VECTOR(X);
  PARAMETER_VECTOR(logtheta);
  vector<Type> th=exp(logtheta);
  Type ans=0, m;
  for(int i=1; i<Y.size(); i++){
    m = X[i-1]+th[0]*(1.0-pow(exp(X[i-1])/th[2],th[1]));
    ans -= dnorm(X[i],m,sqrt(th[3]),true);
  }
  for(int i=0; i<Y.size(); i++){
    ans -= dnorm(Y[i],X[i],sqrt(th[4]),true);
  }
  return ans;
}
```

```
Y<-scan("theta.dat", quiet=TRUE)
library(TMB)
compile("theta.cpp")
dyn.load(dynlib("theta"))
data <- list(Y=Y)
param <- list(X=data$Y * 0, logtheta=c(0,0,6,0,0))
obj <- MakeADFun(data,param,random="X")
opt <- nlminb(obj$par,obj$fn,obj$gr)
```

Optimizations

User's code goes through several phases of optimization:

1. Cpp file is at compile time optimized via the Eigen library.
2. Operations are put on a tape.
3. Tape is runtime optimized by CppAD.

Thereafter the user's cpp function is no longer used.

- User need not worry about optimizing the code (e.g Passing objects by const reference or eliminating temporaries etc)

Laplace and sparse Hessian

- Recall the Laplace approximation of the negative log-likelihood:

$$-\log L^*(\theta) = -n \log \sqrt{2\pi} + \frac{1}{2} \log \det(H(\hat{u}, \theta)) + f(\hat{u}, \theta). \tag{1}$$

$$\hat{u}(\theta) = \arg\min_u f(u, \theta) \ . \tag{2}$$

We use $H(u, \theta)$ to denote the Hessian of $f(u, \theta)$ w.r.t. u

$$H(u, \theta) = f''_{uu}(u, \theta) \ . \tag{3}$$

- Sparseness pattern of H: Which entries are zero for any choice of prameters (u, θ) ?
- For efficiency: utilizing 'sparseness pattern' of $H(\theta)$ greatly reduces the time to calculate the log-determinant.
- Key feature of TMB: Can detect the pattern automatically.
- How? Numerical test is not sufficient. A 'symbolic' analysis is required.

Example:

- Simple test function:

$$f(u_1, \ldots, u_8) = u_1^2 + \sum_{i=2}^{8} (u_i - u_{i-1})^2 \; .$$

- Implementation in TMB (examp.cpp):

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  PARAMETER_VECTOR(u);
  Type ans=0;
  ans+=pow(u[0],2);
  for(int i=1;i<u.size();i++)ans+=pow(u[i]-u[i-1],2);
  return ans;
}
```

Run from R

- Compile and construct function object:

```
library(TMB)
compile("examp.cpp")
```

```
Note: Using Makevars in /home/kaskr/.R/Makevars
[1] 0
```

```
dyn.load(dynlib("examp"))
obj <- MakeADFun(data=list(),parameters=list(u=rep(0,8)),random="u")
```

- Get the Hessian:

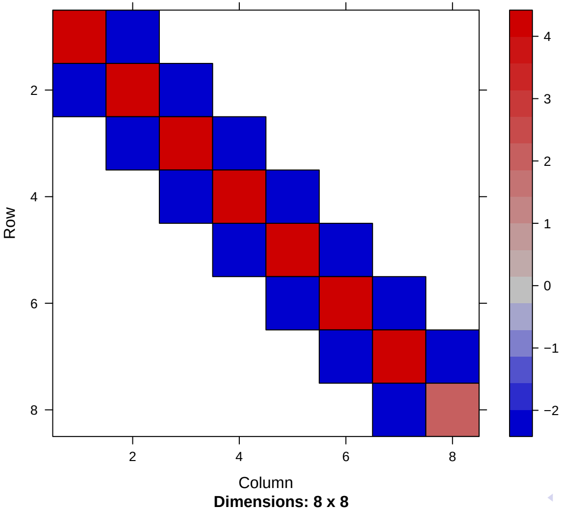
```
obj$env$spHess(random=TRUE)
```

```
8 x 8 sparse Matrix of class "dsCMatrix"

[1,]  4 -2  .  .  .  .  .  .
[2,] -2  4 -2  .  .  .  .  .
[3,]  . -2  4 -2  .  .  .  .
[4,]  .  . -2  4 -2  .  .  .
[5,]  .  .  . -2  4 -2  .  .
[6,]  .  .  .  . -2  4 -2  .
[7,]  .  .  .  .  . -2  4 -2
[8,]  .  .  .  .  .  . -2  2
```

Visualize from R

```
h <- obj$env$spHess(random=TRUE)
range <- c(0,nrow(h)) + .5
Matrix::image(h,xlim=range,ylim=range)
```



Comments

- TMB detected the correct banded structure: Hessian entries outside the band are zero for any u.
- Works by analysing the internal representation of the computional graph.
- This is currently a modification of the AD tool (CppAD).
- Take a look at the computational graphs...

Tape of the test function

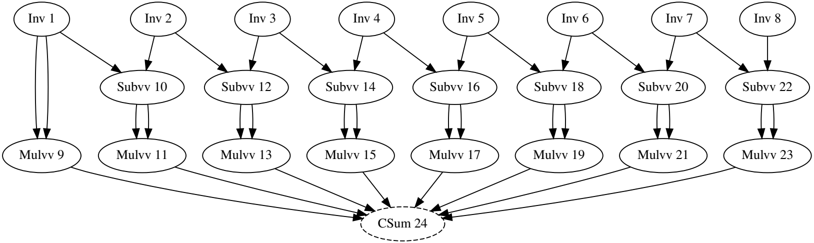


Figure: CppAD tape T1 for $f(u_1, \ldots, u_8) = u_1^2 + \sum_{i=2}^{8}(u_i - u_{i-1})^2$. Nodes "Inv 1"–"Inv 8" corresponds to $u_1, \ldots, u_8$ and node "CSum 24" corresponds to $f(u_1, \ldots, u_8)$.

Tape of the test function's gradient

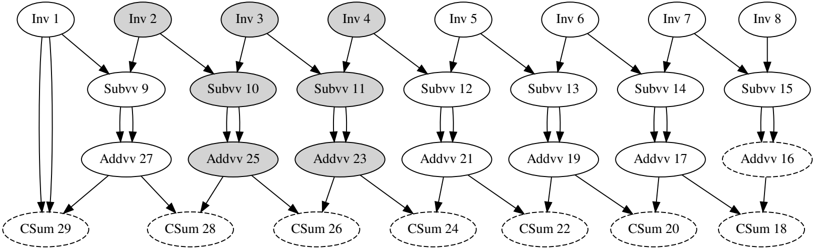


Figure: CppAD tape T2 for $f'(u)$, when $f(u)$ is defined as in previous figure. For example, node 26 corresponds to the partial of f w.r.t. u_3; i.e., $f_3'(u) = 2(u_3 - u_2) - 2(u_4 - u_3)$.

Summary sparsity detection

- Requiremnet: Fixed graph !
- Detection algorithm is a primitive dependency analysis: Never *false positives* but ther may be *false negatives* (e.g. $xy - xy$).
- Manual detection (like ADMB) does not have this problem !
- CONS: If statement flexibility.
- What happens to the sparsity pattern if an observation of the sum $\sum_i u_i$ is added:

$$x|u \sim N(\sum_i u_i, 1)$$

Atomic functions

- What is an atomic function?
- Example: 'ppois'

```
TMB_ATOMIC_VECTOR_FUNCTION(
    // ATOMIC_NAME
    ppois
    ,
    // OUTPUT_DIM
    1
    ,
    // ATOMIC_DOUBLE
    ty[0] = Rmath::Rf_ppois(
                 tx[0], tx[1], 1, 0);
    ,
    // ATOMIC_REVERSE
    Type value = ty[0];
    Type n = tx[0];
    Type lambda = tx[1];
    CppAD::vector<Type> arg(2);
    arg[0] = n - Type(1);
    arg[1] = lambda;
    px[0] = Type(0);
    px[1] = (-value + ppois(arg)[0])
         * py[0];
)
```

Definition

$$ppois(n, \lambda) = \sum_{k=0}^{n} \frac{\lambda^k}{k!} e^{-\lambda}$$

Partial derivatives:

$$\partial_n ppois(n, \lambda) = 0$$

$$\partial_\lambda ppois(n, \lambda) = -ppois(n, \lambda) + ppois(n-1, \lambda)$$

Overview of existing atomic functions

- Matrix multiply

```
atomic::matmul(x, y);   // T(grad) < 3 * T(func)
```

- Matrix exponential

```
atomic::matexp(x);      // T(grad) < 4 * T(func)
```

- Matrix inverse

```
atomic::matinv(x);      // T(grad) < 2.5 * T(func)
```

NOTE

- Atomic functions have dense sparsity pattern in TMB !
- Cheap gradient preserved for these atomic operations (tested $10^3 \times 10^3$).

Checkpointing

- Purpose: reduce memory of AD when the same operation sequence is repeated multiple times.
- This technique is known as *checkpointing*.
- Given a function

```
template<class Type>
vector<Type> work(vector<Type> parms) {
  // Long computation
  return ans;
}
```

- Generate 'work' and its derivatives as native symbols:

```
REGISTER_ATOMIC(work);
```

Other new stuff

- ?precompile
- ?gdbsource
- ?config
- ?as.list.sdreport
- ?sdreport -> bias.correction
- ?oneStepPredict
- ?MakeADFun -> profile
- namespace autodiff