Annotated Reading Notes:
TMB: Automatic Differentiation and Laplace Approximation
(Kristensen et al., 2016)

1 Introduction (Big picture)

Summary. TMB is an R package to implement latent-variable (random-effects) models efficiently by writing a C++ template for the joint negative log-likelihood $f(u, \theta)$. It obtains the Laplace approximation to the marginal likelihood by integrating out random effects via automatic differentiation (AD) and sparse linear algebra.

Why it matters. If you fit state-space, GLMM, GMRF, or other hierarchical models with many random effects, the bottleneck is often derivatives and sparse factorizations. TMB offloads both: you implement $f(u, \theta)$ once; TMB differentiates it and optimizes the Laplace objective automatically. Compared with ADMB, TMB leans on modern C++ libraries (CppAD, Eigen, CHOLMOD/Matrix, OpenMP/BLAS), which is key to speed and maintainability.

2 The Laplace approximation (Core math)

Setup. Let $u \in \mathbb{R}^n$ be random effects and $\theta \in \mathbb{R}^m$ parameters. Define $f(u, \theta)$ as the negative joint log-likelihood. The marginal likelihood is

$$L(\theta) = \int_{\mathbb{R}^n} \exp\{-f(u, \theta)\}\, du.$$

Let $\hat{u}(\theta) = \arg\min_u f(u, \theta)$ and $H(\theta) = \partial^2 f / \partial u^2 \big|_{u=\hat{u}(\theta)}$. The Laplace approximation reads

$$L^*(\theta) = (2\pi)^{n/2} \det\big(H(\theta)\big)^{-1/2} \exp\big(-f(\hat{u}(\theta), \theta)\big),$$

so the optimization target is $-\log L^*(\theta) = \frac{1}{2} \log\det H(\theta) + f(\hat{u}, \theta) + \text{const}$.

What to remember. (1) The *mode* $\hat{u}(\theta)$ and the *curvature* $H(\theta)$ drive the approximation. (2) You need derivatives w.r.t. both u (for the inner optimization) and θ (for the outer optimization). (3) Regularity: a unique, well-conditioned $\hat{u}$ and positive-definite H are essential near the optimum.

3 Automatic differentiation and CppAD (How derivatives are obtained)

Key idea. TMB records your C++ template as a computational graph ("tape") and applies forward/reverse AD to compute gradients, Hessian-vector products, and selected Hessian entries without hand-coding derivatives.

Practice tip. Reverse-mode AD makes *scalar* objectives cheap to differentiate ("cheap gradient principle"): one reverse sweep is a small multiple of the function cost. TMB also tapes derivative computations themselves to reach up to third-order derivatives efficiently when needed by the Laplace machinery.

4 Software implementation (What runs where)

Pipeline. In R: evaluate $-\log L^*(\theta)$ and its gradient. In C++: (i) solve for $\hat{u}(\theta)$, (ii) build sparse $H(\theta)$, (iii) factorize using CHOLMOD, (iv) compute log-det and selected inverse entries.

Design win. Sparse factorizations dominate time on large problems; use tuned/parallel BLAS (e.g. MKL) to accelerate CHOLMOD. Meanwhile, AD sweeps dominate when the template itself is heavy; OpenMP-parallel accumulation in TMB can split sums across cores.

5 Inverse subset algorithm (Why log-det and selected inverse are fast)

Identity. $\frac{\partial}{\partial \xi_i} \frac{1}{2} \log\det H(\xi) = \frac{1}{2} \operatorname{tr}\big(H(\xi)^{-1}\, \partial H(\xi)/\partial \xi_i\big)$. Since H is sparse, TMB only needs the entries of H^{-1} where H is nonzero (the "subset"). The inverse-subset algorithm transforms the Cholesky factor into those needed inverse entries cheaply.

Takeaway. You avoid forming a dense H^{-1}: compute *only* what the trace identity needs on the sparsity pattern. This preserves near "cheap-gradient" scaling for the Laplace gradient.

6 Automatic sparsity detection (Why you don't hand-code sparsity)

Mechanism. TMB infers which state-time blocks depend on which others by analyzing the derivative tape, then builds H's sparsity pattern automatically.

Modeling tip. Write the joint log-likelihood in a sum-of-local-contributions form (e.g. over times, sites, individuals). TMB will detect block-banded or GMRF-like patterns and exploit them in H.

7 Parallelization (Two levers)

BLAS/CHOLMOD. Parallel/tuned BLAS speeds sparse Cholesky and inverse subset. **OpenMP.** Parallel "accumulator" splits $f = \sum_k f_k$ across threads for AD and sparsity discovery.

Rule of thumb. If your profile shows $> 50\%$ time in Cholesky/inverse-subset, focus on BLAS. If it shows $> 50\%$ in AD sweeps, focus on OpenMP-parallelizing your template's sum.

8 Using TMB (What the user writes)

Workflow. (1) C++: write `objective_function<Type>` that returns $f(u, \theta)$. (2) R: `compile()`, `MakeADFun(data, parameters, random=...)`, then optimize `obj$fn` with `obj$gr` and inspect `sdreport`.

Check-list. Stable initial values for u and θ, sensible scaling (log/softplus/logit reparameterizations to enforce constraints), and guarding against invalid domains (e.g. `if(!R_FINITE(val)) return INFINITY;`) make optimizers happy.

9 Case studies & results (What speeds to expect)

Empirics. Across GLMMs, state-space, and spatial models, TMB achieved speedups from ~ 1.5 to ~ 100 vs. ADMB, with larger gains for larger/sparser random-effect structures.

Interpretation. When n (the dimension of u) is large and H is sparse, automatic sparsity plus fast Cholesky dominate the win. For tiny models or models without random effects, TMB and ADMB are comparable.

10 Small reproducible demo: Laplace vs. numerical integration in 1D

We illustrate the Laplace approximation in a one-dimensional setting purely in base R, to visualize the *mode + curvature* idea without compiling TMB.

Model

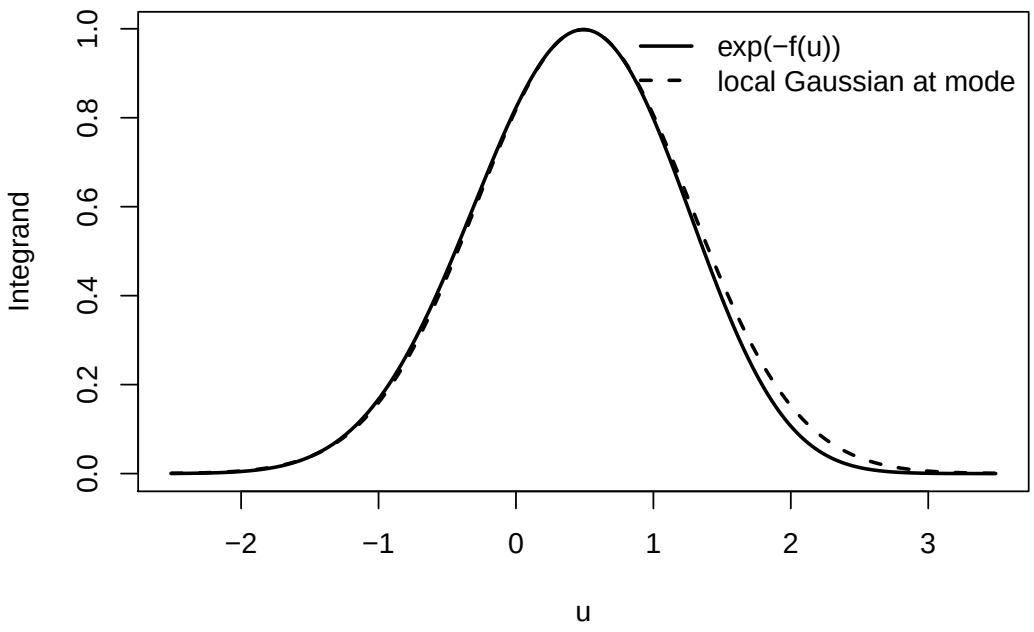
Consider the "joint" negative log-likelihood $f(u; \theta)$ for a non-Gaussian latent u:

$$f(u; \theta) = \frac{(u - \mu)^2}{2\sigma^2} + \alpha\, u^4,$$

with parameters $\theta = (\mu, \sigma, \alpha)$. We treat u as the random effect and the marginal likelihood is $L(\theta) = \int_{\mathbb{R}} e^{-f(u;\theta)}\, du$.

Computation: exact integral vs. Laplace

```r
# Parameters (mildly non-Gaussian)
mu <- 0.5; sigma <- 0.8; alpha <- 0.03

# f(u) and its derivatives
f  <- function(u) (u-mu)^2/(2*sigma^2) + alpha*u^4
df <- function(u) (u-mu)/(sigma^2) + 4*alpha*u^3
d2f<- function(u) 1/(sigma^2) + 12*alpha*u^2

# Find mode: minimize f(u)
u_hat <- uniroot(df, interval=c(-3,3))$root
H     <- d2f(u_hat)

# Exact (numerical) marginal likelihood
exact <- integrate(function(u) exp(-f(u)), lower=-Inf, upper=Inf,
                   rel.tol=1e-10)$value

# Laplace approximation: (2*pi)^{1/2} * H^{-1/2} * exp(-f(u_hat))
lap   <- sqrt(2*pi) * (1/sqrt(H)) * exp(-f(u_hat))

rel_err <- (lap - exact)/exact

# Plot the integrand and the local Gaussian at the mode
uu <- seq(u_hat-3, u_hat+3, length.out=600)
integrand <- exp(-f(uu))
gauss     <- exp( - (H/2)*(uu - u_hat)^2 ) * exp(-f(u_hat))

par(mar=c(4,4,1,1))
plot(uu, integrand, type="l", lwd=2, xlab="u", ylab="Integrand",
     main="")
lines(uu, gauss, lwd=2, lty=2)
legend("topright", bty="n",
       legend=c("exp(-f(u))", "local Gaussian at mode"),
       lwd=c(2,2), lty=c(1,2))
```




```
# Print a tiny summary
cat(sprintf("Mode u_hat = %.4f, curvature H = %.4f\n", u_hat, H))

## Mode u_hat = 0.4909, curvature H = 1.6492

cat(sprintf("Exact integral  = %.8f\n", exact))

## Exact integral  = 1.90014418

cat(sprintf("Laplace approx  = %.8f\n", lap))

## Laplace approx  = 1.94832849

cat(sprintf("Relative error  = %.4f\n", rel_err))

## Relative error  = 0.0254
```

What you see. The dashed curve is the local Gaussian using the mode and curvature; the filled curve is the true integrand. Even with a quartic perturbation ($\alpha > 0$), the Laplace value tracks the true integral closely when the mode dominates the mass. Misspecification grows as the distribution becomes flatter or more skewed near the mode.

11 Common pitfalls & practical advice

- **Non-unique or flat modes.** If $f(u, \theta)$ is poorly identified, H becomes ill-conditioned and the Laplace approximation degrades. Reparameterization and weakly-informative priors (in Bayesian settings) or constraints/penalties (in ML) can help.

- **Boundaries and transforms.** Use log/softplus/logit transforms in templates so positivity and simplex constraints are respected; this stabilizes both inner and outer optimizers.
- **Scaling.** Standardize covariates and rescale states so that H has reasonable condition number; this speeds Cholesky and reduces numerical noise in gradients.