Wavelet-Based Functional Mixed Model Analysis: Computation Considerations

Richard C. Herrick and Jeffrey S. Morris
Department of Biostatistics and Applied Mathematics
The University of Texas M. D. Anderson Cancer Center, Houston, TX

Abstract

Wavelet-based Functional Mixed Models (WFMM) is a new Bayesian method extending mixed models to irregular functional data (Morris and Carroll, 2006). These data sets are typically very large and can quickly run into memory and time constraints unless these issues are carefully managed in the software. In this paper we discuss optimization of a C++ implementation of WFMM and test its performance on a variety of functional data sets representative of biomedical applications. We reduced runtime and memory use by 1.) identifying and optimizing hotspots, 2.) using wavelet compression and other approximations to do less computation with minimal impact on results, and 3.) dividing the code into multiple executables to be run in parallel using a grid computing resource. We present rules of thumb for estimating memory requirements and computation times in terms of model and data set parameters. Finally we present examples and benchmarks demonstrating that it is practical to analyze very large data sets with readily available computing resources. This code is freely available on our website.

1. Introduction

WFMM can present a number of computational challenges when applied to the large spectral and imaging data sets that are frequently generated by modern biomedical experiments. Although input and output files can be on the order of gigabytes, they can be readily handled by today's low-cost hard drives. Physical memory, on the other hand, is still expensive and limited, and therefore has to be managed carefully. Larger data sets can easily require gigabytes of RAM and exceed the physical memory of the system. Depending on the data set, the design matrices applied, and number of MCMC samples required, execution time on a single processor can range from a few hours to several days.

This poster presents a C++ implementation of WFMM. Although scripting languages such as in Matlab, Splus, and R may be easier to develop statistical applications in, C++ provides a greater opportunity to achieve efficiencies and more economical use of memory since it is closer to the hardware and allows more performance tuning. Ease of programming can still be achieved through the construction of a good class library.

Representative Data Sets

- SELDI-TOF data set from organ-by-cell line experiment. Tumors from either A375P human melanoma or PC3MM2 prostate cancer cell lines were implanted in either the brain or lung of 16 nude mice. 32 spectra were obtained from blood serum samples.
- MALDI-TOF data set from organ-by-cell-line experiment. Tumors from one of 3 cell lines were implanted in one of 4 organs of 147 nude mice. Spectra were obtained from blood serum of these mice and 37 controls.
- MALDI-TOF data from pancreatic cancer experiment. Blood serum was taken from 139 pancreatic cancer patients and 117 controls.
- Planet Health Children's Activity Study data set was recorded with TriTrac activity monitor (Hemokentics, Inc., Madison,WI), which provides minute-by-minute acceleration counts in three-dimensions. Two data sets were benchmarked consisting of weekday and weekend complete accelerometer profiles.
- Rat colon carcinogenesis study – 30 rats were fed two diets rich in either fish-oil or corn oil, then sacrificed at 5 different time intervals after exposure to a carcinogen. MGMT levels were measured as a function of cell depth.
- 2D gel data sets –7 mice in three groups, "abstainers", "social drinkers", and "alcoholics", were sacrificed after a period of time and brain tissue from the amygdale region extracted, and run on 3 replicate 2d gels.

Wavelet-Based Functional Mixed Model

The wavelet-based functional mixed model is described in detail by Morris and Carroll (2006). Suppose we have a sample of N "observed functions", e.g. mass spectra, let $Y(t) = \{Y1(t), \ldots, YN(t)\}'$ be the N observed functions, "stacked" as rows.

Then sampling these functions on a common grid $\boldsymbol{t}=\{t1, \ldots, tT\}$ of length T gives rise to the discrete version of the functional mixed model given by:

$$\underbrace{Y}_{N\times \mathrm{T}} = \underbrace{X}_{N\times p}\underbrace{B}_{p\times \mathrm{T}} + \underbrace{Z}_{N\times m}\underbrace{U}_{m\times \mathrm{T}} + \underbrace{E}_{N\times \mathrm{T}},$$
$$U \sim MN(P,Q), E \sim MN(R,S)$$

P and R are correlation matrices (m x m and N x N, respectively). Q and S are T x T covariance matrices, and MN refers to the Matrix Normal distribution of Dawid(1981). X is the design matrix, B are fixed effect functions, Z*U are random effects, and E is residual error.

Rather than fitting this model to Y directly, we project the data into the wavelet space, and do the modeling there. Wavelets are orthogonal basis functions that can be used to represent functions. The projection can be represented by orthogonal projection D=YW' to each row of Y. The wavelet space version of the functional mixed model is:

$$\underbrace{D}_{N\times \mathrm{T}} = \underbrace{X}_{N\times p}\underbrace{B^*}_{p\times \mathrm{T}} + \underbrace{Z}_{N\times m}\underbrace{U^*}_{m\times \mathrm{T}} + \underbrace{E^*}_{N\times \mathrm{T}},$$
$$U^* \sim MN(P,Q^*), E^* \sim MN(R,S^*)$$

B*=BW' & U*=UW' are wavelet coefficients for fixed/random effect functions. We assume Q* and S* are diagonal, implicitly

assuming wavelet coefficients within a given function are independent.

We put shrinkage priors on the elements of B* that consist of a mixture of a point mass at zero and a normal distribution. When placed on wavelet coefficients, this type of prior results in functional estimates that are denoised but without substantial attenuation of the peaks of the function. We put vague priors on the Ω, the set of parameters indexing the covariance matrices P, Q*, R, and S*. Given estimates of B* and U*, we can easily obtain B=B*W and U=U*W.

Given *Y, X, Z,* wavelet basis, and choice of structure for P and R, we can use MCMC to obtain posterior samples of the fixed and random effect functions *B(t)* and *U(t)* and covariance matrices/surfaces *P, Q, R,* and *S* on the grid ***t*** : Working with the likelihood w/ U* integrated out, repeat for $g=1, \ldots G$ iterations: 1.) Sample from $f(B^*|D,\Omega)$, which are mixtures of normals and point masses at 0. 2.) Sample from $f(\Omega|D,B^*)$, which is done using Metropolis-Hastings steps. Then use IDWT to obtain posterior samples of *B(t)* on grid ***t*** from *B**.

Application Architecture

The computation readily separates into three components: Initialization, MCMC Loop, and Summarization. These are linked into a single executable , WFMM, for serial execution on a single PC or into three executables, WFMM1, WFMM2, and WFMM3 for parallel computing. A Windows bat file runs WFMM1 to compute initial parameters, then submits WFMM2 to N processors using the grid computing facility Condor (www.cs.wisc.edu/condor) and waits for them to finish. It then runs WFMM3 to summarize the results.

WFMM1 (Initialization): Apply discrete wavelet transform (DWT) to each function to obtain sets of wavelet coefficients. Compute starting values for fixed effect functions and variance components using maximum likelihood; estimate variance of MLE for variance components to use as proposal variance in random-walk Metropolis-Hastings step. Use Empirical Bayes method to estimate smoothing parameters from the data, and set up vague, proper priors with 1 unit of information for each variance component.

WFMM2(MCMC Loop): Random Walk Metropolis Hastings sampling of variance components. Gibbs sampling of Beta (mixtures of point masses at zero and Normals).

WFMM3(Summarization): Apply inverse DWT (IDWT) to MCMC samples of wavelet coefficients for fixed effect functions to obtain MCMC samples of fixed effect functions. Compute mean, pointwise variance, and pointwise credible intervals for fixed effect functions.

2. Methods

The WFMM application was implemented as a Windows console application in C++, using Visual Studio 2005 as the IDE. A class library developed internally was used to provide the vector and matrix handling as well as random-number generation. Linear algebra operations were provided by Intel's Math Kernel Library 8.1. Benchmarks were run on a Dell Precision Workstation 650 with Intel Xeon 2.4 GHz processor and 2 GB RAM.

Software Tools

The following applications and system functions were used to improve performance and minimize memory usage:

- VTune Performance Analyzer (Intel Corp.): Estimates time spent on each instruction to quickly identify hot spots.
- Timing Macros (Windows): Computes elapsed time for each component in order to provide benchmarks.
- Perfmon: (Windows): Runs concurrently with WFMM to monitor available memory in real-time.
- GetMemoryStatus function (Windows): Allows WFMM to determine how much memory is available.
- _CrtDumpMemoryLeaks() (Windows): Checks for memory not being deallocated from the heap and which line of code allocated the memory.

Optimizing Runtime

VTune and timing macros identified the most time-consuming function, which was computation of the generalized cross products (GCP) . We then:

- Identified the most efficient algorithm, which involved computing the generalized cross products X'Σjk-1X, X'Σjk-1djk, etc. using a SWEEP operator (a la PROC MIXED, Wolfinger, et al. 1994). This avoids computing large inverses or matrix multiplications. GCP's are sufficient pivotal quantities for updating the fixed effect functions, and variance components in the Metropolis-Hastings computation.
- Made sure the 10% of code that takes up 90% of CPU time (the SWEEP step for updating the GCP) is coded as efficiently as possible in C++.
- Used VTune to identify the top 10 to 15 most time consuming functions; this method identified the new operator as a major time consumer because large buffers were allocated locally. Buffers were then made persistent and reused each step to save time.

Optimizing Memory Usage

The point of maximum memory allocation was determined with perfmon during a WFMM run. Breakpoints can be set in the debugger if necessary to freeze execution at a specific line. Then the following steps were taken to maximize available memory at that point:

- Used _CRTDumpMemoryLeaks() system call to check for memory leaks in case some lower level functions are not releasing memory.
- Ensured memory buffers were deallocated when no longer needed so that subsequent objects could use it.
- Avoided double buffering when possible, such as passing by reference rather than returning a large array on the stack.

Computational Shortcuts

These were implemented to save memory and CPU time. The premise is that significant computation is spent on elements that have little influence on the final results. Since their effect is data dependent, they are controlled by the user:

- Wavelet compression; Sort wavelet coefficients in decreasing magnitude, then keep only the K' largest coefficients that pre-

serve a specified percentage of the total sum squared data values. Fill the dropped coefficients with 0's on reconstruction. This easily speeds up computations by a factor of 5 to 100, without substantially impacting results in many cases.

- Set qjk =0 and do not update in MCMC if the MLE starting value is less than a specified threshold (10-6). This saves considerable time without strongly impacting results, since we expect parsimony in the wavelet space for the covariance parameters.
- For other qjk that are updated in the MCMC, if the current value is very small, especially relative to sjk, then when using the sweep operator to update the GCP's, substitute qjk=0. Again, this results in substantial savings and qjk is so small that the GCP remain virtually the same.

3. Results

Estimating Memory and Runtime

Since the large data sets often fit with this method present tremendous challenges in memory requirements, computation times, and disk space, these should be checked before implementing this method on a given system. Memory requirements and computation times can be estimated from data set size and design matrix parameters:

N = Number of Functions
T = Number of observations/function
B = Number of MCMC samples
K = Number of wavelet coefficients
p = Number of fixed effect functions
K' = Number of non-thresholded wavelet coefficients
m = Number of random effect functions
H = Number of levels of random effect functions
c = Number of strata for residual error functions
B'= Number of samples in memory

Disk Usage ≈ 8 [BK'(p+H+c+1) + pT(B+4) + 3NT+ 2NK' + K'(p^2+m^2+pm+m+3+7p+7(H+c))]

RAM Usage ≈ 8[B'(2pT+(H+c+2)K'+T) + (0.05B+4)pT + 3(H+c)K']

Total runtime is more difficult to estimate but its dependence on parameters can be inferred by inspection; runtime is dominated by the MCMC loop and in particular by the sweep calculation; initialization and postprocessing runtimes have not been a significant consideration.

Runtime depends linearly on number of iterations = B*(thin)+burnin and the number of nonthresholded wavelet coefficients K'. From the sweep calculation, we can estimate runtime dependence by estimating floating point operations as a function of the swept matrix size and number of sweeps (m).

$$\text{Sweep Runtime} \sim K'm(p+2m+1)^2, \text{ or } O(K'm^3)$$
$$\text{for } m >> p \text{ and } O(K'p^2) \text{ for } p >> m.$$

Actual Runtime and Memory Usage

Table 1 gives measured runtimes, RAM, and hard disk usage for seven functional data sets and demonstrates effect of parameters. Runtimes are based on 11,000 iterations (burnin=1000, thin=5, B=2000) and K' = K ~ T for all cases except 2D Gel, where wavelet compression was used to reduce K' to 9529. Note strong dependence of runtime and RAM on m demonstrated by TriTrac weekday set and effect of large K demonstrated by proteomics and 2D Gel data sets.

	N	T	m	p	Runtime (hr/min)			RAM(MB)			HD(MB)
					Init	MCMC	Post	Init	MCMC	Post	
Rat colon carcinogenesis	256	256	30	13	0/1	1/8	0/1	42	97	148	138
TriTrac weekday	237	660	148	4	0/38	125/2	0/1	157	271	92	136
TriTrac weekend	52	540	45	4	0/2	1/1	0/1	28	103	144	112
Pancreatic Cancer MALDI	256	12096	0	5	0/11	3/7	0/9	132	840	930	2532
Organ-by-Cell-Line SELDI	32	7985	16	5	0/15	10/50	0/6	78	1093	600	1629
Organ-by-Cell-Line MALDI	184	6518	0	14	0/12	3/15	0/13	62	901	1211	3079
2D Gel data	57	490448	21	3	0/55	15/39	4/15	807	803	1569	24034

Conclusions

- It is practical to analyze even very large data sets with readily available computing resources using WFMM methodology.
- This application is available for free download from: http://odin.mdacc.tmc.edu/~jeffmo/papers_files/wfmm_supplement.html

References

1. Morris JS and Carroll RJ: Wavelet-Based Functional Mixed Models. *Journal of the Royal Statistical Society, Series B*, 68(2): 179-199, 2006.
2. Wolfinger, R., Tobias, R. and Sall, J. (1994) Computing Gaussian likelihoods and their derivatives for general linear mixed models. *SIAM J. Scient. Comput.*, **15**, 1294–1310.j
3. Dawid, A.P. (1981). Some matrix-variate distribution theory: Notational considerations and a Bayesian application. *Biometrika*, 68, 265-274.