GPUs and the computational efficiency of Gaussian process based models

Colin Rundel

April 25, 2016

Duke University

What is a Gaussian Process

A statistical distribution that describes observations that arise from a continuous domain (e.g. space, time), whereby any subset of points within that domain have a multivariate normal distribution.

$$\underset{n\times 1}{X} \sim \mathcal{N}(\underset{n\times 1}{\mu}, \underset{n\times n}{\Sigma})$$

In general, we can think about the Gaussian process as a way of representing an infinite dimensional object (e.g. a smooth continuous surface over a region of space) that we observe at finite locations.

The problem with GPs ...

Gaussian process models are difficult to scale to large problems:

Evaluate the (log) likelihood?

$$-\frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(x-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu}) - \frac{n}{2}\log 2\pi \qquad \mathcal{O}\left(n^3\right)$$

Want a sample?

$$\boldsymbol{\mu} + \text{Chol}(\boldsymbol{\Sigma}) \times \boldsymbol{Z} \text{ with } Z_i \sim \mathcal{N}(0,1) \qquad \mathcal{O}\left(n^3\right)$$

Update covariance parameter?

$$\{\Sigma\}_{ij} = \sigma^2 \exp(-\{d\}_{ij}\phi) \qquad \mathcal{O}\left(n^2\right)$$

A simple guide to computational complexity

Linear complexity?

A simple guide to computational complexity

Linear complexity? - Go for it

A simple guide to computational complexity

Linear complexity? - Go for it

Quadratic complexity?

A simple guide to computational complexity

Linear complexity? - Go for it

Quadratic complexity? - Pray

A simple guide to computational complexity

Linear complexity? - Go for it

Quadratic complexity? - Pray

Cubic complexity?

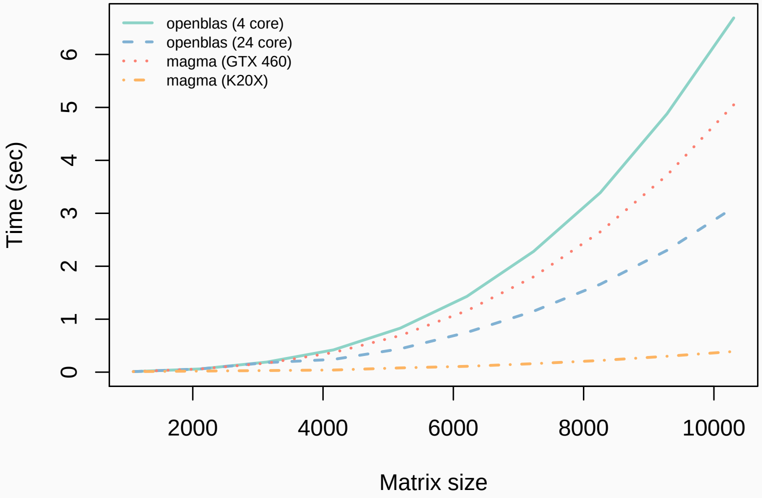
A simple guide to computational complexity

Linear complexity? - Go for it

Quadratic complexity? - Pray

Cubic complexity? - Give up

Improving Cholesky



Tools and Optimization

$$\begin{tikzcd}
\text{R} \arrow[r, "\text{Rcpp*}"] \arrow[rr, bend right, "\text{HiPLAR, gmatrix}"'] & \text{C++} \arrow[r, "\text{magma, cublas}"] & \text{CUDA}
\end{tikzcd}$$

Regardless of tools or workflow, measuring / profiling performance is critical.

Background

Using intrinsic markers (genetic and isotopic signals) for the purpose of inferring migratory connectivity.

- Existing methods are too coarse for most applications
- Large amounts of data are available (>150,000 feather samples from >500 species)
- Genetic assignment methods are based on Wasser, et al. (2004)
- Isotopic assignment methods are based on Wunder, et al. (2005)

Data - DNA microsatellites and $\delta^2\mathrm{H}$

Hermit Thrush
(*Catharus guttatus*)

- 138 individuals
- 14 locations
- 6 loci
- 9-27 alleles / locus

Wilson's Warbler
(*Wilsonia pusilla*)

- 163 individuals
- 8 locations
- 9 loci
- 15-31 alleles / locus

Allele Frequency Model

For the allele *i*, from locus *l*, at location *k*

$$\boldsymbol{y}_{\cdot lk} | \boldsymbol{\Theta} \sim \text{Mult}\left(\textstyle\sum_i y_{ilk},\ \boldsymbol{f}_{\cdot lk}\right)$$

$$f_{ilk} = \frac{\exp(\Theta_{ilk})}{\sum_i \exp(\Theta_{ilk})}$$

$$\boldsymbol{\Theta}_{il} | \boldsymbol{\alpha}, \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}_{il},\ \boldsymbol{\Sigma})$$

$$\{\Sigma\}_{ij} = \alpha_0\ \exp\left(-\left(\{d\}_{ij}/\alpha_1\right)^{\alpha_2}\right) + \alpha_3\, \mathbb{1}_{i=j}$$

Genetic Assignment Model

Assignment model using Hardy-Weinberg equilibrium allowing for genotyping (δ) and single amplification (γ) errors.

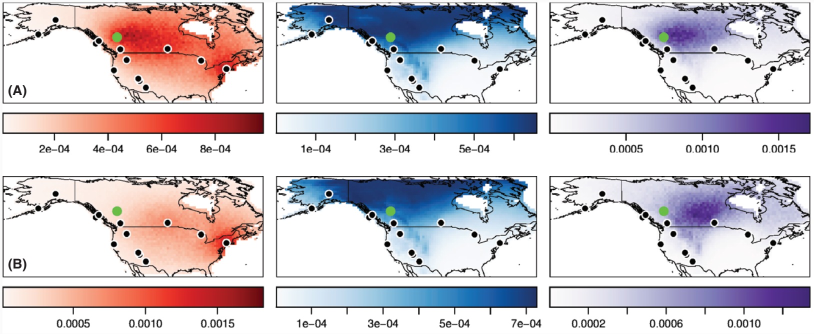
$$P(S_G|\boldsymbol{f}, k) = \prod_l P(i_l, j_l|\boldsymbol{f}, k)$$

$$P(i_l, j_l|\boldsymbol{f}, k) = \begin{cases} \gamma P(i_l|\boldsymbol{f}, k) + (1-\gamma)P(i_l|\tilde{\boldsymbol{f}}, k)^2 & \text{if } i = j \\ (1-\gamma)P(i_l|\boldsymbol{f}, k)P(j_l|\boldsymbol{f}, k) & \text{if } i \neq j \end{cases}$$

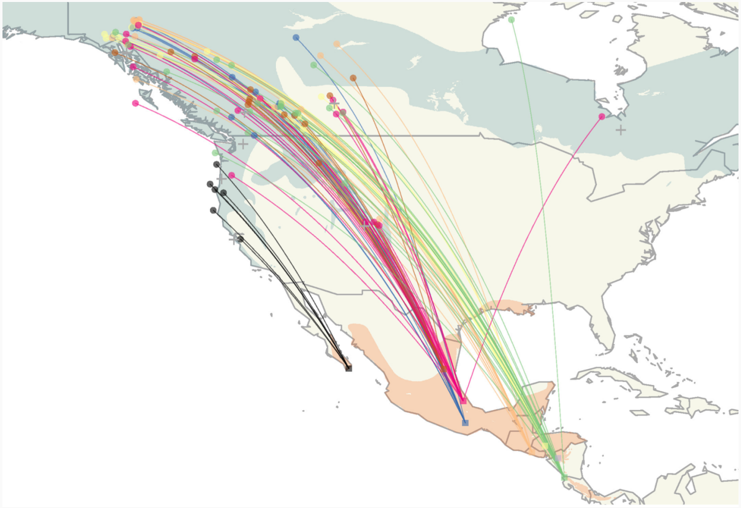
$$P(i_l|\boldsymbol{f}, k) = (1-\delta)f_{lik} + \delta/m_l$$

Combined Model

Genetic Isotopic Combined



Migratory Connectivity



Implementation

Model fitting is done via MCMC (MH within Gibbs)

- Original implementation in pure C++ with minimal dependencies (Wasser, et al. (2004))
- Rewritten using R / C++ via Rcpp(Armadillo)
  - Code closer to matrix notation (and R)
  - Transparent use of high performance LAPACK implementations
  - R Package - isoscatR - `https://github.com/rundel/isoscatR`
- Model fitting performance is quite good
  - 300,000 iterations in $\sim$ 5.5 minutes
- Bottleneck in drawing posterior predictive samples
  - 1,000 iterations in $\sim$ 30 minutes

Prediction details

Why is the prediction slow?

Prediction details

Why is the prediction slow?

Predicting allele frequencies for Hermit thrush at 3318 novel locations, in order to do this we sample from:

$$\boldsymbol{\Theta}_p | \boldsymbol{\Theta}_m \sim \mathcal{N}(\boldsymbol{\mu}_p + \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\Theta}_m - \boldsymbol{\mu}_m),\ \boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$$

Prediction details

Why is the prediction slow?

Predicting allele frequencies for Hermit thrush at 3318 novel locations, in order to do this we sample from:

$$\boldsymbol{\Theta}_p | \boldsymbol{\Theta}_m \sim \mathcal{N}(\boldsymbol{\mu}_p + \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\Theta}_m - \boldsymbol{\mu}_m),\ \boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$$

Algorithm steps

1. Calculate $\boldsymbol{\Sigma}_{pm}$, $\boldsymbol{\Sigma}_p$, and $\boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp}$
2. Calculate Chol$(\boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$
3. Sample from MVN
4. Calculate allele frequencies

Posterior predictive sampling timings

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Performance
1.	Covariances	1.080	0.046	23.0
2.	Cholesky	0.467	0.208	2.3
3.	Sample	0.049	0.052	0.9
4.	Allele Freq	0.129	0.127	1.0
	Total	1.732	0.465	3.7

Posterior predictive sampling timings

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Performance
1.	Covariances	1.080	0.046	23.0
2.	Cholesky	0.467	0.208	2.3
3.	Sample	0.049	0.052	0.9
4.	Allele Freq	0.129	0.127	1.0
	Total	1.732	0.465	3.7

Total run time:

- CPU - 28.9 mins
- CPU+GPU - 7.8 mins

Posterior predictive sampling timings

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Performance
1.	Covariances	1.080	0.046	23.0
2.	Cholesky	0.467	0.208	2.3
3.	Sample	0.049	0.052	0.9
4.	Allele Freq	0.129	0.127	1.0
	Total	1.732	0.465	3.7

Total run time:

- CPU - 28.9 mins
- CPU+GPU - 7.8 mins

$\times$ CV runs $\begin{bmatrix}\text{166 for Hermit Thrush}\\ \text{179 for Wilson's Warbler}\end{bmatrix}$

Lessons

Relatively small changes in one function resulted in 3 - 4x improvement

- Cross validation results in two days instead of a week
- 1-2 weeks of implementation, 1 week of tweaking / testing
- Started with Cholesky, other optimizations followed

Lessons

Relatively small changes in one function resulted in 3 - 4x improvement

- Cross validation results in two days instead of a week
- 1-2 weeks of implementation, 1 week of tweaking / testing
- Started with Cholesky, other optimizations followed

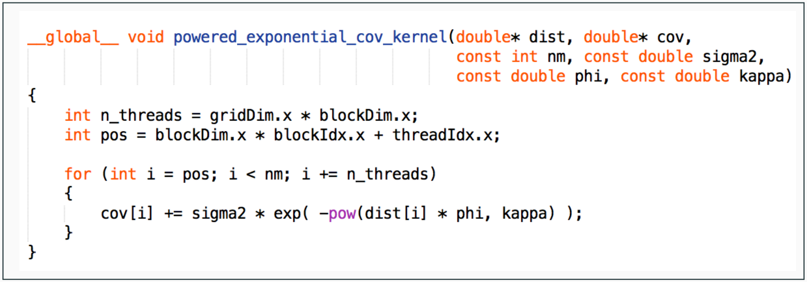
Issues:

- External library dependency makes package development (much) more complicated
- Additional code verbosity and complexity

Improving Covariance Calculations

Covariance is assumed to be stationary and isotropic

- Elements of the covariance matrix can be calculated independently
- Small scale “embarrassingly parallel”
- Implementation is straight forward
  (if we don’t worry about things like symmetry)



```
__global__ void powered_exponential_cov_kernel(double* dist, double* cov,
                                               const int nm, const double sigma2,
                                               const double phi, const double kappa)
{
    int n_threads = gridDim.x * blockDim.x;
    int pos = blockDim.x * blockIdx.x + threadIdx.x;

    for (int i = pos; i < nm; i += n_threads)
    {
        cov[i] += sigma2 * exp( -pow(dist[i] * phi, kappa) );
    }
}
```

Building core tools

Common set of (expensive) tasks for GP models

- Covariance calculation
- Cholesky of Cov.
- Inverse of Cov.

Goal is to make performing these tasks on a GPU as painless as possible and allow interoperability with GPU (magma, CUBLAS) and CPU (Armadillo) libraries.

- GPU matrix class
- Modern resource management (RAII, move semantics)
- Simple translation between GPU and CPU memory

R Package - RcppGP - `https://github.com/rundel/RcppGP`

CPU vs GPU code

```cpp
arma::mat prop_Sigma = arma::exp(-prop_phi * d_CIF);
arma::mat prop_Sigma_U = arma::chol(prop_Sigma);

double prop_Sigma_log_det = 2*arma::accu(arma::log(prop_Sigma_U.diag()));

arma::mat prop_Sigma_U_inv = arma::inv(arma::trimatu(prop_Sigma_U));
arma::mat prop_Sigma_inv = prop_Sigma_U_inv * prop_Sigma_U_inv.t();
```

```cpp
exponential_cov_gpu(d_CIF_gpu.mat, cov_gpu.mat, nr_CIF, nr_CIF, 1.0, prop_phi, 64);
cov_gpu.chol('L',false);

double prop_Sigma_log_det = 2*arma::accu(arma::log(cov_gpu.get_mat().diag()));

cov_gpu.inv_chol('L',true);
arma::mat prop_Sigma_inv = cov_gpu.get_mat();
```

Back

Background

Fine particulate matter ($PM_{2.5}$) is an EPA regulated air pollutant linked to a variety of adverse health effects

- Classified based on particle size (< 2.5 μm diameter)
- Major species: Sulfate, Nitrate, Ammonium, Soil, Carbon.
- Minor species: trace elements (K, Mg, Ca), heavy metals (Cu, Fe), etc.
- Complex spatio-temporal dependence between species

Data

Speciated $PM_{2.5}$ Sources

- Chemical Speciation Network (CSN) - 221 stations
- Interagency Monitoring of Protected Visual Environments (IMPROVE) - 172 stations

Total $PM_{2.5}$ Sources

- Federal Reference Method (FRM) - 949 stations

Model Output

- Community Multi-scale Air Quality (CMAQ) - 12 km grid

Data Issues

- Monitoring frequency
- Total vs Sum of Species

Species Model Details

For the 5 major species (Sulfate, Nitrate, Ammonium, Soil, Carbon) and the two networks (CSN, IMPROVE):

$$C_t^i(\mathbf{s}) = Z_t^i(\mathbf{s}) + \epsilon_{C,t}^i(\mathbf{s})$$
$$I_t^i(\mathbf{s}) = Z_t^i(\mathbf{s}) + \epsilon_{I,t}^i(\mathbf{s})$$

where $Z_t^i(\mathbf{s})$ are the latent "true" concentrations of species i at time t and locations $\mathbf{s}$, and is given by

$$Z_t^i(\mathbf{s}) = \max\left(0,\ \widetilde{Z}_t^i(\mathbf{s})\right)$$
$$\widetilde{Z}_t^i(\mathbf{s}) = \beta_{0,t}^i + \beta_{0,t}^i(\mathbf{s}) + \beta_{1,t}^i\, Q_t^i(B_{\mathbf{s}})$$

Total $PM_{2.5}$ Model Details

For total $PM_{2.5}$ from the three networks (CSN, IMPROVE, FRM):

$$\begin{aligned} C_t^{tot}(\mathbf{s}) &= Z_t^{tot}(\mathbf{s}) + \epsilon_{C,t}^{tot}(\mathbf{s}) \\ I_t^{tot}(\mathbf{s}) &= Z_t^{tot}(\mathbf{s}) + \epsilon_{I,t}^{tot}(\mathbf{s}) \\ F_t^{tot}(\mathbf{s}) &= Z_t^{tot}(\mathbf{s}) + \epsilon_{F,t}^{tot}(\mathbf{s}) \end{aligned}$$

where $Z_t^{tot}(\mathbf{s})$ are the latent “true” concentration of total $PM_{2.5}$ at time t and locations $\mathbf{s}$, which is given by the sum of the major species and the “other” species concentrations.

$$Z_t^{tot}(\mathbf{s}) = \sum_{i=1}^{5} Z_t^i(\mathbf{s}) + Z_t^o(\mathbf{s})$$

$$Z_t^o(s) = \max\left(0,\ \widetilde{Z}_t^o(s)\right) \qquad \widetilde{Z}_t^o(\mathbf{s}) = \beta_{0,t}^o + \beta_{0,t}^o(\mathbf{s}) + \beta_{1,t}^o\, Q_t^o(B_{\mathbf{s}})$$

Spatial Dependence

Spatial dependence enters the model through the $\beta^i_{0,t}(s)$ parameters for $i \in \{o, 1, 2, 3, 4, 5\}$.

$$\beta^i_{0,t}(\boldsymbol{s}) = \sigma^i_t \, w^i_t(\boldsymbol{s})$$

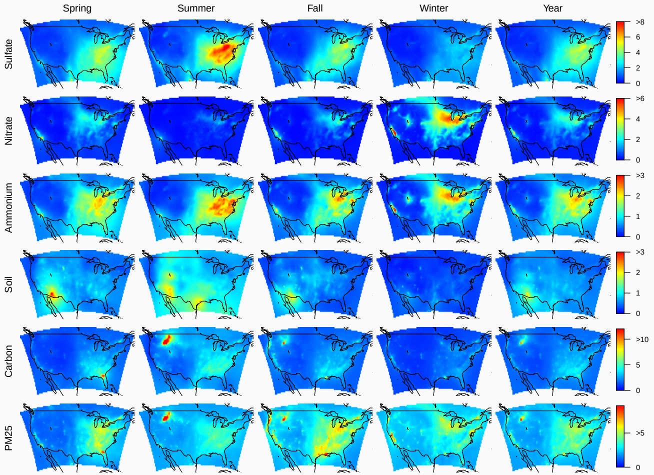
where $w^i_t(\boldsymbol{s})$ are zero mean, variance 1, Gaussian processes with exponential correlation given by

$$\mathrm{corr}(w^i_t(\boldsymbol{s}), w^i_t(\boldsymbol{s}')) = \exp(-\phi^i_t |\boldsymbol{s} - \boldsymbol{s}'|)$$

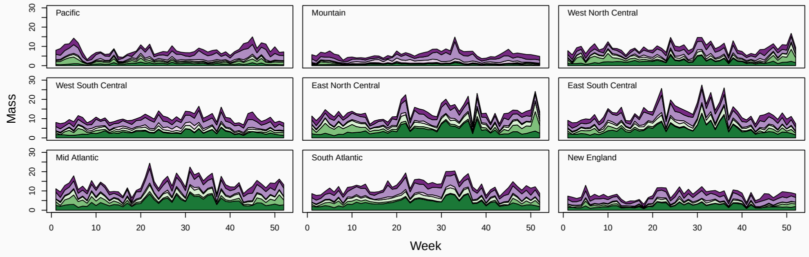
Additional dependence between species is introduces via coregionalization,

$$\begin{pmatrix} \beta^i_{0,t}(\boldsymbol{s}) \\ \beta^j_{0,t}(\boldsymbol{s}) \end{pmatrix} = \boldsymbol{A}_t \begin{pmatrix} w^i_t(\boldsymbol{s}) \\ w^j_t(\boldsymbol{s}) \end{pmatrix}.$$

Model results



Model results



■ Sulfate ■ Nitrate □ Ammonium □ Soil ■ Carbon ■ Other

MCMC performance

Parameter	CPU (secs)	CPU+GPU (secs)	Rel. Performance
$\beta_0,\ \beta_1$	0.00029	0.00030	0.97
$\beta_0(s)$	0.09205	0.09132	1.00
σ^2	0.00383	0.00385	0.99
ϕ	0.46084	0.25174	1.83
$\tau_i^2,\ \tau_{tot}^2$	0.00003	0.00003	1.00
Total	0.55708	0.34729	1.60

Run times

Total run time for model fitting (50,000 iterations):

- CPU - 7.7 hours
- CPU+GPU - 4.8 hours

$\times$ 52 weeks

Run times

Total run time for model fitting (50,000 iterations):

- CPU - 7.7 hours
- CPU+GPU - 4.8 hours

$\times$ 52 weeks

Total run time for model prediction at 5950 locations (1,000 iterations):

- CPU - 7.2 hours
- CPU+GPU - 4.3 hours

$\times$ 52 weeks

Run times

Total run time for model fitting (50,000 iterations):

- CPU - 7.7 hours
- CPU+GPU - 4.8 hours

$\times$ 52 weeks

Total run time for model prediction at 5950 locations (1,000 iterations):

- CPU - 7.2 hours
- CPU+GPU - 4.3 hours

$\times$ 52 weeks

One run takes about 775 hours (4.6 days) total on CPU alone, 473 (2.8 days) on CPU and GPU.

Run times

Total run time for model fitting (50,000 iterations):

- CPU - 7.7 hours
- CPU+GPU - 4.8 hours

$\times$ 52 weeks

Total run time for model prediction at 5950 locations (1,000 iterations):

- CPU - 7.2 hours
- CPU+GPU - 4.3 hours

$\times$ 52 weeks

One run takes about 775 hours (4.6 days) total on CPU alone, 473 (2.8 days) on CPU and GPU.

$\times$ 3 model variants

$\times$ 10 for cross validation

Lessons

Established infrastructure makes a huge difference in development time

- 1 hour to go from CPU implementation to CPU+GPU implementation
- Code shown previously is 2/3 of the changes necessary
  Code

In practice, was easier to run CPU only code across more servers (configuration time / effort)

- Not possible (or at least easy) for models variants that are not independent in time.
- There are $\sim$ 20 desktops with GPUs available in the department (available via Condor)

Low rank approximations

For a Gaussian process

$$Y(\boldsymbol{s}) = x(\boldsymbol{s})'\,\boldsymbol{\beta} + w(\boldsymbol{s}) + \epsilon, \quad \epsilon \sim N(0,\ \tau^2\, I)$$
$$w(\boldsymbol{s}) \sim N(0,\ \boldsymbol{C}(\boldsymbol{s})), \quad \boldsymbol{C}(\boldsymbol{s}, \boldsymbol{s}') = \sigma\, \rho(\boldsymbol{s}, \boldsymbol{s}'|\theta)$$

if we can approximate $\boldsymbol{C}(\boldsymbol{s})$ with a low rank approximation with the form $\boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}'$ where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $n \times k$ and $\boldsymbol{S}$ is $k \times k$.

Low rank approximations

For a Gaussian process

$$Y(\boldsymbol{s}) = x(\boldsymbol{s})'\,\boldsymbol{\beta} + w(\boldsymbol{s}) + \epsilon, \quad \epsilon \sim N(0,\ \tau^2\, I)$$
$$w(\boldsymbol{s}) \sim N(0,\ \boldsymbol{C}(\boldsymbol{s})), \quad \boldsymbol{C}(\boldsymbol{s}, \boldsymbol{s}') = \sigma\,\rho(\boldsymbol{s}, \boldsymbol{s}'|\theta)$$

if we can approximate $\boldsymbol{C}(\boldsymbol{s})$ with a low rank approximation with the form $\boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}'$ where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $n \times k$ and $\boldsymbol{S}$ is $k \times k$.

We can the use of the Sherman-Morrison-Woodbury formula for the inverse (and determinant),

$$\boldsymbol{C}(\boldsymbol{s})^{-1} \approx \left(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}'\right)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{S}^{-1} + \boldsymbol{V}'\boldsymbol{A}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}'\boldsymbol{A}^{-1}.$$

Gaussian Predictive Processes

For a rank k approximation,

- Pick k knot locations $\boldsymbol{s}^*$
- Calculate knot covariance ($\boldsymbol{C}(\boldsymbol{s}^*)$) and knot cross-covariance ($\boldsymbol{C}(\boldsymbol{s}^*)^{-1}$)
- Approximate full covariance

$$C(s) \approx C(s, s^*)\, C(s^*)^{-1}\, C(s^*, s).$$

- Systematically underestimates variance, inflates τ^2.
- Modified predictive process corrects this using

$$C(s) \approx C(s, s^*)\, C(s^*)^{-1}\, C(s^*, s) + \text{diag}\Big(C(s) - C(s, s^*)\, C(s^*)^{-1}\, C(s^*, s)\Big).$$

Banerjee, Gelfand, Finley, Sang (2008), Finley, Sang, Banerjee, Gelfand (2008)

Low Rank Approximations via Random Projections

1. Starting with an $m \times n$ matrix $\boldsymbol{A}$.
2. Draw an $n \times k + p$ Gaussian random matrix $\boldsymbol{\Omega}$.
3. Form $\boldsymbol{Y} = \boldsymbol{A}\,\boldsymbol{\Omega}$ and compute its QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\,\boldsymbol{R}$
4. Form the $k + p \times n$ matrix $\boldsymbol{B} = \boldsymbol{Q}'\,\boldsymbol{A}$.
5. Compute the SVD of the small matrix $\boldsymbol{B}$, $\boldsymbol{B} = \hat{\boldsymbol{U}}\,\boldsymbol{S}\,\boldsymbol{V}'$.
6. Form the matrix $\boldsymbol{U} = \boldsymbol{Q}\,\hat{\boldsymbol{U}}$.

Low Rank Approximations via Random Projections

1. Starting with an $m \times n$ matrix $\boldsymbol{A}$.
2. Draw an $n \times k + p$ Gaussian random matrix $\boldsymbol{\Omega}$.
3. Form $\boldsymbol{Y} = \boldsymbol{A}\,\boldsymbol{\Omega}$ and compute its QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\,\boldsymbol{R}$
4. Form the $k + p \times n$ matrix $\boldsymbol{B} = \boldsymbol{Q}'\,\boldsymbol{A}$.
5. Compute the SVD of the small matrix $\boldsymbol{B}$, $\boldsymbol{B} = \hat{\boldsymbol{U}}\,\boldsymbol{S}\,\boldsymbol{V}'$.
6. Form the matrix $\boldsymbol{U} = \boldsymbol{Q}\,\hat{\boldsymbol{U}}$.

Resulting approximation has nicely bounded expected error,

$$\mathrm{E}\ \|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}'\| \leq \left[1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min(m,n)}\right]\sigma_{k+1}.$$

Halko, Martinsson, Tropp (2011)

Random Matrix Low Rank Decompositions and GPs

Preceding algorithm can be modified slightly to take advantage of the positive definite structure of a covariance matrix.

1. Starting with an $n \times n$ covariance matrix $\boldsymbol{A}$.
2. Draw an $n \times k+p$ Gaussian random matrix $\boldsymbol{\Omega}$.
3. Form $\boldsymbol{Y} = \boldsymbol{A}\,\boldsymbol{\Omega}$ and compute its QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\,\boldsymbol{R}$
4. Form the $k+p \times k+p$ matrix $\boldsymbol{B} = \boldsymbol{Q}'\,\boldsymbol{A}\,\boldsymbol{Q}$.
5. Compute the eigen decomposition of the small matrix $\boldsymbol{B}$, $\boldsymbol{B} = \hat{\boldsymbol{U}}\,\boldsymbol{S}\,\hat{\boldsymbol{U}}'$.
6. Form the matrix $\boldsymbol{U} = \boldsymbol{Q}\,\hat{\boldsymbol{U}}$.

Once again we have a bound on the error,

$$\mathrm{E}\,\|\boldsymbol{A} - \boldsymbol{Q}(\boldsymbol{Q}'\boldsymbol{A}\boldsymbol{Q})\boldsymbol{Q}'\| = \mathrm{E}\,\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{S}\boldsymbol{U}'\| \lessapprox c \cdot \sigma_{k+1}.$$

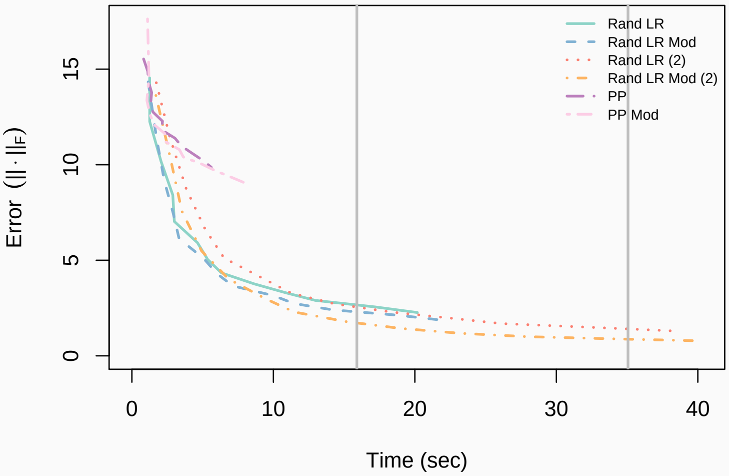
Halko, Martinsson, Tropp (2011), Banerjee, Dunson, Tokdar (2012)

Low Rank Approximations and GPUs

Both predictive process and random matrix low rank approximations are good candidates for acceleration using GPUs.

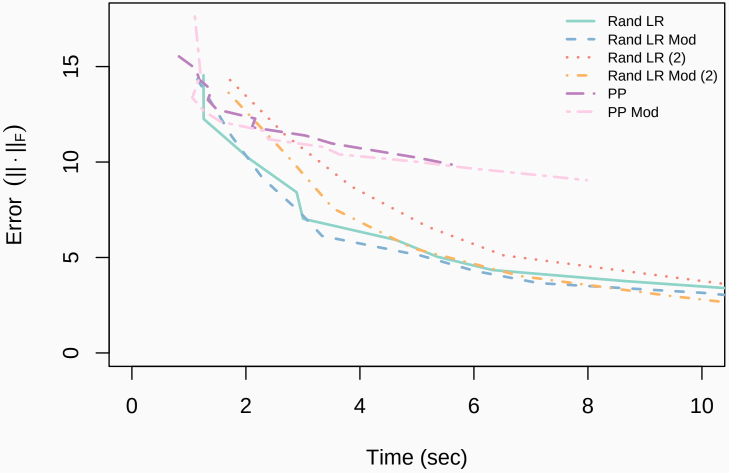
- Both use Sherman-Woodbury-Morrison to calculate the inverse (involves matrix multiplication, addition, and a small Matrix inverse).
- Predictive processes involves several covariance matrix calculations (knots and cross-covariance) and a small matrix inverse.
- Random matrix low rank involves a large matrix multiplication ($\boldsymbol{A\Omega}$) and several small matrix decompositions (QR, eigen).
- Functionality for both approaches included in current version of RcppGP (inv_lr and inv_pp).

Matrix inverse (fixed rank, strong dependence)



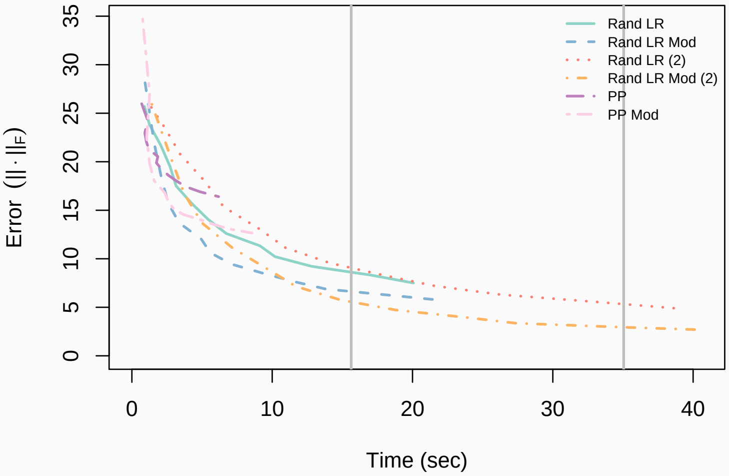
$$n = 15000, \quad k = \{100, \ldots, 4900\}$$

Matrix inverse (fixed rank, strong dependence)



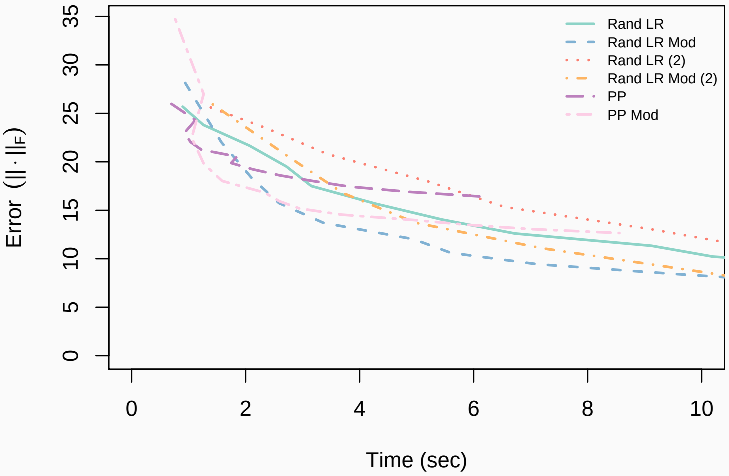
$$n = 15000, \quad k = \{100, \ldots, 4900\}$$

Matrix inverse (fixed rank, weak dependence)



$$n = 15000, \quad k = \{100, \dots, 4900\}$$

Matrix inverse (fixed rank, weak dependence)



$$n = 15000, \quad k = \{100, \ldots, 4900\}$$

Rand. Matrix Low Rank Decompositions for Prediction

This approach can also be used for prediction, if we want to sample

$$\mathcal{N}(0, \mathbf{\Sigma}) \text{ with } \mathbf{\Sigma} \approx \boldsymbol{USU'} = (\boldsymbol{US}^{1/2}\boldsymbol{U'})(\boldsymbol{US}^{1/2}\boldsymbol{U'})'$$

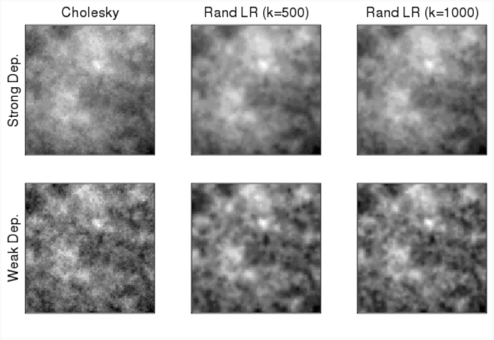
then $X_{pred} = (\boldsymbol{U}\,\boldsymbol{S}^{1/2}\,\boldsymbol{U'}) \times \boldsymbol{Z}$ where $Z_i \sim \mathcal{N}(0, 1)$.

Rand. Matrix Low Rank Decompositions for Prediction

This approach can also be used for prediction, if we want to sample

$$\mathcal{N}(0, \mathbf{\Sigma}) \text{ with } \mathbf{\Sigma} \approx \boldsymbol{USU'} = (\boldsymbol{US}^{1/2}\boldsymbol{U'})(\boldsymbol{US}^{1/2}\boldsymbol{U'})'$$

then $X_{pred} = (\boldsymbol{U}\,\boldsymbol{S}^{1/2}\,\boldsymbol{U'}) \times \boldsymbol{Z}$ where $Z_i \sim \mathcal{N}(0,1)$.



$$n = 1000, \quad p = 10000$$

Future Directions

- Refinement of RcppGP
  - Transition to header only implementation
  - Transparent GPU to CPU failover
  - Support for fixed error (instead of rank) random matrix low rank decomposition
  - Thinking about out-of-memory based approaches
- Future of GPUs, CUDA, and Magma
  - Single vs. Multi-GPU algorithms
  - Mixed precision algorithms
  - NVBLAS
  - Unified memory
  - cuSolver

Acknowledgments

Migratory Connectivity

- John Novembre - UChicago
- Thomas Smith - UCLA
- Kristen Ruegg - UCLA, UCSC
- Center for Tropical Research, UCLA IoES

Speciated $PM_{2.5}$

- Alan Gelfand - Duke
- Dave Holland - EPA
- Erin Schliep - Missouri

RcppGP `https://github.com/rundel/RcppGP`