Lecture 22

Computational Methods for GPs

Colin Rundel

04/12/2017

GPs and Computational Complexity

The problem with GPs

Unless you are lucky (or clever), Gaussian process models are difficult to scale to large problems. For a Gaussian process $y \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

The problem with GPs

Unless you are lucky (or clever), Gaussian process models are difficult to scale to large problems. For a Gaussian process $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

Want to sample $\boldsymbol{y}$?

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

The problem with GPs

Unless you are lucky (or clever), Gaussian process models are difficult to scale to large problems. For a Gaussian process $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

Want to sample $\boldsymbol{y}$?

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

Evaluate the (log) likelihood?

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

The problem with GPs

Unless you are lucky (or clever), Gaussian process models are difficult to scale to large problems. For a Gaussian process $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

Want to sample $\boldsymbol{y}$?

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

Evaluate the (log) likelihood?

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

Update covariance parameter?

$$\boxed{\{\Sigma\}_{ij}} = \sigma^2 \exp(-\{d\}_{ij}\phi) + \sigma_n^2 1_{i=j} \qquad \mathcal{O}\left(n^2\right)$$

A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity

A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity - Go for it

A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity - Go for it

$\mathcal{O}(n^2)$ - Quadratic complexity

A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity - Go for it

$\mathcal{O}(n^2)$ - Quadratic complexity - Pray

A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity - Go for it

$\mathcal{O}(n^2)$ - Quadratic complexity - Pray

$\mathcal{O}(n^3)$ - Cubic complexity

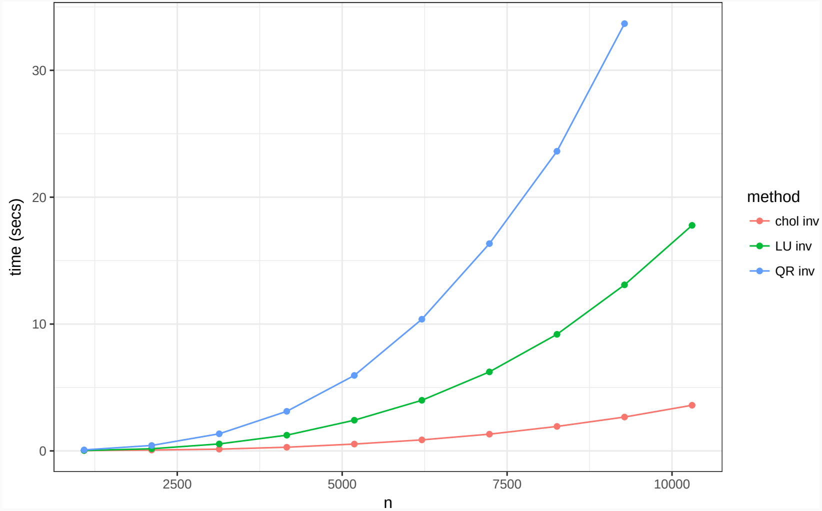
A simple guide to computational complexity

$\mathcal{O}(n)$ - Linear complexity - Go for it

$\mathcal{O}(n^2)$ - Quadratic complexity - Pray

$\mathcal{O}(n^3)$ - Cubic complexity - Give up

How bad is the problem?



Practice - Migratory Model Prediction

After fitting the GP need to sample from the posterior predictive distribution at $\sim$ 3000 locations

$$\boldsymbol{y}_p \sim \mathcal{N}\left(\mu_p + \Sigma_{po}\Sigma_o^{-1}(y_o - \mu_o),\ \Sigma_p - \Sigma_{po}\Sigma_o^{-1}\Sigma_{op}\right)$$

Practice - Migratory Model Prediction

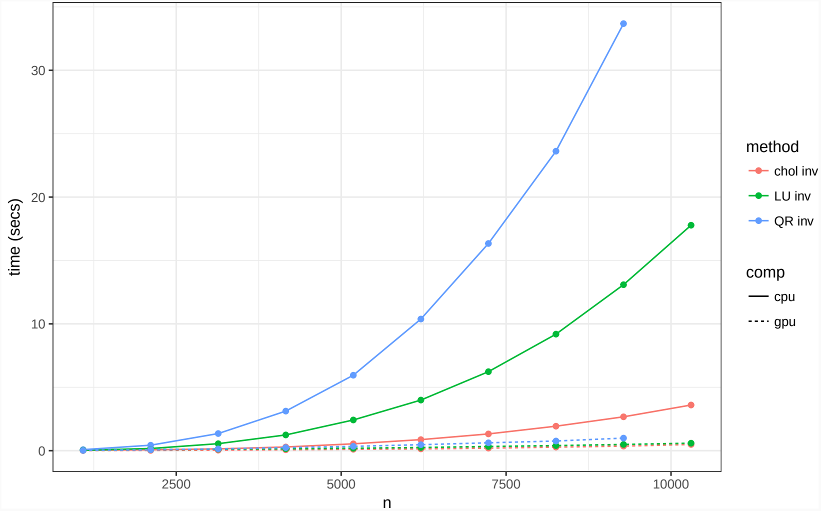
After fitting the GP need to sample from the posterior predictive distribution at $\sim$ 3000 locations

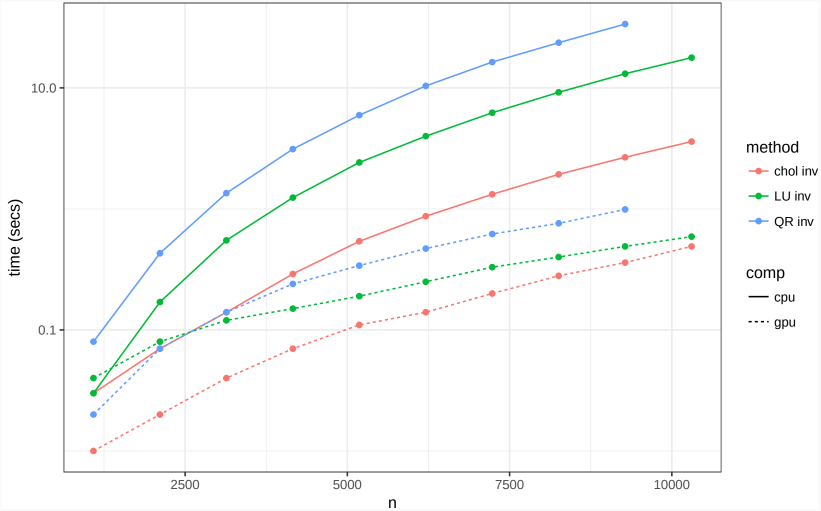
$$y_p \sim \mathcal{N}\left(\mu_p + \Sigma_{po}\Sigma_o^{-1}(y_o - \mu_o),\ \Sigma_p - \Sigma_{po}\Sigma_o^{-1}\Sigma_{op}\right)$$

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Performance
1.	Calc. Σ_p, Σ_{po}	1.080	0.046	23.0
2.	Calc. $\text{chol}(\Sigma_p - \Sigma_{po}\Sigma_o^{-1}\Sigma_{op})$	0.467	0.208	2.3
3.	Calc. $\mu_{p\|o} + \text{chol}(\Sigma_{p\|o}) \times Z$	0.049	0.052	0.9
4.	Calc. Allele Prob	0.129	0.127	1.0
	Total	1.732	0.465	3.7

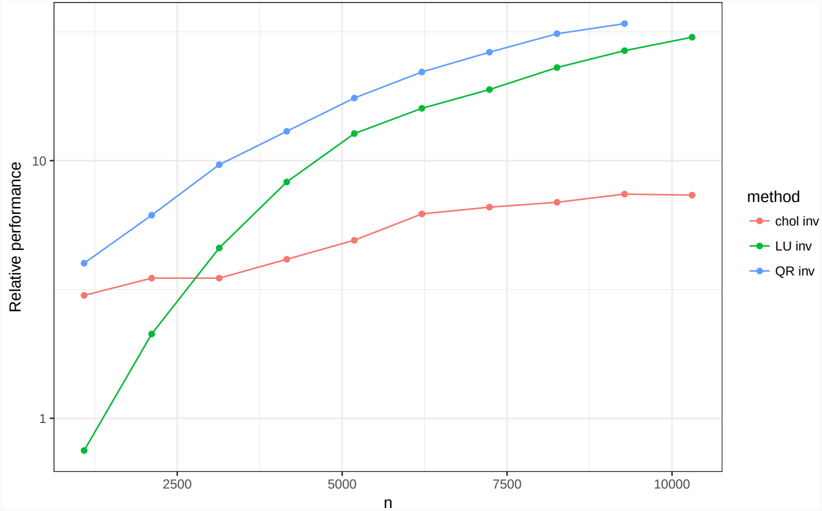
Total run time: CPU (28.9 min), CPU+GPU (7.8 min)

Cholesky CPU vs GPU (P100)

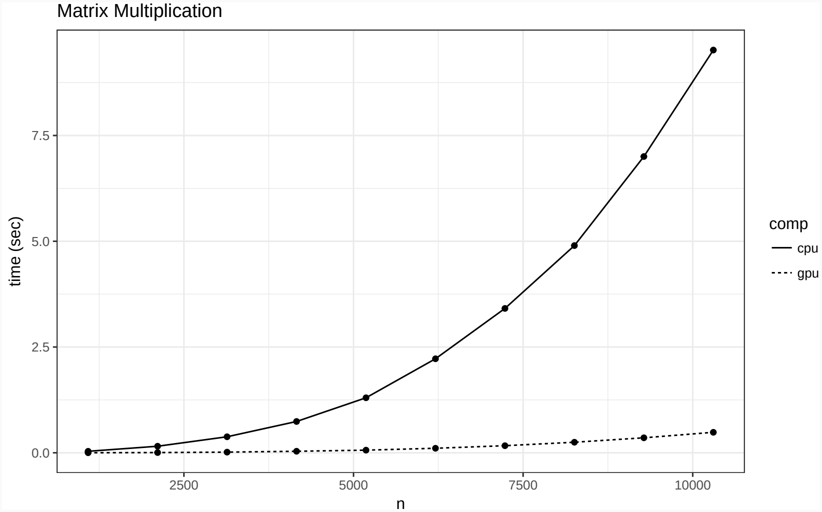



time (secs)
10.0
0.1
2500
5000
7500
10000
n
method
chol inv
LU inv
QR inv
comp
cpu
gpu

Relative Performance



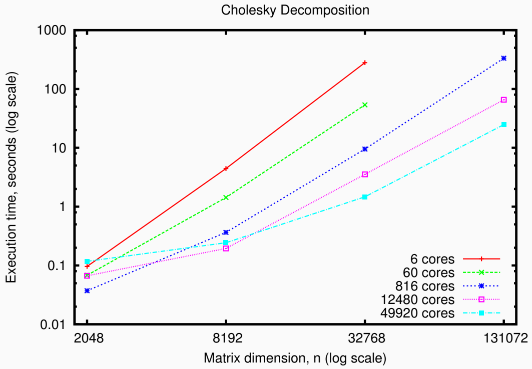
Aside - Matrix Multiplication



An even bigger hammer

`bigGP` is an R package written by Chris Paciorek (UC Berkeley), et al.

- Specialized distributed implementation of linear algebra operation for GPs
- Designed to run on large super computer clusters
- Uses both shared and distributed memory
- Able to fit models on the order of $n = 65$k (32 GB Cov. matrix)



More scalable solutions?

- Spectral domain / basis functions
- Covariance tapering
- GMRF approximations
- Low-rank approximations
- Nearest-neighbor models

Low Rank Approximations

Low rank approximations in general

Lets look at the example of the singular value decomposition of a matrix,

$$\underset{n\times m}{M} = \underset{n\times n}{U} \underset{n\times m}{\mathrm{diag}(S)} \underset{m\times m}{V^t}$$

where U are called the left singular vectors, V the right singular vectors, and S the singular values. Usually the singular values and vectors are ordered such that the signular values are in descending order.

Low rank approximations in general

Lets look at the example of the singular value decomposition of a matrix,

$$\underset{n\times m}{M} = \underset{n\times n}{U}\,\underset{n\times m}{\mathrm{diag}(S)}\,\underset{m\times m}{V^t}$$

where U are called the left singular vectors, V the right singular vectors, and S the singular values. Usually the singular values and vectors are ordered such that the signular values are in descending order.

It turns out (Eckart–Young theorem) that we can approximate M as having rank r by defining $\tilde{S}$ to only have the r largest singular values (others set to zero).

$$\underset{n\times m}{\tilde{M}} = \underset{n\times n}{U}\,\underset{n\times m}{\mathrm{diag}(\tilde{S})}\,\underset{m\times m}{V^t} \quad = \underset{n\times k}{\tilde{U}}\,\underset{k\times k}{\mathrm{diag}(\tilde{S})}\,\underset{k\times m}{\tilde{V}^t}$$

Example

$$M = \begin{pmatrix} 1.000 & 0.500 & 0.333 & 0.250 \\ 0.500 & 0.333 & 0.250 & 0.200 \\ 0.333 & 0.250 & 0.200 & 0.167 \\ 0.250 & 0.200 & 0.167 & 0.143 \end{pmatrix} = U \operatorname{diag}(S)\, V^t$$

$$U = V = \begin{pmatrix} -0.79 & 0.58 & -0.18 & -0.03 \\ -0.45 & -0.37 & 0.74 & 0.33 \\ -0.32 & -0.51 & -0.10 & -0.79 \\ -0.25 & -0.51 & -0.64 & 0.51 \end{pmatrix}$$

$$S = \begin{pmatrix} 1.50 & 0.17 & 0.01 & 0.00 \end{pmatrix}$$

Example

$$M = \begin{pmatrix} 1.000 & 0.500 & 0.333 & 0.250 \\ 0.500 & 0.333 & 0.250 & 0.200 \\ 0.333 & 0.250 & 0.200 & 0.167 \\ 0.250 & 0.200 & 0.167 & 0.143 \end{pmatrix} = U\,\mathrm{diag}(S)\,V^t$$

$$U = V = \begin{pmatrix} -0.79 & 0.58 & -0.18 & -0.03 \\ -0.45 & -0.37 & 0.74 & 0.33 \\ -0.32 & -0.51 & -0.10 & -0.79 \\ -0.25 & -0.51 & -0.64 & 0.51 \end{pmatrix}$$

$$S = \begin{pmatrix} 1.50 & 0.17 & 0.01 & 0.00 \end{pmatrix}$$

Rank 2 approximation:

$$\tilde{M} = \begin{pmatrix} -0.79 & 0.58 \\ -0.45 & -0.37 \\ -0.32 & -0.51 \\ -0.25 & -0.51 \end{pmatrix} \begin{pmatrix} 1.50 & 0.00 \\ 0.00 & 0.17 \end{pmatrix} \begin{pmatrix} -0.79 & -0.45 & -0.32 & -0.25 \\ 0.58 & -0.37 & -0.51 & -0.51 \end{pmatrix}$$

$$= \begin{pmatrix} 1.000 & 0.501 & 0.333 & 0.249 \\ 0.501 & 0.330 & 0.251 & 0.203 \\ 0.333 & 0.251 & 0.200 & 0.166 \\ 0.249 & 0.203 & 0.166 & 0.140 \end{pmatrix}$$

Approximation Error

We can measure the error of the approximation using the Frobenius norm,

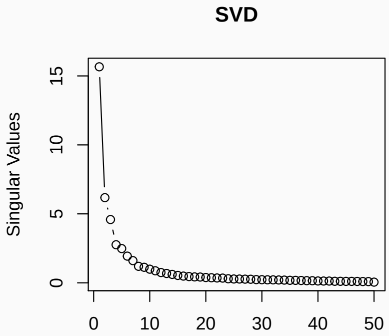
$$\|M - \tilde{M}\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} (M_{ij} - \tilde{M}_{ij})^2 \right)^{1/2}$$

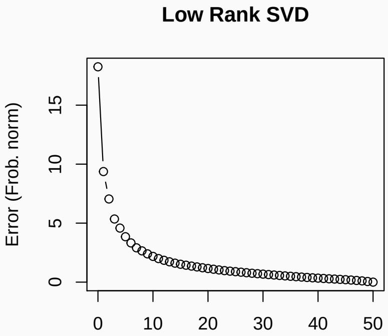
Approximation Error

We can measure the error of the approximation using the Frobenius norm,

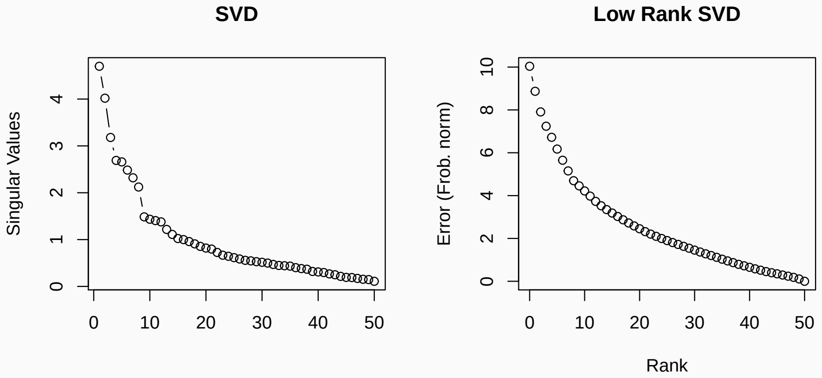
$$\|M - \tilde{M}\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} (M_{ij} - \tilde{M}_{ij})^2 \right)^{1/2}$$

Strong dependence (large eff. range):





Weak dependence (short eff. range):



How does this help? (Sherman-Morrison-Woodbury)

There is an immensely useful linear algebra identity, the Sherman-Morrison-*Woodbury* formula, for the inverse (and determinant) of a decomposed matrix,

$$\begin{aligned}\underset{n\times m}{\tilde{M}}^{-1} &= \left(\underset{n\times m}{A} + \underset{n\times k}{U}\,\underset{k\times k}{S}\,\underset{k\times m}{V^t}\right)^{-1} \\ &= A^{-1} - A^{-1}U\left(S^{-1} + V^tA^{-1}U\right)^{-1}V^tA^{-1}.\end{aligned}$$

How does this help? (Sherman-Morrison-Woodbury)

There is an immensely useful linear algebra identity, the Sherman-Morrison-*Woodbury* formula, for the inverse (and determinant) of a decomposed matrix,

$$\begin{aligned}\underset{n\times m}{\tilde{M}}^{-1} &= \left(\underset{n\times m}{A} + \underset{n\times k}{U}\;\underset{k\times k}{S}\;\underset{k\times m}{V^t}\right)^{-1} \\ &= A^{-1} - A^{-1}U\left(S^{-1} + V^tA^{-1}U\right)^{-1}V^tA^{-1}.\end{aligned}$$

How does this help?

- Imagine that $A = \operatorname{diag}(A)$, then it is trivial to find A^{-1}.
- S^{-1} is $k \times k$ which is hopefully small, or even better $S = \operatorname{diag}(S)$.
- $\left(S^{-1} + V^tA^{-1}U\right)$ is $k \times k$ which is hopefully small.

Aside - Determinant

Remember for any MVN distribution when evaluating the likelihood

$$-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(x - \boldsymbol{\mu}) - \frac{n}{2}\log 2\pi$$

we need the inverse of Σ as well as its *determinant*.

Aside - Determinant

Remember for any MVN distribution when evaluating the likelihood

$$-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu}) - \frac{n}{2}\log 2\pi$$

we need the inverse of Σ as well as its *determinant*.

- For a full rank Cholesky decomposition we get the determinant for "free".

$$|M| = |LL^t| = \prod_{i=1}^{n} \left(\text{diag}(L)_i\right)^2$$

Aside - Determinant

Remember for any MVN distribution when evaluating the likelihood

$$-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu}) - \frac{n}{2}\log 2\pi$$

we need the inverse of Σ as well as its *determinant*.

- For a full rank Cholesky decomposition we get the determinant for “free”.

$$|M| = |LL^t| = \prod_{i=1}^{n}\left(\text{diag}(L)_i\right)^2$$

- For a low rank approximation the Sherman-Morrison-Woodbury / Determinant lemma gives us,

$$\begin{aligned}\det(\tilde{M}) &= \det(A + USV^t) \\ &= \det(S^{-1} + V^t A^{-1} U)\ \det(S)\ \det(A)\end{aligned}$$

Low rank approximations for GPs

For a standard spatial random effects model,

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

if we can replace $\boldsymbol{\Sigma}(\boldsymbol{s})$ with a low rank approximation of the form

- $\boldsymbol{\Sigma}(\boldsymbol{s}) \approx \boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}^t$ where
- $\boldsymbol{U}$ and $\boldsymbol{V}$ are $n \times k$,
- $\boldsymbol{S}$ is $k \times k$, and
- $A = \tau^2 I$ or a similar diagonal matrix

Predictive Processes

Gaussian Predictive Processes

For a rank k approximation,

- Pick k knot locations $s^\star$
- Calculate knot covariance, $\boldsymbol{\Sigma}(s^\star)$, and knot cross-covariance, $\boldsymbol{\Sigma}(s, s^\star)$
- Approximate full covariance using

$$\boldsymbol{\Sigma}(s) \approx \underset{n\times k}{\boldsymbol{\Sigma}(s, s^\star)}\, \underset{k\times k}{\boldsymbol{\Sigma}(s^\star)^{-1}}\, \underset{k\times n}{\boldsymbol{\Sigma}(s^\star, s)}.$$

- PPs systematically underestimates variance (σ^2) and inflate τ^2, Modified predictive processs corrects this using

$$\begin{aligned}\boldsymbol{\Sigma}(s) \approx & \boldsymbol{\Sigma}(s, s^\star)\, \boldsymbol{\Sigma}(s^\star)^{-1}\, \boldsymbol{\Sigma}(s^\star, s) \\ & + \operatorname{diag}\Big(\boldsymbol{\Sigma}(s) - \boldsymbol{\Sigma}(s, s^\star)\, \boldsymbol{\Sigma}(s^\star)^{-1}\, \boldsymbol{\Sigma}(s^\star, s)\Big).\end{aligned}$$

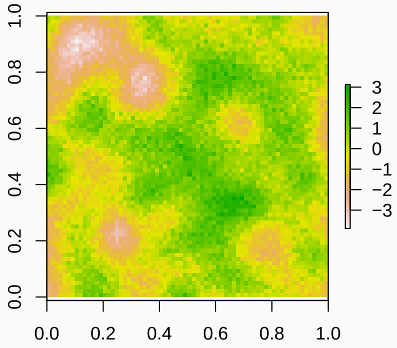
Banerjee, Gelfand, Finley, Sang (2008) Finley, Sang, Banerjee, Gelfand (2008)

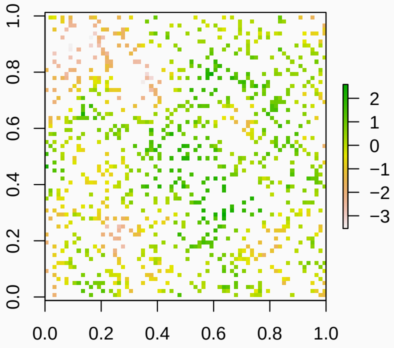
Example

Below we have a surface generate from a squared exponential Gaussian Process where

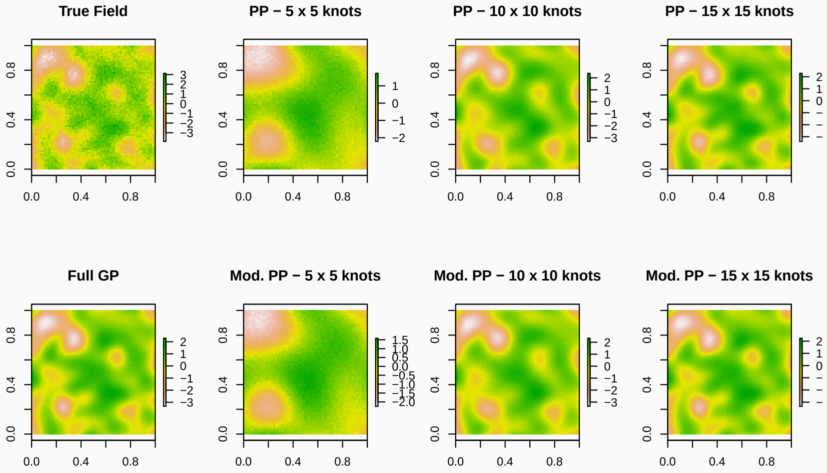
$$\{\Sigma\}_{ij} = \sigma^2 \exp\left(-(\phi\, d)^2\right) + \tau^2 I$$

$$\sigma^2 = 1 \quad \phi = 9 \quad \tau^2 = 0.1$$

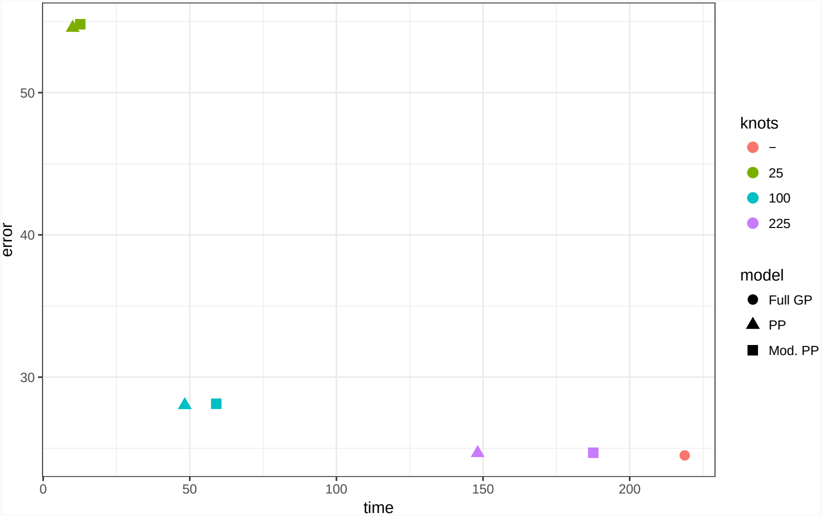




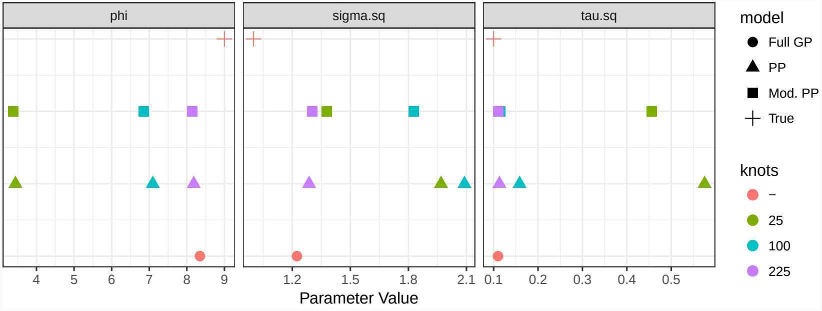
Predictive Process Model Results



Performance



Parameter Estimates



Random Projections

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 a 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 Approximations and GPs

Preceeding 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,

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

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

Low Rank Approximations and GPUs

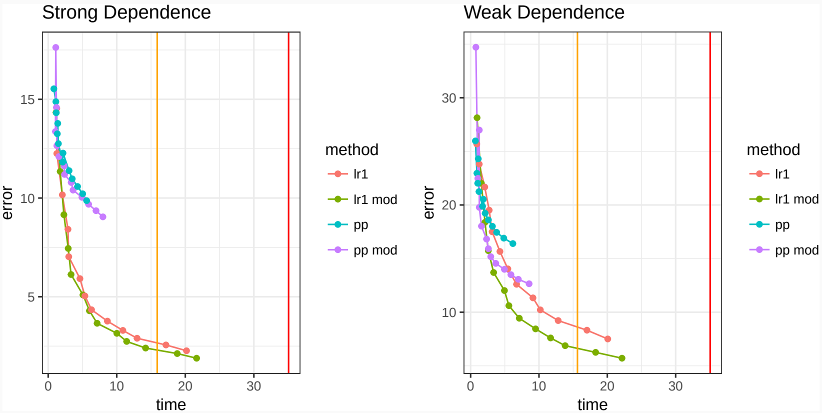
Both predictive process and random matrix low rank approximations are good candidates for acceleration using GPUs.

- 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 approximations involves a large matrix multiplication ($\boldsymbol{A}\,\boldsymbol{\Omega}$) and several small matrix decompositions (QR, eigen).

Comparison ($n = 15,000$, $k = \{100, \ldots, 4900\}$)



Rand. Projection LR Decompositions for Prediction

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

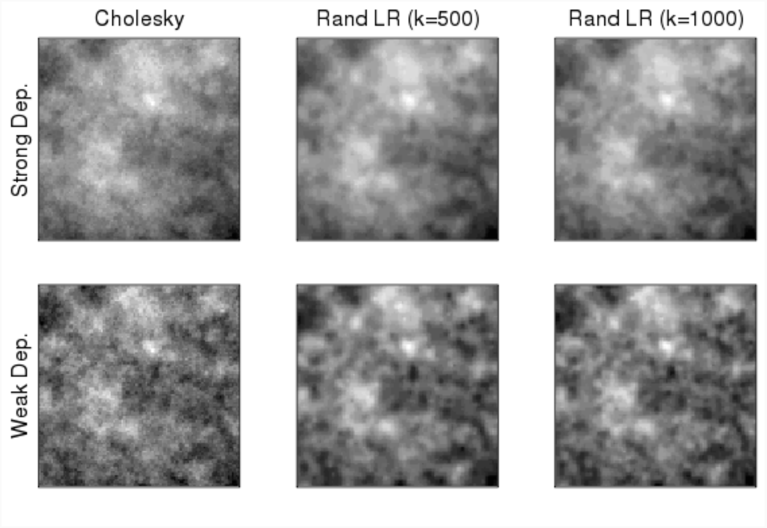
$$y \sim \mathcal{N}(0, \boldsymbol{\Sigma})$$
$$\Sigma \approx \boldsymbol{U}\boldsymbol{S}\boldsymbol{U}^t = (\boldsymbol{U}\boldsymbol{S}^{1/2}\boldsymbol{U}^t)(\boldsymbol{U}\boldsymbol{S}^{1/2}\boldsymbol{U}^t)^t$$

then

$$y_{\text{pred}} = \left(\boldsymbol{U}\,\boldsymbol{S}^{1/2}\,\boldsymbol{U}^t\right) \times \boldsymbol{Z} \text{ where } Z_i \sim \mathcal{N}(0, 1)$$

because $\boldsymbol{U}^t\,\boldsymbol{U} = I$ since $\boldsymbol{U}$ is an orthogonal matrix.

Dehdari, Deutsch (2012)



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