CAMCOS Project – Fall 2006
Improved Linear Algebra Methods for
Redshift Computation from Limited
Spectrum Data
Final Report

**Students:** Miranda Braselton, Kelley Cartwright, Michael Hurley, Maheen Khan, Miguel Rodriguez, David Shao, Jason Smith, Jimmy Ying, Genti Zaimi

**Faculty:** Bem Cayco and Wasin So
## Contents

1 Introduction 7  
2 Redshift and photometric observations 8  
3 Gaussian processes 8  
4 Special polynomial kernel of degree 2 10  
5 Reduced rank 12  
5.1 Iterative Refinement 12  
6 Cholesky update 14  
6.1 Forward and back substitutions 14  
6.2 Rank-one update 17  
7 Conjugate Gradient 19  
8 Training and Testing Procedure 21  
8.1 Data set 21  
8.2 Baseline—Quadratic Regression 22  
8.3 Bootstrap 23  
8.4 Hardware and software environment 24  
9 Results 25  
9.1 Time for 100 Bootstrap Samples 25  
9.2 RMSE Graphs 26  
10 Conclusions 30  
11 Further research 31  
11.1 Definition of hyperparameters 31  
11.1.1 One hyperparameter $\lambda^2$ 31  
11.1.2 Three hyperparameters $(\lambda^2, c_0, c_1, c_2)$ 32  
11.1.3 Twenty-two hyperparameters 32  
11.2 Marginal likelihood 32  
11.3 Preliminary results on optimal hyperparameters 33  
11.4 Final note 36
A Redshift 37

B Sherman-Morrison-Woodbury Formula 39
  B.1 Application to reduced rank reduction 39
  B.2 Proof 39
  B.3 Motivation 40

C Block Matrix Inversion 40
  C.1 Proof of block matrix inversion 41
  C.2 Motivation for block matrix inversion 41
  C.3 Application to covariance matrices 42

D Conditional distribution for multivariate normal 42

E Matrix multiplication is time, space $O(d^r n)$ 45

F Fast computation of $k(x^*)$ 47

G Cholesky rank-one update 49

H Gibbs Sampler 51

I Source code 55
  I.1 Data structures 55
  I.2 Source code listing 55
List of Figures

1  Comparison of Methods for $n = 1000$ ........................ 26
2  Comparison of Methods for $n = 10000$ ......................... 27
3  Comparison of Methods for $n = 180045$ ....................... 28
4  Comparison of reduced rank for varying $n$ .................... 29
5  RMSE for varying $\lambda^2$ using Cholesky update and $n = 180045$ . 35
6  RMSE for varying $\lambda^2$ using Cholesky update and $n = 180045$ . 36
7  $\text{genq.m}$ .................................................. 56
8  $\text{genqprime.m}$ ............................................ 56
9  $\text{ReducedRank.m}$ ........................................ 56
10 $\text{forwardsolve.m}$ .......................................... 57
11 $\text{backwardsolve.m}$ ........................................ 57
12 $\text{getp.m}$ .................................................. 58
13 $\text{getbeta.m}$ ................................................. 58
14 $\text{CU.m}$ ...................................................... 59
15 $\text{CG.m}$ ...................................................... 60
List of Tables

1  Error before and after iterative refinement ............... 13
2  Time (seconds) for 100 Bootstrap RMS Calculations .... 25
3  Number of iterations for conjugate gradient around 200 . 25
4  Estimation of $\lambda^2$ alone .................................. 34
5  Estimation of four hyperparameters not consistent ....... 34
List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Iterative refinement</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td><strong>Forward substitution</strong> for $L = L(p, \beta)$</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td><strong>Back substitution</strong> for $L = L(p, \beta)$</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Solving for $Lp = q$</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Cholesky update solver</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Conjugate gradient method</td>
<td>20</td>
</tr>
</tbody>
</table>
1 Introduction

Way and Srivastava compared various models for predicting galactic redshifts from easier to obtain photometric filter observations. For one particular type of model called Gaussian process regression they were unable to use the full size $N$ of the set of observations due to the necessity of solving a linear system whose apparent solution time was $O(N^3)$ [Way and Srivastava (2006)].

Having noticed the special form of a kernel function used by Way and Srivastava for their Gaussian process regression, we implement three well known numerical methods that reduce the running time of the problematic linear system solution to $O(N)$ instead of $O(N^3)$. We extend Way and Srivastava’s analysis of the prediction error of Gaussian process regression for their set of redshift and photometric filter observations to the full size of the observation set. Our analysis indicates that for the full size of the observation set, Gaussian process regression models do not have prediction error smaller than that of the simplest baseline method, quadratic regression. Our preliminary analysis also indicates that optimizing a parameter corresponding to an estimate of the magnitude of random noise in redshift observation does not substantially decrease prediction error.

For this report we leave background discussion of the physics behind redshift and photometric observation to Appendix A. We begin with a discussion of how the special form of the kernel function used in Gaussian process regression enables us to reduce the effective dimensionality of the linear system that needs to be solved. We then present our three implemented fast methods for solving such systems—reduced rank, Cholesky update, and conjugate gradient. We include in Appendix H discussion of a partially implemented method called Gibbs sampling that would perhaps be applicable if we could not use the special form of the kernel function. We report on our fast methods’ running times and prediction errors, using Way and Srivastava’s bootstrap methodology of repeatedly sampling from the same observation set, refitting models, and comparing predicted to observed redshifts in order to estimate the range of prediction error.

We start with a condensed explanation of redshift and photometric observations and of the notation used throughout this report.
2 Redshift and photometric observations

Let \( y = (y_1, \ldots, y_n)^T \) be a sequence of redshift observations of \( n \) galaxies. In Way and Srivastava’s observation set, for each galaxy \( i \) there are five photometric observations denoted as \( ugriz \). We use \( X \) to denote an \( n \times 5 \) matrix such that the \( i \)-th row \( x_i \) of \( X \) represents the five photometric observations corresponding to galaxy \( i \), and we let \( U, G, R, I, Z \) denote the columns of \( X \) in their respective order.

The goal is to fit a model from given observations \( X \) and \( y \) such that using this model, we can find a prediction \( \hat{y}^* \) of the redshift \( y^* \) of some galaxy after inserting the galaxy’s photometric observations \( x^* \) into the fitted model.

3 Gaussian processes

In Gaussian process regression we assume that the \( n \) given redshifts \( y \) and the redshift to be predicted \( y^* \) are together generated by a multivariate normal distribution—a generalization to \( n + 1 \) dimensions of the bell-shaped normal (Gaussian) distribution that also allows for modeling interactions between redshift values, so that one redshift may be correlated in sign and/or magnitude with another. Here we append redshift \( y^* \) to \( n \)-vector \( y \) to form an \( (n + 1) \)-vector. The \( (n + 1) \times (n + 1) \) matrix that aggregates these estimates of the interactions is known as the multivariate normal distribution’s covariance matrix \( A^{(n+1)} \).

As we wish to predict redshifts from photometric observations, covariance matrix \( A^{(n+1)} \) is a function \( A^{(n+1)} = A^{(n+1)}(X, x^*) \) of the photometric observations \( X \) and \( x^* \).

The choice of function \( A^{(n+1)}(X, x^*) \) is somewhat arbitrary, dependent both on mathematical convenience and empirical considerations. It is reasonable to express the covariance matrix as the sum of a random noise term and a term generated from a kernel function. Let \( I = I_{n,n} \) be the \( n \times n \) identity matrix, then the random noise term is modeled by \( \lambda^2 I^{(n+1)} \) where \( I^{(n+1)} \) is the \( (n + 1) \times (n + 1) \) diagonal matrix with \( I \) as its upper left hand corner and the rest of its entries 0, and where \( \lambda^2 \) represents the expected value of the square of the magnitude of the random noise. The term generated from kernel function \( K \) is a matrix \( K^{(n+1)} \) where the \((a, b)\) entry \( K^{(n+1)}_{a,b} \) represents the covariance, a measure of interaction, between the \( a \)-th and
b-th redshifts. Hence

\[ A^{(n+1)} = \lambda^2 I^{(n+1)} + K^{(n+1)}. \]

Since we wish to predict a galaxy’s redshift \( y^* \) from its photometric filter observations \( x^* \), the kernel function \( K \) is actually a function on two photometric filter observations. The expected covariance contributed by the kernel function between redshifts \( y_a \) and \( y_b \) is \( K(x_a, x_b) \), where \( y^* \) may be neither, either, or both of \( y_a \) and \( y_b \). In this model the random noise term only affects the spread or range of a previous redshift observation \( y_i \) from \( y \) and does not affect the covariance between two distinct observations.

However we are not interested in simultaneously analyzing what occurs in all \( n + 1 \) dimensions of the \( n + 1 \) random vector of redshifts \( (y^T, y^*) \); instead, we are given in advance previous redshift observations \( y \), and previous photometric filter observations \( X \). Given all of these values and given a photometric filter observation \( x^* \), we wish to know the distribution of the one dimensional random variable \( y^* \)—in particular, the expected value of \( y^* \) that we will use to predict its value.

Fortunately it is well known that the properties of multivariate normal distributions imply that the distribution of \( y^* \) given the values \( X, y, x^* \) is an ordinary normal distribution whose mean and variance can be expressed as a function of \( X, y, x^* \), a proof of which we include in Appendix D. Let \( K^{(n)} \) be the \( n \times n \) upper left hand corner of \( K^{(n+1)} \) that corresponds to the covariances between the previously observed \( y \), and let \( k(x^*) \) be the \( n \times 1 \) upper right hand corner of \( K^{(n+1)} \) that corresponds to the covariances between \( y^* \) and \( y \). Then the expected value \( E[y^*|X, y, x^*] \) of \( y^* \), which we use as the prediction \( \hat{y}^* \), is given by

\[ \hat{y}^* = E[y^*|X, y, x^*] = k(x^*)^T (\lambda^2 I + K^{(n)})^{-1} y \] \( (1) \)

For the right hand side of equation (1), the noise term \( \lambda^2 I \) depends only on \( \lambda^2 \) and the \( K = K(X) \) term depends only on the previously observed photometric filter observations. The influence of the previously observed redshifts is therefore strictly in the \( y \) term, and the influence of the photometric filter observation \( x^* \) is strictly in the \( k(x^*) \) term.

We next discuss the special form of a certain kernel function that enables us to rapidly fit its corresponding Gaussian process regression model.
4 Special polynomial kernel of degree 2

The standard **polynomial kernel of degree** $r$ is the function

$$K(x_a, x_b) = (1 + \langle x_a, x_b \rangle)^r$$  \hspace{1cm} (2)

where $\langle x_a, x_b \rangle$ is the ordinary dot product of $x_a$ and $x_b$.

For $r = 2$ the polynomial kernel has the form

$$K(x_a, x_b) = (1 + \langle x_a, x_b \rangle)^2$$  \hspace{1cm} (3)

$$= 1 + 2 \langle x_a, x_b \rangle + \langle x_a, x_b \rangle^2$$  \hspace{1cm} (4)

For representing values that are constant across all galaxy observations we add an $n \times 1$ column $\mathbf{1}$ of all 1’s as its entries. Recall for $X$ an $n \times 5$ matrix of $ugriz$ photometric filter observations, the 5 columns of $X$ are denoted respectively by $\mathbf{U}$, $\mathbf{G}$, $\mathbf{R}$, $\mathbf{I}$, $\mathbf{Z}$. For columns $A$ and $B$ let $AB$ denote the componentwise multiplication of $A$ by $B$ such that entry $(AB)_i$ is the product $A_iB_i$. Interactions between 5 photometric filter observations are therefore represented by the 15 columns $UU$, $UG$, $UR$, $UI$, $UZ$, $GG$, $GR$, $GI$, $GZ$, $RR$, $RI$, $RZ$, $II$, $IZ$, and $ZZ$. Note that we do not duplicate columns such as $GU$ since $UG = GU$—this reduces the number of interaction columns from $25 = 5^2$ to $15 = 5 + 4 + 3 + 2 + 1$.

From Corollary E.2 of Appendix E we obtain covariance matrix $A$ for $d = 5$ and $(c_0, c_1, c_2) = (1, 2, 1)$ to be a sum of the form

$$A = \lambda^2 I_{n,n} +$$

$$1 \{1\}^T +$$

$$2 \left[ U \{U^T\} + G \{G^T\} + R \{R^T\} + I \{I^T\} + Z \{Z^T\} \right] +$$

$$UU \{UU\}^T +$$

$$2 \left[ UG \{UG\}^T + UR \{UR\}^T + UI \{UI\}^T + UZ \{UZ\}^T \right] +$$

$$GG \{GG\}^T +$$

$$2 \left[ GR \{GR\}^T + GI \{GI\}^T + GZ \{GZ\}^T \right] +$$

$$RR \{RR\}^T + 2 \left[ RI \{RI\}^T + RZ \{RZ\}^T \right] +$$

$$II \{II\}^T + 2IZ \{IZ\}^T + ZZ \{ZZ\}^T$$  \hspace{1cm} (7)
Note that there is a factor of 2 in front of each mixed interaction term such as $UG$ because $UG = GU$, and therefore each such mixed term actually accounts for 2 columns.

Define matrix $Q$ to be the matrix whose 21 columns are in respective order $1, \sqrt{2}U, \sqrt{2}G, \sqrt{2}R, \sqrt{2}I, UU, \sqrt{2}UG, \sqrt{2}UR, \sqrt{2}UI, \sqrt{2}UZ, GG, \sqrt{2}GR, \sqrt{2}GI, \sqrt{2}GZ, RR, \sqrt{2}RI, \sqrt{2}RZ, II, \sqrt{2}IZ$, and $ZZ$. Then we can rewrite equation (7) in the far simpler form

$$A = \lambda^2 I + QQ^T$$

(8)

A similar analysis holds for polynomial kernels of any degree $r \geq 1$ so that the resulting matrix $Q$ has dimension $n \times m$ where $m = m(r, d) = O(d^r)$ is a function of $r$ and the number of columns $d$, with $d = 5$ for $ugriz$ observations.

We next discuss our three implemented fast methods for fitting Gaussian process regression models, starting with the simplest to implement, reduced rank.
5 Reduced rank

Using the Sherman-Morrison-Woodbury formula as derived in equation (17) of Appendix B, we obtain that the inverse of the covariance matrix $A$ of dimension $n \times n$ is given by

\[
(\lambda^2 I + QQ^T)^{-1} = \frac{1}{\lambda^2} \left[ I - Q \left( \lambda^2 I + Q^T Q \right)^{-1} Q^T \right] \tag{9}
\]

Let the dimension of $Q$ be $n \times m$. Note that the matrix inverse in the left hand side of equation (9) is for a matrix of dimension $n \times n$, but the matrix inverse in the right hand side is for a matrix of dimension of $m \times m$. Then for a polynomial kernel of degree $r$, the reduced rank method simply directly solves equation (9) using the simpler solution $z$ of $m \times m$ linear system $(\lambda^2 I + Q^T Q) z = Q^T y$. Thus solving the linear system $A w = y$ for $w$ is performed by calculating the solution of

\[
w = \frac{1}{\lambda^2} \left[ y - Q \left( \left( \lambda^2 I + Q^T Q \right)^{-1} Q^T \right)y \right].
\]

We note that for this observation set of ugriz photometric observations the columns of $Q$ are highly correlated so that matrix $Q^T Q$ is almost singular. The numerical stability of this reduced rank method is questionable.

5.1 Iterative Refinement

**Algorithm 1**: Iterative refinement

<table>
<thead>
<tr>
<th>input</th>
<th>$A$ and $y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Estimated solution $w$ of $A w = y$</td>
</tr>
</tbody>
</table>

begin
  Estimate $\hat{w}$ that solves $A \hat{w} = y$;
  Calculate residual $r = y - A \hat{w}$;
  Estimate $\hat{z}$ that solves $A \hat{z} = r$;
  Return $w = \hat{w} + \hat{z}$;

end

We use the method known as iterative refinement to improve the accuracy of a computed solution to a linear system. As shown in Algorithm 1, suppose we have a method for estimating a solution $\hat{w}$ to a linear system
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Size $n$ for $n \times n \ A$ & Error without iterative refinement & Error after one round of iterative refinement \\
\hline
50 & $1.7 \times 10^{-6}$ & $9.1 \times 10^{-9}$ \\
100 & $1.2 \times 10^{-5}$ & $2.2 \times 10^{-8}$ \\
500 & $1.1 \times 10^{-4}$ & $2.8 \times 10^{-7}$ \\
1000 & $3.5 \times 10^{-4}$ & $8.4 \times 10^{-7}$ \\
5000 & $4.2 \times 10^{-3}$ & $1.3 \times 10^{-5}$ \\
\hline
\end{tabular}
\caption{Error before and after iterative refinement}
\end{table}

$Aw = y$, then one can apply this same method replacing $y$ by the residual $r = y - \hat{A}\hat{w}$.

Because of the reduced rank method’s speed and low memory requirements and because of the special danger of roundoff error with this method, we apply iterative refinement once. In Table 1 we show the results of numerical experiments demonstrating error reduction of approximately two orders of magnitude (two to three significant digits) when applying one round of iterative refinement to our reduced rank method for solving a special linear system using $A$ derived from the special polynomial kernel of degree $r = 2$.

An additional iteration however did not produce a significant improvement.

We next discuss our implementation of the Cholesky update method, a method that is intermediate for our three fast methods in terms of run time, but one that also provides an $O(n)$ method for calculating the determinant of $A$ for special polynomial kernels.
6 Cholesky update

An $n \times n$ matrix $A$ is **symmetric positive definite** if it is equal to its transpose (symmetric) and if for any nonzero $n$-vector $v \neq 0$, the quadratic form evaluated at $v$ always satisfies $v^T A v > 0$. The choice of $A$ is deliberate as the noise term $\lambda^2 I$ ensures that covariance matrix $A = \lambda^2 I + QQ^T$ for Gaussian process regression and for the special polynomial kernel of degree $r$ is always symmetric positive definite.

For symmetric positive definite $A$, it is well known that there exists a unique decomposition of $A$, called its **Cholesky factorization (decomposition)** into the product of a lower triangular matrix $L$ with diagonal entries all 1, a diagonal matrix $\Lambda$ with strictly positive diagonal entries, and upper triangular $L^T$. See for example [Stoer and Bulirsch, Theorem 4.3.3, page 182].

$$A = L\Lambda L^T$$

6.1 Forward and back substitutions

The following ideas we use are hardly new and can be found in references such as [Seeger (2006)]. The basic idea of the Cholesky update method is to decompose $A = \lambda^2 I + QQ^T$ into the product of a lower triangular matrix $L$ with diagonal entries all 1, a diagonal matrix $\Lambda$ with strictly positive diagonal entries, and upper triangular $L^T$. See for example [Stoer and Bulirsch, Theorem 4.3.3, page 182].

Given such a decomposition, the system $Aw = y$ can be solved by a sequence of $m$ **forward and back substitutions**. The sequence can be described as solving for, in succession, $w_1, \ldots, w_{2m} = w$ using

\[
\begin{align*}
L_1 w_1 &= y \\
L_2 w_2 &= w_1 \\
& \vdots \\
L_m w_m &= w_{m-1} \\
L_m^T w_{m+1} &= \Lambda_m^{-1} w_m \\
& \vdots \\
L_1^T w_{2m} &= w_{2m-1} 
\end{align*}
\]
**Definition 6.1.** Suppose that for unit lower triangular $n \times n$ matrix $L$ there exists $n$-vectors $\mathbf{p}$ and $\mathbf{\beta}$ such that each entry of $L$ satisfies the following.

$$L_{i,j} = \begin{cases} p_i\beta_j & \text{if } i > j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Then we say that $L = L(\mathbf{p}, \mathbf{\beta})$.

Let $L = L(\mathbf{p}, \mathbf{\beta})$. Then we claim that Algorithm 2 for any $n$-vector $y$ correctly solves for $w$ the solution of $Lw = y$, and we claim that Algorithm 3 similarly works to find the solution of $L^T w = y$. For observe that if $Lw = y$, then for $i = 1$, $w_1 = y_1$, and for $2 \leq i \leq n$,

\[
y_i = \left( \sum_{j=1}^{i-1} p_i\beta_j w_j \right) + w_i
\]

\[
w_i = y_i - p_i \left( \sum_{j=1}^{i-2} \beta_j w_j \right) + \beta_{i-1} w_{i-1}
\]

Due to the special form of the $L_{i,j}$’s, the operational count and storage for this algorithm are both $O(n)$ for $n$ the number of observations used to fit the model. In fact we can count at most $4n$ total additions and multiplications and an $n$ cost allocation for $w$ for a total of at most $5n$ operations for one call to either Algorithm 2 or to Algorithm 3.

**Algorithm 2: Forward substitution** for $L = L(\mathbf{p}, \mathbf{\beta})$

```
input : $\mathbf{p}$ and $\mathbf{\beta}$ are $n$-vectors such that $L = L(\mathbf{p}, \mathbf{\beta})$ $y$ an $n$-vector
output: $w = (w_1, \ldots, w_n)^T$ is the solution of $Lw = y$
begin
    $w_1 \leftarrow y_1$ ;
    $c \leftarrow 0$ ;
    for $i \leftarrow 2$ to $n$ do
        $c \leftarrow c + \beta_{i-1} \ast w_{i-1}$ ;
        $w_i \leftarrow y_i - p_i \ast c$ ;
    end
end
```
Algorithm 3: Back substitution for $L = L(p, \beta)$

**input**: $p$ and $\beta$ are $n$-vectors such that $L = L(p, \beta)$, $y$ an $n$-vector

**output**: $w = (w_1, \ldots, w_n)^T$ is the solution of $L^T w = y$

**begin**

$w_n \leftarrow y_n$;
$c \leftarrow 0$;

for $i \leftarrow n - 1$ to 1 do

$c \leftarrow c + w_{i+1} * p_{i+1}$;

$w_i \leftarrow y_i - \beta_i * c$;

end

**end**

Suppose that $L = L_1 \cdots L_k$ where $L_i = L_i(p_i, \beta_i)$ for $i = 1, \ldots, k$. If we have matrix $\tilde{p}$ for which its first $k$ columns are $p_1, \ldots, p_k$ and matrix $\tilde{\beta}$ for which its first $k$ columns are $\beta_1, \ldots, \beta_k$, then Algorithm 4 for any $n$-vector $q$ will find the solution $p$ of $Lp = q$.

Algorithm 4: Solving for $Lp = q$

**input**: $\tilde{p}$ and $\tilde{\beta}$ are $n \times m$ matrices

$k$ an integer such that $1 \leq k \leq m$

$q$ an $n$-vector

**output**: $p$ the solution of $L_1 \cdots L_k p = q$

where each $L_i = L_i(\tilde{p}_{*,i}, \tilde{\beta}_{*,i})$ for $i = 1, \ldots, k$, with the $i$-th column of $\tilde{p}$ denoted by $\tilde{p}_{*,i}$ and the $i$-th column of $\tilde{\beta}$ denoted by $\beta_{*,i}$.

**begin**

$p \leftarrow q$;

for $i \leftarrow 1$ to $k$ do

Use Algorithm 2 to solve $p \leftarrow L_i^{-1} p$;

end

**end**
6.2 Rank-one update

Recall that for the special polynomial kernel of degree \( r = 2 \), we can write covariance matrix \( A \) in either of two equivalent forms

\[
A = \lambda^2 I + QQ^T = \lambda^2 I + \sum_{i=1}^{m} q_i \{ q_i^T \}
\]

The idea then is to start with \( \lambda^2 I \) and to proceed by adding one \( q_i \{ q_i^T \} \) at a time, recomputing each time the Cholesky factorization. At the \( j \)-th stage for \( 1 \leq j \leq m \), we have

\[
\lambda^2 I + \sum_{i=1}^{j} q_i \{ q_i^T \} = LL^T
\]

where \( L \) is unit lower triangular and \( \Lambda \) is diagonal.

Let \( p \) be the solution to \( Lp = q \), then

\[
LL^T + qq^T = \Lambda L^T + (Lp)(Lp)^T
\]

\[
= \Lambda L^T + Lpp^TL^T
\]

\[
= L \left( \Lambda + pp^T \right) L^T
\]

Given positive diagonal matrix \( \Lambda \), unit lower triangular matrix \( L \), and \( n \)-vector \( q \), Cholesky rank-one update finds \( p \) the solution of \( Lp = q \) and finds \( n \)-vector \( \beta \) and positive diagonal matrix \( \tilde{\Lambda} \) such that for \( \tilde{L} = \tilde{L}(p, \beta) \),

\[
\Lambda + pp^T = \tilde{L}(p, \beta)\tilde{\Lambda}\tilde{L}(p, \beta)^T
\]

\[
LL^T + qq^T = LL(p, \beta)\tilde{\Lambda}LL(p, \beta)^TL^T
\]

\[
= \left( LL(p, \beta) \right) \tilde{\Lambda} \left( LL(p, \beta) \right)^T
\]

Using Theorem G.1 from Appendix G we obtain correctness of Algorithm 5 for computing \( m \) rank-one updates for the decomposition of equation (10). Observe that in one call to Algorithm 5 there are exactly \( 0 + 1 + \cdots + (m - 1) = \binom{m}{2} \) calls to forward substitution Algorithm 2. The innermost for loop over \( j \) and \( i \) has at most \( 7mn \) assignments, additions, subtractions, multiplications, and divisions. The initialization of \( t_{new} \) adds another \( m \) operations. The total number of operations for one call to Algorithm 5 is therefore bounded by \( 5\binom{m}{2}n + 7mn + m = O(m^2n) \). The storage
for one call to Algorithm 5 is bounded by the sum of the sizes of input array \( Q \), output arrays \( \tilde{p} \), \( \tilde{\beta} \), and the \( n \)-vector that represents the diagonal of \( \tilde{\Lambda} \), for a total of \( mn + 2mn + n = O(mn) \).

If the output arrays from one call to Algorithm 5 are retained, then finding the solution \( w \) of \( Aw = y \) using equations (11) from Section 6.1 takes \( m \) calls to Algorithm 2, one multiplication of an \( n \)-vector by \( n \) scalars, and \( m \) calls to Algorithm 3, for a total bound of \( 5mn + n + 5mn = O(mn) \) operations.

Algorithm 5: Cholesky update solver

<table>
<thead>
<tr>
<th>input</th>
<th>( n \times m ) matrix ( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda^2 &gt; 0 ) a scalar</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output</th>
<th>( n \times m ) dimension matrices ( \tilde{p} ) and ( \tilde{\beta} ) and ( n \times n ) diagonal matrix ( \tilde{\Lambda} ) such that for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ L_i = L_i(\tilde{p}<em>{*,i}, \tilde{\beta}</em>{*,i}) \text{ for } i = 1,\ldots,m ]</td>
</tr>
</tbody>
</table>

we have the identity

\[ L_1 \cdots L_m \tilde{\Lambda} L_m^T \cdots L_1^T = \lambda^2 I + QQ^T \]

begin

\[ L_0 \leftarrow I ; \]

\[ \text{for } j \leftarrow 1 \text{ to } m \text{ do } \]

\[ \text{Use Algorithm 4 to find } \tilde{p}_{*,j} \leftarrow L_{j-1}^{-1} \cdots L_0^{-1} Q_{*,j} ; \]

\[ t_{\text{new}} \leftarrow 1 ; \]

\[ \text{for } i \leftarrow 1 \text{ to } n \text{ do } \]

\[ t_{\text{old}} \leftarrow t_{\text{new}} ; \]

\[ t_{\text{new}} \leftarrow t_{\text{old}} + \tilde{p}_{i,j}^2 / \tilde{\Lambda}_i ; \]

\[ \tilde{\beta}_{i,j} \leftarrow \tilde{p}_{i,j} / \left( \tilde{\Lambda}_i * t_{\text{new}} \right) ; \]

\[ \tilde{\Lambda}_i \leftarrow \tilde{\Lambda}_i * t_{\text{new}} / t_{\text{old}} ; \]

end

dep

We next discuss the last of our three fast methods for Gaussian process regression, conjugate gradient, a method whose speed for the special polynomial kernel relies on matrix multiplication by \( A \) having run time \( O(n) \) and not \( O(n^2) \).
7 Conjugate Gradient

Because matrix multiplication for the special polynomial kernel with degree $r$ is $O(n)$ and because covariance matrix $A$ is symmetric positive definite, we implement an iterative method, the conjugate gradient algorithm, to approximate the solution $w$ to the linear system $Aw = y$. Expositions can be found in numerous sources such as Algorithm 10.2.6 of [Golub and Van Loan, page 529] or Algorithm 8.7.3 of [Stoer and Bulirsch, page 607]. In Algorithm 6 we show pseudocode almost identical to that of Golub and Van Loan’s or that of Stoer and Bulirsch’s.

Because conjugate gradient’s search is along directions that are perpendicular relative to some norm, assuming exact arithmetic, conjugate gradient is guaranteed to converge to the exact solution in at most $n$ iterations. To deal with non-exact arithmetic, we halt search when the difference $\|y - Aw_k\|$ is within some specified tolerance $\epsilon > 0$ of $\|y\|$. As can be seen in Algorithm 6, each iteration has all steps $O(n)$ except for the one matrix multiplication which has number of operations $O(mn)$. Only four $n$-vectors $w$, $r$, $p$, and $Ap$ are required to be stored for an upper bound on the amount of storage required of $O(n)$.

The number of iterations of conjugate gradient is minimized when the eigenvalues of $A$ are relatively the same. Unfortunately for the particular form of polynomial kernels $K$ such a good condition on the eigenvalues is unlikely to hold—the $\lambda^2 \delta_{ij}$ term is what prevents $A$ from being singular, that is, non-positive definite. For this system determining the speed of convergence of conjugate gradient is simply a matter of trial and error.

We next turn to a discussion of how we evaluate each of our three fast methods’ performance versus that of a baseline method, quadratic regression, that represents the simplest alternative method to Gaussian process regression.
Algorithm 6: Conjugate gradient method

**input**: $Q$ an $n \times m$ matrix

- $\lambda^2 > 0$ an estimate of magnitude of noise term
- $y$ an $n$-vector
- $\epsilon > 0$, residual tolerance relative to $\|y\|$
- $w_0$ an initial estimate of $Aw = y$ solution

**output**: $w$ an estimate of the solution of $Aw = y$ such that $\|y - Aw\| < \epsilon \|y\|$

**begin**

$w \leftarrow w_0$

$r \leftarrow y - Aw = y - Aw_0$, initial residual$

p \leftarrow r$, new search direction for conjugate gradient$

\delta^2_{\text{next}} \leftarrow \|r\|^2$

**while** $\delta^2_{\text{next}} \geq ||y||^2 \epsilon$ **do**

Calculate $Ap = \lambda^2 p + Q(Q^T p)$

$\alpha \leftarrow \langle r, p \rangle / \langle p, Ap \rangle$

$r \leftarrow r - \alpha \cdot Ap$

$\delta^2 \leftarrow \delta^2_{\text{next}}$

$\delta^2_{\text{next}} \leftarrow \|r\|^2$

$\beta \leftarrow \delta^2_{\text{next}} / \delta^2$

$w \leftarrow w + \alpha \cdot p$

$p \leftarrow r + \beta \cdot p$

**end**

**end**
8 Training and Testing Procedure

8.1 Data set

We are given a training set and a testing set of, respectively, 180,045 and 20,229 observed \textit{ugriz} photometric filter and observed redshift values. We use samples \((X, y)\) from the training set to fit a model and then use the resulting model to compare predicted red shifts with observed redshifts from the testing set \((X^*, y^*)\).

Recall from Section 4 that from \(n \times d\) matrix \(X\) one can produce an \(n \times m\) matrix \(Q = Q(X)\) such that for \(q_1, \ldots, q_m\) the columns of \(Q\)

\[
A = \lambda^2 + \sum_{i=1}^{m} q_i \{q_i\}^T
\]

We can regard \(Q(\cdot)\) as a function that takes for any \(n' \geq 1\) length \(n'\) columns \(X'_1, \ldots X'_d\) and produces a new \(n' \times m\) matrix whose columns are the componentwise products of \(X'_1, \ldots, X'_d\) up to order \(r\), where \(r\) is the degree of the special polynomial kernel and where \(m = m(r, d) = O(d^r)\).

Let the testing set sample’s photometric filter observations \(X^*\) be expressed as a matrix

\[
X^* = \begin{bmatrix} U^* & G^* & R^* & I^* & Z^* \end{bmatrix}
\]

of dimension \(n^* \times d\) for \(d = 5\), and let the corresponding known testing set redshift vector be \(y^*\) of length \(n^*\). Then from equation (1) of Section 3 the predicted redshifts can be computed as follows

\[
y^* = E[y^*|X, y, x^*] = k(x^*)^T w
\]

where \(Aw = y\).

From Theorem F.1 of Appendix F, we obtain for one \(d\)-vector \(x^*\)

\[
\begin{align*}
k(x^*) &= Q(X)Q(x^*)^T \\
k(x^*)^T &= Q(x^*)Q(X)^T
\end{align*}
\]

We can therefore extend \(k(x^*)\) to be a function over all of the testing set photometric filter portion \(X^*\)

\[
k(X^*)^T = Q(X^*)Q(X)^T
\]
The $n^*$-vector of predicted redshifts for the testing set can therefore be calculated using matrix operations
\[
\hat{y}^* = Q(X^*)Q(X)^T w
\]
where $Aw = y$.

From the predicted redshifts the root mean square error (RMSE), a measure of the average distance between predicted and observed values, can be calculated as follows
\[
RMSE = \sqrt{\frac{\sum_{i=1}^{n^*} (y_i - \hat{y}_i^*)^2}{n^*}}
\]

### 8.2 Baseline—Quadratic Regression

Let a training set sample matrix have columns $U, G, R, I, Z$ of dimension $n \times 1$ and let $1$ denote the column of dimension $n \times 1$ whose entries are all 1. For two $n \times 1$ columns $A$ and $B$ let $AB$ denote the $n \times 1$ column whose entries come from the componentwise product of $A$ and $B$: $(AB)_i = A_i B_i$ for all $i = 1, \ldots, n$. Then quadratic regression is the simplest linear model that use $1$, columns $U, G, R, I, Z$ and interactions between columns. For this application, for quadratic regression one determines $m = 21$ coefficients $\hat{β}_1, \ldots, \hat{β}_m$ that minimize the sum of squares error $\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ of the residuals $y - \hat{y}$, where predicted redshifts $\hat{y}$ on the training set sample are given by
\[
\hat{y} = \hat{β}_1 U + \hat{β}_2 G + \hat{β}_3 R + \hat{β}_4 I + \hat{β}_5 Z + \\
\hat{β}_6 UU + \hat{β}_7UG + \hat{β}_8 UR + \hat{β}_9 UI + \hat{β}_{10} UZ + \\
\hat{β}_{11} GG + \hat{β}_{12} GR + \hat{β}_{13} GI + \hat{β}_{14} GZ + \\
\hat{β}_{15} RR + \hat{β}_{16} RI + \hat{β}_{17} RZ + \\
\hat{β}_{18} II + \hat{β}_{19} IZ + \\
\hat{β}_{20} ZZ
\]
\[
\hat{y} = \tilde{Q} \hat{β}
\]

Note that for some positive diagonal matrix $D$, $\tilde{Q} = QD$. For the special polynomial kernel of degree $r = 2$, the columns of $Q$ and $\tilde{Q}$ are the same except for factors of $\sqrt{2}$ that correspond either to duplicated interaction terms or to the coefficient 2 in
\[
(1 + \langle x_a, x_b \rangle)^2 = 1 + 2 \langle x_a, x_b \rangle + \langle x_a, x_b \rangle^2
\]
The non-1 elements of the diagonal $D$ are these factors of $\sqrt{2}$.

By the theory of least squares, the predicted values $\hat{y}$ for observations $X$ and $y$ are given by

$$\hat{y} = \hat{Q}\hat{\beta}$$

where $\hat{\beta}$ is the solution of

$$\hat{Q}^T\hat{Q}\hat{\beta} = \hat{Q}^Ty$$

Observe that $\hat{Q}^T\hat{Q}\hat{\beta} = \hat{Q}^Ty$ is an $m \times m$ linear system since $\hat{Q}$ has the same $n \times m$ dimensions as $Q$.

But also observe that for any positive diagonal matrix $D$, if $Q$ satisfies $QD = \hat{Q}$, while the solution $\hat{\beta}'$ of

$$Q^TQ\hat{\beta}' = Qy$$

(12)

differs from $\hat{\beta}$, the predicted values

$$\hat{y}' = Q\hat{\beta}'$$

(13)

are the same as predicted values $\hat{y}$.

Our implementation of quadratic regression uses equations (12) and (13) to predict redshift values.

Fast fitting of Gaussian process regression models using the special polynomial kernel enables increasing the size of the sample from the training set, and the hope is that the RMSE for the resulting Gaussian process regression models will be substantially less than the RMSE for the baseline quadratic regression.

8.3 Bootstrap

As noted in [Way and Srivastava (2006)], one RMSE may or may not be representative of the values one would obtain from having different training and or testing sets. To partially overcome the difficulty of having only one training and one testing set, as suggested by [Way and Srivastava (2006)], one can use a statistical method known as the bootstrap. Given training set $X$, one assumes that $X$ can serve as the entire universe of possible observations, with the assumption that $X$ has the correct frequencies. One then creates samples of some size $n$ by sampling with replacement from the rows of $X$, so that a row of $X$ representing one set of observations on a galaxy may be repeated or omitted in the new sample. For each additional sample
from \( X \), one can fit a new model, using either Gaussian process regression or quadratic regression, and from this new model one can compute a new \( \text{RMSE} \). For each size \( n \) ranging from 1000 to 180045 and for each of reduced rank, conjugate gradient, cholesky update, and quadratic regression, we randomly generate 100 samples, recording the resulting \( \text{RMSE} \)'s and run times.

### 8.4 Hardware and software environment

We have run our experiments on 3.40GHz Pentium D CPU machines with 3.24GB of RAM and with Matlab R2006a Version 7.2.0.232 using operating system Microsoft Windows XP SP2.
Table 2: Time (seconds) for 100 Bootstrap RMS Calculations

<table>
<thead>
<tr>
<th>Training Set Size</th>
<th>QR</th>
<th>Reduced Rank</th>
<th>Conjugate Gradient</th>
<th>Cholesky Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10</td>
<td>10</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>5000</td>
<td>11</td>
<td>12</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>10000</td>
<td>14</td>
<td>15</td>
<td>41</td>
<td>39</td>
</tr>
<tr>
<td>50000</td>
<td>37</td>
<td>40</td>
<td>205</td>
<td>163</td>
</tr>
<tr>
<td>100000</td>
<td>68</td>
<td>74</td>
<td>590</td>
<td>383</td>
</tr>
<tr>
<td>180045</td>
<td>116</td>
<td>126</td>
<td>1090</td>
<td>687</td>
</tr>
</tbody>
</table>

Table 3: Number of iterations for conjugate gradient around 200

<table>
<thead>
<tr>
<th>Training Set Sample Size</th>
<th>Average Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>150.88</td>
</tr>
<tr>
<td>5000</td>
<td>172.78</td>
</tr>
<tr>
<td>10000</td>
<td>181.03</td>
</tr>
<tr>
<td>50000</td>
<td>196.42</td>
</tr>
<tr>
<td>100000</td>
<td>198.56</td>
</tr>
<tr>
<td>180045</td>
<td>200.30</td>
</tr>
</tbody>
</table>

9 Results

9.1 Time for 100 Bootstrap Samples

As shown in Table 2, as we vary the size of the sample from the training set from 1,000 through 180,045, the time to run 100 bootstrap samples of the RMSE increases roughly linearly for all of the quadratic regression, reduced rank, conjugate gradient, and cholesky update methods. As expected the simplest model, quadratic regression, is the fastest, but the next simplest in computation, reduced rank, is almost as fast. The slowest method, conjugate gradient, still completes all 100 RMSE estimates in less than 20 minutes.

In Table 3 we show the average number of iterations as sample set size $n$ varies for 100 bootstrap linear solutions of $Aw = y$ using conjugate gradient. We see this average number increases slowly once past $n = 50000$ to approximately 200 iterations per solution.
In Figure 1 we graph the RMSE for baseline method quadratic regression and fast methods reduced rank, conjugate gradient, and Cholesky update. The horizontal axis represents sorting each 100 RMSE values from smallest to highest so that the RMSE at model number 10 corresponds to the 10th smallest RMSE for that method and that sample size. We see that for small $n$ much smaller than the size of the testing set, the RMSE curves for all three fast methods, methods that correspond to fitting using Gaussian process regression models, are below the curve for baseline quadratic regression. For such small $n$ bootstrap sample RMSE’s range all the way from about 0.024 to more than 0.2, an order of magnitude increase.

Figure 1: Comparison of Methods for $n = 1000$
In Figure 2 we obtain the same graph but now for size $n = 10000$. Here we see that the differences between the RMSE curves for baseline quadratic regression versus our fast methods has disappeared. Note that the range of RMSE values has also decreased compared to Figure 1.

Finally in Figure 3 we compare quadratic regression only with the fast conjugate gradient and Cholesky update methods. Now the range of RMSE values has shrunk to be between 0.024 to 0.025, a difference exaggerated by the scale chosen for the graph. If anything the baseline quadratic regression now performs slightly better than the two fast methods. Note that the reduced rank method’s curve is not shown because including it would have distorted the scale of the graph so that all three of the other methods would have been smashed together in a near horizontal line.
Figure 3: Comparison of Methods for $n = 180045$
The reduced rank RMSE curves for increasing $n$ are shown in Figure 4. For increased $n$ the curves decrease until $n = 180045$ after which the curve rapidly increases among its highest RMSE values. We believe this is consistent with reduced rank having the worst theoretical vulnerability to the effects of roundoff error.

Figure 4: Comparison of reduced rank for varying $n$

![Reduced Rank RMSE for different n](image-url)
10 Conclusions

We have implemented three fast methods for fitting Gaussian process regression using special polynomial kernels of degree 2, methods whose run time increases approximately linearly not cubically relative to the size of the number of observations used to fit the model and whose storage also only increases linearly. We are able to fit 100 Gaussian process regression models and make 20,229 predictions using each model in under 20 minutes on commodity hardware. Of these methods, reduced rank is the fastest, but it appears to be vulnerable to roundoff error. A compromise between the fastest, reduced rank, and the slowest, conjugate gradient, may be Cholesky rank one update.

Our experiments indicate that while for small sample sizes Gaussian process regression produces smaller root mean square errors than does quadratic regression, for larger sample sizes Gaussian regression performs no better.
11 Further research

11.1 Definition of hyperparameters

Define matrix $Q'$ to be the matrix whose $m = 21$ columns are in respective order $1, U, G, R, I, Z, UU, \sqrt{2}UG, \sqrt{2}UR, \sqrt{2}UI, \sqrt{2}UZ, GG, \sqrt{2}GR, \sqrt{2}GI, \sqrt{2}GZ, RR, \sqrt{2}RI, \sqrt{2}RZ, II, \sqrt{2}IZ,$ and $ZZ$. Note that matrix $Q'$ is the same as matrix $Q$ for the special polynomial kernel from equation (4) of Section 4 except that the columns $U, G, R, I,$ and $Z$ do not have the factor of $\sqrt{2}$. That factor of $\sqrt{2}$ is in some sense an artifact from the specific form of the kernel in equation (4) that requires a factor of 2 in front of the first order terms corresponding to $\langle x_a, x_b \rangle$.

Let $q'_1, \ldots, q'_m$ be the columns of $Q'$. Suppose we choose $e_1, \ldots, e_m \geq 0$ and define $m \times m$ diagonal matrix $D$ so that the $i$th diagonal element of $D$ is $\sqrt{e_i}$ for $i = 1, \ldots, n$. Then if we write

$$Q = Q'D$$

we observe that

$$QQ^T = \sum_{i=1}^{m} e_i q'_i \{q'_i\}^T$$

That is, we can obtain $Q$ from $Q'$ and can generalize $Q$ to allow for up to $m$ scaling factors for the columns of $Q'$. Observe the columns of $Q'$ include all interactions term up to order $r$ between the previously given photometric filter observations.

The estimate of the noise magnitude $\lambda^2$ and whatever scaling factors are chosen to form diagonal matrix $D$ are parameters that influence the resulting Gaussian process regression model and that need to somehow be estimated before a model can be fitted. Such parameters are referred to in the literature as hyperparameters $\Theta$, for example, as discussed in [Rasmussen and Williams (2006), Chapter 5].

11.1.1 One hyperparameter $\lambda^2$

Suppose we decide to have only one hyperparameter $\Theta = (\lambda^2)$. Then as in equation (8), we can set $(e_1, \ldots, e_{21})$ to be the concatenation of the sequences $(1)$, for the 1 column, $(2, 2, 2, 2)$, for the 5 first order columns $U, G, R, I,$
and $Z$, and $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$ for the 15 second order interactions between columns—a choice of the $e_k$ that corresponds to the special polynomial kernel of degree $r = 2$ of the form

$$K(x_a, x_b) = (1 + \langle x_a, x_b \rangle)^2.$$  

### 11.1.2 Three hyperparameters ($\lambda^2, c_0, c_1, c_2$)

Suppose we decide that the zero-order one’s vector $I$, the first-order linear terms $U, G, R, I, Z$, and the second-order interaction terms $UU, UG, UR, UI, UZ, GG, GR, GI, GZ, RR, RI, RZ, II, IZ$, and $ZZ$ possibly should have three different scaling factors, $c_0$, $c_1$, and $c_2$, respectively, each scaling factor corresponding to a different order. Then the hyperparameter vector has dimension 4 with $\Theta = (\lambda^2, c_0, c_1, c_2)$, and we can set $(e_1, \ldots, e_{21})$ to be the concatenation of sequences $(c_0)$, $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,)$.

Note that this case actually generalizes the following: suppose we decide that each photometric measurement column $U, G, R, I, Z$ should have its own scaling factor $f_U, f_G, f_R, f_I, f_Z$, respectively. But each term of a polynomial kernel is of the form $\langle x_a, x_b \rangle^s$ so that the combined scaling factor for this term is $f_s = (f_U f_G f_R f_I f_Z)^s$. There are therefore for $r = 2$ only three scaling factors $c_0$ for the 1’s term, $c_1 = (f_U f_G f_R f_I f_Z)$ for the $s = 1$ term, and $c_2 = (f_U f_G f_R f_I f_Z)^2$ for the $s = 2$ term.

### 11.1.3 Twenty-two hyperparameters

Suppose we decide that each column of $Q^\prime$ should have its own scaling factor, then $\Theta = (\theta_1, \ldots, \theta_{m+1})$ where $\theta_1 = \lambda^2$ and $\theta_{i+1} = e_i$ for $i = 1, \ldots, m$.

### 11.2 Marginal likelihood

Let **marginal likelihood** $p(y | X, \Theta)$ be the probability of observing redshift data $y$ given photometric filter observations $X$ and hyperparameters $\Theta$. Then one method of optimizing prediction with respect to $\Theta$ is to simply find the maximum value of $p(y | X, \Theta)$ for fixed $y, X$, varying $\Theta$. Note that maximizing $p(y | X, \Theta)$ is equivalent to maximizing its natural logarithm $\log p(y | X, \Theta)$, and thus maximizing $p(y | X, \Theta)$ is the same as minimizing the negative of the logarithm $- \log p(y | X, \Theta)$.
For Gaussian process regression where the covariance matrix is \( A = A(X, \Theta) \), as shown in [Rasmussen and Williams (2006), Section 5.4, pages 113-114], the log of the maximum likelihood is given by

\[
\log p(y|y, \Theta) = -\frac{1}{2} y^T A^{-1} y - \frac{1}{2} \log \det (A) - \frac{n}{2} \log 2\pi
\]

and its partial derivatives with respect to \( \Theta \) are given by

\[
\frac{\partial}{\partial \theta_j} \log p(y|X, \Theta) = \frac{1}{2} \text{trace} \left[ \left( A^{-1} y \right) \left( A^{-1} y \right)^T - A^{-1} \right] \frac{\partial A}{\partial \theta_j}
\]

In the following numerical experiments optimizing hyperparameters, we use Matlab’s function \texttt{fminsearch}; therefore, we only use the expression for marginal likelihood, equation (14), and not its gradient. The only quantity in equation (14) not previously calculated is \( \log \det (A) \). But the Cholesky update method of Section 6 calculates a diagonal matrix \( \Lambda_m \) such that the product of the elements of \( \Lambda_m \) gives the determinant of covariance matrix \( A \). The sum of the logarithms of the diagonal elements of \( \Lambda_m \) therefore gives \( \log \det (A) \).

### 11.3 Preliminary results on optimal hyperparameters

In Table 4 we show single hyperparameter \( \lambda^2 \) that is estimated to maximize the marginal likelihood for varying training set sample size \( n \) from 100 to 180,045. We see that the estimates approach a value of approximately 0.0006.

In Table 5 we extend optimization of hyperparameters for special polynomial kernels, \( r = 2 \), of the form

\[
K(x_i, x_j) = \theta_2 + \theta_3 \langle x_i, x_j \rangle + \theta_4 \langle x_i, x_j \rangle^2
\]

We see that although the estimates for \( \lambda^2 \) are consistent, still approximately 0.0006, the estimates for \( \theta_2, \theta_3, \theta_4 \) simply show no consistency between training set sample sizes. The second order interaction term \( \theta_4 \) in particular for most estimates almost if not completely vanishes.

As shown in Figure 5, neither the value nor the range of the RMSE varies much for varying \( \lambda^2 \). Here we use values of \( \lambda^2 \) from 0.01 to the value 0.0006 estimated from maximizing the marginal likelihood. In fact the highest RMSE appear to occur for smaller \( \lambda^2 \).
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Training Set Size & Time (seconds) & \hat{\lambda}^2 \\
\hline
100 & 1 & 0.0003258 \\
500 & 1 & 0.0003960 \\
1000 & 1 & 0.0003908 \\
5000 & 4 & 0.0004674 \\
10000 & 6 & 0.0004618 \\
50000 & 37 & 0.0005523 \\
100000 & 82 & 0.0005935 \\
150000 & 134 & 0.0005976 \\
180045 & 160 & 0.0005896 \\
\hline
\end{tabular}
\caption{Estimation of \(\hat{\lambda}^2\) alone}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
Training Set Size & Time (seconds) & \hat{\lambda}^2 & \hat{\theta}_2 & \hat{\theta}_3 & \hat{\theta}_4 \\
\hline
100 & 2 & 0.0004 & 0.0194 & 3.5360 & 0.0000 \\
1000 & 19 & 0.0004 & 2.4089 & 0.0000 & 0.2895 \\
10000 & 69 & 0.0005 & 2.6416 & 2.1047 & 0.0463 \\
100000 & 1186 & 0.0006 & 3.1826 & 1.1381 & 0.0008 \\
180045 & 2023 & 0.0006 & 1.6444 & 3.4223 & 0.0009 \\
\hline
\end{tabular}
\caption{Estimation of four hyperparameters not consistent}
\end{table}
Figure 5: RMSE for varying $\lambda^2$ using Cholesky update and $n = 180045$
To show that relatively small change in root mean square error is not caused by the relatively small range of $\lambda^2$ tested, we also used Cholesky update to examine much larger $\lambda^2$. As shown in Figure 6, we see that the RMSE’s values increase as we increase $\lambda^2$ from $10^2 = 100$ to $100^2 = 10000$ and finally to $1000^2 = 1000000$.

### 11.4 Final note

We agree with [Crunk (2006)] that further progress is likely to only occur with deducing a model that better fits the data.
A Redshift

A galaxy’s redshift $z$ is the change in the wavelength divided by the initial wavelength of the electromagnetic waves that are emitted by the galaxy.

$$z = \frac{\Delta \lambda}{\lambda_0}$$

Redshifts originate from the Doppler formula which determines how rapidly the distance between the earth and a galaxy is increasing. A redshift of a galaxy indicates that it is moving away from the earth. By calculating the redshift of a galaxy, scientists can determine many characteristics of that galaxy and the universe. For example, a redshift can determine the distance between the galaxy and the earth. Also the redshifts of galaxies help scientists theorize more precisely the origins of the universe. The calculations of the redshift have given more credit to the popular Big Bang Theory [Seeds, page 382].

There are two methods that are used to collect the data needed to calculate a red shift: photometry and spectroscopy. Photometry uses multiple filters, each designed to collect particular wavelengths of the electromagnetic spectrum. These filters collect data from 5 band passes- $u$, $g$, $r$, $i$, and $z$, which range from the ultraviolet to the infrared. Broadband photometry is designed to collect 5 pieces of data (one from each band pass) which comes from many objects, in our case galaxies, in a particular region of space. Spectroscopy, on the other hand, uses a diffraction grating which will split the light emitted from the galaxies into the different wavelengths to collect the spectral data. Spectroscopy is often preferred because there it collect more data from its object and therefore it is more accurate. However, since photometry is cheaper and faster, because it collects more data at one time, it was the preferred method used by Dr. Way and Dr. Srivastava.

There are two main approaches used to calculate photometric redshifts. The spectral energy distribution fitting (SED fitting), also known as template fitting, compares the spectral energy distribution converted from the observed data with the spectral energy distribution of a known template. The training-set method (TS method) uses the redshifts calculated of other similar galaxies and the newly observed data to calculate the redshifts of these new galaxies. The SED fitting approach has typically been preferred because the TS method requires samples of galaxies that are similar in mag-
Dr. Way and Dr. Srivastava described various TS methods have been recently used and developed in their report, that enabled them to estimate the photometric redshifts of galaxies. Using the TS method, the scientists wanted to find the best mathematical model for the redshift data. Some of the techniques they used, both linear and non-linear, were polynomial fittings, support vector machines, and artificial neural networks. In particular, the scientists compared the neural network ANNz model, the linear and quadratic models, the ensemble model, and Gaussian processes to find the best model for the red shifts. The Gaussian process regression was the focus of our problem.
B Sherman-Morrison-Woodbury Formula

Equation (A.9) from [Rasmussen and Williams (2006), Appendix A, page 201], the Sherman-Morrison-Woodbury formula, states that if all matrix inverses on the right hand side exist

\[(Z + UWV^T)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^TZ^{-1}U)^{-1}V^TZ^{-1}\] (16)

for \(n \times n\) matrix \(Z\), \(n \times m\) matrix \(U\), \(m \times m\) matrix \(W\), and \(n \times m\) matrix \(V\).

B.1 Application to reduced rank reduction

Let \(I_{n,n}\) be the \(n \times n\) identity matrix, \(Q\) be a \(n \times m\) matrix, \(I_{m,m}\) be the \(m \times m\) identity matrix, and \(\lambda^2 > 0\). Then using \(Z = \lambda^2I_{n,n}\), \(U = V = Q\), and \(W = I_{m,m}\) in equation (16) we obtain

\[(\lambda^2I_{n,n} + QQ^T)^{-1} = \frac{1}{\lambda^2} \left[I_{n,n} - Q(\lambda^2I_{m,m} + Q^TQ)^{-1}Q^T\right]\] (17)

B.2 Proof

This formula can be verified after some algebra by multiplying \(Z + UWV^T\) by the formula’s right hand side. Let \(F\) be this product, then

\[
\begin{align*}
F &= ZZ^{-1} - F_1 + F_2 - F_3 \\
&= I - F_1 + F_2 - F_3 \\
F_1 &= U(W^{-1} + V^TZ^{-1}U)^{-1}V^TZ^{-1} \\
F_2 &= UWV^TZ^{-1} \\
F_3 &= UWV^TZ^{-1}U(W^{-1} + V^TZ^{-1}U)^{-1}V^TZ^{-1}
\end{align*}
\]

Note that \(F_1, F_2, F_3\) all begin with \(U\) and end with \(V^TZ^{-1}\).

\[F_1 - F_2 + F_3 = UF_4V^TZ^{-1}\]

where

\[
\begin{align*}
F_4 &= (W^{-1} + V^TZ^{-1}U)^{-1} - W + WV^TZ^{-1}U(W^{-1} + V^TZ^{-1}U)^{-1} \\
&= WW^{-1}(W^{-1} + V^TZ^{-1}U)^{-1} - W + WV^TZ^{-1}U(W^{-1} + V^TZ^{-1}U)^{-1} \\
&= 0
\end{align*}
\]
since collecting the first and third terms in $F_4$ gives

$$I = (W^{-1} + V^T Z^{-1} U)(W^{-1} + V^T Z^{-1} U)^{-1}$$

$$W = (W W^{-1} + W V T Z^{-1} U)(W^{-1} + V^T Z^{-1} U)^{-1}$$

### B.3 Motivation

As a partial motivation, suppose we wanted to write $(Z + H)^{-1}$ in a form symmetric in $Z$ and $H$ with as many $Z^1$ terms as we can find. Pretend $Z$ and $H$ are simple numbers not matrices.

$$\frac{1}{Z + H} = \frac{Z}{Z(Z + H)}$$

$$= \frac{Z + H - H}{Z(Z + H)}$$

$$= \frac{1}{Z} - \frac{H}{Z(Z + H)}$$

$$= \frac{1}{Z} - \frac{1}{Z(Z + H)}$$

$$= \frac{1}{Z} - \frac{1}{Z(\frac{1}{Z} + \frac{1}{Z})}$$

$$= \frac{1}{Z} - \frac{1}{\frac{1}{Z} + \frac{1}{Z}} * \frac{1}{Z}$$

Now if we pretend that $U$ and $V^T$ are invertible, substituting $H = U W V^T$ and aiming for the dimensionalities inside the innermost inversion to be $m \times m$, we obtain the form of the matrix inversion lemma.

### C Block Matrix Inversion

We have the following from equation (A.13) of [Rasmussen and Williams (2006), page 202] Let block matrix $A$ and its inverse $A^{-1}$ have form

$$A = \begin{pmatrix} P & Q \\ R & S \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} \tilde{P} & \tilde{Q} \\ \tilde{R} & \tilde{S} \end{pmatrix}$$
Then if all inverses shown are well-defined

\[
\begin{align*}
\tilde{P} &= N \\
\tilde{Q} &= -NQ^{-1} \\
\tilde{R} &= -S^{-1}RN \\
\tilde{S} &= S^{-1} + S^{-1}RNQS^{-1}
\end{align*}
\]

\[
\begin{cases}
\tilde{P} = N \\
\tilde{Q} = -NQ^{-1} \\
\tilde{R} = -S^{-1}RN \\
\tilde{S} = S^{-1} + S^{-1}RNQS^{-1}
\end{cases}
\]

where \( N = (P - QS^{-1}R)^{-1} \)

### C.1 Proof of block matrix inversion

We simply compute

\[
\begin{align*}
P\tilde{P} + Q\tilde{R} &= PN + Q(-S^{-1}RN) \\
&= (P - QS^{-1}R)N \\
&= I \\
P\tilde{Q} + Q\tilde{S} &= P(-NQS^{-1}) + Q(S^{-1} + S^{-1}RNQS^{-1}) \\
&= (-PN + I + QS^{-1}RN)QS^{-1} \\
&= (-P + QS^{-1}R)NQS^{-1} + QS^{-1} \\
&= -IQS^{-1} + QS^{-1} \\
&= 0 \\
R\tilde{P} + S\tilde{R} &= RN + S(-S^{-1}RN) \\
&= 0 \\
R\tilde{Q} + S\tilde{S} &= R(-NQS^{-1}) + S(S^{-1} + S^{-1}RNQS^{-1}) \\
&= -RNQS^{-1} + I + RNQS^{-1} \\
&= I
\end{align*}
\]

### C.2 Motivation for block matrix inversion

Suppose that one regarded \( P, Q, R, S \) as real numbers, then by the standard formula for inverting a \( 2 \times 2 \) matrix

\[
A^{-1} = \frac{1}{PS - QR} \begin{pmatrix} S & -Q \\ -R & P \end{pmatrix} = \begin{pmatrix} \frac{S}{PS - QR} & -\frac{Q}{PS - QR} \\ -\frac{R}{PS - QR} & \frac{P}{PS - QR} \end{pmatrix}
\]
C.3 Application to covariance matrices

Let covariance matrix $\Sigma$ and its inverse $\Sigma^{-1}$ be given by

$$
\Sigma = \begin{pmatrix} A & C \\ C^T & B \end{pmatrix}, \quad \Sigma^{-1} = \begin{pmatrix} \tilde{A} & \tilde{C} \\ \tilde{C}^T & \tilde{B} \end{pmatrix}
$$

Then using the above we obtain

$$
\begin{align*}
\tilde{A} &= N \\
\tilde{C} &= -NCB^{-1} \\
\tilde{C}^T &= -B^{-1}C^TN \\
\tilde{B} &= B^{-1} + B^{-1}C^TN CB^{-1}
\end{align*}
$$

where $N = (A - CB^{-1}C^T)^{-1}$

Rewrite the equation for $\tilde{B}$ as

$$
\begin{align*}
\tilde{B} &= B^{-1} + B^{-1}C^TN CB^{-1} \\
&= B^{-1} + B^{-1}C^T(A - CB^{-1}C^T)^{-1}CB^{-1} \\
\tilde{B} &= B^{-1} - B^{-1}((-C)^T)((A^{-1})^{-1} + (C^T)^TB^{-1}(-C)^T)^{-1}(C^T)^TB^{-1}
\end{align*}
$$

Setting $Z = B, U = (-C)^T, V = (C^T)^T = C, W = A^{-1}$, we obtain from the matrix inversion lemma

$$
\tilde{B} = (B - C^TA^{-1}C)^{-1}
$$

D Conditional distribution for multivariate normal

If we see a probability density function of the form

$$
p(x) = c \cdot \exp\left(-\frac{1}{2}(x - m)^T\Sigma^{-1}(x - m)\right)
$$

then we conclude that the random vector given by $x$ has a multivariate normal distribution with mean $\mu_x = m$ and covariance matrix $\Sigma_x = \Sigma$. Note that constant $c$ is assumed to be whatever value is needed so that the total integral of the density function $p$ is 1 so we often don’t have to explicitly compute it.
Suppose we know that \( p(x) \) is the probability density function of some multivariate normal distribution. If we can write

\[
p(x) = c \cdot \exp \left( -\frac{1}{2} g(x) \right)
\]

\[
g(x) = x^T Dx - y^T Dx - x^T Dy + h
\]

where \( h \) does not contain \( x \), then \( \Sigma = D^{-1} \) must be the covariance matrix and \( \mu_x = y \) must be the mean of \( x \).

Let the joint density of random vectors \( b \) and \( a \) be given by a multivariate normal distribution

\[
x = \begin{pmatrix} a \\ b \end{pmatrix}
\]

\[
\mu_x = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}
\]

\[
\Sigma_x = \begin{pmatrix} \Sigma_a & \Sigma_{a,b} \\ \Sigma_{a,b}^T & \Sigma_b \end{pmatrix} = \begin{pmatrix} A & C \\ C^T & B \end{pmatrix}
\]

\[
\Sigma_x^{-1} = \begin{pmatrix} \tilde{A} & \tilde{C} \\ \tilde{C}^T & \tilde{B} \end{pmatrix}
\]

By definition the conditional density of \( b \) given \( a \) is

\[
p(b|a) = \frac{p(a,b)}{p(a)}
\]

But note that if \( a \) is fixed since it is assumed given, \( p(a) \) is a constant and can be ignored since “constants work out”.

\[
g(b) = \begin{pmatrix} a - \mu_a \\ b - \mu_b \end{pmatrix}^T \begin{pmatrix} \tilde{A} & \tilde{C} \\ \tilde{C}^T & \tilde{B} \end{pmatrix} \begin{pmatrix} a - \mu_a \\ b - \mu_b \end{pmatrix}
\]

Let us denote \( \tilde{a} = a - \mu_a \) and \( \tilde{b} = b - \mu_b \)

\[
g(b) = b^T \tilde{B} \tilde{b} + \tilde{a}^T \tilde{C} \tilde{b} + b^T \tilde{C} \tilde{a} + h_1
\]

\[
= b^T \tilde{B} b - \mu_b^T \tilde{B} b - b^T \tilde{B} \mu_b + (a - \mu_a)^T \tilde{C} b + b^T \tilde{C} (a - \mu_a) + h_2
\]

We see that the term involving \( b^T \) on the left is

\[
b^T \tilde{B} m = b^T \left( \tilde{B} \mu_b - \tilde{C}^T (a - \mu_a) \right)
\]

\[
= b^T \tilde{B} \left( \mu_b - \tilde{B}^{-1} \tilde{C}^T (a - \mu_a) \right)
\]

43
Recalling that

\[ \tilde{B} = (B - C^T A^{-1} C)^{-1} \]
\[ \tilde{B}^{-1} = B - C^T A^{-1} C \]

We are able to simplify

\[-\tilde{B}^{-1} C^T = -(B - C^T A^{-1} C)(-B^{-1} C^T N)\]
\[= (B - C^T A^{-1} C)(B^{-1} C^T)(A - CB^{-1} C^T)^{-1}\]
\[= (1 - C^T A^{-1} C B^{-1} C^T)C^T (A - CB^{-1} C^T)^{-1}\]
\[= C^T - C^T A^{-1} C B^{-1} C^T (A - CB^{-1} C^T)^{-1}\]
\[= C^T (1 - A^{-1} C B^{-1} C^T)(A - CB^{-1} C^T)^{-1}\]
\[= C^T A^{-1} (A - CB^{-1} C^T)(A - CB^{-1} C^T)^{-1}\]
\[= C^T A^{-1} \]

We conclude that \( b | a \) has a multivariate normal distribution with

\[ \Sigma_{b|a} = \tilde{B} \]
\[= B - C^T A^{-1} C \] (18)
\[ \mu_{b|a} = \mu_b - \tilde{B}^{-1} \tilde{C} (a - \mu_a) \] (19)
\[= \mu_b + C^T A^{-1} (a - \mu_a) \] (20)
Matrix multiplication is time, space $O(d^r n)$

**Theorem E.1.** Let $X$ be an $n \times d$ matrix whose $k$th row is denoted by $x_k$. Let $b_{M_1, \ldots, M_s}$ for $M_1, \ldots, M_s = 1, \ldots, d$ be defined by componentwise multiplication $\ast$ of columns $M_1, \ldots, M_s$ of $n \times d$ matrix $X$.

$$b_{M_1, \ldots, M_s} \overset{\text{def}}{=} X_{\cdot M_1} \ast \cdots \ast X_{\cdot M_s}$$

Suppose $n \times n$ matrix $A$ has entries of the form

$$A_{i,j} = \lambda^2 \delta_{i,j} + \sum_{s=0}^{r} c_s \langle x_i, x_j \rangle^s$$

Let $w$ be an $n$-vector, then $v = Aw$ can be computed in time and space $O(d^r n)$ from the formula

$$Aw = \lambda^2 w + \sum_{s=0}^{r} c_s \left[ \sum_{1 \leq M_1, \ldots, M_s \leq d} b_{M_1, \ldots, M_s} \left\{ b_{\overline{M_1}, \ldots, \overline{M_s}}^T w \right\} \right] \tag{22}$$

**Proof.** The $i$th component $v_i$ of $v = Aw$ is given by

$$v_i = \sum_{j=1}^{n} A_{i,j} w_j$$

$$= \lambda^2 w_i + \sum_{j=1}^{n} \left( \sum_{s=0}^{r} c_s \langle x_i, x_j \rangle^s w_j \right)$$

$$= \lambda^2 w_i + \left( \sum_{s=0}^{r} c_s \sum_{j=1}^{n} \langle x_i, x_j \rangle^s w_j \right)$$

We have therefore reduced the problem of fast matrix multiplication to calculating for $0 \leq s \leq r$ in time and space $O(d^s n)$ all of

$$v_i^{(s)} = \sum_{j=1}^{n} \langle x_i, x_j \rangle^s w_j \text{ for } i = 1, \ldots, n$$

For the following we adopt the convention that a product of zero terms
is by default the number 1. The $i$-th component $v_i^{(s)}$ of $v^{(s)}(w)$ is given by

$$
v_i^{(s)} = \sum_{j=1}^{n} \langle x_i, x_j \rangle^s w_j
$$

$$
= \sum_{j=1}^{n} \left( \sum_{1 \leq M \leq d} X_{i,M} X_{j,M} \right)^s w_j
$$

$$
= \sum_{j=1}^{n} \prod_{t=1}^{s} \left( \sum_{1 \leq M_t \leq d} X_{i,M_t} X_{j,M_t} \right) w_j
$$

$$
v_i^{(s)} = \sum_{j=1}^{n} \left( \sum_{1 \leq M_1, \ldots, M_s \leq d} X_{i,M_1} \cdots X_{i,M_s} X_{j,M_1} \cdots X_{j,M_s} \right) w_j
$$

Changing the order of summation we finish the proof of our theorem as follows.

$$
v_i^{(s)} = \sum_{1 \leq M_1, \ldots, M_s \leq d} \sum_{j=1}^{n} (X_{i,M_1} \cdots X_{i,M_s} X_{j,M_1} \cdots X_{j,M_s} w_j) \quad (23)
$$

$$
v_i^{(s)} = \sum_{1 \leq M_1, \ldots, M_s \leq d} X_{i,M_1} \cdots X_{i,M_s} \left( \sum_{j=1}^{n} X_{j,M_1} \cdots X_{j,M_s} w_j \right) \quad (24)
$$

**Corollary E.2.** Under the same assumptions as Theorem E.1,

$$
A = \lambda^2 I + \sum_{s=0}^{r} c_s \left[ \sum_{1 \leq M_1, \ldots, M_s \leq d} b_{M_1, \ldots, M_s} b_{M_1, \ldots, M_s}^T \right] \quad (25)
$$
Fast computation of $k(x^*)$

**Theorem F.1.** Let $x_i$ denote the $i$-th row and $X(\bullet, j)$ the $j$-th column of $n \times d$ matrix $X$. Define for $d$-vector $x^*$ $n$-vector $k = k(x^*)$ such that its $i$-th component $k_i(x^*)$ for $1 \leq i \leq n$ is given by

$$k_i(x^*) = K(x_i, x^*) = \sum_{s=0}^{r} c_s (x_i, x^*)^s$$

Define for $1 \leq M_1, \ldots, M_s \leq d$ $n$-vector $b_{M_1, \ldots, M_s}$ to be the componentwise product of columns $M_1, \ldots, M_s$ of $X$.

$$b_{M_1, \ldots, M_s} = X_{\bullet, M_1} \cdots \cdots \cdots X_{\bullet, M_s}$$

Then we have the identity

$$k(x^*) = \sum_{s=0}^{r} c_s \left( \sum_{1 \leq M_1, \ldots, M_s \leq d} x^*_{M_1} \cdots x^*_{M_s} b_{M_1, \ldots, M_s} \right) \quad (26)$$

In addition, for any $n \times n$ matrix $A$

$$A^{-1}k(x^*) = \sum_{s=0}^{r} c_s \left( \sum_{1 \leq M_1, \ldots, M_s \leq d} x^*_{M_1} \cdots x^*_{M_s} A^{-1}b_{M_1, \ldots, M_s} \right) \quad (27)$$

Therefore once all the $O(d^r)$ vectors $A^{-1}b_{M_1, \ldots, M_s}$ are computed, space $O(d^r n)$ is required to store them; and each inversion $A^{-1}k(x^*)$ and each bilinear form $k(x^*)^T A^{-1}k(x^*)$ only requires time $O(d^r n)$ to calculate.

**Proof.** Because multiplication by $A^{-1}$ is linear, (27) follows if we can prove
Define \( k^{(s)}(x^*) \) so that its \( i \)th component is \( k^{(s)}_i(x^*) \overset{\text{def}}{=} \langle x_i, x^* \rangle^s \), then

\[
k(x^*) = \sum_{s=0}^{r} c_s k^{(s)}(x^*)
\]
\[
k_i^{(s)}(x^*) = \left( \sum_{1 \leq M \leq d} X_{i, M} x_M^* \right)^s
\]
\[
= \prod_{t=1}^{s} \left( \sum_{1 \leq M_t \leq d} X_{i, M_t} x_{M_t}^* \right)
\]
\[
= \sum_{1 \leq M_1, \ldots, M_s \leq d} X_{i, M_1} x_{M_1}^* \cdots X_{i, M_s} x_{M_s}^*
\]
\[
= \sum_{1 \leq M_1, \ldots, M_s \leq d} x_{M_1}^* \cdots x_{M_s}^* b_{M_1, \ldots, M_s}
\]

We conclude that for \( k^{(s)}(x^*) \) the \( n \)-vector whose \( i \)-th component for \( 1 \leq i \leq n \) is \( k^{(s)}_i(x^*) \)

\[
k^{(s)}(x^*) = \sum_{1 \leq M_1, \ldots, M_s \leq d} \left( x_{M_1}^* \cdots x_{M_s}^* \right) X_{\bullet, M_1} \ast \cdots \ast X_{\bullet, M_s}
\]
\[
= \sum_{1 \leq M_1, \ldots, M_s \leq d} x_{M_1}^* \cdots x_{M_s}^* b_{M_1, \ldots, M_s}
\]
G   Cholesky rank-one update

Theorem G.1. Suppose $\Lambda = \text{diag}(d_1, \ldots, d_n)$ is an $n \times n$ positive diagonal matrix and $\mathbf{p}$ is an $n$-vector. Then there exists positive diagonal matrix $\tilde{\Lambda} = \text{diag}(\tilde{d}_1, \ldots, \tilde{d}_n)$ and $n$-vector $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ such that for unit lower triangular $\tilde{L} = \tilde{L}(\mathbf{p}, \beta)$ we have the matrix identity

$$\Lambda + \mathbf{p}\mathbf{p}^T = \tilde{L}\tilde{\Lambda}\tilde{L}^T \quad (28)$$

Proof. The row $i$, column $j$ entry of $\Lambda + \mathbf{p}\mathbf{p}^T$ is

$$(\Lambda + \mathbf{p}\mathbf{p}^T)_{i,j} = d_i \delta_{i,j} + p_i p_j \quad (29)$$

For any $n$-vector $\beta$ and diagonal $\tilde{\Lambda}$ the $i, j$ entry of $\tilde{L}\tilde{\Lambda}\tilde{L}^T$ for $i \geq j$ is

$$(\tilde{L}\tilde{\Lambda}\tilde{L}^T)_{i,j} = \begin{cases} p_i p_i \left( \sum_{k=1}^{i-1} \tilde{d}_k \beta_k^2 \right) + \tilde{d}_i & \text{if } i = j \\ p_i p_j \left( \sum_{k=1}^{i-1} \tilde{d}_k \beta_k^2 \right) + p_i \beta_j \tilde{d}_j & \text{if } i > j \end{cases} \quad (30)$$

Therefore if for $i = 1, \ldots, n$ we set

$$\tilde{d}_i = d_i + p_i^2 \left( 1 - \sum_{k=1}^{i-1} \tilde{d}_k \beta_k^2 \right) \quad (31)$$

$$\beta_i = \frac{p_i}{\tilde{d}_i} \left( 1 - \sum_{k=1}^{i-1} \tilde{d}_k \beta_k^2 \right) \quad (32)$$

then we obtain equality of the left hand sides of equation (29) and equation (30). Let us denote

$$t_i = \begin{cases} 1 & \text{if } i = 0 \\ 1 / \left( 1 - \sum_{k=1}^{i} \tilde{d}_k \beta_k^2 \right) & \text{if } 1 \leq i \leq n \end{cases} \quad (33)$$

Then we claim

$$t_i = t_{i-1} + \frac{p_i^2}{\tilde{d}_i} \quad (34)$$

$$\beta_i = \frac{p_i}{\tilde{d}_i} \frac{1}{t_i} \quad (35)$$

$$\tilde{d}_i = \tilde{d}_i \frac{t_i}{t_{i-1}} \quad (36)$$
Rewriting equation (31) as

\[
\tilde{d}_i = d_i \left[ 1 + \frac{p_i^2}{d_i} \left( 1 - \sum_{k=1}^{i-1} \tilde{d}_k \beta_k^2 \right) \right]
\]

(37)

we see that equation (36) follows from equations (33), (34), and (37). And equation (35) follows from equations (32), (33), and (36). Therefore only equation (34) remains to be proven.

Define

\[
c_i = \frac{1}{t_i}
\]

\[
c_{i-1} = \frac{1}{t_{i-1}}
\]

Then

\[
c_i = 1 - \sum_{k=1}^{i} \tilde{d}_k \beta_k^2 = c_{i-1} - \tilde{d}_i \beta_i^2
\]

(38)

From equation (32) and equation (38) we obtain

\[
c_i = c_{i-1} - p_i \beta_i c_{i-1} = c_{i-1} (1 - p_i \beta_i)
\]

(39)

From equations (31) and (32) we obtain

\[
1 - p_i \beta_i = 1 - \frac{p_i^2}{d_i} c_{i-1}
\]

\[
= 1 - \frac{p_i^2 c_{i-1}}{d_i + p_i^2 c_{i-1}}
\]

\[
= \frac{d_i}{d_i + p_i^2 c_{i-1}}
\]

(40)

Since \(d_i > 0\) by assumption that \(\Lambda\) is a positive diagonal matrix, if \(c_{i-1} > 0\) then \(1 - p_i \beta_i > 0\), and thus \(c_i > 0\) by equation (39). Substituting equation (40) into equation (39) we obtain equation (34).
H Gibbs Sampler

Since covariance matrix $A$ is symmetric positive definite, $A^{-1}$ is also symmetric positive definite can be realized as the covariance matrix of a sample of random vectors $S = [s_1, s_2, s_3, \ldots, s_k]$ from some multivariate normal distribution. **Gibbs sampling** is a procedure to generate each entry of a new random vector from a univariate normal distribution with mean and standard deviation computed from a previous random vector and the matrix $A$. Then the solution $w$ of $Aw = y$ can be estimated using

$$w = A^{-1}y \approx \frac{1}{k} S(S^T y)$$

It is unknown how large the sample should be to give a good approximation of $w$. If $A$ is well-conditioned, then the convergence is known to be reasonable.

As suggested by the results of Guo and Thompson, Gibbs sampling works best in situations where large standard error is tolerated, when diagonal entries of the matrix in question are of particular interest, and when more exact methods that may be too slow for use in larger matrices.

After this sample of random vectors is created they can be denoted by a matrix $S$, and the covariance matrix of $S$, which can be calculated by $\frac{SS^T}{k}$ will be our inverse of $B$. A computer algorithm created by Geman and Geman serves to estimate an inverse of a positive definite matrix by use of the Gibbs Sampler. Our study entails taking a matrix denoted by $X$ filled with photometric data and a vector denoted by $y$ filled with values of redshift data in order to solve the matrix equation $Aw = y$ in terms of $w$ such that $A = I(\lambda)^2 + QQ^T$, where $I$ is the $n$ by $n$ identity matrix, $QQ^T$, is from the kernel function of our system, and $\lambda$ is the noise level. The algorithm generates $A$ using codes that simulate the equation for $A$ seen above. It starts by creating $Q$, which is a series of dot products of the rows of $X$ with the row transposes of $X$. After $Q$ is created, we next create one column of $A$, which is denoted as $v_i$. Going by the equation seen above, the element of $A$ resting in the $i$th row and the $j$th column is the dot product of the $i$th row of $Q$ with the $j$th row of $Q$ transpose, but if the element of $A$ in question lies on the diagonal, we also add $\lambda^2$. After $v_i$ is filled up, we are now ready to randomly estimate one of the elements of $S$, but first there is the matter of determining the mean and standard deviation of our prediction. The formula
used in Geman and Geman’s algorithm was denoted by

\[ m = -v_{ii}^{-1}\left(\sum_{j=1}^{i-1} v_{ij}s_{j}^{k} + \sum_{j=i+1}^{n} v_{ij}s_{j}^{k-1}\right) \]

(Harville, 206). In this formula, \( n \) is the number of rows (or columns) in \( A \), \( v_{ij} \) represents the element in the \( i \)th row and \( j \)th column of \( A \), and \( s_{j}^{k} \) is the \( j \)th element in the \( k \)th column of \( S \). This formula for the mean is asking us to take the negative reciprocal of the \( i \)th diagonal entry of \( A \) \( (v_{ii}) \), and multiply it by a dot product consisting of elements of \( A \) and the already known elements of \( S \). The dot product can be expressed as taking the dot product of \( v_{i} \) with \( s_{i} \), where \( s_{i} \) is a vector whose first \( i-1 \) elements are taken from the \( i-1 \) elements of \( S \)’s \( k \)th column that have already been determined, whose \( i \)th element is zero, and whose final \( n-i \) elements are taken from the final \( n-i \) elements in the column of \( S \) that came before it. The standard deviation is then calculated by \( \sqrt{\frac{1}{v_{ii}}\text{Tr}Ac1v_{ii}} \) (Harville, 206), which is simply the square root of the reciprocal \( i \)th diagonal entry of \( A \). We will never have to worry about any values on the diagonal being negative however, since our definition of \( A \) forbids negative values. With scalar values for the mean and standard deviation calculated, we then randomly predict what the next value of \( S \) should be. Most computer programs provide a command for a pseudo-random number generator that will churn out values at a standard normal distribution (an average value of 0, and a standard deviation of 1). To condition this command for our situation, we merely scale the random value by the standard deviation and then shift the random value by the mean. This new value goes into the \( i \)th row and \( k \)th column of \( S \), and the process is repeated until the \( k \)th column of \( S \) is filled up before moving on to the next random vector which will be the next column of \( S \). We can see from the calculations that no more than 1 column of \( A \) is needed at any given time to determine a particular value of \( S \), so the code we use does not require us to completely compute \( A \) first (or even store what \( A \) needs to be) before using it’s values to create \( S \), which aids us to making the calculations take less time. However, we must also notice that new values of \( S \) are dependent on the values that came before it, and since no values come before the first values, we create must make the first column of \( S \) nothing but zeros to run the code without skewing the results in any way. After the \( S \) matrix is completely determined, we have the option of possibly making our prediction of the inverse of \( A \) more accurate by eliminating certain values from \( S \). The fact that each new random
vector depends on the vector that came before it, and the general principle behind the convergence of random values over indefinite repetition suggests that the random vectors within S that came further down the line are likely to be more converged about an expected value than the earlier vectors. This gives us the helpful (but not always necessary) opportunity to consider only a portion of S rather than all of S in our inverse prediction. Whether we use S or a portion of S which we can denote by S’, we calculate our predicted inverse by \( \frac{SS^T}{k} \), where k is the number of random vectors we generated (or included in our portion). Our code however, goes a step further by not only predicting the inverse of A, but also by outputting w in our equation earlier, or \( A^{-1}y \). Thus, our code is completed, but unfortunately, our code has costly drawbacks and limitations. The Gibbs sampler’s predictions are known to be accurate as long as the matrix A in question is well-conditioned, but the guidelines for a well-conditioned matrix are not entirely met for our experiments. Whether A is a reasonable matrix for the Gibbs sampler is determined by a condition number which is the quotient of the largest eigenvalue of A divided by the smallest eigenvalue of A. Looking back at our equation of A, we recall that \( A = I(\lambda)^2 + QQ^T \), so finding the eigenvalues of A is equal to \( eig(A) = eig(I(\lambda)^2 + QQ^T) = eig(I(\lambda)^2) + eig(QQ^T) \). The eigenvalues of \( I(\lambda)^2 \) are all simply \( \lambda^2 \), and the eigenvalues of are positive and zero, so if we denote the largest eigenvalue of \( QQ^T \) by z, we find that for matrix A, the condition number is approximately \( \frac{\lambda^2 + z}{\lambda^2} = 1 + \frac{z}{\lambda^2} \). A well-conditioned matrix for the Gibbs sampler is one that is approximately one, and the code works reasonably well for matrices that have a condition number as high as 100, however, given our data set, this poses many problems. Our photometric data set is very large with many potential eigenvalues, and the value of the noise level that was agreed upon was 0.1. The large set opens the opportunity for A to have very large eigenvalues, which when divided by \( \lambda^2 \) (or .01) will cause the condition number for A to be very large. We saw earlier that the calculations for both the mean and standard deviation of the values of S were dependent on the diagonal entries of A, which means that if A is a matrix with particularly low eigenvalues, the entries of S are in danger of inflating excessively to high values which can cause great inaccuracy in our prediction. The Gibbs sampler can work very well on relatively small matrices with high eigenvalues that vary little, but even given the properties of randomization, large matrices with small eigenvalues are in danger of having inflated predictions that aren’t accurate. These are of great concern to us, and in the end they are the main reason behind why the Gibbs Sampler is not
a preferable or even valuable method of matrix inversion that we are looking for. In the end we are left with conclusions that leave us with knowledge, but not results. The Gibbs sampler is an alternate method of inverting positive definite matrices and under the proper conditions sacrifices a small bit of accuracy for speed, but does work. In the case of our study of redshifts however, it ends up being a poor method due to its strong conditions that work against our situations. Under better circumstances, this may have been a useful alternative to the Gaussian process, but unfortunately, it’s not meant to be.
I Source code

I.1 Data structures

We use the following conventions throughout our programs for associating variable names with data structures.

\( r \) degree of the special polynomial kernel. The implemented programs only work for \( r = 1, 2 \).

\( \mathbf{X} \) an \( n \times d \) matrix that represents photometric filter observations; therefore, \( d = 5 \). We refer to the columns of \( \mathbf{X} \) in order as \( U, G, R, I, \) and \( Z \).

\( \mathbf{\theta} \) vector of hyperparameters. \( \mathbf{\theta}(1) \) is always the noise magnitude \( \lambda^2 \). For both \( r = 1, 2 \), \( \mathbf{\theta}(2 : 6) \) are the scaling factors for columns \( U, G, R, I, \) and \( Z \), respectively. For \( r = 2 \) there are 15 additional scaling factors \( \mathbf{\theta}(7 : 22) \) that correspond to columns \( UU, \sqrt{2}UG, \sqrt{2}UR, \sqrt{2}UI, \sqrt{2}UZ, GG, \sqrt{2}GR, \sqrt{2}GI, \sqrt{2}GZ, RR, \sqrt{2}RI, \sqrt{2}RZ, II, \sqrt{2}IZ, \) and \( ZZ \).

\( \mathbf{Q} \) The \( n \times m \) matrix such that for covariance matrix \( A \), \( A = \lambda^2 + \mathbf{QQ}^T \). For \( r = 1 \) we have \( m = 6 \), while for \( r = 2 \) we have \( m = 21 \).

I.2 Source code listing

In Figure 7 we show a version of a Matlab function \texttt{genq} to generate matrix \( \mathbf{Q} \) given polynomial degree \( r \) and hyperparameter vector \( \mathbf{\Theta} \). Function \texttt{genq} actually just multiplies \( \mathbf{Q}' \) by the diagonal matrix produced from \( \mathbf{\Theta} \). Matrix \( \mathbf{Q}' \) is created using Matlab function \texttt{genqprime} shown in Figure 8.

Figure 9 is our Matlab implementation of reduced rank with one round of iterative refinement and corresponds to Algorithm 1 of Section 5.

Figures 10, 11, 12, and 13 show our Matlab implementation of Choleksy update and correspond respectively to Algorithms 2, 3, 4, and 5 of Section 6.

Figure 15 shows our Matlab implementation of conjugate gradient and corresponds to Algorithm 6 of Section 7.
function Q = genq(X, r, theta)
    R = genqprime(X, r);
    Q = R * diag(sqrt(max(theta(2: length(theta)), 0)));

    Figure 7: genq.m

function Q = genqprime(X, r)
    [n, d] = size(X);
    Z = [ones(n, 1) X];
    h = d + 1;
    y = [];
    if r == 1
        Q = Z;
    elseif r == 2
        for i = 1:h
            for j = i:h
                y = [y; i j];
            end
        end
        Q = [];
    for t = 1: size(y, 1)
        if y(t, 1) == 1 % Either column of 1’s or original column
            Q = [Q Z(:, y(t, 1)) .* Z(:, y(t, 2))];
        elseif y(t, 1) == y(t, 2) % Column is squared by itself
            Q = [Q Z(:, y(t, 1)) .* Z(:, y(t, 2))];
        else % Product of two different columns must account for duplicate
            Q = [Q sqrt(2) * Z(:, y(t, 1)) .* Z(:, y(t, 2))];
        end
    end
end

    Figure 8: genqprime.m

function w = ReducedRank(Q, lambdasq, y)
    n = size(Q, 1);
    k = size(Q, 2);
    temp = lambdasq * eye(k) + Q’ * Q;
    w = (1/lambdasq) * (y - Q * (temp \ (Q’ * y)));

    Figure 9: ReducedRank.m
function \( w = \text{forwardsolve}(p, b, y, n) \)
\[
\begin{align*}
    w &= \text{zeros}(n,1); \quad \text{define } w, \text{ set to all } 0s \\
    w(1) &= y(1); \quad \text{set initial } w(1) \\
    c &= 0; \quad \text{initialize } c \\
    \text{for } i = 2:n \\
    &\quad c = c + b(i - 1) \times w(i - 1); \quad \text{updates } c \\
    &\quad w(i) = y(i) - p(i) \times c; \quad \text{calculates } w(i) \\
    \text{end}
\end{align*}
\]

Figure 10: forwardsolve.m

function \( w = \text{backwardsolve}(p, b, y, n) \)
\[
\begin{align*}
    w &= \text{zeros}(n, 1); \quad \text{defines } w, \text{ sets to all } 0s \\
    w(n) &= y(n); \quad \text{sets first } w(n) \\
    c &= 0; \quad \text{initialize } c=0 \\
    \text{for } i = (n-1):-1:1 \quad \text{loop goes backwards} \\
    &\quad c = c + w(i + 1) \times p(i + 1); \quad \text{updates } c \\
    &\quad w(i) = y(i) - b(i) \times c; \quad \text{calculates } w(i) \\
    \text{end}
\end{align*}
\]

Figure 11: backwardsolve.m
function p = getp(pmatrix, k, betas, Q)

n = size(Q,1);
u = zeros(n,1);
rhs = zeros(n,1);

for j = 1:k-1
    if j == 1
        rhs(:) = Q(:,k);
    else
        rhs = u;
    end
    p = pmatrix(:, j);
    b = betas(:, j);
    u = forwardsolve(p, b, rhs, n);
end
p = u;

Figure 12: getp.m

function [b,d] = getbeta(p, dold)

n = size(p, 1);
b = zeros(n, 1);
d = zeros(n, 1);
told = 1;
tnew = told;
for j = 1:n
    told = tnew;
    tnew = told + p(j)^2 / dold(j);
    b(j) = p(j) / (dold(j) * tnew);
    d(j) = dold(j) * tnew / told;
end

Figure 13: getbeta.m
function w = CU(Q, lambdasq, y)
n = size(Q,1);
dold=zeros(n,1); %initialize vector
for j=1:n
    dold(j)=lambdasq; %dold nx1 vector is supplied values
end
p=Q(:,1);
[b,d]=getbeta(p,dold);
dold=d; %update dold
pmatrix(:,1)=p(:,);
betas(:,1)=b(:,);
for i=2:size(Q,2)
    p=getp(pmatrix,i,betas,Q);
    pmatrix(:,i)=p(:,); %each p_i stored in the pmatrix matrix
    [b,d]=getbeta(p,dold);
    betas(:,i)=b(:,); %each b_i stored in the betas matrix
    dold=d; %dold updated
end
for i=1:size(Q,2)
    p=pmatrix(:,i);
    b=betas(:,i);
    w=forwardsolve(p, b, y,n);
    y=w; %after each run the w is updated
end
y=y./d;
for i=size(Q,2):-1:1 %loop goes backwards
    p=pmatrix(:,i);
    b=betas(:,i);
    w=backwardsolve(p, b, y, n);
    y=w;
end

Figure 14: CU.m
function w = CG(Q, lambdasq, y, eps, winit)
n = size(Q,1);
if nargin < 4
    eps = 10^(-10);
end
if nargin < 5
    winit = zeros(n, 1);
end
w = winit;
r = y - ((lambdasq * w) + Q * (Q' * w));
p = r;

niter = 0;
for iteration = 1:n
    Ap = ((lambdasq * p) + Q * (Q' * p));
    alpha = dot(r, p) / dot(Ap, p);
    rnext = r - alpha * Ap;
    beta = dot(rnext, rnext) / dot(r, r);
    w = w + alpha * p;
    r = rnext;
    p = rnext + beta * p;
    if norm(r, 2)< eps * norm(y, 2)
        niter = iteration;
        break
    end
end

Figure 15: CG.m
References


