Improved Linear Algebra Methods for Redshift Computation from Limited Spectrum Data – II

Faculty Advisor: Leslie Foster (foster@math.sjsu.edu)

Student Team: Alex Waagen (team leader), Nabeela Aijaz, Michael Hurley, Apolo Luis, Joel Rinsky, Chandrika Satyavolu

NASA Ames Research Center Advisors: Paul Gazis, Ashok Srivastava and Michael Way

CAMCOS Project (www.math.sjsu.edu/camcos)
Mathematics Department
San Jose State University
July 6, 2007

We are grateful to the Woodward fund for partial support of this project.

Contents

1 Redshift and photometric observations **4**
1.1 What is a Redshift? 4

2 Gaussian Processes **6**

3 Gibbs Sampler **6**

4 Covariance Functions and Kernels **9**

5 Low Rank Approximation **11**
5.1 Singular Value Decomposition 11
5.2 Partial Cholesky Decomposition 11
5.3 Partial Cholesky Decomposition With Pivoting 12
5.4 Algorithm: Partial Cholesky Decomposition with Pivoting 13
5.5 Pivoting 13

6 V Formulation **14**
6.1 V Formulation without QR factorization 14
6.2 QR Factorization in V Formulation 15

7 Subset of Regressors **16**
7.1 Subset of Regressor without QR Factorization 16
7.2 QR Factorization in Subset of Regressors 17

8 Numerical Stability **18**
8.1 Squaring the condition number 18
8.2 Pivoting 20
8.3 Related methods 22

9 Choice of Rank **22**

10 Optimization of Hyperparameters **22**
10.1 Nelder-Mead Simplex Minimization 22
10.2 Marginal Likelihood and Minimize 23
10.3 Optimization Method 23

11 Timings **26**

12 Accuracy and Stability **29**
12.1 Bootstrap resampling results 29
12.2 The choice of rank 35
12.3 The effect of pivoting 36
12.4 The choice of the testing set size 36
12.5 Comparison with Traditional Method 37
12.6 Comparison with alternate approaches 38

13 Outliers **39**
13.1 Characterizing Outliers 39
13.2 Classifying Outliers 43

14 Appendix - Code **44**

Abstract

Given photometric broadband measurements of a galaxy, Gaussian processes may be used with a training set to solve the regression problem of approximating the redshift of this galaxy. However, in practice solving the traditional Gaussian processes equation is too slow and requires too much memory. We employed several methods to avoid this difficulty using algebraic manipulation and low-rank approximation, and were able to quickly approximate the redshifts in our testing data within 17 percent of the known true values using limited computational resources. The accuracy of one method, the V Formulation, is comparable to the accuracy of the best methods currently used for this problem. However, the V Formulation is quicker, and its numerical stability is superior.

If 10 percent of outliers are removed from the testing data, the error is reduced from 17 percent to 12 percent. A necessary condition was found for identifying these outliers, that the norm of a collection of redshifts corresponding to these outliers is comparatively large when compared to the norm of a randomly selected group of redshifts. Moreover, viewing characterization of outliers as a classification problem, it was possible to reduce the error to approximately 15 percent.

1 Redshift and photometric observations

Let $y = (y_1, ..., y_n)^T$ be a sequence of redshift observations of n galaxies. In Way and Srivastava's [Way and Srivastava, 2006] observation set which is based on the SLOAN sky survey, for each galaxy i there are five photometric observations denoted as U, G, R, I, Z. We use X to denote an $n \times 5$ matrix such that the ith 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.

1.1 What is a Redshift?

The redshift of a galaxy, z, is the change in the wavelength divided by the initial wavelength of the electromagnetic waves that are emitted by the galaxy. Hence, the redshift of z is $\frac{d\lambda}{\lambda_0}$. 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.

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 infared. 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, and because it collects more data at one time, it was the preferred method used by Dr. Way and Dr. Srivastava in their research [Way and Srivastava, 2006], upon which the research in this paper is based.

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 magnitude, color, and red shift. These samples have not always been available because the telescopes could not reach as far into space. However, now that the surveys can go deeper into space, the TS method can be used because there are more samples of galaxies with the same characteristics. Dr. Way and Dr. Srivastava described various TS methods that 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 red-shift 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 Guassian processes to find the best model for the red shifts. The Guassian process regression was the focus of our problem.

2 Gaussian Processes

The set of $180,045$ measurements through the 5 broadband spectrums (UGRIZ) are given by a $180,0455 \times 5$ matrix X , and the corresponding redshift measurements are stored in a a $180,0455 \times 1$ vector y. Together, these form the training dataset. A second set of $20,229$ more measurements X^* along with $20,229$ corresponding redshifts stored in y^* together form the testing dataset. This second set was the testing data, and in practice would be a set of broadband spectrum values whose redshifts were unknown. For the purposes of our research the measured redshifts y^* were known. The goal is to predict the value of y^* given X, y, and X^* using Gaussian processes.

The prediction of y^* involved both covariances between rows from X and itself, forming a covariance matrix $K(x_i, x_j)$ and between rows from X and X^*, forming a covariance matrix $K^*(x_i^*, x_j)$. The prediction $\hat{y}^*$ of y^* is given by the traditional Gaussian processes equation, shown below:

$$\hat{y}^* = K^*(\lambda^2 I + K)^{-1} y$$

It is not completely clear how to choose the kernel function K , and there exist many different kernel functions that apply broadly to many cased. Last semester, the polynomial kernel was used, a function that involved matrix multiplication of X and X^* to generate K. Namely, the kernel took the form $K = (\lambda^2 I + XX^T).\hat{}2$, where the symbol $.\hat{}$ here indicates componentwise square (so that if $B = (\lambda^2 I + XX^T)$ then $k_{ij} = bij^2$). Since X is a $180,045 \times 5$ matrix, K is a $180,045 \times 180,045$ matrix. This semester, the study went further by experimenting with many other kernels. Last semester four methods that were used and perfected were the reduced rank method, the Cholesky update, the conjugate gradient, and the classic method, quadratic regression. We refer the reader to last semester's report for detail of these methods [Cayco, So, et. al., 2006]

3 Gibbs Sampler

The Gibbs Sampler is another method of solving a system of linear equations. The Gibbs Sampler is based off of the Monte Carlo idea, a process of finding the expected value of a population by creating a random sample of the population. Given that this random sample is representative of the population, the Monte Carlo idea states that the average of this sample is the expected value of the population. The best way to ensure that this sample is representative is to take a large sample. In the case of redshifts however, this process is translated to vectors and matrices. The population in this case is the covariance matrix A, and the random sample will be a set of vectors.

This set of random vectors will form a matrix S whose covariance matrix will ideally be A^{-1}. The Gibbs Sampler works best in situations where large tolerance for error is allowed and standard procedures are too slow. Other factors can come into play however, one of which is the condition ratio. The condition ratio of A is the largest eigenvalue of A divided by the smallest, and the smaller the condition ratio, the more accurate the Gibbs Sampler will be. It was seen last semester that by generating A by a polynomial kernel, the condition ratio was too large for the Gibbs Sampler to work. So instead this semester the squared-exponential kernel was used for the majority of experimentation of the Gibbs Sampler to better success.

The Gibbs Sampler code used in this project was a computer algorithm put forth by [Geman and Geman, 1984]. Given A, the elements of S must be randomly selected such that they accurately represent A. This is done with a series of dot products and matrix multiplication. S is denoted by a series of vectors s_i, the more vectors, the better. These vectors are generated one entry at a time and are dependent both on A, whose columns are denoted as v_i, and the vectors of S that have already been created. To ensure that the vectors are representative of A, the mean and standard deviation of the entry that is being created are computed first. The algorithm put forth by Geman and Geman was that the entry of S that lies in the kth column and ith row of S has a mean of:

$$m = \sum_{j=1}^{i+1} v_{ij} s_{j^k} + \sum_{j=i+1}^{n} v_{ij} s_{j^{k-1}}$$

The mean is found by taking the semi-dot product of the first i-1 entries of the jth column of A with the entries in the kth column of S that have already been created, as well as the last n-(i+1) entries of the jth column of A with the entries in the column of S previous to the kth column. The entire product is then multiplied by the negative reciprocal of the ith diagonal entry of A. Thus, this entire process doesn't require that A be completely computed beforehand either. However, it should be noted that each new vector of S depends on the vector that came before it, thus to avoid confusion on generating the first vector of S, a column of zeroes is automatically made the first column of S. The standard deviation is simply computed as:

$$t = \frac{1}{\sqrt{v_{ii}}}$$

After the mean and standard deviation are computed, the entry in the ith row and kth column of S is computed to be $s_{ik} = tz + m$, where z is a random number created by a pseudo-random number generator at standard normal distribution, as most computer systems have such a generator. The process is repeated until all the vectors (and columns of S) have been generated, and the covariance of S is computed to be $(SS\hat{}T)/k$ where k is the number of vectors that were generated to form S. For the various different kernels that were sampled, the one that was used for the majority of the study of the Gibbs Sampler was the squared-exponential kernel. However, despite the improvements on the Gibbs Sampler from last year's studies, obstacles still stood in the way.

The biggest obstacle was that of time. Time was one of the primary reasons why alternate methods to quadratic regression were sought after, but the Gibbs Sampler in this form was not a time-efficient code. Due to the complicated ordering factors involved in the production of the mean values, the MATLAB code that was used required looping codes with looping codes (creating double loops). Therefore, these loops caused the running time for the Gibbs Sampler to be very large, especially for larger data sets. Alternate methods were eventually devised to decrease running times, like Fortran codes with MATLAB, but in the end the Gibbs Sampler was still simply not as fast as the other methods devised over this semester. Another factor was that of accuracy, even after careful reconstruction through the Fortran code, the accuracy of the Gibbs Sampler was only about half as good as that of the other methods examined both this semester and last semester. Perhaps the biggest reason behind why the Gibbs Sampler was never fully realized was that the code was one of several different algorithms that have been proposed for the Gibbs Sampler, some of which are much more advanced, complex, and less prone to errors than the one used in these experiments. The RMS values for the Gibbs Sampler would tend to converge to values twice that of the values in other methods used, so while the Gibbs Sampler could work, it wasn't efficient in relation to the other methods. Further study in the Gibbs Sampler would involve finding a quicker or more complex algorithm that would allow for more accurate results without sacrificing time efficiency.

4 Covariance Functions and Kernels

According to [Rasmussen and Williams, 2006] (for much of this section we follow the development in this source) a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. To characterize a Gaussian process it is critical to specify its covariance function. A covariance function is a function $k(x, x')$ of two variables, sometimes called a kernel, with the property

$$\int k(x,x')f(x)f(x')d\mu(x)d\mu(x' \geq 0$$

for a suitable function $f(x)$ and measure μ. Covariance functions characterize the nearness or similarity between input data points. A covariance function $k(x, x')$ gives the covariance of the values of the random field at the two locations x and x'.

Let $\{x_i | i = 1, ..., n\}$ be a set of input points. Then the Gram matrix or covariance matrix is defined as K whose entries are $K_{ij} = k(x_i, x_j)$. A Gram matrix corresponding to a general kernel function need not be positive semi-definite but the Gram matrix corresponding to the covariance function is positive semi-definite.

We now summarize some of the features of commonly used covariance functions:

The **Squared Exponential (SE)** covariance function is defined by the following equation:

$$k_{SE}(r) = \exp(-\frac{r^2}{2\ell^2})$$

where ℓ is the length scale. This covariance function is infinitely differentiable and hence is very smooth. With such strong smoothness, it is sometimes unrealistic for use in modelling real physical processes.

The **Matern class** covariance function is defined by the following equation:

$$k_{Matern}(r) = \frac{2^{\ell-v}}{\Gamma(v)}(\frac{\sqrt{2vr}}{\ell})^v \ K_v(\frac{\sqrt{2vr}}{\ell})$$

where v, ℓ are positive parameters and K_v is a modified Bessel function. The **Matern class** covariance function reduces to the Squared Exponential covariance function as $v \to \infty$. It becomes simpler when v is a half integer. For example, if $v = p + \frac{1}{2}$ (where p is a non-negative integer), it reduces to a covariance function that is a sum of an exponential and a polynomial function:

$$k_{Matern\left(v=p+\frac{1}{2}\right)}(r) = \exp\left(-\frac{2vr}{l}\right)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\left(\frac{\sqrt{8vr}}{l}\right)^{p-i}$$

The process becomes very rough for $v = \frac{1}{2}$ and for values of $v \geq \frac{7}{2}$, the function is as rough as noise. The Matern class covariance function is mean square differentiable k times if and only if $v > k$. Hence, the Matern class of covariance functions can be used to model real physical processes and is more realistic than the Squared Exponential (SE) covariance function.

The **Rational Quadratic (RQ)** covariance function is defined by the following equation:

$$k_{RQ}(r) = \left(1 + \frac{r^2}{2\alpha l^2}\right)^{-\alpha}$$

The Rational Quadratic (RQ) covariance function reduces to a Squared Exponential (SE) covariance function as $\alpha \to \infty$. This function is mean square differentiable for every α as opposed to Matern class covariance function.

The **polynomial** covariance function, in its most general form, can be defined by the equation:

$$k(x, x') = (\sigma_0^2 + x^T \Sigma_p x')^p$$

where Σ_p is a positive semidefinite matrix and p is a positive integer. If $\sigma_0^2 = 0$, the kernel is homogeneous and linear, and otherwise it is inhomogeneous. In principle it may not be suitable for regression problems as the variance grows with $\mid x \mid$ for $\mid x \mid > 1$. However there are applications where it has turned out to be effective [Rasmussen and Williams, 2006].

The **Neural Network (NN)** covariance function is defined by the following equation:

$$k_{NN}(x, x') = \frac{2}{\pi}\sin^{-1}\left(\frac{2x^T\Sigma x'}{\sqrt{(1+2x^T\Sigma x)(1+2x'^T\Sigma x')}}\right)$$

This covariance function is named after Neural networks because the function can be derived from the limiting case of a model of a neural network.

Two or more covariance functions can be combined to produce a new covariance function. For example sums, products, convolutions, tensor products and other combinations of covariance functions can be used to form new covariance functions. Details are described in [Rasmussen and William, 2006].

5 Low Rank Approximation

Definition: A positive semi-definite matrix $\hat{K}$ is a low rank approximation of covariance matrix K if $\left\| \hat{y}^*_K - \hat{y}^*_{\hat{K}} \right\|$ is small, where $\hat{y}^*_X$ is the approximation $\hat{y}^*$ calculated with covariance matrix X.

It is difficult to calculate a low rank approximation of K using this definition directly. However, if $\hat{K}$ is sufficiently numerically stable and $\left\| K - \hat{K} \right\|$ is small, then $\hat{K}$ is a low rank approximation of K. Singular value decomposition and partial Cholesky decomposition may be used to find such a $\hat{K}$.

5.1 Singular Value Decomposition

Covariance matrices are n x n, which for large n makes them impossible to calculate in a reasonable amount of time or store in memory. A possible solution is to instead calculate low rank approximations of these matrices. There is a well-known bound on how good these approximations can be. If $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$ are the eigenvalues of any matrix K, and $\hat{K}$ is any rank m matrix, then $\left\| K - \hat{K} \right\| \geq \alpha_{m+1}$. The singular value decomposition of K can be used to find a $\hat{K}$ such that $\left\| K - \hat{K} \right\| = \alpha_{m+1}$, proving $\hat{K}$ to be the be the best low rank approximation of K [Golub and Van Loan, 1996, p. 72]. Unfortunately, singular value decomposition is expensive in terms of time and memory, requiring $O(n^3)$ operations, and so it is not feasible with large matrices. Therefore, a more economic method is required.

5.2 Partial Cholesky Decomposition

The partial Cholesky decomposition is a generalization of the Cholesky decomposition which may be calculated for a positive semi-definite $n \times n$ matrix K. For any $m < n$, there exists an $n \times m$ matrix V such that $\hat{K} = VV^T$ is a rank m partial Cholesky decomposition of K, and it follows from this that $\hat{K}$ is a low rank approximation of K. Moreover, this decomposition is not unique – many different possible choices of V exist. Rather than lower triangular as in the Cholesky decomposition, V is "lower trapezoidal" as illustrated below.

$$\hat{K} = VV^T =$$

$$\begin{bmatrix} v_{11} & 0 & \cdots & 0 \\ v_{21} & v_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ v_{(m-1)1} & v_{(m-1)2} & \cdots & 0 \\ v_{m1} & v_{m2} & \cdots & v_{mn} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nn} \end{bmatrix} \begin{bmatrix} v_{11} & v_{21} & \cdots & v_{(m-1)1} & v_m & \cdots & v_n \\ 0 & v_{22} & \cdots & v_{(m-1)2} & v_{2m} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & v_{mn} & \cdots & v_{nn} \end{bmatrix}$$

5.3 Partial Cholesky Decomposition With Pivoting

Partial Cholesky decomposition with pivoting is an algorithm which is based on an algorithm to calculate the Cholesky decomposition, simply halted after m steps. It may be used to calculate a unique rank m partial Cholesky decomposition of K in $O(nm^2)$ operations. Pivoting need not be used, but without pivoting the decomposition is not unique and there is not clear bound on the inaccuracy of the approximation. If pivoting is used, then$\left\|K - VV^T\right\|$ is bounded by $c_1 = \sqrt{\frac{1}{3}(n-m)(4^m+1)}\alpha_{m+1}$. [Higham, 2002, p. 204] In practice, this bound is not necessarily a good indication of the accuracy of the approximation, and usually $\left\|K - VV^T\right\|$ is well under $10\alpha_{m+1}$ [Higham, 2002, p. 207]. The algorithm for calculating the partial Cholesky decomposition with pivoting is given below.

5.4 Algorithm: Partial Cholesky Decomposition with Pivoting

Given a positive semi-definite K, this algorithm [Higham, 2002, p. 202] computes the partial Cholesky decomposition with pivoting algorithm. If there are no ties in the choice of pivot elements the algorithm produces a unique decomposition. At each step let *piv* be a vector such that $\bar{K}(piv, piv)$ is a matrix in which its first $m-1$ diagonal entries equal 0, and its mth diagonal entry is $\max(diag(\bar{K}))$. Note the the output of this algorithm is not $\bar{K}$, but $V = (v_{ij})$.

```
$\bar{K} = K$
$for\ j = 1 : m$
    $\bar{K} = \bar{K}(piv, piv)$
    $for\ i = 1 : j-1$
        $v_{ij} = \frac{1}{v_{ii}} \left( k_{ij} - \sum_{k=1}^{i-1} v_{ki} v_{kj} \right)$
    $end$
$v_{jj} = \sqrt{\left( k_{jj} - \sum_{k=1}^{i-1} v_{kj} \right)}$
$\bar{K} = \bar{K} - v_m v_m^T$
$end$
```

The line $\bar{K} = \bar{K}(piv, piv)$ is the pivoting step, and without this step the above algorithm would be the Partial Cholesky Decomposition without Pivoting.

5.5 Pivoting

Pivoting is not only useful for forming a more accurate, numerically stable low rank approximation of a positive semi-definite matrix, but also to identify the rows of the training data which are the "best" in some way. A pivot of the matrix K, which is simply a permutation of K of the form PKP^T corresponds to the permutation PX of X . The above partial Cholesky algorithm with pivoting will move columns and rows of K so that the m by m leading principal submatrix of PKP^T has the condition number that is with a constant c_2 of α_1/α_m. The proof follows from arguments similar to those in [Higham, 2002, pp. 195-208]. The constant c_2 is related to the constant c_1 above.

Thus pivoting will tend to construct a low rank approximation whose condition number is related to the condition number (α_1/α_m) of the low rank approximation produced by the singular value decomposition. Since pivoting limits the condition number, it will tend to improve the numerical stability of the algorithm and reduce the effect of computer arithmetic errors.

6 V Formulation

6.1 V Formulation without QR factorization

Previously, to solve the original equation to predict the redshift, we used the Gaussian approach. This required forming an $n \times n$ covariance matrix K by using some pre-defined formula $K_{ij} = k(x_i, x_j)$.

If ${x_i}^*$ is taken to be the ith row of X^*, then a $p \times n$ matrix K^* can then be formed by the following formula:

$$K_{ij}^* = k(x_i^*, x_j)$$

Substituting these K and K^* in the original linear equation:

$$\hat{y}^* = K^*(\lambda^2 I + K)^{-1} y$$

But the size of $(\lambda^2 I + K)^{-1}$is $n \times n$, and for large n (in our case $n = 180,045$), it is not practical to calculate $(\lambda^2 I + K)^{-1}$ directly. Inverting a matrix takes $O(n^3)$ floating point operations. When n is large, an $O(n^3)$ operation quickly becomes intractable. This problem may be by passed using simple linear algebra and the low rank approximations detailed in the previous section.

To do this we will approximate K with VV^T where V is produced by the partial Cholesky factorization. If pivoting was used in constructing V we will assume, for simplicity of presentation, that K is the covariance matrix after its rows and columns have been moved. Now K^* is a p by n matrix. Let K_1^* be the first m columns of K^* and let V_{11} be the $m \times m$ matrix consisting of the first m rows of V. Also let $V^* = K_1^* V_{11}^{-T}$. Now in addition to approximating K with VV^T we also approximate K^* with $V^* V^T$. Using these approximations we see that $K^*(\lambda^2 I + K)^{-1} y$ can be approximated by $V^* V^T (\lambda^2 I + VV^T)^{-1} y$.

We now have the following key Lemma:

Lemma: $V^T(\lambda^2 I + VV^T)^{-1} = (\lambda^2 I + V^T V)^{-1} V^T$

Proof

$$\begin{gathered}
\lambda^2 V^T + V^T V V^T = V^T V V^T + \lambda^2 V^T \\
= (\lambda^2 I + V^T V) V^T = V^T (V V^T + \lambda^2 I) \\
= (\lambda^2 I + V^T V)^{-1} (\lambda^2 I + V^T V) V^T (\lambda^2 I + V V^T)^{-1} \\
= (\lambda^2 I +^T V^T V)^{-1} V^T (V V^T + \lambda^2 I)(\lambda^2 I + V V^T)^{-1} \\
\to V^T (\lambda^2 I + V V^T)^{=1} = (\lambda^2 I + V^T V)^{-1} V^T \\
\blacksquare
\end{gathered}$$

From this lemma, it follows that by substitution that:

$$\text{Corollary: } \hat{y}^* = V^*(\lambda^2 I + V^T V)^{-1} V^T y$$

This corollary is the basis of the V Formulation method. The matrix $(\lambda^2 I + V^T V)^{-1}$ is $m \times m$ rather than $n \times n$, and so for small enough m, the equation is tractable, and in fact can be solved quite quickly. The flop count for solving this new equation is $O(nm^2)$.

6.2 QR Factorization in V Formulation

The condition number of $V^T V$ is the square of the condition number of V, and so if V has a condition number that is sufficient large, then there is a potential to introduce numerical instability in the V Formulation. Therefore, it may be useful to make the equation more stable. To do this let $A = \begin{pmatrix} V \\ \lambda I \end{pmatrix}$ and let $b = \begin{pmatrix} y \\ 0 \end{pmatrix}$, where I is an $m \times m$ identity matrix and 0 is an $m \times 1$ zero vector. Here A is an $(n+m) \times m$ matrix and b is a $(n+m) \times 1$ vector. Consider the least square problem:

$$\min ||Ax - b||$$

where the norm (—— ——) indicates the usual Euclidean norm. The normal equations solution to this least squares problem is [Golub and Van Loan, 1996, p. 237] $x = (A^T A)^{-1} A^T b = (\lambda^2 I + V^T V)^{-1} V^T y$. Therefore

$$\hat{y}^* = V^*(\lambda^2 I + V^T V)^{-1} V^T y = V^* x = K_1^* V_{11}^{-T} x.$$

However, it can be more accurate to solve a least squares problem using the QR factorization [Golub and Van Loan, 1996, p. 244]. In this approach [Golub and Van Loan, 1996, p. 239] one first factors $A = QR$ where Q is an $(n+m) \times m$ matrix with orthonormal columns and R is an $m \times m$ right triangular matrix. Then $x = R^{-1} Q^T b = R^{-1} Q^T \begin{pmatrix} y \\ 0 \end{pmatrix}$ so that

$$\hat{y}^* = V^* x = K_1^* V_{11}^{-T} x = K_1^* V_{11}^{-T} R^{-1} Q^T b = K_1^* V_{11}^{-T} R^{-1} Q^T \begin{pmatrix} y \\ 0 \end{pmatrix}.$$

With the above algorithm $\hat{y}^*$ can still be solved quickly. The flop count for solving this new equation is $O(nm^2)$.

7 Subset of Regressors

7.1 Subset of Regressor without QR Factorization

The subset of regressors method is similar to V Formulation in that an approximation of the traditional Gaussian processes equation is formed. However, low rank approximation is not used directly, rather the matrices K, K^* are partitioned into submatrices, only some of which are used in the calculation. Linear algebra can then be used to manipulate the original equation into an approximately equal equation that may be solved.

Suppose there exists a $n \times m$ matrix V such that $K = VV^T$ (This is of course not possible if K is of rank greater than m, and that is why the solution given by this method is only approximate). Then:

$$\hat{y}^* = V^*(\lambda^2 I + VV^T)^{-1}V^T y$$

We partition these matrices as follows:

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} = \begin{pmatrix} K_1 & K_2 \end{pmatrix}$$
$$K^* = \begin{pmatrix} K_1^* & K_2^* \end{pmatrix} \text{ and } V = \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix}$$

K_{11} is $m \times m$.
K_{21} is $(n-m) \times m$.
K_1 is $n \times m$.
K_1^* is $p \times m$.
V_{11} is $m \times m$.

It follows that:

$$\begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} = \begin{pmatrix} K_1 & K_2 \end{pmatrix} = \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix} \begin{pmatrix} V_{11}^T & V_{21}^T \end{pmatrix} = V \begin{pmatrix} V_{11}^T & V_{21}^T \end{pmatrix}$$

Therefore $K_1 = VV_{11}^T$ and $K_{11} = V_{11}V_{11}^T$, and similarly $K_1^* = V^*V_{11}^T$.

Assuming $V_{11} \in R^{m \times m}$ is invertible we have $K_1 V_{11}^{-T} = V$ and $k_1^* V_{11}^{-T} = V^*$

Substituting into $\hat{y}^* = V^*(\lambda^2 I + VV^T)^{-1}V^T y$ yields:

$$\begin{aligned}
\hat{y}^* &= K_1^* V_{11}^{-T}(\lambda^2 I + V_{11}^{-1} K_1^T K_1 V_{11}^{-T})^{-1} V_{11}^{-1} K_1^T y \\
\hat{y}^* &= K_1^* [V_{11}(\lambda^2 I + V_{11}^{-1} K_1^T K_1 V_{11}^{-T}) V_{11}^T]^{-1} K_1^T y \\
\hat{y}^* &= K_1^* [\lambda^2 V_{11} V_{11}^T + K_1^T K_1]^{-1} K_1^T y \\
\hat{y}^* &= K_1^* [\lambda^2 K_{11} + K_1^T K_1]^{-1} K_1^T y
\end{aligned}$$

Note that $\lambda^2 K_{11} + K_1^T K_1$ is an $m \times m$ matrix, so that this method is also $O(nm^2)$. Also note that V is not present in the final equation, so that it does not need to be calculated. For that reason, the subset of regressors method is the fastest of our methods. However, it is also the least stable.

Above we developed the subset of regressor method by first presenting the V method. We should add that the subset of regressors method can be derived directly from $\hat{y}^* = K^*(\lambda^2 I + K)^{-1} y$. To do this approximate K and K^* using

$$K \cong \widehat{K} \equiv K_1 K_{11}^{-1} K_1^T$$

and

$$K^* \cong \widehat{K}^* \equiv K_1^* K_{11}^{-1} K_1^T.$$

Next in $\hat{y}^* = K^*(\lambda^2 I + K)^{-1} y$ we replace K with $\widehat{K}$ and K^* with $\widehat{K}^*$ so that

$$\begin{gathered}
\widehat{y}^* \cong \widehat{K}^*(\lambda^2 I + \widehat{K})^{-1} y = \\
K_1^* K_{11}^{-1} K_1^T (\lambda^2 I + K_1 K_{11}^{-1} K_1^T)^{-1} y = \\
K_1^* K_{11}^{-1} (\lambda^2 I + K_1^T K_1 K_{11}^{-1})^{-1} K_1^T y = \\
K_1^* (\lambda^2 K_{11} + K_1^T K_1)^{-1} K_1^T y.
\end{gathered}$$

Also in this section note that if, in the subset of regressors formula, we substitute $K_1 = V V_{11}^T$ and $K_{11} = V_{11} V_{11}^T$ then the V formulation equations follow from the subset of regressor formula.

Finally, we note that the above derivations show that the V formulation and the subset of regressors formulas are mathematically equivalent, assuming that the order of the columns and rows in K are the same when applying the formulas. This is the case, for example, if the V formulation uses Cholesky factorization without pivoting to form V or if, assuming that pivoting is used in the Cholesky factorization, then the columns and rows of K are pivoted in the same manner prior to applying the subset of regressors formulas. Therefore the primary difference is in the numerical properties of the formulas, which we address shortly.

7.2 QR Factorization in Subset of Regressors

The subset of Regressors method using QR factorization requires only a slight modification to the Subset of Regressors method. As in the previous section, an approximation of the traditional equation is given by:

$$\hat{y}_* \; = K_1^*[\lambda^2 K_{11} + K_1^T K_1]^{-1} K_1^T y$$

Now we factor the $m \times m$ matrix $K_{11} = V_{11} V_{11}^T$. The $m \times m$ matrix V_{11} can be calculated by Cholesky factorization of K_{11}. Now let $A = \begin{pmatrix} K_1 \\ \lambda V_{11}^T \end{pmatrix}$ and let $b = \begin{pmatrix} y \\ 0 \end{pmatrix}$, where 0 is an $m \times 1$ zero vector. Here A is an $(n+m) \times m$ matrix and b is a $(n+m) \times 1$ vector. Consider the least square problem:

$$\min ||Ax - b||.$$

The normal equations solution to this least squares problem is [Golub and Van Loan, 1996, p. 237] $x = (A^T A)^{-1} A^T b = (\lambda^2 V_{11} V_{11}^T + K_1^T K_1)^{-1} K_1^T y = (\lambda^2 K_{11} + K_1^T K_1)^{-1} K_1^T y$. Therefore

$$\hat{y}^* = K_1^*[\lambda^2 K_{11} + K_1^T K_1]^{-1} K_1^T y = K_1^* x$$

However, it can be more accurate to solve a least squares problem using the QR factorization [Golub and Van Loan, 1996, p. 244]. In this approach [Golub and Van Loan, 1996, p. 239] one first factors $A = QR$ where Q is an $(n+m) \times m$ matrix with orthonormal columns and R is an $m \times m$ right triangular matrix. Then $x = R^{-1} Q^T b = R^{-1} Q^T \begin{pmatrix} y \\ 0 \end{pmatrix}$ so that

$$\hat{y}^* = K_1^* x = K_1^* R^{-1} Q^T b = K_1^* R^{-1} Q^T \begin{pmatrix} y \\ 0 \end{pmatrix}.$$

With the above algorithm $\hat{y}^*$ can still be solved quickly. The flop count for solving this new equation is $O(nm^2)$.

8 Numerical Stability

8.1 Squaring the condition number

Consider the subset of regressor method without QR (SR), the subset of regressor method with QR (SRQR) and the V formulation (VF). They all provide a solution to

$$\min ||Ax - b||$$

where $A = \begin{pmatrix} K_1 \\ \lambda V_{11}^T \end{pmatrix}$ and let $b = \begin{pmatrix} y \\ 0 \end{pmatrix}$, where 0 is an $m \times 1$ zero vector. Here A is an $(n+m) \times m$ matrix and b is a $(n+m) \times 1$ vector.

$$\text{SR: } x = [\lambda^2 K_{11} + K_1^T K_1]^{-1} K_1^T y$$

$$\text{SRQR: } x = R^{-1} Q^T \begin{pmatrix} y \\ 0 \end{pmatrix}$$

$$\text{VF: } x = V_{11}^{-T} (\lambda^2 I + V^T V)^{-1} V^T y.$$

Here $A = QR$ where Q is a $(m+n) \times m$ matrix with orthonormal columns and R is a $m \times m$ right triangular matrix. Also $K_1 = VV_{11}^T$ and $K_{11} = V_{11}V_{11}^T$ where V is an $n \times m$ matrix and the $m \times m$ matrix V_{11} is the first m rows of V.

The SR formula is just the normal equation solution to the above least squares problem and the SRQR formula is the QR solution. As indicated in [Golub and Van Loan, 1996, p. 244-245] there is potential for numerical instability in using normal equations (since the error growth is always governed by the square of the condition number of A) whereas as use of the QR factorization is numerically stable. Here the phrase "numerical stability" is used to indicate that the algorithm produces a calculated answer that is close to the accuracy of the answer permitted by the condition number of the problem (see [Golub and Van Loan]). Therefore we should expect that the SRQR formulation will produce more accurate results than the SR formulation.

To consider the VF formulation we will begin by considering the special case where $\lambda = 0$ and later consider the more general case. In the case that $\lambda = 0$ we have:

$$\text{VF: } x = V_{11}^{-T}(V^TV)^{-1}V^Ty.$$

where $K_1 = VV_{11}^T$. There is a potential concern in using this equation since to construct x the linear system of equations

$$(V^TV)z = V^Ty$$

must be solved. Forming V^TV squares the condition number of V which, potentially could lead to the introduction of undesirable computer arithmetic errors. However we will argue that the matrix $B = V^TV$ is diagonally equivalent to matrix that, in practice, is well conditioned and that this will limit the growth of computer arithmetic errors.

Now V is formed by a partial Cholesky factorization of the symmetric semi definite (SSD) matrix K. For a SSD matrix it follows, for each $i = 1, \ldots, m$, that the i^{th} diagonal entry is at least as large in magnitude as any off diagonal entry in row i or column i [Trefethen, 1996, p. 176]. From properties of the Cholesky factorization it then follows that the lower trapezoidal matrix V has the property that, for each $i = 1, \ldots, m$, the i^{th} diagonal entry in V is at least as large in magnitude as any entry in column i. Therefore we can write V as $V = WD$ where D is an $m \times m$ diagonal matrix and W is an $n \times m$ lower trapezoidal matrix with all entries one or less in magnitude and with ones on the diagonal. Indeed this matrix W is identical to the lower trapezoidal matrix produced if Gaussian elimination with partial pivoting is applied to K (and indeed, in this case, Gaussian elimination with partial pivoting will not pivot any entries). However it is known [Trefethen, 1997, p. 169] that, in practice, the lower trapezoidal matrices produced by Gaussian elimination with partial pivoting are well conditioned.

Thus V is a diagonal rescaling of a well conditioned matrix W. Now it follows that we can write $V = WD_1D_2$ where the entries of the diagonal matrix D_1 are between 1 and 2 and where entries in D_2 are exact powers of 2. Since W is well conditioned then so is $U = WD_1$ (since $cond(WD_1) \leq cond(W)cond(D_1) \leq$

2 *cond*(W)). Now note, since V^TV is symmetric positive definite, that algorithms for solving $V^TVz = V^Ty$ do not require pivoting. However, in this case, scaling V^TV by exact powers of two does not effect the numerical stability of the solution z to $V^TVz = V^Ty$ [Forsythe and Moler, 1967, pp. 37-46]. Therefore the growth of computer arithmetic errors in solving $V^TVz = V^Ty$ is indicated by the condition number of U^TU (the rescaling of V^TV by exact powers of two). However since U has a modest condition number (since it is well conditioned in practice) so does U^TU (since its condition number is the square of the condition number of U).

The conclusion from the above argument is that we do not square the condition number of a ill-conditioned matrix in the VF formulation and that we expect that this will limit the error growth in using the VF formulation.

To consider the case that $\lambda \neq 0$ we note that in this case the condition number of $B = (\lambda^2 I + V^TV)$ will be important in solving

$$(\lambda^2 I + V^TV)z = V^Ty.$$

However we have

Theorem 1 *For any* $\lambda \geq 0$, $cond(\lambda^2 I + V^TV) \leq cond(V^TV)$.

Proof. If V^TV has eigenvalues $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_m \geq 0$ then the eigenvalues of $(\lambda^2 I + V^TV)$ are $(\lambda^2 + \alpha_i)$, $i = 1, \ldots, m$. Therefore $cond(V^TV) = \alpha_1/\alpha_m$ and $cond(\lambda^2 I + V^TV) = (\alpha_1 + \lambda^2)/(\alpha_m + \lambda^2)$. However it follows easily that $\alpha_1/\alpha_m \geq (\alpha_1 + \lambda^2)/(\alpha_m + \lambda^2)$. ∎

Since $cond(\lambda^2 I + V^TV) \leq cond(V^TV)$ we expect that solving $(\lambda^2 I + V^TV)z = V^Ty$ with $\lambda \neq 0$ will be more accurate than solving this equation with $\lambda = 0$. Since we have argued that the error growth in solving this equation for $\lambda = 0$ should be limited we expect that this should also be true when $\lambda \neq 0$.

In summary our the above argument suggest that the SRQR and VF formulations should have improved numerical stability in comparison to the the SR formulation. This will be supported by the numerical experiments that we present later.

We should add that the SR formulation has the advantage that it can be more efficient and require less memory than the other approaches. We will see this in our later experiments and also see that the VF formulation can also be quicker and require less memory than the SRQR method.

8.2 Pivoting

When using the SRQR formulation we solved the least squares problem

$$\min ||Ax - b||$$

using the QR factorization. Although the SRQR formulation should have improved accuracy compared to the SR formulation the accuracy, the accuracy of the solution to $\min ||Ax - b||$ still depends on the the condition number of of A (see [Golub and Van Loan, 1996, p. 236-245] for details). Therefore it may be useful to try to control the condition number of $A = \begin{pmatrix} K_1 \\ \lambda V_{11}^T \end{pmatrix}$.

Also in the VF formulation after one solves for z in $(\lambda^2 I + V^T V)z = V^T y$ to calculate x one must solve

$$V_{11}x = z.$$

Therefore the condition number of V_{11} will be a factor in the accuracy of the calculated x. We did not include this in our earlier analysis.

For the above reasons we would like to limit the condition numbers of A and V_{11}. These condition numbers can be affected by the ordering of the columns of the covariance matrix K. For example consider the simple 3 by 3 SSD matrix

$$K = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

If we consider a rank two approximation to K, without reordering the columns and rows of K, then it follows easily that

$$V_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},$$

$$K_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

and

$$A = \begin{pmatrix} K_1 \\ \lambda V_{11} \end{pmatrix}.$$

Therefore in this case V_{11}, K_1 and A are all exactly singular and therefore have condition numbers $= \infty$.

Now consider a reordering of of K, switching the second and third rows and columns. For $\tilde{K}$, the reordered K we have

$$\tilde{K} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

If we consider a rank two approximation to $\tilde{K}$ then it follows easily that

$$\tilde{V}_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\tilde{K}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and

$$\tilde{A} = \begin{pmatrix} \tilde{K}_1 \\ \lambda \tilde{V}_{11} \end{pmatrix}.$$

After reordering the new matrices $\tilde{V}_{11}$, $\tilde{K}_1$ and $\tilde{A}$ are well conditioned. For example $cond(\tilde{V}_{11}) = cond(\tilde{K}_1) = 1$.

As mentioned in Section 5.5 the partial Cholesky factorization with pivoting can be used to reorder the columns of K in a suitable manner.

8.3 Related methods

The V formulation is related to a general least squares algorithm due to Peters and Wilkinson [Bjork, 1996, pp. 73-76]. Also the formulas in [Wahba, 1990, p. 136] are related to to our V formulation formulas. The formulas in Wahba were used to develop practical techniques for the computations related to generalized cross validation.

9 Choice of Rank

In using low rank approximation the choice of rank will affect the accuracy of the approximation. It may be impractical to repeat the computations for a variety of different ranks and is useful to have techniques to facilitate determination of the accuracy of a variety of low rank approximations.

It is well known that there are efficient techniques to remove columns (downdate) and add columns (update) in solving general least squares problems [Bjork, 1996, pp. 127-152]. These techniques can be adapted to calculations for all of our low rank approximation techniques (VF, VQ, SRQR). For example one can calculate the solution for a rank m low rank approximation and use these techniques to calculate the rmse (root mean square errors) of the approximations for low rank approximation for all ranks $k, k = 1, \ldots, m$. This can be implemented efficiently and require only a small additional amount of computing time. We have included this option in our code.

10 Optimization of Hyperparameters

The hyperparameters for any particular covariance function K may be optimized by minimizing the output of a function ϕ which takes as input a vector of hyperparameters θ_0 and outputs $\hat{y}^*_K$, the root mean square error of the approximation with respect to K. We used two available MATLAB functions, fminsearch.m and minimize.m, to accomplish this and find a good value of θ to use.

10.1 Nelder-Mead Simplex Minimization

The standard MATLAB function fminsearch uses an initial estimate to minimize a multi-variable function, which is known as unconstrained nonlinear optimization. The algorithm used by fminsearch is the Nelder-Mead simplex[Nelder and Mead, 1965] search method, which does not make use of gradients. This method becomes extremely slow as the number of hyperparameters increases, but it can output the true optimum values but it is possible for fminsearch.m to output a local minimum.

10.2 Marginal Likelihood and Minimize

Marginal likelihood $p(y|X,\theta)$ is the probability of observing a set of redshift measurements in y given photometric filter observations X and a series of hyperparameters, stored as a vector θ. To better predict the values of redshifts y, one can set X and y as fixed constants whose values are determined by previously-observed training data sets and maximize the marginal likelihood function, setting θ as variable. An "optimal" set of hyperparameters can then be found. One can alternatively determine optimal hyperparameters by maximizing the natural logarithm of the likelihood function, which according to [Rasmussen and Williams, 2006, p. 113] is:

$$\log p(y|X,\theta) = -\tfrac{1}{2} y^\top A^{-1} y - \tfrac{1}{2}\log\det(A) - \tfrac{n}{2}\log 2\pi$$

We utilize minimize.m, a Matlab program that minimizes a multivariate differentiable function provided by Rasmussen [2006], to minimize the negative logarithm of the marginal likelihood function–an equivalent procedure to the maximization of the original marginal likelihood function–and thus determine optimal hyperparameters given each of the covariance functions under consideration. Computer memory issues have limited the minimization process to 2000 data training points in X and y.

10.3 Optimization Method

The speed of minimize.m allowed us to use it to calculate good values for the vector of hyperparameters θ for each method and covariance matrix at our disposal. The vector θ was used to compare the ability of different method/covariance matrix combinations to predict y^*. The function fminsearch.m can then be used with the combinations which seem to best predict y^* in order to attempt to improve the accuracy. However, the new results found with fminsearch did not constitute a significant improvement.

We should note that minimize is not tailored to any of the low rank approximation algorithms. To choose the hyperparameters using minimize one must selected a subset of the data (for example 1000 galaxies). Minimize then uses the traditional Gaussian process formula in it optimization calculations. This procedure was followed in the examples presented in [Rasmussen and Williams, 2006, p. 182-184]. We also followed this procedure. We did not find a significant change in our results if we varied the subset used to select the hyperparameters. Although it might be of interest to develop a method for selecting hyperparameters that is tailored to low rank approximations we did not do so in this project.

With fminsearch here is the danger of overfitting since our use of fminsearch makes use of y^*. Our hypothesis was that the dataset of 180,045 inputs that we were given was sufficiently large to ensure that overfitting would not result in significantly different results for new datasets. This hypothesis was confirmed by splitting y^* into two smaller datasets indexed by I_1 and I_2, where I_1 and I_2 are distinct indexes containing $10,114$ numbers and $10,115$ numbers respectively (recall that y^* contains $20,229$ elements total). The fminsearch function was used to find optimized hyperparameters corresponding to $y^*(I_1)$ and $y^*(I_2)$, θ_{I_1} and θ_{I_2}. The number $|\theta_{I_1} - \theta_{I_2}|$, if large enough, is clear evidence of overfitting in our optimization. However, in our case $|\theta_{I_1} - \theta_{I_2}|$ was not large enough to be significant in terms of our computations (on the order of 10^{-3} or 10^{-4} in each case), especially with regard to the V formulation and V formulation with QR factorization methods, which are relatively independent of the hyperparameters chosen.

Our calculations with covariance functions made use of the covariance functions that are included with the software associated with [Rasmussen and Williams, 2006]. The hyperparameters that we calculated include λ and other parameters that are detailed in Rasmussen and Williams software. We refer the reader to http://www.gaussianprocess.org/gpml/code/matlab/doc/ for a detailed definition of the hyperparameters. Some of Rasmussen and Williams' covariance functions came with two versions. The ARD (for automatic relevance determination) version had more hyperparameters the the simpler versions. Also Rasmussen and Williams find it convenient to use the logarithm of hyperparameters as input to there routines. Below is a list of the logarithms (base e) of the hyperparameters used in our calculations. The were all determined using minimize (selecting a subset of 2000 galaxies in the minimize calculations):

Table of Hyperparameters Used

Covariance Matrix	$\log \theta_1$	$\log \theta_2$	$\log \theta_3$	$\log \theta_4$	$\log \theta_5$	$\log \theta_6$	$\log \theta_7$	$\log \theta_8$
Rational Quadratic ARD	0.3942	0.4083	0.6969	1.0122	1.3051	-1.0845	0.8351	-3.9720
Rational Quadratic	0.7061	-0.9527	0.1628	-3.9692				
Squared Exponential ARD	0.1247	0.2648	0.6504	0.9302	0.9864	-1.2208	-3.9639	
Squared Exponential	0.2725	-1.3061	-3.9765					
Matern 3	1.8813	-0.6291	-3.9842					
Matern 5	1.1055	-0.9277	-3.9724					
Neural Net	-2.8870	-0.5853	-4.1261					
Polynomial, r = 2	-3.7483	-8.3263	-3.8932					
Polynomial, r = 3	-3.7483	-8.3263	-3.8932					
Polynomial, r = 4	0.6935	-0.4289	-3.9343					

11 Timings

The tic toc commands in MATLAB were used to time different methods under the same conditions. The times shown in the table below represent the amount of time it took to calculate $\hat{y}^*$, the predicted redshift. In terms of speed, the methods used this semester were at least as good as those from last semester.The following table lists the computing times when using the quadratic covariance function with rank = 21:

K\N	1000	10,000	180,000
V Formulation with Partial Cholesky	0.1410	0.2340	4.3280
V Formulation	0.2340	0.1560	1.5000
V Formulation with QR Factorization	0.5930	0.1560	2.0470
Reduced Rank with Cholesky	0.5620	0.2650	4.5940
Reduced Rank	0.0940	0.1570	1.5470
Cholesky Update with Partial Cholesky	0.3120	0.4690	10.2810
Cholesky Update with QR Factorization	0.1250	0.3440	6.5460
Subset of Regressors	0.0150	0.0460	0.2660
Subset of Regressors with QR Factorization	0.0470	0.2500	5.1250
Subset of Regressors with Partial Cholesky	0.0780	0.2500	4.3750
Conjugate Gradient with Partial Cholesky	0.7030	0.8440	21.3600
Conjugate Gradient	0.1410	0.6720	14.1090
Gibb's Sampler	241.5460	232.1250	232.5150

Using results in the above table we can picture several of the methods (SR and V) introduced in this semester's project with several of the methods used in the first semester's portion of the project (RR- reduced rank, CU - Cholesky update, CG - conjugate gradient):

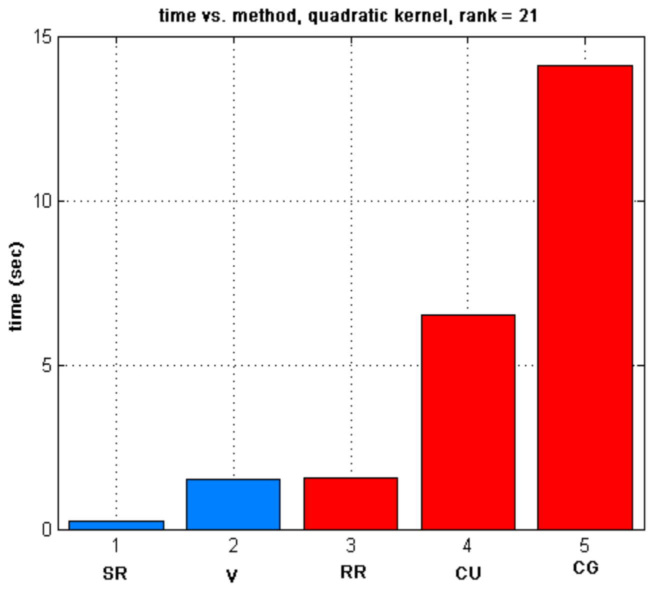

Figure 1: Times of two methods used in the current semester and three methods used in the previous semester.

Note that the SR and V method are faster than the methods (RR, CU and CG) used in the first part of this project. The times of the V method and the RR method are close but it should be added that the RR method can be less accurate than the V method.

We have done additional timings that compare the SR, V method without pivoting, SRQR, and the V method with pivoting (using the column and row order determined by the partial Cholesky factorization). In the SR, V method without pivoting, and SRQR method we left the column and row ordering of the covariance matrix unchanged. In some cases we also did timings of the result of calculating the RMSE errors for all low rank approximations. In the graphs we have used "histor" to indicate these calculations. These calculations were done using the neural network covariance function and with a rank of 100 and also with a rank of 1000.

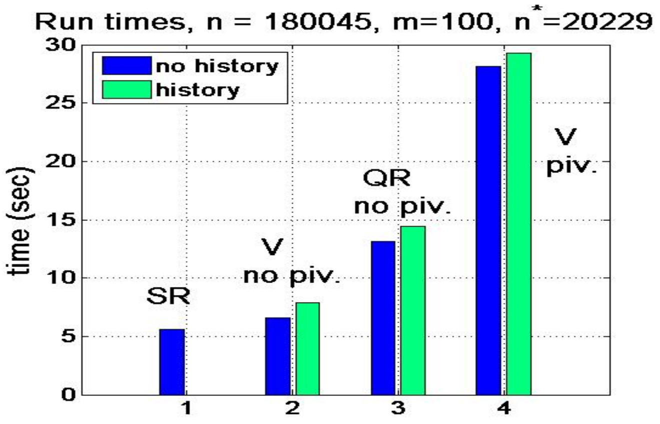

Figure 2: Comparision of the times of four methods, rank = 100

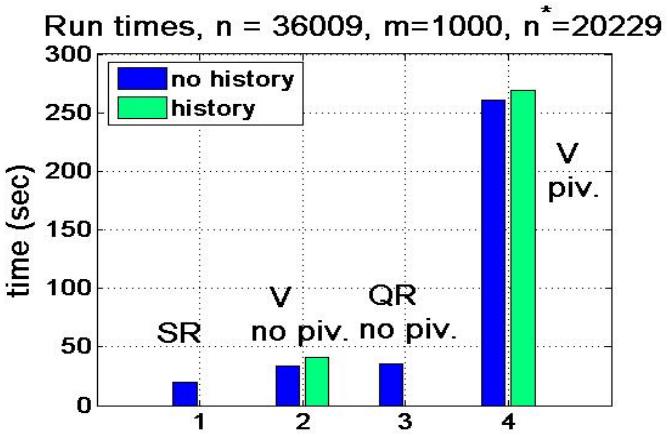

Figure 3: Comparision of the times of four methods, rank = 1000

We see from these plots that the extra computations involved in calculating the RMSE errors for all lower order approximations is small compared to the time for the initial calculations in computing the low order approximation. We also see that the SR method is the fastest, the V method without pivoting is next fastest and this is followed by the SRQR method. The V method with pivoting is slowest.

We should add that part of the reason that the V method with pivoting is slower is that Matlab does not have a built in routine for the Cholesky factorization with pivoting. We therefore wrote the code to do this using Matlab's interpreted code which is slower than compiled code. Although we did not do so, it is possible that one could write this code in a compilable language like Fortran or C (and then use Matlab's MEX utility). Such code would significantly reduce the run times for the V method with pivoting.

12 Accuracy and Stability

12.1 Bootstrap resampling results

One possible way to minimize the error in our prediction is to simply use the entire training dataset of 180,045 broadband measurements to train each of our methods to predict future redshifts given a separate set of broadband "testing" points. A limitation of this approach is that essentially only one sample is available from which to gauge the accuracy of our methods in the general case. It may be the case that is different training data were introduced, the result would be quite different. Although we have a limited set of data, somehow we want to determine how the method will respond to new sets of data.

Given the size of our training dataset, it is reasonable to assume that it is a representative sample of the galaxies in the known universe, so that the galaxies with the various properties contained in the dataset occur in the universe at approximately the same frequency. If this hypothesis is correct, then a statistical inference technique called bootstrap resampling may be employed. Bootstrap resampling takes the data that we have as the total known "population," from which we can randomly sample to generate new training datasets. The standard approach to bootstrap resampling is to sample with replacement. If the size of the sample is less than the size of the training set one could also use resampling without replacement. Using each of these bootstrap samples, all of size 180,045, we can test our methods by fitting a prediction model, and compare the errors between the predictions and the actual redshifts. In this way we no longer rest all of our inferences on a single result for each method; instead we have a range of values, similar to the situation of having multiple datasets from which to choose. The bootstrap method is implemented as follows.

With each method, we generate a random sample of galaxy broadband measurements and corresponding redshifts from the entire 180,045 dataset, X;

each sample is of size 180,045 and we adhere to the property of traditional bootstrapping of replacement, in which a galaxy can be selected more than once for a sample. We then run the model on a separate testing dataset, which we call X^*, to predict the corresponding redshifts of each galaxy. These redshift predictions generated by the model are then compared to the actual observed redshifts y^* by calculating the root mean square error (RMSE) $||y^* - \hat{y}^*||/\sqrt{p}$ where the norm is the usual Euclidean norm and p is the number of entries in y^* and $\hat{y}^*$. We then repeat this process 100 times with 100 random samples from X across each of the methods and covariance functions.

Figures 4-7 show the results of the 100 bootstrap runs of both the V Formulation and Subset of Regressors method using several different covariance functions. The covariance function which performed best in these trials was the neural network covariance function. The 100 model runs are ordered from low to high according to RMSE. While the SR method matches the VQ method in terms of accuracy for the best 40 runs, the second half of the runs produced steadily worse errors. We attribute this lack of stability to computer arithmetic issues inherent in the SR method. For the neural network covariance function, due to the instability of the SR method, its RMSE error is driven up to .0267. The V Formulation method is shown to be both stable and accurate. For the neural network covariance function the RMSE error equal to .0215 with the V method and the variation over individual samples is less than .001. The covariance function which performed best with all methods was the Neural Net covariance function, as shown in figure 7.

Comparing the bootstrap accuracy results of the SR and V methods with methods explored in the first semester of the project [Cayco, So, et. al., 2007] such as Cholesky Update and Conjugate Gradient (figure 8), we find that the V Formulation provides an absolute improvement in both accuracy and stability. Contrarily the subset of regressors method is comparably lacking in both accuracy and stability, especially when the covariance matrix used is of low rank. The quadratic kernel, for example, is of rank 21, and the SR method performs badly with the quadratic kernel as is clearly indicated in figure 8. However, when a larger rank is used such as a low rank approximation of the Neural Network covariance function with $m = 100$, the accuracy and numerical stability is greatly improved as in figures 4-7. Recall that when the partial Cholesky algorithm is used to pivot the covariance matrix K used in the SR method, it is equivalent to the V formulation. This indicates that pivoting as used in the partial Cholesky algorithm is essential to the greater numerical stability of the V formulation.

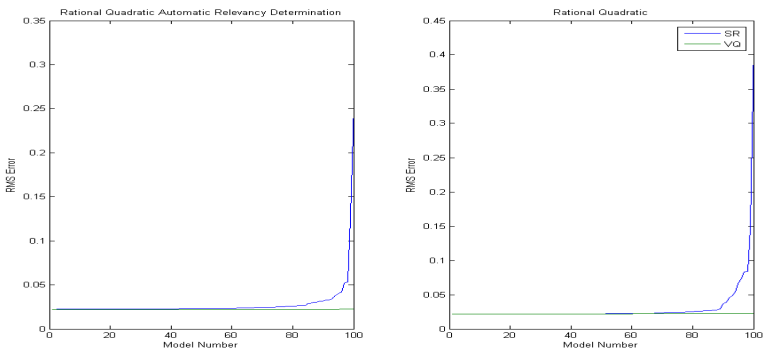

Figure 4: Bootstrap resampling with the Rational Quadratic covariance functions, rank = 100.

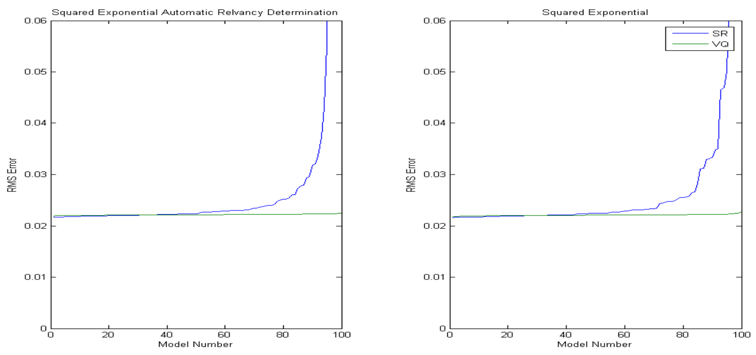

Figure 5: Bootstrap resampling with Squared Exponential covariance functions, rank = 100.

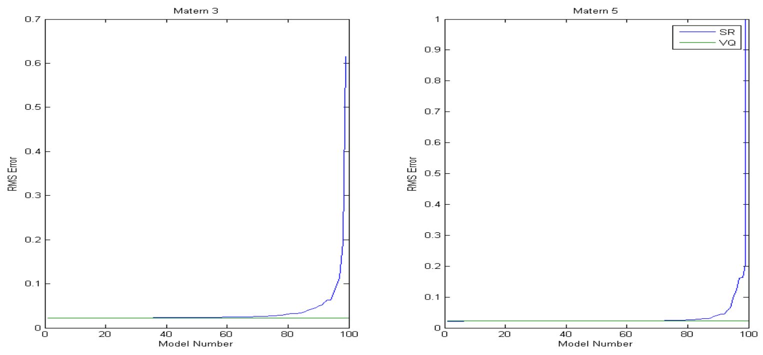

Figure 6: Bootstrap resampling with Matern class covariance functions, rank = 100.

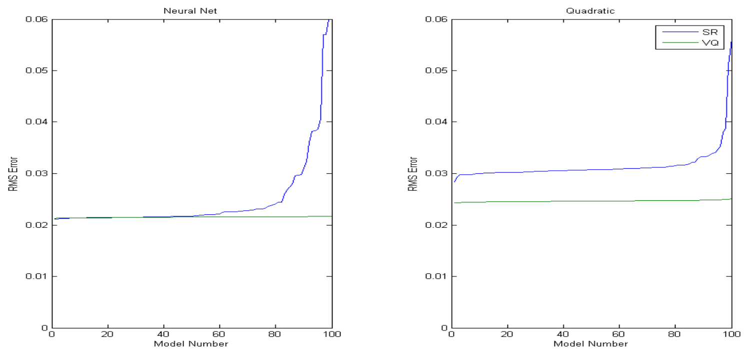

Figure 7: Bootstrap resampling with the Neural Net and Quadratic covariance functions, rank = 100.

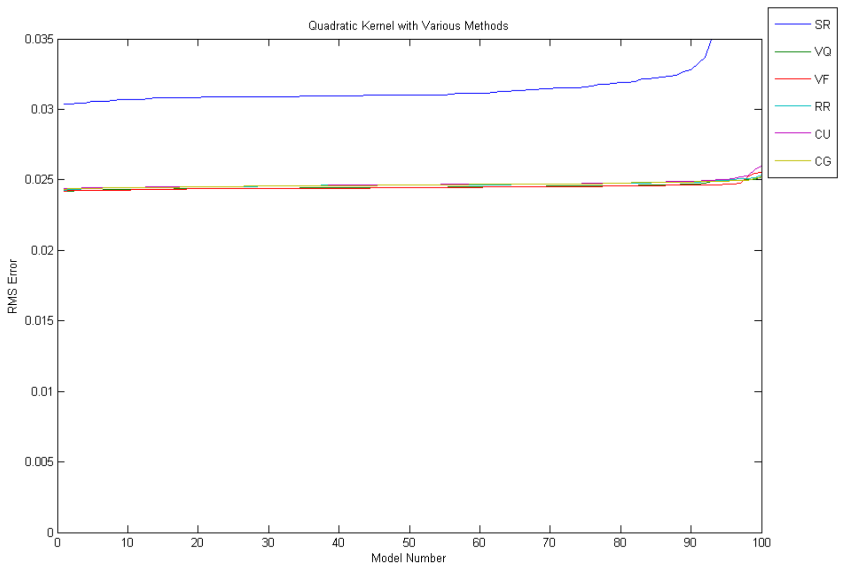

Figure 8: Bootstrap resampling for various other methods with the Quadratic kernel, rank = 100

VF: V Formulation
VQ: V Formulation with QR Decomposition
SR: Subset of Regressors
RR: Reduced Rank
CU: Cholesky Update
CG: Conjugate Gradiant

The most accurate covariance function in the above graphs was the neural network covariance function. The following graph provides more detail for this covariance function using the SR and V method with rank = 100. For comparison we also include the result of the V method using the quadratic covariance function with rank = 21 (the mathematical rank of the quadratic covariance function is 21). Note that the graph clearly indicates that the inaccuracy of the SR method approximately 80 of the 100 samples and also indicates that the neural network kernel is more accurate than the quadratic kernel.

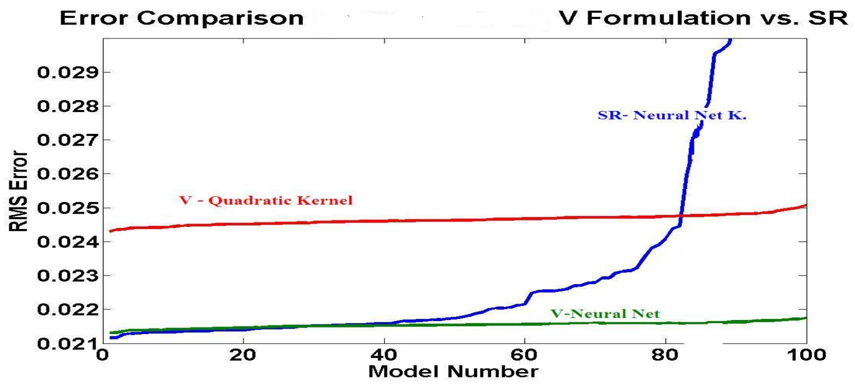

Figure 9: Bootstrap resampling for neural network kernel, rank =100, with the SR and VF methods, and the Quadratic kernel, rank = 21, V method.

12.2 The choice of rank

As mentioned in Section 9 and Section 11 we can calculate the RMSE errors corresponding to all lower rank approximations with a modest amount of extra effort. This can be used to illustrate the effect of varying the rank of the low rank approximation. For example we can do bootstrap resampling with a low rank approximation with rank $m = 1000$ and also obtain the result of bootstrap resampling with any lower rank.

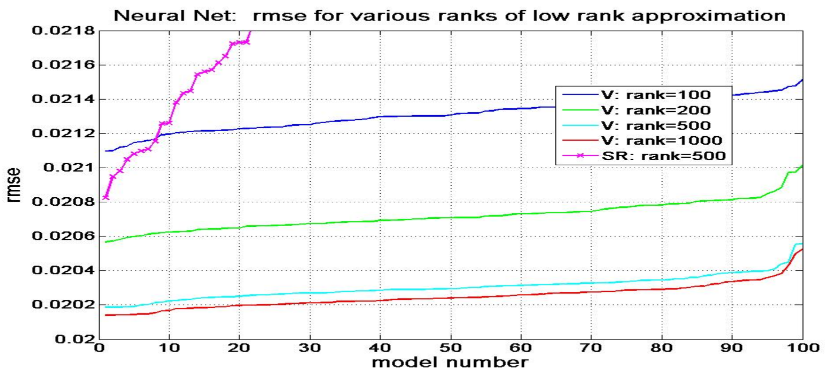

Figure 10: Bootstrap resampling with the neural network covariance function and ranks 100, 200, 500 and 1000 and, for comparison, the SR method with rank 500.

We can also use these calculations to plot the median rmse error versus rank.

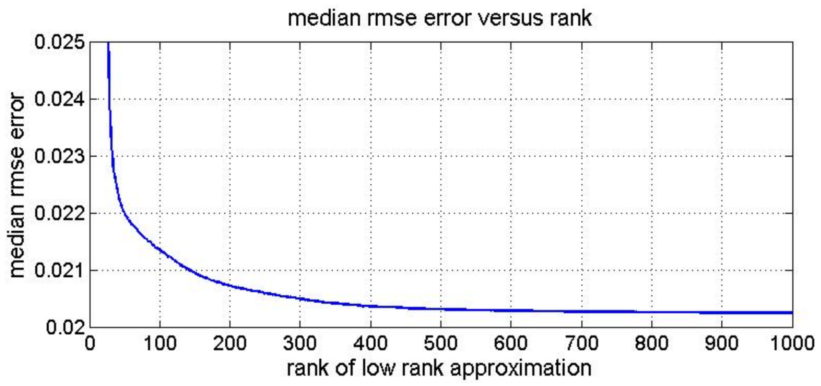

Figure 11: The median rmse error versus rank.

For the neural network covariance function the above results indicate that the approximations become significantly more accurate as the ranks increase from 100 to 200 and to 500, but that the improvement in accuracy is small for ranks larger than 1000. Also note that Figure 10 again indicates the potential instability of the SR approach.

We should note that due to memory limitations with our rank = 1000 computations, we reduced the training set size to 36009 for the above runs. Also note that in the above runs we used bootstrap resampling without replacement.

12.3 The effect of pivoting

The following figure compares the result of using the V method without pivoting and the V method with pivoting:

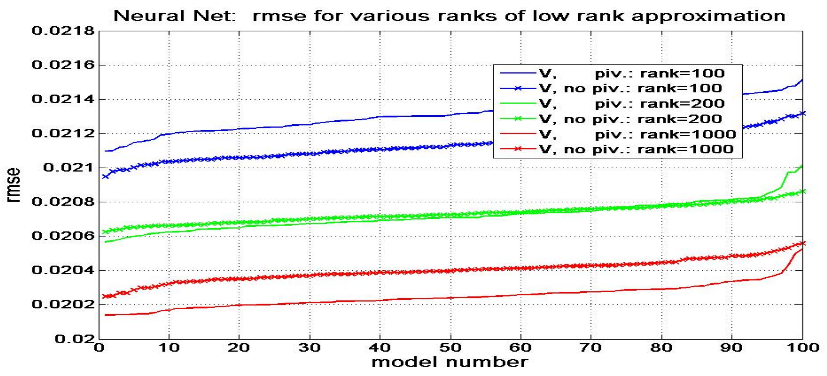

Figure 12: Comparison of the V method without pivoting and the V method with pivoting with the neural network covariance function and ranks of 100, 200 and 1000.

These results suggest that, for the redshift data, pivoting does not have a large effect on the accuracy of the low rank approximation.

12.4 The choice of the testing set size

The training set for the redshift calculations is large – it consists of more than 180000 galaxies. Is all this data necessary to develop a good model?

We can explore this by repeating calculation using training sets of various sizes. The results of this calculation is summarized in the Figure 13.

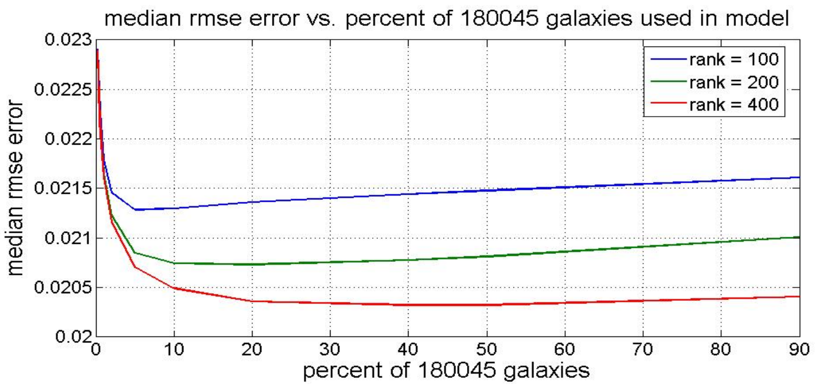

Figure 13: The effect of varying the training set size.

This figure suggests that the entire data set is not needed to get good approximations. Indeed after the training set size is approximately 36000 (20 % of 180045) the is little or no improvement in the accuracy of the approximation as the training set size is increased.

12.5 Comparison with Traditional Method

Although it is not practical to use the traditional Gaussian processes method with all of the available training data, it is possible with a small subset of the training data. In the following graph, 4000 of the $180,045$ rows of X were selected randomly without replacement to form $X(I)$ for some index I. The equation $\hat{y}^* = K^*(\lambda^2 I + K)^{-1}y$ was then used with K being the covariance matrix of $X(I)$. This process was repeated 100 times with different random indices (figure 14).

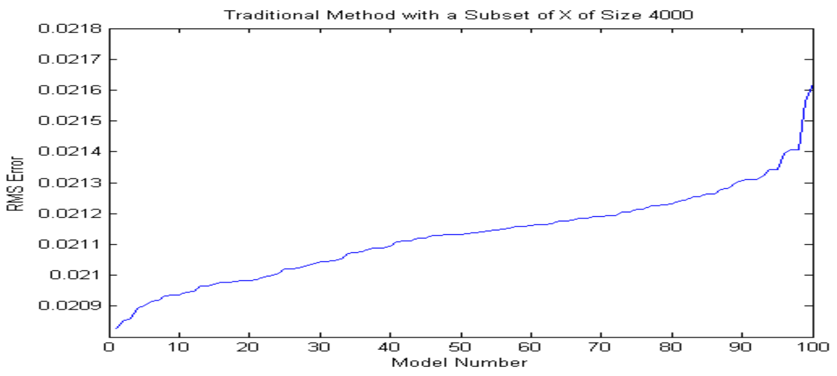

Figure 14: Bootstrap resampling (without replacement) with training sets consisting of 4000 galaxies.

A subset of 4000 inputs is the practical limit given the available resources. With 4000 galaxies the accuracy is worse than that the V Formulation with low rank approximations of ranks 200, 500 or 1000 (see figure 10). We can also redo the computations for training set sizes of less than 4000 galaxies to obtain the results in figure 15.

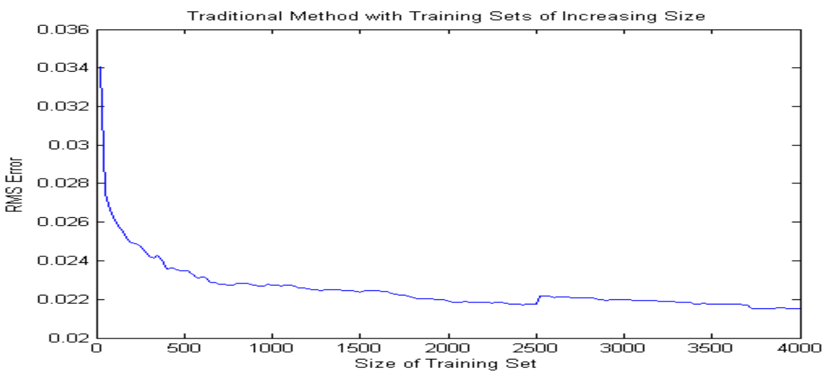

Figure 15: Error versus training set size. As the training set increases in size, the RMS seems to converge to 0.021.

However, this is not to say that the traditional method does not show any promise. It is not necessary to choose the training set randomly, as was done in the tests graphed below. There may be a way to choose a particular training set which is ideal in some way, and in that case the traditional method (or the V formulation with dramatically increased rank) may be more effective. One possible way to do this is to use the partial Cholesky decomposition to find an index of X which places the most linearly independent rows first. Unfortunately, the practical limit on the partial Cholesky decomposition on a $180,045 \times 5$ matrix is only $m = 1000$ or so, and so the effectiveness of this method may not currently be tested.

12.6 Comparison with alternate approaches

In [Way and Srivastava, 2006] redshift prediction are made with a variety of traditional and novel approaches. These approaches include traditional linear regression, traditional quadratic regression, artificial neural networks, an E-model approach and Gaussian processes using the quadratic kernel. See [Way and Srivastava, 2006] for details. In figure 16 we compare the results of our Gaussian process calculations with their results. The results labeled GP-V are our results using the neural network covariance function with rank 1000. The other results pictured are obtained from [Way and Srivastava, 2006]

The primary conclusion from this figure is that our results are competitive with the best known alternative results.

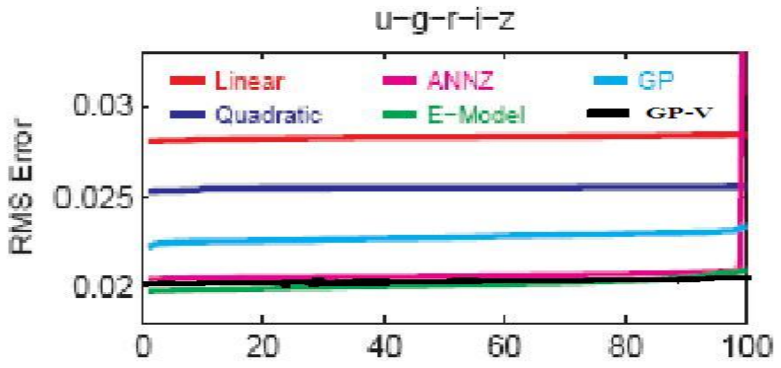

Figure 16: Comparison of a variety of methods of redshift prediction.

13 Outliers

Removing outliers from the testing matrix X^* may allow us to significantly lower the root mean square error of our prediction. Outliers may be calculated by sorting the vector $y^* - \hat{y}^*$. The graphs of $y^* - \hat{y}^*$ so far indicate that there is a set of outliers which is constant relative to choice of method and kernel. In other words, it is so far evidenced that there is a set of outliers common to the general method of approximating y^* as a Gaussian Process.The sorted graphs indicate that a significant set of outliers exist for each method and each covariance function. In fact, if 10 percent of the outliers are removed from the testing data, the RMS error on the remaining points drops to nearly 0.016. It cannot be concluded from the sorted graphs that the set of outliers is invariant with respect to choice of covariance function or method. However, the unsorted graphs do indicate that the outliers are a common set of points regardless of which covariance function is chosen and which method is used. This is because the "spikes" in these graphs tend to overlap.

13.1 Characterizing Outliers

Given the hypothesis that there is a constant set of outliers, they still may not be discarded unless they can be characterized, so that all inputs sharing a particular characteristic may be removed from the testing set. Let I be a vector corresponding to the index of the sorted $y^* - \hat{y}^*$ vector, with $\hat{y}^*$ corresponding to the Neural Net covariance matrix and V-formulation with QR factorization. The input columns of the testing data corresponding to U, G, R, I, Z were each sorted by the index I, meaning that the outliers should correspond to the first and last thousand entries or so of each of these resorted columns. However, examining the columns one at a time does not reveal any pattern. It is possible that some sort of multivariate correlation analysis would be able to detect a more complicated pattern among these columns. Similarly, $\hat{y}^*$ was resorted

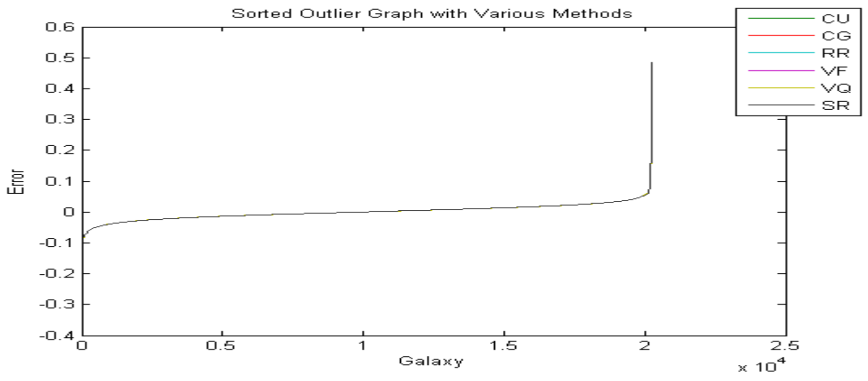

Figure 17: Graph which shows the presence of outliers.

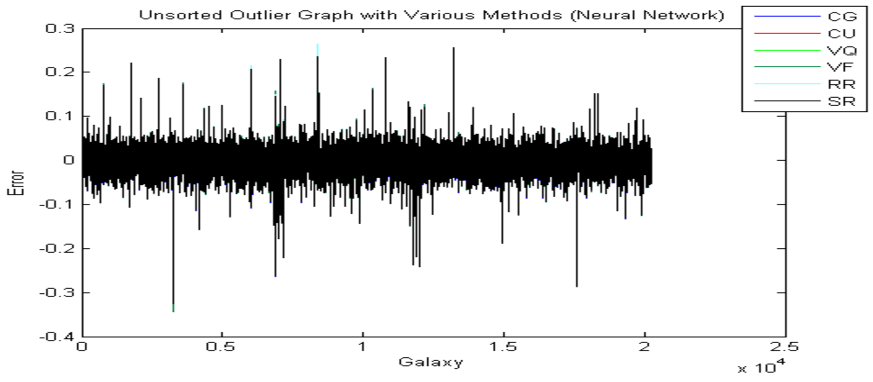

Figure 18: Graph which shows the distribution of outliers.

by the index I, and no pattern was readily apparent. This data can also be included in a more in-depth analysis. One approach that might help is to block groups of galaxies. It is also possible to view the characterization of outliers as a classification problem, and this idea is addressed later in this section.

However, if the inputs of these graphs are blocked together, and the output is taken to be the norm of these blocks, then a pattern begins to emerge.

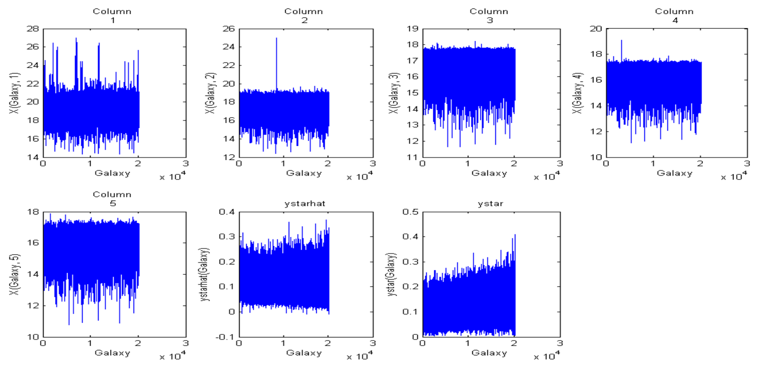
Figure 19: Columns of X^*, y^* and $\hat{y}^*$ for the testing set.

Block Size = 20

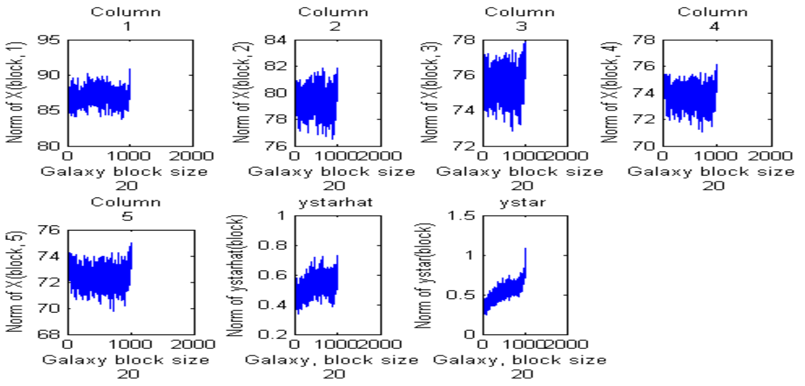
Figure 20: Columns of X^*, y^* and $\hat{y}^*$ for the testing set in blocks of 20.

A slight U-shaped pattern appears in the columns of X^*. A linear correlation seems to exist in y^*, although this has limited usefulness since y^* is not available for characterization purposes.

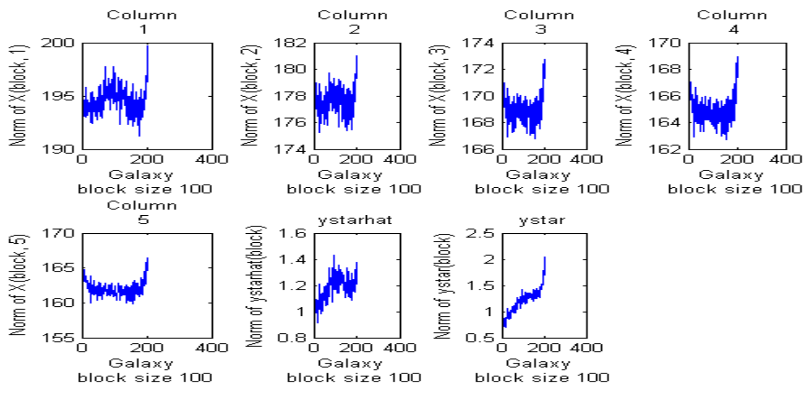

Figure 21: Columns of X^*, y^* and $\hat{y}^*$ for the testing set in blocks of 100.

A stronger pattern exists here, especially in columns 3, 4, and 5.

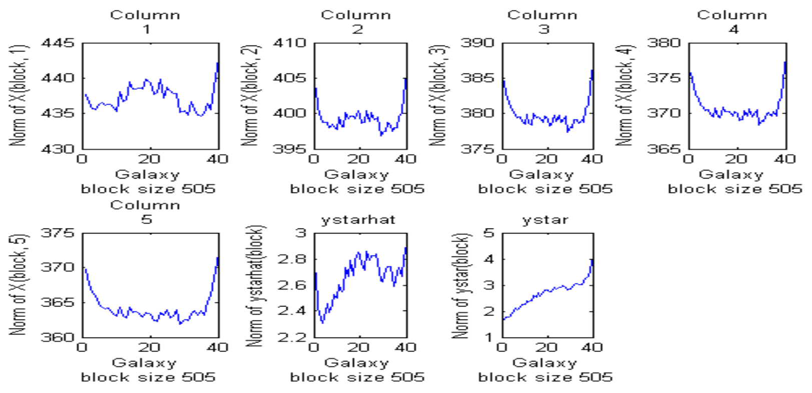

Figure 22: Columns of X^*, y^* and $\hat{y}^*$ for the testing set in blocks of 505.

The norm of a the set of 505 entries which all correspond to the outliers in the fourth column of X^*, for example, is greater than 375 even if we take 20% of the set to be outliers. It may thus be reasonable to take this as a necessary condition for a set which contains only outliers. Moreover, it seems reasonable that for a set which contains a percentage α% of non-outliers, the norm must be at least $\frac{(1-\alpha)375+(\alpha)371.5}{2}$.

13.2 Classifying Outliers

Calculating the outliers of a testing set X^* can be viewed as a classification problem. In other words, given training data X and a vector $\tilde{y}$ with the property that $\tilde{y}(i) = 1$ if x_i is an outlier, and $\tilde{y}(i) = 0$ if x_i is a non-outlier, and similarly with testing data X^* and $\tilde{y}^*$, calculate an approximation of $\tilde{y}^*$. This can be recast as a regression problem by taking $\tilde{y} = |y - \hat{y}|$, where $\hat{y}$ is an approximation of y, and $\tilde{y}^* = |y^* - \hat{y}^*|$. To calculate $\tilde{y}$ it is thus necessary to take a subset of the training data, and use one subset as training data to approximate another as testing data. This is equivalent to solving the classification problem, because it is the index of $\tilde{y}^*$ that is used to define outliers and non-outliers. Therefore, the V Formulation can be used indirectly to classify the outliers. First, approximate $\tilde{y}^*$ as γ. It then follows that the index of the sorted γ can be used to predict a set of outliers in X^*(those highest in the index), and after these have been removed the original regression problem may be solved. This results in a significant reduction in rms error if 10% of the data is removed, as shown in figure 13. This method may be further refined by recursively finding outliers in the approximated set of outliers, adding these back in, finding outliers in that set and taking them back out, and so on.

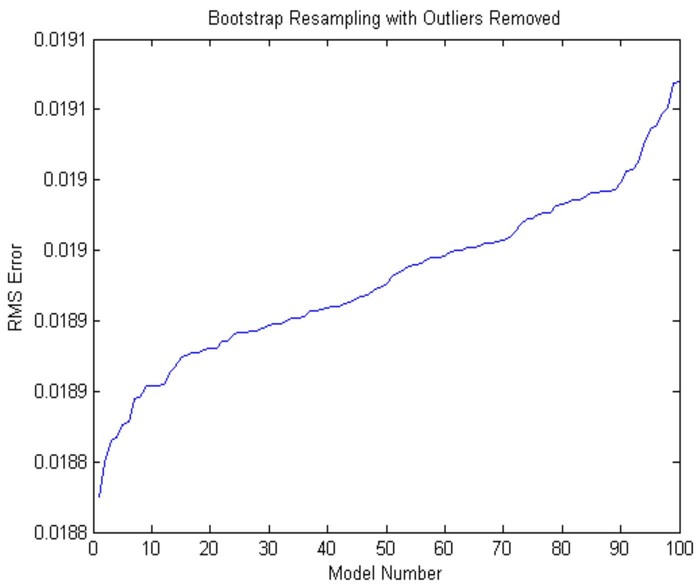

Figure 23: Bootstrap resampling with outliers removed.

14 Appendix - Code

Code for this project is based on the code used in Rasmussen and Williams text [Rasmussen and William, 2006]. This code is available at: www.gaussianprocess.org/gpml/code/matlab/doc/.

We wrote a number of additional functions to supplement the Rasmussen and Williams code. Here we briefly describe each function. Detailed description of each function is contained in the code. Contact Leslie Foster (foster@math.sjsu.edu) for the actual code.

Note that a number of the functions (gprSRPP0, gprSRPP1, gprSRPP1b, gprSRPP2, gprSRPP3, and gprSRPP4) are written to be compatible, including useage and parameter lists, with Rasmussen and Williams function gprSRPP which does low rank approximation. Any call to Rasmussen and William function gprSRPP can be replaced by a call to one of these functions. There are additional options available in our code that are not part of Rasmussen and Williams code. However one one does not need to use these.

To use the above code in Matlab one needs to add to the Matlab path the location of the above files (a zip file with the code will be sent upon request to to foster@math.sjsu.edu) and also add to the Matlab path the location of the gpml-matlab\gpml folder of Rasmussen and Williams code (available at www.gaussianprocess.org/gpml/code/matlab/doc/). Also one may wish to load the data file redshift_data mention below.

The SR method: gprSRPP0.m – Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set specified by INDEX. gprSRPP0 is a minor change from the gprsRPP code provided by Rasmussen and Williams. gprSRPP0 avoids repeating certain calculations.

The V Method without pivoting: gprSRPP1.m - Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set specified by INDEX. This version uses a variation of the Peters-Wilkinson approach for solving least squares problems to increase the numerical stability of the algorithm.

The V method without pivoting (implemented using partial Cholesky factorization without pivoting): gprSRPP1b.m - Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set specified by INDEX. This version uses a variation of the Peters-Wilkinson approach for solving least squares problems to increase the numerical stability of the algorithm. The version uses a partial Cholesky factorization to factor the columns of the covariance matrix specified by INDEX. This code is slower than gprSRPP1 but requires less memory.

The SRQR method: gprSRPP2.m - Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set specified by INDEX. This version uses a QR factorization to increase the numerical stability of the algorithm.

The V method with pivoting: gprSRPP3.m - Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set selected by a partial Cholesky decomposition. This version uses a variation of the Peters-Wilkinson approach for solving least squares problems (and partial Cholesky factorization) to increase the numerical stability of the algorithm.

The V method with QR factorization (VQ): gprSRPP4.m - Carries out approximate Gaussian process regression prediction using the subset of regressors (SR) or projected process approximation (PP) and the active set selected by a partial Cholesky decomposition. This version uses a QR factorization (and partial Cholesky) to increase the numerical stability of the algorithm.

Partial Cholesky: chol_part.m –This function does a partial cholesky decomposition, with pivoting or optionally without pivoting, of the kernel matrix K defined by a covariance function that follows Rasmussen and Williams style for covariance functions.

Quadratic Covariance function: covQUADiso.m – Quadratic covariance function with isotropic distance measure. Written in the style of Rasmussen and Williams covariance functions.

Quadratic Covariance function for Automatic Relevance Determination: covQUADard.m – Quadratic covariance function with Automatic Relevance Determination (ARD) distance measure. Written in the style of Rasmussen and Williams covariance functions.

Cubic Covariance function: covCUBICiso.m – Cubic covariance function with isotropic distance measure. Written in the style of Rasmussen and Williams covariance functions.

Cubic Covariance function for Automatic Relevance Determination: covCUBICard.m – Cubic covariance function with Automatic Relevance Determination (ARD) distance measure. Written in the style of Rasmussen and Williams covariance functions.

Polynomial Covariance function: covPOLYiso.m – Polynomial covariance function with isotropic distance measure. Default degree is 4. Written in the style of Rasmussen and Williams covariance functions.

Polynomial Covariance function for Automatic Relevance Determination: covPOLYard.m – Polynomial covariance function with Automatic

Relevance Determination (ARD) distance measure. Default degree is 4. Written in the style of Rasmussen and Williams covariance functions.

Some utilities:

sq_dist.m - a function to compute a matrix of all pairwise squared distances between two sets of vectors, stored in the columns of the two matrices, a (of size D by n) and b (of size D by m). If only a single argument is given or the second matrix is empty, the missing matrix is taken to be identical to the first. NOTE: This version of sq_dist calls a FORTRAN mex file and will be more efficient than the sq_dist.m file supplied with Rasmussen and Williams code.

sq_distf.dll - a dynamic link library file created by Matlab's mex utility. It is called by sq_dist.m. This utility works for Matlab 7.0, 7.1 and 7.2 but has not been successfully tested for Matlab 7.4.

A data file:

redshift_data.mat – contains the training set data X and y and the testing set data Xtest and ytest for the redshift problem. Also contain the logarithms of the hyperparameters that are listed in Section 10.

Demonstration files:

Several demonstration files will be distributed with the code upon request. These are not currently available but will be ready in August. However note that the functions gprSRPP0, gprSRPP1, gprSRPP1b, gprSRPP2, gprSRPP3, gprSRPP4 are compatible with Rasmussen and Williams gprSRPP. Therefore these new files can be used in an identical manner to gprSRPP. Rasmussen and Williams (www.gaussianprocess.org/gpml/code/matlab/doc/) has examples of the use of gprSRPP.

References

A. Bjorck. Numerical Methods for Least Squares Problems. SIAM Press. Philadelphia. 1996.

Bem Cayco, Wasin So, Miranda Braselton, Kelley Cartwright, Michael Hurley, Maheen Khan, Miguel Rodriguez, David Shao, Jason Smith, Jimmy Ying, and Genti Zaimi. CAMCOS Project Fall 2006 Improved Linear Algebra Methods for Redshift Computation from Limited Spectrum Data Final Report. For availability see http://www.math.sjsu.edu/camcos/. 2006.

S. Geman and D. Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". IEEE Transactions on Pattern Analysis and Machine Intelligence, 6:721-741, 1984.

G. Golub and C. Van Loan. Matrix Computations, third edition. John Hopkins Press. Baltimore and London. 1996.

G. Forsythe and C. Moler. Computer Solutions of Linear Algebraic Systems. Prentice Hall, Englewood Cliffs, N. J., 1967.

Higham, N. J. "Accuracy and Stability of Numerical Algorithms Second Edition." Manchester, England. 2002.

Nelder, J. A. and Mead, R. "A Simplex Method for Function Minimization." Comput. J. 7, 308-313, 1965.

Rasmussen, Carl E. and Williams, Christopher K. I. "Gaussian Processes for Machine Learning." MIT Press, Cambridge, Massachusetts. 2006.

Srivastava, A.N and Way, M.J. "Novel Methods for Predicting Photometric Redshifts from Broadband Data Using Virtual Sensors." The Astrophysical Journal. 647: 102:115, 2006 August 8.

L. Trefethen and D. Bau. Numerical Linear Algebra. SIAM Press, Philadelphia. 1997.

G. Wahba, Spline Models for Observation Data, SIAM Press, Philadelphia, 1990.