

# An Online Condition Number Query System \*

Dianwei Han,<sup>†</sup> Jun Zhang<sup>‡</sup>

Department of Computer Science, University of Kentucky,  
Lexington, KY 40506-0046, USA

Shuting Xu<sup>§</sup>

Department of Computer Information Systems, Virginia State University,  
Petersburg, VA 23806, USA

May 5, 2006

## Abstract

Condition number of a matrix is an important measure in numerical analysis and linear algebra. It is a measure of stability or sensitivity of a matrix to numerical operations. However, the direct computation of the condition number of a matrix is very expensive in terms of CPU and memory cost, and becomes prohibitive for large size matrices. We propose to use data mining techniques to estimate the condition number of a given sparse matrix. In particular, we will use Support Vector Machine (SVM) to predict the condition numbers. That is, after computing the sparsity pattern features of a matrix, we use support vector regression (SVR) to predict its condition number. This Online Condition Number Query System (CONQS) allows the users to submit their matrices and to obtain predicted condition numbers for their matrices. The accuracy of our prediction methods may not be as precise as the direct computation methods, but it is much faster. Our online system accepts matrices in Harwell-Boeing (HB) format and in standard MATLAB format. The users can use our system to estimate the condition number of their matrices through LAPACK software as well.

## 1 Introduction

The condition number of a matrix is one of many important matrix features and is widely used in many areas, such as in numerical analysis and linear algebra theory. The condition number  $k(A)$  of a nonsingular matrix  $A$  with respect to a matrix norm is formally defined as  $\|A\| \cdot \|A^{-1}\|$  [6]. The condition number corresponding to the Frobenius norm will

---

\*Technical Report No. 458-06, Department of Computer Science, University of Kentucky, Lexington, KY, 2006. This research was supported in part by the Kentucky Science and Engineering Foundation under the grant KSEF-148-502-05-132.

<sup>†</sup>dianweih@csr.uky.edu

<sup>‡</sup>E-mail: jzhang@cs.uky.edu, URL: <http://www.cs.uky.edu/~jzhang>.

<sup>§</sup>sxu@vsu.edu.

be denoted by  $k_F(A)$  and that corresponding to the  $p$ -norm will be denoted by  $k_p(A)$ . In this paper, we only consider the condition numbers corresponding to the 1-norm  $k_1(A)$ . If  $k(A)$  is relatively small, the matrix  $A$  is called a *well-conditioned matrix*; but if  $k(A)$  is very large, it is an *ill-conditioned matrix*. Many application scientists and engineers may want to know if their matrix is well-conditioned or ill-conditioned in order to choose appropriate numerical methods. In numerical analysis, the condition number is basically a measure of stability or sensitivity of a matrix to numerical operations. For example, the condition number associated with the linear equation  $Ax = b$  gives a bound on how inaccurate the solution will be after the numerical solution. Condition number can also be used to predict the convergence of iterative methods. For instance, for the conjugate gradient (CG) method, the error can be bounded in terms of  $k_2(M^{-1}A)$ [2], where  $M$  is a preconditioner. If  $A$  is symmetric positive definite, then for CG with a symmetric positive definite preconditioner  $M$ , it is well-known that:  $\|\hat{x}^{(i)} - x\|_A \leq 2\alpha \|\hat{x}^{(0)} - x\|_A$ , where  $\alpha = (\sqrt{k_2(M^{-1}A)} - 1)/(\sqrt{k_2(M^{-1}A)} + 1)$ , and  $\hat{x}^{(i)}$  is the  $i$ th step approximate solution [10].

There are several ways to obtain a condition number of a matrix. The direct method is to compute  $A^{-1}$  first and multiply its norm with the norm of  $A$ . However, computing  $k(A)$  for even the simplest matrix norms is three times as expensive as solving  $Ax = b$  in the first place [1]. Another method uses a less expensive algorithm to estimate the condition number. There are some estimation algorithms in literature [4, 5, 8]. For example, LAPACK uses subroutine SGECON to compute the condition number. Its time cost is  $O(n^2)$  extra beyond the  $O(n^3)$  cost of solving  $Ax = b$ , where  $n$  is the dimension of  $A$ . If the size of the matrix is relatively large the process of solving  $Ax = b$  could be a problem on many desktop computers. In most cases, the estimated condition number is within a factor of 10 of the true condition number, but there exist some counter-examples with large estimation errors. MATLAB uses LINPACK for computing the reciprocal of  $k_1(A)$  and uses Higham's modification of Hager's method to estimate  $k_1(A)$  [8].

We propose a completely new approach to estimating the condition number of a sparse matrix. We predict the condition number from the matrix sparsity features using data mining techniques instead of direct computation or estimation. Given a sparse matrix, we first extract the matrix features based on the locations and values of the nonzero entries, then use some machine learning methods to predict its condition number. We build our on line condition number query system (OCNQS) with the following considerations: (1) Matrix features for prediction. We associate the condition number of a matrix with some features describing the structure and value distribution of the matrix. (2) Specific prediction method. We use SVM regression (SVR) [14] and test different kernels. (3) Highly efficient system. As more and more matrices enter the system, it is inefficient to retrain the SVR every time a new matrix is added, we will rebuild the training model periodically to reduce training time while maintaining the prediction accuracy of the system. Although the condition number predicted is not as precise as the above-mentioned direct computation methods, it has much smaller time and memory cost, especially for large size matrices, which is suitable for online matrix condition number query. Furthermore, many

users are only interested in knowing whether a matrix is *well-conditioned* or *ill-conditioned*, or the approximate order of the condition number. In these situations, response time and reliability are at least as important as accuracy.

The rest of this paper is organized as follows: In Section 2, we introduce the matrix features used to predict the condition number. Section 3 discusses individual components of OCNQS. The experimental results are presented in Section 4, followed by some concluding remarks in Section 5.

## 2 Matrix Feature Extraction

The features of a matrix are computed according to its sparsity pattern, i.e., the locations and values of its nonzero entries, and are directly related to the precision of the prediction system. We will compute the features of a matrix first and then use such information to predict its condition number [18]. We extract 68 features in our project. Here, we just list five major types: structure, value, bandwidth, diagonal, and other related statistics. For more detailed description on the matrix features, please see [18].

### 2.1 Structure

This group of features describe the distribution of nonzero elements of a matrix. For example, we consider the sparsity rate: the number of nonzero elements divided by the number of all elements of the whole matrix, the lower nonzero rate: the number of nonzero in the lower part divided by the number of all elements, the diagonal nonzero rate: the number of nonzero in the diagonal part divided by the number of all elements, and the upper nonzero rate: the number of nonzero in the upper part divided by the number of all elements. Sometimes we also want to know the total number of nonzero diagonals, i.e., the number of diagonals which have at least one nonzero element among the  $2n - 1$  diagonals of the matrix. The attribute *symmetric* measures whether a matrix is symmetric, i.e.,  $A = A^T$ . The attribute *relsymm* describes the relative symmetric rate of a matrix. It is the ratio of the number of elements that matches divided by the number of nonzero (*nnz*). An element  $a(i, j)$  in the matrix  $A$  is *matched* if it satisfies the following condition: if  $a(i, j)$  is nonzero then  $a(j, i)$  is nonzero.

### 2.2 Value

This group of features show the value distribution of a matrix. We compute the average value of all nonzero entries and its standard deviation. We take the average of the main diagonal entries and the standard deviation, the average of the upper triangular entries and the standard deviation, as well as the average of the lower triangular entries and the standard deviation. Additionally, the matrix norms are very important attributes as well. For example, the one norm, the infinity norm, and the Frobenius norm of a matrix are computed. This group also includes the minimum of the sum of the columns, and the minimum of the sum of the rows.

### 2.3 Diagonal

All the features in this category are diagonal related. They include the percentage of weakly diagonally dominant columns and the percentage of weakly diagonally dominant rows. We compute the total number of non-void diagonals, the maximum and the minimum values of the diagonal elements. We include the average distance from each entry to the diagonal and the standard deviation. Furthermore, we compute the average of the difference from each of the entry to its diagonal value and the standard deviation, etc.

### 2.4 Bandwidth

This group of features describe the bandwidth of a matrix. For example, the lower bandwidth of a matrix is defined as the largest value of  $i - j$ , where  $a(i, j)$  is nonzero, the upper bandwidth of a matrix is defined as the largest value of  $j - i$ . The maximum bandwidth is defined as  $\max(\max(j) - \min(j))$ , where  $a(i, j)$  is nonzero. The average bandwidth is defined as the average width of all columns.

## 3 Components of OCNQS

We will discuss two issues concerning the OCNQS implementation. First of all, we will briefly introduce SVM regression technique. Then, we will describe how our system is implemented and how it works.

### 3.1 SVM regression

SVM was developed at AT&T Bell Laboratories by Vapnik and co-workers [15], and has been gaining popularity due to its many attractive features. Because of this industrial context, SV research has had a sound orientation towards real-world applications [14]. SVMs were developed to solve the classification problem, but recently they have been extended to the domain of regression problems [16]. SVM regression (SVR) is introduced with an alternative loss function [13]. There mainly are four loss functions that can be used here: Gaussian, Laplacian, Huber's robust, and  $\varepsilon$ -insensitive [14]. Quadratic (Gaussian) corresponds to the conventional least squares error criterion. Laplacian loss function is less sensitive to outliers than the Gaussian loss function. Huber's robust has optimal properties when the distribution of the data is unknown [7]. SVM regression using the  $\varepsilon$ -insensitive loss function is called  $\varepsilon$ -SV regression [15]. In  $\varepsilon$ -SV regression, the goal is to find a function  $f(x)$  that has at most  $\varepsilon$  deviation from the actually obtained targets  $y_i$  for all the training data, and at the same time is as flat as possible [14]. We can visualize this as a tube of size  $2\varepsilon$  around  $f(x)$  and data fall out of the tube are errors [3].

Suppose a linear function  $f(x)$  is of the form:  $f(x) = \langle w, x \rangle + b, w \in X, b \in R$ , where  $\langle \cdot, \cdot \rangle$  denotes the dot product in  $R$ . Flatness in this case means that one seeks small  $w$ . One way to achieve this objective is to minimize  $\|w\|^2$ . We can state this

problem formally as a convex optimization:

$$\begin{aligned} & \text{minimize } \frac{1}{2} \| w \|^2 \\ & \text{subject to } \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases} \end{aligned}$$

It implies that such a function  $f$  actually exists with  $\varepsilon$  precision, and the convex optimization is feasible. However, sometimes, we may allow some errors. In this case, one can introduce slack variables  $\xi_i$  and  $\xi_i^*$ .  $\xi_i$  computes the error for underestimating the function while  $\xi_i^*$  computes the error for overestimating the function. Then we have the following convex optimization problem [15]:

$$\begin{aligned} & \text{minimize } \frac{1}{2} \| w \|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) \\ & \text{subject to } \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \\ C > 0 \end{cases} \end{aligned}$$

where  $C$  is a user-selected regularization parameter. Using a standard dualization method, we can get the following function [14]:

$$f(x) = \sum_{i=1}^l (\alpha_i - \alpha_i^*) \langle x_i, x \rangle + b. \quad (1)$$

where  $0 \leq \alpha_i, \alpha_i^* \leq C$ .

For the nonlinear case, we apply a mapping  $\Phi : X \rightarrow F$  to map the input space into some feature space  $F$ . Here we use a kernel function,  $K(x, x_i) = \langle \Phi(x), \Phi(x_i) \rangle$ , which is a symmetric function and satisfies the Mercer's condition. We substitute  $K(x, x_i)$  for the dot product, which maps the input space into some reproduced kernel feature space. Then Equation (1) can be rewritten as:

$$f(x) = \sum_{i=1}^l (\alpha_i - \alpha_i^*) K(x, x_i) + b. \quad (2)$$

In our project, we use a RBF kernel function:

$$K(x, x_i) = e^{-\frac{\|x - x_i\|^2}{2\sigma^2}}.$$

### 3.2 OCNQS implementation

We build up the OCNQS to allow the web users to submit their matrices and predict the condition number of the matrix. Basically, We prefer the users to submit their matrices

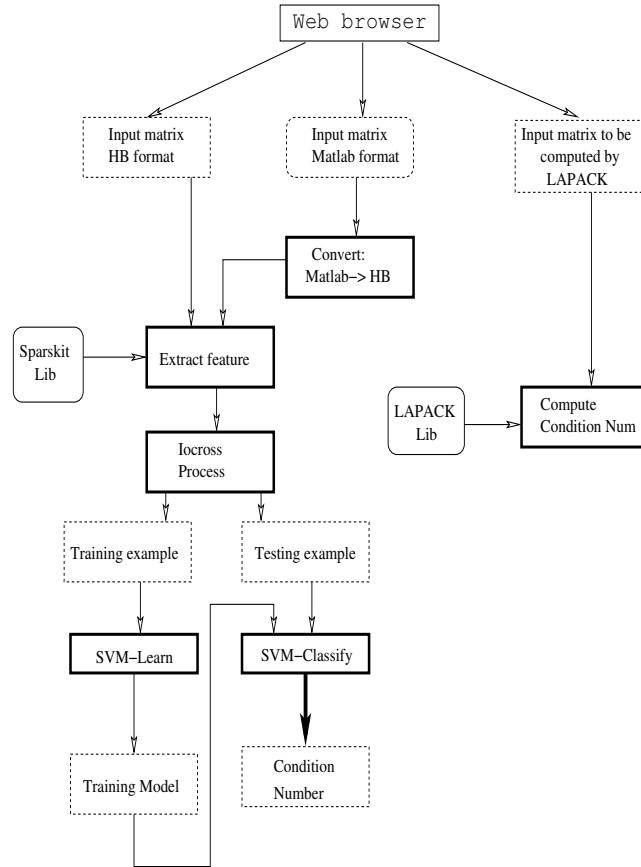


Figure 1: The structure and individual components of OCNQS.

in the Harwell-Boeing (HB) format. We can also convert a matrix from the MATLAB format to the HB format. However, the matrix has to be submitted through a different input box. We illustrate the individual components of OCNQS in Fig. 1.

In the OCNQS system, We provide two entries for the web clients. One is the interface to predict the condition number using our SVM technique, regardless of the input matrix format. The other is the interface to directly compute the condition number using a LAPACK routine.

### 3.2.1 Predict the condition number

From Fig. 1, we can see that if the given matrix is in the HB format, we directly apply our program to extract the features of this matrix. Otherwise, we first need to run a program to convert the matrix from the MATLAB format to the HB format. At this stage, we call some SPARSKIT libraries to extract the matrix features. Then, we run IOcross program to generate the training set and the testing set. Next, We run SVM-learn to train the training set to get the training model. Then we input the testing set and the training model to the SVM-Classify program to compute our prediction result.

### 3.2.2 Periodically update the training model

Instead of using an incremental algorithm as used by Ma *et al.* [11], we just periodically update the training model. In their method, they first defined three vectors:  $E$ ,  $S$ , and  $R$ ; the  $E$  set: error support vectors; the  $S$  set: margin support vectors; and the  $R$  set: remaining samples. Their algorithm basically is to find the appropriate Kuhn-Tucker (KT) conditions for the new data by modifying its influence ( $\beta$ ) in the regression function while maintaining consistency in the KT conditions for the rest of the data used for learning. This algorithm is good and always converges. We take 319 matrices as our initial database and we take the entire database as the training set. Our algorithm runs SVM-learn of SVM<sup>light</sup> code to get the training model and then takes the user input matrix as testing set and runs SVM-Classify program to get the prediction result. The new data will become a new part of the training set and will participate in the next training phase. In order to reduce the time cost during the training phase, we do not produce new training model every time. As the number of input matrices increases to a certain level, we may retrain the training model and update it periodically. We may choose to use an incremental algorithm when our database is very large and our system becomes stable.

### 3.2.3 Directly compute the condition number

We also allow the users to submit their matrices to compute the condition number directly. But if the matrix size is too large, i.e.,  $n \geq 1000$ , the computation time will be so long that the network connection will be disconnected. Sometimes, it will take up to several hours to finish the computation. Here, we just use LAPACK software to compute the condition number.

### 3.2.4 Decision strategy

When to use the prediction method and when to use the direct computation method partly depends on the matrix characteristics. If the matrix is of small size we prefer to directly compute its condition number, because we can get a very accurate condition number within a short period of time. Unfortunately, most matrices encountered in practical applications are relatively large or huge. Condition number is theoretically important, but it does not have much practical use in real matrix computation because of its prohibitively high computational cost (CPU time plus storage cost). Therefore, if the matrix is very large we are better off to predict its condition number because we can get an approximate condition number without a high time cost.

## 4 Experiments and Results

We conduct some experiments to test the performance of our OCNQS software that predicts the condition number of a given matrix. We focus on accuracy and response time of the condition number prediction methods. We use SVM<sup>light</sup> [9] for SVM regression.

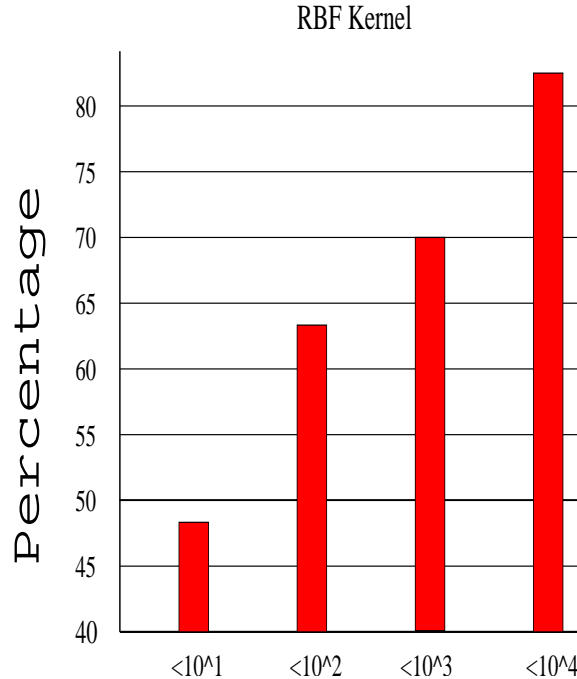


Figure 2: Accuracy of the prediction method.

There are 319 matrices from the Matrix Market [12] used in our experiments. We use up to 66 matrix attributes, most of which are described in Section 2. The experiments are conducted on a SunBlade 100 workstation.

#### 4.1 Accuracy

We test how accurate the predicted condition numbers are, compared with the directly computed condition numbers. We use 5-fold cross validation and choose the parameter  $C$  in SVR to be 10000 according to the results of the 5-fold cross validation. The RBF kernel uses  $\sigma = 0.1$ .

Fig. 2 shows the percentage of all matrices for which the relative differences between the computed values and predicted values of the condition number are within  $10^1, 10^2, 10^3$ , and  $10^4$ . 70% of the matrices have relative differences smaller than  $10^3$ . 82.5% of the matrices have relative differences smaller than  $10^4$ .

#### 4.2 Response time

Given a matrix, the time used to get the condition number is referred as to the response time. When using LAPACK, we take the time for computing the condition number by LAPACK routines as the response time. The response time for the prediction method includes the time to extract the matrix features and the time for prediction.

The prediction method is especially advantageous in response time for large size matrices. Table 1 shows that the average LAPACK response time for the 141 matrices

Table 1: Average response time for large size matrices (in seconds).

Size	Number of Matrix	<i>LAPACK</i>	<i>Prediction</i>
$\geq 1000$	141	283.86	20.98
$\geq 2000$	99	396.36	22.89

with size greater than 1000 is around 4.7 minutes and the prediction method only needs 21 seconds. For the 99 matrices with size greater than 2000, the LAPACK computation takes an average response time around 6.6 minutes while the prediction method just takes 23 seconds. The later is almost 17 times faster. In our experiments, matrix size greater than 1000 is considered to be large. As LAPACK will run out of memory on our machines with the matrices of size larger than 20000, we can only test matrices whose size is under this number for comparison. Basically, we encourage users to use LAPACK to directly compute condition number when the matrix size is under 1000, because the computation time is acceptable and the computed condition number is more accurate than that from the prediction method.

### 4.3 Very large matrices and very small matrices

We conduct experiments on several very large size matrices and very small size matrices. The following table shows the results for some large size matrices.

Table 2: Performance comparison for some very large matrices (in seconds).

Matrix Name	<i>Size</i>	<i>nnz</i>	<i>LAPACK(time)</i>	<i>Prediction(time)</i>
add20	2395	13151	8955.5	8
af23560	23560	460598	N/A	61
cry10000	10000	49699	2517.5	13
fidap015	6867	96421	318.5	22
gemat11	4929	33108	220.24	9
lns_3937	3937	25407	3251.8	8

In table 2, we show some very interesting results. For predicting the condition number of the matrix add20, using prediction method can be 1000 times faster than using LAPACK. LAPACK used almost one hour to compute the condition number of lns\_3937, the prediction method just needed 8 seconds. Due to the size of the matrix af23560, the LAPACK program on our SunBlade100 machine failed to compute its condition number. We can get it by the prediction method successfully. The significantly reduced response time is our motivation for advocating the prediction method. When we conduct some experiments on very small matrices, the response time and accuracy of the LAPACK method

are better than the prediction method. For example, for a trivial matrix

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 2 & 3 & 1 & 0 \\ 0 & 0 & 12 & 9 \\ 2 & 0 & 0 & 8 \end{bmatrix}$$

the condition number predicted by the prediction method is the same as that computed by LAPACK (infinity). But the response time used by LAPACK is 2 seconds while the response time of the prediction method is 12 seconds.

## 5 Concluding Remarks

We proposed to use the features of sparse matrices to predict the condition number of a given sparse matrix. We use SVR in our system. The experimental results show that our prediction method works best when the input matrix size is very large and the accuracy requirement of the condition number is not very high. The accuracy of the prediction method is low compared with direct computation, but it is good enough for those people who only want to know if a matrix is *well-conditioned* or *ill-conditioned*. The advantage of the prediction method is that the response time is very short [18]. It is a desirable utility component to be used in our Intelligence Preconditioner Recommendation System (IPRS), where we use condition number as one of the matrix features to help estimate the solvability of a matrix by a preconditioned iterative method [19]. OCNQS can be considered as one part of the IPRS.

## References

- [1] Z. Bai, J. Demmel, A. Mckenkkey. *On computing condition numbers for the nonsymmetric eigenproblem*. ACM Tran. Math. Soft., 19:202-223 (1993).
- [2] R. Barrett, M. Berry, T.F. Chan, *et al.* *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 2nd Ed., 1994.
- [3] K.P. Bennett, C. Campbell. *Support vector machines: hype or hallelujah?* SIGKDD Explorations, 2:1-13 (2000).
- [4] C. Bischof. *Incremental condition estimation*. SIAM J. Matrix Anal. Appl., 11:31-322 (1990).
- [5] C.H. Bischof, P.T.P. Tang. *Generalizing incremental condition estimation*. J. Numer. Linear Alg. Appl., 1:149-164 (1992).
- [6] G.H. Golub and C.F. Van Loan *Matrix Computation*. John Hopkins Univ. Press, Baltimore, 3rd Ed., 1996.

- [7] S. R. Gunn *Support vector machines for classification and regression*. Technical Report, ISIS-1-98, Department of Electronics and Computer Science, University of Southampton, 1998.
- [8] N.J. Higham. *Fortran codes for estimation the one-norm of a real or complex matrix, with applications to condition estimation*. ACM Tran. Math. Soft., 14:381-396 (1988).
- [9] T. Joachims. *Making large-scale SVM learning practical*, Advances in Kernel Methods Support Vector Learning, B. Schölkopf, C. Burges and A. Smola (ed.), MIT-Press, 1999.
- [10] S. Kaniel. *Estimates for some computational techniques in linear algebra*. Math. Comput., 20:369-378 (1966).
- [11] J. Ma, J. Theiler, S. Perkins. *Accurate on-line support vector regression*. Neur. Comput., 15:2683-2703 (2003).
- [12] <http://math.nist.gov/MatrixMarket/>
- [13] A.J. Smola. *Regression Estimation with Support Vector Learning Machines*. Master's Thesis, Technische Universität München, 1996.
- [14] A.J. Smola, B. Schölkopf. *A tutorial on support vector regression*. NeuroCOLT Technical Report Series, NC2-TR-1998-030, 1998.
- [15] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
- [16] V.N. Vapnik, S. Golowich, A. Smola. *Support vector method for function approximation, regression estimation, and signal processing*. Advances in Neural Information Processing System 9. MIT Press, Cambridge, 1997.
- [17] V.N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.
- [18] S. Xu, J. Zhang. *A new data mining approach to predicting matrix condition numbers*. Communications in Information and Systems, 4:325-340 (2004).
- [19] S. Xu, J. Zhang. *A data mining approach to matrix preconditioning problem*. In *Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD'05)*, pp. 49-57, Newport Beach, CA, 2005.