

CS294 Final Project: An Application of Non-parametric Regression

Ryan Feeley

1 Introduction

The dominant theme of cs294 is learning patterns or relations from data. These may be simple binary relations suitable for classification algorithms, or more complicated relationships, such as functional dependences. This report concerns the later. For my final project I have examined several regression algorithms (including k-nearest neighbor, support vector machine, and Gaussian random process) in the context of a problem relevant to my research.

Due in part to their generality and the ease in which they may incorporate prior knowledge, many models arising from the natural sciences (chemistry, molecular biology, and the like) are mechanistic in nature, and yield parametric models when expressed mathematically. When such a model is proposed, routine questions include “is it consistent with existing observations?” and “what does it predict about quantities that have not been observed?”. In a manner that will be made clear in the background section, both inquiries can be formulated as questions about the set of parameter values that are consistent with the existing observations.

This set of parameter values is described implicitly by three factors: the observed values (data), a description of the uncertainty in each observation, and functions representing the dependence of the observed quantities on the model parameters. In many situations, it is this third component that makes the analysis of a hypothesized model difficult. Often there are no closed form expressions for such functions—they may only be evaluated (with some algorithm-dependent numerical error) through a computer simulation. Furthermore, such simulations may be expensive—possibly taking hours or days (e.g., flame simulations).

For my final project in cs294, I have first analyzed the ability of several standard machine learning algorithms to represent a more mild function (≈ 0.5 second evaluation time) that is generated by a biological signaling model. The contents of the report are as follows. Section 2 contains background information. How experimental observations carve out a set of consistent parameter values is formulated in a simple deterministic setting. Additionally this section contains a description of the function that I investigated using regression algorithms covered in class. Section 3 contains a description of the goals and objectives of the research project. The general methodology I used to perform the regressions is outlined in section 4, Further context specific details and the precise forms of the learning algorithms that I employed (and their associated parameters) are described in section 5. Results and discussion are provided in section 5, and the final section provides concluding remarks.

2 Background

This section briefly introduces a setting in which accurate function approximation is important. This serves to motivate the goals of this research project. The section half of the section describes the function studied in the work.

Consider a system and an associated parametric model that is hypothesized to describe the system’s behavior. The parameter vector x is assumed to lie in a bounded n -dimensional hyperrectangle H . Suppose an observable with scalar values y , has been measured experimentally.

Let d denote the corresponding measured value (data). The measurement error $|y - d|$ is assumed to be bounded above by a known scalar u . Restricted to the conditions of the experiment, the parametrized model for the system behavior induces a parametrized model for the observable, $M : \mathbb{R}^n \rightarrow \mathbb{R}$. Combining this model with the experimental observation constraints the parameter vector to satisfy $|M(x) - d| \leq u$. Consequently the experiment carves out the set $F = \{x \in H : |M(x) - d| \leq u\}$ as the set of parameter consistent with the observation. (In the general case where multiple experiments have been performed, the intersection of the corresponding sets F is the object of study.)

In this deterministic framework, F is an exact characterization of the uncertainty in the parameter vector. Determining whether or not F is empty is useful for model validation, and can be decided by examining the sign of the optimal solution of

$$\min_{x \in H, \gamma} \gamma \quad \text{subj. to } |M(x) - d| \leq u - \gamma. \quad (1)$$

Unfortunately, unless M is special (e.g., convex), even determining if F is nonempty is an instance of an NP-hard problem.

This leads us to consider which classes of functions M would lead to a set F that is amenable to efficient algorithms. For nonlinear functions very little is available. However, if M were quadratic, the S-procedure can be used to formulate a relaxation of (1) as a semidefinite program. In the general case suppose Q is an indefinite quadratic approximation to M and b is a uniform bound on the approximation error over H . Then

$$F_Q = \{x \in H : |Q(x) - d| \leq u + b\} \supset F. \quad (2)$$

If the S-procedure suffices to show F_Q is empty, then clearly F is as well. This would indicate that hypothesized model for the system's behavior has been invalidated by the observational data.

Although the above background is necessarily terse, I hope it has motivated why learning the behavior of a complicated and expensive to evaluation function M is useful, and further why it may be desirable to have a quadratic approximation to this behavior, together with a uniform bound on the approximation error. In the remainder of this report, I will document my efforts to learn the behavior of a particular function M , which I describe below.

The function M is derived from hypothesized model for the ligand induced activation of G-protein in yeast. A simplified model for this process is diagrammed in figure 1. The model includes formation of a ligand-receptor complex RL, and subsequent activation of a subunit of the heterotrimeric G-protein complex. This later reaction is catalyzed by the bound receptor RL.

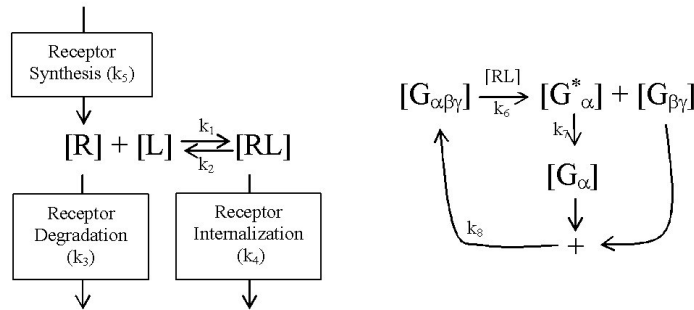


Figure 1: Schematic of the yeast signaling pathway

A mathematical description for the dynamics of the system is provided by the following

state-space system of equations.

$$\begin{aligned}\dot{x}_1 &= -k_1x_1u + k_2x_2 - k_3x_1 + k_5 \\ \dot{x}_2 &= k_1x_1u - k_2x_2 - k_4x_2 \\ \dot{x}_3 &= -k_6x_2x_3 + k_8(G_t - x_3 - x_4)(G_t - x_3) \\ \dot{x}_4 &= k_6x_2x_3 - k_7x_4 \\ y &= (G_t - x_3)/G_t\end{aligned}$$

The states are $x_1 = [R]$, $x_2 = [RL]$, $x_3 = [G_{\alpha\beta\gamma}]$, $x_4 = [G_{\alpha}^*]$ (activated G-protein subunit). The input u is the concentration of ligand $[L]$, while the output y is the normalized concentration of $G_{\beta\gamma}$. $[G_t]$ is the total concentration of (say) α subunits of the G-protein complex: $[G_t] = [G_{\alpha\beta\gamma}] + [G_{\alpha}^*] + [G_{\alpha}]$ (equivalently, $[G_t] = [G_{\alpha\beta\gamma}] + [G_{\beta\gamma}]$). The initial conditions are $x_1(0) = k_5/k_3$, $x_2(0) = 0$, $x_3(0) = G_t$, $x_4(0) = 0$.

Measurements of the output $y(t = 60s)$ have been obtained experimentally for 8 different input levels (or doses) of ligand L [1]. As part of a model validation exercise, we would like to characterize the dependence of the output for these input levels as a function of the model parameters. For my cs294 project, I have chosen to study the relationship between the parameters and the model predicted output $y(t = 60s)$ for an input level of $u = 10nM$. In the remainder of the work, the symbol M will denote this function mapping \mathbb{R}^n to \mathbb{R} .

The uncertain parameter vector of the above model comprises k_1, \dots, k_8 , and G_t . The model and nominal values of the parameters are taken from [1]. The set H over which M is to be studied is a hyperrectangle formed by assuming an order of magnitude uncertainty in each parameter. This level of uncertainty, while larger than that of citefosbe, is more realistic, based on personal communication with the authors.

3 Problem Statement

Before I began work on this project, I knew that the function M that I have chosen to investigate is not very quadratic. Consequently the function must be evaluated a high ($> 10^5$) number of times in order to determine a near optimal (in sup norm) quadratic approximation to it. My hope with this project is that I would discover that one of the more data-driven regression methods covered in cs294 could determine an accurate representation of M with far fewer function evaluations. Since the regression model can presumably be evaluated much faster than M , this “surrogate” model could be heavily sampled to a degree for which a very near optimal quadratic approximation to it could be ascertained. The resulting quadratic function would then be a near optimal quadratic approximation to M . Additionally, due to the heavy sampling of the surrogate model (and further exploiting that it can be cheaply studied with a local search routine) I would be able to estimate a uniform bound on the difference between M and the quadratic approximation to the surrogate.

To accomplish this goal, I set out to fulfill the following tasks:

- Estimate the best possible fitting error that can be obtained with a direct quadratic approximation to M .
- Quantify the performance of various non-parametric or semi-parametric regression algorithms studied in cs294 applied to evaluations of M . The performance metrics include peak and average error on the training and test datasets, plus an estimate of the peak global (over H) fitting error. Additionally discover how this performance depends on the size of training dataset.
- Determine if a surrogate model developed with the best performing algorithm could be used in the manner described above to realize a good sup-norm quadratic approximation to the original function M .

4 Methodology

The section covers general techniques and methods I used in the course of the project. Ideas that are specific to individual algorithms are described in the section 5.

I began my work on the project by generating 5 million input/output pairs (x_i, y_i) from the function M . This would be infeasible for most functions I encounter in my research. I chose to study a function from this simple yeast model primarily because it could be evaluated rapidly enough to generate this magnitude of samples. The inputs x_i were determined by a Latin hypercube design on the 9-dimensional rectangular domain H . Only a small fraction of these points were used at any one time for either training, validation, or testing. The bulk of these points were used to estimate the global approximation error in the manner described below. Prior to performing the regressions, the points (x_i, y_i) were centered and normalized.

I measured the performance of each regression algorithm for three sizes of training data: 2000, 5000, and 50000 points. A five fold cross-validation using these training sets was used where appropriate to determine relevant parameters. Because the peak error of the approximation was an important performance metric, I used test datasets that were the same size as the training data. I lieu of determining a global bound on the approximation error (which would be extremely difficult) I used as many of the remaining evaluations of the model M as possible (given time constraints) to determine good initial seed points with which to examine $|M(x) - \hat{y}(x)|$ via constrained optimization, where $\hat{y}(x)$ denotes the learned regression function. This provided a (hopefully good) estimate of the global fitting error. In the sequel, the “global error” of an approximation should be taken to mean this estimate.

5 Regression algorithms

In the course of my work on the project, I researched several regression algorithms. Here I describe the ones that seemed promising for the present study.

5.1 Linear regression

The linear regression approach I used to fit a quadratic polynomial to evaluations (x_i, y_i) of M was standard. The only point of interest is that since I determined the polynomial coefficients by fitting in sup-norm, the fitting optimization was a linear program with the number of constraints equal to $2 \times \#$ of samples. Do to memory limitations, I could only employ 500000 samples points to determine a fit. Because the order was fixed, and a linear approximation was woefully inadequate to represent M , there was no need to perform cross-validation to select the fitting order. However, I did find the the performance on validation data was slightly improved if the coefficients in the final linear regression model were obtained by averaging those from several runs with subsets of the training data.

5.2 K-nearest neighbor

Because it is naturally biased near the perimeter of the fitting domain H , I did not expect nearest neighbor regression to perform very well. However, because of its simplicity, I chose to include this method in my study. Additionally, examining nearest neighbor schemes seemed appropriate since they are in some sense on the opposite end of the spectrum from linear regression in what is assumed about the behavior of the function being approximated.

The regression model I used was the obvious one. Given samples of the function M , $(x_i, y_i)_{i=1}^m$, let

$$\hat{y}(x) = y_{i^*}, \quad \text{where } i^* = \operatorname{argmin} \|x - x_i\|_2 \quad (3)$$

I also implemented k-nearest neighbors using the average value of y_i over the k -nearest neighbors to determine \hat{y} .

The value of k was determined through consideration of both the peak error and average error on the withheld blocks of the cross-validation exercise.

5.3 Weighted k-nearest neighbor

As will be seen in the next section where I discuss algorithm performance, both nearest neighbor and k-nearest neighbor performed poorly on this problem. For completeness, I additionally tried a distance weighted k-nearest neighbor regression scheme. I used weights in the form of a Gaussian kernel. The final regression function $\hat{y}(x)$ was defined as the average of

$$y_i \exp\left(-\frac{1}{\beta}(x - x_i)^T P(x - x_i)\right) \quad (4)$$

over the k nearest neighbors, where P is a diagonal matrix whose components sum to one, and β is a scalar parameter. The parameters were determined via a 5-fold cross validation by gridding k and determining β and P by local search, with the objective to minimize the average error over the withheld fold.

I chose to include component-wise weighting of the distances (through the matrix P) because there is no reason the function M derived from the yeast signaling model should not have a strong directional dependence on the parameters.

The nearest neighbor regression methods described in this and the previous subsection 5.2 were implemented in MATLAB using the TSTOOL software package [2].

5.4 ϵ -tube support vector regression

I used a standard implementation of ϵ -support vector regression provided by the MATLAB port of LIB-SVM produced by M. Vogy and colleagues [4]. Best performance was achieved with a Gaussian kernel

$$k(x, x') = e^{-\frac{\|x - x'\|^2}{\beta}}. \quad (5)$$

Values for ϵ , β , and C were determined by 5-fold cross-validation and gridding. Since there is essentially no noise in the data, I expected ϵ to be small.

5.5 Gaussian random process regression

Assessing the performance of Gaussian random process regression (GPR) was my original goal in considering this project. After considering the problem and performing a preliminary literature search, I decided to additionally study the aforementioned methods for completeness. The GPR technique was attractive to me because it is an interpolator. I am performing regression on essentially noise-free evaluations of a deterministic function, so it seems logical that a smooth interpolator would have good performance.

The basic form of this regression function is taken from [3]. Consider the random process

$$\tilde{y}(x) = g(x, \theta_1) + z(x, \theta_2) \quad (6)$$

where g is a deterministic function parametrized by θ_1 and z is a stationary zero mean Gaussian random process whose autocorrelation function is parametrized by θ_2 . The conditional mean function of \tilde{y} given $(x_i, y_i)_{i=1}^m$ becomes the regression function \hat{y} .

I carried out the Gaussian random process regression using the MATLAB Kriging toolbox DACE [5]. This toolbox restricts g to be a 2nd or lower order polynomial and provides several common autocorrelation functions for z .

One limitation of the method that I did not anticipate is the high memory requirements. Because the $m \times m$ correlation matrix of the size m training set must be stored, the training size I could consider was limited to $m = 5000$. I didn't have time to explore any of the more elaborate version of GPR that make various approximations to mitigate this effect.

Preliminary experimentation determined that a linear mean function g was most appropriate. This left selection of the correlation function and associated parameters are the last step. These were chosen via 5-fold cross-validation, with an inner loop used to select the parameters, and an outer loop to select the form of the function.

6 Results and Discussion

As I had expected, the baseline linear regression algorithm which directly sought a sup-norm quadratic approximation to M performed rather poorly. Even when this method was applied to a training set of 200,000 points (which was the largest I could handle with my computational resources) the peak global error was more than twice the peak error on the training data or on an identically sized test dataset. This indicates that in the general case where a great number of function evaluations cannot be afforded, when a quadratic approximation fits poorly, the fitting error on the test set is not a good indicator of the global fitting error. Table 1 contains information showing the quality of the approximations for training datasets ranging from 2,000 to 200,000 points. To place these numbers in context, the swing of M over H is approximately 0.96. The average error is defined to be the average value of $|M(x_i) - \hat{y}(x_i)|$ over the relevant dataset while the interval displaying the peak error contains the minimum and maximum of this quantity over the dataset. The global error numbers are estimates of the extreme values of $M(x) - \hat{y}(x)$ over H .

Training size	Training error		Test error		Global error
	avg.	peak	avg.	peak	
2×10^3	0.0413	[-0.137, 0.115]	0.0417	[-0.140, 0.151]	[-0.325, 0.302]
5×10^3	0.0383	[-0.131, 0.124]	0.0380	[-0.171, 0.163]	[-0.389, 0.278]
5×10^4	0.0467	[-0.148, 0.145]	0.0471	[-0.165, 0.162]	[-0.375, 0.216]
2×10^5	0.0501	[-0.156, 0.157]	0.0530	[-0.169, 0.1773]	[-.300, .234]

Table 1: Data on the linear regression which fit a quadratic polynomial to the training data in sup-norm. The peak error on the training set in the last row of the table reveals that such a model will never achieve a uniform error bound smaller than 0.16. The best uniform error achievable by this model class lies somewhere between this value and 0.3.

The nearest-neighbor based algorithms performed even worse than linear regression. This surprised me. Even a training dataset of 50,000 points led to a regression model with a very large global error. Table 2 contains information showing the quality of the nearest neighbor approximations for training datasets of 2000, 5000, and 50000 points. The sample standard deviation of the approximation error $|M(x_i) - \hat{y}(x_i)|$ over the relevant dataset is given to illustrate that while the global error is quite large, these were rare events. The training error is not displayed in the table since it is zero for the method. In all respects the nearest neighbor regression algorithm performs worse than the baseline linear regression models.

Training size	Test error			Global error
	avg.	std.	peak	
2×10^3	0.0881	0.0810	[-0.572, 0.580]	[-0.639, 0.706]
5×10^3	0.0769	0.0703	[-0.533, 0.513]	[-0.622, 0.648]
5×10^4	0.0601	0.0569	[-0.458, 0.451]	[-0.468, 0.568]

Table 2: Performance summary of the nearest neighbor regression models.

The k-nearest neighbor regression performed slightly better, achieving performance on par with that of the linear regression for a training dataset of a given size. One notable difference is that peak fitting error on the test points was similar to the global fitting error. Fitting error statistics from this class of regression models, accompanied by the optimal k values, are shown in table 3.

The kernel distance weighted k-nearest neighbor regression algorithm described in section 5.3 performed similarly to the k-nearest neighbor regression algorithm so I will not tabulate its performance here. However, a few features of the regression model are worth noting. First, recall

Training size	k	Training error			Test error			Global error
		avg.	std.	peak	avg.	std.	peak	
2×10^3	13	0.056	0.049	[-0.21, 0.34]	0.061	0.054	[-0.24, 0.37]	[-0.29, 0.40]
5×10^3	11	0.047	0.044	[-0.20, 0.29]	0.061	0.055	[-0.28, 0.34]	[-0.33, 0.49]
5×10^4	10	0.034	0.033	[-0.21, 0.33]	0.037	0.036	[-0.24, 0.35]	[-0.27, 0.40]

Table 3: Performance summary of the k -nearest neighbor regression models.

that kernel took the form

$$k(x, x') = \exp\left(-\frac{1}{\beta}(x - x')^T P(x - x')\right) \quad (7)$$

where P was diagonal. This allowed each coordinate to exhibit a different weight in the final regression output. For increased values of k , the optimal matrix P becomes less and less like a scaled identity, with each direction being assigned different weights. This indicates that the different coordinate directions contain varying amounts of useful information for the regression. This same theme was encountered with the Gaussian random process regression models.

The ϵ -tube support vector regression algorithm was the first regression algorithm I examined that out-performed the linear regression model. Due to its high computational burden, I was only able to use this algorithm with training datasets of 2000 and 5000 points. After several hours of waiting, I had to terminate the 50000 training point run in order to submit this report. The optimal ϵ value determined by my gridding and cross-validation procedure was small (1% of the swing of M) for both cases. This is not surprising, since random error was not an issue for this problem. Table 4 shows the performance of the ϵ -SV regression model on the problem at hand, together with the optimal values of C . The optimal values for ϵ and β were 0.01 and 0.3, respectively, in both cases.

Training size	C	Training error			Test error			Global error
		avg.	std.	peak	avg.	std.	peak	
2×10^3	180	0.0081	0.0068	[-0.069, 0.060]	0.012	0.011	[-0.094, 0.088]	[-0.22, 0.25]
5×10^3	160	0.0075	0.0069	[-0.083, 0.0829]	0.0089	0.0089	[-0.81, 0.078]	[-0.21, 0.29]

Table 4: Performance statics for the ϵ -SVR model.

The Gaussian process regression model was the last algorithm I examined. I considered several forms for the correlation function of the random process which were provided by the DACE toolbox in the outer-loop of a cross-validation exercise. A Gaussian function of the same form as the kernel I employed for the weighted nearest neighbor algorithm achieved the best performance.

This regression algorithm was vastly superior to even ϵ -SVR. The comparison isn't entirely fair, since that SV kernel I used didn't allow for individual weights for each coordinate. However, even when I made a similar restriction to the GPR it still outperformed ϵ -SVR. The biggest drawback to the GPR model was its high memory requirements. This limited the size of training set I could consider to 3000 points. The performance of this regression model is summarized in table 5.

Figure 2 illustrates histograms of the fitting error of the GPR model and the linear regression (sup-norm quadratic) model on a test dataset comprising 200000 points when both are trained in the same 3000 point training set. The difference is striking. The GPR regression model is unbiased, the mass is concentrated at 0, and the tails of its histogram are narrower than the corresponding tails of histogram from the linear regression model. In all respects, the GPR makes much better use of the training data, realizing a highly accurate approximation to M .

While the GPR model provides a very accurate representation of M when trained on a modest sized training set, it was unable to achieve my final project goal. Recall that I had hoped that a

Training size	Test error			Global error
	avg.	std.	peak	
2×10^3	0.0029	0.0039	[-0.047, 0.034]	[-0.11, 0.13]
3×10^3	0.0027	0.0040	[-0.044, 0.033]	[-0.12, 0.13]

Table 5: Training, test, and global performance of the Gaussian process regression models.

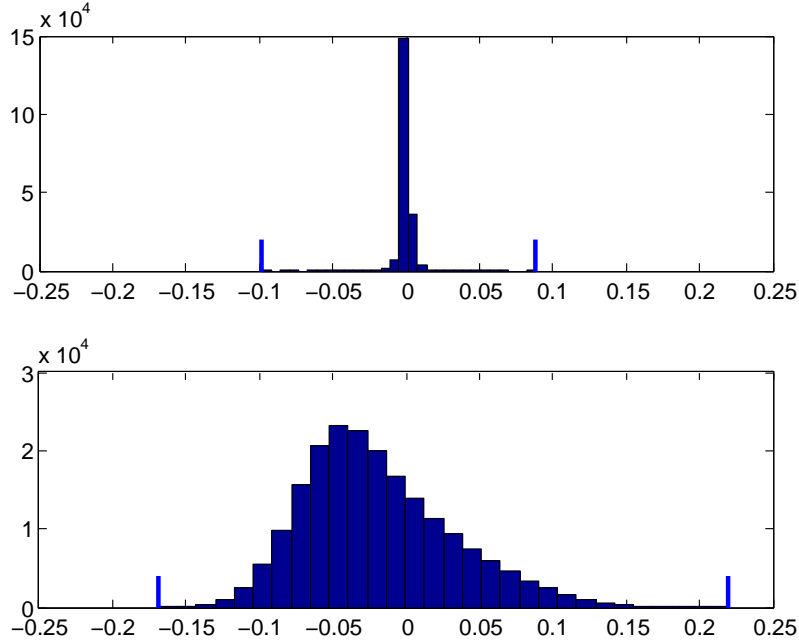


Figure 2: Comparison of the GPR regression model (top panel) and quadratic polynomial model determined by linear regression (bottom panel) on a test set of 200000 points. The small vertical lines depict the tails of the two histograms.

quadratic model trained on the GPR model (which can cheaply facilitate a very large training set) would provide a good sup-norm approximation to the original function M . Unfortunately, when I carried out this procedure by training a linear regression model on the best GPR approximation of M using a training set of 200000 points, the resulting quadratic function had similar peak error characteristics with respect to the original function M as did a quadratic approximation trained on the 3000 sample points of M used to develop the GPR model. While this is a negative result, as I describe in the final section, some good may still come out of the idea.

7 Conclusions

In this report I described my experience training several regression models covered in cs294 on a deterministic function M having a 9-dimensional domain. I discovered that regression models based on nearest neighbor ideas performed rather poorly, achieving at best an approximation accuracy similar to that obtained using linear regression with 2nd-order polynomial basis functions. A ϵ -tube support vector regression model provided better results, while a Gaussian random process regression model was able to capture the behavior of M quite well (see figure 2).

The motivation for the work is developing a methodology for model invalidation, where by an efficient algorithm could be used to certify that the set $F = \{x \in H : |M(x) - d| \leq u\}$ is empty (see section 2). An existing method to achieve this is to approximate M in sup-norm by a quadratic function. However, obtaining such approximations often require an impractical number of evaluations of M . My hope was that one of the algorithms I considered would provide an accurate cheaply evaluatable surrogate model for M that could then be approximated by a quadratic function. In the case considered, I was unable to achieve this, likely due to the fact that quadratic functions are just not that expressive. However, the results of my investigation indicate that the GPR model does provide an accurate cheaply evaluatable surrogate model for M . Perhaps this can be the launching point for other tractible model invalidation ideas.

References

- [1] T.-M. Yi, M. Fazel, X. Liu, T. Otitoju, A. Papachristodoulou, S. Prajna, and J. Doyle, Application of Robust Model Validation Using SOSTOOLS to the Study of G-Protein Signaling in Yeast. In *Proceedings of FOSBE*, Aug 2005.
- [2] C. Merkwirth, U. Parlitz, I. Wedekind, and W. Lauterborn, *TSTOOL*, available <http://www.physik3.gwdg.de/tstool/>.
- [3] J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn, em Design and Analysis of Computer Experiments, *Statistical Science*, **4**, 1989.
- [4] M. Vogt, A. Ayarov, and B. Gedan, *MATLAB Interface for LIBSVM*, available <http://www.iat.tu-darmstadt.de/~vogt/en/software.html>.
- [5] S. N. Lophaven, H. B. Nielsen, and J. Sondergaard, *DACE: A MATLAB Kriging Toolbox* available <http://www2.imm.dtu.dk/~hbn/dace/>.