On-line Gaussian Processes for Robotics

Forschungspraxis

from

Jonas Hess

Lehrstuhl für

STEUERUNGS- UND REGELUNGSTECHNIK

Technische Universität München

Supervisors:	M.S. Matteo Saveriano, Prof. Dongheui Lee
Begin:	26.10.2015
End:	18.01.2016

Abstract

This project focused on the challenges of an on-line Gaussian process for regression applications. Thereby, incremental and decremental inversion methods have been developed as a means to shorten computing time. In tests, the incremental and decremental regression algorithms performed significantly faster than two state-of-the-art implementations of GP while still retaining high estimation accuracy.

Zusammenfassung

Der Fokus dieser Forschungsarbeit liegt auf den Herausforderungen einer on-line Regression mit Gaußschen Prozessen. Dazu wurden inkrementelle und dekrementelle Matrixinversiontechniken entwickelt, die Rechenzeit einsparen sollen. In Tests erwiesen sich die entwickelten inkrementellen und dekrementellen Regressionsalgorithmen gegenüber zwei anerkannten aktuellen GP Methoden als signifikant schneller ohne Einbußen an Schätzungsgenauigkeit.

Contents

1. Introduction

Gaussian Processes (GP) are statistical modeling tools that have been successfully used in a number of robotics applications, such as imitation and reinforcement learning. To face the increasing computational time problem in large-scale on-line applications, incremental algorithms for GP have been proposed in numerous literatures [1][2][3][4][5]. These algorithms are useful to update the learned parameters according to new incoming data. This project investigated an approach of using incremental and decremental inversion methods in order to increase computation efficiency.

1.1 Gaussian Process as means for Regression

"When concerned with a general Gaussian process regression problem, it is assumed that for a Gaussian process *f* observed at coordinates x, the vector of values $f(\boldsymbol{x})$ is just one sample from a multivariate Gaussian distribution of dimension equal to number of observed coordinates $|\boldsymbol{x}|$. Therefore under the assumption of a zero-mean distribution, $f(\boldsymbol{x}) \sim N(0, \boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}'))$, where $\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}')$ is the covariance matrix between all possible pairs $(\boldsymbol{x},\boldsymbol{x}')$ for a given set of hyperparameters $\boldsymbol{\theta}$. As such the log marginal likelihood is:

$$\log p(f(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{x}) = -\frac{1}{2}f(\boldsymbol{x})^T\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}')^{-1}f(\boldsymbol{x}) - \frac{1}{2}log\ det(\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}')) - \frac{|\boldsymbol{x}|}{2}log\ 2\pi \quad (1)$$

and maximizing this marginal likelihood towards $\boldsymbol{\theta}$ provides the complete specification of the Gaussian process *f*. One can briefly note at this point that the first term corresponds to a penalty term for a model's failure to fit observed values and the second term to a penalty term that increases proportionally to a model's complexity. Having specified $\boldsymbol{\theta}$ making predictions about unobserved values $f(\boldsymbol{x}^*)$ at coordinates $\boldsymbol{x}^*$ is then only a matter of drawing samples from the predictive distribution $p(\boldsymbol{y}^*|\boldsymbol{x}^*,f(\boldsymbol{x}),\boldsymbol{x}) = N(\boldsymbol{y}^*|A,B)$ where the posterior mean estimate A is defined as:

$$A = \boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x}^*,\boldsymbol{x})\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}')^{-1}f(\boldsymbol{x}) \quad (2)$$

and the posterior variance estimate B is defined as:

$$B = \boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x}^*,\boldsymbol{x}^*) - \boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x}^*,\boldsymbol{x})\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x},\boldsymbol{x}')^{-1}\boldsymbol{K}(\boldsymbol{\theta},\boldsymbol{x}^*,\boldsymbol{x})^T \quad (3)$$

where ***K(θ,x*,x)*** is the covariance between the new coordinate of estimation ***x**** and all other observed coordinates ***x*** for a given hyperparameter vector **θ**, ***K(θ,x,x')*** and *f(x)* are defined as before and ***K(θ,x*,x*)*** is the variance at point ***x**** as dictated by ***θ***." From Rasmussen, *Gaussian processes for machine learning* [6].

1.2 Matrix inversion is most complex operation in the calculation of posterior estimates

As shown in (2) and (3), the posterior mean estimate and posterior variance estimate both contain the inverse of the covariance matrix ***K(θ,x,x')***. Matrix inversion is a very costly operation in terms of processing time with a complexity between $O(n^3)$ (Gauß-Jordan) and $\sim O(n^{2.3})$ and is a known bottleneck for Gaussian processes.
The objective of this project is to investigate various inversion methods, identify the most efficient ones, integrate those methods into a GP model and benchmark these solutions against state-of-the-art solutions.

1.3 Related work

A commonly used optimization is the utilization of the Cholesky decomposition of the covariance ***K(θ,x,x')*** in equations (2) and (3) instead of calculating its inverse. This can e.g. be found it Rasmussen's Gaussian Process Machine Learning toolbox and many other implementations [6]. See also chapter 2 for a more detailed explanation of the Cholesky decomposition and its utilization possibilities.

In 2002, Csató published an efficient Gaussian Processes capitalizing from the sparsity of the covariance matrix and demonstrated the efficiency in terms of regression, classification and density estimation [7].

In 2008, Naish-Guzman et al. presented the FITC approximation, a low-rank plus diagonal approximation to the exact covariance using inducing points, basing the computations on cross-covariances between training, test and inducing points only [8]. We tested our solution against the FITC approximation and found that in our tests the FITC performed better in cases where very little learning information are given. However, in all other cases, the FITC approximation performed equally fast or even worse than the exact estimation. See chapter 3 for more information.

In 2014, Anitescu et al. presented an inversion-free estimating equation approach for Gaussian Process models that requires only a small fraction of the computational effort of maximum likelihood calculations [9]. For specific test cases with data sets of up to 1 million data points, the estimating equation method returned an accuracy close to the optimal one as measured but at a fraction of the cost (1% or even less). However, the presented approach only finds an efficient solution for calculating the maximum likelihood estimate and does not offer an alternative solution of calculating the posterior estimate of a Gaussian process.

Moreover, Kronecker methods have been used to exploit structure in the GP covariance matrix for scalability, while allowing for expressive kernel learning [10].

However, Kronecker methods have been confined to Gaussian likelihoods. In 2015, Flexman et al. proposed new scalable Kronecker methods for Gaussian processes with non-Gaussian likelihoods, using a Laplace approximation which involves linear conjugate gradients for inference, and a lower bound on the GP marginal likelihood for kernel learning [11]. The main application for Kronecker methods are extrapolation of learning data, making it difficult to transfer the method to other regression applications. In the scope of this project, Kronecker methods were therefore not applied to our regression problem, but it remains a potential topic for future research.

1.4 Chapter overview

Chapter 2 describes examined inversion methods and their performance, while in chapter 3 the most efficient method is benchmarked against state-of-the-art solutions. Chapter 4 gives a summary of the results and an outlook to future research possibilities.

2. Examination of inversion methods

Various inversion methods have been examined over the course of the project. They all span around solving the operation

$$\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{y} \qquad (4)$$

which represents the critical part of equations (2) and (3).

2.1 Inversion methods

There are various possibilities to mathematically correctly calculate equation (4). Below is a short explanation of the ones most popular or deemed most promising.

Inv(): Matlab has a comprehensive, built-in matrix-inversion function inv(). This represents a native, un-optimized approach and unsurprisingly delivered the slowest results.

M\y: Also a built-in Matlab method that is used to solve equations Mx = y. It is based on the cholmod()-function which is an efficient built-in implementation of the Choleksy decomposition [12]: This algorithm is a decomposition of a Hermitian, pdf matrix into the product of a lower triangular matrix L and its conjugate transpose U. It is a numerically very efficient way of solving Ax = b using forward and back substitution.

Chol(): We also investigated explicit Cholesky solutions, first of all using the built-in chol()-function from Matlab and then solving the equation:

```
    U = chol(M);
    Result = U\(U.'\Y);
```

Not shown in figure 1 is an alternative implementation of the Cholesky decomposition coded in Matlab which turned out to be not as efficient.

Cholesky increment: There is also the possibility of calculating a Cholesky decomposition based on a prior Cholesky decomposition of a Matrix that is equal to M except that the last row and column are removed. This represents the case that one entry is added to the covariance matrix and (3) has to be recalculated [13].

```
N = size(M);
for l=1:NumIt
```

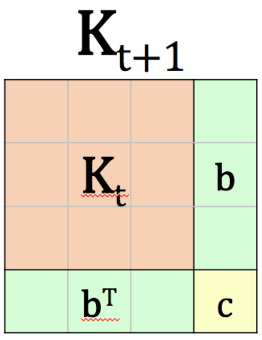
```
    L = [LOld zeros(N-1,1); zeros(1,N)];
    for j=1:N-1
           sum = 0;
           for i=1:j-1
               sum = sum + L(N,i)*L(j,i);
           end
           L(N,j)=(M(N,j)-sum)/L(j,j);
    end

    sumDiag = L(N,1:N-1)*L(N,1:N-1).';
    L(N,N)=sqrt(M(N,N)- sumDiag);

    RIncrChol = L.'\(L\Y);
end
```

A Matlab-based incremental Cholesky implementation turned out to be less efficient than recalculating the Cholesky decomposition from scratch using the built-in chol()-function. However, we also implemented this method in C with a Mex-Matlab-Interface, which gave a much better performance and performed significantly better than the recalculating the whole Cholesky decomposition.

Increment using block form: What turned out to be most efficient was an incremental way of calculating the inverse $\mathbf{M}^{-1}$.



If a new learning point is added to the GP model, ***K(θ,x,x')*** is incremented by 1 dimension:

$$\boldsymbol{K}_{t+1} = \begin{bmatrix} \boldsymbol{K}_t & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{bmatrix}$$

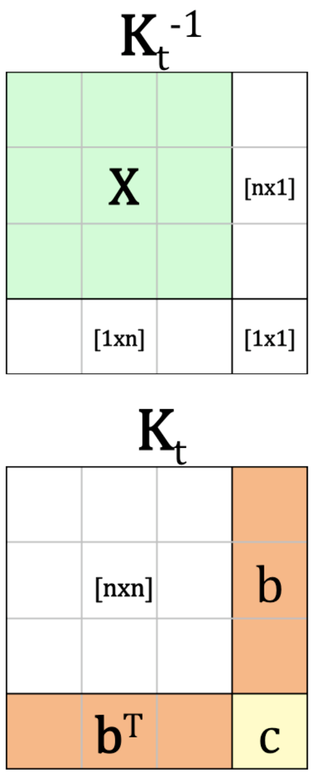
Then the inverse of $\boldsymbol{K}_{t+1}$ can be calculated as:

$$\boldsymbol{K}_{t+1}^{-1} = \begin{bmatrix} \boldsymbol{K}_t^{-1} + \frac{1}{k}\boldsymbol{K}_t^{-1}\boldsymbol{b}\boldsymbol{b}^T\boldsymbol{K}_t^{-1} & -\frac{1}{k}\boldsymbol{K}_t^{-1}\boldsymbol{b} \\ -\frac{1}{k}\boldsymbol{b}^T\boldsymbol{K}_t^{-1} & \frac{1}{k} \end{bmatrix} \quad (5)$$

$, where\ k = c - \boldsymbol{b}^T\boldsymbol{K}_t^{-1}\boldsymbol{b}$.

This algorithm is derived from the block-form inversion [14].

Decremental inversion using Sherman-Morrison:
Similarily, the inverse of a decremented matrix can be calculated based on the inverse of the prior matrix.



If a learning point is removed from the GP model, ***K(θ,x,x')*** is decremented by 1 dimension. To calculate the inverse of the decremented matrix $\boldsymbol{M}_{t+1}^{-1}$, the previous inverse $\boldsymbol{M}_t^{-1}$ and $\boldsymbol{M}_t$ (shown as $\boldsymbol{K}_t^{-1}$ and $\boldsymbol{K}_t$ in the right figure) are then split in the parts **X**, **b**, **b**^{T} and c.

The inverse $\boldsymbol{M}_{t+1}^{-1}$ of the decremented matrix $\boldsymbol{M}_t$ can be calculated as:

$$\boldsymbol{M}_{t+1}^{-1} = \boldsymbol{X} - \frac{\boldsymbol{Xb}\,(\boldsymbol{Xb})^T}{c + \boldsymbol{b}^T\boldsymbol{Xb}} \quad (6)$$

This is derived from the block-form inversion in combination with the Sherman-Morrison formula [14][15]:

$$(\boldsymbol{A} + \boldsymbol{uv}^T)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{uv}^T\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}} \quad (7)$$

Swapping of entries: Using the decremental method, the last entry of the covariance matrix **K**(**θ**,**x**,**x**') can be removed. However, if another entry of **K** is supposed to be removed, entries need to be swapped first. This can be done easily by swapping the corresponding rows and columns of the inverse of the prior matrix and the y as shown in figure 2.

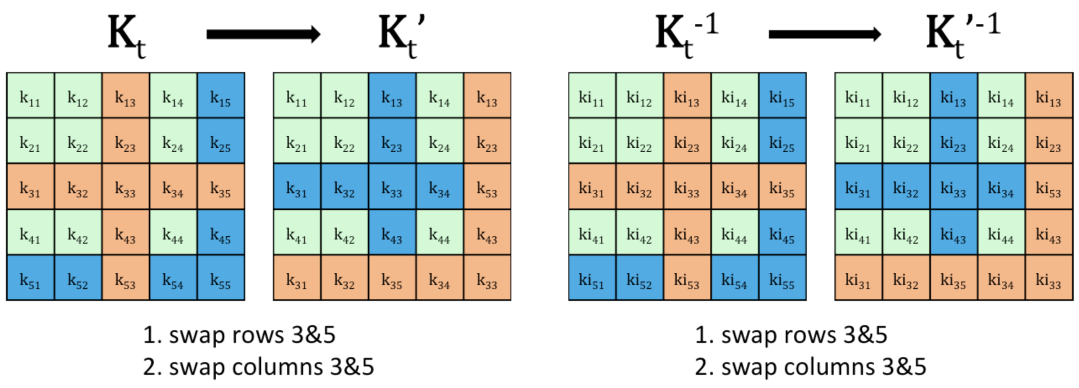


*Figure 2: Visualization of entry swapping: If entries **x**$_1$ and **x**$_2$ of the covariance matrix **K(θ,x,x')** are swapped this results in a swap of the corresponding rows and columns in **K**, as well as in a swap of the the corresponding rows and columns in the inverse of **K**. This allows to efficiently remove any entry from the covariance matrix and recalculate the posterior estimates using the decremental inversion.*

2.2 Inversion benchmark setup and results

All methods were tested in Matlab R2015a. They calculated equation (4) given a random, symmetric and postive-definite (pdf) **M** and a random vector **y**. Figure 1 shows the relation between the size of **M** and the average computing time for 30 iterations (15 iterations for matrices >1000).

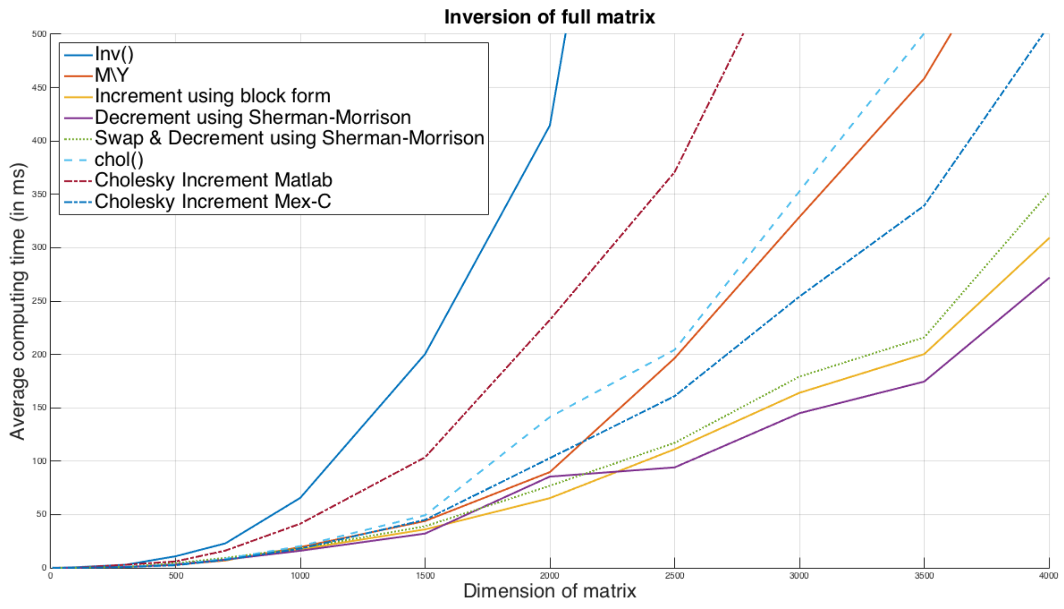


Figure 1: Benchmark between various inversion methods. Incremental and (swapping) decremental methods have been identified to be most efficient and have therefore been further investigated.

Thereby, the incremental and decremental inversions (including swapping of entries) have been identified as most efficient way of calculating expression (4). The next chapter shows how they were integrated into an existing GP-library and benchmarked against other GP-methods.

3. Benchmarking of GP implementations

As described in chapter 2, the incremental and decremental inversion methods have been identified as most efficient compared to other inversion methods. In order to find out by how much GP calculations can be optimized using those two methods, we are currently integrating them into the latest version (July 2015) of the Gaussian Process Machine Learning Toolbox by Rasmussen and Williams.
This toolbox offers a wide range of state-of-the-art implementations of likelihoods, inference methods and covariance functions, which enables a direct comparison of different GP methods.

The incremental and decremental inversion methods have been integrated into the toolbox by creating two new inference methods infIncremental() and infDecremental(). The two implementations can be found in appendix A and B. Both of them are based on the built-in infExact() and represent an exact inference method. They can be called just like any other inference method using the gp()-function of the toolbox. Therefore, the results can easily be reproduced on any other computer by simply inserting the respective two inference Matlab files.
Furthermore, in order to compare only the posterior mean estimation methods, the gp()-function was slightly adjusted to prohibit it from also calculating the posterior variance estimate.

3.1 Benchmark setup

The investigated regression methods are three exact inference methods (built-in, incremental as well as swapped decremental version) as well as FITC approximation. Those methods were analyzed in terms of computing time and estimation accuracy in relation to the number of learning points, as well as the density of learning points (the higher the density, the more distinct the regression).

A sinus wave was the basis of the regression task. From this sinus function, individual learning points were derived randomly and a white Gaussian noise with a signal-to-noise-ratio of 10 added. The number of learning points as well as the density of learning points per unit determines the width of the regressed function. An excerpt of such a regression task is shown in figure 3.

As performance indicators, computing time and estimation accuracy were measured. Thereby, the computing time (in ms) was averaged over 25 iterations of every calculation for each regression methods. Accuracy was measured by taking the mean of the squared distance of each posterior-estimate to the true value (MSE).

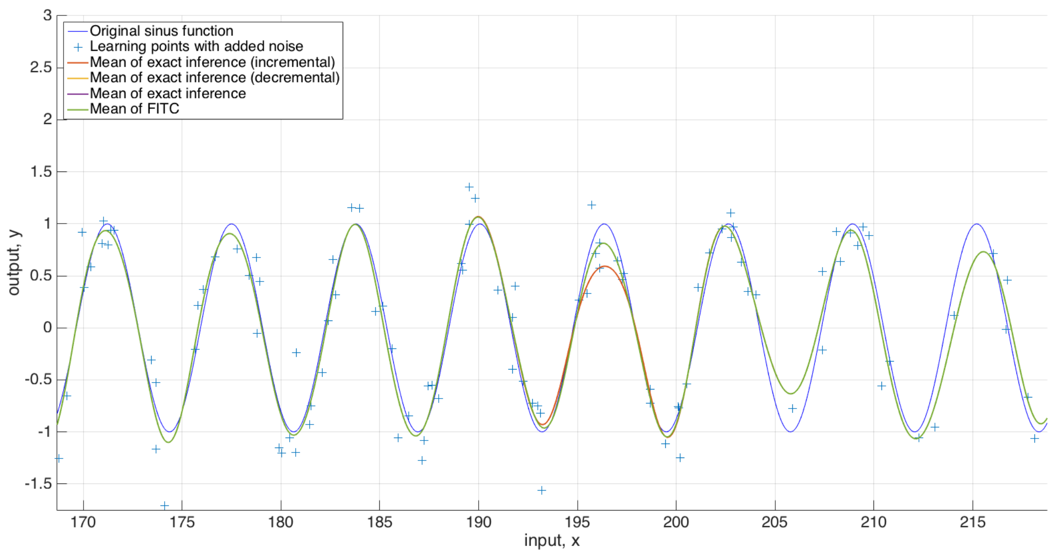


Figure 3: Regression results using the built-in exact inference method from the GPML-toolbox as well as the exact inference method based on the incremental and swapping decremental inversion method. Learning points are derived from a sinus-function with signal-to-noise-ratio of 10 and a point density of 2.0/unit. As expected, all exact inference regression results are nearly identical and therefore individual regression lines are mostly not be visible.

For this benchmark scenario, the following test parameters were used, resulting in 4x9=36 test cases for each regression method.

```
tDensity     = [0.5,  1,   3,   5];
% Average number of learning points per width unit
tNumPoints   = [50, 100, 200, 300, 500, 700, 1000, 1300, 1500];
% Number of learning points
```

3.2 Benchmark results

Throughout our benchmarks, the incremental and decremental exact inference methods needed less or way less computing time than the built-in exact inference methods, as shown in figure 4. In terms of estimation accuracy, they performed nearly identical to the built-in method, as expected. This can be seen in figure 5.
The FITC approximation showed mixed results. In test cases with a very low learning point density of 0.5/unit, the FITC approximation was the fastest regression method. However, the tradeoff for the gained computing speed is a significant drop in terms of accuracy as it can be seen in figures 5 and 6. Such a test case is not a realistic scenario for a real life application.
For test cases with a higher density of learning points (density of 1.0 or higher) the FITC approximation showed a computing time equal to or even slower than the built-in exact inference method and was significantly outperformed by the

incremental and decremental methods.
In summary, for this test scenario, the incremental and decremental methods performed as desired - they used significantly less computing time than the built-in exact inference method while keeping the same estimation accuracy. If those gains in computing time suffice for online applications of Gaussian processes with a large number of learning points is up to be seen.

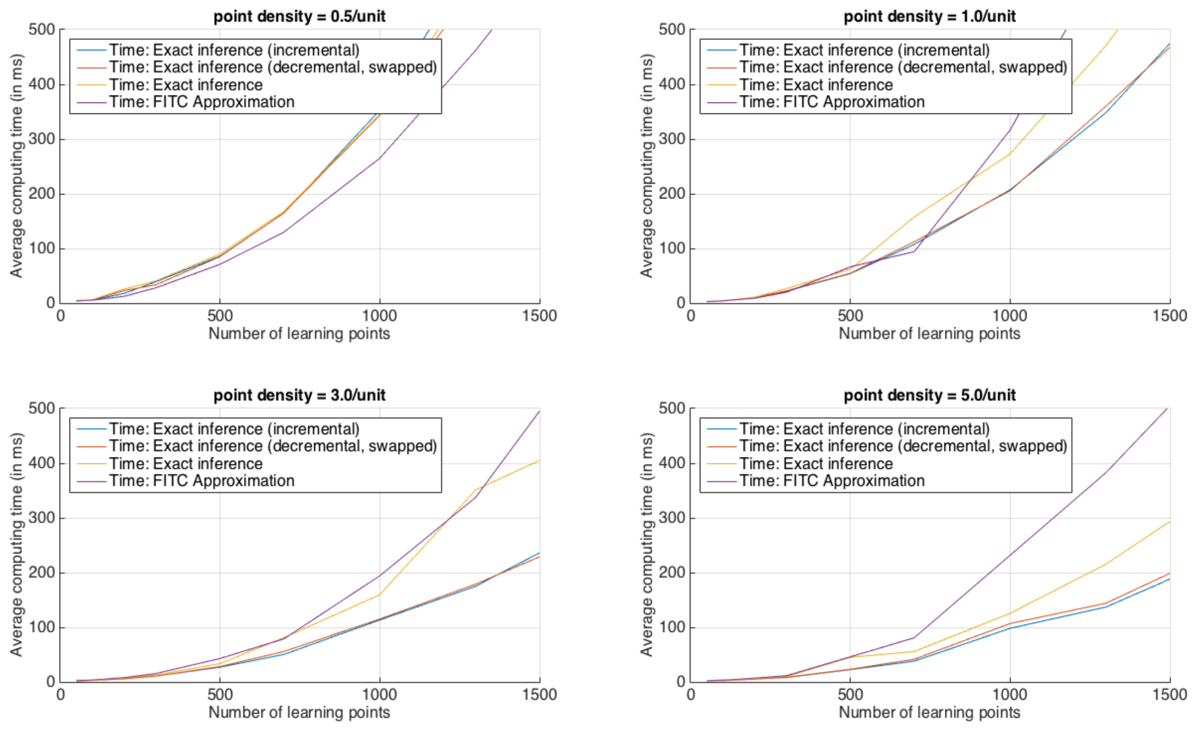


Figure 4: Benchmark between exact inference methods and FITC approximation in terms of computing time. For point densities of 1.0/unit or higher, the incremental and decremental methods computed significantly faster than the other methods. For a low point density of 0.5/unit the FITC approximation calculated the fastest but showed a significant drop in estimation accuracy as shown in figures 5 and 6.

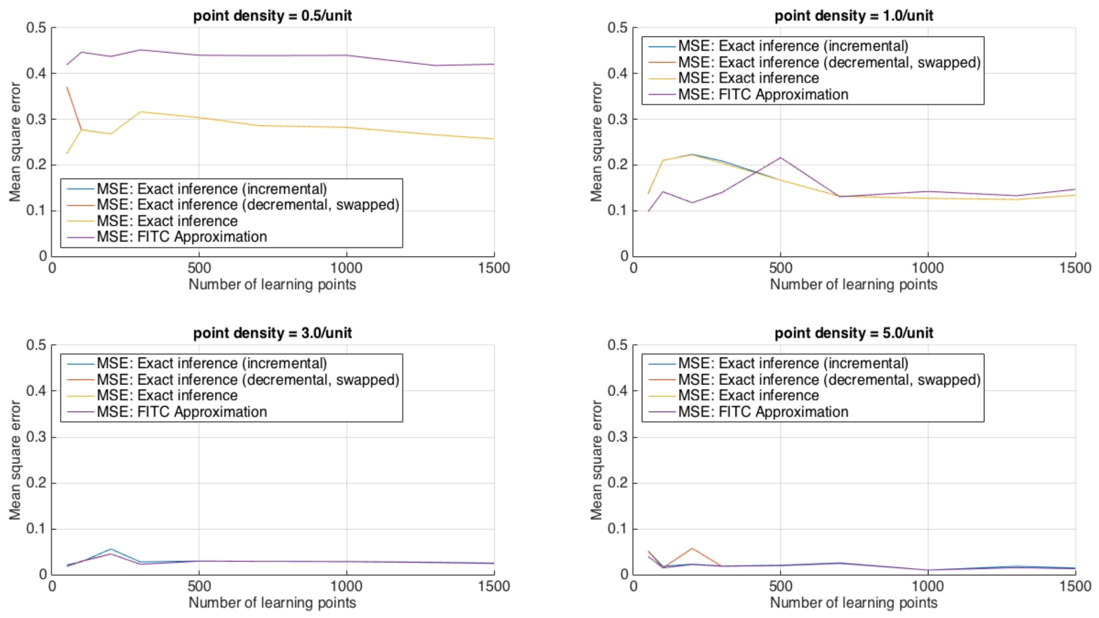


Figure 5: Benchmark between exact inference methods and FITC approximation in terms of estimation accuracy, measured as mean squared error (MSE). For high point densities of 1.0/unit or higher, all methods achieved sufficiently low MSEs. For a low point density of 0.5/unit the FITC approximation (purple) shows a significant drop of accuracy as visualized in figure 6.

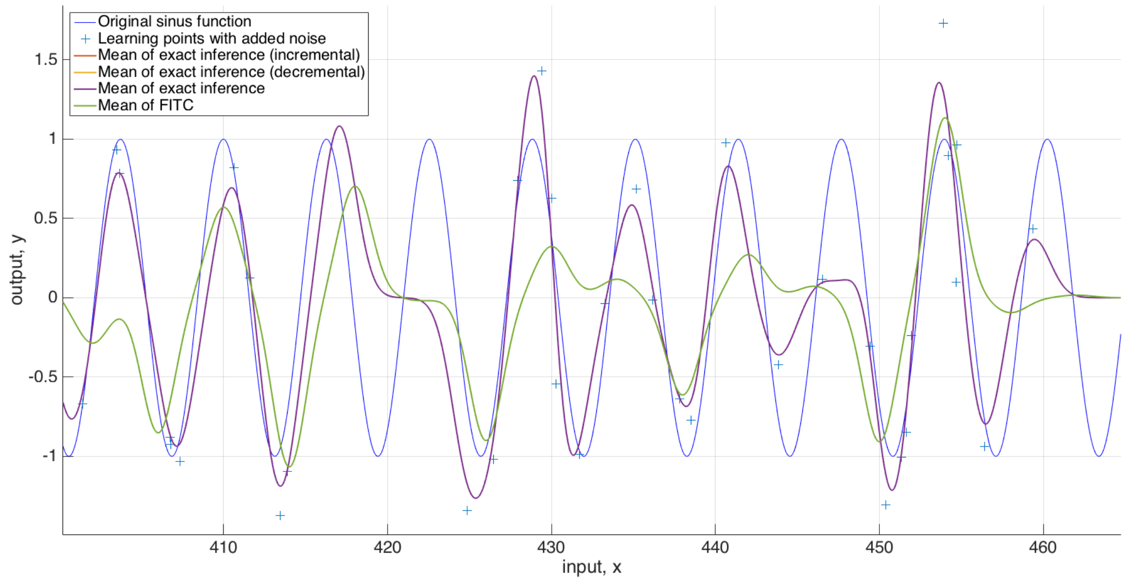


Figure 6: A test case with 500 learning points and a density of 0.5 points/unit. The low point density causes a significant drop in estimation accuracy. The exact inference methods (purple) still estimate a function somehow close to the original sinus wave (blue) while the FITC approximation (green) fails.

4. Summary and outlook

This project focused on the challenges of an online gaussian process for regression applications. First of all, a matrix inversion was identified as the most significant bottleneck for processing time. Therefore, different inversion methods were investigated and benchmarked against each other. Eventually, an incremental and an decremental inversion method that build upon prior results were identified as most efficient.
Secondly, those two inversion methods were integrated in a Gaussian process and benchmarked against state of the art solutions. This was done using the latest version (July 2015) of the Gaussian Process Machine Learning Toolbox by Rasmussen and Williams. Using the incremental and decremental inversion methods, two new inference methods were developed. In a final step they were benchmarked against a built-in exact inference method and the FITC approximation inference. As a result, the incremental and decremental needed less or way less computing time than the built-in exact inference method while keeping a nearly identical estimation accuracy. The FITC approximation was only faster in cases with a low point density which in return resulted in a low estimation accuracy and therefore nullifying the saved processing time.

The ultimate goal is to integrate an online GP solution into a robotic application. However, further research is necessary to realize a real life integration.
A future point of research could be the integration of incremental and decremental solutions into the FITC approximation. Also, further comparisons with other GP solutions should be performed.
Another future research focus lies in the likelihood function. While this project focused solely on the regression step of a Gaussian process, assuming that a system's hyper-parameters are fix, in real life applications hyper-parameters often need to be reevaluated. This brings up more challenges for an online GP as the reevaluation of hyper-parameters is an optimization problem that can have an even higher complexity than the regression step. Hence, for a comprehensive online GP solution, the calculation efficiency of the likelihood function should be further researched.

List of figures

References

[01] Nguyen-Tuong, Duy, Matthias Seeger, and Jan Peters. "Model learning with local gaussian process regression." *Advanced Robotics* 23, no. 15 (2009): 2015-2034.

[02] Kronander, Klas, Mohammad Khansari, and Aude Billard. "Incremental Motion Learning with Gaussian Process Modulated Dynamical Systems."

[03] Lütz, Alexander, Erik Rodner, and Joachim Denzler. "EFFICIENT MULTI-CLASS INCREMENTAL LEARNING USING GAUSSIAN PROCESSES."

[04] Quinonero-Candela, Joaquin, and Ole Winther. "Incremental gaussian processes." In *Advances in neural information processing systems*, pp. 1001-1008. 2002.

[05] Vijayakumar, Sethu, Aaron D'souza, and Stefan Schaal. "Incremental online learning in high dimensions." *Neural computation* 17, no. 12 (2005): 2602-2634.

[06] Rasmussen, Carl Edward. "Gaussian processes for machine learning." *MIT Press*, 2006.

[07] Csató, Lehel. "Gaussian processes: iterative sparse approximations." PhD diss., Aston University, 2002.

[08] Naish-Guzman, Andrew, and Sean Holden. "The generalized FITC approximation." In *Advances in Neural Information Processing Systems*, pp. 1057-1064. 2007.

[09] Anitescu, Mihai, Jie Chen, and Michael L. Stein. "An inversion-free estimating equation approach for Gaussian process models." *submitted for publication*(2014).

[10] Wilson, Andrew, Elad Gilboa, John P. Cunningham, and Arye Nehorai. "Fast kernel learning for multidimensional pattern extrapolation." In *Advances in Neural Information Processing Systems*, pp. 3626-3634. 2014.

[11] Flaxman, Seth R., Andrew Gordon Wilson, Daniel B. Neill, Hannes Nickisch, and Alexander J. Smola. "Fast Kronecker Inference in Gaussian Processes with non-Gaussian Likelihoods." (2015).

[12] Davis, Timothy A., and W. Hager. "CHOLMOD: supernodal sparse cholesky factorization and update/downdate." (2005).

[13] Polok, Lukas, Marek Solony, Viorela Ila, Pavel Smrz, and Pavel Zemcik. "Incremental Cholesky Factorization for Least Squares Problems in Robotics⋆." In *Intelligent Autonomous Vehicles*, vol. 8, no. 1, pp. 172-178. 2013.

[14] Drakos, Nikos. "Computer based learning unit." *University of Leeds.“Network Intrusion Detection: Evasion, Traffic normalization and End-to-End protocol semantics* (1997).

[15] Eldén, Lars, Misha E. Kilmer, and Dianne P. O’Leary. "Updating and Downdating Matrix Decompositions." In *GW Stewart*, pp. 45-58. Birkhäuser Boston, 2010.

Appendix

A - Implementation of the incremental exact inference method

```
function [post nlZ dnlZ] = infIncrement(hyp, mean, cov, lik, x, y)

   % Needs to be passed hyp.Increment.KOld and hyp.Increment.AOldInv!
   % KOld = old squared exponential Covariance Matrix with isotropic distance measure
   %
   % Exact inference for a GP with Gaussian likelihood. Compute a parametrization
   % of the posterior, the negative log marginal likelihood and its derivatives
   % w.r.t. the hyperparameters. See also "help infMethods".
   %
   % Copyright (c) by Carl Edward Rasmussen and Hannes Nickisch, 2015-07-13.
   %                                File automatically generated using noweb.
   %
   % See also INFMETHODS.M.

   if iscell(lik), likstr = lik{1}; else likstr = lik; end
   if ~ischar(likstr), likstr = func2str(likstr); end
   if ~strcmp(likstr,'likGauss')              % NOTE: no explicit call to likGauss
     error('Exact inference only possible with Gaussian likelihood');
   end

   %% This part has been changed!
   n = length(x);
   KOld = hyp.Increment.KOld;
   % xs = x(n);                                         % incremental point xs
   % xOld = x(1:n-1);                   % old x-vector without incremental point xs

   ell = exp(hyp.cov(1));                          % characteristic length scale
   sf2 = exp(2*hyp.cov(2));                                  % signal variance


   p = sq_dist(x(1:n-1)'/ell,x(n)/ell);          % evaluate new covariance vector
   p = sf2*exp(-p/2);                              % for incremental point xs
   %K = [KOld p; p' KOld(1,1)];                    % evaluate covariance matrix

   m = feval(mean{:}, hyp.mean, x);                        % evaluate mean vector
   sn2 = exp(2*hyp.lik);                              % noise variance of likGauss

   %pL = -inv(K+sn2*eye(n));                           % simple inversion function
   pL = -myIncrementalInversion(hyp.Increment.AOldInv, p, KOld(1,1) );    % increm.
inversion

   alpha = -pL*(y-m);

   %%

   post.alpha = alpha;                           % return the posterior parameters
   post.sW = ones(n,1)/sqrt(sn2);                  % sqrt of noise precision vector
   post.L = pL;

   %% Not applicable if nargin==8 for gp()-function
   if nargout>1                               % do we want the marginal likelihood?
     nlZ = (y-m)'*alpha/2 + sum(log(diag(L))) + n*log(2*pi*sl)/2;   % -log marg lik
     if nargout>2                                         % do we want derivatives?
       dnlZ = hyp;                                 % allocate space for derivatives
       Q = solve_chol(L,eye(n))/sl - alpha*alpha';     % precompute for convenience
       for i = 1:numel(hyp.cov)
         dnlZ.cov(i) = sum(sum(Q.*feval(cov{:}, hyp.cov, x, [], i)))/2;
       end
       dnlZ.lik = sn2*trace(Q);
       for i = 1:numel(hyp.mean)
         dnlZ.mean(i) = -feval(mean{:}, hyp.mean, x, i)'*alpha;
       end
     end
   end

end

function [AincrInv] = myIncrementalInversion(AInv, b, c )
%MYINCREMENTALINVERSION Summary of this function goes here
%   Calculates the inverse of an incremented matrix Aincr based on the
%   inverse of a matrix A
%
%   Inputs:
%   AInv = the inverse of the original matrix
%
%   b = the new column of the increased matrix without the element in the
%   bottom right corner A(N,N)
%
%   c = the element of the bottom right corner of the increased matrix
%   A(N,N)
%
%   Outputs:
%   Aincr = Increased matrix, consisting of A, b, transpose of b, and c
%   AincrInv = Inverse of the increased matrix Aincr

bT = b.';
AinvB = AInv*b;
BAinv = AinvB.';
kinv = 1/(c - bT*AinvB);

partA = AInv + kinv*AinvB*BAinv;
partb = -kinv*AinvB;
partc = -kinv*BAinv;
partd = kinv;

AincrInv = [partA partb; partc partd];
%Aincr = [A b; bT, c] ;

end
```

B - Implementation of the decremental exact inference method

```
function [post nlZ dnlZ] = infDecrement(hyp, mean, cov, lik, x, y)
   % Needs to be passed hyp.Decrement.b, hyp.Decrement.c and hyp.Decrement.AOldInv!
   %
   % Exact inference for a GP with Gaussian likelihood. Compute a parametrization
   % of the posterior, the negative log marginal likelihood and its derivatives
   % w.r.t. the hyperparameters. See also "help infMethods".
   %
   % Copyright (c) by Carl Edward Rasmussen and Hannes Nickisch, 2015-07-13.
   %                               File automatically generated using noweb.
   %
   % See also INFMETHODS.M.

   if iscell(lik), likstr = lik{1}; else likstr = lik; end
   if ~ischar(likstr), likstr = func2str(likstr); end
   if ~strcmp(likstr,'likGauss')             % NOTE: no explicit call to likGauss
     error('Exact inference only possible with Gaussian likelihood');
   end

   %% This part has been changed!
   n = length(x);

   ell = exp(hyp.cov(1));                         % characteristic length scale
   sf2 = exp(2*hyp.cov(2));                                  % signal variance

   p = sq_dist(x(1:n-1)'/ell,x(n)/ell);          % evaluate new covariance vector
   p = sf2*exp(-p/2);                              % for incremental point xs

   m = feval(mean{:}, hyp.mean, x);                      % evaluate mean vector
   sn2 = exp(2*hyp.lik);                          % noise variance of likGauss
   pL = -myDecrementalInversion(hyp.Decrement.AOldInv, hyp.Decrement.b,
hyp.Decrement.c);   % decrem. inversion

   alpha = -pL*(y-m);

   %%

   post.alpha = alpha;                         % return the posterior parameters
   post.sW = ones(n,1)/sqrt(sn2);                 % sqrt of noise precision vector
   post.L = pL;
```

```
    %% Not applicable if nargin==8 for gp()-function
    if nargout>1                              % do we want the marginal likelihood?
      nlZ = (y-m)'*alpha/2 + sum(log(diag(L))) + n*log(2*pi*sl)/2;   % -log marg lik
      if nargout>2                                      % do we want derivatives?
        dnlZ = hyp;                               % allocate space for derivatives
        Q = solve_chol(L,eye(n))/sl - alpha*alpha';     % precompute for convenience
        for i = 1:numel(hyp.cov)
          dnlZ.cov(i) = sum(sum(Q.*feval(cov{:}, hyp.cov, x, [], i)))/2;
        end
        dnlZ.lik = sn2*trace(Q);
        for i = 1:numel(hyp.mean)
          dnlZ.mean(i) = -feval(mean{:}, hyp.mean, x, i)'*alpha;
        end
      end
    end

end

function [ AdecrInv] = myDecrementalInversion(AInv, b, c)
%MYDECREMENTALINVERSION Summary of this function goes here
%   Calculates the inverse of a decremented Matrix Adecr based on the inverse of
%   the Matrix A; Possibility to swap entry in order to decrement specific entry
%
%   Input
%   A = Original matrix
%   AInv = Inverse of matrix A

%   Output:
%   Adecr = Decremented matrix of A, last row and column removed
%   AdecrInv = Inverse of decremented matrix Adecr

N = length(AInv)-1;

%% Calculate inverse
Xi = AInv(1:N,1:N);
Xib = Xi*b;
AdecrInv = Xi - (Xib*Xib.')/(c+b.'*Xib);
%Adecr = A(1:N,1:N);
end
```

C - Implementation of the regression benchmark test suite

```
recalculateEverything = true;
close all;
if(recalculateEverything)
   clear all;

   showRegressionPlot = false;

   testcase = 1;

   switch testcase
      case 1
         % Testparameters
         tNumPoints   = [50, 100, 200, 300, 500, 700, 1000, 1300, 1500]; % Number of learning
points
         tDensity     = [0.5,  1,   3,   5];     % Average number of learning points per
width unit
         tStructNames = ['d=0.5','d=1','d=3','d=5'];       % Structure Names for results

      case 2
         % Testparameters
         tNumPoints   = [50, 100, 200]; % Number of learning points
         tDensity     = [0.5,  1,   3];     % Average number of learning points per width
unit
         tStructNames = ['d=0.5','d=1','d=3'];       % Structure Names for results
      otherwise
         disp('Undefined test case!');
         return;
   end

   NumIt       = 25;     % Number of iterations (more iterations = higher benchmark accuracy)

   inputs      = [];
   resultsTime  = tNumPoints;
   resultsMse   = tNumPoints;

   for i=1:length(tDensity)

      avrgTIncrExactMilis = [];
      mseIncrExact       = [];
```

```
avrgTDecrExactMilis = [];
mseDecrExact       = [];

avrgTExactMilis    = [];
mseExact           = [];

avrgTFITCMilis     = [];
mseFITC            = [];

%resultsData        = [];

for j=1:length(tNumPoints)

    %% Initialization
    numPoints = tNumPoints(j)
    density = tDensity(i)
    width = numPoints/density
    n = ceil(width*density);                  % number of learning points
    z = (0:0.1:width)';
    val = sin(z);
    x = rand(n,1)*width;
    y = awgn(sin(x),10);

    likfunc = @likGauss;
    covfunc = @covSEiso;

    % evaluating hyper parameters using infExact
    hyp.cov = [0; 0]; hyp.lik = log(0.1);
    hyp = minimize(hyp, @gp, -25, @infExact, [], covfunc, likfunc, x, y);

    sn2                = exp(2*hyp.lik);

    % Adding old (decremented) covariance matrix KOld as well as old inverse
    % Aold = inv(KOld+sn2*eye(n-1)) for incremental regression
    hyp.Increment.KOld    = feval(covfunc, hyp.cov, x(1:n-1));
    hyp.Increment.AOldInv = inv(hyp.Increment.KOld+sn2*eye(n-1));

    % Adding old (incremented) covariance matrix + sn2 AOld as well as old
    % inverse AOldInv = inv(KOld+sn2*eye(n-1)) for decremental regression
    xDecr               = rand()*width;
    yDecr               = awgn(sin(xDecr),10);
    randSwap             = randi([1,n]);
    xn = [x; xDecr]; xn([randSwap,n+1]) = xn([n+1,randSwap]);
```

```
yn = [y; yDecr]; yn([randSwap,n+1]) = yn([n+1,randSwap]);

Decrement.AOldSwapped    = feval(covfunc, hyp.cov, xn)+sn2*eye(n+1);
Decrement.AOldInvSwapped = inv(Decrement.AOldSwapped);

disp('Determined hyperparameters, starting regression...');

%% GP regression step
%% Incremental exact inference
tic;
for k=1:NumIt

  mIncr = gp(hyp, @infIncrement, [], covfunc, likfunc, x, y, z);

end
elpsdTIncrExact = toc;
avrgTIncrExactMilis = [avrgTIncrExactMilis (elpsdTIncrExact / NumIt) * 1000];

dIncrExact = mIncr-val;
mseIncrExact = [mseIncrExact (dIncrExact'*dIncrExact)/length(z)];



%% Decremental exact inference with Swap
tic;
for k=1:NumIt

  % (Re-)Swapping before regression step
  l = randSwap;
  hyp.Decrement.c = Decrement.AOldSwapped(l,l);
  hyp.Decrement.b = Decrement.AOldSwapped(1:n,l);
  hyp.Decrement.b(l) = Decrement.AOldSwapped(l,n);
  hyp.Decrement.AOldInv            = Decrement.AOldInvSwapped;
  hyp.Decrement.AOldInv(:,[l,n+1])  = hyp.Decrement.AOldInv(:,[n+1,l]);
  hyp.Decrement.AOldInv([l,n+1],:)  = hyp.Decrement.AOldInv([n+1,l],:);
  yn([l,n+1]) = yn([n+1,l]);
  xn([l,n+1]) = xn([n+1,l]);

  mDecr = gp(hyp, @infDecrement, [], covfunc, likfunc, xn(1:n), yn(1:n), z);

end
elpsdTDecrExact = toc;
avrgTDecrExactMilis = [avrgTDecrExactMilis (elpsdTDecrExact / NumIt) * 1000];

dDecrExact = mDecr-val;
```

```
          mseDecrExact = [mseDecrExact (dDecrExact'*dDecrExact)/length(z)];

          %% Exact inference
          tic;
          for k=1:NumIt
            mExact = gp(hyp, @infExact, [], covfunc, likfunc, x, y, z);
          end
          elpsdTExact = toc;
          avrgTExactMilis = [avrgTExactMilis (elpsdTExact / NumIt) * 1000];

          dExact = mExact-val;
          mseExact = [mseExact (dExact'*dExact)/length(z)];

          %% FITC
          nu = fix(n/2);
          border = width/(2*nu);
          u = linspace(border,width-border,nu)';
          covfuncF = {@covFITC, {covfunc}, u};

          tic;
          for k=1:NumIt
              mFITC = gp(hyp, @infFITC, [], covfuncF, likfunc, x, y, z);
          end
          elpsdTFITC = toc;
          avrgTFITCMilis = [avrgTFITCMilis (elpsdTFITC / NumIt) * 1000];
          dFITC = mFITC-val;
          mseFITC = [mseFITC (dFITC'*dFITC)/length(z)];
          %%
      end

      resultsTime = [resultsTime; avrgTIncrExactMilis; avrgTDecrExactMilis;
avrgTExactMilis; avrgTFITCMilis];
      resultsMse = [resultsMse; mseIncrExact; mseDecrExact; mseExact; mseFITC];
      %Data = struct(tStructNames(i), resultsData);
   end
end
%% Time benchmark graphs

numInfMthds = 4;
numDnsts = length(tDensity);

figure(1)
hold on
plotNumber = 1;
```

```
plotRow = 2;

subplot(2, 2, plotNumber); hold on
t05 = plot(resultsTime(1,:),resultsTime(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
title('point density = 0.5/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Average computing time (in ms)','FontSize',16);
ylim([0 500]);
LEG = legend('Time: Exact inference (incremental)', 'Time: Exact inference (decremental,
swapped)', 'Time: Exact inference', 'Time: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t10 = plot(resultsTime(1,:),resultsTime(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
title('point density = 1.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Average computing time (in ms)','FontSize',16);
ylim([0 500]);
LEG = legend('Time: Exact inference (incremental)', 'Time: Exact inference (decremental,
swapped)', 'Time: Exact inference', 'Time: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t30 = plot(resultsTime(1,:),resultsTime(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
title('point density = 3.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Average computing time (in ms)','FontSize',16);
ylim([0 500]);
LEG = legend('Time: Exact inference (incremental)', 'Time: Exact inference (decremental,
swapped)', 'Time: Exact inference', 'Time: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t50 = plot(resultsTime(1,:),resultsTime(plotRow:numInfMthds+plotRow-1,:));
title('point density = 5.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Average computing time (in ms)','FontSize',16);
ylim([0 500]);
LEG = legend('Time: Exact inference (incremental)', 'Time: Exact inference (decremental,
swapped)', 'Time: Exact inference', 'Time: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
set(gca,'YTick',0:100:500);
grid on

%% Accuracy benchmark graphs
numInfMthds = 4;
numDnsts = length(tDensity);

figure(2)
hold on
plotNumber = 1;
plotRow = 2;

subplot(2, 2, plotNumber); hold on
t05 = plot(resultsMse(1,:),resultsMse(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
title('point density = 0.5/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Mean square error','FontSize',16);
ylim([0 0.5]);
LEG = legend('MSE: Exact inference (incremental)', 'MSE: Exact inference (decremental,
swapped)', 'MSE: Exact inference', 'MSE: FITC Approximation', 'Location', 'SouthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
%set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t10 = plot(resultsMse(1,:),resultsMse(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
```

```
title('point density = 1.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Mean square error','FontSize',16);
ylim([0 0.5]);
LEG = legend('MSE: Exact inference (incremental)', 'MSE: Exact inference (decremental,
swapped)', 'MSE: Exact inference', 'MSE: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
%set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t30 = plot(resultsMse(1,:),resultsMse(plotRow:numInfMthds+plotRow-1,:));
plotRow = plotRow + numInfMthds; plotNumber = plotNumber +1;
title('point density = 3.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Mean square error','FontSize',16);
ylim([0 0.5]);
LEG = legend('MSE: Exact inference (incremental)', 'MSE: Exact inference (decremental,
swapped)', 'MSE: Exact inference', 'MSE: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
%set(gca,'YTick',0:100:500);
grid on

subplot(2, 2, plotNumber); hold on
t50 = plot(resultsMse(1,:),resultsMse(plotRow:numInfMthds+plotRow-1,:));
title('point density = 5.0/unit','FontSize',16);
xlabel('Number of learning points','FontSize',16);
xlim([0 1500]);
ylabel('Mean square error','FontSize',16);
ylim([0 0.5]);
LEG = legend('MSE: Exact inference (incremental)', 'MSE: Exact inference (decremental,
swapped)', 'MSE: Exact inference', 'MSE: FITC Approximation', 'Location', 'NorthWest');
set(LEG,'FontSize',16);
set(gca, 'FontSize', 16);
%set(gca,'YTick',0:100:500);
grid on
```