Project in Artificial Intelligence and Machine Learning

Balancing a pole

Karol Sobierajski 336307
Piotr Sikora

WiSe10/11

Abstract
Balancing a pole is a classical problem in control theory. In this project we made attemt to implement new method based on Gaussian Process Regression and Reinforcement Learning, developed recently by Marc Deisenroth and Carl Rasmussen.

1. INTRODUCTION

The pole-balancing problem requires a close-loop feedback control system with the desired behaviour of an inverted pendulum which is connected to a motor driven cart. The movement of the cart is restricted to the horizontal axis by a track, and the pole is free to move about the horizontal axis of the pivot.The state of the system is defined by four real values; the angle of the pole, the angular velocity of the pole, the position of the cart relative to the centre of the track and the velocity of the cart . The output of the control system is a forward or backward movement for the cart as a fixed force. The pendulum has to be swung up and balanced by just pushing the cart to left and right. In desired state both cart and pole are not moving – and pole is swung up in desired point and is not falling down.
In classical control theory, the solution of the task is usually based on understanding of the system dynamics, by knowing exact ODE equation. In our attempt however, we do not know this equations. System dynamics is learned by experience using reinforcement learning (RL) algorithm.
As a branch of machine learning, reinforcement learning is a computational approach to learning from interactions with the surrounding world and concerned with sequential decision making in unknown environments to achieve high-level goals. Usually, no sophisticated prior knowledge is available and all required information to achieve the goal has to be obtained through trials. No teacher is available. To gather information about the world, the RL agent has to explore it on his own. An agent interacts with its surrounding world by taking actions. In result, the agent perceives sensory inputs that reveal some information about the state of the world. In addition, the agent perceives a reward/penalty signal that reveals information about the quality of the chosen action and the stateof the world. The history of taken actions and perceived information gathered from interacting with the world forms the agent's experience. The agent's objective in RL is to find a sequence of actions, a strategy, that minimizes an expected long-term cost. In the context of control, the world can be identified as the dynamic system, the decision-making algorithm within the agent corresponds to the controller, and actions correspond to control signals.
A central issue in RL is to increase the learning efficiency by reducing the number of interactions with the world that are necessary to find a good strategy. One way to increase the learning efficiency is to extract more useful information from collected experience.
The RL setup requires to automatically extract information and to learn structure from collected data. The learned structure is captured in the form of a statistical model that compactly represents the data. Bayesian data analysis aims to make inferences for quantities about which we wish to learn by using probabilistic models for quantities we observe. Model of the world summarizes collected experience and can be used for generalization and hypothesizing about the consequences in the world of taking a particular action. The model of the world is often described by a transition function that maps state-action pairs to successor states.
Gaussian process regression allows for an appropriate uncertainty treatment in RL when approximating unknown functions. This allows us to reason about things we do not know for sure.

2. Fast Reinforcement Learning Framework

Adaptive probabilistic world model learned on previous experience is used. There are two moajor phases: **interaction** and **simulation**.
Algorithm is initialized by initialization of policy (which is set to random). In the interaction phase, we apply a single action to the real world according to current policy. We observe a new state and employ the policy again. Subsequently, the probabilistic world model is updated using new and historical experience collected during the interaction phases.
During the planning phase, we take a state distribution and simulate the world with the corresponding distribution over actions. The probabilistic world model determines a distribution over successor states. The controller computes the corresponding distribution over costs and the system is being simulated by applying the policy to the entire state distribution.
During learning, the policy will be optimized in the planning phase based onthe evidence so far. However, it may be that because of limited experience the simulations do not correspond to the real world. When this situation occurs and we apply the model-optimized policy to the real system, the model will discover the discrepancy between the model's predictions and the world. This new insight will be incorporated into the subsequent model update.

2.1 Algorithm:

```
1. Set policy to random                                    policy initialization
2. loop                                                    main loop
3.          execute policy                                 interaction with world
4.          record experience
5.          learn prob. dynamic model
6.          loop                                           simulation loop
7.                simulate system with policy π
8.                compute long-term                        policy evaluation
                  expected cost
9.                improve policy
10.         end loop
11. end loop
```

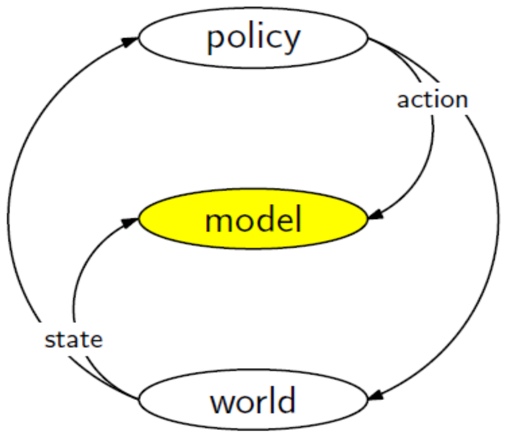
algorithm1: Fast reinforcement learning

First when interacting with real system (line 3) – this is when experience is collected (line 4) and model is updated, update is based on both new and historical data (line 5)

Second when algorithm refines the policy (line 9), it is based on simulation with a probabilistic model (line 7)

Interaction phase:

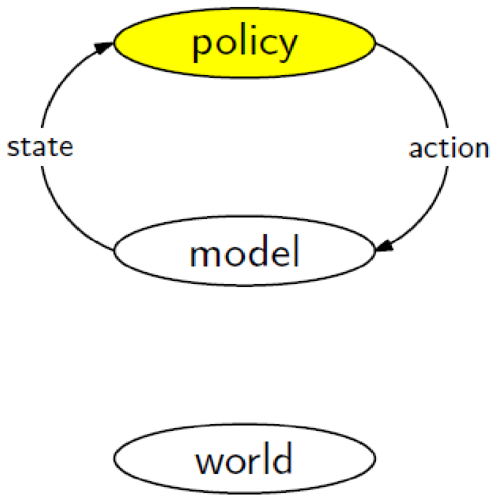
- Action is applied to the real world
- World changes its state and return it to the policy
- Policy selects next action and apply it to real world again
- Model refines itself using actions and states from real world



interaction

Simulation phase:

- Action is applied to the model
- Model simulates real world
- Policy determines actions according to state returned by model and applies again
- Using simulated experience, policy refines itself



simulation

2.2 MODEL:

Learning the dynamics is done using Gaussian process models.The GP dynamics model takes as input a representation of the current state and action. As output the GP computes a representation of the distribution over consecutive states.

2.3 COST FUNCTION:

The state s of the system consists of cart position, cart velocity, angle of the pole , and angular velocity of the pole . $s=[x\,\dot{x}\,\phi\,\dot{\phi}]$

The only feedback the controller gets about the quality of its applied action is the squared distance between the tip of the pendulum and its desired position:

$$d(s)^2=x^2+2\mathrm{xl}\sin(\phi)+2\mathrm{l}^2+2\mathrm{l}^2\cos(\phi)$$

We choose the immediate cost:

$$l(s)=1-\exp(-0.5\,d(s)^2/c^2)$$

where c=0.25m is a constant giving the distance at which the cost function switches between locally quadratic and saturation. The goal cost is zero.

2.4 POLICY:

The controller implements a nonlinear deterministic policy.

A policy denoted as π is a function that maps states to actions. It has parameters ψ. Alorithm searches policy indirectly, using approximate inference for policy evaluation.

For a state distribution we need to be able to compute a corresponding distribution over actions.

Policy must be able to deal with constrained control signals

Policy GP is parameterized by a pseudo-training set, consisting of pairs of states (training inputs) and corresponding actions (training targets). By modifying this training set, we can control the policy being implemented.

Assume a given policy and a given probabilistic dynamics model. To evaluate the quality of the policy starting from a particular state distribution p(s0), the expected cost of the trajectory has to be determined. The expected undiscounted nite-horizon cost:

$$V^{\pi}:=E_{\tau}[\sum_{t=0}^{T} l(s_t)|\pi, p(s_0)]=\sum_{t=0}^{T} E[l(s_t)|\pi, p(s_0)]$$

Expectation is taken with respect to the probability distribution over trajectories. To evaluate horizon we have to compute E for cost function l.

2.5 Policy Optimization

To optimize the policy, we minimize the expected longterm cost V^{π} with respect to the policy parameters using a gradient-basedmethod. Two dierent kinds of parameters are involved in the Gaussian processcontroller.

- hyper-parameters of the kernel
- pseudo training set of the controller itself.

They are treated as free variables to be optimized to find an optimal strategy. They are stored in parameter vector Θ .

Gradient of V^{π} is computed:

$$\frac{dV^{\pi}(\tau)}{d\Theta}=\sum_{t=0}^{T}\frac{d}{d\Theta}E[l(s_t)|\pi_{\Theta},p(s_0)]$$

Next we have to compute:

$$\frac{d}{d\mu_t}E[l(s_t)|\pi_{\Theta},p(s_0)]\frac{d\mu_t}{d\Theta}+(\frac{d}{d\Sigma_t}E[l(s_t)|\pi_{\Theta},p(s_0)])\frac{d\Sigma_t}{d\Theta}$$

for $t=0,...,T, where\, \mu_t,\Sigma_t$ are mean and covariance of $p(s_t)$

3. GAUSSIAN PROCESS REGRESSION

Regression is the problem of estimating a function f given a set of input vectors and observations of the corresponding function values.

Definition 2 (Gaussian process) A Gaussian process is a collection of random variables,any finite number of which have a consistent joint Gaussian distribution

---

A Gaussian process is completely specified by its mean function and co- covariance and variance function:

$m(x)=E[f(x)]$ mean function

$k(x,x')=E[(f(x)-m(x))(f(x')-m(x'))]$ covariance function

Gaussian process can be denoted as:

$f(x)\;\; GP(m(x),k(x,x'))$

Random variables represent the value of the function f(x) at location x.

A simple example of a Gaussian process is linear regression model $f(x)=\phi(x)^T w$ where w are weights with prior $w\;\; N(0,\Sigma_p)$ and $x:\phi(x)=(1,x,x^{2,}x^{3,}...)^T$ is projection of scalar input into polinominal regression. For prediction we need: input - x value for which state were observed, output y – observed state and x* - test set for which prediction is made.

Mean and covariance of this example:

$E[f(x)]=\phi(x)^T E[w]=0$

$E[f(x)f(x')]=\phi(x)^T E[ww^T]\phi(x')=\phi(x)^T \Sigma_p \phi(x')$

We choose squared expotential (SE) as a covariance function.

This cov function specifies covariance between pair of random variables:

$cov(f(x_p), f(x_q))=k(x_p, x_q)=\exp(-0.5 \text{abs}\, x_p - x_q^2)$

Prediction using noisy observations

In realistic modelling we dont have access to function values themselve, but only to noisy versions of them. $y=f(x)+\epsilon$. Prior distribution then becomes:

$cov(y_p, y_q)=k(x_p, x_q)+\sigma_n^2 \delta_{pq}$ or $cov(y)=K(X,X)+\sigma_n^2 I$, where delta is a Kronecker delta which is one if p=q and zero otherwise.

We can then write joint distribuiton of observed target values and function values at test locations under the prior as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X)+\sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

The key predictive equations are:

$$\begin{aligned} \mathbf{f}_*|X,\mathbf{y},X_* &\sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)), \text{ where} \\ \bar{\mathbf{f}}_* &\triangleq \mathbb{E}[\mathbf{f}_*|X,\mathbf{y},X_*] = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}, \\ \text{cov}(\mathbf{f}_*) &= K(X_*,X_*) - K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,X_*). \end{aligned}$$

and when there is a single test point x* we write k(x*)=k* to denote the vector of covariances between the test point and the n training points. Then, using K=K(X,X) we can denote previous formulas as:

$$\begin{aligned} \bar{f}_* &= \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{y}, \\ \mathbb{V}[f_*] &= k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}_*^\top (K+\sigma_n^2 I)^{-1}\mathbf{k}_*. \end{aligned}$$

Covariance functions have some tunable variables which are called hyperparameters. They are different for each function. Right choice of hyperparameters can improve quality of prediction, while bad choice can make prediction completely false.

4. Implementations

4.1 IMPLEMENTED GP REGRESSION FUNCTION:

For implementing gp regression in matlab we used following algorithm, which is practical implementation of theory from previous section. Matlab code is to be found in appendix.

```
Algorithm2
input: inputs, targets, cov function, noise level, test input
1 $L := cholesky(K + \sigma_n^2 I)$
2 $\alpha := L^T \backslash (L \backslash y)$
3 $\bar{f}_* := \mathbf{k}_*^\top \alpha$
4 $v := L \backslash k_*$
5 $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
6 $log p(y|X) := -0.5 y^T \alpha - \Sigma_i \log L_{ii} - 0.5 n \, log 2\pi$
return: predicted mean, predicted variance
```

Results:

We tested our implemented algorithm on simple function – polynomial of sixth degree. We generated few exact "observation" values and added noise. As test set we used values from set range, every 0.1. Test values are far more numerous then observed ones. Then we generated values how output for test set should like. At the end we compared predicted values with comparision values.

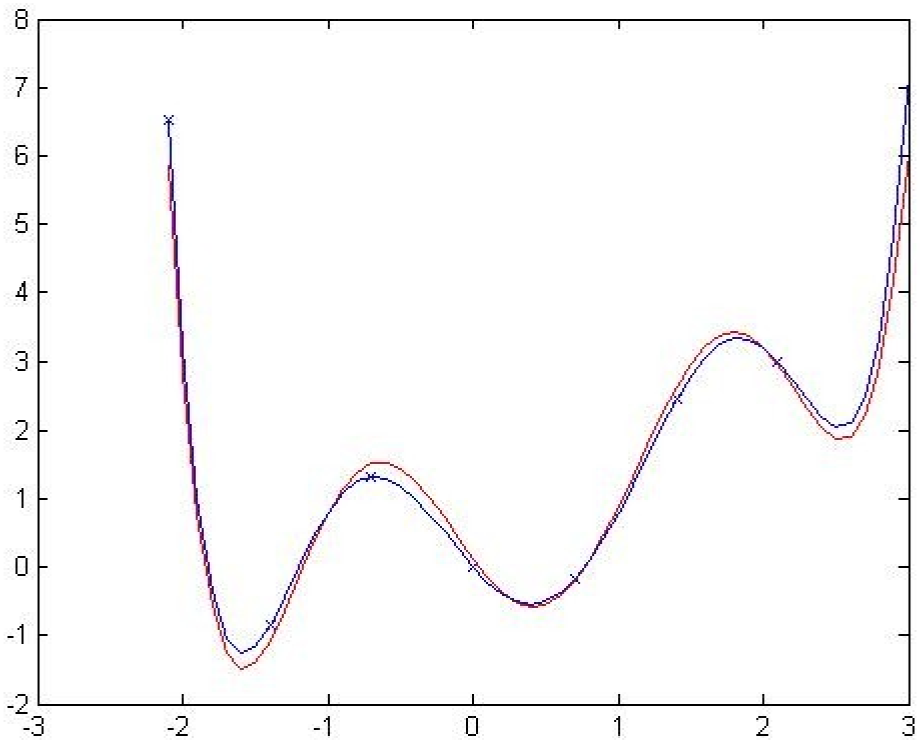


Figure 1 – comparision between prediction and exact values. Blue line – exact value (with x as observations), red – predicted function.

As we can see on figure 1, regression worked good. Predicted function resembles original one, shape is the same, differences are not big and in acceptable range. To achieve this result we chose right hyperparameters. This choice can influence the result.

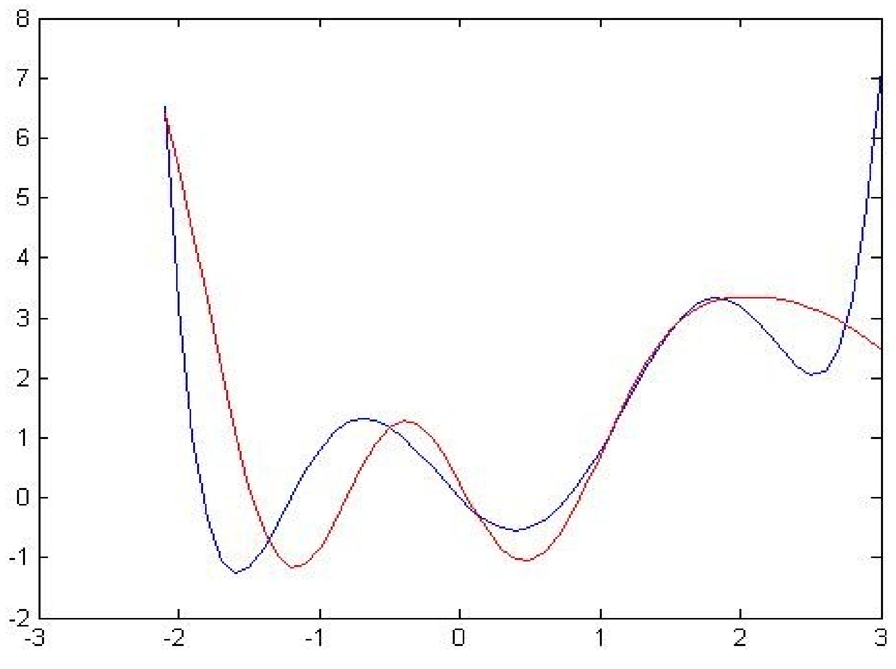


Figure2 – prediction with different hyperparameters

As we can see on figure 2, prediction differs from original function. Reason of this is wrong choice of hyperparameters. However predicted function still resembles original one in some way, shape is more or less the same but function is shifted.

4.2 DYNAMICS PREDICTOR

Next step was to use gaussian process regression to learn system dynamics. We simulated real world using simulink model. Following equation were used:

$$\ddot{x} = \frac{\left(\frac{u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2 \sin\varphi}{I + m(\frac{l}{2})^2} + m^2(\frac{l}{2})^2 g \sin\varphi\cos\varphi\right)}{(M+m)(I + m(\frac{l}{2})^2) - m^2(\frac{l}{2})^2(\cos\varphi)^2},$$

$$\ddot{\varphi} = \frac{m\frac{l}{2}\cos\varphi(u - b\dot{x} + m\frac{l}{2}\dot{\varphi}^2\sin\varphi) + (M+m)gm\frac{l}{2}\sin\varphi}{m^2(\frac{l}{2})^2(\cos\varphi)^2 - (M+m)(I + m(\frac{l}{2})^2)}$$

where x is cart position and φ is pendulum angle, their derivatives are velocities and second derivatives are acceleration.
M = 0.5 kg is the mass of the cart
m = 0.5 kg the mass of the pole,
b = 0.1 Ns/m the friction between cart and ground,
l = 0.6 m the length ofthe pole,
I = 0.06 kg m^2 the moment of inertia around the tip of the pole
g = 9.82 m/s^2 the gravity acceleration

The control signal is restricted to values between -10 and 10 N. U is a horizontal force pushing the cart to left or right and is our only input.

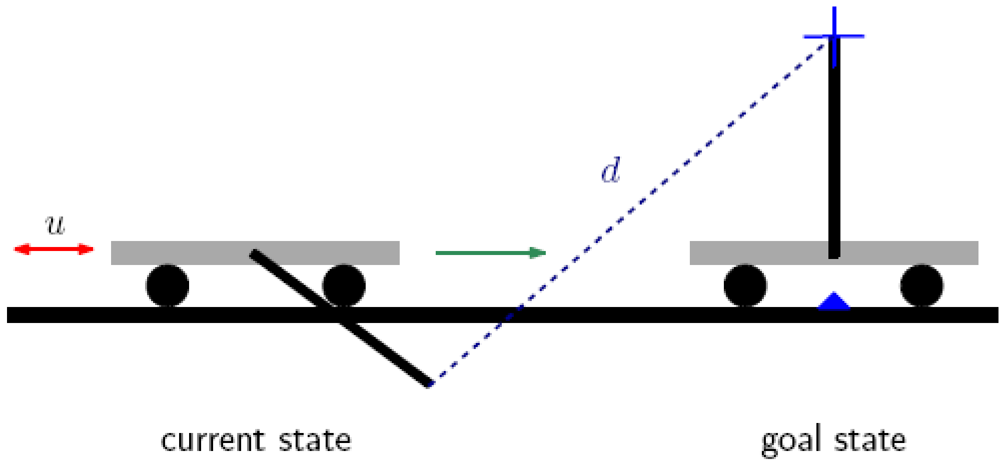


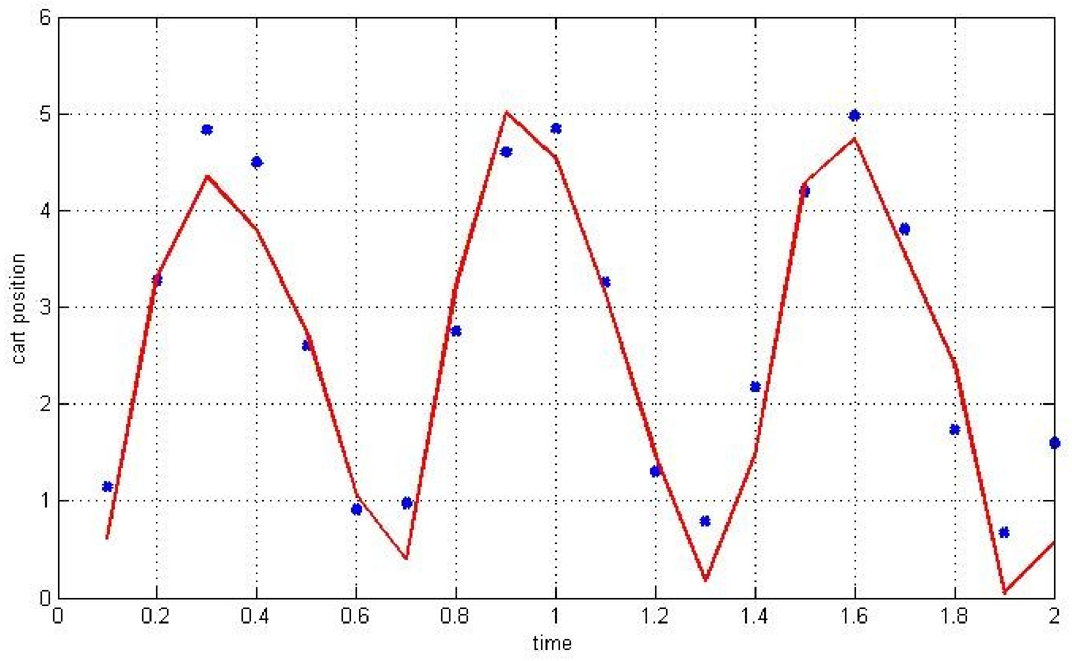
Figure 3 The pendulum has to be swung up and balanced at the cross by just pushing the cart to left and right. The Euclidean distance between the tip of the pendulum and the goal state is shown by the dashed line.

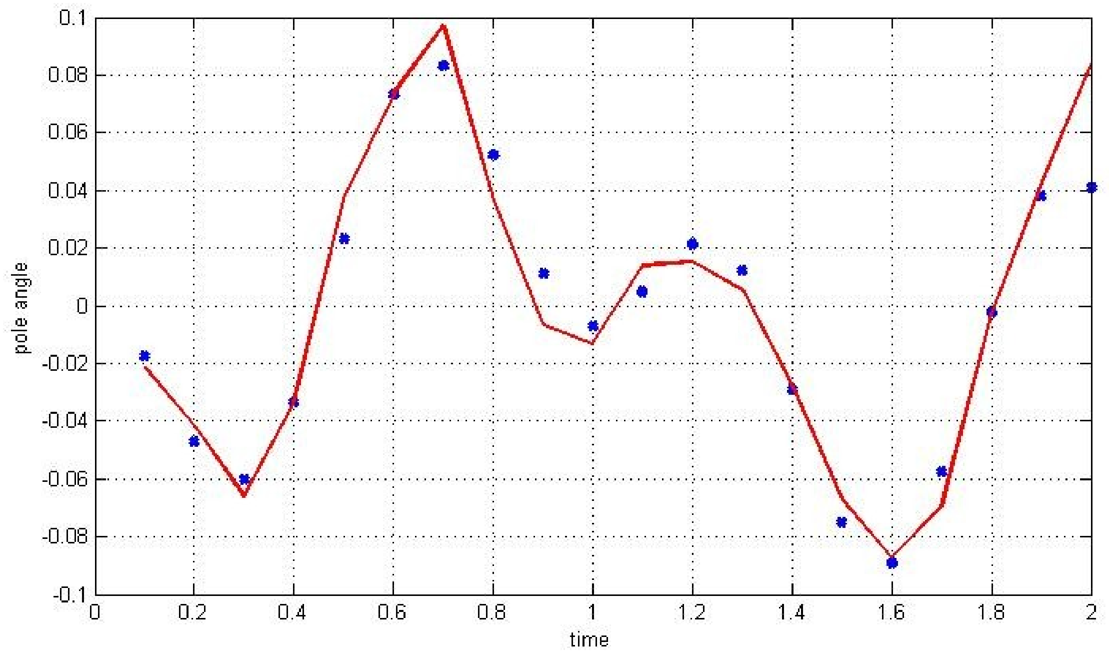
Goal of our predictor is to predict next state, based on current one.

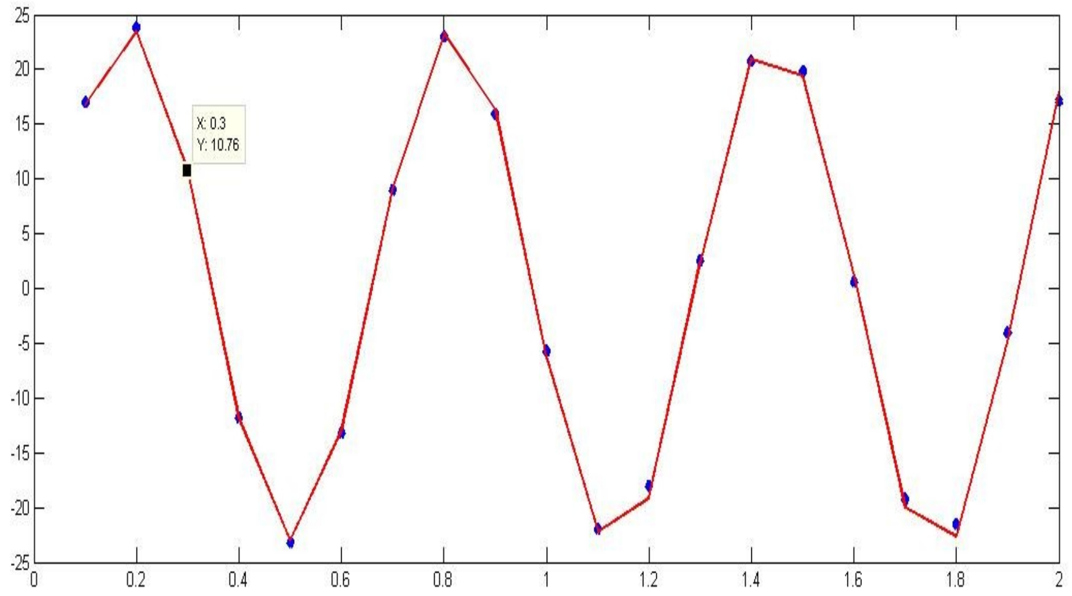
State consists of four variables: cart position, cart velocity, pole angle and pole velocity. Together with control signal at moment n they form one row of input matrix X. Output matrix Y consist of noised next state n+1 only, without control signal. Test input is state n+1 and state n+2 is predicted. As input values we used sinusoidal force (

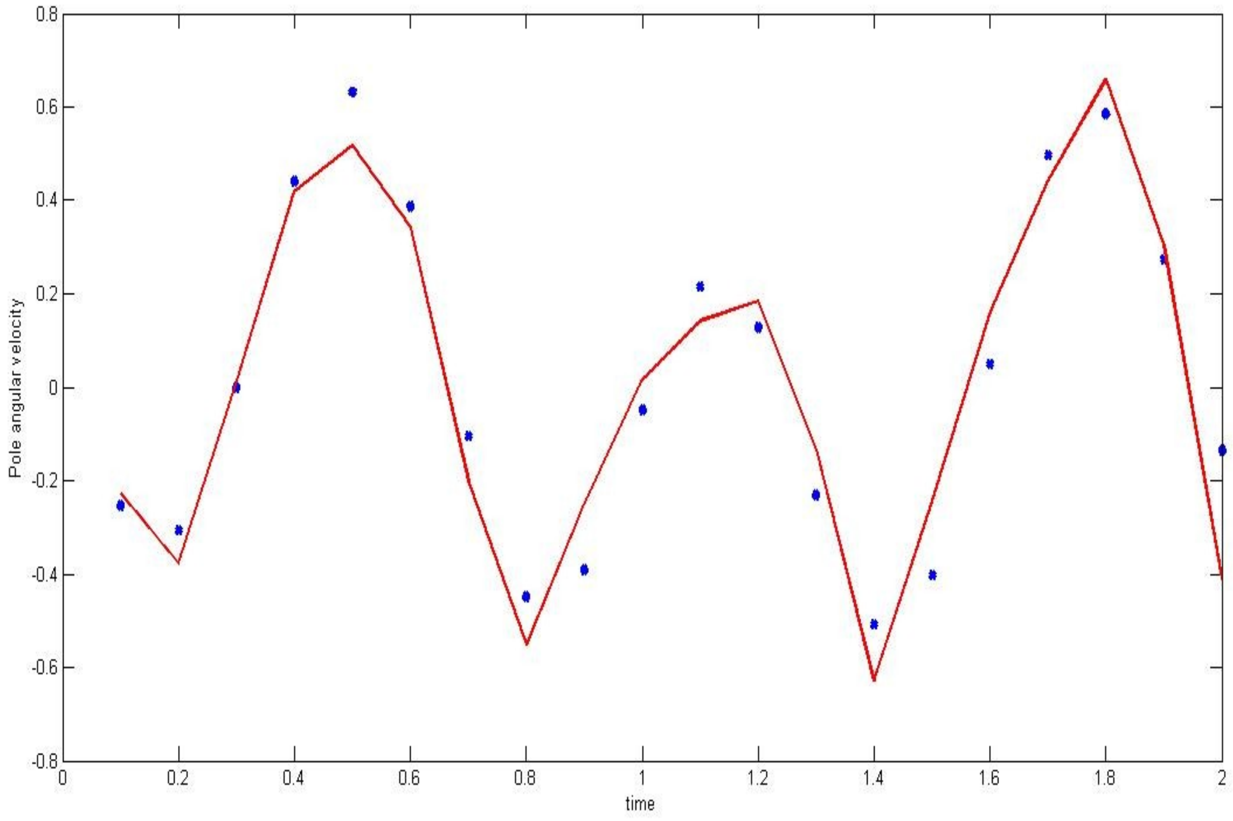
Results:

To see if result is correct, we extracted every variable from state vector and plot them with time. Then they were compared with predicted ones.




pole angle
time


X: 0.3
Y: 10.76



Blue dots represent observed states, red line is predicted function

As we can see, predictor is vorking correctly and we sucessfully predicted next state.

After that we experimented with control signal type – we set it to random. Predictor worked as well. Later on we changed covariance functions. Neural network and quadratic covariance functions also worked fine, but squared expotential roked best. We dont paste plots here, because it looks the similar as ones above.

5. SUMMARY

Gaussian processes are sucessfully used to make predictions. Main advantage of this recently developed method is that does not require prior knowedgle about the system and can be used to nearly any system. It can use the function without actually knowing it and is very fast.

Unrortunately we did not manage to finish the project. Reinforcement Learning part was not done, but we are hoping to finish it in the future.

Table of contents

BIBLIOGRAPHY:

Marc Peter Deisenroth *Ecient Reinforcement Learning using Gaussian Processes*
Carl Edward Rasmussen, Marc Peter Deisenroth *Probabilistic Inference for Fast Learning in Control*
Marc Peter Deisenroth, Jan Peters, Marc Peter Deisenroth *Gaussian Process Dynamic Programming*
C. E. Rasmussen, C. K. I. Williams, *Gaussian Processes for Machine Learning*