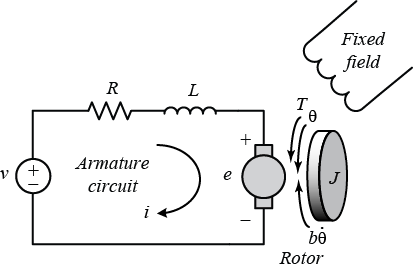
**Intelligent Control Homework 2**

due date: 2022/10/25

1. (**Control System Simulation**) Consider a common actuator in control systems is the DC motor. It directly provides rotary motion and, coupled with wheels or drums and cables, can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure (more details can be found in Automatic Control Text book).



As above, we will assume that the input of the system is the voltage source ($V$) applied to the motor's armature, while the output is the rotational speed of the shaft $\dot{\theta}$. The rotor and shaft are assumed to be rigid. We further assume a viscous friction model, that is, the friction torque is proportional to shaft angular velocity. The physical parameters are given as

*J*: moment of inertia of the rotor 0.01 kg.m2;

b: motor viscous friction constant 0.1 N.m.s;

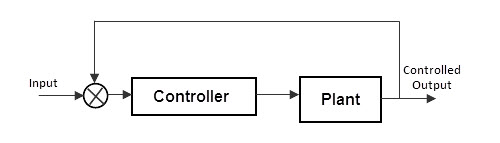
*Ke*: electromotive force constant 0.01 V/rad/sec;

*Kt*: motor torque constant 0.01 N.m/Amp;

*R*: electric resistance 1 Ohm;

*L*: electric inductance 0.5 H.

1. Find the corresponding transfer function of DC motor (*Ω*(*s*)/*V*(*s*));
2. Write the corresponding state space representation;
3. Sketch the corresponding block diagram;
4. Consider the unity-feedback closed-loop control system as follows, and the controller is P controller, let us utilize the delta learning to implement the adaptive control for DC motor, i.e., the motor system is first digitalized; and the adaptive control signal is ** and **. Note that the learning rate should be selected properly.



In the following discussion, simulations using MATLAB or SIMULINK are encouraged; and proper parameters can be chosen according to the literature or Web site, please given your reference.

1. As the block diagram shown in Lecture 1 (page 9), when the current, velocity, and position loops are selected to be P controller, please discuss the advantage of multiple loop control in bandwidth).
2. As above (consider the practical used in commercial products), when the current, and velocity loops are selected to be PI controller, and position loop is P controller, please discuss the advantage of multiple loop control in bandwidth).
3. Repeat again, how about all controller are PID terms.

2. (**Implementing perceptron algorithm**) Let’s consider the linear classiﬁcation model with a bias term,

, where sign(*v*) outputs +1 if *v*≥0, and -1 otherwise. The perceptron learning algorithm can be expressed as follows (for a given dataset *D*={x*i***,** *yi*}, *i*=1, …, n, x*i*∈ℜ*d*, *yi*∈{+1, -1}):

|  |
| --- |
| Perceptron algorithm on training samples  Let **w** = **0**, i = 0, *numIter* = 100  while !(all samples correctly classiﬁed) AND *i < numIter* do  for *s* = 1, 2, ..*n* do  if *y*s(**wTxs** + *b*) ≤ 0 then  **w** = **w** + *y*s**x**s;  *b* = *b* + *y*s ;  end  end  *i* = *i* + 1 ;  end  return **w**, *b*, *i* |

This algorithm can be used to obtain **w** and *b*, which can then be used to predict labels of the test samples using . Implement this algorithm for the task of classiﬁcation on the dataset given here (ﬁlenames with ‘parta’ preﬁx). Report your accuracy on the test set, and the number of iterations it took the algorithm to converge on the training set.

// <http://terrence.logdown.com/posts/290508-python-simple-perceptron-learning-algorithm-implementations>

3. (**Implementing kernel perceptron algorithm**) Let’s consider a kernel-deﬁned feature space φ(**x**) (φ(**x**) ∈ ***R****E*, *E* >> *d*), and the linear classiﬁcation model in that space. More precisely, the model is deﬁned as:

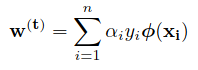
 (\*)

(Note, that we are not considering the bias term in the new feature space). So in order to apply the perceptron learning algorithm, the update rule will have to be modiﬁed. The update rule then becomes

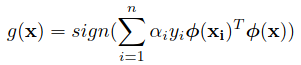
**w**(*k*+1) = **w**(*k*)+*yiφ*(**x***i*)

or the misclassiﬁed example (**x***i*, *yi*). But this generally turns out to be computationally expensive, so we consider the dual version of the problem. Assuming n dual variables αi corresponding to each training input vector (which are initialized to 0), the dual update rule is *αi = αi*+ 1 (for the above *i*th misclassiﬁed sample).

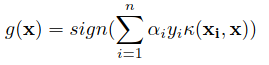
The weight vector (at iteration t) can be expressed as:



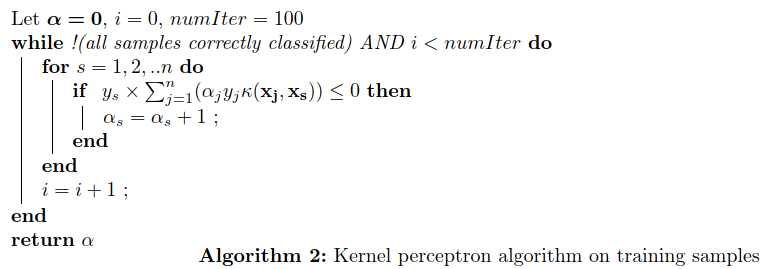
Subsequently, the function g (from equation (\*)), for an input x, can be written as:



(Note, **x**i is the *i*th point in the dataset). Now, we can see that we can use kernel inner products instead of using φ(**x**) directly. Moreover, we don’t need an explicit form of the weight vector to compute *g*(**x**), it can written as:



In conclusion, we only need values of *αi* for predicting labels for test samples. The kernel perceptron algorithm can be written as follows (note: *α* ∈ *Rn*, *αi* is *i*th component (hence a scalar) ):



<https://gist.github.com/mblondel/656147>