8. Approximate Policy Iteration

PDF Version

Note: On March 13, 2021, these notes were updated as follows:

1. Tighter bounds are derived; the old analysis was based on bounding ; the new analysis directly bounds , which leads to a better dependence on the approximation error;
2. Unbiased return estimates are introduced that use rollouts of random length.

One simple idea to use function approximation in MDP planning is to take a planning method that uses internal value functions and add a constraint that restrict the value functions to have a compressed representation.

As usual, two questions arise:

- Does this lead to an **efficient** planner? That is, can the computation be carried out in time polynomial in the relevant quantities, but not the size of the state space? In the case of linear functions the question is whether we can calculate the coefficients efficiently.
- Does this lead to an **effective** planner? In particular, how good a policy can we arrive at with a limited compute effort?

In this lecture, as a start into exploring the use of value function approximation in planning, we look at modifying policy iteration in the above described way. The resulting algorithm belongs to the family of **approximate policy iteration** algorithms, which consists of all algorithms derived from policy iteration by adding approximation to it.

We will work with linear function approximation. In particular, we will assume that the planner is given as a hint a feature-map . In this setting, since policy iteration hinges upon evaluating the policies obtained, the hint given to the planner is

considered to be "good" if the (action-)value functions of **all** policies are well-represented with the features.

This means, that we will work under assumption B2 from the previous lecture, which we copy here for convenience. In what follows we fix .

Assumption B2 (approximate universal value function realizibility) The MDP and the featuremap are such that for any memoryless policy of the MDP, .

Recall that here the notation means that can be approximated up to a uniform error of using linear combinations of the basis functions underlying the feature-map :

For any policy ,

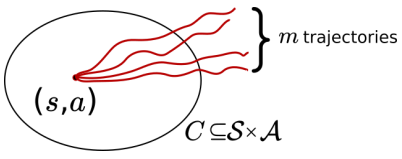
One may question whether it is reasonable to expect that the value functions of all policies can be compressed. We will come back to this question later.

Approximate Policy Evaluation: Done Well

Recall that in phase of policy iteration, given a policy , the next policy is obtained as the policy that is greedy with respect to . If we found some coefficients such that

then when it comes to "using" policy , we could just use when an action is needed at state . Note that this action can be obtained at the cost of elementary operations, a small overhead compared to a table lookup (with idealized access times).

Hence, the main question is how to obtain this parameter in an efficient manner. To be more precise, here we want to control the uniform error committed in approximating .



To simplify the notation, let . A simple idea is **rolling out** with the policy from a fixed set to "approximately" measure the value of at the pairs in . For concreteness, let . Rolling out with policy this pair means using the simulator to simulate what would happen if we used policy for a number of consecutive time steps when the initial state is , the first

action , but for subsequent time steps the actions are chosen using policy for whatever states are encountered. If the simulation goes on for steps, this way we get trajectories starting in . For let the trajectory obtained be . Thus,

,

where , , and for , , and

. The figure on the right illustrates these trajectories.

Given these trajectories, the empirical mean of the discounted sum of rewards along these trajectories is used for approximating :

—

Under the usual condition that the rewards are in the interval, the expected value of is in the vicinity of the and by averaging a large number of independent trajectories, we also achieve that the empirical means are tightly concentrated around their mean.

Using a randomization device, it is possible to remove the error ("bias") introduced by truncating the trajectories at a fixed time. For this, just let be independent **geometrically distributed** random variables with parameter , which are also independently chosen from the trajectories. By definition is the number of - parameter Bernoulli trials needed to get one success. With the help of these variables, define now by

—

Note that in the expression of the discount factor is eliminated. To calculate one can just perform a rollout with policy as before, just in each time step , after obtaining , draw a Bernoulli variable with parameter to decide whether the rollout should continue.

To see why the above definition works, fix and note that by definition, for , and thus . Therefore,

All in all, this means, that we expect that if we solve for the **least-squares problem**

we expect to be a good approximation to . Or at least, we can expect this hold at the points of , where we are taking our measurements. The question is what happens **outside of** : That is, what guarantees can we get for **extrapolating** to points of . The first thing to observe that unless we are choosing carefully, there is no guarantee about the extrapolation error will be kept under control. In fact, if the choice of is so unfortunate that all the feature vectors for points in are **identical**, the least-squares problem will have many solutions.

Our next lemma gives an explicit error bound on the extrapolation error. For the coming results we slightly generalize least-squares by introducing a weighting of the various errors in . For this, let be a weighting function assigning a positive weight to the various error terms and let

be the minimizer of the resulting weighted squared-loss. A simple calculation gives that provided the (weighted) **moment matrix**

is nonsingular, the solution to the above weighted least-squares problem is unique and is equal to

From this expression we see that there is no loss of generality in assuming that the weights in the weighting function sum to one: . We will denote this by writing (here, refers to the fact that we can see as an element of a simplex). To state the lemma recall the notation that for a positive definite, matrix and vector ,

Lemma (extrapolation error control in least-squares): Fix any , , and such that the moment matrix is nonsingular. Define

Then, for any we have

Before the proof note that what his lemma tells us is that as long as we guarantee that the moment matrix is full rank, the extrapolation errors relative to predicting with some can be controlled by controlling

1. the value of ; and
2. the maximum deviation of the targets used in the weighted least-squares problem and the predictions with .

Proof: First, we relate to :

Then for a fixed ,

To get a sense of how to control the sum notice that if in the last sum was somehow replaced by , using the definition of could greatly simplify the last expression. To get here, one may further notice that having the term in absolute value squared would help. Now, to get the squares, recall Jensen's inequality, which states that for any convex function and probability distribution , . Of course, this also works when is a finitely supported, which is the case here. Thus, applying Jensen's inequality with , we thus get

Plugging this back into the previous inequality gives the desired result.

It remains to be seen of whether can be kept under control. This is the subject of a classic result of Kiefer and Wolfowitz:

Theorem (Kiefer-Wolfowitz): Let be finite. Let be such that the underlying feature matrix is rank . There exists a set and a distribution over this set, i.e. , such that

1 ;

2 ;

3 In the previous line, the inequality is achieved with equality and the value of $\bar{}$ is best possible under all possible choices of and .

We will not give a proof of the theorem, but we give references at the end where the reader can look up the proof. When is not full rank (i.e., is not rank), one may reduce the dimensionality (and the cardinality of reduces accordingly). The problem of choosing and such that is minimized is called the -optimal design problem in statistics. This is a specific instance of optimal experimental design.

Combining the Kiefer-Wolfowitz theorem with the previous lemma shows that least-squares amplifies the "measurement errors" by at most a factor of $\bar{}$:

Corollary (extrapolation error control in least-squares via optimal design): Fix any full rank. Then, there exists a set with at most elements and a weighting function such that for any and any ,

$\bar{}$

where is given by

Importantly, note that **and are chosen independently of and** , that is, they are independent of the target. This suggests that in approximate policy evaluation, one should choose as in the Kiefer-Wolfowitz theorem and use the weighted moment matrix. This leads to

where is defined by Eq. and is defined by Eq. . We call this procedure **least-square policy evaluation based on rollouts from -optimal design points**, or LSPE- , for short. Note that we stick to the truncated rollouts, because this allows a simpler probabilistic analysis. That this properly controls the extrapolation error is as attested by the next result:

Lemma (LSPE- extrapolation error control): Fix any full-rank feature-map and take the set and the weighting function as in the Kiefer-Wolfowitz theorem. Fix an arbitrary policy and let and such that and assume that immediate rewards belong to the interval . Let be as in Eq. . Then, for any , with probability ,

Notice that that from the Kiefer-Wolfowitz theorem, and therefore nothing in the above expression depends on the size of the state space. Now, say we want to make the above error bound at most with some value of . From the above we see that it suffices to choose and so that

This, together with gives

Proof: In a nutshell, we use the previous corollary, together with Hoeffding's inequality and using that , which follows since the rewards are bounded in .

▶ Click here for the full proof.

In summary, what we have shown so far is that if the features can approximate well the action-value function of a policy, then there is a simple procedure (Monte-Carlo rollouts and least-squares estimation based on an optimal experimental design) to produce an reliable estimate of the action-value function of the policy. The question remains whether if we use these estimates in policy iteration, the whole procedure will still give good policies after a sufficiently large number of iterations.

Progress Lemma with Approximation Errors

Here we give a refinement of the geometric progress lemma of policy iteration that allows for "approximate" policy improvement steps. This previous lemma stated that the value function of the improved policy is at least as large as the Bellman operator applied to the value function of the policy to be improved. Our new lemma is as follows:

Lemma (Geometric progress lemma with approximate policy improvement): Consider a memoryless policy and its corresponding value function . Let be any policy and define via

Then,

Proof: First note that for the optimal policy , . We have

Using the value difference identity and that , we calculate

where the inequality follows because , the sum of positive linear operators, is a positive linear operator itself and hence is also monotone. Plugging the inequality obtained into gives

Taking the maximum norm of both sides and using the triangle inequality and that gives the desired result.

Approximate Policy Iteration

Notice that the progress lemma makes no assumptions about the origin of the errors. This motivates considering a generic version of **approximate policy iteration** where for in the th update set, the new policy is approximately greedy with respect to in that sense that

The progress lemma implies that the resulting sequence of policies will have value functions that converge to a neighborhood of where the size of the neighborhood is governed by the magnitude of the error terms .

Theorem (Approximate Policy Iteration): Let , be such that holds for all . Then, for any ,

---

Proof: Left as an exercise.

Consider now a version of approximate policy iteration where the sequence of policies is defined as follows:

That is, for each , is greedy with respect to .

Corollary (Approximate Policy Iteration with Approximate Action-value Functions): The sequence defined in is such that

---

Proof: To simplify the notation consider policies and functions over the state-action space such that and . We have

where we used that is linear, monotone, and that is monotone, and both are nonexpansions in the maximum norm.

Hence, if is defined by then and the result follows from the previous theorem.

Global planning with least-squares policy iteration

Putting things together gives the following planning method:

1 Given the feature map , find and as in the Kiefer-Wolfowitz theorem
2 Let
3 For do
4 Roll out with policy for steps to get the targets where and
5 Solve the weighted least-squares problem given by Eq. to get .
6 Return

We call this method least-squares policy iteration (LSPI) for obvious reasons. Note that this is a **global planning method**: The method makes no use of an input state and the parameter vector returned can be used to get the policy (as in the method above).

Theorem (LSPI performance): Fix an arbitrary full rank feature-map and let . Assume that B2 holds. Then, for any , with probability at least , the policy which is greedy with respect to is -suboptimal with

In particular, for any , choosing so that

policy is -optimal with

$$\frac{}{}$$

while the total computation cost is .

Thus, with a polynomial cost, LSPI with the specific configuration at the cost of polynomial computation cost, but importantly, with a cost that is independent of the size of the state space, can result in a good policy as long as , the worst-case error of approximating action-value functions of policies using the features provided, is sufficiently small.

Proof: Note that B2 and that is full rank implies that for any memoryless policy there exists a parameter vector such that (cf. Part 2 of Question 3 of Assignment 2). Hence, we can use the "LSPE extrapolation error bound" (cf.). By this result, a union bound and of course by B2 , we get that for any , with probability at least , for any ,

$$\frac{}{}$$

where we also used that . Call the quantity on the right-hand side in the above inequality .

Take the event when the above inequalities hold and for now assume this event holds. By the previous theorem, is -optimal with

$$\frac{}{}$$

To obtain the second part of the result, we split into two equal parts: is set to force the iteration error to be at most , while and are chosen to force the policy evaluation error to be at most . Here, to choose and , is again split into two equal parts. The details of this calculation are left to the reader.

Notes

Approximate Dynamic Programming (ADP)

Value iteration and policy iteration are specific instances of dynamic programming methods. In general, dynamic programming refers to methods that use value functions to calculate good policies. In **approximate dynamic programming** the methods are modified by introducing "errors" when calculating the values. The idea is that the origin of the errors does not matter (e.g., whether they come due to imperfect function approximation, linear, or nonlinear, or due to the sampling): The analysis is done in a general form. While here we met approximate policy iteration, one can also use the same ideas as shown here to study an approximate version of value iteration. A homework in problem set 2 asks you to study this method, which is usualy called **approximate value iteration**. In an earlier homework you were asked to study how linear programming can also be used to compute optimal value functions. Adding approximations we then get **approximate linear programming**.

What function approximation technique to use?

We note in passing that fans of neural networks should like that the general, ADP-style results, like the theorem in the middle of this lecture, can be also applied to the case when neural networks are used as the function approximation technique. However, one main lesson of the lecture is that to control extrapolation errors, one should be quite careful in how the training data is chosen. For linear prediction and least-squares fitting, optimal design gives a complete answer, but the analog questions are completely open in the case of nonlinear function approximation, such as neural networks. There is also a sizable literature that connects nonparametric techniques (an analysis friendly relative of neural networks) to ADP methods.

Concentrability coefficients and all that jazz

The idea of introducing approximate calculations has been introduced at the same time people got interested in Markov Decision Processes in the 1960s. Hence, the literature is quite enormous. However, the approach taken here which asks for error bounds where the algorithmic (not approximation-) error is uniformly controlled regardless of the MDP is quite recent and where the term that involves the approximation error is also uniformly bounded (for a fixed dimension and discount factor).

Earlier literature often presented bounds where the magnification factor of the approximation and the algorithmic error involved terms which depended on the MDP. Often these came in the form of "concentrability coefficients" (and yours truly was quite busy with working on these results a while ago). The main conclusion of this earlier analysis is that more stochasticity in the transitions means less control, less concentrability, which is advantageous for the ADP algorithms. While this makes sense and this indicates that these earlier results are complementary to the results presented here, the issue is that these

results are quite pessimistic for example when the MDP is deterministic (as in this case the concentrability coefficients can be as large as the size of the state space).

While here we emphasized the importance of using a good design to control the extrapolation errors, in these earlier results, no optimal design was used. The upshot is that this saves the effort of coming up with a good design, but the obvious downside is that the extrapolation error may become uncontrolled. In the batch setting (which we will come back to later), of course, there is no way to control the sample collection, and this is in fact the setting where this earlier analysis was done.

The strength of hints

A critical assumption in the analysis of API was that the approximation error is controlled uniformly for all policies. This feels limiting. Yet, there are some interesting sufficient conditions when this assumption is clearly satisfied. In general, these require that the transition dynamics and the reward are both "compressible". For example, if the MDP is such that , the immediate reward as a function of the state-action pairs satisfies and the transition matrix, satisfies with some matrix , then for any policy policy , has a range which is a subset of . Since is the fixed-point of , i.e., , it follows that is also necessarily in the range space of . As such, and . MDPs that satisfy the above two constraints are called **linear in** (or sometimes, just "linear MDPs"). Exact linearity can be relaxed: If and , then for any policy , with —— . Nevertheless, later we will investigate whether this assumption can be relaxed.

The tightness of the bounds

It is not known whether the bound presented in the final result is tight. In fact, the dependence of on the is almost certainly not tight; in similar scenarios it has been shown in the past that replacing Hoeffding's inequality with Bernstein's inequality allows the reduction of this factor. It is more interesting whether the amplification factor of the approximation error, , is best possible. In the next lecture we will show that the approximation error amplification factor cannot be removed while keeping the runtime under control. In a later lecture, we will show that the dependence on cannot be improved either – at least for this algorithm. However, we will see that if the main concern is the amplification of the approximation error, while keeping the runtime polynomial (perhaps with a higher order though) then under B2 better algorithms exist.

The cost of optimal experimental design

The careful reader would not miss that to run the proposed method one needs to find the set and the weighting function . The first observation here is that it is not crucial to find the best possible pair. The Kiefer-Wolfowitz theorem showed that with this best possible choice, . However, if one finds a pair such that , the price of this is that wherever appears in the final performance bound, a submultiplicative factor of will also need to be introduced. This should be acceptable. In relation to this note that by relaxing this optimality requirement, the cardinality of can be reduced. For example, by introducing the factor of as suggested above allows one to reduce the cardinlity to ; which may actually be a good tradeoff as this can save much on the runtime.

However, the question still remains of who computes these (approximately) optimal designs and at what cost. While this calculation only needs to be done once and is independent of the MDP (just depends on the feature map), the value of these methods remains unclear because of this compute cost. General methods to compute approximately optimal designs needed here are known, but their runtime for our case will be proportional to the number of state-action pairs. In the very rare cases when simulating transitions is very costly but the number of state-action pairs is not too high, this may be a viable option. However, these cases are rare. For special choices of the feature-map, optimal designs may be known. However, this reduces the general applicability of the method presented here. Thus, a major question is whether the optimal experimental design can be avoided. What is known is that for linear prediction with least-squares, clearly, they cannot be avoided. One suspects that this is true more generally.

Can optimal designs be avoided while keeping the results essentially unchanged? Of particular interest would be if the feature-map would also be only "locally explored" as the planner interacts with the simulator. Altogether, one suspects that two factors contributed here for the appearance of optimal experimental design: One factor is that the planner is global: It comes up with a parameter vector that leads to a policy that can be used regardless of the state. The other (perhaps) factor is that the approach was based on simple "patching up" a dynamic programming algorithm with a function approximator. While this is a common approach, controlling the extrapolation errors in this approach is critical and is likely only possible with something like an optimal experimental design. As we shall see soon, there are indeed approaches that avoid the optimal experimental design step and which are based on online planning and they also deviate from the ADP approach.

Policy evaluation alternatives

The policy evaluation method presented here feels unsophisticated. It uses simple Monte-Carlo rollouts, with truncation, averaging and least-squares regression. The reinforcement learning literature offers many alternatives, such as the "temporal difference" learning

type methods that are based on solving the fixed point equation . One can indeed try to use this equation to avoid the crude Monte-Carlo approach presented here, in the hope of reducing the variance (which is currently rather crudely upper bounded using the term in the Hoeffding bound). Rewriting the fixed point as , and then plugging in , we see that the trouble is that to control the extrapolation errors, the optimal design must likely depend on the policy to be evaluated (because of the appearance of).

Alternative error control: Bellman residuals

Let and be so that

Here, is called the "Bellman residual" of . The policy evaluation alternatives above aim at controlling these residuals. The reader is invited to derive the analogue of the "approximate policy iteration" error bound in for this scenario.

The role of in the Kiefer-Wolfowitz result

One may wonder about how critical is the presence of in the results presented. For this, we can say that it is not critical. Unweighted least-squares does not perform much worse.

Least-squares error bound

The error bound presented for least-squares does not use the full power of randomness. When part of the errors with are random, some helpful averaging effects can appear, which we ignored for now, but which could be used in a more refined analysis.

Optimal experimental design – a field on its own

Optimal exoerimental design is a subfield of statistics. The design considered here is just one possibility. In fact, this design which is called G-optimal design (G stands, uninspiringly, for the word "general"). The Kiefer-Wolfowitz theorem actually also states that this is equivalent to the D-optimal designs.

Lack of convergence

The results presented show convergence to a ball around the optimal target. Some people think this is a major concern. While having a convergent method may look more appealing, as long as one controls the size of the ball, I will not be too concerned.

Approximate value iteration (AVI)

Similarly to what is done here, one can introduce an approximate version of value-iteration. This is the subject of Question 3 of homework 2. While the conditions are different, the qualitative behavior of AVI is similar to that of approximate policy iteration.

In particular, as for approximate policy iteration, there are two steps to this proof: One is to show that the residuals can be controlled and the second is that if they are controlled then the policy that is greedy with respect to (say) is -optimal with controlled by . For this second part, we have the following bound:

where . The procedure that uses least-squares fitting to get the iterates is known under various names, such as **least-squares value iteration** (LSVI), **fitted Q-iteration** (FQI), **least-squares Q iteration** (LSQI). This proliferation of abbreviations and names is unfortunate, but there is not much that can be done at this stage. To add insult to injury, when neural networks are used to represent the iterates and an incremental stochastic gradient descent algorithm is used for "fitting" the weights of these networks by resampling old data from a "replay buffer", the resulting procedure is coined "Deep Q-Networks" (training), or DQN for short.

Bounds on the parameter vector

The Kiefer-Wolfowitz theorem implies the following:

Proposition: Let and be such that and . Then, there exist a matrix such that for

there exists such that the following hold:

1 , ;

2 ;

3 .

Proof: Let be the -optimal design whose existence is guaranteed by the Kiefer-Wolfowitz theorem. Let be the underlying moment matrix. Then, by the definition of , .

Define and . The first property is clearly satisfied. As to the second property,

Finally, for the third property,

finishing the proof.

Thus, if one has access to the full feature-map then knowing that a function realized is bounded, one may as well assume that the feature map is bounded and the parameter vector is bounded just by .

Regularized least-squares

The linear least-squares predictor given by a feature-map and data predicts a response at via where

with

Here, by abusing notation for the sake of minimizing clutter, we use , . The problem is that may not be invertible (i.e., may not be defined as written above). "By continuity", it is nearly equally problematic when is ill-conditioned (i.e., its minimum eigenvalue is "much smaller" than its maximum eigenvalue). In fact, this leads to poor "generalization". One remedy, often used, is to modify by shifting it with a small constant multiple of the identity matrix:

Here, is a tuning parameter, whose value is often chosen based on cross-validation or with a similar process. The modification guarantees that is invertible and it overall improves the quality of predictions, especially when is tuned base on data.

Above, the choice of the identity matrix, while is common in the literature, is completely arbitrary. In particular, invertibility will be guaranteed if is replaced with any other positive definite matrix . In fact, the matrix one should use here should be one that makes small (while, say, keeping the minimum eigenvalue of at constant). That this is the choice that makes sense can be argued for by noting that with

the vector defined in is the minimizer of

and thus, the extra penalty has the least impact for the choice of that makes the norm of the smallest. If we only know that , by our previous note, a good choice is , where where is a -optimal design. Indeed, with this choice, . Note also that if we apply the feature-standardization transformation of the previous note, we have

showing that the choice of using the identity matrix is justified when the features are standardized as in the proposition of the previous note.

References

We will only scratch the surface now; expect more references to be added later.

The bulk of this lecture is based on

- Tor Lattimore, Csaba Szepesvári, and Gellért Weisz. 2020. “Learning with Good Feature Representations in Bandits and in RL with a Generative Model.” ICML and

arXiv:1911.07676,

who introduced the idea of using -optimal designs for controlling the extrapolation errors. A very early reference on error bounds in "approximate dynamic programming" is the following:

- Whitt, Ward. 1979. "Approximations of Dynamic Programs, II." Mathematics of Operations Research 4 (2): 179–85.

The analysis of the generic form of approximate policy iteration is a refinement of Proposition 6.2 from the book of Bertsekas and Tsitsiklis:

- Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-Dynamic Programming. Athena Scientific, Belmont, Massachusetts, 1996.

However, there are some differences between the "API" theorem presented here and Proposition 6.2. In particular, the theorem presented here appears to capture all sources of errors in a general way, while Proposition 6.2 is concerned with value function approximation errors and errors introduced in the "greedification step". The form adopted here appears, for example, in Theorem 1 of a technical report of Scherrer, who also gives earlier references:

- Scherrer, Bruno. 2013. "On the Performance Bounds of Some Policy Search Dynamic Programming Algorithms." arxiv.

The earliest of these references is perhaps

- Munos, R. 2003. "Error Bounds for Approximate Policy Iteration." ICML.

Least-squares policy iteration appears in

- Lagoudakis, M. G. and Parr, R. Least-squares policy iteration. The Journal of Machine Learning Re-search, 4:1107–1149, 2003.

The particular form presented in this work though uses value function approximation based on minimizing the Bellman residuals (using the so-called LSTD method).

Two books that advocate the ADP approach:

- Powell, Warren B. 2011. Approximate Dynamic Programming. Solving the Curses of Dimensionality. Hoboken, NJ, USA: John Wiley & Sons, Inc.
- Lewis, Frank L., and Derong Liu. 2013. Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. Hoboken, NJ, USA: John Wiley & Sons, Inc.

And a chapter:

- Bertsekas, Dimitri P. 2009. “Chapter 6: Approximate Dynamic Programming,” January, 1–118.

A paper that is concerned with API and least-squares methods, but uses concentrability is:

Antos, Andras, Csaba Szepesvári, and Rémi Munos. 2007. “Learning near-Optimal Policies with Bellman-Residual Minimization Based Fitted Policy Iteration and a Single Sample Path.” Machine Learning 71 (1): 89–129.

Optimal experimental design has a large literature. A nice book concerned with computation is this:

- M. J. Todd. Minimum-volume ellipsoids: Theory and algorithms. SIAM, 2016.

The Kiefer-Wolfowitz theorem is from:

- J. Kiefer and J. Wolfowitz. The equivalence of two extremum problems. Canadian Journal of Mathematics, 12(5):363–365, 1960.

More on computation here:

- E. Hazan, Z. Karnin, and R. Meka. Volumetric spanners: an efficient exploration basis for learning. Journal of Machine Learning Research, 17(119):1–34, 2016
- M. Grötschel, L. Lovász, and A. Schrijver. Geometric algorithms and combinatorial optimization, volume 2. Springer Science & Business Media, 2012.

The latter book is a very good general starting point for convex optimization.

That the features are standardized as shown in the notes is assumed (and discussed), e.g., in

- Wang, Ruosong, Dean P. Foster, and Sham M. Kakade. 2020. “What Are the Statistical Limits of Offline RL with Linear Function Approximation?” arXiv [cs.LG]. arXiv

which we will meet later.