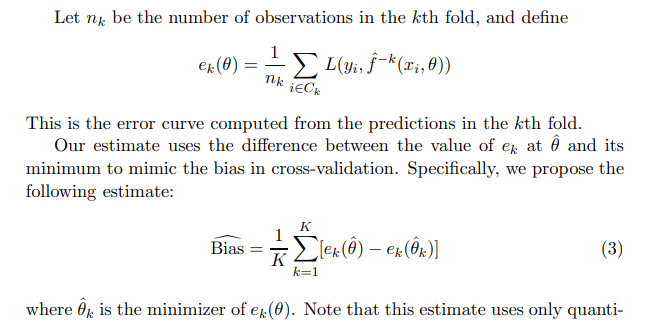
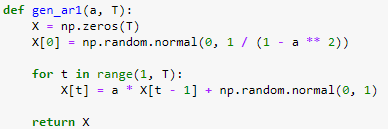
[A Bias Correction for the Minimum Error Rate in Cross-validation](https://www.stat.cmu.edu/~ryantibs/papers/cvbias.pdf) 

# Either one of the two Gaussian Processes using Cross-Validation.

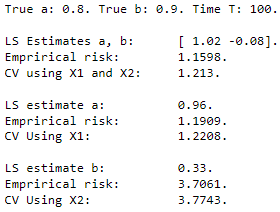
Generated to AR(1) models X1, X2 as follows:





Generated Y as X1 + noise.

Emprirical Risk is minimized using X1 and X2. What does cross validated risk do?

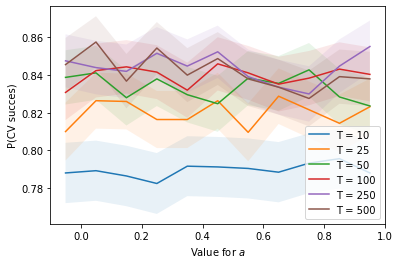
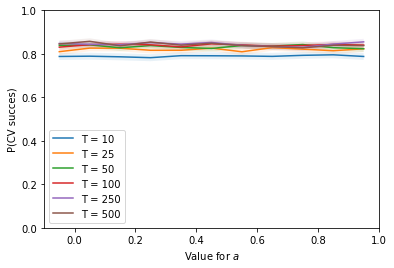


Indeed! In the example, the cross validated risk minimizer indeed picks the correct model. Now, how often is this the case? And do the values *a, b,* and *T* influence this?

## Success rate as a function of a, fixed T

To find the success rate, we sample X accordingly 1.000 times, for different as. The values are between 0.05 (almost noise), and 0.95 (strong dependence on previous lag). Intuitively, the higher the value of a, the more its proportional influence would be. So, intuitively, we expect for a large a that the success rate is higher.

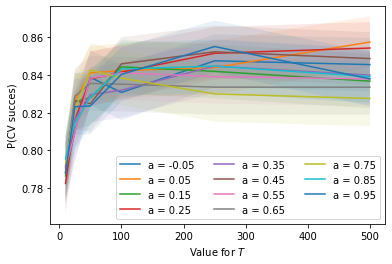
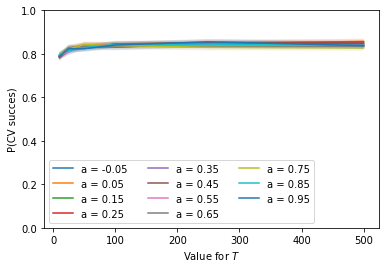
As we need to use a T, we will use T = 50 (so that we have 50 Y\_i, X\_i pairs). Other values are: 10, 25, 50, 100, 250, 500.

We clearly see that lower values of T achieve a lower probability of success, which is as expected. The difference between 50 and 500 is not very large, so the difference is really in the lower part.

## Success rate as a function of T, fixed a

We will also look at the influence of T, for a fixed a. We use T = 10, 25, 50, 100, 200, and a = 0.5. Again, the larger T would be, the less likely we are to be fooled by random correlation for AR(2). So, we would expect the success rate to be higher for larger T.

It seems that the probability of success is significantly lower for small values of T, around 10, 25. However, when T becomes 250, the difference becomes much smaller. It is hard to tell which *a* results in a lower or higher value. There does not seem to be a significant difference.

## Two-Dimensional Grid

The a have value in [-0.05 + 0.10 \* i], i = 0, …, 10. The T have value in [10, 25, 50, 100, 250, 500, 1.000]:

T\a | -0.05 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95.

10 |[[0.79 0.79 0.79 0.78 0.79 0.79 0.79 0.79 0.79 0.80 0.79]

25 | [0.81 0.83 0.83 0.82 0.82 0.83 0.81 0.83 0.82 0.81 0.82]

50 | [0.84 0.84 0.83 0.84 0.83 0.82 0.84 0.84 0.84 0.83 0.82]

100 | [0.83 0.84 0.84 0.84 0.83 0.85 0.84 0.84 0.84 0.84 0.84]

250 | [0.85 0.84 0.84 0.85 0.84 0.85 0.84 0.83 0.83 0.84 0.86]

1000| [0.85 0.86 0.84 0.85 0.84 0.85 0.84 0.83 0.83 0.84 0.84]]

## 95% Confidence Interval

Approximately 1.96 \* sqrt(p (1 – p) / n), so for p = 0.8, n = 2500 => +/- 0.028.

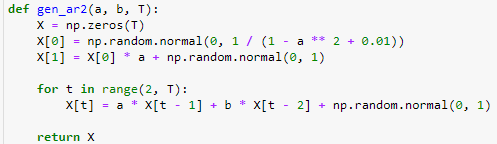
**Question:**

Why does the success rate of the LOOCV model selection barely depend on a?

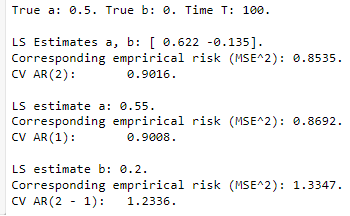
Why the small increase for larger value for *T*.

# AR(1), AR(2), or AR(2-1) using Cross-Validation.

Generated AR(1), AR(2), and AR(2 – 1) Models using Cross Validation:



In all scenarios, AR(2) will achieve minimal empirical risk. However, cross-validation should pick the correct one hopefully.

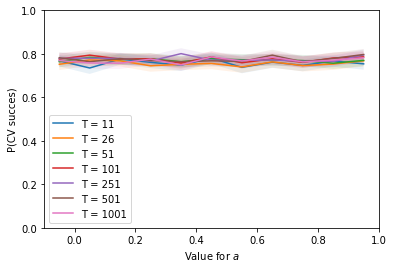
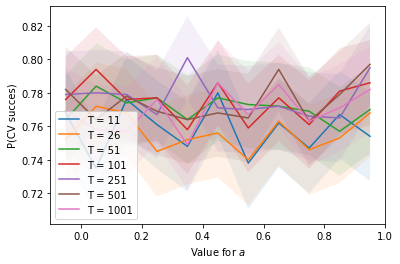


In this example, we see that indeed, the correct model is chosen! But, how often is this the case?

## Success rate as a function of a, fixed T

To find the success rate, we sample X accordingly 1.000 times, for different as. The values are between 0.05 (almost noise), and 0.95 (strong dependence on previous lag). Intuitively, the higher the value of a, the more its proportional influence would be. So, intuitively, we expect for a large a that the success rate is higher.

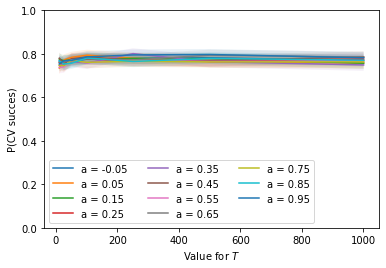
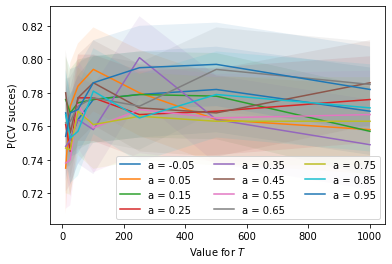
As we need to use a T, we will use T = 51 (so that we have 50 X\_i, X\_i+1 pairs). Other values are: 11, 26, 51, 101, 201.



Seems like T does not affect it at all, or very neglible. Doing more samples revealed a slight increase for larger *T*.

## Success rate as a function of T, fixed a

We will also look at the influence of T, for a fixed a. We use T = 11, 26, 51, 101, 201, and a = 0.5. Again, the larger T would be, the less likely we are to be fooled by random correlation for AR(2). So, we would expect the success rate to be higher for larger T.



We see that the value for *a* does not really affect anything, all are around the same line. It seems to drop from 10 to 25, but after that it shoots up a bit to 50. And remains quite constant for larger T.

## Two-Dimensional Grid

The a have value in [0.05 + 0.10 \* i], i = 0, …, 9. The T have value in [11, 26, 51, 101, 251, 501, 1.001]:

T\a|-0.05 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85 0.95

11 |[[0.768 0.735 0.776 0.761 0.748 0.78 0.738 0.762 0.747 0.767 0.754]

26 | [0.752 0.772 0.768 0.745 0.752 0.756 0.74 0.763 0.746 0.753 0.768]

51 | [0.764 0.784 0.774 0.777 0.764 0.777 0.773 0.772 0.769 0.757 0.77 ]

101 | [0.776 0.794 0.776 0.777 0.758 0.786 0.759 0.777 0.761 0.781 0.786]

251 | [0.779 0.78 0.779 0.767 0.801 0.771 0.77 0.772 0.766 0.765 0.795]

501 | [0.782 0.764 0.778 0.769 0.764 0.768 0.765 0.794 0.763 0.779 0.797]

1001| [0.769 0.758 0.757 0.776 0.749 0.786 0.767 0.785 0.763 0.771 0.782]]

## 95% Confidence Interval

Approximately 1.96 \* sqrt(p (1 – p) / n), so for p = 0.8, n = 1000 => +/- 0.025.

**Question:**

Why does the success rate of the LOOCV model selection barely depend on T or a?

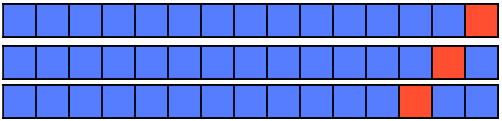
Why the constant value of 0.8?

# Cross Validation on AR(1)

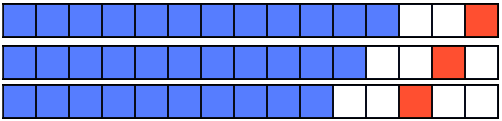
Different Cross-Validation Methods.

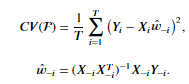
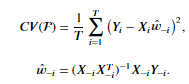
1. **LOOCV, NonDep(*h*).** Leave One Out Cross Validation. *T-1* sets, train on *T – 2* (X\_i, X\_i+1) pairs, test on one (X\_j, X\_j+1) pair. Alternatively, to remove dependency between train en test set, we can remove the *h* pairs before and after the test set from the training set.

LOOCV with h = 0.



LOOCV with h = 2.



Model definition, derivation. For LOOCV with h = 0. Consider an AR(1) model *X\_t = a X\_t-1 + e\_t*. Then, the cross validation error is , where  is the ordinary least squares solution.

Let us consider one Y\_i – X\_i w\_-i. It seems that they have low bias, but high variance. Every time, it seems to be about equal to *s* in mean, similar to the empirical risk and the true risk, and has covariance *2s^2*. Think this is due to the sum of T samples, each having variance *2s^2*.

Generate AR(1) with a = 0.8 for T = 51. Iterate 1000 times.

Expected Risk: 1.025 +/- 0.003.

True Risk: 1.007 +/- 0.042.

LOOCV Risk Mean: 1.026 +/- 0.045.

LOOCV Risk Variance: 2.134 +/- 1.371.

Empirical Risk: 0.986 +/- 0.042.

Larger *T* means that the values will be closer to 1.000.

Generate AR(1) with a = 0.8 for T = 251. Iterate 5000 times.

Expected Risk: 1.004 +/- 0.0.

LOOCV Risk: 1.006 +/- 0.008.

LOOCV Risk Variance: 2.025 +/- 0.23.

True Risk: 1.002 +/- 0.008.

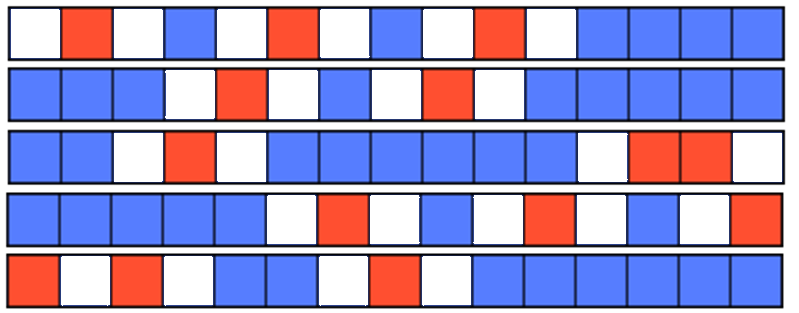
Empirical Risk: 0.998 +/- 0.008.

It seems as if the LOOCV risk is slightly higher, meaning that it overestimates the expected risk.

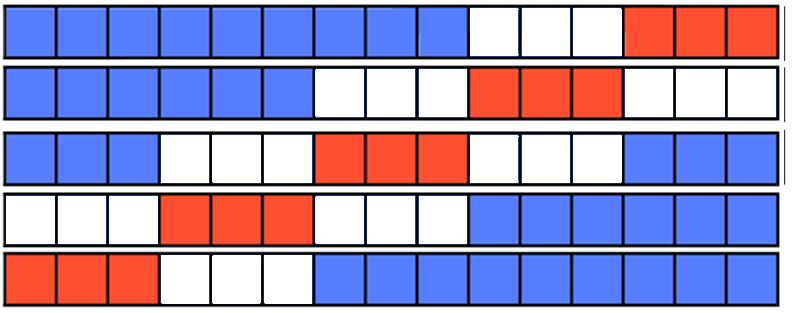
*Influence of h*. For larger *h*, the training set will be smaller resulting in less accurate estimates. Therefore, the value for *h* should be small for autoregressive models of order 1.

1. **K-Fold CV, +/- Random Shuffle, +/- NonDep(h).**

Five folds, random shuffle, h = 1.



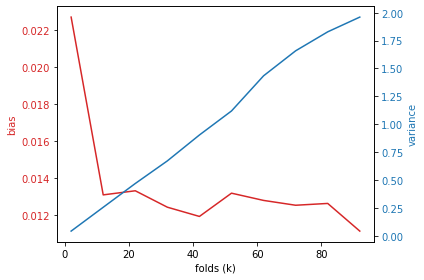
Five folds, no random shuffle, h = 3.



Larger folds decrease the variance between the fold estimates, as the test size is larger. Now, there are fewer “unlucky” small test sizes. However, as the train set is smaller, the bias will be larger.

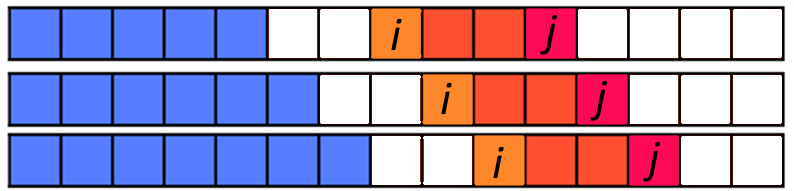
*Bias and variance of K-Fold*.

We see the variance increases as the folds increase, and the bias decreases as the folds increase. This is in line with what we expect.



1. **LFOCV, Test What (i, j), NonDep(h).**

We only train on the past and predict the future, commonly used in time series, as using the future to predict the past is “too good”, and hence, we often underestimate the true risk, as we think the problem is easier than it is supposed to be.



Furthermore, what do we test? Do we only test the next sample (h = 0, i = j)? Or de we test on more, say the next three samples (j = i + 3)? Or do we use the whole remaining samples (j = T)? Many questions.

A big issue is that the relative sizes between the train and test split change. In the beginning train / test is quite small, as train is e.g. T / 2. However, as we roll further into the future, the train size will be larger, and the test set not larger. This may be problematic.

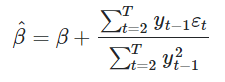
1. **OOS, split size, NonDep(h).**

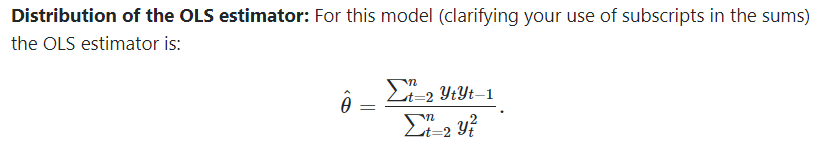
We simply split the time series in two parts, one is the train set, and the other is the test set, and there is no cross-validation, the test set is really mimicked to be *out of sample*. Now, to make the test set independent of the training set, some elements might need to be deleted to remove dependencies.

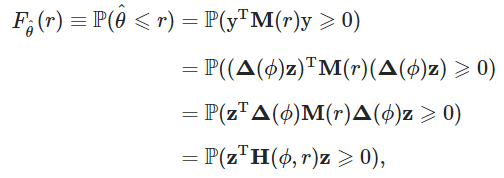


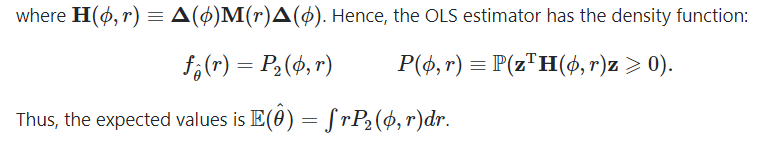
What is the split size, and how many elements need to be deleted to remove dependencies?

# Derivation of OLS Solution

Can be decomposed as a function of beta and the data: . What is its distribution?



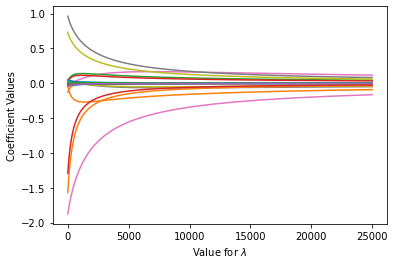




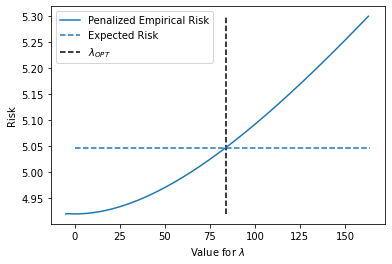
Very difficult to compute! Simulations indeed agree, but not something very nice.

# Ridge Regression

Uses the two-norm on the variables, is much easier to compute, even closed form solution. Can draw the solution path as well. Now, we see curves that do not go to zero, so not very useful for sparsity.



Now, what is a good value for lambda such that the penalized risk is equal to the true risk?



Value differs a lot, depending on T and a. Also, sometimes there is no such positive lambda, as the OLS estimate already over estimates the true risk sometimes.

# Bootstrapping

Some resources:

[Properties of the nonparametric autoregressive bootstrap](https://www.econstor.eu/bitstream/10419/61278/1/721981283.pdf)

[The bootstrap approach to autoregressive time series analysis](https://repository.up.ac.za/bitstream/handle/2263/29608/dissertation.pdf;sequence=1)

[The Bootstrap for the Functional Autoregressive Model FAR(1)](https://kluedo.ub.uni-kl.de/frontdoor/deliver/index/docId/4410/file/Thesis.pdf)

[New Bootstrap Method for Autoregressive Models](https://www.researchgate.net/publication/263648838_New_Bootstrap_Method_for_Autoregressive_Models?enrichId=rgreq-f807f223ee86e0e879574e78ff8379a4-XXX&enrichSource=Y292ZXJQYWdlOzI2MzY0ODgzODtBUzoxNDU0NzQ2NDQzNTMwMjVAMTQxMTY5NTI2NjkzMA%3D%3D&el=1_x_2&_esc=publicationCoverPdf)

[Discussion: Bootstrap methods for dependent data: A review](http://mapageweb.umontreal.ca/goncals/documents/Goncalves-Politis-2011.pdf)

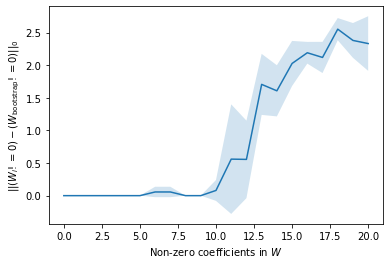
[3.3 Boostrap methods for time series](https://lbelzile.github.io/timeseRies/boostrap-methods-for-time-series.html)

Procedure:

1. Do OMP, and get a sequence of Ws = [W\_1, …, W\_K], where W\_i has i non-zero coefs.
2. For each W\_i:
   1. Compute residuals from X, centralize them.
   2. For the desired amount of iterations:
      1. Use these with W\_i to generate ~X.
      2. Do OMP on ~X, get sequence ~Ws.
      3. 1: Compare ~Ws\_i to W\_i. See if ~Ws\_i recoverts W\_i
      4. 2: Compare W\_i to ~W\_i. See where MSE diff stabilizes
   3. Save means / covariances of 1 and 2.
3. Plot 1. And 2. With their covariances.

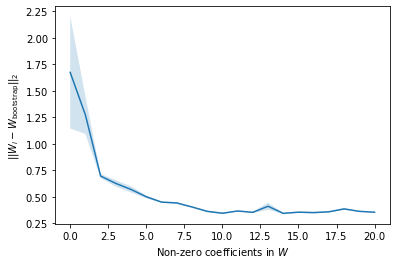
Bootstrap: We estimate a time series X using W. We calculate the residuals, and we simulate with W using the old residuals. We simulate multiple boostrap sets, and calculate their W.

Originally, for model selection, we argued that if an entry w is important enough, then it will be recovered from the bootstrap sample. If it is not important enough, then it will not be recovered.

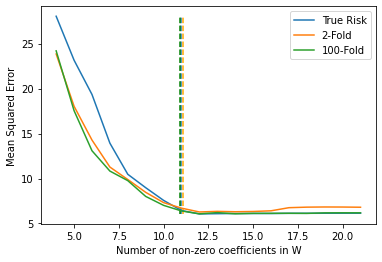


Additionally, we are now checking at the difference between W and its bootstrap in the norm. If we are close to the true datagenerating matrix, the residuals should be good, and also the estimation procedure should become **stable**.

We are therefore looking for a point where the difference between the estimated W and the bootstrapped W will become stable.



Comparing these to the cross-validation settings, we get similar results.



We see that #non-zero = 11 is really the first part where the line becomes flat. All parts after that achieve similar scores, but are not detrimental. This is similar to the bootstrap that recognizes incorrectly recovered. We see that from 10 onwards (which corresponds to 11 here), the line is going up, but the covariance indicates that there still quite often we recover all until 12 (which corresponds to 13 here). So, that bootstrap approach gives a nice way to look at the importance of edges.