Generative Adversarial Networks for Supervised Machine Learning with Limited Training Data

B. Bheeman, H. Byrne, J. Heidecke, S. Hoeksma Palazuelos, M. Prieto

January 23, 2017

Abstract

The growing research interest in Generative Adversarial Networks (GAN) has mostly focused on applications in computer vision so far. Our experiments show the potential of generated data as an approach to enhance training data for supervised machine learning. We train Deep Convolutional GANs (DCGAN) based on different amounts of original data from the MNIST data set and observe improved accuracy with a linear classifier, especially when adding generated data to previously small or medium-sized training sets. Even though the generated data could be distinguished from original data by humans in most cases, it often increases classification accuracy when using it as training data. This shows potential for other machine learning tasks beyond the scope of our experiments.

1 Introduction

Challenges finding the most appropriate data to build models for supervised machine learning can range from large data sets including a lot of redundancy to small data sets containing insufficient information to train a reliable model. In this paper we aim to show that by adding data generated with Deep Convolutional Generative Adversarial Networks (DCGANs) to insufficient-sized datasets, the accuracy of the trained models increases.

Deep learning and deep convolutional neural networks have so far mostly been used in a supervised way to classify data. DCGANs introduce a promising way of learning a generative model of some probability distribution in an unsupervised way. Most of GAN research has been focused on image generation, but there is a variety of other application domains. For example, DCGAN can be used to generate data to pre-train neural networks where labeled data is expensive to obtain. This approach is interesting since it is a recent development with ongoing research that might potentially have a substantial effect on a variety of machine learning practices. Therefore, evaluating the applicability of generated data to improve supervised classification tasks is the main focus of our report, to generate labeled data and evaluate its results and potential for further machine learning tasks.

In section 2 we review important publications and recent development in the field of DCGAN research. The setup of our experiment is detailed in section 3 and the results are presented in section 4. Finally, we conclude our observations and provide an outlook on future work in section 5.

2 Literature Review

Yann LeCun, Director of AI Research at Facebook and Professor at NYU, describes GANs as the "most interesting idea in the last 10 years in Machine Learning" [3]. Since the original work of Goodfellow et al. in 2014 [8], there has been a rapid increase in research in this area. This is witnessed when searching Google Scholar for papers including the phrase "generative adversarial nets"; we find 9 papers from 2014, 39 from 2015 and 260 from 2016. As this is a fairly new area of study, we expect to see a continuing trend of intensified research to better stabilise the models, as suggested by Radford et al. [10] and apply the structure to an extended domain (thus far the vast majority of research has been in the field of computer vision). Below, we start by discussing some of the basic theory and history of the models, briefly highlight some research successes so far and develop ideas for our own experimentation and analysis.

2.1 GANs

Generative Adversarial Networks (GANs) are comprised of two networks battling against each other, a generative and a discriminative model. The fundamental difference between them is the probability distribution that they model [5]. A discriminative model is used to predict some output values for given input values, more succinctly: given a set of data (x, y), a discriminative model learns the conditional probability distribution $P(y|x)$ for classification or regression. Contrastingly, a generative model learns the joint probability distribution $P(x, y)$, used to generate data that resembles some probability distribution of the data and furthermore, can be trained in unsupervised ways [13].

At the turn of the century it was general consensus that discriminative models were almost always preferred to generative models [9] and this was reflected in the exuberant level of research that followed. Discriminative models have achieved remarkable successes since, particularly in areas of natural language processing and image recognition [5]. Research into generative models developed less-enthusiastically but has increased in recent years and seen successful applications in image processing, speech recognition and information retrieval [13].

The merits of a hybrid generative-discriminative model were reported by Erhan et al. in 2010 [7] and then the formulation of GANs, by Goodfellow, was realised in 2014 [8]. In Generative Adversarial Networks the generative model is trained to capture the data distribution and the discriminative model is setup to decipher whether samples are real or generated and this feedback is backpropagated to the generative model. The discriminator is trained to maximise the probability that it correctly identifies whether samples are "real" or "fake", whilst the generator is trained to maximise probability of the discriminator making a mistake. It is adversarial because the two models are "fighting" against one another, and the framework is equivalent to a "minimax two-player game" with the value function:

$$V(G, D) : minmaxV(D, G) = Ex \sim pdata(x)[logD(x)] + Ez \sim pz(z)[log(1 - D(G(z)))]$$

The generated samples start as a uniform noise distribution Z (a random bit of code) taken as input, which the generative model wants to transform to appear like the training data. It uses the feedback from the discriminative model to adapt the many parameters used to create samples. In this way the generated samples start out extremely noisy and far in likeness to the real data, and are iteratively transformed to be indistinguishable from the real data's probability distribution. GANs offer lower computational cost compared to other frameworks, but have some shortcomings [8]. Notably the two models must be well-synchronized during training so that the respective models learn at the right pace. One model can be easily left behind if it is not learning at a comparatively fast enough pace, and will be stuck learning nothing if it is always losing its 'fight'.

2.2 DCGANs

The original framework for GANs has had many suggested revisions to try and improve the model stability, namely the Laplacian pyramid GAN [6], which uses multiple pairs of generative and discriminative models that learn to generate augmented versions of the previous image. Another such improvement is the adaption to Deep Convolutional GANs, detailed by Radford et al. [10] The result of their experimentation is to describe "architectural guidelines" for the stable training of Deep Convolutional GANs. They achieved far higher quality of image generation using their chosen architectures than previous research, which is why we will be experimenting with this framework for generative modelling.

Architectural changes to GANS [10]:

- Replace pooling layers with strided convolutions for discriminator and fractional strided convolutions for generator.
- Use batch normalisation in generator and discriminator.
- Remove fully connected hidden layers.
- Use ReLU activation in generator (all layers except output layer which uses Tanh).
- Use LeakyReLU activation in discriminator (all layers).

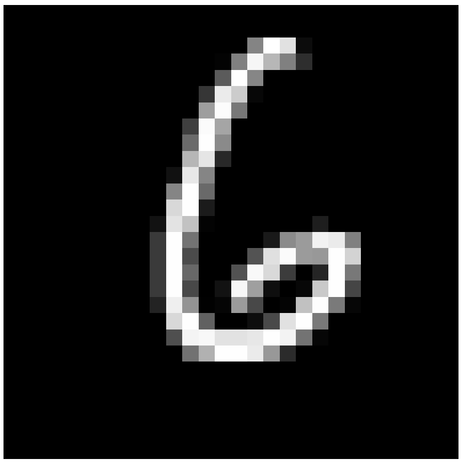
In our experiments we will be implementing code [1] that has been derived based on these architectures.

2.3 Data selection: MNIST

In machine learning research in general and more specifically on image generation, the MNIST dataset [2] has been well utilised as training data. Radford et al. [10], in fact, previously showed that MNIST samples generated by DCGAN framework can achieve a slightly lower test error than the MNIST training data. In our experiments we want to show whether adding generated data to a training set will provide additional value by creating a better model than with limited training data. As we are confident in the relevance and usability of this dataset and as the goals for our experiments differ to any that we have read about in our review of relevant literature, we have chosen to use the MNIST dataset. The MNIST dataset [2] is large and is assumed to include some redundancy (60,000 samples). We will be using a subset of

[1]https://github.com/carpedm20/DCGAN-tensorflow

5,000 samples and we assume the redundancy to be minimal. These 5,000 samples have 784 features (28x28 pixels), which are stored as unsigned chars (1 byte) and range from 0 to 255. These features allow for the classification of each sample into one of the ten digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). We will be solely experimenting with the 6 & 8 digits. These were chosen after brief, initial experimentation into which two digits were most difficult to classify in binary classification. Some example digits from the dataset are shown below.


(a) 6

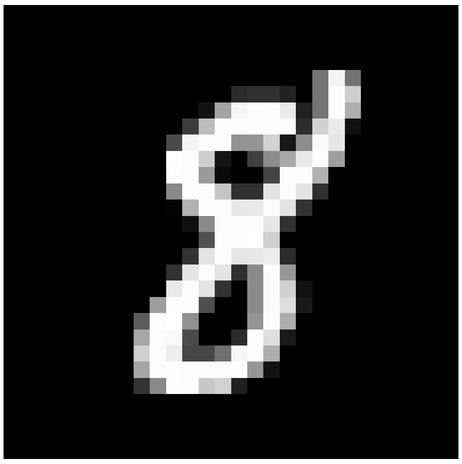

(b) 8

Figure 1: Example images of the MNIST data set

2.4 Applications

For simple image data sets such as MNIST, DCGANs are already able to generate samples of images which are almost indistinguishable from the real datasets. We speculate that it is only a matter of time before DCGANs and their adaptions are able to generate indistinguishable images for the most complex image data sets. OpenAI [?] predict applications such as "on-demand generated art, or Photoshop++ commands such as "make my smile wider"." The applications so far have been fairly limited to the domain of computer vision - namely generating images and more recently video [4], however it isn't inconceivable that in the future DCGANs could be used for generating much more in many different domains. We also see the potential benefit for using these frameworks to generate labelled data for supervised learning, where data is only available in small size and/or is expensive to obtain, which is why we have set out to research this in our own experimentation.

2.5 Labelled data

Whilst the amount of labelled training data available has increased exponentially over the past few decades, it is still a costly and laborious task to generate. There has been some efforts into generating more data by augmenting existing data, e.g. The infiMNIST code produces digit images derived from the MNIST dataset "using pseudo-random deformations and translations" [1].

The concept of using generated data as training data when the size of real data is small is certainly not a new idea. There has been fairly developed research into generating labelled training data from a large unlabelled dataset and a 3D model [12, 11]. The work of Sixt et al. [11] specifically uses DCGANs and concludes that a DCNN performs better trained on the generated data rather than the real data. It seems that the next obvious step is to be able to generate labelled training data for any domain of computer vision images.

We set out to explore how successfully we can generate samples to be used as training data for a supervised learning task, using different amounts and proportions of real training data vs generated training data.

3 Experiment

The main goal of our experiments is to evaluate if we can improve the performance of a classifier by adding more data to its training phase with generated data. This would be highly valuable in a scenario where not enough original training data is available, since obtaining labeled data is often a costly process that needs to be executed by human experts.

3.1 Experimental Setup

In this experiment we compare the impact of using different ratios of generated and original data to train the classifier. We are using the MNIST data set digits 6 & 8, so we will be generating and classifying handwritten characters (digits). The experiment has two main parts; first the generation of the data and second the classification of the data to compare the results.

Generation of Data:

The first part is the generation of the data using the DCGAN. The DCGAN is trained using the different amounts of original images from the MNIST data set. The amount of original data examples used for the generation is noted with k, $k \in K$ and K:

$$K = \{10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120\}$$

For the two digits $d \in \{6, 8\}$ and all combinations (d, k), we train 10 DCGANs to make the results less dependent on the random initial weights. The DCGANs are trained for 100 epochs. The implementation is based on a *tensorflow* version[2] of the original DCGAN paper [10] implementation. After training the DCGANs, they produce 20,480 generated samples each. This amounts to more than four million generated digits (based on: 2 different digits, 10 different values of k, 10 iterations, 20,480 samples per trained network).

Classification:

Once we have the generated images we proceed to train our classifier to evaluate if it performs better or not using the generated data. We are using a simple linear classifier to classify if the image presented is a 6 or an 8. We prepare different training

[2]https://github.com/carpedm20/DCGAN-tensorflow

sets, each with different ratios of original vs. generated data. The k originals are combined with m generated samples to form the training set (k, m). These m generated samples are generated solely based on the corresponding samples in k for each training set. The amount of generated data m is selected from M:

$$M = \{0, 10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480\}$$

The following table gives an overview of the training sets, especially how much original data (k) and generated data (m) they contain and what the proportion of generated data is. If all the rows were filled out, the table would be $|K \times M| = |K| \cdot |M| = 130$ rows long - one row for each combination of original and generated data. Note that the actual training data given to the classifier will be twice as big since it contains $k + m$ examples both for the digit 6 and 8.

k	m	$k+m$	$\frac{m}{k+m}$
10	0	10	0%
10	10	20	50%
10	20	30	67%
...	...	...	...
10	20480	20490	99.95%
20	0	10	0%
...	...	...	...
20	20480	20500	99.9%%
...	...	...	...
...	...	...	...
5120	20480	25600	80%

Table 1: Overview of combinations of original data k and generated data m.

After the classifier is trained on a specific combination of original and generated data, it is asked to classify the 1948 examples for 6 and 8 from the official MNIST test set, 974 examples for each digit. The metric we are using to evaluate our results from the classifier is the error of the classifier, based on the confusion matrix of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN):

$$Error = \frac{FP + FN}{TP + FP + TN + FN}$$

Since we trained ten DCGANs for each combination of k and m, we obtain 10 error measures for all combinations. This results in 1300 error measures, which we present in the following section.

Classifier

In our experiments the model is trained using a linear classifier.

A linear classifier computes the score of a class as a weighted sum of all of its pixel values across all of its features. Depending on precisely what values we set for these weights, the function has the capacity to like or dislike (depending on the sign of each weight) certain features in the input (in our case, the input image). Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

In this project, we used the "LogisticRegression" module from the scikit learn package as our classifier. This choice was made to ensure that we have no randomness in the training stage. Our input 'dataset' is a real valued nx28x28 3D array, where each n is a 28x28 grayscale image. We have normalized the data to maintain a 0-mean and and standard deviation of 1.

3.2 Quality of Generated Samples

We have designed an additional experiment in order to get an understanding of the subjective quality measured in how "real" the generated examples look to humans, for samples generated from small and large number of real samples. To do this we are asking 20 human testers to look at 100 pairs of digits and choose which digit they believe to be human generated (from the MNIST database). Each pair of digits will include one real MNIST digit and one DCGAN generated digit. We are testing for digits 6 and 8, for k=10 and k=5120, a total of four, 100-pair tests for each human participant. We are measuring quality as the accuracy of a human when discriminating between originals and generated samples. A low accuracy corresponds to very good generated data. An accuracy of 50% means the generated data is not distinguishable from real data anymore.

4 Results

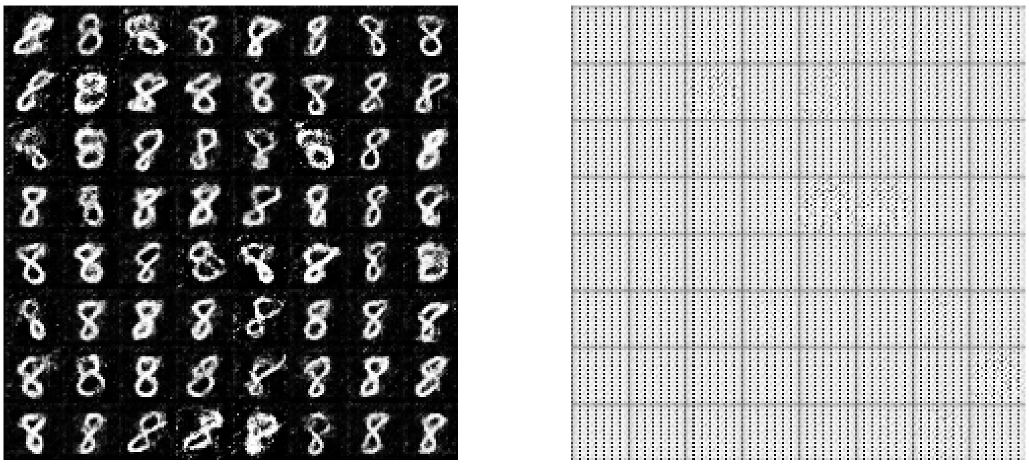
4.1 Quality of Generated Samples

In our experiment, for digit 6 we found that the mean accuracy for humans identifying the real data was 87.5% for $k = 10$ and 76.3% for $k = 5120$. We can see that when k is large i.e. $k = 5120$, the DCGAN is generating higher quality images that are more indistinguishable from the real MNIST data. This is intuitive as 10 training samples may not be sufficient to train well the DCGAN. Similarly for digit 8 we found that the mean accuracy for humans identifying the real data was 93.2% for $k = 10$ and 83.9% for $k = 5120$. Furthermore, we noticed that for each individual human "tester" their accuracy scores were lower for $k = 5120$, for 6 and 8, in every case. Additionally, these results clearly show that our generated data is not indistinguishable from the real data, as humans are able to tell the difference the majority of the time. These results also lead us to infer that the digit 8 was more difficult for the DCGAN to generate good quality, indistinguishable samples, as the accuracy for identifying 8 was higher in both values of k (than for 6). The mean accuracy of the human testers is summarized in the following table:

digit	$k = 10$	$k = 5120$
6:	87.5%	76.3%
8:	93.2%	83.9%%

Table 2: Mean accuracy of human testers distinguishing original and generated samples

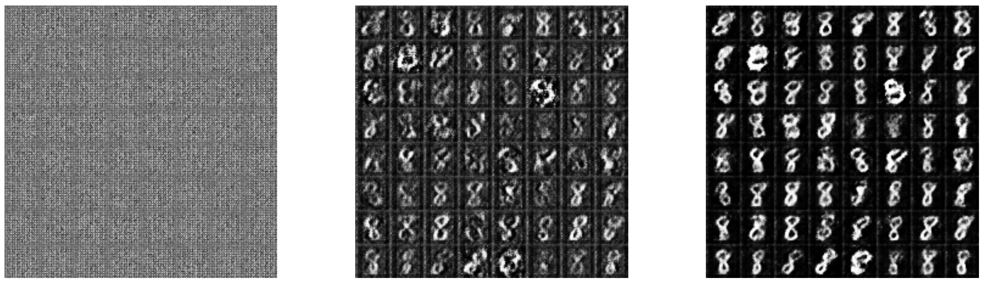
Furthermore, we discovered that for small k the network sometimes fails entirely to learn how to generate realistic data. This was seen, for example, in the 9th iteration of generating 8s based on 10 original samples. The original papers of GAN and DCGAN warned about this happening when the discriminator of the GAN improves too quickly in the beginning which leads to the generator not being able to find out how to improve its current weights. We can see examples generated based on 10 originals by a successful DCGAN in figure 2a and by a failed DCGAN in figure 2b.



(a) Results of successful DCGAN (b) Results of failed DCGAN

Figure 2: DCGAN results for $k = 10$

The failed DCGAN only produces random dots that are clearly not at all similar to the original data distribution of handwritten 8s. We can observe the progress of the training of successful networks by looking at the examples they generate after different epochs.



(a) 5th epoch (b) 25th epoch (c) 30th epoch

Figure 3: DCGAN training process for $k = 10$, successful (iteration 8)

Figure 3 shows images that were generated by an successful DCGAN after the 5th, 25th and 30th epoch. We can clearly see that when reaching a quarter of training time in epoch 25, the network already has a pretty good understanding of what the distribution of 8s looks like. The failed network, however, fails entirely to use the discriminators output to learn which patterns look more or less like the desired distribution. The training just produces noisy patterns, as can be seen in figure 4.

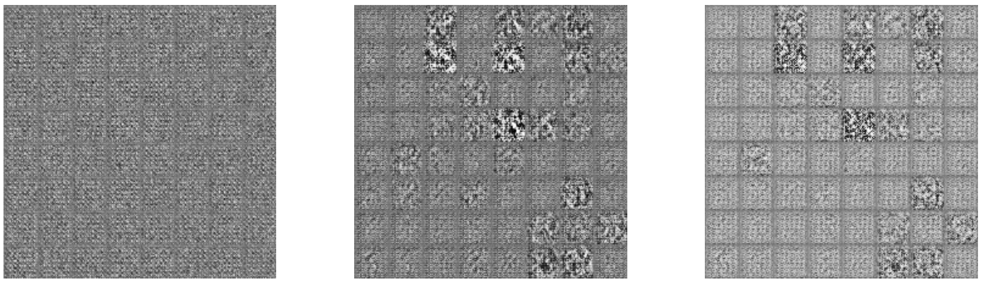

(a) 5th epoch (b) 25th epoch (c) 30th epoch

Figure 4: DCGAN training process for $k = 10$, failed (iteration 9)

4.2 Quality as Training Data

In this section we present our findings on how different proportions and amounts of both original and generated data in the training set influence the error of the classifier on the test set.

We can see clearly from our results that when the size of training data is small, e.g. $k = 10$, adding generated data to increase its size, has obvious benefits. In figure 5 we can see that the smaller the dataset is, the higher the reduction when adding even just a few generated samples to the training data. This reduction in error with added data shows a diminishing return as m increases (more generated data is added). This is intuitive because a sufficient training set size, for a classifier to learn from, needn't be exponentially large. When comparing $k = 10$ and $k = 5120$ on the same scale, we can see almost no change on the error for $k = 5120$, not even when adding four times the amount of the original data as generated data.

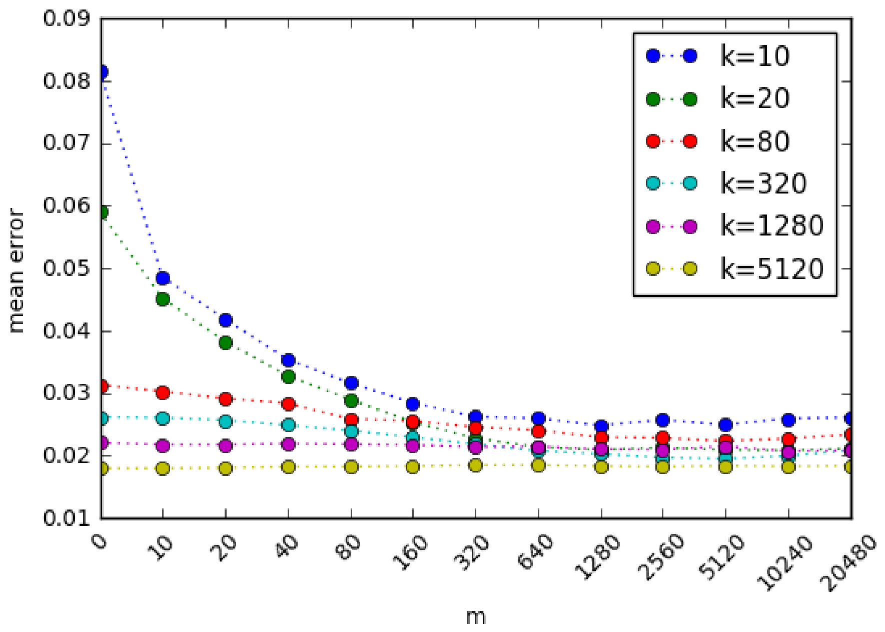


Figure 5: Effects of m on error for different k

In figure 6, when k=10, we see that for training sets with 20+ generated samples included, we obtain outlier error scores (crosses on the plot). These are likely to be examples when the DCGAN has failed to learn how to generate a digit. In this

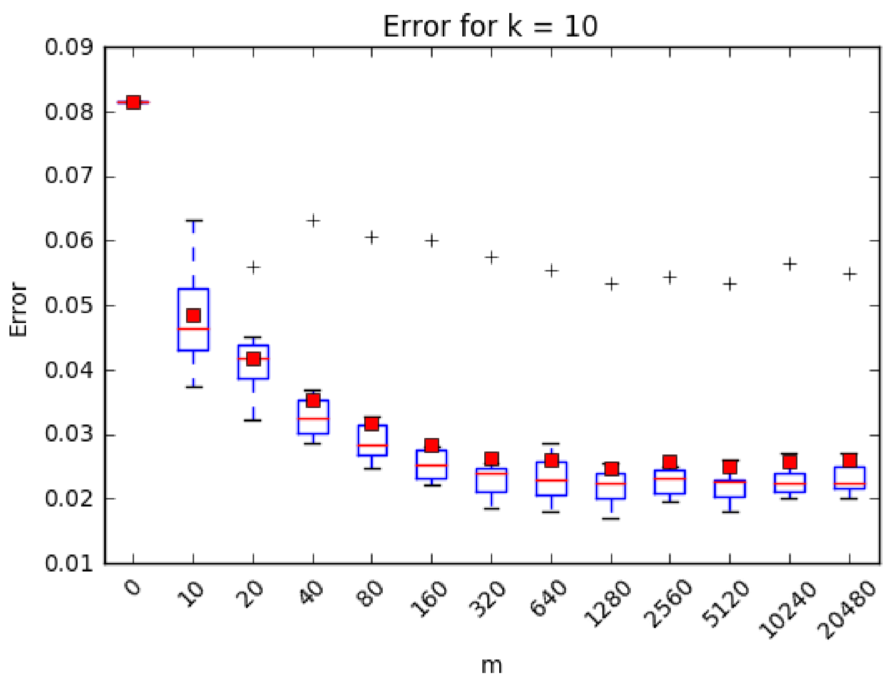
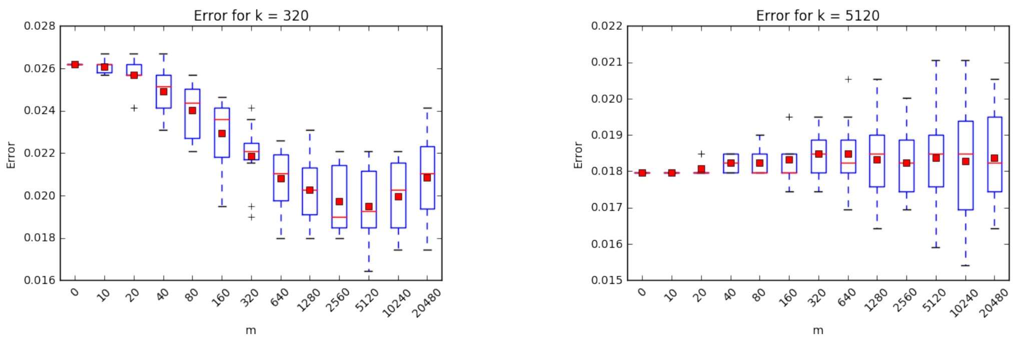


Figure 6: Effects of m on error for k = 10

case, we found the outlier to be based on the failed network for digit 8 in iteration 9, see previous section. These images, which look extremely unlike the MNIST digits have been incorporated in the training data and caused an outlier error rate. We see that for k=10 and m=320+ (training data size = 330+), there is no obvious improvement in error reduction for increasing size of generated data and therefore total size of training data. In fact, we notice that the rate of error begins to fluctuate very slightly.

We included the results of all measured k in the appendix in order to keep this result section overseeable. To contrast the results for a very small dataset of $k = 10$ in figure 6, we show the results for a very large dataset ($k = 5120$) in figure 7. We can see here that when the amount of original data is already sufficiently large, adding generated data does not really change the expected error - it merely adds variance to the results. For medium sized numbers of original data, such as $k = 320$ we observe slight positive effects of adding smaller amounts of generated data, but not comparable to the effect on small data sets. We see that as we increase k (larger original data size), the benefit of adding generated data also diminishes. In particular we notice that for big enough k, i.e. $k \geq 320$, by adding generated data we sometimes, in fact, increasing the error. Similarly, the standard deviation of error is much higher for large m.

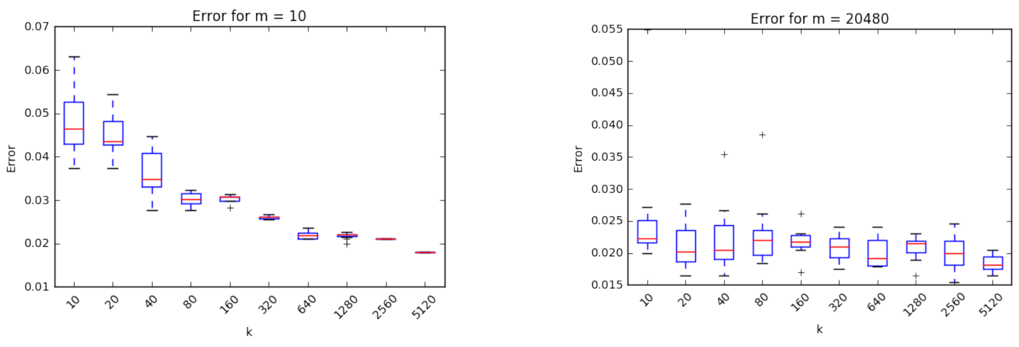
We can see from our results holding m constant and increasing the size of the training set by including original samples; that the larger the proportion of m of the set, the larger the variance in the error. For m=10 and $k \leq 80$ we see a wider, approximately reducing error. When m=10, and comprises 50% of the data, we see the largest variance. When m=20480, we notice that there is variance in the error results for all training set sizes. This is likely to be because the generated samples are comprising 80%+ of the training data. The generated samples are inferior in quality to the original data which leads to variance in the classification error. This



(a) (b)

Figure 7: Effects of m on error for k = 320, 5120

can be observed in figure 8



(a) (b)

Figure 8: Effects of k on error for m = 10, 20480

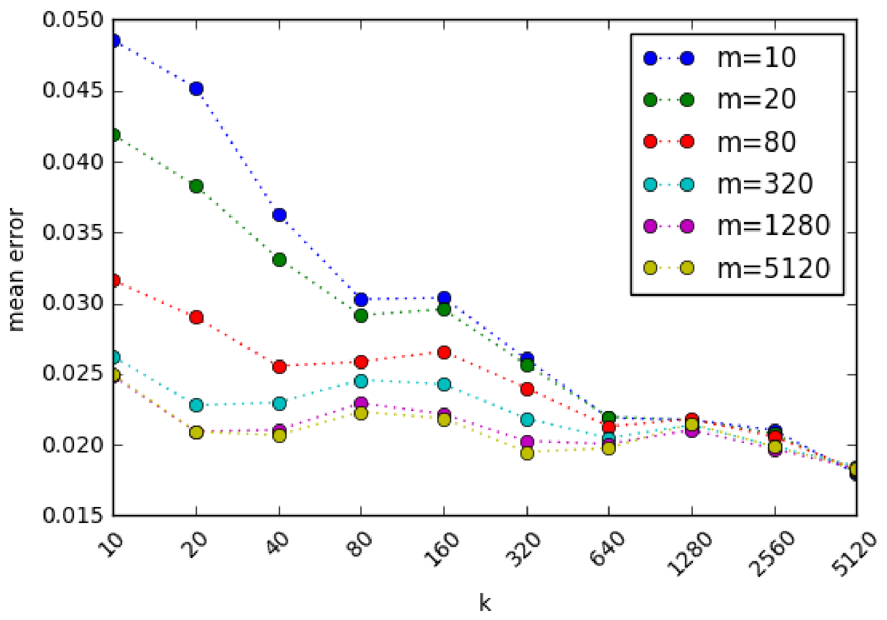


Figure 9: Effects of k on error for different m

In figure 9 we observe that the qualitative differences of adding different amounts of generated data are largest when the amount of original data is very small. For larger amounts of original data, it makes almost no difference how much generated data is added. This also shows, that the highest rewards can be earned for small data sets.

Finally, we observe the combined effect of k and m in figure 10. As we can see here, the error is obviously highest when there is both a small amount of original and generated data. However, both adding original data or generated data seems to have a strong effect on the classifier's error, especially when the total size of the training set is rather small. While overall it is mostly preferable to add original data, generated data often did almost as good and does not come with the costs of needing human experts to label the original samples for supervised learning tasks.

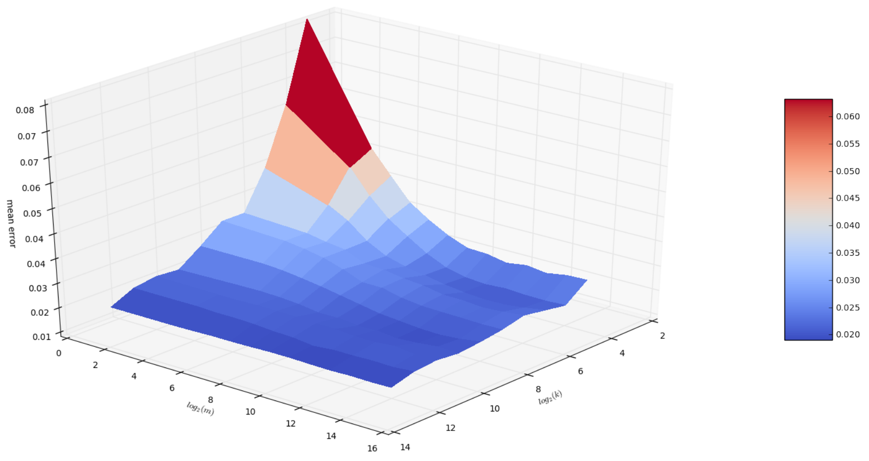


Figure 10: 3D plot of effect of both k and m

5 Conclusions

Our experiments have shown that DCGANs can, at least in some cases, be used to enhance the available base of training data for supervised machine learning tasks without costly manual labelling of new samples. While the scope of this report is limited to one dataset and one supervised classifier, the very clear results give rise to the expectation that other machine learning tasks could also benefit from enhancing their training sets with generated data.

As seen when comparing the effect of generated data with different amounts of original data, the biggest beneficial effect could be observed for rather small data sets. Bigger data sets can still benefit from generated data, but to a much lesser degree.

However, adding generated data did not always lead to a decrease of error. While mean and median error decreased for small and medium-sized data sets, there are few cases where negative effects could be observed. This is especially the case when the DCGAN fails entirely to converge to a good results, a scenario that was especially likely to observe with small datasets.

For this reason, we conclude that when using generated data, the effects should always be observed by means of using a validation set to give an estimate of the change of error. Since the DCGANs can converge to local minima of different quality, based on the randomized initial weights, it is recommended to train several DCGANs for the same problem and then pick the best generated data with a validation set.

While our experiment showed strong results for this particular combination of training data and supervised machine learning technique, the actual "search space" of machine learning problems is obviously much bigger. For this reason, further experiments are required to test the suitability of generated data for more types of data sets as well as for more different kinds of classifiers. These additional experiments will help to get an understanding of the scope of problems in which generating training data can be beneficial.

References

[1] The infinite mnist database. http://leon.bottou.org/projects/infimnist#credits. Accessed: 2017-01-23.

[2] The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed: 2017-01-23.

[3] Recent and potentially upcoming breakthroughs in deep learning. http://qr.ae/TPVp4H. Accessed: 2017-01-23.

[4] Carl Vondrick, H. P., and Torralba, A. Generating videos with scene dynamics. *CoRR*.

[5] Deng, L., and Jaitly, N. Deep discriminative and generative models for pattern recognition. *USENIX–Advanced Computing Systems Association* (2015).

[6] Denton, E. L., Chintala, S., Fergus, R., et al. Deep generative image models using a [U+FFFC] laplacian pyramid of adversarial networks. In *Advances in neural information processing systems* (2015), pp. 1486–1494.

[7] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research 11*, Feb (2010), 625–660.

[8] I. Goodfellow, J. Pouget-Abadie, M. M. B. X. D. W.-F. S. O. A. C., and Bengio, Y. Generative adversarial nets. *NIPS*, 2672–2680.

[9] Ng, A. Y., and Jordan, M. I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2002, pp. 841–848.

[10] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks.

[11] Sixt, L., Wild, B., and Landgraf, T. Rendergan: Generating realistic labeled data. *CoRR abs/1611.01331* (2016).

[12] Su, H., Qi, C. R., Li, Y., and Guibas, L. J. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 2686–2694.

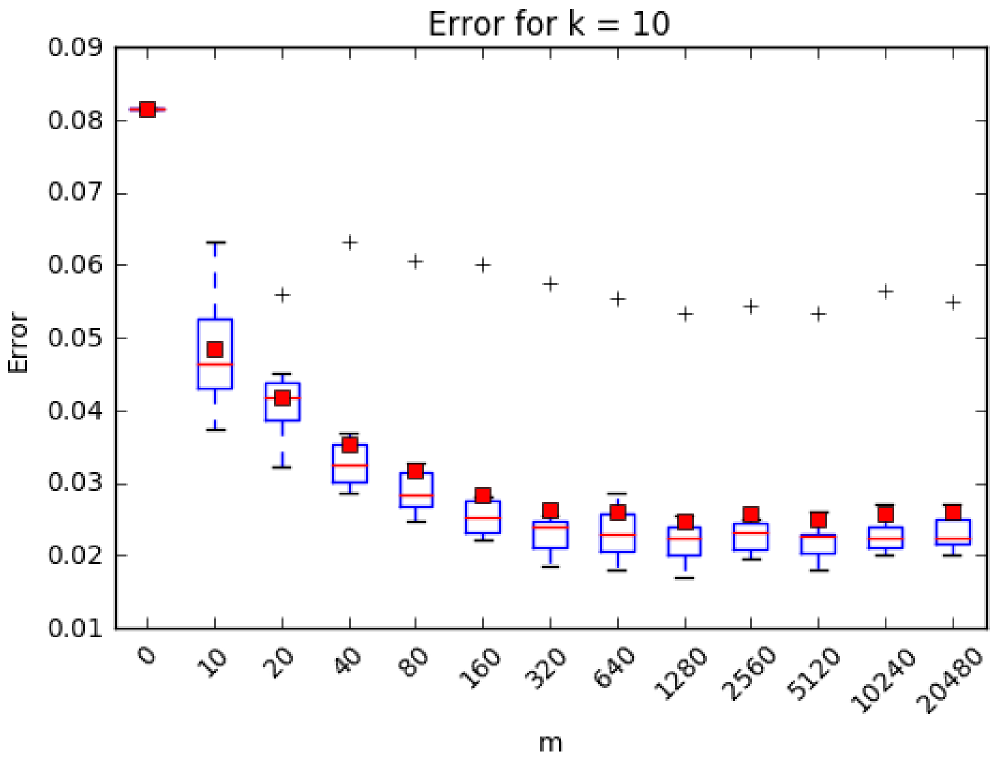
[13] Xu, J., Li, H., and Zhou, S. An overview of deep generative models. *IETE Technical Review 32*, 2 (2015), 131–139.

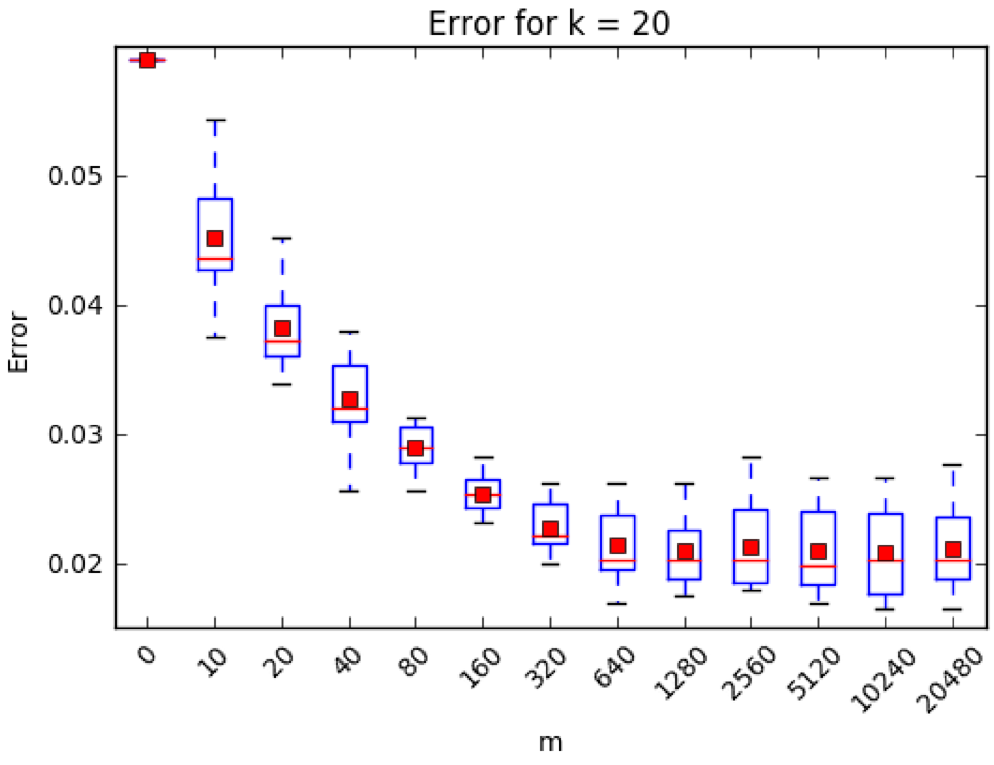
A Information about provided code

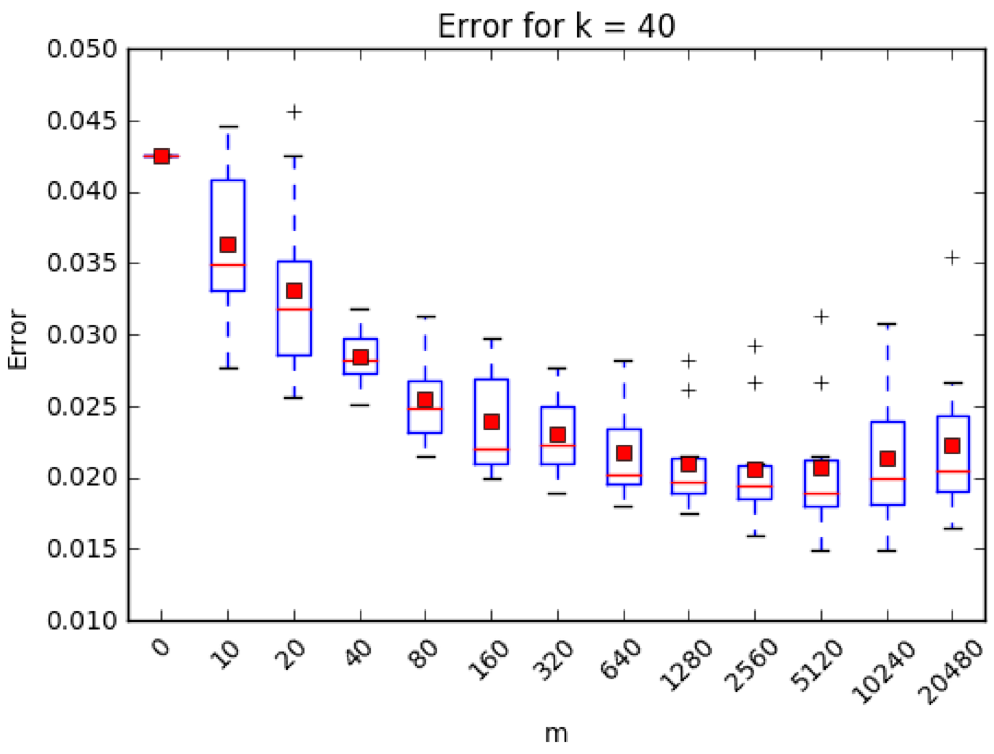
Attached to this report there is a folder titled `Code/` containing all code that was used to execute the experiment. We did not include the data that was generated during the experiment since it exceeds 16GB of storage space. In case the reader is interested in repeating the experiment, the following files and folders are of interest:

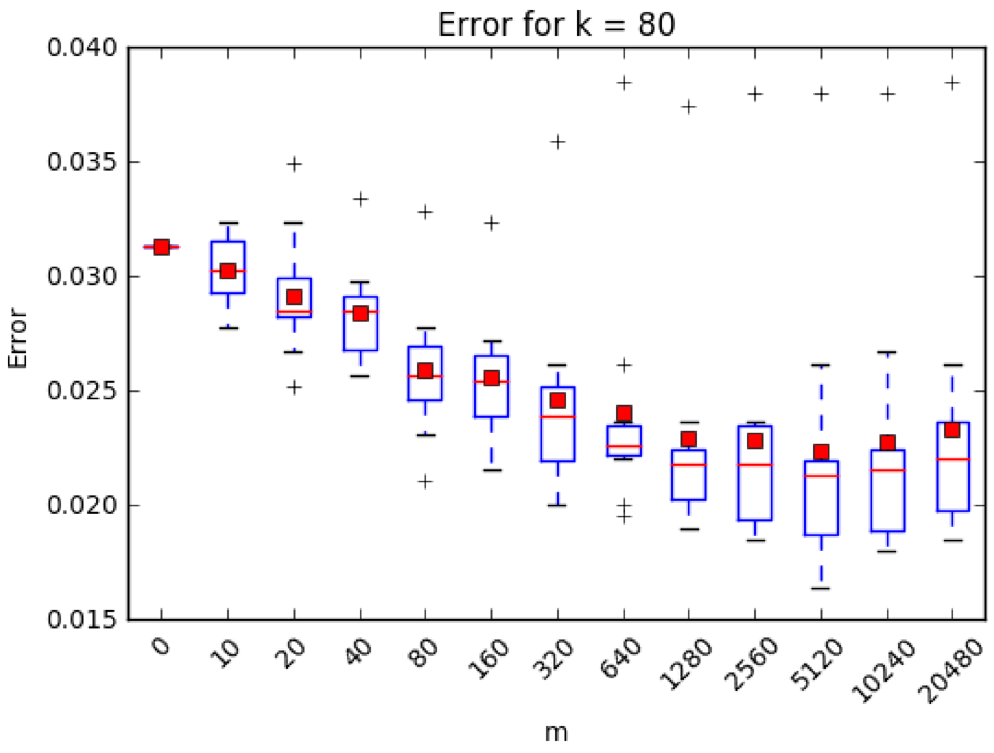
- Before executing any other code, one should download the MNIST dataset with the command `python download.py mnist`
- `generator.py`: This python script executes the training of DCGAN for all different combinations of k and d. The generated samples will be saved to the `/samples` subdirectory.
- `main.py` and `model.py` contain the implementation of DCGAN in the tensorflow framework. The implementation is taken from `https://github.com/carpedm20/DCGAN-tensorflow` and adapted to our needs.
- `linear_classifier.py` contains the code of the classifier used to obtain errors for the experiment.
- The script in `experiment.py` should be run after the generator process is done. It will iteratively provide the classifier with the right amounts of training data and then obtain a confusion matrix for the test data in the subdirectory `test/`.
- Results of the experiment are persisted in `experimentResults.pickle`. These results can be accessed and rendered with the jupyter notebook `Results.ipynb`.
- For any questions regarding the execution of the experiment, please do not hesitate to contact our group.

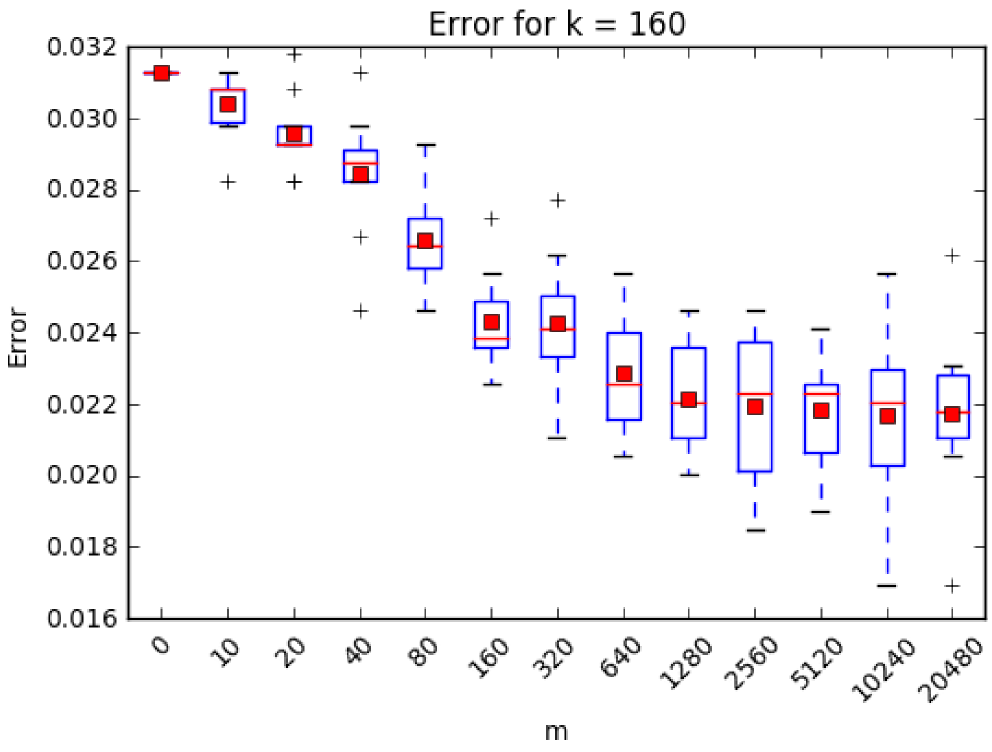
B Appendix: Results for different k

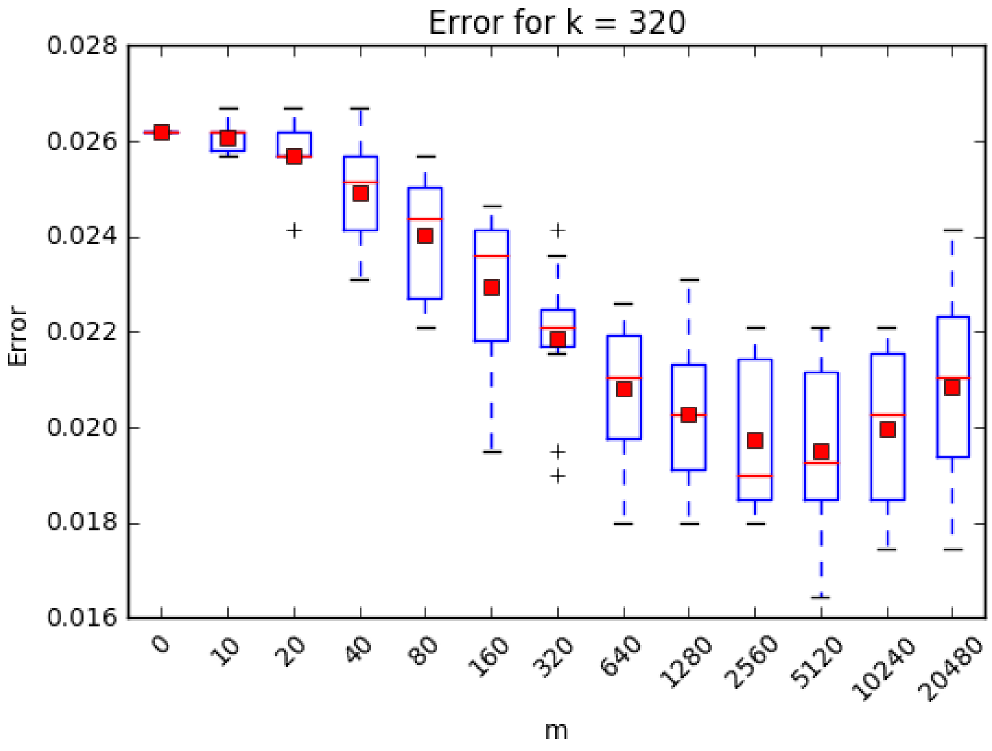


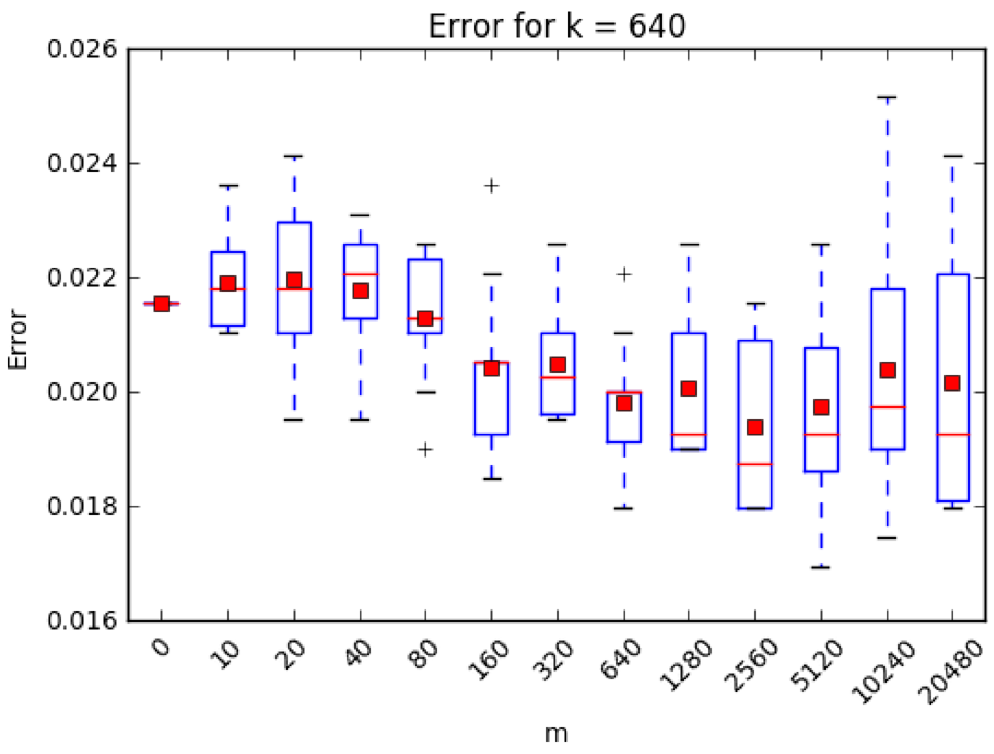



Error for k = 40
Error
m
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480

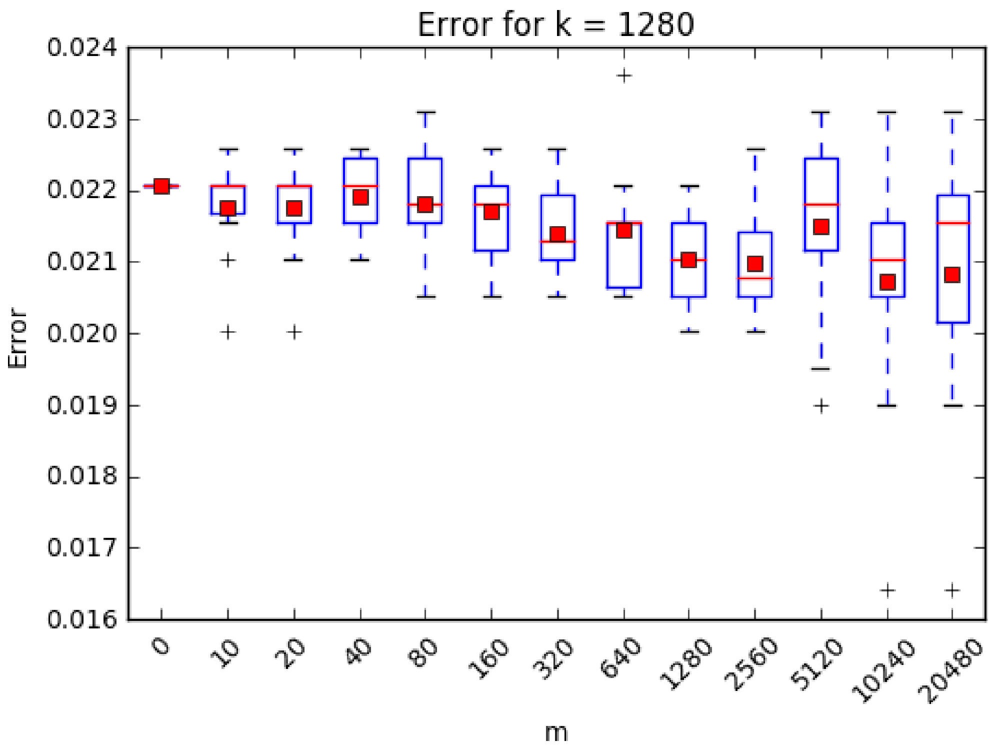

Error for k = 80
Error
m
0.040
0.035
0.030
0.025
0.020
0.015
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480


Error for k = 160
Error
m
0.032
0.030
0.028
0.026
0.024
0.022
0.020
0.018
0.016
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480

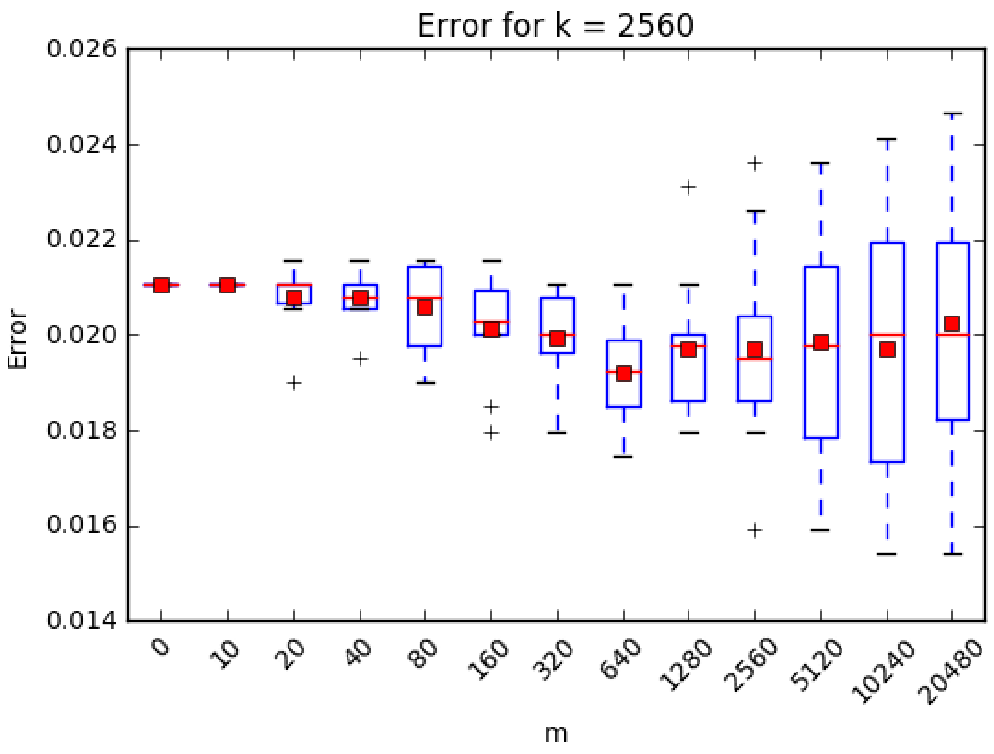

Error for k = 320
Error
m
0.028
0.026
0.024
0.022
0.020
0.018
0.016
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480

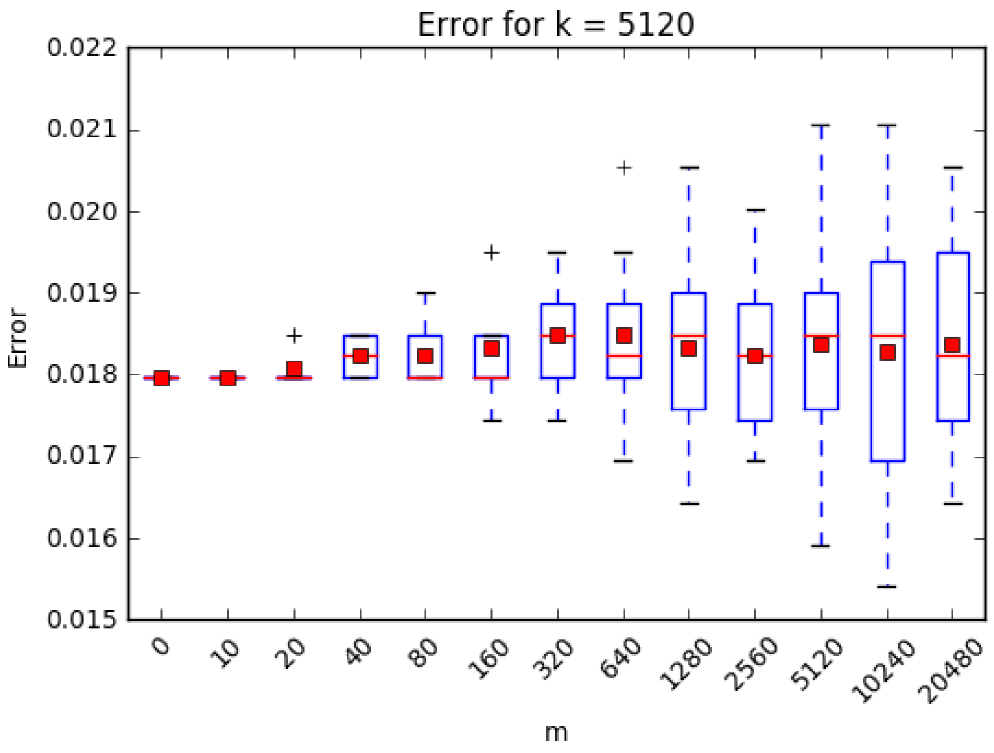


Error for k = 640
Error
m
0.016
0.018
0.020
0.022
0.024
0.026
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480

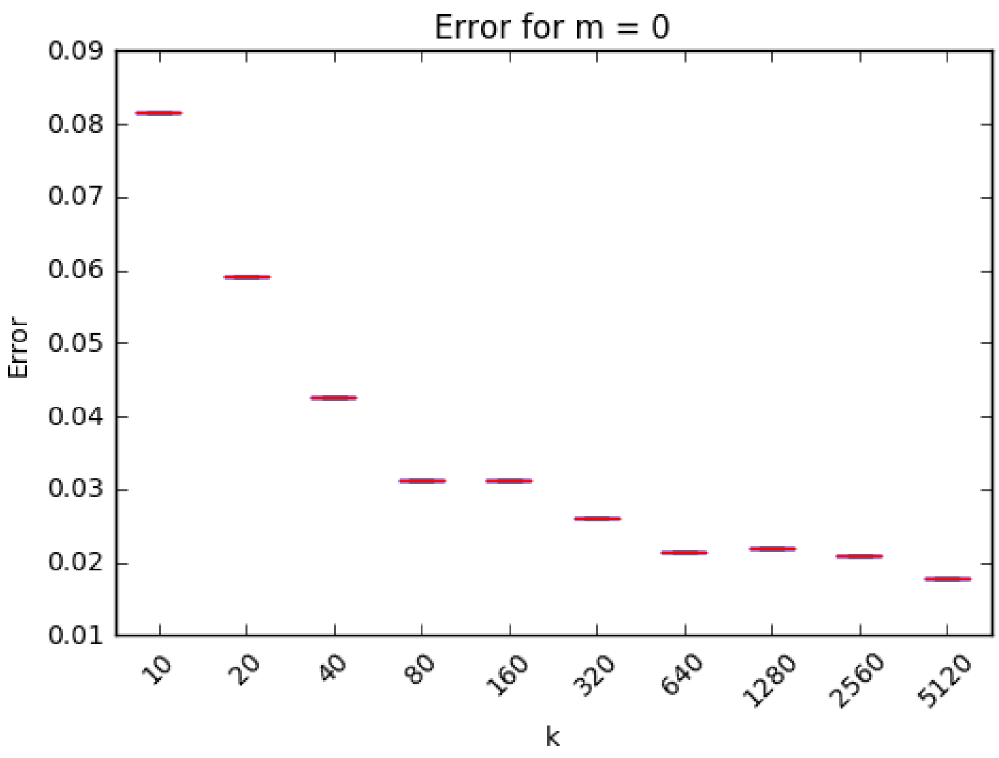


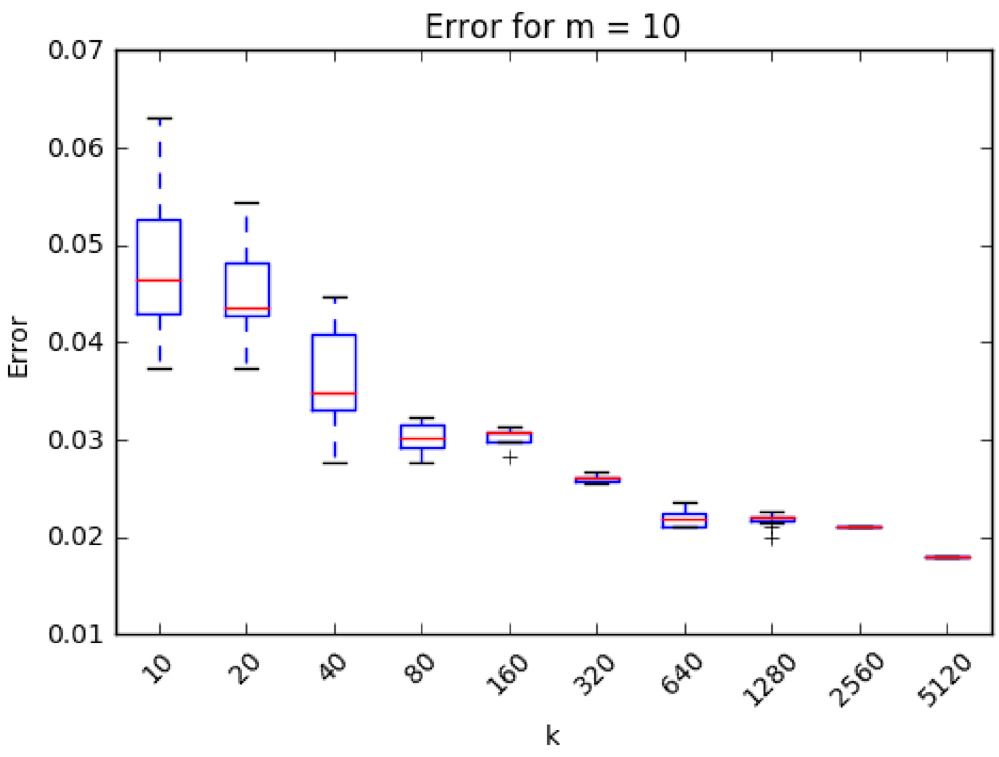
Error for k = 1280
Error
m
0.016
0.017
0.018
0.019
0.020
0.021
0.022
0.023
0.024
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480

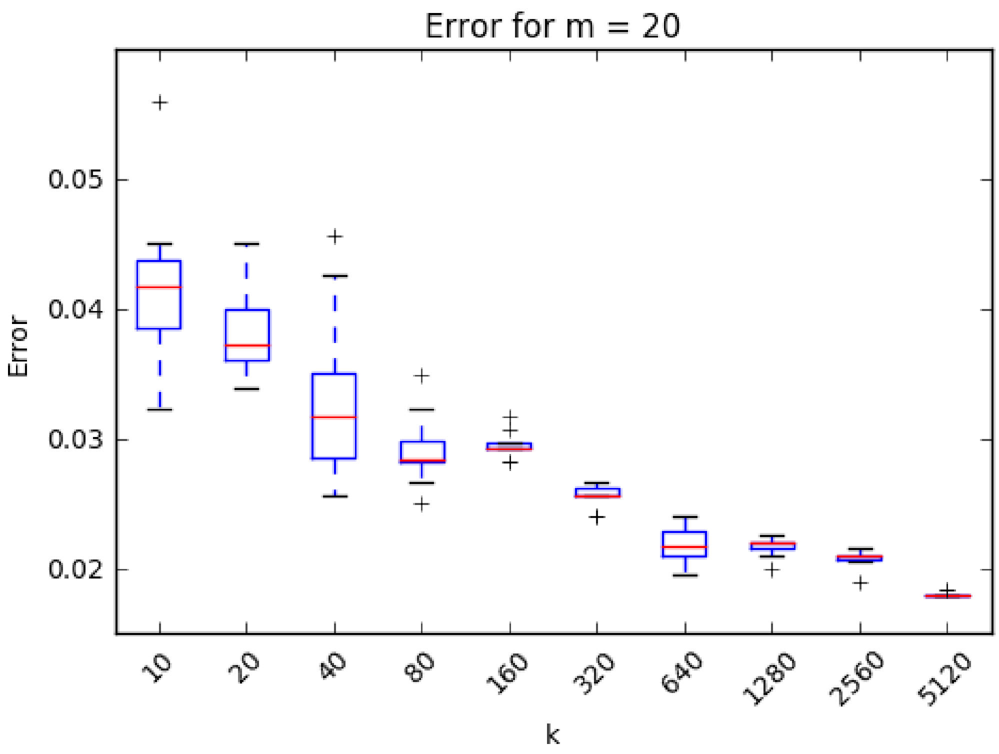

Error for k = 2560
0.026
0.024
0.022
0.020
0.018
0.016
0.014
Error
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480
m

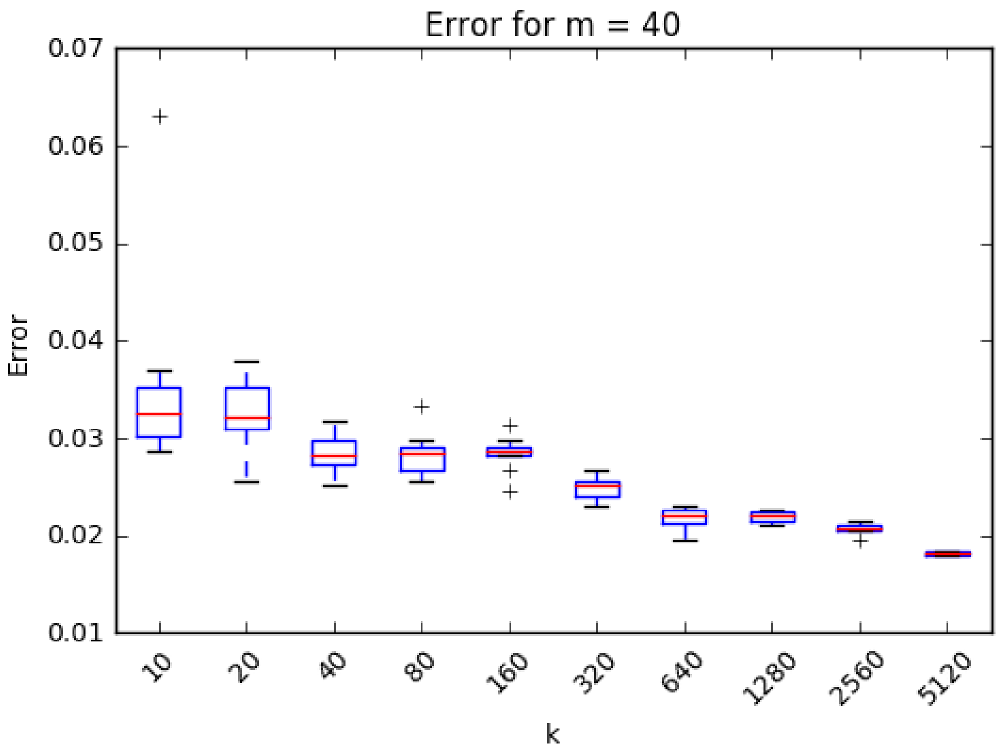

Error for k = 5120
0.022
0.021
0.020
0.019
0.018
0.017
0.016
0.015
Error
0
10
20
40
80
160
320
640
1280
2560
5120
10240
20480
m

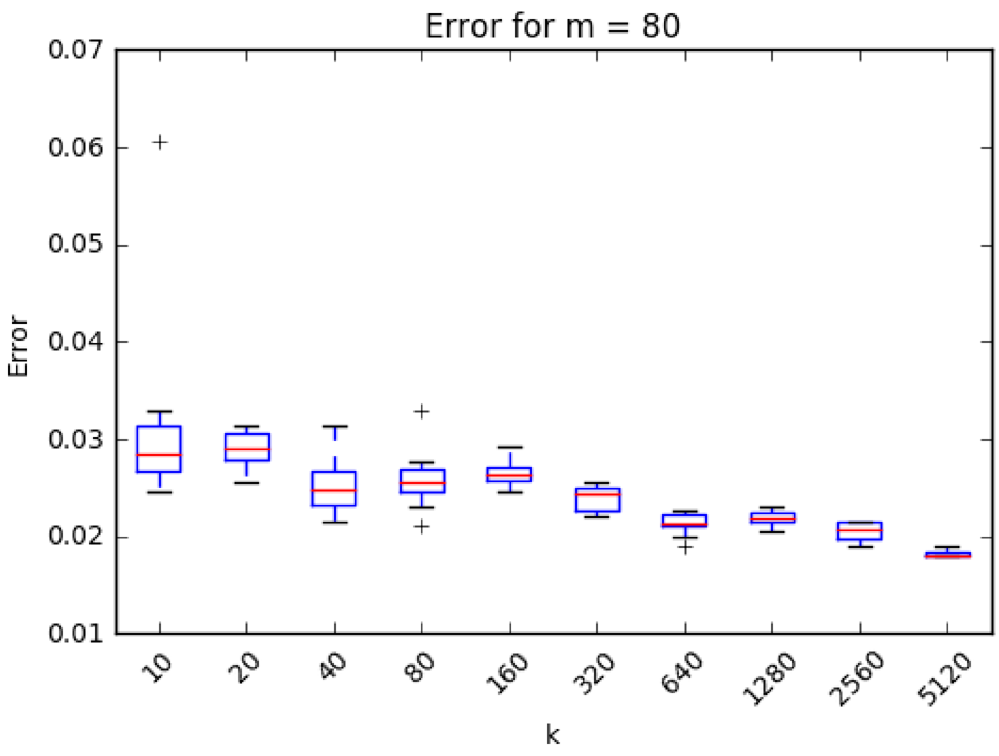
C Appendix: Results for different m

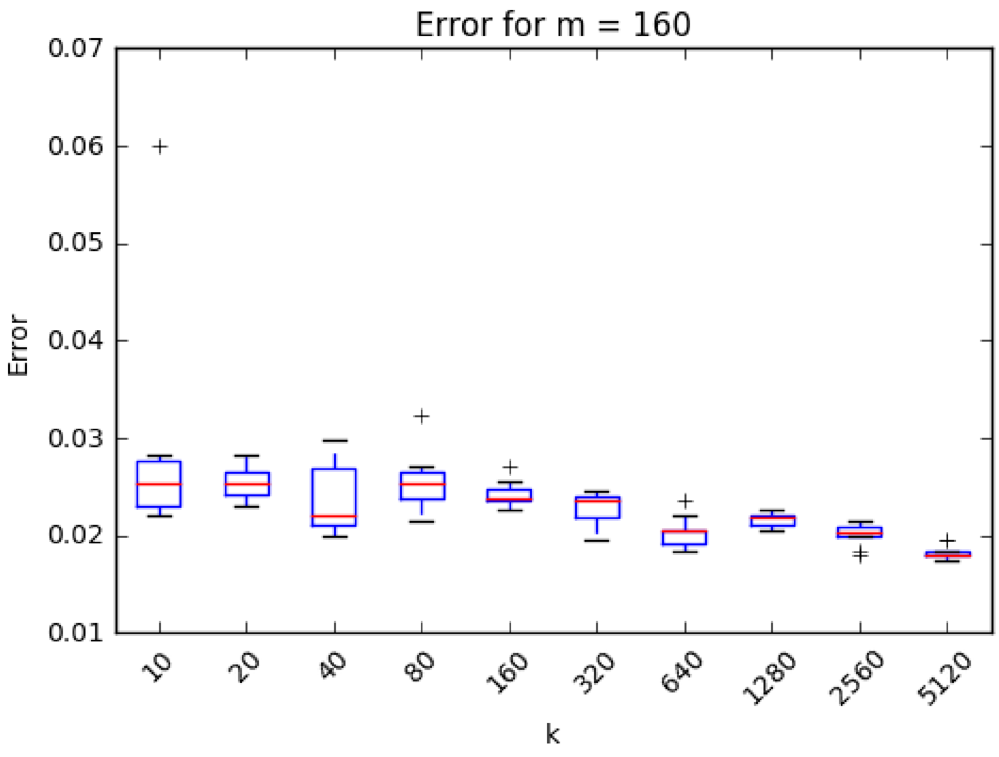


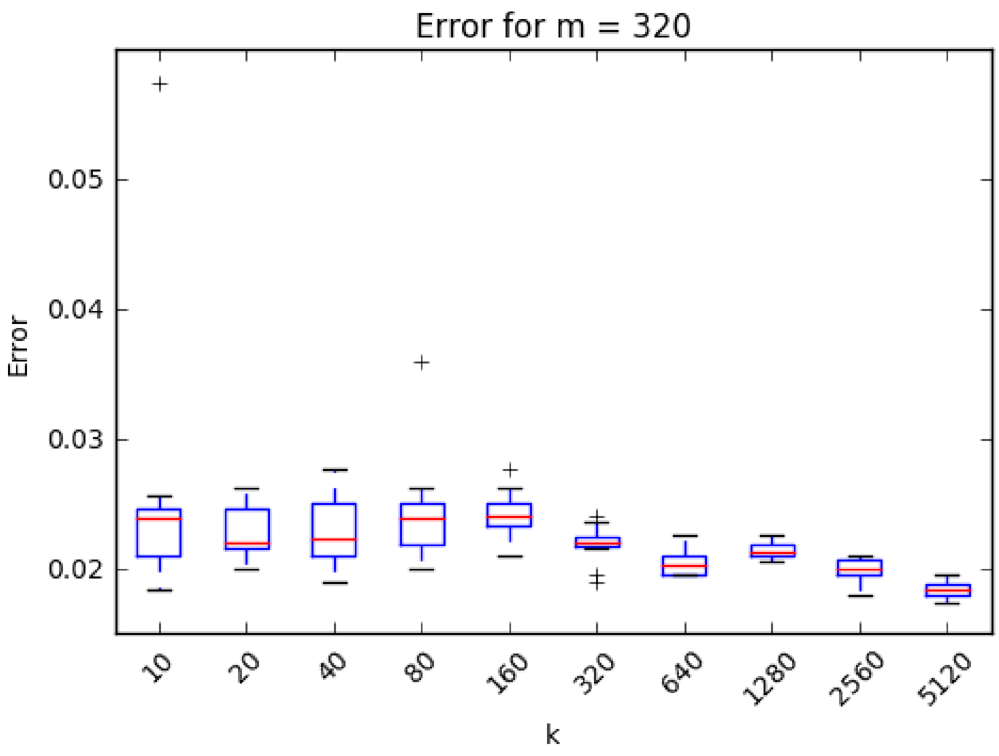



Error for m = 20
Error
0.05
0.04
0.03
0.02
10
20
40
80
160
320
640
1280
2560
5120
k

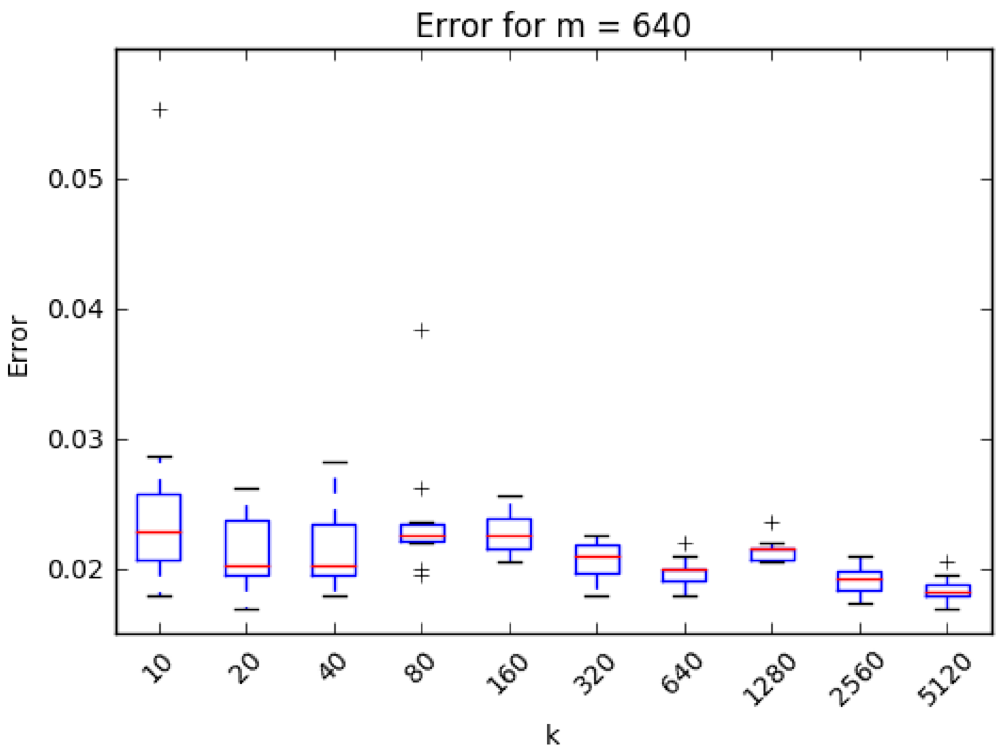

Error for m = 40
Error
0.07
0.06
0.05
0.04
0.03
0.02
0.01
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 80
Error
k
0.07
0.06
0.05
0.04
0.03
0.02
0.01
10
20
40
80
160
320
640
1280
2560
5120

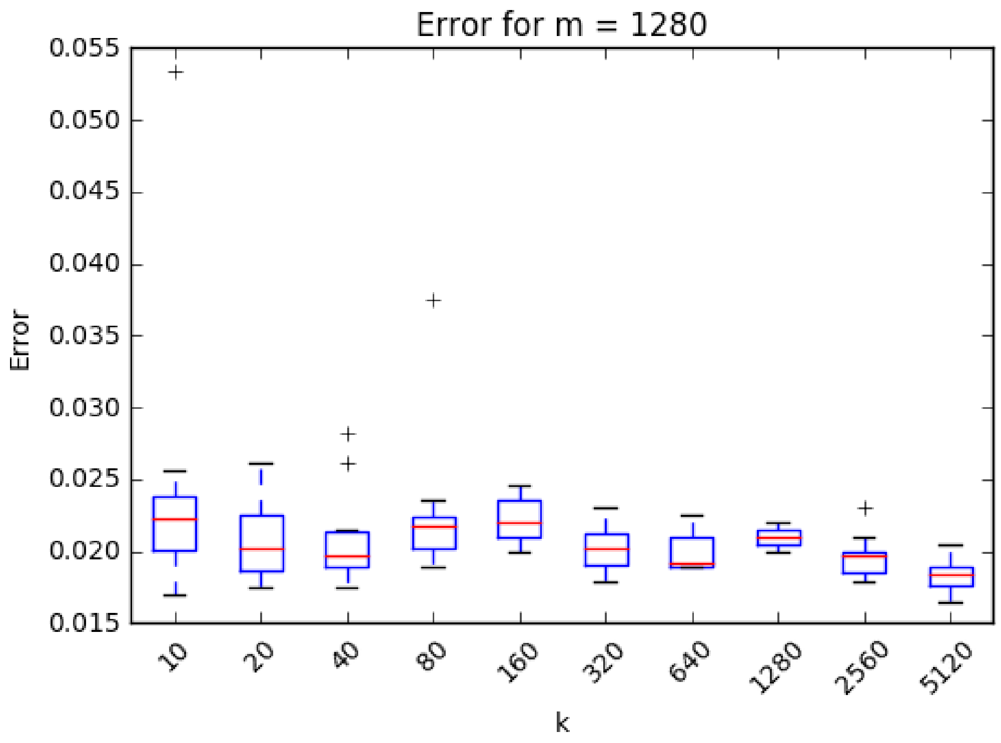

Error for m = 160
Error
k
0.07
0.06
0.05
0.04
0.03
0.02
0.01
10
20
40
80
160
320
640
1280
2560
5120

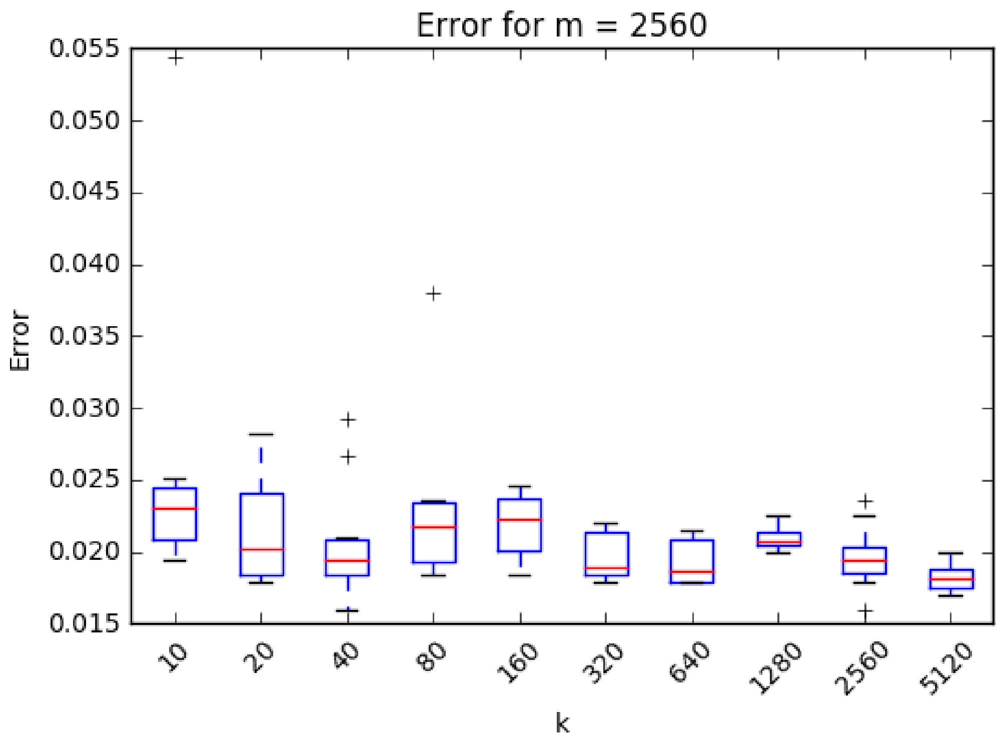


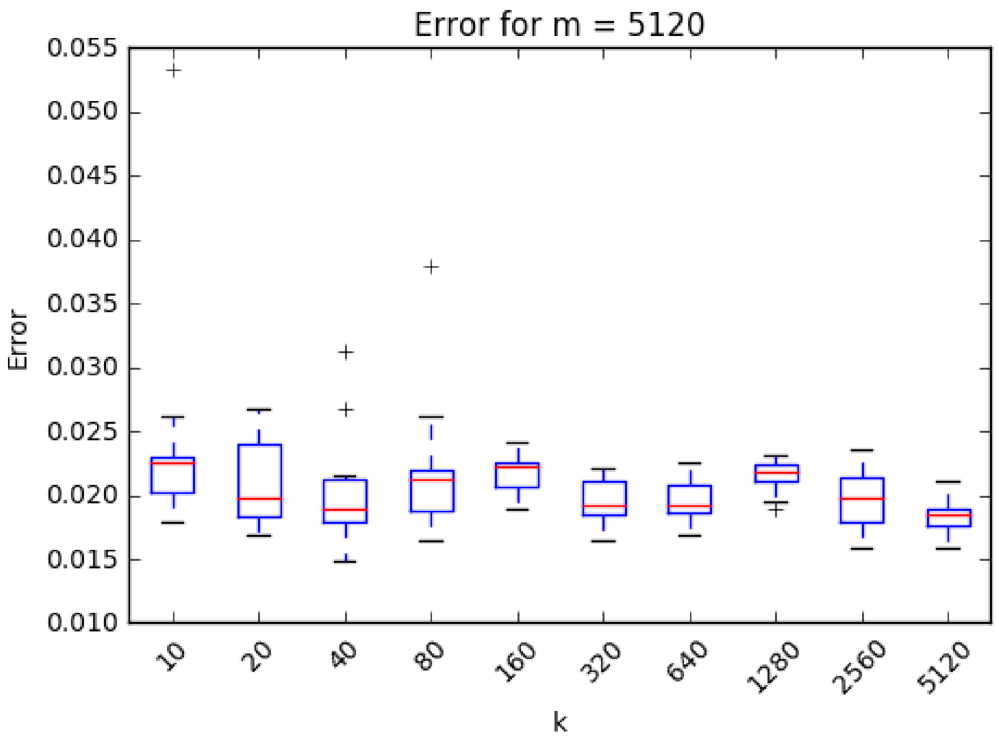
Error for m = 320
Error
0.05
0.04
0.03
0.02
10
20
40
80
160
320
640
1280
2560
5120
k

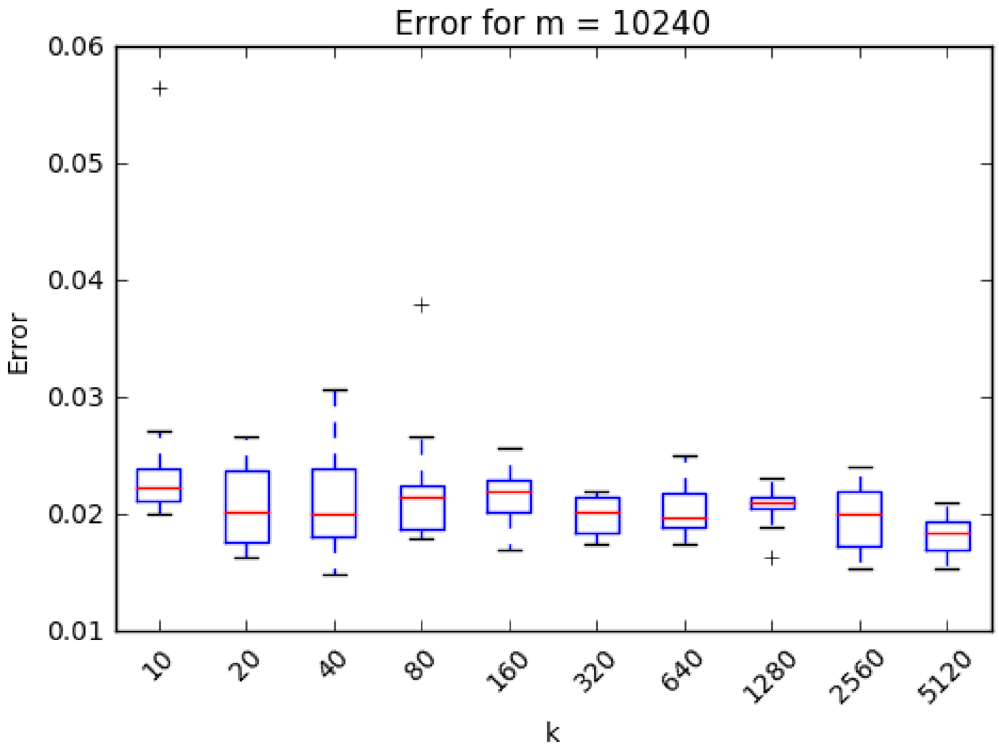


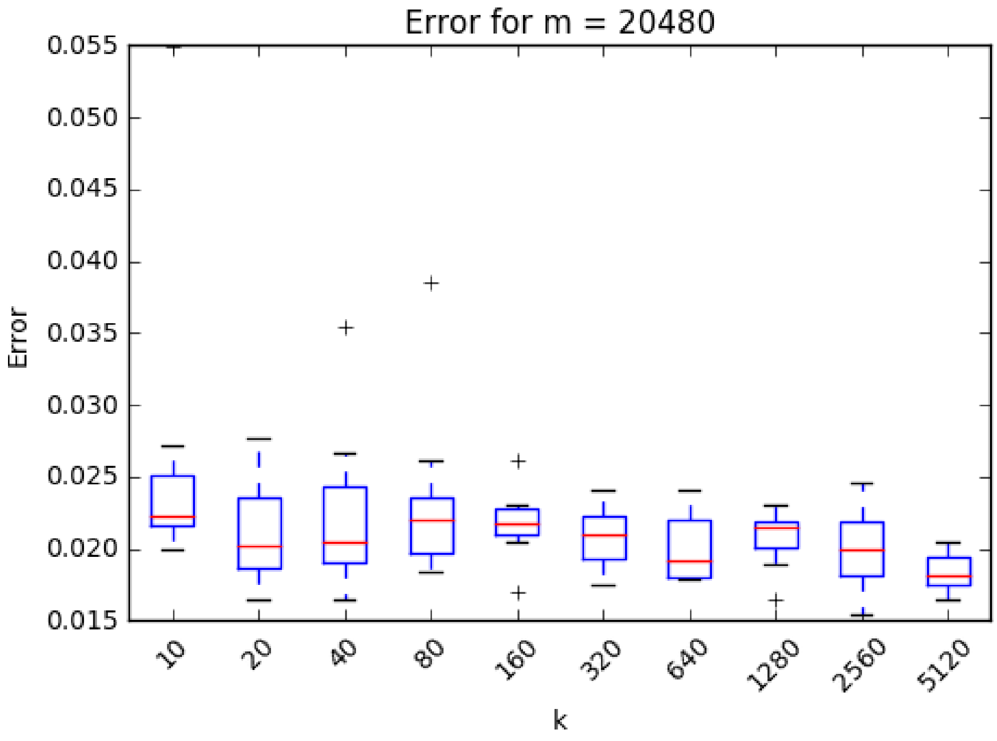
Error for m = 640
Error
0.05
0.04
0.03
0.02
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 1280
Error
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 2560
Error
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 5120
Error
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 10240
Error
0.06
0.05
0.04
0.03
0.02
0.01
10
20
40
80
160
320
640
1280
2560
5120
k


Error for m = 20480
Error
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
k
10
20
40
80
160
320
640
1280
2560
5120