Optimisation of the Muon Injection for a Muon EDM Experiment

Semester Project

written by

Anton Holmberg

supervised by

Dr. A. Adelmann (PSI & ETH)
Dr. P. Schmidt-Wellenburg (PSI)

August 2, 2021

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Optimisation of the Muon Injection for a Muon EDM Experiment

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Holmberg	Anton

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 21.07.2021	Anton Holmberg

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

1 Introduction

At the Paul Scherrer Institute (PSI) the feasibility of the search of a muon electric dipole moment (EDM) using the frozen spin technique, [1], is currently being explored, see letter of intent [2]. This project aims to optimise the injection of muons for one of the proposed setups. This is important because better injection efficiency essentially means the same statistical power in less time. The setup in question is a vertical injection similar to what is proposed by the J-PARC (g-2) group [3]. The muons are injected into the a storage ring form above, mostly horizontally but at an angle so they approach the storage ring in a helix. The vertical component of the velocity is then stopped by a magnetic kicker field. The mouns are then stored in the storage ring until they decay. Because of the electric field freezing the spin, and the fact that the muon decay is asymmetrical with regards to spin, there will, in the presence of an EDM, be an up/down asymmetry in the decay positrons that can then be measured.

The motivation behind such an experiment, to try to measure a particle EDM, is that a successful measurement of one would indicate a charge-parity violation (CPV). If one is not found it would place a new upper bound on the muon EDM. CPV constitutes the violation of the combination of charge conjugation symmetry and parity symmetry, meaning that if a particle is changed for its antiparticle and the spatial coordinates are mirrored, the laws of physics are still the same. Plenty of CPV:s have been found in the weak interactions of the standard model, but this far none have been found in the strong interactions, see the review by Particle Data Group [4]. One of the reasons a CPV is interesting is that it would be a part of explaining the baryon asymmetry in the universe [5].

The experiment is intended to be placed at the muE1 beamline at PSI. Simulations of the injection of the muE1 beam into the storage ring are done in G4beamline (3.06) [6] and fieldmaps for the simulations are created using Agrossuite and G4beamline. Because of the high number of parameters, optimisation of the injection using traditional optimisation techniques is not feasible due to prohibitive time to solution. Instead the goal is to create a surrogate model using polynomial chaos expansions (PCE) to then be used to either find an optimum, or at least limit the search space for the parameters affecting the injection efficiency. This is done following [7], a surrogate model could also be used later to give an uncertainty quantification (UQ) of the found optimum.

As a comparison to the PCE surrogate, a surrogate model is also constructed using a feed-forward neural net. This is a simple neural net predicting the injection efficiency based on the values of the parameters affecting the injection. An overview of machine learning based surrogate modelling can be found in the review Machine-Learning Methods for Computational Science and Engineering [8].

2 Simulation setup for the vertical injection

The vertical injection scheme works by injecting the muon beam into the solenoid field from above at an angle. So that the muon approaches the desired orbit in a helix. This is done to, among other things, not have the muon beam go through the electrodes. A magnetic kicker field is then used to remove the vertical component of the particles velocity. The described setup can be seen in figure 1.

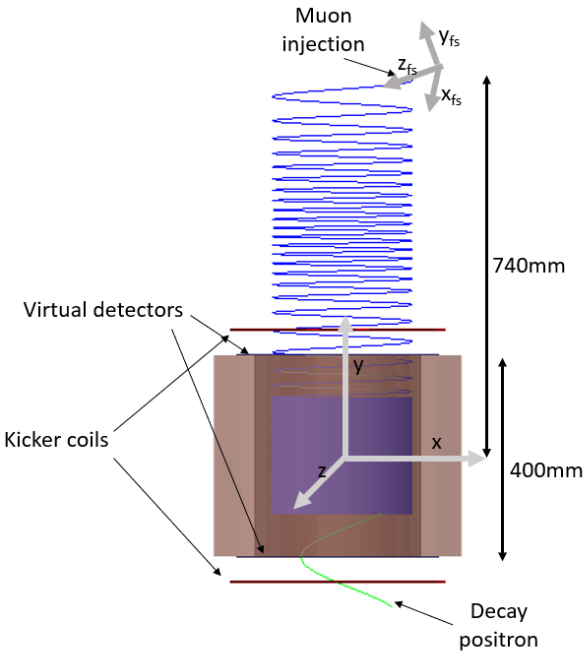
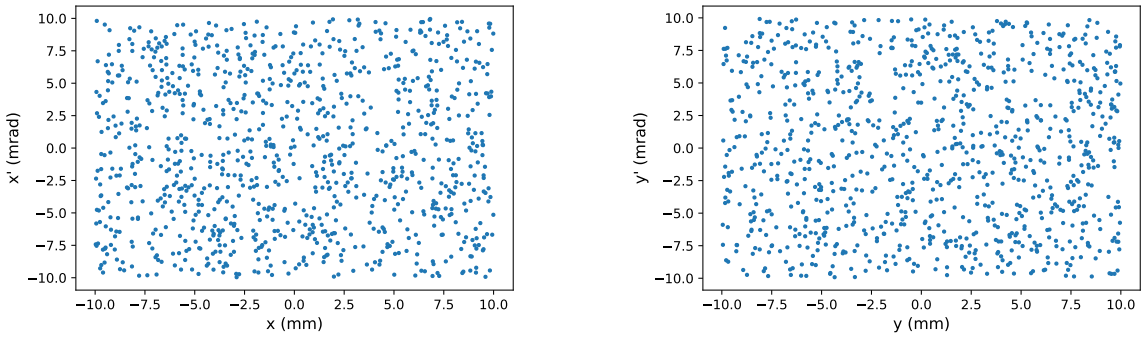


Figure 1: The setup for simulating the injection in G4beamline

The 4-dimensional injection phase space for the setup is defined by a uniform distribution for now as it is a good enough approximation of the true phase space of the real beam. The real beam is gaussian with a large sigma but due to mainly space constraints, e.g. spacing of electrodes and detectors, only the center of the beam is relevant hence, a uniform distribution is a good approximation. The beam is 20 mm $\times$ 20 mrad in both the vertical and horizontal phase space see figure 2. All of the particles were given the same total momentum of 125 MeV.



(a) **(b)**

Figure 2: Plots of the transversal injection phase space. In both cases a uniform distribution $\pm$10mm and $\pm$10 mrad.

A successful injection was first defined as a muon that decays between the top and bottom virtual detectors. The non successful injections were classified as deflected if the muon gets

deflected by the kicker and/or the storage field, and as through if the moun does not stop inside the storage ring but rather continues through, as the kicker did not slow it down enough. The different outcomes can be seen in figure 3. Later, to speed up simulations, all the tracks that had not yet decayed after 1.5 µs were terminated. To classify the injection attempts only information on where the tracks end is necessary, this is provided by the *beamlossntuple* in G4beamline. This method classified all of the injections the same as the previous method but with only one output file instead of three and half the computation time.

A fourth, less interesting outcome is also possible, that the muons decay before reaching the storage ring. However, since it is difficult to distinguish between those and the deflected particles and since there is not much that can be done about it, they are grouped together.

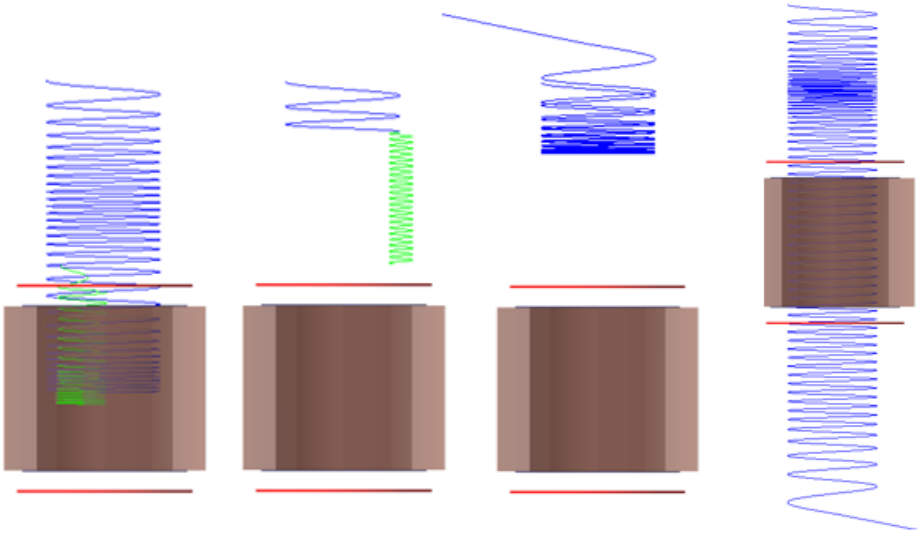


Figure 3: The possible results of the muon injection in G4beamline. The furthest left is a successful injection, second is where the muon decays before reaching the storage ring, third is the case where the muon gets deflected by the kicker field and last is where the muon goes straight through.

There are many parameters that affect the rate of successful injections. Firstly there are the EM-fields, the electric field can not be changed to optimise the injection since it is specified by what is required by the frozen spin technique. Then there are the two magnetic fields, the main solenoid field and the kicker field. This project was limited to changing the kicker field, however, the solenoid field can obviously also have a large impact on performance by varying things such as the strength of the focusing or how large the storage region is. The kicker field is computed separately from the beamline simulation and implemented from a text file. It is computed, first using Agros2d and later with G4beamline, as a static field, which is described further in the next section. In the fieldmap text file a time dependency is added, the magnetic pulse is approximated as a simple half period sine function. Only a few time steps need to be defined and then G4beamline interpolates between the values. The

time duration of the pulse is referred to as the ΔT and the timing of when the pulse reaches its peak strength as the δt.

The two other parameters controlled by G4beamline are the injection angle of the muon beam ϕ_{inj}, only vertical, and the current of the fieldmap for the kicker field i.e. a scaling of the strength of the field referred to as S.

3 The EM-fields

This project focuses on the optimisation of the magnetic field of the kicker but two more fields are needed for the simulations, the solenoid field that actually keeps the muons on an orbit and provides some week focusing and the electric field required for the frozen-spin technique.

The kicker is in this case generated with a pair of anti-helmholtz coils, see figure 4. The parameters varied in the calculations of the different configurations were the height h, the vertical distance from the desired orbit of the stored muons ($z = 0$) to the bottom of the coil, and the width w, the horizontal distance from the design orbit ($r = 140$) mm and the inside of the coils.

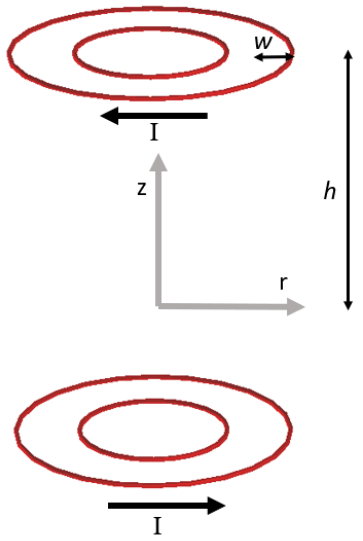
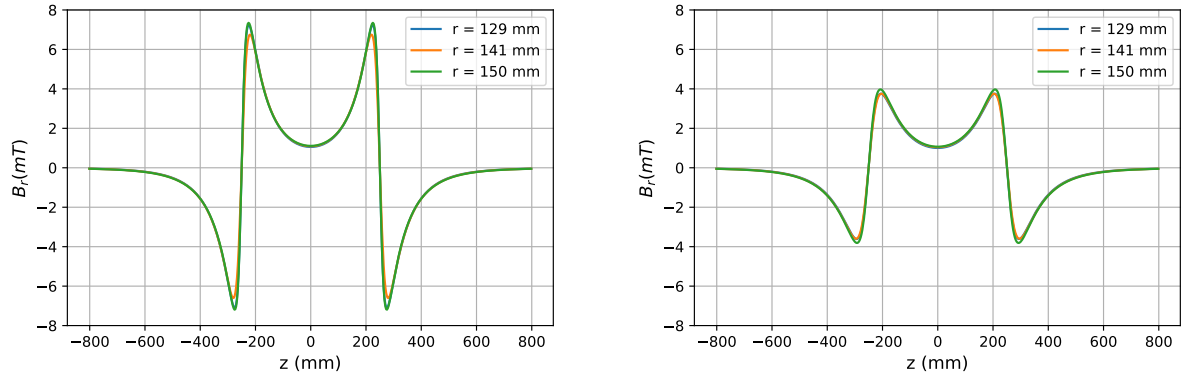


Figure 4: The anti-helmholtz coil pair setup with the parameters w and h. All currents have the same current I.

At first the fields were computed using Agros2d/Agrossuite but later, after it became apparent that it was basically impossible to run the Agrossuite python package on merlin, the PSI compute cluster, it was switched for doing the calculations in G4beamline using the *coil* and *solenoid* commands. This was possible since the calculation were computationally not very demanding.

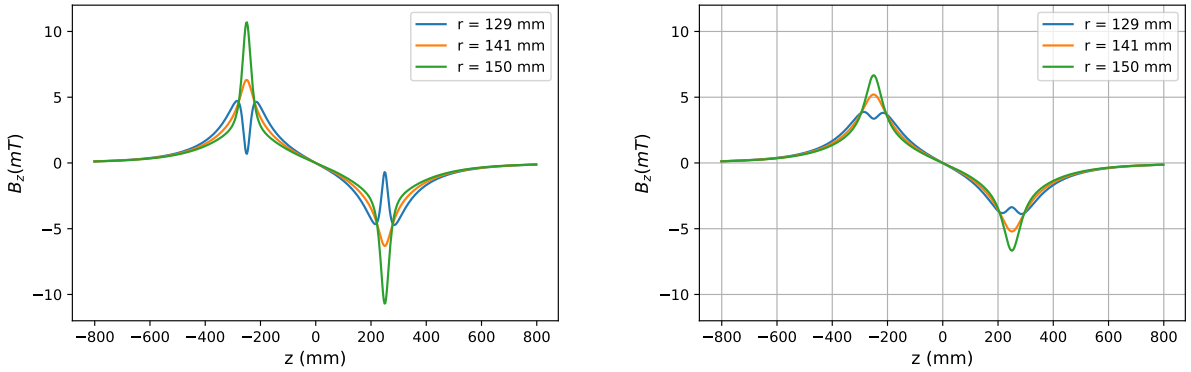
The kicker field should ideally be solely in the radial direction, since a vertical component would have an effect on the radius of the orbit. With this setup there is a z-component to the field, see figure 6, it is, however, smaller than it seems because of the time dependence.

Since the field will not yet have ramped up to full strength when the muon passes the region of of high B_z. The z-component is also pretty insignificant compared to the solenoid field, see fig 8b, at less than 1% of it. So the fact that there is a z-component should not affect the results much, if at all.



(a) $w = 30$ mm **(b)** $w = 50$ mm

Figure 5: The radial component of the kicker field for coils placed at $h = 250$ mm and two different values of w, h being the vertical distance of the coil to $z = 0$ and w, the distance from the coil to the design radius ($r = 14$ mm)



(a) **(b)**

Figure 6: The vertical component of the kicker field for coils placed at $h = 250$ mm and two different values of w, h being the vertical distance of the coil to $z = 0$ and w, the distance from the coil to the design radius ($r = 14$ mm).

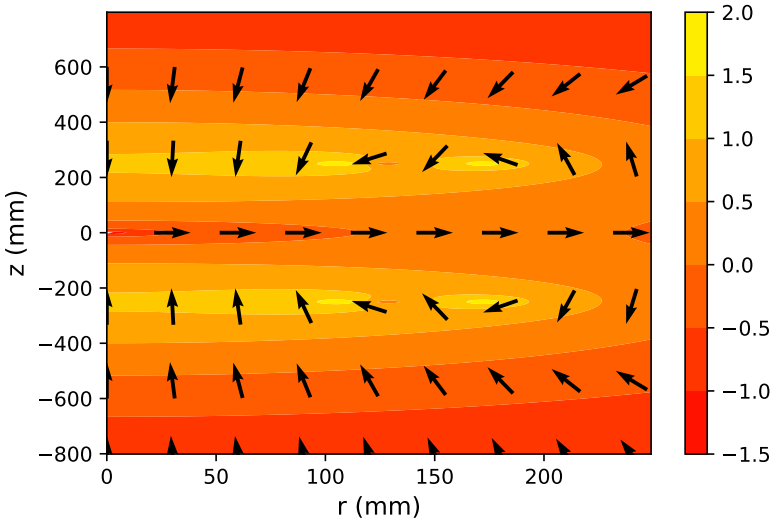
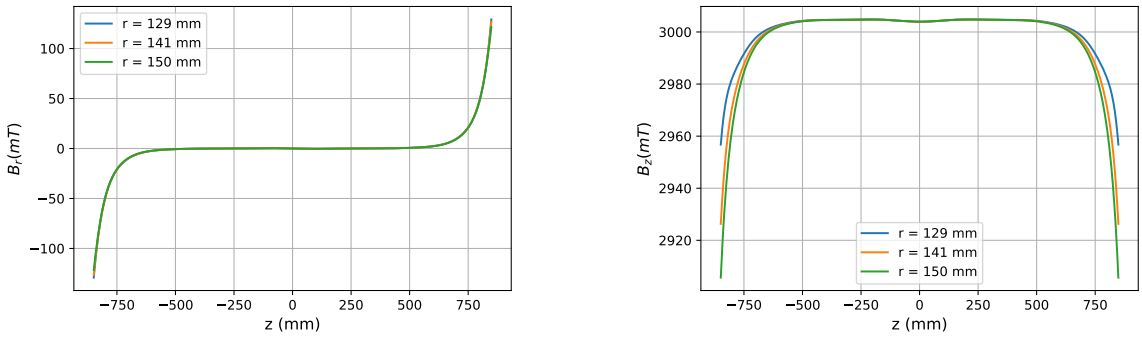


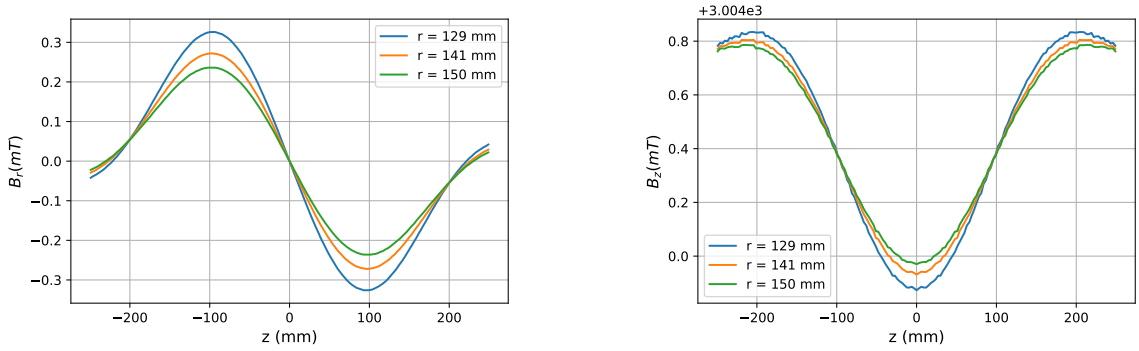
Figure 7: The magnitude and direction of the kicker field for coils placed at $h = 250$ mm and $w = 30$ mm, h being the vertical distance of the coil to $z = 0$ and w, the distance from the coil to the design radius ($r = 14$ mm). The colour map of the B-field magnitude is scaled logarithmically and is in units of (mT).

The solenoid field, see figure 8, is highly homogeneous to make the injection very nonsensitive to the horizontal phase space at injection, the field strength is virtually the same no matter what radius, within the relevant height. The solenoid field also provides some weak focusing from its radial component, even though it is not very visible in figure 8a. Both the homogenety and the focusing can be more clearly seen in figure 9, the same solenoid field but only for the storage region.



(a) **(b)**

Figure 8: The radial and vertical component of the highly uniform main solenoid field. In the entire defined region.



(a) **(b)**

Figure 9: The radial and vertical component of the highly uniform main solenoid field. In only the storage region to better display the uniformity and the focusing.

4 Methods

First a parameter scan was done on all of the parameters individually. Keeping the other parameters constant, one parameter at a time was varied in a range. This was done for two purposes: first to make sure that the parameters actually have an impact on the injection efficiency and second, tho give an indication of a range of the parameters for the surrogate model.

4.1 Polynomial chaos expansion surrogate

Polynomial chaos expansions (PCE) were first introduced by Wiener in 1938 [9] but was not used much until fairly recently. This project uses the generalised polynomial chaos introduced by [10]. PCE is used for surrogate modelling, i.e. it replaces the real model with an approximation to speed up computations. The speedup can be many orders of magnitude, which can be very useful but only if the the approximation is good enough. Good news is that PCE is known to converge very fast, sometimes exponentially, for certain problems [10]. This means PCE can be a very good option for doing uncertainty quantification or optimisation.

In principle PCE works by exploiting the fact that a quantity of interest, as long as it is a square integrable, second order random variable with a finite variance output can be expressed:

$$u(\boldsymbol{\xi}) = \sum_{i=0}^{\infty} \alpha_i \Psi_i(\boldsymbol{\xi}).$$

Where Ψ are the polynomial basis functions, the basis is chosen according to the Wiener-Askey scheme [10] and it depends on the distribution of $\boldsymbol{\xi}$. The coefficients α_i are deterministic. In this case the quantity of interest u is the injection efficiency depending on the parameters: $S, \phi_{inj}, w, h, \delta t, \Delta T$.

The expansion is truncated to:

$$\hat{u}(\boldsymbol{\xi}) = \sum_{i \in I_{d,p}}^{K} \alpha_i \Psi(\boldsymbol{\xi}).$$

Where d is the dimension of the parameter space and p is the total order of the multivariate polynomials.

There are many ways of obtaining the coefficients α_i however, in this case only the non-intrusive methods are interesting since intrusive methods would require modifications of the solver (in this case G4beamline), which would be way too time consuming. Instead the solver is viewed as a black box, denoted $\mathcal{M}$. A number N samples ξ^n are then drawn from the distribution of the parameters, in this case a uniform distribution, and the model is then evaluated on these points: $u^n = \mathcal{M}(\xi^n)$ to create a training set used to fit the expansion.

There are two methods used in this project for fitting the expansion pseudo-spectral projection and point collocation. The first one, pseudo-spectral projection is described in algorithm 1 and is based on quadrature. The second one is point collocation and is described in algorithm 2 and is based on regression to fit the coefficients. The training points used are preferably a low discrepancy sequence, in this case sobol points, see [11]. It was implemented in python with the package ChaosPy [12].

Algorithm 1 Fitting a PCE using pseudo-spectral projection

1: Define the distribution function of the parameters $\rho(\boldsymbol{\xi})$. In our case a 6-dimensional uniform distribution. Create with *chaospy.Uniform* and *chaospy.J*.
2: Generate a set of quadrature point-weight pairs from the distribution $\{\xi_i, w_i\}_{i=1}^n$ with $n = (p+1)^d$ where $d = 6$ and p is the polynomial order. This is done with *chaospy.generate_quadrature*.
3: Generate training points by using the quadrature points as model inputs $u_i = \mathcal{M}(\xi_i)$, into G4beamline.
4: Generate the PCE depending on the distribution of the parameters. This is done with *chaospy.generate_expansion*.
5: Fit the expansion with quadrature using the point-weight pairs, i.e. compute the coefficients. Done using *chaospy.fit_quadrature*.

Algorithm 2 Fitting a PCE using point collocation

1: Define the distribution function of the parameters $\rho(\boldsymbol{\xi})$. In our case a 6-dimensional uniform distribution. Create with *chaospy.Uniform* and *chaospy.J*.
2: Generate a set of sobol points $\{\xi_i\}_{i=1}^n$, or any other low discrepancy sequence for that matter, where n is a number to be chosen, higher is better but more time consuming for the simulations. This is done with *.sample* on the distribution.
3: Generate training points by using the sobol points as model inputs $u^i = \mathcal{M}(\xi^i)$, into G4beamline.
4: Generate the PCE depending on the distribution of the parameters. This is done with *chaospy.generate_expansion*.
5: Fit the expansion to the training points using regression, in our case least squares. This is done with *chaospy.fit_regression*.

Some initial tests showed that in this case the regression based point collocation got a better fit than the quadrature based pseudo-spectral projection. So most of the effort was put

into improving the regression based method. The other advantage is that with the regression based one the parameter space is also much better covered because of the use of sobol points instead of quadrature points. To improve the convergence of the PCE the ranges of the parameters were successively narrowed since, at first, most of the training points were just zero and convergence can be improved with a flatter distribution of training points. The number of training points is decided by the polynomial order in the case of pseudo-spectral projection and after order four it becomes prohibitively many, order four is already 15000 training points. For the regression based point collocation only 4000 points were used for training and an additional 1000 unseen points for testing. As the regression model, least squares was used as it provided the best results.

4.2 Feed-forward neural network surrogate model

Neural networks (NN) are very good at approximating models and are therefor used in a wide range of applications within computational science, one of them being surrogate modelling [8]. A NN is especially good at learning a model when using data with very low noise which suits simulated data perfectly. Though the G4beamline simulations has got a random component, this is very small when using a large number of particles, so the data is practically without noise. The network is trained to make predictions on injection efficiency based on values of the parameters.

For training and testing the same set of data as for the regression based PCE was used, 4000 points for training and 1000 for validation. This was because low discrepancy based training sets have been shown to outperform the more standard random samples, for a lot of cases [13].

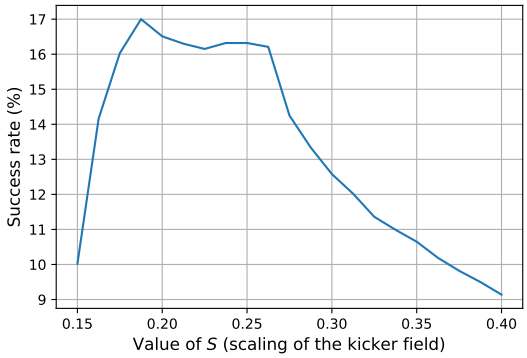
Not much was done to optimise the training but some light, manual tuning was done. So the performance can definitely be improved quite a bit. The final configuration was a 6 layer deep and 512 neurons wide on all levels. As activation function leaky ReLU was used [14], and as optimiser ADAM was chosen [15]. Some standard optimisers and activation functions were compared but these gave the lowest validation error. Some regularisation was used, even though the data had low noise this gave a lower validation error then no using it.

The NN was implemented in pytorch [16].

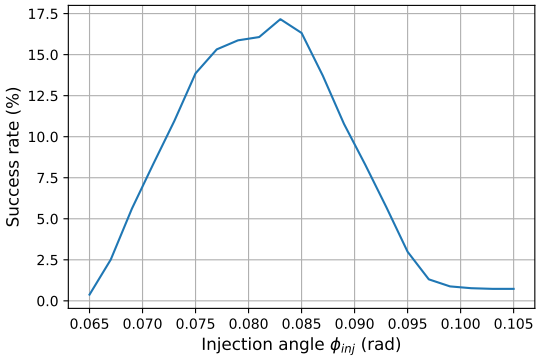
5 Results

5.1 Parameter scan

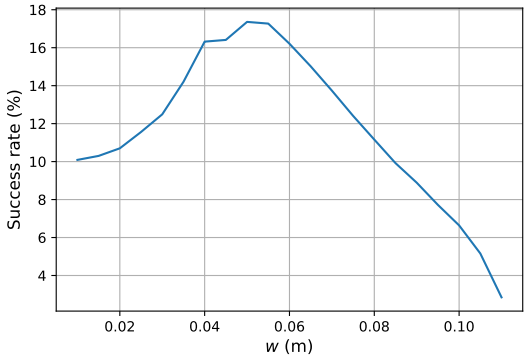
The results from the parameter scan seen in figure 10 show clearly that all of the chosen parameters have a big impact on the injection efficiency. While it gives a clear optimum for the parameters, it is not that interesting since all of the parameters affect each other. The most simple example is how the strength of the kicker field and the duration of the pulse together gives the change to the trajectory. However, the results give an indication of the interesting range for the parameters. The ranges of the parameters chosen for the surrogate modelling can be seen in table 1. They were intentionally picked to be very wide ranges as to try to not miss any optima.



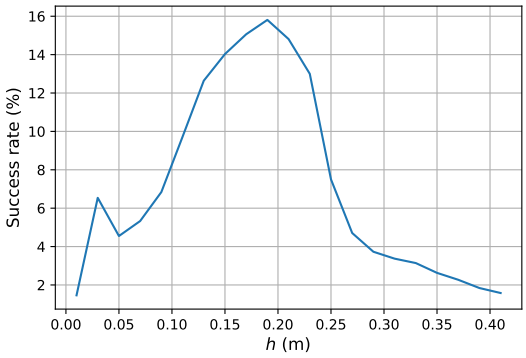
(a)



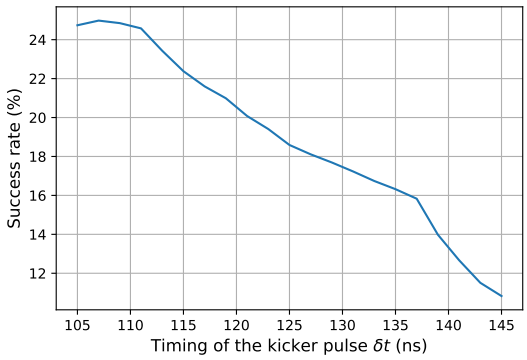
(b)



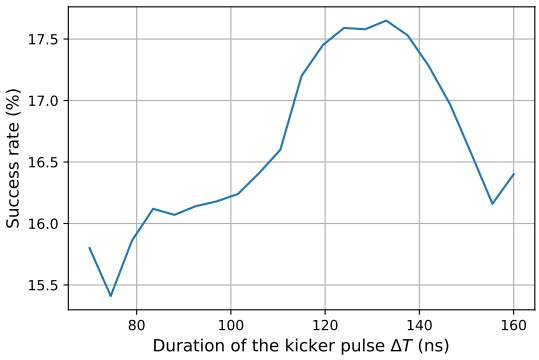
(c)



(d)



(e)



(f)

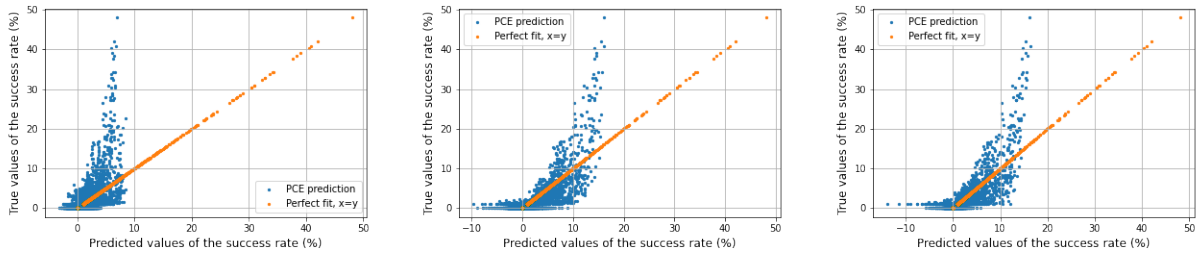
Figure 10: A scan of each individual parameter. The parameters not being changed has the values: $S = 0.25, \phi_{inj} = 0.085rad, w = 0.04m, h = 0.2m, \delta t = 135ns, \Delta T = 90ns$

Parameter	Lower bound	Upper bound
S	0.1	1
ϕ_{inj}	0.04 (rad)	0.1 (rad)
w	0.02 (m)	0.08 (m)
h	0.05 (m)	0.4 (m)
δt	70 (ns)	250 (ns)
ΔT	50 (ns)	200 (ns)

Table 1: The first bounds for the surrogate modelling decided from the parameter scan.

5.2 PCE using pseudo-spectral projection

The convergence of the PCE with pseudo-spectral projection using gaussian optimal quadrature were a lot worse than expected, see figure 11. Even with fourth order, meaning over 15000 training points, the model is not very accurate, see fig 11c. A quick test with fitting a PCE using regression on these training points showed a better convergence so no further effort was put in to improving the convergence of this model, instead focus was put on the regression based model. Even tough this model probably also could have improved from narrower parameter bounds.

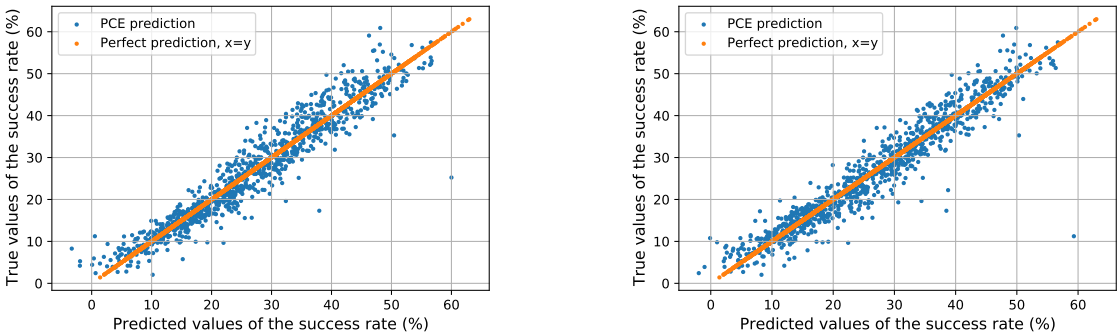


(a) Polynomial order 2 **(b)** Polynomial order 3 **(c)** Polynomial order 4

Figure 11: The predicted values of the PCE surrogate using pseudo-spectral projection vs true values for a few different orders of the polynomial chaos expansion.

5.3 PCE using point collocation

Over a few iterations of creating training points and looking at the PCE fit and the distribution of training points, then subsequently changing the parameter bounds, the final bound for the parameters was decided and can be seen in table 2. The fit of the regression model is quite good, see figure 12, it has almost stopped giving negative predictions but still has some outlier predictions that are very far from the true value. Using all of the 4000 training points PCE with polynomial order 5 gave the lowest mean square error of $MSE = 13.31$, slightly lower than the $MSE = 13.75$ that order 6 gave. The convergence of the PCE for different polynomial orders, with respect to mean square error can be seen in figure 13.



(a) Polynomial order 5

(b) Polynomial order 6

Figure 12: The predicted values of the PCE surrogate using point collocation vs true values for a couple different orders of the polynomial chaos expansion. Done with 4000 training points and 1000 unseen validation points.

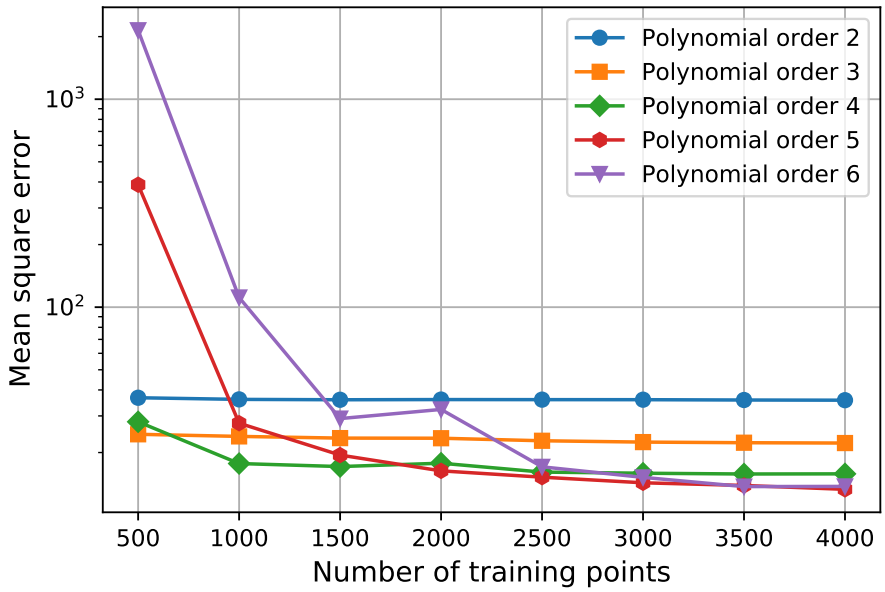


Figure 13: Convergence in the mean square sense of the PCE fitted using point collocation, with regards to number of training points, for different total polynomial order.

Parameter	Lower bound	Upper bound
S	0.65	1
ϕ_{inj}	0.075 (rad)	0.095 (rad)
w	0.02 (m)	0.05 (m)
h	0.2 (m)	0.3 (m)
δt	70 (ns)	95 (ns)
ΔT	70 (ns)	120 (ns)

Table 2: The final bounds for the regression based surrogate model.

5.4 Feed-forward neural network surrogate model

The neural network based surrogate model actually turned out to be more accurate than the PCE surrogates. The mean square error for the model is $MSE = 4.27$. Looking at the predictions in figure 14, the model still has some outliers in the predictions but are improved with respect to the PCE.

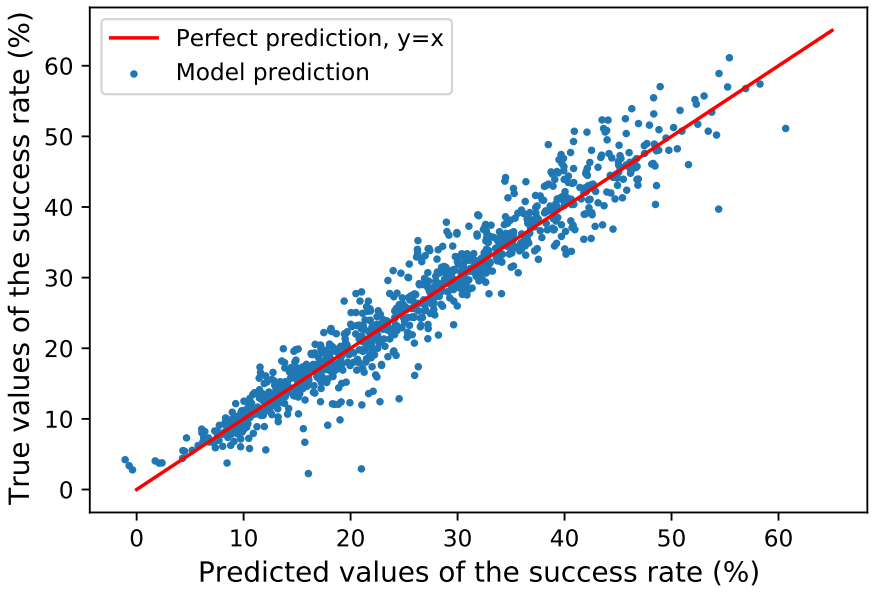


Figure 14: Predicted values from the neural network surrogate model vs true values. Trained on 4000 data points and validated with 1000 unseen data points.

5.5 Summary of the different surrogate models

As you can see in table 3 the NN based surrogate outperforms the other two models in terms of MSE. The other observation one can make from the table is that the quadrature based PCE outperforms the regression based PCE, this however, seems to be because of the good predictions close to zero injection efficiency. Predictions away from zero can be quite poor

Model	MSE
NN model	4.27
Point collocation order 6	13.7
Point collocation order 5	13.3
Point collocation order 4	15.8
Point collocation order 3	22.2
Pseudo-spectral projection order 4	6.59
Pseudo-spectral projection order 3	7.84
Pseudo-spectral projection order 2	11.9

Table 3: The found mean square errors for the different models. This is not a completely fair comparison since the parameter bounds are different in the case of pseudo-spectral projection compared to the other two methods.

and still give low MSE since most points are close to zero. Although the comparison is not quite fair since parameter bounds are different from the pseudo-spectral projection to the two other models.

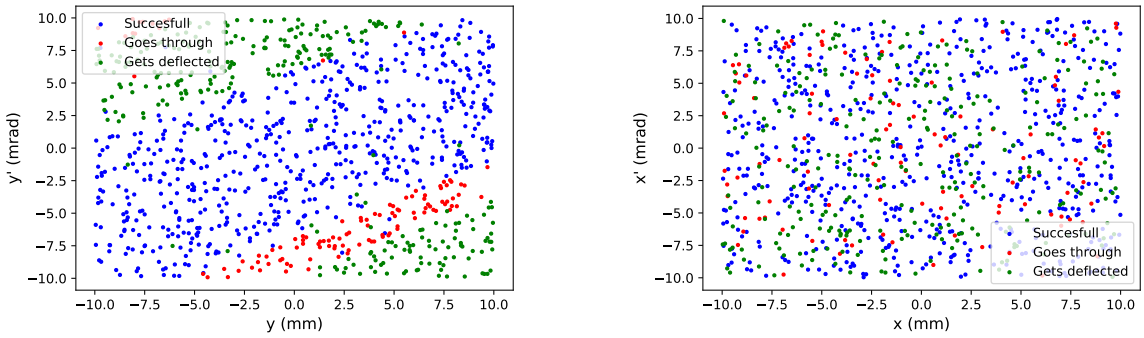
5.6 Interesting training points

Some of the training points already have a very high injection efficiency of over 60%. The ten points with the highest injection efficiency can be seen in table 4. The most obvious trend that can be observed is that all of the ten best points have a really short timing δt. Good points in terms of injection efficiency with slightly longer δt (not in the table) tend to have longer pulse length ΔT which means that the start of the pulse is still at about the same time. So a lot of the high efficiency points have a pulse that starts after around 30 ns even if it sometimes peaks later.

	S	ϕ_{inj} (rad)	δt (ns)	ΔT (ns)	w (m)	h (m)	transmission %
1	0.996	0.0871	70.87	80.93	0.0395	0.263	62.99
2	0.959	0.0858	71.78	82.35	0.0247	0.251	62.79
3	0.972	0.0861	73.36	92.35	0.0491	0.281	61.88
4	0.970	0.0877	72.80	88.29	0.0348	0.264	61.13
5	0.830	0.0867	72.12	87.14	0.0221	0.261	60.75
6	0.969	0.0868	70.68	72.15	0.0375	0.244	60.50
7	0.955	0.0876	73.61	92.85	0.0260	0.262	60.25
8	0.931	0.0859	70.41	101.5	0.0448	0.288	60.14
9	0.860	0.0843	70.02	100.7	0.0331	0.287	59.61
10	0.820	0.0853	72.08	95.65	0.0265	0.275	59.55

Table 4: The ten best points from the training set.

The injection phase space for the highest efficiency case is plotted in figure 15. It is easy to see from figure 15b that in this case, with this injection phase space, the horizontal phase



(a) **(b)**

Figure 15: The transversal injection phase space divided into sets depending on whether the injection is successful, goes through or gets deflected.

space does not matter for whether the injection is successful or not. However, in figure 15a, a clear pattern is visible. There are some data points in the plot that might look odd but for example the ones in the successful region that get labeled as deflected are muons that decay before reaching the storage region. The successful muons in the "goes through" region are muons that would have gone through but happened to decay in before exiting the storage ring.

In figure 16 correlations of the different parameters are plotted for three groups of the success rate: under 20, between 20 and 40, and above 40. Most interesting is the histograms on the diagonal since the distributions of the different groups are quite clear. While all of the parameters have a very big impact on the success rate on their own, for some of them it seems like for every value in the range there are combinations of the other parameters which are good.

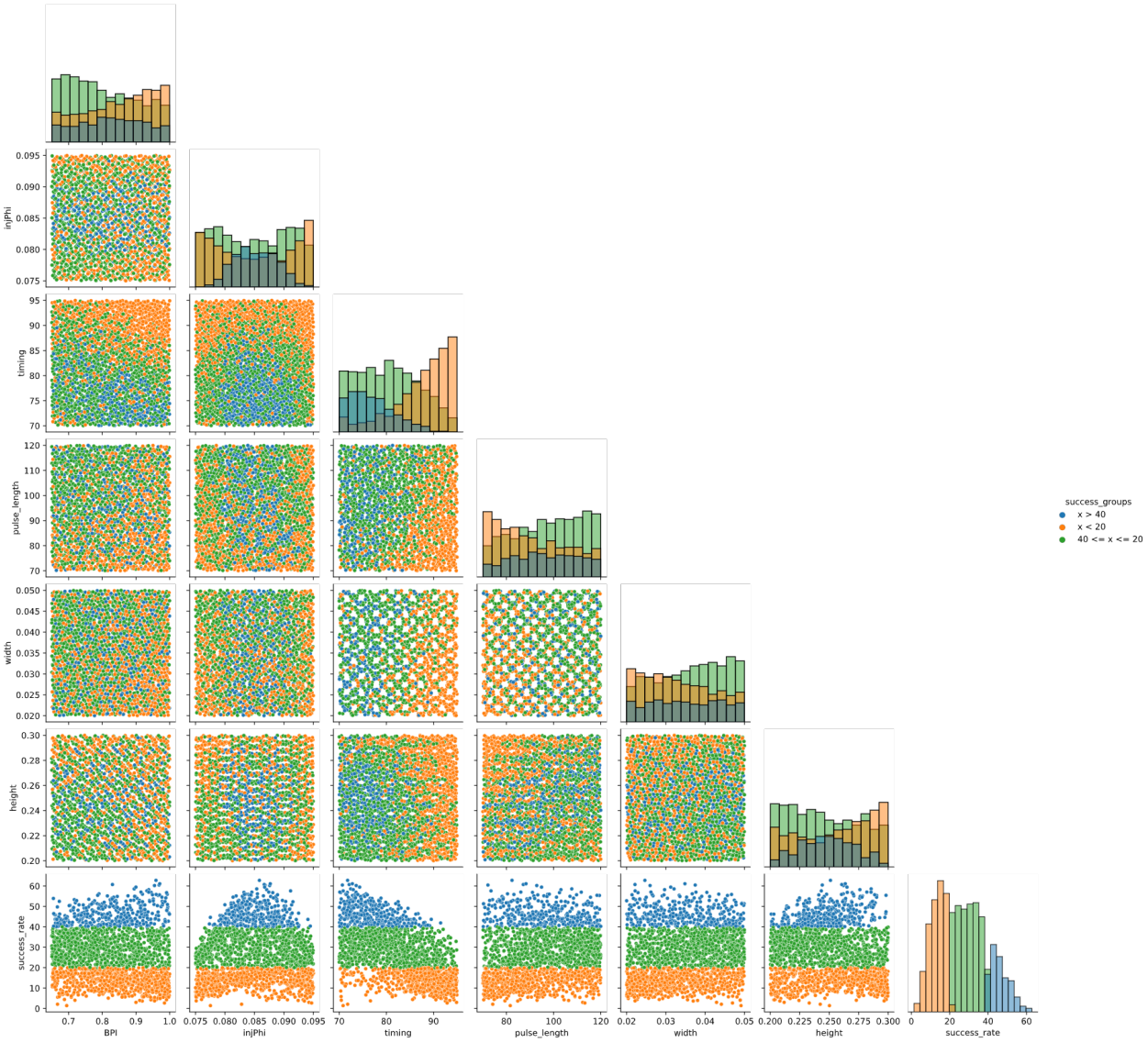


Figure 16: Correlations of the sobol training points used for point collocation.

6 Conclusions

In the efforts to fit a surrogate model some points were found with high injection efficiency which is very promising for the continuation of the project. The surrogate fit, though not perfect, should be good enough to be used for optimisation or uncertainty quantification, although that is unfortunately out of the scope of this project. For optimisation my suggestion would probably be to try to apply a genetic algorithm to the surrogate model since if correctly designed they can be good at avoiding local minima [17]. Otherwise conventional algorithms that require gradient information should be very efficient since the PCE is based on polynomials whose gradients are easy to compute analytically, and the neural network model which has gradient computation built in because it is necessary for the backpropagation during training. Then, use a large number of, either different random seeds or a set of the best training points, as initial guesses for the conventional algorithms and they should also be able to find a global maxima. Because of how fast the surrogate models evaluate, time to completion should not be a problem for either.

The NN surrogate model should still has a decent room for improvement. This is based on the fact that the training parameters where not optimised thoroughly but only manually tuned. For proper optimisation one should consider ensemble training following what is proposed in [18] or something similar.

However, the next step in optimising the injection should probably not be to further optimise this setup since there were variables left out of this project. Most notably the main solenoid field was not changed at all. At the moment, for low injection angles, a large percentage of the muons get deflected, not because of the kicker, but because of the potential barrier that the main solenoid field creates. At small enough angles no muons get injected successfully. If this problem can be mitigated, so that smaller angles are possible, it should be possible to get high injection efficiencies with timings δt that are longer. It not only would take longer for the muons to reach the storage ring, but they would also spend more time in the region, this could mean that the setup also gets less sensitive with regards to the timing δt. Additionally it means that the muons will have a smaller downward momentum in the storage region, which could lead to weaker fields being possible.

At the moment the injection efficiency is very good in the best cases but it might prove challenging to actually build a setup that can deliver these conditions. For the 62.99% efficiency case the kicker pulse starts at about 30ns after injection and reaches full strength after an additional 40ns, this would be very challenging to build if not impossible. This is why it is very important to try to make smaller injection angles possible.

References

[1] F. J. M. Farley, K. Jungmann, J. P. Miller, W. M. Morse, Y. F. Orlov, B. L. Roberts, Y. K. Semertzidis, A. Silenko, and E. J. Stephenson, "New method of measuring electric dipole moments in storage rings," *Phys. Rev. Lett.*, vol. 93, p. 052 001, 5 Jul. 2004. DOI: 10.1103/PhysRevLett.93.052001. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.93.052001.

[2] A. Adelmann, M. Backhaus, C. C. Barajas, N. Berger, T. Bowcock, C. Calzolaio, G. Cavoto, R. Chislett, A. Crivellin, M. Daum, M. Fertl, M. Giovannozzi, G. Hesketh, M. Hildebrandt, I. Keshelashvili, A. Keshavarzi, K. S. Khaw, K. Kirch, A. Kozlinskiy, A. Knecht, M. Lancaster, B. Märkisch, F. M. Aeschbacher, F. Méot, A. Nass, A. Papa, J. Pretz, J. Price, F. Rathmann, F. Renga, M. Sakurai, P. Schmidt-Wellenburg, A. Schöning, M. Schott, C. Voena, J. Vossebeld, F. Wauters, and P. Winter, *Search for a muon edm using the frozen-spin technique*, 2021. arXiv: 2102.08838 [hep-ex].

[3] H. Iinuma, H. Nakayama, K. Oide, K.-i. Sasaki, N. Saito, T. Mibe, and M. Abe, "Three-dimensional spiral injection scheme for the g-2/edm experiment at j-parc," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 832, Jun. 2016. DOI: 10.1016/j.nima.2016.05.126.

[4] P. A. Zyla *et al.*, "Review of particle physics," *Progress of Theoretical and Experimental Physics*, vol. 2020, no. 8, Aug. 2020, 083C01, ISSN: 2050-3911. DOI: 10.1093/ptep/ptaa104. eprint: https://academic.oup.com/ptep/article-pdf/2020/8/083C01/34673722/ptaa104.pdf. [Online]. Available: https://doi.org/10.1093/ptep/ptaa104.

[5] A. D. Sakharov, "Violation of CP Invariance, C asymmetry, and baryon asymmetry of the universe," *Pisma Zh. Eksp. Teor. Fiz.*, vol. 5, pp. 32–35, 1967. DOI: 10.1070/PU1991v034n05ABEH002497.

[6] T. J. Roberts and D. M. Kaplan, "G4beamline simulation program for matter-dominated beamlines," in *2007 IEEE Particle Accelerator Conference (PAC)*, 2007, pp. 3468–3470. DOI: 10.1109/PAC.2007.4440461.

[7] A. Adelmann, "On nonintrusive uncertainty quantification and surrogate model construction in particle accelerator modeling," *SIAM/ASA Journal on Uncertainty Quantification*, vol. 7, pp. 383–416, Jan. 2019. DOI: 10.1137/16M1061928.

[8] M. Frank, D. Drikakis, and V. Charissis, "Machine-learning methods for computational science and engineering," *Computation*, vol. 8, p. 15, Mar. 2020. DOI: 10.3390/computation8010015.

[9] N. Wiener, "The homogeneous chaos," *American Journal of Mathematics*, vol. 60, no. 4, pp. 897–936, 1938, ISSN: 00029327, 10806377. [Online]. Available: http://www.jstor.org/stable/2371268.

[10] D. Xiu and G. Karniadakis, "The wiener–askey polynomial chaos for stochastic differential equations," *SIAM J. Sci. Comput.*, vol. 24, pp. 619–644, Oct. 2002. DOI: 10.1137/S1064827501387826.

[11] I. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967, ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(67)90144-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555367901449.

[12] J. Feinberg and H. P. Langtangen, "Chaospy: An open source tool for designing methods of uncertainty quantification," *Journal of Computational Science*, vol. 11, Aug. 2015. DOI: 10.1016/j.jocs.2015.08.008.

[13] S. Mishra and K. Rusch, "Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences," *SIAM Journal on Numerical Analysis*, vol. 59, pp. 1811–1834, Jan. 2021. DOI: 10.1137/20M1344883.

[14] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[17] K. Man, K. Tang, and S. Kwong, "Genetic algorithms: Concepts and applications [in engineering design]," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, 1996. DOI: 10.1109/41.538609.

[18] K. Lye, S. Mishra, and D. Ray, "Deep learning observables in computational fluid dynamics," *Journal of Computational Physics*, vol. 410, p. 109 339, Mar. 2020. DOI: 10.1016/j.jcp.2020.109339.

A Installing programs required for the simulations

Here is a short guide on how to replicate these results and how to modify things. This is assuming that you already have access to the gitlab repository for the muonEDM project and have an account with access to the PSI compute cluster merlin. It is also assuming there is not yet a project directory set up for the project. It is written from a windows users point of view but should be easily adapted to other operating systems.

1. First step is to get VPN access since it will simplify things. If you have physical access to PSI this is very easy but if you are not able to go to PSI it can be a bit more tedious. The problem is that the page detailing how to set up your VPN account is a PSI internal website, which you can not see since you do not have VPN access yet. So either get someone to send the instructions to you or use SSH to see the website following the guide on the PSI website. Though the PSI guide on how to SSH tunnel to view internal websites is for an old version of Firefox so keep in mind that you will also have to check the box "Proxy DNS when using SOCKS v5". Then following the instructions is very simple.

2. Then for working with merlin on windows I would suggest using winSCP for transferring files and PuTTY for actually using merlin. You could also use the Windwos Subsystem for Linux (WSL) to SSH but I would suggest not to since WSL stops having internet access with a VPN running (there are probably workarounds but I do not see the point).

3. Then, installing G4beamline is fairly straight-forward. First download the latest SFL Linux version since merlin runs a red-hat distribution of Linux. Then, before doing the G4beamline installation you will need a X Window System Server for windows since the installation requires a GUI, I used Xming. Then, just follow the instructions on the G4beamline website. At the end of the installation a window should pop up for downloading the required Geant4 data files. However, this did not work for me and might not work for you but the fix is simple: go to the Geant4 download page and download the data files manually. Keep in mind that G4beamline does not always use the latest data file so you will find some of the files in the archive.

   Where you install G4beamline should not matter but the data files are large so they should be on the data directory, not on the home directory. Since the home directory in much smaller.

4. Now you can clone the gitlab repository to get the required files. At the moment they are in the holmberg_a branch but some files might be incorporated into the main branch.

5. To run the python scripts you will now need to create a python environment with the required packages. How to create one is described well in the merlin guide at: `https://lsm-hpce.gitpages.psi.ch/merlin6/python.html`. Basically what needs to be don is to create a CondaRC file to change the location in which the environment is created, so that it is in the data directory instead of the home directory. Then first load anaconda with `$ module load anaconda`, then create the environment using `$ conda create -q --yes -n 'your_environment_name' numpy chaospy pandas`.

Then, any time you want to run the python scripts just load anaconda and then activate the environment.

If you also want to run the pythorch code for the neural net on merlin you can add that package to the environment as well. But it does not take long to run so it is not necessary to use merlin for it. But if you want to really optimise the parameters for the training for example using ensemble training that could benefit from running on merlin.

B Running the simulations

The simulations are run in a few steps: first create the kicker files, second add the time dependency in the kicker files, third actually run the beamline simulation and finally analyse the G4beamline output to get the results. G4beamline allows parameters for the simulation to be set at the command line when executing the program but parameters can also be set in a separate configuration file.

What is done now is that parameters are set in configuration files, one for every setup, and what file is included in the simulation is then set in the batch script submitted to merlin. This could possibly be changed to having one configuration file with every row representing one setup with those rows being the command line input for G4beamline. This would limit the number of files required but I do not see any other advantages.

For the creation of the kickers *kicker_field.g4bl* is used. A configuration file specifies the location of the coils and the name of the kicker file, the location of this configuration file is specified in the command that starts the simulation. To create these configuration files setting the parameters use the python script *kicker_params.py*. Then, the files need a time dependency, this can be added using the *append_timedep* function in *generate_kicker_files.py*

Then, to set up the beamline simulations, configuration files for them need to e created. This is to set the parameters not controlled by the kicker files, such as the injection angle, but also to set the path to the kicker file for that simulation and the path of the output file(s) from the run. Configuration files are written by either *write_file* or *write_file_trackcuts* in *write_config_files.py*, depending on which method you want to use for classifying the muon injection, the "trackcuts" is version results in faster simulations, less output files and seems to be just as accurate. Keeping both methods so it is possible to compare if changes are made to the "trackcuts" version, e.g. cutting the tracks earlier. Then, the simulations are run using either *WeakFocusSolenoid_II.g4bl* or *WeakFocusSolenoid_trackcuts.g4bl* depending on the simulation.

To run the simulations on merlin a batch script needs to be submitted and there are two ways to submit many parallel jobs either packed jobs or array jobs. The first one is packed jobs which is more suitable for many short jobs, less than 5 minutes each, so that is more suitable for the kicker computations which take roughly 3 minutes each. The second type is array jobs which have a bit more overhead for each job so therefore are not suitable for the really short jobs, but are fine for the beamline simulations.

The packed jobs have two downsides, first is that sometimes a few of the simulations will fail and have to be rerun, second is that all g4beamline console outputs are written to the same file instead of one per simulation as in the case for array jobs. However, these

problems might be solvable in the batch script. Read more about the job types in the merlin documentation at: `https://lsm-hpce.gitpages.psi.ch/merlin6/running-jobs.html`.

Below is an example of a packed job:

```
#! /bin/bash
#SBATCH --job-name=YOUR JOB NAME
#SBATCH --partition=daily
#SBATCH --ntasks=1
#SBATCH --time=03:00:00
#SBATCH --ntasks=44    # defines the number of parallel tasks
#SBATCH --output=--/G4_.%j.out

for i in {0..999}
do
   srun -N1 -n1 -c1 --exclusive --/G4beamline-3.06/bin/g4bl --/kicker_field.g4bl configPath=--/kicker_params_${i}.txt &
done
wait
```

Where −− is of course replaced by the path to wherever the file is located for you.

Below is an example of an array job:

```
#!/bin/bash
#SBATCH --job-name=YOUR JOB NAME
#SBATCH --partition=daily
#SBATCH --ntasks=1
#SBATCH --time=02:00:00
#SBATCH --array=0-1999
#SBATCH --output=/data/user/holmberg_a/G4_logs/G4_surrogate_sobol_2000_II.%j.out
srun --/G4beamline-3.06/bin/g4bl --/WeakFocusSolenoid_new.g4bl configPath=--/params_surrogate_${SLURM_ARRAY_TASK_ID}.txt
```

Where, again, −− is of course replaced by the path to wherever the file is located for you.

Then, to get the results use *check_for_particles.py* or *check_for_particles_trackcuts.py* depending again on which method was used in the simulations.

The idea is then to write some scripts that use these functions mentioned to set up your simulation and get results. As an example this is how to set up the simulations using sobol points for the parameters, from a distribution defined using chaospy, and in the ed get a csv file with the parameters and the injection efficiency corresponding to the parameters, using the trackcuts version:

1. First step is to change the paths in the scripts to where you want the files to be created.

2. Load anaconda and activate the environment.
   `$ module load anaconda`
   `$ conda activate YOUR_ENVIRONMENT_NAME`

3. Run *PCE_setup.py* (possibly changing number of points first). This creates the two sets of configuration files and a csv with the parameters for the different runs.
   `$ python PCE_setup.py`

4. Submit the batch script that starts the *kicker_fiels.g4bl* to merlin. Make sure this is done in the data directory since G4beamline creates creates a lot of temporary files for the field map, two per run, this will fill up the home directory very fast.
   `$ sbatch YOUR_BATCH_SCRIPT`

5. Add time dependency to the kicker files using *timing_of_g4kickers.py*. This reads the csv from earlier to get the timing and pulse length.
   `$ python timing_of_g4kickers.py`

6. Now run the actual beamline simulation with by submitting the next batch script. You might want to wait for a lot of nodes on merlin to be empty before submitting otherwise

this can take a long time. Check using `$ sinfo`
`$ sbatch YOUR_BATCH_SCRIPT`

7. Now get the results by running *get_results_trackcuts.py*.
   `$ python get_results_traccuts.py`

8. You can now use this set of points for the surrogate modelling.

An example of the PCE surrogate fitting can be found in *PCE_sobol.ipynb* and of the NN surrogate in *torch_surrogate_model.ipynb*.