SPACE-PINN: A NEW PARADIGM FOR TRAJECTORY OPTIMIZATION WITH KINEMATIC STATE-TRANSFORMED NEURAL NETWORKS *

Samuel Schönherr
TU Graz
Univ
City
{Author1, Author2}email@email

Jaemin Seo
Chung-Ang University
Univ
City
email@email

Sascha Ranftl
NYU Courant
Univ
City
email@email

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

***K*eywords** First keyword · Second keyword · More

1 Introduction

1.1 PINNs

[1, 2]

1.2 PINNs with exact constraints

[3, 4, 5, 6, 7, 8, 9][10]

[11]

[12][13]

2 Methodology

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula. See Section 2.

* *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

2.1 PINNs

2.2 Trajectory optimization with NNs

In order to use NNs to optimize spacecraft trajectories, the applicable physical laws must be converted into residual form, using the gravitational potential Φ as a starting point. For three masses M_i at positions (x_i, y_i), Φ is knowingly given by

$$\Phi(\mathbf{r}) = \sum_{i=1}^{3} \frac{-GM_i}{\mathbf{r}_i}, \quad \text{where } \mathbf{r}_i = \sqrt{(x-x_i)^2 + (y-y_i)^2}. \tag{1}$$

Now one wants to know what force $\mathbf{F}_G$ is acting on the spaceship with mass m_S at this potential. To derive the force at work the negative gradient of Φ is formed and multiplied with m_S.

$$\mathbf{F}_G = -m_S \nabla \Phi = \sum_{i=1}^{3} -\frac{Gm_S M_i}{{\mathbf{r}_i}^3} \begin{pmatrix} x - x_i \\ y - y_i \end{pmatrix} \tag{2}$$

The next step is to form newton's equation of motion with the gravitational force $\mathbf{F}_G$. This then yields

$$m_S \mathbf{a} = m_S \frac{d^2\mathbf{r}}{dt^2} = \sum_{i=1}^{3} -\frac{Gm_S M_i}{{\mathbf{r}_i}^3} \begin{pmatrix} x - x_i \\ y - y_i \end{pmatrix} = \mathbf{F}_G \tag{3}$$

Finally, the time derivative is replaced by the normalised time derivative which comes with dividing by the total time squared t_N^2 which is needed to travel the trajectory. Now the spacecraft needs 1 s for every trajectory given by a boundary problem. Bringing the r.h.s of eq. (3) to 0, and dividing by m_S gives the desired mathematical form of a thrust vector $\mathbf{T}$ which can be minimzed in the training process of a PINN.

$$\mathbf{T} = \frac{1}{t_N^2} \frac{d^2}{dt_N^2} \begin{pmatrix} x \\ y \end{pmatrix} - \sum_{i=1}^{3} -\frac{GM_i}{{\mathbf{r}_i}^3} \begin{pmatrix} x - x_i \\ y - y_i \end{pmatrix} \tag{4}$$

A system of two second-order ODEs needs two boundary conditions to be completely defined. These boundary conditions are the spatial starting point $\mathbf{R}_0(t_N = 0) = (-1, -1)$ and the endpoint $\mathbf{R}_N(t = 1) = (1, 1)$ of the spacecraft.

The coordinates of the astronomic objects in the (x, y) plane are stated in tab. 1.

Table 1: (x, y) positions and gravitational masses GM_i of the astronomic objects

i	(x, y) / m	GM_i / m^3/s^2
1	$(-0.5, -1.0)$	0.5
2	$(-0.2, +0.4)$	1.0
3	$(+0.8, +0.3)$	0.5

2.3 Transformation regarding initial and final kinematic state

Rewrite problem as follows:

$$\mathbf{r}(t) = \mathbf{R}(t) + \psi(t)\mathcal{N}_\theta(t) + \phi(t)\Big(\mathbf{V}(t) - \dot{\mathbf{R}}(t)\Big), \tag{5}$$

where $\mathbf{r}(t), \mathbf{R}(t), \mathbf{V}(t), \mathcal{N}_\theta(t) : \mathbb{R} \to \mathbb{R}^d : t \mapsto \Xi$ and $\phi(t), \psi(t) : \mathbb{R} \to \mathbb{R}$ with spatial dimension d. Then it follows that

$$\dot{\mathbf{r}}(t) = \dot{\mathbf{R}}(t) + \dot{\psi}(t)\mathcal{N}_\theta(t) + \psi(t)\dot{\mathcal{N}}_\theta(t) + \phi(t)\Big(\dot{\mathbf{V}}(t) - \ddot{\mathbf{R}}(t)\Big) + \dot{\phi}(t)\Big(\mathbf{V}(t) - \dot{\mathbf{R}}(t)\Big) \tag{6}$$

Add footnote that derivation in 3d is trivial or change to 3D

Agree. I may consider to extend notation to 3D later as its only minor notation addition

Need to resolve inconsistency with t, tN : $\tau = t/tN$ - maybe. but all the code is consistent with t being normalized time

Agree. Let's do the polish later and get the big picture ready first

In particular we may choose

$$\mathbf{R}(t) := \frac{t}{t_N}\big(\mathbf{x}_{t_N} - \mathbf{x}_{t_0}\big) + \mathbf{x}_{t_0} \ , \tag{7}$$

$$\mathbf{V}(t) := \frac{t}{t_N}\big(\mathbf{v}_{t_N} - \mathbf{v}_{t_0}\big) + \mathbf{v}_{t_0} \ , \tag{8}$$

where the initial and final state conditions are $\mathbf{x}_{t_N} = \mathbf{x}(t_N)$, $\mathbf{x}_{t_0} = \mathbf{x}(0)$, $\mathbf{v}_{t_N} = \mathbf{v}(t_N)$, $\mathbf{v}_{t_0} = \mathbf{v}(0)$, where the elements of the vectors are the spatial directions $\mathbf{x}_{t_0} = (x_{1,0}, x_{2,0}, x_{3,0})^T$, etc.. Then, to satisfy the kinematic conditions $\forall\theta$, we need to construct $\psi(t)$ s.t. $\psi(0) = 0$, $\psi(t_N) = 0$, $\dot{\psi}(0) = 0$, $\dot{\psi}(t_N) = 0$, and $\psi(t) > 0 \ \forall 0 < t < t_N$. The function presumably need not be convex. Two solutions are:

$$\psi(t) = 1 - \cos\Big(\frac{2\pi t}{t_N}\Big) \ , \tag{9}$$

$$\psi(t) = t^2(t_N - t)^2 \ . \tag{10}$$

Further, we need to construct $\phi(t)$ s.t. $\phi(0) = 0$, $\phi(t_N) = 0$, $\dot{\phi}(0) = 1$, $\dot{\phi}(t_N) = 1$, and $\phi(t) > 0 \ \forall 0 < t < t_N$. Two solutions are:

$$\phi(t) = \sin(\cdots) \tag{11}$$

$$\phi(t) = 2\frac{t^3}{t_N^2} - 3\frac{t^2}{t_N} + t \ . \tag{12}$$

Remark 0: Note that the polynomial suggestion $\phi(t) \not> 0 \ \forall 0 < t < t_N$ and $\phi(t) \not< 0 \ \forall 0 < t < t_N$. It is unclear at the moment whether this can lead to trouble, and the search for a better function should continue.

Remark 1: It may be easier construct another function $\chi(t)$ for a product $\chi(t)\mathbf{R}(t)$ that fulfills positivity inside the domain to elimante $\dot{\mathbf{R}}(t)$ at $t \in \{0, t_N\}$ and substitute $\Big(\mathbf{V}(t) - \mathbf{R}(t)\Big) \rightarrow \mathbf{R}(t)$. The behaviour of $\chi(t)$ must be the "reverse"of $\phi(t)$.

Remark 2: $\phi(t), \psi(t), \chi(t)$ can contain learnable parameters to learn the boundary conditions.

Remark 3: This shows the principle of generalization from kinematic to dynamic states, i.e. $\ddot{\mathbf{r}}$. When the second-order correction functions are parametrized with thrust time window and force magnitude, one can also learn maneuvers with impulses.

Remark 4: In this construction, the learned kinematic states are always consistent, i.e. $\mathbf{v}(t) \equiv \dot{\mathbf{r}}(t)$. This is not strictly fulfilled in the vanilla-PINN, nor in the case when one constructs two independent NNs to learn position and velocity, respectively, nor in the case of a single NN with multiple outputs learning position and velocity jointly.

2.4 Implementation

We need to construct a neural network $\mathcal{N}_\theta(t) : \mathbb{R} \rightarrow \mathbb{R}^d : t \mapsto (\tilde{x}, \tilde{y}, \tilde{z})^T$ and transform the neural network to a new function $\mathbf{r}(t)$ according to (5). This transformed NN fulfills our kinematic state conditions for all NN parametrizations and can be auto-differentiated with respect to both t, as would only necessary to train on velocity data, and with respect to θ, which is necessary to minimize thrust. Our optimization objective thrust does not contain the velocity, hence $\dot{\mathbf{r}}$ and auto-differentiation with respect to t are not necessary in the implementation. Above considerations are to proof the reparametrization and demonstrate the procedure for generalization.

2.5 Alternative transformation with χ

Rewrite problem as follows:

$$\mathbf{r}(t) = \chi(t)\mathbf{R}(t) + \psi(t)\mathcal{N}_\theta(t) + \phi(t)\mathbf{V}(t) \ , \tag{13}$$

where $\mathbf{r}(t), \mathbf{R}(t), \mathbf{V}(t), \mathcal{N}_\theta(t) : \mathbb{R} \rightarrow \mathbb{R}^d : t \mapsto \Xi$ and $\chi(t), \phi(t), \psi(t) : \mathbb{R} \rightarrow \mathbb{R}$ with spatial dimension d. Then it follows that

$$\dot{\mathbf{r}}(t) = \dot{\chi}(t)\mathbf{R}(t) + \chi(t)\dot{\mathbf{R}}(t) + \dot{\psi}(t)\mathcal{N}_\theta(t) + \psi(t)\dot{\mathcal{N}}_\theta(t) + \phi(t)\dot{\mathbf{V}}(t) + \dot{\phi}(t)\mathbf{V}(t) \tag{14}$$

In particular we may choose again

$$\mathbf{R}(t) := \frac{t}{t_N}\left(\mathbf{x}_{t_N} - \mathbf{x}_{t_0}\right) + \mathbf{x}_{t_0} \ , \tag{15}$$

$$\mathbf{V}(t) := \frac{t}{t_N}\left(\mathbf{v}_{t_N} - \mathbf{v}_{t_0}\right) + \mathbf{v}_{t_0} \ , \tag{16}$$

where the initial and final state conditions are as before. Note that the requirements for ψ, ϕ have changed slightly under this new formulation. We need $\dot{\chi}(0) = 0$, $\dot{\chi}(t_N) = 0$ while $\chi(0) = \chi(t_N) = 1$. This construction always leads to a contradiction no matter ϕ, ψ except for vanishing $\dot{\mathbf{R}}$ or specifically designed compensation terms. It may be ameliorated by choosing $\mathbf{R}(t)$ and $\mathbf{V}(t)$ more wisely. All that however poses no obvious advantage over the original method at at least the same complexity.

3 Experiments

3.1 Position-state transformed PINN

Architecture & Hyperparameters: For both networks, 3 layers with 50 densely connected neurons are used. The input is the normalized time and the output a position in 2d-space. The ADAM and LBFGS optimizers are used consecutively. For the vanilla PINN 7000 steps are made in total, where the first 3000 were made with ADAM. The position state-transformed experiment was trained with 200 iterations of ADAM and LBFGS afterwards.

Maybe introduce table format for all experiments - or one table for all experiments

To carry out an experiment with exact initial and final positions, the PINNs output must be transformed. Therefore eq. (5) is used with the functions $\mathbf{R}(t) = (2t-1, 2t-1)^T$, $\psi(t) = t(1-t)$ and $\phi(t) = 0$. With t being normalized time, one can easily obtain that the initial and final position of the trajectory must be $\mathbf{r}_0 = (-1, -1)$ and $\mathbf{r}_N = (1, 1)$, respectively. This problem was originally posed by Seo [4], but he solved it without the exact boundary conditions.

3.1.1 Vanilla PINNs vs. position state-transformed PINNs

To solve the Swingby-Trajectory problem using a vanilla PINN, it's necessary to introduce separate loss terms: one to enforce the boundary conditions L_{BC} and another to minimize the thrust by bringing in the physical laws L_P. These loss terms are weighted by factors of ω_{BC} and $\omega_P = 1.0$, inside the loss function. For the sake of comparability, we leave the physics loss term weighted by unity in all further experiments. The boundary loss weight ω_{BC} needs some tuning, in order to predict the best possible outcome. This is achieved by performing a one-dimensional brute force grid search. We start by training 1000 models with logarithmically spaced ω_{BC} values in the range of $[10^{-3}, 10^3]$. The networks are trained for 2000 and 5000 iterations using the ADAM and LBFGS optimizers, consecutively. For every weight parameter, the minimal boundary and total loss values are stored and further evaluated. It should be noted that only the local minima are valid solutions because the global ones (for $\omega_{BC} = 10^{-3}$) fail to comply the boundary conditions heavily. The result is shown in Figure 1.

SR: best practise would be a compromise. Denote major architectural choices such as dpeth, width, optimizer, learning rate here. Provide a table with details such as schedule/curriculum hyperparameters in appendix. Optionally, we can also just publish the code on GitHub.

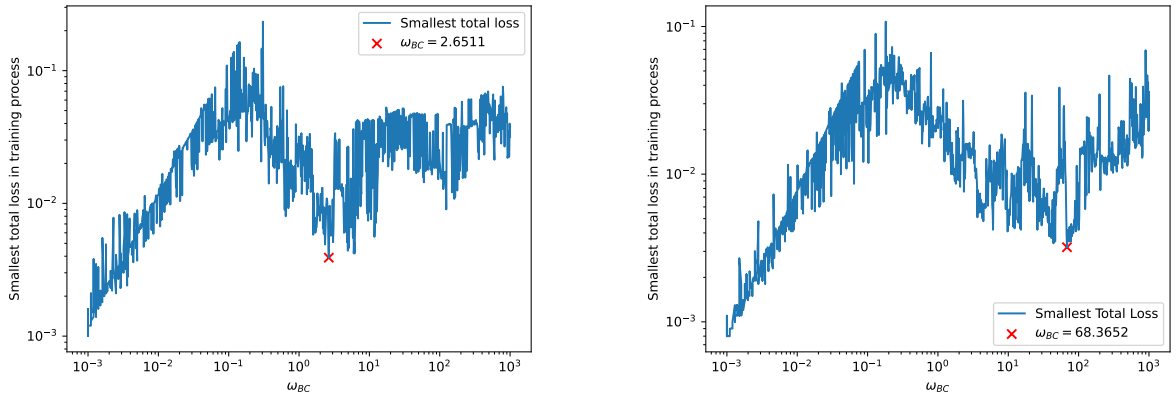
By using the position-state-transformed PINN the boundary loss term is made redundant and only the physics loss term remains. So the aim of this work is to showcase a comparison between eliminating the boundary loss term, and leaving it with a properly tuned weight parameter.

The optimized Vanilla, as well as the Position state transformed Swingby-Trajectories can be seen in fig. 2a. The initial and end positions are marked with a red dot and cross respectively. The vanilla trajectory is plotted in the lighter blue and the acting gravities on the trajectories are indicated by a 20-times smaller arrow of the same color. The thrust magnitude would be indicated by a black arrow but is vanishingly small. The astronomical objects are plotted in green. The vanilla PINN doesn't fulfill the boundary conditions as well as the exactly implemented version, but besides that, the trajectory is qualitatively similar.

Fig. 2c shows the acting gravity (dashed) and the force (solid) required to fly along the trajectory with respect to normalized time for the vanilla and position transformed PINN. As these two pairs of lines are almost fully overlapping, the thrust magnitude which is shown in Fig. 2d is small compared to the required force magnitude.

Furthermore, Fig. 2b shows the total loss function as well as its individual terms as a function of training epochs for the vanilla and position transformed PINN. It takes around 6000 iterations of training for the vanilla PINN to converge. In this case, a slightly longer training with LBFGS leads to overfitting, causing the total loss to explode. The position state-transformed PINN on the other hand, converges already after around 1500 training epochs. The loss term is given by the mean-square-error of the thrust differential equation in residual form evaluated on 100 linearly spaced collocation points $t_{colloc} \in [0, 1]$.

Introduce MSE / thrust properly



(a) Smallest total loss as a function of boundary loss weight in 2D

(b) Smallest total loss as a function of boundary loss weight in 3D

Figure 1: Boundary loss weight ω_{BC} grid search for the 2D (left) and 3D (right) vanilla PINN. The blue curve shows the smallest total loss value of a training process with N_{ADAM} = 2000 and N_{LBFGS} = 5000 iterations, for a given ω_{BC} and fixed total time of flight $t_{total} = 1$. The networks were trained 1000 times, respectively.

3.1.2 Adding time of flight to PINN parameters

The total time of flight (TOF) is decisive for its efficiency. Since it is not possible to know in advance how long the optimal trajectory will take, we add this to the trainable parameters of the PINN where the initial total TOF is set to $t_{total} = 1$. The results are pictured in Fig. 3. The improvement of the result becomes immediately evident when inspecting the time evolution of thrust in Fig. 3d of the experiment. The thrust magnitude of the position transformed PINN is at max half the size of the thrust in the experiment with static total time 2d. The vanilla PINNs thrust in the static experiment shows two maxima where in the dynamic experiment the thrust is concentrated to one peak in normalized time. Comparing thrust of the vanilla and position transformed PINN in Fig. 3d indicates the superiority of our novel transformation approach, as the thrust is significantly smaller.

3.1.3 Four-body problem on a plane in three-dimensions

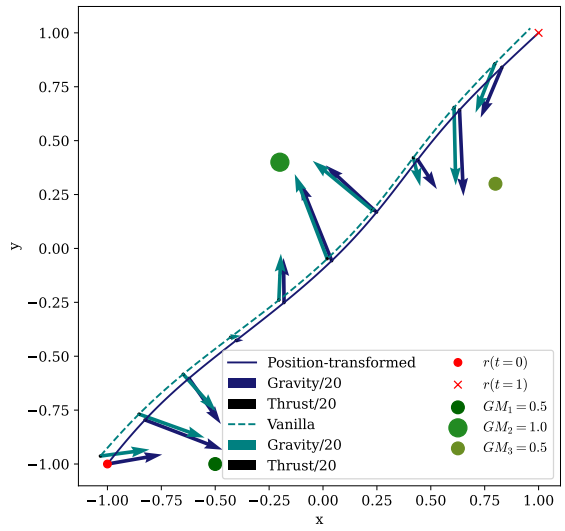
In order to test the generalizability of our position state-transformed PINN we look at a "Pseudo-3d" problem, where we add the z direction as a degree of freedom to our already discussed four-body problem in the (x, y) plane. Hence, our position state-transformed PINN is now a function from normalized time into 3d space $t \mapsto (x, y, z)$. Our boundary conditions are represented by $\mathbf{R}(t) = (2t-1, 2t-1, 0)^T$ now. The two functions $\psi(t) = t(1-t)$ and $\phi(t) = 0$ remain unchanged. The PINNs output is again transformed by eq. (5). And once again, we compare our results with the vanilla PINN.

The results are pictured in Fig. 4. In Fig. 4a and 4b we see the Swingby trajectories of our vanilla and position-transformed PINN in 3D perspective as well as projected into 2D. The benefit of exact imposed boundary conditions gets immediately clear when one looks at the 3D plot. The vanilla PINN doesn't fulfill the boundary conditions in the z-direction. With that comes a deviation of the trajectory in the third dimension. The needed thrust for the trajectories in Fig. 4e is equally good, but the training for the vanilla PINN takes twice as long as for the position state-transformed PINN.

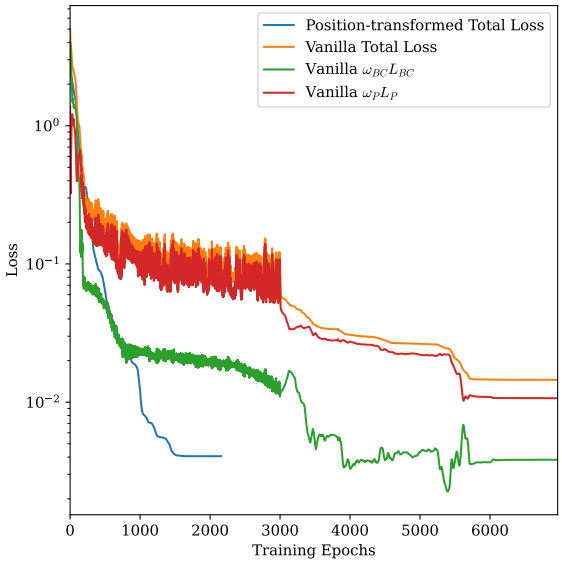
Architecture & Hyperparameters: 3 layers, 50 densely connected neurons. 200 ADAM, rest LBFGS. $N_{colloc} = 100$.

3.1.4 Four-body problem in three-dimensions with shifted boundary conditions

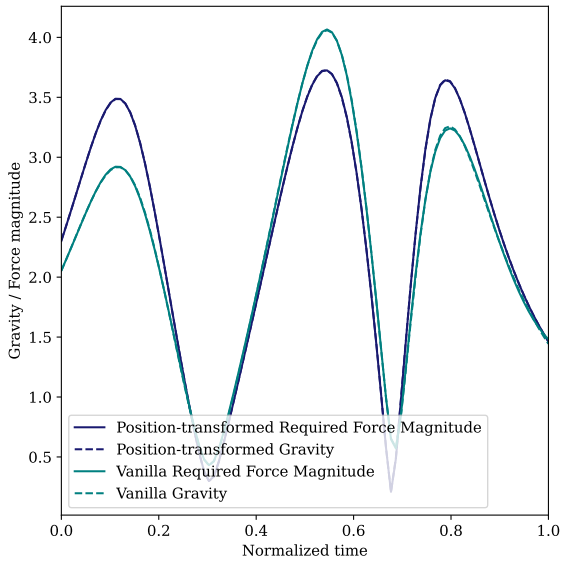
To test the stability of the method, we redo the four-body problem in three dimensions, but with shifted boundary conditions. Shifted such that our BC are represented by $\mathbf{R}'(t) = (2t-1, 2t-1, A(1-2t))$. This corresponds to moving $\mathbf{r}_0$ and $\mathbf{r}_N$ a distance of $\pm A$ into the $z-$direction. The experiment is conducted using $A \in \{0, 0.1, ..., 0.5\}$ and the results are shown in Fig. XX.



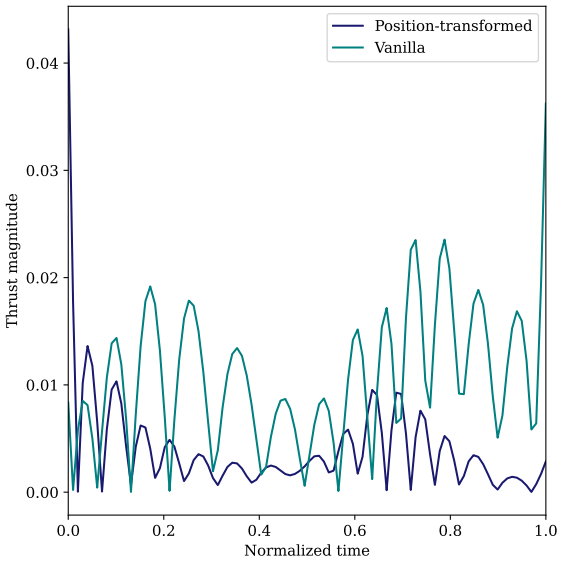
(a) Swingby Trajectories of the position state transformed and vanilla PINN



(b) Loss over training epochs of the position state transformed and vanilla PINN

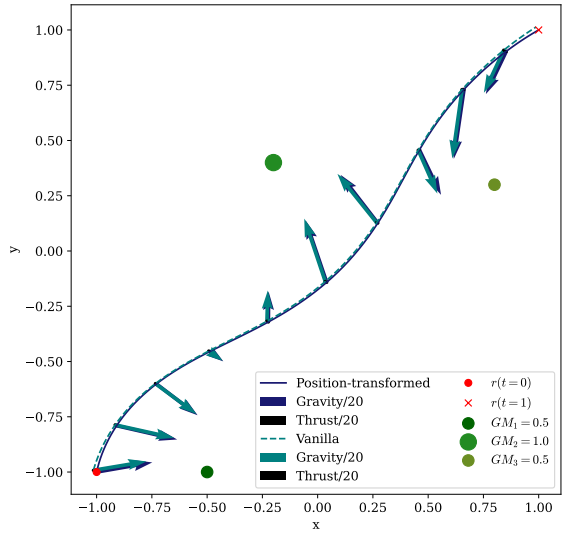


(c) Total gravity and required force magnitude of the position state transformed and vanilla PINN

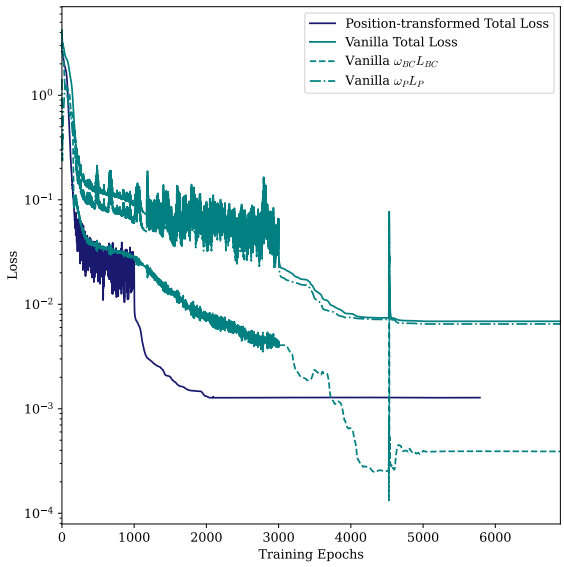


(d) Thrust magnitude of the position state transformed and vanilla PINN

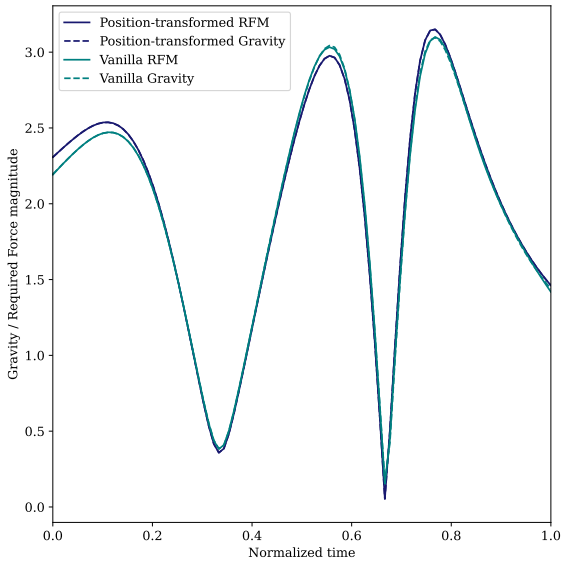
Figure 2: Comparison of vanilla PINN vs. position-transformed PINN



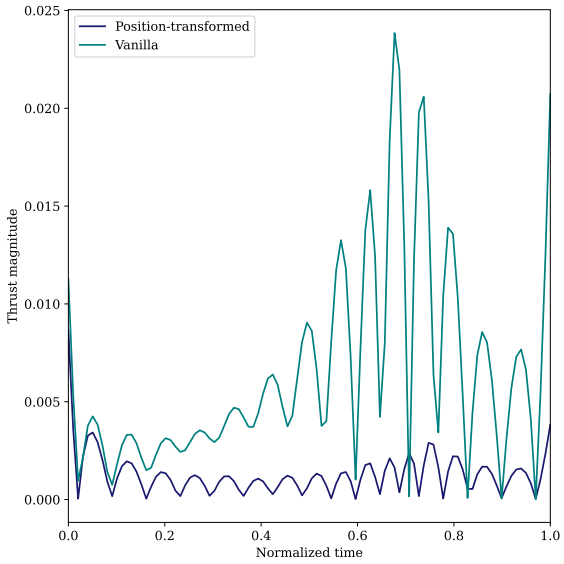
(a) Swingby Trajectories of the position state transformed and vanilla PINN



(b) Loss over training epochs of the position state transformed and vanilla PINN

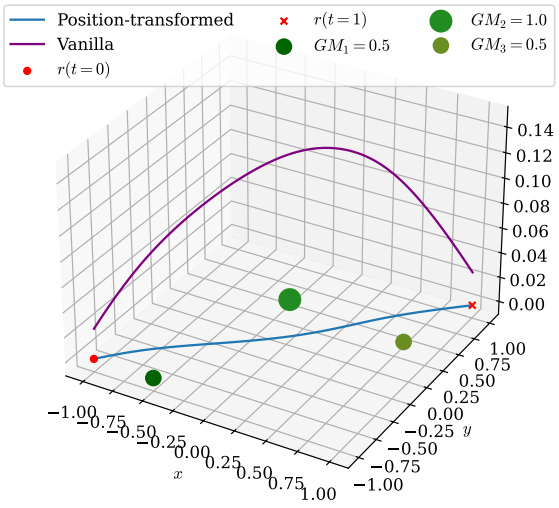


(c) Total gravity and required force magnitude of the position state transformed and vanilla PINN

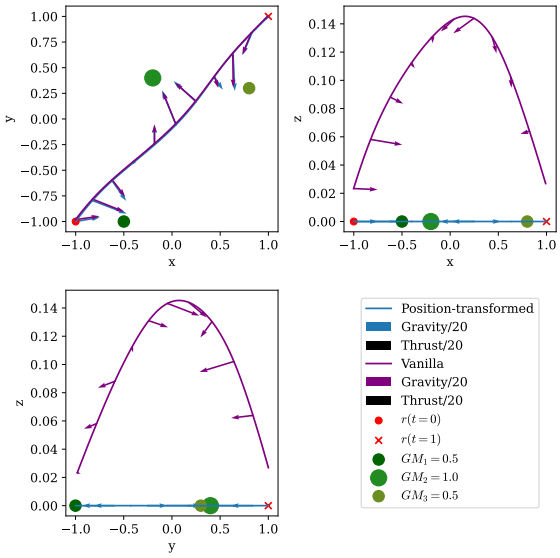


(d) Thrust magnitude of the position state transformed and vanilla PINN

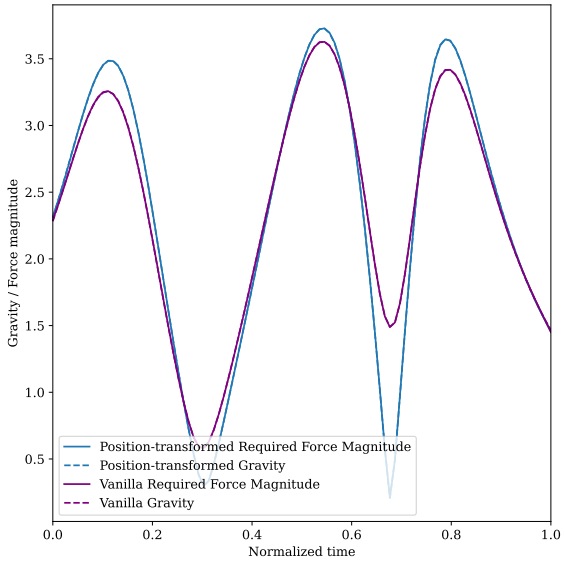
Figure 3: Comparison of vanilla PINN vs. position-transformed PINN with time of flight as trainable parameter



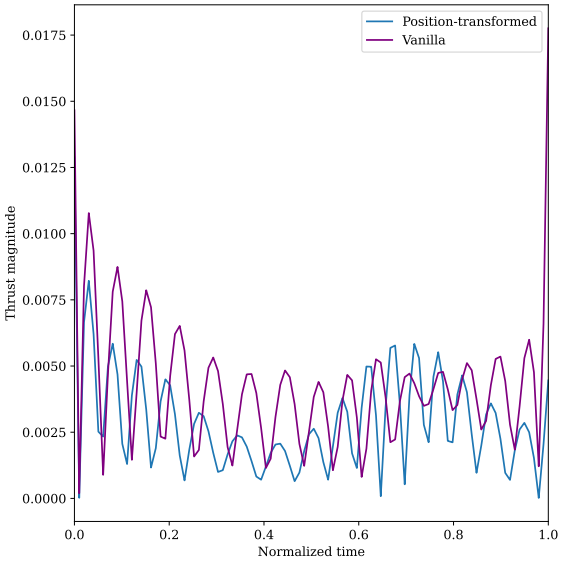
(a) Swingby Trajectories of the position state transformed and vanilla PINN on a plane in 3d



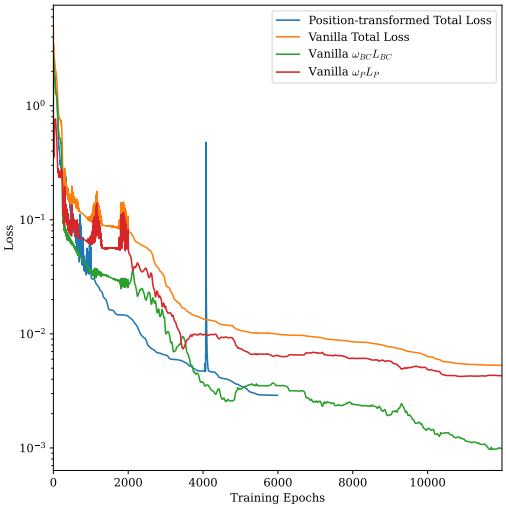
(b) Swingby Trajectories of the position state transformed and vanilla PINN on a plane in 3d projected into 2d



(c) Total gravity and required force magnitude of the position state transformed and vanilla PINN on a plane in 3d



(d) Thrust magnitude of the position state transformed and vanilla PINN on a plane in 3d



(e) Loss of the position state transformed and vanilla PINN on a plane in 3d

Figure 4: Comparison of vanilla PINN vs. position-transformed PINN on a plane in three dimensions

3.2 Extension to kinematic state-transformed PINN

Most real-world problems require boundary conditions in phase space. Therefore, we also study such problems with our transformation approach. What changes is that now a function $\phi \neq 0$ is chosen in our transformation from eq. (5). Additionally, the initial and final velocities are declared by $\mathbf{V}(t)$ in eq. (16). The construction is analogous to the initial and final position $\mathbf{R}(t)$.

- Following notes are for fixed t_total = 1
- Choosing intial/final velocity from converged Kirche-ums-Kreuz trajectory doesn't converge to a swingby trajectory. In the start and end there is always around 10-40 thrust needed to correct from boundary velocities to trajectory –> Can't reproduce results from old code, Still don't know why. Training too long yields loss explosion
- Another try was to qualitatively guess v_0 pointing to heavy mass and +1 into z direction, v_N same but -1 into z. Also z directions -1,+1 –> Works better
- Fact: Kinematic approach doesn't work for random guessed boundary velocities
- Found out that old code didn't yield correct boundary velocities. Direction is correct but magnitude differs. Implementation is identical
- Next idea: Learn x-y components of initial/final velocities for given z component.

3.2.1 Vanilla PINNs vs. kinematic state-transformed PINNs

Didn't do Vanilla 3d

3.2.2 Four-body problem on a plane in three dimensions

The experiments with the position state transformed PINN for a plane in 3d show that it is hard to find a trajectory that lays in the (x, y) plane that minimizes thrust. At least by using exact boundary conditions in only position space instead of phase space. The idea now is to define the approximate direction of the desired trajectory in terms of initial and final velocities to prevent immediate drift in the z-direction, in the vicinity of our start and end point. Therefore we choose $\mathbf{v}_{t_{0,N}} = (1, 1, 0)$. Our first transformation function remains $\psi = t^2(1-t)^2$ and we define $\phi = 2t^3 - 3t^2$ from the suggestion above.

Fig. 5a and 5b show that the idea of guiding the trajectory on the boundaries in the desired direction does indeed work. The solution is qualitatively the same as in our 2d problem. The only difference is that we can observe a small thrust arrow pictured orange in Fig. 5a. The thrust at the beginning and end of the trajectory is needed in order to make up for our initially guessed velocity. This behavior can be seen in 5c very well. Approximately 20-25% of the normalized time is used to accelerate from the trial-velocities start and end to the trajectory that is really swingby and uses no thrust. Besides these correction times, the required thrust magnitude is vanishingly small.

Although the total loss is orders of magnitude larger compared to our other experiments, the network converges noticeably faster. In Fig. 5d it seems like convergence takes place at around 350 training epochs already. Furthermore, the first 100 iterations are optimized by using ADAM. This, however, doesn't have a big impact on the network's prediction. Afterwards, LBFGS is used.

Fig. 10a and 10b show a qualitatively different trajectory compared to the 2d case. The found swingby solution leaves the (x, y)-plane immediately after the start and only comes back at the desired final destination. However, the vanishing thrust magnitude in Fig. 10c confirms this trajectory to be a valid solution. This behavior suggests that the solution of our four body problem in two dimensions is very unstable. The training process is finished at around 1400 iterations, as pictured in Fig. 10d, which takes approximately as long as in 2d (**??**).

Idea: Do experiment where v0=(0,0,1) and vN=(0,0,-1) and unchanged **R** in order to show that we can force the trajectory into different Loss basins using our method. (Only releveant if we can reproduce Kirche ums Kreuz trajectory probably)

3.2.3 Four-body problem in three-dimensions

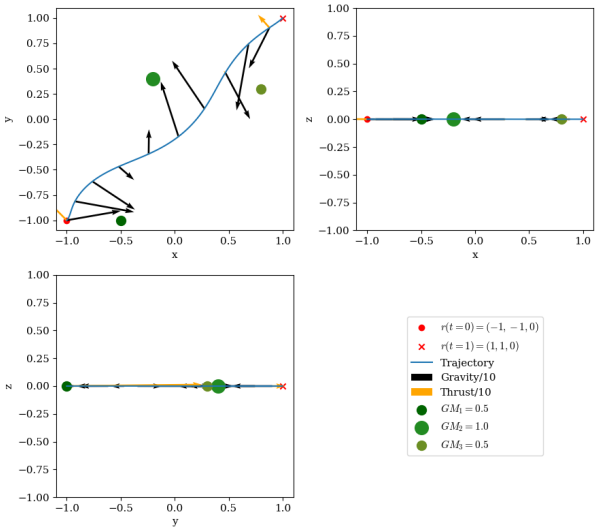
Didn't do such an experiment

ah yes, that could actually be very beneficial to show another example how the transformed PINN can solve problems where the vanilla PINNs get stuck. But let's first

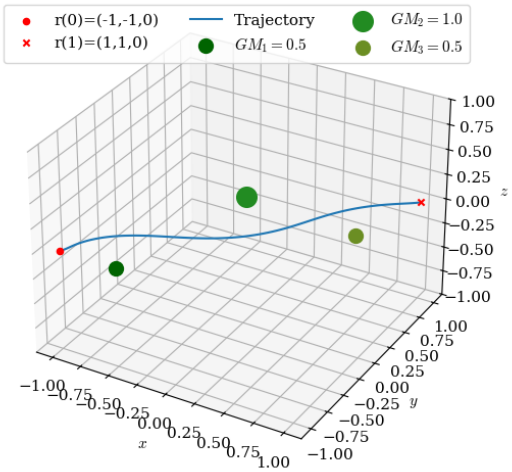
3.3 Hohnmann-Transfer

4 Conclusion

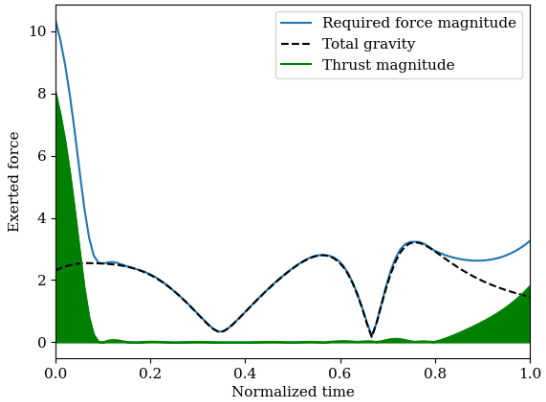
Your conclusion here



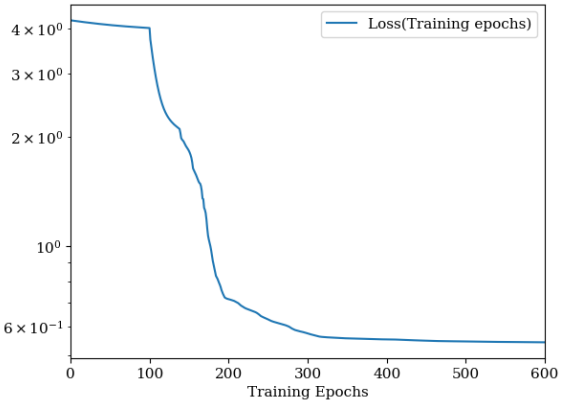
(a)



(b)



(c) Force magnitude



(d) Loss as a function of training epochs

Figure 5: Phase-space PINN in 3D

Acknowledgments

This research was funded in whole or in part by the Austrian Science Fund (FWF), Grant DOI: 10.55776/J4774.

References

[1] Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[2] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[3] David Dalton, Alan Lazarus, Hao Gao, and Dirk Husmeier. Boundary constrained gaussian processes for robust physics-informed machine learning of linear partial differential equations. *Journal of Machine Learning Research*, 25(272):1–61, 2024.

[4] Jaemin Seo. Solving real-world optimization tasks using physics-informed neural computing. *Scientific Reports*, 14(1):202, 2024.

[5] N. Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

[6] M.R. Jones, T.J. Rogers, and E.J. Cross. Constraining gaussian processes for physics-informed acoustic emission mapping. *Mechanical Systems and Signal Processing*, 188:109984, 2023.

[7] Hailong Sheng and Chao Yang. Pfnn: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries. *Journal of Computational Physics*, 428:110085, 2021.

[8] Songming Liu, Hao Zhongkai, Chengyang Ying, Hang Su, Jun Zhu, and Ze Cheng. A unified hard-constraint framework for solving geometrically complex pdes. *Advances in Neural Information Processing Systems*, 35:20287–20299, 2022.

[9] Sascha Ranftl. A connection between probability, physics and neural networks. In *Physical Sciences Forum*, volume 5, page 11. MDPI, 2022.

[10] KA Luong, MA Wahab, and JH Lee. Simultaneous imposition of initial and boundary conditions via decoupled physics-informed neural networks for solving initial-boundary value problems. *Applied Mathematics and Mechanics*, 46(4):763–780, 2025.

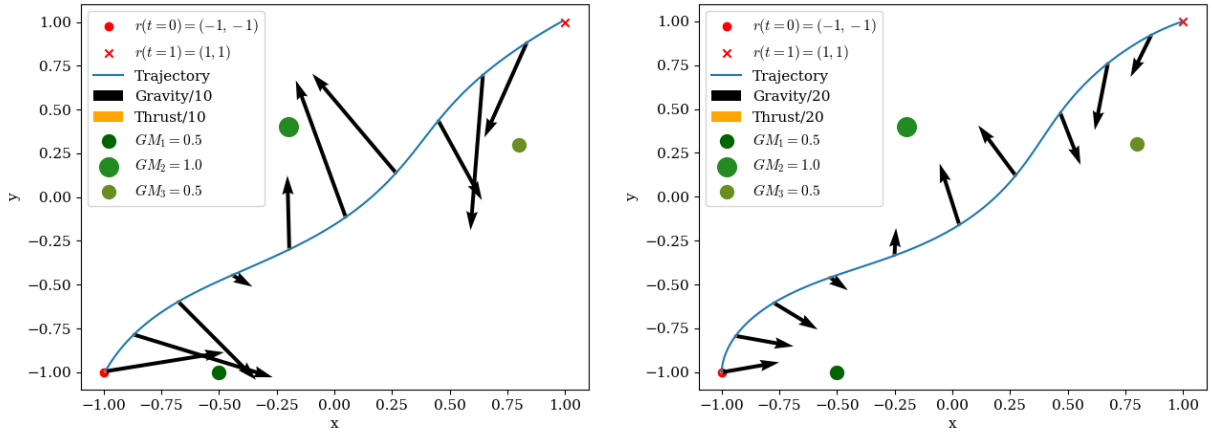
[11] Christopher Straub, Philipp Brendel, Vlad Medvedev, and Andreas Rosskopf. Hard-constraining neumann boundary conditions in physics-informed neural networks via fourier feature embeddings, 2025.

[12] Markus Lange-Hegermann. Algorithmic linearly constrained gaussian processes. In *NeurIPS*, 2018.

[13] Marc Härkönen, Markus Lange-Hegermann, and Bogdan Raiţă. Gaussian process priors for systems of linear partial differential equations with constant coefficients. In *ICML*, 2023.

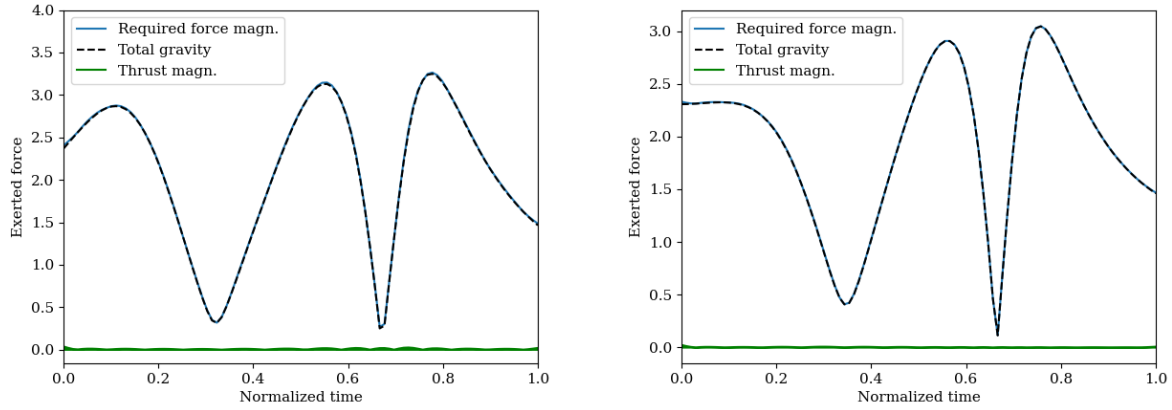
5 Friedhof

5.0.1 Vanilla vs Positon-transformed OLD



(a) Swingby-Trajectory of the vanilla PINN (b) Swingby-Trajectory of the position state transformed PINN

Figure 6: Vanilla PINNs (left) vs. Spacy PINNs (right): Trajectory



(a) Force magnitudes vanilla PINN (b) Force magnitudes of the position state transformed PINN

Figure 7: Vanilla PINNs (left) vs. Spacy PINNs (right): Force magnitudes

5.0.2 Four-body problem in three dimensions

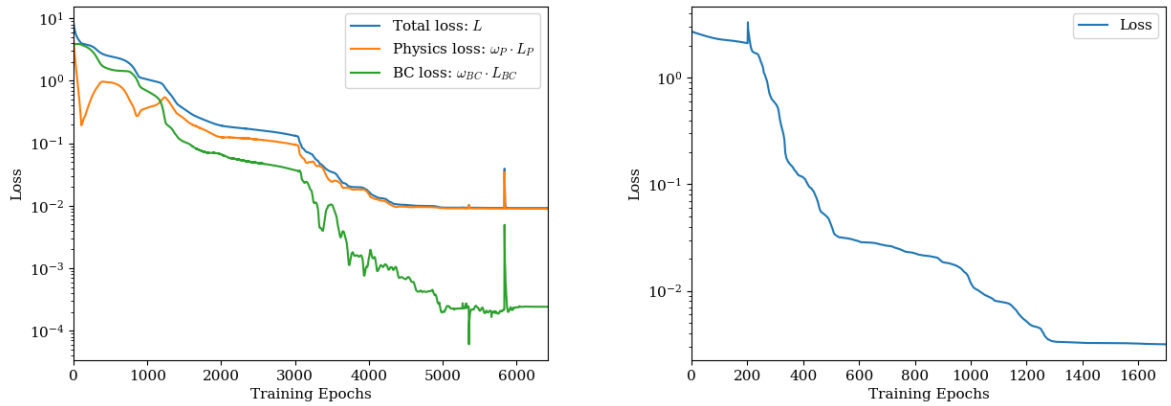
The solution of our four-body problem on a plane in three dimensions doesn't give a lot of insight. Thus, we further investigate this problem by introducing three kinds of variations, which we shall discuss now. The goal of these variations is to find out if and when our model starts to predict a trajectory that is qualitatively similar to the 2d case.

The inclined planet plane: In the first variation, we take the plane where the planets lay in and tilt it by an angle ϕ. The rest of the experiment stays the same. The axis of rotation $a = \frac{1}{\sqrt{2}}(1,-1,0)^T$ goes through the origin and is perpendicular to the vector that connects the start and endpoint. It was started with a rotation by $\phi = 1\,°$. If the trajectory didn't change qualitatively, the experiment was repeated with a larger angle ϕ. The smallest angle that coincides with a trajectory that is not making a large circle around the astronomical objects is $\phi = 23\,°$. The trajectory is shown in Fig. 9a and 9b respectively. However, Fig. 9b clearly shows that this setup has little to do with our original problem in 2d.

I used the nomenclature from Github. To be changed afterwards

SR: We will find scientifically appealing semantics eventually

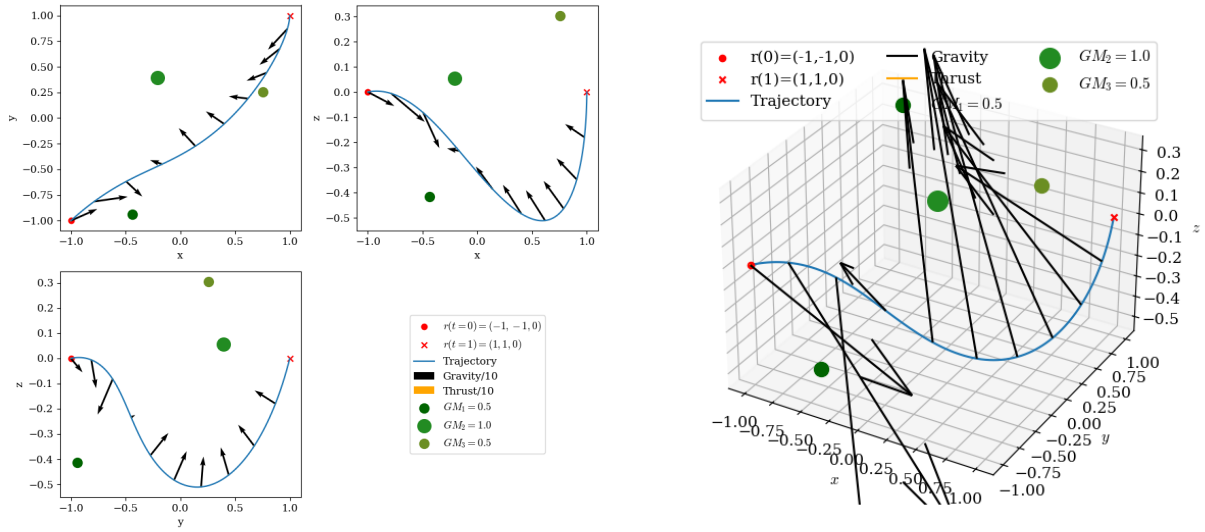
ideally,



(a) Total loss, weighted physics and BC loss terms as a function of training epochs of the vanilla PINN

(b) Loss as a function of training epochs of the position state transformed PINN

Figure 8: Vanilla PINNs (left) vs. Spacy PINNs (right): Loss function



(a) Trajectory of the inclined planet-plane experiment with angle $\phi = 23\,^\circ$

(b) 3d Trajectory of the inclined planet-plane experiment with angle $\phi = 23\,^\circ$

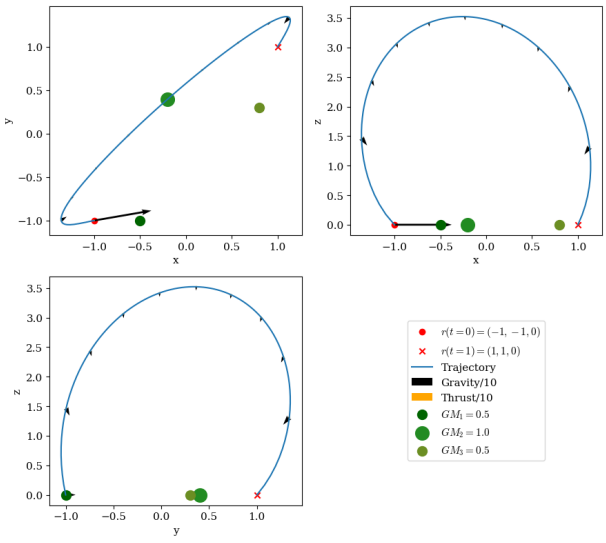
Figure 9: 3D Problem

Planets moved along z In some other experiments, we played with the z-positions of our astronomical objects. We either move all three gravitational masses GM_i or only the heaviest one (GM_2). When moving all of them, we move $GM_{1,3}$ the same distance up and GM_2 down or vice versa. The outcome of all these experiments is the same as for the inclined planet-plane. Namely, the trajectories only change when the configuration is much different from our initial configuration. The planets must be shifted so far (0.6-0.7 when moving all planets and even further when moving only one) that a comparison to the 2d case is rather nonsensical.

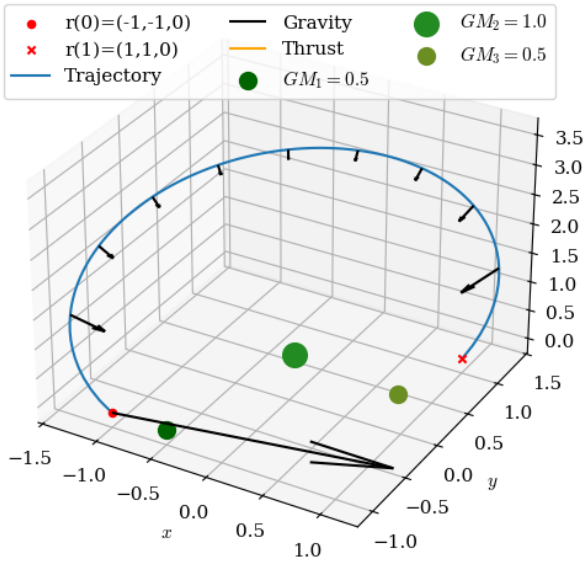
Shifted initial end conditions For the sake of easier comparison, we also make a variation by shifting the initial and end conditions by a factor A instead of the planets. The planets stay in the (x, y)-plane just like in the 2d problem setup. In that case, our boundary conditions are represented by $\mathbf{R}(t) = (2t-1, 2t-1, (2t-1)A)^T$. Once again the experiment is started with a small shift of $A = 0.1$ and repeated with a larger offset if nothing qualitatively changed. A

Decide/Select if/which plot we want to show here

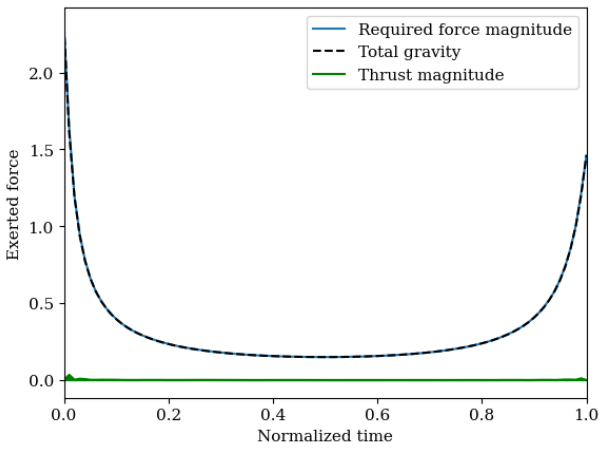
these figures are not in the manuscript yet right lets keep it / include it for now and



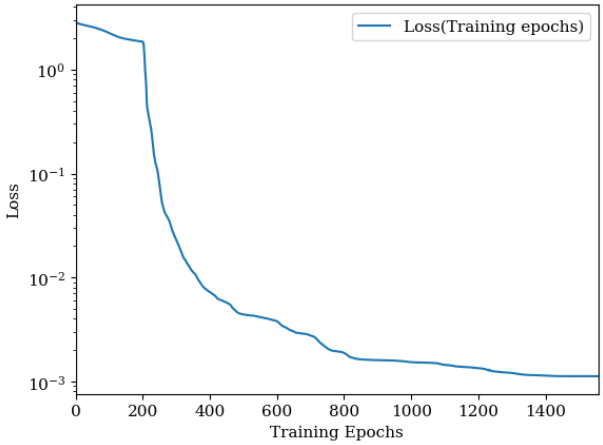
(a) Trajectories of the four-body problem in three dimensions



(b) 3d-trajectory of the four-body problem in three dimensions



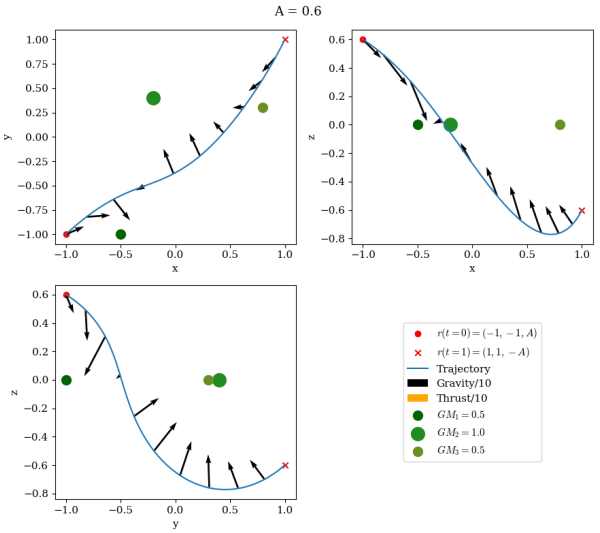
(c) Force magnitude of the four-body problem in three dimensions



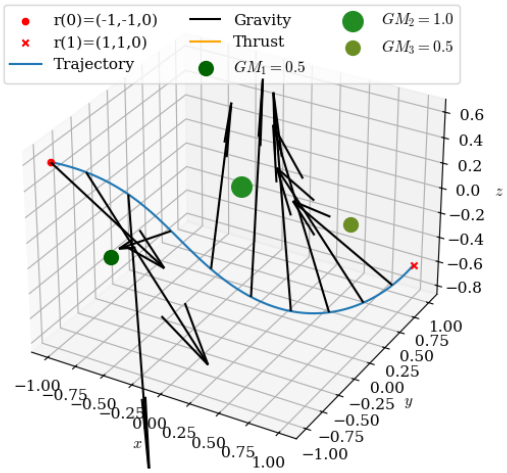
(d) Loss as a function of training epochs of the four-body problem in three dimensions

Figure 10: 3D-problem on a plane

trajectory that goes rather directly from the start to end point only emerges for a large shift of $A > 0.6$ The resulting trajectory is shown in Fig. 11a and 11b.



(a) Trajectories of the four-body problem in three dimensions with shifted BC



(b) 3d-trajectory of the four-body problem in three dimensions with shifted BC

Figure 11: 3D problem sensitivity analysis with respect to plane inclination angle

All the models above are trained using only the LBFGS optimizer. The variations from above all aim to describe small deviations from a 2d problem. It turns out the deviations must be rather large in order to reproduce a result that goes directly from the start to the end position. To summarize, all three variations give the same result, namely: the solution of the 2d problem (6b) is very unstable, compared to the orbit-like behavior in 3d (10c).