Documentation Notes for 'bergman_epinn.ipynb'

Written by Hai Siong Tan
✉haisiong.tan@gryphonai.com.sg
November 2025

This document provides more detailed explanations for the main notebook **'bergman_epinn.ipynb'** in this repository. This project implements an application of *Evidential Physics-Informed Neural Networks* (E-PINN) as presented in [1] to *Bergman's model* [2, 3] defined by the following coupled differential equations.

$$\frac{dG}{dt} = -p_1(G - G_b) - X(t)G(t), \tag{1}$$

$$\frac{dX}{dt} = -p_2X(t) + p_3\left(I(t) - I_b\right), \tag{2}$$

where G, X represent glucose level and insulin intake respectively, and I represents insulin level, with G_b, I_b being constant base values and p_1, p_2, p_3 are the unknown rate constants. In our model, G, X are taken to be the main model target outputs with $I(t)$ being known. We used the data presented in the paper by Kartono et al. in [4], specifically Figure 1 where glucose and insulin values were read off from the graphs using *WebPlotDigitizer*. In applying E-PINN to this dataset, the model predicts the following

1. the glucose vs time curve together with an uncertainty band defined by a t-distribution as formulated in Evidential Deep Learning (EDL) [5, 6]
2. a posterior distribution for the parameters p_1, p_2, p_3 that best describes the dataset

The base model was taken to be a typical multi-layer-perceptron model with time as the input and five targets representing G and its associated uncertainty (defined by 3 other outputs) and X. Since there is no directly observed data for X, we treat X as a standard output without its uncertainty learned through the formalism of EDL. This means that the model's output description for X consists solely of X itself, and there are no uncertainty targets associated with it, unlike G.

In the following, we provide some explanatory notes for the main sections of the notebook. Each section is intended to be read alongside the corresponding one in the notebook **'bergman_epinn.ipynb'**.

1 Data-Fitting phase

This phase involves model training without alluding to Bergman's model at all, i.e. the ODEs are not taken into account in the loss function. We took the perceptron model to have two hidden layers each equipped with 32 neurons (**'nn_I = bergman_helpers.eFCN(1,32,2).to(device)'**). Increasing this level of complexity by a large extent will lead to overfitting, e.g. an oscillatory model curve that deviates significantly from the form of the numerical solutions of Bergman's model. In general, the complexity of the model should be adopted such that the geometrical features (e.g. extrema points, asymptotes, etc.) of the predicted curve do not deviate too much from the numerical solutions of the ODEs that we are asserting as loss terms later on. We can then align model training to the goal of seeking a model prediction that fits the data and can also reasonably admit the ODE description.

There is a hyperparameter **'coeff_ini = 1e8 '** that should be set to assert the model's predicted curve to begin at the same point as that suggested by the data. Model training proceeds using the EDL loss function ('**bergman_helpers.evidential_data_loss()**'). In the line '**Lalpha, Lbeta, Lnu, L_glucose, L_X = nn_I(t_z)**', the outputs '**L_glucose, L_X**' refer to the (mean) glucose target and X target respectively, while '**Lalpha, Lbeta, Lnu**' are the three outputs that are used to compute uncertainty of the glucose target (see eqn.7 of [1]).

2 Intermediate phase

In this phase, we first use the data-fitted model (obtained from the first phase) to construct a prior for the unknown parameters $\{p_1, p_2, p_3\}$. However, if there is a reasonable choice for the prior strongly motivated by physical considerations, etc., one should use it in place of the one that we propose here.

One has to set a finite domain for each parameter that reflects its expected minimum and maximum value. Our choices for '**p1_vals, p2_vals, p3_vals**' are motivated by the values inferred in [4]. The snippet computing '**results**' may take some time depending on the resolutions of the domains chosen. The array '**prob_grid**' refers to the density function $f(\Omega)$ as defined in eqn.9 of [1], but without the normalization constant.

The line '**p1_prior_mean, p2_prior_mean, p3_prior_mean, p1_prior_std, p2_prior_std, p3_prior_std = bergman_helpers.three_stats_prior(prob_grid, p1_vals, p2_vals, p3_vals)**' generates the parameters of the prior distribution $\pi(\Omega)$ for the parameters (product of three Gaussians). The mode and IQR of $f(\Omega)$ were used to construct each Gaussian (see discussion surrounding eqns.10,11 of [1]). There is also a code cell that plots both the numerical solution corresponding to the prior's mean and the data-fitted model's prediction. They should be close to each other as a check of the constructed prior.

We also construct the prior for the loss weight as described in Section 2.3 of [1]. The loss weight parameter is σ_R^2 in the formula for the PDE residual term as follows.

$$\frac{1}{2\sigma_R^2} \sum_{k=1}^{N_D} \mathcal{R}_k^2$$

where $\mathcal{R}_k^2$ is the PDE residual loss term. In this context, this is the sum[1]

$$\mathcal{R}^2 \equiv \left(\frac{dG}{dt} + p_1(G - G_b) + X(t)G(t) \right)^2 + \left(\frac{dX}{dt} + p_2 X(t) - p_3 \left(I(t) - I_b \right) \right)^2 . \tag{3}$$

In standard PINN models, the loss weight – here parametrized by σ_R^2 – is often taken as a hyperparameter that sets the importance of the asserting PDE constraint relative to the data loss term. In our formalism, σ_R^2 is a learnable parameter so this resolves the usual ambiguity of how to determine the loss weight. Our approach is as follows.

1. We assume the prior for σ_R^2 to be an inverse-gamma function (eqn.12 of [1]) parametrized by α_r, β_r which we compute in the code section titled **'Constructing the prior for the loss weight of the PDE residual term'**.

[1]See **'bergman_helpers.bergman_likelihood_array'**.

2. We set the initial σ_R^2 such that the likelihood function (which is the exponential of the PDE residual loss term) is close to the parameter prior $\pi(\Omega)$ in the sense of the Kullback-Leibler(KL) divergence[2] between them. This is done in the line **'mean_var = bergman_helpers. mean_var_KL_3D(1e-4, likelihood_function, p1_prior, p2_prior, p3_prior)'** . Intuitively, we wish to seek a value of σ_R^2 such that as a distribution in the parameters p_1, p_2, p_3, the likelihood function is close to the prior before the start of training in the second phase. We also set such a value to be the mean of the prior for σ_R^2, which is $\beta_r/(\alpha_r - 1)$ (see eqn.12 of [1] for the prior formula).

   As the model is trained using the PDE constraint and data-fitting in the second phase, we would like σ_R^2 to evolve towards its minimal value. A natural candidate for this minimal value is the one that yields a likelihood function that is close to a density function with standard deviations characterized by the resolution of the parameter domains. This is done in the line **'min_var = bergman_helpers.mean_var_KL_3D(1e-5, likelihood_function, l_p1_prior, l_p2_prior, l_p3_prior)'**. We also set this to be the mode of the inverse-gamma prior which is $\alpha_r/(\beta_r + 1)$. This equation and the preceding one that sets it to be the mean $\alpha_r/(\beta_r - 1)$ allow us to solve for α_r, β_r which is done in the line **'alpha, beta = bergman_helpers.PDE_prior (min_var, mean_var)'**. Thus σ_R^2 evolves during model training in the second phase, starting from the initial value (**'mean_var'**) to the asymptotic value (**'min_var'**). Since we are taking the negative logarithm of the prior for σ_R^2 in the loss function, gradient-descent based training then induces a decrease in σ_R^2 from the mean to the mode of the prior for σ_R^2 (eqn.12 of [1]).

3. In the snippet preceding line, there is a float named **'reg_s'**. This should be initially set to 1, representing the exact resolution of the parameter domain, i.e. the minimal spacing between values of p_1, p_2, p_3. But if there is issue with the KL divergence minimization procedure, e.g. it doesn't converge, or the plotted graph shows that the optimal variance value doesn't lie at a local minimum, etc., then one can try slightly increasing this value to be larger than 1. If it requires an adjustment to a value beyond 2, then this implies that the form of the likelihood function is quite different from the prior and this method will not work. One should then simply use a constant value for the σ_R^2, such that the one obtained by the code line **'min_var = bergman_helpers.mean_var_KL_3D(1e-5, likelihood_function, l_p1_prior, l_p2_prior, l_p3_prior) '**.

3 Second training phase

This training phase involves the full loss function as defined in eqn.17 of [1], in particular this includes the PDE residual loss term of standard PINN. From physical considerations, we would like the learned p_1, p_2, p_3 to be positive and thus we expressed each as the sigmoid function of an auxiliary parameter. Apart from the weights and biases of the base perceptron model, $p_1, p_2, p_3, \sigma_R^2$ are the learnable parameters. σ_R^2 is represented by the torch tensor s in the notebook. We set the initial values of p_1, p_2, p_3 to be those of the mean of the prior, and σ_R^2 to be the mean of the inverse-gamma prior (as performed by the line **'s_initial = torch.tensor(mean_var)'**)

The Bergman model involves two coupled ODEs. For the PDE residual loss, we took it as the sum of each ODE (see the line **'mse_physics_loss = mse_physics_loss_G +**

[2]This is an information-theoretic distance between two probability distributions. See e.g. [7]

mse_physics_loss_X') although in principle, one could consider adjusting their relative weights.

In our formalism, apart from the loss weight that imposes the initial condition (**'coef_ini'**), there are no other hyperparameters to tune within the loss function. But we have defined various coefficients : **coef_physics**, **coef_prior**, **coef_pp**, just in case their adjustments to values away from 1 are needed.

References

[1] Tan HS, Wang K, McBeth R. Evidential Physics-Informed Neural Networks for Scientific Discovery; 2025. Available from: `https://arxiv.org/abs/2509.14568`.

[2] Bergman RN. Origins and History of the Minimal Model of Glucose Regulation. Frontiers in Endocrinology. 2021;Volume 11 - 2020.

[3] Bergman RN, Ider YZ, Bowden CR, Cobelli C. Quantitative estimation of insulin sensitivity. The American journal of physiology. 1979;236 6:E667-77.

[4] Kartono A, Nurullaeli, Syafutra H, Wahyudi ST, Sumaryada T. A mathematical model of the intravenous glucose tolerance test illustrating an n-order decay rate of plasma insulin in healthy and type 2 diabetic subjects. IOP Conference Series: Materials Science and Engineering. 2019 may;532(1):012016.

[5] Amini A, Schwarting W, Soleimany A, Rus D. Deep evidential regression. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS '20. Red Hook, NY, USA: Curran Associates Inc.; 2020. .

[6] Sensoy M, Kaplan L, Kandemir M. Evidential Deep Learning to Quantify Classification Uncertainty; 2018. Available from: `https://arxiv.org/abs/1806.01768`.

[7] Shlens J. Notes on Kullback-Leibler Divergence and Likelihood; 2014. Available from: `https://arxiv.org/abs/1404.2000`.