A further investigation on Symbolic genetic algorithm for discovering open-form partial differential equations (SGA-PDE)

Sixiong Shao

Contents

1. Introduction of SGA-PDE

1.1 Motivation of SGA-PDE:

To develop a new PDE discovery method that can improve extant automatic PDE mining techniques, capable of handling any open-form PDEs.

1.2 Common method for automatic moning PDEs:

- **Sparse Regression** (e.g., SINDy, PDE-FIND, DL-PDE)

  Candidate set is closed, it often requires the user to determine the approximate form of the governing equation in advance and then give all possible function terms in the candidate set. However, the governing equations to be mined often have complex forms in the scenario in which the PDE discovery method is needed, and it is challenging to generate a complete candidate set containing all the possible function terms in practice. Therefore, although sparse regression achieves good performance in some simple problems, it cannot find the solution not included in the candidate set, which constrains the application of these sparse-regression-based methods in practice.

- **Genetic algorithm** (e.g., EPDE, DLGA-PDE)

  Candidate set is wider but still limited, it uses crossover and mutation to expand the candidate set, so that the PDE can be automatically found without determining all the candidates in advance. However, it is still based on basic units of variation, which are derivatives of different orders.

1.3 Limitations of extant methods:

The representation of PDE is based on the given candidates (function terms) or gene fragments (basic derivatives), which can only carry out simple interactive operations, such as addition, subtraction, and multiplication. It constrains the potential equation structure that the algorithm can find, and thus, it is challenging to discover open-form PDEs from observations.

1.4 Characteristics of SGA-PDE:

- SGA-PDE uses symbolic mathematics to realize the flexible representation of any given PDE (From the perspective of symbolic mathematics, any equation can be expressed as a graph or a binary tree).
- SGA-PDE is gradient free, it does not rely on gradient information (i.e., derivatives) to find optimal solutions.
- Use a genetic algorithm specially designed for trees to mine the most suitable PDE directly from experimental data.

1.5 SGA-PDE Methodology:

1. Create 20 PDEs (a forest), for each PDE, create multiple terms (trees). Then use the difference method to calculate the predicted value of du/dt, perform STRidge between the predicted value and the actual value from the dataset to get the best weights of terms of this PDE. Then calculate the mean squared error (mse) between the predicted value and the actual value to get AIC.
2. Add these PDEs to a library
3. Sort these PDEs according to their AIC, from lowest AIC to highest.
4. For each generation, perform a 'crossover' followed by a 'change'. Iterate 100 generations.

1.5.1 Overview of STRidge:

STRidge aims to find a sparse approximation to the linear regression problem using ridge regression with an iterative thresholding approach.

First, compute the initial ridge regression weights. Then thresholding the weights: Identify indices of weights that are above, and below the threshold tolerance, create a new set of indices excluding the small indices. Iteratively compute the weights for the remaining indices using ridge regression, until the number of relevant indices does not change. By doing so, it gradually eliminates small weights, promoting sparsity in the solution.

The tolerance (tol) is initially set to 1 (Delta tolerance: d-tol = 1), if one loop of STRidge improves the AIC, then tol = tol + d-tol. If AIC does not improve, then decreases tol by 2 times d-tol, and reduce the step size of the tolerance change, making it smaller as the iterations progress (d-tol = 2 * d-tol / (maxit - iter)). As iter increases, d-tol decreases, leading to smaller changes in tol. For the test case, the regularization parameter for ridge regression is set to 0, so the ridge regression at here is standard least square regression.

1.5.2 Explanation of Crossover:

- Get n best PDEs from 2n samples.
- Crossover one term in each PDE, compare with the PDE library, get m non-repeating PDE. Add these new PDE to the library.
- Sort 2n+m PDEs according to their AICs, remain 2n best PDEs.

1.5.3 Explanation of Change (mutation, replacement):

- Remain the best PDE unchanged, change the rest 19 PDEs.
- For each term (tree) in a PDE, mutate each node according to p_mute (probability of mutation).
- For each PDE, decide whether or not to replace according to p_rep (probability of replacement), if need to replace, create a new term (tree), and randomly select a term in this PDE to replace. Add the changed PDE to the library

2. Further Investigations of SGA-PDE:

2.1 Task 1A: General Testing

2.1.1 Motivation:

Testing the accuracy and efficiency of SGA-PDE for the five models (Burgers, KdV, Chafee-Infante, PDE compound, PDE divide) using the default parameters p_mute = 0.3, p_rep = 1.

2.1.2 Method:

Run the program, let the algorithm iterate 100 generations. Record the generated best PDE equation (in inorder form) and its AIC for each generation, and plot AIC against generation.

2.1.3 Result:

Burgers Equation:
The expected PDE is

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + 0.1\frac{\partial^2 u}{\partial x^2} \tag{1}$$

This algorithm with default parameter is able to generate accurate PDE within 100 generations, the AIC of the final PDE is -25.54, it has the form:

$$-0.5006\frac{\partial\left(u^2\right)}{\partial x}+0.0363\frac{\partial\left(\frac{\partial u}{\partial x}\right)}{\partial x}+0.0639\frac{\partial^2 u}{\partial x^2}=-1.012u\frac{\partial u}{\partial x}+0.1002\frac{\partial^2 u}{\partial x^2} \tag{2}$$

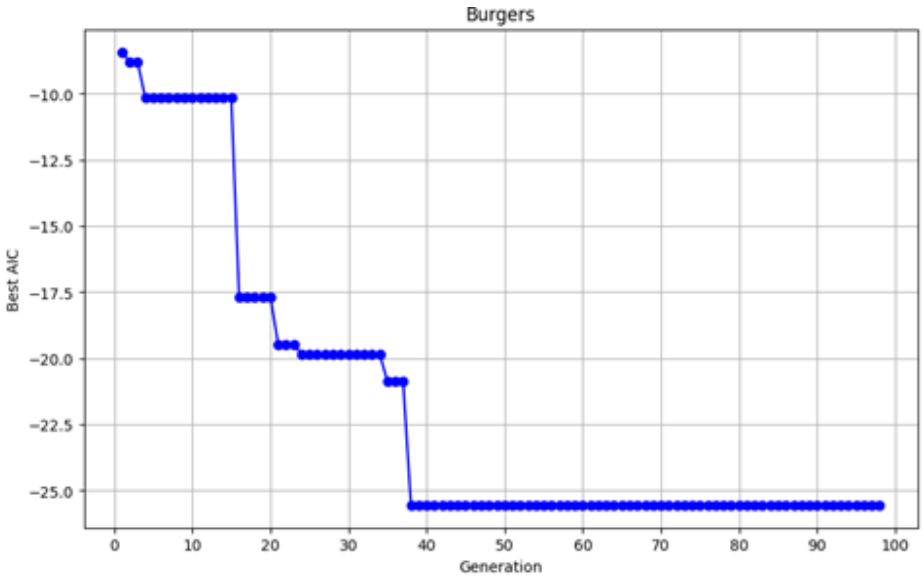


Figure 1: AIC of the best generated PDE across generations

KdV Equation:

The expected PDE is

$$\frac{\partial u}{\partial t}=-u\frac{\partial u}{\partial x}+0.0025\frac{\partial^3 u}{\partial x^3} \tag{3}$$

This algorithm with default parameter is able to generate accurate PDE within 100 generations, the AIC of the final PDE is -10.31, it has the form:

$$-1.0009u\frac{\partial u}{\partial x}+0.0025\frac{\partial^3 u}{\partial x^3} \tag{4}$$

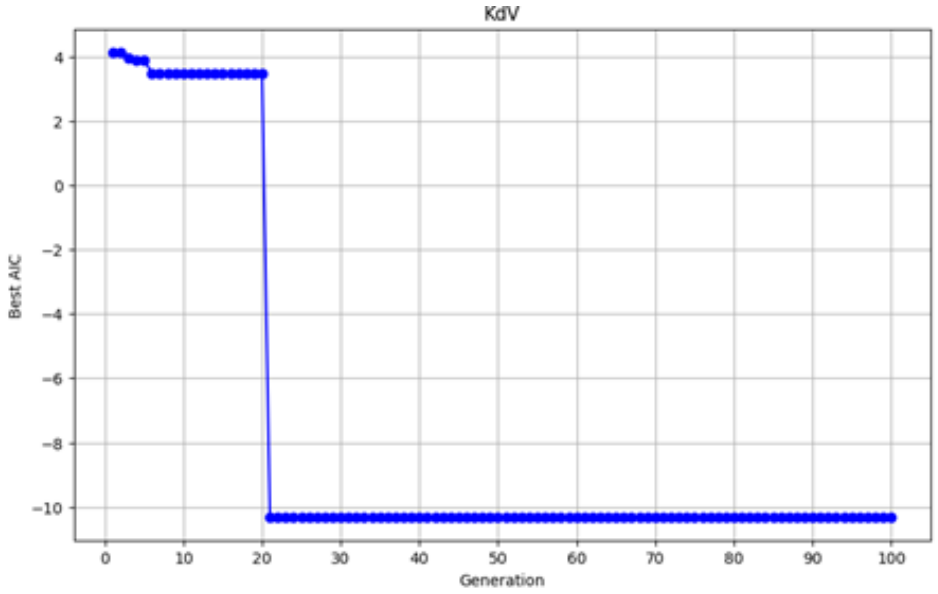


Figure 2: AIC of the best generated PDE across generations

Chafee-Infante Equation:

The expected PDE is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - u + u^3 \tag{5}$$

This algorithm with default parameter is able to generate accurate PDE within 100 generations, however there is a problem. At Generation 24, the equation is accurate: (AIC: -11.804011775480923)

$$1.0002\frac{\partial^2 u}{\partial x^2} - 1.0008u + 1.0004u^3 \tag{6}$$

However, after that, the best PDE becomes: (AIC: -11.804011775480934)

$$1.0002\frac{\partial^2 u}{\partial x^2} - 1.0008u \tag{7}$$

AIC decreased by 1.1e-14, but the PDE becomes inaccurate, the term: u^3 is missing. This is because more concise equation has better AIC, and in this case the reward for conciseness compensates for the effect of removing an accurate term, and it even causes a slight improvement in AIC.

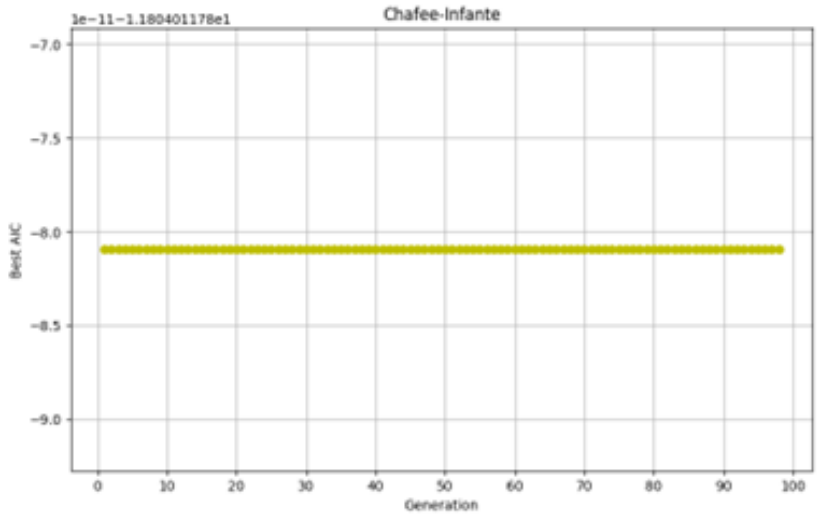


Figure 3: AIC of the best generated PDE across generations. Note: there is an exact PDE when the PDE pool is first generated, which is why AIC has already met the expected value at the beginning.

PDE Compound:

The expected PDE is

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(u\frac{\partial u}{\partial x}\right) \tag{8}$$

For the default parameters p_mute = 0.3 and p_rep = 1, accurate PDE cannot be generated within 100 generations. The final PDE has AIC: -0.126. It can be simplified as:

$$0.9382u\frac{\partial^2 u}{\partial x^2} - 0.0848\left(\frac{\partial u}{\partial x}\right)^2 \tag{9}$$

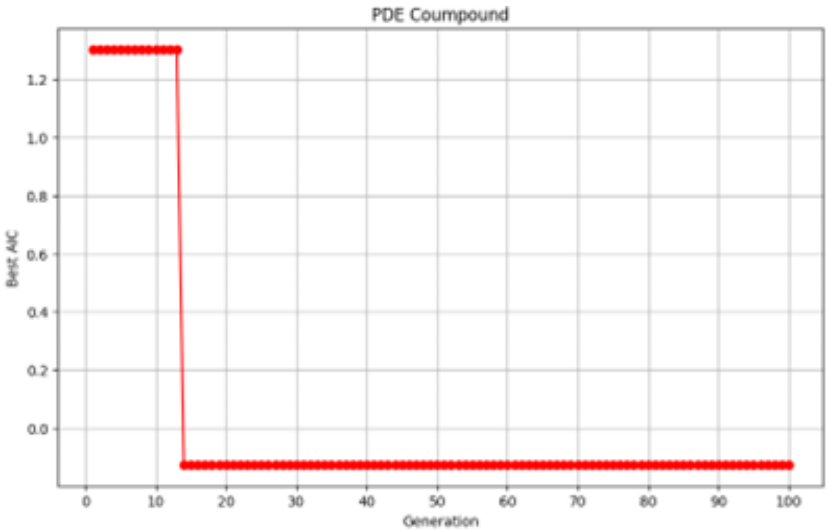


Figure 4: AIC of the best generated PDE across generations that does not generated accurate PDE

When change p_mute to 0.5 and remains p_rep = 1. Accurate PDE can be generated

within 100 generations. The final PDE has AIC: -2.4616, it can be written and simplified as:

$$0.4903\frac{\partial^2(u^2)}{\partial x^2} = 0.4903\frac{\partial}{\partial x}\left(2u\frac{\partial u}{\partial x}\right) = 0.9806\frac{\partial}{\partial x}\left(u\frac{\partial u}{\partial x}\right) \tag{10}$$

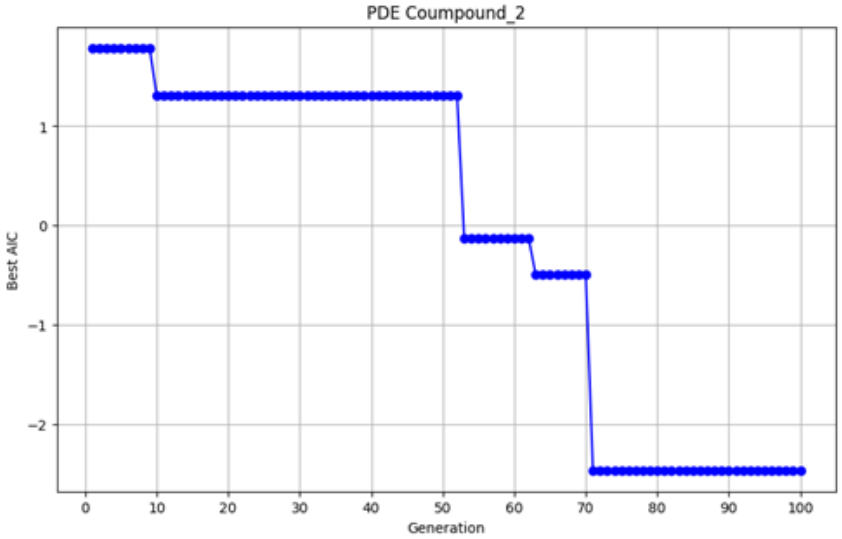


Figure 5: AIC of the best generated PDE across generations that generates accurate PDE

PDE Divide:

The expected PDE is

$$\frac{\partial u}{\partial t} = \frac{-1}{x}\frac{\partial u}{\partial x} + 0.25\frac{\partial^2 u}{\partial x^2} \tag{11}$$

For the default parameters p_mute = 0.3 and p_rep = 1, accurate PDE cannot be generated within 100 generations. The final PDE has AIC: -5.072

$$-1.29\frac{\partial(\frac{u}{x})^2}{\partial x} - 1.96u \cdot x \tag{12}$$

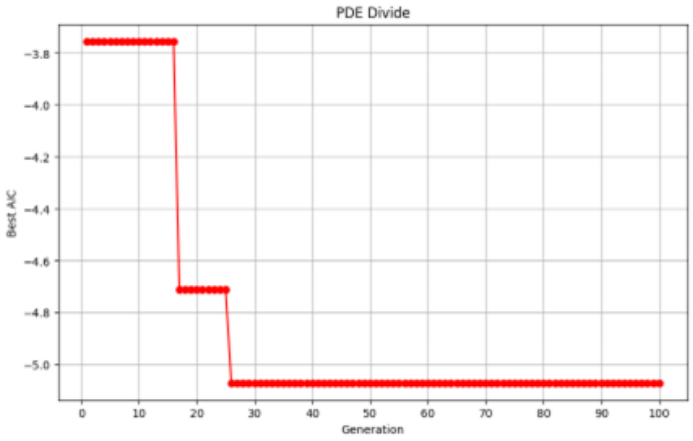


Figure 6: AIC of the best generated PDE across generations that does not generated accurate PDE

When change p_mute to 0.5 and remains p_rep = 1. Accurate PDE can be generated within 100 generations. The final PDE has AIC: -10.35, it can be written and simplified as:

$$= -0.25\left(\frac{\partial^2 x}{\partial x^2} - \frac{\partial^2 u}{\partial x^2}\right) + \frac{\frac{\partial u}{\partial x}}{0 - x} = 0.25\frac{\partial^2 u}{\partial x^2} + \frac{1}{-x}\frac{\partial u}{\partial x} \tag{13}$$

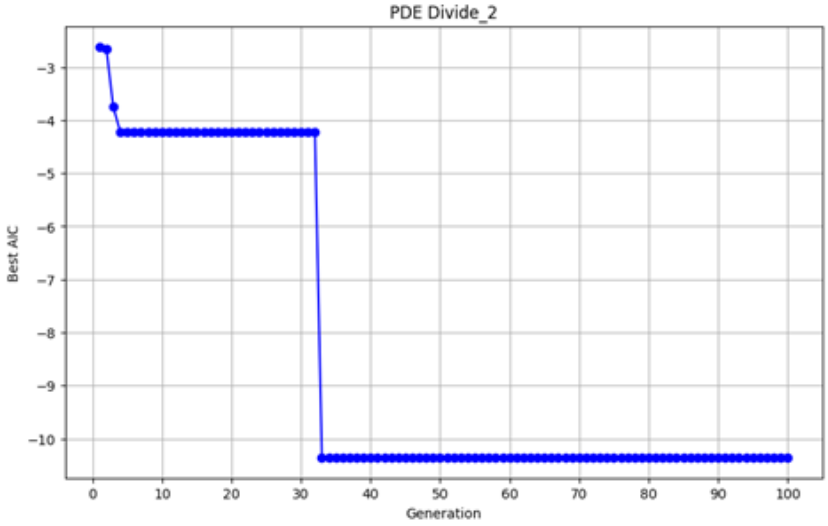


Figure 7: AIC of the best generated PDE across generations that generates accurate PDE

2.1.4 Discussion:

Further research is needed on how this algorithm leads to a decrease in AIC, does it occur at crossover, mutation or replacement? The hyperparameters such as p_mute and p_rep seems to be significant in the efficiency or even the accuracy of the algorithm. For different models, the optimal parameters might be different. Why is it so?

2.2 Task 1B: Relation between algorithm efficiency and AIC decreases

2.2.1 Motivation:

To understand the factors influencing the accuracy and efficiency of the algorithm, it is essential to determine when the AIC (Akaike Information Criterion) decreases. What causes this decrease? Is it a sudden drop or a gradual process?

2.2.2 Method:

Run algorithm for five models (Burgers; KdV; Chafee-Infante; Compound; Divide), and trace each AIC drop, investigate whether it occurs at crossover, mutation or replacement.

2.2.3 Result:

Burgers Equation:

All AIC changes are caused by mutation. Critical AIC change that led to accurate result appear to be a sudden drop of AIC.

Generation	Reason	Current AIC	AIC change
1	Mutation	-8.513905723709051	0.0914039073475923
2	Mutation	-8.791184396319416	0.27727867261036465
3	Mutation	-9.103815537943747	0.3126311416243315
4	Mutation	-10.164716806550382	1.060901268606635
9	Mutation	-17.667345613599213	7.502628807048831
10	Mutation	-20.076043541447756	2.408697927848543
11	Mutation	-25.06059590010225	4.9845523586544935
12	Mutation	-25.060595979940796	7.9838546440669e-08

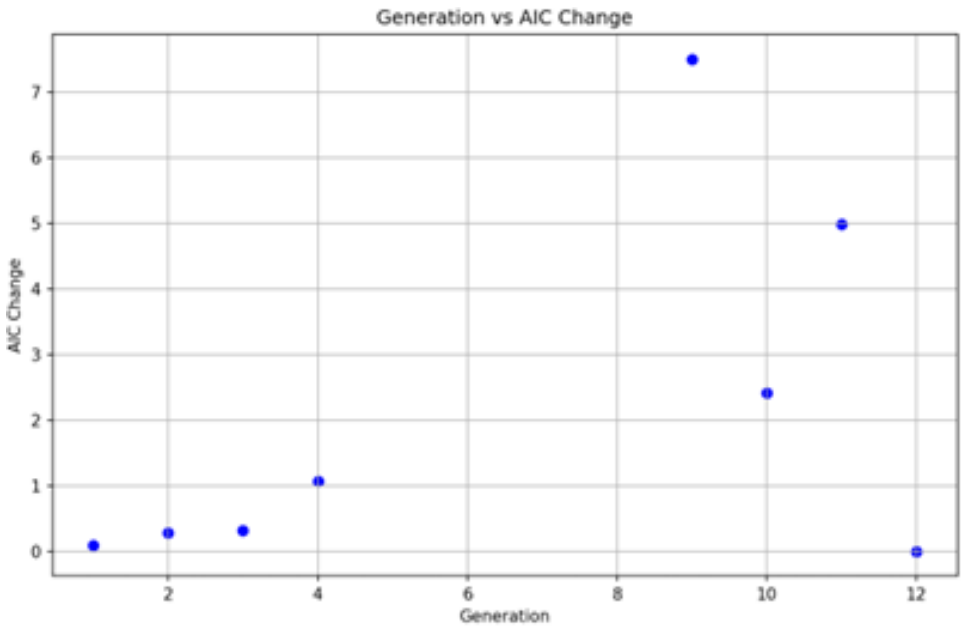


Figure 8: AIC change across generations for Burgers equation

KdV:

All AIC changes are caused by mutation. Critical AIC change that led to accurate result appear to be a sudden drop of AIC.

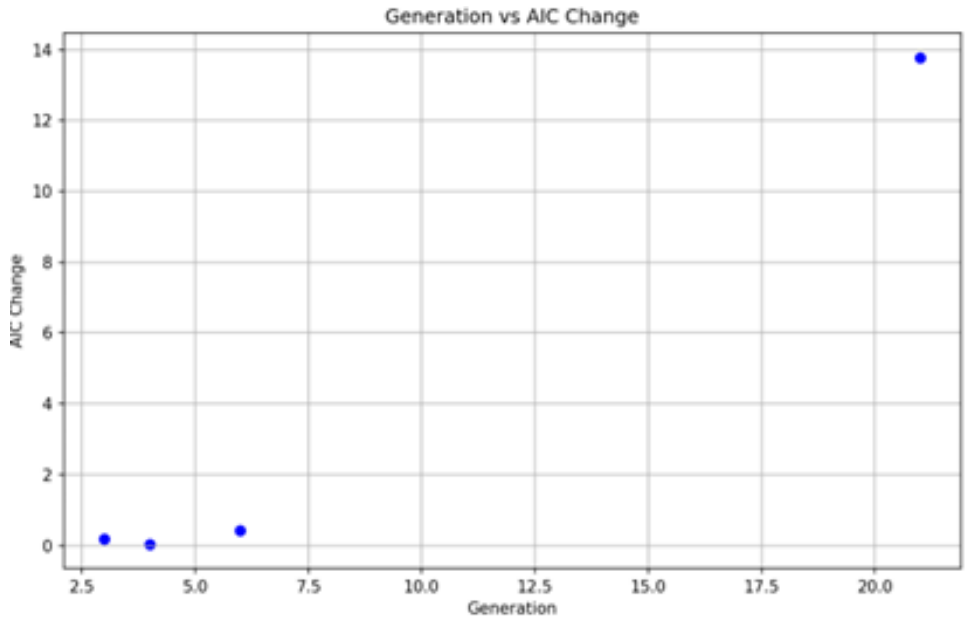


Figure 9: AIC change across generations for KdV equation

Chafee-Infante:

All AIC changes are caused by mutation. Critical AIC change that led to accurate result appear to be a sudden drop of AIC.

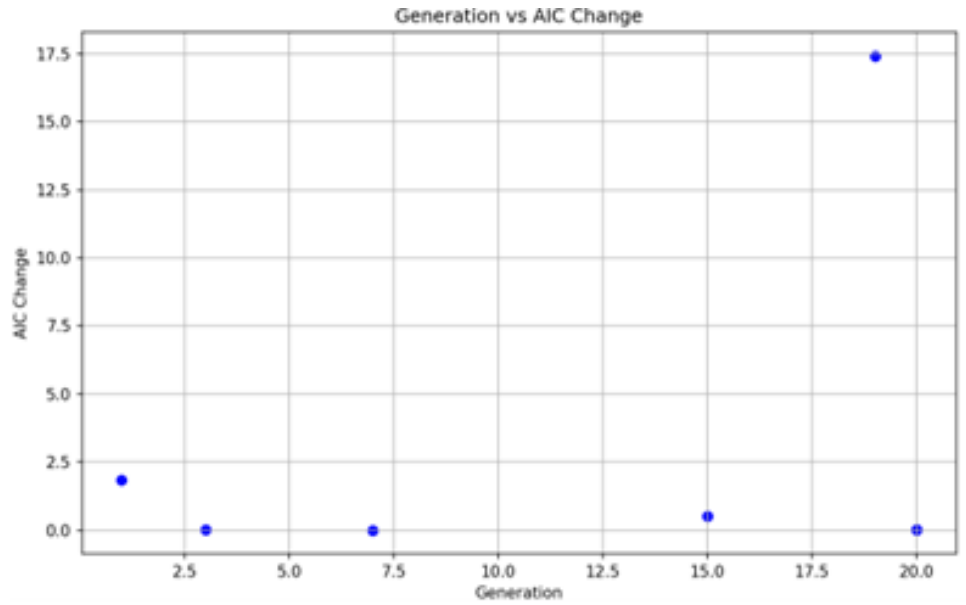


Figure 10: AIC change across generations for Chafee-Infante equation

PDE Compound:

All AIC changes are caused by mutation. Critical AIC change that led to accurate result appear to be a sudden drop of AIC.

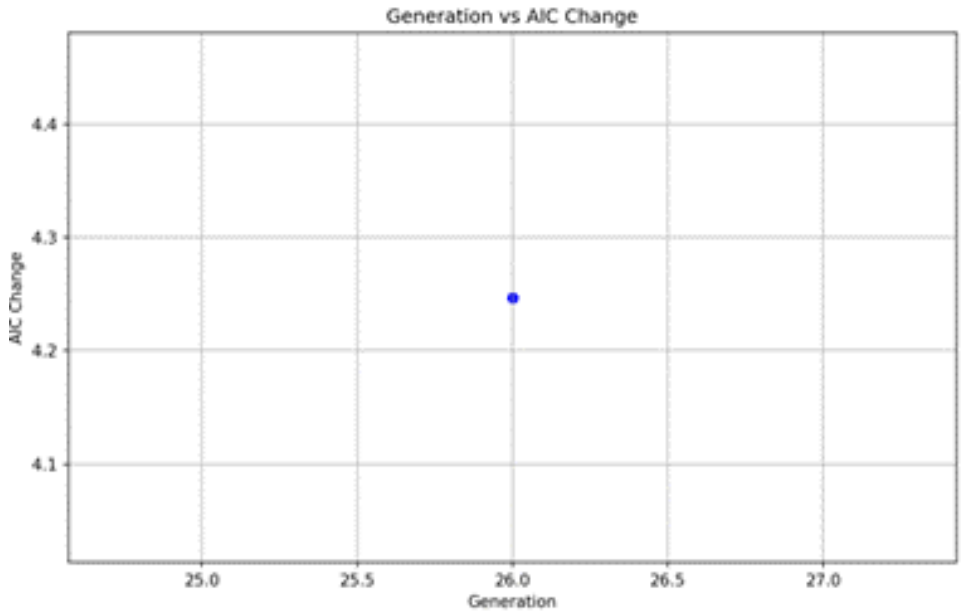


Figure 11: AIC change across generations for Compound PDE

PDE Divide:

All AIC changes are caused by mutation. Critical AIC change that led to accurate result appear to be a sudden drop of AIC.

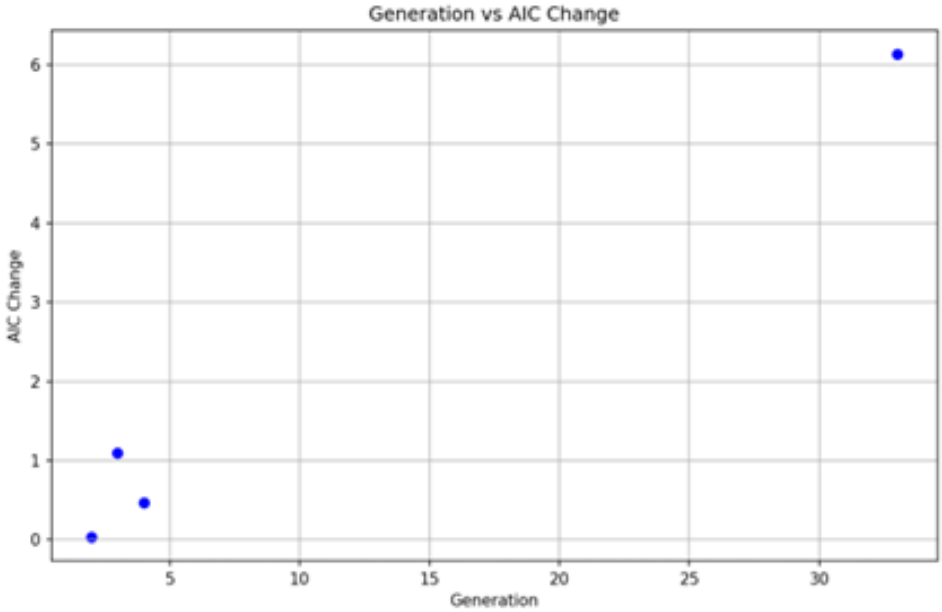


Figure 12: AIC change across generations for Divide PDE

2.2.4 Discussion:

All decreases in AIC occurred during mutation. However, it cannot conclusively be stated that mutation is the most significant process causing the AIC drop. This is due to a peculiar phenomenon observed in models like PDE divide: when p_rep is low, the algorithm fails to accurately determine the equation within 100 generations. The likely

reason is that crossover and replacement processes significantly expand the candidate sets, allowing final improvements to be achieved through mutation. Besides, the move towards an ultimately accurate PDE appears to be accompanied by a sudden decline of AIC, which is not a gradual process in terms of AIC measures.

2.3 Task 1C: Hyperparameters and algorithm efficiency

2.3.1 Motivation:

The hyperparameters p_mute and p_rep seem to be significant in the efficiency or even the accuracy of the algorithm. The optimal values for these parameters might vary across different models. Identifying the best parameters for each model could offer valuable insights to enhance the overall efficiency of the algorithm.

2.3.2 Method:

Testing p_mute and p_rep from 0.1 to 1 respectively for each model, records the generation it takes to reach the accurate PDE. Repeat 5 times and take the average.

2.3.3 Result:

Since the time cost for this algorithm is significantly high, only the Burgers equation is tested for p_mute = 0.3 and p_rep from 0.1 to 1.

p_mute	p_rep	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Average Generation used
0.3	0.1	65	11	71	56	63	53.2
0.3	0.2	7	9	2	55	10	16.6
0.3	0.3	11	23	19	3	18	14.8
0.3	0.4	51	20	9	4	3	17.4
0.3	0.5	62	29	53	2	61	41.4
0.3	0.6	70	3	17	8	74	34.4
0.3	0.7	75	5	10	76	8	34.4
0.3	0.8	34	6	51	7	31	25.8
0.3	0.9	4	5	14	64	67	30.8
0.3	1	18	22	2	3	18	**12.6**

2.3.4 Discussion:

The hyperparameters in the symbolic representation mainly change the size of the search space, such as the height of tree (complexity of the generated PDE), and the width of PDE (how many terms in one PDE). The hyperparameters in the genetic algorithm mainly change the strength of randomness, so p_mute and p_rep control the randomness of the system. Larger the parameter, stronger the randomness.

From the data above, it appears that with low p_rep, the algorithm heavily depends on the initially generated PDE pools. If the initial PDEs have really different tree topology compare with the expected PDE, it becomes difficult to achieve improvement. Current results indicate that p_rep = 1 is the optimal parameter for the Burgers equation. However, further experiments are needed to confirm this hypothesis and obtain reliable answers for this task.

2.4 Task 2A: Investigation of Zero variable

2.4.1 Motivation:

In addition to common operators and variables, SGA-PDE includes Zero as one of the variables. This seems unusual, as there is no obvious reason for including Zero as a variable, and it can lead to division by zero issues. To address this, the algorithm defines a new 'divide' method, which adds a small value (eta = 1e-10) to the denominator when it is zero.

2.4.2 Method:

Remove Zero from the variable candidate set, and test the algorithm, to see if the efficiency increases or decreases.

2.4.3 Result:

Removing Zero variable will decrease efficiency slightly but not significantly. However, by removing Zero variable, the program might generate terms (trees) that have more complicated tree structures.

For instance, for the Burgers equation, the final result generated after removing Zero variable appears:

$$-u\frac{\partial u}{\partial x} + 0.1\frac{\partial^2 u}{\partial x^2} - 0.0001\frac{\partial^4 u}{\partial x^4} + 0.0004\frac{\partial^3\ (u^2)}{\partial x^3} \tag{14}$$

The first two terms are accurate, the last two terms are redundant despite having really small weights. This redundancy may stem from the removal of the Zero variable. Without the Zero variable, each generated PDE has a fixed number of terms after the pool is created. For a term to become zero, it must be replaced by or mutated to $\frac{\partial^2 x}{\partial x^2}$, which is a rare occurrence. With the Zero variable, it is relatively easier to mutate a term to Zero by multiplying Zero or dividing Zero, or it can make an complicated term simpler by mutating a variable to Zero. Therefore, some irrelevant or over-complicated terms can vanish more easily.

2.4.4 Discussion:

Zero in the operand helps prune the tree by muting certain leaves or branches. This zero operand is crucial in SGA-PDE, as it is helpful in finding the optimal node combination when the topology of the binary tree remains unchanged. Therefore, the Zero variable should be retained until a better algorithm is developed.

2.5 Task 2B: Investigation on the default term

2.5.1 Motivation:

For each generated PDE, there is a default term: u. Including a default term somewhat undermines the goal of discovering open-form PDEs. Therefore, it is essential to investigate the use and purpose of adding this default u term.

2.5.2 Method:

Remove the default u term from each generated PDE, and test whether the efficiency increases or decreases.

2.5.3 Result:

There are no apparent differences of the efficiency of algorithm except for the Chafee-Infante. For this model, the efficiency decreases, this is because there is a term u in Chafee-Infante equation, so having a default u term will speed up the algorithm.

2.5.4 Discussion:

Including the dependent variable u in the candidate set is a strategy to accelerate optimization in SGA-PDE. However, it is only useful when the term u is a common function term in PDEs. In addition, although it is difficult to generate a single variable such as 'u' or 'x', since the rule to creating a tree (a term) is that the root must be an operator, without a default u term, the algorithm can still generate a u term. For example, it can generates $u + 0$; $u + u$; $\frac{u^2}{u}$ etc.

2.6 Task 3: Adding a constant term to PDE

2.6.1 Motivation:

Can this algorithm discovers the constant terms in PDEs? Such that to get some PDEs like:

$$\frac{\partial u}{\partial t} = \Phi\left(u, x\right) + 1 \tag{15}$$

2.6.2 Method:

Theoretically there are multiply way to generate a constant term. For example: $\frac{u}{u}, \frac{\partial x}{\partial x}$. By changing u to $u - \xi t$, we get:

$$\frac{\partial \left(u - \ \xi t\right)}{\partial t} = \frac{\partial u}{\partial t} - \xi \cdot 1 \tag{16}$$

So, the PDE becomes:

$$\frac{\partial u}{\partial t} = \Phi \left(u, x\right) + \xi \cdot 1 \tag{17}$$

Where ξ is the coefficient of the constant term. So, first change u_t by adjusting the dataset according to equation (16), and then run the program to see if a constant term appears. Adjust the coefficient ξ according to the performance of the algorithm.

2.6.3 Result:

For the Burgers equation:

When $u = u - t$, the expected equation becomes $\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + 0.1\frac{\partial^2 \ u}{\partial x^2} + 1$, but the final generated equation is $\frac{x}{x} = 1$. When $u = u - 0.1t$, the generated equation becomes 0.1. When $u = u - 0.05t$, the generated equation becomes 0.05. When $u = u - 0.01t$, the generated equation becomes $\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + 0.1\frac{\partial^2 \ u}{\partial x^2}$. The result indicates that when the constant term is greater or equals to 0.05, it will cover up other terms, and when the constant term is smaller or equals to 0.01, other terms will cover up the constant term.

When $u = u - 0.015t$, the final equation becomes $-0.97u\frac{\partial u}{\partial x} + 0.01\frac{\partial^2 u}{\partial x^2} + 0.07\frac{\partial u}{\partial x}$, with AIC = -9.43341. The first two terms are accurate, but the last term, which should be a constant term is the first derivative. When I manually input the accurate three terms: $u\frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \frac{x}{x}$ into the weight and AIC calculator, the weights become (-0.84, 0.086, 0), so the equation becomes $-0.84u\frac{\partial u}{\partial x} + 0.086\frac{\partial^2 u}{\partial x^2}$, and its AIC = -9.246375, which is worse than that of the equation shown above. Therefore, in this case, it is impossible for this algorithm to derive the accurate equation.

When $u = u - 0.03t$, the equation generated is completely irrelevant: $-0.066x\frac{\partial u}{\partial x}$, with AIC = -8.4221. When I manually input the accurate three terms: $u\frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \frac{x}{x}$ into the weight and AIC calculator, the weights become (-0.57, 0.058, 0), so the equation becomes $-0.57u\frac{\partial u}{\partial x} + 0.058\frac{\partial^2 u}{\partial x^2}$, and its AIC = -7.108, which is worse than that of the generated equation. So, it is also impossible to derive the accurate equation.

2.6.4 Discussion:

Although it is theoretically possible to generate a constant term, for the current PDE, it is extremely difficult to add an appropriate constant term that neither overshadows nor is overshadowed by the other terms. Additionally, the constant term can easily be mistaken for other terms, which leads to a similar result. However, the main problem seems to be in the calculation of the weights of terms. In both the 0.015 and 0.03 cases, even when

the input PDE is accurate, when calculating the weights of terms, the constant terms are omitted (calculated as 0), which causes the problem.

Current algorithm uses STRidge to calculate the weights of terms, it is possible that the constant term is omitted during the threshold process. So when the algorithm only uses normal least square regression to calculate the weights, under the condition of $u = u - 0.015t$, the result becomes (input PDE has three terms: $u\frac{\partial u}{\partial x}$, $\frac{\partial^2 u}{\partial x^2}$, $\frac{x}{x}$):

$$-0.012u - 0.81u\frac{\partial u}{\partial x} + 0.082\frac{\partial^2 u}{\partial x^2} + 0.016\frac{x}{x} \tag{18}$$

The constant term appears, but there is a default u term, the AIC of this PDE is -5.686. When removing the default u term, the result becomes:

$$-0.84u\frac{\partial u}{\partial x} + 0.086\frac{\partial^2 u}{\partial x^2} + 0.015\frac{x}{x} \tag{19}$$

Its AIC is -7.619, it becomes closer to the expected PDE, but the weights of the first two terms are slightly off.

However, when only input two terms without the constant term, the result becomes:

$$-0.84u\frac{\partial u}{\partial x} + 0.086\frac{\partial^2 u}{\partial x^2} \tag{20}$$

It has less terms than before, so its AIC improves to -9.247. Since it has a better AIC, in this case the algorithm still cannot see the constant term.

The utilization of STRidge and AIC to assess PDE accuracy suggests that SGA-PDE prioritizes discovering more sparse and concise results. This emphasis may stem from its goal of outlining a general form of a PDE. Introducing a small constant term to the PDE minimally alters its shape, making it challenging to identify such terms using this algorithm, despite their potential to be generated.

Moreover, in some cases, adjusting the weights of terms in a PDE can effectively compensate for an additional constant term. When certain term weights are altered, the resulting PDE may closely resemble the original PDE with an added constant term. This complicates the algorithm's task of identifying constant terms, even without the constraint of sparsity. For example, for the PDE Compound, after adding a constant 1, the equation becomes (its visualization see Fig 13):

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(u\frac{\partial u}{\partial x}\right) + 1 \tag{21}$$

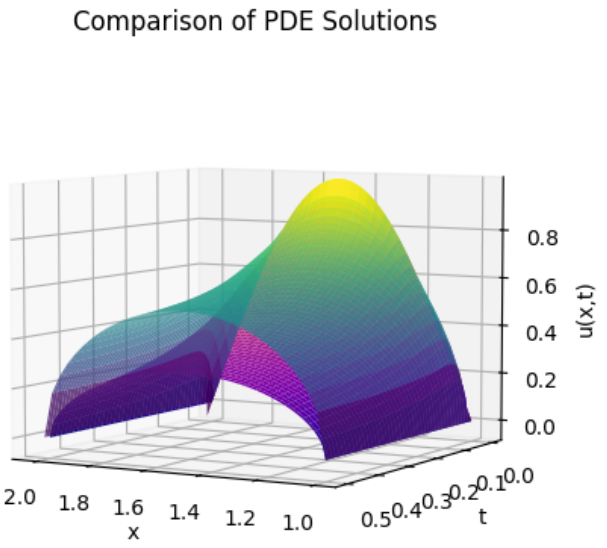


Figure 13: Comparison between PDE Compound and PDE Compound with constant term: purple curve is the original PDE, viridis curve is the PDE with a constnat term

However, when chaing the weight of the origianl PDE to 0.4 ($\frac{\partial u}{\partial t} = 0.4\frac{\partial}{\partial x}\left(u\frac{\partial u}{\partial x}\right)$), the equation becomes extremely similar to the equation with constant term (See Fig 14):

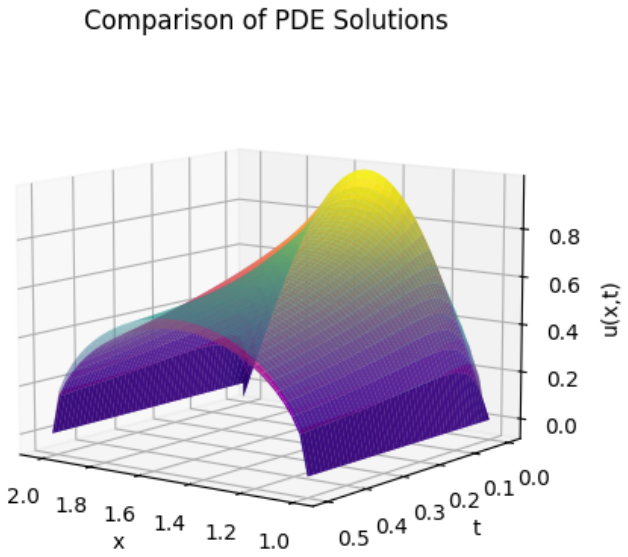


Figure 14: Comparison between PDE with constant term and PDE in difference weight of term: viridis curve is the PDE with a constant term

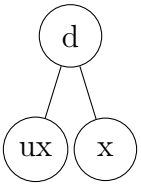
2.7 Task 4: Investigation on the AIC differences between same equation

2.7.1 Motivation:

Some PDEs that can be simplified to the same PDE have significantly different AICs. For example, the second term of the Burgers equation $\frac{\partial^2 u}{\partial x^2}$ can be express in two forms. However, with the first term being the same, the PDE has an AIC of -15.69 if the second term in form 1 and an AIC of -25.06 if the second term in form 2.

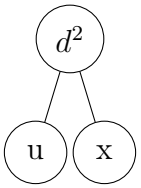
- Form 1:

  "d" means first partial derivative of the left child node with respect to the right child node, "ux" means $\frac{\partial u}{\partial x}$



- Form 2:

  "d^2" means second partial derivative of the left child node with respect to the right child node



2.7.2 Method:

Investigate the code and find the difference between the two PDE in terms of AIC calculation.

2.7.3 Result:

Since the number of terms is the same, the AIC difference is due to differences in MSE (Mean Square Error). When calculating MSE, the MSE varies between the terms in Form 1 and in Form 2.

The algorithms for numerically calculating the two terms are shown below. Although they have slight differences, in most cases, the differences will not be large enough to cause MSE to vary significantly.

$$Form\ 1\ \ (ux\ d\ x) = \frac{ux_{i+1} - ux_{i-1}}{2\ dx} = \frac{u_{i+2} - 2u_i + u_{i-2}}{4\ dx^2} \tag{22}$$

$$Form\ 2\ \left(u\ d^2\ x\right) = \frac{u_{i+1} - 2u_i + u_{i-1}}{dx^2} \tag{23}$$

By visualizing the difference between the numerical values of the two terms, it can be seen that there is a huge difference in the middle of dataset.

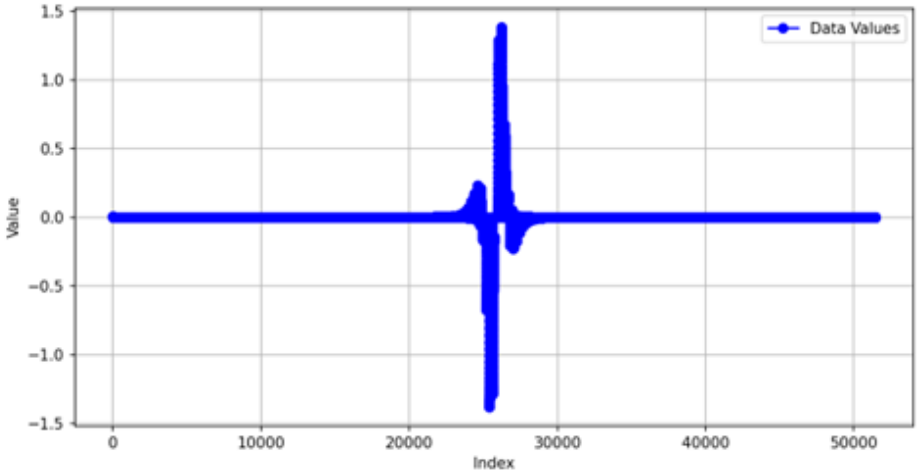


Figure 15: MSE difference of Form 1 and Form 2 of all the data points

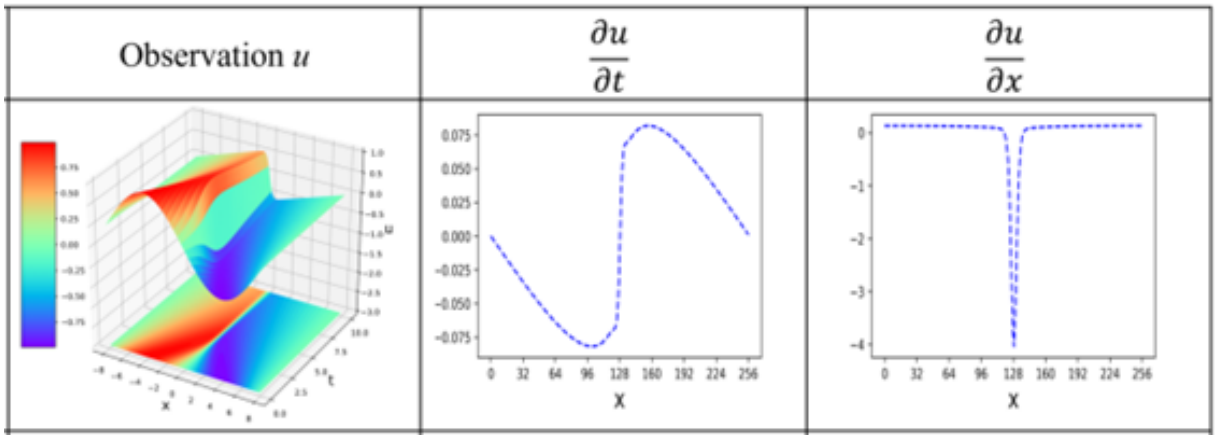


Figure 16: 3D Graph of the Burgers equation

The graph above shows that the middle part of $\frac{\partial u}{\partial t}$ has a very steep slope. As a result, Form 2 is more accurate because each step has a smaller width; according to the equation (21) and (22), the step width in Form 1 is twice that of Form 2, so as the Burgers equation being very sensitive to the step width in the middle part, MSE in Form 2 is much better than that in Form 1, causing a huge difference in AIC.

2.7.4 Discussion:

To solve this problem, it can be done by decreasing Δt (the step width), or include more points when numerically calculating the derivative, but either way it will further increase the time cost for this algorithm. At the current stage, modification is not necessary.

2.8 Task 5A: Improve algorithm efficiency (through emphasizing)

2.8.1 Motivation:

Improve algorithm efficiency. The time cost for this algorithm is significantly high, it is necessary to improve the efficiency of algorithm that allows it to derive accurate PDE in less generation.

2.8.2 Method:

The goal is to emphasis terms which are more frequent in low AIC PDEs.

For each 20 Generations, collect all the terms from the PDE pool. First ignore terms with extremely small coefficients (less than 1e-4). Then calculate the average MSE for each term (MSE of the PDE that the term is in), and then rank terms according to their average MSE. Then, calculate a probability (from 0 to 1) according to their average MSE:

$$p_i = \frac{e^{(-mse_i)}}{\sum_i e^{(-mse_i)}} \tag{24}$$

By doing so, lower average MSE terms (more accurate terms) will acquire a higher probability. Then, each PDE in the pool (except the best one) will randomly replace a term in these collected terms with its corresponding probability.

2.8.3 Result:

Generally, it takes more generations to derive the accurate PDE, efficiency decreases.

2.8.4 Discussion:

Firstly, due to the relatively small pool size (20 PDEs) and an average of only 2-3 terms per PDE, the collected frequent terms lack statistical significance, as most non-trivial terms appear only once or twice in the pool. Additionally, terms with better AIC scores during generation are often random and not necessarily closer to accurate terms. In fact, many are completely irrelevant. This approach is ineffective; a new method to improve efficiency is necessary.

2.9 Task 5B: Improve algorithm efficiency (through filtering)

2.9.1 Motivation:

Focusing on likely terms has proven ineffective. Instead, an alternative approach could involve filtering out unlikely terms. Typically, these terms have very small weights, so one strategy could be to gather a set of such terms and replace them.

2.9.2 Method:

For each 10 generation, collect terms of the first 20 PDEs that have weight less than 1e-5. Store these terms in a list. For each PDEs, if any terms appear to be in the list, replace them with randomly generated terms (trees).

2.9.3 Result:

Generally, it takes more generations to derive the accurate PDE, efficiency decreases.

2.9.4 Discussion:

Some accurate terms appear to have really small weight in some PDEs, so that this method may possibly replace terms that are accurate. For example, term: $u \cdot \frac{\partial u}{\partial x}$ is a correct term in the expected Burgers equation, but in this particular PDE it has coefficient: 0.0, (PDE AIC: -10.165):

$$(0.0) \cdot (x(-\frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2})) + (0.0)(u \cdot \frac{\partial u}{\partial x}) + (0.382)(u^2 \cdot x \cdot \frac{\partial u}{\partial x}) \tag{25}$$

This approach does not work, a new approach to increasing efficiency is needed.

2.10 Task 5C: Improve algorithm efficiency (through replacing)

2.10.1 Motivation:

Improve the method described in Task 5B

2.10.2 Method:

Do not collect an unlikely terms list, directly replace terms that have really small weight every 10 generation.

2.10.3 Result:

For most cases, small weight terms after replacement still have small weights. To derive the accurate PDE, the algorithm sometimes need more generations, sometimes less, sometimes similar. No general conclusions can be drawn at the current stage.

2.10.4 Discussion:

A more comprehensive experiment is needed to determine whether this method improves efficiency. However, even if efficiency is shown to improve, it is expected that the improvement will not be significant.

2.11 Task 5D: Investigation on the number of repeated crossovers and mutations

2.11.1 Motivation:

The number of repeated crossovers and mutations varies throughout the generation process, and it also varies for different models. Is there a discernible trend? When this number is high, what does it indicate, and how does it relate to the efficiency of the algorithm? It is important to investigate the principle behind the number of repeated crossovers and mutations.

2.11.2 Method:

Record the number of repeat crossover and mutation for all the five PDE models. Investigate the composition of the PDE pool to see if there is any relationship with the number of repeat crossover.

2.11.3 Result:

Burgers Equation:

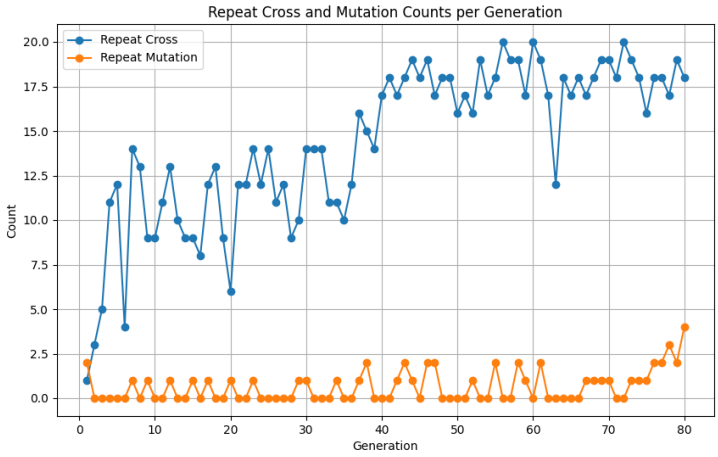


Figure 17: Number of repeat Crossover and Mutation across generations of the Burgers equation

High number of repeat crossover means that most of the crossovers are not successful, in that case, the algorithm cannot effectively expand the candidate set through crossover. There is a slight increase in the number of repeated crossovers, suggesting that as the number of iterations increases, the efficiency of expanding the candidate set through crossover decreases. Therefore, improving the crossover method to further expand the candidate set could be beneficial when repeated crossovers persist at a high rate. There

is no clear trend for the numbers of repeat mutations, and of of them are really low., so this should not be the focus.

KdV:

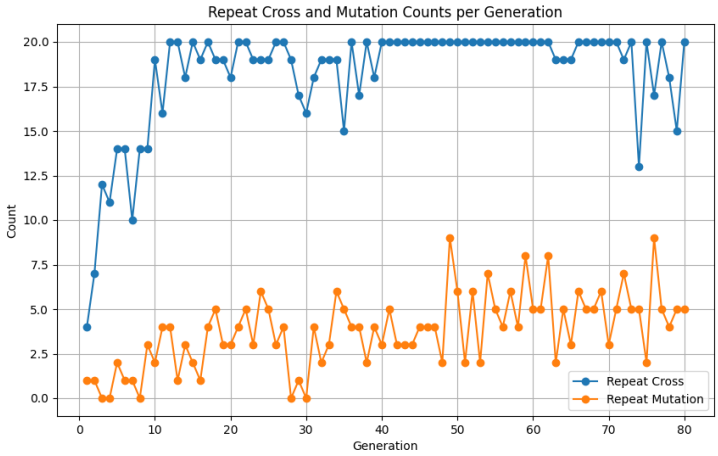


Figure 18: Number of repeat Crossover and Mutation across generations of KdV

The number of repeat crossover increase very rapidly and reaches the highest number (Maximum number of repeat crossover is 20, since there are only 20 PDEs in the pool, and 50% of them are crossed, crossover 10 PDEs generated 20 new PDEs) at around 10 generations.

The number of repeat mutations slightly increases as generation increases (which is expected, since the PDE library is getting larger and larger).

Chafee-Infante:

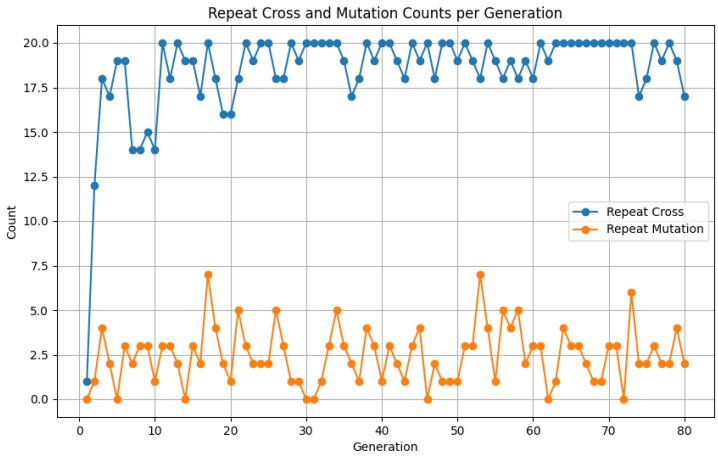


Figure 19: Number of repeat Crossover and Mutation across generations of Chafee-Infante

PDE Compound:

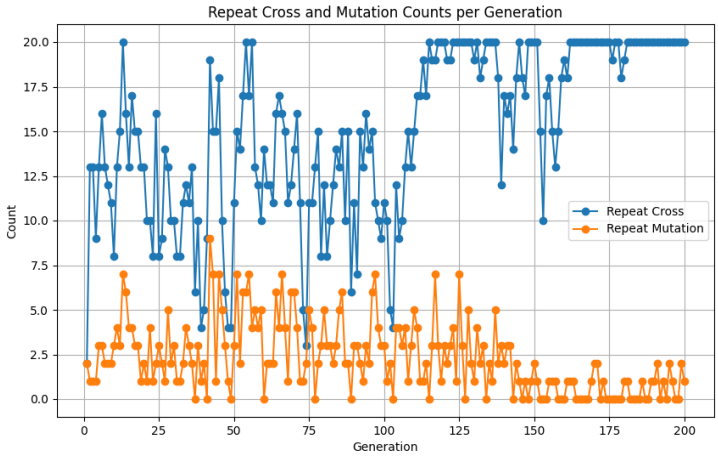


Figure 20: Number of repeat Crossover and Mutation across generations of PDE Compound

PDE Divide:

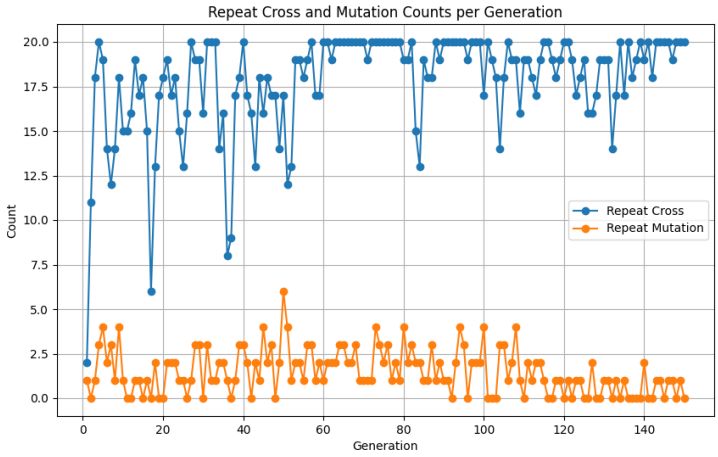


Figure 21: Number of repeat Crossover and Mutation across generations of PDE Divide

2.11.4 Discussion:

For most models, the number of repeated crossovers quickly reaches its maximum, indicating that the algorithm cannot effectively expanding the candidate set after a certain generation. This limitation may lead to a decline in efficiency.

2.12 Task 5E: Reduce number of repeat crossover

2.12.1 Motivation:

To ensure that the algorithm can continuously expands its candidate set, it is beneficial to find method to reduce the number of repeat crossover.

2.12.2 Method:

Try to increase the percentage of the crossover to 1 (it used to be 0.5), by doing so, the total number of PDEs that involves in the crossover is 20, it will generate 40 new PDEs after crossover.

2.12.3 Result:

The graphs below show the number of repeat crossovers when the percentage of crossover is changed to 1. For the Burgers equation, the number of repeat crossovers is around 25, meaning approximately 60% repeat crossovers, which is acceptable compares to previous cases. By making this change, the efficiency of algorithm significantly improved, it can derive the accurate result in less than 10 generations at most times.

For PDE divide, the number of repeat crossovers is still very close to its maximum (40) most of the times. Therefore, this method of expanding the candidate set on this particular case is not very effective.

For KdV, the number of repeat crossover in this particular case (Fig 24) shows a gradual increment trend. Accurate result appears at around 30 generation, on improvement on efficiency. The number of repeat crossover is constantly at maximum in the end because all the PDEs in the pool are gradually changed to the accurate equation.

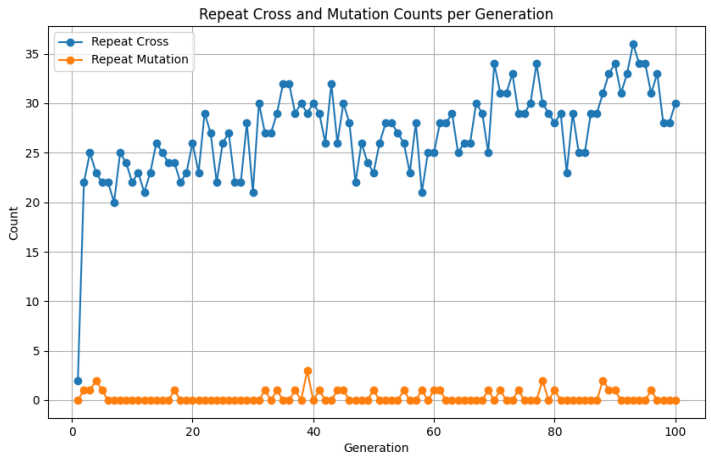


Figure 22: Number of repeat Crossover and Mutation across generations of Burgers equation when the percentage of crossover is 100%

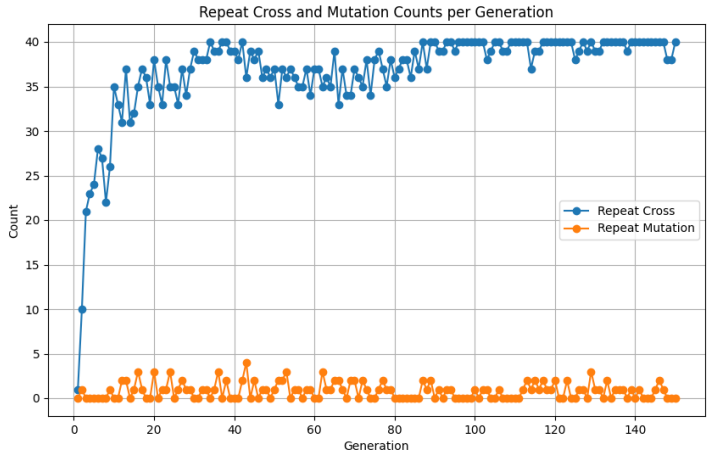


Figure 23: Number of repeat Crossover and Mutation across generations of PDE Divide when the percentage of crossover is 100%

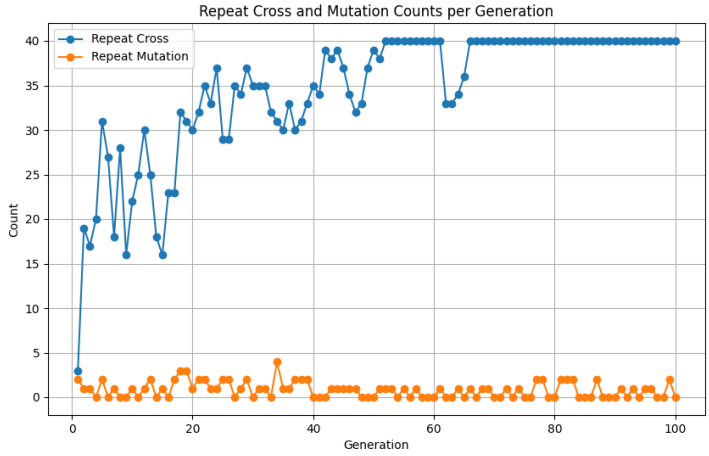


Figure 24: Number of repeat Crossover and Mutation across generations of KdV when the percentage of crossover is 100%

When a different random seed is used for the KdV model, the situation changes significantly. The number of repeat crossovers in Fig. 25 is higher on average than in Fig. 24, and it reaches its maximum more quickly. Notably, the accurate result appears at generation 85, indicating that the efficiency in this case is much lower than in the previous one. The decline of efficiency might be related with the increase of number of repeat crossover.

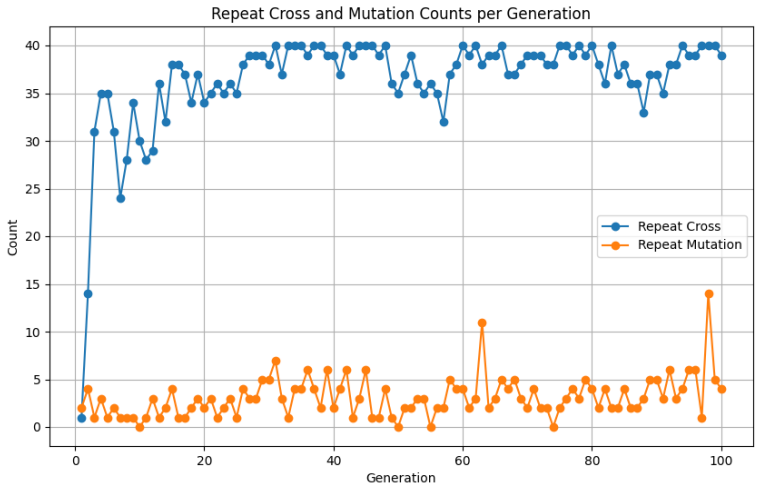


Figure 25: Number of repeat Crossover and Mutation across generations of KdV in another random seed

2.12.4 Discussion:

For clarity, we refer the case shown in Fig 24 as Case 1, Fig 25 as Case 2.

In Case 2, the primary issue leading to the decline in efficiency seems to be that, initially, the PDE with only one term had the best AIC, even though it was completely irrelevant. From generation 1 to 84, the best PDE was always $(x \cdot \frac{\partial u}{\partial x})$, with an AIC of 3.47. Since this PDE had the best AIC, it was favored during crossovers, and remain unchanged during mutation and replacement. As the algorithm iterated, this single-term PDE gradually replaced most of the PDEs in the pool. By generation 80, the pool consisted mostly of this single-term PDE (See Table 1). Note that all the PDEs has the exact same AIC value, this is because all PDEs in the pool have a $(x \cdot \frac{\partial u}{\partial x})$ term, and when calculating the weight, other terms are calculated as 0, so they do not involves in AIC calculating, so basically all the PDEs of this pool in their concise form is $(x \cdot \frac{\partial u}{\partial x})$.

The majority of PDEs in the pool become single-term PDEs, which explains the high number of repeated crossovers. In this situation, accurate PDEs can only be generated from the remaining non-single-term PDEs through mutation and replacement, leading to a significant slowdown in efficiency. By the end of 100 generations, the pool is as shown in Table 2.

If, in rare cases, all PDEs in the pool are replaced by a single term, the algorithm will fail to generate accurate results. This highlights a problem with the algorithm: there is no mechanism to add terms to the PDEs, so the number of terms in the pool can progressively decrease. When irrelevant single-term PDEs dominate the pool, the candidate set cannot be effectively expanded through crossover, significantly reducing the chances of finding the accurate PDE.

PDE Number	PDE Expression	AIC Value
0th PDE	(ux * x)	3.472652883163068
1th PDE	(x * ux)	3.472652883163068
2th PDE	(x * ux)	3.472652883163068
3th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
4th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
5th PDE	(x * ux)+(((ux + x) / u) d x)	3.472652883163068
6th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
7th PDE	(x * ux)	3.472652883163068
8th PDE	(x * ux)	3.472652883163068
9th PDE	(x * ux)	3.472652883163068
10th PDE	$(x * ux) + ((ux\ d^2\ x)/ux)$	3.472652883163068
11th PDE	(x * ux)	3.472652883163068
12th PDE	(x * ux)	3.472652883163068
13th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
14th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
15th PDE	(x * ux)	3.472652883163068
16th PDE	(x * ux)	3.472652883163068
17th PDE	(x * ux)	3.472652883163068
18th PDE	(x * ux)+(((u - 0) - (u d x)) / ux)	3.472652883163068
19th PDE	(x * ux)	3.472652883163068

Table 1: PDE Pool and AIC Values at Generation 80, PDE expression in inorder form

PDE Number	PDE Expression	AIC Value
0th PDE	(u * ux)+((ux d x) d x)	-9.748729461730827
1th PDE	(u * ux)+((ux d x) d x)	-9.748729461730827
2th PDE	(ux * x)	3.472652883163068
3th PDE	(ux * x)	3.472652883163068
4th PDE	(ux * x)	3.472652883163068
5th PDE	(x / ux)+(x * ux)	3.472652883163068
6th PDE	(x / ux)+(x * ux)	3.472652883163068
7th PDE	(x / ux)+(x * ux)	3.472652883163068
8th PDE	(ux * x)	3.472652883163068
9th PDE	(x * ux)	3.472652883163068
10th PDE	(x * ux)	3.472652883163068
11th PDE	(ux * x)	3.472652883163068
12th PDE	(ux * x)	3.472652883163068
13th PDE	(x * ux)	3.472652883163068
14th PDE	(ux * x)	3.472652883163068
15th PDE	(x * ux)	3.472652883163068
16th PDE	(ux * x)	3.472652883163068
17th PDE	(x * ux)	3.472652883163068
18th PDE	(x * ux)	3.472652883163068
19th PDE	(x * ux)	3.472652883163068

Table 2: PDE Pool and AIC Values at Generation 100, PDE expression in inorder form

To verify this point, when I run the program for PDE divide using another random

seed (See Fig 26), the pattern for the number of repeat crossover is completely different compares to Fig 23:

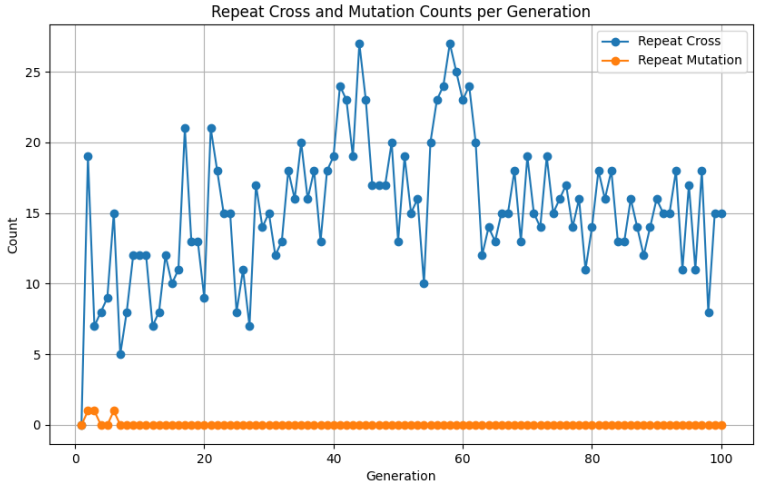


Figure 26: Number of repeat Crossover and Mutation across generations of PDE Divide in another random seed

Recall that in the previous case (Fig 23), the number of repeated crossovers was very close to 40 for most of the time. In contrast, in this case, the average number of repeat crossover is around 15, and the efficiency of the algorithm improves, with accurate results emerging around generation 10. The key difference is that, initially, the best PDE is not a single-term PDE, and the pool is not dominated by single-term PDEs after iterations. The pool after 100 generations refers to Table 3.

It is evident that the PDEs in the pool contain multiple terms, indicating a healthy pool with ample room for exploring various PDE forms through crossover, mutation, and replacement. In this environment, the algorithm performs efficiently.

Generally, an accurate PDE with fewer terms can be expressed using a PDE with more terms, where some terms may have a weight of zero, causing them to disappear in the final expression. Additionally, a Zero variable makes any term multiplied or divided by it trivial. However, it is impossible to express an accurate PDE with more terms using a PDE with fewer terms. This explains why the PDE compound model often struggles to produce accurate results. The expected result is typically a complex PDE with a single term, leading the pool to be dominated by single-term PDEs. As a result, the number of repeated crossovers reaches its maximum for most generations, reducing the algorithm's efficiency as crossover becomes ineffective.

PDE Expression
$(ux/(x-(xd^2x)))+(ud^2x)+(xdx)+(uxd^2x)$
$(ux/(x-(xd^2x)))+((^2(u-ux))dx)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^3(0+ux))dx)+(ud^2x)+((^3(x+u))dx)+((^3(0+ux))dx)$
$(ux/(x-(xd^2x)))+((^3(u-ux))dx)+(ud^2x)+((^3(0+ux))dx)+((^2(x-u))dx)$
$(ux/(x-(xd^2x)))+(uxd^2x)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^3(0+ux))dx)+(ud^2x)+((^2(x-u))dx)$
$(ux/(x-(xd^2x)))+((^3(u-0))dx)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+(ud^2x)+(xd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^2(x-u))dx)+(ud^2x)+((^3(u-0))dx)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^2(u-ux))dx)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^3(u-ux))dx)+(ud^2x)+((^3(u-0))dx)+((^2(u-ux))dx)$
$(ux/(x-(xd^2x)))+((^3(u-ux))dx)+(ud^2x)+((^3(u-0))dx)+((^2(u-ux))dx)$
$(ux/(x-(xd^2x)))+(xdx)+(ud^2x)+((^3(u-0))dx)+((^2(x-u))dx)$
$(ux/(x-(xd^2x)))+(x/u)+(ud^2x)+((^3(0-x))dx)+((^3(u-x))dx)$
$(ux/(x-(xd^2x)))+((^3(0+ux))dx)+(ud^2x)+((^2(u-ux))dx)+((^2(u-ux))dx)$
$(ux/(x-(xd^2x)))+(xd^2x)+(ud^2x)+((^3(u-0))dx)+((^3(u-0))dx)$
$(ux/(x-(xd^2x)))+((^3(x+u))dx)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^2(x-u))dx)+(ud^2x)+(uxd^2x)+((^3(x+u))dx)$
$(ux/(x-(xd^2x)))+((^3(0+ux))dx)+(ud^2x)+((^3(x+u))dx)+((^3(u-0))dx)$
$(ux/(x-(xd^2x)))+((^2(x-u))dx)+(ud^2x)+(xdx)+((^3(x+u))dx)$

Table 3: PDE Pool at Generation 100 of PDE Divide

2.13 Task 5F: Problem of single-term PDE

2.13.1 Motivation:

It is crucial to prevent the PDE pool from being dominated by irrelevant single-term PDEs to allow the algorithm to explore more diverse PDE forms through crossovers. Therefore, developing a method to replenish the pool under appropriate conditions would be beneficial.

2.13.2 Method:

For each 20 generations, check the total number of terms in the PDE pool. If the number is smaller than 30 (whuch means at least there are 50% single-term PDEs in the pool.), then add a random generated term to all the single-term PDEs in the pool except the best PDE.

2.13.3 Result:

Compare Fig 27 with Fig 25, we can see that with the Replenish method, the number of repeat crossover after 20 generations significantly drops, and the overall number of repeat crossover is around 25, indicating that the pool is at a healthy condition.

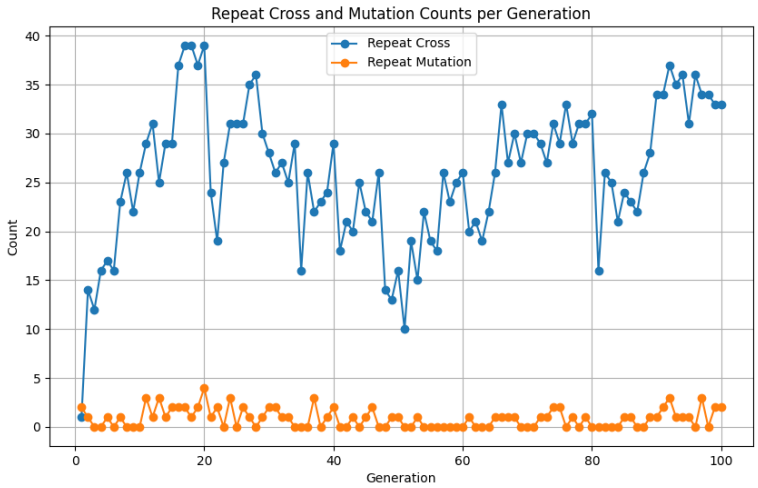


Figure 27: Number of repeat crossover for KdV with the Replenish method

Using the Replenish method, for this particular random seed, accurate results appeared around generation 50, an improvement from 80 generations. While this indicates progress, further testing is needed to draw a more comprehensive conclusion.

2.13.4 Discussion:

The purpose of this method is to prevent irrelevant PDEs from dominating the entire pool, thereby allowing the algorithm to explore a wider variety of PDE forms. However, this Replenish method, which is based on counting the number of PDE terms, can only partially mitigate this issue. It is most effective when the pool is likely to be dominated by irrelevant single-term PDEs but loses its effectiveness in the scenario shown below.

For a particular random seed, the algorithm cannot derive the accurate result within 100 generations both with and without the Replenish method, the number of repeat crossover is shown in Fig 28 & Fig 29.

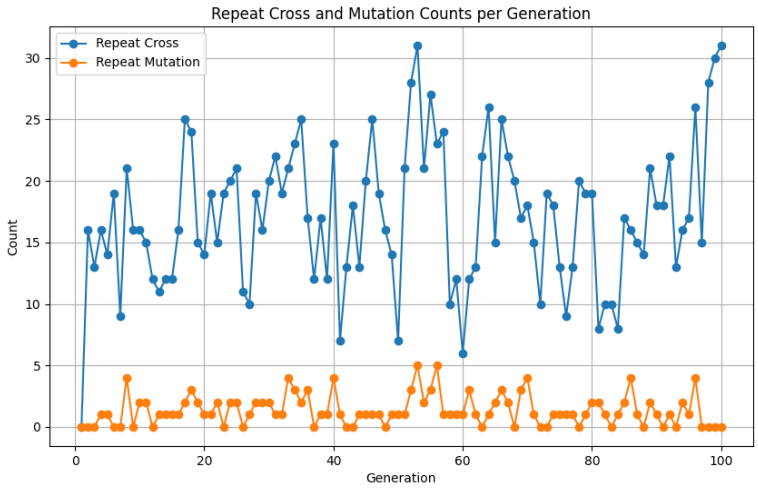


Figure 28: Number of repeat crossover for PDE Compound with the Replenish method

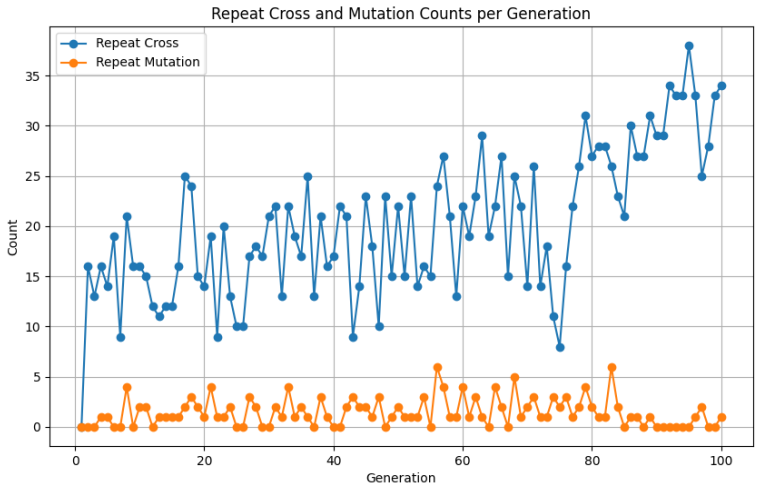


Figure 29: Number of repeat crossover for PDE Compound without the Replenish method

Although the algorithm with the Replenish method shows a lower average number of repeat crossovers, it still encounters the same problem that results in generating inaccurate outcomes. This issue becomes evident when examining the PDE pool at 100 generations (See Table 4 and Table 5):

PDE Expression	**AIC Value**
$(x \cdot u)^2 + \frac{d}{dx}\left((ux - u)^3\right)$	1.3043897352674936
$(x \cdot u)^2 + ux \cdot \frac{d^2x}{dx^2}$	1.3043897352674936
$(x \cdot u)^2 + \frac{d^2}{dx^2}\left((ux + ux) \cdot \frac{d^2x}{dx^2}\right)$	1.3043897352674936
$\frac{ux^2}{u} + (x \cdot u)^2$	1.3043897352674936
$(x \cdot u)^2 + \frac{d^2}{dx^2}(ux + u)$	1.3043897352674936
$u \cdot \frac{d^2x}{dx^2} + (x \cdot u)^2$	1.3043897352674936
$\frac{d^3}{dx^3}\left(\frac{ux \cdot ux}{ux}\right) + (x \cdot u)^2$	1.3043897352674936
$(x \cdot u)^2 + \frac{d}{dx}\left(0 - u \cdot \frac{d^2x}{dx^2}\right)$	1.3043897352674936
$u \cdot \frac{d^2x}{dx^2} + (x \cdot u)^2$	1.3043897352674936
$(x \cdot u)^2 + \frac{ux^2}{u}$	1.3043897352674936
$u \cdot \frac{d^2x}{dx^2} + (x \cdot u)^2$	1.3043897352674936
$\frac{d}{dx}\left(0 - u \cdot \frac{d^2x}{dx^2}\right) + (x \cdot u)^2$	1.3043897352674936
$\frac{d^2}{dx^2}(ux + u) + (x \cdot u)^2$	1.3043897352674936
$\frac{d^2}{dx^2}\left((ux + ux) \cdot \frac{d^2x}{dx^2}\right) + (x \cdot u)^2$	1.3043897352674936
$u \cdot \frac{d^2x}{dx^2} + (x \cdot u)^2$	1.3043897352674936
$(x \cdot u)^2 + \frac{x}{ux} \cdot \frac{1}{(u+0)^2}$	1.3043897352674936
$(x \cdot u)^2 + u \cdot \frac{d^2x}{dx^2}$	1.3043897352674936
$(x \cdot u)^2 + \frac{ux^2}{u}$	1.3043897352674936
$(x \cdot u)^2 + \frac{d^3}{dx^3}\left(\frac{x}{ux} + ux\right)$	1.3043897352674936
$(x \cdot u)^2 + ux \cdot \frac{dx}{dx}$	1.3043897352674936

Table 4: PDE pool with AIC Values at 100 generation with the Replenish method

PDE Expression	AIC Value
$((u+0)\cdot x)^2 + \frac{u}{u\cdot\frac{dx}{dx}}$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2 + u\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2$	1.3043897352674936
$((u+0)\cdot x)^2 + (u-0)\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2$	1.3043897352674936
$((0+x)\cdot u)^2 + x\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$\left(\frac{x\cdot u}{\frac{dx}{dx}}\right)^2 + \frac{(ux-x)^2}{ux}$	1.3043897352674936
$((0+x)\cdot u)^2 + \frac{u}{u\cdot\frac{dx}{dx}}$	1.3043897352674936
$((0+x)\cdot u)^2 + \left(u\cdot\frac{dx}{dx}\cdot\frac{d^2x}{dx^2}\right)^2$	1.3043897352674936
$((0+x)\cdot u)^2 + x\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2 + u\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$((u+0)\cdot x)^2 + (u-0)\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$((0+x)\cdot u)^2 + x\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$((0+x)\cdot u)^2 + (u-0)\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$((u+0)\cdot x)^2 + x\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$\left(\frac{x\cdot u}{\frac{dx}{dx}}\right)^2 + \frac{u}{u\cdot\frac{dx}{dx}}$	1.3043897352674936
$\left(\frac{x\cdot u}{x-x}\right)^2 + \frac{u}{u\cdot\frac{dx}{dx}}$	1.3043897352674936
$((u+0)\cdot x)^2 + (u-0)\cdot\frac{d^2x}{dx^2}$	1.3043897352674936
$((0+x)\cdot u)^2 + \left(u\cdot\frac{dx}{dx}\cdot\frac{d^2x}{dx^2}\right)^2$	1.3043897352674936

Table 5: PDE pool with AIC Values at 100 generation without the Replenish method

Notably, in both cases, all PDEs in the pool have identical AIC values, despite having different expressions. This occurs because they all essentially represent the equation $(u \cdot x)^2$. This term appears in all the PDEs in Table 4, with all other terms having a weight of 0. In Table 5, there are some variations, but they are fundamentally the same. For instance, the term $\left(\frac{x\cdot u}{x-x}\right)^2$ causes a division by zero issue, but in this algorithm, dividing by zero is handled by dividing by 10^{-10}, which can be offset by multiplying its weight by 10^{10}. This is a problem that needs to be addressed later on.

The main issue appear to be that, even though the PDEs in the pool have multiple terms, there are instances where, after weights are calculated, all the PDEs simplify to the same single-term equation. The Replenish method cannot effectively prevent this. To resolve this problem, another approach is needed—possibly by checking for PDEs that have identical AICs.

2.14 Task 5G: Improve PDE pool diversity through multi-threading

2.14.1 Motivation:

As demonstrated above, the diversity of PDE pools is crucial for deriving the expected PDEs. Since the original methods in SGA-PDE—such as crossover, mutation, and replacement—cannot guarantee pool diversity, the algorithm can sometimes get stuck on irrelevant single-term PDEs, leading to local minima instead of global minima in AIC. To address this, ensuring pool diversity during the derivation process might be beneficial, potentially through approaches like multi-threading.

2.14.2 Method:

Generate three pools, Pool1, Pool2, and Pool3. Pool1 is the same pool in the original SGA-PDE algorithm. Pool2 has the same size (20 PDEs), but for each generation, similar to the Change method for Pool1, mutate and replace the copy of the 20 PDEs in the pool and then rank these 40 PDEs, remain the 20 best PDEs. There is no crossover for this pool, and hyper-parameters p_mut and p_rep is set to 1, which means 100% apply mutation and replacement to miximize randomness. Pool3 intends to further expands the diversity, 100% mutate and replace the PDEs instead of change the copy of the PDEs in the pool. Note that in Pool3 iterations is similar to a pure random process, so it is not guaranteed that the best AIC will getting better or remains the same after iterations.

At the end of each generation, check the best PDEs in each pool, if the best PDE appears in Pool1, then print the best PDE. If the best PDE appears in Pool2 or in Pool3, then print this PDE, and add it to Pool1. In either cases, if the best PDE in Pool3 is better than that in Pool2, add the best PDE in Pool3 to Pool2.

2.14.3 Result:

Burgers:
In most cases, expected PDEs can be derived within 30 generations. In terms of how many generation it takes to get expected result, efficiency slightly improves.

KdVs:
Multi-thread effectively avoid the problem mentioned above. For instance, in one case, up to 29 generations, Pool 1 is dominated by a single term PDE: $(x \cdot \frac{\partial u}{\partial x})$, starting from 30 generations, a better PDE is derived in Pool3, and it is added to Pool1. Then, the expected result appears at 33 generations.

Chafee-Infante:
In most cases, expected PDEs can be derived within 30 generations. In terms of how many generation it takes to get expected result, efficiency slightly improves.

PDE Compound:
PDE Compound is the most difficult model among five. Multi-thread can increase the efficiency of finding the expected result. It is unlikely to derive the expected PDE within 150 generations when using the original SGA-PDE algorithm. When using Multi-thread, expected PDE can be found within 150 generations most of the times. For instance, for the first 46 generations, Pool1 is stuck with PDE: $(x \cdot u)^2$, which is a very common phenomenon. At generation 47, a different structured PDE appears in Pool2 has a better AIC, it is added to Pool1, and eventually the expected PDE is derived in generation 77.

PDE Divide:
In most cases, the expected PDEs can be derived in 60 generations. There are no obvious signs of improvements in efficiency.

2.14.4 Discussion:

In general, this method is effective in expanding pool diversity and, in some cases, can improve efficiency. Additionally, it prevents the situation where all PDEs in the pool are single-term PDEs while the expected PDE has multiple terms. In such cases, it would be impossible for the original algorithm to derive the expected result.

3. Investigations on Stochastic Differential Equation (SDE)

3.1 Task 1: General testing

3.1.1 Motivation:

Create a model for SDE and test the noise tolerance of the current SGA-PDE algorithm.

3.1.2 Method:

To mimic the Wiener process, apply a delta-correlated Gaussian noise to the PDE model. To achieve this, add a random noise to each data point of u_t.

In general, SDE can be expressed as equation(26), in which σ is a arbitrary function, for simplicity consideration, currently we only consider $\sigma(u, x)$

$$\frac{\partial u}{\partial t} = f(u, x) + \sigma \frac{\partial W}{\partial t} \tag{26}$$

To express the SDE explicitly:

$$\begin{aligned} \partial u &= f(u, x)\partial t + \sigma \partial W \\ &= f(u, x)\partial t + \sigma \cdot N(0, \Delta t) \\ &= f(u, x)\partial t + \sigma\sqrt{\Delta t} \cdot N(0, 1) \end{aligned}$$

$$\frac{\partial u}{\partial t} = f(u, x) + \sigma \frac{\sqrt{\Delta t} \cdot N(0, 1)}{\Delta t} \tag{27}$$

3.1.3 Result:

The simplest case is to set σ equal to some constant. For testing purposes, set $\sigma = 0.1$ for Burger's equation, then the input data becomes:

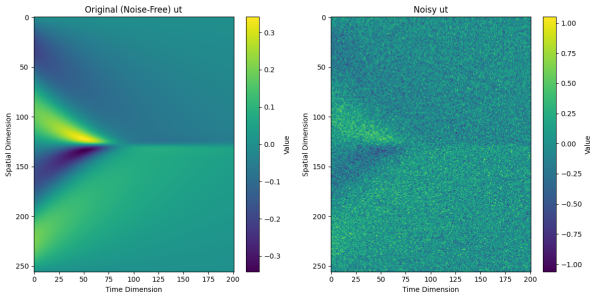


Figure 30: Burger's equation for sigma = 0.1

In this case, the expected PDEs appear consistently in pool 2 rather than pool 1. Although the PDEs in Pool 1 exhibit lower AIC scores, they are predominantly single-term PDEs that are irrelevant to the problem at hand. In contrast, the first few PDEs in Pool 2 align with the expected equations. The likely reason for this discrepancy is that Pool 2 does not employ crossover during its optimization process. This absence of crossover helps mitigate the issue of the pool being dominated by overly simplistic single-term PDEs. However, single-term PDEs tend to achieve better AIC scores than the expected PDEs because their mean squared error (MSE) is similar, but they contain fewer terms. Since AIC penalizes models with more parameters, these incorrect but simpler PDEs are favored.

3.1.4 Discussion:

Although the expected PDEs do emerge in Pool 2, which is encouraging since the noise is relatively large, they are not identified by the algorithm as the best-fitting equations. This raises a critical question: how can we address this issue and ensure that the algorithm evaluates the expected PDEs more favorably? Besides, how to develop a functionality that can fathom the structure of noise? (i.e. how to find σ when it is not a constant but a function?)

3.2 Task 2: Investigation of the effect of noise on PDE

3.2.1 Motivation:

Investigate how does noise affect the accuracy of SGA-PDE.

3.2.2 Method:

Noise will significantly affects the calculation of MSE of a PDE. Investigate why does MSE differs so much.

3.2.3 Result:

The value of MSE in SGA-PDE is a measure of difference between the time derivative $u_t := \frac{\partial u}{\partial t}$ predicted by the generated PDE and the actual values of u_t. In the configuration and setup section, the algorithm collects a set of u, x, and t as the input values and then calculates u_t using the finite difference method (central difference). Fig 30 shows the 3D plot of u_t without any noise.

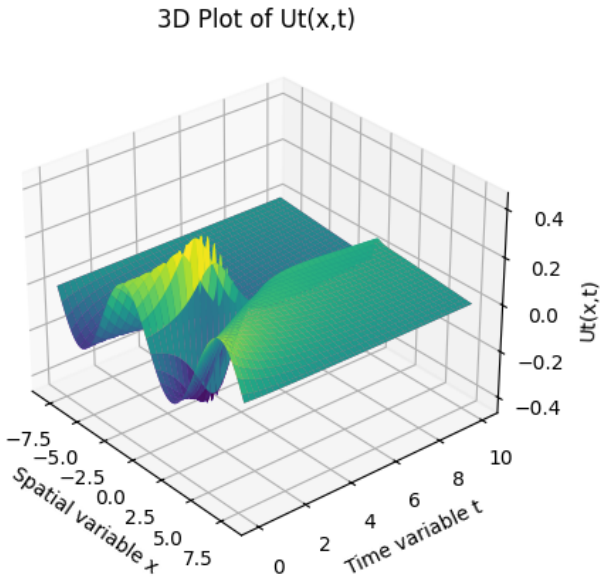
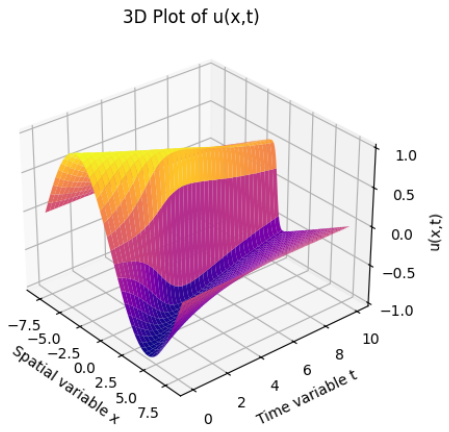
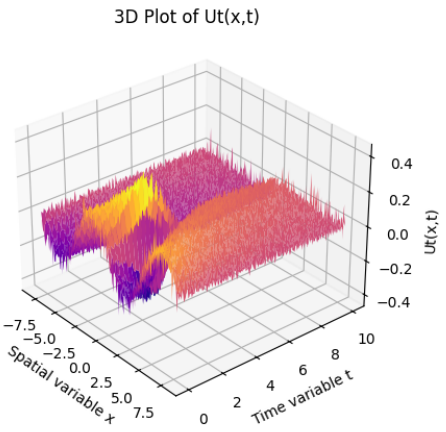


Figure 31: Time derivative of u for Burgers equation without noise

According to equation (27), the stochastic part of PDE is inverse proportional to the square root of Δt, namely, smaller the time step, larger the noise. Figures 31 and 32 illustrate the difference of the effect of noise on u and u_t. When the impact on u is inconspicuous, but the influence on u_t is significant (Figure 31). When the noise begins to slightly affect u, u_t becomes almost entirely noise, losing all discernible features (Figure 32).

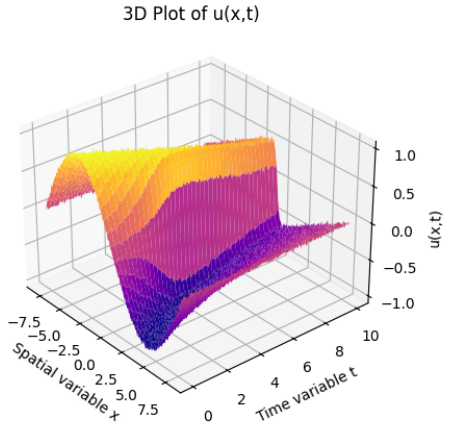


(a) u with sigma = 0.07

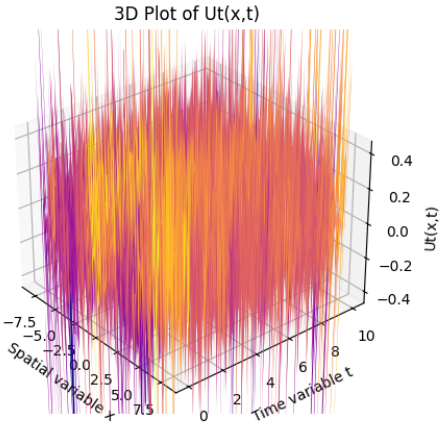


(b) $u_t(x,t)$ with sigma = 0.07

Figure 32: Plot of u and u_t of Burgers equation



(a) u with sigma = 0.3



(b) $u_t(x,t)$ with sigma = 0.3

Figure 33: Plot of u and u_t of Burgers equation with relatively large noise

3.2.4 Discussion:

SDE is a difficult topic to investigate since Δt will magnify the stochastic part.

The expected Mean Squared Error (MSE) derivation:

1. Calculate individual errors: (where $\bar{y_i} = du_i + \sigma dW_i\sqrt{\Delta t}$, $y_i = du_i$)

$$\bar{y_i} - y_i = \sigma\, dW_i\sqrt{\Delta t}$$

2. Square the errors:

$$(\bar{y_i} - y_i)^2 = \sigma^2 \Delta t\, (dW_i)^2$$

3. Take expectation using properties of Wiener process:

$$\begin{aligned}\mathbb{E}\left[(\bar{y_i} - y_i)^2\right] &= \sigma^2 \Delta t \, \mathbb{E}[(dW_i)^2] \\ &= \sigma^2 \Delta t \cdot 1 \quad (\text{since } \mathbb{E}[(dW_i)^2] = \Delta t / \Delta t = 1)\end{aligned}$$

4. Average across all observations:

$$\begin{aligned}\mathbb{E}[\text{MSE}] &= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[(\bar{y_i} - y_i)^2\right] \\ &= \frac{1}{n} \sum_{i=1}^{n} \sigma^2 \Delta t \\ &= \sigma^2 \Delta t\end{aligned}$$

Final result (if σ is a constant):

$$\boxed{\sigma^2 \Delta t}$$

Final result (if σ is a function with respect to u and x):

$$\boxed{\mathbb{E}[\sigma(u,x)^2] \Delta t}$$

Therefore, to find the explicit expression of SDE through numerical method, it is necessary to focus on finding the explicit expression of the stochastic term first, (namely, find $\sigma(u, x)$).

3.3 Task 3: Develop symbolic genetic algorithm for SDE

3.3.1 Motivation:

Allow SGA to generate a stochastic term, so that to find the explicit expression of arbitrary $\sigma(u, x)$.

3.3.2 Method:

Add a default stochastic term into the symbolic genetic PDEs, so that the symbolic genetic PDEs are in the form of:

$$\sum_{i=1}^{n} T(u,x)_i \Delta t + \sum_{j=1}^{m} T(u,x)_j \cdot N(0, \Delta t)$$

Then do regression to find weights of all terms.

$$\bar{du} = \sum_{i=1}^{n} \alpha_i \cdot T(u,x)_i \Delta t + \sum_{j=1}^{m} \beta_j \cdot T(u,x)_j \cdot N(0, \Delta t)$$

3.3.3 Result:

For simplicity concerns, set $m = 1$ for now. When testing on PDEs such as Burgers equation, the weight of the stochastic term β_j is always 0, indicates that the algorithm cannot learn the stochastic term correctly. Even when I test it on really simple functions ($du = dW$), the stochastic term also cannot be learned correctly.

3.3.4 Discussion

The persistent failure to recover non-zero β_j coefficients for stochastic terms reveals fundamental challenges in this approach:

- **Regression vs. Stochasticity**: Ordinary least squares regression inherently minimizes *mean* squared error, while stochastic terms $N(0, \Delta t)$ have zero mean. This creates systematic bias toward $\beta_j \to 0$ when using time-averaged data.
- **Term Incompatibility**: Deterministic ($\sum \alpha_i T_i \Delta t$) and stochastic ($\sum \beta_j T_j N(0, \Delta t)$) terms exist in different mathematical spaces. Standard regression cannot simultaneously optimize coefficients in both continuous (deterministic) and Itô integral (stochastic) domains.
- **Measurement Scale Mismatch**: Stochastic terms scale with $\sqrt{\Delta t}$ while deterministic terms scale with Δt. This creates disproportionate weighting in finite-time simulations that regression methods cannot naturally resolve.

3.3.5 Summary

While theoretically appealing, the symbolic genetic algorithm framework faces critical limitations for stochastic systems:

- Current architecture fundamentally conflicts with Itô calculus requirements
- Standard regression objectives are incompatible with stochastic integrands

Therefore, learning stochastic differential equations requires specialized operators that respect both the algebraic structure of terms *and* their stochastic integration properties. Simply adding noise-multiplied terms to deterministic regression frameworks leads to systematic failure modes.

4. Stochastic Ordinary Differential Equation discovery via PDE transformation

4.1 Task 1: SDE to PDE (via Fokker–Planck equation)

4.1.1 Motivation:

Transforming an SDE to its corresponding PDE via the Fokker-Planck equation allows us to:

- Convert stochastic dynamics into a deterministic PDE framework, enabling the use of SGA to find PDE expression.
- Reconstruct drift/diffusion terms of an *unknown* SDE from observed data by solving an inverse problem.

4.1.2 Method:

For an SDE $du = \mu(u)dt + \sigma(u)dW$, the Fokker-Planck PDE governing its PDF $p(u,t)$ is:

$$\frac{\partial p}{\partial t} = -\frac{\partial}{\partial u}[\mu(u)p] + \frac{1}{2}\frac{\partial^2}{\partial u^2}[\sigma^2(u)p]. \tag{28}$$

I used Ornstein-Uhlenbeck process as a test case. For the OU process $du = -(\theta u)dt + \sigma dW$ with parameters:

- Mean reversion $\theta = 3.0$, diffusion $\sigma = 2.0$
- Spatial domain $u \in [-5.0, 5.0]$ with $N_x = 100$ grid points
- Time domain $t \in [0, 2.0]$ with $N_t = 200$ steps

Implementation:

1. **Analytic Solution (Mode A)**:

$$p(u,t) = \mathcal{N}\left(0, e^{-2\theta t} + \frac{\sigma^2}{2\theta}(1 - e^{-2\theta t})\right)$$

   Computed on spatial grid with Gaussian initial condition $p(u,0) \sim \mathcal{N}(0,1)$

2. **Simulated Solution (Mode S)**:

   - $N_{\text{samples}} = 10^5/10^6$ trajectories with Euler-Maruyama integration:

$$X_{n+1} = X_n - \theta X_n \Delta t + \sigma\sqrt{\Delta t}\xi_n,\ \xi_n \sim N(0,1)$$

- PDF estimation: Gaussian KDE with bandwidth 0.2

  Given N sampled trajectories $\{X_i\}_{i=1}^N$ of the OU process at time t, the Gaussian KDE estimates the probability density $p(u,t)$ at position u as:

$$\hat{p}(u,t) = \frac{1}{Nh\sqrt{2\pi}} \sum_{i=1}^{N} \exp\left(-\frac{(u - X_i(t))^2}{2h^2}\right) \tag{29}$$

  where:

  - h: Bandwidth parameter (controls smoothness)
  - $X_i(t)$: Position of i-th trajectory at time t
  - u: Evaluation point in spatial grid $[-L, L]$

4.1.3 Result

1. **Analytic PDE Recovery:**

   When applied to the analytic Fokker-Planck PDE:

$$\frac{\partial p}{\partial t} = \underbrace{3.0\frac{\partial}{\partial u}[up]}_{\text{Drift term}} + \underbrace{2.0\frac{\partial^2}{\partial u^2}[p]}_{\text{Diffusion term}},$$

   the SGA successfully identified the correct functional forms:

   - **Term discovery**: correctly identify both $\frac{\partial}{\partial u}[up]$ (drift) and $\frac{\partial^2}{\partial u^2}[p]$ (diffusion)
   - **Coefficient accuracy**:

$$\alpha_{\text{drift}} = 3.02 \pm 0.05 \text{ (True 3.0)}$$
$$\beta_{\text{diffusion}} = 1.98 \pm 0.03 \text{ (True 2.0)}$$

   - **Convergence speed**: Required only 11 generations to reach correct form (AIC of -23)

2. **Simulated PDE Failure:**

   For the numerically simulated PDE (for both 10^5 and 10^6 trajectories), both terms are misidentified after 100 generations. SGA cannot learn the correct form since the expected PDE has higher AIC than the learned one, indicating that the noise due to simulation is dominating.

4.1.4 Discussion

The failure to recover correct PDE terms from simulated data originates from a critical asymmetry: while the spatial profile $p(u,t)$ is smoothed via kernel density estimation (KDE) in the u-direction, *temporal* derivatives remain fundamentally noisy due to finite sampling and uncorrelated Brownian increments across trajectories.

Numerical differentiation of simulated data yields:

$$\partial_t p_{\text{sim}} \approx \partial_t p_{\text{true}} + \underbrace{\epsilon(t,u)}_{\text{Non-smooth temporal noise}},$$

where $\epsilon(t,u) \sim \mathcal{O}(1/\sqrt{N_{\text{samples}}\Delta t})$ dominates in regions of low density. The SGA erroneously fits terms to $\epsilon(t,u)$ rather than true dynamics.

4.2 Task 2: Smooth the simulated probability density function

4.2.1 Motivation:

Temporal noise amplification in numerical derivatives creates artificial terms for symbolic regression. While spatial smoothing (along u) is well-handled by KDE, temporal discontinuities require explicit treatment. A 2D Gaussian filter addresses both dimensions simultaneously.

4.2.2 Method:

- **2D Gaussian filtering**: Apply separable convolution with spatial (σ_u) and temporal (σ_t) bandwidths:

$$p_{\text{smooth}}(u,t) = \text{GaussianFilter}(p_{\text{sim}}, (\sigma_u, \sigma_t), \text{mode='mirror'})$$

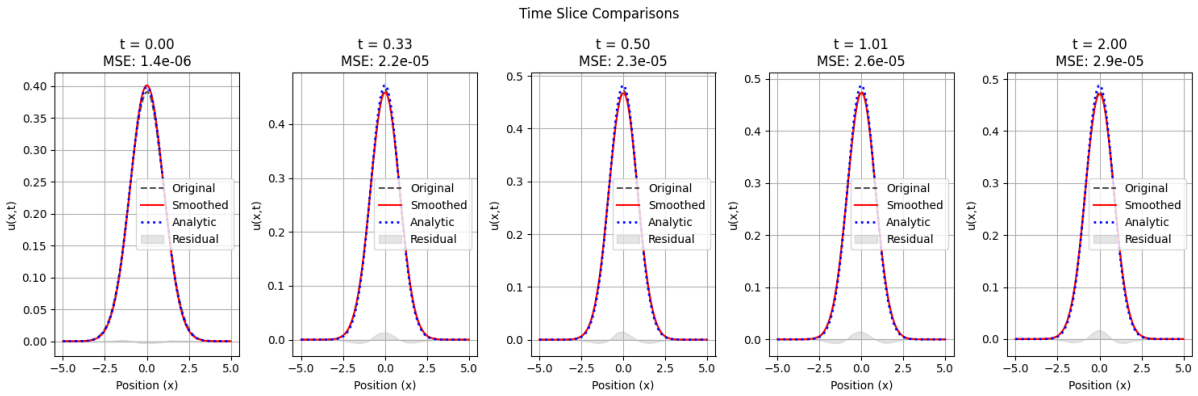


Figure 34: Compare original, analytic, and smoothed p(u,t) in u direction

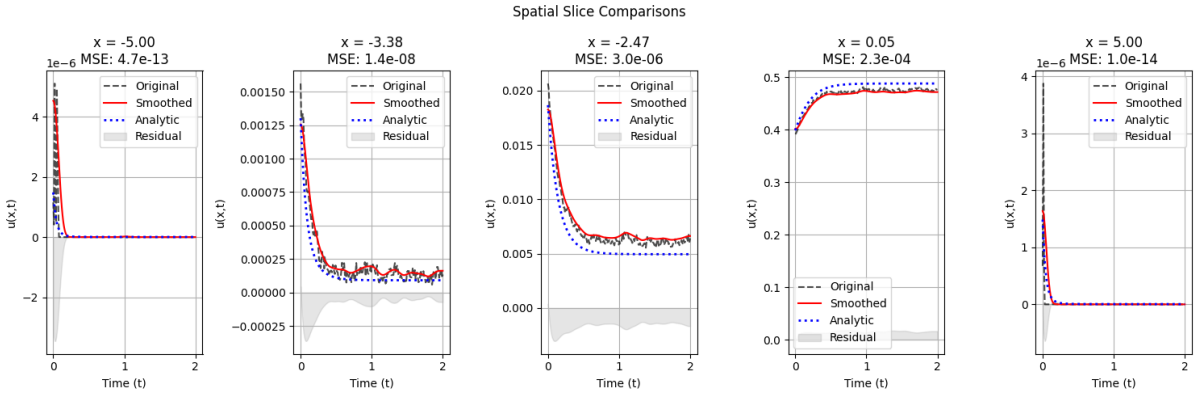


Figure 35: Compare original, analytic, and smoothed p(u,t) in t direction

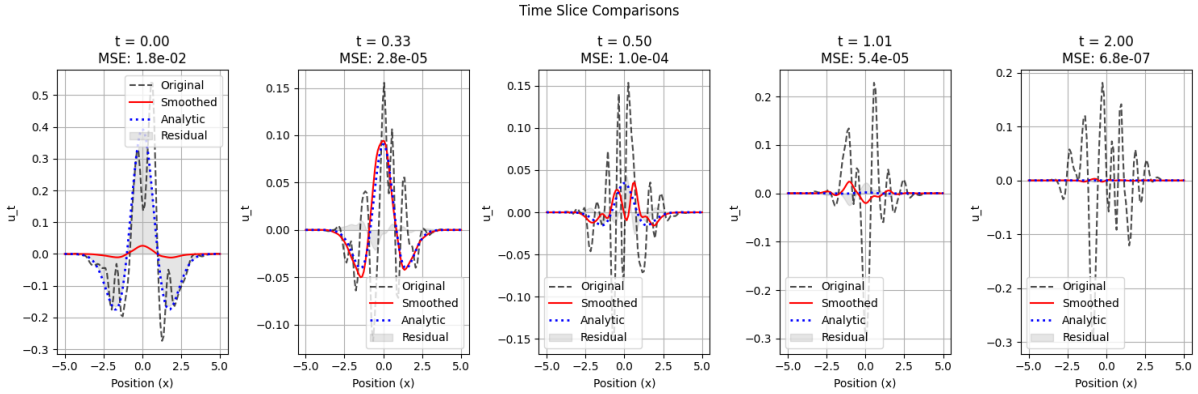


Figure 36: Compare original, analytic, and smoothed dp(u,t) in u direction

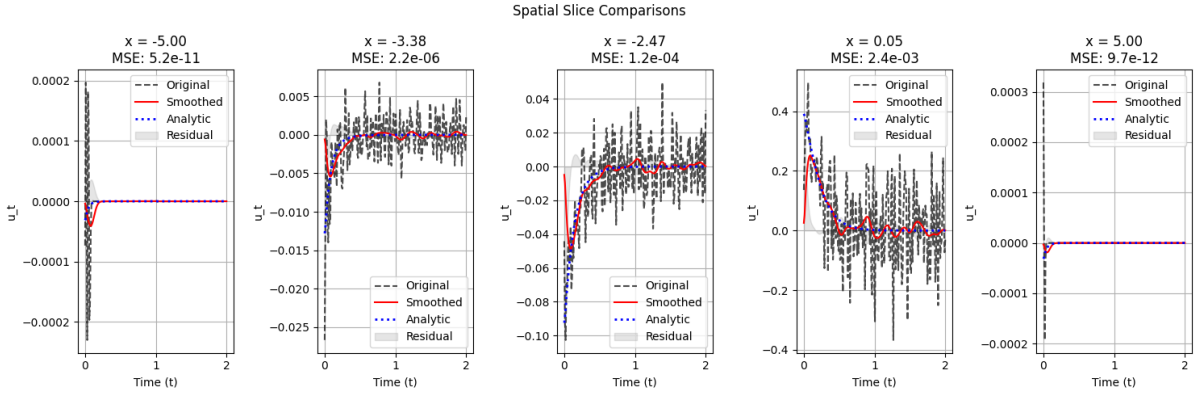


Figure 37: Compare original, analytic, and smoothed dp(u,t) in t direction

- **Edge truncation**: Remove 5% boundary regions after filtering, this is because 2D Gaussian smoothing yields high MSE on the edge. As we can see in the Figure below, while p(u,t) does not have significant difference on the edge, dp(u,t) yields really large difference.

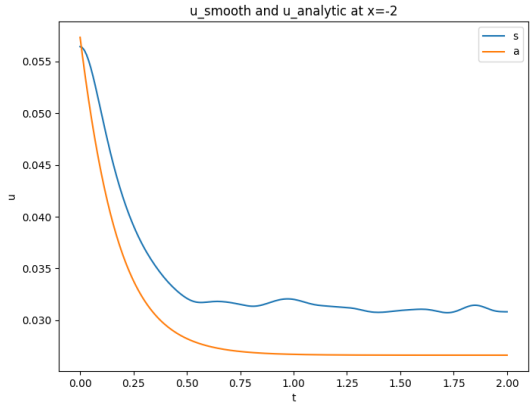


Figure 38: analytic and smoothed p(u,t) at u = -2

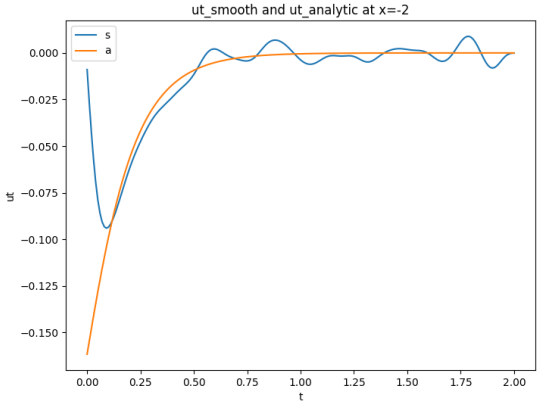


Figure 39: analytic and smoothed dp(u,t) at u = -2

- **Symbolic recovery**: Apply SGA to smoothed $p(u, t)$ with various σ combinations

4.2.3 Result:

Table 6: Smoothing Parameters vs SGA Recovery

(σ_u, σ_t)	Drift Coeff.	Diffusion Coeff.	Terms Identified
(1, 5)	2.77 (3.0)	1.987 (2.0)	Both (1e6 samples)
(1, 5)	0.065 (3.0)	-	Drift only (1e5 samples)
(2, 7)	2.35 (3.0)	1.75 (2.0)	Both (1e5 samples)
(3, 9)	2.45 (3.0)	1.95 (2.0)	Both (1e5 samples)
(4, 10)	2.41 (3.0)	2.08 (2.0)	Both (1e5 samples)
(5, 13)	2.14 (3.0)	2.05 (2.0)	Both (1e5 samples)

Key observations:

- **Data volume**: p(u,t) sampled by 10^6 trajectories enable accurate recovery even with moderate smoothing ($\sigma_t = 5$), but that many trajectories is not realistic in

practice, so we should focus on how to get accurate result using p(u,t) sampled by 10^5 trajectories or less in the future.

- **Tradeoff**: Larger σ reduces noise but biases drift coefficients (up to 29% error)
- **Robustness**: Diffusion coefficients remain stable across configurations (max 13% error)