Answer Sheet for Test Questions

Name: Cen Liu
Application No.: C2117860
Programme: MSc in Biomedical Data Science, NTU

1 Operations for the Right Sum

1.1 Case 1: 9×9 Matrix

Since the problem scale of given 9×9 matrix is relatively small, we can enumerate each feasible operation which is constituted by 8 'R' as well as 8 'D' and check whether these operations can lead to given summed numbers. For a given summed number, there are can be multiple feasible solutions. The solutions are summarized in Table 1 as follows

Table 1 Solutions for 9×9 matrix

Summed Number	Number of Feasible Solutions	Examples of Feasible Solutions
65	63	RRRRRRDDDDRDDDDR or RRRRRDRDDRDDDDDR
72	221	RRRRRDDDRDDDDDRR or RRRRDRDRDDDDDDRR
90	461	RRRDDDDDRDDDRRRR or RRDRDDDRDDDDRRRR
110	15	DRDDDDDDDRRRRRRR or DDRDDDDDRDRRRRRR

Note: The output file which contains only one solution for question 1.a. is named as output_question_1.txt while that contains all feasible solutions is named as output_question_1_AllSolutions.txt. The codes for this problem are written by MATLAB R2020a and are provided as Test_1a.m and bin2oper.m which can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%201.

1.2 Case 2: $90,000 \times 100,000$ Matrix

The scale of this problem is huge since it is even impossible to initialize such a $90,000 \times 100,000$ matrix, so I adopt a heuristic approach to address this problem. At first, let us compute the maximum value as well as the minimum value provided by the feasible paths on this matrix.

$V_{\max} = 1 + 2 + \ldots + 90,000 + 90,000 \times (100,000 - 1) \approx 1.305 \times 10^{10}$

$V_{\min} = 1 + 2 + \ldots + 90,000 + 1 \times (100,000 - 1) \approx 4.050 \times 10^{9}$

Since $87,127,231,192 \approx 8.713 \times 10^{10} > V_{\max}$, there is no feasible operation leading to this number.

For 5,994,891,682 which satisfy $V_{\min} < 5,994,891,682 \approx 5.995 \times 10^9 < V_{\max}$, there is at least one operation leading to this number. Since 5,994,891,682 is closer to $V_{\min}$ than $V_{\max}$, I adopt the path which leads to $V_{\min}$ as initialization. At first, I utilize the largest number '90,000' to fill the gap between 5,994,891,682 and $V_{\min}$. After the gap less than 90,000, the number '2' is selected to fill this gap at a higher precision. The approach can be depicted as follows

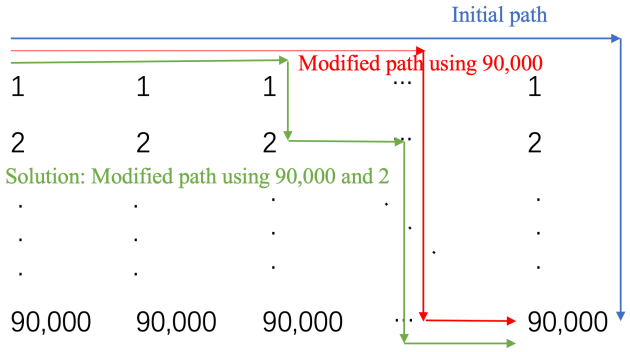


Figure 1 The heuristic approach for question 1.b.

Finally, the feasible operation for 5,994,891,682 can be found as

$$\boldsymbol{R(\times 30100)\ D\ R(\times 48291)\ D(\times 89998)\ R(\times 21608)}$$

where $\times N$ means repeating the given operation for N times.

Note: The codes are provided as Test_1b.m and bin2oper.m which can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%201.

2 Equivalent Networks

At first, we have following equations by observing the structure of Network 1

$$\vec{a}^{(1)} = W^{(1)}\vec{a}^{(0)} + \vec{b}^{(1)}$$
$$\vec{a}^{(2)} = W^{(2)}\vec{a}^{(1)} + \vec{b}^{(2)}$$
$$\vec{a}^{(3)} = W^{(3)}\vec{a}^{(2)} + \vec{b}^{(3)}$$

Then we can derive that

$$\begin{aligned}\vec{a}^{(3)} &= W^{(3)}\vec{a}^{(2)} + \vec{b}^{(3)} \\ &= W^{(3)}\left(W^{(2)}\vec{a}^{(1)} + \vec{b}^{(2)}\right) + \vec{b}^{(3)} \\ &= W^{(3)}\left[W^{(2)}\left(W^{(1)}\vec{a}^{(0)} + \vec{b}^{(1)}\right) + \vec{b}^{(2)}\right] + \vec{b}^{(3)} \\ &= W^{(3)}W^{(2)}W^{(1)}\vec{a}^{(0)} + W^{(3)}W^{(2)}\vec{b}^{(1)} + W^{(3)}\vec{b}^{(2)} + \vec{b}^{(3)}\end{aligned}$$

By observing the structure of Network 2 which has no hidden layers, we can easily get

$$\vec{a}^{(out)} = \tilde{W}\vec{a}^{(in)} + \tilde{b}$$

To guarantee the above two networks are equivalent, following equations must hold which give us the expressions of $\tilde{W}$ and $\tilde{b}$

$$\boldsymbol{\tilde{W} = W^{(3)}W^{(2)}W^{(1)}}$$
$$\boldsymbol{\tilde{b} = W^{(3)}W^{(2)}\vec{b}^{(1)} + W^{(3)}\vec{b}^{(2)} + \vec{b}^{(3)}}$$

3 Multilayer Perceptron for Regression

By using **PyTorch**, the MLP model given in the question is constructed and trained to predict the values of y for the test dataset. The code and output file are provided as Test_3.py and test_predicted.txt, respectively. These files can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%203.

4 Connected Components

4.1 Algorithm Description

An algorithm which can finish this task by visiting each component of the input $M \times N$ binary matrix with time complexity $\mathcal{O}(MN)$ is designed as follows

1. Start from the first component in the matrix. Set current label to 1.

2. If this component is non-zero, give it the current label, set this component to zero and add its index as the first element in an array, then go to (3). Otherwise, repeat (2) for the next component in the matrix.

3. Find all neighbors of the components in the array based on 4-connectivity or 8-connectivity. Set the array to null. If a neighbor is non-zero, give it the current label, set this component to zero and add its index to the array. Repeat (3) until there are no more elements in the array.

4. Go to (2) for the next component in the matrix and increment current label by 1.

Note: The codes are provided as Test_4.m, find_neighbors_4connect.m and find_neighbors_8connect.m which can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%204. The output file named as output_question_4.txt is based on 8-connectivity while the output based on 4-connectivity is also provided as output_question_4_4connectivity.txt.

4.2 Library Functions

The code is written by MATLAB R2020a and the following library functions are used in this work:

a. **size**: sz = size(A) returns a row vector whose elements are the lengths of the corresponding dimensions of A. For example, if A is a 3-by-4 matrix, then size(A) returns the vector [3 4].

b. **zeros**: X = zeros(sz1,...,szN) returns an sz1-by-...-by-szN array of zeros where sz1,...,szN indicate the size of each dimension. For example, zeros(2,3) returns a 2-by-3 matrix.

c. **isempty**: TF = isempty(A) returns logical 1 (true) if A is empty, and logical 0 (false) otherwise.

d. **find**: k = find(X) returns a vector containing the linear indices of each nonzero element in array X. If X is a vector, then find returns a vector with the same orientation as X.

e. **mod**: b = mod(a,m) returns the remainder after division of a by m, where a is the dividend and m is the divisor.

f. **unique**: C = unique(A) returns the same data as in A, but with no repetitions. C is in sorted order.

5 Coloring

5.1 Case 1: 5×5 Matrix, 12 R & 13 B

Since the difference of the number between red beads and blue beads is only one, it is natural to come up with a heuristic idea by arranging the beads in different color alternatively. In this way, the grid configuration can be derived as follows

B R B R B
R B R B R
B R B R B
R B R B R
B R B R B

In this configuration, we can see that the number of red beads and blue beads are 12 and 13, respectively. The neighbors of any given bead are in different color. **The total penalty is minimized as zero in this configuration.**

5.2 Case 2: 64×64 Matrix, 139 R & 1451 B & 977 G & 1072 W & 457 Y

Inspired by Case 1, it is natural to think about whether there also exists a grid configuration leading to zero total penalty for this question. If we can find a configuration like that, then the task is done since zero total penalty is the least one.

As what we did in Case 1, we can choose any two type of beads and arranging them alternatively. In this way, each row of the $L \times L$ matrix contains $\frac{L}{2} = 32$ beads in same color. The number of rows generated by different beads are calculated as follows

$$\text{R: } 139 = \frac{L}{2} \times 4 + 11 \rightarrow \text{Generating 4 rows and leaving 11 beads unplaced}$$
$$\text{B: } 1451 = \frac{L}{2} \times 45 + 11 \rightarrow \text{Generating 45 rows and leaving 11 beads unplaced}$$
$$\text{G: } 977 = \frac{L}{2} \times 30 + 17 \rightarrow \text{Generating 30 rows and leaving 17 beads unplaced}$$
$$\text{W: } 1072 = \frac{L}{2} \times 33 + 16 \rightarrow \text{Generating 33 rows and leaving 16 beads unplaced}$$
$$\text{Y: } 457 = \frac{L}{2} \times 14 + 9 \rightarrow \text{Generating 14 rows and leaving 9 beads unplaced}$$

In an alternative and row-by-row manner, all rows except the last one (the first $L - 1 = 63$ rows) are filled with beads as depicted in Figure 2. This configuration guarantees the total penalty of the first 63 rows is zero.

Last but not least, let us discuss the placement of beads in the last row. As calculated before, we need to place 11 beads in red, 11 beads in blue, 17 beads in cyan, 16 beads in white and 9 beads in yellow in the last row. In addition, we need to take the pattern of 63th row into consideration to avoid extra penalty. One of the placements of the last row which would not incur penalty is depicted in the Figure 3.

Combining the placements of the first 63 rows and the last row showed in Figure 2 and Figure 3, **we manage to get a grid configuration with zero total penalty for this question.**

Note: The output files for question 1 and question 2 are provided as output_question_5_1.txt and output_question_5_2.txt, respectively. The code for generating the solution for question 2 is named as Test_5.m. These files can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%205.

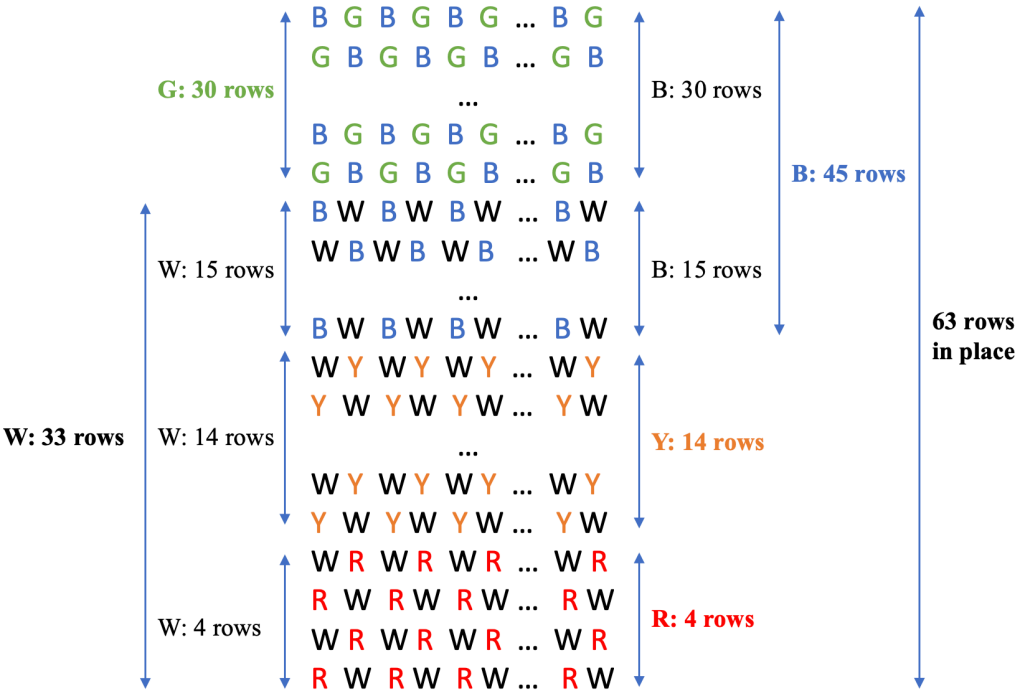


Figure 2 The placement of the first 63 rows.

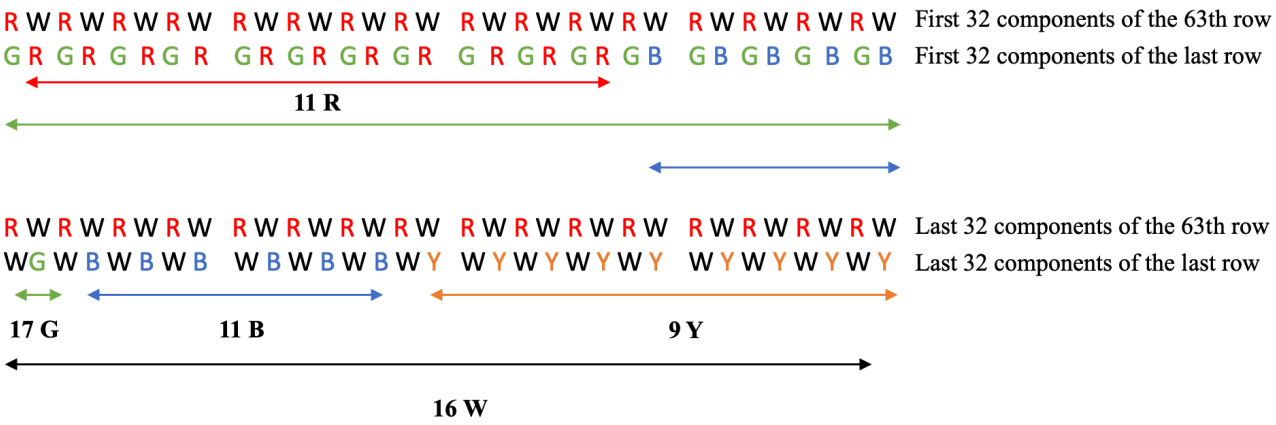


Figure 3 The placement of the last row.

6 Points inside/outside Polygon

6.1 Algorithm Description

A classical algorithm to tell whether a point is either inside or outside a given polygon can be described as follows:

1. For a given test point, making a horizontal line across this point.

2. Count the number of intersections generating by the line and the polygon on both sides of the given point, respectively.

3. If both numbers of intersections are odd, then the test point is inside the polygon. Otherwise, it is outside the polygon.

An application of the above algorithm is depicted in Figure 4. We can see that there are 3 intersections on the left side of the test point while 5 intersections are on the right side. Since both numbers are odd, we can draw a conclusion that this point is inside the polygon.

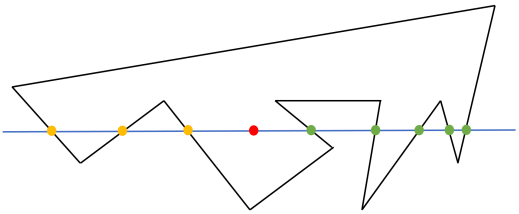


Figure 4 An application of the algorithm.

It will help us obtaining a deeper understanding of the algorithm by observing some special cases. One special case is showed in Figure 5(a). We can see that the horizontal line is right across a vertex of the polygon. The algorithm would fail to work if it mistaking this intersection as 2 different intersections.

(a) The horizontal line is right across a vertex of the polygon. (b) The horizontal line overlaps an edge of the polygon.

Figure 5 Two special cases.

Thus, it is worth giving a criterion on recognizing intersections: *If the Y-coordinate of one endpoint of an edge is strictly less than the Y-coordinate of the test point, while the Y-coordinate of the other endpoint is not less than that, there exists one and only one intersection.* Applying this criterion to Figure 5(a), we success to count the intersections on the left side of the test point only once.

The Figure 5(b) demonstrates another special case where the horizontal line overlaps an edge of the polygon. Applying the above criterion, the corresponding intersections are counted once so the algorithm still works.

6.2 Output and Verification

For given input, the output of the algorithm are

7 11 inside, 10 14 inside, 11 4 outside, 12 21 outside, 16 3 outside,
16 10 inside, 17 4 inside, 18 7 inside, 18 17 outside, 20 7 outside.

To verify the correctness of the output, the input polygon and test points are plotted in Figure 6. It is easy to see that the output of the algorithm for this question is correct.

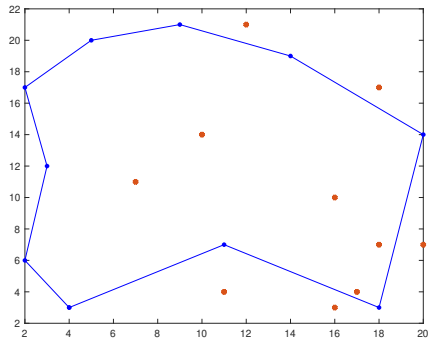


Figure 6 The input polygon and test points.

Note: The output file is provided as output_question_6.txt. The code of the above algorithm and the code for plotting the input polygon and test points (Figure 6) are named as Test_6.m and Test_6_figure.m, respectively. These files can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%206.

6.3 Library Functions

The code is written by MATLAB R2020a and the only library function used in this work is:

length: L = length(X) returns the length of the largest array dimension in X. For vectors, the length is simply the number of elements.

7 Coordinates-to-index & Index-to-coordinates

7.1 2-dimension

For a 2-dimensional grid, the mathematical equation converting coordinates to index is

$$\boldsymbol{I = x_2 L_1 + x_1}$$

Conversely, the equations converting the index into coordinates are

$$\begin{cases} \boldsymbol{x_2 = \lfloor \frac{I}{L_1} \rfloor} \\ \boldsymbol{x_1 = I - x_2 L_1 = I - L_1 \lfloor \frac{I}{L_1} \rfloor} \end{cases}$$

where $\lfloor \cdot \rfloor$ represents the rounding operator which returns the largest integer that no more than the input number.

Given input indices and coordinates, corresponding coordinates and indices can be output in a straight way according to the above equations.

7.2 d-dimension

To gain more insights, let us look at the 3-dimensional case first. The equations between the index I and the 3-D coordinates (x_1, x_2, x_3) can be derived as

$$I = x_3 L_2 L_1 + x_2 L_1 + x_1$$

$$\begin{cases} x_3 = \lfloor \dfrac{I}{L_2 L_1} \rfloor \\ x_2 = \lfloor \dfrac{I - x_3 L_2 L_1}{L_1} \rfloor \\ x_1 = I - x_3 L_2 L_1 - x_2 L_1 \end{cases}$$

Without loss of generality, the above equations can be generalized to the d-dimensional case. On the one hand, for the conversion from d-dimensional coordinates $(x_1, x_2, \ldots, x_d)$ to the scalar index I, we will adopt

$$\boldsymbol{I = x_d L_{d-1} \ldots L_1 + x_{d-1} L_{d-2} \ldots L_1 + \ldots + x_2 L_1 + x_1}$$

On the other hand, the inverse process can be expressed as

$$\begin{cases} \boldsymbol{x_d = \lfloor \dfrac{I}{L_{d-1} L_{d-2} \ldots L_1} \rfloor} \\ \boldsymbol{x_{d-k} = \lfloor \dfrac{I - x_d L_{d-1} \ldots L_1 - \ldots - x_{d-k+1} L_{d-k} \ldots L_1}{L_{d-i+1} L_{d-i} \ldots L_1} \rfloor \ , \ k = 1, 2, \ldots, d-2} \\ \boldsymbol{x_1 = I - x_d L_{d-1} \ldots L_1 - x_{d-1} L_{d-2} \ldots L_1 - \ldots - x_3 L_2 L_1 - x_2 L_1} \end{cases}$$

For the given 6-dimensional input, the output can be derived directly according to the above equations by setting $d = 6$.

Note: The input files of the 2-dimensional case are saved as in_coordinate.mat and in_index.mat, while those of the 6-dimensional case are saved as in_coordinate_2.mat and in_index_2.mat. The output files of the 2-dimensional case are provided as output_index_7_1.txt and output_coordinates_7_1.txt, while those of the 6-dimensional case are output_index_7_2.txt and output_coordinates_7_2.txt. The codes for the 2-dimensional and 6-dimensional cases are named as Test_7_1.m and Test_7_2.m, respectively. These files can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%207.

8 Enzyme Kinetics

8.1 Rate Equations

To start with, denote by $y_E(t)$, $y_S(t)$, $y_{ES}(t)$ and $y_P(t)$ the concentrations of the enzyme E, the substrate S, the intermedia ES and the product P, respectively. For ease of exposition, the argument t is omitted so the concentrations can be expressed by y_E, y_S, y_{ES} and y_P in a simpler way. By using the law of mass action, equations for the rate of changes of the four species are derived as follows

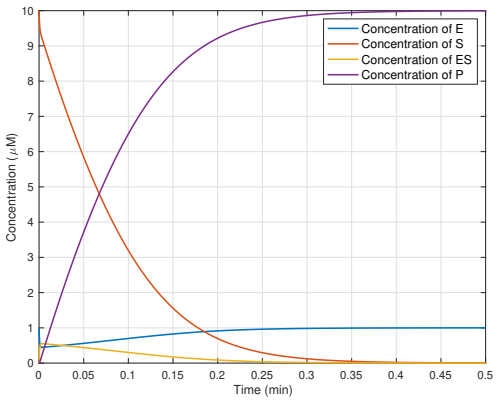
$$\begin{cases} \boldsymbol{\dfrac{\mathrm{d}y_E}{\mathrm{d}t} = -k_1 y_E y_S + (k_2 + k_3) y_{ES}} \\ \boldsymbol{\dfrac{\mathrm{d}y_S}{\mathrm{d}t} = -k_1 y_E y_S + k_2 y_{ES}} \\ \boldsymbol{\dfrac{\mathrm{d}y_{ES}}{\mathrm{d}t} = k_1 y_E y_S - (k_2 + k_3) y_{ES}} \\ \boldsymbol{\dfrac{\mathrm{d}y_P}{\mathrm{d}t} = k_3 y_{ES}} \end{cases}$$

8.2 Numerical Solutions of the Rate Equations

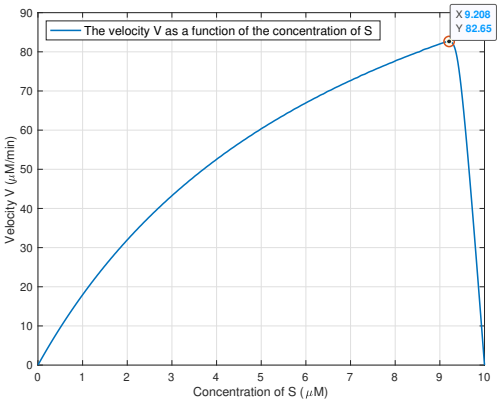
By substituting the given parameters into the above rate equations, we obtain a system of ordinary differential equations (ODEs)

$$\begin{cases} \dfrac{\mathrm{d}y_E}{\mathrm{d}t} = -100y_E y_S + 750y_{ES} \\ \dfrac{\mathrm{d}y_S}{\mathrm{d}t} = -100y_E y_S + 600y_{ES} \\ \dfrac{\mathrm{d}y_{ES}}{\mathrm{d}t} = 100y_E y_S - 750y_{ES} \\ \dfrac{\mathrm{d}y_P}{\mathrm{d}t} = 150y_{ES} \\ y_E(0) = 1, y_S(0) = 10, y_E S(0) = 0, y_P(0) = 0. \end{cases}$$

Based on the fourth-order Runge-Kutta formula, we can numerically solve this system of ODEs by calling the library function *ode45* in MATLAB. The numerical solutions of the rate equations are demonstrated in Figure 7(a).



(a) The numerical solutions of the rate equations.



(b) The velocity V as a function of the concentration of the substrate S.

Figure 7 The numerical solutions and the saturation point.

As shown in Figure 7(a), the concentrations of the substrate S and the intermedia ES decrease to zero as time goes on. The concentration of the product P increases at first and then saturates at a certain value. Since the enzyme E acts as a catalyst, it remains unchanged at the end of the reaction.

8.3 The Saturation Point of the Velocity

As depicted in Figure 7(b), the velocity V is plotted as a function of the concentration of the substrate S. As mentioned by the question, we can find that the velocity V increases approximately linearly when the concentrations of S are small on the one hand, and it saturates to a maximum value V_m at large concentrations of S on the other hand. The saturation point is marked out on the figure. It shows that the maximum value of the velocity is $\boldsymbol{V_m = 82.65\ \mu\mathrm{M/min}}$ when the concentration of S is 9.21 μM.

Note: The code for solving the rate equations as well as plotting the saturation point are provided as RateEquation.m and Test_8.m. They can be accessed through this link: https://github.com/Star-sunny/Answers-for-Test-Questions/tree/main/Test%208.

9 Interpretable Machine Learning

9.1 The Rashomon Effect

In a liter of water, drop 10 grams of salt or sugar. We can discriminate whether you dropped sugar or salt when we taste it. However, add a liter of water, another liter of water, and so on. Eventually, we would not be able to tell what you added. Such is the effect of dilution. When dealing with a high-dimensional dataset, we will see a similar effect. For a fixed-size training sample, if the number of variables reaches a certain point, the sample will be diluted over the space spanned by these variables. This space will also contain many satisfactory models who meets certain precision requirements. However, for real-world problems, maybe only one or a few of those models will be useful. This means that for real world observations outside the training sample, most of these satisfactory models will have a bad performance [1]. This phenomenon is caused by the Rashomon Effect.

The *Rashomon Effect*, also known as the *multiplicity of good models*, is a phenomenon identified by Leo Breiman [2] where for a given set of data, it is possible to construct many equally well-performing models that may differ greatly in their internal structure and hence in their attendant explanations. In the Rashomon Effect, there are multiple models that perform similarly well, their accuracies are so close to one another that we cannot be sure that the differences are not due to random factors. Each of those models suggests a different explanation, even though they all have similar input-output mappings [3].

9.2 The Rashomon Set

The mathematical definition of Rashomon Set is given as follows

Definition (Rashomon Set) *Given $\theta \geq 0$, a dataset S, a hypothesis space $\mathcal{F}$, and a loss function ϕ, the Rashomon Set $\hat{R}_{set}(\mathcal{F}\theta)$ is the subspace of the hypothesis space defined as follows* [4]

$$\hat{R}_{\text{set}}(\mathcal{F}\theta) := \{f \in \mathcal{F} : \hat{L}(f) \leq \hat{L}(\hat{f}) + \theta\}$$

where $\hat{f}$ is an empirical risk minimizer for the training data S with respect to a loss function ϕ : $\hat{f} \in \arg\min_{f\in\mathcal{F}} \hat{L}(f)$.

An illustration of the Rashomon set is shown in Figure 8. Models below the red plane belong to the Rashomon set $\hat{R}_{\text{set}}(\mathcal{F}\theta)$ where the height of the red plane is adjusted by the Rashomon parameter θ.

(a) From the side. (b) From above.

Figure 8 An illustration of a possible Rashomon set in two dimensional hypothesis space $\mathcal{F}$.

9.3 The Theoretical Guarantee for the Explainable Models

The real-world data usually permit a large set of reasonably accurate predictive models to exist. Since the set of satisfactory models is large, it often contains at least one model that is explainable. Thus, the conception

of Rashomon Set can serve as a theoretical guarantee for the existence of the explainable models since it is realistic and some inspirations provided by it.

9.3.1 The Rashomon Set is a Realistic Idea to Capture the Explainable Models

For a given data set, the Rashomon set is defined as the set of reasonably accurate predictive models [5]. Due to the real-world data must be finite, a quite large Rashomon set would be generated since the data could admit many close-to-optimal models. As long as the volume of the Rashomon set is large enough, it probably contains functions that can be approximated well by simpler functions. In other words, uncertainty arising from the data leads to a Rashomon set (probably very large), a larger Rashomon set probably contains explainable models, thus explainable accurate models often exist.

9.3.2 Inspirations Provided by the Rashomon Sets

The first inspiration is that large Rashomon sets can embed models from simpler hypothesis spaces. Actually, this statement has been proved in the Section 4 of the literature [4]. The second inspiration is that similar performance across different machine learning algorithms may correlate with large Rashomon sets. In practice, many different machine learning algorithms sometimes perform similarly on the same dataset, despite they have different functional forms. These phenomenon now have a reasonable explanation that the Rashomon set decided by the training dataset is quite large.

References

[1] Jue Wang and Qing Tao. Machine learning: The state of the art. *IEEE Intelligent Systems*, 23(6):49–55, 2008.

[2] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[3] Leif Hancox-Li. Robustness in machine learning explanations: does it matter? In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 640–647, 2020.

[4] Lesia Semenova, Cynthia Rudin, and Ronald Parr. A study in rashomon curves and volumes: A new perspective on generalization and model simplicity in machine learning. *arXiv preprint arXiv:1908.01755*, 2019.

[5] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.