Lorenz System

[*]Report 5 on the course "Numerical Analysis".

1st Chen Yihang

Peking University

1700010780

Abstract

We implment the explicit schemes, the predictor-corrector schemes, Runge–Kutta schemes and embeded (adaptive) Runge-Kutta schemes and perform tests on the Lorenz system. Tables and figures have been produced on various hyperparameters of the Lorenz system.

CONTENTS

I. Problem Statement

Consider the famous Lorenz equation

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x) \\ \frac{\mathrm{d}y}{\mathrm{d}t} = \rho x - y - xz \\ \frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z \end{cases} \tag{1}$$

1) For $\sigma = 10, \rho = 28, \beta = 8/3$, choose different initial values, and observe the results of simulation: whether the solution is bounded, periodic or converged.

2) Choose different σ, ρ, β and initial values, and observe the results of simulation.

II. ODE schemes

A. Fixed step size

We implment the following algorithms

1) **Forward schemes**: Euler, Adams-2.
2) **Predictor-corrector schemes**: Modified Euler, Milne-3, Adams-4.
3) **Runge-Kutta**: classical Runge-Kutta, Kutta-33, Heun-33, Kutta-44, Gill-44.
4) **Embeded Runge-Kutta**: Bogacki-Shampine[1], Runge–Kutta–Fehlberg[2], Cash-Karp[3], Dormand-Prince[4].

The Bogacki–Shampine method is implemented in the ode23 function in MATLAB. The Dormand–Prince method is currently the default method in the ode45 solver for MATLAB and GNU Octave and is the default choice for the Simulink's model explorer solver. Despite the fact that embeded Runge-Kutta algorithms are adaptive step size algorithms, in this part, we regard it as fixed step size ones.

The descriptions of the algorithms can be found in the textbooks and the links above. Hence, we will not restate the algorithms here.

B. Adaptive Runge–Kutta methods

Adaptive methods are designed to produce an estimate of the local truncation error of a single Runge–Kutta step. This is done by having two methods, one with order p and one with order

[1]https://en.wikipedia.org/wiki/Bogacki-Shampine_method
[2]https://en.wikipedia.org/wiki/Runge-Kutta-Fehlberg_method
[3]https://en.wikipedia.org/wiki/Cash-Karp_method
[4]https://en.wikipedia.org/wiki/Dormand-Prince_method

$p-1$. These methods are interwoven, i.e., they have common intermediate steps. Thanks to this, estimating the error has little or negligible computational cost compared to a step with the higher-order method.

During the integration, the step size is adapted such that the estimated error stays below a user-defined threshold: If the error is too high, a step is repeated with a lower step size; if the error is much smaller, the step size is increased to save time. This results in an (almost) optimal step size, which saves computation time. Moreover, the user does not have to spend time on finding an appropriate step size.

The lower-order step is given by

$$y^*_{n+1} = y_n + h\sum_{i=1}^{s} b^*_i k_i$$

where k_i are the same as for the higher-order method. Then the error is :

$$\Delta = e_{n+1} = y_{n+1} - y^*_{n+1} = h\sum_{i=1}^{s}(b_i - b^*_i)k_i$$

which is $O(h^p)$.

The Butcher tableau for this kind of method is extended to give the values of b^*_i:

Hence, the error Δ scales as h^p. If we take a step h_1 and produce an error Δ_1 therefore, the step h_0 that would have given some other value Δ_0 is readily estimated as

$$h_0 = h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{\frac{1}{p}} \tag{2}$$

Henceforth, we will let Δ_0 denote the desire accuracy. Then, the equation is used in two ways: If Δ_1 is larger than Δ_0 in magnitude, the equation tells how much to decrease the stepsize when we retry the present (failed) step. If Δ_1 is smaller than Δ_0, on the other hand, then the equation tells how much we can safely increase the stepsize for the next step.

In the following, we implment the adaptive Fehlberg, Cash-Karp and Dormand-Prince methods in "schemes_adp.py". The main function "ode45" implments the adaptive schemes. Despite the codes are quite clear, we still give a brief description of the algorithm.

We use the function "ode45_step" to calculate the fifth and fourth order solutions, and the Δ_1 can be obtained via calculating the difference of these two solutions. The desired accuracy is set to be

$$\Delta_0 = \varepsilon(|y| + h|f(x,y)|) \tag{3}$$

to which some small values, such as ε^2 in our codes, can be added in order to prevent any item of Δ_0 from being zero. Then, we can readily define the new step size as

$$h' = \begin{cases} \max(Sh\gamma^{0.2}, h/10), & \gamma \geq 1 \\ \min(Sh\gamma^{0.25}, 5h), & \gamma < 1 \end{cases} \tag{4}$$

where $\gamma = \max\left(\frac{\Delta_0}{\Delta_1}\right)$, and the division is performed element-wisely. S is a safety factor, which is a few percent smaller than 1.

If on the step k, $t_k + h_k > T$, then we take $h_k = T - t_k$.

In our codes, we take $S = 0.9, \varepsilon = 2^{-52}$.

C. Comparision with MATLAB default ode45

I found that the default solver "ode45" in MATLAB is open-source. I read its implmentation and makes the following comments. They use the Dormand–Prince method while my implementation includes all types of 4-5 embeded methods. They use $S = 0.8$ while we use $S = 0.9$. They use a unified power of 0.2 while we use 0.2 and 0.25 for different occasions.

D. Codes description

1) schemes.py: implment one step of all the schemes in the report.
2) ode_solver.py: solves the ode via schemes from "schemes.py".
3) schemes_adp.py: implment the adaptive Runge–Kutta method, whose main function is "ode45".
4) Lorenz_plot.py: generate all the figures in the report.
5) test.py: generate all the tables in the report.
6) ode45.m: MATLAB's recommended ODE solver.

E. Numerical results

We set the Lorenz system to be $\sigma = 10, \rho = 28, \beta = 8/3$. We plot its trajectory until $T = 1, 10, 100$. We regard the solution obtained by "ode45" with Dormand–Prince method as the groundtruth and compare the ℓ_2 norm of other solutions.

We test $T = 1$ for all the schemes in I, but will test $t = 10$ only for Runge-Kutta schemes in II.

We can summarize our results by

1) Vanilla Euler method will easily encounter overflow while calculation.
2) Predictor-corrector methods are not stable when numerious steps are taken.

Table I
$T = 1, (1, 1, 1)^\top$

Schemes	Order	Step=100	Step=200	Step=500	Step=1000	Step=2000	Step=5000	Step=10000
Kutta33	3	0.0218308	0.0027743	0.0001779	2.22e-05	2.78e-06	1.78e-07	2.22e-08
Heun33	3	0.0220577	0.0027766	0.0001777	2.22e-05	2.77e-06	1.77e-07	2.22e-08
Runge Kutta	4	9.45e-05	2.53e-06	7.68e-08	5.80e-09	3.96e-10	8.95e-12	1.76e-12
Kutta44	4	0.0001048	3.65e-06	1.08e-07	7.64e-09	5.08e-10	1.17e-11	1.51e-12
Gill44	4	0.0001145	4.02e-06	1.14e-07	8.06e-09	5.36e-10	1.24e-11	1.56e-12
Bogacki-Shampine-3	3	0.0220435	0.0027769	0.0001778	2.22e-05	2.77e-06	1.78e-07	2.22e-08
Bogacki-Shampine-2	2	0.0201408	0.0024306	0.0001967	4.52e-05	1.18e-05	1.98e-06	5.05e-07
Fehlberg-5	5	2.92e-05	9.12e-07	9.25e-09	2.87e-10	7.97e-12	1.91e-12	2.36e-12
Fehlberg-4	4	6.76e-05	1.89e-06	1.99e-08	1.20e-09	8.64e-11	4.27e-12	2.46e-12
Cash-Karp-5	5	4.28e-06	1.32e-07	1.34e-09	4.06e-11	1.19e-12	1.95e-12	2.36e-12
Cash-Karp-4	4	1.23e-05	3.43e-07	3.23e-09	1.77e-10	1.42e-11	2.35e-12	2.40e-12
Dormand-Prince-5	5	4.87e-06	1.78e-07	1.94e-09	6.30e-11	3.19e-12	1.98e-12	2.35e-12
Dormand-Prince-4	4	2.08e-05	5.90e-07	1.21e-08	8.63e-10	6.01e-11	3.61e-12	2.46e-12
Euler	1	12.427199	4.6086493	1.5965305	0.765861096	0.376567753	0.14905459	0.074272394
Adams2	2	0.129439	0.023051	0.0037252	0.000972439	0.011953608	0.00472911	0.002355916
Euler2	2	0.0619742	0.0090694	0.0011692	0.000296198	0.011809117	0.00470615	0.00235019
Milne3	3	0.0057503	0.0022439	0.0003135	5.09e-05	0.011739047	0.00469499	0.002347406
Adams4	4	0.0049548	0.0001974	7.73e-06	9.86e-07	0.011738449	0.00469495	0.002347401
ode45-Fehlberg	5				3.16e-12			
ode45-Cash-Karp	5				2.87e-12			
ode45-Dormand-Prince	5				0			

Table II
$T = 10, (1, 1, 1)^\top$

Schemes	Order	Step=1000	Step=2000	Step=5000	Step=10000	Step=20000	Step=50000	Step=100000
Kutta33	3	1.41e-01	1.81e-02	1.16e-03	1.45e-04	1.82e-05	1.16e-06	1.45e-07
Heun33	3	1.44e-01	1.81e-02	1.16e-03	1.45e-04	1.81e-05	1.16e-06	1.45e-07
Runge Kutta	4	1.14e-03	3.44e-05	4.08e-07	2.14e-08	1.34e-09	3.06e-10	3.75e-10
Kutta44	4	1.19e-03	3.52e-05	4.22e-07	2.33e-08	1.48e-09	3.03e-10	3.76e-10
Gill44	4	1.27e-03	3.76e-05	4.39e-07	2.38e-08	1.52e-09	3.01e-10	3.76e-10
Bogacki-Shampine-3	3	1.44e-01	1.81e-02	1.16e-03	1.45e-04	1.82e-05	1.16e-06	1.45e-07
Bogacki-Shampine-2	2	2.24e-01	3.82e-02	4.44e-03	9.75e-04	2.27e-04	3.48e-05	8.56e-06
Fehlberg-5	5	2.52e-04	7.90e-06	8.01e-08	2.36e-09	1.05e-10	3.09e-10	3.74e-10
Fehlberg-4	4	6.45e-04	1.96e-05	1.86e-07	5.62e-09	2.71e-10	3.12e-10	3.75e-10
Cash-Karp-5	5	3.71e-05	1.14e-06	1.15e-08	2.18e-10	1.72e-10	3.09e-10	3.74e-10
Cash-Karp-4	4	1.15e-04	3.52e-06	3.36e-08	9.41e-10	1.74e-10	3.12e-10	3.74e-10
Dormand-Prince-5	5	4.01e-05	1.50e-06	1.66e-08	6.68e-10	2.00e-10	3.10e-10	3.74e-10
Dormand-Prince-4	4	2.15e-04	6.22e-06	5.91e-08	2.48e-09	2.66e-10	3.14e-10	3.75e-10
ode45-Fehlberg	5				4.45e-10			
ode45-Cash-Karp	5				3.56e-10			
ode45-Dormand-Prince	5				0			

We may consider whether we can produce satisfactory results for larger T. We take $T = 50$. No method, even my ode45 solver, can produce promising results (basically every two schemes produce significantly different results), which reflect the chaotic behavior of the systen. We present the results in III.

We also consider other initialization.

Table III
$T = 50, (1, 1, 1)^\top$, CHAOTIC SYSTEM.

Schemes	Order	Step=1000	Step=5000	Step=10000
Kutta33	3	21.722371	12.71423417	1.12e+01
Heun33	3	25.032635	1.919558767	9.73e+00
Runge Kutta	4	3.3026864	9.247784466	2.09e+01
Kutta44	4	9.6745926	8.947122728	1.25e+01
Gill44	4	22.838871	9.877826653	2.33e+01
Bogacki-Shampine-3	3	20.823048	33.70047057	1.57e+01
Bogacki-Shampine-2	2	8.3741834	22.52080048	8.68e+00
Fehlberg-5	5	5.1341464	8.746708108	1.49e+01
Fehlberg-4	4	12.95801	22.01023818	2.17e+01
Cash-Karp-5	5	21.286549	17.4042821	1.34e+01
Cash-Karp-4	4	5.4040717	10.94574968	1.02e+01
Dormand-Prince-5	5	10.903763	11.40696783	3.36e+00
Dormand-Prince-4	4	12.358719	9.957384526	3.29e+00
ode45-Fehlberg	5		18.04508859	
ode45-Cash-Karp	5		1.756454731	
ode45-Dormand-Prince	5		0	

Table IV
$T = 1, (100, 1, 1)^\top$

Schemes	Order	Step=100	Step=200	Step=500	Step=1000	Step=2000	Step=5000	Step=10000
Kutta33	3	2.1939726	0.339955145	0.02316552	2.93e-03	3.68e-04	2.36e-05	2.95e-06
Heun33	3	2.5101006	0.361725167	0.0235087	2.92e-03	3.64e-04	2.32e-05	2.90e-06
Runge Kutta	4	3.34e-01	1.29e-02	1.95e-04	9.78e-06	5.49e-07	1.33e-08	8.12e-10
Kutta44	4	0.3522172	1.35e-02	2.03e-04	1.02e-05	5.69e-07	1.37e-08	8.40e-10
Gill44	4	0.3282259	1.27e-02	1.93e-04	9.70e-06	5.44e-07	1.31e-08	8.05e-10
Bogacki-Shampine-3	3	2.2903341	0.345494236	0.0230409	2.89e-03	3.61e-04	2.31e-05	2.89e-06
Bogacki-Shampine-2	2	2.077238	0.229148257	0.01272715	4.23e-03	1.33e-03	2.41e-04	6.26e-05
Fehlberg-5	5	2.93e-02	1.25e-03	1.39e-05	4.40e-07	1.38e-08	1.43e-10	6.34e-12
Fehlberg-4	4	1.02e-01	4.02e-03	4.84e-05	1.91e-06	9.05e-08	2.00e-09	1.23e-10
Cash-Karp-5	5	1.10e-02	2.62e-04	2.37e-06	7.20e-08	2.23e-09	2.40e-11	3.34e-12
Cash-Karp-4	4	3.02e-02	1.10e-03	1.57e-05	7.28e-07	3.77e-08	8.50e-10	5.30e-11
Dormand-Prince-5	5	2.40e-03	2.22e-04	3.62e-06	1.24e-07	4.01e-09	4.08e-11	2.53e-12
Dormand-Prince-4	4	4.48e-02	1.45e-03	1.77e-05	8.28e-07	4.67e-08	1.16e-09	7.44e-11
ode45-Fehlberg	5				4.26e-12			
ode45-Cash-Karp	5				3.62e-12			
ode45-Dormand-Prince	5				0			

III. Plot the Lorenz equation

A. Simulation method

For the equation

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y) \tag{5}$$

We use the classical Runge-Kutta formula:

$$y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

where,

$$\begin{cases} K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_1) \\ K_3 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_2) \\ K_4 = f(x_n + h, y_n + hK_3) \end{cases} \tag{6}$$

B. Results

The results are plotted in Lorenz.py. We can have the following oberservation.

a) $\sigma = 10, \rho = 28, \beta = 8/3$*:* We analyze the case in Figure 1, Figure 2.

- Figure 1 shows that small perturbations of initialization results in significant difference as the system evolves.
- Figure 2 shows that significant different initialization will eventually become periodic in the same region as the system evolves.

b) Other settings: We adjust the paramters in Lorenz system and perform simulations. We find that

- In general, σ, ρ, β should be positive to ensure the trajectory is bounded, as Figure 7 shows.
- The size of the paramters greatly influences the shape of attractor. It could have two symmetric attractors with shape "o", or one attractor with shape "8". Middle cases are more complicated.

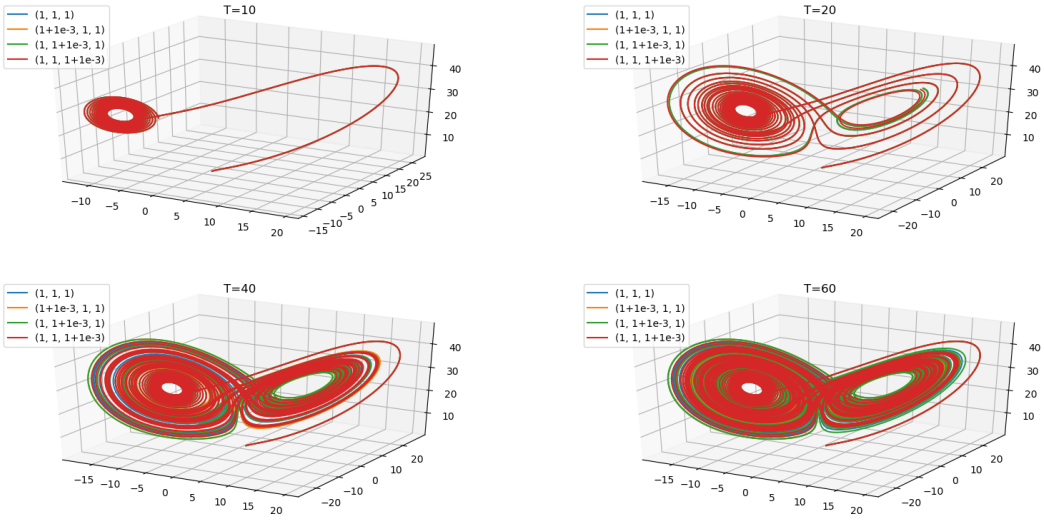

Figure 1. Case 1

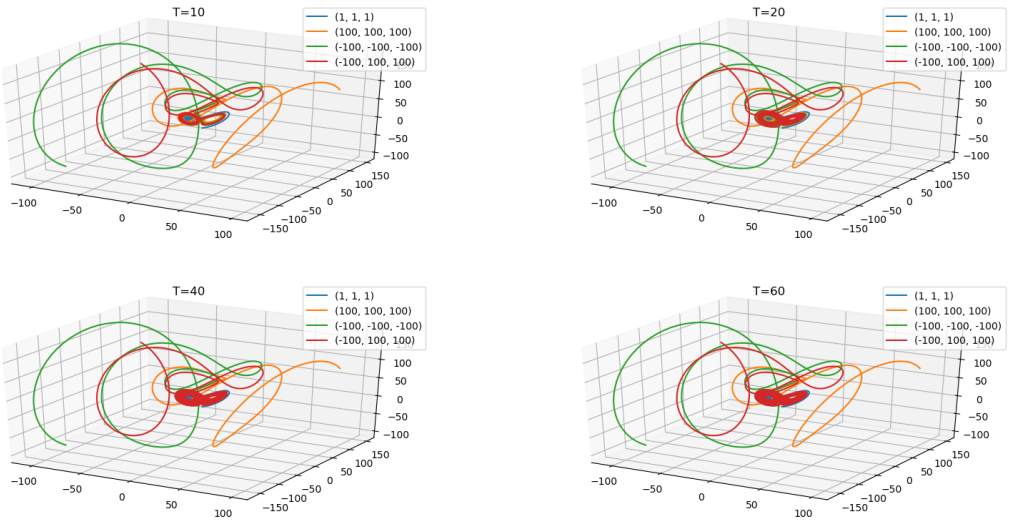

Figure 2. Case 2

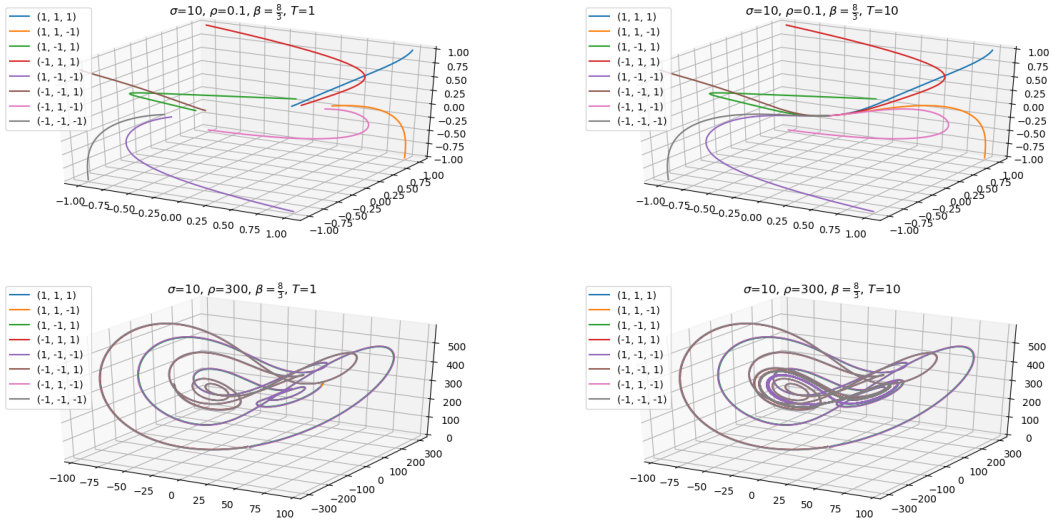

Figure 3. Case 3

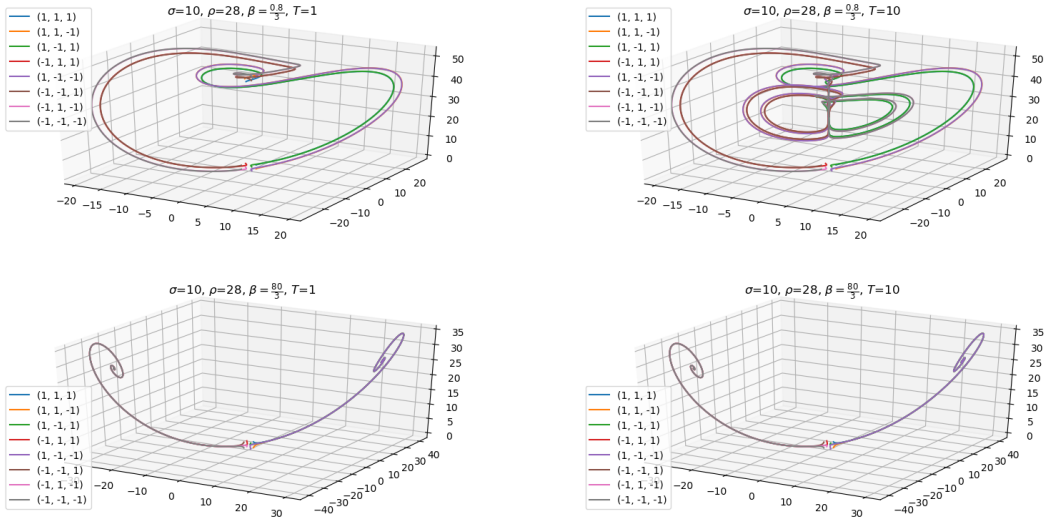

Figure 4. Case 4

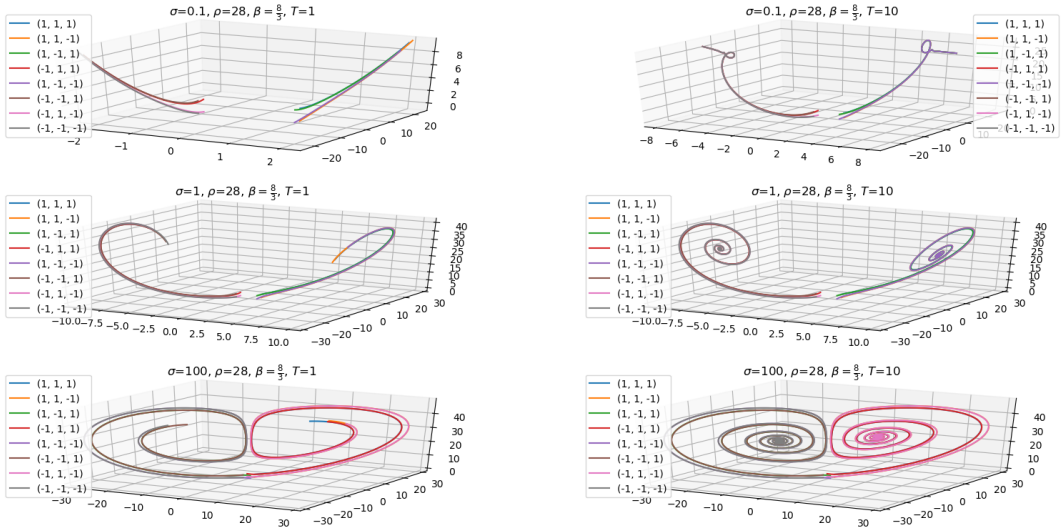

Figure 5. Case 5

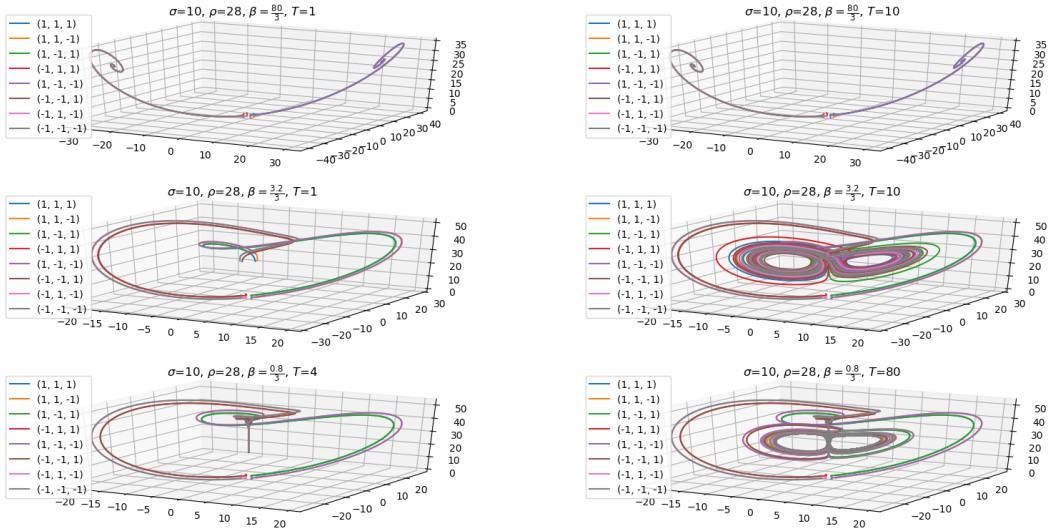

Figure 6. Case 6

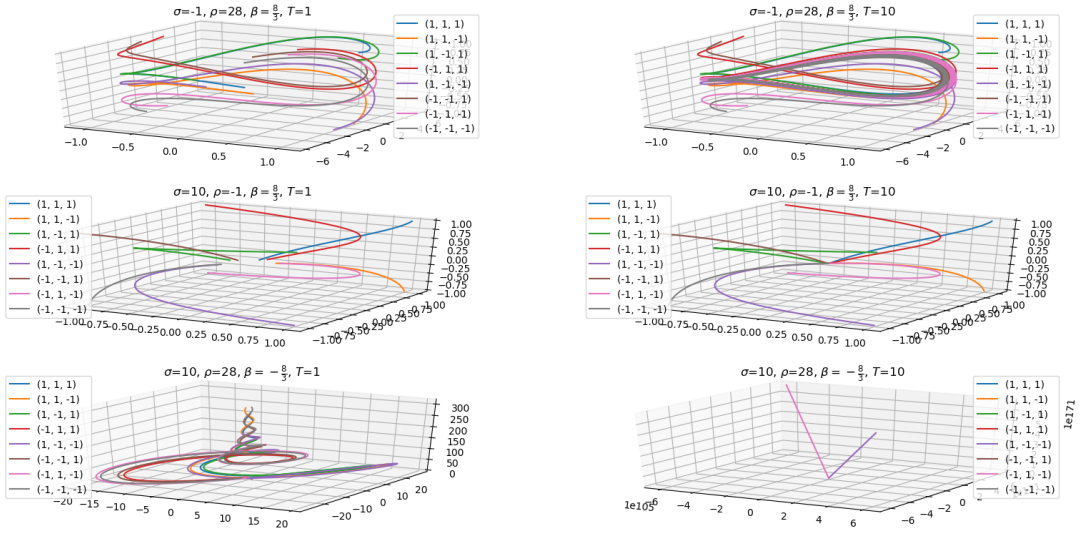

Figure 7. Case 7