数値解析 期末レポート

テーマ

構造保存型数値解法を中心とした種々の常微分方程式の数値解法の比較

目次

1. 導入

講義では様々な常微分方程式の解法を習ったが，その中でも構造保存型数値解法は物理的な側面を保存する解法となっていることが興味深く感じられ，また実用上重要であると考えたため，この解法の性能を調べるテーマを設定した．性能の比較においては複数の観点により網羅的に比較するため，精度（解法の次数），安定性，速度について調べた．

2. 計画・方針

2.1 扱う問題

本レポートでは図 1 のようなおもりが 3 個の連成振動を扱った．ここで講義と同じように 3 つのおもりの質量を 1，バネ定数を 1 とした．従ってこの系の Hamiltonian H は，

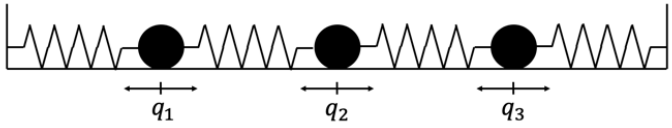
$$H(\boldsymbol{p},\boldsymbol{q}) = \frac{p_1^2}{2} + \frac{p_2^2}{2} + \frac{p_3^2}{2} + \frac{q_1^2}{2} + \frac{(q_1-q_2)^2}{2} + \frac{(q_2-q_3)^2}{2} + \frac{q_3^2}{2}.$$

正準運動方程式は，

$$\begin{pmatrix}\boldsymbol{p}\\ \boldsymbol{q}\end{pmatrix} = J\nabla H(\boldsymbol{p},\boldsymbol{q}).$$

ここでI を 3 次元の単位行列として$J = \begin{pmatrix} O & I \\ -I & O \end{pmatrix}$ である．

図 1. おもりが 3 個の連成振動．両端が壁に繋がれており，床との摩擦はなし．



この問題を選んだ理由は，系が単純であり誤差の解析に必要な解析解を求める（設定する）ことが容易だからである．また，連立常微分方程式を今まで扱ったことがなかったため，運動方程式が少し大きめの連立常微分方程式になるようにした．

2.2 扱う数値解法

比較する数値解法として，表 2 に示した解法を扱った（構造保存型数値解法については [1]にあり解法を用いた）．RK 法については，安定性領域の観点からも基本的に 4 次を使うと考えられるため他の次数のものは省略したが，BDF については優劣がわからなかったので全て試すことにした．付録に示したコードでは表 2 で示した略記法を用いており，以下でもこれらを用いる．

本レポートで扱う問題は Hamiltonian が可分な場合であり，Symplectic Euler 法は陽的になることに注意する．この場合，[1]によると高次の陽的な Symplectic 解法が存在し，表 2 の通りとなっている（それぞれの解法についての説明は省略するが，付録 6.2 節にコードを示した）．これらの各陽的 Symplectic 解法については，[1]に倣い「陽的 n 次法」と表記する．また，2 次 GL 法は[1]にある通り陰的中点則であり，今回は講義で扱われた離散勾配法の離散勾配の作り方のうち，Average Vector Field，あるいは因数分解による方法と一致する．このため以下ではこれらを同一視して議論する．

表 2. 本レポートで扱った解法．（今回 2 次 GL 法，陰的中点則は離散勾配法に等しいことに注意する．以下の本文中では各陽的 Symplectic 解法を「陽的 n 次法」と表記する．）

解法\次数	1 次	2 次	3 次	4 次
汎用解法（陽的）	陽的 Euler 法			4 次 Runge-Kutta（RK）法
汎用解法（陰的）	陰的 Euler 法	BDF2，陰的中点則	3 次 BDF	4 次 BDF
Symplectic 解法（陽的）	Symplectic Euler（SE）法	Störmer-Verlet（SV）法	Ruth の公式	Sanz-Serna の（SS）公式
Symplectic 解法（陰的）		2 次 Gauss-Legendre（GL）法		4 次 GL 法

2.3 解法の性能比較の方法

まず精度（次数）の比較（3.1 節）については，[1]p.73 で行われている方法と同じ方法を用いた．具体的には，分点数をNとし，n番目の分点でのq_i 数値解を$q_{i,n}$，解析解を$q_i(t_n)$ として誤差ε を

$$\varepsilon = \max_{1\le n\le N}\{\max_{1\le i\le 3}|q_{i,n} - q_i(t_n)|\}$$

と定義した上で$-\log_2(\varepsilon)$ の挙動を見ることにより精度（次数）を調べた．

次に安定性（3.2 節）については，安定領域を中心に議論した．

最後に各解法の速度（3.3 節）については，分点数を固定した上で，Python の time.perf_counter()（返り値の単位は sec）を用いて 10 回計算を実行し，これを統計処理して算出した．まず，time.perf_counter() についてはリファレンスにある通り高分解能であるようだが，具体的にどの程度の分解能があるのかわからなかったが，大小関係の比較の参考として用いるため，今回は小数点以下 2 桁に丸めることにした．統計処理については駒場の物理実験の教科書[2]を根拠に，実験標準偏

差 $\Delta\bar{x} = \frac{\sqrt{\sum(x_i-\bar{x})^2}}{\sqrt{n(n-1)}}$ （ここでx_i を一回あたりの計測時間の実測値，$\bar{x}$ をその 10 回に渡る平均値とした）を用いて，$\bar{x} \pm \Delta\bar{x}$ を 1 回あたりの計算時間として比較対象とすることにした．

2.4 使用した計算環境

使用した計算環境は

CPU: Intel Core i5, メモリ: 8 GB, OS: macOS, プログラミング言語: Python 3.8.5

である．

3. 本論

今回の実験では，2.1 で述べた問題で初期値が $(\boldsymbol{p}, \boldsymbol{q})^{\mathrm{T}} = (1, 1, 1, 0.5, 0, 0.5)^{\mathrm{T}}$ の場合を扱った（解析解の挙動を付録 6.1 節に示した）．また，最終時刻は常に 20.0 と固定して計算した．

3.1 各解法の精度（次数）についての実行結果

2.3 で定義した誤差ε に基づき，$-\log_2(\varepsilon)$ を求め，これを表にまとめると表 3 のようになった．これを見ると，p 次の解法について，高次の BDF を除き分点数を 2 倍にすると誤差ε が$\frac{1}{2^p}$ となっていることが分かる．高次の BDF については[3]に示されている安定性領域に 3.2 節で示す固有値が入っていないことが原因と考えた．

表 3. 初期条件 1 での$-\log_2(\varepsilon)$ について，分点数を 50 から 2 倍ずつ増加させたときの結果

解法\分点数	50	100	200	400	800	1600	3200
陽的 Euler 法	-13.33	-6.74	-2.47	-0.27	1.21	2.38	3.46
4 次 RK 法	5.66	9.61	13.62	17.62	21.62	25.62	29.62
陰的 Euler 法	-0.24	0.04	0.47	1.07	1.82	2.68	3.61
2 次 BDF	0.12	1.20	2.95	4.91	6.90	8.90	10.90
3 次 BDF	0.07	3.41	6.80	9.00	10.92	12.74	14.65
4 次 BDF	0.71	4.85	6.53	8.47	10.46	12.46	14.46
陽的 1 次法	2.02	3.06	4.03	5.01	6.00	6.97	7.95
陽的 2 次法	1.88	3.88	5.88	7.88	9.88	11.88	13.88
陽的 3 次法	7.73	11.01	14.13	17.19	20.22	23.24	26.24
陽的 4 次法	13.02	17.08	21.10	25.10	29.10	33.10	37.10
2 次 GL 法	1.06	2.91	4.90	6.90	8.90	10.90	12.90
4 次 GL 法	8.28	12.22	16.21	20.21	24.20	28.20	32.20

3.2 各解法の安定性とそれらを実行した時の挙動について

2.1 で示した運動方程式を $\begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{pmatrix} = J\nabla H = A\begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{pmatrix}$ とすると，$A = \begin{pmatrix} O & I \\ D & O \end{pmatrix}$ と表せ（ここでI は 3 次の単位行列，$D = \mathrm{Diag}(-2-\sqrt{2},\ -2,\ -2+\sqrt{2})$），固有値を求めると$\lambda = \pm\sqrt{2}i, \pm\sqrt{2 \pm \sqrt{2}}i$ となり全て純虚数である．第 6 章講義資料に示されている安定性領域と比較して，陽的 Euler 法はうまく動かないと分かる．また，3.1 節で述べたように同様の理由で高次の BDF もうまく動かないと考えられ，実際に 3 次の BDF は表 3 において，分点数が 50 のときに誤差が大きくなっている．4 次の BDF では表 3 の分点数が 50 の時の誤差が比較的大きいものの，2 次，3 次に比べると小さくなっていたが，この理由についてはよく分からなかった．

さらに数値解の挙動の確認として，分点数$N = 50$ のときの Hamiltonian の変化を Symplectic Euler 法，離散勾配法，4 次 RK 法について調べたところ，図 4 のようになった．図 4 より離散勾配法では Hamiltonian がきちんと保存していることがわかる．また，Symplectic Euler 法の場合では Hamiltonian の振動が見られるが，これは第 6 章の講義資料にあるように，Symplectic Euler 法による数値解は真の Hamiltonian から少しずれた影の Hamiltonian が定める Hamilton 系の軌道上に乗ることの現れであると考えられる．最後に 4 次の RK 法については，時間発展とともに徐々に発散するような傾向が見られ，構造保存型解法との違いが現れている．

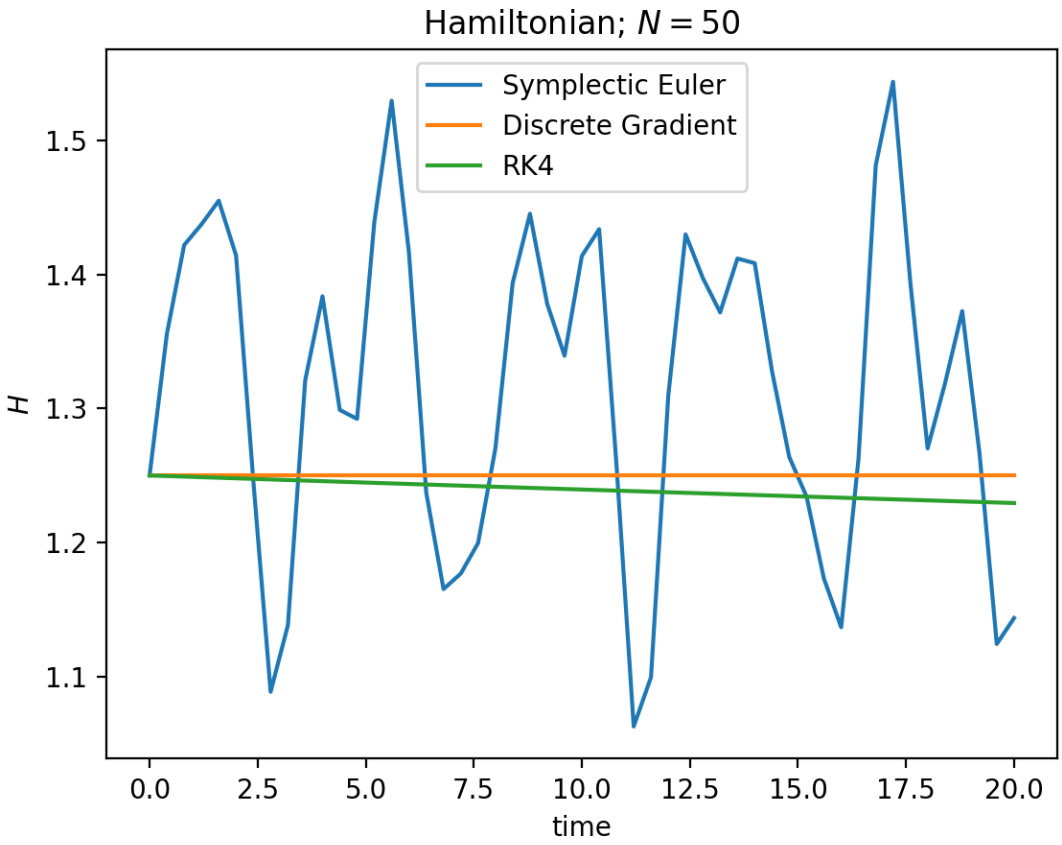


図 4. 分点数$N = 50$ のときの Hamiltonian の時間変化の比較

3.3 各解法の速度についての実行結果

2.3 で述べた比較方法により各解法の計算時間を比較した．表 3 より分点数$N = 1600$ 付近ではどの解法も比較的高い精度を持っていたことから，この分点数での速度は比較するのにふさわしいと考え，この分点数で固定して計算を実行すると表 5 の様になった（実装は付録 6.3 節の最後に示したが，平均と標準偏差を求めていたので，後者を後で$\sqrt{9} = 3$ で割ったものを実験標準偏差として表 5 に示した）．表 5 を見ると，陽的解法と陰的解法には実行時間に顕著な差が現れており，より大きな計算をする際には問題になりうると考えた．

表 5. 各解法の 1 回あたりの実行時間の比較（分点数$N = 1600$と固定，単位は全て sec）

解法	陽的 Euler 法	RK4	陰的 Euler 法	BDF2	BDF3	BDF4
平均値	0.0186164	0.08863958	0.21691384	0.29258357	0.323796261	0.361197227
実験標準偏差	0.00232799	0.0016938	0.00116614	0.00203118	0.0026014	0.001952313
丸めた後の結果	**0.02**	**0.09**	**0.22**	**0.29**	**0.32**	**0.36**
解法	陽的 1 次法	陽的 2 次法	陽的 3 次法	陽的 4 次法	GL2	GL4
平均値	0.01890945	0.03402499	0.05568568	0.10146848	0.684402868	1.501237229
実験標準偏差	0.00071477	0.00192863	0.00041315	0.00292427	0.002852309	0.004284237
丸めた後の結果	**0.02**	**0.03**	**0.06**	**0.10**	**0.68**	**1.50**

4. まとめと考察

今回の実験結果は，この問題に関しては精度と速度のいずれの観点から見ても，陽的な（になる）Symplectic 法を使うのが良いということを表している．しかし，[1]での高次の陽的な Symplectic 解法の導入部分では，Hamiltonian が可分であることを仮定しており，不可分の場合には Symplectic Euler 法が陰的になるだけでなく，このことからいずれの解法も成立しないと考えられる．これらのから，一般に構造保存型解法の間で比べた際 Symplectic 法か離散勾配法のどちらを使うべきは問題毎に異なると考えた．しかし Hamiltonian が可分な場合であれば，陽的であるため十分な速度を持ちかつ高精度である高次の陽的 Symplectic 法を選択するのが大抵の場合は良さそうである．他の解法については，RK 法が特に高精度かつ高速であり，使える場合には非常に有用であると実感した．

今回はかなり簡単なモデルを扱ってしまったため，離散勾配法の良さがわかりにくかったが，今後様々な問題に対して適用するとともに，他の方法も併せて構造保存型解法の理解を深めていきたい．

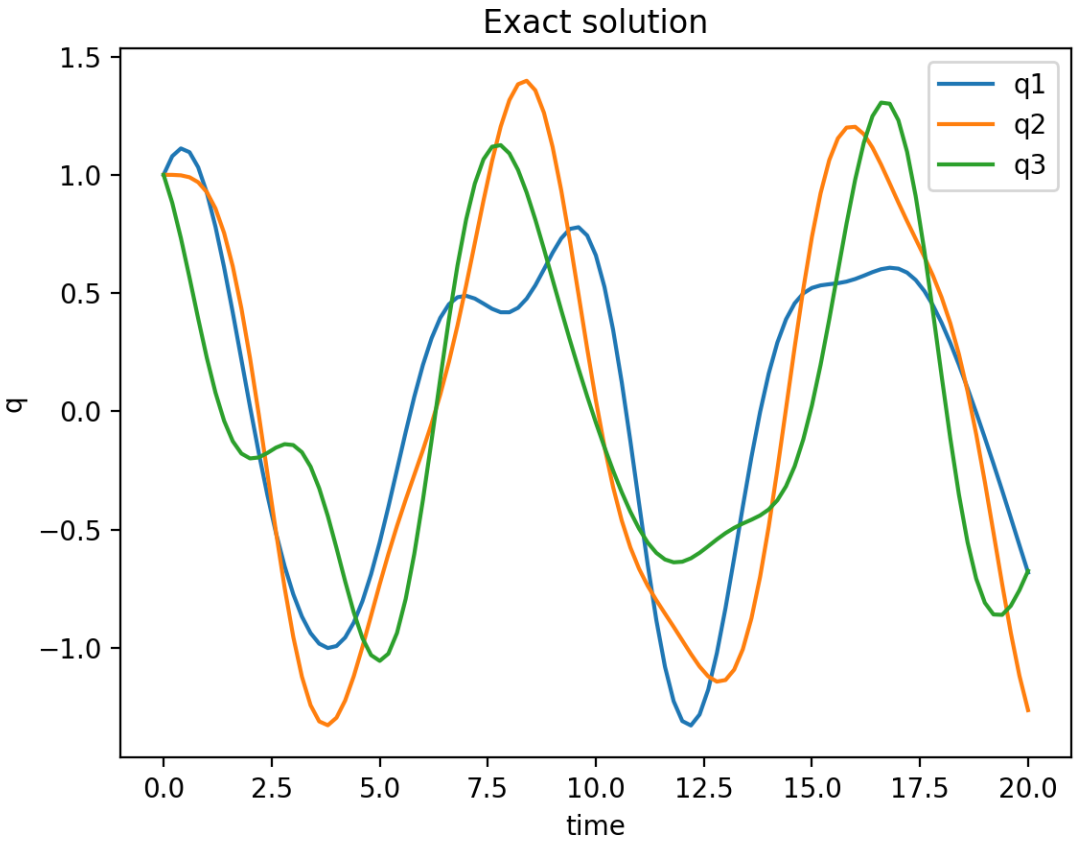
5. 参考文献

[1] 三井斌友. 小藤俊幸. 齋藤善弘. 微分方程式による計算科学入門. 共立出版, 2004, 43-80.

[2] 東京大学教養学部基礎物理学実験テキスト編集委員会 編. 基礎物理学実験. 学術図書出版社, 2019, 12-13.
[3] 後退微分法.Wikipedia. https://ja.wikipedia.org/wiki/後退微分法, (参照 2021-2-12).

6. 付録

6.1 初期値が $(\boldsymbol{p}, \boldsymbol{q})^{\mathrm{T}} = (1, 1, 1, 1, 0.5, 0, 0.5)^{\mathrm{T}}$ の時の解析解の挙動



6.2 メインのコード

```
# -*- coding: utf-8 -*-
import numpy as np
from scipy.optimize import root
import matplotlib.pyplot as plt
from time import perf_counter

init1 = np.array([[1.0,1.0,1.0],[0.5,0.0,-0.5]])
init2  = np.array([[0.5,-np.sqrt(2)/2,0.5],[0.0,0.0,0.0]])
init3  = np.array([[np.sqrt(2)/2,0.0,-np.sqrt(2)/2],[0.0,0.0,0.0]])
init4  = np.array([[0.5,np.sqrt(2)/2,0.5],[0.0,0.0,0.0]])
et = 20.0
```

```
n = 50
dt = et/n
tv = np.linspace(0,et,n+1)

def exp_E(init):
    label = 'symp_E'
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        q0 = np.array(q)
        p0 = np.array(p)
        q += p0*dt
        p -= np.array([2*q0[0]-q0[1], -q0[0]+2*q0[1]-q0[2], -q0[1]+2*q0[2]])*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def RK4(init):
    label = 'RK4'
    def f(y0):
        q0,p0 = y0[0],y0[1]
        return np.array([p0, -np.array([2*q0[0]-q0[1], -q0[0]+2*q0[1]-q0[2], -q0[1]+2*q0[2]])])
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        y0 = np.array([q,p])
        k1 = f(y0)
        k2 = f(y0 + k1*dt/2)
        k3 = f(y0 + k2*dt/2)
        k4 = f(y0 + k3*dt)
        q += (k1/6 + k2/3 + k3/3 + k4/6)[0]*dt
        p += (k1/6 + k2/3 + k3/3 + k4/6)[1]*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def BDF(init,mtd): #BDFの準備
    def f(y0):
        q0,p0 = np.array(y0[0:3]),np.array(y0[3:6])
        return np.array([p0[0],p0[1],p0[2], -(2*q0[0]-q0[1]), -(-q0[0]+2*q0[1]-q0[2]), -(-
q0[1]+2*q0[2])])
    def BDF1(y,y0): #1次
        return y - y0 - dt*f(y)
    def BDF2(y,y0,y1): #2次
        return 3/2*y - 2*y0 + 1/2*y1 - dt*f(y)
    def BDF3(y,y0,y1,y2): #3次
        return 11/6*y - 3*y0 + 3/2*y1 - 1/3*y2 - dt*f(y)
    def BDF4(y,y0,y1,y2,y3): #4次
        return 25/12*y - 4*y0 + 3*y1 - 4/3*y2 + 1/4*y3 - dt*f(y)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]] #解を格納
```

```
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    mtdtxt = str() #k次のBDFのラベル
    for i in range(n): #BDFの実行
        if  mtd == 1 or i==0:
            mtdtxt = 'BDF1'
            z0 = np.array([y1[i][0],y2[i][0],y3[i][0], y1[i][1],y2[i][1],y3[i][1]])
            sol = root(lambda x: BDF1(x,z0), x0=z0)
            y1.append([sol.x[0],sol.x[3]])
            y2.append([sol.x[1],sol.x[4]])
            y3.append([sol.x[2],sol.x[5]])
        elif mtd == 2 or i==1:
            mtdtxt = 'BDF2'
            z0 = np.array([y1[i][0],y2[i][0],y3[i][0], y1[i][1],y2[i][1],y3[i][1]])
            z1 = np.array([y1[i-1][0],y2[i-1][0],y3[i-1][0], y1[i-1][1],y2[i-1][1],y3[i-1][1]])
            sol = root(lambda x: BDF2(x,z0,z1), x0=z0)
            y1.append([sol.x[0],sol.x[3]])
            y2.append([sol.x[1],sol.x[4]])
            y3.append([sol.x[2],sol.x[5]])
        elif mtd == 3 or i==2:
            mtdtxt = 'BDF3'
            z0 = np.array([y1[i][0],y2[i][0],y3[i][0], y1[i][1],y2[i][1],y3[i][1]])
            z1 = np.array([y1[i-1][0],y2[i-1][0],y3[i-1][0], y1[i-1][1],y2[i-1][1],y3[i-1][1]])
            z2 = np.array([y1[i-2][0],y2[i-2][0],y3[i-2][0], y1[i-2][1],y2[i-2][1],y3[i-2][1]])
            sol = root(lambda x: BDF3(x,z0,z1,z2,), x0=z0)
            y1.append([sol.x[0],sol.x[3]])
            y2.append([sol.x[1],sol.x[4]])
            y3.append([sol.x[2],sol.x[5]])
        else:
            mtdtxt = 'BDF4'
            z0 = np.array([y1[i][0],y2[i][0],y3[i][0], y1[i][1],y2[i][1],y3[i][1]])
            z1 = np.array([y1[i-1][0],y2[i-1][0],y3[i-1][0], y1[i-1][1],y2[i-1][1],y3[i-1][1]])
            z2 = np.array([y1[i-2][0],y2[i-2][0],y3[i-2][0], y1[i-2][1],y2[i-2][1],y3[i-2][1]])
            z3 = np.array([y1[i-3][0],y2[i-3][0],y3[i-3][0], y1[i-3][1],y2[i-3][1],y3[i-3][1]])
            sol = root(lambda x: BDF4(x,z0,z1,z2,z3), x0=z0)
            y1.append([sol.x[0],sol.x[3]])
            y2.append([sol.x[1],sol.x[4]])
            y3.append([sol.x[2],sol.x[5]])
    return y1,y2,y3,mtdtxt #解のリストとラベルを出力


def symp_E(init):
    label = 'symp_E'
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        q += np.array(p*dt)
        p += - np.array([2*q[0]-q[1], -q[0]+2*q[1]-q[2], -q[1]+2*q[2]])*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def SV(init):
    label = 'SV'
    def Hq(q):
        return np.array([2*q[0]-q[1], -q[0]+2*q[1]-q[2], -q[1]+2*q[2]])
```

```
    def Hp(p):
        return np.array(p)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        p += - 1/2*Hq(q)*dt
        q += Hp(p)*dt
        p += - 1/2*Hq(q)*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def Ruth(init):
    label = 'Ruth'
    def Hq(q):
        return np.array([2*q[0]-q[1], -q[0]+2*q[1]-q[2], -q[1]+2*q[2]])
    def Hp(p):
        return np.array(p)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        q += 7/24*Hp(p)*dt
        p += - 2/3*Hq(q)*dt
        q += 3/4*Hp(p)*dt
        p += - -2/3*Hq(q)*dt
        q += -1/24*Hp(p)*dt
        p += - Hq(q)*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def SS(init):
    label='SS'
    def Hq(q):
        return np.array([2*q[0]-q[1], -q[0]+2*q[1]-q[2], -q[1]+2*q[2]])
    def Hp(p):
        return np.array(p)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        q += 7/48*Hp(p)*dt
        p += - 1/3*Hq(q)*dt
        q += 3/8*Hp(p)*dt
        p += - -1/3*Hq(q)*dt
        q += -1/48*Hp(p)*dt
        p += - Hq(q)*dt
        q += -1/48*Hp(p)*dt
        p += - -1/3*Hq(q)*dt
        q += 3/8*Hp(p)*dt
```

```
        p += - 1/3*Hq(q)*dt
        q += 7/48*Hp(p)*dt
        y1.append([q[0],p[0]])
        y2.append([q[1],p[1]])
        y3.append([q[2],p[2]])
    return y1,y2,y3,label

def GL4(init):
    label = 'GL4'
    def f(y):
        q,p = np.array(y[0:3]),np.array(y[3:6])
        return np.array([p[0],p[1],p[2], -(2*q[0]-q[1]), -(-q[0]+2*q[1]-q[2]), -(-q[1]+2*q[2])])
    def g(X,y0):
        X1,X2 = np.array(X[0:6]),np.array(X[6:12])
        return X - np.concatenate([y0 + dt*(1/4*f(X1)+(3-2*np.sqrt(3))/12*f(X2)), y0 +
dt*((3+2*np.sqrt(3))/12*f(X1)+1/4*f(X2))], axis=0)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        y0 = np.array([y1[-1][0],y2[-1][0],y3[-1][0], y1[-1][1],y2[-1][1],y3[-1][1]])
        X0 = np.concatenate([y0,y0],axis=0)
        sol = root(lambda X: g(X,y0),x0=X0)
        X1,X2 = sol.x[0:6], sol.x[6:12]
        y = y0 + dt*(f(X1)+f(X2))/2
        y1.append([y[0],y[3]])
        y2.append([y[1],y[4]])
        y3.append([y[2],y[5]])
    return y1,y2,y3,label

def disc_grad(init,ind):
    label = 'disc_grad'
    def H(y):
        return (y[0]**2 + (y[0]-y[1])**2 + (y[1]-y[2])**2 + y[2]**2 + y[3]**2 + y[4]**2 +
y[5]**2)/2
    def nablaH(y):
        q,p = np.array(y[0:3]),np.array(y[3:6])
        return np.array([2*q[0]-q[1],-q[0]+2*q[1]-q[2],-q[1]+2*q[2], p[0],p[1],p[2]])
    def g(y,y0):
        q,p = np.array(y[0:3]),np.array(y[3:6])
        q0,p0 = np.array(y0[0:3]),np.array(y0[3:6])
        qm,pm = (q+q0)/2,(p+p0)/2
        J = np.array([[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1],[-1,0,0,0,0,0],[0,-
1,0,0,0,0],[0,0,-1,0,0,0]])
        if ind == 1: #AVF, GL2(陰的中点則（台形則))
            dg = nablaH((y+y0)/2)
        else: #Gonzalez
            z = (y+y0)/2
            dg = nablaH(z) + (H(y)-H(y0)-np.dot(z,y-y0))/np.dot(y-y0,y-y0) * (y-y0)
        return y - y0 - dt*np.matmul(J,dg)
    q = np.array(init[0])
    p = np.array(init[1])
    y1 = [[q[0],p[0]]]
    y2 = [[q[1],p[1]]]
    y3 = [[q[2],p[2]]]
    for i in range(n):
        y0 = np.array([y1[-1][0],y2[-1][0],y3[-1][0], y1[-1][1],y2[-1][1],y3[-1][1]])
        if ind == 1:
```

```
            sol = root(lambda x: g(x,y0),x0=y0)
        else:
            sol = root(lambda x: g(x,y0),x0=y0+np.ones(6)*0.1)
        y1.append([sol.x[0],sol.x[3]])
        y2.append([sol.x[1],sol.x[4]])
        y3.append([sol.x[2],sol.x[5]])
    return y1,y2,y3,label
```

6.3 グラフ描画などに用いたコード

```
def exactq(ini):
    if ini == 1:
        return ((2-np.sqrt(2))/2 * np.matmul(np.array([[1/2,-np.sqrt(2)/2,1/2]]).T,
np.array([np.cos(np.sqrt(2+np.sqrt(2))*tv)]))
                    + 1/2 * np.matmul(np.array([[np.sqrt(2)/2,0,-np.sqrt(2)/2]]).T,
np.array([np.sin(np.sqrt(2)*tv)]))
                    + (2+np.sqrt(2))/2 * np.matmul(np.array([[1/2,np.sqrt(2)/2,1/2]]).T,
np.array([np.cos(np.sqrt(2-np.sqrt(2))*tv)])))
    elif ini == 2:
        return np.matmul(np.array([[1/2,-np.sqrt(2)/2,1/2]]).T,
np.array([np.cos(np.sqrt(2+np.sqrt(2))*tv)]))
    elif ini == 3:
        return np.matmul(np.array([[np.sqrt(2)/2,0,-np.sqrt(2)/2]]).T,
np.array([np.cos(np.sqrt(2)*tv)]))
    elif ini == 4:
        return np.matmul(np.array([[1/2,np.sqrt(2)/2,1/2]]).T, np.array([np.cos(np.sqrt(2-
np.sqrt(2))*tv)]))

def H_t(sol):
    q1,p1 = [y[0] for y in sol[0]],[y[1] for y in sol[0]]
    q2,p2 = [y[0] for y in sol[1]],[y[1] for y in sol[1]]
    q3,p3 = [y[0] for y in sol[2]],[y[1] for y in sol[2]]
    q1,p1 = np.array(q1),np.array(p1)
    q2,p2 = np.array(q2),np.array(p2)
    q3,p3 = np.array(q3),np.array(p3)
    T = (p1**2 + p2**2 + p3**2)/2
    U = (q1**2 + (q2-q1)**2 + (q3-q2)**2 + q3**2)/2
    return T+U

def tq_graph(sol):
    q1,q2,q3 = [y[0] for y in sol[0]],[y[0] for y in sol[1]],[y[0] for y in sol[2]]
    lab = sol[3]
    fig,ax = plt.subplots()
    ax.set_xlabel('time')
    ax.set_ylabel('$q$')
    ax.plot(tv,q1,label='$q_1$')
    ax.plot(tv,q2,label='$q_2$')
    ax.plot(tv,q3,label='$q_3$')
    ax.legend()
    plt.show()

def qp_graph(sol):
    q1,p1 = [y[0] for y in sol[0]],[y[1] for y in sol[0]]
    q2,p2 = [y[0] for y in sol[1]],[y[1] for y in sol[1]]
    q3,p3 = [y[0] for y in sol[2]],[y[1] for y in sol[2]]
    fig,ax = plt.subplots()
    ax.set_xlabel('$q$')
    ax.set_ylabel('$p$')
    ax.plot(q1,p1,label='1')
```

```
    ax.plot(q2,p2,label='2')
    ax.plot(q3,p3,label='3')
    ax.legend()
    plt.show()

def tH_graph(sol):
    lab = sol[3]
    exactH = np.ones(n+1)*H_t(sol)[0]
    fig,ax = plt.subplots()
    ax.set_xlabel('time')
    ax.set_ylabel('$H$')
    ax.plot(tv,exactH,label="exact $H$")
    ax.plot(tv,H_t(sol),label=lab)
    ax.legend()
    plt.show()

def diff_graph(sol,ini):
    q1,q2,q3 = [y[0] for y in sol[0]],[y[0] for y in sol[1]],[y[0] for y in sol[2]]
    lab = sol[3]
    exact_sol = exactq(ini)
    fig,ax = plt.subplots()
    ax.set_xlabel('time')
    ax.set_ylabel('$\Delta q$')
    ax.plot(tv,q1-exact_sol[0],label='$\Delta q_1$')
    ax.plot(tv,q2-exact_sol[1],label='$\Delta q_2$')
    ax.plot(tv,q3-exact_sol[2],label='$\Delta q_3$')
    ax.legend()
    plt.show()

def calc_error(func,init,ini):
    global n
    global dt
    global tv
    n = 25
    errs = []
    for i in range(7):
        n = n*2
        dt = et/n
        tv = np.linspace(0,et,n+1)
        eq = exactq(ini)
        sol = func(init)
        q = np.array([[y[0] for y in sol[0]],[y[0] for y in sol[1]],[y[0] for y in sol[2]]])
        idx = np.unravel_index(np.argmax(np.abs(q-eq)),eq.shape)
        error = np.abs(q-eq)[idx]
        errs.append(-np.log2(error))
    print(errs)

def H_comp(init): #HamiltonianをSymplectic Euler法，離散勾配法，4次RK法について比較
    sol_SE = symp_E(init)
    sol_DG = disc_grad(init,1)
    sol_RK4 = RK4(init)
    fig,ax = plt.subplots()
    ax.set_xlabel('time')
    ax.set_ylabel('$H$')
    ax.plot(tv,H_t(sol_SE),label="Symplectic Euler")
    ax.plot(tv,H_t(sol_DG),label="Discrete Gradient")
    ax.plot(tv,H_t(sol_RK4),label="RK4")
    ax.set_title('Hamiltonian; $N=50$')
    ax.legend()
    plt.show()
```

```
def calc_time(func,init):
    rt =[]
    for i in range(10):
        start = perf_counter()
        func(init)
        end = perf_counter()
        rt.append(end-start)
    rt = np.array(rt)
    print(np.mean(rt),np.std(rt))
```