計算物理学 第二単元レポート

佐々木良輔

3-A.

3-A-1. **調和振動子の厳密解**

調和振動子の微分方程式は以下の通りである.

$$\begin{aligned} \frac{\mathrm{d}v}{\mathrm{d}t} &= -x \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= v \end{aligned} \tag{1}$$

すなわち

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -x \tag{2}$$

であり初期条件を $x(0) = 0$, $v(0) = 1$ とすれば解は以下のようになる.

$$\begin{aligned} x(t) &= \sin t \\ v(t) &= \cos t \end{aligned} \tag{3}$$

3-A-2. **メインプログラムについて**

図 1 にメインプログラムの PAD 図を示す. またソースコードは補遺に示す. このプログラムでは 2^{-2} から 2^{-12} までの各時間刻み τ について, (1) 式の微分方程式を $t = 0$ から $t_{\mathrm{end}} = 8$ まで Runge-Kutta 法で解く. Runge-Kutta 法関数に関しては次節で説明する. ここで初期値は $x = 0$, $v = 1$ としている. したがって微分方程式の厳密解は (3) 式であり, 出力段では時間刻み τ 及び時刻 $t = 8$ での厳密解と数値解の差を出力している.

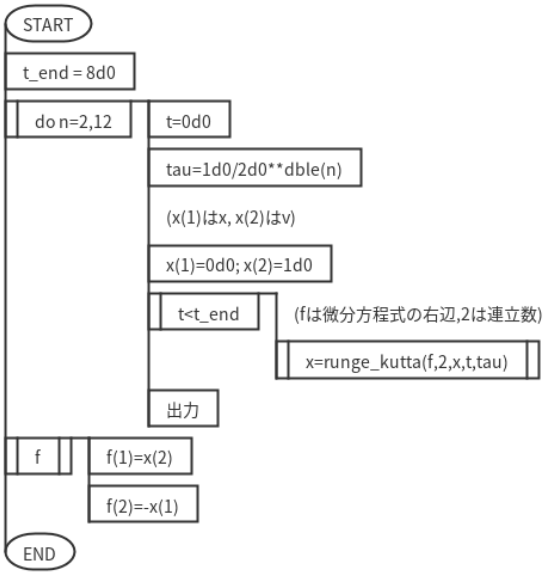

図 1　メインプログラムの PAD 図

3-A-3. Runge-Kutta 関数について

図 2 に Runge-Kutta 法関数の PAD 図を示す. またソースコードは補遺に示す. この関数では関数 arg を 4 段 4 次の Runge-Kutta 法で 1 ステップ (時間刻み τ) 計算する. arg は倍精度 1 次元配列型の関数であり以下のような微分方程式の右辺 $\vec{f}(\vec{q},t)$ に相当する.

$$\frac{\mathrm{d}\vec{q}}{\mathrm{d}t} = \vec{f}(\vec{q},t) \tag{4}$$

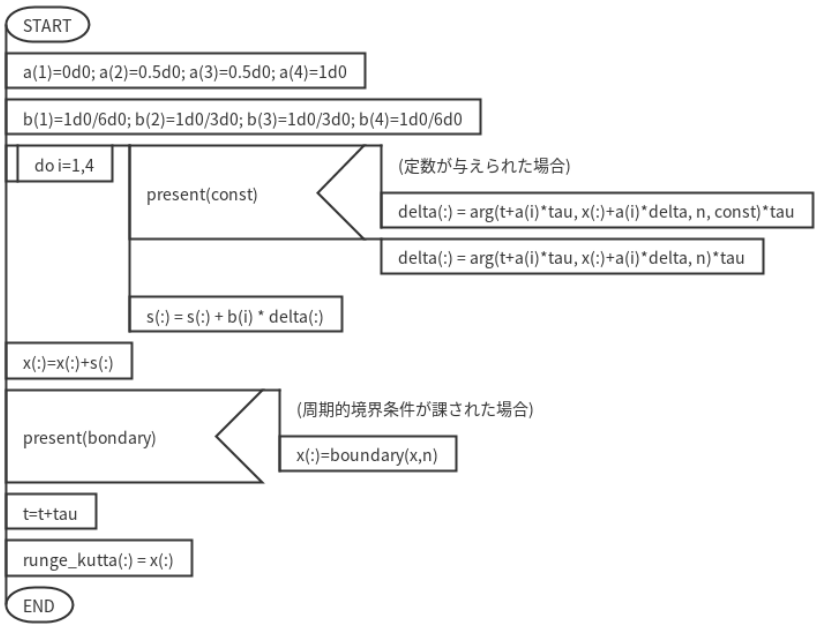

図 2　Runge-Kutta 法の PAD 図

3-A-4. **出力結果**

図 3 にプログラムの出力結果を示す．これは両対数グラフであり横軸は時間刻み τ, 縦軸は厳密解からの誤差である．また直線は誤差の線形な部分を最小二乗 Fit した直線である．

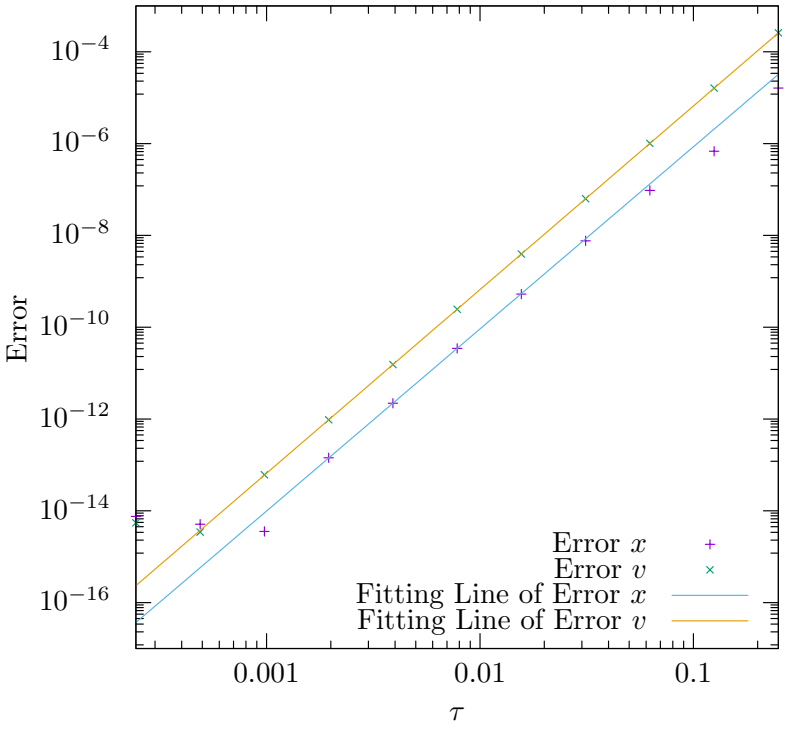

図 3　3-A. の出力結果

3-A-5. **考察**

図 3 において, 各 Fitting 曲線 err_x, err_v は以下のようになった．

$$\mathrm{err}_x(\tau) = 0.00792(1 \pm 0.5) \times \tau^{3.96(1\pm0.0)} \tag{5}$$

$$\mathrm{err}_v(\tau) = 0.0662(1 \pm 0.0) \times \tau^{4.00(1\pm0.0)} \tag{6}$$

以上から Runge-Kutta 法の誤差は τ に対して 4 次で変化していることがわかる．今回実装した Runge-Kutta 法は 4 段 4 次のものだったので, 期待した通りの挙動をしていることがわかる．一方で τ が小さい領域では誤差が直線の系列から離れていることがわかる．これは分割数が増えたことに依る丸め誤差の蓄積が原因であると考えられる．実際, 倍精度浮動小数点の精度は 10 進数で 16 桁程度であり, 近い桁で誤差が大きくなっていることがわかる．

3-A-6. **感想**

Runge-Kutta 法の実装では最初に誤差が τ の 2 乗程度の挙動を示し, この問題の解決に多大な時間を要した. 原因は Runge-Kutta のループにおいて時間が最後の 1 ステップ分ずれて計算されていたことであった. 私は普段組み込み機器でプログラムをしているため, プログラムの記述自体には慣れていたが, 数値計算特有の難しさや注意すべき点を認識する良い機会だった.

3-B.

3-B-1. **振動的外力の加わった減衰振り子**

外力, 減衰の無い振り子の微分方程式は

$$ml\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + mg\sin\theta = 0 \tag{7}$$

である. これに減衰項 $(k\mathrm{d}\theta/\mathrm{d}t)$ 及び角振動数 ω_e の周期的な外力が加わったとすると微分方程式は

$$ml\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + k\frac{\mathrm{d}\theta}{\mathrm{d}t} + mg\sin\theta = F\cos\omega_e t \tag{8}$$

となる. ここで $\hat{t} = t\sqrt{g/l}$ と無次元化すると

$$\begin{aligned} \frac{\mathrm{d}^2\theta}{\mathrm{d}\hat{t}^2} + \frac{k}{m}\sqrt{\frac{1}{lg}}\frac{\mathrm{d}\theta}{\mathrm{d}\hat{t}} + \sin\theta &= \frac{F}{mg}\cos\omega_e\sqrt{\frac{l}{g}}\hat{t} \\ \frac{\mathrm{d}^2\theta}{\mathrm{d}\hat{t}^2} + \frac{1}{Q}\frac{\mathrm{d}\theta}{\mathrm{d}\hat{t}} + \sin\theta &= G\cos\Omega\hat{t} \end{aligned} \tag{9}$$

と無次元化される. これは 1 階の微分方程式に分解すると $\omega = \dot{\theta}$, $\phi = \Omega\hat{t}$ として

$$\begin{aligned} \frac{\mathrm{d}\omega}{\mathrm{d}\hat{t}} &= -\frac{1}{Q}\omega - \sin\theta + G\cos\phi \\ \frac{\mathrm{d}\theta}{\mathrm{d}\hat{t}} &= \omega \\ \frac{\mathrm{d}\phi}{\mathrm{d}\hat{t}} &= \Omega \end{aligned} \tag{10}$$

を得る.

3-B-2. **メインプログラムについて**

図 4 にメインプログラムの PAD 図を示す. またソースコードは補遺に示す. このプログラムでは以下の振動的外力の加わった減衰振り子の微分方程式 (10) を Runge-Kutta 法で解く. Runge-Kutta 法関数は前問で用いたものと同じものである. ここで定数 Q, G, Ω の値は表 1 の値とする. 時間刻みは $\tau = 2\pi/\Omega \times 10^{-3}$ とし $\phi \in [0, 2\pi)$, $\theta \in (-\pi, \pi]$ の周期的境界条件を課している.

表 1　定数 Q, G, Ω

条件	Q	G	Ω
(a)	2	0.9	2/3
(b)	2	1.07	2/3
(f)	2	1.47	2/3
(g)	2	1.5	2/3

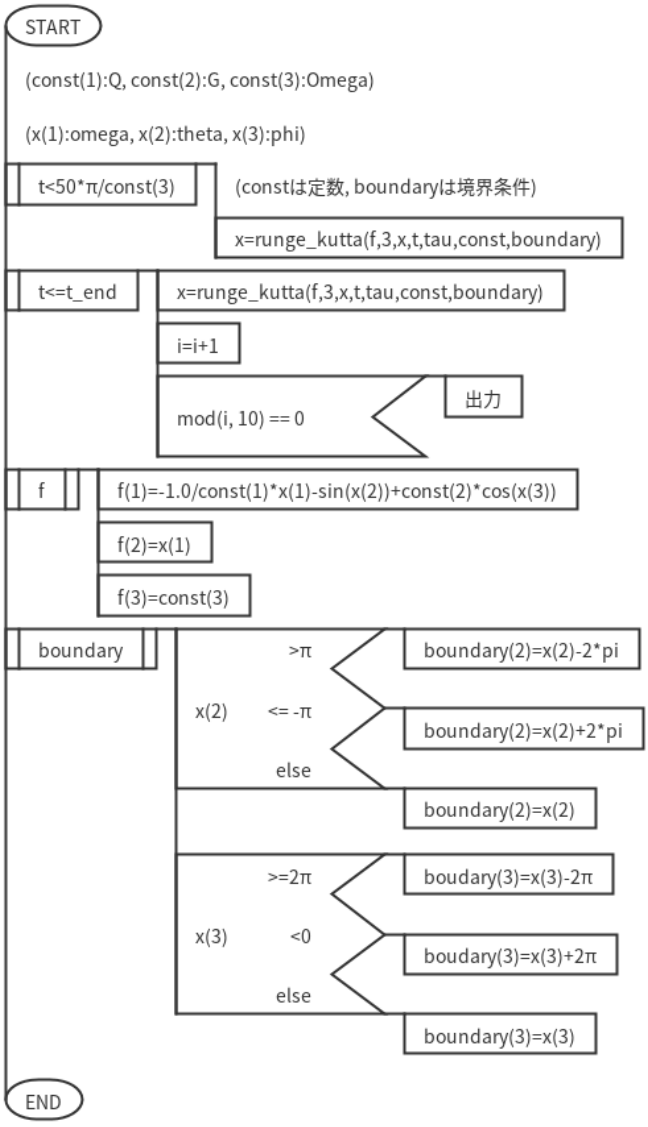

図 4　メインプログラムの PAD 図

3-B-3. **出力結果**

図 5 から図 8 に各条件での出力結果を示す. 1 枚目の画像は Attractor の画像, 2 枚目はその θ-ω 平面への射影, 3 枚目は Poincaré 断面である. Attractor の画像は $50 \times 2\pi/\Omega < t \leq 150 \times 2\pi/\Omega$ の範囲で $2\pi/\Omega \times 10^{-2}$ ごとにプロットしている. また Poincaré 断面の画像は $50 \times 2\pi/\Omega < t \leq 1050 \times 2\pi/\Omega$ の範囲で $2\pi/\Omega$ ごとにプロットしている.

3-B-4. **考察**

(a) の場合の Poincaré 断面を見ると常に 1 点に集中していることがわかる. これは ϕ の 1 周期ごとに周期的な運動をしていることを示している. もともと ϕ は周期的な外力項の位相であったので, 定常的な運動がその周期に沿って周期的であるということは運動が強制振動振り子のような挙動をとっていると理解できる. 一方で (b), (f) の場合は Poincaré 断面は 2 点, 4 点に集中していることがわかる. これはそれぞれ ϕ の 2 周期, 4 周期で周期的な運動であることを示している. また (g) の場合は Poincaré 断面は非常に多数の点が見えている. これは外力の周期に対して運動がほとんど周期性を持たず, カオスを示していることがわかる.

3-B-4. **感想**

Runge-Kutta 法の関数を汎用的に設計したので実装自体は容易だった. 一方でこれだけシンプルな系とプログラムからカオスが現れるということからも, 現実の複雑系のシミュレーションが非常に困難であることが想像できた. またカオスの数値計算においては浮動小数点の丸め誤差やプロセッサのアーキテクチャの差が問題になることも有りうると思い, 定量的な議論が難しいと感じた.

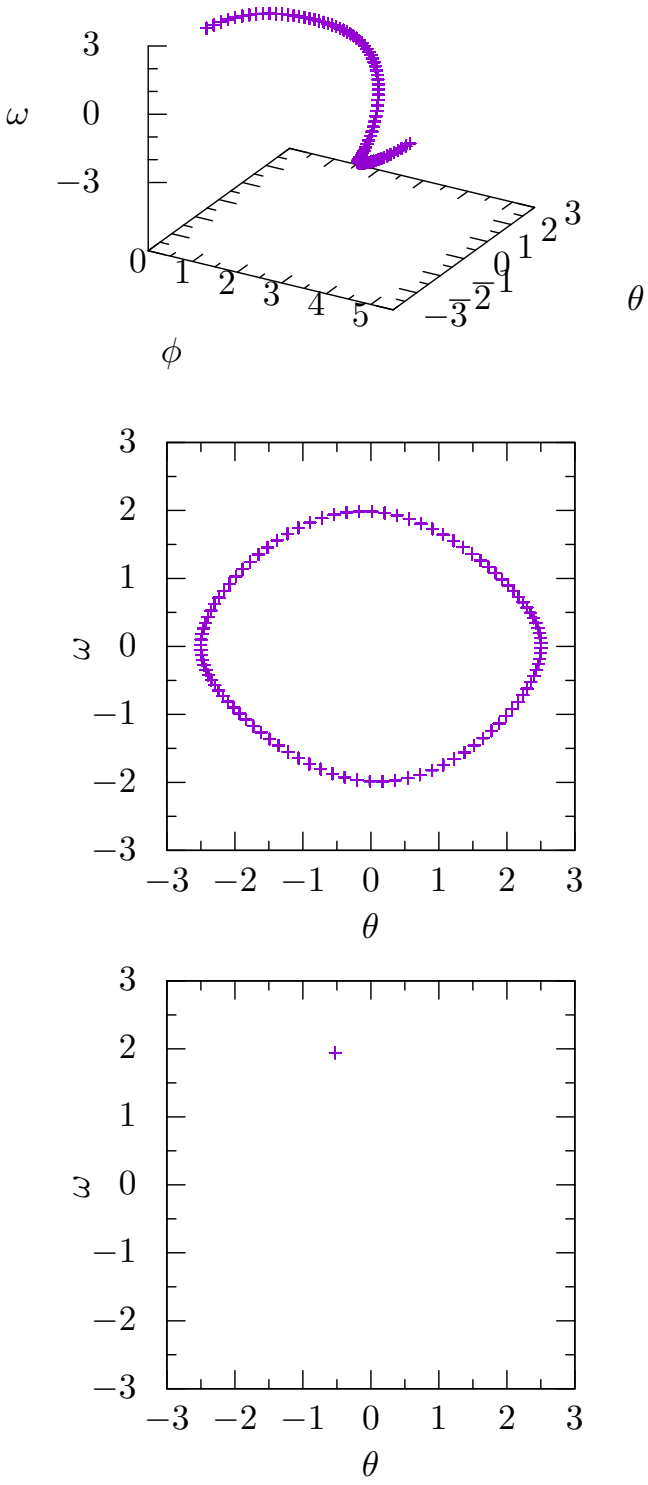

図 5　(a) の結果

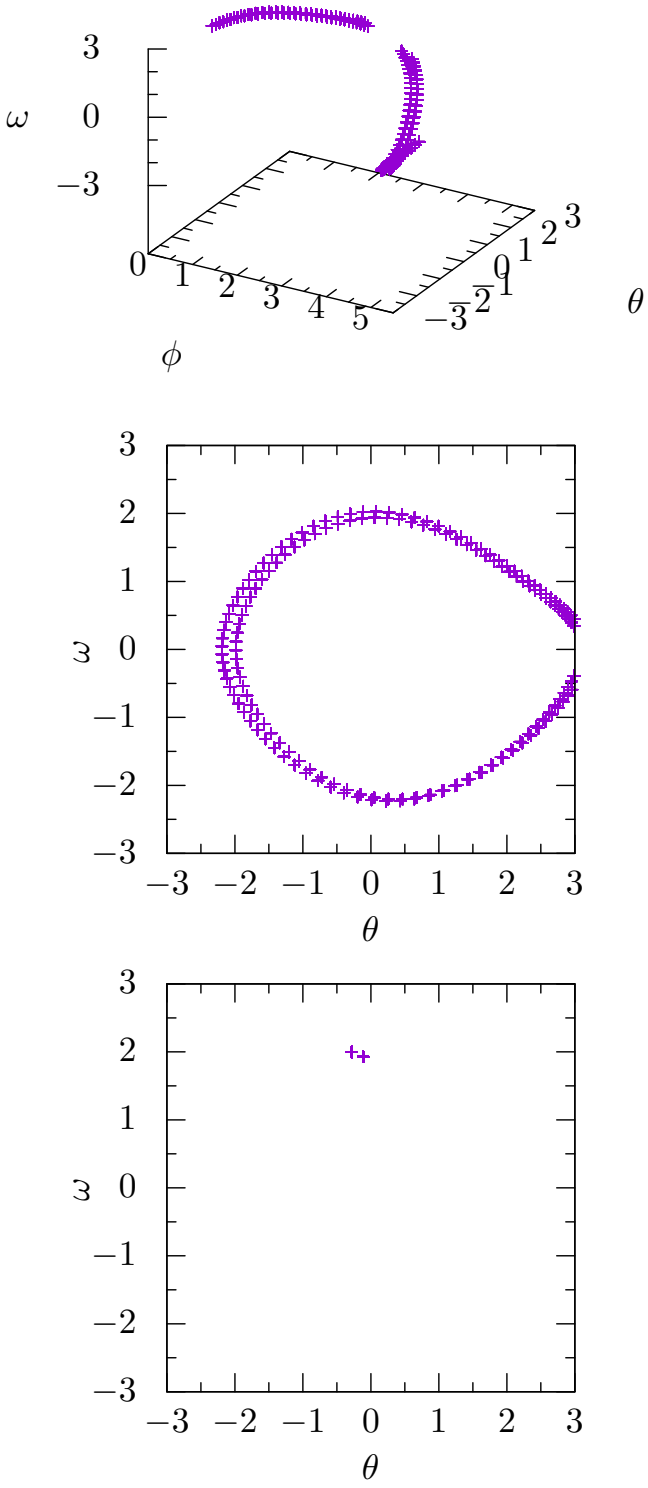

図 6　(b) の結果

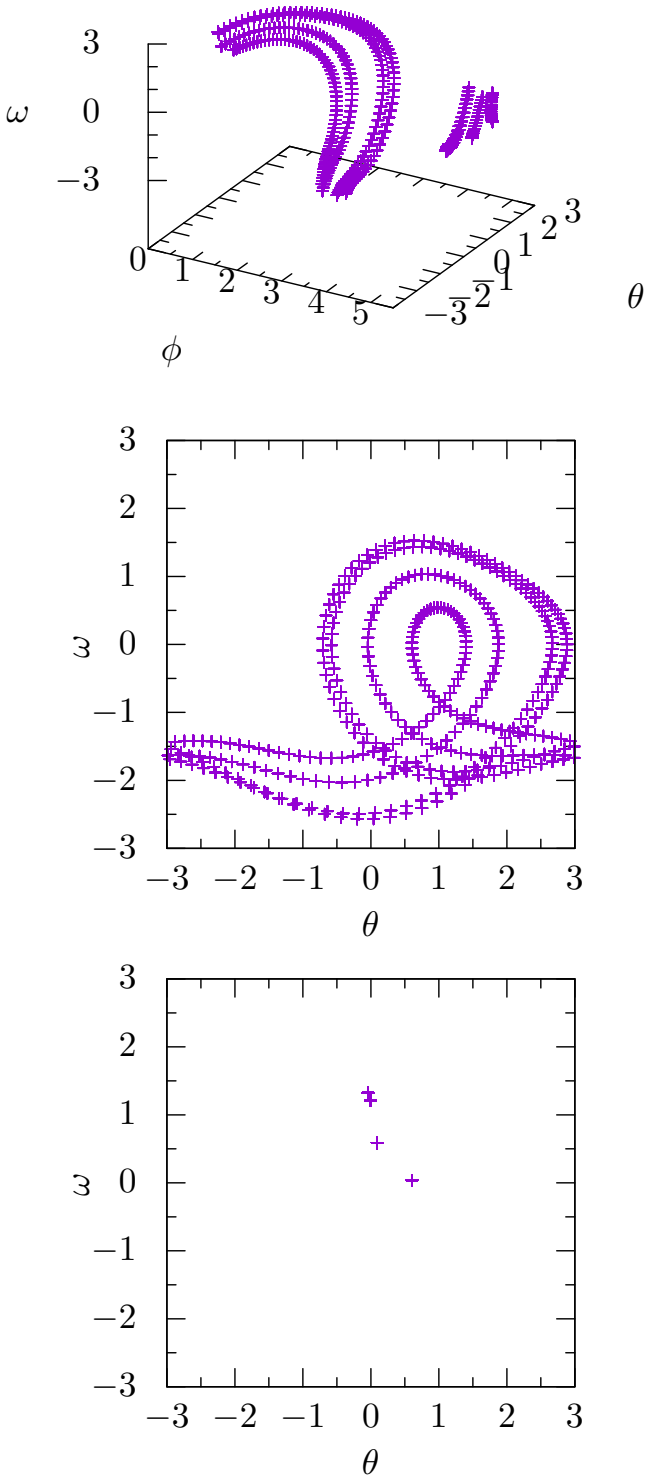

図 7　(f) の結果

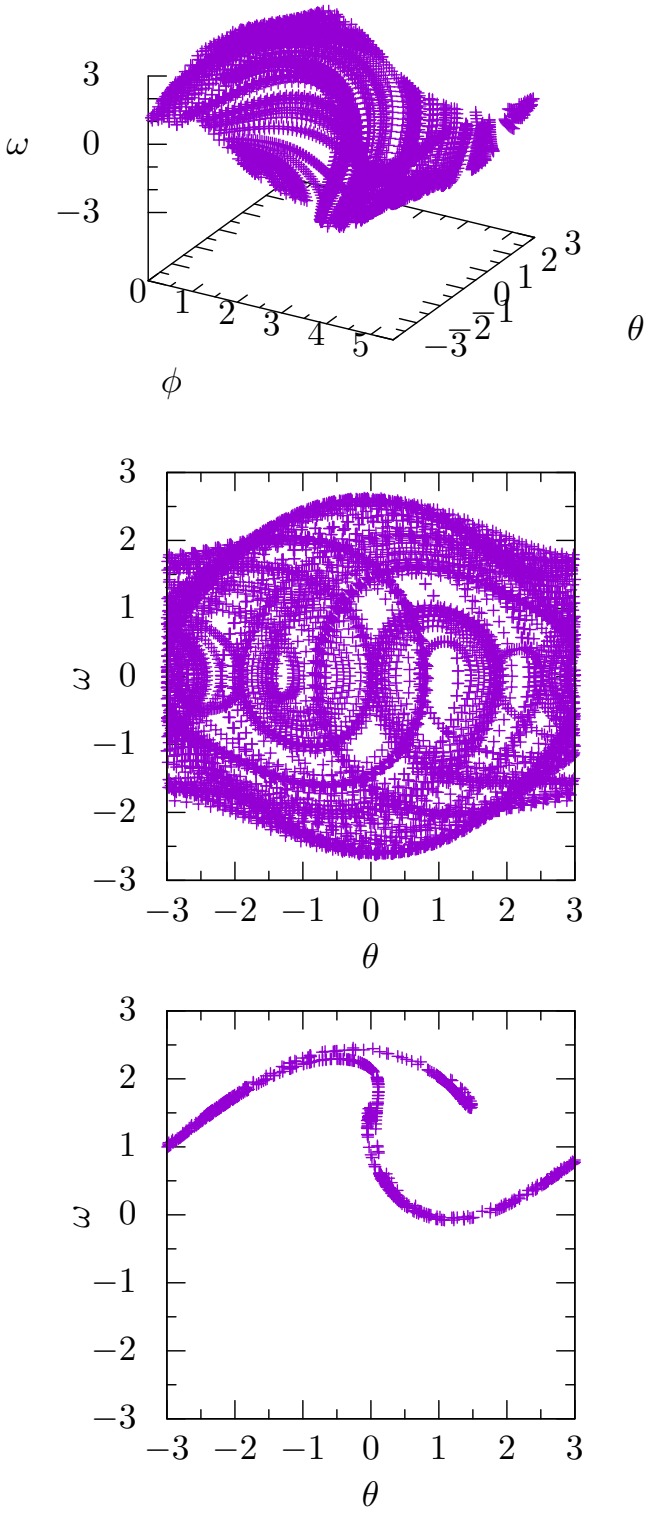

図 8　(g) の結果

補遺 A　ソースコード

Listing A.1　Runge-Kutta 関数のソースコード

```
module differential
  implicit none

  contains
  !runge_kutta の計算を1 ステップ進める
  !arg: 微分方程式; n: 連立する数; init: 初期値; t_begin: 計算開始; t_end: 計
      算終了; tau: 刻み幅;
  !const: 定数 (optional); boundary: 境界条件 (optional)
  function runge_kutta(arg, n, init, t_begin, tau, const, boundary)
    implicit none
    interface微分方程式
      !
      function arg(t, x, n, const)
        INTEGER, INTENT(IN) :: n
        DOUBLE PRECISION :: arg(n)
        DOUBLE PRECISION, INTENT(IN) :: x(:), t
        DOUBLE PRECISION, OPTIONAL :: const(:)
      end function arg
      ! 境界条件の計算関数
      function boundary(x, n)
        implicit none
        INTEGER, INTENT(IN) :: n
        DOUBLE PRECISION, INTENT(IN) :: x(:)
        DOUBLE PRECISION, OPTIONAL :: boundary(n)
      end function boundary
    end interface
    INTEGER :: i
    ! 段数
    INTEGER, PARAMETER :: order = 4
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION, INTENT(in) :: init(:), tau
    DOUBLE PRECISION, INTENT(INOUT) :: t_begin
    DOUBLE PRECISION :: runge_kutta(n), x(n), t, s(n), delta(n)
    DOUBLE PRECISION, OPTIONAL :: const(:)
    DOUBLE PRECISION :: a(order), b(order)
    a(1)=0d0; a(2)=0.5d0; a(3)=0.5d0; a(4)=1d0
    b(1)=1d0/6d0; b(2)=1d0/3d0; b(3)=1d0/3d0; b(4)=1d0/6d0
    x(:) = init(:)
```

```
38      t = t_begin
39
40      s = 0; delta = 0
41      do i = 1, order
42        ! optional の定数が与えられている場合はそれを含む計算を実行
43        if (PRESENT(const)) then
44          delta(:) = arg(t+a(i)*tau, x(:) + a(i) * delta, n, const)*tau
45        else
46          delta(:) = arg(t+a(i)*tau, x(:) + a(i) * delta, n)*tau
47        end if
48        s(:) = s(:) + b(i) * delta(:)
49      end do
50      x(:) = x(:) + s(:)
51      t = t + tau
52      t_begin = t
53      ! optional の境界条件が与えられている場合はそれを考慮
54      if (PRESENT(boundary)) then
55        x(:) = boundary(x, n)
56      end if
57      runge_kutta(:) = x(:)
58    end function runge_kutta
59  end module differential
```

Listing A.2　3-A のソースコード

```
program main
  use differential
  implicit none
  DOUBLE PRECISION :: t = 0d0, x(2), tau = 0.01d0, t_end = 8d0
  INTEGER :: n, i
  x(1) = 0d0; x(2) = 1d0

  do n = 2, 12
    t = 0d0
    tau = 1d0 / 2d0**dble(n)
    x(1) = 0d0; x(2) = 1d0
    do while (t < t_end)
        x = runge_kutta(f, 2, x, t, tau)
    end do
    WRITE(*, *) tau, t, abs(x(1) - x_true(t)), abs(x(2) - v_true(t))
  end do

  contains
  function f(tp, xp, n, const)
    implicit none
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION :: f(n)
    DOUBLE PRECISION, INTENT(IN) :: xp(:), tp
    DOUBLE PRECISION, OPTIONAL :: const(:)
    f(1) = xp(2)
    f(2) = -xp(1)
  end function f
  function x_true(tp)
    implicit none
    DOUBLE PRECISION :: x_true
    DOUBLE PRECISION, INTENT(IN) :: tp
    x_true = sin(tp)
  end function x_true
  function v_true(tp)
    implicit none
    DOUBLE PRECISION :: v_true
    DOUBLE PRECISION, INTENT(IN) :: tp
    v_true = cos(tp)
  end function v_true
end program main
```

Listing A.3　3-b のソースコード

```
program main
  use differential
  implicit none
  DOUBLE PRECISION, PARAMETER :: pi = 4d0*atan(1d0)
  DOUBLE PRECISION :: t_begin = 0d0, t_end
  DOUBLE PRECISION :: tau, t = 0d0, x(3) = 0d0
  DOUBLE PRECISION :: const(3)
  INTEGER :: interval = 10, i = 0
  !const(:) : (Q, G, Omega)
  !xp(:) : (omega, theta, phi)
  const(1) = 2d0; const(3) = 2d0/3d0;
  t_end = 150d0*2d0*pi/const(3)
  tau = 2d0*pi/const(3)*10d0**(-3d0)

  const(2) = 1.47d0
  do while (t < 50d0*2d0*pi/const(3))
    x = runge_kutta(f, 3, x, t, tau, const, boundary)
  end do
  do while (t <= t_end)
    x = runge_kutta(f, 3, x, t, tau, const, boundary)
    i = i + 1
    if (mod(i, interval) == 0) then
      WRITE(*, *) x(3), x(2), x(1)
    end if
  end do

  contains
  function f(tp, xp, n, const)
    implicit none
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION :: f(n)
    DOUBLE PRECISION, INTENT(IN) :: xp(:), tp
    DOUBLE PRECISION, OPTIONAL :: const(:)
    !xp(:) : (omega, theta, phi)
    !const(:) : (Q, G, Omega, pi)
    f(1) = -1.0/const(1)*xp(1) - sin(xp(2)) + const(2)*cos(xp(3))
    f(2) = xp(1)
    f(3) = const(3)
  end function f
  function boundary(xp, n)
    implicit none
```

```
42      INTEGER, INTENT(IN) :: n
43      DOUBLE PRECISION, INTENT(IN) :: xp(:)
44      DOUBLE PRECISION :: boundary(n)
45      !xp(:) : (omega, theta, phi)
46      boundary(1) = xp(1)
47
48      if (xp(2) > pi) then
49        boundary(2) = xp(2) - 2*pi
50      elseif (xp(2) <= -pi) then
51        boundary(2) = xp(2) + 2*pi
52      else
53        boundary(2) = xp(2)
54      end if
55      if (xp(3) >= 2*pi) then
56        boundary(3) = xp(3) - 2*pi
57      elseif (xp(3) < 0) then
58        boundary(3) = xp(3) + 2*pi
59      else
60        boundary(3) = xp(3)
61      end if
62    end function boundary
63  end program main
```