情報数理工学実験第2/コンピュータサイエンス実験第2
テキスト
2次元渦運動の数値シミュレーション

担当：緒方秀教 ogata@im.uec.ac.jp

2015年12月2日（水）（2015年度後学期）

1　序：本実験の目的

2次元渦運動は，モデル方程式が簡単な割には条件によっていろいろ面白い現象が見られ，興味ある物理現象である（図1参照）．

ところで，物理現象のほとんどは，微分方程式を用いた数理モデルで記述され，2次元渦運動もそのひとつである．本実験では，常微分方程式の初期値問題の数値解法，とくに，Runge-Kutta（ルンゲ= クッタ）法について学び，それを用いて2次元渦運動のシミュレーションを行う．すなわち，次の2項目を本実験の目的とする．

- 常微分方程式の数値解法（とくにRunge-Kutta法）を理解し，　実際にプログラムを書いて常微分方程式の数値解を求めることが出来る．
- 前項でまなんだ常微分方程式の数値解法を応用して，2次元渦運動を例に，物理現象のシミュレーションが出来る．

2　本実験のスケジュール

本実験では，おおまかに次の2項目を1ラウンド中に行う．

1. 常微分方程式の数値解法
    - Euler法，Runge-Kutta法
    - 数値解法の誤差
2. 2次元渦運動の数値解析
    - 2次元流体の複素関数論による解析
    - 2次元渦運動の基礎方程式
    - 2次元渦運動の数値シミュレーション

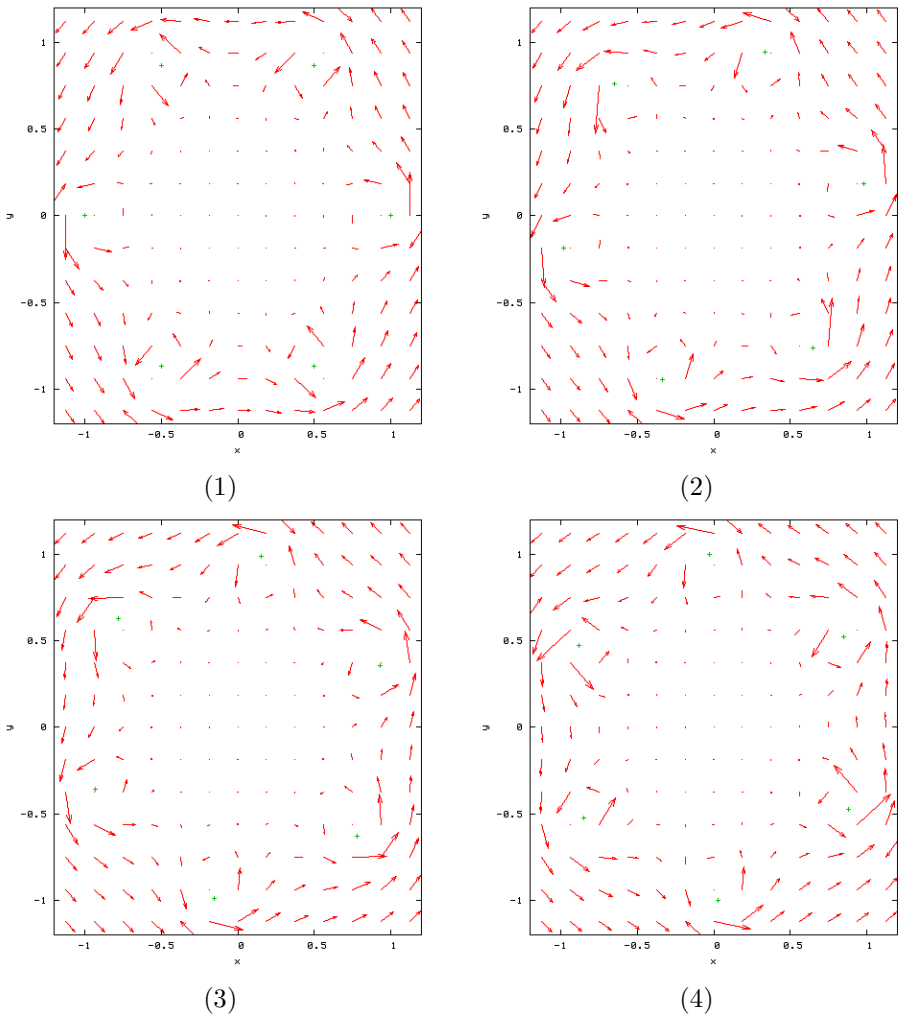


図 1: 6 個の渦の存在する流体場における流速ベクトル場の時間変化．6 個の渦の中心が円に沿って回転しているのが分かる．

3　常微分方程式の数値解法

3.1　常微分方程式の初期値問題

次の m 元連立常微分方程式の初期値問題を考える．

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = f_i(t, u_1(t), u_2(t), \ldots, u_m(t)), \quad i = 1, 2, \ldots, m \tag{1a}$$

$$u_i(t_0) = u_i^{(0)}, \quad i = 1, 2, \ldots, m. \tag{1b}$$

これらはベクトルを用いると，次のように表される．

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}(t, \boldsymbol{u}(t)), \quad \boldsymbol{u}(t_0) = \boldsymbol{u}_0, \tag{2}$$

ここで，

$$\boldsymbol{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_m(t) \end{bmatrix}, \quad \boldsymbol{u}_0 = \begin{bmatrix} u_1^{(0)} \\ u_2^{(0)} \\ \vdots \\ u_m^{(0)} \end{bmatrix}, \quad \boldsymbol{f}(t, \boldsymbol{u}) = \begin{bmatrix} f_1(t, u_1, \ldots, u_m) \\ f_2(t, u_1, \ldots, u_m) \\ \vdots \\ f_m(t, u_1, \ldots, u_m) \end{bmatrix} \tag{3}$$

である．以降はもっぱら，上のベクトル表記を用いることにする．

3.2 Euler 法

我々は時間 $t_0 \leqq t \leqq t_{\mathrm{end}}$ における解 $\boldsymbol{u}(t)$ を近似的に求めたいとする．このとき，常微分方程式の数値解法では，時間 $[t_0, t_{\mathrm{end}}]$ を適当な間隔で

$$t_{\mathrm{init}} = t_0 < t_1 < t_2 < \ldots < t_N = t_{\mathrm{end}}, \quad t_n = t_0 + nh, \quad h = \frac{t_{\mathrm{end}} - t_{\mathrm{init}}}{N}$$

と区切り，時刻 $t = t_n$ における解 $\boldsymbol{u}(t_n)$ の近似値 $\boldsymbol{u}_n$ を用いて，次の時間ステップ $t = t_{n+1}$ における解の近似値 $\boldsymbol{u}_{n+1} \simeq \boldsymbol{u}(t_{n+1})$ を求める．

こうした解法で最も簡単なものは，次の **Euler（オイラー）法**である．

Euler 法

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\boldsymbol{f}(t_n, \boldsymbol{u}_n), \quad n = 0, 1, 2, \ldots, N-1. \tag{4}$$

これは，微分の差分近似 $\dfrac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \simeq \dfrac{\boldsymbol{u}(t+h) - \boldsymbol{u}(t)}{h}$ によって得られる．この公式は計算が簡単があるが，時間ステップを増やしていくうちに誤差が累積していき，厳密解とのずれが大きくなっていく（図 2 参照）．

3.3 Runge-Kutta 法

Euler 法より精度を改善した数値解法としてよく用いられるのは，**Runge-Kutta 法**と呼ばれる一連の解法である [2]．これは，ステップ $t = t_n$ の近似解 $\boldsymbol{u}_n$ から次のステップ $t = t_{n+1}$ の近似解 $\boldsymbol{u}_{n+1}$ を求めるのに，区間 $t_n \leqq t \leqq t_{n+1} = t_n + h$ の途中のいくつかの点の関数値 $\boldsymbol{f}(t, \boldsymbol{v})$ を用いるものである．実際の計算でよく用いられているものは，1 ステップ中の 4 点を用いる次の公式（4 段 4 次 Runge-Kutta 公式）である．

4 段 4 次 Runge-Kutta 法

$$\begin{cases} \boldsymbol{k}_1 = \boldsymbol{f}(t_n, \boldsymbol{u}_n), \\ \boldsymbol{k}_2 = \boldsymbol{f}\left(t_n + \dfrac{h}{2}, \boldsymbol{u}_n + \dfrac{h}{2}\boldsymbol{k}_1\right), \\ \boldsymbol{k}_3 = \boldsymbol{f}\left(t_n + \dfrac{h}{2}, \boldsymbol{u}_n + \dfrac{h}{2}\boldsymbol{k}_2\right), \\ \boldsymbol{k}_4 = \boldsymbol{f}(t_n + h, \boldsymbol{u}_n + h\boldsymbol{k}_3) \end{cases} \tag{5a}$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \frac{h}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4), \quad n = 0, 1, 2, \ldots, N-1. \tag{5b}$$

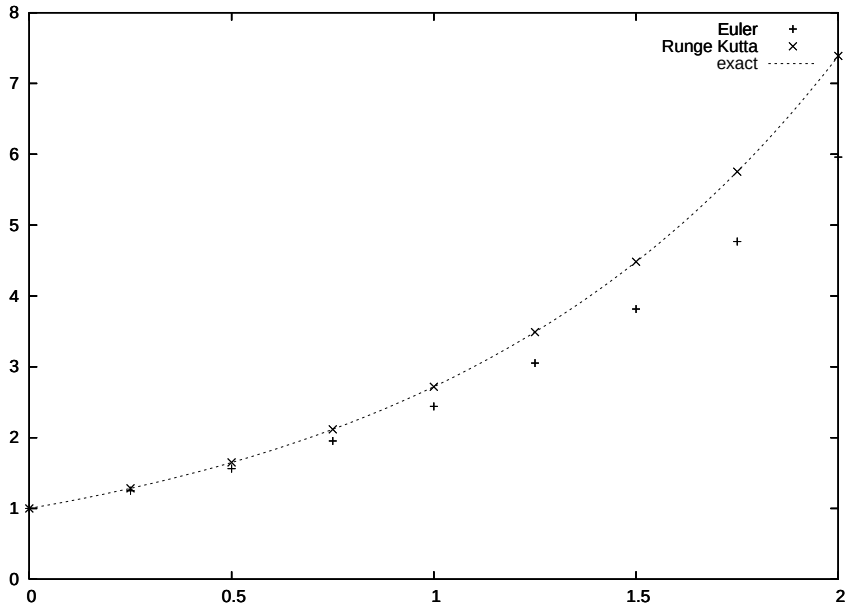


図 2: 常微分方程式の初期値問題 (6) に対する数値解法.

3.4 数値例とプログラム

常微分方程式の初期値問題

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y, \quad y(0) = 1 \tag{6}$$

に Euler 法，Runge-Kutta 法を適用して数値解を求めた結果を図 2 に載せた．ただし，$t_{\mathrm{init}} = 0$, $t_{\mathrm{end}} = 2$, $h = 0.25$ ($N = 8$) としてある．以下に，Euler 法の C プログラム例 (`euler.c`) を載せる．

```
/*
  常微分方程式の初期値問題
  du(t)/dt = f(t, u(t)), t(t_init) = u_init ( t_init<=t<=t_end )
  を Euler 法で解く.
*/
#include<stdio.h>
#include<math.h>
#define NMAX 100
/*
  関数 f(t, u(t))
  the function f(t, u(t))
*/
double func(double t, double u)
{
  return u;
}
/*
   Euler 法の各ステップ
   n 段目の解 "un"=u(t(n)) を受け取り，n+1 段目の解 u(t(n+1)) を返す.
   each step of the Euler method
*/
double euler_step(double tn, double un, double h, double (*func)(double, double))
{
  double un1;
```

```
  /* */
  un1 = un + h * func(tn, un);
  /* */
  return un1;
}
/*
   Euler 法
   t=t(0)(=t_init), t(1), ..., t(n_time)(=t_end)
   ( t(n) = t(0) + n*h, h=(t_end-t_init)/n_time )
   における近似解
   u(0)=u(t(0))(=u_init), u(1)=u(t(1)), ..., t(n_time)=u(t(n_time))
   を計算する.
   時間ステップ t(0), t(1), ..., t(n_time) は配列"t[]"に記憶し,
   近似解 u(0), u(1), ..., u(n_time) は配列"u[]"に記憶する.
*/
void euler(double (*func)(double, double), double t_init, double t_end, double u_init,
   int n_time, double t[NMAX+1], double u[NMAX+1])
{
  double h, un, un1, tn;
  int n;
  /* */
  h = (t_end - t_init) / n_time;
  /* */
  un1 = u_init;
  t[0] = t_init;
  u[0] = un1;
  for (n=0; n<n_time; ++n)
    {
      tn = t_init + n * h;
      un = un1;
      un1 = euler_step(tn, un, h, func);
      t[n+1] = tn + h;
      u[n+1] = un1;
    }
}
/*
  gnuplot 用のデータファイル"euler.dat"に,
  t(n), u(n) (n=0,1,...,n_time)
  を書きこむ.
*/
void record_euler(int n_time, double t[NMAX+1], double u[NMAX+1])
{
  FILE *fp;
  int n;
  /* */
  fp = fopen("euler.dat","w");
  for (n=0; n<=n_time; ++n)
    fprintf(fp,"%10.3e %10.3e\n", t[n], u[n]);
  fprintf(fp,"\n");
}
/*
  main program
*/
main()
{
  double t_init, t_end, u_init;
  double t[NMAX+1], u[NMAX+1];
  int n_time;
```

```
  /* */
  t_init = 0.0;
  t_end = 2.0;
  u_init = 1.0;
  n_time = 8;
  euler(func, t_init, t_end, u_init, n_time, t, u);
  record_euler(n_time, t, u);
  /* */
  return 0;
}
```

課題 1　初期値問題 (6) の厳密解を求めよ．

課題 2　上記の Euler 法プログラムを実行し，問題 (6) の近似解を求めよ．

課題 3　上記のプログラムをもとに 4 段 4 次 Runge-Kutta 法のプログラムを作成し，問題 (6) の近似解を求めよ（図 2 参照）．計算結果は数値を出して終わりにするのでなく，**gnuplot などでグラフを描くなどして視覚的に分かるよう工夫する**こと．以降の課題も同様である．

課題 2 の Euler 法の C プログラム `euler.c` は，Web ページ
`http://www.im.uec.ac.jp/~ogata/jikken2_2015/jikken2_2015.html`
に置いてあるので，課題にはそれを使うこと [1]．そして，課題 3 の Runge-Kutta 法のプログラムは `euler.c` を書き直してつくること．

注意　コンピュータ・プログラムは次のように作成すべきである．

1. これから計算しようとすることを，いくつかの計算過程に分ける．
2. 各計算過程ごとに副プログラム（C でいう “関数”）を書く．
3. これらの副プログラムをメイン・プログラム（main 関数）でまとめて実行する．

これをプログラムの構造化という．上記のプログラムもそのように書いてある．何でもかんでもメイン・プログラムに全部押し込むべきでない．

3.5　数値解法の誤差

最終ステップ $t = t_N = t_{\mathrm{end}}$ における数値解の誤差（大域的誤差）

$$\boldsymbol{e}_N = \boldsymbol{u}_N - \boldsymbol{u}(t_N) \tag{7}$$

の大きさは表 1 のとおりになる．表 1 において，記号 $\mathrm{O}(h^p)$ は h^p のオーダーの大きさを意味する．

常微分方程式の数値解法で大域的誤差の大きさが $\mathrm{O}(h^p)$ のものを，p 次の公式と呼ぶ．よって，Euler 法は 1 次の公式，Runge-Kutta 法 (5a,5b) は 4 次の公式である．

p 次の公式に対する大域的誤差の厳密な評価について，次の不等式が成り立つことが知られている [2]：ある条件の下で不等式

$$\|\boldsymbol{e}_n\| \leqq C\left(h^p + \frac{\varepsilon}{h}\right)(t_n - t_0)\mathrm{e}^{L(t_n - t_0)} \tag{8}$$

[1] コメント文中の日本語は shift-JIS で書いてありますので，文字化けして読めない場合は `nkf` コマンドを使うなりして EUC に変換してください．

表 1: 常微分方程式の数値解法の大域的誤差

数値解法	誤差
Euler 法	$\mathrm{O}(h)$
Runge-Kutta 法 (5a,5b)	$\mathrm{O}(h^4)$

が成立する．ここで，C, L は公式によって定まる正の定数，ε は数値解法の各ステップで生じる丸め誤差の絶対値の最大値である．

不等式 (8) から分かるとおり，きざみ幅 h を小さくしていくと，丸め誤差による項 ε/h が大きくなる．したがって，むやみにきざみ幅 h を小さくすることは危険である（後述の「丸め誤差の影響」参照）．

課題 4　Euler 法，Runge-Kutta 法 (5a,5b) について，きざみ幅 h をいろいろな大きさにとって大域的誤差の大きさを調べ，h の変化にともなう誤差の振る舞いが実際に表 1 のとおりになっていることを確かめよ．

gnuplot による最小自乗近似　常微分方程式の数値解法の誤差の刻み h に対する依存性を調べるには，gnuplot による最小自乗近似を用いるとよい．

たとえば，関係式 $y = ax^2 + b$（a, b は定数）に従う 2 つの量 x, y があるとして，実験等により次の x, y の値が得られたとする．

x	y
0.000	1.005
1.000	1.981
2.000	5.101
3.000	9.899
4.000	17.123

これをつぎのようなテキストファイル（ファイル名を `fitting.dat` とする）に記録する．

```
#
#  fitting.dat
#
#  x       y
#-----------------
 0.000   1.005
 1.000   1.981
 2.000   5.101
 3.000   9.899
 4.000  17.123
#-----------------
```

gnuplot を立ち上げ，次のコマンドを実行する．

```
gnuplot> f(x) = a * x**2 + b
gnuplot> fit f(x) "fitting.dat" via a, b
```

すると，gnuplot は最小自乗近似の計算を行い，最終的に次のような結果を返す．

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 1.00454          +/- 0.007575     (0.7541%)
```

```
b               = 0.994559          +/- 0.06374      (6.409%)

...
```

これより，実験的に得られる定数 a, b の値は $a = 1.00454$, $b = 0.994559$ であることが分かる．さらに，次のコマンドにより，`fitting.dat` のデータと最小自乗近似で得られた $y = ax^2 + b$ のグラフを一緒にプロットすることができる（図 3 参照）.

```
gnuplot> set xlabel "x"
gnuplot> set ylabel "y"
gnuplot> plot "fitting.dat" title "", f(x) w l title ""
```

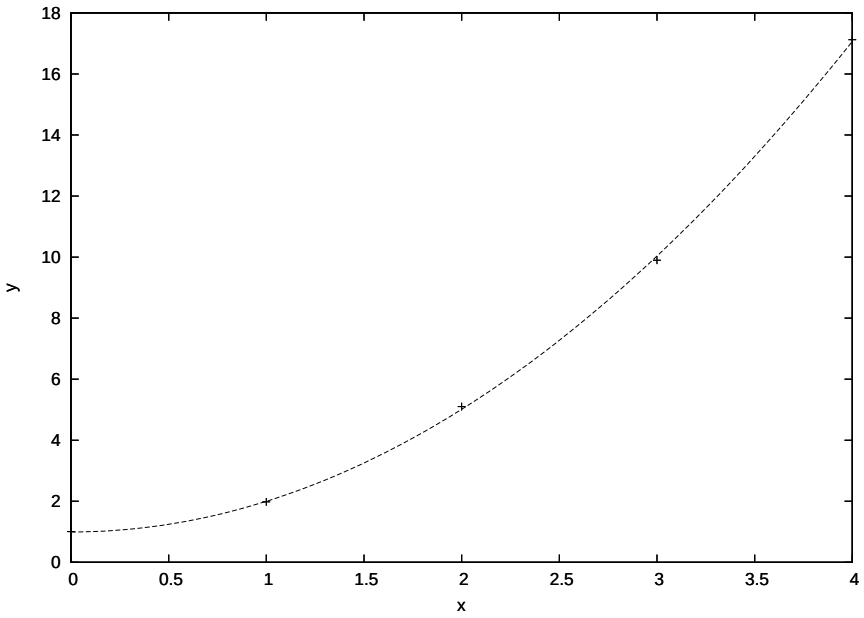


図 3: 最小自乗近似.

課題 4 の場合，誤差評価式 $e_n = Ch^p$ ($e_n = \|\boldsymbol{e}_n\|$) における p の値を実験データから求めたいならば，$\log_{10} h, \log_{10} e_n$ の値を調べれば，関係式

$$\log_{10} e_n = p \log_{10} h + \log_{10} C$$

に最小自乗近似を適用して p の値を求めることができる。

丸め誤差の影響　常微分方程式の数値解法で丸め誤差の影響が現れるのは，大域的誤差の評価式 (8) 右辺において 2 項 h^p と ϵ/h が同程度の大きさになるとき，すなわち，

$$Ch^p \approx \frac{\epsilon}{h}$$

となるときである．そこで，仮に ϵ はマシンイプシロン ϵ_{MAC}（計算機上の浮動小数点演算で $1.0 + \epsilon > 1.0$ とみなされる最小の正数 ϵ）程度の大きさであるとして，丸め誤差の影響が現れる h の値を見積もる．昔の JED の計算機で倍精度計算する場合 [2]，$\epsilon_{\mathrm{MAC}} \sim 1.084 \times 10^{-19} \sim 10^{-19}$ となる（この ϵ_{MAC} の求め方は後で記す）．そして，Runge-Kutta 法の場合 $p = 4$ であるから，

$$h^4 \sim \frac{10^{-19}}{h}, \quad \therefore \quad h \sim (10^{-19})^{1/5} \approx 1.6 \times 10^{-4} \sim 10^{-4}.$$

[2] 2009 年 3 月以前の古い計算機で計算した場合である.

よって，Runge-Kutta 法の場合 $h \sim 10^{-4}$ で丸め誤差の影響が現れると予想される．

一方，著者が数値実験した結果では，h を徐々に小さくしていくと，上の見積もりより 10 進 1 桁大きい $h \sim 10^{-3}$ あたりで誤差の減衰が止まり，丸め誤差の影響が現れた．そして，そこでの大域的誤差の大きさは $10^{-12} = (10^{-3})^4$ 程度であった．

マシンイプシロンは文献 [3] に載っているプログラムを C に書き直したつぎのプログラムで計算できる（**eps** がマシンイプシロンである）．

```
#include <stdio.h>
main()
{
  double eps = 1.0;
  while (1.0+eps!=1.0)
    {
      eps *= 0.5;
    }
  eps *= 2.0;
  printf("machine epsilon: %10.3e\n", eps);
  //
  return 0;
}
```

4　2 次元渦の運動の数値シミュレーション

4.1　2 次元ポテンシャル流の基礎事項

4.1.1　複素流速ポテンシャル

ここでは，2 次元ポテンシャル流，すなわち，2 次元の非圧縮・非粘性流体の定常（時間非依存）渦無し流れについての基礎事項を簡潔に説明する．詳細は，流体力学の教科書，たとえば [1] を参照すること．

2 次元直交座標で表された点 (x, y) における流速を $\boldsymbol{v}(x, y) = (u(x, y), v(x, y))$ で表す．このとき，渦無しの仮定から

$$\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0 \tag{9}$$

が成り立ち，定常流・非圧縮性の仮定から連続方程式は

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{10}$$

と表される．式 (9) からある関数 $\Phi(x, y)$ が存在して流速は

$$u = \frac{\partial \Phi}{\partial x}, \quad v = \frac{\partial \Phi}{\partial y} \tag{11}$$

と表される．一方，式 (10) からある関数 $\Psi(x, y)$ が存在して流速は

$$u = \frac{\partial \Psi}{\partial y}, \quad v = -\frac{\partial \Phi}{\partial x} \tag{12}$$

と表される．Φ を速度ポテンシャル，Ψ を流れの関数とよぶ．

さて，式 (9), (10) から Φ, Ψ は

$$\frac{\partial \Phi}{\partial x} = \frac{\partial \Psi}{\partial y}, \quad \frac{\partial \Phi}{\partial y} = -\frac{\partial \Psi}{\partial x} \tag{13}$$

を満たすことが分かる．これは，(Φ, Ψ) が複素関数論における Cauchy-Riemann の関係式を満たすことに他ならない．したがって，複素関数 $f(z)$ ($z = x + \mathrm{i}y$) を

$$f(z) = \Phi(x, y) + \mathrm{i}\Psi(x, y) \tag{14}$$

で定義すると，$f(z)$ は z の正則関数である．$f(z)$ を**複素速度ポテンシャル**とよぶ．なお，式 (9), (10) および公式 $\dfrac{\partial}{\partial z} = \dfrac{1}{2}\left(\dfrac{\partial}{\partial x} - \mathrm{i}\dfrac{\partial}{\partial y}\right)$ を使えば，流速は

$$u - \mathrm{i}v = f'(z) \tag{15}$$

で与えられることが分かる．

流れの場に閉曲線 C をとり（図 4(a) 参照），z を C に沿って一周動かしたときの $f(z)$ の変化を調べる．式 (15) を用いると，

$$\begin{aligned}
[f(z)]_C &= \oint_C f'(z)\,\mathrm{d}z = \oint_C (u - \mathrm{i}v)(\mathrm{d}x + \mathrm{i}\mathrm{d}y) \\
&= \oint_C (u\mathrm{d}x + v\mathrm{d}y) + \mathrm{i}\oint_C (u\mathrm{d}y - v\mathrm{d}x) \\
&= \oint_C v_t \mathrm{d}s + \mathrm{i}\oint_C v_n \mathrm{d}s
\end{aligned}$$

を得る．ここで，$\mathrm{d}s$ は閉曲線 C 上の微小線素の長さ，v_t は流速 $\boldsymbol{v}$ の接線方向成分，v_n は外向き法線方向成分である（図 4(a) 参照）．

$$\Gamma(C) = \oint_C v_t \mathrm{d}s = \mathrm{Re}\,[f(z)]_C\,, \quad Q(C) = \oint_C v_n \mathrm{d}s = \mathrm{Im}\,[f(z)]_C \tag{16}$$

と置き，$\Gamma(C)$ を閉曲線 C についての**循環**，$Q(C)$ を閉曲線 C からの**湧き出し**（$Q(C) < 0$ の場合は吸い込み）の強さとよぶ．$Q(C)$ は閉曲線 C を通って内側から外側へ流れ出す単位時間当たりの流量を表す．$f(z)$ が C およびその内部を含む領域で正則ならば，$[f(z)]_C = 0$ なので，循環 $\Gamma(C)$ および湧き出しの強さ $Q(C)$ はゼロである．

流れ関数 Ψ の意味　C を流れの場の中の（必ずしも閉じていない）閉曲線とする（図 4(b) 参照）．このとき，上記と同様にして

$$[\Psi]_C = \Psi(\mathrm{B}) - \Psi(\mathrm{A}) = \int_C v_n \mathrm{d}s \tag{17}$$

が導かれる．したがって，$[\Psi]_C$ は曲線 C の左側から右側へ通り抜ける単位時間あたりの流量を意味する．とくに，$[\Psi]_C = 0$ の場合，流体は C を通り抜けないので，C は流れの流線に一致する．すなわち，曲線 $\mathrm{Im}\, f = \Psi = \mathrm{const.}$ は流れの流線である．

4.1.2　簡単な例

一様な流れ　x 軸（実軸）に対して角度 α をなす一様な流れ（図 5(a) 参照）は，複素ポテンシャル

$$f(z) = U\mathrm{e}^{-\mathrm{i}\alpha} z$$

（$U(>0)$ は流速の大きさを表す定数）で与えられる．実際，流速 $\boldsymbol{v} = (u, v)$ は $u - \mathrm{i}v = f'(z) = U\mathrm{e}^{-\mathrm{i}\alpha}$ より $u = U\cos\alpha$，$v = U\sin\alpha$ と与えられる．

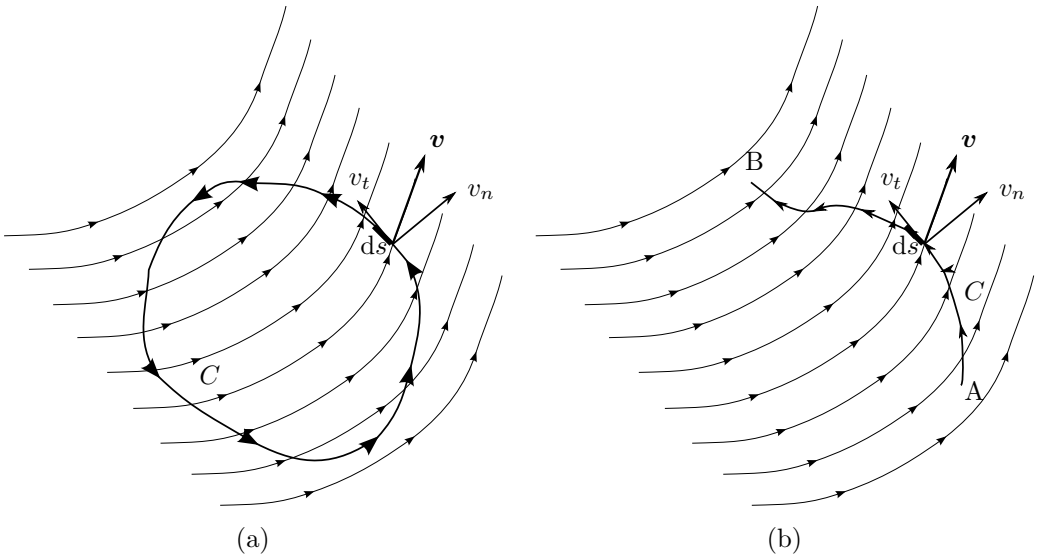


図 4: 流れの場における閉曲線 C.

湧き出し　複素速度ポテンシャル

$$f(z) = \frac{Q}{2\pi} \log z$$

(Q は実定数) は原点 $z = 0$ から放射状に湧き出す流れを表す（図 5(b) 参照）. 実際, 流速 $\boldsymbol{v} = (u, v)$ は

$$u - \mathrm{i}v = f'(z) = \frac{Q}{2\pi} \frac{1}{z} = \frac{Q}{2\pi} \frac{x - \mathrm{i}y}{x^2 + y^2}$$

より,

$$u = \frac{Q}{2\pi} \frac{x}{x^2 + y^2}, \quad v = \frac{Q}{2\pi} \frac{y}{x^2 + y^2}$$

で与えられる. そして, C を原点の周りを 1 周する閉曲線とすると

$$\Gamma(C) + \mathrm{i}Q(C) = [f(z)]_C = \frac{Q}{2\pi} [\log z]_C = \frac{Q}{2\pi} \cdot 2\pi\mathrm{i} = \mathrm{i}Q$$

より,

$$\Gamma(C) = 0, \quad Q(C) = Q$$

を得る. したがって, 原点 $z = 0$ からの単位時間あたりの湧き出し量は Q である.

定数 Q を, **湧き出しの強さ**とよぶ.

渦糸　複素速度ポテンシャル

$$f(z) = \frac{\Gamma}{2\pi\mathrm{i}} \log z \tag{18}$$

(Γ は実定数) は原点 $z = 0$ を中心とする 2 次元渦 (渦糸) を表す（図 5(c) 参照）. 実際, 流速 $\boldsymbol{v} = (u, v)$ は

$$u - \mathrm{i}v = f'(z) = \frac{\Gamma}{2\pi\mathrm{i}} \frac{1}{z} = \frac{\Gamma}{2} \frac{-y - \mathrm{i}x}{x^2 + y^2}$$

より,

$$u = -\frac{\Gamma}{2\pi} \frac{y}{x^2 + y^2}, \quad v = \frac{\Gamma}{2\pi} \frac{x}{x^2 + y^2}$$

で与えられる．そして，C を原点の周りを 1 周する閉曲線とすると，

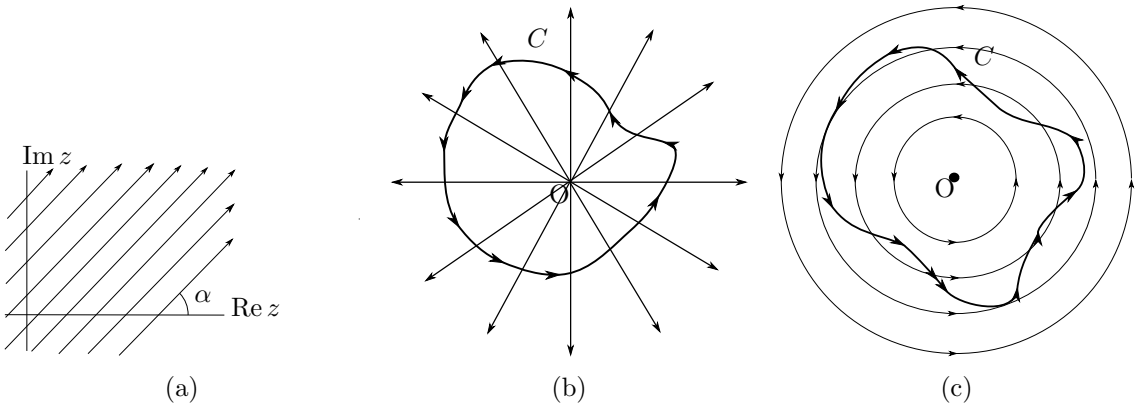
$$\Gamma(C) + \mathrm{i}Q(C) = [f(z)]_C = \frac{\Gamma}{2\pi\mathrm{i}} [\log z]_C = \frac{\Gamma}{2\pi\mathrm{i}} \cdot 2\pi\mathrm{i} = \Gamma$$

より

$$\Gamma(C) = \Gamma, \quad Q(C) = 0$$

である．したがって，原点まわりの循環は Γ である．

定数 Γ を**渦糸の循環**とよぶ．



(a) (b) (c)

図 5: (a) 一様な流れ，(b) 湧き出し，(c) 渦糸．

4.2 2 次元渦の運動方程式

平面上を相互作用しあいながら運動する N 個の渦を考える．各渦は中心 $z_j = x_j + \mathrm{i}y_j$, 循環 Γ_j ($j = 1, 2, \ldots, N$) を持つとする，すなわち，各渦は次の複素速度ポテンシャルを持つとする．

$$f_j(z) = \frac{\Gamma_j}{2\pi\mathrm{i}} \log(z - z_j) \quad (\, j = 1, 2, \ldots, N\,). \tag{19}$$

ここで，j 番目の渦の中心 z_j の運動を考える．渦の中心 z_j は，自分自身以外の渦のつくる速度場，すなわち，複素速度ポテンシャル $\sum_{\substack{i=1 \\ (i\neq j)}}^{N} f_i(z)$ をもつ速度場に乗って運動する．したがって，渦の中心 z_j の速度 $\mathrm{d}z_j/\mathrm{d}t$ は

$$\overline{\frac{\mathrm{d}z_j}{\mathrm{d}t}} = \left[\frac{\mathrm{d}}{\mathrm{d}z} \sum_{\substack{i=1 \\ i\neq j}}^{N} f_i(z) \right]_{z=z_j} = \sum_{\substack{i=1 \\ i\neq j}}^{N} \frac{\Gamma}{2\pi\mathrm{i}} \frac{1}{z_j - z_i},$$

すなわち，

$$\frac{\mathrm{d}\overline{z_j}}{\mathrm{d}t} = \sum_{\substack{i=1 \\ (i\neq j)}}^{N} \frac{\Gamma_i}{2\pi\mathrm{i}(z_j - z_i)} \quad (\, j = 1, 2, \ldots, N\,) \tag{20}$$

で与えられる．これが，N 個の渦の中心 $z_1, z_2, \ldots, z_N$ に対する運動方程式である．

課題 5 運動方程式 (20) を，z_j の実部 x_j，虚部 y_j ($j = 1, 2, \ldots, N$) に対する微分方程式に書き直せ．

渦運動の性質　渦群の運動について次の性質が知られている：

V1. 渦が 2 個の場合 ($N = 2$)，これらは互いの距離を変えずに運動する．

V2. N 個の渦（中心 $z_1, \ldots, z_N$，循環 $\Gamma_1, \ldots, \Gamma_N$）があるとする．

(a) $\sum_{j=1}^{N} \Gamma_j \neq 0$ の場合，それらの重心を

$$z_0 = \frac{\sum_{j=1}^{N} \Gamma_j z_j}{\sum_{j=1}^{N} \Gamma_j}$$

で定義する．このとき，渦群の重心 z_0 は動かない．

(b) $\sum_{j=1}^{N} \Gamma_j = 0$ の場合，渦群を任意の 2 群に分けると，それらの重心の相対的位置は不変である．すなわち，渦群を 2 群

$$\begin{cases} \text{中心 } z'_j, \text{ 循環 } \Gamma'_j & (j = 1, 2, \ldots, N') \\ \text{中心 } z''_j, \text{ 循環 } \Gamma''_j & (j = 1, 2, \ldots, N'') \end{cases}$$

に分け，各群の重心を $z'_0 = \sum_j \Gamma'_j z'_j / \sum_j \Gamma'_j$，　$z''_0 = \sum_j \Gamma''_j z''_j / \sum_j \Gamma''_j$ とするとき，$z'_0 - z''_0$ は一定である．

V3. 原点周りの “慣性モーメント” $\sum_j \Gamma_j ({x_j}^2 + {y_j}^2)$ は不変である．

V4. 原点周りの角運動量

$$\sum_j \Gamma_j \left(x_j \frac{\mathrm{d}y_j}{\mathrm{d}t} - y_j \frac{\mathrm{d}x_j}{\mathrm{d}t} \right)$$

も不変である．

課題 6　2 個の渦対について，次の条件のもとで運動を数値的に（Runge-Kutta 法により）求め，その結果について考察せよ．

1. 第 1 渦対：循環 $\Gamma(> 0)$，初期位置 $z_1(0) = a(> 0)$.
   第 2 渦対：循環 Γ，初期位置 $z_2(0) = -a$.
2. 第 1 渦対：循環 $\Gamma(> 0)$，初期位置 $z_1(0) = \mathrm{i}a$ $(a > 0)$.
   第 2 渦対：循環 $-\Gamma$，初期位置 $z_2(0) = -\mathrm{i}a$.

参考までに，1 番目の課題を Euler 法で解いた C プログラム `euler1.c` を，下記に載せる．このプログラムは Web ページ

`http://www.im.uec.ac.jp/~ogata/jikken2_2015/jikken2_2015.html`

に置いておくので，これを Runge-Kutta 法のプログラムに書き直して使うこと．なお，渦の運動方程式の未知関数 x_1, y_1, x_2, y_2 と連立常微分方程式 (1a) の未知関数 $u_1, u_2, \ldots$ とは，次のように対応付けている．

$$u_1 = x_1,\ u_2 = y_1,\ u_3 = x_2,\ u_4 = y_2.$$

```
//
// euler1.c
//
#include <stdio.h>
#include <math.h>
#include <string.h>
#define NDIM 20
#define NVTX 10
#define NTIME 500
double Gamma[NVTX+1];
//
```

```
// Euler 法の各ステップ
// 時間ステップ t=t0 における解 u0 を与えて,
// 次の時間ステップ t=t1=t0+h における解 u1 を得る.
// func, ndim については, 関数 euler のコメントを参照すること.
//
void euler_step(void (*func)(int, double, double [], double []), int ndim,
                double  t0, double u0[NDIM+1], double h,
                double* t1, double u1[NDIM+1])
{
  double k1[NDIM+1];
  int i;
  //--------------------------------
  func(ndim, t0, u0, k1);
  *t1 = t0 + h;
  for (i=1; i<=ndim; ++i) u1[i] = u0[i] + h * k1[i];
}
//
// Euler 法.
// (input)
// func: 常微分方程式 du/dt = f(t,u) の右辺を与える関数
// t_init, t_end: 計算時間 t_init <= t <= t_end
// ntime: 時間ステップ数
// ndim: ベクトル値関数 u(t) の次元
// u_init: 常微分方程式の初期値
// (output)
// t[k] (k=0,1,...,ntime): 時間ステップ（等間隔）
// u[k][i] (i=1,...,ndim): 時間ステップ t=t[k] における解 u
//
void euler(void (*func)(int, double, double [], double []),
           int ntime, double t_init, double t_end,
           int ndim, double u_init[NDIM+1],
           double t[NTIME+1], double u[NTIME+1][NDIM+1])
{
  double t0, t1, h;
  int k;
  double u0[NDIM+1], u1[NDIM+1]; int i;
  int n_vortex = ndim/2;
  //
  h = (t_end - t_init) / ntime;
  t1 = t_init;
  t[0] = t_init;
  for (i=1; i<=ndim; ++i) u[0][i] = u1[i] = u_init[i];
  for (k=1; k<=ntime; ++k)
    {
      t0 = t1;
      for (i=1; i<=ndim; ++i) u0[i] = u1[i];
      euler_step(func, ndim, t0, u0, h, &t1, u1);
      t[i] = t1;
      for (i=1; i<=ndim; ++i) u[k][i] = u1[i];
    }
}
//
// 常微分方程式 du/dt = f(t, u) の右辺を与える.
//
void func(int ndim, double t, double u[NDIM+1], double f[NDIM+1])
{
  double dx, dy, dr2;
  double fx, fy;
```

```
  int n_vortex, j, k;
  //
  n_vortex = ndim / 2;
  //
  for (k=1; k<=n_vortex; ++k)
    {
      fx = fy = 0.0;
      for (j=1; j<=n_vortex; ++j)
        {
          if (j!=k)
            {
              dx = u[2*k-1] - u[2*j-1];
              dy = u[2*k  ] - u[2*j  ];
              dr2 = dx*dx + dy*dy;
              fx += - Gamma[j] * dy / dr2;
              fy +=   Gamma[j] * dx / dr2;
            }
        }
      f[2*k-1] = fx;
      f[2*k  ] = fy;
    }
}
//
// 渦数 n_vortex, 渦の初期位置 (x_init[j],y_init[j]) (j=1,...,n_vortex),
// 計算時間 [t_init, t_end] および時間ステップ数 ntime を与えて,
// 等間隔にとった時間ステップ t=t[k](k=0,1,...,ntime) における
// 渦の中心の位置 (x[k][j],y[k][j]) (j=1,...,n_vortex) を得る.
//
void get_motion(int ntime, double t_init, double t_end,
                int n_vortex, double x_init[NVTX+1], double y_init[NVTX+1],
                double t[NTIME+1],
                double x[NTIME+1][NVTX+1], double y[NTIME+1][NVTX+1])
{
  int ndim = 2 * n_vortex;
  double u_init[NDIM+1], u[NTIME+1][NDIM+1];
  int i, k;
  //
  for (i=1; i<=n_vortex; ++i){
    u_init[2*i-1] = x_init[i];
    u_init[2*i  ] = y_init[i];
  }
  euler(func, ntime, t_init, t_end, ndim, u_init, t, u);
  for (k=0; k<=ntime; ++k)
    for (i=1; i<=n_vortex; ++i) {
      x[k][i] = u[k][2*i-1];
      y[k][i] = u[k][2*i  ];
    }
}
//
// 時間ステップ t=t(k) における渦の中心の座標
// (x[k][j], y[k][j]) (j=1, 2, ..., n_vortex)
// を与えて, 点 (xx, yy) における流速 (vx, vy) を得る.
//
void get_velocity(double xx, double yy, int n_vortex, int k,
                  double x[NTIME+1][NVTX+1], double y[NTIME+1][NVTX+1],
                  double* vx, double* vy)
{
  double vvx, vvy;
```

```
  double dx, dy, bunbo;
  double cc = 0.5 / M_PI;
  double r_vortex = 0.1;
  int j;
  //
  vvx = vvy = 0.0;
  for (j=1; j<=n_vortex; ++j)
    {
      dx = xx - x[k][j];
      dy = yy - y[k][j];
      if (hypot(dx,dy)<=r_vortex)
        {
          vvx = vvy = 0.0;
          break;
        }
      else
        {
          bunbo = dx * dx + dy * dy;
          vvx += - cc * Gamma[j] * dy / bunbo;
          vvy +=   cc * Gamma[j] * dx / bunbo;
        }
    }
  *vx = vvx;
  *vy = vvy;
}
//
// 時間ステップ t=t(k) (k=0,1,...,n_time) における渦の中心の座標
// (x[k][j],y[k][j]) (j=1,2,...,n_vortex) を与えて,
// 各時間ステップ t=t(k) (k=0,k_itvl,2*k_itvl,...) における
// 流速ベクトル場をファイル"velocity00", "velocity01", ... に,
// 渦の中心の座標をファイル"vortex00", "vortex01", ... に書き込む.
//
void record_velocity(int ntime, int n_vortex,
                     double x[NTIME+1][NVTX+1], double y[NTIME+1][NVTX+1])
{
  FILE *fp;
  char filename[20] = "velocity";
  char filename_tmp[20], filenum[3];
  char filename1[20] = "vortex";
  int nfile = -1;
  int i, k, k_itvl = 12;
  //
  double xx, yy, xmin, xmax, ymin, ymax;
  int nx, ny, ix, iy, j;
  double dx, dy;
  double vx, vy, c_scale = 0.15;
  //
  // 流速ベクトル場を書き込む領域を
  // 長方形領域 [xmin,xmax]x[ymin,ymax] とする.
  xmin = ymin = - 1.5;
  xmax = ymax =   1.5;
  // 長方形領域 [xmin,xmax]x[ymin,ymax] 中の
  // 格子点 (xmin+ix*dx, ymin+iy*dy)
  // (ix=0,1,...,nx, iy=0,1,...,ny)
  // に流速ベクトルを書き込む.
  nx = ny = 16;
  dx = (xmax - xmin) / nx;
  dy = (ymax - ymin) / ny;
```

```
  //
  for (k=0; k<=ntime; ++k)
    if (k%k_itvl==0){
      // ファイル"velocity**"を開く.
      ++nfile;
      filenum[0] = (char)nfile/10 + '0';
      filenum[1] = (char)nfile%10 + '0';
      filenum[2] = '\0';
      strcpy(filename_tmp, filename);
      strcat(filename_tmp, filenum);
      fp = fopen(filename_tmp,"w");
      for (ix=0; ix<=nx; ++ix)
        {
          xx = xmin + ix * dx;
          for (iy=0; iy<=ny; ++iy)
            {
              yy = ymin + iy * dy;
              get_velocity(xx, yy, n_vortex, k, x, y, &vx, &vy);
              vx *= c_scale;
              vy *= c_scale;
              fprintf(fp,"%10.3e %10.3e %10.3e %10.3e\n", xx, yy, vx, vy);
            }
        }
      fclose(fp);
      // ファイル"vortex**"を開く.
      strcpy(filename_tmp, filename1);
      strcat(filename_tmp, filenum);
      fp = fopen(filename_tmp,"w");
      for (j=1; j<=n_vortex; ++j)
        fprintf(fp,"%10.3e %10.3e\n", x[k][j], y[k][j]);
      fclose(fp);
    }
}
//
// main program
//
main()
{
  double x_init[NVTX+1], y_init[NVTX+1], x[NTIME+1][NVTX+1], y[NTIME+1][NVTX+1];
  double t_init = 0.0, t_end = 10.0, t[NTIME+1];
  int n_vortex=2, ntime=256;
  double theta, d_theta;
  int j;
  //
  Gamma[1]  =   1.0;
  Gamma[2]  =   1.0;
  x_init[1] =   1.0;
  y_init[1] =   0.0;
  x_init[2] = - 1.0;
  y_init[2] =   0.0;
  //
  get_motion(ntime, t_init, t_end, n_vortex, x_init, y_init, t, x, y);
  record_velocity(ntime, n_vortex, x, y);
  //
  return 0;
}
```

ここで，同じ循環（$\Gamma > 0$ とする）の N 個の渦を正 N 角形状に配列した場合を考える．すなわち，次の初期条件での渦運動を考える．

$$z_k(0) = R\exp\left(\mathrm{i}\frac{2\pi(k-1)}{N}\right), \quad k = 1, 2, \ldots, N \quad (\, R > 0\,).$$

この場合，原点の周りを一定の角速度 ω で運動する解が知られている．

$$z_k(t) = R\exp\left(\mathrm{i}\left(\frac{2\pi(k-1)}{N} + \omega t\right)\right), \quad k = 1, 2, \ldots, N.$$

課題 7　上記の N 個の渦群の運動を，さまざまな N の値に対して数値的に（Runge-Kutta 法により）求めよ．そして，計算結果の一つを GIF アニメーションにせよ．

アニメーションの作成　物理の運動方程式の計算結果は，アニメーションにすることにより感覚的に理解することができる．ここでは，gnuplot を用いてアニメーションを作成する方法を述べる．

先述の `euler1.c` をコンパイル･実行すると，時間ステップ順の流速ベクトル場のデータファイル `velocity00`, `velocity01`, ...，渦の中心のデータファイル `vortex00`, `vortex01`, ... ができあがる．そこで，テキストエディタで次のファイル `vortex.gpl` をつくっておく．

```
#
# vortex.gpl
#
set size square
set xrange [-1.2:1.2]
set yrange [-1.2:1.2]
set xlabel "x"
set ylabel "y"
plot "velocity00" with vector title "", "vortex00" title ""
pause 1
plot "velocity01" with vector title "", "vortex01" title ""
pause 1
（途中略）
plot "velocity21" with vector title "", "vortex21" title ""
pause 1
```

gnuplot を起動し，コマンド

```
gnuplot> load "vortex.gpl"
```

を実行すると，流速ベクトル場および渦の中心の時間変化がコマ送りで表示される．

さらにアニメーション GIF を作成するには，次のようにすればよい．テキストエディタで次のファイル `vortex_png.gpl` をつくっておく．

```
#
# vortex_png.gpl
#
set size square
set xrange [-1.2:1.2]
set yrange [-1.2:1.2]
set xlabel "x"
set ylabel "y"
set terminal png
set output "velocity00.png"
plot "velocity00" with vector title "", "vortex00" title ""
set output "velocity01.png"
plot "velocity01" with vector title "", "vortex01" title ""
（途中略）
set output "velocity21.png"
plot "velocity21" with vector title "", "vortex21" title ""
set terminal x11
```

gnuplot を起動し，次のコマンドを実行する．

```
gnuplot> load "vortex_png.gpl"
```

すると，PNG 画像ファイル velocity00.png, velocity01.png が生成される．これは，各時間ステップにおける流速場・渦の中心を描いた画像である．gnuplot を終了し，次のコマンドを実行する．

```
$ convert -loop 3 -delay 30 velocity*.png vortex.gif
```

これにより，PNG ファイル velocity00.png, velocity01.png, . . . がひとつのアニメーション GIF vortex.gif にまとめられる．オプション-loop の後の数字は，アニメーションを繰り返す回数である．この数字を 0 とすれば，アニメーションが無限に繰り返される．

参考文献

[1] 今井功：流体力学，岩波書店，1993 年.

[2] 森正武：数値解析　第 2 版，共立出版，2002 年.

[3] 森正武，FORTRAN77 数値計算プログラミング（増補版），岩波書店，1987 年.