常微分方程式の数値計算を行うにあたって

――― 数値の表現、誤差、近似、マス・バネ・ダンパ系の例 ―――

1. 数値の表現

1.1 整数型の数の表現

・４ビット整数の表現（２進数、８進数、１６進数）

10 進数	2 進数	16 進数	8 進数
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	a	12
11	1011	b	13
12	1100	c	14
13	1101	d	15
14	1110	e	16
15	1111	f	17

・１バイト（８ビット）整数の表現

符号なし整数			符号付き整数
10 進数	2 進数	16 進数	10 進数
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
3	00000011	03	3
4	00000100	04	4
:	:	:	:
:	:	:	:
126	01111110	7e	126
127	01111111	7f	127
128	10000000	80	-128
129	10000001	81	-127
130	10000010	82	-126
131	10000011	83	-125
132	10000100	84	-124
:	:	:	:
:	:	:	:
254	11111110	fe	-2
255	11111111	ff	-1

負の数(-n)は正の数(n)の 2 の補数
$2^8 - n = (2^8 - 1) - n + 1$
（ビットを反転させ 1 を加えた数）により表現され、その先頭ビットには 1 がたつ。
「符号付き 2 バイト整数の表現」参照のこと

・符号付き２バイト整数（short int 型）の表現

10 進数	2 進数	16 進数（2 の補数）		10 進数	2 進数	16 進数
0	00000000 00000000	0000				
1	00000000 00000001	0001	<--->	-1	11111111 11111111	ffff
2	00000000 00000010	0002	<--->	-2	11111111 11111110	fffe
3	00000000 00000011	0003	<--->	-3	11111111 11111101	fffd
4	00000000 00000100	0004	<--->	-4	11111111 11111100	fffc
5	00000000 00000101	0005	<--->	-5	11111111 11111011	fffb
6	00000000 00000110	0006	<--->	-6	11111111 11111010	fffa
7	00000000 00000111	0007	<--->	-7	11111111 11111001	fff9
8	00000000 00001000	0008	<--->	-8	11111111 11111000	fff8
9	00000000 00001001	0009	<--->	-9	11111111 11110111	fff7
10	00000000 00001010	000a	<--->	-10	11111111 11110110	fff6
11	00000000 00001011	000b	<--->	-11	11111111 11110101	fff5
12	00000000 00001100	000c	<--->	-12	11111111 11110100	fff4
13	00000000 00001101	000d	<--->	-13	11111111 11110011	fff3
14	00000000 00001110	000e	<--->	-14	11111111 11110010	fff2
15	00000000 00001111	000f	<--->	-15	11111111 11110001	fff1
16	00000000 00010000	0010	<--->	-16	11111111 11110000	fff0
17	00000000 00010001	0011	<--->	-17	11111111 11101111	ffef
:	:	:		:	:	:
:	:	:		:	:	:
32764	01111111 11111100	7ffc	<--->	-32764	10000000 00000100	8004
32765	01111111 11111101	7ffd	<--->	-32765	10000000 00000011	8003
32766	01111111 11111110	7ffe	<--->	-32766	10000000 00000010	8002
32767	01111111 11111111	7fff	<--->	-32767	10000000 00000001	8001
				-32768	10000000 00000000	8000

2 の補数による負の数の表現：

2 バイトで表現できる整数は 2^{16} (= 65536) 通りであるが、負の数もいれるため正負それぞれ 65536/2 (= 2^{15} = 32768) 通りの数が表現できる。0 を入れて正の側では 0 ~ 32767 まで、負の側では -1 ~ -32768 まで表す。負の数(-n)は、正の数(n)の 2 の補数（ $2^{16}-n = (2^{16}-1)-n+1$ ：ビットを反転させ 1 を加えた数）により表され、その先頭ビットは 1 である。

1.2 実数型の数の表現（浮動小数点数表示）

計算機における“実数”とは、数学的な意味での実数ではなく、有限桁から成る浮動小数点数により表される不連続（離散的）で有限な数である。例えば-1234.56 なる 10 進数は、有効桁と 10 のべきとに分けて、

$$-1.23456\times10^{3}$$

と書くことができるように、一般に計算機内部における N 進 t 桁の浮動小数点数は次の形に表される。

$$\pm d_0.d_1d_2\cdots d_{t-1}\times N^q = \pm(d_0+\frac{d_1}{N}+\frac{d_2}{N^2}+\cdots+\frac{d_{t-1}}{N^{t-1}})N^q$$

とくに $d_0d_1d_2\cdots d_{t-1}$ を仮数部、N^q を指数部と呼ぶ。通常の計算機では N は 2, 8, 16 のいずれかであ

る。桁数 t および指数 q の範囲は個々の計算機により異なる。浮動小数点システムでは表される数が 0 でない限り、最初の数字 d_0 は 0 以外の数である（正規表現）。

パーソナルコンピュータやワークステーションで通常用いられる IEEE システムでは N=2 であり、
IEEE SP (Single Precision; 単長実数型)では、
符号に 1 ビット、仮数部に t=23 ビット、指数部に 8 ビット(符号込み)の計 32 ビット（4 バイト）
IEEE DP (Double Precision; 倍長実数型)では、
符号に 1 ビット、仮数部に t=52 ビット、指数部に 11 ビット(符号込み)の計 64 ビット（8 バイト）
が用いられる。なお 0 は仮数=0、指数 q=0 で表される。
ここで IEEE SP で表現できる実数の範囲を概算してみよう。
10 進数で表したときの有効桁数を x 桁とすると、

$$2^{23} = 10^{x} \qquad \text{より、} \quad x = 23 \times \log 2 = 6.923\cdots \qquad (\log 2 \approx 0.3010)$$

つまり有効桁数約 7 桁となる。

次に指数部の範囲は、q が符号付き 8 ビット整数、つまり $2^{-7} \sim 2^{7} - 1$ の範囲の数をとり得ることから、

$$2^{\pm 128} = 10^{y} \qquad \text{より、} \quad y = \pm 128 \times \log 2 = \pm 36.368\cdots$$

つまり概算で $10^{-36} \sim 10^{+36}$ の範囲の指数部をとり得ることになる。

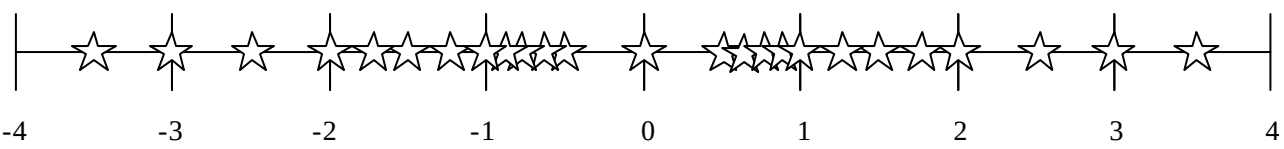
ある計算機で表すことのできる浮動小数点数の絶対値の最大値を D_{max}、最小値を D_{min} とする。IEEE SP では、$D_{max} \sim 10^{+36}$、$D_{min} \sim 10^{-36}$ である。D_{max} よりも大きい数が生ずるとオーバーフローとなり、計算は通常はその時点で中断されてしまう。D_{min} よりも絶対値の小さい数（0 は除く）が生ずるとアンダーフローとなるが、この場合には通常はその数を 0 で置き換えて計算を続行する。
以上のように、浮動小数点数は、離散的で有限な数である。

例） 簡単化のため、仮数部に t=3 ビットを用い、指数に-1, 0, 1 をとりうる 2 進 3 桁の浮動小数点システムを考えてみよう。とり得る浮動小数点数は、

0

$\pm(1.00)_2 \times 2^{0} = \pm(1.0)_{10}$	$\pm(1.00)_2 \times 2^{-1} = \pm(0.5)_{10}$	$\pm(1.00)_2 \times 2^{1} = \pm(2.0)_{10}$
$\pm(1.01)_2 \times 2^{0} = \pm(1.25)_{10}$	$\pm(1.01)_2 \times 2^{-1} = \pm(0.625)_{10}$	$\pm(1.01)_2 \times 2^{1} = \pm(2.5)_{10}$
$\pm(1.10)_2 \times 2^{0} = \pm(1.5)_{10}$	$\pm(1.10)_2 \times 2^{-1} = \pm(0.75)_{10}$	$\pm(1.10)_2 \times 2^{1} = \pm(3.0)_{10}$
$\pm(1.11)_2 \times 2^{0} = \pm(1.75)_{10}$	$\pm(1.11)_2 \times 2^{-1} = \pm(0.875)_{10}$	$\pm(1.11)_2 \times 2^{1} = \pm(3.5)_{10}$

の 25 個である。ここで括弧外下添え字、すなわち $(\ )_N$ の N は括弧内数値が N 進数であることを表す。これらの数値は下の数値軸にマーク☆ を記したところにあたる。



-4 -3 -2 -1 0 1 2 3 4

オーバーフローレベル=D_{max}= $(1.11)_2 \times 2^{1} = (3.5)_{10}$

アンダーフローレベル=D_{min}= $(1.00)_2 \times 2^{-1} = (0.5)_{10}$

2. 丸め誤差と打ち切り誤差

・丸め誤差

計算機における“実数”（浮動小数点数）とは、不連続（離散的）で有限な数であるのに対し、数学的な意味での実数は連続で無限個の数の集合を形成する。一般の実数 x は計算機の中では x に最も近い浮動小数点数 x_R に近似的に置き換えられる。これを、「x を x_R に丸める」といい、$x_R - x$ を数値の表現における丸め誤差と呼ぶ。

一般に丸め誤差とは、与えられたアルゴリズムを用いて厳密な算術によって得られる結果と、同じアルゴリズムを用いるけれども丸められた数値や丸められた算術演算によって得られる結果との差のことである。

・打ち切り誤差

打ち切り誤差とは、真の結果と、与えられたアルゴリズムを用いて厳密な算術によって得られる結果との差である。これは、差分法で無限級数を有限個の和で打ち切る近似や、イタレーションを収束するまえに打ち切る近似などのために生じる。

例）微分可能な関数 $f(x)$ の一階微係数に対する下記の差分近似（前進差分近似）

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} \qquad (1)$$

を考えてみよう。Taylor の定理により、$q \in [x, x+h]$ の範囲のある q に対して

$$f(x+h) = f(x) + \frac{h}{1!} f'(x) + \frac{h^2}{2!} f''(q)$$

が成り立つ。これは

$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{h}{2} f''(q)$$

と変形されるので、差分近似式（1）の打ち切り誤差は $Mh/2$ により見積もられる。ただし x 近傍の t に対して $\left| f''(t) \right| \le M$ が満たされるとする。

関数値の誤差は e で見積もられるとすれば、差分近似式（1）における丸め誤差は $2e/h$ となる。このとき誤差の合計は、打ち切り誤差と丸め誤差をあわせて

$$\frac{Mh}{2} + \frac{2e}{h}$$

となる。h が減少するにつれ、第1項は減少し第2項は増加する。従がって刻み幅 h の選択において、打ち切り誤差と丸め誤差のあいだに tradeoff が存在する。上記の誤差を h で微分して 0 とおけば

$$h = 2\sqrt{e/M}$$

となるので、h がこの値をとるとき誤差は最小となることがわかる。

打ち切り誤差を減少させるには、より高精度の差分近似、例えば中心差分近似

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h} \qquad (2)$$

を用いればよい。Taylor の定理

$$f(x+h) = f(x) + \frac{h}{1!} f'(x) + \frac{h^2}{2!} f''(x) + \frac{h^3}{3!} f'''(q_+), \qquad \exists q_+ \in [x, x+h]$$

$$f(x-h) = f(x) - \frac{h}{1!} f'(x) + \frac{h^2}{2!} f''(x) - \frac{h^3}{3!} f'''(q_-), \qquad \exists q_- \in [x-h, x]$$

より

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}(f'''(q_+) + f'''(q_-))$$

が求まるので、差分近似式（２）の打ち切り誤差は $Nh^2/6$ により見積もられる。ただし x 近傍の t_+ と t_- に対して $\left|f'''(t_+) + f'''(t_-)\right| \le N$ が満たされるとする。

丸め誤差を減少させるには、高精度の数値表現と数値演算（例えば double 型）を用いればよい。

3. 科学技術計算におけるモデル化と近似

計算科学においては、様々な近似（すなわち不正確さ）が導入される。

数値計算を行う以前におこる近似： Usually Uncontrollable !!!

・物理モデル：系の物理的な特徴の単純化，例えば粘性法則 $t = m\frac{du}{dy}$ など

・経験的な計測：重力加速度 g，プランクの定数，粘性係数 m など

・以前の計算：入力データが、近似を含む以前の計算により作成されている場合など

数値計算を行う時におこる近似：

・打ち切り、あるいは離散化：数学モデルのある特徴が省略あるいは簡単化されること、例えば微係数を差分により置き換えること

・丸め：計算機で扱える実数の表現や算術演算は、有限な精度に限られるので、厳密ではない。

最終的な数値計算結果の精度は上記の近似の組み合わせを反映する。

例）地球の表面積を

$A = 4\pi r^2$ （r ：半径）

により計算する時、以下のようないくつかの近似が含まれる。

・地球は球としてモデル化されているが、それは実際の形の理想化である。

・半径の値 $r \approx 6370[km]$ は経験的な計測と以前の計算に基づいている。

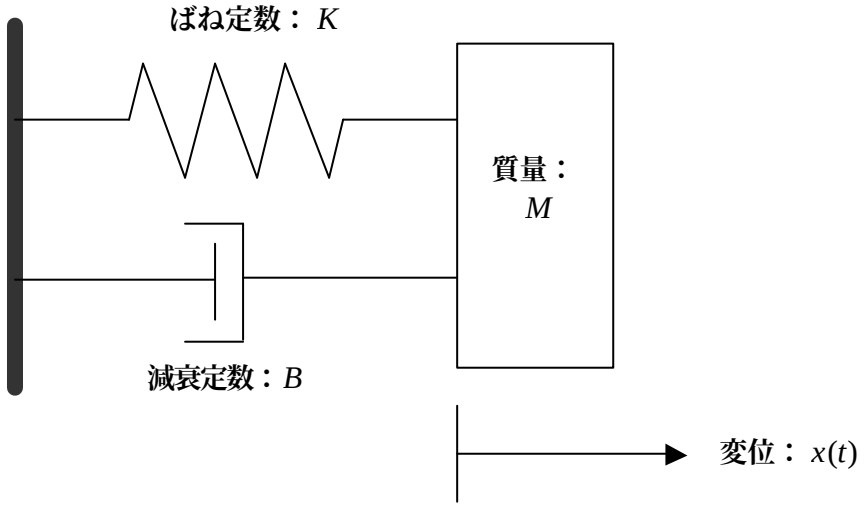
・$\pi(=3.1415\cdots)$ の値は無限に続く数であるが、あるところで打ち切られざるを得ない。

・入力データの数値やそれらの算術演算の結果は計算機により丸められる。

計算結果の精度はこれらの近似すべてに依存する。

4. 常微分方程式の数値計算（マス・バネ・ダンパ系の例）

多くの力学系は下図に示した Mass-Spring-Damper（マス・バネ・ダンパ）系にモデル化される。ここではマス・バネ・ダンパ系の数値シミュレーションを考える。



変位を $x(t)$ とすると、この系の運動方程式は以下のように二階常微分方程式で記述される。

$$M\frac{d^2x(t)}{dt^2}+B\frac{dx(t)}{dt}+Kx(t)=0$$

一般に高階の常微分方程式は、連立一階常微分方程式系に帰着される。例えばこの系の場合、速度 $v(t)\equiv\dfrac{dx(t)}{dt}$ を導入すると、上記の二階常微分方程式は連立一階常微分方程式系

$$\begin{cases}\dfrac{dx(t)}{dt}=v(t)\\ \dfrac{dv(t)}{dt}=-\dfrac{B}{M}v(t)-\dfrac{K}{M}x(t)\end{cases}$$

となり、これを更に以下のように記述する。

$$\frac{d}{dt}\begin{Bmatrix}x\\v\end{Bmatrix}=\begin{Bmatrix}f_x(t,x,v)\\f_v(t,x,v)\end{Bmatrix}\qquad\qquad\begin{cases}f_x(t,x,v)=v\\f_v(t,x,v)=-\dfrac{B}{M}v-\dfrac{K}{M}x\end{cases}$$

常微分方程式系の数値計算法の一つにルンゲ・クッタ法がある。2 次精度ルンゲ・クッタ法においては、$t=t_n$ における値 $x_n=x(t_n),\quad v_n=v(t_n)$ から $t_{n+1}=t_n+h$ における近似値 $x_{n+1},\quad v_{n+1}$ を以下のように求める。

$$\begin{cases}k_{x1}=hf_x(t_n,x_n,v_n)\\k_{v1}=hf_v(t_n,x_{n,},v_n)\\k_{x2}=hf_x(t_n+h,x_n+k_{x1},v_n+k_{v1})\\k_{v2}=hf_v(t_n+h,x_n+k_{x1},v_n+k_{v1})\end{cases}$$

$$x_{n+1}=x_n+\frac{1}{2}(k_{x1}+k_{x2})$$

$$v_{n+1}=v_n+\frac{1}{2}(k_{v1}+k_{v2})$$

課題（常微分方程式の数値計算）

マス・バネ・ダンパ系の運動の数値シミュレーションを 2 次精度ルンゲ・クッタ法を用いて行うプログラムを作成せよ。また結果をパソコン画面上にグラフ表示するプログラム（配布）をバージョンアップすること。

計算アルゴリズムは以下のとおり：

1. パラメータ値をコンソールから入力する。
   物理的な問題を規定するパラメータ値の入力：
   ・バネ定数 K、減衰定数 B、質量 M
   ・変位と速度の初期値 $x0, v0$
   数値計算に必要なパラメータ値の入力：
   ・時間刻み幅 h
2. 初期値の設定： $x_o = x0, \quad v_0 = v0, \quad t_0 = 0.0$
3. $k_{x1,} \quad k_{v1}$ を計算する。
4. $k_{x2,} \quad k_{v2}$ を計算する。
5. $x_{n,} \quad v_n$ から、$x_{n+1,} \quad v_{n+1}$を計算する。
6. 時間の値を h だけ進める： $t_{n+1} = t_n + h$
7. 所定の時間に達するまで、3 ~ 6 を繰り返す。
8. 数値計算結果をファイルに出力する。（計算結果を配列に蓄えておけば、最後にまとめてファイルに出力できる。結果を配列に蓄えずに時間を進めるたびにファイルに出力してもよい。）

SI 単位系を用いてパラメータ値を設定し、20 秒間の変位と速度を数値計算し、与えたパラメータ値と数値計算結果を以下のようにファイルに記入する。

出力ファイル

第 1 行：与えたパラメータ値
第 2 行：記入する数値計算結果の種類の記述
第 3 行以降：数値計算結果（時間番号、時間、変位、速度）

出力ファイルの例

```
K=50.000000 B=10.000000 M=10.000000 x0=10.000000 v0=0.000000 h=0.100000
    n       Time        Position        Velocity
    0  0.0000000e+00  1.0000000e+01  0.0000000e+00
    1  1.0000000e-01  9.7500000e+00 -4.7500000e+00
    2  2.0000000e-01  9.0550003e+00 -8.8112497e+00
    3  3.0000001e-01  7.9915566e+00 -1.2055025e+01
    :       :              :              :
  199  1.9900040e+01  3.5566781e-04  2.4688433e-04
  200  2.0000040e+01  3.7023012e-04  4.8315997e-05
```

上記のファイルを読み込んでパソコン画面上にグラフを表示するプログラムを後に添付する。

提出期限：　１月１０日（木）

提出物：　**数値計算プログラム**　　**ファイル名は**　p○○××e81.c **とすること**
グラフ表示プログラム　　**ファイル名は**　p○○××e82.c **とすること**
数値計算結果の出力ファイル　　**ファイル名は**　p○○××e8.txt **とすること**
○○は学籍番号の上２桁の数字
××は学籍番号の下２桁の数字

＊ **プログラムの第１行目には、コメントで（** /* */ **でくくって)、学籍番号と名前を記述すること**
＊ **数値計算結果の出力ファイル** p○○××e8.txt **は、各自が作成した計算プログラム** p○○××e81.c **によって数値計算を行った結果の出力ファイル**

提出形態：　E-mail **に上記提出ファイルを添付すること**
- **あて先**　cp2-69@cc.tuat.ac.jp
- **題**　cp2ex8
- **本文**　**グラフ表示プログラムをどのようにバージョンアップしたかを記述**
感想、学籍番号、名前を記述

ファイルからデータを読み、グラフを描くプログラム

```
#include <stdio.h>
#include <gucc/gucc.h>        /* グラフィックス関数のためのヘッダーファイルを取り込む */

#define NSIZE 101             /* 時間、変位、速度を蓄える配列のサイズをマクロで定義 */

void G_INIT(void);                    /* 使用するグラフィックス関数のプロトタイプ宣言 */
void G_CLS(void);                     /* 上記に同じ */
void G_LINE(int, int, int, int);      /* 上記に同じ */

int f_input(float *, float *, float *);                          /* 関数のプロトタイプ宣言 */
void graph(int, float *, float *, float *);                      /* 上記に同じ */
float max(int, float *);                                         /* 上記に同じ */
float min(int, float *);                                         /* 上記に同じ */
float g_curve(int, float *, float *, float, float, int, int);    /* 上記に同じ */

void main(void)
{
    int   nmax;
    float t[NSIZE],x[NSIZE],v[NSIZE];
    float tmax,xmax,vmax,tmin,xmin,vmin;
    float tmax0,xmax0,vmax0,tmin0,xmin0,vmin0;
    float tscale, xscale, vscale;
    int   dxpix,dypix, xpix0, ypix0;
    int   xpixel=640, ypixel=480;             /* 画面座標の pixel 数 */

    nmax=f_input(t,x,v);     /* ファイルから数値計算結果を読む関数の返り値はデータ点数 */
    printf("\nnmax=%d",nmax);

    tmax=t[nmax-1];                           /* 時間 t の最大値 */
    tmin=t[0];                                /* 時間 t の最小値 */
    xmax=max(nmax, x);                        /* 変位 x の最大値 */
    xmin=min(nmax, x);                        /* 変位 x の最小値 */
    vmax=max(nmax, v);                        /* 速度 v の最大値 */
    vmin=min(nmax, v);                        /* 速度 v の最小値 */

    printf("\n 時間 t、変位 x、速度 v の最小値と最大値：\n");
    printf("(tmin,tmax)=(%f,%f)\n",tmin,tmax);
    printf("(xmin,xmax)=(%f,%f)\n",xmin,xmax);
    printf("(vmin,vmax)=(%f,%f)\n",vmin,vmax);

    tmax0=tmax;                          /* グラフに表示する時間軸（横軸）の最大値 */
    if(xmax>-xmin)                       /* グラフに表示する変位軸（縦軸）の最大値 */
      xmax0=xmax;
    else
      xmax0=-xmin;
    if(vmax>-vmin)                       /* グラフに表示する速度軸（縦軸）の最大値 */
      vmax0=vmax;
    else
      vmax0=-vmin;
```

```
    tmin0=0.0;                          /* グラフに表示する時間軸（横軸）の最小値 */
    xmin0=-xmax0;                       /* グラフに表示する変位軸（縦軸）の最小値 */
    vmin0=-vmax0;                       /* グラフに表示する速度軸（縦軸）の最小値 */

    printf("\n グラフの時間軸（横軸）、変位軸（縦軸）、速度軸（縦軸）の最小値と最大値：\n");
    printf("(tmin0,tmax0)=(%f,%f)\n",tmin0,tmax0);
    printf("(xmin0,xmax0)=(%f,%f)\n",xmin0,xmax0);
    printf("(vmin0,vmax0)=(%f,%f)\n",vmin0,vmax0);

    dxpix=xpixel/10;                        /* 画面の横方向余白 pixel 数 */
    dypix=ypixel/10;                        /* 画面の縦方向余白 pixel 数 */
    tscale=(xpixel-2*dxpix)/(tmax0-tmin0);      /* 時間軸のスケール */
    xscale=(ypixel-2*dypix)/(xmax0-xmin0);      /* 変位軸のスケール */
    vscale=(ypixel-2*dypix)/(vmax0-vmin0);      /* 速度軸のスケール */
    xpix0=dxpix;                            /* (xpic0, ypic0)：グラフ原点の画面座標 */
    ypix0=ypixel/2;

    G_INIT();                       /* 画面の初期化 */
    G_CLS();                        /* 画面消去 */

    G_COLOR(15);                     /* 色番号 15（黒）を指定 */
    G_LINE(dxpix,ypix0,xpixel-dxpix,ypix0);   /* 横軸（時間）を描く */
    G_LINE(xpix0,dypix,xpix0,ypixel-dypix);   /* 縦軸（変位・速度）を描く */

    G_COLOR(5);                                     /* 色番号 5（高輝度の赤）を指定 */
    g_curve(nmax, t, x, tscale, xscale, xpix0, ypix0);   /* 変位の履歴(t,x)をグラフ表示 */

    G_COLOR(3);                                      /* 色番号 3（高輝度の青）を指定 */
    g_curve(nmax, t, v, tscale, vscale, xpix0, ypix0);   /* 速度の履歴(t,v)をグラフ表示 */

    getch();   /* 標準入力から 1 文字読み込む（任意のキーが押されるまで画面を保持） */
}

/* 最大値を求める */
float max(int nmax, float *x)
{
    int n;
    float xmax;

    xmax=x[0];
    for(n=0; n<nmax; n++){
       if(x[n]>xmax)  xmax=x[n];
    }
    return(xmax);
}

/* 最小値を求める */
float min(int nmax, float *x)
{
    int n;
    float xmin;
```

```
    xmin=x[0];
    for(n=0; n<nmax; n++){
       if(x[n]<xmin)  xmin=x[n];
    }
    return(xmin);
}

/* 点列 (x[n],y[n]), n=0,1,...,nmax-1 を描く */
float g_curve(int nmax, float *x, float *y, float xscale, float yscale, int xpix0, int ypix0)
{
    int n,px0, py0, px1, py1;

    px0=xpix0+(int)(x[0]*xscale);
    py0=ypix0-(int)(y[0]*yscale);

    for(n=1; n<nmax; n++){
        px1=xpix0+(int)(x[n]*xscale);
        py1=ypix0-(int)(y[n]*yscale);   /* 画面座標の正の方向とグラフの正の方向は逆 */
        G_LINE(px0,py0,px1,py1);        /* 画面座標の(px0,py0)と(px1,py1)を線分で結ぶ */
        px0=px1;
        py0=py1;
    }
}

/* ファイルから数値計算結果を読み込む */
int f_input(float *t, float *x, float *v)
{
    int  dummy, n;
    FILE *fp;
    char fname[20], buf[80];

    printf("数値計算結果が保存されているファイルの名===>");
    scanf("%s",fname);                              /* ファイル名を読み込む */

    if( (fp=fopen(fname,"r"))==NULL ){              /* ファイルオープンとエラー処理 */
       printf("ファイル %s をオープンできません。\n",fname);
       exit(1);
    }
    fgets(buf, 80, fp);      /* ファイルから第１行（計算条件）を読み込む */
    printf("\n%s",buf);
    fgets(buf, 80, fp);      /* ファイルから第２行（出力値の記述）を読み込む */
    printf("%s",buf);
    n=0;
    while( feof(fp)==0 ){    /* ファイルに記述されたデータが終わらない間、以下を繰り返す */
       fscanf(fp, "%d %f %f %f\n", &dummy, &t[n], &x[n], &v[n] );  /* ファイルから読む */
       printf("%5d \t %f \t %f \t %f\n", n, t[n], x[n], v[n] );
       n++;                                     /* n はファイルから読み込んだデータ点数 */
    }
    fclose(fp);                                 /* ファイルのクローズ */
    return(n);                                  /* データ点数を関数の返り値とする */
}
```