数式処理と保存則のエントロピー解の波面追跡

吉川　敦
九州大学大学院数理学研究院

目 次

1　近似保存則のエントロピー解と Maple

数式処理ソフト Maple を用いて，初期値が階段関数の場合の保存則のエントロピー解を構成したい．まず，与えられた流量[1] $f(v)$ の（N 次の）折れ線近似は

$$f_N(v) = \begin{cases} f(v), & v = v_m \\ (1-\theta)\, f(v_m) + \theta\, f(v_{m+1}), & v = (1-\theta)\, v_m + \theta\, v_{m+1} \end{cases} \tag{1}$$

（$0 < \theta < 1$）である．ただし，

$$v_m = \frac{m}{2^N},\ m = 0,\ \pm 1,\ \pm 2,\ \cdots$$

[1]以下のプロシデュアが機能するためには，流量が v の有理数係数の多項式であるとの想定が必要である．

とする．N 次の近似流量を伴う保存則[2]

$$\frac{\partial}{\partial t}u(t,x)+\frac{\partial}{\partial x}f_N(u(t,x))=0,\ t>0,\ -\infty<x<+\infty \tag{2}$$

$$u(0,x)=u_0(x) \tag{3}$$

を，初期関数が階段関数

$$u_0(x)=\begin{cases} vlist_1\,, & x<xlist_1 \\ \quad\cdots & \\ vlist_{i+1}\,, & xlist_i<x<xlist_{i+1} \\ \quad\cdots & \\ vlist_{n+1}\,, & xlist_n<x \end{cases} \tag{4}$$

の場合に考える．ただし，$xlist$, $vlist$ は実数（実は，有理数）の組

$$xlist=[xlist_1,\,xlist_2,\,\cdots,\,xlist_n] \tag{5}$$

$$vlist=[vlist_1,\,vlist_2,\,\cdots,\,vlist_n,vlist_{n+1}] \tag{6}$$

であって，しかも，$xlist$ の成分は単調増大であり，一方，$vlist$ の成分は 2 進有理数 $vlist_i=\dfrac{m_i}{2^N}$ であるとする．

このとき，数式処理ソフト Maple によって，エントロピー解を構成し，また，解に現れる不連続線をすべて求めるプロシデュアを与えることができる．例えば，

```
> kai:=entropy_solution:-zenbukai
> ([0,1,2],[0,1,2,0],2,u->u^3,15):
```

によって（出力はしていないが），$N=2$, $f(u)=u^3$ の場合に，初期関数が $xlist=[0,1,2]$, $vlist=[0,1,2,0]$ に対応するときのエントロピー解 kai を求めることができる[3]．また，この解の不連続線の全体 $hurenzokusen$ は，

```
> hurenzokusen:=tuiseki:-zensujisen
> (t,[0,1,2],[0,1,2,0],2,u->u^3,15):
```

によって，求めることができる．ここで，`entropy_solution:-zenbukai` あるいは `tuiseki:-zensujisen` は，Maple のモデュール `entropy_solution` あるいは `tuiseki` の中に当該のプロシデュア `zenbukai` や `zensujisen` が記述してあるからである．本来は，これらも整理して，例えば，パッケージ化してしまうのがよいのだろうが，ここでは，そこまではやらない[4]．

[2](2) において $f_N(v)$ ではなく $f(v)$ が現れるのがもともとの流量 $f(u)$ を伴う保存則である．保存則のエントロピー解の概念については別に説明する．

[3]数値 15 は反復過程の回数を手動で制御するためである．本来，反復回数は自動的に判定すべきであるが，計算時間の事前評価が難しいためにこのようにしてある．ただし，この数値が特に `zensujisen` の出力個数よりも大きくないと正しい結果は返らない．

[4]`zenbukai` あるいは `zensujisen` に至る手順のつながりを個々に説明しているために反復判定過程や無駄な重複が多い．これらを整理するだけで随分と違うはずではあるが，多少とも実用化を図るためには一層の工夫が必要である．

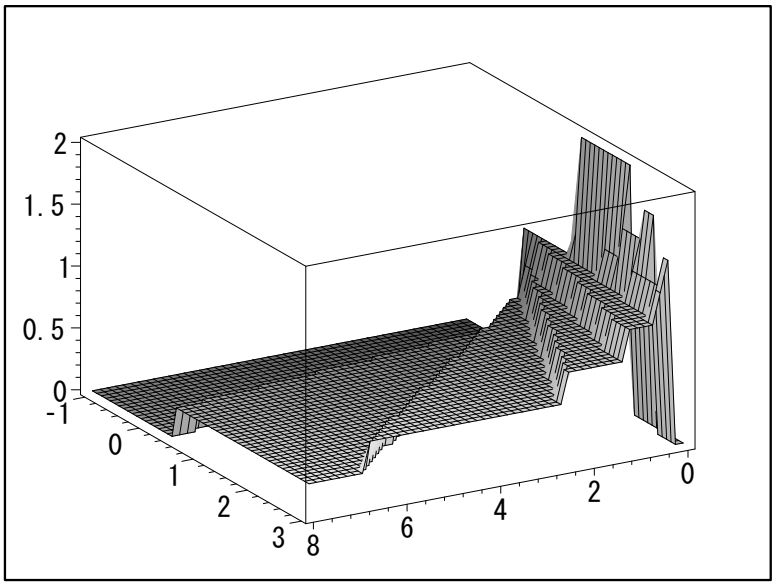

図 1: $kai(t,x)$, $0<t<8$, $-1<x<3$

エントロピー解 $kai(t,x)$ および不連続線 $hurenzokusen$ の画像化を図 1 および図 2 に示す[5].

次節で，エントロピー解構成と関連するプロシデュアについて詳細に説明する．アイデアは Dafermos[2] の手順を丹念に実行することだけである（Bressan[1] も見よ）.

2 Maple によるコード化の説明

2.1 Maple による折れ線グラフの凸化

近似 Riemann 問題のエントロピー解の構成のためには，近似流量 $f_N(u)$ に対し，$u_->u_+$ または $u_-<u_+$ に応じて，上への凸化 $f_N^*(u;u_-,u_+)$ または下への凸化 $f_{N*}(u;u_-,u_+)$ を構成する必要がある（§4.2.2).

この区間での近似流量 $f_N(u)$ のグラフは，$u_-=\frac{\ell}{2^N}$，$u_+=\frac{r}{2^N}$ として，u_-, u_+ の間を等分した

$$u_i=\frac{i}{2^N},\quad \ell \lesseqgtr i \lesseqgtr r$$

と，そこでの f の値 $f(u_i)$ とを組合わせた点 $(u_i,\, f(u_i))$ を結んだ折れ線図形である．この折れ線図形の下方および上方の領域の凸包の境界となる折れ

[5]図 2 には，干渉前に不自然な曲がり方を示している線があるが，これは画像化に際してのエラーであり，画像の局所化により改善される.

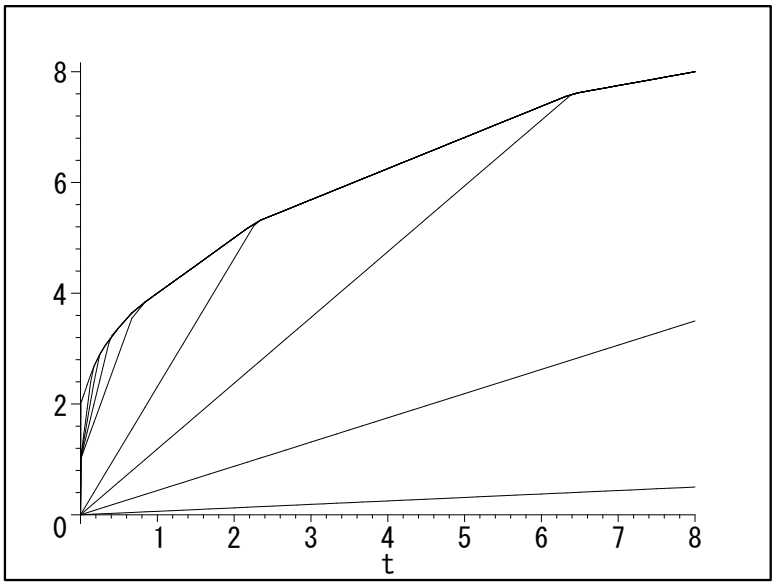

図 2: 不連続線全体 $0 < t < 8$

線が，それぞれ，$f_N(u)$ の上への凸化，下への凸化のグラフに相当する（後出の図 7，図 8 参照）.

近似流量 $f_N(u)$ は N に応じて $f(u)$ から求められるものである．その際，$u_\pm$ の管理は，ℓ, r を N に応じて選びなおす可能性もあり，凸化の手続きと近似流量とは一旦切り離しておくべきであろう．そこで，以下では，近似流量のグラフとは限らない折れ線図形から，その下方領域，上方領域の凸包の境界となる折れ線図形を求める手順を Maple のプロシデュア群に分解し，一括して，module 化したもの `convexify` を示す.

話の順序として，`convexify` には，折れ線図形を作るためのプロシデュア `gen` と `nawabari` を収めておく[6]．`gen` は $xpair = [x_1, x_2]$，$ypair = [y_1, y_2]$ をデータとして，２点 $(x_1, y_1), (x_2, y_2)$ を結ぶ弦，すなわち，

$$\mathrm{gen}(xpair, ypair)(x) = \begin{cases} 0, & x < x_1 \\ \dfrac{x_2 - x}{x_2 - x_1}\, y_1 + \dfrac{x - x_1}{x_2 - x_1}\, y_2, & x_1 < x < x_2 \\ 0, & x_2 < x \end{cases} \qquad (7)$$

を出力させるものである．`nawabari` は，入力データ $xlist = [x_1, \cdots, x_n]$，$ylist = [y_1, \cdots, y_n]$ に対し，n 個の点 $(x_1, y_1), \cdots, (x_n, y_n)$ を結ぶ折れ線を，$\mathrm{gen}([x_i, x_{i+1}], [y_i, y_{i+1}])$ を逐次接続して，描くものである．したがって，

$$\mathrm{nawabari}(xlist, ylist)(x) = \sum_{i=1}^{n-1} \mathrm{gen}([x_i, x_{i+1}], [y_i, y_{i+1}])(x)$$

[6] `gen` は弦，`nawabari` は縄張りのつもりである.

を出力する.

以上を Maple のプロシデュアとしてコード化したものを掲げる.

工程 2.1.1　プロシデュア gen 及び nawabari のコードは次の通りである. ただし, 以下では, $xpair = [xpair_1, xpair_2]$ などのように, リスト（例えば $list$ ）の第 j 成分は添え数 j を付して（例えば $list_j$ のように）表している.

プロシデュア gen では入力データ $xpair, ypair$ の整合性と $x_1 = xpair_1 < x_2 = xpair_2$ をまず検査してから, (7) を計算する[7].

```
gen := proc(xpair, ypair)
local x;
  if 0 < (nops(xpair) − 2)^2 + (nops(ypair) − 2)^2 then
    RETURN(‘convexify : −gen = input_error0‘)
  end if;
  if xpair_2 ≤ xpair_1 then RETURN(‘convexify : −gen = input_error1‘) end if;
  x → piecewise(x < xpair_1, 0, xpair_1 < x and x < xpair_2,
    (xpair_2 − x) ∗ ypair_1/(xpair_2 − xpair_1)
     + (x − xpair_1) ∗ ypair_2/(xpair_2 − xpair_1), xpair_2 < x, 0)
end proc

nawabari := proc(xlist, ylist)
local i, j, n, gn, x;
  n := nops(xlist) ;
  if n < 2 then RETURN(‘convexify : −nawabari = input_error0‘) end if;
  if nops(ylist) ≠ n then RETURN(‘convexify : −nawabari = input_error1‘) end if;
  for i to n − 1 do gn_i := gen([xlist_i, xlist_{i+1}], [ylist_i, ylist_{i+1}]) end do ;
  x → sum(gn_j(x), j = 1..n − 1)
end proc
```

nawabari では入力データ $xlist, ylist$ の整合性が最初に検査されてから, gen を利用する.

注意 2.1.1　既述の（モデュール action （§2.3.3）の）プロシデュア dan は, (convexify の）プロシデュア gen の特別な場合と考えられる. すなわち,

$$\mathrm{dan}([b_1, b_2], v) = \mathrm{gen}([b_1, b_2], [v, v])$$

である.

例 2.1.1　gen 及び nawabari の出力例をグラフ化したものが, 図 3, 図 4 である.

[7]エラー・メッセージに, モデュールとプロシデュアへの参照を含めてある. 他のものでも同様.

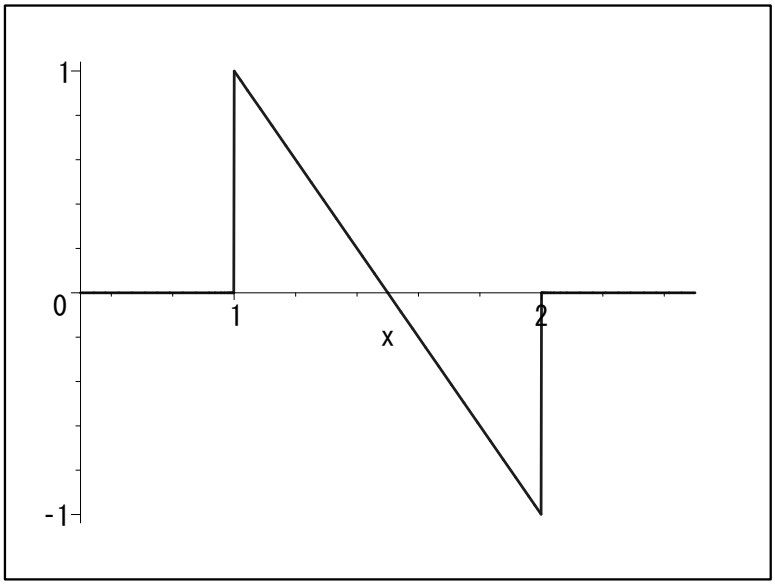

図 3: gen$([1,2],[1,-1])(x)$ のグラフ

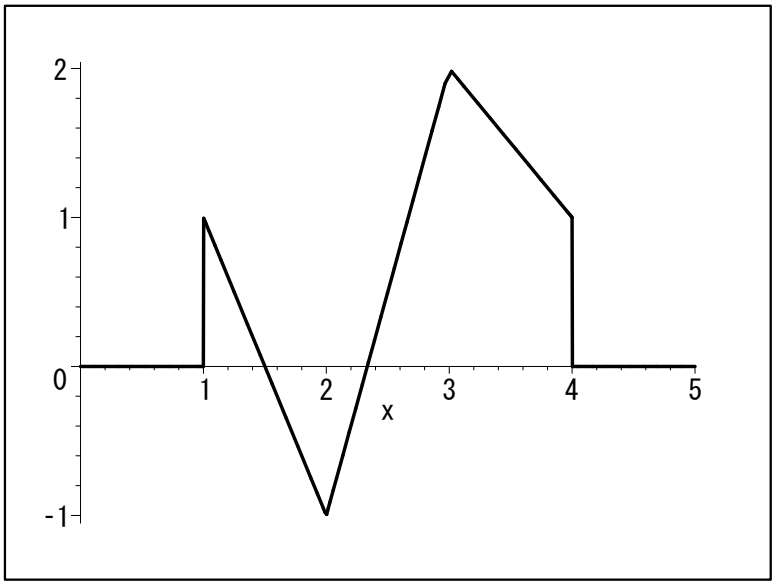

図 4: nawabari$([1,2,3,4],[1,-1,2,1])(x)$ のグラフ

折れ線グラフ nawabari($xlist, ylist$) の上下への凸化を計算するために，各弦の勾配の管理をしたい．そこで，module `convexify` には，プロシデュア `katamuki`，`migikata`，`hidarikata` も用意する．

`katamuki` は入力データ $xlist = [x_1, \cdots, x_n]$，$ylist = [y_1, \cdots, y_n]$ から定まる n 個の点 $(x_1, y_1), \cdots, (x_n, y_n)$ の隣り合う 2 点を結ぶ弦の勾配

$$dir_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad i = 1, \cdots, n-1,$$

を計算し，リスト $[dir_1, \cdots, dir_{n-1}]$ として出力するものである．これに対し，`migikata` は，点 (x_1, y_1) と残りの点 $(x_2, y_2), \cdots, (x_n, y_n)$ を結ぶ弦の勾配

$$dir_i = \frac{y_{i+1} - y_1}{x_{i+1} - x_1}, \quad i = 1, \cdots, n-1,$$

を計算し，リスト $[dir_1, \cdots, dir_{n-1}]$ として出力する．また，`hidarikata` は，点 (x_n, y_n) と残りの点 $(x_1, y_1), \cdots, (x_{n-1}, y_{n-1})$ を結ぶ弦の勾配

$$dir_i = \frac{y_i - y_n}{x_i - x_n}, \quad i = 1, \cdots, n-1,$$

を計算し，リスト $[dir_1, \cdots, dir_{n-1}]$ として出力する．すなわち，プロシデュアとしてコード化すれば次のようになる．

工程 2.1.2　以下は，プロシデュア `katamuki`，`migikata`，`hidarikata` のコードである．いずれも，入力データ $xlist$, $ylist$ の整合性の検査を経て，必要な計算を実行する．

```
katamuki := proc(xlist, ylist)
local i, n, dir;
  n := nops(xlist);
  if n < 2 then RETURN(`convexify : -katamuki = input_error0`) end if;
  if nops(ylist) ≠ n then RETURN(`convexify : -katamuki = input_error1`) end if;
  for i to n - 1 do dir_i := (ylist_{i+1} - ylist_i)/(xlist_{i+1} - xlist_i) end do;
  [seq(dir_i, i = 1..n - 1)]
end proc
```

```
migikata := proc(xlist, ylist)
local i, n, dir;
  n := nops(xlist);
  if n < 2 then RETURN(`convexify : -migikata = input_error0`) end if;
  if nops(ylist) ≠ n then RETURN(`convexify : -migikata = input_error1`) end if;
  for i to n - 1 do dir_i := (ylist_{i+1} - ylist_1)/(xlist_{i+1} - xlist_1) end do;
  [seq(dir_i, i = 1..n - 1)]
end proc
```

```
hidarikata := proc(xlist, ylist)
local i, n, dir;
  n := nops(xlist);
  if n < 2 then RETURN(`convexify : −hidarikata = input_error0`) end if;
  if nops(ylist) ≠ n then RETURN(`convexify : −hidarikata = input_error1`) end if;
  for i to n − 1 do dir_i := (ylist_n − ylist_i)/(xlist_n − xlist_i) end do;
  [seq(dir_i, i = 1..n − 1)]
end proc
```

いずれも出力データはリストである．`katamuki`，`migikata`，`hidarikata` いずれにおいても，成分数は入力データのリスト $xlist$ または $ylist$ よりも 1 成分少ない．

出力例を示す．

$$\text{katamuki}([1,\,2,\,3,\,4],\,[1,\,-1,\,2,\,1]) = [-2,\,3,\,-1]$$

$$\text{migikata}([1,\,2,\,3,\,4],\,[1,\,-1,\,2,\,1]) = [-2,\,\frac{1}{2},\,0]$$

$$\text{hidarikata}([1,\,2,\,3,\,4],\,[1,\,-1,\,2,\,1]) = [0,\,1,\,-1]$$

この結果を見やすくするために，`migikata`，`hidarikata` を画像化しておこう[8]．

例えば，migikata$([1,2,3,4],[1,-1,2,1])$ を，点 $(1,1)$ と点 $(2,-1)$，$(3,2)$，$(4,1)$ を結ぶ弦によって画像化したものが，図 5 である．

[8] プロシデュア `migikataplot`，`hidarikataplot` による．すなわち，

```
migikataplot := proc(xlist, ylist)
local i, n, x, g;
  n := nops(xlist);
  for i from 2 to n do g_i := plot(
    convexify : −gen([xlist_1, xlist_i], [ylist_1, ylist_i])(x),
    x = xlist_1 − 1/2..xlist_n + 1/2, thickness = 2)
  end do;
  plots_display(seq(g_i, i = 2..n))
end proc
```

```
hidarikataplot := proc(xlist, ylist)
local i, n, x, g;
  n := nops(xlist);
  for i to n − 1 do g_i := plot(convexify : −gen([xlist_i, xlist_n], [ylist_i, ylist_n])(x),
    x = xlist_1 − 1/2..xlist_n + 1/2, thickness = 2)
  end do;
  plots_display(seq(g_i, i = 1..n − 1))
end proc
```

を利用する．ここで，$convexify : -gen$ などは，module `convexify` に含まれているプロシデュア gen を呼んでいることを示す．その他，Maple の plots パッケージ も呼んでいる．ただし，画像化のためだけのプロシデュアであり，module `convexify` には収めない．プロシデュアの詳細な解説も省略する．

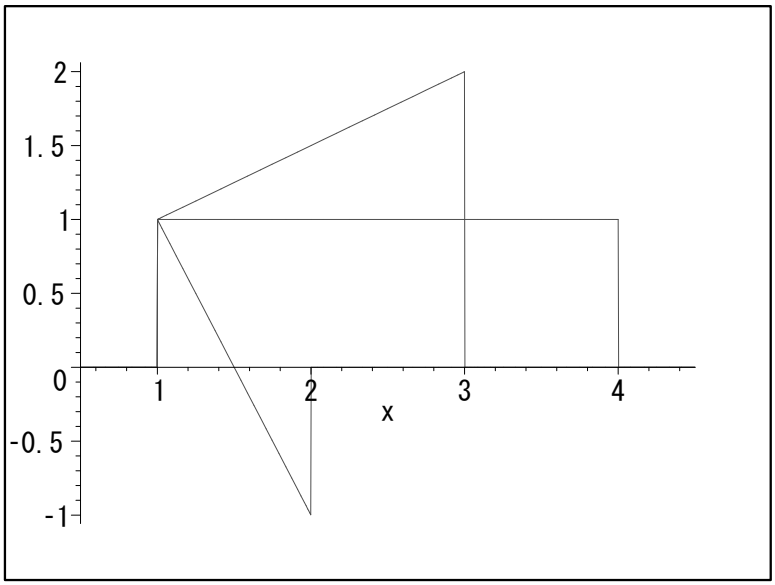

図 5: migikata($[1,2,3,4],[1,-1,2,1]$) の画像化

また，hidarikata($[1,2,3,4],[1,-1,2,1]$) の画像化には，点 $(4,1)$ と点 $(1,1)$，$(2,-1)$，$(3,2)$ とを結ぶ弦を利用する．すなわち，図 6 である．

折れ線関数 nawabari$(xlist,ylist)(x)$ のグラフの左端の点 $(xlist_1,ylist_1)$ とグラフ上の他の折れ目の点 $(xlist_{i+1},ylist_{i+1})$ を結ぶ弦のうち勾配が最小のものは，この折れ線関数の下への凸化のグラフの弦でもあるはずである．そこで，migikata$(xlist,ylist)$ の最小値を最初に実現する点 $(xlist_{i+1},ylist_{i+1})$ の添え数を求めるプロシデュア `migisita` を用意する．同様に考えて，折れ線関数 nawabari$(xlist,ylist)(x)$ のグラフの右端の点と他の折れ目の点とを結ぶ弦で勾配が最小のものは，この折れ線関数の上への凸化の弦でもあるはずである．この勾配は hidarikata$(xlist,ylist)$ の最小値である．この最小値を実現する右端にもっとも近い折れ目の点の添え数を求めるプロシデュア `hidarisita` も用意する[9]．

工程 2.1.3　プロシデュア `migiasi`，`hidariasi` のコードを与える．

[9] migikata$(xlist,ylist)$ の最大値を求めても上への凸化への道が拓けるはずである．ただし，今の段階では，最適なコード化を目指すことは課題にはしていない．

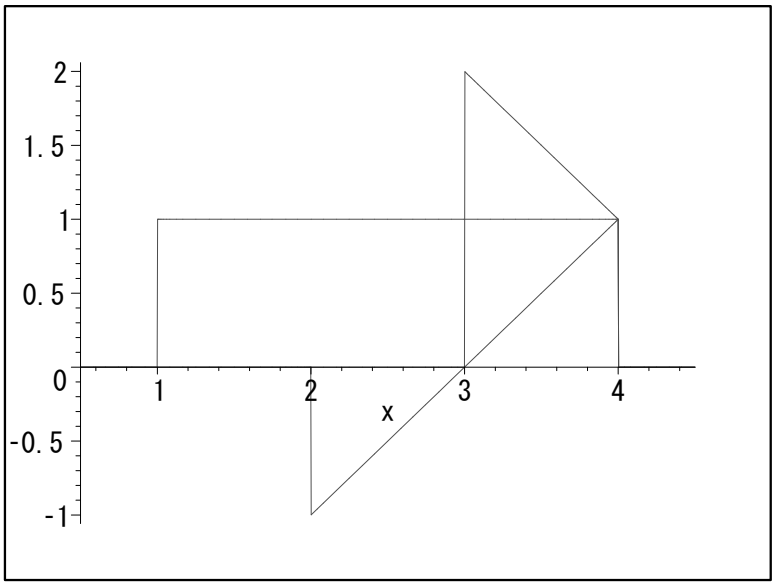

図 6: hidarikata($[1,2,3,4],[1,-1,2,1]$) の画像化

migisita := **proc**(*xlist*, *ylist*)
local *a*, *b*, *i*, *n*, *ser*, *bango*;
a := migikata(*xlist*, *ylist*) ;
n := nops(*a*) ;
ser := seq(a_i, $i = 1..n$) ;
b := min(*ser*) ;
bango := {} ;
for *i* **to** *n* **do if** $[ser]_i = b$ **then** *bango* := *bango* union $\{i\}$ **end if end do** ;
$1 + \min(\mathrm{seq}(bango_i,\ i = 1..\mathrm{nops}(bango)))$
end proc

hidarisita := **proc**(*xlist*, *ylist*)
local *a*, *b*, *i*, *n*, *ser*, *bango*;
a := hidarikata(*xlist*, *ylist*) ;
n := nops(*a*) ;
ser := seq(a_i, $i = 1..n$) ;
b := min(*ser*) ;
bango := {} ;
for *i* **to** *n* **do if** $[ser]_i = b$ **then** *bango* := *bango* union $\{i\}$ **end if end do** ;
$\max(\mathrm{seq}(bango_i,\ i = 1..\mathrm{nops}(bango)))$
end proc

なお，上のプロシデュアでは，いずれも該当する添え数すべての集合 *bango* を一旦構成しているが，手順上は無駄である．

`migikata`, `hidarikata` と比較した出力例[10]を挙げる．

$$\text{migikata}([1,\,2,\,3,\,4,\,5],\,[1,\,-1,\,2,\,-5,\,3]) = [-2,\,\frac{1}{2},\,-2,\,\frac{1}{2}]$$

$$\text{migisita}([1,\,2,\,3,\,4,\,5],\,[1,\,-1,\,2,\,-5,\,3]) = 2$$

$$\text{hidarikata}([1,\,2,\,3,\,4,\,5],\,[1,\,-1,\,2,\,-5,\,3]) = [\frac{1}{2},\,\frac{4}{3},\,\frac{1}{2},\,8]$$

$$\text{hidarisita}([1,\,2,\,3,\,4,\,5],\,[1,\,-1,\,2,\,-5,\,3]) = 3$$

ところで，われわれの本来の目標は，折れ線関数 $\text{nawabari}(xlist, ylist)(x)$ の上または下への凸化である．プロシデュア `migisita`，`hidarisita` は凸化の第一歩の情報しか与えないので，さらに，この手順を繰り返す必要がある．このためには，入力データ $xlist$, $ylist$ を更新しなければならない．

そこで，入力データ $xlist = [x_1, x_2, \cdots, x_n]$，$ylist = [y_1, y_2, \cdots, y_n]$ をもとに，$\text{migisita}(xlist, ylist) = m$ として，$[x_m, \cdots, x_n]$，$[y_m, \cdots, y_n]$ を出力させるプロシデュア `miginokori` を利用すれば，`migisita` によって下への凸化のグラフにおける次の折れ目の点の添え数がわかる．しかも，この操作の繰り返しをプロシデュア化すれば，折れ線関数 $\text{nawabari}(xlist, ylist)(x)$ の下への凸化の折れ目の点の添え数の集合を作ることができる．これが，プロシデュア `migiasi` である．

工程 2.1.4　プロシデュア `miginokori`，`migiasi` のコードは次のように与えられる．

```
miginokori := proc(xlist, ylist)
local m, n, i;
    m := migisita(xlist, ylist) ;
    n := nops(xlist) ;
    [seq(xlist_i, i = m..n)], [seq(ylist_i, i = m..n)]
end proc
```

[10] これらのプロシデュアはモデュール `convexify` に一括して収めることを前提にコード化されてあり，したがって，単独でプロシデュアを走らせることができないことがある．その場合は，`convexify:-migisita` などの形で適用する．

```
migiasi := proc(xlist, ylist)
local i, m, l, L, asi;
    L_0 := xlist, ylist ;
    m_0 := 0 ;
    l_0 := 1 ;
    asi := {l_0} ;
    for i to nops(xlist) do
        m_i := migisita(L_{i-1_1}, L_{i-1_2}) ;
        l_i := l_{i-1} + m_i - 1 ;
        if l_i ≤ nops(xlist) then
            asi := asi union {l_i} ; L_i := miginokori(L_{i-1_1}, L_{i-1_2})
        else i := nops(xlist) + 1
        end if
    end do;
    asi
end proc
```

詳細の解説は省略する.

上で計算した `migikata`, `migisita` の続きの出力[11]を示す.

$$\text{miginokori}([1, 2, 3, 4, 5], [1, -1, 2, -5, 3]) = ([2, 3, 4, 5], [-1, 2, -5, 3])$$

$$\text{migiasi}([1, 2, 3, 4, 5], [1, -1, 2, -5, 3]) = \{1, 2, 4, 5\}$$

一方，折れ線関数 $\text{nawabari}(xlist, ylist)(x)$ のグラフの右端の点から出発し，$\text{hidarisita}(xlist, ylist) = m$ のとき，リスト $[x_1, \cdots, x_m]$, $[y_1, \cdots, y_m]$ を出力するプロシデュア `hidarinokori` に，さらに，`hidarisita` を適用反復して，折れ線関数の上への凸化のグラフの折れ目の点の添え数の集合を求めるプロシデュア `hidariasi` が得られる． すなわち，

工程 2.1.5

```
hidarinokori := proc(xlist, ylist)
local m, i;
    m := hidarisita(xlist, ylist) ; [seq(xlist_i, i = 1..m)], [seq(ylist_i, i = 1..m)]
end proc
```

[11] 脚注 10 参照.

```
hidariasi := proc(xlist, ylist)
local i, m, l, L, asi;
    asi := {nops(xlist)};
    L_0 := xlist, ylist;
    m_0 := nops(xlist);
    for i to nops(xlist) − 1 do
        m_i := hidarisita(L_{i−1_1}, L_{i−1_2});
        if 1 ≤ m_i then
            asi := asi union {m_i}; L_i := hidarinokori(L_{i−1_1}, L_{i−1_2})
        else i := nops(xlist)
        end if
    end do;
    asi
end proc
```

出力例を示す.

hidarinokori([1, 2, 3, 4, 5], [1, −1, 2, −5, 3]) = ([1, 2, 3], [1, −1, 2])

hidariasi([1, 2, 3, 4, 5], [1, −1, 2, −5, 3]) = {1, 3, 5}

プロシデュア `migiasi`，`hidariasi` があれば，凸化まではすぐである．

実際，$xlist$，$ylist$ から migiasi($xlist, ylist$) に属する添え数の成分だけを抜き出すことにより，折れ線関数 nawabari($xlist, ylist$)(x) の下への凸化のグラフを描くための折れ目の点の候補がすべて得られる．その際，グラフの折れ目の候補の点が見かけだけ，つまり，その点の前後での勾配が一致して，候補の点が折れ目にならないことがあれば，そのような点は除いておかなければならない．こうして整理されたリストを出力するのがプロシデュア `sitaetotu` であり，さらに，これにプロシデュア `nawabari` を施して関数化したものがプロシデュア `sitaenototuka` である．すなわち，折れ線関数 nawabari($xlist, ylist$)(x) の下への凸化は sitaenototuka($xlist, ylist$)(x) である．

折れ線関数 nawabari($xlist, ylist$)(x) の上への凸化を得るためには，プロシデュア `hidariasi` からプロシデュア `ueetotu`，`ueenototuka` を，上と全く同様に，定めることができ，ueenototuka($xlist, ylist$)(x) が求める上への凸化となる．

工程 2.1.6　プロシデュア `sitaetotu`，`ueetotu`，`sitaenototuka`，および `ueenototuka` は次のようにコード化される．

```
sitaetotu := proc(xlist, ylist)
local i, j, asi, assi, Lx, Ly, X, Y, K, A;
    asi := migiasi(xlist, ylist);
    Lx := [seq(xlist_i, i = asi)];
    Ly := [seq(ylist_i, i = asi)];
    K := katamuki(Lx, Ly);
    if nops(K) = nops({seq(K_j, j = 1..nops(K))}) then X := Lx; Y := Ly
    else
        A := {};
        for i to nops(K) - 1 do if K_i = K_{i+1} then A := A union {asi_{i+1}} end if
        end do;
        assi := asi minus A;
        X := [seq(xlist_i, i = assi)];
        Y := [seq(ylist_i, i = assi)]
    end if;
    X, Y
end proc

ueetotu := proc(xlist, ylist)
local i, j, asi, assi, Lx, Ly, X, Y, K, A;
    asi := hidariasi(xlist, ylist);
    Lx := [seq(xlist_i, i = asi)];
    Ly := [seq(ylist_i, i = asi)];
    K := katamuki(Lx, Ly);
    if nops(K) = nops({seq(K_j, j = 1..nops(K))}) then X := Lx; Y := Ly
    else
        A := {};
        for i to nops(K) - 1 do if K_i = K_{i+1} then A := A union {asi_{i+1}} end if
        end do;
        assi := asi minus A;
        X := [seq(xlist_i, i = assi)];
        Y := [seq(ylist_i, i = assi)]
    end if;
    X, Y
end proc

ueenototuka := proc(xlist, ylist) local x; x → nawabari(ueetotu(xlist, ylist))(x) end proc
sitaenototuka := proc(xlist, ylist) local x; x → nawabari(sitaetotu(xlist, ylist))(x) end proc
```

出力例を示す（図 7，図 8）.

最後に，以上のプロシデュアを module `convexify` にまとめたコードを示す.

工程 2.1.7　module `convexify` の export 変数はつぎの通り.

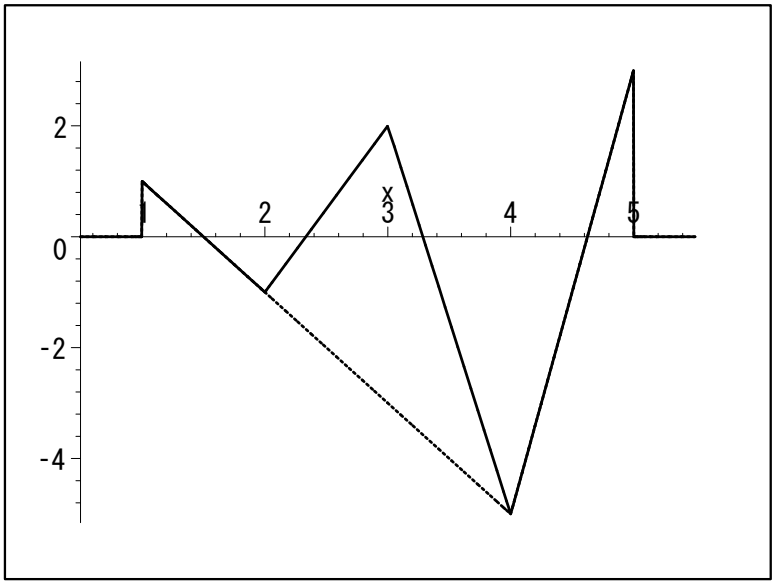

図 7: sitaenototuka$([1,2,3,4,5],[1,-1,2,-5,1])(x)$ の模式図（破線）

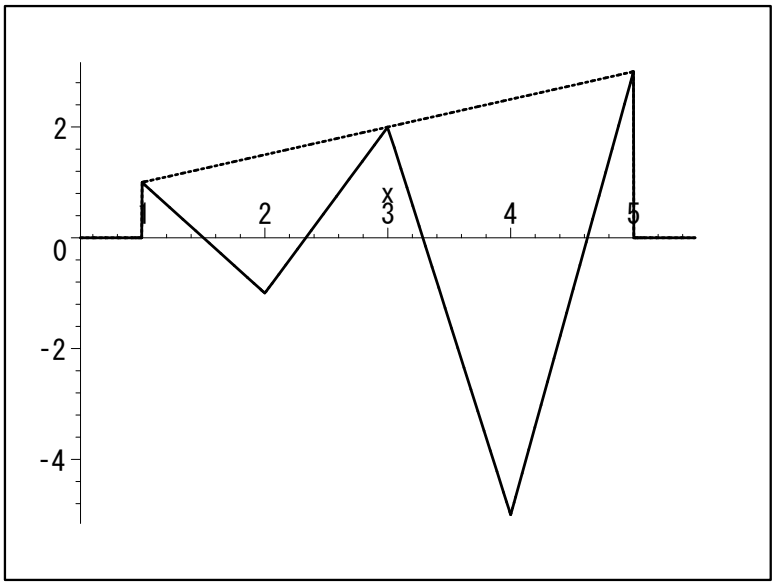

図 8: ueenototuka$([1,2,3,4,5],[1,-1,2,-5,1])(x)$ の模式図（破線）

convexify := **module**()
export*gen*, *nawabari*, *katamuki*, *migikata*, *hidarikata*, *migisita*, *hidarisita*, *miginokori*, *hidarinokori*, *migiasi*, *hidariasi*, *ueetotu*, *sitaetotu*, *ueenototuka*, *sitaenototuka*;

end module

ここで，例えば，convexify : −ueenototuka($xlist, ylist$) とすれば，上述のプロシデュア ueenototuka($xlist, ylist$) が起動されるのである[12].

注意 2.1.2　プロシデュア `ueetotu`，`sitaetotu` などは，`ue`，`sita` もデータに入れて設定しなおすことができる．もちろん，このような精神で最初からプロシデュアの設定を行なうのであれば，中間的なプロシデュア `ueetotu`, `sitaetotu` を，現行の形で用意することは必ずしも適当ではない．同様のことは，`migi`，`hidari` をデータとして扱う場合にも言える．例えば，

totu := **proc**($xlist$, $ylist$, d)
if $d = ue$ **then** *convexify* : $-ueetotu(xlist,\ ylist)$
elif $d = sita$ **then** *convexify* : $-sitaetotu(xlist,\ ylist)$
else RETURN(‘*meaningless direction*‘)
end if
end proc

とできる．この場合の出力は，

totu([1, 2, 3], [−1, 2, 2], *ue*) = ([1, 2, 3], [−1, 2, 2])
totu([1, 2, 3], [−1, 2, 2], *sita*) = ([1, 3], [−1, 2])
totu([1, 2, 3], [−1, 2, 2], *naka*) = *meaningless direction*

となる．

2.2 Maple による流量の近似

一般の保存則の流量 $f(u)$ に対する区間 $u_\ell, u_r]$ における N 次 2 進近似 $f_N(u)$ を，データ u_ℓ, u_r, N, f から構成する手順を数式処理ソフト Maple のモデュール `fcndata` として具体化してみる．

工程 2.2.1

fcndata := **module**() **export** *fvalues*, *xdata*, *initdata*, *kinjidata*, *fcnkinji*; **end module**

`fcndata` の export 変数の説明をしよう．

まず，該当する部分の近似流量を作る．`fvalues` は，x-座標のリスト $xlist = [x_1, \cdots, x_n]$ と流量 f とから関数値のリスト $[f(x_1), \cdots, f(x_n)]$ を作り出す

[12] これは Maple というソフトにおける module の約束事である．

プロシデュアである．すなわち，

fvalues := **proc**(*xlist*, *f*)
local *i*, *n*;
  if *xlist* ≠ sort(*xlist*) **then** RETURN(‘*fcndata* : −*fvalues* = *input_error0*‘) **end if**;
  n := nops(*xlist*) ;
  if nops({seq($xlist_i$, *i* = 1..*n*)}) < *n* **then**
    RETURN(‘*fcndata* : −*fvalues* = *input_error1*‘)
  end if;
  map(*f*, *xlist*)
end proc

ただし，ここではデータ *xlist* の整合性の検査が中心である[13]．

出力例を挙げる[14]．プロシデュア `xdata` は後述する．

例 2.2.1　データを

$$\mathrm{xdata}(-1,\,2,\,3) = [\frac{-1}{8},\,0,\,\frac{1}{8},\,\frac{1}{4}]$$

で与える．まず，流量が抽象的な関数（記号） f のとき `fvalues` の出力は

$$\mathrm{fvalues}(\mathrm{xdata}(-1,\,2,\,3),\,f) = [\mathrm{f}(\frac{-1}{8}),\,\mathrm{f}(0),\,\mathrm{f}(\frac{1}{8}),\,\mathrm{f}(\frac{1}{4})]$$

となる．流量が関数 $x \mapsto x^3$ で与えられる場合は

$$\mathrm{fvalues}(\mathrm{xdata}(-1,\,2,\,3),\,x \to x^3) = [\frac{-1}{512},\,0,\,\frac{1}{512},\,\frac{1}{64}]$$

である．

つぎに，m, n, N から 2 進数 $\dfrac{i}{2^N}$, $i = m, \cdots, n$, を出力するプロシデュアが `xdata` である．

xdata := **proc**(*m*, *n*, *N*)
local *i*;
  if *n* ≤ *m* **then** RETURN(‘*fcndata* : −*xdata* = *input_error0*‘) **end if** ;
  [seq($i * 2^{(-N)}$, *i* = *m*..*n*)]
end proc

この出力を `fvalues` のデータ *xlist* としたい．しかし，`xdata` は，端点を（特定の整数）値ではなく，$\dfrac{m}{2^N}$, $\dfrac{n}{2^N}$ の形に予め想定して，N を動かしたい場合には不都合である．そこで，プロシデュア `initdata` では対 *pair* と N から 2 進数のリストを生成させる．

initdata := **proc**(*pair*, *N*)
local *m*, *n*;
  m := $2^N * pair_1$; *n* := $2^N * pair_2$; xdata(*m*, *n*, *N*)
end proc

[13]つまり，*xlist* の成分が単調増大かどうかを確認してから関数値を求めている．
[14]`fvalues` はモデュール `fcndata` の export 変数だから，本来は *fcndata* : −*fvalues* として計算する．以下の他の計算例でも同様である．

実際は，$2^N * pair_1$，$2^N * pair_2$ が整数であることを確認しなければならない（が，ここでは手を抜いている）．

出力例を挙げる．

例 2.2.2

$$\text{initdata}([\frac{-1}{8}, \frac{1}{4}], 3) = [\frac{-1}{8}, 0, \frac{1}{8}, \frac{1}{4}]$$
$$\text{initdata}([\frac{-1}{8}, \frac{1}{4}], 4) = [\frac{-1}{8}, \frac{-1}{16}, 0, \frac{1}{16}, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}]$$

以上のプロシデュア `fvalues`，`initdata` を組合わせれば，所期の近似流量を得ることができる．すなわち， `kinjidata` は初期値 $pair$，近似の精度 N，流量 f をデータとして，区間 $pair_1 \le x \le pair_2$ の間隔 $\frac{1}{2^N}$ での細分点のリストと対応する流量 f の値のリストとを併せて出力するプロシデュアであり，これを，さらに，module `convexify` の export 変数 `nawabari` で処理して，近似流量を折れ線関数として出力するプロシデュアが `fcnkinji` である．

kinjidata := **proc**($pair$, N, f)
local X, Y;
　X := initdata($pair$, N) ; Y := fvalues(X, f) ; X, Y
end proc

fcnkinji := **proc**($pair$, N, f)
local Z, x;
　Z := kinjidata($pair$, N, f) ; $x \to convexify : -nawabari(Z)(x)$
end proc

`kinjidata` の出力例を示す．

例 2.2.3

$$\text{kinjidata}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3) = ([\frac{-1}{8}, 0, \frac{1}{8}, \frac{1}{4}], [\frac{-1}{512}, 0, \frac{1}{512}, \frac{1}{64}])$$

$$\text{kinjidata}([\frac{-1}{8}, \frac{1}{4}], 4, x \to x^3) =$$
$$([\frac{-1}{8}, \frac{-1}{16}, 0, \frac{1}{16}, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}], [\frac{-1}{512}, \frac{-1}{4096}, 0, \frac{1}{4096}, \frac{1}{512}, \frac{27}{4096}, \frac{1}{64}])$$

`fcnkinji` の出力はグラフを示す（図 9，図 10）．関数 $x \to x^3$ のグラフを破線で重ねる．図 10 では，$fcnkinji([-1/8, 1/4], 3, x \to x^3)(x)$ のグラフも破線で示してある．

モデュール `fcndata` と `convexify` を組合わせれば，$u_- \le u \le u_+$ における f の近似流量 f_N の上下への凸化を計算することができる．ここでは例で

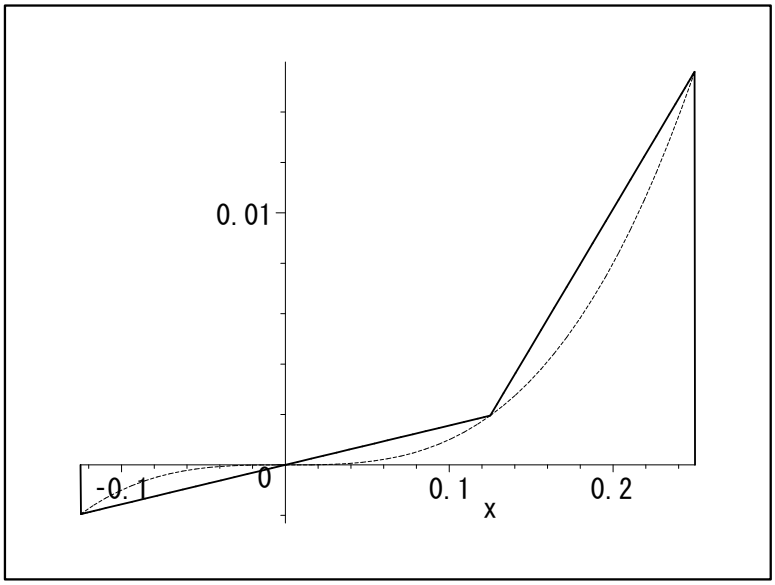

図 9: *fcnkinji*$([-1/8, 1/4], 3, x \to x^3)(x)$ のグラフ

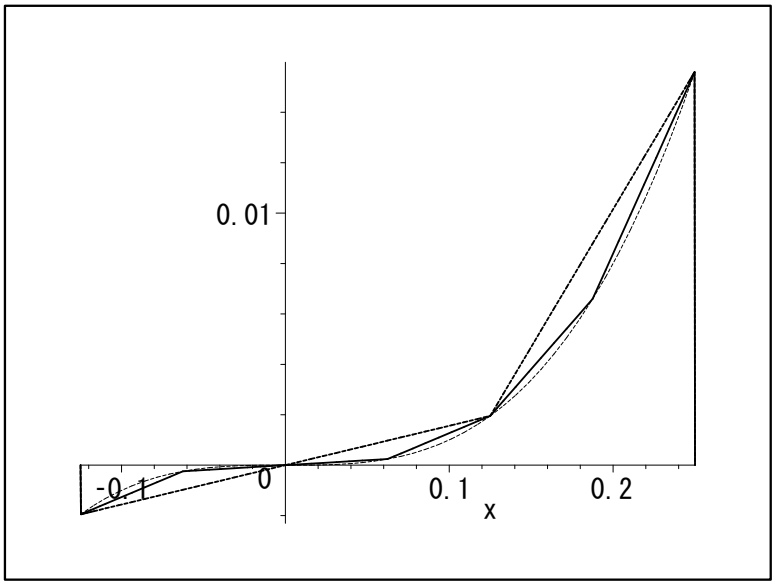

図 10: *fcnkinji*$([-1/8, 1/4], 4, x \to x^3)(x)$ のグラフ

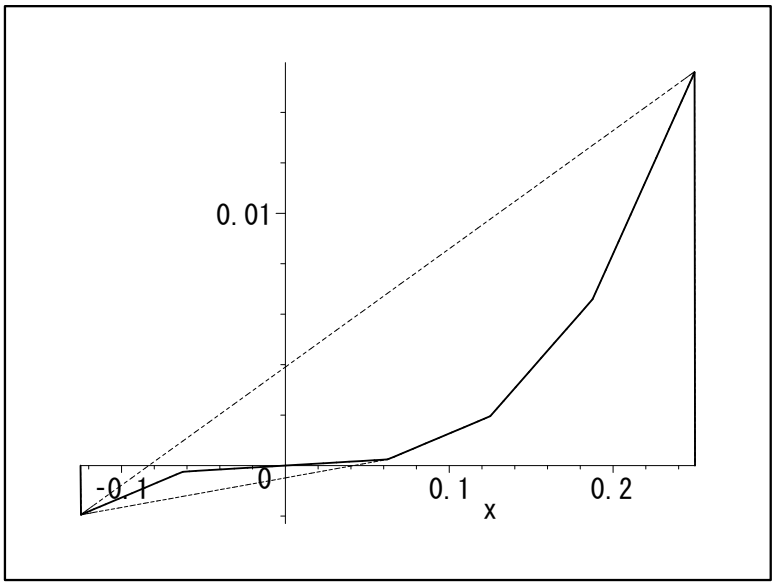

図 11: 近似と凸化のグラフ

示そう．便宜上，例 2.2.3 の kinjidata($[\frac{-1}{8}, \frac{1}{4}]$, 4, $x \to x^3$) の出力を $Fd4$ と表そう．すると，つぎのようになる．

> `'convexify:-ueetotu(Fd4)'=convexify:-ueetotu(Fd4);`

$$\text{ueetotu}(Fd4) = ([\frac{-1}{8}, \frac{1}{4}], [\frac{-1}{512}, \frac{1}{64}])$$

> `'convexify:-sitaetotu(Fd4)'=convexify:-sitaetotu(Fd4);`

$$\text{sitaetotu}(Fd4) = ([\frac{-1}{8}, \frac{1}{16}, \frac{1}{8}, \frac{3}{16}, \frac{1}{4}], [\frac{-1}{512}, \frac{1}{4096}, \frac{1}{512}, \frac{27}{4096}, \frac{1}{64}])$$

他方，対応する上への凸化は convexify : −ueenototuka($Fd4$) となり，下への凸化 convexify : −sitaenototuka($Fd4$) である．例えば，

$$\text{ueenototuka}(Fd4)(x) = \begin{cases} 0 & x < \frac{-1}{8} \\ \frac{1}{256} + \frac{3\,x}{64} & -x < \frac{1}{8} \textbf{ and } x < \frac{1}{4} \\ 0 & \frac{1}{4} < x \end{cases}$$

となる．

これらのグラフを破線で表し，fcndata : −fcnkinji($[\frac{-1}{8}, \frac{1}{4}]$, 4, $x \to x^3$) のグラフと重ねて描けば，図 11 が得られる．

2.3 Maple による近似 Riemann 問題のエントロピー解（一）

流量 $f(u)$ の第 N-近似流量 $f_N(u)$ に対する保存則について，初期関数が

$$g(x) = \begin{cases} u_-, & x < 0 \\ u_+, & x > 0 \end{cases} \qquad \left(u_- = \frac{\ell}{2^N},\ u_+ = \frac{r}{2^N} \right)$$

のときの Riemann 問題のエントロピー解（Riemann 波）を Maple で計算しよう．ただし，ℓ, r は相異なる整数である．このための計算を，いくつかのモデュールにまとめておくと便利である．

工程 2.3.1　まず，各種の波の計算のために，Maple のモデュール `wavedata` を用意する[15]．

wavedata := **module**()
export *sazanami*, *tatunami*, *namisuji*, *namisen*, *namisendata*, *majiwari*, *migiriemann*, *hidaririemann*, *riemann*, *migiriemanndata*, *hidaririemanndata*, *riemanndata*, *riemannnamisen*, *riemannnamisuji*;
end module

モデュール `wavedata` の export 変数（プロシデュア）については，以下で説明をする．

注意 2.3.1　求めるエントロピー解は `wavedata` の export 変数 `riemann` によって，

$$u(t,x) = wavedata : -riemann([u_-, u_+], N, f)\left(\frac{x}{t}\right)$$

として計算される．しかし，このままでは不足があるので，さらに，モデュール `riemann`，`kansho` を用意し，さらに，モデュール `entropy_solution` 内のプロシデュアによって，大域的なエントロピー解の生成も行なおうというのである．

工程 2.3.2　まず，`sazanami` と `tatunami` を説明する．

sazanami := **proc**(*xlist*, *ylist*)
local *i*, *n*, *w*, *c*;
　c := *convexify* : −*katamuki*(*xlist*, *ylist*) ;
　n := nops(*c*) ;
　for *i* **to** *n* **do** $w_i := [xlist_i,\ c_i,\ ylist_i,\ ylist_{i+1}]$ **end do** ;
　seq(w_i, $i = 1..n$)
end proc

[15] 既述のモデュール `action`，`convexify`，`fcndata` は当然利用する．

このプロシデュア sazanami は入力データ $xlist = [x_1, \cdots, x_{n+1}]$, $ylist = [y_1, \cdots, y_{n+1}]$ から[16]，モデュール convexify 中のプロシデュア katamuki によって，勾配の列 $[c_1, \cdots, c_n] = convexify : -katamuki(xlist, ylist)$ を計算した上で，リストの列

$$[x_1, c_1, y_1, y_2],\ \cdots,\ [x_n, c_n, y_n, y_{n+1}]$$

を出力するものである．$[x_i, c_i, y_i, y_{i+1}]$ は波線 $x = x_i + c_i\, t$ と，その左右にそれぞれ値 y_i, y_{i+1} があることを示唆するリストである．sazanami の入力データ $xlist$, $ylist$ に時間変数 t を補ったものから，sazanami の波線の部分に相当する列だけを（関数化して）出力するプロシデュアが tatunami である．

$tatunami := \mathbf{proc}(xlist,\ ylist,\ t)$
local $i,\ n,\ w,\ c$;
　$c := convexify : -katamuki(xlist,\ ylist)$;
　$n := \mathrm{nops}(c)$;
　for i **to** n **do** $w_i := \mathrm{unapply}(xlist_i + c_i * t,\ t)$ **end do**;
　$[\mathrm{seq}(w_i(t),\ i = 1..n)]$
end proc

sazanami と tatunami の出力データの違いは注意に値する．

出力例を見よう．

例 2.3.1　sazanami の出力は 4 成分のリスト（列）である．tatunami の出力は，実は，これらの第 1，第 2 成分のみに関わる．

$\mathrm{sazanami}([1,\ 2,\ 3,\ 4],\ [2,\ 1,\ 3,\ 1]) = ([1,\ -1,\ 2,\ 1],\ [2,\ 2,\ 1,\ 3],\ [3,\ -2,\ 3,\ 1])$

念のために，convexify:-katamuki と比較する．

$$\mathrm{katamuki}([1,\ 2,\ 3,\ 4],\ [2,\ 1,\ 3,\ 1]) = [-1,\ 2,\ -2]$$

すなわち，sazanami の出力列に現れるリストそれぞれの第 2 成分が新たなリストを作っていることが確かめられる．一方，tatunami については，つぎの通り．

$$\mathrm{tatunami}([1,\ 2,\ 3,\ 4],\ [2,\ 1,\ 3,\ 1],\ t) = [1 - t,\ 2 + 2\,t,\ 3 - 2\,t]$$

この出力を画像化したものが図 12 である．

工程 2.3.3　namisuji，namisen，namisendata を説明する．個別に波線を表すためのプロシデュアが namisuji である．入力データは出発点の座標 a と勾配 c を対にした $list = [a, c]$ で，出力は波線 $t \mapsto a + ct$ である．

$namisuji := \mathbf{proc}(list)\ \mathbf{local}\ t;\ \mathrm{unapply}(list_1 + list_2 * t,\ t)\ \mathbf{end\ proc}$

一方，プロシデュア namisen は，sazanami の出力に現れる個々のリストと同じ形の $[a, c, b, bb]$ を入力データとし，その最初の 2 成分のなす対 $[a, c]$ の

[16] プロシデュア中では x_i は $xlist_i$ となっている．y_i も同様.

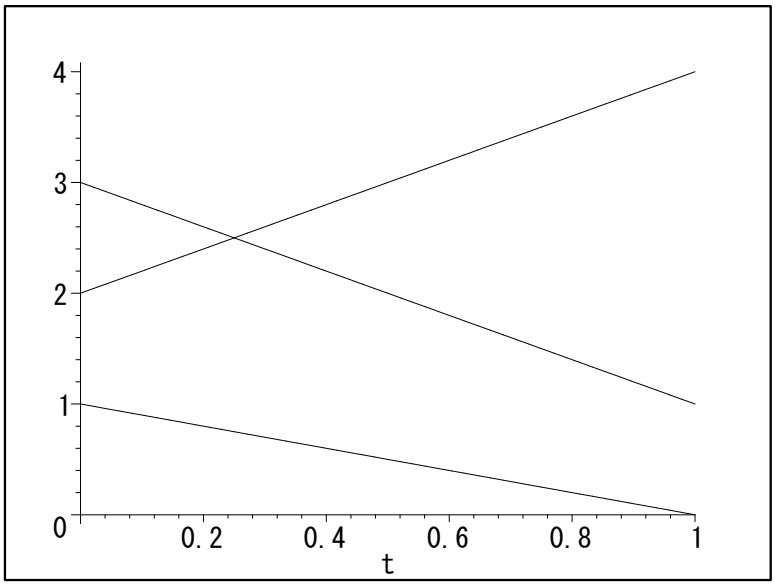

図 12: tatunami([1, 2, 3, 4], [2, 1, 3, 1], t) の画像化

部分に `namisuji` 相当の計算を行なって，$[t \mapsto a + ct, b, bb]$ を出力するものである．

$namisen := \mathbf{proc}(datalist)$
$\mathbf{local}\, t,\ nami;$
$\quad nami := datalist_1 + datalist_2 * t\,;\ [\mathrm{unapply}(nami,\ t),\ datalist_3,\ datalist_4]$
$\mathbf{end\ proc}$

これに対し，入力データ $[t \mapsto a + ct, b, bb]$ から $[a, c, b, bb]$ を出力させるプロシデュアを `namisendata` とする．

$namisendata := \mathbf{proc}(datalist)$
$\mathbf{local}\, t,\ u,\ a,\ c,\ b,\ bb;$
$\quad u := \mathrm{unapply}(datalist_1(t),\ t)\,;$
$\quad a := \mathrm{coeff}(u(t),\ t,\ 0)\,;$
$\quad c := \mathrm{coeff}(u(t),\ t,\ 1)\,;$
$\quad b = datalist_2\,;$
$\quad bb := datalist_3\,;$
$\quad [a,\ c,\ b,\ bb]$
$\mathbf{end\ proc}$

すなわち，`namisen` と `namisendata` は互いに他の逆になっている（裏表をなすと言ってもよい）．

出力例を挙げる．

例 2.3.2　記号についての演算の形で示す．

$$\text{namisuji}([a,\, c]) = (t \to a + c\,t)$$
$$\text{namisen}([a,\, c,\, b,\, bb]) = [t \to a + c\,t,\, b,\, bb]$$
$$\text{namisendata}([t \to a + c\,t,\, b,\, bb]) = [a,\, c,\, b,\, bb]$$

これらの例の入出力データの形に注意してほしい.

工程 2.3.4　`wavedata` の export 変数 `majiwari` は，隣接する二つの波線の交差を管理するプロシデュアである[17]．入力データは `namisen` の出力と同じ形のリスト 2 個，すなわち，$namidata1 = [t \mapsto a_1 + c_1\,t, b_1, b_1']$，$namidata2 = [t \mapsto a_2 + c_2\,t, b_2, b_2']$ である．

majiwari := **proc**(*namidata1*, *namidata2*)
local t, *nami1*, *nami2*, T, a, b, bb;
　nami1 := *namidata1*$_1$;
　nami2 := *namidata2*$_1$;
　if *nami1*(0) = *nami2*(0) **then** RETURN(‘*wavedata* : −*majiwari* = *input_error0*‘)
　elif *nami1*(0) < *nami2*(0) **then**
　　if *namidata1*$_3$ ≠ *namidata2*$_2$ **then**
　　　RETURN(‘*wavedata* : −*majiwari* = *input_error1*‘)
　　elif coeff(*nami1*(t), t) ≤ coeff(*nami2*(t), t) **then** $T := \infty$
　　else
　　　T := solve(*nami1*(t) − *nami2*(t), t);
　　　a := *nami1*(T);
　　　b := *namidata1*$_2$;
　　　bb := *namidata2*$_3$
　　end if
　else
　　if *namidata1*$_2$ ≠ *namidata2*$_3$ **then**
　　　RETURN(‘*wavedata* : −*majiwari* = *input_error2*‘)
　　elif coeff(*nami2*(t), t) ≤ coeff(*nami1*(t), t) **then** $T := \infty$
　　else
　　　T := solve(*nami1*(t) − *nami2*(t), t);
　　　a := *nami1*(T);
　　　b := *namidata2*$_2$;
　　　bb := *namidata1*$_3$
　　end if
　end if;
　$[T,\, a,\, [b,\, bb]]$
end proc

`majiwari` は，波線 $x = a_1 + c_1 t,\ x = a_2 + c_2 t$ が $a_1 < a_2$ （または $a_2 < a_1$）をみたし，かつ交点 $t = T > 0,\, x = a$ をもつときに，$[T, a, b_1, b_2']$（または

[17] Riemann 問題の解を個別に扱う段階では干渉を想定していないので，まだ，`majiwari` は必要にならない.

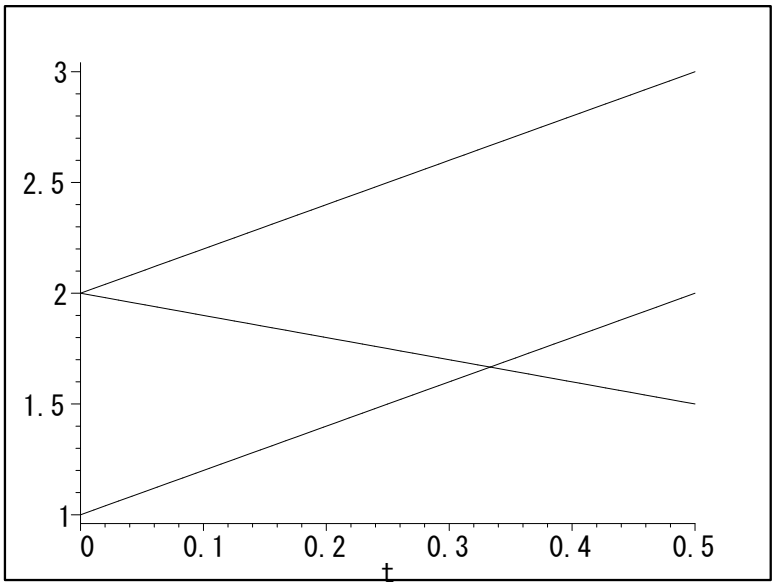

図 13: （下から）f, g, h のグラフ

$[T, a, b_2, b_1']$ を出力する[18].

出力例を示す.

例 2.3.3　波線を 3 本とって,

$$f = (t \to 1 + 2\,t),\ g = (t \to 2 - t),\ h = (t \to 2 + 2\,t)$$

とし，`majiwari` の入力データは

$$namidata1 = [f,\ 1,\ 2],\ namidata2 = [g,\ 2,\ 3],\ namidata3 = [h,\ 2,\ 3]$$

のうちの 2 個をとることにする（図 13 参照）.
以下が `majiwari` の出力結果である.

$$\text{majiwari}([f,\ 1,\ 2],\ [g,\ 2,\ 3]) = [\frac{1}{3},\ \frac{5}{3},\ [1,\ 3]]$$

$$\text{majiwari}([f,\ 1,\ 2],\ [h,\ 2,\ 3]) = wavedata : -majiwari = input\_error1$$

$$\text{majiwari}([g,\ 2,\ 3],\ [f,\ 1,\ 2]) = [\frac{1}{3},\ \frac{5}{3},\ [1,\ 3]]$$

$$\text{majiwari}([g,\ 2,\ 3],\ [h,\ 2,\ 3]) = wavedata : -majiwari = input\_error0$$

[18] $a_1 = a_2$ ならば $wavedata : -majiwari = input\_error0$ を出力し，$a_1 \neq a_2$ であっても $T > 0$ が成り立たないときは $wavedata : -majiwari = input\_error1$ を出力する．なお，二つの波線 $namidata1$，$namidata2$ は $a_1 < a_2$ のときは $b_1' = b_2$，$a_1 > a_2$ のときは $b_1 = b_2'$ が成り立つときに隣接する，あるいは，隣り合っていると言うことにする．`majiwari` は，入力データの波線が隣接しているかどうかも判定している．

いよいよ近似 Riemann 問題のエントロピー解の構成の準備をする．ただし，現段階では，発見的考察めいた中間的なものである[19]．

工程 2.3.5　入力データ *Ulist*，*Flist* に対応する折れ線関数の下への凸化または上への凸化を求め，それらのグラフの勾配と対応する区分点（右左の脚）の座標から階段関数を求める．階段関数の各段を定めるのは凸化グラフの勾配であって，段の高さ（値）が区分点の座標（*Ulist* の部分リスト）である．下への凸化に対応するのが `migiriemann` であり，上への凸化に対応するのが `hidaririemann` である．

$$
\begin{aligned}
&\mathit{migiriemann} := \mathbf{proc}(\mathit{Ulist},\ \mathit{Flist})\\
&\mathbf{local}\ s,\ \mathit{Lu},\ \mathit{Lf},\ \mathit{dir},\ \mathit{ddir},\ n,\ i;\\
&\quad \mathit{Lu} := \mathit{convexify} : -\mathit{sitaetotu}(\mathit{Ulist},\ \mathit{Flist})_1\,;\\
&\quad \mathit{Lf} := \mathit{convexify} : -\mathit{sitaetotu}(\mathit{Ulist},\ \mathit{Flist})_2\,;\\
&\quad \mathit{dir} := \mathit{convexify} : -\mathit{katamuki}(\mathit{Lu},\ \mathit{Lf})\,;\\
&\quad n := \mathrm{nops}(\mathit{dir})\,;\\
&\quad \mathit{ddir} := [-\infty,\ \mathrm{seq}(\mathit{dir}_i,\ i = 1..n),\ \infty]\,;\\
&\quad s \to \mathit{action} : -\mathit{kaidan}(\mathit{ddir},\ \mathit{Lu})(s)\\
&\mathbf{end\ proc}
\end{aligned}
$$

`hidaririemann` では上への凸化の勾配の列が単調減少であることとプロシデュア `action:-kaidan` のデータへの要請とを整合させるために，中間的な出力データの成分の並べ替えを必要とする．このため，`migiriemann` との対称性が崩れているようにみえるところがある．

$$
\begin{aligned}
&\mathit{hidaririemann} := \mathbf{proc}(\mathit{Ulist},\ \mathit{Flist})\\
&\mathbf{local}\ s,\ \mathit{Lu},\ \mathit{Lf},\ \mathit{dir},\ \mathit{ddir},\ n,\ \mathit{dddir},\ i,\ \mathit{Luu};\\
&\quad \mathit{Lu} := \mathit{convexify} : -\mathit{ueetotu}(\mathit{Ulist},\ \mathit{Flist})_1\,;\\
&\quad \mathit{Lf} := \mathit{convexify} : -\mathit{ueetotu}(\mathit{Ulist},\ \mathit{Flist})_2\,;\\
&\quad \mathit{dir} := \mathit{convexify} : -\mathit{katamuki}(\mathit{Lu},\ \mathit{Lf})\,;\\
&\quad \mathit{ddir} := \mathrm{sort}(\mathit{dir})\,;\\
&\quad n := \mathrm{nops}(\mathit{ddir})\,;\\
&\quad \mathit{dddir} := [-\infty,\ \mathrm{seq}(\mathit{ddir}_i,\ i = 1..n),\ \infty]\,;\\
&\quad \mathit{Luu} := \mathit{ListTools}_{\mathit{Reverse}}(\mathit{Lu})\,;\\
&\quad s \to \mathit{action} : -\mathit{kaidan}(\mathit{dddir},\ \mathit{Luu})(s)\\
&\mathbf{end\ proc}
\end{aligned}
$$

`migiriemann`，`hidaririemann` のいずれも中間的なものである．

`migiriemann`，`hidaririemann` の出力例を画像として示す（図 14，図 15）．

本来の Riemann 問題との関係では *Ulist* の両端の成分が，初期関数の値に相当し，*Ulist* の他の成分および *Flist* は，流量 $f(u)$ がらモデュール `fcndata`

[19] 我々は，もっとも経済的なアルゴリズムを追求しているわけではなく，数学解析に内在するものを探っていることに注意されたい．

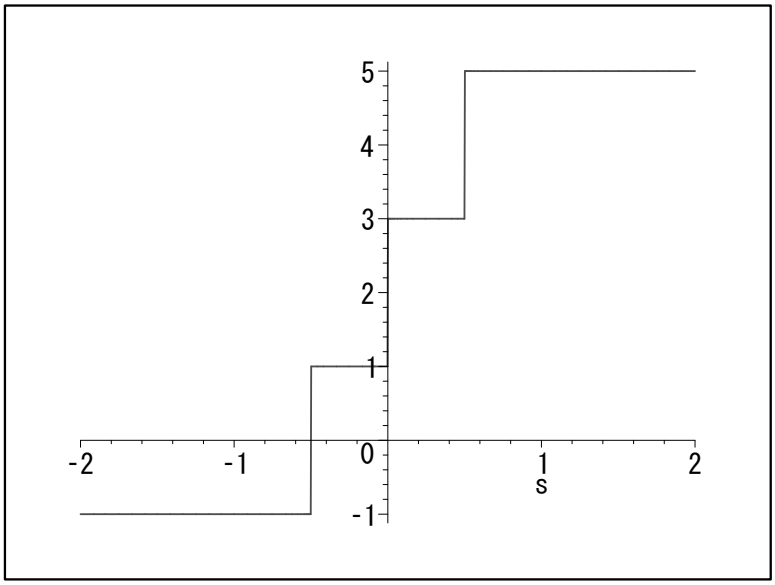

図 14: $migiriemann([-1,0,1,3,5],[2,3,1,1,2])(s)$ のグラフ

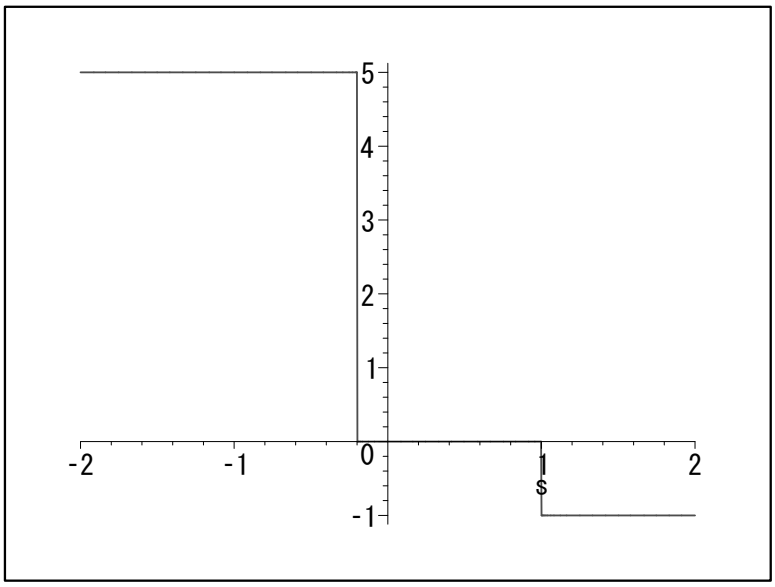

図 15: $hidaririemann([-1,0,1,3,5],[2,3,1,1,2])(s)$ のグラフ

によって補間されるべきものである．次に示す `riemann` はこの点を考慮に入れている．

工程 2.3.6　プロシデュア `riemann` を説明する．入力データは初期値の対 $Pair$ と N, f であり，これから $Ulist$, $Flist$ を `fcndata:-kinjidata` によって生成する．初期値の設定に応じて，`migiriemann` あるいは `hidaririemann` を適用する．なお，`riemann` それ自体は擬似 Riemann 問題のエントロピー解の一歩手前である．

$riemann := \mathbf{proc}(Pair,\, N,\, f)$
local $pair$, $pseudoriemann$, s;
　if $Pair_1 = Pair_2$ **then** RETURN(‘$wavedata : -riemann = input\_error0$‘)
　elif $Pair_2 < Pair_1$ **then**
　　$pair := [Pair_2,\, Pair_1]$;
　　$pseudoriemann := s \to \mathrm{hidaririemann}(fcndata : -kinjidata(pair,\, N,\, f))(s)$
　else
　　$pair := Pair$;
　　$pseudoriemann := s \to \mathrm{migiriemann}(fcndata : -kinjidata(pair,\, N,\, f))(s)$
　end if;
　unapply($pseudoriemann(s)$, s)
end proc

出力例を示す．流量は $f(u) = u^3$，$N = 3$ とする．$u_- = -\frac{1}{8}$, $u_+ = \frac{1}{4}$ のときは次の通り．

例 2.3.4　　まず，$u_- = -\frac{1}{8}$, $u_+ = \frac{1}{4}$ のときを見る．

$$\mathrm{riemann}([\frac{-1}{8},\, \frac{1}{4}],\, 3,\, x \to x^3)(s) = \left(\begin{cases} 0 & s < -\infty \\ \dfrac{-1}{8} & -\infty - s \le 0 \textbf{ and } s < \dfrac{1}{64} \\ 0 & \dfrac{1}{64} \le s \end{cases}\right)$$

$$+ \left(\begin{cases} 0 & s < \dfrac{1}{64} \\ \dfrac{1}{8} & \dfrac{1}{64} - s \le 0 \textbf{ and } s < \dfrac{7}{64} \\ 0 & \dfrac{7}{64} \le s \end{cases}\right) + \left(\begin{cases} 0 & s < \dfrac{7}{64} \\ \dfrac{1}{4} & \dfrac{7}{64} - s \le 0 \textbf{ and } s - \infty < 0 \\ 0 & \infty \le s \end{cases}\right)$$

他方，$u_- = \frac{1}{4}$, $u_+ = -\frac{1}{8}$ のときは次のようになる．

$$\mathrm{riemann}([\frac{1}{4},\, \frac{-1}{8}],\, 3,\, x \to x^3)(s) =$$

$$\left(\begin{cases} 0 & s < -\infty \\ \dfrac{1}{4} & -\infty - s \le 0 \textbf{ and } s < \dfrac{3}{64} \\ 0 & \dfrac{3}{64} \le s \end{cases}\right) + \left(\begin{cases} 0 & s < \dfrac{3}{64} \\ \dfrac{-1}{8} & \dfrac{3}{64} - s \le 0 \textbf{ and } s - \infty < 0 \\ 0 & \infty \le s \end{cases}\right)$$

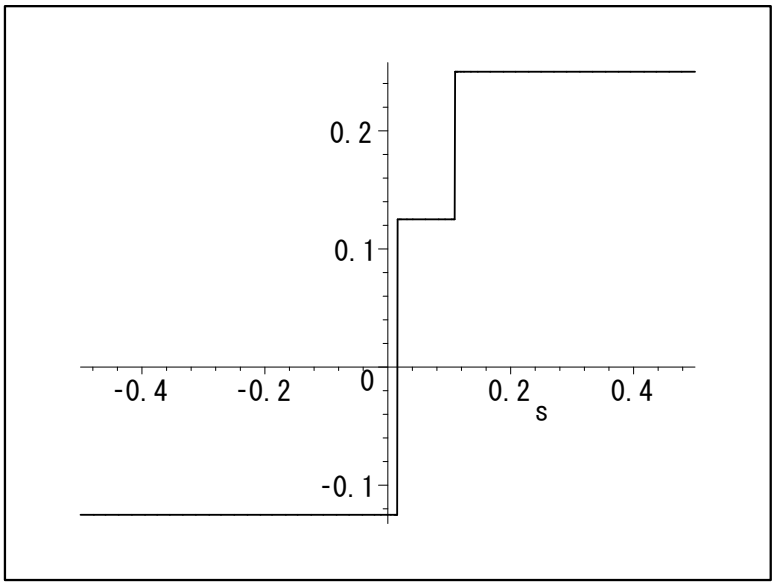

図 16: riemann($[\frac{1}{4}, \frac{-1}{8}]$, 3, $x \to x^3$)(s) のグラフ

画像化したものは，図 16，図 17 に掲げる．

さて，初期条件として $t = 0$ のとき $x < 0$ で値 $-\frac{1}{8}$ をとり，$x > 0$ では $\frac{1}{4}$ となるような満たす擬似 Riemann 問題のエントロピー解 $U(t, x)$ は

$$U(t,x) = \text{riemann}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3)\left(\frac{x}{t}\right)$$

として求められる．同様に考えると，初期値が $x < 1$ で値 $\frac{1}{4}$，$x > 1$ では $-\frac{1}{8}$ のとき，擬似 Riemann 問題のエントロピー解 $W(t, x)$ は

$$W(t,x) = \text{riemann}([\frac{1}{4}, \frac{-1}{8}], 3, x \to x^3)\left(\frac{x-1}{t}\right)$$

となる．これらのグラフを 3 次元画像として表すことができる（図 18，図 19）．

工程 2.3.7 プロシデュア `migiriemann`, `hidaririemann` および `riemann` の出力は加工しにくい．そこで，入力データ $Ulist$, $Flist$ から `sazanami` の出力と同じ形の出力を果たしたい．このために，さらに波線の発生点に相当するデータ a を補う．

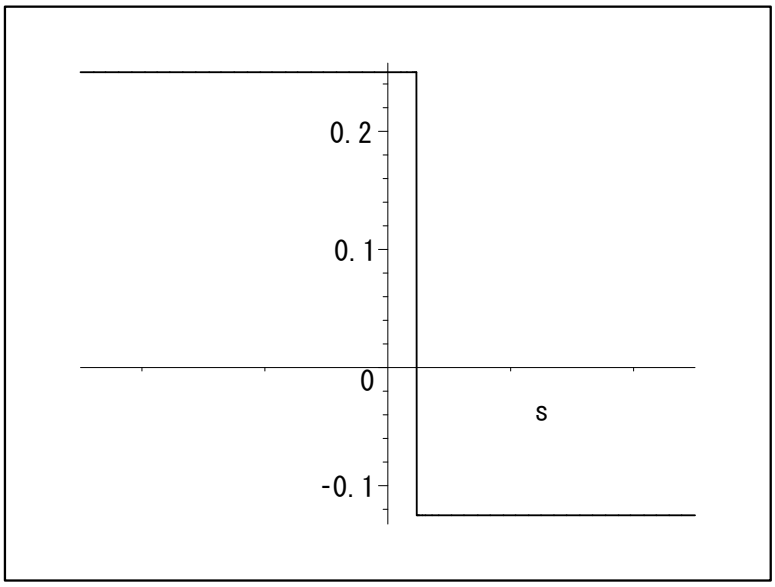

図 17: riemann($[\frac{-1}{8}, \frac{1}{4}]$, 3, $x \to x^3$)(s) のグラフ

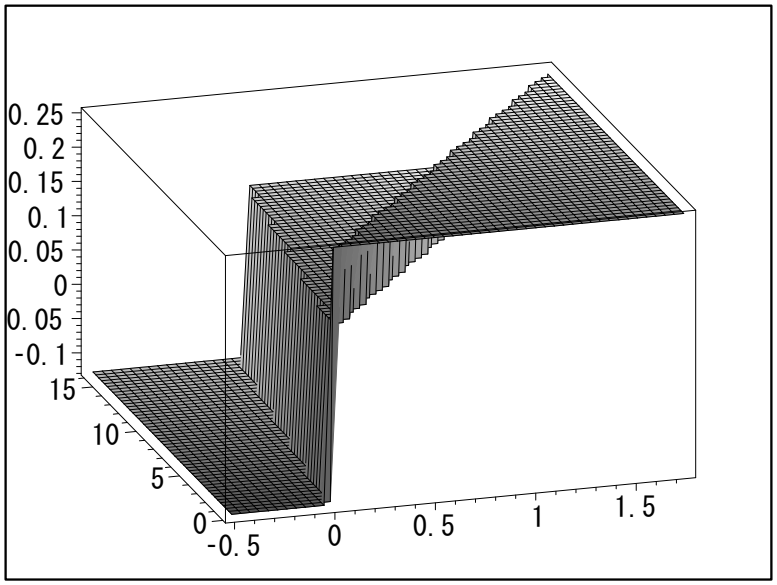

図 18: $U(t, x)$ の立体画像

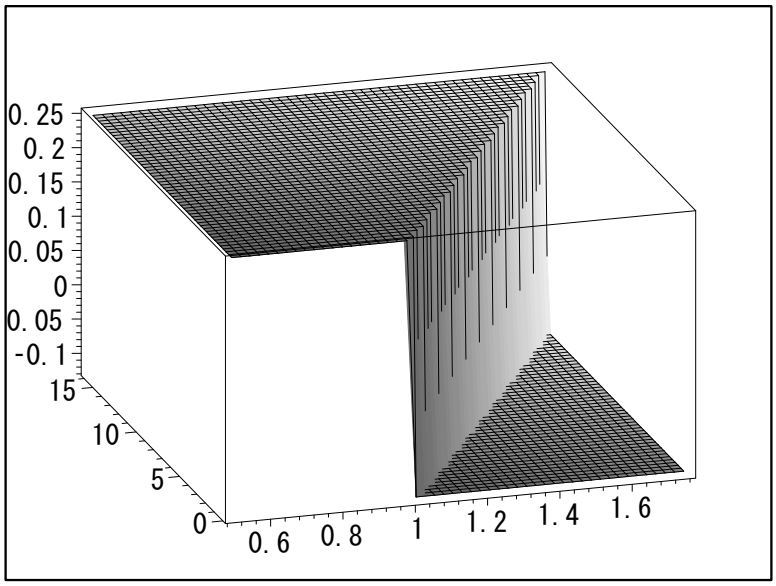

図 19: $W(t,x)$ の立体画像

$migiriemanndata := \mathbf{proc}(Ulist,\ Flist,\ a)$
$\mathbf{local}\, s,\ Lu,\ Lf,\ dir,\ namidata,\ n,\ i;$
$Lu := convexify : -sitaetotu(Ulist,\ Flist)_1\,;$
$Lf := convexify : -sitaetotu(Ulist,\ Flist)_2\,;$
$dir := convexify : -katamuki(Lu,\ Lf)\,;$
$n := \mathrm{nops}(dir)\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, namidata_i := [a,\ dir_i,\ Lu_i,\ Lu_{i+1}]\, \mathbf{end\ do}\,;$
$\mathrm{seq}(namidata_i,\ i = 1..n)$
$\mathbf{end\ proc}$

$hidaririemanndata := \mathbf{proc}(Ulist,\ Flist,\ a)$
$\mathbf{local}\, s,\ Lu,\ Lf,\ dir,\ ddir,\ n,\ namidata,\ i,\ Luu;$
$Lu := convexify : -ueetotu(Ulist,\ Flist)_1\,;$
$Lf := convexify : -ueetotu(Ulist,\ Flist)_2\,;$
$dir := convexify : -katamuki(Lu,\ Lf)\,;$
$ddir := \mathrm{sort}(dir)\,;$
$n := \mathrm{nops}(ddir)\,;$
$Luu := ListTools_{Reverse}(Lu)\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, namidata_i := [a,\ ddir_i,\ Luu_i,\ Luu_{i+1}]\, \mathbf{end\ do}\,;$
$\mathrm{seq}(namidata_i,\ i = 1..n)$
$\mathbf{end\ proc}$

`riemanndata` は，プロシデュア `riemann` が初期値の対 $Pair$ と流量 $f(u)$，近似の精度 N に基づく補間によって， $Ulist$，$Flist$ を生成したように，

`fcndata:-kinjidata` を内蔵する.

riemanndata := **proc**(*Pair*, *N*, *f*, *a*)
local *pair*, *pseudoriemanndata*, *s*;
  if $Pair_1 = Pair_2$ **then** RETURN(‘*check data pair*‘)
  elif $Pair_2 < Pair_1$ **then**
    pair := [$Pair_2$, $Pair_1$] ;
    pseudoriemanndata := hidaririemanndata(*fcndata* : −*kinjidata*(*pair*, *N*, *f*), *a*)
  else
    pair := *Pair* ;
    pseudoriemanndata := migiriemanndata(*fcndata* : −*kinjidata*(*pair*, *N*, *f*), *a*)
  end if;
  pseudoriemanndata
end proc

出力例を示す.

例 2.3.5

$$\text{migiriemanndata}([-1, 0, 1, 3, 5], [2, 3, 1, 1, 2], a) = ([a, \frac{-1}{2}, -1, 1], [a, 0, 1, 3], [a, \frac{1}{2}, 3, 5])$$

$$\text{hidaririemanndata}([-1, 0, 1, 3, 5], [2, 3, 1, 1, 2], a) = ([a, \frac{-1}{5}, 5, 0], [a, 1, 0, -1])$$

$$\text{riemanndata}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3, a) = ([a, \frac{1}{64}, \frac{-1}{8}, \frac{1}{8}], [a, \frac{7}{64}, \frac{1}{8}, \frac{1}{4}])$$

$$\text{riemanndata}([\frac{1}{4}, \frac{-1}{8}], 3, x \to x^3, a) = [a, \frac{3}{64}, \frac{1}{4}, \frac{-1}{8}]$$

工程 2.3.8　`riemannnamisuji` および `riemannnamisen` は `riemanndata` に `namisuji` および `namisen` を合成したものである.

riemannnamisen := **proc**(*Pair*, *N*, *f*, *a*)
local *n*, *i*, *RD*, *RN*;
  RD := [riemanndata(*Pair*, *N*, *f*, *a*)] ;
  n := nops(*RD*) ;
  for *i* **to** *n* **do** RN_i := namisen(RD_i) **end do** ;
  seq(RN_i, *i* = 1..*n*)
end proc

```
riemannnamisuji := proc(Pair, N, f, a)
local n, i, RD, RN;
    RD := [riemanndata(Pair, N, f, a)];
    n := nops(RD);
    for i to n do RN_i := namisuji(RD_i) end do;
    seq(eval(RN_i), i = 1..n)
end proc
```

例を挙げる.

例 2.3.6

$$\text{riemannnamisuji}([\frac{1}{4}, \frac{-1}{8}], 3, x \to x^3, 1) = (t \to 1 + \frac{3}{64}\,t)$$

$$\text{riemannnamisuji}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3, 0) = (t \to \frac{1}{64}\,t, t \to \frac{7}{64}\,t)$$

$$\text{riemannnamisen}([\frac{1}{4}, \frac{-1}{8}], 3, x \to x^3, 1) = [t \to 1 + \frac{3}{64}\,t, \frac{1}{4}, \frac{-1}{8}]$$

$$\text{riemannnamisen}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3, 0) = ([t \to \frac{1}{64}\,t, \frac{-1}{8}, \frac{1}{8}], [t \to \frac{7}{64}\,t, \frac{1}{8}, \frac{1}{4}])$$

なお，今計算した $\text{riemannnamisen}([\frac{1}{4}, \frac{-1}{8}], 3, x \to x^3, 1)$ および

$$\text{riemannnamisen}([\frac{-1}{8}, \frac{1}{4}], 3, x \to x^3, 0)_2 \quad \text{（第 2 成分）}$$

をそれぞれ A, B とおくと

$$\text{majiwari}(A, B) = [16, \frac{7}{4}, [\frac{1}{8}, \frac{-1}{8}]]$$

となる（図 20）.

最後に，プロシデュア `majiwari` の利用に言及しておこう．A, B の定める波線は $t = 16, x = \frac{7}{4}$ で交わり，A の右（図 20 では上方）に $-\frac{1}{8}$，B の左（図 20 では下方）に $\frac{1}{8}$ が配置されていることを示す.

このことは，初期条件

$$u(0,x) = \begin{cases} \frac{1}{8}, & x < 0 \\ \frac{1}{4}, & 0 < x < 1 \\ -\frac{1}{8}, & 1 < x \end{cases} \tag{8}$$

として， $f(u) = u^3$, $N = 3$, のときの擬似 Riemann 問題のエントロピー解が，$0 < t < 16$ である限り，上で構成した $U(t,x)$ と $W(t,x)$ の貼りあわせによって得られることを意味する．すなわち，このエントロピー解を $Z(t,x)$ とすれば,

$$Z := (t, x) \to \text{piecewise}(x < \frac{7}{64}\,t, \text{U}(t, x), \frac{7}{64}\,t < x \textbf{ and } x < 1 + \frac{3}{64}\,t, \frac{1}{4}, 1 + \frac{3}{64}\,t < x, \text{W}(t, x))$$

となる．図 21 に，$Z(t,x)$ の 3 次元画像化を示す（$0 < t < 16$）.

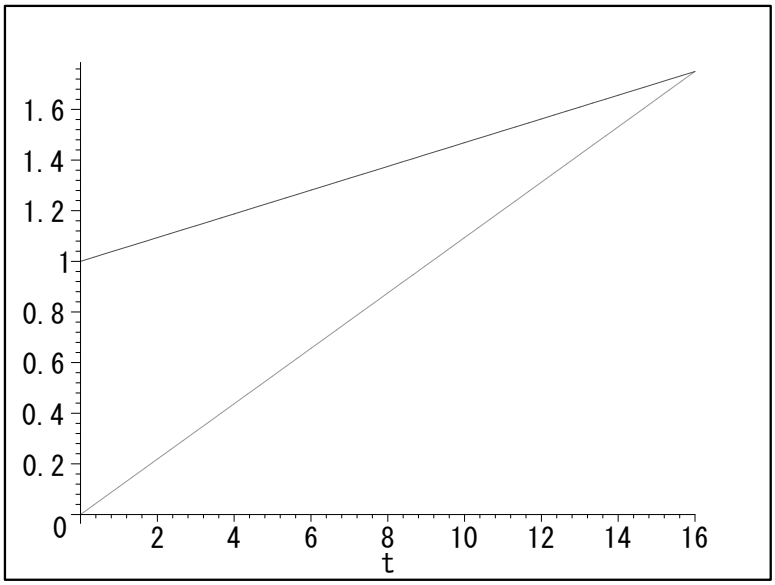

図 20: 波線 A, B の模式図（上 A，下 B）

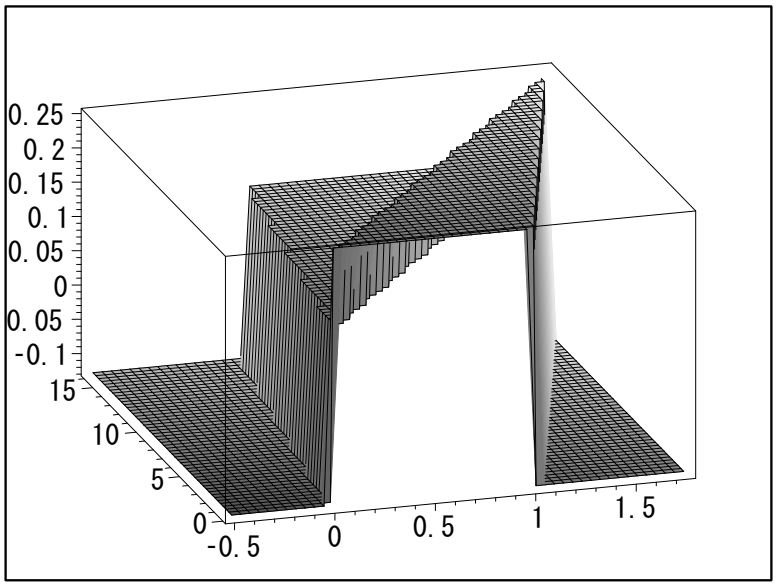

図 21: エントロピー解 $Z(t,x)$

それでは，$t > 16$ のときはどう考えるか．Riemann 波の干渉の処理については後に組織的にモデュール `kansho` および `entropy_solution` として考察するが，その準備も兼ねて手動で極めて見やすい今の場合を説明する．`majiwari` の出力にしたがって，

$$\text{riemann}([\frac{1}{8}, \frac{-1}{8}], 3, x \to x^3)(s) =$$

$$\left(\begin{cases} 0 & s < -\infty \\ \frac{1}{8} & -\infty - s \le 0 \textbf{ and } s < \frac{1}{64} \\ 0 & \frac{1}{64} \le s \end{cases}\right) + \left(\begin{cases} 0 & s < \frac{1}{64} \\ \frac{-1}{8} & \frac{1}{64} - s \le 0 \textbf{ and } s - \infty < 0 \\ 0 & \infty \le s \end{cases}\right)$$

を計算する．これを $uw(s)$ とおけば，$t > 16$ において

$$UW(t,x) = uw\left(\frac{x - \frac{7}{4}}{t - 16}\right)$$

は有効である．ただし，$U(t,x)$ のもう一本の波線（ $[0, \frac{1}{64}, -\frac{1}{8}, \frac{1}{8}]$ に相当するもの）は B の定める波線とは交わらないから，エントロピー解として $t > 16$ で有効なものは，次に掲げる $ZZ(t,x)$ である：

$$ZZ := (t,\, x) \to \text{piecewise}(x < \frac{7}{64}\, t,\, \text{U}(t,\, x),\, \frac{7}{64}\, t < x,\, \text{UW}(t,\, x))$$

したがって，初期条件 (8) に対する $t > 0$ で有効なエントロピー解 $SOL(t,x)$ は

$$SOL := (t,\, x) \to \text{piecewise}(t < 16,\, \text{Z}(t,\, x),\, 16 < t,\, \text{ZZ}(t,\, x))$$

として与えられるはずである．

次に掲げるのは，$SOL(t,x)$ の 3 次元画像である．図 22 は全体像の想像のためであるが，波線の交点の近傍で画像が（数値計算の関係で）乱れている．もともとは有理数計算をしているものを画像化するために浮動小数点計算にしているのである．図 23 は，交点の近くに制限して画像化したものである．ここでは有理数演算が先行していることの反映として，交点の近くでの解の様子がよくわかるであろう．

2.4 Maple による近似 Riemann 問題のエントロピー解（二）

モデュール `riemann` は，モデュール `wavedata` を整理し，近似 Riemann 問題のエントロピー解の生成に必要なプロシデュアをまとめ直したものである．

工程 2.4.1　モデュール `riemann` は次の通り：

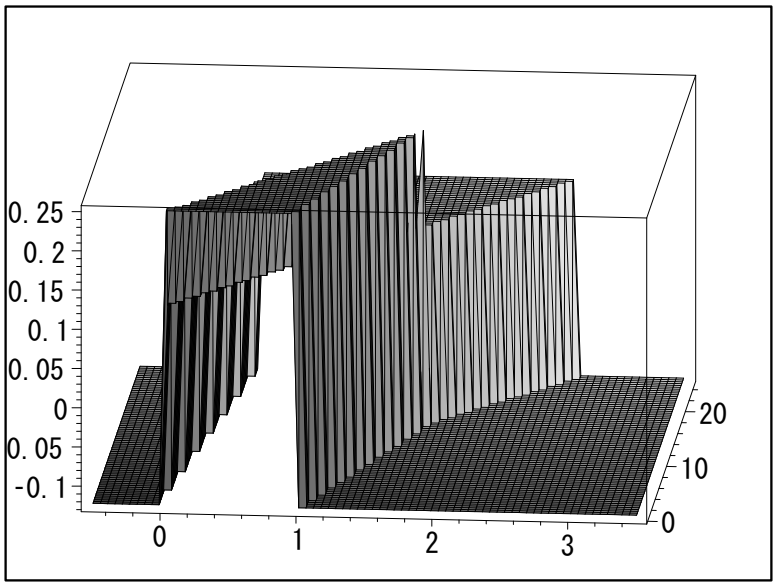

図 22: $SOL(t,x)$ のグラフ（1）

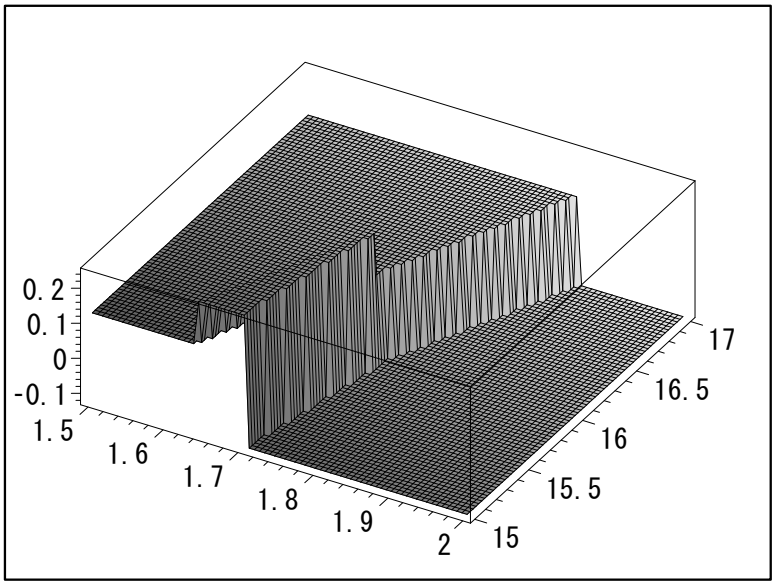

図 23: $SOL(t,x)$ のグラフ（2）

module()
export*tandata*, *migidata*, *hidaridata*, *tann*, *zendata*, *zendataretu*,
zenvdata, *namisuji*, *zennamisuji*, *namisujiplot*, *zennamisujiplot*, *zen*,
migihajidata, *hidarihajidata*,
migihajinamisuji, *hidarihajinamisuji*, *migihajinamisen*,
hidarihajinamisen, *tannamisen*, *zennamisen*;

end module

export されるプロシデュアは，基本的に `wavedata` にあるものも多いが，その場合でも名称は必ずしも一致しない．個々の内容は以降で説明する．

工程 2.4.2　プロシデュア `tandata` が基本である．`tandata` は，プロシデュア `hidaridata`，`migidata` を内包する．これらは，それぞれモデュール `wavedata` の `riemanndata`，`hidaririemanndata`，`migiriemanndata` に他ならないことは，両者を比較して見ればわかる．

tandata := **proc**(*a*, *Pair*, *N*, *f*)
local *pair*, *RD*;
  if $Pair_1 = Pair_2$ **then** RETURN(‘*riemann* : −*tandata* = *input_error0*‘)
  elif $Pair_2 < Pair_1$ **then**
    pair := $[Pair_2, Pair_1]$;
    RD := hidaridata(*a*, *fcndata* : −*kinjidata*(*pair*, *N*, *f*))
  else *pair* := *Pair* ; *RD* := migidata(*a*, *fcndata* : −*kinjidata*(*pair*, *N*, *f*))
  end if;
  RD
end proc

migidata := **proc**(*a*, *vlist*, *flist*)
local *s*, *Lu*, *Lf*, *dir*, *namidata*, *n*, *i*;
  Lu := *convexify* : −*sitaetotu*(*vlist*, *flist*)$_1$;
  Lf := *convexify* : −*sitaetotu*(*vlist*, *flist*)$_2$;
  dir := *convexify* : −*katamuki*(*Lu*, *Lf*) ;
  n := nops(*dir*) ;
  for *i* **to** *n* **do** $namidata_i := [a, dir_i, Lu_i, Lu_{i+1}]$ **end do** ;
  seq($namidata_i$, $i = 1..n$)
end proc

$hidaridata := \mathbf{proc}(a,\ vlist,\ flist)$
$\mathbf{local}\ Lu,\ Lf,\ dir,\ ddir,\ n,\ Luu,\ i,\ namidata;$
$Lu := convexify : -ueetotu(vlist,\ flist)_1\,;$
$Lf := convexify : -ueetotu(vlist,\ flist)_2\,;$
$dir := convexify : -katamuki(Lu,\ Lf)\,;$
$ddir := \mathrm{sort}(dir)\,;$
$n := \mathrm{nops}(ddir)\,;$
$Luu := ListTools_{Reverse}(Lu)\,;$
$\mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ namidata_i := [a,\ ddir_i,\ Luu_i,\ Luu_{i+1}]\ \mathbf{end\ do}\,;$
$\mathrm{seq}(namidata_i,\ i = 1..n)$
$\mathbf{end\ proc}$

なお，モデュール `wavedata` と `riemann` の関係はいずれ整理する予定ではある．

出力例を `tandata` について示す．

例 2.4.1

$$\mathrm{tandata}(-1,\ [0,\ -1],\ 3,\ x \to x^3) = ([-1,\ \frac{1}{64},\ 0,\ \frac{-1}{8}],\ [-1,\ \frac{7}{64},\ \frac{-1}{8},\ \frac{-1}{4}],$$
$$[-1,\ \frac{19}{64},\ \frac{-1}{4},\ \frac{-3}{8}],\ [-1,\ \frac{37}{64},\ \frac{-3}{8},\ \frac{-1}{2}],\ [-1,\ \frac{61}{64},\ \frac{-1}{2},\ \frac{-5}{8}],$$
$$[-1,\ \frac{91}{64},\ \frac{-5}{8},\ \frac{-3}{4}],\ [-1,\ \frac{127}{64},\ \frac{-3}{4},\ \frac{-7}{8}],\ [-1,\ \frac{169}{64},\ \frac{-7}{8},\ -1])$$

$$\mathrm{tandata}(0,\ [-1,\ 1],\ 3,\ x \to x^3) = ([0,\ \frac{3}{4},\ -1,\ \frac{1}{2}],$$
$$[0,\ \frac{61}{64},\ \frac{1}{2},\ \frac{5}{8}],\ [0,\ \frac{91}{64},\ \frac{5}{8},\ \frac{3}{4}],\ [0,\ \frac{127}{64},\ \frac{3}{4},\ \frac{7}{8}],\ [0,\ \frac{169}{64},\ \frac{7}{8},\ 1])$$

$$\mathrm{tandata}(1,\ [1,\ 0],\ 3,\ x \to x^3) = [1,\ 1,\ 1,\ 0]$$

ちなみに，近似の精度の変更に対し，希薄波が敏感であり，ショックが安定であることも `tandata` の出力から判断できる．

例 2.4.2

$$\mathrm{tandata}(-1,\ [0,\ -1],\ 4,\ x \to x^3) = ([-1,\ \frac{1}{256},\ 0,\ \frac{-1}{16}],\ [-1,\ \frac{7}{256},\ \frac{-1}{16},\ \frac{-1}{8}],$$
$$[-1,\ \frac{19}{256},\ \frac{-1}{8},\ \frac{-3}{16}],\ [-1,\ \frac{37}{256},\ \frac{-3}{16},\ \frac{-1}{4}],\ [-1,\ \frac{61}{256},\ \frac{-1}{4},\ \frac{-5}{16}],$$
$$[-1,\ \frac{91}{256},\ \frac{-5}{16},\ \frac{-3}{8}],\ [-1,\ \frac{127}{256},\ \frac{-3}{8},\ \frac{-7}{16}],\ [-1,\ \frac{169}{256},\ \frac{-7}{16},\ \frac{-1}{2}],$$
$$[-1,\ \frac{217}{256},\ \frac{-1}{2},\ \frac{-9}{16}],\ [-1,\ \frac{271}{256},\ \frac{-9}{16},\ \frac{-5}{8}],\ [-1,\ \frac{331}{256},\ \frac{-5}{8},\ \frac{-11}{16}],$$
$$[-1,\ \frac{397}{256},\ \frac{-11}{16},\ \frac{-3}{4}],\ [-1,\ \frac{469}{256},\ \frac{-3}{4},\ \frac{-13}{16}],\ [-1,\ \frac{547}{256},\ \frac{-13}{16},\ \frac{-7}{8}],$$
$$[-1,\ \frac{631}{256},\ \frac{-7}{8},\ \frac{-15}{16}],\ [-1,\ \frac{721}{256},\ \frac{-15}{16},\ -1])$$

$$\mathrm{tandata}(1,\,[1,\,0],\,5,\,x \to x^3) = [1,\,1,\,1,\,0]$$

次に掲げるプロシデュアは $x = a$ から出発する近似 Riemann 波を生成するものである．$t > 0$ であれば，

$$wavedata : -riemann\,([u_-, u_+], N, f)\,\left(\frac{(x-a)}{t}\right) =$$
$$= riemann : -tann(a, [u_-, u_+], N, f)$$

となるはずであるが，左辺の $t = 0$ における困難の回避を図ったものである．

工程 2.4.3　単純 Riemann 波の生成のプロシデュア `tann` を示す[20]：

$tann := \mathbf{proc}(a,\,Pair,\,N,\,f)$
$\mathbf{local}\,t,\,x,\,RD,\,n,\,i,\,j,\,RDD,\,TN;$
　$RD := [\mathrm{tandata}(a,\,Pair,\,N,\,f)]\,;$
　$n := \mathrm{nops}(RD)\,;$
　$RDD_0 := [a,\,-\infty,\,null,\,null]\,;$
　$RDD_{n+1} := [a,\,\infty,\,RD_{n4},\,null]\,;$
　$\mathbf{for}\,i\,\mathbf{to}\,n\,\mathbf{do}\,RDD_i := RD_i\,\mathbf{end\ do}\,;$
　$\mathbf{for}\,i\,\mathbf{to}\,n+1\,\mathbf{do}\ TN_i := \mathrm{piecewise}(x < RDD_{i-1_2} * t + a,\,0,$
　　$RDD_{i-1_2} * t + a \le x\ \mathbf{and}\ x < RDD_{i2} * t + a,\,RDD_{i3},$
　　$RDD_{i2} * t + a < x,\,0)$
　$\mathbf{end\ do};$
　$\mathrm{unapply}(\mathrm{sum}(TN_j,\,j = 1..n+1),\,t,\,x)$
$\mathbf{end\ proc}$

出力例を挙げる．

例 2.4.3　例 2.4.2 の 2 例目，3 例目に相当する．まず，2 例目への対応：

$$\mathrm{tann}(0,\,[-1,\,1],\,3,\,x \to x^3) = ((t,\,x) \to$$
$$\mathrm{piecewise}(x < -\infty\,t,\,0,\,-\infty\,t - x \le 0\ \mathbf{and}\ x - \frac{3}{4}\,t < 0,\,-1,\,\frac{3}{4}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{3}{4}\,t,\,0,\,-x + \frac{3}{4}\,t \le 0\ \mathbf{and}\ x - \frac{61}{64}\,t < 0,\,\frac{1}{2},\,\frac{61}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{61}{64}\,t,\,0,\,-x + \frac{61}{64}\,t \le 0\ \mathbf{and}\ x - \frac{91}{64}\,t < 0,\,\frac{5}{8},\,\frac{91}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{91}{64}\,t,\,0,\,-x + \frac{91}{64}\,t \le 0\ \mathbf{and}\ x - \frac{127}{64}\,t < 0,\,\frac{3}{4},\,\frac{127}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{127}{64}\,t,\,0,\,-x + \frac{127}{64}\,t \le 0\ \mathbf{and}\ x - \frac{169}{64}\,t < 0,\,\frac{7}{8},\,\frac{169}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{169}{64}\,t,\,0,\,-x + \frac{169}{64}\,t \le 0\ \mathbf{and}\ x - \infty\,t < 0,\,1,\,\infty\,t < x,\,0))$$

[20]正接関数 `tan` との競合を回避した．

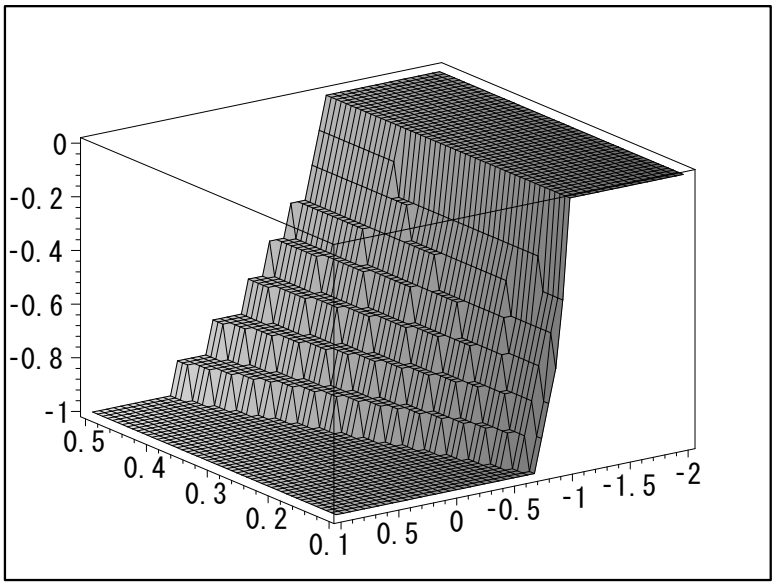

図 24: riemann : $-\mathrm{tann}(-1, [0, -1], 3, x- > x^3)(t, x)$ の画像

ここで，当然ながら $\pm\infty\, t$ は $\pm\infty$ である．
3 例目への対応：

$\mathrm{tann}(1,\ [1,\ 0],\ 3,\ x \to x^3) = ((t,\ x) \to$
$\mathrm{piecewise}(x < -\infty\, t + 1,\ 0,\ -\infty\, t + 1 - x \leq 0$ **and** $x - t < 1,\ 1,\ 1 + t < x,\ 0)$
$+ \mathrm{piecewise}(x < 1 + t,\ 0,\ 1 - x + t \leq 0$ **and** $x - \infty\, t < 1,\ 0,\ \infty\, t + 1 < x,\ 0))$

右辺第 1 項と第 3 項は，実は，定数関数 0 であるが，プロシデュアの構成上，見掛け上の出力がある．

$riemann : -tann(a, Pair, N, f)$ の出力は画像化した方が印象的である．例 2.4.2 の 1 例目に対応する場合を示す[21] （図 24）．

初期関数が階段関数ならば，各分点から発する Riemann 波，すなわち，分点の両側の関数値によって定まる Riemann 問題のエントロピー解を組合わせることにより，少なくとも，これらの波の干渉が生じるまでは，対応する初期値問題のエントロピー解を構成できるはずである．事実，この目的のために，プロシデュア `zen` を作ることができる．しかし，その前にも手順がいくつかある．

[21]出力用コード：

```
> plot3d([t,x,riemann:-tann(-1,[0,-1],3,x- > x^3)(t,x)],
> t=0.1..1/2,x=-2..1,
> axes=boxed,grid=[50,50],orientation=[145,65]);
```

工程 2.4.4　有理点の列 $xlist$，$vlist$ に対し，リスト $xlist$ の成分数だけリスト $tandata(xlist_i, [vlist_i, vlist_{i+1}], N, f)$ を出力するプロシデュア `zendata` と，これらのリストの成分である 4 成分リストを全体列として出力するプロシデュア `zendataretu` を与える．

$zendata := \mathbf{proc}(xlist,\ vlist,\ N,\ f)$
local $i,\ n,\ RD$;
　if $xlist \neq \mathrm{sort}(xlist)$ **then** RETURN(‘$riemann : -zendata = input\_error0$‘) **end if**;
　$n := \mathrm{nops}(xlist)$;
　if $n+1 \neq \mathrm{nops}(vlist)$ **then** RETURN(‘$riemann : -zendata = input\_error1$‘) **end if**;
　for i **to** n **do** $RD_i := [\mathrm{tandata}(xlist_i,\ [vlist_i,\ vlist_{i+1}],\ N,\ f)]$ **end do**;
　$\mathrm{seq}(RD_i,\ i = 1..n)$
end proc

$zendataretu := \mathbf{proc}(xlist,\ vlist,\ N,\ f,\ k)$
local $RZD,\ ZD,\ n,\ m,\ i,\ j$;
　$RZD := [\mathrm{zendata}(xlist,\ vlist,\ N,\ f)]$;
　$n := \mathrm{nops}(RZD)$;
　for i **to** n **do** $m_i := \mathrm{nops}(RZD_i)$ **end do**;
　$ZD := \mathrm{seq}(\mathrm{seq}(RZD_{ij},\ j = 1..m_i),\ i = 1..n)$
end proc

出力例を挙げる．なお，

注意 2.4.1　以下で例として考察しているのは，初期関数が

$$g_0(x) = \begin{cases} 0, & x < -1 \text{ または } x \geq 1 \\ -1, & -1 \leq x < 0 \\ 1, & 0 \leq x < 1 \end{cases}$$

の場合（図 25）の保存則

$$\frac{\partial}{\partial t}\, u(t,x) + \frac{\partial}{\partial x}\, u(t,x)^3 = 0$$

に対する 2^{-3} 刻みの近似保存則である．

例 2.4.4　第 1 例は成分がそれぞれ 8 リスト，5 リスト，1 リストの 3 個のリストからなり，第 2 例は 14 個の 4 成分リストの列である．

$\mathrm{zendata}([-1,\ 0,\ 1],\ [0,\ -1,\ 1,\ 0],\ 3,\ x \to x^3)$
$= ([[-1,\ \frac{1}{64},\ 0,\ \frac{-1}{8}],\ [-1,\ \frac{7}{64},\ \frac{-1}{8},\ \frac{-1}{4}],\ [-1,\ \frac{19}{64},\ \frac{-1}{4},\ \frac{-3}{8}],\ [-1,\ \frac{37}{64},\ \frac{-3}{8},\ \frac{-1}{2}],$
$[-1,\ \frac{61}{64},\ \frac{-1}{2},\ \frac{-5}{8}],\ [-1,\ \frac{91}{64},\ \frac{-5}{8},\ \frac{-3}{4}],\ [-1,\ \frac{127}{64},\ \frac{-3}{4},\ \frac{-7}{8}],\ [-1,\ \frac{169}{64},\ \frac{-7}{8},\ -1]],$
$[[0,\ \frac{3}{4},\ -1,\ \frac{1}{2}],\ [0,\ \frac{61}{64},\ \frac{1}{2},\ \frac{5}{8}],\ [0,\ \frac{91}{64},\ \frac{5}{8},\ \frac{3}{4}],\ [0,\ \frac{127}{64},\ \frac{3}{4},\ \frac{7}{8}],\ [0,\ \frac{169}{64},\ \frac{7}{8},\ 1]],$
$[[1,\ 1,\ 1,\ 0]])$

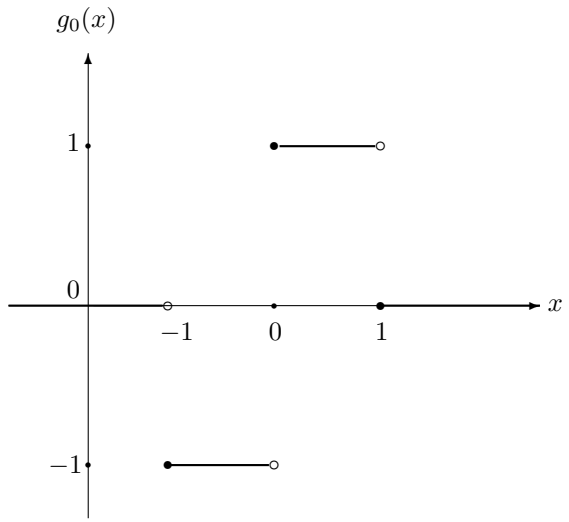

図 25: 初期関数 $g_0(x)$

zendataretu([−1, 0, 1], [0, −1, 1, 0], 3, $x \to x^3$)
$= ([-1, \frac{1}{64}, 0, \frac{-1}{8}], [-1, \frac{7}{64}, \frac{-1}{8}, \frac{-1}{4}], [-1, \frac{19}{64}, \frac{-1}{4}, \frac{-3}{8}],$
$[-1, \frac{37}{64}, \frac{-3}{8}, \frac{-1}{2}], [-1, \frac{61}{64}, \frac{-1}{2}, \frac{-5}{8}], [-1, \frac{91}{64}, \frac{-5}{8}, \frac{-3}{4}],$
$[-1, \frac{127}{64}, \frac{-3}{4}, \frac{-7}{8}], [-1, \frac{169}{64}, \frac{-7}{8}, -1], [0, \frac{3}{4}, -1, \frac{1}{2}], [0, \frac{61}{64}, \frac{1}{2}, \frac{5}{8}],$
$[0, \frac{91}{64}, \frac{5}{8}, \frac{3}{4}], [0, \frac{127}{64}, \frac{3}{4}, \frac{7}{8}], [0, \frac{169}{64}, \frac{7}{8}, 1], [1, 1, 1, 0])$

`zendataretu` の応用として，与えられた初期関数に対応する凸化の過程で定められる近似流量の分点の全体を列として抜き出すプロシデュア `zenvdata` が得られる．

工程 2.4.5　`zendataretu` の出力のうち，第 1 項となるリストからのみ第 3 成分を取り出したものを，残りのリストからは第 4 成分を取り出して作り上げられた列と並置して得られたリストが `zenvdata` の出力結果になる．

$zenvdata := \mathbf{proc}(xlist,\ vlist,\ N,\ f)$
$\mathbf{local}\, n,\ i,\ ZDR;$
$ZDR := \mathrm{zendataretu}(xlist,\ vlist,\ N,\ f)\,;$
$n := \mathrm{nops}([ZDR])\,;$
$[ZDR_{13},\ \mathrm{seq}(ZDR_{i4},\ i = 1..n)]$
$\mathbf{end\ proc}$

`zenvdata` の出力は，初期関数から定められる（干渉前の）エントロピー解の値の列でもある．

出力例を挙げる．例 2.4.4 と比較されたい．

例 2.4.5

$$\mathrm{zenvdata}([-1,\ 0,\ 1],\ [0,\ -1,\ 1,\ 0],\ 3,\ x \to x^3)$$
$$= [0,\ \frac{-1}{8},\ \frac{-1}{4},\ \frac{-3}{8},\ \frac{-1}{2},\ \frac{-5}{8},\ \frac{-3}{4},\ \frac{-7}{8},\ -1,\ \frac{1}{2},\ \frac{5}{8},\ \frac{3}{4},\ \frac{7}{8},\ 1,\ 0]$$

干渉前の複数の Riemann 問題の解の組み合わせは，区分線によって定数を区切る形で行なわれる．個々の Riemann 問題の解毎に `wavedata:-namisuji` によって区分線は計算できた．これをモデュール `riemann` にプロシデュア `namisuji` として取り入れておく．求めるべき区分線の全体は，初期関数の分点に `namisuji` を繰り返し適用したものを集めればよく，これがプロシデュア `zennamisuji` である．

工程 2.4.6 `namisuji`, `zennamisuji` は次の通り：

$namisuji := \mathbf{proc}(a,\ Pair,\ N,\ f)$
$\mathbf{local}\, n,\ i,\ RD,\ RN;$
$RD := [\mathrm{tandata}(a,\ Pair,\ N,\ f)]\,;$
$n := \mathrm{nops}(RD)\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, RN_i := wavedata : -namisuji(RD_i)\, \mathbf{end\ do}\,;$
$\mathrm{seq}(\mathrm{eval}(RN_i),\ i = 1..n)$
$\mathbf{end\ proc}$

$zennamisuji := \mathbf{proc}(xlist,\ vlist,\ N,\ f)$
$\mathbf{local}\, n,\ i,\ NS,\ t;$
$\mathbf{if}\, xlist \neq \mathrm{sort}(xlist)\, \mathbf{then}\, \mathrm{RETURN}(\text{`}riemann : -namisuji = input\_error0\text{`})\, \mathbf{end\ if}\,;$
$n := \mathrm{nops}(xlist)\,;$
$\mathbf{if}\, n + 1 \neq \mathrm{nops}(vlist)\, \mathbf{then}\, \mathrm{RETURN}(\text{`}riemann : -namisuji = input\_erro1\text{`})\, \mathbf{end\ if}\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, NS_i := \mathrm{namisuji}(xlist_i,\ [vlist_i,\ vlist_{i+1}],\ N,\ f)\, \mathbf{end\ do}\,;$
$\mathrm{seq}(\mathrm{eval}(NS_i),\ i = 1..n)$
$\mathbf{end\ proc}$

計算例を示す．

例 2.4.6　まず，namisuji の場合を示し，ついで，それらがまとめられた形になる zennamisuji を見る．

$$\text{namisuji}(-1,\,[0,\,-1],\,3,\,x\to x^3)=(t\to -1+\frac{1}{64}\,t,\,t\to -1+\frac{7}{64}\,t,\,t\to -1+\frac{19}{64}\,t,$$
$$t\to -1+\frac{37}{64}\,t,\,t\to -1+\frac{61}{64}\,t,\,t\to -1+\frac{91}{64}\,t,\,t\to -1+\frac{127}{64}\,t,\,t\to -1+\frac{169}{64}\,t)$$
$$\text{namisuji}(0,\,[-1,\,1],\,3,\,x\to x^3)=(t\to \frac{3}{4}\,t,\,t\to \frac{61}{64}\,t,\,t\to \frac{91}{64}\,t,\,t\to \frac{127}{64}\,t,\,t\to \frac{169}{64}\,t)$$
$$\text{namisuji}(1,\,[1,\,0],\,3,\,x\to x^3)=(t\to 1+t)$$

これらが zennamisuji では一括される．

$$\text{zennamisuji}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x\to x^3)=(t\to -1+\frac{1}{64}\,t,\,t\to -1+\frac{7}{64}\,t,$$
$$t\to -1+\frac{19}{64}\,t,\,t\to -1+\frac{37}{64}\,t,\,t\to -1+\frac{61}{64}\,t,\,t\to -1+\frac{91}{64}\,t,\,t\to -1+\frac{127}{64}\,t,$$
$$t\to -1+\frac{169}{64}\,t,\,t\to \frac{3}{4}\,t,\,t\to \frac{61}{64}\,t,\,t\to \frac{91}{64}\,t,\,t\to \frac{127}{64}\,t,\,t\to \frac{169}{64}\,t,\,t\to 1+t)$$

出力例は図 26 を見よ[22]．

予告した階段関数を初期値とする（干渉までの短い時間だけ有効の）エントロピー解を生成するプロシデュア zen は，zenvdata，zennamisuji があれば定義できる．しかし，その前に，namisuji，zennamisuji を画像化するプロシデュア namisujiplot，zennamisujiplot を示そう．

工程 2.4.7　namisujiplot は，単一の近似 Riemann 波の波線を時間について区間 $Tpair_1 \le t \le Tpair_2$ に制限して描くものである．$Tpair_1 \ge Tpair_2$ の事態はエラーとしている．一方，zennamisujiplot は，namisujiplot を複数の近似 Riemann 波に対して組合わせたものである．ただし，干渉による Riemann 波の変形は考慮していない．

namisujiplot := **proc**(a, *Pair*, N, f, *Tpair*)
local n, i, t, *RN*, *RNP*;
　if $Tpair_2 \le Tpair_1$ **then** RETURN(‘*riemann* : −*namisujiplot* = *input_error0*‘)
　end if;
　RN := [namisuji(a, *Pair*, N, f)] ;
　n := nops(RN) ;
　for i **to** n **do** RNP_i := plot($RN_i(t)$, $t = Tpair_1..Tpair_2$, *color* = *black*,
　　xtickmarks = 3, *ytickmarks* = 3)
　end do;
　$plots_{display}$(seq(RNP_i, $i = 1..n$))
end proc

[22] $0 < t < \frac{64}{121}$ におけるグラフである．右端の値は，干渉時刻（後述．例 2.5.1）である．

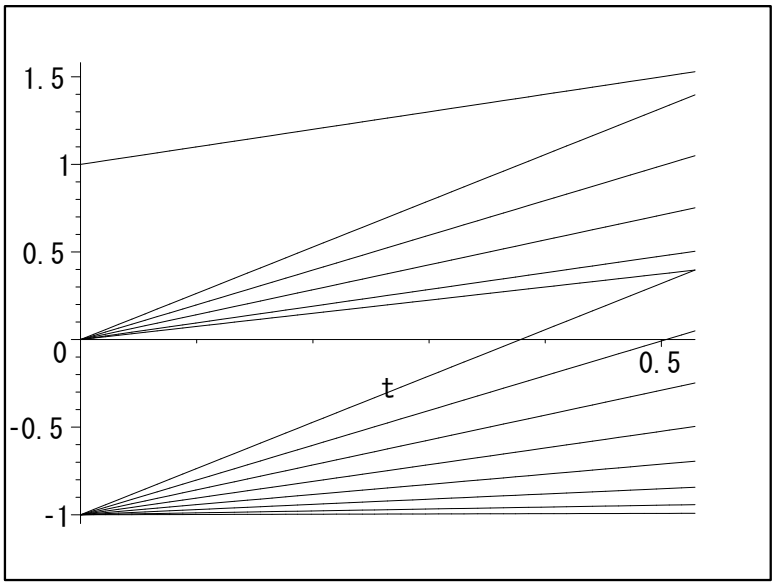

図 26: zennamisujiplot([-1,0,1],[0,-1,1,0],3,$x \to x^3$,[0,$\frac{64}{121}$]) のグラフ

zennamisujiplot := **proc**(*xlist*, *vlist*, *N*, *f*, *Tpair*)
local *ZR*, *t*, *n*, *i*, *d*;
　ZR := zennamisuji(*xlist*, *vlist*, *N*, *f*) ;
　n := nops([*ZR*]) ;
　for *i* **to** *n* **do** d_i :=
　　plot($ZR_i(t)$, *t* = $Tpair_1$..$Tpair_2$, *color* = *black*, *xtickmarks* = 2, *thickness* = 2)
　end do;
　$plots_{display}$(seq(d_i, *i* = 1..*n*))
end proc

出力例は `zennamisujiplot` の場合だけにする（図 26).

さて，いよいよプロシデュア `zen` を示す.

工程 2.4.8

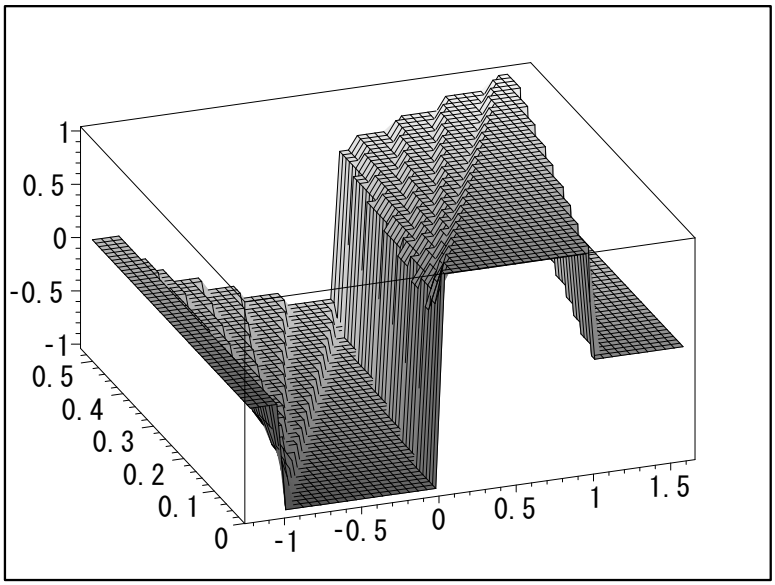

図 27: riemann : $-\mathrm{zen}([-1,0,1],[0,-1,1,0],3,x \to x^3)$

$zen :=$ **proc**($xlist$, $vlist$, N, f)
local t, x, i, j, n, ZS, ZV, TN;
  $ZS :=$ [zennamisuji($xlist$, $vlist$, N, f)] ;
  $ZV :=$ zenvdata($xlist$, $vlist$, N, f) ;
  $n :=$ nops(ZS) ;
  for i **to** $n-1$ **do** $TN_i :=$ piecewise($x < ZS_i(t)$, 0, $ZS_i(t) < x$ **and** $x < ZS_{i+1}(t)$,
    ZV_{i+1}, $ZS_{i+1}(t) < x$, 0)
  end do;
  unapply(sum(TN_j, $j = 1..n-1$), t, x)
end proc

例 2.4.7　例として riemann : $-\mathrm{zen}([-1,0,1],[0,-1,1,0],3,x->x^3)$ を挙げる．余り見やすくはないので，合せて，画像化する（図 27）.

$$\mathrm{zen}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) = ((t,\,x) \to$$
$$\mathrm{piecewise}(x < -1 + \frac{1}{64}\,t,\,0,\,\frac{1}{64}\,t - x < 1 \textbf{ and } x - \frac{7}{64}\,t < -1,\,\frac{-1}{8},\,-1 + \frac{7}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{7}{64}\,t,\,0,\,-x + \frac{7}{64}\,t < 1 \textbf{ and } x - \frac{19}{64}\,t < -1,\,\frac{-1}{4},\,-1 + \frac{19}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{19}{64}\,t,\,0,\,-x + \frac{19}{64}\,t < 1 \textbf{ and } x - \frac{37}{64}\,t < -1,\,\frac{-3}{8},\,-1 + \frac{37}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{37}{64}\,t,\,0,\,-x + \frac{37}{64}\,t < 1 \textbf{ and } x - \frac{61}{64}\,t < -1,\,\frac{-1}{2},\,-1 + \frac{61}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{61}{64}\,t,\,0,\,-x + \frac{61}{64}\,t < 1 \textbf{ and } x - \frac{91}{64}\,t < -1,\,\frac{-5}{8},\,-1 + \frac{91}{64}\,t < x,\,0)$$
$$+$$
$$\mathrm{piecewise}(x < -1 + \frac{91}{64}\,t,\,0,\,-x + \frac{91}{64}\,t < 1 \textbf{ and } x - \frac{127}{64}\,t < -1,\,\frac{-3}{4},\,-1 + \frac{127}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{127}{64}\,t,\,0,\,-x + \frac{127}{64}\,t < 1 \textbf{ and } x - \frac{169}{64}\,t < -1,\,\frac{-7}{8},$$
$$-1 + \frac{169}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < -1 + \frac{169}{64}\,t,\,0,\,-x + \frac{169}{64}\,t < 1 \textbf{ and } x - \frac{3}{4}\,t < 0,\,-1,\,\frac{3}{4}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{3}{4}\,t,\,0,\,-x + \frac{3}{4}\,t < 0 \textbf{ and } x - \frac{61}{64}\,t < 0,\,\frac{1}{2},\,\frac{61}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{61}{64}\,t,\,0,\,-x + \frac{61}{64}\,t < 0 \textbf{ and } x - \frac{91}{64}\,t < 0,\,\frac{5}{8},\,\frac{91}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{91}{64}\,t,\,0,\,-x + \frac{91}{64}\,t < 0 \textbf{ and } x - \frac{127}{64}\,t < 0,\,\frac{3}{4},\,\frac{127}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{127}{64}\,t,\,0,\,-x + \frac{127}{64}\,t < 0 \textbf{ and } x - \frac{169}{64}\,t < 0,\,\frac{7}{8},\,\frac{169}{64}\,t < x,\,0)$$
$$+ \mathrm{piecewise}(x < \frac{169}{64}\,t,\,0,\,-x + \frac{169}{64}\,t < 0 \textbf{ and } x - t < 1,\,1,\,1 + t < x,\,0))$$

さて，こうして作った近似 Riemann 波に関する干渉状況の特定などのためには，細かい観察に適したプロシデュアが必要である．まず，`tandata` の両端の区分線を抜き出すプロシデュアを用意する．

工程 2.4.9　プロシデュア `migihajidata`, `hidarihajidata` は，それぞれ，`tandata` の出力の最終リストと最初のリストを与える．

```
migihajidata := proc(a, Pair, N, f)
local n;
  n := nops([tandata(a, Pair, N, f)]) ; [tandata(a, Pair, N, f)][n]
end proc
hidarihajidata := proc(a, Pair, N, f) [tandata(a, Pair, N, f)][1] end proc
```

なお，`tandata` の出力が 1 個のときは，`hidarihajidata`, `migihajidata` の出力と一致する．

migihajidata，hidarihajidata は，基本的には，隣接する 2 個の Riemann 波の干渉の処理のために用いられる．つまり，migihajidata は左方の Riemann 波の最速の波線を与え，hidarihajidata は右方の Riemann 波の最遅の波線を与えるので， wavedata:-majiwari によって両者の干渉が記述されることが期待される（ただし，wavedata:-majiwari が要求するデータ型に変換するためには後述の段階が必要になる）．

例 2.4.8　次の出力例と例 2.4.4 における textttzendata の出力例と比べてみよ．

$$\text{migihajidata}(-1,\, [0,\, -1],\, 3,\, x \to x^3) = [-1,\, \frac{169}{64},\, \frac{-7}{8},\, -1]$$

$$\text{hidarihajidata}(0,\, [-1,\, 1],\, 3,\, x \to x^3) = [0,\, \frac{3}{4},\, -1,\, \frac{1}{2}]$$

$$\text{migihajidata}(1,\, [1,\, 0],\, 3,\, x \to x^3) = [1,\, 1,\, 1,\, 0]$$

$$\text{hidarihajidata}(1,\, [1,\, 0],\, 3,\, x \to x^3) = [1,\, 1,\, 1,\, 0]$$

最後の 2 例は，対応する tandata の出力と一致する．

データ型を wavedata:-majiwari に適合するものに変換する．

工程 2.4.10　migihajidata, hidarihajidata に wavedata:-namisen を施してプロシデュア migihajinamisen，hidarihajinamisen を得る．

migihajinamisen :=
proc(a, *Pair*, N, f) *wavedata* : $-$*namisen*(migihajidata(a, *Pair*, N, f)) **end proc**

hidarihajinamisen :=
proc(a, *Pair*, N, f) *wavedata* : $-$*namisen*(hidarihajidata(a, *Pair*, N, f)) **end proc**

出力例を示す．

例 2.4.9

$$\text{migihajinamisen}(-1,\, [0,\, -1],\, 3,\, x \to x^3) = [t \to -1 + \frac{169}{64}\, t,\, \frac{-7}{8},\, -1]$$

$$\text{hidarihajinamisen}(0,\, [-1,\, 1],\, 3,\, x \to x^3) = [t \to \frac{3}{4}\, t,\, -1,\, \frac{1}{2}]$$

これらを wavedata:-majiwari と絡めた結果も示しておく．

$$\text{majiwari}([t \to -1 + \frac{169}{64}\, t,\, \frac{-7}{8},\, -1],\, [t \to \frac{3}{4}\, t,\, -1,\, \frac{1}{2}]) = [\frac{64}{121},\, \frac{48}{121},\, [\frac{-7}{8},\, \frac{1}{2}]]$$

なお，wavedata:-riemannnamisen と同じものをモデュール riemann では tannamisen として取り込んである．

工程 2.4.11　tannamisen および複合化 zennamisen を示す．

tannamisen := **proc**(a, *vpair*, N, f) *wavedata* : $-$*riemannnamisen*(*vpair*, N, f, a) **end proc**

zennamisen := **proc**(*xlist*, *vlist*, *N*, *f*)
local *n*, *i*, *NS*, *t*;
if *xlist* ≠ sort(*xlist*) **then** RETURN(‘*riemann* : −*zennamisen* = *input_error0*‘) **end if**;
n := nops(*xlist*) ;
if *n* + 1 ≠ nops(*vlist*) **then** RETURN(‘*riemann* : −*zennamisen* = *input_error1*‘)
end if;
for *i* **to** *n* **do** NS_i := tannamisen($xlist_i$, [$vlist_i$, $vlist_{i+1}$], *N*, *f*) **end do** ;
seq(eval(NS_i), *i* = 1..*n*)
end proc

出力例をみる.

$$\text{tannamisen}(-1,\,[0,\,-1],\,3,\,x\to x^3) = ([t\to -1+\frac{1}{64}\,t,\,0,\,\frac{-1}{8}],\,[t\to -1+\frac{7}{64}\,t,\,\frac{-1}{8},\,\frac{-1}{4}],$$
$$[t\to -1+\frac{19}{64}\,t,\,\frac{-1}{4},\,\frac{-3}{8}],\,[t\to -1+\frac{37}{64}\,t,\,\frac{-3}{8},\,\frac{-1}{2}],\,[t\to -1+\frac{61}{64}\,t,\,\frac{-1}{2},\,\frac{-5}{8}],$$
$$[t\to -1+\frac{91}{64}\,t,\,\frac{-5}{8},\,\frac{-3}{4}],\,[t\to -1+\frac{127}{64}\,t,\,\frac{-3}{4},\,\frac{-7}{8}],\,[t\to -1+\frac{169}{64}\,t,\,\frac{-7}{8},\,-1])$$

$$\text{zennamisen}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x\to x^3) = ([t\to -1+\frac{1}{64}\,t,\,0,\,\frac{-1}{8}],$$
$$[t\to -1+\frac{7}{64}\,t,\,\frac{-1}{8},\,\frac{-1}{4}],\,[t\to -1+\frac{19}{64}\,t,\,\frac{-1}{4},\,\frac{-3}{8}],\,[t\to -1+\frac{37}{64}\,t,\,\frac{-3}{8},\,\frac{-1}{2}],$$
$$[t\to -1+\frac{61}{64}\,t,\,\frac{-1}{2},\,\frac{-5}{8}],\,[t\to -1+\frac{91}{64}\,t,\,\frac{-5}{8},\,\frac{-3}{4}],\,[t\to -1+\frac{127}{64}\,t,\,\frac{-3}{4},\,\frac{-7}{8}],$$
$$[t\to -1+\frac{169}{64}\,t,\,\frac{-7}{8},\,-1],\,[t\to \frac{3}{4}\,t,\,-1,\,\frac{1}{2}],\,[t\to \frac{61}{64}\,t,\,\frac{1}{2},\,\frac{5}{8}],\,[t\to \frac{91}{64}\,t,\,\frac{5}{8},\,\frac{3}{4}],$$
$$[t\to \frac{127}{64}\,t,\,\frac{3}{4},\,\frac{7}{8}],\,[t\to \frac{169}{64}\,t,\,\frac{7}{8},\,1],\,[t\to 1+t,\,1,\,0])$$

2.5 Maple による波線の干渉の処理

波線の干渉を処理するためのモデュール `kansho` を与える.

工程 2.5.1

kansho := **module**()
export*jikokudata*, *jikoku*, *jikokuindices*, *zendatalist*, *kanshodata*, *gattai*, *datalist*, *sinxdata*, *sinvdata*, *jikokuretu*, *kurikaesikanshodataretu*;
end module

以上の export 変数の説明をしよう．まず，隣接する Riemann 波が干渉する時の状況を記録し，また，その時刻を計算するプロシデュアを与えよう.

工程 2.5.2　プロシデュア `jikokudata`, `jikoku` を与える．`jikokudata` は，プロシデュア `riemann:-migihajinamisen`, `riemann:-hidarihajinamisen`

によって，$xlist$ の成分の点それぞれの両側での初期値が $vlist$ の成分で指定される Riemann 波の右端左端の波線列を構成し，隣接する Riemann 波についてこれらの波線の交差状況を `wavedata:-majiwari` によって調べる．交差時刻のうちの最小値を求め，最小値を実現する Riemann 波の添え数に対応する結果を列として出力する．

$jikokudata := \mathbf{proc}(xlist,\ vlist,\ N,\ f)$
$\mathbf{local}\, n,\ i,\ R,\ L,\ T,\ Tm,\ S;$
$\mathbf{if}\, xlist \neq \mathrm{sort}(xlist)\, \mathbf{then}\, \mathrm{RETURN}(`kansho : -jikokudata = input\_error0`)\, \mathbf{end\ if};$
$n := \mathrm{nops}(xlist);$
$\mathbf{if}\, n+1 \neq \mathrm{nops}(vlist)\, \mathbf{then}\, \mathrm{RETURN}(`kansho : -jikokudata = input\_error1`)\, \mathbf{end\ if};$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, R_i := riemann : -migihajinamisen(xlist_i,\ [vlist_i,\ vlist_{i+1}],\ N,\ f)$
$\mathbf{end\ do};$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, L_i := riemann : -hidarihajinamisen(xlist_i,\ [vlist_i,\ vlist_{i+1}],\ N,\ f)$
$\mathbf{end\ do};$
$\mathbf{for}\, i\, \mathbf{to}\, n-1\, \mathbf{do}\ T_i := wavedata : -majiwari(R_i,\ L_{i+1})_1\, \mathbf{end\ do};$
$Tm := \min(\mathrm{seq}(T_i,\ i = 1..n-1));$
$S := \{\};$
$\mathbf{for}\, i\, \mathbf{to}\, n-1\, \mathbf{do}\ \mathbf{if}\, T_i = Tm\, \mathbf{then}\, S := S\ \mathrm{union}\ \{i\}\, \mathbf{end\ if}\, \mathbf{end\ do};$
$\mathrm{seq}(wavedata : -majiwari(R_i,\ L_{i+1}),\ i = S)$
$\mathbf{end\ proc}$

`jikokudata` の出力の第 1 成分が `jikoku` である．

$jikoku := \mathbf{proc}(xlist,\ vlist,\ N,\ f)\ [\mathrm{jikokudata}(xlist,\ vlist,\ N,\ f)]_{1\,1}\ \mathbf{end\ proc}$

`jikokudata` の出力が複数項からなっても，構成上，どの項の第 1 成分も同じ値を与える[23]．

出力例を示す．

例 2.5.1

$$\mathrm{jikokudata}([-1,\ 0,\ 1],\ [0,\ -1,\ 1,\ 0],\ 3,\ x \to x^3) = [\frac{64}{121},\ \frac{48}{121},\ [\frac{-7}{8},\ \frac{1}{2}]]$$

$$\mathrm{jikoku}([-1,\ 0,\ 1],\ [0,\ -1,\ 1,\ 0],\ 3,\ x \to x^3) = \frac{64}{121}$$

次に，干渉時刻を与える Riemann 波の添え数を求めるプロシデュアを掲げよう．

工程 2.5.3　`jikokuindices` は，`jikoku` が交差時刻を出力する隣接 Riemann 波のうち，右端の波線に対応する添え数の集合を出力する．

[23]ただし，`jikokudata` がエラー・メッセージを返す場合は考慮されていない．

```
jikokuindices := proc(xlist, vlist, N, f)
local n, i, R, L, T, Tm, S;
  if xlist ≠ sort(xlist) then RETURN('kansho : −jikokuindices = input_error0') end if;
  n := nops(xlist) ;
  if n + 1 ≠ nops(vlist) then RETURN('kansho : −jikokuindices = input_error1')
  end if;
  for i to n do R_i := riemann : −migihajinamisen(xlist_i, [vlist_i, vlist_{i+1}], N, f)
  end do;
  for i to n do L_i := riemann : −hidarihajinamisen(xlist_i, [vlist_i, vlist_{i+1}], N, f)
  end do;
  for i to n − 1 do  T_i := wavedata : −majiwari(R_i, L_{i+1})_1 end do ;
  Tm := min(seq(T_i, i = 1..n − 1)) ;
  if Tm = ∞ then S := {}
  else S := {} ; for i to n − 1 do if T_i = Tm then S := S union {i}  end if end do
  end if;
  S
end proc
```

ある Riemann 波の隣接する組の干渉時点において，エントロピー解を構成するすべての Riemann 波成分の値は，干渉時点を所期時刻とする新たな初期値問題のの初期関数の決定のために不可欠である．

工程 2.5.4　`riemann:-zendata` の出力リストの各成分は $[x_i, c_i, v_i, v_{i+1}]$ の形のリストである．`kansho:-zendatalist` は，干渉時刻 T における対応する波筋の値 $x_i + c_i T$ と v_i, v_{i+1} を組合わせたリストを出力する．

```
zendatalist := proc(xlist, vlist, N, f)
local ZR, n, m, i, j, b, T, ZK;
  ZR := riemann : −zendata(xlist, vlist, N, f) ;
  n := nops(xlist) ;
  for i to n do m_i := nops(ZR_i) end do ;
  T := jikoku(xlist, vlist, N, f) ;
  for i to n dofor j to m_i do
        ZK_{ij} := [ZR_{ij_1} + T ∗ ZR_{ij_2}, ZR_{ij_3}, ZR_{ij_4}]
    end do
  end do;
  for i to n do b_i := [seq(ZK_{ij}, j = 1..m_i)] end do ;
  seq(b_i, i = 1..n)
end proc
```

出力例を挙げる．例 2.4.4 と比較せよ．

例 2.5.2

zendatalist([−1, 0, 1], [0, −1, 1, 0], 3, $x \to x^3$)
$= ([[\frac{-120}{121}, 0, \frac{-1}{8}], [\frac{-114}{121}, \frac{-1}{8}, \frac{-1}{4}], [\frac{-102}{121}, \frac{-1}{4}, \frac{-3}{8}], [\frac{-84}{121}, \frac{-3}{8}, \frac{-1}{2}],$
$[\frac{-60}{121}, \frac{-1}{2}, \frac{-5}{8}], [\frac{-30}{121}, \frac{-5}{8}, \frac{-3}{4}], [\frac{6}{121}, \frac{-3}{4}, \frac{-7}{8}], [\frac{48}{121}, \frac{-7}{8}, -1]],$
$[[\frac{48}{121}, -1, \frac{1}{2}], [\frac{61}{121}, \frac{1}{2}, \frac{5}{8}], [\frac{91}{121}, \frac{5}{8}, \frac{3}{4}], [\frac{127}{121}, \frac{3}{4}, \frac{7}{8}], [\frac{169}{121}, \frac{7}{8}, 1]],$
$[[\frac{185}{121}, 1, 0]])$

この出力例の第 1 リストの第 8 成分[24]と第 2 リストの第 1 成分[25]は，それぞれ，

$$[\frac{48}{121}, \frac{-7}{8}, -1], \quad [\frac{48}{121}, -1, \frac{1}{2}]$$

であり，これらのリストの第 1 成分が一致している．すなわち，干渉を起こしている `zendatalist` の成分リストである．このようなリストの取り出しを行なうプロシデュアが `kanshodata` である[26]．

工程 2.5.5　`kanshodata` は `zendatalist` の出力のうち，干渉を起こしているリスト成分の対をまとめてリスト化したものを列として出力するプロシデュアである．

kanshodata := **proc**(*xlist*, *vlist*, *N*, *f*)
local *ZKD*, *KJI*, *n*, *m*, *i*, *j*, *KD*;
　ZKD := zendatalist(*xlist*, *vlist*, *N*, *f*) ;
　KJI := jikokuindices(*xlist*, *vlist*, *N*, *f*) ;
　n := nops([*ZKD*]) ;
　for *i* **to** *n* **do** m_i := nops(ZKD_i) **end do** ;
　[seq([ZKD_{im_i}, ZKD_{i+1_1}], *i* = *KJI*)]
end proc

出力例を示す．

例 2.5.3　この例は上の注意に対応する．

kanshodata([−1, 0, 1], [0, −1, 1, 0], 3, $x \to x^3$) = $[[[\frac{48}{121}, \frac{-7}{8}, -1], [\frac{48}{121}, -1, \frac{1}{2}]]]$

`kansho:-kanshodata` によって得られるリストの対 $[a, b_1, b_2]$, $[a, b_1', b_2']$ から干渉時点における新たな Riemann 問題が定義できるためには，$b_2 = b_1'$ に加えて，$b_1 \neq b_2'$ でなければならない．つまり，$b_1 = b_2'$ ならば，干渉点の近くでは干渉後波の値が b_1 になってしまって新たな波は生じないのである．

[24]つまり，$riemann : -migihajidata(-1, [0, -1], 3, x \to x^3)$.
[25]つまり，$riemann : -hidarihajidata(0, [-1, 1], 3, x \to x^3)$.
[26]ここでは，3 個以上の異なる Riemann 波の干渉が一点で起こることはないことを前提にしている．この成立について数学的に予め検証しておくべきことである．

そこで，干渉するデータの対について，このような管理をするプロシデュア gattai を用意する．

工程 2.5.6　入力データは $[a, b_1, b_2]$, $[a', b'_1, b'_2]$ であるが，$[a, b_2] = [a', b'_1]$ でない限りエラー・メッセージが返る．次に，$b_1 \neq b'_2$ ならば $[a, b_1, b'_2]$ が出力されるが，$b_1 = b'_2$ ならば何も出力されない．

gattai := **proc**(*triplet1*, *triplet2*)
local *triplet*;
　if [*triplet1*$_1$, *triplet1*$_3$] $\neq$ [*triplet2*$_1$, *triplet2*$_2$] **then**
　　RETURN(‘*kansho* : $-$*gattai* = *input_error*‘)
　end if;
　if *triplet1*$_2 \neq$ *triplet2*$_3$ **then** *triplet* := [*triplet1*$_1$, *triplet1*$_2$, *triplet2*$_3$]
　else *triplet* := *NULL*
　end if;
　triplet
end proc

出力例を示す．

例 2.5.4　$a \neq a1$, $b_2 \neq b1_2$ などとして，gattai の出力は，次の通り．

gattai([a, b_1, b_2], [$a1$, $b1_1$, $b1_2$]) = *kansho* : $-$*gattai* = *input_error*
gattai([a, b_1, b_2], [a, b_2, $b1_2$]) = [a, b_1, $b1_2$]
gattai([a, b_1, b_2], [a, b_2, b_1]) = ()

zendatalist の出力に gattai の効果を取り入れて整理し，干渉時点でのデータの列を出力するプロシデュア datalist を与える．我々の目論見は，干渉時刻を初期時刻としたときに想定される初期関数の分点列と関数値の列を得ることである．

工程 2.5.7　プロシデュア datalist の入力は，*xlist*, *vlist* および N, f であり，出力は，リスト $[x_i, v_i, v_{i+1}]$ の列である．構成上，x_1, x_2, $\cdots$ は単調増大で，すべて相異なる．また，$v_i \neq v_{i+1}$ でもある．

```
datalist := proc(xlist, vlist, N, f)
local ZDL, DI, n, m, i, j, KD, KDD;
   ZDL := zendatalist(xlist, vlist, N, f);
   n := nops(xlist);
   DI := jikokuindices(xlist, vlist, N, f);
   for i to n do
      if i in DI then m_i := nops(ZDL_i) - 1 else m_i := nops(ZDL_i) end if
   end do;
   for i to n do for j to m_i do KD_ij := ZDL_ij end do end do;
   for i to n do
      if (i - 1) in DI then
         KDD_i1 := gattai(ZDL_{i-1 m_{i-1}+1}, ZDL_i1);
         for j from 2 to m_i do KDD_ij := ZDL_ij end do
      else for j to m_i do KDD_ij := ZDL_ij end do
      end if
   end do;
   seq(seq(KDD_ij, j = 1..m_i), i = 1..n)
end proc
```

出力例を示す．例 2.5.3 と比較せよ．

例 2.5.5

$$\text{datalist}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) = ([\frac{-120}{121},\,0,\,\frac{-1}{8}],$$
$$[\frac{-114}{121},\,\frac{-1}{8},\,\frac{-1}{4}],\,[\frac{-102}{121},\,\frac{-1}{4},\,\frac{-3}{8}],\,[\frac{-84}{121},\,\frac{-3}{8},\,\frac{-1}{2}],\,[\frac{-60}{121},\,\frac{-1}{2},\,\frac{-5}{8}],$$
$$[\frac{-30}{121},\,\frac{-5}{8},\,\frac{-3}{4}],\,[\frac{6}{121},\,\frac{-3}{4},\,\frac{-7}{8}],\,[\frac{48}{121},\,\frac{-7}{8},\,\frac{1}{2}],\,[\frac{61}{121},\,\frac{1}{2},\,\frac{5}{8}],$$
$$[\frac{91}{121},\,\frac{5}{8},\,\frac{3}{4}],\,[\frac{127}{121},\,\frac{3}{4},\,\frac{7}{8}],\,[\frac{169}{121},\,\frac{7}{8},\,1],\,[\frac{185}{121},\,1,\,0])$$

`datalist` を整理して，分点（x-座標）の列および関数値の列を作り出すプロシデュア `sinxdata`，`sinvdata` を与えよう．

工程 2.5.8　入力は *xlist*, *vlist* および N, f であり，出力は対応する列をリスト化したものである．

```
sinxdata := proc(xlist, vlist, N, f)
local n, DL;
   DL := datalist(xlist, vlist, N, f); n := nops([DL]); [seq(DL_i1, i = 1..n)]
end proc
```

```
sinvdata := proc(xlist, vlist, N, f)
local n, i, DL, VL;
    DL := datalist(xlist, vlist, N, f);
    n := nops([DL]);
    for i to n do VL_i := DL_{i2} end do;
    VL_{n+1} := DL_{n3};
    [seq(VL_i, i = 1..n + 1)]
end proc
```

出力例を示す．例 2.5.5 と比較せよ．

例 2.5.6

$$\text{sinxdata}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) =$$
$$[\frac{-120}{121}, \frac{-114}{121}, \frac{-102}{121}, \frac{-84}{121}, \frac{-60}{121}, \frac{-30}{121}, \frac{6}{121}, \frac{48}{121}, \frac{61}{121}, \frac{91}{121}, \frac{127}{121}, \frac{169}{121}, \frac{185}{121}]$$

$$\text{sinvdata}([-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) =$$
$$[0, \frac{-1}{8}, \frac{-1}{4}, \frac{-3}{8}, \frac{-1}{2}, \frac{-5}{8}, \frac{-3}{4}, \frac{-7}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1, 0]$$

さて，一般に，`kansho:-sinxdata`, `kansho:-sinvdata` の出力を，例えば，

$$xlist_1 := kansho : -sinxdata(xlist, vlist, N, f)$$
$$vlist_1 := kansho : -sinvdata(xlist, vlist, N, f)$$

とすると，$xlist$ の成分を分点とし，値が $vlist$ の成分の階段関数

$$g(x) = convexify : -nawabari(xlist, vlist, N, f)(x)$$

を初期関数とする近似保存則のエントロピー解

$$riemann : -zen(xlist, vlist, N, f)(t, x)$$

は，時刻 $T = kansho : -jikoku(xlist, vlist, N, f)$ で成分の Riemann 波に干渉が起きるので，$t > T$ では，もはや，有効ではない．しかし，ここから，先は，$t = T$ を初期時刻とする新たなエントロピー解

$$riemann : -zenn(xlist_1, vlist_1, N, f)(t - T, x)$$

で接続すればよい．この解の成分の波に干渉が起きるまで，この解は有効である．以下，同様の手続きの反復によって，近似保存則の大域的なエントロピー解が得られる．

そこで，干渉時刻を反復して求めるプロシデュア `jikokuretu` と，`sinxdata`, `sinvdata` の反復と組合わせたプロシデュア `kurikaesikanshodataretu` を与えよう．ただし，反復は M 回までとする．

工程 2.5.9　kurikaesikanshodataretu の出力は 3 個のリストからなる．第 1 リストは jikoku の反復結果，第 2 リストは sinxdata，第 3 リストは sinvdata の反復結果である．

kurikaesikanshodataretu := **proc**(*xlist*, *vlist*, N, f, M)
local i, j, *sinxlist*, *sinvlist*, *JKK*, n;
$sinxlist_1 := xlist$;
$sinvlist_1 := vlist$;
$JKK_1 :=$ jikoku($sinxlist_1$, $sinvlist_1$, N, f);
$n := 1$;
for i **to** M **do**
if type(JKK_i, *rational*) = *true* **then**
$sinxlist_{i+1} :=$ sinxdata($sinxlist_i$, $sinvlist_i$, N, f);
$sinvlist_{i+1} :=$ sinvdata($sinxlist_i$, $sinvlist_i$, N, f);
$JKK_{i+1} :=$ jikoku($sinxlist_i$, $sinvlist_i$, N, f);
$n := i$
end if
end do;
[seq(JKK_i, $i = 2..n$)], [seq($sinxlist_i$, $i = 1..n$)], [seq($sinvlist_i$, $i = 1..n$)]
end proc

kurikaesikanshodataretu 出力中の第 1 リストが jikokuretu の出力列をリストにしたものである．

jikokuretu := **proc**(*xlist*, *vlist*, N, f, M)
local i, *sinxlist*, *sinvlist*, *JKK*;
$sinxlist_1 := xlist$;
$sinvlist_1 := vlist$;
$JKK_1 :=$ jikoku($sinxlist_1$, $sinvlist_1$, N, f);
for i **from** 2 **to** M **do**
$sinxlist_i :=$ sinxdata($sinxlist_{i-1}$, $sinvlist_{i-1}$, N, f);
$sinvlist_i :=$ sinvdata($sinxlist_{i-1}$, $sinvlist_{i-1}$, N, f);
$JKK_i :=$ jikoku($sinxlist_i$, $sinvlist_i$, N, f)
end do;
seq(JKK_i, $i = 1..M$)
end proc

プロシデュア kansho:-kurikaesikanshodataretu の出力例は厖大になる．ここでは，kansho:-jikokuretu のみ示す．

例 2.5.7

jikokuretu([−1, 0, 1], [0, −1, 1, 0], 3, $x \to x^3$, 15) =
$(\frac{64}{121}, \frac{1024}{12705}, \frac{64}{385}, \frac{2432}{15015}, \frac{824}{3003}, \frac{56}{165}, \frac{608}{1155}, \frac{2656}{3465}, \frac{608}{495}, \frac{96}{55}, \frac{48}{11}, \frac{48}{11}, \frac{288}{11}, \frac{224}{11}, \infty)$

2.6　近似保存則のエントロピー解の大域的構成

以上のモデュールを総合して，近似保存則のエントロピー解を大域的に構成できる．モデュール entropy_solution として掲げておく．

工程 2.6.1

entropy_solution := **module**() **export** *bubunkai*, *zenbukai*; **end module**

プロシデュア bubunkai は，kansho:-kurikaesikanshodataretu の各出力に応じて，riemann:-zen を計算し，結果を列として出力させる．

工程 2.6.2

bubunkai := **proc**(*xlist*, *vlist*, N, f, M)
local i, n, *KKD*, *SOL*, t, x;
　KKD := *kansho* : −*kurikaesikanshodataretu*(*xlist*, *vlist*, N, f, M) ;
　n := nops(KKD_1) ;
　for i **to** $n+1$ **do**
　　SOL_i := unapply(*riemann* : −*zen*(KKD_{2i}, KKD_{3i}, N, f), t, x)
　end do;
　seq($SOL_i(t, x)$, $i = 1..n+1$)
end proc

次に，プロシデュア zenbukai を示す．これは，エントロピー解の大域的な構成を与えるものである．

工程 2.6.3　zenbukai は，プロシデュア bubunkai の出力にプロシデュア kansho:-jikokuretu の出力に相当する反復干渉時刻の効果を加味して総和をとった区分的に定数の関数を出力する．反復回数の M は，プロシデュアが終了しない場合の保険であるが，干渉回数が M より小さいときは，最後の干渉までしか反復をしないようにはしてある．

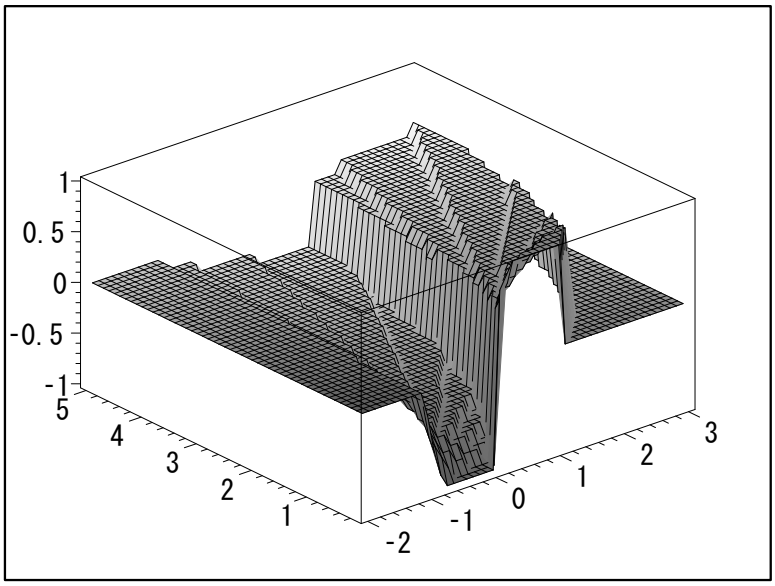

図 28: zenbukai($[-1,0,1],[0,-1,1,0],3,x\to x^3,15$) のグラフ (1)

zenbukai := **proc**(*xlist*, *vlist*, N, f, M)
local n, i, j, k, *KKD*, *BBK*, T, *SOL*, t, x;
KKD := *kansho* : −*kurikaesikanshodataretu*(*xlist*, *vlist*, N, f, M);
n := nops(KKD_1);
for j **to** n **do** T_j := sum(KKD_{1i}, $i = 1..j$) **end do**;
BBK := bubunkai(*xlist*, *vlist*, N, f, M);
SOL_1 := unapply(piecewise($t < T_1$, $BBK_1(t, x)$, $T_1 < t$, 0), t, x);
for j **from** 2 **to** n **do** SOL_j := unapply(piecewise($t < T_{j-1}$, 0,
$T_{j-1} < t$ **and** $t < T_j$, $BBK_j(t - T_{j-1}, x)$, $T_j < t$, 0), t, x)
end do;
SOL_{n+1} :=
unapply(piecewise($t < T_n$, 0, $T_n < t$, $BBK_{n+1}(t - T_n, x)$), t, x);
unapply(sum($SOL_k(t, x)$, $k = 1..n + 1$), t, x)
end proc

図 28 （$0 < t < 10$, $-2 < x < 3$）および 図 29（$0 < t < 65$, $-5 < t < 5$）に，画像化した zenbukai($[-1,0,1],[0,-1,1,0],3,x\to x^3,15$) の出力を示す.

ちなみに，$u_3(t,x)$ = zenbukai($[-1,0,1],[0,-1,1,0],3,x\to x^3,15$)$(t,x)$ と $u_4(t,x)$ = zenbukai($[-1,0,1],[0,-1,1,0],4,x\to x^3,31$) の差 $u_3(t,x) - u_4(t,x)$ の画像化を図 30 と図 31 に示す．図 30 では，恐らく，画像化する際の浮動小数点計算による誤差の効果であろうスパイクが見えているが，局所化した図 31 では，振動範囲の極めて小さいグラフが得られている.

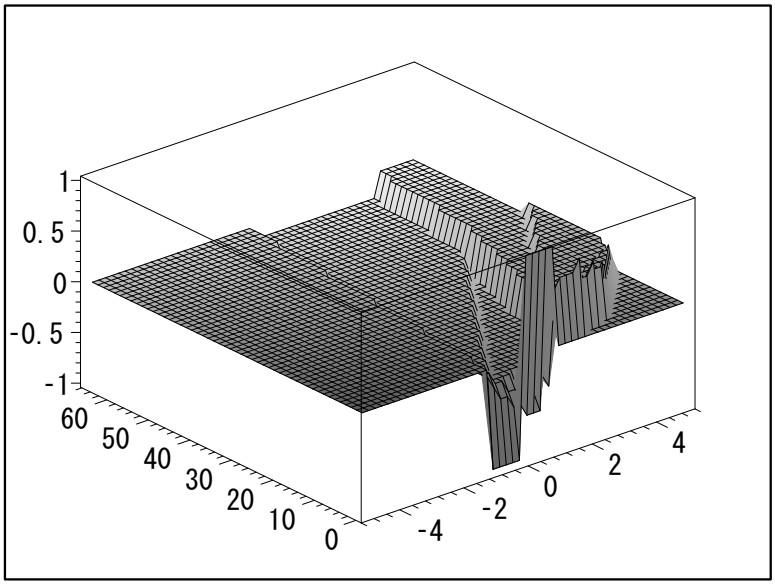

図 29: zenbukai($[-1,0,1],[0,-1,1,0],3,x \to x^3,15$) のグラフ (2)

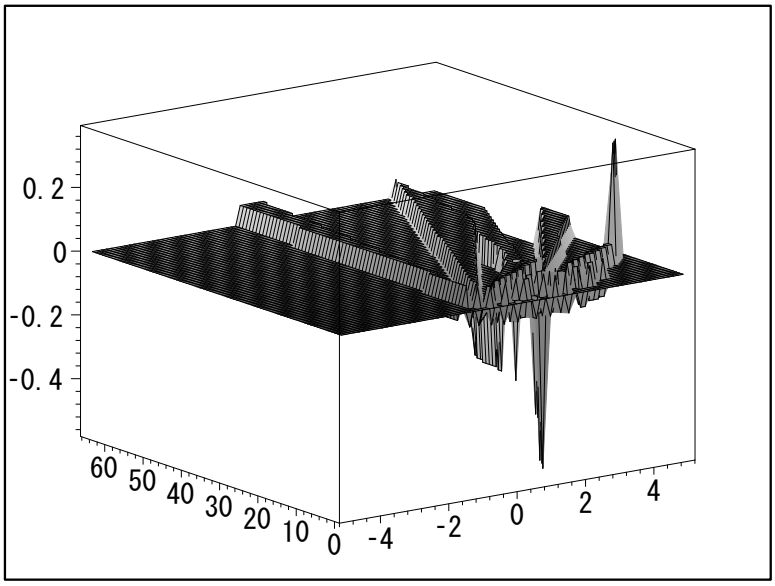

図 30: $u_3(t,x) - u_4(t,x)$ のグラフ (1)

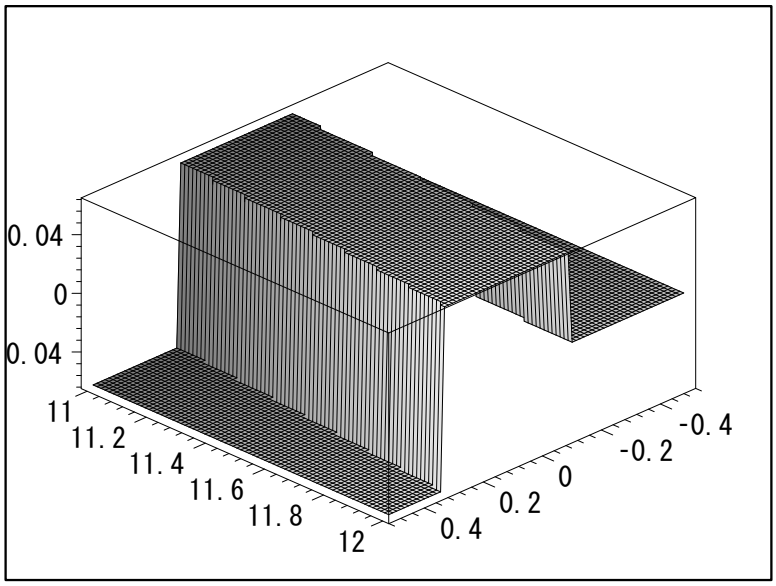

図 31: $u_3(t,x) - u_4(t,x)$ のグラフ (2)

2.7 Maple による波線追跡

近似保存則のエントロピー解には，初期関数の値と近似流量の値の間の，言わば，相性に基く不連続性が観察される．これらの不連続性は波の境界を表す不連続線であり，波面と解釈することができる．ただし，1 次元的なものであるから，波線と呼んで来たのあるが．波線は，近似保存則のエントロピー解の構造を知るために重要である．以下で，Maple のモデュール `tuiseki` による波線の追跡を示そう．

工程 2.7.1　出発点の個数とそれらから発する Riemann 波の本数は一般に異なり，さらに， Riemann 波の干渉が起きた時点で，ここまでの Riemann 波の波線と新たに発生する Riemann 波の波線の本数が変わる．モデュール `tuiseki` は，これらの変化を管理・記述するためのものである．これにより，最初の出発点から干渉を繰り返しつつ進行していく波線を追跡することができる．

tuiseki := **module**()
export*husibusi*, *zenhusibusi*, *kusari*, *tuinarabe2*, *tuinarabe3*, *juzutunagitui*, *juzutunagi*, *zenjuzutunagi*, *sujimejun*, *bubunsuji*, *sujisen*, *sujisenplot*;
end module

export 変数の大半は，以下に説明するように，中間的なものである．

工程 2.7.2　プロシデュア `husibusi` は，入力データの *xlist* の添え数と

モデュール `riemann` のプロシデュア `sinxdata` の出力の添え数について，Riemann 波の波線で結ばれるものを対にして出力するものである．したがって，各出力 $[i, j]$ は $xlist_i$ から発する Riemann 波の波線が `sinxcata` の出力の j 番目の点に達することを意味しており，$xlist_i$ から発する Riemann 波の波線中の順位を示すものではない．

$husibusi := \mathbf{proc}(xlist,\ vlist,\ N,\ f)$
$\mathbf{local}\, n,\ i,\ j,\ k,\ ZD,\ KJI,\ HSM,\ HSM1,\ hb;$
$n := \mathrm{nops}(xlist)\,;$
$ZD := riemann : -zendata(xlist,\ vlist,\ N,\ f)\,;$
$KJI := kansho : -jikokuindices(xlist,\ vlist,\ N,\ f)\,;$
$HSM_0 := 0\,;$
$HSM1_0 := 0\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}$
$\quad \mathbf{if}\, i \,\mathrm{in}\, KJI\, \mathbf{then}\, HSM_i := \mathrm{nops}(ZD_i) - 1\, \mathbf{else}\, HSM_i := \mathrm{nops}(ZD_i)\, \mathbf{end\ if}$
$\mathbf{end\ do};$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, HSM1_i := HSM1_{i-1} + HSM_i\, \mathbf{end\ do}\,;$
$\mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, \mathbf{for}\, j\, \mathbf{to}\, \mathrm{nops}(ZD_i)\, \mathbf{do}\, hb_{ij} := HSM1_{i-1} + j\, \mathbf{end\ do}\, \mathbf{end\ do}\,;$
$\mathrm{seq}(\mathrm{seq}([i,\ hb_{ij}],\ j = 1..\mathrm{nops}(ZD_i)),\ i = 1..n)$
$\mathbf{end\ proc}$

なお，`husibusi` は添え数の対を列として出力する．

出力例を示す．

例 2.7.1

husibusi($[-1,\ 0,\ 1]$, $[0,\ -1,\ 1,\ 0]$, 3, $x \to x^3$) = ([1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 8], [2, 9], [2, 10], [2, 11], [2, 12], [3, 13])

出力中の [1, 8], [2, 8] の意味は，第 1 点 -1 から出発する Riemann 波の第 8 の波線が第 2 点 0 , からの Riemann 波の最初の波線と交わり，しかも，その点は，この時点で波線群を切ったとき得られる（x-座標の）点の（左から）8 番目になるということである．

プロシデュア `husibusi` を（M 回までの）干渉の反復をまとめて一括して出力するプロシデュアが `zenhusibusi` である．ただし，初回分の添え数から終回分の添え数までが該当する Riemann 波の波線の経過をたどって追跡できるようには整理することは，別のプロシデュアに任せることにしてある．

工程 2.7.3

```
zenhusibusi := proc(xlist, vlist, N, f, M)
local KKD, n, i, HB;
    KKD := kansho : −kurikaesikanshodataretu(xlist, vlist, N, f, M);
    n := nops(KKD_2);
    for i to n do HB_i := [husibusi(KKD_{2i}, KKD_{3i}, N, f)] end do;
    seq(HB_i, i = 1..n)
end proc
```

次に掲げるのは説明のための出力例である.

例 2.7.2　$M = 3$ の場合を出力する.

zenhusibusi([−1, 0, 1], [0, −1, 1, 0], 3, $x \to x^3$, 3) =
([[1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 8], [2, 9],
[2, 10], [2, 11], [2, 12], [3, 13]],
[[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6], [7, 7], [8, 8], [9, 9],
[10, 10], [11, 11], [12, 12], [13, 12]],
[[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6], [7, 7], [8, 7], [9, 8], [10, 9],
[11, 10], [12, 11]])

次のプロシデュア `kusari` は，`zenhusibusi` の出力を整理するための第一歩である.

工程 2.7.4　`kusari` は，対を成分とするリストについて，先行の対の第 2 成分と後続の対の第 1 成分が一致しない隣接する対の組がなければ，もとのリストを返し，そうでないときは何も返さない（*NULL* を返す）.

```
kusari := proc(pairlist)
local n, i, j, S1, S2, Output;
    n := nops(pairlist);
    S1 := [seq(pairlist_{i2}, i = 1..n − 1)];
    S2 := [seq(pairlist_{i+1_1}, i = 1..n − 1)];
    if S1 = S2 then Output := pairlist else Output := NULL end if;
    Output
end proc
```

要するに，`kusari` は，添え数の対のリストに対するフィルターになる．出力例は省略する.

つぎのプロシデュア `tuinarabe2` は，対を成分とするリストの集合 *pairset1*，*pairset2* のそれぞれの成分のリストから取り出した対を並置して得られるリストにフィルター `kusari` を掛けて整理したものを出力する.

工程 2.7.5

tuinarabe2 := **proc**(*pairset1*, *pairset2*)
local *a*, *b*;
[seq(seq(kusari([*a*, *b*]), *a* = *pairset1*), *b* = *pairset2*)]
end proc

次の出力例は，例 2.7.2 の出力の対を成分とする第 1，第 2 のリストをそれぞれ対を成分とする集合に変換して tuinarabe2 を適用したものである．

例 2.7.3　出力は対 2 個を成分とするリストである．

[[[1, 1], [1, 1]], [[1, 3], [3, 3]], [[1, 2], [2, 2]], [[1, 4], [4, 4]],
[[1, 5], [5, 5]], [[1, 6], [6, 6]], [[1, 7], [7, 7]], [[1, 8], [8, 8]],
[[2, 8], [8, 8]], [[2, 9], [9, 9]], [[2, 10], [10, 10]],
[[2, 11], [11, 11]], [[2, 12], [12, 12]], [[3, 13], [13, 12]]]

この場合，tuinarabe2 によって，$14 \times 13 = 182$ 個の可能な 2 成分リストのうちから 14 個を選び出していることになる．

一方，対を成分とするリストの集合 *pairset1*，*pairset2*，*pairset3* のおのおののリストの成分から選んだ対を並置した三つ組リストに対しプロシデュア kusari の篩を掛けて出力するプロシデュアが tuinarabe3 である．

工程 2.7.6

tuinarabe3 := **proc**(*pairset1*, *pairset2*, *pairset3*)
local *a*, *b*, *c*;
[seq(seq(seq(kusari([*a*, *b*, *c*]), *a* = *pairset1*), *b* = *pairset2*), *c* = *pairset3*)]
end proc

例 2.7.3 と同様に，例 2.7.2 の出力を集合に変換したものに，プロシデュア tuinarabe3 を適用した出力例を示す．

例 2.7.4　出力は 3 個の対をまとめたリストの列である．

[[[1, 1], [1, 1], [1, 1]], [[1, 3], [3, 3], [3, 3]],
[[1, 2], [2, 2], [2, 2]], [[1, 4], [4, 4], [4, 4]], [[1, 5], [5, 5], [5, 5]],
[[1, 6], [6, 6], [6, 6]], [[1, 7], [7, 7], [7, 7]], [[2, 12], [12, 12], [12, 11]],
[[3, 13], [13, 12], [12, 11]], [[2, 11], [11, 11], [11, 10]], [[1, 8], [8, 8], [8, 7]],
[[2, 8], [8, 8], [8, 7]], [[2, 9], [9, 9], [9, 8]], [[2, 10], [10, 10], [10, 9]]]

しかし，この方式で特定の波線の干渉の追跡をするための対のリストを生成することは現実的ではない．実際，高い回数の zenhusibusi の出力のリストを集合に変換し，tuinarabe の類似工程で処理しようとすると，入力の

集合の個数に応じて計算時間や必要なメモリが際限なく増えて行き，簡単な例でも出力が得られないことが起きる．別なアプローチも必要である．

工程 2.7.7　juzutunagitui の入力 *pairlist1*，*pairlist2* はそれぞれ対を成分とするリストをさらにリスト化したものである．成分リストの長さはそれぞれ一定である．プロシデュア juzutunagitui は，前者の第 i-成分リストの最後尾の対の第 2 成分と後者の第 j-成分リストの最前首の対の第 1 成分とを比較し，一致する場合は添え数の対 $[i, j]$ の集合を出力する．

juzutunagitui := **proc**(*pairlist1*, *pairlist2*)
local *PL*, *n1*, *n2*, *s1*, *s2*, *i*, *j*, *n*;
　n1 := nops(*pairlist1*) ;
　n2 := nops(*pairlist2*) ;
　n := nops($pairlist1_1$) ;
　s1 := {seq(nops($pairlist1_i$), $i = 1..n1$)} ;
　s2 := {seq(nops($pairlist2_j$), $j = 1..n2$)} ;
　if {nops(*s1*)} union {nops(*s2*)} $\neq$ {1} **then**
　　RETURN(‘*juzutunagitui* = *input_error*‘)
　end if;
　PL := {} ;
　for *i* **to** *n1* **dofor** *j* **to** *n2* **do**
　　if $pairlist1_{in2} = pairlist2_{j11}$ **then** *PL* := *PL* union {[*i*, *j*]} **end if**
　end do
　end do;
　PL
end proc

例 2.7.3 の出力結果を *pairset1*，例 2.7.4 の出力結果を *pairset2* として，プロシデュア tuiseki:-juzutunaitui に入力した結果を挙げる．

例 2.7.5

{[1, 2], [1, 1], [3, 14], [3, 8], [3, 13], [3, 10], [1, 11], [3, 12], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [2, 9]}

添え数の対と例 2.7.3，例 2.7.4 と比較せよ．

プロシデュア juzutunagi は，入力リスト *pairlist1*，*pairlist2* に対する juzutunagitui の出力結果の添え数対に対応する *pairlist1*，*pairlist2* のリスト成分を接続した新たなリストをまとめてリストとして出力するプロシデュアである．

工程 2.7.8

```
juzutunagi := proc(pairlist1, pairlist2)
local JZ, n1, n2, n, m, i, j, k, l, JT, p, q;
    JZ := juzutunagitui(pairlist1, pairlist2);
    n1 := nops(pairlist1);
    n := nops(pairlist1[1]);
    n2 := nops(pairlist2);
    m := nops(pairlist2[1]);
    JT := {};
    for i to n1 dofor j to n2 do
        if [i, j] in JZ then
            JT := JT union {[seq(pairlist1[i,k], k = 1..n), seq(pairlist2[j,l], l = 1..m)]}
        end if
    end do
    end do;
    p := nops(JT);
    [seq(JT[q], q = 1..p)]
end proc
```

出力例を示す.

例 2.7.6　例 2.7.5 に対応するものである.　例 2.7.5 の出力が集合のために，添え数の対との対応の順序が乱れている.

```
[[[1, 3], [3, 3], [3, 13], [13, 12], [12, 11]], [[1, 2], [2, 2], [2, 12], [12, 12], [12, 11]],
[[1, 2], [2, 2], [2, 11], [11, 11], [11, 10]], [[1, 2], [2, 2], [2, 8], [8, 8], [8, 7]],
[[1, 2], [2, 2], [2, 9], [9, 9], [9, 8]], [[1, 2], [2, 2], [2, 10], [10, 10], [10, 9]],
[[1, 1], [1, 1], [1, 1], [1, 1], [1, 1]], [[1, 1], [1, 1], [1, 3], [3, 3], [3, 3]],
[[1, 1], [1, 1], [1, 2], [2, 2], [2, 2]], [[1, 1], [1, 1], [1, 4], [4, 4], [4, 4]],
[[1, 1], [1, 1], [1, 5], [5, 5], [5, 5]], [[1, 1], [1, 1], [1, 6], [6, 6], [6, 6]],
[[1, 1], [1, 1], [1, 7], [7, 7], [7, 7]], [[1, 1], [1, 1], [1, 8], [8, 8], [8, 7]]]
```

エントロピー解は，近似流量と階段関数初期値から Riemann 波の干渉の繰り返しで得られる．関与する波線の状況の理解には，ある干渉時点から次の干渉時点に至ったときの波線の番号の変化を記述する番号の対を把握することが必要である．このために，プロシデュア `tuinarabe2`，`tuinarabe3` とプロシデュア `juzutunagi` を組合わせた `zenjuzutunagi` を用意する．`juzutunagi` の入力は，$xlist$，$vlist$，N，f の他に反復回数 M を要求する．M は，言わば，保険であって，M が十分に大きいときは，`zenhusibusi` の出力の個数（`nops`）までしか計算に介入しない.

工程 2.7.9　プロシデュアのアイデアは，$zenhusibusi(xlist, vlist, N, f, M)$ の出力リストを適当に区分けして，それぞれを `tuinarabe3`，`tuinarabe2` な

どで計算し[27]，それらを `juzutunagi` で処理することである．

```
zenjuzutunagi := proc(xlist, vlist, N, f, M)
local ZHB, ZHBS, n, i, j, q, r, TN, a, b, c, JT;
  ZHB := zenhusibusi(xlist, vlist, N, f, M);
  n := nops([ZHB]);
  for i to n do ZHBS_i := convert(ZHB_i, set) end do;
  r := irem(n, 3, 'q');
  if q = 0 then
    if r = 1 then TN_{q+1} := [seq([ZHB_j], j = 1..nops(ZHB))]
    elif r = 2 then TN_{q+1} := tuinarabe2(ZHBS_1, ZHBS_2)
    end if
  else
    for j to q do TN_j := tuinarabe3(ZHBS_{3*j-2}, ZHBS_{3*j-1}, ZHBS_{3*j})
    end do;
    if r = 2 then TN_{q+1} := tuinarabe2(ZHBS_{n-1}, ZHBS_n)
    elif r = 1 then TN_{q+1} := [seq([ZHB_{nj}], j = 1..nops(ZHB_n))]
    else TN_{q+1} := NULL
    end if
  end if;
  if q = 0 then JT_{q+1} := TN_{q+1}
  else
    JT_1 := TN_1;
    for i to q - 1 do  JT_{i+1} := juzutunagi(JT_i, TN_{i+1}) end do;
    if r = 0 then JT_{q+1} := JT_q
    else JT_{q+1} := juzutunagi(JT_q, TN_{q+1})
    end if
  end if;
  JT_{q+1}
end proc
```

なお，ここに掲げるプロシデュアが，この目的のために最良であるとは思わないが，一応役には立つようである．

例えば，$zenjuzutunagi([-1,0,1],[0,-1,1,0],3,x \to x^3,17)$ を出力してみたいが，各成分が対（2 成分リスト）であるような 15 成分のリスト 29 個をさらにリスト化したものが得られることになる．出力例は省略する．

次に，添え数の対 $[p,q]$ に対応する波線を特定するために，第 p 番目の Riemann 波の中で q に対応する波線の順位を決定しなければならない．このためのプロシデュアが `sujimejun` である．

工程 2.7.10　入力データの *pair* が添え数の対である．

[27]出力リストそのものを `juzutunagi` の適用向けに加工することを含む．

```
sujimejun := proc(pair, xlist, vlist, N, f)
local HB, n, p, q, i, j, m, mm, S;
    HB := husibusi(xlist, vlist, N, f);
    n := nops([HB]);
    if nops({HB} minus {pair}) = n then RETURN('sujijun = input_error') end if;
    p := pair_1;
    q := pair_2;
    if p = 1 then m := q
    else
        S := {};
        for i to n do if [p, i] in {HB} then S := S union {i} end if end do;
        mm := min(seq(j, j = S));
        m := q - mm + 1
    end if;
    [p, m]
end proc
```

出力例を挙げる．

例 2.7.7　図 26 と比較していただきたい（図の波線の右端を下から 1 〜 13 と数える．9 に相当するのは第 2 の Riemann 波の下から 2 番目になる）．

$$\text{sujimejun}([2,\,9],\,[-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) = [2,\,2]$$
$$\text{sujimejun}([3,\,9],\,[-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3) = \mathit{sujijun} = \mathit{input\_error}$$

最後の例は，エラー・メッセージである．

添え数の対に相当する波線を切り出すプロシデュアを与える．

工程 2.7.11　入力データの *Tpair* は，指定された添え数の対に対し本来意味を持つ時間間隔を示すためである．ただし，プロシデュアとしては任意に与えられるものになっている．

```
bubunsuji := proc(pair, Tpair, xlist, vlist, N, f)
local SJ, t, a, b, m, NS, BBS;
    SJ := sujimejun(pair, xlist, vlist, N, f);
    a := xlist_SJ_1;
    b := [vlist_SJ_1, vlist_SJ_1+1];
    m := SJ_2;
    NS := unapply([riemann : -namisuji(a, b, N, f)]_m(t), t);
    BBS :=
        piecewise(t < Tpair_1, 0, Tpair_1 < t and t < Tpair_2, NS(t), Tpair_2 < t, 0);
    unapply(BBS, t)
end proc
```

このプロシデュアは完全に部分的なもので，例を与えにくいものではあるが，理論的な例というべきものは作ることができる．

例 2.7.8

$$\text{bubunsuji}([2,\,11],\,[1,\,2],\,[-1,\,0,\,1],\,[0,\,-1,\,1,\,0],\,3,\,x \to x^3)(t) =$$

$$\begin{cases} 0 & t < 1 \\ \dfrac{127\,t}{64} & -t < -1 \textbf{ and } t < 2 \\ 0 & 2 < t \end{cases}$$

$[2, 11]$ とは何か．これは `sujimejun` の適用で確かめられるべきことである．今の場合は，0 から発する左から 4 番目の波線の一部に相当する（ 出力は $1 < t < 2$ の部分．この波線が，本来，意味を持つところではないが）．

さて，初期関数から発する Riemann 波の波線から（干渉回数 M までの）波線の接続を，添え数の対の系列の指定に応じて構成するプロシデュア `sujisen` を示す．

工程 2.7.12　入力データは，添え数の対の系列をリスト化した *pairlist* と初期関数のデータ *xlist*，*vlist* および近似度 N，流量関数 f，反復回数 M である．

sujisen := **proc**(*pairlist*, *xlist*, *vlist*, N, f, M)
local *M1*, *KKD*, T, S, i, j, k, n, t, *bbs*;
　KKD := *kansho* : −*kurikaesikanshodataretu*(*xlist*, *vlist*, N, f, M) ;
　M1 := nops(KKD_1) ;
　T := [0, seq($KKD_{1\,i}$, $i = 1..M1$), ∞] ;
　S := [seq(sum(T_j, $j = 1..i$), $i = 1..M1 + 2$)] ;
　n := nops(*pairlist*) ;
　if $M < n$ **then** RETURN(‘*tuiseki* : −*sujisen* = *input_error*‘) **end if** ;
　for k **to** n **do**
　　bbs_k := bubunsuji($pairs_k$, $[T_1,\, T_{k+1}]$, KKD_{2k}, KKD_{3k}, N, f)
　end do;
　unapply(sum($bbs_i(t - S_i)$, $i = 1..n$), t)
end proc

出力は，画像化したものを次節で示す．なお，画像化のためのプロシデュアもモデュールの中に `sujisenplot` として用意してみた．利用状況[28]を考えると，発想を変える必要がありそうな代物であるが，一応，次に掲げる．

[28]事実，以下の出力例では利用しない．

工程 2.7.13

sujisenplot := **proc**(*pairlist*, *xlist*, *vlist*, N, f, M)
local *sjs*, t, n, KKD, TT, i, j;
KKD := *kansho* : $-$*kurikaesikanshodataretu*(*xlist*, *vlist*, N, f, M) ;
n := nops(*pairs*) ;
sjs := sujisen(*pairlist*, *xlist*, *vlist*, N, f, M) ;
TT := sum($KKD_{1\,i}$, $i = 1..n$) ;
plot(*sjs*(t), $t = 0..TT$, *color* = *black*)
end proc

不満は，プロシデュアによる画像化の範囲が TT までと固定されていることだけではない．

さて，以上を整理して，すべての不連続線を（関数の値として）出力させるプロシデュアが `zensujisen` である．

工程 2.7.14　出力はリストの形であり，成分は，入力データ t における関数値である．

zensujisen := **proc**(t, *xlist*, *vlist*, N, f, M)
local PL, n, *sjs*, i;
PL := zenjuzutunagi(*xlist*, *vlist*, N, f, M) ;
n := nops(PL) ;
for i **to** n **do** sjs_i := sujisen(PL_i, *xlist*, *vlist*, N, f, M)(t) **end do** ;
[seq(sjs_i, $i = 1..n$)]
end proc

次節で，プロシデュア `zensujisen` のアイデアを説明する．

`zensujisen` の出力例は省略する．これと Maple の組込みコマンド `plot` を合成した出力画像の例が，図 2 である．また，次節で掲げる図 32〜図 37 も，`zensujisen` で描くことができる（文脈上はそうなっていない）．

2.8　波線の追跡画像

プロシデュア `tuiseki:-sujisen` で計算した初期時刻からたどった波線の全体を画像化して出力しよう．まず，計算をする[29]．

例 2.8.1　最初に，$N = 3$ として，`zenjuzutunagi` を出力させずに計算させる（$xlist = [-1, 0, 1]$, $vlist = [0, -1, 1, 0]$, $f(u) = u^3$）．これによって，この場合，波線の全体像を計算で求めるために必要な連続する添え数の対の情報が得られたことになる．

[29] ここでは Maple のワークシートでの計算を示す．

```
> start:=time():
> pairlists:=tuiseki:-zenjuzutunagi
> ([-1,0,1],[0,-1,1,0],3,x->x^3,17):
> elapsed3:=time()-start;
```

$$elapsed3 := 8.553$$

すなわち，計算時間は 8.553 であった[30].

同様の計算を $N = 4$ の場合に試みたのが次の例である.

例 2.8.2

```
> start:=time():
> pairlists4:=tuiseki:-zenjuzutunagi
> ([-1,0,1],[0,-1,1,0],4,x->x^3,90):
> elapsed4:=time()-start;
```

$$elapsed4 := 34.209$$

この場合の計算時間は 34.209 であった.

以上の結果を sujisen に反映させる.

例 2.8.3 $N = 3$ の場合である．反復回数が 29 なのは，*pairsets* の成分リストの個数が 29 だからである.

```
> for i from 1 to 29 do
> sjs[i]:=tuiseki:-sujisen
> (pairlists[i],[-1,0,1],[0,-1,1,0],3,x->x^3,15)
> end do:
```

なお，ここまでの計算で 263.419 掛かっている．これによって，しかし，近似保存則の対応するエントロピー解の波線が $sjs_i(t), i = 1, \cdots, 29,$ としてすべて得られたことになる.

次は $N = 4$ の場合である.

例 2.8.4 反復回数 89 は事前の知識である.

```
> for i from 1 to 89 do
> sjs4[i]:=tuiseki:-sujisen^3
> (pairlists4[i],[-1,0,1],[0,-1,1,0],4,x->x^3,90)
> end do:
```

[30]Pentimum III. Windows XP. これは長め：バックグランドの状況で若干相違が出る.

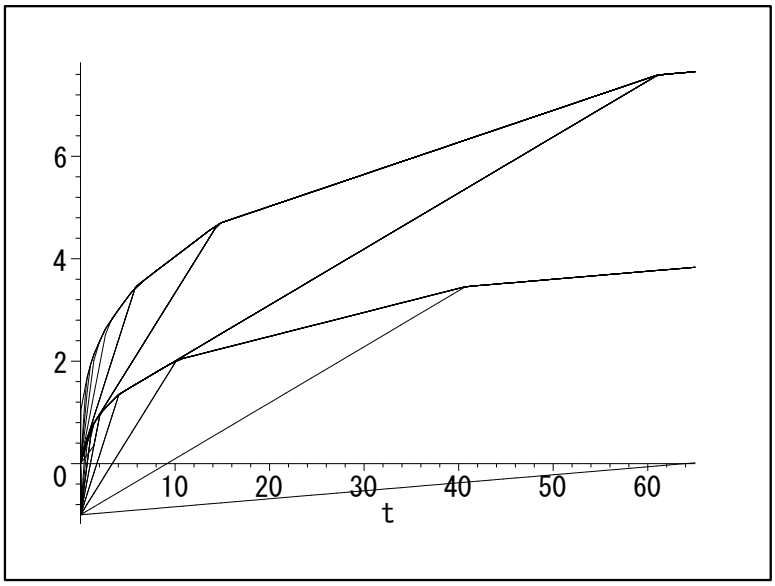

図 32: 波線図（１） $N = 3$，$0 < t < 65$

一方，この場合は 3286.749 掛かってしまった．この計算で得られた波線は $sjs4_j(t)$，$j = 1, \cdots, 89$，である．

後は，画像化のためのコードである．一例だけ挙げる．

例 2.8.5　$disp30$ と名前を付しているのは，他の画像と重ね合せたいからである．

```
> disp30:=plot([seq(sjs[i](t),i=1..29)],
> t=0..65,color=black,xtickmarks=5):
```

出力は図 32 である[31]．図 32 では，t が小さいところが混んでいて見にくい．$0 < t < 3$ に限定したのは，図 33 である．

同様の図を $N = 4$ の場合に描いたのが，図 34，図 35 である．図 36 は，図 33，図 35 の重ね合わせに相当する．図 37 は $60 < t < 450$ での重ねあわせである．

[31]出力用のコードは

```
> plots[display](disp30);
```

である．

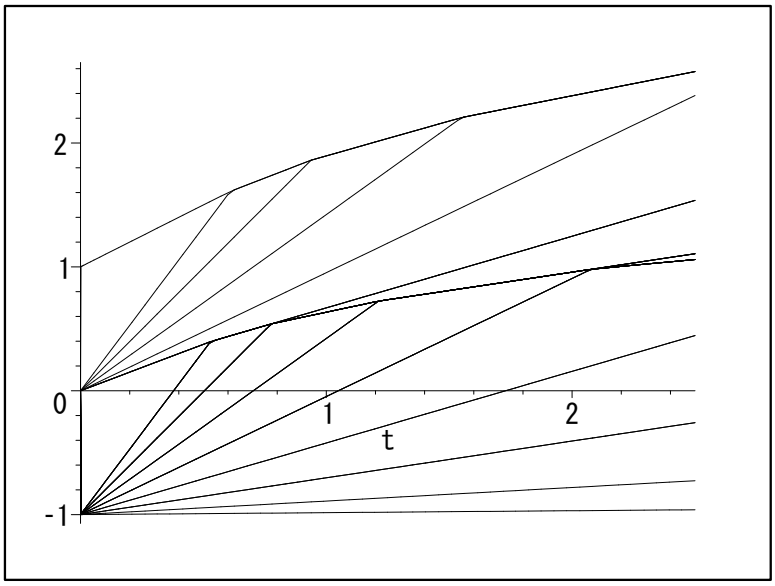

図 33: 波線図（２） $N = 3,\ 0 < t < 2.5$

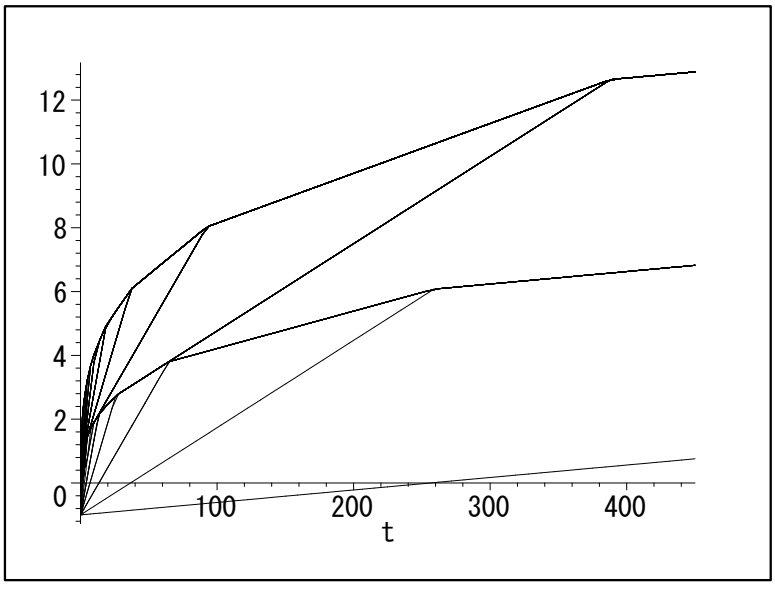

図 34: 波線図（３） $N = 4,\ 0 < t < 450$

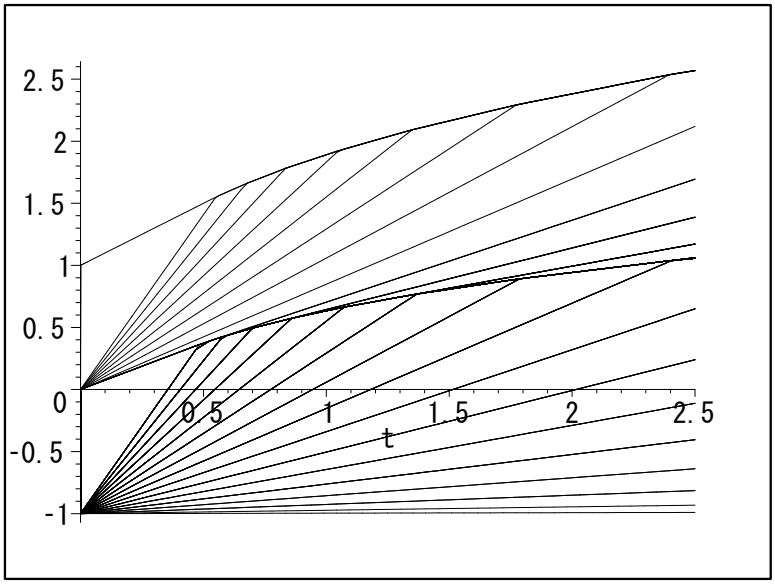

図 35: 波線図（４） $N = 4,\ 0 < t < 2.5$

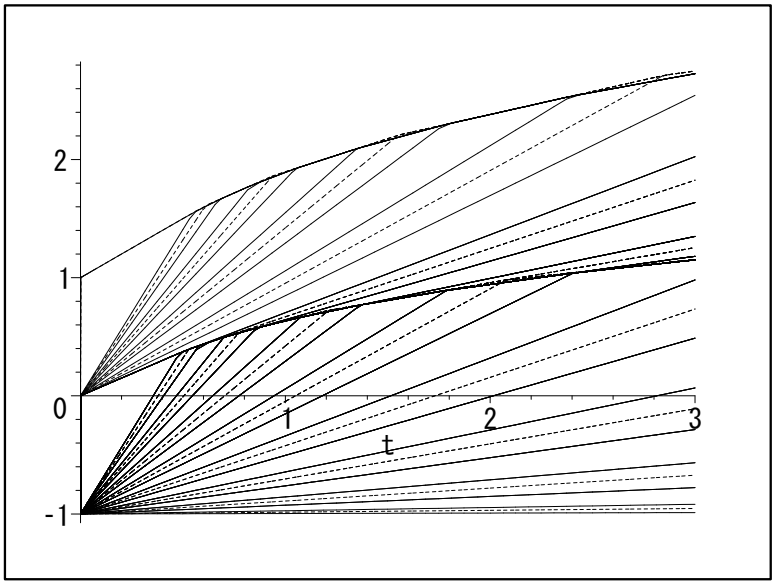

図 36: 波線図（５） $N = 3$（点線），$N = 4$（実線），$0 < t < 2.5$ の重ね合わせ

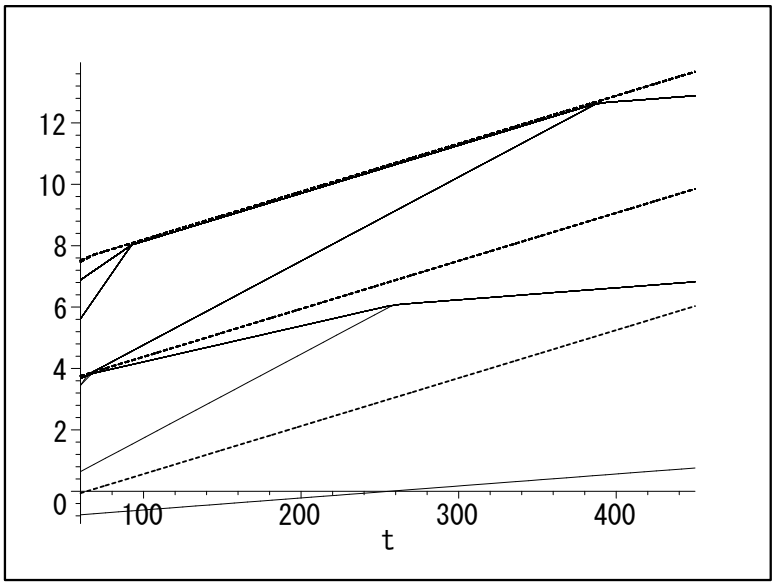

図 37: 波線図（６） $N = 3$（点線），$N = 4$（実線），$60 < t < 450$ の重ね合わせ

参考文献

[1] Alberto Bressan. *Hyperbolic systems of conservation laws*. Oxford University Press, 2000.

[2] C. Dafermos. Polygonal approximations of solutions of the initial value problems for a conservation law. *J. Math. Anal. Appl.*, Vol. 38, pp. 33–41, 1972.