卒業論文

特殊相対性理論における光行差

大阪工業大学
情報科学部　情報科学科
学生番号　A03-017

犬束 高士

2007年2月13日

目 次

1　序論

相対性理論とは, 時間・空間の物理学である. この相対性理論はアインシュタインが, 一人で作り上げたものである. 世界・宇宙でどこでも, またどのような運動をしていようとも, 過去から未来へと流れている. また空間も四方上下に広がっていると考えられていた.

アインシュタインは, マックスウェルの電磁気学とニュートン力学での間に生じる矛盾を解決しようと研究をはじめた. その結果ニュートン力学での時間や空間での概念に誤りがあること, 矛盾が生じていることを発見した.

1905 年に発表された特殊相対性理論は, 光の速さはどの観測者からも不変であり, 最大速度であることを基礎としている. その結果, 光の速度に近い速度で運動している人と静止している人では, 時間の流れ・見える色や物体を観測する角度が異なることになる. 特殊相対性理論をもとに動いている観測者と静止している観測者から見える景色がどのように違いが生じてくる現象を「光行差」という.

この光行差を身近なものと考えた物語が、「不思議宇宙のトムキンス」(文献 [3]) の冒頭に描かれている。この物語は、自然界の制限速度が 30km と我々の身近にある速度に置き換えてのお話である。その世界で移動すると、景色・周りの人は細く見え、時間の遅延も生じてくるが、実際に自然界の制限速度に近い速度で運動すると、どのような景色になるのかを本研究では, この「光行差」の概念を中心に考察していく.

ニュートン力学を用いたガリレイ変換を光速不変に対応させ拡張した変換式, ローレンツ変換から光行差の式を求める. 求めた式から, 光速に近い速度で移動するロケットから見える光景が, 速度と共にどのように変化するのかを,Java を用いたプログラムを作成して可視化した.

2　特殊相対性理論

この章では参考文献 [2] を引用している。

2.1　ガリレイ変換

まずはじめにニュートン力学での時間と空間を考える.
ニュートン力学は３つの法則からなる.

- 第一法則（慣性の法則）.
  外力が加わらなければ, 質点はその運動状態又は静止状態を維持する.

- 第二法則（ニュートンの運動方程式）.
  質点の運動又は運動量の時間的変化は, 質点にかかる力の大きさに比例し, 力の方向に作用する.m を質点の質量,x を質点の座標,t を時間,f を質点にかかる力とするとき, 次の式が成り立つ

$$m\frac{d^2x}{dt^2} = f \tag{1}$$

- 第三法則（作用反作用の原理）
  二つの質点の間に働く力には一方の質点に作用する力だけでなく, 他方への反作用の力がある. これらの力は大きさが等しく, 方向が逆である. ただし, 方向が逆だと言っても２点間力の方向は一直線上とは限らない.

第二法則で示した座標系を x 系と呼ぶ. このとき別の慣性系（慣性の法則が成り立つ座標系）x' 系でも同様に式 (1) が成り立つはずである.

$$m\frac{d^2x'}{dt^2} = f \tag{2}$$

これは２つの座標系を結ぶ座標変換により証明される.x 系を静止している観測者とし,x' 系を速度 v で運動している観測者とする.

横 x, 縦 y, 奥行き z, 時間 t の４次元空間で考えたとき, ここでは簡単化のために,x 軸方向のみ運動している座標系とすると, 座標変換は

$$\begin{aligned} t' &= t \\ x' &= x - vt \\ y' &= y \\ z' &= z \end{aligned} \tag{3}$$

と表すことができる（図 1）.x 系,x' 系. この 2 つの慣性系の間を結ぶ変換をガリレイ変換と呼ぶ. 式 (2) が成立することは, 式 (3) を式 (1) に代入することで証明される. 時間は不変的に流れるものなので, 両座標系の時間を一致させることにより, 式 (1) は不変だと言える.

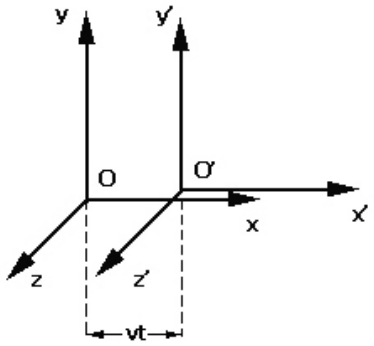

図 1: vt グラフ (出展 [2])

2.2 ローレンツ変換

しかし, このガリレイ変換には限界がある. それは光速不変の原理に当てはめることができない. なぜならニュートン力学で速度の合成を行うと矛盾が生じるからである.
例として, 合成した速度を V, 電車の速度を v_1, 電車の中で移動する人の速度を v_2 として,v_1・v_2 を供に光速 c で動くとした場合

$$\begin{aligned} V &= v_1 + v_2 \\ &= c + c = 2c \end{aligned} \tag{4}$$

となり計算上では, 光の速度を超えてしまい光速不変の原理との矛盾が生じてしまう. よって, ニュートン力学での速度の合成が矛盾を生じるので, ガリレイ変換も矛盾が生じる.
しかしこのガリレイ変換は速度 v が光の速さより十分小さい場合は近似的に正しいはずである.
アインシュタインはガリレイ変換と光速不変の原理との矛盾に気づき, 解決するために次の 3 つの条件のもとガリレイ変換を拡張した.

- 相対性原理（座標系（慣性系）の相対性）
- 光速不変の原理
- 速度 v/光速 $c \ll 1$ の極限とガリレイ変換が一致する.

まず座標系の相対性により, 座標変換は次式のようになる.

$$\begin{pmatrix} ct' \\ x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} ct \\ x \\ y \\ z \end{pmatrix} \tag{5}$$

式 (5) のようにする理由としては, もし t や x,y,z に関して 2 次や 3 次の項を含む場合,x 系で等速運動している質点は x' 系で力を受けないにも関わらず加速度運動をしていることになるからである.x 系での等速運動が x' 系でも等速運動であるためには, その逆も成立するために線形変換でなければならない.
ここでは時間 t に光速 c を乗算することにより座標を同じ長さの次元をもった量に変換すると行列が無次元となり, きれいな形式で表現されるからである.

ガリレイ変換と同様に二つの慣性系を考え, 計算を容易にするため x 軸方向のみの運動を考える (図 1). このとき,x 軸方向に速度 v で運動しているとする. つまり,$y = y'$,$z = z'$ であり,$a_{22} = a_{33} = 1$,$a_{20} = a_{21} = a_{23} = a_{30} = a_{31} = a_{32} = 0$ である. 同様に,$a_{02} = a_{03} = a_{12} = a_{13} = 0$ でなければならない. なぜなら式 (5) の逆変換をしたときに,$y = y'$,$z = z'$ とならず,y や z が ct' や x' の関数になってしまうからである.
代入した式が

$$\begin{pmatrix} ct' \\ x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & 0 & 0 \\ a_{10} & a_{11} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} ct \\ x \\ y \\ z \end{pmatrix} \tag{6}$$

となり

$$\begin{aligned} ct' &= a_{00}ct + a_{01}x \\ x' &= a_{10}ct + a_{11}x \\ y' &= y \\ z' &= z \end{aligned} \tag{7}$$

と書ける. では a_{00},a_{01},a_{10},a_{11} を考えるとする. x 系の原点 O は x' 上では,x' 軸の負の方向に速度 v で移動するので

$$x' = -vt' \tag{8}$$

である. これは式 (7) に $x = 0$ で求められる原点 O のときの運動式

$$\begin{aligned} ct' &= a_{00}ct \\ x' &= a_{10}ct \end{aligned} \tag{9}$$

と一致しなければならない. 式 (8)(9) より

$$\frac{a_{10}}{a_{00}} = -\frac{v}{c} \tag{10}$$

である. 同様に x' 系からの時も考えると, 原点 O' は x 系では x 軸上を正の方向に速度 v で運動しているので

$$x = vt \tag{11}$$

であり, 式 (7) に $x' = 0$ を代入して求められる運動は

$$x = -\frac{a_{10}}{a_{11}}ct \tag{12}$$

と一致しなければならない. 式 (11)(12) より

$$\frac{a_{10}}{a_{11}} = -\frac{v}{c} \tag{13}$$

である. 式 (10)(13) より,$a_{00} = a_{11}$ でなければならない. ここで $a_{00} = a_{11} = \gamma$ と便宜上置くことにする.

次に原点から x 軸方向に光を放出したとする. 放出された光は x 系において,$x = ct$ と運動していく. これを式 (7) より x' 系の座標に変換すると

$$\begin{aligned} ct' &= (a_{00} + a_{01})ct \\ x' &= (a_{10} + a_{11})ct \end{aligned} \tag{14}$$

となる. ここで光速不変の原理より光速はどの慣性系でも同じ値なので,x' において放出された光も同様に $x' = ct'$ で運動するはずである. これを式 (14) に代入することで $(a_{00} + a_{01})$=$(a_{10} + a_{11})$ が得られる. これと γ より

$$a_{01} = a_{10} \tag{15}$$

でなければならない. これらから,ct,x の変換式は

$$\begin{aligned} ct' &= \gamma x - \gamma(\frac{v}{c})x \\ x' &= -\gamma(\frac{v}{c})ct + \gamma x \end{aligned} \tag{16}$$

となる. この座標変換の逆変換は, 式 (16) を ct,x について解くと

$$\begin{aligned} ct &= \frac{1}{\gamma[1 - (\frac{v}{c^2})]}ct' + \frac{\frac{v}{c}}{\gamma[1 - (\frac{v}{c^2})]}x' \\ x &= \frac{\frac{v}{c}}{\gamma[1 - (\frac{v}{c^2})]}ct' + \frac{1}{\gamma[1 - (\frac{v}{c^2})]}x' \end{aligned} \tag{17}$$

となる. この逆変換は,x' 系からそれに対して x' 方向に速度 $-v$ で運動している x 系への座標変換に対応する. つまりこれは, いま求めようとしている座標変換式 (7) において, 座標変換に ’ が

ついているものといないものとを入れ替えて, 速度 v を $-v$ に置き換えたものと同じであるはずである. 式 (7) を書き換えた式 (16) において置き換えた式

$$\begin{aligned} ct &= \gamma ct' + \gamma(\frac{v}{c})x \\ x &= \gamma(\frac{v}{c})ct' + \gamma x' \end{aligned} \tag{18}$$

は式 (17) と一致しなければならい. これより

$$\gamma = \frac{1}{\sqrt{1-(\frac{v}{c})^2}} \tag{19}$$

であることがわかる. 式 (19)(16) より, 式 (7) は

$$\begin{aligned} x &= \frac{x' + vt'}{\sqrt{1-\frac{v^2}{c^2}}} \\ y &= y' \\ z &= z' \\ t &= \frac{t' + \frac{v}{c^2}x'}{\sqrt{1-\frac{v^2}{c^2}}} \end{aligned} \tag{20}$$

となる. 式 (20) のような変換をローレンツ変換と言う.

このローレンツ変換を用いることにより, 速度 v が光速 c に比べて十分に小さい極限では, ガリレイ変換と一致することを以下で示す.

$y = (v/c)^2$,$f(y) = x' + vt'/\sqrt{1-y}$ と置く, この時 $f(y)$ を級数展開すると

$$\begin{aligned} f(y) &= f(0) + f(0)' + \frac{f(0)''}{2!} + \cdots + \frac{f(0)^n}{n!} \\ &= x' + vt' + \frac{y}{\sqrt{(1-y)^3}} + \cdots \\ &= x' + vt' + \frac{0}{\sqrt{(1-0)^3}} + \cdots \\ &= x' + vt' \end{aligned} \tag{21}$$

となり, 1 次以降のオーダーは無視される. 式 (21) により,$x' = x - vt$ となり速度 v が光速 c に比べて十分に小さい極限をとるとき, ローレンツ変換はガリレイ変換 (式 (3)) に帰着することがわかる.

3　速度の合成と光行差

この章では文献 [1] を引用している
光行差とは, 静止観測者と動的観測者とでは観測する物体の見かけの方向がずれるという現象である. 本研究では、光速に近い観測者（ロケット操縦者）からの光行差を考える。この章では必要となる式をまとめる。

3.1　ニュートン力学においての光行差

まずニュートン力学での光行差について考える. 例として雨の日に電車の中から見た窓の様子を考えると (図 2), 電車の速度を V_1, 雨の落下速度を V_0 としたとき, 電車の窓から見える雨の角度は、電車の速度分だけ雨が後方に向かうと考えて

$$\tan\theta = \frac{|\vec{V_0}|}{|\vec{V_1}|} \tag{22}$$

となる (図 2)。これより動いてる電車の中から見える雨の角度と静止している人から見える雨の降る角度は異なって見える.

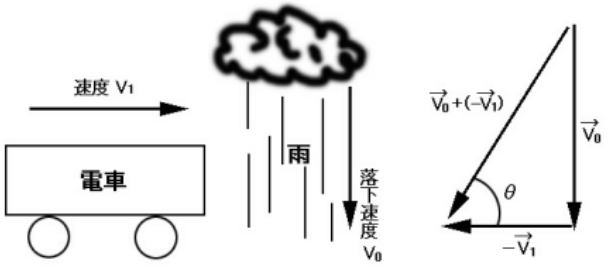

図 2: 電車と雨

3.2　特殊相対性理論における光行差

本研究のテーマである特殊相対性理論における光行差を求めるにあたって, ある物体の 1 つの基準系での速度と第 2 の基準系での速度を関係付ける公式を導く.

§2 で導いたローレンツ変換を用いて,x 系と x' 系の速度の変換公式を導くことができる. いま, 座標系間の移動速度は図 1 のように x 軸方向に速度 v であるとする.x 系で見た物体の速度 (v_x, v_y, v_z) と x' 系で見た物体の速度 (v'_x, v'_y, v'_z) の変換は, まず各成分の微小量を取ると

$$\begin{aligned} dx &= \frac{dx' + vdt'}{\sqrt{1-\frac{v^2}{c^2}}} \\ dy &= dy' \end{aligned}$$

$$
\begin{aligned}
dz &= dz' \\
dt &= \frac{dt' + \frac{v}{c^2}dx'}{\sqrt{1-\frac{v^2}{c^2}}}
\end{aligned}
\tag{23}
$$

となる. 式 (23) で求めた各成分を時間の微小量で割ることにより x 系での速度を x' 系での速度より対応させる式が求められる.

$$
\begin{aligned}
v_x &= \frac{dx}{dt} = \frac{v'_x + v}{1 + v'_x \frac{v}{c^2}} \\
v_y &= \frac{dy}{dt} = \frac{v'_y \sqrt{1-\frac{v^2}{c^2}}}{1 + v'_x \frac{v}{c^2}} \\
v_z &= \frac{dz}{dt} = \frac{v'_z \sqrt{1-\frac{v^2}{c^2}}}{1 + v'_x \frac{v}{c^2}}
\end{aligned}
\tag{24}
$$

同様に x' 系での物体の速度は,

$$
\begin{aligned}
v'_x &= \frac{dx'}{dt'} = \frac{v_x - v}{1 - v_x \frac{v}{c^2}} \\
v'_y &= \frac{dy'}{dt'} = \frac{v_y \sqrt{1-\frac{v^2}{c^2}}}{1 - v_x \frac{v}{c^2}} \\
v'_z &= \frac{dz'}{dt'} = \frac{v_z \sqrt{1-\frac{v^2}{c^2}}}{1 - v_x \frac{v}{c^2}}
\end{aligned}
\tag{25}
$$

である.

式 (24) から相対性理論における速度の合成法則を導くことができる. 簡単のため, 物体が x 軸に平行に動くとき, すなわち

$$
\begin{aligned}
v_x &= v \\
v_y &= v_z = 0
\end{aligned}
\tag{26}
$$

を考えると式 (24)(26) より

$$
v = \frac{v' + v}{1 + v' \frac{v}{c^2}}
\tag{27}
$$

となる. 式 (27) は, 2 つの速度の和は, 決して光速より大きくならないという特殊相対性理論における速度の合成法則を表している. これにより光速は常に最大速度であるという理論を構築していることになる.

次に物体の見かけの角度の変化を考える, 計算を容易にするため、与えられた瞬間における

物体の速度が $x-y$ 平面にあるように 2 次元で考え座標軸を選ぶ。
このとき観測者が物体を見上げた時の角度を θ とすると, 動的観測者の速度は x 系での成分では

$$\begin{aligned} v_x &= v\cos\theta \\ v_y &= v\sin\theta \end{aligned} \tag{28}$$

を持つ.(図 3)

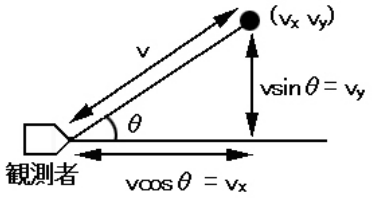

図 3: 観測者から見る物体の座標

同様に x' 系でも,

$$\begin{aligned} v'_x &= v'\cos\theta' \\ v'_y &= v'\sin\theta' \end{aligned} \tag{29}$$

の成分を持つはずである. この時 θ' を観測者が物体を見上げた角度とする.
これらの成分を式 (24) に代入すると,

$$\begin{aligned} v_x &= v\cos\theta = \frac{v'\cos\theta' + V}{1 + v'\cos\theta'\frac{V}{c^2}} \\ v_y &= v\sin\theta = \frac{v'\sin\theta'\sqrt{1-\frac{V^2}{c^2}}}{1 + v'\cos\theta'\frac{V}{c^2}} \end{aligned} \tag{30}$$

と書ける. 式 (30) で求めた成分を割ると

$$\tan\theta = \frac{v_y}{v_x} = \frac{v'\sqrt{1-\frac{V^2}{c^2}}}{v'\cos\theta' + V}\sin\theta' \tag{31}$$

と導くことができる. 式 (31) は速度の向き（角度）が同じ基準系中での変換によってどのように変わるかを表している.

式 (31) で $v' = c$ とすれば光の見かけの角度のズレが表現できる. これを光行差という. このとき $v' = c$ となり, 式 (31) は次のようになる。

$$\tan\theta = \frac{\sqrt{1-\frac{V^2}{c^2}}}{\frac{V}{c} + \cos\theta'}\sin\theta' \tag{32}$$

式 (32) は x 系から x' 系への角度の変換式を表す.
例えば、速度 V で動く観測者が遠方の星を見上げる角度 θ' は、相対的に $-V$ で外部が運動していると考えられるので、

$$\tan\theta' = \frac{\sqrt{1-\frac{(-V)^2}{c^2}}}{\cos\theta - \frac{(-V)}{c}}\sin\theta \tag{33}$$

となる。次章では式 (33) を用いて、光行差の可視化を行う。

4　光行差の可視化

動的観測者が光速 c に近い速度で進むロケットから静止した星の光を観測するモデルを考える。すなわち、静止しているときの星の見上げる角度を θ、ロケットから星を見上げる角度を θ' として、式 (33) から θ と θ' の関係を、ロケットの速度 V を変数として調べることを考える。

§3 で導いた式 (33) に実際に静的観測者から物体を見る角度と動的観測者の速度を定め, 数値化したものが表 1,2 のようになる. 表 2 は、式 (33) にロケットの速度 V と静的観測者から物体を見上げた角度 θ を求めたものである。$\tan\theta'$ から θ' を求めるには式 (34) を用いた.

$$\theta' = \arctan(\tan\theta') \tag{34}$$

表 1: $\tan\theta$ から $\tan\theta'$ の変換値

速度V	θ（度）	0	30	60	90
	$\tan\theta$	0.00	0.58	1.73	
		$\tan\theta'$			
0		0.00	0.58	1.73	
0.1c		0.00	0.51	1.44	9.95
0.2c		0.00	0.46	1.21	4.90
0.3c		0.00	0.41	1.03	3.18
0.4c		0.00	0.36	0.88	2.29
0.5c		0.00	0.32	0.75	1.73
0.6c		0.00	0.27	0.63	1.33
0.7c		0.00	0.23	0.52	1.02
0.8c		0.00	0.18	0.40	0.75
0.9c		0.00	0.12	0.27	0.48
0.99c		0.00	0.04	0.08	0.14

表 2: 各速度での θ' の値

速度V	角度(度)			
	θ			
	0	30	60	90
	θ'			
0	0	30.00	60.00	90.00
0.1c	0	27.25	55.15	84.26
0.2c	0	24.68	50.48	78.46
0.3c	0	22.25	45.92	72.54
0.4c	0	19.90	41.41	66.42
0.5c	0	17.59	36.87	60.00
0.6c	0	15.26	32.20	53.13
0.7c	0	12.84	27.27	45.57
0.8c	0	10.21	21.79	36.87
0.9c	0	7.04	15.09	25.84
0.99c	0	2.18	4.69	8.11

これらの数値をグラフ化したものが図 4 である.

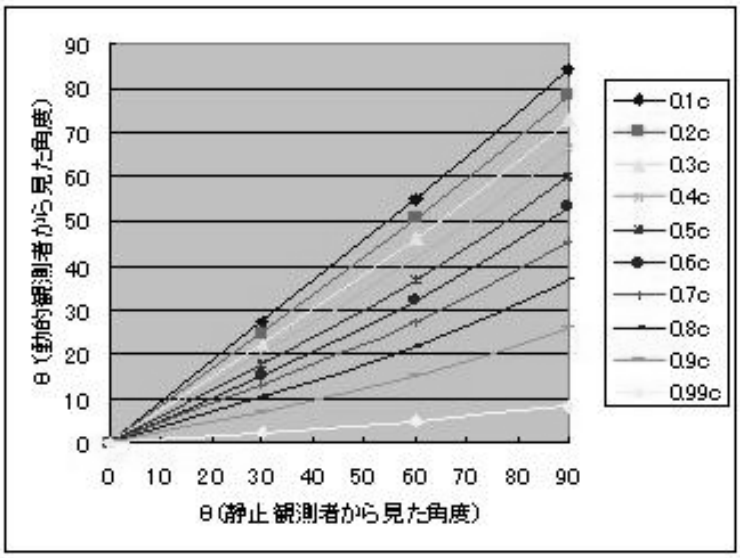

図 4: 各速度での θ と θ' のグラフ

図 4 より, 速度 v が光の速度 c に近づくと, 物体の見かけの角度が 0 度に収束してゆくことがわかる. このことを踏まえて,java 言語を用いて動的観測者から見える物体の可視化を行うプログラムを作成した.

4.1 見上げる星が一つの場合

観測者は光速 c に近いロケットに乗っており、その中から静止している星を観測する。

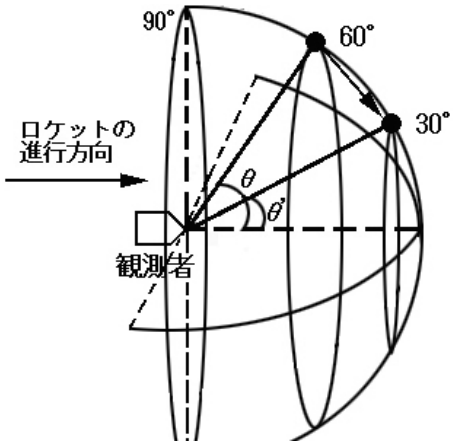

図 5: 観測者から物体を見上げた角度の変位

まず見上げる星が一つの場合を考えてプログラムを作成した. 図 5 のように θ は静止観測者から物体を見上げた角度とし,θ' は動的観測者から物体を見上げた角度とする. 観測者の速度が

光の速度に近づくつれに物体の見上げる角度が小さくなるので, 最終的には, 物体は観測者の真正面に見える. java アプレットで作成したシミュレーション画面が図 6 である.90 °までの見かけの角度を表示するために同心円を描き、前方と左右の視野をまとめて表すように工夫した。このアプレットは静止している観測者から星を見上げたときの角度（すなわち、本来の星の見かけの角度）をテキストフィールドに入力し,「計算」ボタンを押すことでロケットの速度が $0.1c$,$0.2c$,...,$0.9999c$ と増加していき, 各速度での星の見かけの角度の変化を可視化できる. 光速に近い速度から見える景色は観測者の真正面の一点に収束して見えることがわかる.

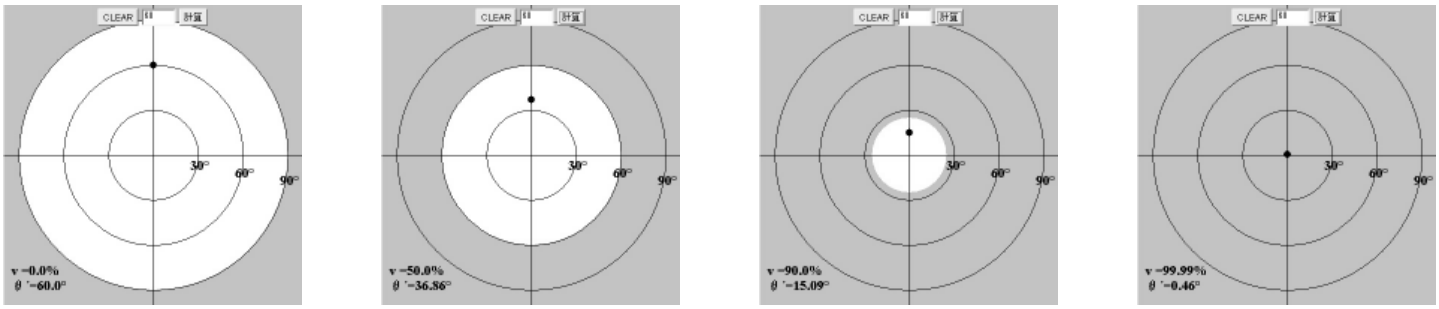

図 6: 光行差可視化プログラム

4.2 見上げる星が複数の場合

今度は見上げる星が複数の場合を考える. 計算方法としては

1. 図示する投影面 (同心円の図) 上での星の座標 (x, y) を決める

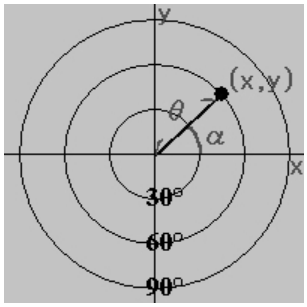

図 7: 方位格 α

2. 1 より静止観測者から見上げた星角度 θ が求まる

3. 求めた角度 θ を式 (33) へ当てはめることにより動的観測者の角度 θ' が導ける

4. 始めに定義した x, y 座標から $\sin\alpha$,$\cos\alpha$ を求める。(角度 α は左右の方向を決める方位角。図 7 参照)

5. $\sin\alpha$,$\cos\alpha$ に θ' を乗算することにより動的観測者から見上げた星の位置が決まる

6. 各速度に対応させていき, 速度が $0.1c$,$0.2c$,...,$0.9999c$ と増加させた時の変化を可視化できるようにする

例としてオリオン座を挙げると図 8 のようになった. 今回はオリオン座なので 7 箇所座標を指定することにより, オリオン座の光行差の可視化プログラムが作成できた. 結果、ロケットの速度 v が光速 c に近づくにつれ, 観測者の真正面に収束する様子がわかる。

なお、θ =90 °を変換した値 θ' は、ロケットから観測者の視野を表すと考えられる。何故なら、観測者が正面を向いて観測できる角度は 0 °から 90 °の範囲であるからである。よってロケットの速度が光速に近づくにつれ、観測者の視界が狭くなる。その様子を Java アプレット内では白円で表し、白円内が観測者の視界であることがわかるようにした。ロケットの速度がほぼ光速になるときには視界は真正面の一点のみになることがわかる。

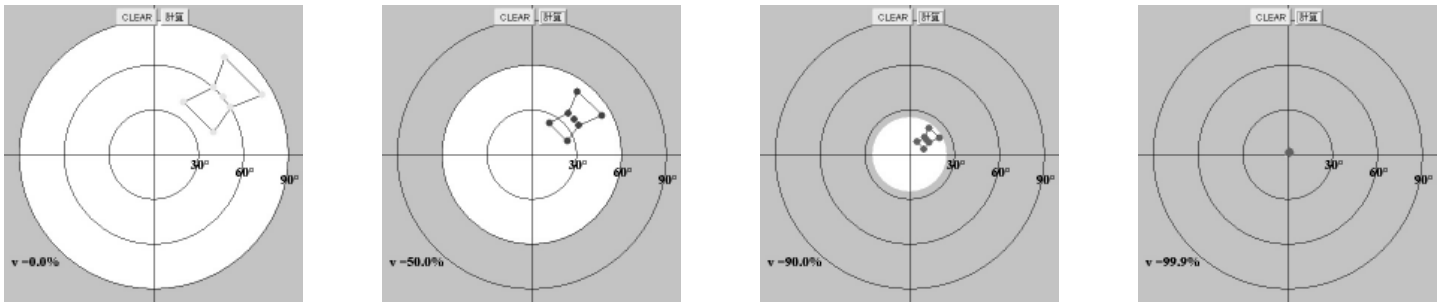
CLEAR
計算
30°
60°
90°
v =0.0%
CLEAR
計算
30°
60°
90°
v =50.0%
CLEAR
計算
30°
60°
90°
v =90.0%
CLEAR
計算
30°
60°
90°
v =99.9%

図 8: オリオン座

5　まとめ

本研究では観測者が光速に近い速度で移動するロケットから静止している星を観測するというモデルで、光行差を考察してきた。その結果、観測者が光速に近い速度で移動しながら星を観測すると、静止している時と比べ観測者から見た星の角度は見かけ上小さくなり、光速とほぼ同じ速度に達したとき、星は観測者の真正面の一点に収束される。またこのことから、光速に近づくにつれに観測者の視野も狭くなり最終的に真正面の一点のみが視野となることがわかった。この事から、「不思議宇宙のトムキンス」の物語で描かれている世界は景色・人が細く見えることは正しいと言える。

本研究では詳しく考察しなかったが、観測者から見える色もまた光速に近い速度で移動することによって、静止している時と比べると異なった色が見える。これは光のドップラー効果が影響するものであり、星からの波長が短くなるため色も変化していく。本研究の Java アプレット (オリオン座) では、星の初期の色を黄色とし、速度が光速に近づくと、黄色→緑→青→藍→紫と変化していくよう作成している。しかし、この色の変化は正確なものではないため今後の課題となる。

また、色だけでなく星の明るさも光速で移動することで変わってくると考えられる。なぜなら、見かけの角度が小さくなることにより星の面積も見かけ上小さくなっているので、星から放出されるエネルギーが同じと考えるのなら、面積が小さくなるにつれ発せられる光が強くなると考えられるからである。静止観測者から見た星の面積を S_1、動的観測者から見た星の面積を S_2、R_1 を静的観測者から星を見上げた角度、R_1 を動的観測者から星を見上げた角度、微小量 Δr を星の直径とする。(図 9)

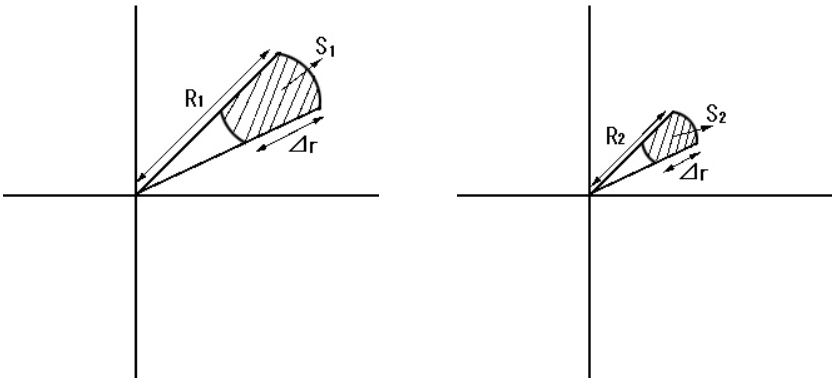

図 9: 面積比

このとき静的観測者から見た星と動的観測者から見た星の面積比を考える。
まず、静的観測者から見た星の面積 S_1 を考えると

$$\begin{aligned} S_1 &= \pi {R_1}^2 - \pi(R_1 - \Delta r)^2 \\ &= \pi {R_1}^2 - \pi({R_1}^2 - 2R_1\Delta r + (\Delta r)^2) \\ &\cong 2\pi R_1 \Delta r \end{aligned} \tag{35}$$

となる。同様に動的観測者から見た星の面積 S_2 は

$$\begin{aligned} S_2 &= \pi {R_2}^2 - \pi(R_2 - \Delta r)^2 \\ &= \pi {R_2}^2 - \pi({R_2}^2 - 2R_2\Delta r + (\Delta r)^2) \\ &\cong 2\pi R_2 \Delta r \end{aligned} \tag{36}$$

となる。よって面積比は

$$\frac{2\pi R_1 \Delta r}{2\pi R_2 \Delta r} = \frac{R_2}{R_1} \tag{37}$$

となるので星の明るさは、観測者が光速に近づくつれに R_1/R_2 倍になると考えられる。
準光速で飛行するロケットの実現は遠い未来のことかもしれないが、本研究で取り上げた光行差・ドップラー効果や星の明るさの要素を考え、シミュレーションを行うことで準光速で動くときの現象を正確に予測でき、準光速の世界を擬似体験することが可能である。

参考文献

[1] ランダウ, リフシッツ, 場の古典論＝電気力学, 特殊および一般相対性理論=, 東京図書株式会社 (1978)

[2] 佐藤勝彦，相対性理論（岩波基礎物理シリーズ９），岩波書店 (1996)

[3] ジョージ・ガモフ/ラッセル・スタナード, 不思議宇宙のトムキンス, 白揚社 (2001)

A　プログラムソース

(見上げる星が一つの場合)

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class jt extends Applet
                 implements ActionListener{
 Button clear,keisan;  //変数の定義
 TextField kakudo;
 Font font;
 Point p1 = new Point(200,200);
 double PI = 3.141592654;
 double c = 300000.0;
 double b,b1,x,y,z,w,n,n1,s,s1,R,R1,V1;
 double V = 0;
 int i = 0;
 int h;
 int R2;

 public void init(){
  font = new Font("Serif", Font.BOLD,18);

  clear = new Button("CLEAR");  //ボタン・テキストフィールドの定義
  add(clear);
  clear.addActionListener(this);

  kakudo = new TextField("",3);
  add(kakudo);

  keisan = new Button("計算");
  add(keisan);
  keisan.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e){
  int kk = Integer.parseInt(kakudo.getText());

   if(e.getSource() == keisan){
   if(kk > 90 && kk < 270){    // 入力角度が観測者が後ろを向いた（90 °～270 °）の
時の処理
    if(i == 0){
     R = kk;

     if(R >90 && R < 180){
     R = 180 - R;
     }else
      if(R > 180 && R < 270){
     R = 180 - R;
     }

     p1.setLocation(200,200-(2*R));

     V = 0;
     b = (((double)R)/180) * PI;
     b1 = (((double)R1)/180)* PI;
     i++;

  }else
   if(i < 10){
     V = V + 10;
     V1 = (double)(V/100) * c;
     x = (V1 * V1);
     y = (c * c);
     z = x / y;
     w = Math.sqrt(1-z);
```

```
    n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
    n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
    s = Math.atan(n);
    s1 = Math.atan(n1);
    R = (180*s)/PI;
    R1 = (180*s1)/PI;
    p1.setLocation(200,200-(2*R));
    i++;
 }else
  if(i == 10){
   V = 99.9;
   V1 = (double)(V/100) * c;
   x = (V1 * V1);
   y = (c * c);
   z = x / y;
   w = Math.sqrt(1-z);
   n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
   n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
   s = Math.atan(n);
   s1 = Math.atan(n1);
   R = (180*s)/PI;
   R1 = (180*s1)/PI;
   p1.setLocation(200,200-(2*R));
   i++;

 }else
  if(i == 11){
   V = 99.99;
   V1 = (double)(V/100) * c;
   x = (V1 * V1);
   y = (c * c);
   z = x / y;
   w = Math.sqrt(1-z);
   n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
   n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
   s = Math.atan(n);
   s1 = Math.atan(n1);
   R = (180*s)/PI;
   R1 = (180*s1)/PI;
   p1.setLocation(200,200-R);
 }
}else{        //観測者が前を向いた(0 °～90 °と 270 °～360 °)時の処理
   if(i == 0){
    R = kk;
   if(R > 270 && R < 360){
    R = R - 360;
    }
    p1.setLocation(200,200-(2*R));
    b = (((double)R)/180) * PI;
    b1 = (((double)R1)/180)* PI;
    V = 0;
    i++;

 }else
```

```
    if(i < 10){
      V = V + 10;
      V1 = (double)(V/100) * c;
      x = (V1 * V1);
      y = (c * c);
      z = x / y;
      w = Math.sqrt(1-z);
      n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
      n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
      s = Math.atan(n);
      s1 = Math.atan(n1);
      R = (180*s)/PI;
      R1 = (180*s1)/PI;
      p1.setLocation(200,200-(2*R));
      i++;
    }else
     if(i == 10){
      V = 99.9;
      V1 = (double)(V/100) * c;
      x = (V1 * V1);
      y = (c * c);
      z = x / y;
      w = Math.sqrt(1-z);
      n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
      n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
      s = Math.atan(n);
      s1 = Math.atan(n1);
      R = (180*s)/PI;
      R1 = (180*s1)/PI;
      p1.setLocation(200,200-(2*R));
      i++;

    }else
     if(i == 11){
      V = 99.99;
      V1 = (double)(V/100) * c;
      x = (V1 * V1);
      y = (c * c);
      z = x / y;
      w = Math.sqrt(1-z);
      n = w*(Math.sin(b))/((Math.cos(b))+(V1/c));
      n1 = w*(Math.sin(b1))/((Math.cos(b1))+(V1/c));
      s = Math.atan(n);
      s1 = Math.atan(n1);
      R = (180*s)/PI;
      R1 = (180*s1)/PI;
      p1.setLocation(200,200-(2*R));
    }
   }
}

 if(e.getSource() == clear){  //clear ボタンを押したときの処理
    R = kk;
    R1 = 90;
    if(R >90 && R <= 180){
      R = 180 - R;
```

```
      }else
       if(R > 180 && R <= 270){
      R = 180 - R;
      }else
       if(R > 270 && R <= 360){
      R = R - 360;
      }
    V = 0;
    i = 0;
    p1.setLocation(200,200-(2*R));
   }
  repaint();
}

  public void paint(Graphics g){  //描画処理
  g.setColor(Color.white);        //視野の範囲を表す白円を表示
  h = (int)R1;
  g.fillOval(200-(2*h),200-(2*h),(4*h),(4*h));
  g.setColor(Color.black);  //同心円を表示
  g.drawLine(200,0,200,400);
  g.drawLine(0,200,400,200);
  g.drawOval(20,20,360,360);
  g.drawOval(80,80,240,240);
  g.drawOval(140,140,120,120);
  g.fillOval(p1.x - 5 , p1.y - 5,10,10);  //観測する星を表示
  g.setFont(font);
  g.drawString("v ="+V+"%",10,360);       //観測者の速度
  R = R*100;
  R2 = (int)R;
  R =(double)R2/100;
  g.drawString("θ ´="+R+" °",10,380);  //動的観測者から見える星の角度
  g.drawString("30 °",250,220);
  g.drawString("60 °",310,230);
  g.drawString("90 °",370,240);
 }
}
```

(見上げる星が複数の場合 (オリオン座))

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class orion2 extends Applet
                implements ActionListener{
 Button clear,keisan;                          //変数の定義
 Font font;
 double b,bb1,d,j,f,R1,V1;
 double g,g1,g2,g3,g4,g5,g6,g7;
 double h,h1,h2,h3,h4,h5,h6,h7;
 double s,s1,s2,s3,s4,s5,s6,s7;
 double x2,x4,x6,x8,x10,x12,x14;
 double y2,y4,y6,y8,y10,y12,y14;
 double PI = 3.141592654;
 double c = 300000.0;
 double V = 0;
 int i = 0;
 int R = 255;
 int G = 255;
 int B = 0;
 int R2;

  int a1 = 280;                              //投影面での星の x,y 座標 (初期値) を定義
  int b1 = 170;
  double x1 = a1-200;
  double y1 = 200-b1;
  double z1 = Math.sqrt((x1*x1)+(y1*y1));   //三平方の定理より観測者から星を見た
  double o1 = z1/2;                          角度 ( θ ) を求める
  double n = (o1*PI)/180;                   //θをラジアンへ直す
  double q = Math.atan(y1/x1);              //方位角αを求める
  Point p1 = new Point(a1,b1);              //星を描く
  int a2 = 240;
  int b2 = 130;
  double x3 = a2-200;
  double y3 = 200-b2;
  double z2 = Math.sqrt((x3*x3)+(y3*y3));
  double o2 = z2/2;
  double n1 = (o2*PI)/180;
  double q1 = Math.atan(y3/x3);
  Point p2 = new Point(a2,b2);
  int a3 = 303;
  int b3 = 137;
  double x5 = a3-200;
  double y5 = 200-b3;
  double z3 = Math.sqrt((x5*x5)+(y5*y5));
  double o3 = z3/2;
  double n2 = (o3*PI)/180;
  double q2 = Math.atan(y5/x5);
  Point p3 = new Point(a3,b3);
  int a4 = 280;
  int b4 = 110;
  double x7 = a4-200;
  double y7 = 200-b4;
  double z4 = Math.sqrt((x7*x7)+(y7*y7));
  double o4 = z4/2;
```

```
 double n3 = (o4*PI)/180;
 double q3 = Math.atan(y7/x7);

 Point p4 = new Point(a4,b4);

 int a5 = 295;
 int b5 = 70;
 double x9 = a5-200;
 double y9 = 200-b5;
 double z5 = Math.sqrt((x9*x9)+(y9*y9));
 double o5 = z5/2;

 double n4 = (o5*PI)/180;
 double q4 = Math.atan(y9/x9);

 Point p5 = new Point(a5,b5);

 int a6 = 345;
 int b6 = 120;
 double x11 = a6-200;
 double y11 = 200-b6;
 double z6 = Math.sqrt((x11*x11)+(y11*y11));
 double o6 = z6/2;

 double n5 = (o6*PI)/180;
 double q5 = Math.atan(y11/x11);

 Point p6 = new Point(a6,b6);

 int a7 = 293;
 int b7 = 123;
 double x13 = a7-200;
 double y13 = 200-b7;
 double z7 = Math.sqrt((x13*x13)+(y13*y13));
 double o7 = z7/2;

 double n6 = (o7*PI)/180;
 double q6 = Math.atan(y13/x13);

 Point p7 = new Point(a7,b7);

public void init(){
 font = new Font("Serif", Font.BOLD,18);

 clear = new Button("CLEAR");          //ボタン・テキストフィールドの定義
 add(clear);
 clear.addActionListener(this);

 keisan = new Button("計算");
 add(keisan);
 keisan.addActionListener(this);
}

public void actionPerformed(ActionEvent e){
   if(e.getSource() == keisan){
    if(i == 0){
     R1 = 90;
     bb1 = (((double)R1)/180)* PI;
     p1.setLocation(a1,b1);              //星の初期値
     p2.setLocation(a2,b2);
     p3.setLocation(a3,b3);
     p4.setLocation(a4,b4);
     p5.setLocation(a5,b5);
     p6.setLocation(a6,b6);
     p7.setLocation(a7,b7);

     i++;
    }else
    if(i > 0 && i < 10){        //観測者の速度Vが0〜90%までの処理
```

```
V = V + 10;
V1 = (double)(V/100) * c;
b = (V1 * V1);
d = (c * c);
j = b / d;
f = Math.sqrt(1-j);

 g = f*(Math.sin(n))/((Math.cos(n))+(V1/c));
g1 = f*(Math.sin(n1))/((Math.cos(n1))+(V1/c));
g2 = f*(Math.sin(n2))/((Math.cos(n2))+(V1/c));
g3 = f*(Math.sin(n3))/((Math.cos(n3))+(V1/c));
g4 = f*(Math.sin(n4))/((Math.cos(n4))+(V1/c));
g5 = f*(Math.sin(n5))/((Math.cos(n5))+(V1/c));
g6 = f*(Math.sin(n6))/((Math.cos(n6))+(V1/c));
g7 = f*(Math.sin(bb1))/((Math.cos(bb1))+(V1/c));

 h = Math.atan(g);
h1 = Math.atan(g1);
h2 = Math.atan(g2);
h3 = Math.atan(g3);
h4 = Math.atan(g4);
h5 = Math.atan(g5);
h6 = Math.atan(g6);
h7 = Math.atan(g7);

 s = (180*h)/PI;
s1 = (180*h1)/PI;
s2 = (180*h2)/PI;
s3 = (180*h3)/PI;
s4 = (180*h4)/PI;
s5 = (180*h5)/PI;
s6 = (180*h6)/PI;
s7 = (180*h7)/PI;

R1 = s7;

x2 = (2*s)*Math.cos(q);          //角度θ，を方位角αを乗算して
y2 = (2*s)*Math.sin(q);             動的観測者から見える位置に直す

x4 = (2*s1)*Math.cos(q1);
y4 = (2*s1)*Math.sin(q1);

x6 = (2*s2)*Math.cos(q2);
y6 = (2*s2)*Math.sin(q2);

x8 = (2*s3)*Math.cos(q3);
y8 = (2*s3)*Math.sin(q3);

x10 = (2*s4)*Math.cos(q4);
y10 = (2*s4)*Math.sin(q4);

x12 = (2*s5)*Math.cos(q5);
y12 = (2*s5)*Math.sin(q5);

x14 = (2*s6)*Math.cos(q6);
y14 = (2*s6)*Math.sin(q6);

p1.setLocation(200+x2,200-y2);
p2.setLocation(200+x4,200-y4);
p3.setLocation(200+x6,200-y6);
p4.setLocation(200+x8,200-y8);
p5.setLocation(200+x10,200-y10);
p6.setLocation(200+x12,200-y12);
p7.setLocation(200+x14,200-y14);
```

```
  i++;
}else
 if(i == 10){                          //速度 V が光速の 99%の場合も同様に処理を
  V = 99.0;                              行う
  V1 = (double)(V/100) * c;
  b = (V1 * V1);
  d = (c * c);
  j = b / d;
  f = Math.sqrt(1-j);
   g = f*(Math.sin(n))/((Math.cos(n))+(V1/c));
  g1 = f*(Math.sin(n1))/((Math.cos(n1))+(V1/c));
  g2 = f*(Math.sin(n2))/((Math.cos(n2))+(V1/c));
  g3 = f*(Math.sin(n3))/((Math.cos(n3))+(V1/c));
  g4 = f*(Math.sin(n4))/((Math.cos(n4))+(V1/c));
  g5 = f*(Math.sin(n5))/((Math.cos(n5))+(V1/c));
  g6 = f*(Math.sin(n6))/((Math.cos(n6))+(V1/c));
  g7 = f*(Math.sin(bb1))/((Math.cos(bb1))+(V1/c));
   h = Math.atan(g);
  h1 = Math.atan(g1);
  h2 = Math.atan(g2);
  h3 = Math.atan(g3);
  h4 = Math.atan(g4);
  h5 = Math.atan(g5);
  h6 = Math.atan(g6);
  h7 = Math.atan(g7);
   s = (180*h)/PI;
  s1 = (180*h1)/PI;
  s2 = (180*h2)/PI;
  s3 = (180*h3)/PI;
  s4 = (180*h4)/PI;
  s5 = (180*h5)/PI;
  s6 = (180*h6)/PI;
  s7 = (180*h7)/PI;
  R1 = s7;

  x2 = (2*s)*Math.cos(q);
  y2 = (2*s)*Math.sin(q);
  x4 = (2*s1)*Math.cos(q1);
  y4 = (2*s1)*Math.sin(q1);
  x6 = (2*s2)*Math.cos(q2);
  y6 = (2*s2)*Math.sin(q2);
  x8 = (2*s3)*Math.cos(q3);
  y8 = (2*s3)*Math.sin(q3);
  x10 = (2*s4)*Math.cos(q4);
  y10 = (2*s4)*Math.sin(q4);
  x12 = (2*s5)*Math.cos(q5);
  y12 = (2*s5)*Math.sin(q5);
  x14 = (2*s6)*Math.cos(q6);
  y14 = (2*s6)*Math.sin(q6);

  p1.setLocation(200+x2,200-y2);
  p2.setLocation(200+x4,200-y4);
  p3.setLocation(200+x6,200-y6);
```

```
  p4.setLocation(200+x8,200-y8);
  p5.setLocation(200+x10,200-y10);
  p6.setLocation(200+x12,200-y12);
  p7.setLocation(200+x14,200-y14);

  i++;
}else
 if(i ==11){
  V = 99.9;
  V1 = (double)(V/100) * c;
  b = (V1 * V1);
  d = (c * c);
  j = b / d;
  f = Math.sqrt(1-j);
   g = f*(Math.sin(n))/((Math.cos(n))+(V1/c));
  g1 = f*(Math.sin(n1))/((Math.cos(n1))+(V1/c));
  g2 = f*(Math.sin(n2))/((Math.cos(n2))+(V1/c));
  g3 = f*(Math.sin(n3))/((Math.cos(n3))+(V1/c));
  g4 = f*(Math.sin(n4))/((Math.cos(n4))+(V1/c));
  g5 = f*(Math.sin(n5))/((Math.cos(n5))+(V1/c));
  g6 = f*(Math.sin(n6))/((Math.cos(n6))+(V1/c));
  g7 = f*(Math.sin(bb1))/((Math.cos(bb1))+(V1/c));
   h = Math.atan(g);
  h1 = Math.atan(g1);
  h2 = Math.atan(g2);
  h3 = Math.atan(g3);
  h4 = Math.atan(g4);
  h5 = Math.atan(g5);
  h6 = Math.atan(g6);
  h7 = Math.atan(g7);
   s = (180*h)/PI;
  s1 = (180*h1)/PI;
  s2 = (180*h2)/PI;
  s3 = (180*h3)/PI;
  s4 = (180*h4)/PI;
  s5 = (180*h5)/PI;
  s6 = (180*h6)/PI;
  s7 = (180*h7)/PI;
  R1 = s7;

  x2 = (2*s)*Math.cos(q);
  y2 = (2*s)*Math.sin(q);
  x4 = (2*s1)*Math.cos(q1);
  y4 = (2*s1)*Math.sin(q1);
  x6 = (2*s2)*Math.cos(q2);
  y6 = (2*s2)*Math.sin(q2);
  x8 = (2*s3)*Math.cos(q3);
  y8 = (2*s3)*Math.sin(q3);
  x10 = (2*s4)*Math.cos(q4);
  y10 = (2*s4)*Math.sin(q4);
  x12 = (2*s5)*Math.cos(q5);
  y12 = (2*s5)*Math.sin(q5);
  x14 = (2*s6)*Math.cos(q6);
  y14 = (2*s6)*Math.sin(q6);
```

```
   p1.setLocation(200+x2,200-y2);
   p2.setLocation(200+x4,200-y4);
   p3.setLocation(200+x6,200-y6);
   p4.setLocation(200+x8,200-y8);
   p5.setLocation(200+x10,200-y10);
   p6.setLocation(200+x12,200-y12);
   p7.setLocation(200+x14,200-y14);

   i++;
}else
 if(i == 12){                          //速度 V が光速の 99.99%の場合も同様な処理
   V = 99.99;
   V1 = (double)(V/100) * c;
   b = (V1 * V1);
   d = (c * c);
   j = b / d;
   f = Math.sqrt(1-j);

    g = f*(Math.sin(n))/((Math.cos(n))+(V1/c));
   g1 = f*(Math.sin(n1))/((Math.cos(n1))+(V1/c));
   g2 = f*(Math.sin(n2))/((Math.cos(n2))+(V1/c));
   g3 = f*(Math.sin(n3))/((Math.cos(n3))+(V1/c));
   g4 = f*(Math.sin(n4))/((Math.cos(n4))+(V1/c));
   g5 = f*(Math.sin(n5))/((Math.cos(n5))+(V1/c));
   g6 = f*(Math.sin(n6))/((Math.cos(n6))+(V1/c));
   g7 = f*(Math.sin(bb1))/((Math.cos(bb1))+(V1/c));

    h = Math.atan(g);
   h1 = Math.atan(g1);
   h2 = Math.atan(g2);
   h3 = Math.atan(g3);
   h4 = Math.atan(g4);
   h5 = Math.atan(g5);
   h6 = Math.atan(g6);
   h7 = Math.atan(g7);

    s = (180*h)/PI;
   s1 = (180*h1)/PI;
   s2 = (180*h2)/PI;
   s3 = (180*h3)/PI;
   s4 = (180*h4)/PI;
   s5 = (180*h5)/PI;
   s6 = (180*h6)/PI;
   s7 = (180*h7)/PI;

   R1 = s7;


   x2 = (2*s)*Math.cos(q);
   y2 = (2*s)*Math.sin(q);

   x4 = (2*s1)*Math.cos(q1);
   y4 = (2*s1)*Math.sin(q1);

   x6 = (2*s2)*Math.cos(q2);
   y6 = (2*s2)*Math.sin(q2);

   x8 = (2*s3)*Math.cos(q3);
   y8 = (2*s3)*Math.sin(q3);

   x10 = (2*s4)*Math.cos(q4);
   y10 = (2*s4)*Math.sin(q4);

   x12 = (2*s5)*Math.cos(q5);
```

```
      y12 = (2*s5)*Math.sin(q5);
      x14 = (2*s6)*Math.cos(q6);
      y14 = (2*s6)*Math.sin(q6);

      p1.setLocation(200+x2,200-y2);
      p2.setLocation(200+x4,200-y4);
      p3.setLocation(200+x6,200-y6);
      p4.setLocation(200+x8,200-y8);
      p5.setLocation(200+x10,200-y10);
      p6.setLocation(200+x12,200-y12);
      p7.setLocation(200+x14,200-y14);
    }
   }
 if(e.getSource() == clear){                    //clear ボタンを押した時の処理
   V = 0;
   i = 0;
   R1 = 90;
   R = 255;
   G = 255;
   B = 0;
   x2  = a1;  y2  = b1;
   x4  = a2;  y4  = b2;
   x6  = a3;  y6  = b3;
   x8  = a4;  y8  = b4;
   x10 = a5;  y10 = b5;
   x12 = a6;  y12 = b6;
   x14 = a7;  y14 = b7;
   p1.setLocation(a1,b1);
   p2.setLocation(a2,b2);
   p3.setLocation(a3,b3);
   p4.setLocation(a4,b4);
   p5.setLocation(a5,b5);
   p6.setLocation(a6,b6);
   p7.setLocation(a7,b7);
  }
 repaint();
}
public void paint(Graphics g){      //描画処理
  g.setColor(Color.white);          //視野の範囲を表す白円を表示
  R2 = (int)R1;
  g.fillOval(200-(2*R2),200-(2*R2),(4*R2),(4*R2));
  g.setColor(Color.black);
  g.drawLine(200,0,200,400);
  g.drawLine(0,200,400,200);
  if(V == 0){                       //星と星を結ぶ線を表示
   g.drawLine(a1,b1,a2,b2);
   g.drawLine(a1,b1,a3,b3);
   g.drawLine(a2,b2,a4,b4);
   g.drawLine(a4,b4,a5,b5);
   g.drawLine(a5,b5,a6,b6);
   g.drawLine(a6,b6,a3,b3);
  }else{
   g.drawLine((int)(200+x2),(int)(200-y2),(int)(200+x4),(int)(200-y4));
   g.drawLine((int)(200+x4),(int)(200-y4),(int)(200+x8),(int)(200-y8));
   g.drawLine((int)(200+x8),(int)(200-y8),(int)(200+x10),(int)(200-y10));
   g.drawLine((int)(200+x10),(int)(200-y10),(int)(200+x12),(int)(200-y12));
```

```
   g.drawLine((int)(200+x12),(int)(200-y12),(int)(200+x6),(int)(200-y6));
   g.drawLine((int)(200+x6),(int)(200-y6),(int)(200+x2),(int)(200-y2));
  }
  g.drawOval(20,20,360,360);          //同心円を表示
  g.drawOval(80,80,240,240);
  g.drawOval(140,140,120,120);
  g.setFont(font);
  g.drawString("v ="+V+"%",10,350);
  g.drawString("30 °",250,220);
  g.drawString("60 °",310,230);
  g.drawString("90 °",370,240);
  if(i == 0){                         //速度に応じて星から見える色の変化
   g.setColor(new Color(R,G,B));
   }else
    if(i > 0 && i < 3){
     R = R - 85;
     g.setColor(new Color(R,G,B));
  }else
   if(i >= 3 && i < 6){
     G = G - 85;
     B = B + 85;
     g.setColor(new Color(R,G,B));
   }else
    if(i >= 6 && i < 8){
     R = R + 64;
     g.setColor(new Color(R,G,B));
   }else
    if(i >= 8){
    R = 255;
    g.setColor(new Color(R,G,B));
   }
  g.fillOval(p1.x - 5 , p1.y - 5,10,10);       //観測する星を表示
  g.fillOval(p2.x - 5 , p2.y - 5,10,10);
  g.fillOval(p3.x - 5 , p3.y - 5,10,10);
  g.fillOval(p4.x - 5 , p4.y - 5,10,10);
  g.fillOval(p5.x - 5 , p5.y - 5,10,10);
  g.fillOval(p6.x - 5 , p6.y - 5,10,10);
  g.fillOval(p7.x - 5 , p7.y - 5,10,10);
 }
}
```