物理シミュレーション音源を用いた音情報による状況推定に関する研究

2017年3月

北海道科学大学大学院

工学研究科

電気工学専攻

氏名　木下　雄太

目　次

第1章　緒言

音情報による状況認知に関する研究は、音環境知覚、音環境理解として既に研究が進められており人間の聴覚機能のモデル化とその応用として成果が報告されている。群ロボットの行動制御情報としての活用も同様に期待されるが、本研究テーマはロボットの学習機能も状況認知に関わるものとしている点に特徴をもち、音発生の状況シミュレーションと学習機能との連携による精度の高い状況認知機能をモデル化することを目的としている。群ロボットによる機能分散は、一個体の多機能化による機能低下を補完する役割を持っており、センサは協調作業の入力情報となり状況の認知に用いられる。物体群と群ロボットの間の状況の推定において物体音源センサ(VOSS:Vibrating Object Sound Sensor)を設定し、複数の物体の振動(物体音源)から伝搬された波が群ロボットのそれぞれの位置で観測され、統合された音情報を提供するセンサして機能することを想定している。状況を推定するためのシステムではコンピュータ上で生成された振動を音源とし、その音源が存在する周囲の状況を仮想空間内で物理設定を施し表現する。この振動と状況を音波形データとして生成するのがVOSS、Sourced Simulation(SS)と Transmitted Simulation(TS)である。VOSSは仮想のセンサでありサンプリングのための指向性を持つ。またこのVOSSは複数設置可能である。SSは振動を生成するためのシミュレーションモデルであり、生成された振動は物体音源として扱われる。物理シミュレーションとAction-Sourced-Transmitted Simulation(ASTS)を結びつけることにより、VOSSから得られる音情報がActionに対応する物理状況設定のマーカー点で聞き取ることで認知が容易となり推定に至るモデルをイメージするに至った。SSで生成できる振動として考えられるのは4種類あり点音源、線音源、面音源、立体音源となる。なお本論文で扱っているのは線音源までである。TSは物理設定を反映した伝搬を再現するシミュレーションモデルである。TSの伝搬は物体音源からVOSSまでの伝搬過程を再現する。本論文では等速直線運動、加速度運動、反射板が存在する状況、複数の反射板が存在する状況についてシミュレーションを行っている。状況推定を行うためにはSS、TS、VOSSを組み合わせてシミュレーションする必要があり、そこから得られる音波形データに含まれている音情報を利用する必要がある。また音情報を利用するためにはその音情報と元になった状況を対応付けて処理する必要がある。その状況の種類や状態によって対応付けられる音情報は異なるため扱う情報量は膨大になる。状況と対応付けられた音情報のペアを集積して管理するためのシステムと情報の整理手法としてランドルト環方式の知識ベースを活用することについても言及している。

第2章　音情報による状況推定概要

ある音を人間が知覚したときにそれは経験的、または直観的に他の音とは違うことを区別することができる。これは音にはその音をその音とする何らかの固有的な音情報を内包しているからだと考えることができる。わかりやすい例は音の高さである。異なる2つ音叉を続けて鳴らした音を知覚した場合、その音の高さが十分に離れていれば人はその違いを判別することができる。これは人の脳は周波数という音情報を参照にしてその2つの音を区別しているからだと考えることができる。また音の大きさも同様である。この2つの音叉から音が発せられたときの音量の差から人は音の大きさを区別することができる。しかし、この時の音量は先の周波数の場合の音情報とは事情が異なっている。それは音量には状況に関する音情報も含まれている可能性があるからだ。この時2つの状況が考えられる。ひとつは2つの音叉を打つ時の力に差がある場合である。もう1つは同じ力で打っているが音叉同士の間隔が、音量が異なるほどの距離を有している場合である。後者の場合は距離という状況に関する情報が音に含まれていると考えられる。

この考え方をさらに進めた場合、同じ音源でもより色濃く状況の音情報が含まれているものがある。例えばグラスが割れるときの音には状況に関する情報が含まれている。グラスを硬い棒で叩いたときの澄んだ音と床に叩きつけた時の割れる音では同じグラスから出ている音色であるが人間の脳は知覚する音情報は明らかに異なっている。もしグラスと陶器のツボを続けて落とした場合、音を聞いた人が注意をひきつけられるのはその音源から出る音色ではなく何かが割れているという音情報である。人間は条件が整っていれば音情報をもとに状況を読み取り、推定することができる。もしこれらの音情報と状況の関係性をうまく対応付けることができた場合、コンピュータにおいても音情報から状況を認知できる可能性がある。wavファイルやmp3ファイルといった音波形データを解析すると周波数や音量など様々な音情報が手に入る。これらの音情報は人間の耳で聞いた時と同様にその音波形データを録音した状況と結びつく場合がある。音情報としてドップラー効果が特徴となって現れている場合を例にとると周波数の変化からその音源(または観測者)がどのような速度で移動しているのか、また周波数の変化と音量の関係から観測地点とどれだけ離れているのかといったことが波形データから得られる音情報で推測できる。つまりその音情報がどのような状況に起因して発生しているものかを明らかにした場合、ある音情報に対しある状況を対応付けるペアを作ることができる。この組を作成する作業を本論文ではペアリングと呼ぶ。

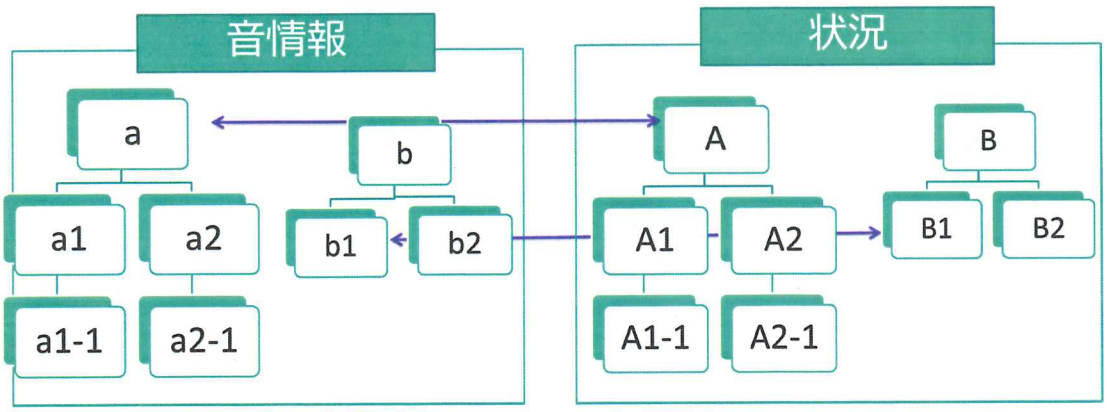

図 2-1.ペアリングのイメージ図

2.1.物理シミュレーションのためのインターフェイス

ペアリングを行うためには状況を表す音波形データと音情報を対にする必要がある。しかし現実世界から得られる音波形データからでは定義がしづらい音情報がノイズとして混じりあっており音情報から状況を正確に定義できない。そのために必要な情報のみを内包している音波形データが必要となる。そこで任意の状況を物理設定したシミュレーションから音波形データを作成する。本論文では後述する振動をシミュレーションする SS と伝搬を再現する TS、また観測者としての役割を果たす仮想のセンサである VOSS の 3 つを利用したシミュレーションにより音波形データを生成している。VOSS は仮想空間内に存在するセンサであり音のセンシングを行う。VOSS は複数台設置可能であり、感度やセンシング範囲を設定できる。このため 2 台の VOSS を同位置に設置することでステレオ音源の作成が可能である。

このときの物理設定を行うために TOOLBOOK を用いたインターフェイスを利用している。TOOLBOOK は Windows アプリケーションを開発するソフトウェア構造体であり、オンライン百科事典や対話型トレーニングアプリケーション、データベースアプリケーションやゲームなどを作成できる。TOOLBOOK は本をメタファーにしてデザインされている。また TOOLBOOK では GUI として画面上に表示されているオブジェクトはプロパティを持っており、そこにスクリプトを記述できる。この特徴を利用してイメージ図として表示されているオブジェクトのパラメータを入力し、そのまま外部のプログラム言語で記述されたプログラムへパラメータを受け渡すことによって直観的に物理設定がで

きるインターフェイスとして利用している。図 2.1-1.はこのインターフェイスの一例である。

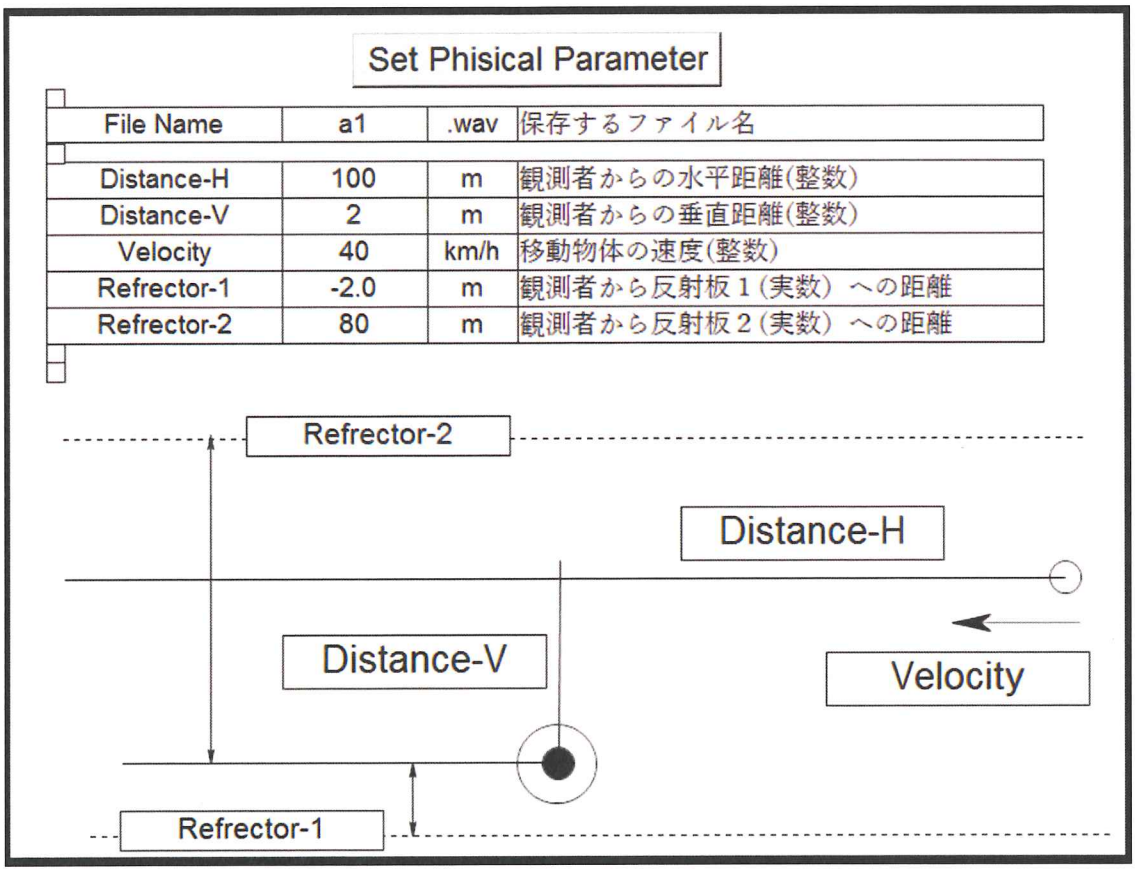

図 2.1-1.TOOLBOOK を利用したインターフェイスの全体図

画面下部は設定している物理状況を表している。この図では Distance-H 上の右端に存在する音源が Velocity のスピードで左方向へ直進する場合を表している。また Distance-V の位置に観測点である VOSS が黒丸として表現されている。この黒丸の外側の円は VOSS がセンシングを行える角度を示しており、この場合は 360 度全方向からの音波を観測している。これらの値は画面上部にある表の値によって決定される。この表の a1 列に任意の値を書き込むことでその値が物理設定として扱われ、最上段にあるボタンを押すと図 2.1-2 のよう a1 の値が物理シミュレーションプログラムの対応した変数に反映される。実行結果が図 2.1-3 のように値が反映されたプログラムとして生成される。この画面上部左の Execute ボタンでシミュレーションが開始され、右の Sound ボタンを実行することでシミュレーションの結果を音源として聞くことができる。

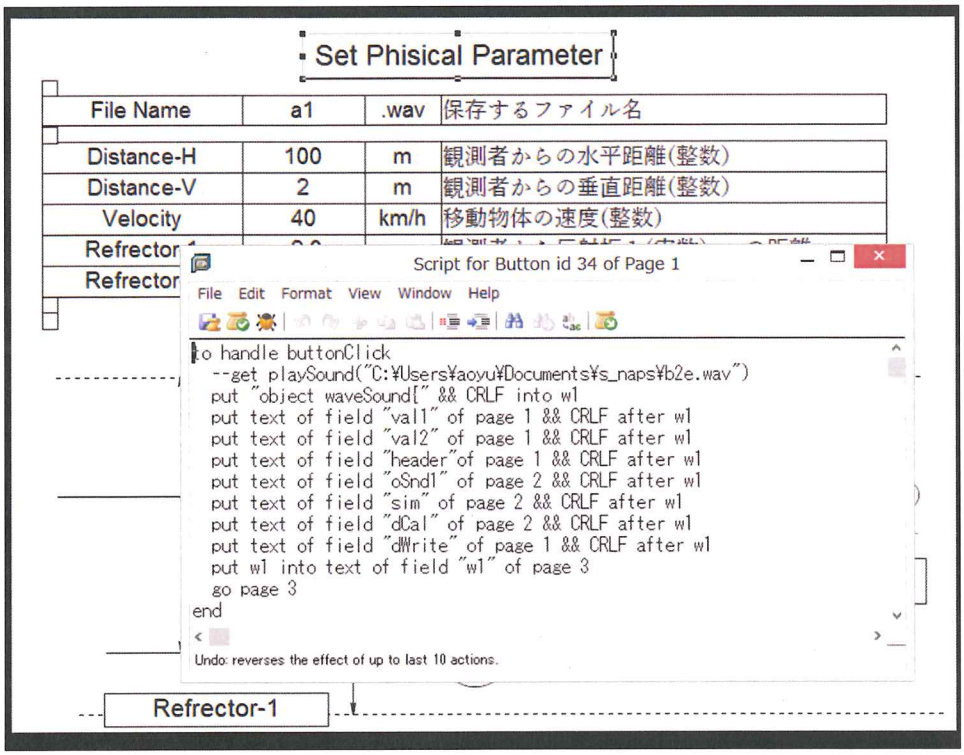

図 2.1-2.オブジェクトに記述したスクリプトの例

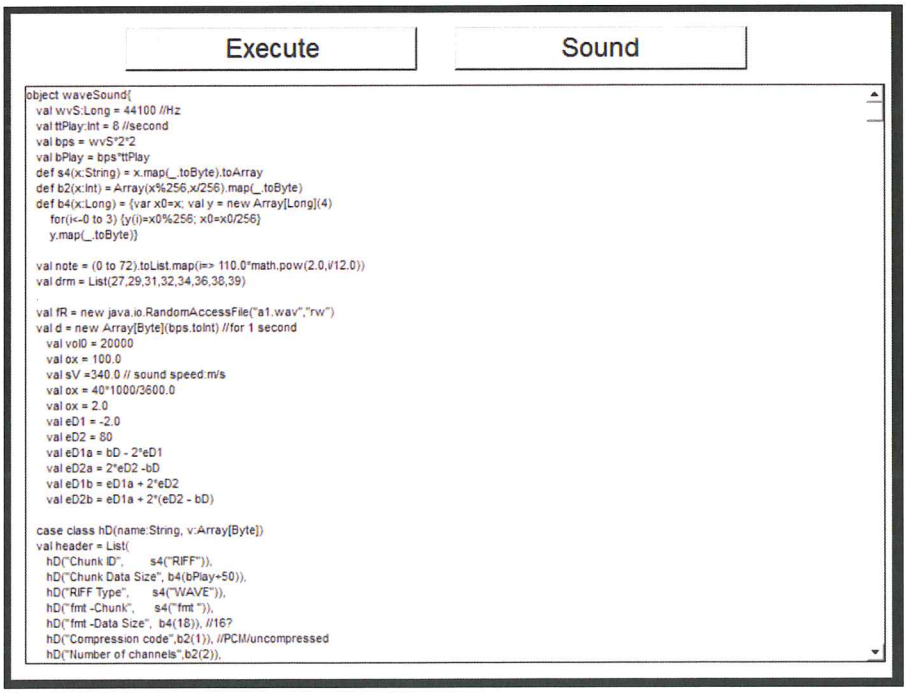

図 2.1-3.物理設定の反映されたプログラム

2.2.ペアリング正規化と状況推定

状況と音情報を組にするペアリングを利用して状況推定を行う場合に考えられる問題の一つとして同じ種類の状況であっても物理状態に差異があるため状況を同定することが困難であるという問題が存在する。例を挙げると音源が移動している状況ではドップラー効果というその状況独自の特徴を観察することができるが移動速度が 50km/h と 100km/h の場合では得られる音情報は異なる。この問題を解決するためには、正規化された音情報を基準として比較することでその音情報の状態を推測することができると考えられる。正規化された音情報を得るための考えられる手法の一つは、周波数特性を利用した「音色」としての正規化が考えられる。もう一つの手法は得られた音情報に統計的なアプローチから処理を施し正規化された音情報を生成するという手法である。

どちらの手法でも汎用的な推定のためには膨大なペアを管理する必要がある。そのため音情報による状況推定のためには知識ベースが必要となってくると本論文は考えている。知識ベースとは知的資産を文書化されたノウハウとして扱うのではなく体系化された知識として扱いそれを蓄積するためのベースであると興膳・喜多・北守(2009)が述べている。そして知識ベースモデルとしてランドルト環方式を検討している。ランドルト環方式は興膳・三上・北守(2010)が提案しているｅラーニング向けのモデルであるがこれを応用することで音情報による状況推定向けの知識ベースとして運用できる可能性がある。ランドルト環とは視力検査で使われる「C」の形に似た切れ目のある記号のことである。視力検査では視力に対応した記号を認識できるかによって視力を測定する。これを模倣してランドルト環方式(のｅラーニング)では理解度に対応した問題を解答できるかによって理解を測定するという方法をとっている。ランドルト環方式において理解の基準となるものはランドルト環の大きさと切れ目の向きである。図 2.2-1 のようにランドルト環の大きさは問題の難易度を表しており、切れ目の向きは問題のカテゴリを表している。問題はこの二つの基準に沿って整理されている。同様にこれを状況推定のための知識ベースに応用するためには、大きさをその音情報の特徴量と対応させ、切れ目の向きを特徴と対応させ、整理することで音情報の状況推定向けの知識ベースとして利用することが出来ると考えている。音情報による状況推定を行うためにはこの知識ベースに蓄積されているペアを参考にして推定したい状況を絞り込んでいく。データベースでは SQL によって管理・操作される関係と同様にランドルト環方式の知識ベースでは AI がランドルト環内にしまわれた知識に対して SQL と同等の役割を持つ。しかし選択した結果が有用である保証がないため、妥当性のある結果を得られる仕組みが必要である。この仕組みを成立するために基礎となる考え方が図 2.2-2 の解

釈・機能生成モデルである。解釈・機能生成モデルは消費者行動を考察するためのシミュレーションモデル作成手法の一つである。このモデルの特徴はK.Weickによって提唱された方法であるイナクトメントとGergernによって提示されたヴィヴィフィケーションの二つの方法を内包したモデルである。北守らは解釈主義的なアプローチを手法であるイナクトメントからモデルを作成し、作成されたモデルからのシミュレーション結果を機能主義的なアプローチであるヴィヴィフィケーションで検証することによりそのモデルの厳密性を判断する。そしてその結果の厳密性が適合していないようであれば、もう一度そのモデルを基にイナクトメントを通じてより適合性の高いモデルへと精緻化することでシミュレーション結果の精度を高めていき、妥当性のある結果を得ることが可能である。この仕組みを、ランドルト環方式を用いた知識ベースを用いた状況推定に当てはめると特徴を参照して知識ベースから選択した結果が適合しているかをAIが判断を行い、その結果がある一定の閾値を超えていない場合は、そのモデルを基に再解釈を行い、モデルを精緻化することでより妥当性のある結果を目指していくこととなる。(図 2.2-3)

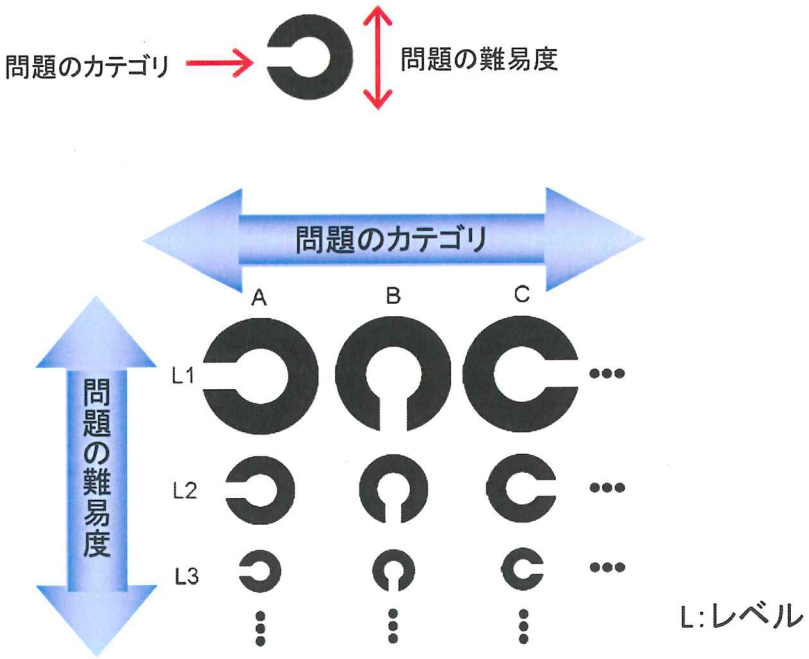

図 2.2-1.ランドルト環方式による問題の整理

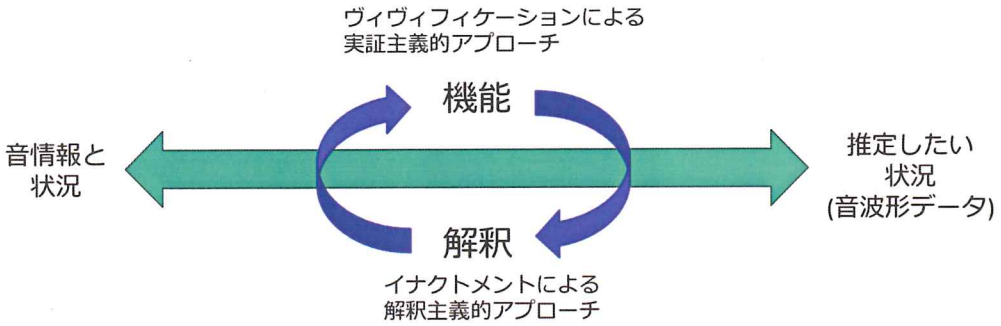

図 2.2-2.解釈・機能生成モデル

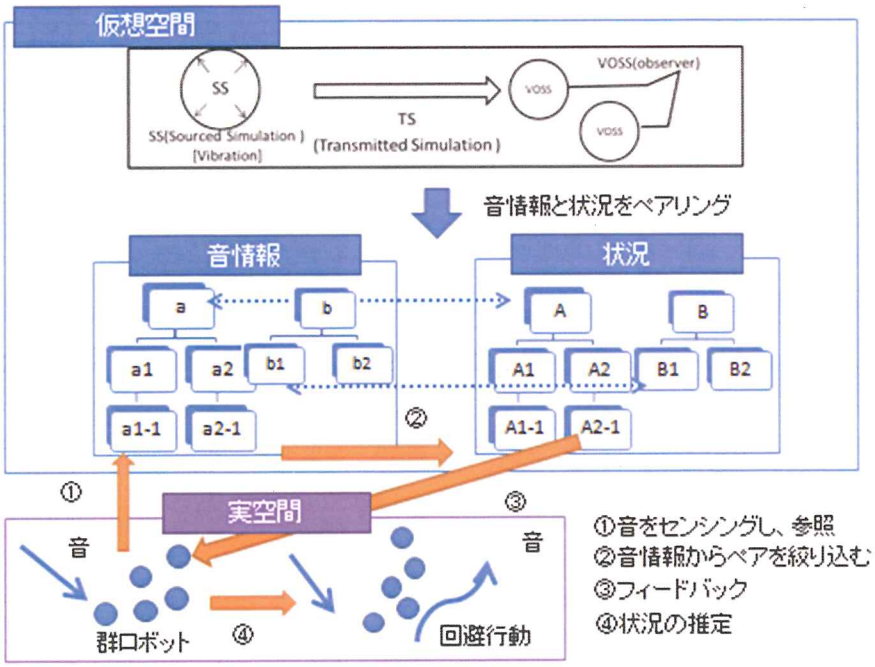

図.2.2-3.群ロボットによる状況推定を例としたイメージ図

第 3 章 状況を特徴づける音情報

本研究で想定している状況とはあらゆる状態に影響を受け、形作られるものである。例えば媒質、音量の大小などの差によって状況は変わってくる。音量の差に注目した場合、その音量差は音源の持つエネルギーに端を発しているものなのか、観測地点との距離に伴った減衰に影響を受けたものなのかは音情報から状況推定をするうえでは重要な点である。また観測者、本論文においては仮想空間内における観測地点の役割を果たす VOSS の設定によっても状況は変化する。例えば VOSS のセンシングを行う角度の設定により波形データから得られる音情報は異なるため、ペアリングの組となる状況も異なる。しかし、同時に状況を推定する上で基礎となる音情報も存在する。この章では状況を形作る音情報について解説を行う。

3.1.音源の振動・伝搬

音情報は音波形データを基にして得ることができる。音波形データを構成するものは振動であり、音も振動を基にした波動現象の一つである。音の特徴的な性質として光や電波などと異なり、気体、液体、固体といった媒質が必要となる。音は振動のエネルギーが媒質を伝わることで成立するため、媒質をバネとおもりの集合体として捉えた場合その動きの伝わり方を音として考えることも出来る。ここでは状況を考察するための音の伝搬から得られる音情報について解説する。

3.2 気体、液体、固体中の音速

最も一般的である空気を媒質とした場合を考えた時、常に約 1000 hPa の大気圧が掛っている。この圧力は変動しない圧力である。ある振動が空気の粒子に接触し、その粒子が大気圧を中心として微細に上下する変動が順次伝わって行き空気中に疎密波が形成される。媒質の動きである音波は粒子速度として、圧力の変動は音圧という形で表現することができる。この時の音を発生させる振動は音源と呼ばれる。音波の伝わる速度は媒質の密度と弾性率によって決まる。媒質が気体である場合の弾性率は気体の変化過程によって決定する。また気体の場合、密度は気温によって変化するため音速も気温の変化に影響される。媒質が液体である場合音速はほとんどの場合 1000~1500m/s の範囲内に収まる。また真水が媒質である場合には断熱変化と等温変化を区別する必要はほとんどない。しかし液体中の音速は気体の時と同様に温度に関係しており温度が上昇

するにつれて、音速も上昇することが確認されている。また真水以外の液体中の音速、例えば海水の場合では塩分、海水温度、深度などに影響を受ける。固体中の音速では固体の温度、構造に影響を受ける。ほとんどの場合その固体の温度が上昇するにつれて音速は減少する。金属から構成されている固体では結晶構造や方向性に音速は影響され、混合物の場合では配合のされる物質や割合、周波数などによって音速は変化する。

表 3.2-1　各種媒質中の音速

(時田 保夫監修 音の環境と制御技術 第Ⅰ巻 基礎技術, 2000, pp.16)

	物質	密度	音速	音響インピーダンス	備考		
		ρ	c	ρc			
気体	空気(0℃)	1.293	331.5	428	1気圧0℃		
	空気(20℃)	1.21	343	415			
	水素	0.09	1269.5	114			
	窒素	1.25055	337	421			
	ヘリウム	0.17847	970	173			
	酸素	1.429	317.2	453			
	一酸化炭素	1.2504	337	421			
	二酸化炭素(低周波)	1.9796	258	512			
	二酸化炭素(高周波)	1.9796	268.6	532			
	水蒸気(100℃)	0.6	404.8	242			
液体	蒸留水	1	1500	1.5	23-27℃		
	海水	1.021	1513	1.54	20℃(塩分30/1000)		
	重水	1.1053	1381	1.53	20℃		
	エチルアルコール	0.786	1207	0.95	23-27℃		
	水銀	13.6	1450	19.8	23-27℃		
固体			縦波	横波	棒の縦波	音響インピーダンス	
		ρ	c1	c2	c3	ρc_1	ρc_2
	アルミニウム	2.96	6420	3040	5000	17.3	8.2
	鉄	7.86	5950	3240	5120	46.4	25.3
	ステンレス鋼(347)	7.91	5790	3100	5000	45.7	24.5
	銅	8.96	5010	2270	3750	44.6	20.2
	鉛	11.34	1960	690	1210	22.4	7.85
	ガラス	2.42	5440			13.2	
	ゴム(天然)	0.97	1500	120	210	1.5	0.12
	ゴム(スチレン-ブタジエン)	1	1760	530		1.76	0.53
	シリコン	2.33	8433	5843		19.64	13.61
	氷	0.917	3230	1600		2.96	1.47
	大理石	2.65	6100	2900	3180	16.2	7.7

[単位]　ρ:10^3kg・m^{-3}、c: m・s^{-1}、ρ c:N・s・m^{-3}

3.3.　音の進行方向

図 3.3-1 は球面波を表した図である。球面波とは均一な媒質中に存在する点音源もしくは波長と比較して充分に小さい音源が存在し、球の中心の一点から点対称に伝搬する波であり、波のエネルギーは距離の 2 乗に反比例する性質を持つ。また、どのような音源でも音源の大きさに対して観測地点と十分に距離がある場合その音源は球面波として取り扱うことができる。また波の形を変えずに平面上に一方向に進行していく波を平面波という。平面波に距離減衰はなく、伝搬中に熱エネルギーに変わらない限り同じ強さで伝搬していく。球面波だとしても音源から十分に距離をとり、観測した場合それはほぼ平面波とみなせる。波の進行状態を部分的に調べる多くの場合は平面波を仮定している。

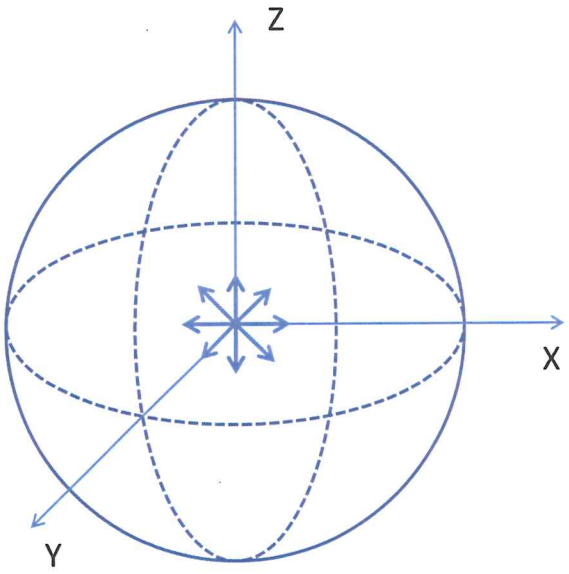

図 3.3-1.　球面波

3.4.　伝搬の際に発生する現象

3.4.1.音の減衰

振動はいつか停止する。この現象は減衰と呼ばれている。減衰が起こる要因は 2 つの事柄から生じる。1 つ目の要因は音が伝搬する際にエネルギーの一部が熱エネルギーに変換され、音エネルギーが減ずるためである。また空気が媒質の場合と比較して熱エネルギーに変換されやすい繊維質な媒質や細い管を伝搬

する場合には音の減衰はより大きくなる。2 つ目の要因は音源から発せられる音の広がり方に関わる。例えば 3.3 で前述した点音源の球面波であり熱エネルギーへの変換を無視した場合、全ての面においてその面の外に向かう音エネルギーはエネルギー保存則によって一定である。音の強さは距離の 2 乗に反比例して減衰することが知られており、この特徴は逆 2 乗則と呼ばれている。

3.4.2.音の反射

ある音が反射した場合やまびこが観測できる。この現象は反射と呼ばれる。反射された音のことを反射音と呼び、1 回の反射ごとにエネルギーが減少するため次第に減衰していく。多重の反射音から現れる現象は残響と呼ばれ、反射音や残響は音質に影響を及ぼす。反射は音の伝わり方を線で表現する音線法を使いあらわすことができる。また反射によって起こる減衰は物質の材質によって変化する。この指標として反射率が用いられる。

$$\text{反射率} = \frac{\text{反射してくる音のエネルギー}}{\text{反射する物質に入る音のエネルギー}} \qquad (3.4.2\text{-}1)$$

また、反射の他に音の一部が反射を行う物質中に吸い取られてしまう現象は吸音と呼ばれ、その吸収される割合を示すために吸音率が使われる、

$$\text{吸音率} = \frac{\text{反射する物質へ吸収された音のエネルギー}}{\text{反射する物質へ入り込んだ音のエネルギー}} \qquad (3.4.2\text{-}2)$$

また反射面が充分に大きく、凹凸が充分に滑らかな場合には鏡面反射という現象が生じる。

3.4.3 音の回折

音波が伝搬するときに障害物が存在する場合、障害物にぶつかり反射するほかに障害物の背後に回り込むことがある。このときの障害物の大きさが波長と

比べ十分に大きくない場合、回折が発生し、障害物背後であっても音が観測できる。

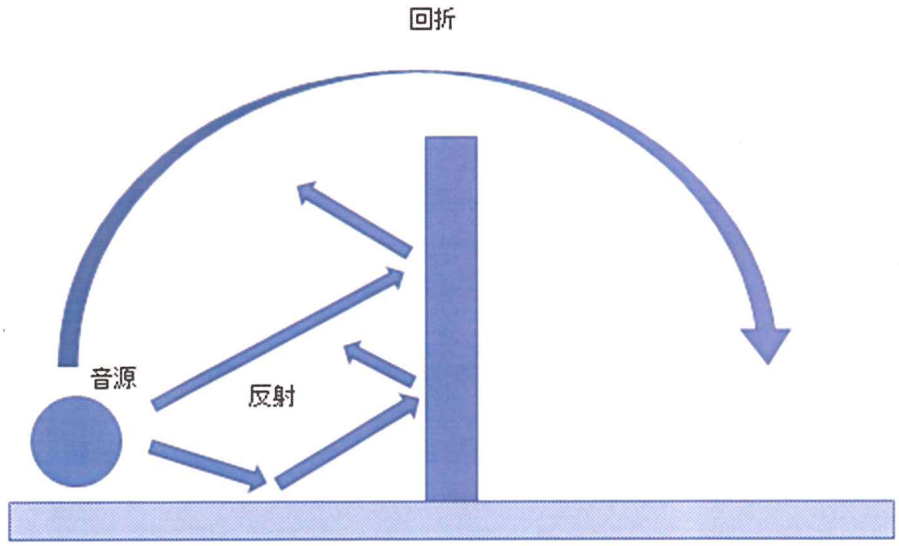

図 3.4.3-1 音の反射と回折

3.4.4 音の屈折

屈折が起こると音の進行方向が変化する。これは伝搬する媒質が伝搬過程で変化した場合にその媒質中の音速が異なるために起こり、密度変化などの条件よりその境界面によって発生しやすい。また同じ媒質であっても媒質の局所的な温度変化が発生している場合は屈折が生じる場合がある。

3.4.5 音の干渉

複数の同一周波数が密接している場合、音波が影響しあう現象であり、お互いが重なり合い、強めたり弱めたりする。例としては戸外で騒音測定を行う際に測定位置によっては壁面や地面の影響を受け、特定の周波数の音圧レベルが変化する場合がある。周波数の差、微細な位相の差が積み重なって合成された場合にはうなりの現象が発生する。また同一の音源から発生した音の場合でも直接音と間接的な音との間には位相差が生じるため、音の干渉が生じる。

3.4.6 移動によって生じるドップラー効果

音源または観測者が移動する場合、音の周波数は変化する。これをドップラー効果と呼ぶ。ドップラー効果は音源が観測地点に向かって移動した場合、観測地点に接近するにつれて周波数が上昇し、遠ざかるにつれて低下して聞こえる現象である。この時音源が移動速度$V_0[m/s]$で移動を行う場合、音源から発せられている、周波数$f_0[Hz]$の音は周波数 f として観測される。

$$f = f_0 \frac{c}{c \pm V_0}[Hz]$$

ただし　c: 音速$\left[\frac{m}{s}\right]$

遠ざかる時は　$c - V$

近づくときは$c + V$

(3.4.6-1)

である。

3.5 生活に関わる音

状況推定を行う上で問題となってくるのは、推定したい音波形データの音環境をどう解釈して取り扱うかである。現実世界では常になんらかの音が生活音として鳴っており一般的に推定したい音波形データ音源の中にもそれがノイズとして残っている。状況推定を行うためにはどのようにして必要な音に焦点を合わせることができるかが重要である。そこで考えられるひとつの手法としてノイズとなりうる可能性が高い状況から出る音を先に明らかにすることによって、状況推定に役立てる音情報を得ることができるのではないかと考えている。また、推定を行いたい状況によってはノイズの類の音を認識したい場合もある。ここではそのひとつとして生活に密着した音について記述する。

集合住宅の住戸内で聞こえる生活音として楽器の音、子供の声または走り回る音、足音、ドアの開閉音、エレベータの作動音、電話の呼び出し音などがある。室内で発生する生活音は基本的に 2 つの種類に分類できる。ひとつは楽器の音や子供の声などは空気を伝わり伝搬する気体伝搬音である。もうひとつは足音や走り回る音は上階の床から下階の天井を伝わって発生する固体伝搬音である。例えばエレベータの作動音も固体伝搬振動であり、大別していくと巻き上げ機械やその他の機械室機器の振動が建物を伝搬して付近の居室へ音を発する。またかごや錘の走行振動も生活音として考えられる。類似した音環境であるオフィスでは呼び出し音やエレベータの作動音に加えてキーボードのタイプ音や会話音、プリンタなどの機器の発生音が加わる。

室外空間においては道路交通騒音や自動車騒音が存在する。自動車騒音はエンジン、変速機、タイヤなどの自動車の構成部品から発生するが何が主要な音源となるかは運転状況の状態によって異なる。タイヤについてはタイヤだけでなく、道路上を走行することで発生しているという見方もできる。自動車による車外騒音の割合は音源寄与率として表されるがこの値は走行状況や自動車の種類によっても変化する。加速走行時の時の騒音音源別寄与率と定常走行時の音源別寄与率の一例は以下の表のようになっている。定常走行時のエンジンの割合は表には示されていないが加速走行時にはエンジンを発生源とする騒音は大きな割合を占めている。二輪車においては自動車とは異なりエンジンが露出しているため、定常走行時でもエンジンを原因とする騒音は大きな割合を占める。またガソリンエンジンよりもディーゼルエンジンのほうがより大きな騒音となることが知られている。

表 3.5-1 加速走行騒音の音源別寄与率

発生源	大型車	中型車	小型車	乗用車	二輪車	原付
エンジン	36.8	47.7	35.8	34.4	27.5	24.5
冷却系	1.6	1.6	3.4	1.9	0	0
呼気系	2.6	3.1	13.0	11.6	17.5	21.2
排気系	22.6	18.6	18.8	23.4	21.4	18.6
駆動系	19.3	14.5	4.7	2.8	0.8	1.1
タイヤ	8.3	13.0	16.1	22.9	7.1	8.1
その他	8.3	1.5	8.2	3	25.7	26.5

表 3.5-2 定常走行騒音の音源別寄与率
(時田 保夫監修 音の環境と制御技術 第Ⅱ巻 応用技術, 1999, pp.333)

発生源	大型車	中型車	小型車	乗用車	二輪車	原付
タイヤ	62.0	52.6	68.1	80.4	15.7	13.1
エンジン等その他	38.0	47.7	31.9	19.6	84.3	86.9

第2章　状況推定のための振動シミュレーションモデル

4.1　音源の振動モデル

この項では、状況推定のためのシミュレーションを行う際に音源の振動となるモデルである、SS(発生源:Sourced Simulation)について説明している。SS は音源となる振動をシミュレーションしている。SS から得られたシミュレーションの結果は音波形データとして出力される。SS から得られる振動は現在 5 種類に分けられる。点音源、線音源、面音源、立体音源と 2 次的な音源となっている。なお本論文で扱う音源は点音源、線音源、2 次的な音源に限る。

表.4.1-1　SS から発生する音源の分類

点音源	球面波として扱われる音源 全ての音源は十分な距離をおいた場合、点音源とみなすことが出来る。
線音源	左右からの張力に影響を受けている音源
面音源	平面が振動している音源
立体音源	面音源が組み合わさり、立体で振動している音源
2 次的な音源	反射によって生じる音波。(反射された地点から生成された新しい音源として考える)

4.2　スプライン関数を用いた線音源の振動方程式

線音源を作成するためには弦の波動方程式を解き、弦の振動を再現する必要があるが一般的に利用されている弦の波動方程式では微小振動であることが前提であるため、最終的に状況推定へ利用することを考慮すると適さない場合がある。そこで微小振動として定義されている$\partial u/\partial x \cong sin\theta$を利用せずに拡張された弦の波動方程式の導出を行った。

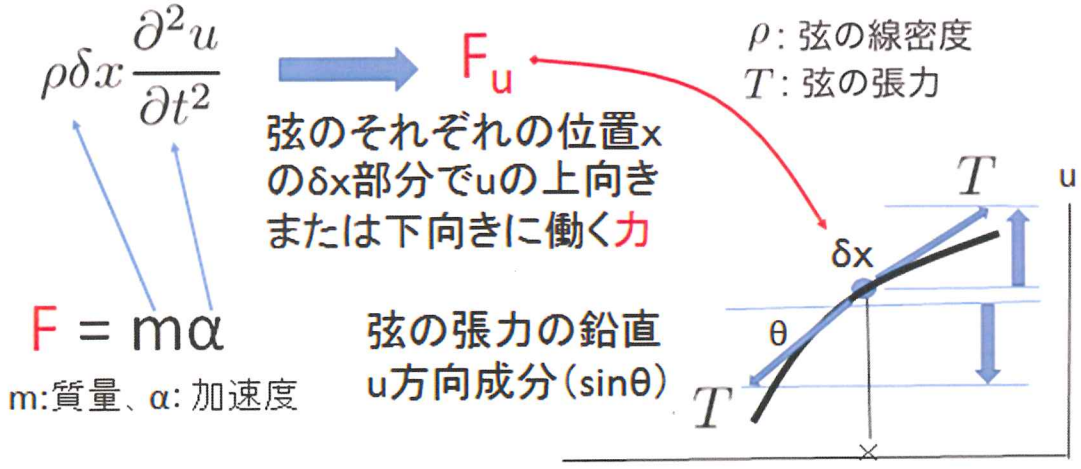

図 4.2-1 波動方程式の導出:鉛直(u)方向

一般的に利用されている弦の波動方程式の導出過程の式である式(4.2-1)において

$$\rho\delta x\frac{\partial^2 u}{\partial t^2} = Tsin\big(\theta(x+\delta x,t)\big) - Tsin(\theta(x,t)) \quad (4.2\text{-}1)$$

$\partial u/\partial x \cong sin\theta$利用せずに$\partial u/\partial x = tan\theta = m$と考える。この時の m は傾きであり、式(4.2-2)と置いた場合、式(4.2-3)と書き表すことができることができる。

$$\cos\theta = \sqrt{\frac{1}{1+m^2}} \quad (4.2\text{-}2)$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{T}{\rho} cos^3\theta \frac{\partial^2 u}{\partial x^2} \quad (4.2\text{-}3)$$

式(4.3-3)から一般的な微小振動である一般的な波動方程式を導く場合
m ≅ 0 or θ ≅ 0の時

$$cos^3\theta \cong 1 (4.2\text{-}4)$$

となりまた

$$\frac{\partial u}{\partial x} = \tan\theta \cong \sin\theta (4.2\text{-}5)$$

だとすると式(4.2-3)に(4.2-4),(4.2-5)が適用されることで式(4.2-6)となり一般的な弦の波動方程式が導出できる。

$$\frac{\partial^2 u}{\partial t^2} = \frac{T}{\rho} \frac{\partial^2 u}{\partial x^2} (4.2\text{-}6)$$

弦の波動方程式では微小振動の場合を想定しているため適応範囲内でしか解が求まらず変形が大きな場合に対応しきれない。しかし拡張された弦の波動方程式では適応範囲部分以外の箇所も扱うことができる。(図 4.2-2)

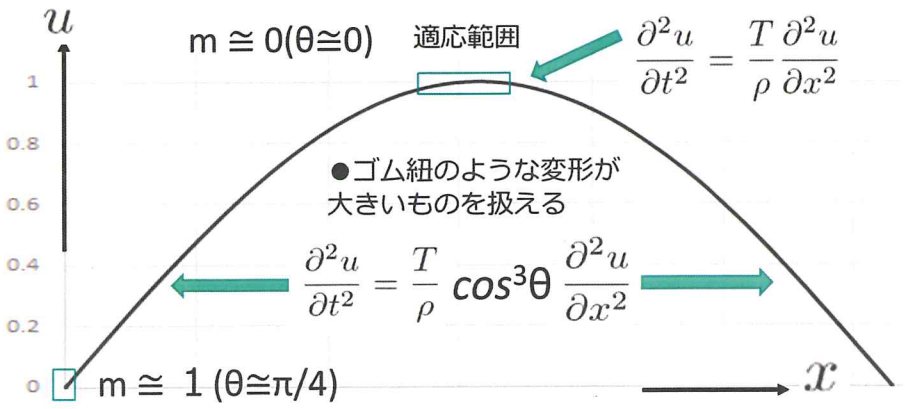

図 4.2-2 t=0 における初期値を正弦波とした場合

しかし偏微分方程式の解を求めようとした時に初期値により解析解が求まらない場合がある。例えばT/σが場所によって変化したときである。これを具体的な状況に当てはめた場合、弾かれたゴムが自身の振動に耐えきれず千切れてしまうことにあたる。このような状態は弦の波動方程式だけでは解を求めることができない。これを解決するためには数値解析からのアプローチが必要になる。しかし、数値解析から解を求める場合にも問題は存在する。ひとつは端数処理によって計算誤差が大きくなる問題、もう一つは数値解析から解を求めるためには数多くの常微分方程式を解く必要があり不安定であるといった問題である。筆者らは線音源を開発するために区分的多項式であるスプライン関数を線状に適応して使用することによってこれらの問題を回避することを提案する。

スプラインとは、自在定規を意味し、任意の点の付近や複数の点を通過する曲線を描くために利用されている。スプライン関数(spline function)は、区分的多項式であり。つなぎ目であるいくつかの制御点が存在している。それらの点は、節点と呼ばれる。その時の節点はξであり、$\xi_1, \xi_2 \cdots, \xi_n (\xi_1 < \xi_2 < \cdots < \xi_n$とする)と表す。節点$\xi_i (i = 1,2, \cdots, n)$をもつ、次数が m のスプライン関数 S(x)は 2 つの条件から定義される。

（ｉ） 各区間(ξ_i, ξ_{i+1})(i = 0,1, ⋯ , n; ただし$\xi_0 = -\infty, \xi_{n+1} = \infty$)においてS(x)は m 次またはそれ以下の多項式である。

（ii） S(x)とその$1,2 \cdots, m-1$次の導関数は$(-\infty, \infty)$で連続している。

S(x)はC^{m-1}級の関数でありこのとき m の値が 0 をとる場合、S(x)は階段関数となる(図 4.2-3)。この時には条件(ii)は当てはまらない。また 1 次のスプライン関数では折れ線グラフと同等の結果を得ることができる。(図 4.2-4)。

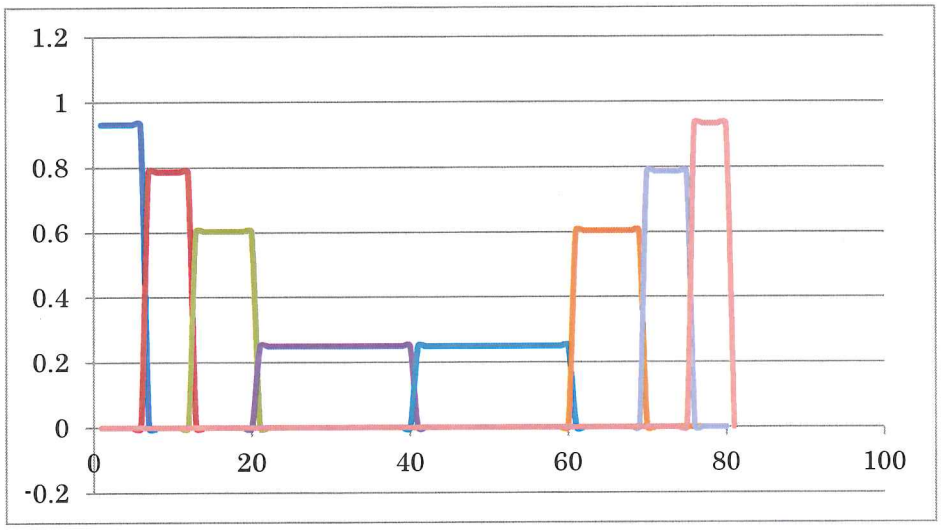

図 4.2-3. 0 次のスプライン関数

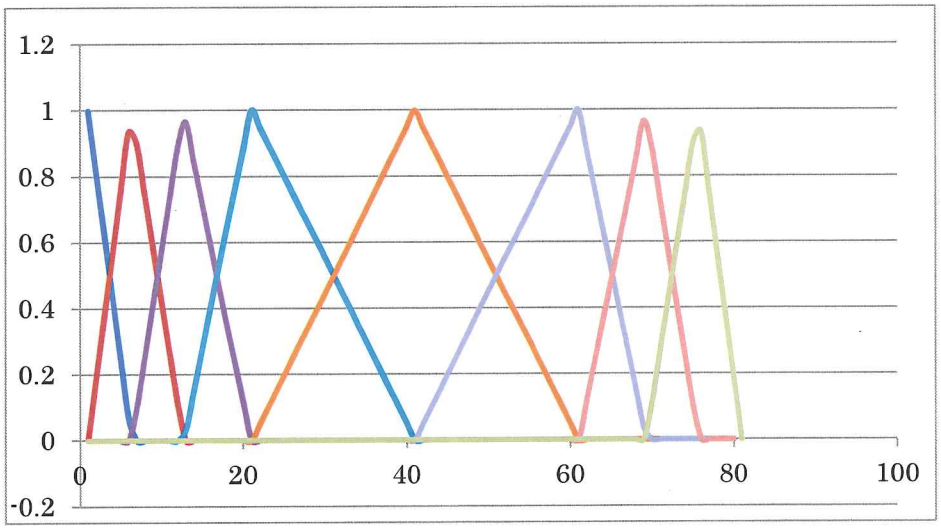

図 4.2-4. 1 次のスプライン関数

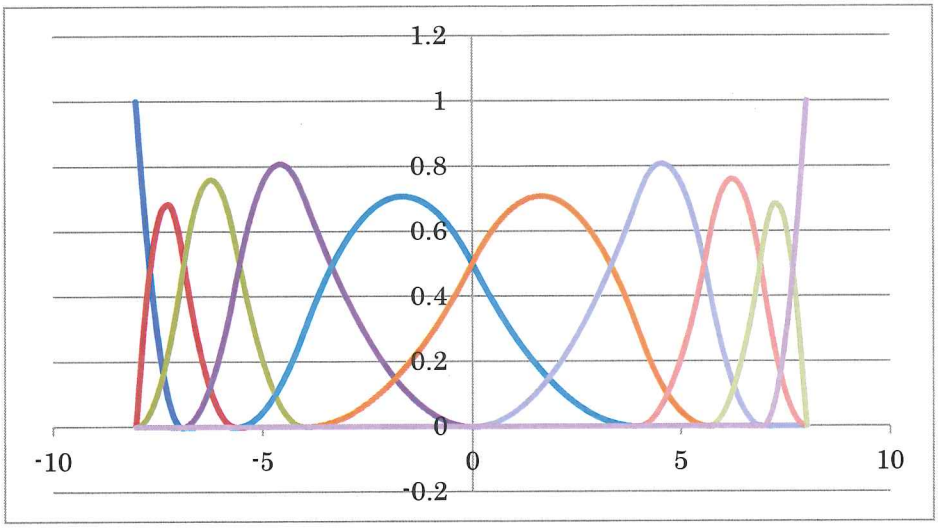

図 4.2-5. 2 次のスプライン関数

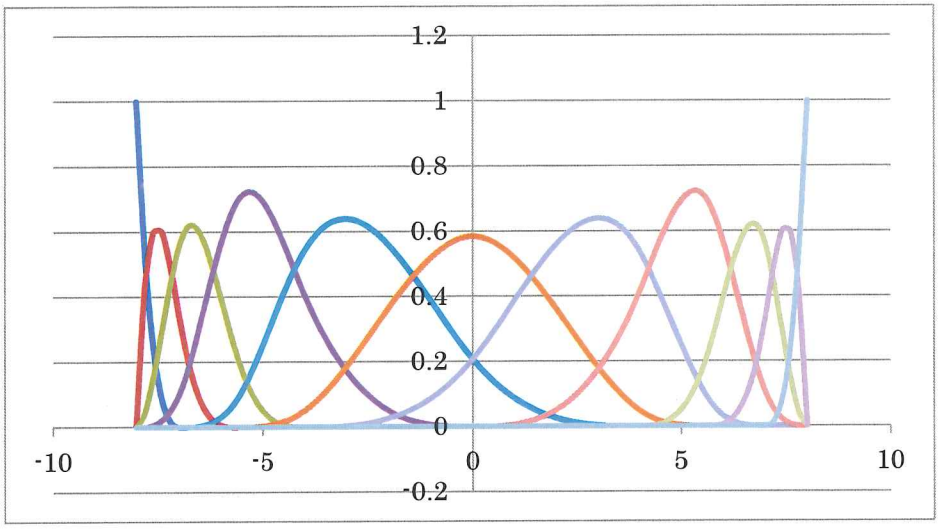

図 4.2-6. 3 次のスプライン関数

線音源の振動の数値モデルを構成するために前述した(4.2-6)式の波動方程式を用いて解説する。(4.2-6)式の解析解は，ρ T が定数の場合についての記述となり、線全体で均一の張りであることが仮定されている。線の状態が均一でない場合には数値解析により解を得ることができる。ここでは区分的多項式であるスプライン関数を線形状に適用し、区分における区間 i での状態を(4.2-6)式に組み込めるように式の導出をした。(4.2-7)式での重み$c_i(t)$を区分における台N_{mi} (m は多項式の次数)に乗ずることで線の形状を表すことができる。またc_iは t の関数であり形状が時間的に変化する。また ρ ,T は区分で異なる状態を持てる。(4.2-6)式の u に(4.2-7)式のスプライン関数を組み込むと(4.2-8)式が得られる。(4.2-8)式の右辺は、高次のスプライン関数を求めるために考案された de Boor-Cox の(4.2-9)式により微分可能であり(4.2-8)より連立の常微分方程式を導きだせる。図 4.2-6 は(4.2-9)式より -8<=x<8,節点（ノット）j=0 から 8 として、重みをかけた B スプライン関数である。

$$u(t,x) = \sum_{i=1}^{n+m} c_i(t) N_{mi}(x) \qquad (4.2\text{-}7)$$

$$\frac{\partial^2 u}{\partial t^2} = \sum_{i=1}^{n+m} \frac{T_i}{\partial_i} c_i(t) \frac{\partial d^2 N_{mi}(x)}{dx^2} \qquad (4.2\text{-}8)$$

$$\left.\begin{aligned} N_{mi}(x) &= (\xi_i - \xi_{i-m}) M_{mi}(x) \\ M_{rj}(x) &= \frac{(x-\xi_{j-r}) M_{r-1,j-1}(x) + (\xi_j - x) M_{r-1,j}(x)}{\xi_j - \xi_{j-r}} \\ M_{1j}(x) &= \begin{cases} (\xi_j - \xi_{j-1})^{-1} & (\xi_{j-1} \le x < \xi_j) \\ 0 & (\text{その他}) \end{cases} \end{aligned}\right\} \qquad (4.2\text{-}9)$$

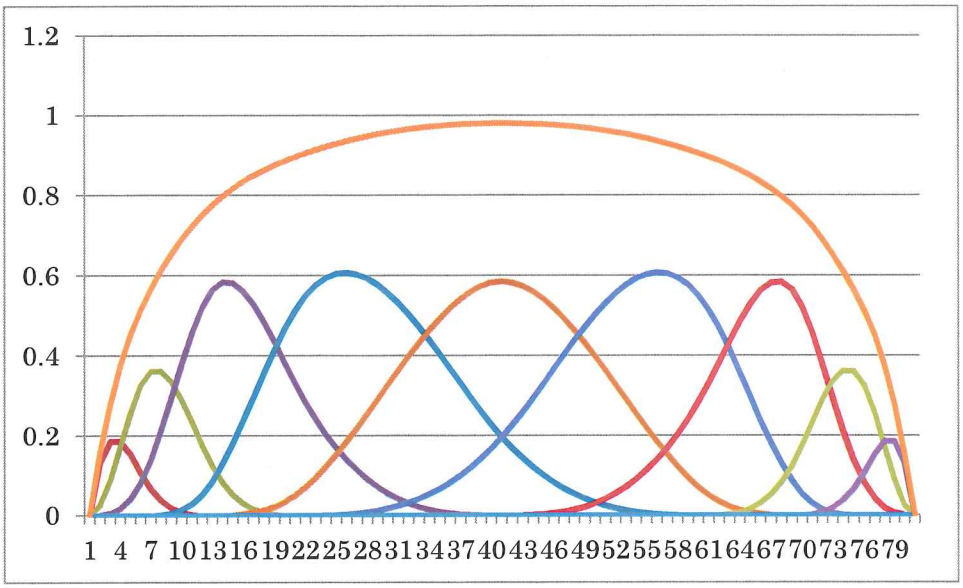

図 4.2-6. 重みをかけた 4 次の B スプライン関数

図 4.2-6 は任意の制御点を通過していないため、ただ平滑化されたものに近い。図 4.2-7 は初期値を正弦波として制御点を通るように調整し 12 区間に分けたものである。このとき x 方向成分はなくなり、時間 t に関する 2 階常微分方程式となる。連立の常微分方程式としたことにより安定した数値解を得られる可能性が高まる。

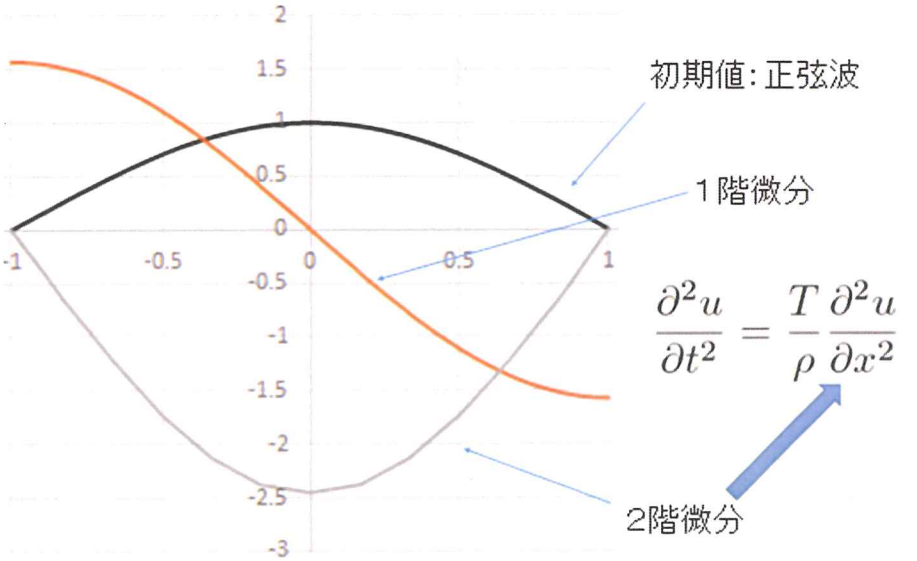

図 4.2-7. 3 次スプライン曲線　制御点：13　初期値：正弦波

図 4.2-7 の正弦波を初期値としてスプライン曲線を時間的に変化させたシミュレーションを行った。(付録 1)この時結果が図 4.2-8 である。図 4.2-8 は波動方程式の解析解(初期値 : 正弦波)であるディリクレ境界条件と一致する結果となった。

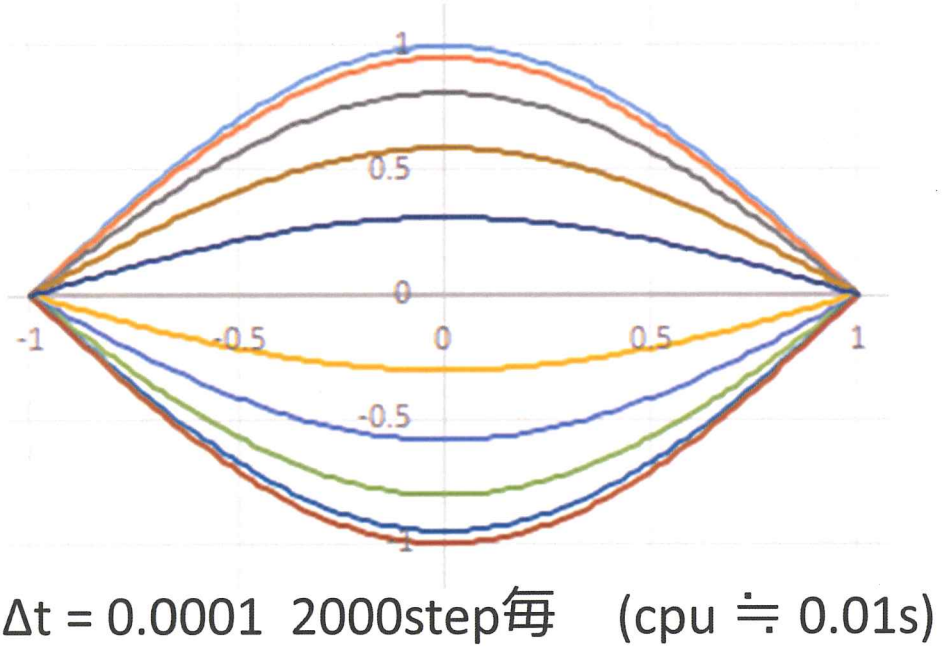

図 4.2-8. シミュレーション結果　制御点：１３　初期値：正弦波

図 4.2-8.のシミュレーションをさらに発展させたものが図 4.2-9 である。このシミュレーションでは初期値が三角形であり、これはゴム製の弦の運動をモチーフにしたものである。このときの条件はサインのように解析解が存在しないため、数値計算から求めなければ難しい。また各部分の張力を減算することで弦が切れる表現ができることも考えられる。

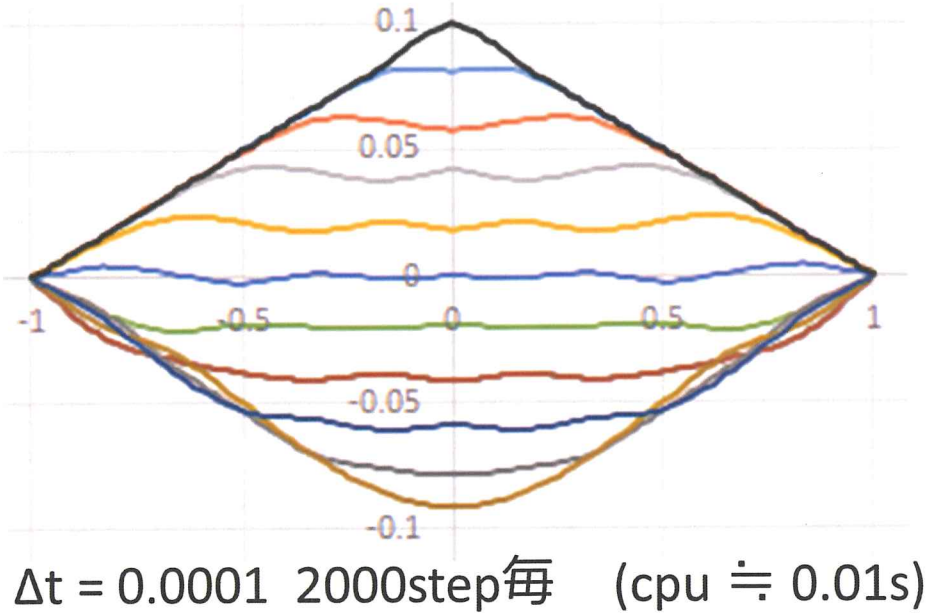

図 4.2-9. シミュレーション結果　制御点：13　初期値：三角形

また線音源を点音源として扱う場合には図 4.2-10 のように VOSS を、線音源全体を取り囲むように配置し、その伝搬の平均値をとることで点音源化することができる。

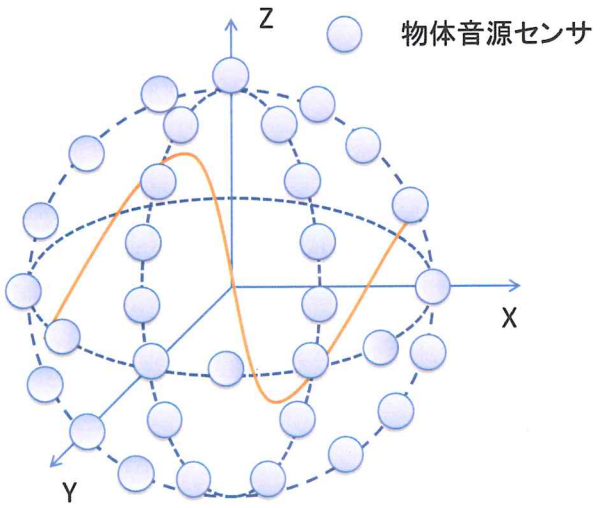

図 4.2-10.物体音源センサを配置した線音源イメージ

4.3 Action-Sourced-Transmitted Simulation (ASTS)

SS でのシミュレーションでは振動が音源として鳴るだけではなく物理設定に応じたアクションに関連した音源も考慮している。例えばある物体が落下し地面との衝突によって、振動が発生した場合がこれである。アクションが由来となる振動は物体のもつ元の音色に依らず一連の特徴的な音が発生する。例えば物体が地面に跳ね返った時では衝突した瞬間のみ振動が発生するため、複数回跳ねる時では次回の衝突までに無音の時間が生じるという音として表現される。(付録 2)

または落下の衝撃によって物体が破壊された場合では衝突の際に複数個の破片が無作為な方向に散りそれが個々に音源となりより低い地点からまた落下のアクションを行うという音となる。(付録 3)

このアクションによる特徴的な音が発生するタイミングをマーカー点とし、これをアクションに対する音情報と考える。アクションに対応する物理状況設定によるマーカー点で聞き取ることで状況を認知することができる可能性がある。

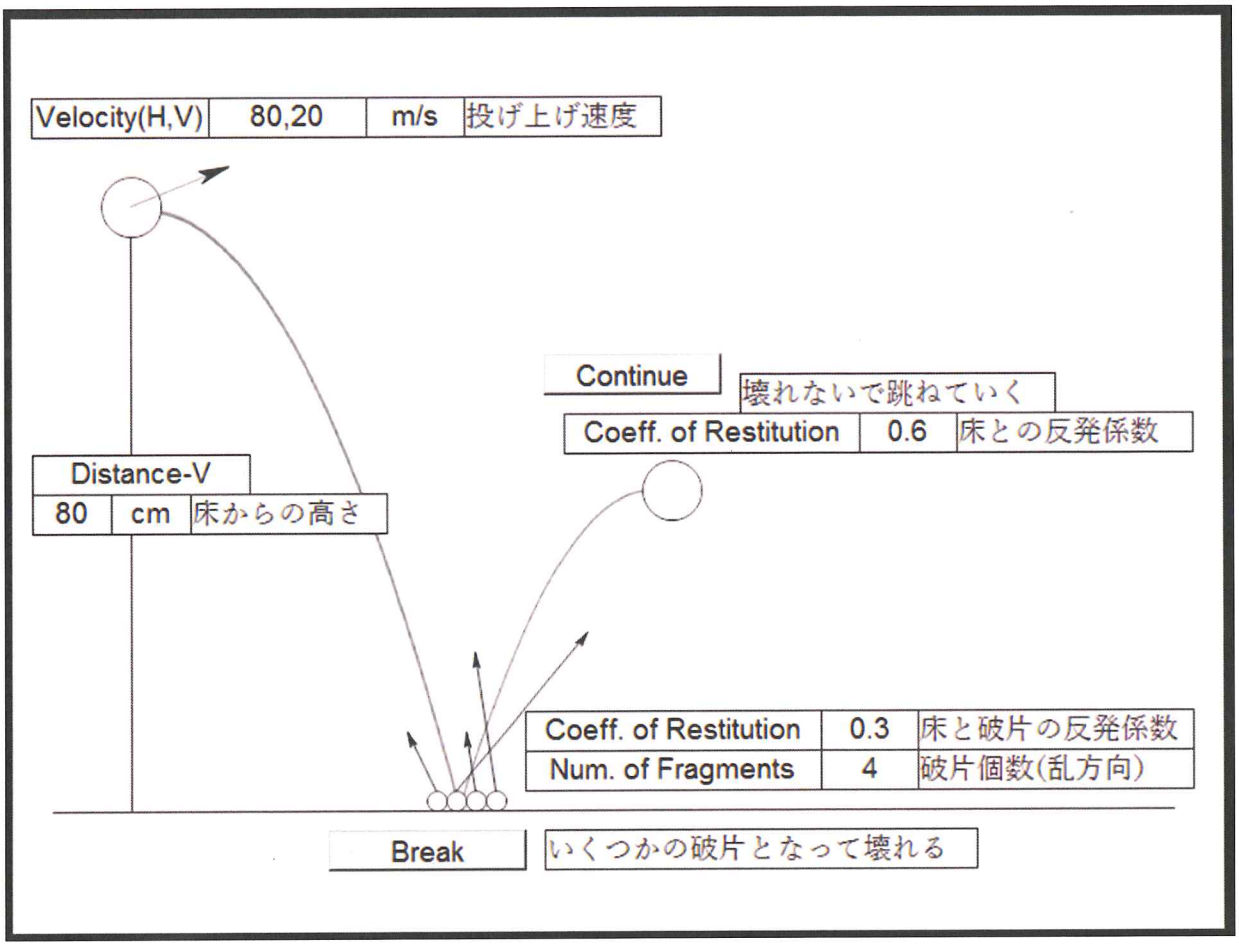

図 4.3-1.落下、破壊の時の特徴を得るための ASTS の状況

第 5 章　状況推定のための伝搬シミュレーションモデル

この章では音の伝搬についてのシミュレーションについて記述する。なおこの章で行われているシミュレーションは付録 4 のプログラムを基にシミュレーションを行っている。

5.1　音の伝搬モデルの作成

ここでは音の伝搬シミュレーション行うためのモデルについて述べる。音の伝搬過程をモデルである TS（Transmitted Simulation)は以下の式から成り立っている。

$$O(t+t_1)=\frac{1}{l^2}S(t-t_2)$$

$$\begin{cases} t_1+t_2=\dfrac{1}{V_{sd}}=\Delta t_{sd} \\ V_s t_1-V_{ob}t_2=(V_s-V_{ob})\Delta t_{sd} \end{cases}$$

$$t_1=\frac{V_s\Delta t_{sd}}{V_s+V_{ob}} \qquad t_2=\frac{V_{ob}\Delta t_{sd}}{V_s+V_{ob}}$$

式(5.1-1)

$O(t)$	観測者が受け取った音 時刻 t の時の振幅値
$S(t)$	音源 時刻 t の時の振幅値
L	音源と観測者の距離 $\sqrt{(x_0 - v_0 t)^2 + {y_0}^2}$
V_{sd}	音速
V_s	音源の速度
V_{ob}	観測者の速度
Δt_{sd}	音が観測されるまでにかかる時間

この式(5.1-1)は音源と観測者との間での音の伝搬の関係性を示す式である。$O(t + t_1)$はある時間において音源から発せられた音が観測地点までに到達する時間を表しており、$S(t - t_2)$はその時間に観測された音はどれだけ前に音源から発せられたかを示している。この式はドップラー効果の一般形であり、この式からドップラー効果の公式を導出することができる。この式(5.1-1)における音源と観測者の関係性を視覚化したものが図 5.1-2 である。

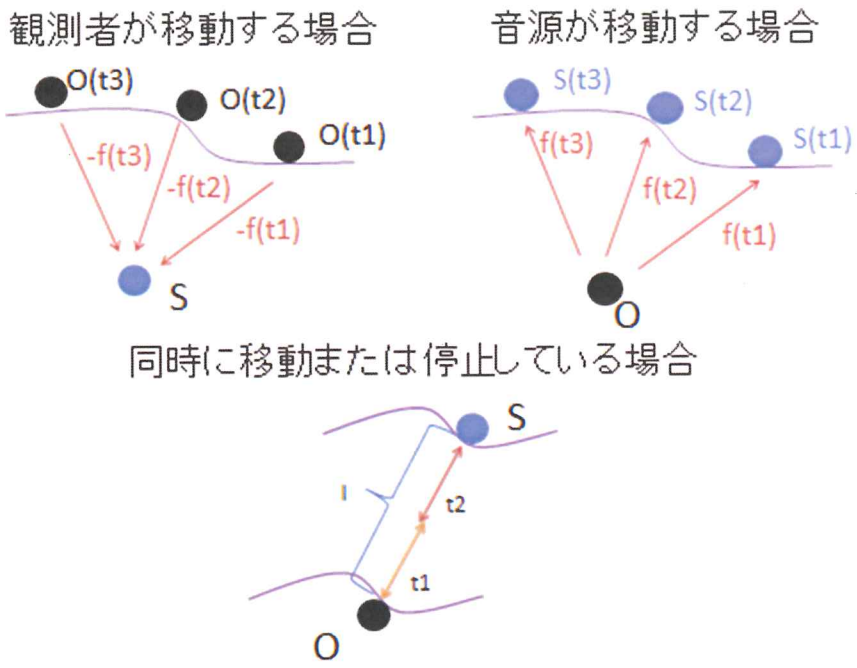

図.5.1-2.音源と観測者の関係性

音源が移動している場合のドップラー効果の公式と観測者が移動している場合のドップラー効果の公式を導出するために、音源をS(t)とし、観測側でのサンプリングされる音をO(t)とする。 S(t)を正弦波振動と仮定し式(5.1-2)とした。

$$S(t) = A\sin(\omega_0 t) \quad (5.1\text{-}2)$$

O(t)がω_0は音源の角周波数。A は 1 とする。媒質が一様と仮定した場合S(t)からの音時間遅れで到着するだけとなる。しかし、音源が移動している場合と観測側が移動している場合では音波の伝搬の仕方が異なり、図.5.1-3 のように観測者が移動する場合は同心円状に伝搬される音波の中を観測者が進むのに対して、音源が移動する場合では進行方向に対して密になるように偏って伝搬する。

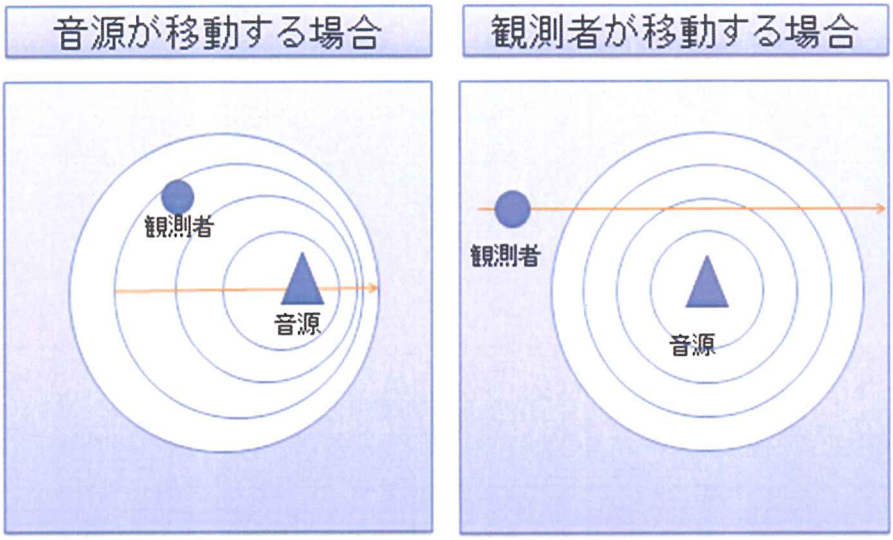

図 5.1-3.　音源が移動する場合と観測者が移動する場合

観測者が移動する場合の観測される音O(t)は、両者の距離lを音速V_{sd}で伝わる分の遅れを 用いて、f(t) = l(t)/V_{sd}を用いて、その時届いた音は音源のf(t)秒前の音であることから

$$O(t) = \frac{1}{l^2} s(t - f(t)) \qquad (5.1\text{-}3)$$

となる。また距離の 2 乗で音が減衰するとしている。音源が移動する場合の観測される音O(t)は、時刻 t で出た音が、f(t)秒遅れて届くため

$$O\big(t + f(t)\big) = \frac{1}{l^2} s(t) \qquad (5.1\text{-}4)$$

となる。

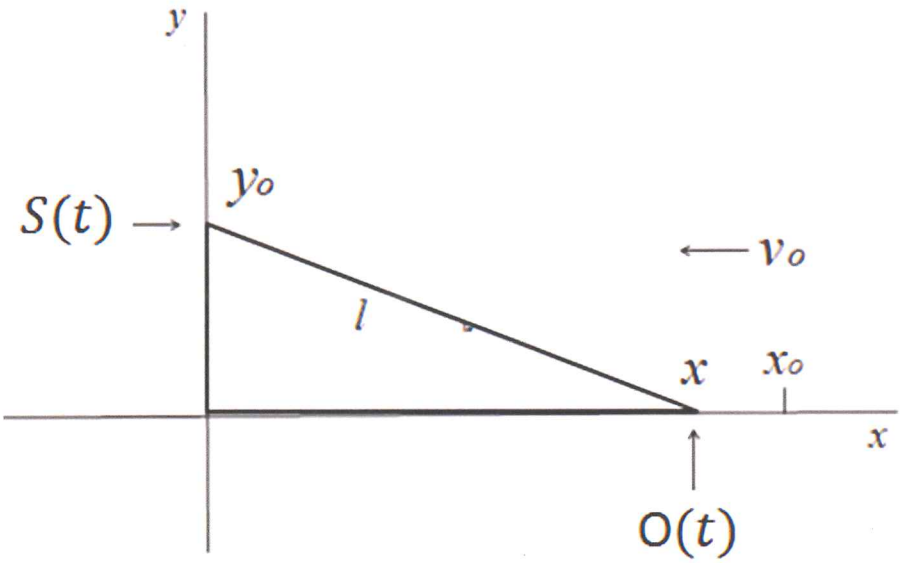

図 5.1-4.音源と観測者の配置

S(t)とO(t)が図 5.1-4 で示される配置であるときO(t)が x 軸上を速度$-v_0$で移動しているとき距離l(t)は

$$\mathrm{l(t)} = \sqrt{(x_0 - v_0 t)^2 + y_0^2} \qquad (5.1\text{-}5)$$

とあらわされるので、式(5.1-4)より式(5.1-5)は

$$\frac{1}{\mathrm{l}^2} \sin\left(\omega_0 \left(t - \frac{l}{V_{sd}}\right)\right) \qquad (5.1\text{-}6)$$

となる。周波数を求めるためには角速度dθ/dtを求めればよいので、t − f(t)を t で微分し式(5.1-7)を得る。

$$\omega(\mathrm{t}) = \omega_0 \left(1 + \frac{v_0}{V_{sd} \cdot l} (x_0 - v_0 t)\right) \qquad (5.1\text{-}7)$$

S(t) が移動しているときは式(5.1-4)においてt′ = t + f(t)とおき(ω_0t)を t'で微分する。このためにはdt′/dtを求める必要がある。

$$\frac{\mathrm{dt}'}{\mathrm{dt}} = 1 + \frac{df}{dt} \qquad (5.1\text{-}8)$$

$$\frac{\mathrm{d}(\omega_0 \mathrm{t})}{dt'} = \frac{d(\omega_0 t)}{dt}\frac{dt}{dt'} \quad (5.1\text{-}9)$$

(5.1-8)式と(5.1-9)式より次の式が求められる。

$$\omega(\mathrm{t}') = \frac{\omega_0}{1-\frac{v_0(x_0 - v_0 t)}{V_{sd} l}} \qquad (5.1\text{-}10)$$

(5.1-7)式と(5.1-10)式においてx_0が十分大きいときはlと等しくなるため(5.1-7a)式、(5.1-10a)式を得る。

$$\omega_0 \left(1 + \frac{v_0}{V_{sd}}\right) \qquad (5.1\text{-}7a)$$

$$\frac{\omega_0}{(1 - v_0/V_{sd})} \qquad (5.1\text{-}10a)$$

上記の導出により一般的なドップラーシフトの公式から導かれた理論値とTS関係式から得られたドップラーシフトの公式は距離を無限大にとった時に一致する。一般的なドップラーシフトの公式では同一直線状でのみ使用可能という前提がある。また観測者と音源とが交わった原点地点は特異点となっており周波数を得ようとするとコサイン成分を計算する必要がある。つまり通常では原点地点の前後にどの程度周波数がシフトしたかということだけがわかる。しかしTSの関係式から得られた公式では観測者と音源の距離関係の概念が式中に存在するために直接ドップラーシフトの変化を得ることができる。

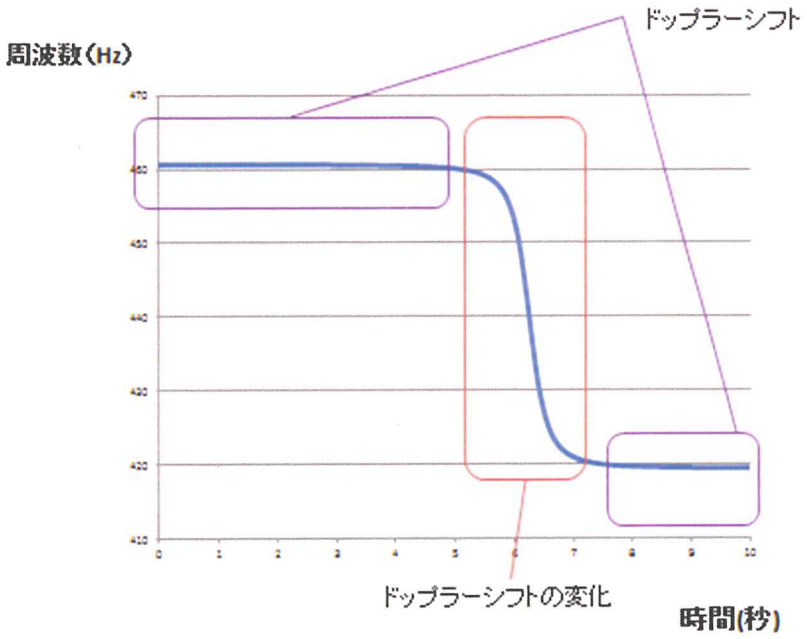

図 5.1-5.ドップラーシフトとドップラーシフトの変化

図 5.1-6 は TS の関係式から導出されたドップラーシフトの公式の理論値と実際にシミュレーションを行って、得られた実数値を比較したものである。この時の理論値と実数値は有効桁数 4 桁で一致している。またこの実数値は音波形データを作成するシミュレーションから得られた数値であるため、理論値からは観察できない原点部分の周波数の変化が波形データとして容易に観察が出来る。

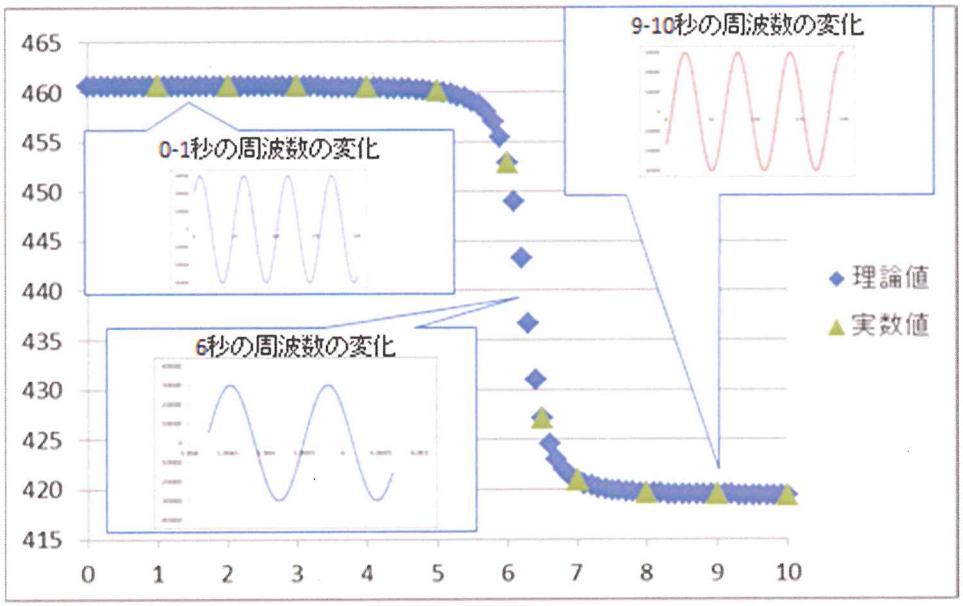

図 5.1-6.TS の関係式から得られた
ドップラーシフトの公式の理論値と実数値の比較

5.2 移動のシミュレーション

5.2.1 等速直線運動

この項では物体の移動時に発生するドップラー効果を移動という状況が持つ特徴的な音響効果という視点から扱い、得られる音情報について考察していく。このシミュレーションでは 40km/h,60km/h,80km/h それぞれの速度を持った観測者 Ob を移動させた。この時 440Hz の正弦波振動を発振している音源 So に対して最も距離が近づいたときに 5m の距離になるよう、移動するように物理設定した。上記の条件をモデル化したものが図 5.2.1-1 である。

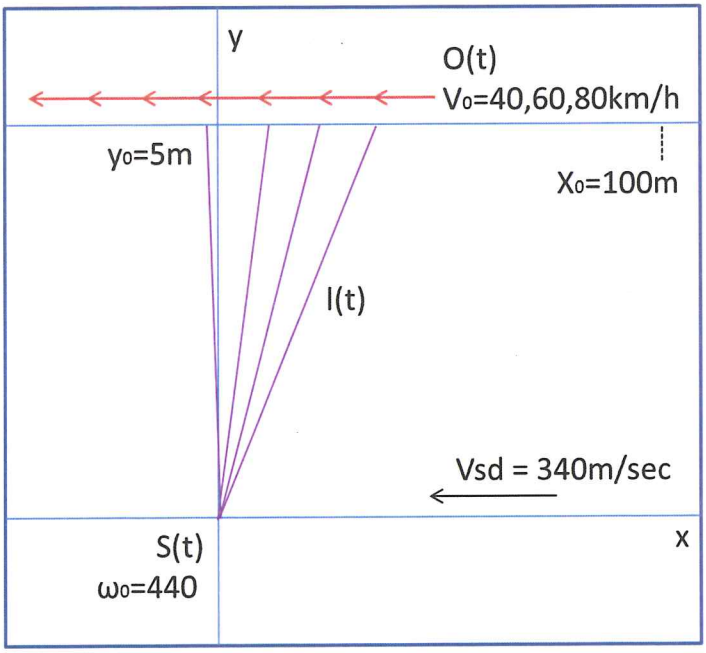

図 5.2.1-1 等速直線運動の状況

上記の条件でシミュレーションを行い、得られた波形データから周波数の変化をプロットしたところ図 5.2.1-2 が得られた。図 5.2.1-2 は通常のドップラーシフトによる周波数の変化は速度が増大するにつれ変化が大きくなった。またドップラーシフトの変化も観察された。

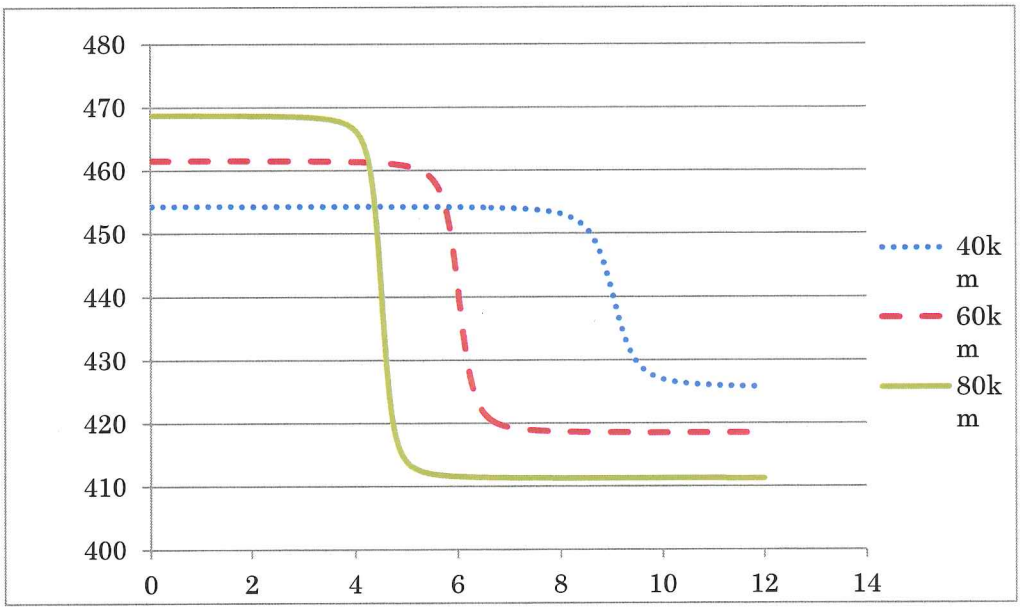

図 5.2.1-2 等速直線運動の周波数の変化

5.2.2 正の値を持つときの加速度運動

この項では正の値で加速度運動を行っている観測者についてシミュレーションしている。80km/h(約 22.2m/s) の速度を持った観測者 Ob を移動させた。この時 440Hz の正弦波振動を発振している音源 So に対して最も距離が近づいたときに 5m の距離になるよう移動し、この時の加速度の値を変化させそれぞれ $\alpha = 1\mathrm{m/s^2}$、$\alpha = 2\mathrm{m/s^2}$、$\alpha = 3\mathrm{m/s^2}$ と取った場合を比較している。

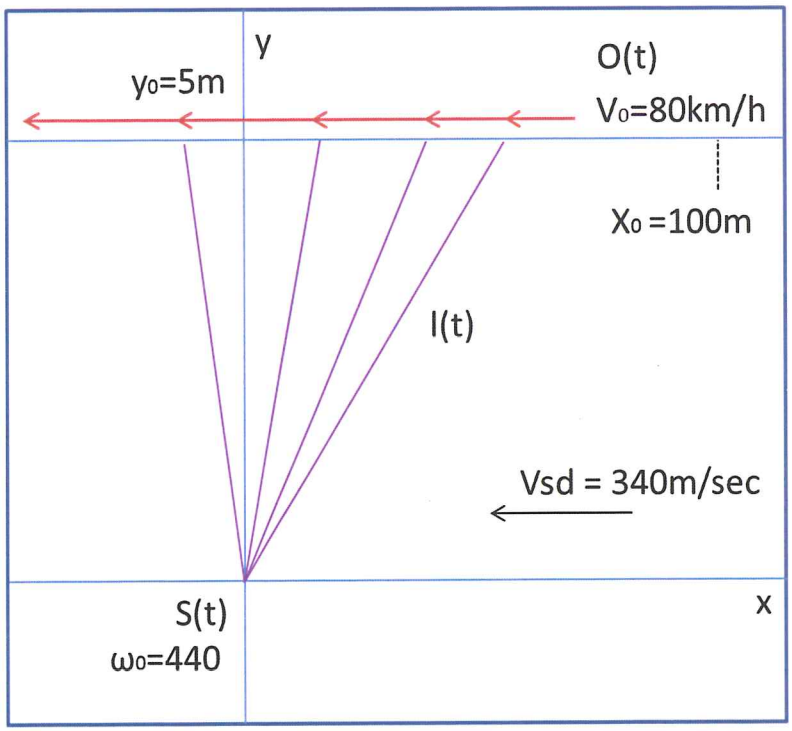

図 5.2.2-1 正の値を持つときの加速度運動の状況

このシミュレーション結果を時間軸に沿った周波数グラフとして表したのが以下の図 5.2.2-2 である。加速度運動を行っている物体はドップラーシフトの変化がない時間においても加速度による直線的な周波数の変化が見られた。等速直線運動の時とは異なり、ドップラーシフトの極小値と極大値が得られた。この変化は α が正の値を取る時では時間に比例して周波数が上昇していく。

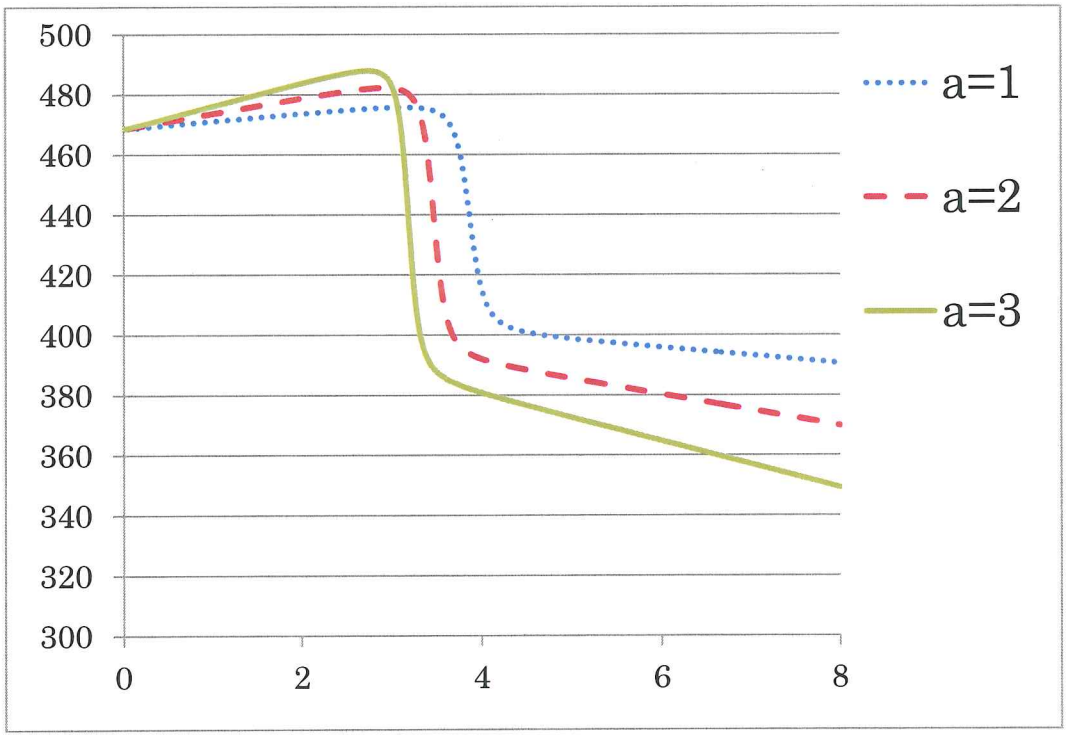

図 5.2.2-2 正の値を持つときの加速度運動周波数の変化

5.2.3 負の値を持つときの加速度運動

この項では負の値をもった加速度運動中の観測者についてシミュレーションを行っている。80km/h(約 22.2m/s) の速度を持った観測者 Ob を移動させた。この時 440Hz の正弦波振動を発振している音源 So に対して最も距離が近づいたときに 5m の距離になるよう移動し、この時の加速度の値を変化させそれぞれ$\alpha = -0.5\mathrm{m/s^2}$、$\alpha = -0.7\mathrm{m/s^2}$、$\alpha = -1\mathrm{m/s^2}$と取った場合を比較している。

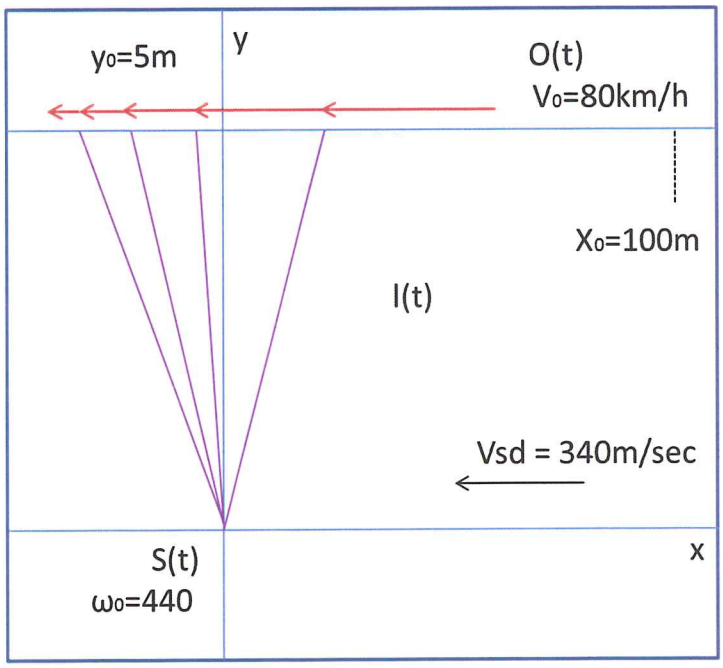

図 5.2.3-1 負の値を持つときの加速度運動の状況

このシミュレーション結果を時間軸に沿った周波数グラフとして表したのが以下の図 5.2.3-2 である。負の加速度運動を行っている物体はドップラーシフトの変化がない時間においても加速度による直線的な周波数の変化が見られた。等速直線運動の時とは異なり、ドップラーシフトの極小値と極大値が得られた。この変化はαが負の値を取る時では時間に反比例して周波数が変化していく。

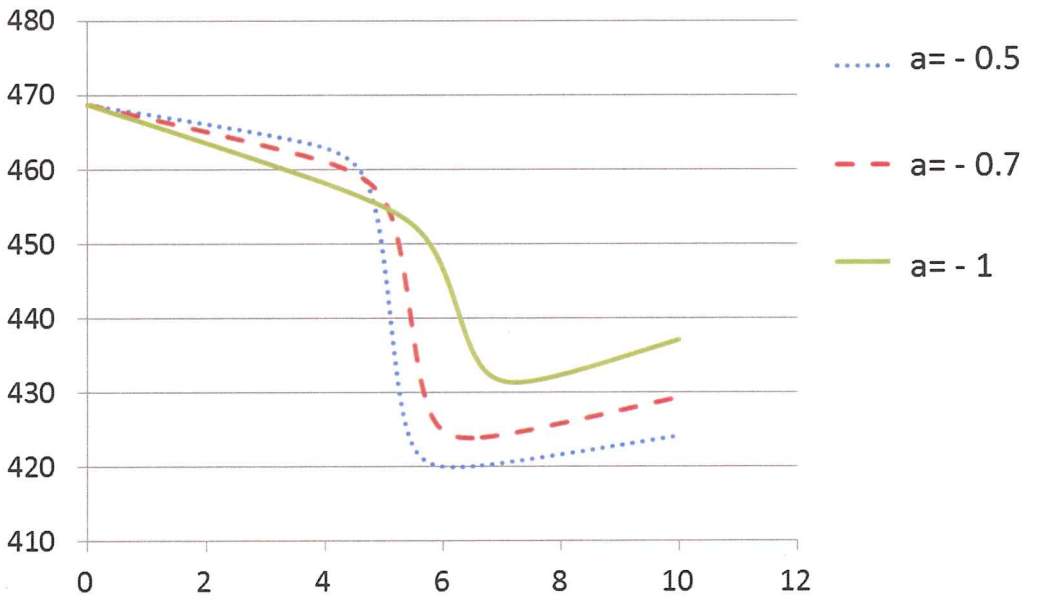

図 5.2.3-2 負の値を持つときの加速度運動周波数の変化

5.3.反射板が存在する状況のシミュレーション

5.3.1 反射-単独の反射-

この項では反射板が存在する状況下で等速直線運動を行っている観測者についてシミュレーションを行っている。80km/h(約 22.2m/s) の速度を持った観測者 Ob を移動させた。この時 440Hz の正弦波振動を発振している音源 So に対して最も距離が近づいたときに 5m の距離になるよう、移動する。反射板は y 軸から-5[m]の地点に存在し、これを反射板 a とする。

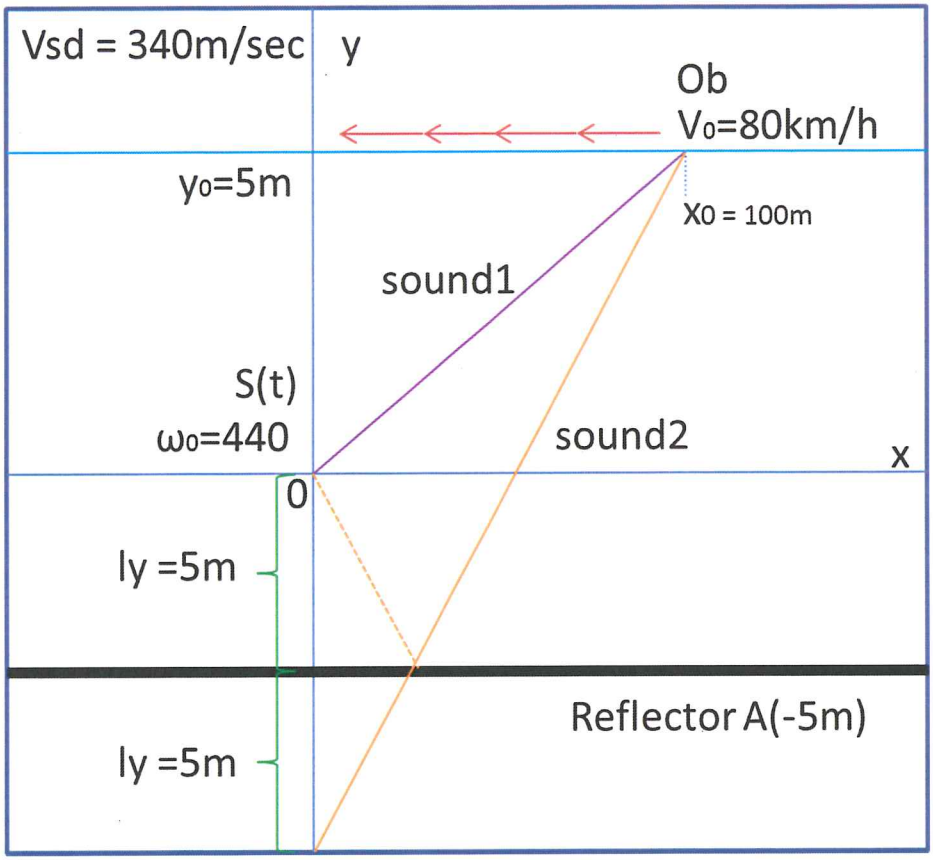

図 5.3.1-1 反射板が 1 枚存在する状況

シミュレーションを行った結果、反射板がない場合と同様にドップラー効果が現れたが反射板 a を置いたことにより、同時にうなりが発生している。これは反射板 a を置いた際のエンベロープを観察することでわかる。図 5.3.1-2 の①を拡大したものが②でありこぶができていることからうなりがわかる。このエンベロープから観察されたうなりが発生している理由はこれまでのように直接観測地点に届く音だけではなく、反射板 a を経由して観測地点に到達した音が存在し、さらにその音は直接観測した場合よりも観測地点に到達するまでの時間が長いことに起因したうなりであり、ディレイ効果として観測されている。

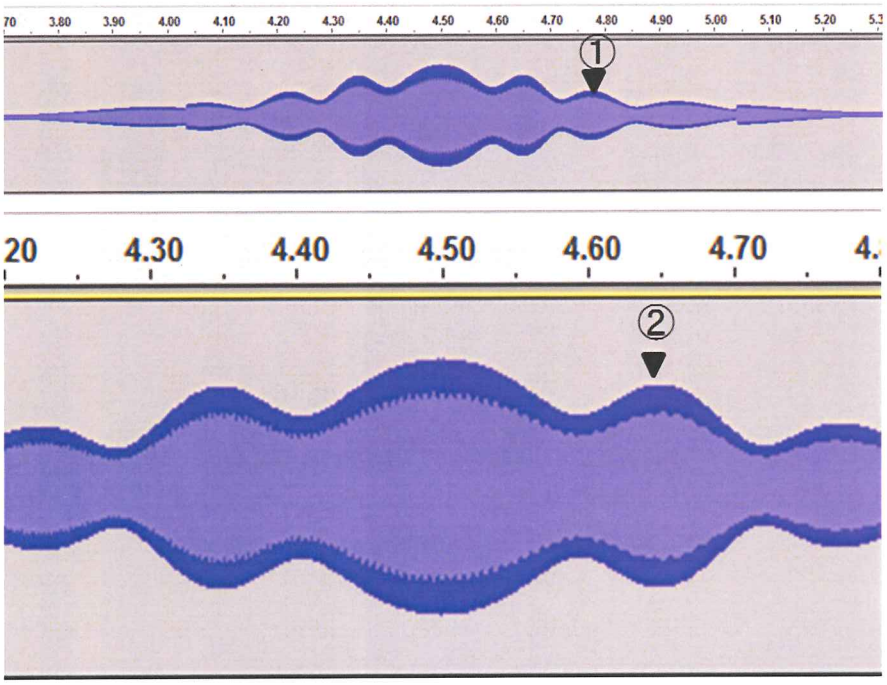

図 5.3.1-2 反射板が 1 枚の時のエンベロープ

5.3.2 反射板-複数の反射-

この項では前項のシミュレーションを発展させ反射板をもう一枚増やしたシミュレーションを行った。反射板が 2 枚存在し、等速直線運動を行っている観測者という状況になっている。80km/h(約 22.2m/s) の速度を持った観測者 Ob を移動させた。この時 440Hz の正弦波振動を発振している音源 So に対して最も距離が近づいたときに 5m の距離になるよう移動する。前項に引き続き、反射板は y 軸から-5[m]の地点に存在し、これを反射板 a とする。同様に y 軸方向 11m に反射板 b を追加したものになっている。

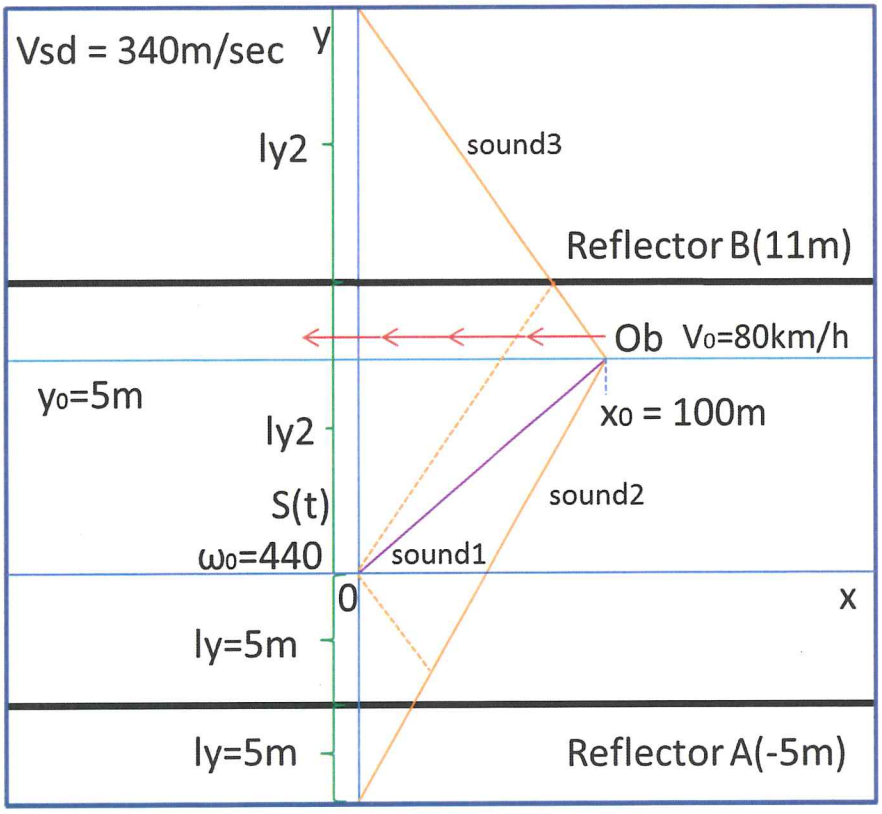

図 5.3.2-1 反射板が 2 枚存在する状況

この物理設定でシミュレーションを行ったところドップラー効果とうなりが現れるという点では反射板が１枚しか存在しない場合と同様である。しかし現れたうなりは図5.3.2-2のエンベロープを観察すると反射板が１枚の時と比べてうなりが細かくなっているのがわかる。これは直接聞こえる音とは他に反射板を経由する音が１枚の時よりも多くなったためでありエコー効果として観測できた。

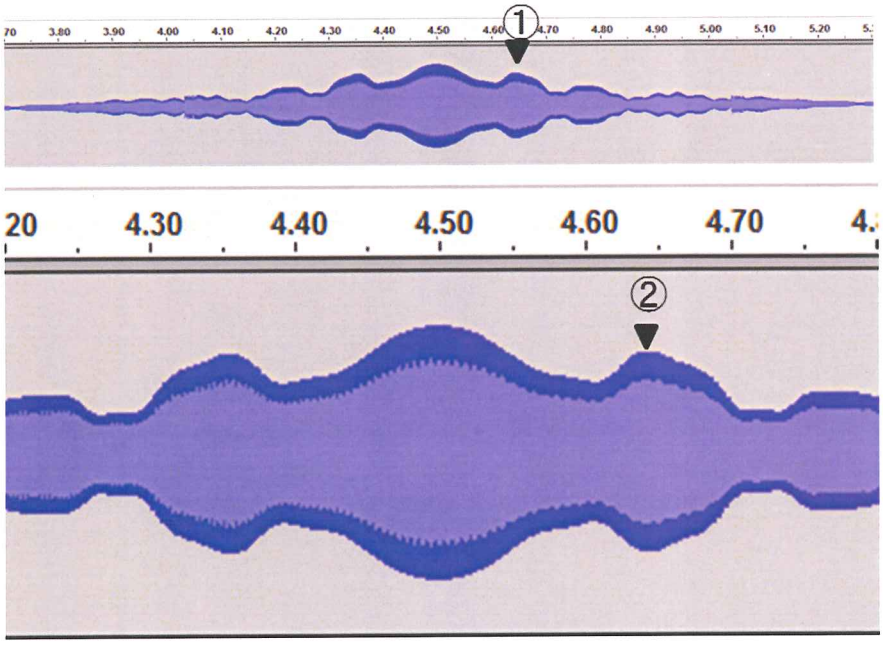

図 5.3.2-2 反射板が 2 枚の時のエンベロープ

第6章　結言

本論文ではSS、TS、VOSSを使用して物理シミュレーションから得られた音波形データを利用した音情報による状況推定に関する考察を行った。音情報による状況推定を行うためには状況と音情報を対応させたペアを利用して推定したい音源とのマッチングを行う手法を考察し、そのために必要となるペアリングを管理するためのシステム基盤として知識ベースの利用について言及した。また知識ベースを利用したマッチングを行ううえで状況推定の精度に関わる解釈・機能生成モデルによるモデルの精緻化について言及した。ペアリングのもととなる物理シミュレーションでは振動のシミュレーションモデルであるSSにおいて線音源のモデルの作成とそのシミュレーションの基礎となる拡張された弦の波動方程式の導出、またASTSの提案を行った。また伝搬のシミュレーションモデルとなるTSを考察しており、移動、加速、反射板という状況を物理設定した音の伝搬についてのシミュレーションを行った。

参考文献

時田保夫　監修,"音の環境と制御技術第Ⅰ巻基礎技術",株式会社フジ・テクノシステム(2000)

時田保夫　監修,"音の環境と制御技術第Ⅱ巻応用技術",株式会社フジ・テクノシステム(1999)

小泉 宣夫　"基礎音響・オーディオ学", コロナ社(2005)

東山三樹夫,"音の物理"コロナ社(2010)

鈴木陽一, 赤木正人, 伊藤彰則, 佐藤洋, 苣木貞史, 中村健太郎, "音響学入門", コロナ社(2011)

市田浩三,吉本富士市, "シリーズ新しい応用の数学 20 スプライン関数とその応用"　新曜社,(1979)

岩宮眞一郎「よくわかる最新音響の基本と応用」　秀和システム,2011

木下雄太,佐藤翔太,西川孝二,北守一隆:システムデザインとしてのランドルト環方式ｅラーニングシステム, 計測自動制御学会 システム・情報部門 学術講演会2015 講演論文集,pp.SS22-1Nov,(2015)

木下雄太,青山ゆう子,大森義行,北守一隆: 消費者行動シミュレーションによるアプローチ-音情報による状況推定モデルとの比較-, 日本情報経営学会第 68 回全国大会予稿集,pp97, May., (2014)

興膳肇,喜多紀史,北守一隆." IT 組織力向上のための知的資産管理を用いたゲーミング・シミュレーション", 生産管理 Vol.16, No.pp.53-58, Oct.(2009)

興膳肇, 三上行生, 北守一隆, "生産管理技術習得のためおランドルト環方式ｅラーニングシステム",　日本生産管理学会　第 31 回全国大会講演論文集, pp.199-204, Mar., (2010)

Yuta Kinoshita,,Kazutaka Kitamori ,Yuko Aoyama :
Situation Presumption by Sound Information:Graphical Physical Configuration Interface, Artificial Intelligence Modelling and Simulation, IEEE, pp.221-223, Nov., (2014)

Yuta Kinoshita, Yuko Aoyama, & Kazutaka Kitamori:
Transmitted Simulation from Vibrating Object Sound, International Journal of Advanced Computer Science,IJACS Vol4, No1, pp.40-44, Jan.,(2014)

Yuta Kinoshita, Yuko Aoyama, Kazutaka Kitamori: Transmitted Simulation from Vibrating Object Sound, The Society for Modeling & Simulation International Spring Simulation Multi-conference 2013, pp.1086-1087, San Diego, USA, Apr., (2013)

KINOSHITA Yuta, TAKEZAWA Megumi, AOYAMA Yuko, KITAMORI, Kazutaka: Situation Presumption for Swarm Robots by Sound Information, pp. http://www.iaria.org/additionalpublications.html Nice, France, Oct,(2014)

木下雄太,畑大介,木下正博,北守一隆: 物体音源センサによる音サンプリング, 2012 年度精密工学会北海道支部学術講演会, pp.5-6, Sept., (2012)

木下雄太,木下正博,北守一隆: 群ロボットにおける状況認知のための物体音源センサ, 2012 年度精密工学会秋季大会学術講演会, pp30, Sept., (2012)

木下雄太,畑大介,竹沢恵,木下正博,北守一隆: 音源モデルによって再現される音響効果を用いた状況推定, 情報処理学会北海道シンポジウム,pp13, Oct., (2012)

木下雄太,畑大介,竹沢恵,木下正博,北守一隆: 物体音源センサによる音サンプリング—音源の移動と反射板を考慮したモデル—, 第 45 回計測自動制御学会北海道支部学術講演会,pp78, Mar., (2013)

木下雄太,畑大介,竹沢恵,木下正博,北守一隆: 音響効果シミュレーションによる音診断, ME とバイオサイバネティックス研究会, 電子情報通信学会技術研究報告書, pp.51-53, June., (2013)

畑大介,木下雄太,竹沢恵,木下正博,北守一隆: 音情報による状況推定におけるグラフィカルインターフェイスを用いた物理設定, 2013 年度精密工学会秋季大会学術講演会, 2013 年度 精密工学会大会秋季大会学術講演会　講演論文集 pp.771-772, Sept. ,(2013)

木下雄太,竹沢恵,木下正博,北守一隆: 弦振動音源を用いた物体音源センサによる状況推定, 2013 年度精密工学会秋季大会学術講演会, 22013 年度 精密工学会大会秋季大会学術講演会　講演論文集 pp.769-770, Sept. ,(2013)

木下雄太,畑大介,竹沢恵,木下正博,北守一隆: 音情報による状況推定のための音サンプリング, 情報処理北海道シンポジウム 2013, 情報処理北海道シンポジウム 2013 講演論文集, pp167, Oct., (2013)

木下雄太,竹沢恵,木下正博,北守一隆:音情報による状況推定のためペアリング-線音源シミュレーション-, 2014 年度精密工学会秋季大会,2014 年度精密工学会秋季大会学術講演会講演論文集,pp.747-748, Sept,(2014)

謝辞

本研究を進めるにあたり、ご指導をいただいた指導教員の北守一隆教授に感謝をいたします。また、研究を博士論文として形にすることができたのは、ご助言により研究について新たな知見をいただいた三田村保教授、多くの知識や重要な指摘を頂いた、木下正博教授のおかげと感謝申し上げます。また本研究に貴重なご助言を頂きました、廣田健一氏、落合永一郎氏に感謝の意を表します。協力していただいた皆様へ心から感謝の気持ちと御礼を申し上げます。

付録

付録１　弦の波動方程式のプログラム(Scala)

```
class s3( n:Int, x1:Double, x2:Double, f:String, dt:Double) extends App{
  val dx = (x2-x1)/n  //ｎ等分の間隔
  val gX = x1::( 1 to n-1).toList.map( i=> x1+dx*i):::List( x2)  //n等分点での節
点のｘ座標
  val gY = gX.map( x=> Math.sin(x)).toArray  //初期値は正弦波
  val gM = ((0.0)::( 1 to n-1).toList.map( i=> {val h = gX( i)-gX( i-1)
                    ( gY(i-1)-2*gY( i)+gY( i+1))/h/h}):::List( 0.0)).toArray  //
節点での２階微分値
  //曲線 d0S の二階微分値
  def d2S = (1 to n).map( i=> { val ( x0, x1) = ( gX( i-1), gX( i))
    val h = x1- x0
    (x:Double) => gM( i-1)*( x1- x)/h+ gM( i)*( x- x0)/h} /*map*/)
  //曲線 d0S の一階微分値
  def d1S = (1 to n).map( i=> { val ( x0, x1) = ( gX( i-1), gX( i))
    val h = x1- x0
    (x:Double) => -gM( i-1)*( x1- x)*( x1- x)/h/2 + gM( i)*( x- x0)*( x- x0)/h/2 +
                ( gY( i)- gY(i-1))/h - ( gM( i)- gM( i-1))/6*h} /*map*/)
  //節点(gX)を通りそこで与えられた二階微分値(gM)を満足する曲線
  def d0S = (1 to n).map( i=> { val ( x0, x1) = ( gX( i-1), gX( i))
    val h = x1- x0
    (x:Double) => gM( i-1)*( x1- x)*( x1- x)*( x1- x)/h/6 +
                  gM( i)*( x- x0)*( x- x0)*( x- x0)/h/6 +
                  ( gY( i-1)- gM( i-1)*h*h/6)*( x1- x)/h +
                  ( gY( i)- gM( i)*h*h/6)*( x- x0)/h} /*map*/)
  def s3( x:Double) = {
    val na = gX.tail.map( x0=> x <= x0).count( _==false)
    val nb = if(na>=n) na-1 else na
    d0S(nb)(x)
  }
  val gV = gX.map( v=> 0.0).toArray
  def dT = {
    (1 to n-1).foreach( i=> {
```

```
      val gv0 = gV(i)
      val gv1 = gv0 + gM(i)*dt
      gY(i) = gY(i) + (gv0+gv1)/2*dt
      gV(i) = gv1} /*foreach*/)

    (1 to n-1).foreach( i=> {val h = gX( i)-gX( i-1)
                    gM(i)=( gY(i-1)-2*gY( i)+gY( i+1))/h/h})
  }
  //main
    val res:Array[Array[Double]] = new Array(10) //結果の保存
    val obX = (x1 to x2 by (x2-x1)/n/10).toArray  //区間をさらに１０等分した観測
点
    res(0) = obX.map( x=>s3(x)) //t=0 時の初期値を保存
    for(i<-1 to 4000) dT  //dt*4000 秒毎の結果
    for(j<-1 to 9) {  //９個の結果を
      for(i<-1 to 200) dT  //dt*200 秒進めながら保存
      res(j) = obX.map( x=>s3(x))}  //観測点での結果
    val sD = (0 to obX.length-1).map(i=>{
      val r1 = obX(i)+","
      val r2 = (0 to 9).map( j=> res(j)(i)).mkString(",")
      r1+r2}).mkString("¥n")
    val sX = <pre> {"¥n"+sD+"¥n"} </pre>  //文字列から XML 形式に変換

    scala.xml.XML.save(f+"0.csv",sX) //XML 形式の結果ファイル出力"b0.csv"
}
val s3a = new s3(8,0.0,3.14159,"b",0.001) //０からπまで８等分した節点、シミュレ
ーション時間 0.001 秒
s3a.main(null)
```

付録２　ASTS 落下のプログラム(Scala)

```
//sounds for fall down .. down

object waveSound extends App{
   val wvS:Long = 44100 //Hz
   val ttPlay:Int = 3 //second
   val bps = wvS*2*2  //total bytes for 1 second
   val bPlay = bps*ttPlay //total bytes for (ttplay) seconds
   def s4(x:String) = x.map(_.toByte).toArray
   def b2(x:Int) = Array(x%256,x/256).map(_.toByte)
   def b4(x:Long) = {var x0=x; val y = new Array[Long](4)
       for(i<-0 to 3) {y(i)=x0%256; x0=x0/256}
       y.map(_.toByte)}

   val note = (0 to 72).toList.map(i=> 110.0*math.pow(2.0,i/12.0))
   val drm = List(27,29,31,32,34,36,38,39)


   case class hD(name:String, var v:Array[Byte])
   val header = Array(
      hD("Chunk ID",          s4("RIFF")),
      hD("Chunk Data Size", b4(bPlay+50)),
      hD("RIFF Type",         s4("WAVE")),
      hD("fmt -Chunk",        s4("fmt ")),
      hD("fmt -Data Size",   b4(18)), //16?
      hD("Compression code",b2(1)), //PCM/uncompressed
      hD("Number of channels",b2(2)),
      hD("Sample rate",       b4(wvS)),
      hD("bytes/second",      b4(bps)),
      hD("Block align",       b2(4)),
      hD("bits/sample",       b2(16)),
      //hD("Extra format bytes",b2(0)),
      //if Extra format bytes != 0 : add n bytes
      //hD("fact-Chunk ID",    s4("fact")),
      //hD("fact-1?",          b4(4)), //4
```

```
      //hD("fact-2?",          b4(0)), //4332672
      hD("data-Chunk ID",    s4("data")),
      hD("data-chunk size", b4(bPlay))
   )
   var header0:Array[hD] = header.map(i=>i)

   val fR0 = new java.io.RandomAccessFile("pon.wav","r")
   val fR = new java.io.RandomAccessFile("b2g.wav","rw")
   val d0 = new Array[Byte](bps.toInt/8) //for 1/8 second
   val d = new Array[Byte](bPlay.toInt) //for ttPlay second
   def vL(x:Array[Byte]):Long = x.map(j=>if(j<0) j+256 else j).
       reverse.foldLeft(0L)((s,i)=>s*256+i)  //4bytes -> Long

   def oSnd1 = dRead  //set sound data to d:Array
   val g = 9.8
  def sim( cR:Double, h:Double) = {
    val v0 = Math.sqrt( 2*g*h)
    val t0 = v0/g
      def tIntv( v:Double, tL:List[Double]=List(t0)):List[Double] = {
        if( v<=v0/1000) tL.reverse else {
          val v1 = cR*v
          val t1 = 2*v1/g
          tIntv( v1, t1 :: tL)
        }
    }
    val tV= tIntv( v0).map(_*bps).map(_.toInt).scan(0)(_+_).tail
//tL:List[Int] = List(43647, 87294, 109117, 120028, 125483, 128210, 129573, 130254,
130594, 130764, 130849)
   println(d.length)
   println(tV.mkString(","))

   val ta = tV(0)
   var fcD = 1.0
   tV.foreach(t=>{ val dt = ((t-ta)/4)*4
   fcD *= 0.9
   for(i<-1 to d0.length) d(i-1+dt) = (d(i-1+dt)+fcD*d0(i-1)).toByte})
```

```
}

  def dRead {
      try{
         header0 = header.map(i=>hD(i.name, (i.v).map(j=>fR0.readByte)))
         header0.map(i=>println(i.name+"  "+vL(i.v)))
         fR0.read(d0)
      }
      finally{ fR0.close; println("dRead")} //
   }
```

付録３　ASTS 破壊のプログラム(Scala)

```
//sounds for fall down and Break

object waveSound extends App{
   val wvS:Long = 44100 //Hz
   val ttPlay:Int = 2 //second
   val bps = wvS*2*2  //total bytes for 1 second
   val bPlay = bps*ttPlay //total bytes for (ttplay) seconds
   def s4(x:String) = x.map(_.toByte).toArray
   def b2(x:Int) = Array(x%256,x/256).map(_.toByte)
   def b4(x:Long) = {var x0=x; val y = new Array[Long](4)
       for(i<-0 to 3) {y(i)=x0%256; x0=x0/256}
       y.map(_.toByte)}

   val note = (0 to 72).toList.map(i=> 110.0*math.pow(2.0,i/12.0))
   val drm = List(27,29,31,32,34,36,38,39)


   case class hD(name:String, var v:Array[Byte])
   val header = Array(
      hD("Chunk ID",          s4("RIFF")),
      hD("Chunk Data Size", b4(bPlay+50)),
      hD("RIFF Type",         s4("WAVE")),
      hD("fmt -Chunk",        s4("fmt ")),
      hD("fmt -Data Size",   b4(18)), //16?
      hD("Compression code",b2(1)), //PCM/uncompressed
      hD("Number of channels",b2(2)),
      hD("Sample rate",       b4(wvS)),
      hD("bytes/second",      b4(bps)),
      hD("Block align",       b2(4)),
      hD("bits/sample",       b2(16)),
      //hD("Extra format bytes",b2(0)),
      //if Extra format bytes != 0 : add n bytes
      //hD("fact-Chunk ID",    s4("fact")),
      //hD("fact-1?",           b4(4)), //4
```

```
      //hD("fact-2?",          b4(0)), //4332672
      hD("data-Chunk ID",    s4("data")),
      hD("data-chunk size", b4(bPlay))
   )
   var header0:Array[hD] = header.map(i=>i)

   val fR0 = new java.io.RandomAccessFile("kash.wav","r")
   val fR = new java.io.RandomAccessFile("b3b.wav","rw")
   val d0 = new Array[Byte](bps.toInt/8) //for 1/8 second
   val d = new Array[Byte](bPlay.toInt) //for ttPlay second
   def vL(x:Array[Byte]):Long = x.map(j=>if(j<0) j+256 else j).
       reverse.foldLeft(0L)((s,i)=>s*256+i)  //4bytes -> Long

   def oSnd1 = dRead  //set sound data to d:Array
   val g = 9.8
  def sim( cR:Double, h:Double) = {
    val v0 = Math.sqrt( 2*g*h)
    val t0 = v0/g
      def tIntv( v:Double, tL:List[Double]=List(t0)):List[Double] = {
        if( v<=v0/1000) tL.reverse else {
          val v1 = cR*v
          val t1 = 2*v1/g
          tIntv( v1, t1 :: tL)
        }
    }
    val tV= tIntv( v0).map(_*bps).map(_.toInt).scan(0)(_+_).tail
//tL:List[Int] = List(43647, 87294, 109117, 120028, 125483, 128210, 129573, 130254,
130594, 130764, 130849)
   println(d.length)
   println(tV.mkString(","))

   val ta = tV(0)
   var fcD = 1.0
   tV.foreach(t=>{ val dt = ((t-ta)/4)*4
   fcD *= 0.8
   val fc = if(dt==0.0) 0.1 else fcD
```

```
    for(i<-1 to d0.length) d(i-1+dt) = (d(i-1+dt)+fc*d0(i-1)).toByte})
}

  def dRead {
      try{
         header0 = header.map(i=>hD(i.name, (i.v).map(j=>fR0.readByte)))
         header0.map(i=>println(i.name+"  "+vL(i.v)))
         fR0.read(d0)
      }
      finally{ fR0.close; println("dRead")} //
   }

   def dWrite {
      header0(1).v = b4(bPlay+50)
      header0(12).v = b4(bPlay)
      try{
         header0.foreach(i=>fR.write(i.v))
         fR.write(d)
      }
      finally{ fR.close; println("dWite")}
   }
   dRead
   sim( 0.6, 0.02);sim( 0.5, 0.07);sim( 0.2, 0.08);sim( 0.3, 0.01);
   dWrite
}
```

付録４　TS のプログラム(Scala)

```
object waveSound{
   val wvS:Long = 44100 //Hz
   val ttPlay:Int = 8 //second
   val bps = wvS*2*2
   val bPlay = bps*ttPlay
   def s4(x:String) = x.map(_.toByte).toArray
   def b2(x:Int) = Array(x%256,x/256).map(_.toByte)
   def b4(x:Long) = {var x0=x; val y = new Array[Long](4)
       for(i<-0 to 3) {y(i)=x0%256; x0=x0/256}
       y.map(_.toByte)}

   val note = (0 to 72).toList.map(i=> 110.0*math.pow(2.0,i/12.0))
   val drm = List(27,29,31,32,34,36,38,39)


   case class hD(name:String, v:Array[Byte])//wav ファイルヘッダ生成
   val header = List(
      hD("Chunk ID",          s4("RIFF")),
      hD("Chunk Data Size", b4(bPlay+50)),
      hD("RIFF Type",         s4("WAVE")),
      hD("fmt -Chunk",        s4("fmt ")),
      hD("fmt -Data Size",   b4(18)), //16?
      hD("Compression code",b2(1)), //PCM/uncompressed
      hD("Number of channels",b2(2)),
      hD("Sample rate",       b4(wvS)),
      hD("bytes/second",      b4(bps)),
      hD("Block align",       b2(4)),
      hD("bits/sample",       b2(16)),
      hD("Extra format bytes",b2(0)),
      //if Extra format bytes != 0 : add n bytes
      hD("fact-Chunk ID",    s4("fact")),
      hD("fact-1?",           b4(4)), //4
```

```
    hD("fact-2?",          b4(0)), //4332672
    hD("data-Chunk ID",    s4("data")),
    hD("data-chunk size", b4(bPlay))
)
val fR = new java.io.RandomAccessFile("TS.wav","rw")
val d = new Array[Byte](bps.toInt) //for 1 second


val sn = 100
val sntp = sn*ttPlay
var is = 0
var smtp = 1.0/sn
val xls = new Array[Double](sntp.toInt+1) //for 1 second smple

def oSnd1(t:Double):Double =  {
    val i = (1*t).toInt
    val f = note(36)//音源の決定
    math.sin(2*math.Pi*f*t)
}
    val vol0 = 15000//ボリューム
    val ox = 100.0 //観測者の位置(m)
    val sV =340.0 // 音速(m/s)
    val oV = 160*1000/3600.0  //  観測者の移動速度(m/s : 80km/h)
    val bD = 5.0 //反射板 a の距離
    var oxp = math.sqrt(ox*ox+bD*bD)
    var tp = 0.0
    val ed = -0.5//反射板 b の距離
    val eD = bD-2*ed

def sim(t:Double):Int = {
    val ox1 = (ox - oV*t)
    val ox2 = math.sqrt(ox1*ox1+bD*bD)
    val dt = ox2/sV
    val vol = vol0*(bD*bD)/(ox2*ox2)

    val ed2 = math.sqrt(ox1*ox1+eD*eD)
```

```
    val edt = ed2/sV

    val vole = vol0*(bD*bD)/(ed2*ed2)*0.5

   if(t>smtp){
   is=is+1
   val tp1 = (t-tp)
   tp = t
   val vp = -(ox2-oxp)/tp1
                oxp=ox2
   val fp = 440*(1+vp/sV)
   smtp = 1.0/sn*is
   xls(is) = fp
    }

    (vol*oSnd1(t-dt)+vole*oSnd1(t-edt)).toInt
}

def dCal(n:Int) {
   val t0 = 1.0/wvS
   for( i<-0 until wvS.toInt){
      val t = i*t0 + (n-1)
      val ix0 = sim(t)
      val ix = if(ix0<0) ix0+256*256 else ix0
      val y1 = (ix%256).toByte
      val y2 = (ix/256).toByte
      d(4*i  ) = y1
      d(4*i+1) = y2
      d(4*i+2) = y1
      d(4*i+3) = y2
   }
}

def dWrite {
   try{
      header.foreach(i=>fR.write(i.v))
```

```
          for(i<-1 to ttPlay) {dCal(i); fR.write(d)}
       }

      finally{ fR.close}

    def csW( n:String, sD:String) = {
    val sX = <pre> {"¥n"+sD+"¥n"} </pre>
    scala.xml.XML.save(n+".csv",sX,"UTF-8")
    }

    val r = (1 to sntp).map{i=>
   {1.0/sn*i+" , "+ xls(i-1)}}.mkString("¥n")

    csW( "TS",r)


    }
}
waveSound.dWrite
```