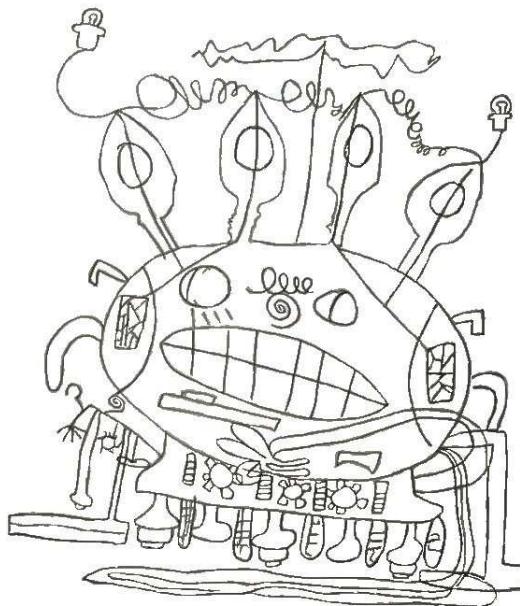


# ロボット知能 研究ノート



表紙絵：A. Honda

# 目次

<b>第Ⅰ部 理論的な側面</b>	<b>23</b>
<b>第1章 構成論的科学としてのロボット知能</b>	<b>27</b>
<b>第2章 反応行動のための知能</b>	<b>35</b>
2.1 走性における感覚行動写像 . . . . .	36
2.1.1 非交差性線形写像 . . . . .	37
2.1.2 超音波センサーによる走性 . . . . .	38
2.1.3 交差性線形写像 . . . . .	38
2.2 ゲインパラメータの取り得る範囲 . . . . .	40
2.3 抑制性感覚運動写像 . . . . .	41
2.4 非線形感覚運動写像 . . . . .	42
2.5 感覚運動写像で行動するロボット . . . . .	42
<b>第3章 収束する感覚行動写像</b>	<b>45</b>
3.1 1軸回転運動 . . . . .	46
3.2 数学的解析の概略地図 . . . . .	48
3.3 行列形式 . . . . .	49
3.4 行列の指數関数を用いた解 . . . . .	50
3.5 行列の固有値と固有ベクトル . . . . .	53
3.6 状態ベクトルに対する解 . . . . .	55
3.7 状態ベクトルの具体的なかたち . . . . .	56
3.8 無矛盾性の確認 . . . . .	59
3.9 振動 . . . . .	61

3.10 減衰振動 . . . . .	62
3.11 比例ゲインと積分ゲイン空間における相図 . . . . .	64
3.12 遷移行列を用いた漸化式 . . . . .	66
<b>第4章 漸化式と数値解析</b>	<b>69</b>
4.1 正しく機能するプログラムを書く . . . . .	69
4.2 構造化プログラミング . . . . .	70
4.3 ニュートン・ラフソン法 . . . . .	72
4.4 再利用して発展できるコード . . . . .	77
4.5 円周率 $\pi$ の近似値を求める . . . . .	79
4.6 数値解析における計算誤差 . . . . .	86
4.7 電子計算機内での数 . . . . .	86
4.7.1 整数型 . . . . .	86
4.7.2 実数型 . . . . .	86
4.7.3 少数点以下の2進数表記 . . . . .	88
4.7.4 有限桁数の2進数で表せない小数 . . . . .	91
4.8 オーバーフローとアンダーフロー . . . . .	92
4.9 誤差と有効精度桁数 . . . . .	94
4.10 丸め誤差 . . . . .	95
4.11 打ち切り誤差 . . . . .	95
4.12 方程式の解法における計算誤差 . . . . .	99
4.13 ガウスの消去法 . . . . .	101
4.13.1 前進消去 . . . . .	102
4.13.2 後退代入 . . . . .	104
4.14 ピボット選択 . . . . .	107
4.15 LU 分解 . . . . .	107
4.16 逆行列 . . . . .	110
4.17 行列の固有値と固有ベクトル . . . . .	111
4.17.1 LR 法 . . . . .	112
4.17.2 逆反復法 . . . . .	113

4.18 関数の近似 . . . . .	119
4.18.1 メリット . . . . .	120
4.18.2 最小二乗法 . . . . .	121
4.18.3 スプライン補間 . . . . .	126
4.19 常微分方程式の数値解 . . . . .	133
4.19.1 1階常微分方程式 . . . . .	134
4.19.2 ルンゲ・クッタ法 . . . . .	135
4.20 微分方程式を数値的に解く . . . . .	136
4.21 振り子運動の数値解析 . . . . .	138
4.22 数学的な準備 . . . . .	139
4.22.1 微分方程式 . . . . .	139
4.22.2 テイラー展開 . . . . .	140
4.23 オイラー法（微分方程式の数値解法） . . . . .	140
4.23.1 1次近似 . . . . .	141
4.23.2 2階微分方程式から連立1階微分方程式へ . . . . .	141
4.23.3 連立1階微分方程式に対するオイラー法 . . . . .	142
4.24 単振り子の数値解析 . . . . .	143
4.24.1 球座標 . . . . .	143
4.24.2 単振り子の運動方程式 . . . . .	145
4.24.3 解析的な近似解が得られる場合 ( $\theta' \ll 1$ ) . . . . .	147
4.25 数値誤差の蓄積とそれを低減させる方法 . . . . .	148
4.25.1 オイラー法による打ち切り誤差 . . . . .	148
4.25.2 修正オイラー法 . . . . .	149
4.25.3 自由振り子の数値解析 . . . . .	151
<b>第5章 時間遅れのある反応行動</b>	<b>157</b>
5.1 回転運動方程式 . . . . .	157
5.2 反応行動としての感覚運動写像 . . . . .	159
5.3 時間遅れと離散制御 . . . . .	164
5.4 ローターの回転運動における緩和時間 . . . . .	165

5.5 時間遅れを考慮した拡張遷移行列 . . . . .	168
5.6 フィボナッチ型遷移行列 . . . . .	172
5.7 時間遅れの固有値への繰り込み . . . . .	174

## 第 II 部 実践的な側面 181

<b>第 6 章 Debian をつかう</b> <span style="float: right;"><b>185</b></span>	
6.1 パッケージの管理 . . . . .	187
6.1.1 既にインストールされているパッケージの確認 . . . . .	188
6.1.2 DVD(CD) ドライブ内のパッケージ認識 . . . . .	188
6.1.3 パッケージ情報のアップデート . . . . .	189
6.1.4 パッケージの検索 . . . . .	189
6.1.5 パッケージのインストール . . . . .	189
6.1.6 ネットワークインストール . . . . .	190
6.1.7 amd64 アーキテクチャで i386 アーキテクチャを使 えるようにする . . . . .	190
6.2 エディター vi . . . . .	191
6.2.1 vi の動作設定 . . . . .	191
6.2.2 vi の使い方 . . . . .	192
6.2.3 GUI 上の vi . . . . .	194
6.3 ネットワーク . . . . .	194
6.3.1 IP アドレス . . . . .	195
6.3.2 名前解決 . . . . .	196
6.3.3 経路解決 . . . . .	198
6.3.4 ネットワークファイルシステム NFS . . . . .	198
6.4 kernel の構築とインストール . . . . .	199
6.4.1 構築 . . . . .	200
6.4.2 インストール . . . . .	201

<b>第 7 章</b>	<b>単純な反応行動</b>	<b>203</b>
7.1	きまぐれロボット . . . . .	203
7.1.1	LEGO ロボット . . . . .	204
7.1.2	アルゴリズム . . . . .	205
7.1.3	LEGO ロボットを動かす C 言語 . . . . .	205
7.2	EV3 上で Debian(ev3dev) . . . . .	211
7.2.1	インストールと起動 . . . . .	211
7.2.2	停止, 再起動 . . . . .	213
7.2.3	USB 有線 LAN を用いたセットアップ . . . . .	213
7.2.4	USB ネットワーク . . . . .	216
7.2.5	USB 無線 LAN . . . . .	216
7.3	PVM . . . . .	218
7.4	ev3c . . . . .	219
7.5	クロスコンパイルと Makefile . . . . .	219
7.5.1	感覚運動写像と PVM . . . . .	222
<b>第 8 章</b>	<b>ボードコンピュータ</b>	<b>227</b>
8.1	BeagleBone Black . . . . .	227
8.1.1	Debian のインストール . . . . .	228
8.1.2	Debian のイメージを直接 microSD にコピー . . . . .	231
8.1.3	ネットワークデバイスに付けられる番号 . . . . .	231
8.1.4	PIC とのシリアル (UART) 通信 . . . . .	232
8.1.5	USB 無線 LAN . . . . .	239
8.1.6	i2c でセンサーの値を読み取る . . . . .	241
8.1.7	PIN 配置 . . . . .	241
8.1.8	超音波センサー値を i2c で読み取る . . . . .	248
8.1.9	BBB の電源 . . . . .	250
8.1.10	PWM でモーターを制御する . . . . .	251

<b>第 9 章 無線 LAN</b>	<b>255</b>
9.1 ツールのインストール . . . . .	255
9.2 設定コマンド . . . . .	255
9.3 シェルスクリプト . . . . .	258
9.4 経路の追加 . . . . .	261
9.5 トラブルシューティング . . . . .	262
9.5.1 SIOCSIFFLAGS: Operation not possible due to RF-kill	262
9.5.2 Network-Manager を使うと DNS が破壊される . . . . .	263
<b>第 10 章 並列計算ライブラリ PVM</b>	<b>265</b>
10.1 インストールと開始 . . . . .	266
10.1.1 apt-get によるインストール . . . . .	266
10.1.2 dpkg を用いたインストール . . . . .	266
10.1.3 PVM を開始する . . . . .	267
10.2 C 言語で PVM を利用する . . . . .	270
10.2.1 ライブラリのインクルード . . . . .	271
10.2.2 構成情報の取得 . . . . .	271
10.2.3 プロセスの生成 (spawn) . . . . .	273
10.2.4 標準出力 . . . . .	275
10.2.5 データの送信と受信 . . . . .	275
10.2.6 遅延のないデータ送受信を行うために . . . . .	278
10.2.7 spawn されたプログラムが正常終了しないで残っている場合 . . . . .	279
10.2.8 整数型データの通信プログラム例 . . . . .	279
10.3 トラブルシューティング . . . . .	281
<b>第 11 章 グラフ作成 (GNUPLOT)</b>	<b>283</b>
11.1 gnuplot のインストール . . . . .	284
11.2 起動と終了 . . . . .	286
11.3 データ描画 (2D) . . . . .	286
11.4 文字の大きさ変更 . . . . .	289

11.5 制御コードファイル . . . . .	290
11.6 対数スケール . . . . .	292
11.7 EPS 出力 . . . . .	292
11.8 EPS 出力で複数のグラフを <code>replot</code> する . . . . .	292
11.9 LaTeX 文章へのグラフの組み込み . . . . .	293
11.10 EPS ファイル内の文字を日本語に置き換える . . . . .	294
11.11 C 言語ソースコードから直接グラフを作る . . . . .	296
11.12 簡易アニメーション . . . . .	298
11.13 演算結果をプロットする . . . . .	299
11.14 データを間引いてプロット . . . . .	299
11.14.1 ブロック . . . . .	299
11.14.2 <code>every</code> オプション . . . . .	300
11.15 範囲指定してプロット . . . . .	303
11.16 データを関数で fit する . . . . .	303
<b>第 12 章 飛行ロボット研究ノート</b>	<b>305</b>
12.1 10.4 号機, 飛行に初成功 . . . . .	305
12.1.1 ゲインなどのパラメータ . . . . .	306
12.1.2 角度, 各加速度データ . . . . .	306
12.2 11 号機 . . . . .	311
12.2.1 初実験 2014/5/17 . . . . .	312
12.2.2 BBB とバッテリー . . . . .	313
12.2.3 ドーム型プロテクター(v11.2) . . . . .	313
12.2.4 ゲインを Self-Avoiding Walk で学習するアルゴリズム . . . . .	316
12.2.5 飛行成功(v11.6) 2014/7/7 . . . . .	322
12.2.6 上下十字ナセルアームのフル型ローター(v11.8) . . .	324
12.2.7 下部センサーによる制御および制御レートの向上 (2014/7/16) . . . . .	327
12.3 v12(8/21 2014) . . . . .	330

12.4 1 3号機 . . . . .	330
12.5 8方向測距のための機体(v14.8.0, 11/20 2014) . . . . .	332
12.6 1 5号機 . . . . .	335
12.7 1 6号機 . . . . .	337
<b>付録 A シリアルコンソール</b>	<b>341</b>
A.1 シリアルポートの読み書き許可 . . . . .	341
A.2 minicom . . . . .	341
<b>付録 B CNC ノート</b>	<b>345</b>
B.1 2次元データの作成(tgif) . . . . .	345
B.2 軌道データの作成(PyCAM) . . . . .	345
B.3 LinuxCNC . . . . .	346
<b>付録 C 起動時に実行されるシェルスクリプト</b>	<b>347</b>
C.1 insserv . . . . .	347
C.2 update-rc.d . . . . .	348
<b>付録 D 数式処理 Maxima</b>	<b>349</b>
D.1 固有値と固有ベクトル . . . . .	349
D.2 右固有ベクトルと左固有ベクトル . . . . .	352
D.3 4つの固有値 . . . . .	354
D.4 遷移行列 $\hat{T}$ の固有値 . . . . .	354
D.5 固有値の1次摂動 . . . . .	355
<b>付録 E 分散バージョン管理システム gitlab</b>	<b>359</b>
<b>付録 F 日本語環境</b>	<b>363</b>
F.1 LaTeX . . . . .	363
F.1.1 インストール . . . . .	363
F.1.2 作業の流れ . . . . .	363
F.2 日本語入力 . . . . .	365

F.2.1	iBus をインストールする . . . . .	366
F.2.2	日本語入力方法を選択する . . . . .	366
F.2.3	Mozc で句読点を変更 . . . . .	367
<b>付録 G モーションキャプチャー</b>		<b>369</b>
G.1	カメラ・ハブ・PC の接続 . . . . .	369
G.2	アプリケーション起動 . . . . .	370
G.3	作業フォルダ . . . . .	370
G.4	Live mode と Postprocess . . . . .	370
G.5	キャリブレーション . . . . .	370
G.6	表示空間の移動 . . . . .	371
G.7	マーカセットの作成 . . . . .	371
G.8	データの取得 . . . . .	373
<b>付録 H pdf で使うフォント</b>		<b>375</b>
H.1	dvipdfmx でフォントを指定する方法 . . . . .	375
H.2	ps2pdf でフォントを指定する方法 . . . . .	376
関連図書	. . . . .	378
<b>索引</b>		<b>380</b>



# 図 目 次

1.1	環境の中で行動するロボット . . . . .	29
1.2	知能（プログラム）が進化するロボット . . . . .	30
1.3	構成論的科学としてのロボット知能の進化 . . . . .	30
2.1	環境と相互作用するロボット . . . . .	35
2.2	2つのセンサーと2つのモーターをもつロボット例(ncross) . . . . .	36
2.3	光センサーをもつ走行ロボット (ncross-light) と光源 . . . . .	38
2.4	超音波センサー（距離センサー）をもつ走行ロボット (ncross-ultra) と障害物 . . . . .	39
2.5	交差性結合をもつ走行ロボット (cross) . . . . .	39
2.6	(2.18) 式による非線形感覚運動写像の例 . . . . .	43
3.1	著者が開発した飛行ロボットの一例 . . . . .	45
3.2	真横からみた飛行ロボット . . . . .	47
3.3	$x$ -軸周りの飛行ロボットの回転角度 $\varphi$ とローター 1,3 による揚力 $L_1, L_3$ . . . . .	47
3.4	飛行ロボットの運動方程式に対する数学的解析地図. 行列の固有値・固有ベクトル, ベクトルの1次独立性, および行列の指數関数などの数学的知識が組み合わせて活用される. . . . .	49
3.5	$c_G = 0$ の場合, $\varphi(t), \dot{\varphi}(t)$ の時間変化は振動をつづける. . . . .	62
3.6	$\omega\varphi(t), \dot{\varphi}(t)$ をそれぞれ横軸と縦軸に表した空間を相空間と呼ぶ. 運動は, 相空間の軌道として表される. 振動運動は, 相空間における円軌道となる. . . . .	63

3.7 (a) 指数関数的な減衰の例. 減衰係数 $\gamma$ が大きくなるほど, 速く減衰する. (b) 振幅が指数関数的に減衰する $\cos$ 関数の例. . . . .	63
3.8 (a) $e^{-\gamma t}(\gamma/\omega) \sin \omega t$ の減衰の様子. (b) $e^{-\gamma t}((\gamma/\omega) \sin \omega t + \cos \omega t)$ の減衰の様子. . . . .	64
3.9 $c_G, c_A$ の関数としての $\text{Im}(\lambda_1)$ と $\text{Re}(\lambda_1)$ . . . . .	65
4.1 ニュートン・ラフソン法の原理 . . . . .	72
4.2 ニュートン・ラフソン法で $\sqrt{2}$ の値を求める. 適切にインデントされておらず, 変数名も混乱を招く. . . . .	74
4.3 適切なインデントと数学的表式を連想できる変数名を用いたプログラム. . . . .	74
4.4 goto 文を用いて $C < 0$ の場合の問題を避けたプログラム. 処理の流れを分かりやすくする可能性がある. . . . .	76
4.5 if 文だけを用いて $C < 0$ の場合の問題を避けたプログラム. while 文をまたいで, 処理を遠くに飛ばさない. また, インデントを用いて, 処理の構造を視覚的にとらえやすくする. . . . .	77
4.6 関数を用いてプログラムを記述すると, 数学的表式と類似性があるので, バグの防止に役立つ. また, 異なる関数についてのプログラムを作成するばあいに, メインルーチンを修正せずに済む. . . . .	78
4.7 半径 1 の円と, それに内接する正 $2^{n+1}$ 角形 ( $n = 1, 2, \dots$ ) . . . . .	80
4.8 多角形から, $\pi$ の近似値を求めるアルゴリズムの主な部分. 平方根の計算にニュートン・ラフソン法を用いた. . . . .	82
4.9 円周率 $\pi$ の近似値を, 円の内接多角形を用いて計算した結果例. 左の図から, $n = 5$ あたりから, 3.14 に近い値が得られていることがわかる. 右の図は, 左の図と同じ結果であるが, 縦軸を拡大して描画した. . . . .	83
4.10 正 $2^{n+1}$ 角形をつかって, $\pi$ の値を求めるプログラム例. 平方根の計算にニュートン・ラフソン法を関数として用いている. . . . .	85
4.11 実数を 32bit(4byte) で表した場合の符号, 指数部および仮数部 . . . . .	87

4.12 オーバーフローを観察するプログラム例 . . . . .	92
4.13 図 4.12 で示したプログラムの実行例 . . . . .	93
4.14 指数関数の値を、漸化式を用いて求めるプログラム例 . . . . .	96
4.15 $e^3$ を級数展開を用いて求めた例 . . . . .	97
4.16 $e^3$ を級数展開を用いて求めた場合の絶対誤差の変化 . . . . .	97
4.17 二分法によって、解の存在する区間を縮小していく . . . . .	100
4.18 ガウスの消去法の実行例 . . . . .	106
4.19 最小二乗法を用いて、関数 $f(x)$ を $n$ 次多項式 $p_n(x)$ で近似する . . . . .	120
4.20 3 次多項式によって、最小二乗法を用いて 2 次元データの関数近似を行った例. . . . .	125
4.21 補間法ではすべての点を通る関数を求める . . . . .	126
4.22 データ例. 図中の曲線は最小二乗法で求めたものである。スプライン曲線と異なり、データ点を通らないことがわかる. . . . .	131
4.23 スプライン補間曲線を求めるアルゴリズムの概略 . . . . .	133
4.24 3 次関数で補間されたスプライン曲線 . . . . .	134
4.25 例えば天気予報も数値解析の結果を利用している. . . . .	137
4.26 $y(t+h)$ の 1 次近似. . . . .	142
4.27 オイラー法のアルゴリズム概略を表す PAD 図. . . . .	144
4.28 球座標 . . . . .	145
4.29 単振り子は $\theta = \pi$ の周りで振動する . . . . .	146
4.30 振り子に働く力 . . . . .	147
 5.1 4 回転翼飛行ロボット上の $xy$ 座標軸およびローター ( $k = 1, 2, 3, 4$ ) の配置. . . . .	158
5.2 $x$ -軸周りの飛行ロボットの回転角度 $\varphi$ とローター 1,3 による揚力 $L_1, L_3$ . . . . .	158
5.3 揚力 $L_k$ は、ローター角速度 $\omega_k$ の 2 次関数である。ローターの番号 $k = 1, 2, 3, 4$ に対して共通. . . . .	159

5.4	飛行ロボットのシステム構成概略図	160	
5.5	$f(\Omega, \infty)$ の実測値をつかったフィッティング. $a = 9.453[\text{rad/s\%}], b = 124.81[\text{rad/s}]$	162	
5.6	ローター操作値を 45%から 50%に増やした場合のローター回転速度の時間変化. $\tau_+ = 47.4[\text{ms}]$ . ローター操作値を 40%から 50%に増やした場合のローター回転速度の時間変化. $\tau_+ = 48.5[\text{ms}]$ .	168	
5.7	ローター操作値を 50%から 45%に下げた場合のローター回転速度の時間変化. $\omega_p = 0.493[\text{rad/ms}], \omega(0) = 0.532[\text{rad/ms}], \tau_- = 41.4[\text{ms}]$	168	
5.8	$\hat{\Lambda}$ 行列の固有値 $\lambda$ に対する繰り込みアルゴリズム	177	
5.9	$\lambda$ の時間遅れによる変化. $\text{Re}(\lambda') > 0$ となる場合の例	177	
5.10	$\lambda$ の時間遅れによる変化. $\text{Re}(\lambda') < 0$ となる場合の例	178	
5.11	修正された $\lambda'$ の実部が負になる領域地図 $\delta = 70[\text{ms}]$	178	
5.12	修正された $\lambda'$ の実部が負になる領域地図 $\delta = 100[\text{ms}]$	179	
5.13	時間遅れが繰り込まれた $\lambda'$ を用いた状態 $\varphi(t)$	179	
6.1	vi(gvim) におけるモードの切り替え	192	
6.2	gvim の実行例.	195	
7.1	(a) レゴ標準ロボット	(b) 「知能」であるプログラ ム例	203
7.2	NXT にはモーターをつなぐインターフェイス (A,B,C) と 4 種類の センサーをつなぐインターフェイス (1,2,3,4) がある. PC と NXT は USB インターフェイスで接続する.	204	
7.3	PAD の構成要素. (a) 演算, (b) 繰り返し, (c) 条件分岐, (d) 関数・サブルーチン	205	
7.4	(a) 1 秒前進して 1 秒後退するためのアルゴリズム. (b) 1 秒 前進して 1 秒後退を無限に繰り返すためのアルゴリズム	206	
7.5	1 秒間前進, 1 秒間後退のプログラム. 図 7.4(a) のアルゴ リズムに対応	206	

7.6 1秒間前進、1秒間後退を繰り返すプログラム。図 7.4(b) のアルゴリズムに対応 . . . . .	207
7.7 4種類のセンサーを有効にする関数 . . . . .	207
7.8 超音波センサーからの入力情報で条件分岐する . . . . .	209
7.9 光センサーからの入力で条件分岐する . . . . .	209
7.10 確率的に方向を変えるための if 文 . . . . .	210
7.11 センサーからの値を変数に記憶する . . . . .	210
7.12 EV3 で microSD から ev3dev を起動した直後の液晶画面 . . . . .	212
7.13 起動リセットを実行する場合は、図中 A と B のボタンを 同時に長押し（数秒）する. . . . .	212
7.14 EV3 の USB ポートに有線 LAN を接続する. . . . .	213
7.15 EV3 と Debian PC によるネットワーク構成 . . . . .	214
7.16 ev3dev における/etc/resolv.conf の例 . . . . .	215
7.17 ev3dev における/etc/apt/sources.list の例 . . . . .	215
7.18 ev3c/というディレクトリがあるディレクトリでクロスコ ンパイルするための Makefile . . . . .	220
8.1 BBB にシリアルケーブルを接続する. . . . .	230
8.2 BBB に USB 無線 LAN アダプタとシリアルケーブルを接 続する. . . . .	240
8.3 BeagleBone Black の P8, P9 ポートピン配列。P9-17, P9-18 を i2c-2 として利用する. . . . .	242
8.4 BeagleBone Black の 5V 電源をレギュレータなどからとる ための配線 . . . . .	251
9.1 GNOME のタスクバーに表示された Network-Manager . . . . .	263
11.1 gnuplot で複数のデータをグラフ化する . . . . .	284
11.2 図 11.1 を gnuplot で描画するためのコード . . . . .	284
11.3 3次元構造を可視化する . . . . .	285
11.4 図 11.3 を gnuplot で描画するためのコード . . . . .	285

11.5 単純な plot 結果 . . . . .	287
11.6 点のサイズを大きく (3) する . . . . .	287
11.7 点と点を線でつなぐ . . . . .	288
11.8 横軸と縦軸にラベルをつける . . . . .	288
11.9 描画する範囲を指定する. . . . .	289
11.10 2つのデータを同時に描画する. . . . .	289
11.11 フォントを "Helvetica,15" に指定. . . . .	290
11.12 title を付ける . . . . .	291
11.13 gnuplot の制御コードファイル例 . . . . .	291
11.14 グラフを EPS 出力するための gnuplot の制御コードファイル例 . . . . .	292
11.15 gnuplot から出力された EPS グラフ . . . . .	294
11.16 EPS ファイル内の文字を日本語に置き換える . . . . .	295
11.17 psfrag と ps2pdf コマンドを使って eps ファイルを生成するための LaTeX コード . . . . .	295
11.18 popen を使って, C 言語から直接 gnuplot を使う例 . . . . .	297
11.19 tail コマンドと reread を使って, 簡易アニメーションを行うための gnuplot コード . . . . .	298
11.20 簡易アニメーションの例 . . . . .	298
11.21 ふたつのブロックにわかれたデータのプロット例 . . . . .	300
11.22 every 2 オプションを用いて, データを 2 行ごとにプロット . . . . .	301
11.23 every オプションを用いて, 開始行と終了行を指定 . . . . .	302
11.24 値の範囲を指定してプロットした例 . . . . .	303
12.1 4 回転翼飛行ロボット (v10.4), 飛行の様子 . . . . .	305
12.2 飛行実験結果 MAP . . . . .	306
12.3 安定時の角度センサー値 . . . . .	306
12.4 安定時角速度 (gyro) センサー値 . . . . .	307
12.5 安定時モーター制御値 . . . . .	308
12.6 不安定時の角度センサー値 . . . . .	309

12.7 不安定時角速度 (gyro) センサー値 . . . . .	309
12.8 不安定時モーター制御値 . . . . .	310
12.9 4回転翼飛行ロボット (v11.0) . . . . .	311
12.10 v11.0 号機裏 . . . . .	311
12.11 スロットル値と各センサー値の例 . . . . .	312
12.12 v11.0 号機に搭載した BBB . . . . .	313
12.13 バッテリー搭載は、4mm 厚シナベニアを 3mm ネジ x4 で 止める . . . . .	313
12.14 v11.2 号機にドーム型のプロテクタを上下につけた . . . . .	314
12.15 BBB やバッテリーを保護すると同時に、水平状態から可 動するので安定ゲインを調べる実験を床のうえで直接行える . . . . .	314
12.16 加速度と角速度の観測結果. 振動の周期から $\omega$ の値を仮 定できそう. ((3.53) 式) . . . . .	315
12.17 PVM を利用してゲイン値を変更しながら飛行実験するア ルゴリズム . . . . .	316
12.18 飛行実験したゲイン (2014.6/2) . . . . .	317
12.19 2 次元 Self Avoiding Walk のアルゴリズム (2014.6/4) . . . . .	318
12.20 11.4 号機に搭載された Hyperion 1709 と 7x3.5 ローター . . . . .	320
12.21 上部ドーム状補強アームの破損状況 . . . . .	321
12.22 v11.6 の構成 . . . . .	323
12.23 v11.8 の構成. 上下十字ナセルアームとプル型ローター . . . . .	325
12.24 v11.8 のセンサーデータ例 (take3,take4) . . . . .	326
12.25 下部センサーを設置. i2c-2 で値を取得 (address X069) . . . . .	327
12.26 下部センサーによる制御データ例 (take1x,take1y) 飛行時間 約 1.4 秒 . . . . .	328

12.27 室内で安定に飛行可能であり、なおかつ 8 方向に測距センサーを配置が容易な機体。写真では、モーターの下部に 4 つの超音波センサーが設置されている。40cm x 40cm，重量はバッテリー含めて約 300 g. バッテリーは 2 セル 7.4V, 600mAh だが、1 つだと BBB もふくめた機体全体に対して電流供給が不足ぎみで、モーターが破損する傾向があるため、2 個を並列使用。 . . . . .	333
12.28 左上： BBB のピンソケットに直接機体の基板を接続した。 右上：加速度とジャイロセンサーは機体下部（写真上部）に配置する。振動を抑えるために、脚部との接続は EPP を利用。左下：ローターは GWS5030 サイズの 3 枚ペラ。プロテクターをモーターマウントに直接ネジ止め。着脱も容易である。右下：モーターはコスモテック CT1811-3800. 非常に小型軽量だが、リード線がエナメル線でハンダ付けはしにくい。また電源供給が不安定だと破損するようだ。 SRF02 はモーターダウン部に配置。アームは気流の流れを考えて基板に対して縦に配置した。 . . . . .	334
12.29 15.4 号機：超音波センサー (SRF02 を 4 方向)。3 葉の折プロペラ . . . . .	335
12.30 マルチスピード SMM . . . . .	336
12.31 MS-SMM アルゴリズム . . . . .	336
12.32 16.2 号機 . . . . .	337
12.33 16.2 号機の構造と部品配置 . . . . .	338
12.34 16.2 号機の基板部（ユニバーサル基板）およびバッテリマウント部品（4mm 厚シナベニア）. . . . .	339
12.35 16.2 号機のアーム部品。4mm 厚シナベニア . . . . .	339
12.36 左から、16.2 号機の基板—アーム結合部品（4mm 厚シナベニア）。モーターマウント部品（4mm 厚シナベニア）。モーターマウント EPP（40mm の EPP）. . . . .	340

A.1	minicom -s を実行した画面	342
A.2	シリアルポートを選択した画面	342
A.3	”F”を入力し, ハードウェアフロー制御「いいえ」を選択	343
A.4	「”df”に設定を保存」を選択し, 設定を保存する.	343
A.5	ボードコンピュータのログインプロンプトが表示される.	343
D.1	Maxima で, フィボナッチ型遷移行列 $T_F$ の定義とその固有値, 右固有ベクトルを求めるためのバッチファイル (Fibonatch.max)	349
D.2	Maxima の起動と, バッチ処理の読み込み	350
D.3	固有値と右固有ベクトルの出力結果. 固有値, 固有値の縮退度, 固有ベクトル (行ベクトル) の順に出力される.	356
D.4	3 番目の固有値の表示, およびその簡略化置き換え (ratsubst)	357
D.5	フィボナッチ型遷移行列の 3 番目の固有値とそれを ratsubst コマンドで簡略化した結果	357
D.6	右固有ベクトルから変換行列 $\hat{V}$ をつくる.	357
D.7	右固有ベクトルが固有ベクトルになっていることの確認	357
D.8	フィボナッチ型行列とその右固有ベクトルの掛け算から得られた固有値の確認結果.	358
D.9	左変換行列 $\hat{U}$ を求める.	358
D.10	左変換行列と右変換行列の積が単位行列となることの確認結果.	358
E.1	gitlab ワークフローの概略	361
F.1	iBus の設定. インプットメソッドとして Mozc や Anthy を追加する.	366
F.2	Mozc プロパティ変更	367
H.1	モトヤフォントをつかった, map ファイルの例	375



# 第I部

## 理論的な側面



ここには、ロボット知能を考える上で基本的に必要になる事柄をまとめた。

反応行動のための単純な感覚運動写像を中心に、知能的な行動の創発という考え方を紹介する。

また、反応行動の基本として系が振動する場合について、基礎数学を用いた解析的な議論と数値解析的な方法も紹介する。時間遅れが存在しない場合、系が不安定になることは理論的にはありえず、せいぜい振動するだけであることを示す。

いっぽう、飛行ロボットの様な行動の特徴的な時間長さがミリ秒単位となるような、系の安定性を考える場合には、身体性がもつ時間遅れがもたらす影響が無視でない。むしろその行動を大きく左右する例を紹介する。反応の時間遅れは、系を大きく不安定にすることを示す。



# 第1章 構成論的科学としてのロボット知能

ロボット知能の研究とは、簡単に言えば人間の持っているような知能を創って、ロボットにそれをもたせようという試みと言えるであろう。似たような言葉として、人工知能というものがある。たとえばよく知られている例をあげると、コンピューターが将棋のプロ棋士に勝ったという話が話題になることがあることはご存知だろう。ここでいうコンピューターというのは、高速な演算処理をすることが可能な計算機ハードウェアと、そこにインストールされた、実際に将棋をさすプログラム（ソフトウェア）双方を一体としてコンピューターと呼んでいる。将棋プログラムは人工知能の典型的な例である。

指し手の数は膨大であるが、有限であることには違いない。巧妙に最善手を探し出すアルゴリズムを考えだされれば、記憶力も計算力も人間より桁外れの能力をもつコンピューターは人間の棋士においつき、いずれ追い越してしまうであろう。将棋のような枠組みがはっきりと定義されている問題については、人工知能のエキスパートシステムはかなりの性能を発揮することが出来るのである。このような能力も、もちろん知能であることには違いない。

われわれ人間が遭遇する状況は、このように枠組みがはっきりしている場合ばかりとは限らない。むしろ、そうではない場合がほとんどであろう。判断の条件や状況は刻一刻と変化しているし、どこまで判断の条件に含めて考えればよいのかも曖昧である。そのような状況でも、われわれは何がしかの判断を下し、適切と思われる行動を起こしている。

知能がどのように構成されていて、どのようなメカニズムの下ではた

らいているのか、はっきりわかっていれば、知能を創ることが可能かもしれない。しかし、それらがはっきり解明されていないばかりか、知能とは何か、どういうことなのかさえはっきりと定義できないのが現状である。

### 構成論的科学

そこで、最も知能的な存在であると考えられる人間を、今までの科学が用いてきた分析的な手法で研究しようとするのは、自然な考え方かもしれない。分析的な手法とは、いわば対象を徹底的に細かく細分化していき、それらの細分化された構成要素の働きの総体として、元の対象を理解しようとする方法である。例えば、物質の性質を、原子や分子が多数集まった総体として理解したり、半導体のように活用したりしようとする。このような物質科学とよばれる分野がもっともその典型的な例であろう。

ところが、知能の研究において分析的手法を用いるということは、おのずと脳を細分化して、その生物学的な性質や、電気的な機能などを調べることになる。このような分析的な努力は多くの研究者によって進められている。しかし、物質科学と同様な分析的な手法を用いることが困難であるということは、容易に想像できるであろう。たとえば、物質科学の場合のようにX線を照射して構造を調べたり電子顕微鏡でのぞいたりすることが、生きた脳に対して行えないということである。もし、行つたとしても、それらの外部刺激の影響が強すぎて、知能の働いている現場をとらえることができないということも考えられる。

そこで、構成論的科学として研究を進める方法を考える。まず「知能」を構成して、つまり、創って、それを実環境の中で行動させる。ここで、われわれ人間などが持っているような生物の知能と区別するために、構成論的に創ろうとする知能の方を「知能」と表した。「知能」はコンピューター内のいわばソフトウェアであるから、かたちとしては、その中にプログラムとして作り込むだけでも十分と思われるかもしれない。しかし、

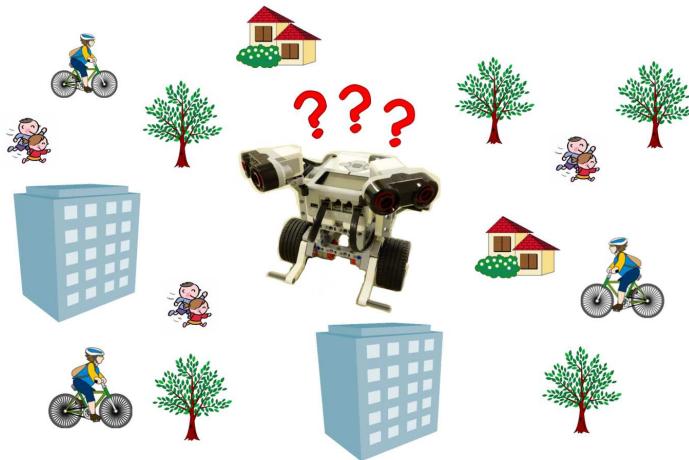


図 1.1: 環境の中で行動するロボット

構成論的科学としてのロボット知能の研究においては、「知能」と「身体」をもったロボットが変化に富んだ実環境の中で行動することが必要であると考える（図 1.1 参照）。もちろん、その「知能」で多様な実環境の中を行動するのに十分であるとは考えられない。生物であれば、そのような知能をもった生物は十分環境に適応していないということで、淘汰されるであろう。すなわち実環境に適した知能（行動原理）をもった生物だけが生き残る。

いっぽう、ロボットの場合にはその「知能」を再構築することによって同じ身体をもつロボットを再利用することができる。構成論的科学とは、いばば「知能」と「身体」を最構成しながら、生物がたどった進化の過程を再現することを通じて、知能を理解しあるいは活用しようというアプローチである。

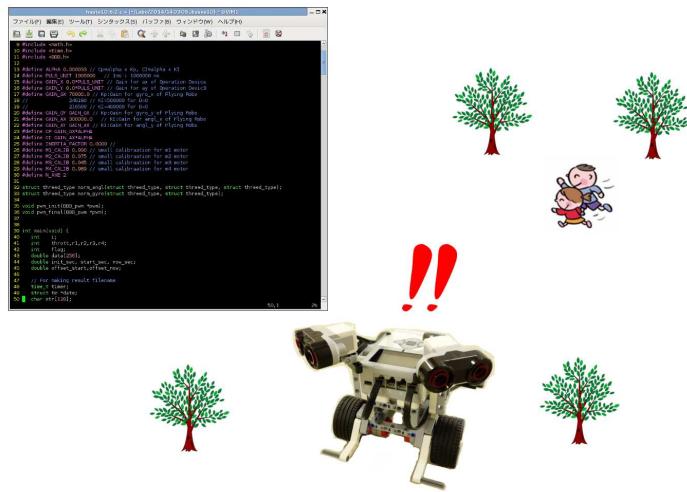


図 1.2: 知能（プログラム）が進化するロボット



図 1.3: 構成論的科学としてのロボット知能の進化

### ロボットの身体性

上でも述べたように、構成論的に知能を研究しようとすると、その「知能」を持った機械がある環境の中で行動することが必要である。いわば脳だけではだめで、身体があって「知能」を活用して行動することが必須である。

「知能」を搭載する機械のことを、筐体、そして「知能」と筐体の総

体のことを、ここではロボットと呼ぶことにしよう。ロボットという言葉の語感からは、アトムやアシモのような人間型の機械を想像してしまうかもしれない。本書では、広く「知能」を搭載した、行動（運動）することのできる機械（筐体）のことをロボットと呼ぶ。

自律運転の「知能」を搭載した自動車なども、単なる人間が操縦する機械ではなくて、もはやロボットと呼んでよいであろう。あるいは、自動飛行できるジャンボジェット機も、広い意味ではロボットと呼んだ方がよいかもしれない。

これらの例で言うと、自動車や飛行機の機械の部分のもつ性質のことを身体性と呼ぶ。また、筐体はロボットの「身体」と呼んでもよいであろう。

自動車と飛行機の例からも想像できるとおり、「知能」の構造や機能は身体性と大きく関わっている。あるいは、同じ「知能」であったとしても身体性の違いによって、生じる行動は大きく異なる可能性が高い。

## 創発

ロボットの行動は「知能」と「身体」だけで決定されるわけではない。どのような外部環境のなかでロボットが行動するかによって、生成される行動は異なる。いわば、「知能」、「身体」および環境の3者がダイナミックに相互作用するなかで、行動が現実に現れてくる。このことを創発とする。

行動のための知能もそうであるが、様々なシステムはこの様に多くの構成要素からなっている。環境のなかでシステム全体が機能する時、それぞれの構成要素の機能の単なる和としてではなく、構成要素それぞれがもともと持っていないような機能が現れることを創発と言うこともできる。個別分断的に構成要素を観察しても、全体としての機能を理解することはできない。

ロボットが置かれる環境は、人間が置かれている環境とは一般に異なる可能性がある。たとえば、深海に潜るロボットであったり、原子炉の

中に入るロボットであったり、空中を浮遊するロボットをイメージすれば、人間が普段置かれている環境条件とは異なる環境条件の中でロボットが行動することがわかる。

「知能」、「身体」および「環境」が相互作用することによって創発する行動がロボットインテリジェンスであるのだから、どのような環境条件の中で「知能」を実現するかということに依存して、結果として得られるロボット知能の形態は、人間のそれとは異なったものになることも十分考えられる。

### 飛行機の歴史から

飛行機の歴史を思い起こしてみる。鳥は、翼を利用して空を飛ぶ人間の先輩である。最初、人間は、翼が羽ばたくということに飛行のメカニズム原理が潜んでいると考えた。いまから振り返って考えると滑稽であるが、人工の翼を腕につけて多くの人々が飛行を試みたが、失敗に終わった。

その後、ライト兄弟につながる先人達は、鳥を徹底的に観察し、グライダーや飛行機を作って飛んだ。失敗を繰り返して飛行機の発明に至るその過程は、まさに飛行のメカニズムを解明するための構成論的科学であると言える。実際に飛行機を作って飛ばすという過程の中で、翼が空気中を前進することによって揚力が発生するという飛行のメカニズムが解明されたのである。

現代のジャンボジェット機を見れば分かる通り、鳥を観察して構成された飛行機は、鳥とは異なる形をしている。しかし、推進力を生み出す方法が異なっているのであって、揚力を生み出す原理は両者に共通しているのである。

### ライト兄弟になれるのか

知能の研究の中で、まだわれわれはライト兄弟以前にいるということになる。知能ロボットを作って、実環境の中で行動させてみるということ

とを繰り返すことで前進しようとしている。先にも述べたとおり、このことを、構成論的科学と呼ぶ。一気に高度な知能を構成しようとすることは、もちろん困難であろう。やはりもっとも基本的な、あるいは単純な知能から出発する。

そのために、ロボットの「知能」、すなわちロボット知能を大きく4つに分類して考えることからはじめる[1]。

ひとつ目は、反応行動のための知能である。感覚器やセンサーから入力してきた情報にたいしてある定型的な反応を示す行動のことである。ここで、注意が必要である。もっとも単純であると考えられる反応行動においても、生成される（創発される）行動が定形的であるとは限らない。定型的なのは、反応を決めるメカニズム（感覚行動写像）である。反応のメカニズムが定型的であったとしても、複数の感覚入力が組み合わせられたり、環境との相互作用を通じて、複雑な行動が創発される可能性がある。

また、反応行動と混同しやすい言葉として、反射行動がある。反応行動の一種として反射行動が含まれる場合もあるが、反射行動とは、行動そのものが定形的な行動のことを指す。たとえば、熱いものに触ったときに、瞬間的に手を離す行動などである。

4つの知的行動の2つ目は、計画行動のための知能である。地図や経路情報をもとに行動の計画を構成する知能といえる。また、実際の行動から地図を構成する学習過程もこの知能に含めて考えられる。

3つ目が、適応行動のための知能である。変化する環境の下では、計画行動だけでは適応できない条件の変化に対応する能力が必要とされる。

4つ目として、協調行動のための知能がある。複数の知能を持った行動体が、それぞれの意図を理解し合いながら、目的を達成するための知能である。

これらの4つの知能は、それぞれの間に明確に線引きできるわけではなく、またそれが補いあったり、関連しあって全体として機能すると考えられる。ロボット知能を構成論的に考える上で、このように分けて考えることから出発しようとするわけである。本書では、これら4つ

の知能のなかで、主に反応行動のための知能に着目する。

### Debian をつかう

ロボットの「知能」は、具体的にはコンピューター内のプログラムの形をとる。なおかつ、環境のなかで実際に行動するロボットに搭載されるコンピューターなので、PCなどとことなり、小型かつ軽量であることが必要である。そのために、ボードコンピューターを利用する。ボードコンピューターにネットワーク接続してプログラムなどの調整を行う必要があるが、ロボットに搭載されているということもあり、PCのような入出力装置が存在しない。つまり、キーボードとディスプレイがない状態でプログラミングを行う必要がある。実際にはボードコンピューターにそれらを接続して利用することも可能であるが、構成論的に「知能」を構築する過程で、入出力装置をロボットに接続したり切り離したりという作業を常に行なうこととは、現実的ではない。

本書では、ボードコンピュータおよび基地となるPC上で利用するオペレーティングシステムとして Debian/GNU Linux（以下、単に Debian とよぶ）を用いる。Debian は GNU という名前がついていることからもわかるとおり、オープンソースとして管理されており、ロボット知能の研究のためには最適な OS のひとつとかんがえられる。

## 第2章 反応行動のための知能

ロボットは、図 2.1 に示したように、センサーを通じて環境からの情報を取り込む。その情報を基に行動を生成し、モーターを通して環境に働きかける。環境はロボットからの行動を受けて、そのダイナミクスにたがって変化し、それを再びロボットがセンサーで受け取る。ロボットが環境の中で行動することは、このようなロボットと環境がつくるループを繰り返して行くことである。ロボットに着目すると、その構成要素

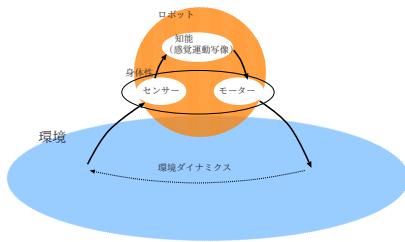


図 2.1: 環境と相互作用するロボット

はセンサー、感覚行動写像（知能）、モーターの 3 つに大別される。動物でいえば、それぞれ感覚器官、中枢神経系、筋肉に対応する。

センサーをロボットのどの部分に、どのような角度でつけるのか、あるいはモーターの数や大きさ角度をどのように形成するのか。これらの配置は、実際に作り出されるロボットの行動に大きく影響するであろう。このような要因は、身体性と呼ばれる。

一方、行動を決定しているのは中枢神経系であり、ロボットであればコンピュータでそれを決定することになる。反応行動を実現するメカニズ

ムの一つとして、感覚行動写像がある[1].

## 2.1 走性における感覚行動写像

感覚入力は一つだけではなく、複数の値が考えられるのでベクトル  $\vec{s}$  でそれを表す。また、行動出力も複数ある場合を考えてベクトル  $\vec{r}$  で表す。従って、感覚行動写像は、感覚ベクトル  $\vec{s}$  から行動ベクトル  $\vec{r}$  への写像  $f$  となる。

$$f : \vec{s} \rightarrow \vec{r} \quad (2.1)$$

たとえば、図2.2に示したように、2つのセンサーと、2つのモーターを持つ走行ロボットを考えてみる。左側のセンサー値を  $s_L$ 、右側のセン

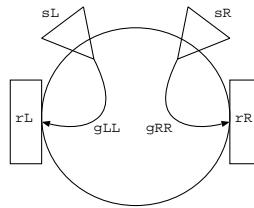


図2.2: 2つのセンサーと2つのモーターをもつロボット例(ncross)

サー値を  $s_R$  とする。また、左側のモーター出力を  $r_L$ 、右側のモーター出力を  $r_R$  とする。感覚ベクトル  $\vec{s}$  と、行動ベクトル  $\vec{r}$  は、それぞれ

$$\vec{s} = \begin{pmatrix} s_L \\ s_R \end{pmatrix}, \quad \vec{r} = \begin{pmatrix} r_L \\ r_R \end{pmatrix} \quad (2.2)$$

と表すことができる。

### 2.1.1 非交差性線形写像

図 2.2 に示した走行ロボットは、感覚入力と行動出力が左右で非交差性結合をしている (noncrossover link) ので、このロボットを ncross ロボットと呼ぶことにしよう。

センサーからの入力値  $s$  にゲイン  $g$  を掛けてそのまま出力値  $r$  とする感覚行動写像の場合、

$$\begin{aligned} r_L &= g_{LL}s_L \\ r_R &= g_{RR}s_R \end{aligned} \quad (2.3)$$

と書ける（線形写像）。ただし、左 (L) のセンサーからの入力を左のモーターに出力する際のゲインを  $g_{LL}$  また、右 (R) のセンサーからの入力を右のモーターに出力する際のゲインを  $g_{RR}$  と表した。行列  $\hat{G}_{\text{ncross}}$  を使ってこれを書くと、

$$\vec{r} = \hat{G}_{\text{ncross}} \vec{s} \quad (2.4)$$

$$\hat{G}_{\text{ncross}} \equiv \begin{pmatrix} g_{LL} & 0 \\ 0 & g_{RR} \end{pmatrix} \quad (2.5)$$

と、ゲイン行列  $\hat{G}_{\text{ncross}}$  は、 $2 \times 2$  の対角行列で表すことができる。

ゲインの値が正の場合、センサー値が大きくなればなるほど、モーター出力も大きくなるので、このことを興奮性結合とよぶ。すなわち、 $g_{LL} > 0, g_{RR} > 0$  のことを興奮性結合という。

このようなセンサーとモーターで構成される比較的単純な走行ロボットにおいてお、センサーの種類によって様々な走性が生じる。

#### 光センサーによる走性

図 2.3 に示したように、センサーを光センサーとして、光源が左前方にある場合、走行ロボットはどのような振る舞いをするだろう？

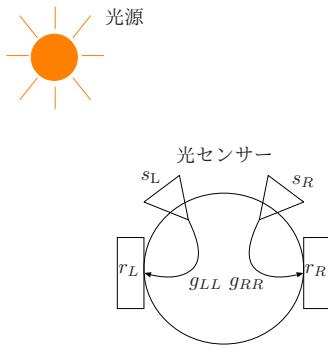


図 2.3: 光センサーをもつ走行ロボット (ncross-light) と光源

左側のセンサーの値の方が大きいので、興奮性結合をもつロボットの場合、左のモーターの方が出力がより大きくなるであろう。結果として、光から遠ざかろうとする行動が生じると考えられる。

### 2.1.2 超音波センサーによる走性

センサーが、超音波センサー、すなわち距離センサーである場合にはどのような行動が生じるだろうか？(図 2.4 参照) 興奮性結合の場合、右側のセンサーの方が障害物までの距離が大きいので、出力も大きく、結果として、障害物の方へぶつかっていく行動が生じると予想される。

### 2.1.3 交差性線形写像

図 2.5 に示した様に、センサーとモーターの結合が交差している場合を cross ロボットと呼ぶことにしよう。

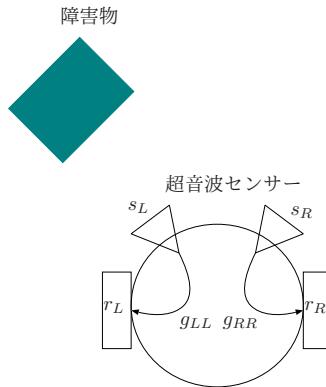


図 2.4: 超音波センサー（距離センサー）をもつ走行ロボット (ncross-ultra) と障害物

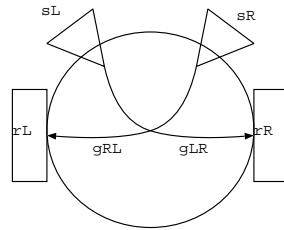


図 2.5: 交差性結合をもつ走行ロボット (cross)

このロボットの場合、出力  $\vec{r}$  は

$$\vec{r} = \hat{G}_{\text{cross}} \vec{s} \quad (2.6)$$

$$\hat{G}_{\text{cross}} \equiv \begin{pmatrix} 0 & g_{RL} \\ g_{LR} & 0 \end{pmatrix} \quad (2.7)$$

と表される。

センターが光センサーの場合、光のある方と逆のモーターの方が出力が大きくなるので、光の方に向かっていく走性を示すと予想される。

いっぽう、距離センサーを用いた場合、逆に障害物を避ける動きが予想される。

## 2.2 ゲインパラメータの取り得る範囲

ここまででは、簡単のためにゲインパラメータ  $\{g_{LL}, g_{RR}\}$  などの値については詳しく述べなかった。これらの値が正の場合、興奮性結合、すなわち、センサー値が大きくなればなるほどモーター出力値も大きくなる場合を述べた。つまり、原理的にはモーター出力値はいくらでも大きな値を取りうる。

しかし、実際のロボットにおいてはモーターに対して無制限に大きな出力を動作させることは不可能である。センサー入力値の取り得る範囲を考慮して、モーター出力可能な範囲になるようにゲインパラメータの値を決める必要がある。

センサー値  $s$  の値が以下の範囲の値をとるとする。

$$s_{\min} \leq s \leq s_{\max} \quad (2.8)$$

線形写像の場合、センサー値にゲインを掛けたものをモーター出力値とするので、

$$\begin{aligned} gs_{\min} &\leq gs \leq gs_{\max} \\ gs_{\min} &\leq r \leq gs_{\max} \end{aligned} \quad (2.9)$$

である。

モーター出力値がとるべき範囲を

$$r_{\min} \leq r \leq r_{\max} \quad (2.10)$$

とすると、 $g$  の最大値に関して、

$$\begin{aligned} gs_{\max} &\leq r_{\max} \\ g &\leq \frac{r_{\max}}{s_{\max}} \end{aligned} \quad (2.11)$$

という条件を満たさなければならない。ここで、センサー値はセンサーからの光の強さや、障害物までの距離の値であるから、 $s_{\max} > 0$ と考えてよいことを用いた。

同様に、 $g$  の最小値について

$$g \geq \frac{r_{\min}}{s_{\min}} \quad (2.12)$$

でなければならない。

まとめると、ゲイン  $g$  は

$$\frac{r_{\min}}{s_{\min}} \leq g \leq \frac{r_{\max}}{s_{\max}} \quad (2.13)$$

の範囲内になければならない。

## 2.3 抑制性感覚運動写像

ここまででは、モーター出力値がセンサー入力値に比例して大きくなる線形写像を考えた。

ゲインの値 ( $-g$ ) が負の値となる場合、センサー入力値が大きくなると、モーター出力値は小さくなる。モーター出力値が負になることが許される場合は、ここまで線形写像の表式をそのまま用いてもよいが、その値が負になることが出来ない場合には、定数  $c > 0$  を用いて

$$r = c - gs \quad (2.14)$$

とすることによってモーター出力値を正に保つことができる。この抑制性写像の場合にも、モーター出力値の取り得る範囲に応じて  $c$  と  $g$  の値の範囲を制限する必要があるのは、先に示した興奮性写像の場合と同様である。

## 2.4 非線形感覚運動写像

ここまででは、興奮性と抑制性の線形写像を用いる場合について述べた。非線形写像を用いることによって、より複雑な走性を創発する可能性がある。

たとえば、モーター出力値がセンサー値に反比例する以下のような場合、

$$r = \frac{g}{(s - s_{\min} + c)} \quad (2.15)$$

センサー値が大きくなればなるほどモーター出力が小さくなることは、抑制性線形写像と同じであるが、センサー値が小さい時にはモーター出力値が大きく変化し、センサー値が大きい場合にはモーター出力があまり変化しないという反応行動をつくることができる。

すなわち、この場合、

$$\frac{dr}{ds} = -\frac{g}{(s - s_{\min} + c)^2} \quad (2.16)$$

であるから、 $s_0 < s_1$  のとき

$$\left| \frac{dr}{ds}(s_0) \right| > \left| \frac{dr}{ds}(s_1) \right| \quad (2.17)$$

という性質をもつ感覚運動写像となる。

もうひとつの例として、興奮性および抑制性の2つの性質を同時にもつようないくつかの写像の例を見てみよう。たとえば、

$$\begin{aligned} r &= \frac{c}{\cosh\{g(s - s_0)\}} \\ &= \frac{2c}{e^{g(s-s_0)} + e^{-g(s-s_0)}} \end{aligned} \quad (2.18)$$

の場合、 $s < s_0$  の領域では、興奮性であり、 $s > s_0$  の領域では抑制性の感覚運動写像となる（図2.6参照）。

## 2.5 感覚運動写像で行動するロボット

LEGO ロボットを使うと、ここまでに挙げた感覚行動写像で実際に走行するロボットを簡単につくることができる。第7章を参照してほしい。

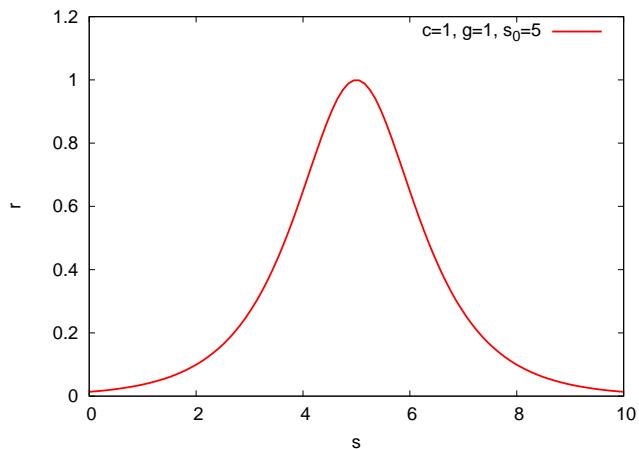


図 2.6: (2.18) 式による非線形感覚運動写像の例



## 第3章 収束する感覚行動写像

2014年ごろから、ドローンという言葉をニュースなどでも耳にするようになった。おもに、4つの回転翼をもつ飛行体（マルチコプター）のことをドローンと呼んでいるようである。単なるホビーとしてのラジコンから、空撮やインフラ調査撮影、また災害調査や農業への応用など爆発的な広がりを見せる勢いである。

このドローンは、単純な感覚運動写像で安定性を保っている飛行ロボットとみなすこともできる（図3.1参照）。



図3.1: 著者が開発した飛行ロボットの一例

ここでは、飛行ロボットの1軸回転運動をとりだし、単純な2階微分方程式でモデル化する[10]。感覚運動写像[1]に時間遅れやノイズが存在

しないと仮定した場合の、時間発展の振る舞いを解析する。

飛行ロボットの例として取り上げたが、この2階微分方程式は、摩擦のある振動や電気回路などをの挙動を表す方程式としてよく現れるものである。

解の求め方はいくつかある。変数分離や定数変化法を使って求める方法や、ラプラス変換して、それを部分分数にした後、逆変換して求める方法などが一般的かもしれない。ここでは、系の安定性を指数関数の指數部の値から直接議論しやすい行列形式を用いて解析する。

ここに示すように、系に時間遅れが存在しない場合、系は振動することはあっても、不安定になることは無いことを理論的に示す。ここでいう不安定とは振動の振幅が徐々に大きくなつて発散することを指す。

### 3.1 1軸回転運動

4つのローターを用いた飛行ロボットは、3次元空間を自由に並行移動し、回転の自由度も3つの軸に関してそれぞれ持っているので、合計6自由度の運動をする。

1軸(x-軸とする)の周りにおける回転運動のみを取り出して考えよう。図3.2に真横から飛行ロボットを見た図を示した。

また、図3.3に回転運動の概略図を示した。

水平状態からの回転角度 $\varphi$ の運動方程式は、慣性モーメント $I$ と回転軸の周りのトルク $I(L_1 - L_3)$ によって

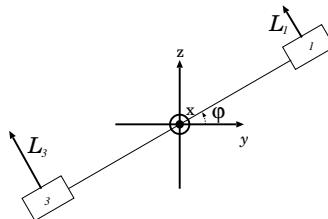
$$I\ddot{\varphi} = I(L_1 - L_3) \quad (3.1)$$

と表される。ここで、 $L_1, L_3$ は1と3のそれぞれの位置における揚力である。また $I$ は回転軸から1および3の位置までの距離を表す。

いま感覚運動写像における感覚に当たるのは、ジャイロセンサーからの角速度 $\dot{\varphi}$ と、加速度センサーからの回転角度 $\varphi$ の2つであるとする。そして、運動に当たる出力は揚力 $L_1, L_3$ である。



図 3.2: 真横からみた飛行ロボット

図 3.3:  $x$ -軸周りの飛行ロボットの回転角度  $\varphi$  とローター 1,3 による揚力  $L_1, L_3$ 

線形写像を用いて感覚運動写像を

$$L_1 = -g_G \dot{\varphi} - g_A \varphi \quad (3.2)$$

$$L_3 = g_G \dot{\varphi} + g_A \varphi \quad (3.3)$$

とする。 $g_G > 0$  はジャイロセンサー値に対するゲイン、 $g_A > 0$  は加速度センサー値に対するゲインである。

揚力 (3.2),(3.3) を回転運動を表す (3.1) に代入すると、

$$I\ddot{\varphi} = -2lg_G \dot{\varphi} - 2lg_A \varphi \quad (3.4)$$

であるので、飛行ロボットの1軸回転運動は、

$$\ddot{\varphi} = -c_G \dot{\varphi} - c_A \varphi \quad (3.5)$$

と2階微分方程式で書ける。ただし、係数  $c_G, c_A$  を

$$c_G \equiv \frac{2lg_G}{I} \quad (3.6)$$

$$c_A \equiv \frac{2lg_A}{I} \quad (3.7)$$

と定義した。 $I > 0, I > 0$  であるから、これらの係数もそれぞれ  $c_G > 0, c_A > 0$  である。

時間  $t$  を明示的に示して書き直せば、

$$\ddot{\varphi}(t) = -c_G \dot{\varphi}(t) - c_A \varphi(t) \quad (3.8)$$

となる。この微分方程式は典型的な2階微分方程式であり、 $\varphi(t)$  の解析的な関数を明示的に求めることが可能である。つまり、飛行ロボットの1軸周りの運動はノイズや筐体の歪による影響を除いて、厳密に数学的にあらかじめ予測することが可能なはずである。

実際には、時間遅れがあるため、飛行ロボットの運動は(3.8)式で表される微分方程式の解とは異なった運動をするが、ここではひとまず、時間遅れがない場合、つまり(3.8)式の性質を明らかにする。

## 3.2 数学的解析の概略地図

飛行ロボットの運動を数学的に解析するためには、行列の固有値やベクトルの一次独立性、また関数の級数展開など、いくつか数学的知識を必要とする。これらの知識は、線形代数や解析学で個別に学ぶ項目であるが、それらの知見が単なるバラバラな存在としてだけでなく、それぞれが関連して活躍し、飛行ロボットの運動という現実世界の問題を解析する上で結びつく。全体的な話の流れを図3.4に示した。

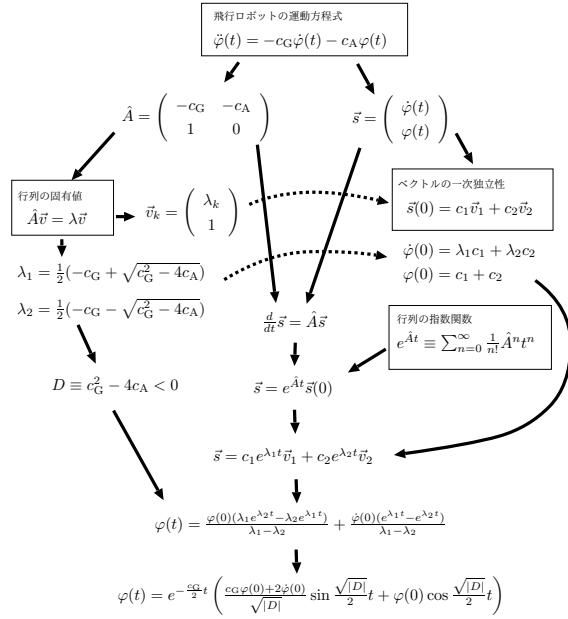


図 3.4: 飛行ロボットの運動方程式に対する数学的解析地図. 行列の固有値・固有ベクトル、ベクトルの1次独立性、および行列の指数関数などの数学的知識が組み合わせて活用される.

### 3.3 行列形式

2階微分方程式の解を求める方法はいくつかあるが、ここでは、行列形式を用いてその解析的な性質を調べることにしよう。行列の固有値がどのような値になるかを調べれば、運動が振動するのか、収束するのか、あるいは発散するのかということを、直接判定しやすくなる。

(3.8) 式から出発しよう。その左辺は、

$$\ddot{\varphi}(t) = \frac{d}{dt} \dot{\varphi}(t) \quad (3.9)$$

と書き改めることができる。また、すこしテクニカルな印象を受けるが、

次の自明な式

$$\frac{d}{dt}\varphi(t) = \dot{\varphi}(t) \quad (3.10)$$

を (3.8) 式と縦に並べて書くと,

$$\frac{d}{dt}\dot{\varphi}(t) = -c_G\dot{\varphi}(t) - c_A\varphi(t) \quad (3.11)$$

$$\frac{d}{dt}\varphi(t) = \dot{\varphi}(t) \quad (3.12)$$

となる. これらの微分方程式を行列形式で書くと

$$\frac{d}{dt} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} = \begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} \quad (3.13)$$

と書ける. さらに, 角速度  $\dot{\varphi}(t)$  と角度  $\varphi(t)$  は, 飛行ロボットの状態を表す量であるから, 状態ベクトル  $\vec{s}(t)$  を

$$\vec{s}(t) \equiv \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} \quad (3.14)$$

と定義し, 行列  $\hat{A}$  を

$$\hat{A} \equiv \begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \quad (3.15)$$

と定義すれば, (3.13) 式の行列形式は

$$\frac{d}{dt}\vec{s}(t) = \hat{A}\vec{s}(t) \quad (3.16)$$

と書きなおすことができる. つまり, (3.8) 式の 2 階微分方程式を行列  $\hat{A}$  を導入することによって, より簡潔な (3.16) 式の 1 階微分方程式の形に書き改めることができた.

### 3.4 行列の指數関数を用いた解

(3.16) 式は時間  $t$  に関する 1 階微分方程式のかたちをしている. その解  $\vec{s}(t)$  を具体的にもとめてみよう.  $\vec{s}(t)$  の  $n$  階微分を  $\vec{s}^{(n)}(t)$  と表す, つまり

$$\vec{s}^{(n)}(t) = \frac{d^n}{dt^n} \vec{s}(t) \quad (3.17)$$

と書き表すことにすると、テイラー級数を用いて、

$$\vec{s}(t) = \vec{s}(0) + \vec{s}'(0)t + \frac{\vec{s}''(0)}{2!}t^2 + \frac{\vec{s}'''(0)}{3!}t^3 + \dots \quad (3.18)$$

と書ける。"..."は、その先に無限に項がつづくことを表している。厳密には、この和が収束する条件を考慮する必要があるが、ここでは気にしないでこのように表せるとして、先に進む。 $\vec{s}(t)$ を1回微分するたびに、 $\hat{A}$ が掛けられるのであるから、 $n$ 回微分するということは、 $n$ 回 $\hat{A}$ を掛けることに等しい。

$$\begin{aligned} \vec{s}^{(n)}(t) &= \frac{d^n}{dt^n} \vec{s}(t) = \frac{d^{n-1}}{dt^{n-1}} \frac{d}{dt} \vec{s}(t) \\ &= \frac{d^{n-1}}{dt^{n-1}} \hat{A} \vec{s}(t) \\ &= \frac{d^{n-2}}{dt^{n-2}} \hat{A} \frac{d}{dt} \vec{s}(t) \\ &= \frac{d^{n-2}}{dt^{n-2}} \hat{A}^2 \vec{s}(t) \\ &\quad \dots \\ &= \hat{A}^n \vec{s}(t) \end{aligned} \quad (3.19)$$

ここで、行列 $\hat{A}$ は時間 $t$ に依存しないので、 $\hat{A}$ と、微分演算子 $\frac{d}{dt}$ が交換可能であることを使った。これを、テイラー展開の式に使うと、

$$\vec{s}(t) = \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!}t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!}t^3 + \dots \quad (3.20)$$

となることがわかる。つまり、(3.16)式が成り立つならば、(3.20)式が成り立つことがわかった。

逆に、(3.20)式が成り立つならば、(3.16)式が成り立つことを示そう。

(3.20) 式の両辺を  $t$  で微分すると,

$$\begin{aligned}\frac{d}{dt} \vec{s}(t) &= \frac{d}{dt} \left( \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!} t^3 + \dots \right) \\ &= \hat{A} \vec{s}(0) + \hat{A}^2 \vec{s}(0)t + \frac{\hat{A}^3 \vec{s}(0)}{2!} t^2 + \dots \\ &= \hat{A} \left( \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \dots \right) \\ &= \hat{A} \vec{s}(t)\end{aligned}\quad (3.21)$$

つまり, (3.16) 式

$$\frac{d}{dt} \vec{s}(t) = \hat{A} \vec{s}(t)$$

が成り立つ. その過程で, “...”の中に無限に項がたくさんあるという事実を使った. どんなにたくさん項があったとしても, その数が有限であるかぎり, このようなことは起こらない. 無限にあるからこそ, (3.16) 式が得られたのである.

ここでわかったことを, 改めて確認すると,

$$\frac{d}{dt} \vec{s}(t) = \hat{A} \vec{s}(t) \quad (3.22)$$

が成り立つならば,

$$\vec{s}(t) = \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!} t^3 + \dots \quad (3.23)$$

がその解として成り立つということである. また, その逆も成り立つことも確認した. つまり, 両者はお互いに必要十分条件になっている, すなわち等価であることがわかる.

上の,  $\vec{s}(t)$  の式をよく見ると,

$$\vec{s}(t) = \hat{f}(\hat{A}t) \vec{s}(0) \quad (3.24)$$

と書けることがわかる. ただし,

$$\hat{f}(\hat{A}t) \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!} t^2 + \frac{\hat{A}^3}{3!} t^3 + \dots \quad (3.25)$$

と定義した。ここで、 $\vec{v}(0)$  は状態ベクトルの初期値であるが、 $\hat{f}(\hat{A}t)$  という行列の右側に書かれていることに注意して欲しい。行列とベクトルの掛け算は、順序が入れ替わると意味が異なってくるのである。

このように、 $\vec{v}(t)$  に対する解は、形式的には簡単なかたちとして求めることができた。しかしそく見ると、こまつたことに、その行列の掛け算を無限回まで行い、それをすべて足し合わせなければならないことがわかる。いくらコンピューターの演算処理が高速化したといえども、無限回掛け算や足し算を繰り返すことはできない。

つぎに示す行列の固有値を用いると、この問題は解決される。

## 3.5 行列の固有値と固有ベクトル

いま、行列  $\hat{A}$  に対して、

$$\hat{A}\vec{v} = \lambda\vec{v} \quad (3.26)$$

を満たすベクトル  $\vec{v}$  とスカラー  $\lambda$  のことを、それぞれ  $\hat{A}$  の右固有ベクトル、および固有値という。これは、 $2 \times 2$  の次元をもつ行列  $\hat{A}$  についてのみ言えることではなく、一般の正方行列に対しても成り立つ。行列  $\hat{A}$  が対称行列であれば、右固有ベクトルと左固有ベクトルの区別はしなくても、両者は同じものになるが、(3.15) 式の行列の様に、非対称行列に関しては、右固有ベクトルと左固有ベクトルが異なるので、注意が必要である。ここではこのことに関して深入りはしない。ただし、後で具体的に求めるが、固有値、固有ベクトルは、行列の次数だけ存在するということは覚えておく必要がある。つまり、 $2 \times 2$  の行列の場合、固有値が 2 つ、そしてそれぞれの固有値に対応して、固有ベクトルも 2 つ存在する。

右固有ベクトルと固有値を求める方法については、後回しにして、ここでは一旦それらが得られたとして、話を進めることにする。(3.25) 式

$$\hat{f}(\hat{A}t) \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!}t^2 + \frac{\hat{A}^3}{3!}t^3 + \dots \quad (3.27)$$

の両辺に、右から  $\hat{A}$  の固有ベクトル  $\vec{v}$  を掛けると、

$$\begin{aligned}
 \hat{f}(\hat{A}t)\vec{v} &= \sum_{n=0}^{\infty} \frac{1}{n!} \hat{A}^n t^n \vec{v} \\
 &= \sum_{n=0}^{\infty} \frac{1}{n!} \hat{A}^n \vec{v} t^n \\
 &= \sum_{n=0}^{\infty} \frac{1}{n!} \lambda^n \vec{v} t^n \\
 &= \left( \sum_{n=0}^{\infty} \frac{1}{n!} \lambda^n t^n \right) \vec{v} \\
 &= e^{\lambda t} \vec{v}
 \end{aligned} \tag{3.28}$$

となる。固有値と固有ベクトルを使うと、行列をスカラーに置き換えられるので、行列の無限回掛け算という困難を指数関数  $e^{\lambda t}$  という形で、すっきり回避できるのである。

この式の意味するところは、行列  $\hat{f}(\hat{A}t)$  を  $\hat{A}$  の固有ベクトル  $\vec{v}$  に左から掛けることは、スカラー  $e^{\lambda t}$  を掛けることに等しいということである。そこで、

$$\hat{f}(\hat{A}t) = e^{\hat{A}t} \tag{3.29}$$

と書き表すことにすれば、

$$e^{\hat{A}t} \vec{v} = e^{\lambda t} \vec{v} \tag{3.30}$$

という、自然な形で表すことが出来る。つまり、行列  $\hat{A}$  の指数関数が、次のようにその無限級数で定義された。

$$e^{\hat{A}t} \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!} t^2 + \frac{\hat{A}^3}{3!} t^3 + \dots \tag{3.31}$$

さて、行列の固有値と右固有ベクトルはそれぞれ2つずつあるので、それらを  $\lambda_k, \vec{v}_k$  ( $k = 1, 2$ ) と表す。 $(3.28)$  式から、

$$e^{\hat{A}t} \vec{v}_k = \vec{v}_k e^{\lambda_k t} \quad (k = 1, 2) \tag{3.32}$$

である。また、右固有ベクトル  $\vec{v}_1, \vec{v}_2$  はそれぞれ、1次独立であり、任意のベクトルをその線形結合で表すことができる。

## 3.6 状態ベクトルに対する解

準備が整ったので、状態ベクトル  $\vec{s}(t)$  の話に戻ろう。 (3.24) 式の  $\vec{s}(0)$  を右固有ベクトルの線形結合で表す。

$$\vec{s}(0) = c_1 \vec{v}_1 + c_2 \vec{v}_2 \quad (3.33)$$

これを (3.24) 式に代入すると、

$$\begin{aligned} \vec{s}(t) &= e^{\hat{A}t} (c_1 \vec{v}_1 + c_2 \vec{v}_2) \\ &= c_1 e^{\lambda_{1t}} \vec{v}_1 + c_2 e^{\lambda_{2t}} \vec{v}_2 \end{aligned} \quad (3.34)$$

となり、状態ベクトルは、行列の固有値によってその時間変化が記述されることがわかる。言い換えると、行列の固有値から、飛行ロボットの運動を記述（予測）できるということである。

ここで、行列  $\hat{A}$  の固有値と右固有ベクトルを求めておこう。行列の特性方程式<sup>1</sup>を用いれば、即座に固有値に対する方程式を求めることができるが、ここでは素直に固有値、固有ベクトルの定義から、それらを求めてみる。 (3.26) 式に具体的に行列 (3.15) を用いると、

$$\begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (3.35)$$

と具体的に書くことができる。ここで、右固有ベクトル  $\vec{v}$  を

$$\vec{v} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (3.36)$$

と、 $\alpha, \beta$  を用いて表した。これらの値も、もちろん既知ではなく、固有値  $\lambda$  を求める過程において同時に求められるべきものである。 $2 \times 2$  行列とベクトルの積を展開してみると、

$$-c_G\alpha - c_A\beta = \lambda\alpha \quad (3.37)$$

$$\alpha = \lambda\beta \quad (3.38)$$

---

<sup>1</sup> $\det(\hat{A} - \lambda\hat{I}) = 0$  を行列  $\hat{A}$  に対する特性方程式よ呼び、これを満たす  $\lambda$  を求めれば、それが固有値となる。ただし、 $\hat{I}$  は単位行列。 $\det$  は行列式をとることを意味する。

となる。 (3.38) 式を (3.37) 式に代入すると,

$$-c_G \lambda \beta - c_A \beta = \lambda^2 \beta \quad (3.39)$$

である。任意の  $\beta$  に対してこれが成り立つためには

$$\lambda^2 + c_G \lambda + c_A = 0 \quad (3.40)$$

である必要があり、2次方程式の解の公式を用いて固有値  $\lambda$  は

$$\lambda_1 = \frac{1}{2} \left( -c_G + \sqrt{c_G^2 - 4c_A} \right) \quad (3.41)$$

$$\lambda_2 = \frac{1}{2} \left( -c_G - \sqrt{c_G^2 - 4c_A} \right) \quad (3.42)$$

と求められる。規格化条件  $|v| = 1$  がない場合には、 $\alpha, \beta$  の値は一意には定まらないが、その比は (3.38) 式によって与えられる。

### 3.7 状態ベクトルの具体的なたたち

ここまで議論では、(3.41),(3.42) 式であたえられる固有値  $\lambda_k$  が複素数になる可能性がある。同様に (3.38) 式から、右固有ベクトルも複素ベクトルになる可能性がある。したがって、(3.34) 式で与えられる状態ベクトルも複素ベクトルになる可能性がある。状態ベクトルの要素は、もともと飛行ロボットの姿勢角度と角速度であるから、実数でなければならぬ。そもそも初期状態  $s(0)$  が実数ベクトルであることからしても、奇妙に思える。そこで、初期状態を用いて状態ベクトルを具体的に表してみよう。初期状態  $t = 0$  においては、

$$\begin{pmatrix} \dot{\varphi}(0) \\ \varphi(0) \end{pmatrix} = c_1 \begin{pmatrix} \lambda_1 \\ 1 \end{pmatrix} + c_2 \begin{pmatrix} \lambda_2 \\ 1 \end{pmatrix} \quad (3.43)$$

であるので、この連立方程式から  $\lambda_1 \neq \lambda_2$  の場合の  $c_1, c_2$  を求めると、

$$c_1 = \frac{\lambda_2 \varphi(0) - \dot{\varphi}(0)}{\lambda_2 - \lambda_1} \quad (3.44)$$

$$c_2 = -\frac{\lambda_1 \varphi(0) - \dot{\varphi}(0)}{\lambda_2 - \lambda_1} \quad (3.45)$$

となる。したがって、これらを(3.34)式

$$\vec{s}(t) = c_1 e^{\lambda_1 t} \vec{v}_1 + c_2 e^{\lambda_2 t} \vec{v}_2$$

に用いると、

$$\varphi(t) = \varphi(0) \left\{ \frac{\lambda_2 e^{\lambda_1 t} + \lambda_1 e^{\lambda_2 t}}{\lambda_1 - \lambda_2} \right\} + \dot{\varphi}(0) \left\{ \frac{e^{\lambda_1 t} - e^{\lambda_2 t}}{\lambda_1 - \lambda_2} \right\} \quad (3.46)$$

と得られる。無論、角速度  $\dot{\varphi}(t)$  も同様にして求めることが可能であるし、この式を時間微分して求めることもできる。

(3.46)式には、行列  $\hat{A}$  の固有値  $\lambda_1, \lambda_2$  が、まだそのまま含まれている。その値がどのような値をとっても、一般的に成り立つ式であると言える。しかし、その値が複素数である場合には、実関数であるべき  $\varphi(t)$  が本当に実数値を取る関数であるかどうか分かりにくい。

そこで、固有値を与える(3.41),(3.42)式

$$\begin{aligned} \lambda_1 &= \frac{1}{2} \left( -c_G + \sqrt{c_G^2 - 4c_A} \right) \\ \lambda_2 &= \frac{1}{2} \left( -c_G - \sqrt{c_G^2 - 4c_A} \right) \end{aligned}$$

を具体的に用いてみよう。固有値の式からわかる通り  $c_G^2 - 4c_A$  の値が正の時には、固有値の虚数部はゼロである。したがって、当然  $\varphi(t)$  の値も実数値となる。いっぽうその値が負の時には、固有値は複素数になる。そこで、

$$D \equiv c_G^2 - 4c_A \quad (3.47)$$

と定義して、 $\varphi(t)$  の式を具体的に表してみると、

$$\begin{aligned} \varphi(t) &= \frac{\varphi(0)}{2} e^{-\frac{c_G}{2}t} \left\{ \frac{c_G}{\sqrt{D}} \left( e^{\frac{\sqrt{D}}{2}t} - e^{-\frac{\sqrt{D}}{2}t} \right) + \left( e^{\frac{\sqrt{D}}{2}t} + e^{-\frac{\sqrt{D}}{2}t} \right) \right\} \\ &\quad + \frac{\dot{\varphi}(0)}{\sqrt{D}} e^{-\frac{c_G}{2}t} \left( e^{\frac{\sqrt{D}}{2}t} - e^{-\frac{\sqrt{D}}{2}t} \right) \end{aligned} \quad (3.48)$$

となる。 $D < 0$  のときに虚数が現れる。

$$\sqrt{D} = i \sqrt{|D|} \quad (3.49)$$

ここで、 $i$  は虚数単位である。これを用いると、

$$\begin{aligned}\varphi(t) &= \frac{\varphi(0)}{2} e^{-\frac{c_G}{2}t} \left\{ \frac{c_G}{i\sqrt{|D|}} \left( e^{\frac{i\sqrt{|D|}}{2}t} - e^{-\frac{i\sqrt{|D|}}{2}t} \right) + \left( e^{\frac{i\sqrt{|D|}}{2}t} + e^{-\frac{i\sqrt{|D|}}{2}t} \right) \right\} \\ &\quad + \frac{\dot{\varphi}(0)}{i\sqrt{|D|}} e^{-\frac{c_G}{2}t} \left( e^{i\frac{\sqrt{|D|}}{2}t} - e^{-i\frac{\sqrt{|D|}}{2}t} \right) \\ &= \varphi(0)e^{-\frac{c_G}{2}t} \left\{ \frac{c_G}{\sqrt{|D|}} \sin \frac{\sqrt{|D|}}{2}t + \cos \frac{\sqrt{|D|}}{2}t \right\} \\ &\quad + \frac{2\dot{\varphi}(0)}{\sqrt{|D|}} e^{-\frac{c_G}{2}t} \sin \frac{\sqrt{|D|}}{2}t\end{aligned}\tag{3.50}$$

となり、すべて実数と実数の三角関数を用いて表されるので、確かに  $\varphi(t)$  は実数関数であることが確認できる。

この表式の物語っていることはどのようなことであろうか？大きき分けると、初期角度  $\varphi(0)$  に関する項と、初期角速度  $\dot{\varphi}(0)$  に関する項の 2 つの項から全体が構成されていることがわかる。

もうすこし  $\varphi(t)$  の式の意味するところを見やすくするためにここで、減衰係数  $\gamma$  を

$$\gamma \equiv \frac{c_G}{2}\tag{3.51}$$

と定義し、角振動数  $\omega$  を

$$\omega \equiv \frac{\sqrt{|D|}}{2}\tag{3.52}$$

と、それぞれ定義すると、

$$\varphi(t) = \varphi(0)e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t\tag{3.53}$$

といふかたちに整理できる。ただし、ここまで話しあくまでも  $D < 0$  の場合にたいするものである。

$c_G, c_A$  は、運動方程式においては、その意味がわかり易かった。しかし、この時間発展の式のなかでそれを直接使うと、煩雑なかたちとなる。むしろ、 $\gamma, \omega$  を用いて表した方が、よりその振る舞いがイメージしやすくなる。

### 3.8 無矛盾性の確認

ここまで、 $D = c_G^2 - 4c_A < 0$  の条件が満たされる場合について  $\varphi(t)$  の具体的な表式をもとめた。どうやって、最終結果  $\varphi(t)$  に間違いがないことを確認すればよいだろう？

(3.53) 式が、実際に (3.8) 式

$$\ddot{\varphi}(t) = -c_G \dot{\varphi}(t) - c_A \varphi(t)$$

を満たすことを確認する。 $\varphi(t)$  を時間微分して、 $\varphi(0), \dot{\varphi}(0)$  および三角関数の種類別にまとめると、

$$\dot{\varphi}(t) = -\left(\frac{\gamma^2}{\omega} + \omega\right)\varphi(0)e^{-\gamma t} \sin \omega t \quad (3.54)$$

$$-\frac{\gamma}{\omega}\dot{\varphi}(0)e^{-\gamma t} \sin \omega t \quad (3.55)$$

$$+\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \quad (3.56)$$

となる。さらに時間微分して、 $\ddot{\varphi}(t)$  を求めると、

$$\begin{aligned} \ddot{\varphi}(t) &= \frac{\gamma}{\omega}(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \sin \omega t \\ &\quad -(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \cos \omega t \\ &\quad +\left(\frac{\gamma^2}{\omega} - \omega\right)\dot{\varphi}(0)e^{-\gamma t} \sin \omega t \\ &\quad -2\gamma\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \end{aligned} \quad (3.57)$$

と求められる。これが、(3.8) 式の左辺である。ここで、

$$\gamma = \frac{c_G}{2}, \quad \omega = \frac{\sqrt{|D|}}{2}, \quad D = c_G^2 - 4c_A < 0$$

を使うと、

$$\gamma^2 + \omega^2 = c_A \quad (3.58)$$

という関係が成り立つ。

さらに、これらの関係を使うと、(3.8)式の右辺は、

$$-c_G\dot{\varphi}(t) - c_A\varphi(t) = -2\gamma\dot{\varphi}(t) - (\gamma^2 + \omega^2)\varphi(t) \quad (3.59)$$

と表される。ここに  $\dot{\varphi}(t)$  および  $\varphi(t)$  の各項を代入すればよい。すこし煩雑になるので、右辺について、各項べつにまとめると、

$$1. \varphi(0)e^{-\gamma t} \sin \omega t \text{ の係数 : } \frac{\gamma}{\omega}(\gamma^2 + \omega^2)$$

$$2. \varphi(0)e^{-\gamma t} \cos \omega t \text{ の係数 : } -(\gamma^2 + \omega^2)$$

$$3. \dot{\varphi}(0)e^{-\gamma t} \sin \omega t \text{ の係数 : } \frac{\gamma^2}{\omega} - \omega$$

$$4. \dot{\varphi}(0)e^{-\gamma t} \cos \omega t \text{ の係数 : } -2\gamma$$

となり、たしかに  $\ddot{\varphi}(t)$  の各項の係数

$$\begin{aligned} \ddot{\varphi}(t) &= \frac{\gamma}{\omega}(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \sin \omega t \\ &\quad -(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \cos \omega t \\ &\quad + \left( \frac{\gamma^2}{\omega} - \omega \right) \dot{\varphi}(0)e^{-\gamma t} \sin \omega t \\ &\quad - 2\gamma\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \end{aligned}$$

と一致する。

$\varphi(t)$  の具体的表式は、その出発点にした2階微分方程式をたしかに満たしていることが確認できた。行列形式や固有ベクトルなどの道具をつかって、たどり着いた目的地がもとの出発点と矛盾していないことが分かった。

数学的な論理展開に象徴されるような、「論理」というものは、意外と脆い側面がある。それぞれの論理には、それが成り立つ条件というものがある。そのような条件付き論理を、いくつも重ねて用いてたどり着く結論というものには、当然それぞれの条件が何重にも課されているので、ただでさえ危うい。さらに加えて、無矛盾性が確認されていない論理展開というものは、そうやすやすと信じてよいとは言えないものである。

### 3.9 振動

はなしを、 $\varphi(t)$  すなわち、飛行ロボットの傾き角度の具体的な振るまいにもどそう。 $D = c_G^2 - 4c_A < 0$  の場合には、 $\varphi(t)$  は(3.53)式

$$\varphi(t) = \varphi(0)e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t \quad (3.60)$$

と表される。

$c_G, c_A$  は、われわれが選べるパラメーターであった。いま、 $c_G = 0$  と選ぶと、 $\gamma = c_G/2 = 0$  であるから、

$$\varphi(t) = \varphi(0) \cos \omega t + \frac{\dot{\varphi}(0)}{\omega} \sin \omega t \quad (3.61)$$

である。また、これを時間微分すると

$$\dot{\varphi}(t) = -\varphi(0)\omega \sin \omega t + \dot{\varphi}(0) \cos \omega t \quad (3.62)$$

が得られる。

これは、式からも分かるとおり、振幅が初期状態  $\varphi(0), \dot{\varphi}(0)$  に依存する時間的な振動運動を表す。角振動数  $\omega$  は、いま  $c_G = 0$  であるから、 $\omega = \sqrt{|D|}/2 = \sqrt{c_A}$  と、 $c_A$  の値だけで決まる。ただし、 $D < 0$  でなければならぬので、 $c_A > 0$  の条件を満たさなければならない。

$\omega = \pi, 3/2\pi, 1/2\pi$  の場合の、 $\varphi(t), \dot{\varphi}(t)$  の時間変化を図 3.5 に表した。 $\omega$  の値が大きいほど振動の周期が短い。 $\omega$  の値は、 $c_A$  の値だけできるのであるから、 $c_A$  の値、すなわち、PI 制御の比例ゲインはロボット運動の振動周期を左右するパラメーターであることがわかる。

$\varphi(t)$  と  $\dot{\varphi}(t)$  はおたがいに位相がすこしづれた、似たような振動を続けることがわかった。

ここで、つぎの量を計算してみると、

$$\{\omega\varphi(t)\}^2 + \dot{\varphi}(t)^2 = \omega^2\varphi(0)^2 + \dot{\varphi}(0)^2 \quad (3.63)$$

と、右辺は時間  $t$  に依存しない一定値となることがわかる。つまり、 $\omega\varphi(t)$  を横軸、 $\dot{\varphi}(t)$  を縦軸としてグラフを描くと、半径が  $\sqrt{\omega^2\varphi(0)^2 + \dot{\varphi}(0)^2}$  の円となることを意味している。

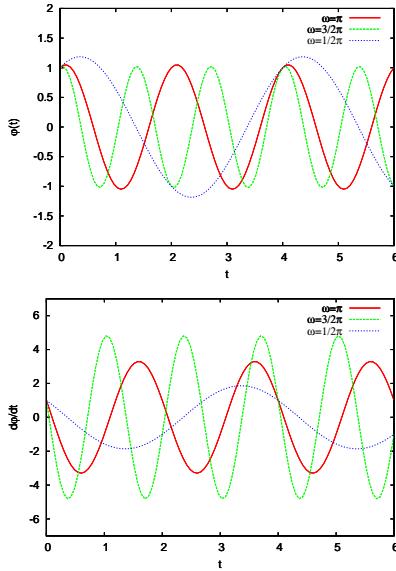


図 3.5:  $c_G = 0$  の場合,  $\varphi(t), \dot{\varphi}(t)$  の時間変化は振動をつづける.

実際に, それらを図 3.6 に描画した. 確かに, 円が描かれた.

この空間のことを, 相空間と呼び, 運動状態はこの空間内の 1 点として表される. また, 運動の時間発展は, 相空間内の軌道として描かることになる.  $c_G = 0$  の場合の振動運動は, 円軌道として表される.

### 3.10 減衰振動

具体的な時間発展の式

$$\varphi(t) = \varphi(0)e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t$$

が意味するところを理解するために, 各項についてそれぞれくわしく見てみる.

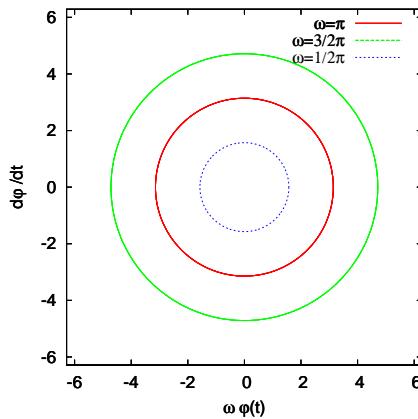


図 3.6:  $\omega\varphi(t), \dot{\varphi}(t)$  をそれぞれ横軸と縦軸に表した空間を相空間と呼ぶ。運動は、相空間の軌道として表される。振動運動は、相空間における円軌道となる。

第1項目は初期角度  $\varphi(0)$  がゼロでない時に現れる項である。その全体に  $e^{-\gamma t}$  が掛かっているので、 $\gamma > 0$  すなわち  $c_G > 0$  の場合には、初期角度の効果は時間が経てば減衰していくことを意味している。そして、減衰係数が大きければ大きいほど、速く減衰する（図 3.7(a) 参照）。次に、こ

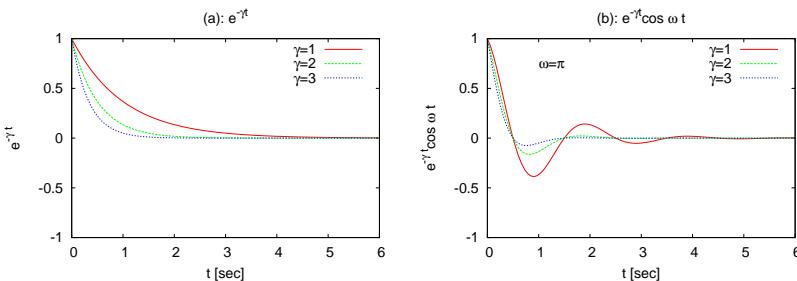


図 3.7: (a) 指数関数的な減衰の例。減衰係数  $\gamma$  が大きくなるほど、速く減衰する。  
(b) 振幅が指数関数的に減衰する  $\cos$  関数の例。

の初期角度の減衰項の中の、 $\cos$  関数で表された項がどのように時間変化

するのかを図3.7(b)に示した。例として角振動数  $\omega = \pi$  の場合を示した。振動しながら、その振幅が減衰していくことがわかる。このような関数の振る舞いの場合に、 $e^{-\gamma t}$  のことを包絡関数、あるいは包絡線と呼ぶ。

$\varphi(t)$  の時間発展を表す式の、第1項にはもうひとつの項が含まれていることを忘れてはならない。この項は

$$e^{-\gamma t} \frac{\gamma}{\omega} \sin \omega t \quad (3.64)$$

というかたちをしている。この項自身の減衰の様子を図3.8(a)に示した。この項が  $\cos$  関数の減衰に加えられたものが実際の振動減衰の時間発展

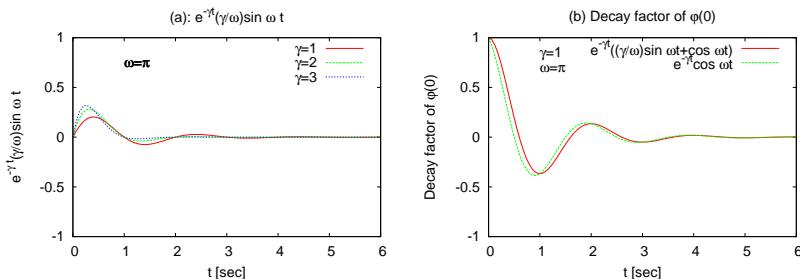


図3.8: (a)  $e^{-\gamma t}(\gamma/\omega)\sin \omega t$  の減衰の様子。 (b)  $e^{-\gamma t}((\gamma/\omega)\sin \omega t + \cos \omega t)$  の減衰の様子。

(図3.8(b)参照)である。 $\cos$  関数だけの場合と比べて、やや位相が遅れ減衰を示すが、結局指数関数的に減衰する包絡線によってその振動は抑えられて、急速にゼロに収束する様子がわかる。

さらに初期角速度の項があるが、この項は初期角度の  $\sin$  関数と類似のかたちをしており、その時間発展の様子も、まったく同様の振る舞いをすることが容易に理解できる。

### 3.11 比例ゲインと積分ゲイン空間における相図

次に、固有値を与える(3.41))式と、 $c_G, c_A$  の値を用いて、 $\lambda_1$  が実際にどのような値をとるかを図示してみよう。それらの実数部の正負によっ

て、状態ベクトルの発散、収束が決まる。図 3.9 に  $\lambda_1$  の実部と虚部の値

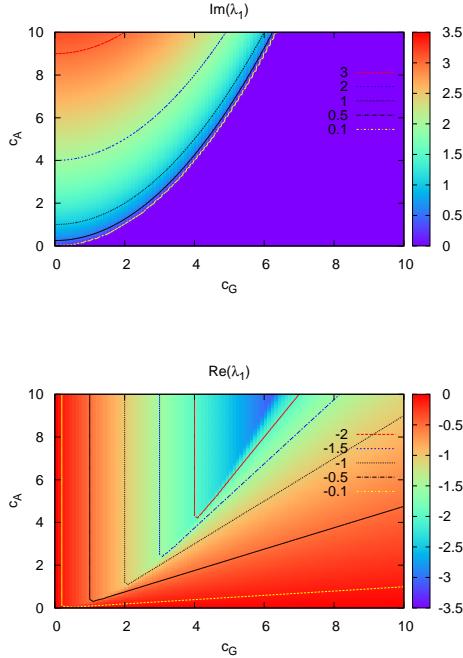


図 3.9:  $c_G, c_A$  の関数としての  $\text{Im}(\lambda_1)$  と  $\text{Re}(\lambda_1)$

を  $c_G, c_A$  の関数として示した。まず、虚部を見ると、 $c_A < c_G^2/4$  の領域では、 $\text{Im}(\lambda_1)=0$  なので、系は振動しない。一方  $c_A > c_G^2/4$  の領域では、 $\text{Im}(\lambda_1) > 0$  となり、系は振動する。 $\text{Im}(\lambda_1)$  の大きさがその振動数に対応するが、 $c_A = c_G^2/4$  から離れれば離れるほど、振動数が大きくなることがわかる。

つぎに、実数部を見るとすべての領域で負の値になる。包絡曲線は図に示したすべての領域でゼロに収束することがわかる。境界  $c_A = c_G^2/4$  の近傍において、比較的  $|\text{Re}(\lambda)|$  のあたいが大きくなる、すなわち速く収

束する傾向にあることがわかる。

### 3.12 遷移行列を用いた漸化式

このように、系の振る舞いは行列の固有値を求める問題に帰着され、その固有値がもともとの系のゲインの値  $c_G, c_A$  によって決まることがわかる。すでに明示的に系の振る舞いが表現されているが、ここでは漸化式の形式を用いても以上の議論を定式化可能であることを示す。

微分方程式 (3.16) から、微小時間  $\Delta t$  に対して

$$\Delta \vec{s}(t) = \hat{A} \vec{s}(t) \Delta t \quad (3.65)$$

が成り立つ。これを用いると  $\Delta t$  後、すなわち  $t + \Delta t$  における状態ベクトル  $\vec{s}(t + \Delta t)$  は

$$\begin{aligned} \vec{s}(t + \Delta t) &= \vec{s}(t) + \Delta \vec{s}(t) \\ &= \vec{s}(t) + \hat{A} \vec{s}(t) \Delta t \\ &= (\hat{I} + \Delta t \hat{A}) \vec{s}(t) \end{aligned} \quad (3.66)$$

となる。ここで、 $\hat{I}$  は単位行列

$$\hat{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (3.67)$$

である。また、スカラー量  $\Delta t$  と行列やベクトルとの積は、順序に依存しない性質を用いた。 $\vec{s}_{i+1} = \vec{s}(t + \Delta t)$ ,  $\vec{s}_i = \vec{s}(t)$  と置き換えて書くと (3.66) 式は

$$\vec{s}_{i+1} = \hat{T} \vec{s}_i \quad (3.68)$$

と漸化式のかたちに書くことが出来る。ただし、ここで遷移行列  $\hat{T}$  を

$$\hat{T} \equiv \hat{I} + \Delta t \hat{A} \quad (3.69)$$

と定義した。行列  $\hat{A}$  の右固有ベクトル  $\vec{v}$  を用いると

$$\begin{aligned}\hat{T}\vec{v} &= (\hat{I} + \Delta t \hat{A})\vec{v} \\ &= (1 + \Delta t \lambda)\vec{v}\end{aligned}\quad (3.70)$$

であるから、遷移行列  $\hat{T}$  の固有値  $\eta$  は

$$\eta = 1 + \Delta t \lambda \quad (3.71)$$

である。したがって、元の  $\hat{A}$  の固有値  $\lambda$  は、遷移行列の固有値  $\eta$  を用いて

$$\lambda = \frac{\eta - 1}{\Delta t} \quad (3.72)$$

と求められる。

この関係を用いれば、漸化式 (3.68) から、系の振る舞いを決める  $\hat{A}$  行列の固有値  $\lambda$  を求めることができる。



# 第4章 漸化式と数値解析

数学は、様々な量を文字記号として象徴的に表し、数式としてそれらの間の関係性を客観的に表現する方法である。人類が2千年前の年月のあいだ使ってきたものである。しかし、万能ではない。数学の力で（解析的に）表現したり答えを求めたりできない場合も多々ある。そのような場合には、コンピュータ上で数値解析を行うのが、一つの強力な解決法である。

つまり、数値解析とは、端的に言えば、解析的に解くことが困難な問題を、コンピュータを用いて数値的に解決する方法である。

ここでは、はじめに数値解析を含めてプログラミングにおいて重要な概念である構造化プログラミングを基に、正しく機能するプログラムをつくるために注意すべき点について述べる。ここで「正しく」と言うのは、「求められている機能を常に果たすことができる」という意味である。具体的な例としてニュートン・ラフソン法について述べる。

また、数値解析を行う上で重要な、打ち切り誤差や丸め誤差などの計算誤差についても述べる。

## 4.1 正しく機能するプログラムを書く

データの流れを明瞭にする

アルゴリズムの具体的なコンピュータ上の実態をプログラムと言う。また、そのプログラムを作成することをプログラミングという。

プログラミングにおいて必須なことは正しく機能するプログラムをつくるということである。より高速に動作することや、技巧的なアルゴリ

ズムを用いて効率を上げることも時には必要であるが、正しさを損なう危険性がある場合には、むしろ単純さを追求することが重要である。

このことは、プログラミング言語の種類に関係なく必要なことである。オブジェクト指向言語であろうが、そうでなかろうが、あるいは、手続き型であろうがイベントドリブン型であろうがなかろうが、Java言語であろうがC言語であろうが、正しく機能するプログラムを求めるることは、あらゆるプログラミングの場面において普遍的に必要なことである。

正しいプログラムをつくるためには、構造化プログラミングの考え方へ則ってそれを行う必要がある。

## 4.2 構造化プログラミング

正しく動作するプログラムとは、言い換えると、常に必要とする機能を果たして結果を返してくれるということである。以下では、一例として実数の平方根を求めるプログラムを考えるが、データによってはエラーを起こしたり、間違った答えを返すプログラムは、正しいとはいえない。

### データの流れを明瞭にする

アルゴリズムとは、言い換えれば、データの流れのことである。プログラムを正しく動作するようにつくるためには、データの流れがわかりやすく記述されている必要がある。データの流れがわかりやすく明瞭であるために、技巧的になりすぎず、単純化されたプログラムをつくる必要がある。つまり、うまいプログラムは必要ない。わかりやすく書こう。

### 基本アルゴリズムを組み合わせる

わかりやすく単純化されたプログラムをつくるといつても、プログラムの果たすべき機能が複雑な場合にはどうすればいいのだろうか？

複雑なアルゴリズムを、より単純な機能の組み合わせとして記述するのである。もっとも基本的なアルゴリズムとして以下の3つの機能がある。

1. 命令（演算や関数）
2. 繰り返し（While, for, …）
3. 条件分岐（if, case, switch, …）

以上の3つの基本アルゴリズムを組み合わせることによって、あらゆる複雑なアルゴリズムを構成することが可能であるというのが、構造化プログラミング[3, 4]の考え方である。

### goto 文をつかわない

正しいプログラムをつくるためには、データの流れを明瞭にする必要がある。goto文をつかうと、プログラムの任意箇所から、任意箇所へ処理を飛ばすことができるが、これを使うと、データの流れが不明瞭になる場合が多い。

### 関数をつかう

単純化のもうひとつ重要な要素として、mainルーチン（main関数）を長く複雑にしがちになることが必要である。main関数では、全体のデータの流れが明瞭にわかる形になっていることが理想的である。

そのためには、関数あるいはサブルーチンを使う。プログラミングにおいて、それらの役割は、一連のひとまとまりの処理を定義して、その処理を行いたい箇所で、その関数（あるいはサブルーチン）を呼び出すだけで済むようにすることである。

関数とサブルーチンの違いは、引数を渡して値を返すかたちに定義するか、あるいは単なる処理の集まりとして定義するかの違いだけである。

関数とは、一連の処理が、ひとつの関数名として定義されたものであるから、関数の役割は、抽象化を実現することと言う事も出来る。つまり

り、関数名が抽象化されたシンボルであり、その具体的な内容がその関数の定義部分の一連の処理ということである。

### 4.3 ニュートン・ラフソン法

漸化式を繰り返し適用することによって、関数のゼロ切片を求める方法の一例として、ニュートン・ラフソン法を説明する。単にニュートン法と呼ばれる場合もある。

変数  $x$  によって微分可能な関数  $f(x), g(x)$  が

$$y = f(x) = g(x) - C \quad (4.1)$$

という関係にあるとき、 $f(x) = 0$  を満たす  $x$  を求めれば、 $g(x) = C$  を満たす  $x$  の値を求めることができる。ここで、 $C$  は定数である。図 4.1 に示

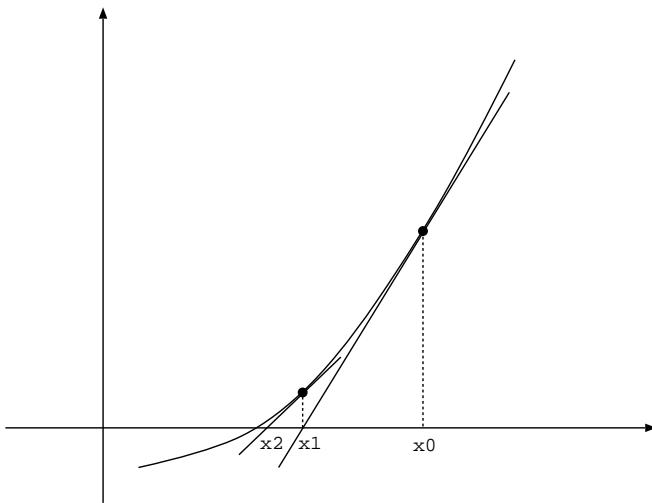


図 4.1: ニュートン・ラフソン法の原理

したように、 $x_0$  における  $f(x)$  の接線が  $x$ -軸と交わる点を  $x_1$  とすると、

微分係数の意味から、

$$\frac{df}{dx}(x_0) = \frac{f(x_0)}{x_0 - x_1} \quad (4.2)$$

である。したがって、 $x_1$  は

$$x_1 = x_0 - \frac{f(x_0)}{\frac{df}{dx}(x_0)} \quad (4.3)$$

と求められる。同様に、 $x_n$  の値から  $x_{n+1}$  の値が

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{df}{dx}(x_n)} \quad (4.4)$$

という漸化式を用いて求めることができる<sup>1</sup>。これを繰り返えすと  $x_n$  は  $g(x) = C$  を満たす  $x$  の値に近づく。

この様に、同様の計算手続きを繰り返すことによって、目的の値に近づいていくアルゴリズムのことを反復法という。

## $\sqrt{2}$ を求める具体例

具体的に、 $g(x) = x^2$  また、 $C = 2$  とすると  $x^2 = 2$  を満たす  $x$  の値、すなわち  $\sqrt{2}$  の値を求ることになる。図 4.2 にそのプログラム例を示した。この例では、基本アルゴリズム要素のなかの、命令と繰り返しを用いている。その意味では、構造化プログラムと言える。

## インデントと数学的表式に近い変数名

しかし、次に示すように、適切なインデント（字下げ）と、数学的表式と混乱を招かない変数名を用いた方が、よりデータの流れが明確になる。また、図 4.3 の例では、 $C = 2$  以外の場合にも  $\sqrt{C}$  の値を求めることができるように、一般化されている。繰り返しの終了判定を while(条

---

<sup>1</sup> 数列などの、項と項の間に一定の関係があり、初項からはじまって、つぎつぎと項を求めることが出来る式を漸化式という。たとえば、等比数列の場合、一つ前の項に等比を掛けたものが次の項となる。

```

1 #include <stdio.h>
2 #include <math.h>
3 int main() {
4     double a,b,c,x;
5     x=2.0;
6     a=x;
7     b=(x/a+a)/2.0;
8     c=b-a;
9     while(fabs(c)>1.0e-6) {
10        c=b-a;
11        printf("%12.10f \n",b);
12        a=b;
13        b=(x/a+a)/2.0;
14    }
15 }
```

図 4.2: ニュートン・ラフソン法で  $\sqrt{2}$  の値を求める。適切にインデントされておらず、変数名も混乱を招く。

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main() {
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    x=C;
16    dx=-(x*x-C)/(2.0*x);
17    while(fabs(dx)>eps) {
18        x=x+dx;
19        printf("%12.10f \n",x);
20        dx=-(x*x-C)/(2.0*x);
21    }
22 }
23 }
```

図 4.3: 適切なインデントと数学的表式を連想できる変数名を用いたプログラム。

件)で行なっているが、そこに現れる微小量を条件の中に直接記述するのではなく、`eps` として定数定義している（3行目）。こうすることによっ

て、プログラムの内容に修正を加えることなく、`#define` 文の部分を修正するだけで、収束条件を変更することが可能となる。この例では、収束判定を行なっているのが 1箇所のみなので、大きな改善には見えないが、収束判定が多く箇所で行われるような大規模なプログラムに発展した際には、プログラミングミス（バグ）を防ぐ重要な役割を果たす。

図 4.3 のプログラムでは、 $C > 0$  が入力された場合には問題ないが、このアルゴリズムは  $f(x)$  の  $x$  切片が存在する場合のものであるから、もし  $C \leq 0$  の値が入力された場合にはその動作が保証できない。実際  $C < 0$  の値が入力されると、無限ループに落ちいり、正常な結果がえられない。これでは、正しく動作するプログラムとは言えない。

### goto 文を用いた処理の流れが分かりにくい例

そこで、 $C < 0$  の場合の問題を避けるために、以下のように `goto` 文を用いることもできる。(15, 24 行目) しかし、`goto` 文は `while` 文をまたいでおり、わかりづらい。この程度の長さのプログラムなら、24行目が飛んだ先だと簡単に認識できるが、100行～1000行先、つまりひと目で認識できない先に処理が飛んだ場合、データの流れを認識するのに混乱をきたす原因となる。なおかつ、24行目は  $C > 0$  の場合にも実行されるので、適切な解決とは言えない。

### 処理の流れを明瞭にする条件分岐の例

図 4.5 に、`if` 文だけを用いて  $C < 0$  の場合の問題を避けたプログラムを示した。このようなアルゴリズム構造にすることによって、処理の流れはを見失うことなくなおかつ、インデントを用いることで、その構造を把握することも容易となる。すなわち、バグが発生する可能性を小さくすることが出来る。

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main() {
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    if(C<=0) goto end;
16    x=C;
17    dx=-(x*x-C)/(2.0*x);
18    while(fabs(dx)>eps) {
19        x=x+dx;
20        printf("%12.10f \n",x);
21        dx=-(x*x-C)/(2.0*x);
22    }
23
24    end: printf("%12.10f \n",x);
25
26 }
```

図 4.4: goto 文を用いて  $C < 0$  の場合の問題を避けたプログラム。処理の流れを分かりづらくする可能性がある。

### 関数をつかう例

図 4.6 に、関数を用いたプログラムを示す。関数を用いることのメリットは、同じ処理を何度もプログラム中に記述することを避けるためであったり、あるいは、複雑な一連の処理をひとかたまりとして抽象化できることである。この例のようにアルゴリズム自体がもともと単純な場合には、それらのメリットはあまりないかもしれない。それでもこのように関数を用いると、数学的な表式とプログラム中の関数の名前を類似したものにすることによって、バグが生じる危険性を軽減できる。また、 $f(x) = x^2 - C$  以外の関数にたいして、ニュートン・ラフソン法を適用する場合にも、main 関数の部分を直接変更することなく、関数の内部だけを修正することによって、変更を済ませることができる。

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main(){
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    if(C<=0){
16        printf("C is zero or negative.\n");
17    }else{
18        x=C;
19        dx=-(x*x-C)/(2.0*x);
20        while(fabs(dx)>eps){
21            x=x+dx;
22            printf("%12.10f \n",x);
23            dx=-(x*x-C)/(2.0*x);
24        }
25    }
26}
27

```

図 4.5: if 文だけを用いて  $C < 0$  の場合の問題を避けたプログラム。while 文をまたいで、処理を遠くに飛ばさない。また、インデントを用いて、処理の構造を視覚的にとらえやすくする。

## 4.4 再利用して発展できるコード

図 4.2 に示されたコード（プログラム）と、図 4.6 に示されたコードを見比べると前者の方が行数が少なくまとまっているので、一見、簡潔で分かりやすいプログラムに見えるかもしれない。どちらがよりデータの流れ、および処理の構造を理解しやすいかよく見比べて欲しい。

コードを書いた時点では、データの流れや処理の構造は脳の中にイメージされている。両者の違いは理解できないかもしれない。むしろ前者のコードの方が理解しやすいと感じるかもしれない。

しかし、プログラムソースコードは使い捨てされると限らない。むしろ数ヶ月あるいは数年後に、そのコードを再び利用することがある

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 double Func(double x, double C);
6 double dFunc(double x);
7
8 int main(){
9
10    double C;
11    double dx;
12    double x;
13
14    printf("C=?");
15    scanf("%lf",&C);
16    printf("%lf\n",C,eps);
17
18    if(C<=0){
19        printf("C is zero or negative.\n");
20    }else{
21        x=C;
22        dx=-Func(x,C)/dFunc(x);
23        while(fabs(dx)>eps){
24            x=x+dx;
25            printf("%12.10f \n",x);
26            dx=-Func(x,C)/dFunc(x);
27        }
28    }
29
30 }
31
32 double Func(double x, double C){
33     double y;
34     y= x*x - C;
35     return y;
36 }
37
38 double dFunc(double x){
39     double y;
40     y=2.0*x;
41     return y;
42 }
```

図 4.6: 関数を用いてプログラムを記述すると、数学的表式と類似性があるので、バグの防止に役立つ。また、異なる関数についてのプログラムを作成するばあいに、メインルーチンを修正せずに済む。

であろう。あるいは改良を加えて他のプログラムの一部として活用するかもしれない。時間を置いた再利用ではないとしても、コードを書いた本人以外がそれを利用する場合もある。

そのような状況においても、データの流れと処理の構造が理解しやすいということは、短い行数で書かれているということよりも、本質的に重要である。

また、新しく作られるコードが、正しく動作するプログラムであるためにも、このことは必要である。正しく動作しないコードを一部にでも含むプログラムは、全体として正しく動作するプログラムとは言えないものである。

## 4.5 円周率 $\pi$ の近似値を求める

半径 1 の円の円周の長さは  $2\pi$  である。すなわち、この長さを  $L_c$  とすると、 $L_c = 2\pi$  である。

次に、この円に内接する正  $2^{n+1}$  角形 ( $n = 1, 2, \dots$ ) を考える(図 4.7 参照)。この多角形の一辺の長さを  $L_n$  とする。

この多角形は  $n$  を大きくしていくと、円に近づく。したがって、辺の長さ  $L = 2^{n+1}L_n$  は  $L_c$  に近づく。半径 1 の円周の長さを 2 で割ったものが  $\pi$  であるから、 $2^n L_n$  の値が  $\pi$  の近似値を与える。

では、具体的に  $L_n$  の値はどのように求められるだろうか？ $n+1$  角形の一辺の長さは、 $n$  角形の一辺の長さから求めることができる。つまり、 $n=1$  の 4 角形から順番に 8 角形、16 角形、32 角形…と求めていくことができる。

図 4.7 とピタゴラスの定理から、

$$x^2 + \left(\frac{L_1}{2}\right)^2 = 1 \quad (4.5)$$

$$y^2 + \left(\frac{L_1}{2}\right)^2 = L_2^2 \quad (4.6)$$

$$x + y = 1 \quad (4.7)$$

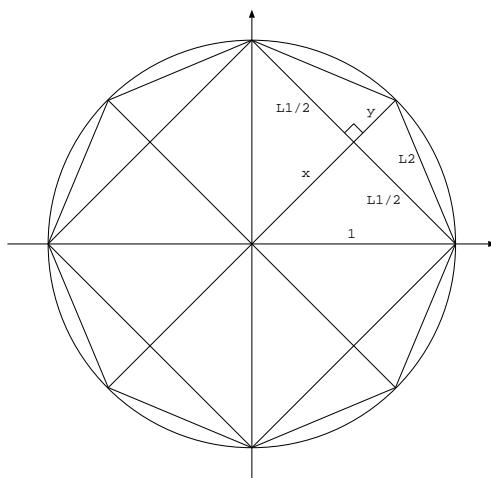


図 4.7: 半径 1 の円と、それに内接する正  $2^{n+1}$  角形 ( $n = 1, 2, \dots$ )

である。これは  $n = 1$  に対する関係であるが、一般の  $n$  に対しても

$$x^2 + \left(\frac{L_n}{2}\right)^2 = 1 \quad (4.8)$$

$$y^2 + \left(\frac{L_n}{2}\right)^2 = L_{n+1}^2 \quad (4.9)$$

$$x + y = 1 \quad (4.10)$$

が成り立つ。

これら 3 つの式から、 $x, y$  を消去して、 $L_{n+1}$  と  $L_n$  の関係式を導けば、円に内接する多角形の一辺の長さ  $L_n$  に関する漸化式が得られる。

## 課題

1.  $x, y$  を消去することにより,  $L_n$  の漸化式を求めなさい.
2.  $L_n$  の漸化式を用いて  $\pi$  の近似値を求めなさい.  
その際、前節でつくった平方根のプログラムを関数として用いること.
3. 求められた  $\pi$  の近似値を  $n$  の関数としてグラフにすること (付録??参照).  
漸化式の繰り返しを多くしすぎると、誤差により正確に  $\pi$  の近似値が求められないので、注意すること.

## 関数を使ったアルゴリズム例

```

Function main() is
  n = 1
  pow = 2 ( $2^n$  を求めるための変数)
  L =  $\sqrt{2}$ 
  while n ≤ Nmax do
    pow × L を出力
    L ← 漸化式の右辺 (SqrtNewton(y)) の計算含む
    pow ← pow × 2
  end
end

Function SqrtNewton(x) is
  r = (ニュートン・ラフソン法を用いた平方根の計算)
  return r
end

```

図 4.8: 多角形から、 $\pi$  の近似値を求めるアルゴリズムの主な部分。平方根の計算にニュートン・ラフソン法を用いた。

## $\pi$ の近似値の計算結果例

図 4.9 に計算結果例をグラフに示した。左の図から  $n = 5$  以上の  $n$  の値に対して、3.14 に近い値が求められていることが分かる。

右の図は、同じ結果を、縦軸だけを拡大したものである。小数点以下 6 衔程度まで求められていることが分かる。

しかし、 $n$  の値を大きくしていくば、順調に精度が向上するというわけではないことも見て取れる。 $n > 15$ においては、この方法で求めた近似値が  $\pi$  の真の値から大きくずれ始めていることがわかる。これは、計算を繰り返したことによる数値誤差の発生が原因であると考えられる(第 ?? 章参照)。

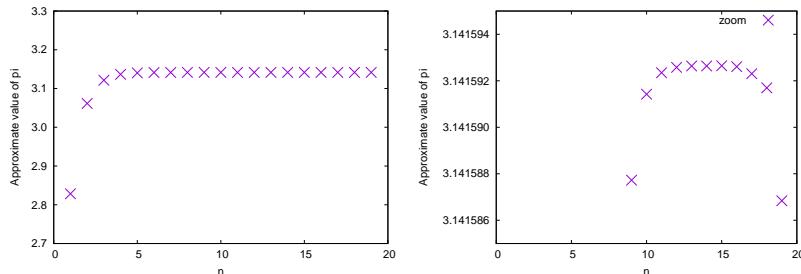


図 4.9: 円周率  $\pi$  の近似値を、円の内接多角形を用いて計算した結果例。左の図から、 $n = 5$ あたりから、3.14に近い値が得られていることがわかる。右の図は、左の図と同じ結果であるが、縦軸を拡大して描画した。

$L_n$  の満たす漸化式

(4.8) 式から (4.9) 式を引くと、

$$\begin{aligned} x^2 - y^2 &= 1 - L_{n+1}^2 \\ L_{n+1}^2 &= 1 - x^2 + y^2 \end{aligned} \quad (4.11)$$

である。いっぽう、(4.10) 式から

$$\begin{aligned} y &= 1 - x \\ y^2 &= 1 - 2x + x^2 \end{aligned} \quad (4.12)$$

であるから、(4.12) 式を (4.11) 式に代入すると、

$$L_{n+1}^2 = 2 - 2x \quad (4.13)$$

である。 $x$  は、(4.8) 式から

$$\begin{aligned} x^2 &= 1 - \left(\frac{L_n}{2}\right)^2 \\ x &= \sqrt{1 - \left(\frac{L_n}{2}\right)^2} \end{aligned} \quad (4.14)$$

この(4.14)式を(4.13)式に用いると、

$$L_{n+1}^2 = 2 - 2 \sqrt{1 - \left(\frac{L_n}{2}\right)^2} \quad (4.15)$$

$$L_{n+1} = \sqrt{2 - 2 \sqrt{1 - \left(\frac{L_n}{2}\right)^2}} \quad (4.16)$$

となり、これが $L_n$ の満たす漸化式である。

### 関数を使ったプログラム例

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-15
4 #define Nmax 20
5
6 double Func(double x, double C);
7 double dFunc(double x);
8 double SqrtNewton(double C);
9
10 int main(){
11     double a,a2;
12     double l;
13     double m;
14     int n;
15
16     n=2;
17     m=2*2;
18     a=SqrtNewton(2.0);
19     a2=2.0;
20     l=SqrtNewton(a2)*m;
21     while(n<Nmax){
22         printf("%d %19.16e %19.16e %19.16e\n",n,m,sqrt(1-a2/4),l/2);
23         a2=2.0-2.0*SqrtNewton(1.0-a2/4.0);
24         m=m*2;
25         l=SqrtNewton(a2)*m;
26         n++;
27     }
28 }
29
30 double SqrtNewton(double C){
31
32     double dx;
33     double x;
34
35     if(C<=0){
36         printf("C is zero or negative.\n");

```

```
37 }  
38     }  
39     x=C;  
40     dx=-Func(x,C)/dFunc(x);  
41     while(fabs(dx)>eps){  
42         x=x+dx;  
43         //printf("%20.17e \n",x);  
44         dx=-Func(x,C)/dFunc(x);  
45     }  
46  
47     return x;  
48 }  
49  
50  
51 double Func(double x, double C){  
52     double y;  
53     y= x*x - C;  
54     return y;  
55 }  
56  
57 double dFunc(double x){  
58     double y;  
59     y=2.0*x;  
60     return y;  
61 }
```

図 4.10: 正  $2^{n+1}$  角形をつかって、 $\pi$  の値を求めるプログラム例。平方根の計算にニュートン・ラフソン法を関数として用いている。

## 4.6 数値解析における計算誤差

## 4.7 電子計算機内の数

電子計算機（コンピューター）の内部では、すべての数値は有限桁の2進数で表されている。

有限桁で内部表現されているのであるから、無理数など小数点以下無限に数値がつづく小数は、厳密に内部表現することができない。また、10進数表記では小数点以下1桁で表される実数、たとえば0.3などでも、2進数表記を用いると循環小数 $0.010011001\dots$ となり、厳密に内部表現することはできない。

のことからも、数値計算において得られる値は、あくまでも近似解であり、誤差を含んでいることが分かる。どの程度誤差を含んでいるか、すなわち計算精度について意識しなくてはならない。

### 4.7.1 整数型

32ビットで整数*i*を表現した場合、1ビットは符号を表すために使われるるので、

$$-2147483647 \leq i \leq 2^{31} - 1 = 2147483647 \quad (4.17)$$

となる。

### 4.7.2 実数型

#### 浮動小数点

大きな数、例えば749321940000のような数を、このように表すと、この数が何桁あるのか読み取りにくい。そこで、

$$749321940000 = 7.4932194 \times 10^{11} = 0.74932194 \times 10^{12} \quad (4.18)$$

の様に指数表記すると、12桁の数であることがすぐに分かる。

ちなみに、日本語でこの数字を読むと七千四百九十三億、二千百九十四万、となる<sup>2</sup>。

また、小さい値、たとえば0.0000003481なども

$$0.0000003481 = 3.481 \times 10^{-7} = 0.3481 \times 10^{-6} \quad (4.19)$$

の様に表す。

小数点をどこに打つかによって、指数部の値も変化する。このような数の表記方法を浮動小数点表記と呼ぶ。科学・技術や数値計算の分野では、ほとんどこの浮動小数点表記を用いる。

### float型とその計算精度

コンピューターの内部で、実数を浮動小数点(float)表記する場合、符号、指数部および仮数部の3つの要素で表される。

$$r = \text{符号} \times \text{仮数部} \times 2^{\text{指数部}} \quad (4.20)$$

32bit(4byte)でfloat型を表す場合、符号1bit、指数部8bit、仮数部23bit用いる。

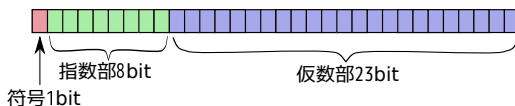


図 4.11: 実数を32bit(4byte)で表した場合の符号、指数部および仮数部

指数部の内部表現は8bitの二進数であるので、0～255の数を表すことができる。

---

<sup>2</sup>お金で読み上げる場合はこのように億や万という単位を用いる。万 =  $10^4$ 、億 =  $10^8$ 、兆 =  $10^{12}$ を覚えておくと便利である。

しかし、指数部は負の数も表さなければならない。そこで、この8bitを使って $-128 \sim 127$ の数が表される。つまり、float型で表記できる数の範囲は

$$10^{-39} \simeq 2^{-128} \leq |r| \leq 2^{127} \simeq 10^{38} \quad (4.21)$$

となる。

仮数部は23bit用いて表される。

$$\begin{aligned} 2^{23} &= 10^x \\ x &= 23 \times \log 2 \simeq 7 \end{aligned} \quad (4.22)$$

であるから、このfloat型で表すことが出来る10進数は約7桁であることが分かる。つまり、float型を用いて数値計算を行う限り、7桁以上の計算精度を望むことはできない。

### double型とその計算精度

いっぽう、実数をdouble型で宣言して数値計算を行う場合、指数部は11bit、仮数部は52bitで表現されるので、表すことの出来る数の範囲は

$$10^{-308} \simeq 2^{-1024} \leq |r| \leq 2^{1023} \simeq 10^{308} \quad (4.23)$$

である。また、計算精度は

$$\begin{aligned} 2^{52} &= 10^x \\ x &= 52 \times \log 2 \simeq 15.6 \end{aligned} \quad (4.24)$$

となる。つまり、15桁程度の数値計算精度を得ることができる。

### 4.7.3 少数点以下の2進数表記

小数点以下の数値、つまり1より小さい数値を2進数で表すと、たとえば

$$0.010011 \quad (4.25)$$

の様に表される。この例の場合、10進数で表すと、 $2^{-2}, 2^{-5}$  および  $2^{-6}$  の桁が 1 であるから、

$$2^{-2} + 2^{-5} + 2^{-6} = 0.25 + 0.03125 + 0.015625 \quad (4.26)$$

$$= 0.296875 \quad (4.27)$$

である。

小数を2進数で表した、一般的な形は

$$0.b_{-1}b_{-2}b_{-3}\cdots b_{-n}\dots \quad (4.28)$$

$$b_{-n} \in \{0, 1\} \quad (n = 1, 2, \dots) \quad (4.29)$$

という形をしている。 $b_{-n}$  は、0か1のどちらかである。

この表記の数値を10進数で表す一般的な形は

$$b_{-1}2^{-1} + b_{-2}2^{-2} + \cdots + b_{-n}2^{-n} + \cdots = \sum_{n=1}^{\infty} b_{-n}2^{-n} \quad (4.30)$$

となる。

この表記を見ると、10進数を2進数に変換するやり方が理解しやすい。  
(4.30)式に2を掛けると、

$$b_{-1} + b_{-2}2^{-1} + \cdots + b_{-n}2^{-n+1} + b_{-n-1}2^{-n} + \cdots \quad (4.31)$$

$$= b_{-1} + \sum_{n=1}^{\infty} b_{-n-1}2^{-n} \quad (4.32)$$

である。(4.32)式の2項目は小数点以下つまり1より小さい数を表しているので、 $b_{-1}$  が1桁目、つまり  $10^0$  の桁を表す。つまり、元の数値に2を掛けて、その小数部分を捨てた整数部が  $b_{-1}$  である。

さらにこの小数部に2を掛けて、その整数部が  $b_{-2}$  である。この手続きを順次繰り返すことによって、 $b_{-n}$  の値を知ることが出来る。

たとえば、最初の例を用いると

$$\begin{aligned}
 0.296875 \times 2 &= 0.593750 \Rightarrow b_{-1} = 0 \\
 0.593750 \times 2 &= 1.187500 \Rightarrow b_{-2} = 1 \\
 0.187500 \times 2 &= 0.375000 \Rightarrow b_{-3} = 0 \\
 0.375000 \times 2 &= 0.750000 \Rightarrow b_{-4} = 0 \\
 0.750000 \times 2 &= 1.500000 \Rightarrow b_{-5} = 1 \\
 0.500000 \times 2 &= 1.000000 \Rightarrow b_{-6} = 1
 \end{aligned} \tag{4.33}$$

小数部が 0 となれば、手続きは終了である。この結果から

$$(0.296875)_{10} = (0.010011)_2 \tag{4.34}$$

となり、確かに (4.25) 式に等しい。

ここで、数値が 10 進数表記なのか、2 進数表記なのか明示するために、 $(\cdots)_{10}$  は数値が 10 進数表記されていることを、また  $(\cdots)_2$  は 2 進数表記されていることを表している。

#### 4.7.4 有限桁数の2進数で表せない小数

では、どんな小数でも2進数で表すことが出来るかというと、そうではない。たとえば、 $(0.3)_{10}$ を2進数表記に変換すると、

$$\begin{aligned}
 0.3 \times 2 &= 0.6 \Rightarrow b_{-1} = 0 \\
 0.6 \times 2 &= 1.2 \Rightarrow b_{-2} = 1 \\
 0.2 \times 2 &= 0.4 \Rightarrow b_{-3} = 0 \\
 0.4 \times 2 &= 0.8 \Rightarrow b_{-4} = 0 \\
 0.8 \times 2 &= 1.6 \Rightarrow b_{-5} = 1 \\
 0.6 \times 2 &= 1.2 \Rightarrow b_{-6} = 1 \\
 0.2 \times 2 &= 0.4 \Rightarrow b_{-7} = 0 \\
 0.4 \times 2 &= 0.8 \Rightarrow b_{-8} = 0 \\
 0.8 \times 2 &= 1.6 \Rightarrow b_{-9} = 1 \\
 0.6 \times 2 &= 1.2 \Rightarrow b_{-10} = 1
 \end{aligned} \tag{4.35}$$

すなわち、

$$(0.3)_{10} = (0.0100110011\cdots)_2 \tag{4.36}$$

となる。2進数表記の小数点以下2桁目以下は、1001を繰り返す、循環小数であることがわかる。

$$(0.3)_{10} = (0.0\underline{1001})_2 \tag{4.37}$$

これは、(4.35)式で、 $b_{-2}$ の計算において $0.6 \times 2 = 1.2$ が現れ、再び $b_{-6}$ において同じ計算が現れるので、以下同じ計算が繰り返されることにより分かる。

課題

$(0.1)_{10}$ を2進数表記に変換せよ。

## 4.8 オーバーフローとアンダーフロー

各型によって表現できる数には範囲があることがわかった。この数値範囲を超えてその絶対値が大きくなることを、オーバーフローという。また小さくなることをアンダーフローという。

例題： $m^n$  から  $m^l$  まで計算するプログラム。 $(m, n, l$  は整数型)

---

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int main(){
6     int m,n,l;
7     int i;
8     int r;
9
10    printf("Input M,N,L. (Positive integer and N<L)\n");
11    scanf("%d,%d,%d",&m,&n,&l);
12    if(m>0 && n>0 && l>0) {
13        printf("M=%d, N=%d, L=%d\n",m,n,l);
14    }else{
15        printf("Illegal number.\n");
16        exit(1);
17    }
18
19    r=pow(m,n);
20    i=n;
21    while(i<=l){
22        printf("%d**%d=%d\n",m,i,r);
23        r=r*m;
24        i++;
25    }
26 }
```

図 4.12: オーバーフローを観察するプログラム例

---

```
1 Input M,N,L. (Positive integer and N<L)
2 M=2, N=29, L=40
3 2**29=536870912
4 2**30=1073741824
5 2**31=-2147483648
6 2**32=0
7 2**33=0
8 2**34=0
9 2**35=0
10 2**36=0
11 2**37=0
12 2**38=0
13 2**39=0
14 2**40=0
```

---

```
1 Input M,N,L. (Positive integer and N<L)
2 M=10, N=8, L=19
3 10**8=100000000
4 10**9=1000000000
5 10**10=1410065408
6 10**11=1215752192
7 10**12=-727379968
8 10**13=1316134912
9 10**14=276447232
10 10**15=-1530494976
11 10**16=1874919424
12 10**17=1569325056
13 10**18=-1486618624
14 10**19=-1981284352
```

図 4.13: 図 4.12 で示したプログラムの実行例

**課題**

$2^{-i}$  を求めるプログラムを作成し、利用する計算機においてアンダーフローを起こす  $i$  の値を確認を確認してみよう。

## 4.9 誤差と有効精度桁数

真値  $a$ , 近似値  $\hat{a}$  とするとき, 誤差  $E(a)$  はその差で定義される。

### 誤差 (error)

$$E(a) = \hat{a} - a \quad (4.38)$$

### 絶対誤差 (absolute error)

その絶対値を絶対誤差と呼ぶ。

$$|E(a)| = |\hat{a} - a| \quad (4.39)$$

### 相対誤差 (relative error)

誤差の真値に対する割合を相対誤差  $rE(a)$  と言うが, 真値は不明な場合がほとんどである。真値がわかっていないから, 数値計算が必要になるわけである。そこで, 真値の代わりに近似値  $\hat{a}$  で置き換えたものを相対誤差の近似値とする場合が多い。

$$rE(a) = \frac{E(a)}{a} = \frac{\hat{a} - a}{a} \quad (a \neq 0) \quad (4.40)$$

$$\approx \frac{E(a)}{\hat{a}} \quad (4.41)$$

### 有効精度桁数

有効桁精度は相対誤差から次のように求められる.

$$-\log|rE(a)| \simeq \log|\hat{a}| - \log|E(a)| \quad (4.42)$$

つまり、値の桁数から誤差の桁数を差し引いたものが有効精度である.

## 4.10 丸め誤差

10進  $l$  桁計算で、 $(l+1)$  桁目を切り捨て、または四捨五入することによる誤差を丸め誤差という。計算末尾の  $l$  桁目に入る。

$$|\text{相対丸め誤差}| < 10^{-l+1} \quad (4.43)$$

$$\text{有効桁精度} = -\log|\text{相対丸め誤差}| > l-1 \quad (4.44)$$

無理数や、先に述べた2進数表記することによる循環小数など、数値計算においては必ず丸め誤差が生じる。

## 4.11 打ち切り誤差

無限級数和を有限和で近似した時の誤差を打ち切り誤差と言う。例として、指數関数をテーラー展開すると、つぎの様な無限級数和で表される。

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots \quad (4.45)$$

$$= \sum_{k=0}^{\infty} \frac{x^k}{k!} \quad (4.46)$$

ここで、 $k!$  は  $k$  の階乗、すなわち  $k! = k \times k-1 \times \dots \times 2 \times 1$  を表す。 $(4.46)$  式には  $0! = 1$  も現れる。 $0! = 1$  と定義されることに注意する必要がある。

無限級数和の中の、数列を  $A_k$  と書くことにすると、

$$A_k = \frac{x^k}{k!} \quad (4.47)$$

であるので、

$$A_{k+1} = A_k \times \frac{x}{k+1} \quad (4.48)$$

という漸化式が成り立つ。

この漸化式を利用して、指数関数  $e^x$  の値を求めるプログラム例を図 4.14 に示した。

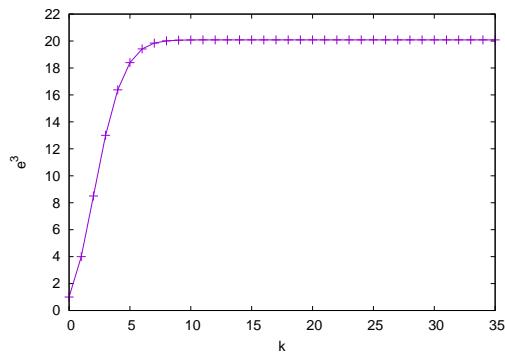
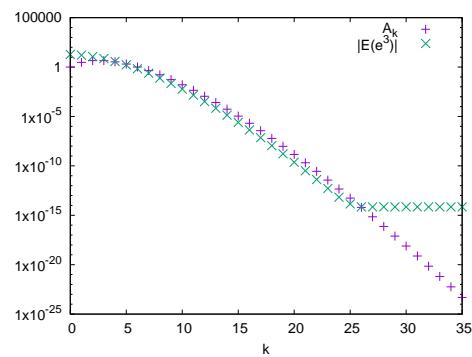
---

```

1 // exp(x) を求める プログラム
2 #include <stdio.h>
3 #include <math.h>
4 #define Nmax 35
5
6 int main(){
7     int k;
8     float x;
9     double a,expo;
10    double error;
11
12    printf("Input x\n");
13    scanf("%f",&x);
14    printf("Exp(%f)\n",x);
15
16    printf("# k %*s ,8," );
17    printf("A %*s ,18," );
18    printf("Exp(x) %*s ,16," );
19    printf("Error\n");
20    k=0;
21    a=1.0;
22    expo=a;
23    while(k<=Nmax){
24        error=expo-exp(x);
25        printf(" %2d %22.15e %22.15e %22.15e\n",k,a,expo,error);
26        a=a*x/(k+1);
27        expo=expo+a;
28        k++;
29    }
30
31 } // main 関数のおわり

```

図 4.14: 指数関数の値を、漸化式を用いて求めるプログラム例

図 4.15:  $e^3$  を級数展開を用いて求めた例図 4.16:  $e^3$  を級数展開を用いて求めた場合の絶対誤差の変化

## 課題

三角関数  $\sin x$  について

1.  $\sin x$  のテーラー展開を無限級数和として表わせ.
2. 級数和の中の数列が満たす漸化式を求めよ.
3. その漸化式を用いたプログラムを作成し,  $x$  の様々な値に関して  $\sin x$  の値を求めよ. また, その結果における打ち切り誤差や丸め誤差について比較検討せよ.

## 収束判定

一般に, 部分和を

$$S_m = \sum_{k=1}^m A_k \quad (4.49)$$

とすると, 計算機内において  $S_m$  の値に変化が生じなくなる, すなわち

$$S_m = S_{m+1} \quad (4.50)$$

のとき, 計算はその時点まで限界であると判定される.

有効精度桁数が分かっている場合は, その桁よりも小さい変化になつたら数値計算が収束したと判定することもできる.

 $S_m$  の絶対誤差 ( $Aerr$ )

$p$  進  $l$  桁の場合における部分和  $S_m$  の絶対誤差は

$$Aerr < \max_{k=1,\dots,m} |A_k| \times p^{-l+1} \quad (4.51)$$

によってその上限を見積もることができる.

## 4.12 方程式の解法における計算誤差

方程式  $f(x) = 0$  を満たす  $x$  を求める方法.  $f(x)$  が  $x$  の多項式であるとき,  $f(x) = 0$  を代数方程式という. たとえば,

$$a_1x + a_2x^2 + \cdots + a_nx^n + a_{n+1} = 0 \quad (4.52)$$

のような形をしている.

超越関数とは多項式で表せない関数のことで, 指数関数, 対数また三角関数などのことである. たとえば,

$$x - c \tanh x - M = 0 \quad (4.53)$$

のような形をした方程式のことである.

2次方程式ならば, 解(根)の公式が存在するが, 代数方程式や超越方程式の解を一般に解析的に求めることは不可能である. そのため, 数値的にその解を求める必要がある. よく知られている反復法として, 二分法, 線形逆補完法, ニュートン法などがある.

### 二分法

連続関数  $f(x)$  が,

$$f(x_1) \times f(x_2) < 0 \quad (4.54)$$

を満たせば, 区間  $(x_1, x_2)$  に少なくとも 1つ解が存在することがわかる. 何らかの方法でその解がただひとつであることが分かったとする.

区間  $(x_1, x_2)$  の中点  $x_3$  として, 2つの区間  $(x_1, x_3)$  と  $(x_3, x_2)$  に二等分する. 解  $a$  はそれらの, いずれかの区間に内に存在するはずである.  $f(x_1)f(x_3)$  と  $f(x_3)f(x_2)$  の値をそれぞれ調べる.もし, 図4.17に示したように  $f(x_1)f(x_3) < 0$  ならば, 区間  $(x_1, x_3)$  に存在することが分かる. さらに  $(x_1, x_3)$  を二等分する. このような処理を  $k$  回繰り返すと, 解の存在する区間の幅  $d_k$  は

$$d_k = |x_2 - x_1|2^{-k} \quad (4.55)$$

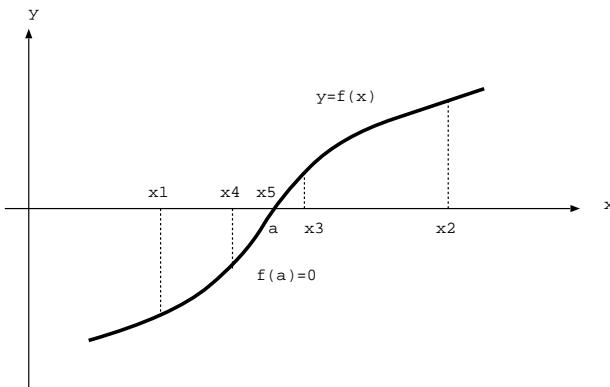


図 4.17: 二分法によって、解の存在する区間を縮小していく

と、指数関数的に小さくなっていく。 $d_k$  が計算限界まで小さくなる区間を  $(x_k, x_{k'})$  とすると、数値的には  $x_k = x_{k'}$  であり、 $x_k$  が  $f(x) = 0$  の数値解である。

以下では、真の解を  $a$  で表す。つまり  $f(a) = 0$  である。

### 課題

関数  $f(x)$  が

$$f(x) = x^2 - b \quad (b > 0) \quad (4.56)$$

のとき、代数方程式  $f(x) = 0$  の数値解を二分法を用いて求めなさい。また、ニュートン・ラフソン法を用いてその解を求め、2分法と比較しなさい。ただし、单精度(float)を用いて、反復計算が進むにつれて  $x_k$  および絶対誤差  $|x_k - a|$  がどのように変化するか求め、片対数グラフ化すること。

## 4.13 ガウスの消去法

ガウスの消去法で、連立一次方程式の解を求める。 $n$  個の変数  $x_i$  ( $i = 1, 2, \dots, n$ ) があるとき、次の  $n$  連立一次方程式が分かれば、変数の値を求めることができる。

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots &\quad \vdots \quad \ddots \quad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{4.57}$$

この連立方程式は行列  $\hat{A}$  を

$$\hat{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \tag{4.58}$$

とし、ベクトル  $\vec{x}, \vec{b}$  を

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{4.59}$$

とすると、(4.57) 式は

$$\hat{A}\vec{x} = \vec{b} \tag{4.60}$$

となる。

### 4.13.1 前進消去

いま、 $n = 4$  の場合を例にとる。

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \quad (4.61)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \quad (4.62)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \quad (4.63)$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \quad (4.64)$$

$a_{11} \neq 0$  のとき、1行目をまず選び（ピボット）(4.62) 式 -  $\frac{a_{21}}{a_{11}}$ (4.61) 式を計算すると、

$$\begin{aligned} & (a_{21} - a_{11})x_1 + (a_{22} - \frac{a_{21}}{a_{11}}a_{12})x_2 + (a_{23} - \frac{a_{21}}{a_{11}}a_{13})x_3 \\ & + (a_{24} - \frac{a_{21}}{a_{11}}a_{14})x_4 = b_2 - \frac{a_{21}}{a_{11}}b_1 \end{aligned} \quad (4.65)$$

1項目はゼロになる。ここで、

$$a_{2j}^{(1)} \equiv a_{2j} - \frac{a_{21}}{a_{11}}a_{1j} \quad (j = 2, \dots, 4) \quad (4.66)$$

$$b_2^{(1)} \equiv b_2 - \frac{a_{21}}{a_{11}}b_1 \quad (4.67)$$

と定義すれば、(4.65) 式は

$$a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 = b_2^{(1)} \quad (4.68)$$

と書ける。同様の計算を(4.63)式および(4.64)式にも行うと、

$$\left( \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ 0 & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{array} \right) \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) = \left( \begin{array}{c} b_1 \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{array} \right) \quad (4.69)$$

となる。ただし

$$a_{ij}^{(1)} \equiv a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j} \quad (i = 2, \dots, 4; j = 1, \dots, 4) \quad (4.70)$$

$$b_i^{(1)} \equiv b_i - \frac{a_{i1}}{a_{11}}b_1 \quad (i = 2, \dots, 4) \quad (4.71)$$

とした。

つぎに  $a_{22}^{(1)} \neq 0$  と仮定し<sup>3</sup> 2行目をピボットに選び

$$a_{ij}^{(2)} \equiv a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)} \quad (i = 3, 4; j = 2, \dots, 4) \quad (4.72)$$

$$b_i^{(2)} \equiv b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)} \quad (i = 3, 4) \quad (4.73)$$

と求めると、

$$\left( \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & 0 & a_{43}^{(2)} & a_{44}^{(2)} \end{array} \right) \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) = \left( \begin{array}{c} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(2)} \end{array} \right) \quad (4.74)$$

最後に 3 行目をピボットに選び

$$a_{ij}^{(3)} \equiv a_{ij}^{(2)} - \frac{a_{i3}^{(2)}}{a_{33}^{(2)}} a_{3j}^{(2)} \quad (i = 4; j = 3, 4) \quad (4.75)$$

$$b_i^{(3)} \equiv b_i^{(2)} - \frac{a_{i3}^{(2)}}{a_{33}^{(2)}} b_3^{(2)} \quad (i = 4) \quad (4.76)$$

と求めると、

$$\left( \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & 0 & 0 & a_{44}^{(3)} \end{array} \right) \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) = \left( \begin{array}{c} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ b_4^{(3)} \end{array} \right) \quad (4.77)$$

となり、 $\hat{A}$  は上三角行列に変換された。

---

<sup>3</sup> $a_{22}^{(1)} = 0$  あるいは非常に絶対値が小さい場合には、行を入れ替えなければならない（ピボット選択）

まとめると、 $m = 1, \dots, n-1$  に対して、(ただしこの例では  $n = 4$ )

$$a_{ij}^{(m)} \equiv a_{ij}^{(m-1)} - \frac{a_{im}^{(m-1)}}{a_{mm}^{(m-1)}} a_{mj}^{(m-1)} \quad (i = m+1, \dots, n; j = m, \dots, n) \quad (4.78)$$

$$b_i^{(m)} \equiv b_i^{(m-1)} - \frac{a_{im}^{(m-1)}}{a_{mm}^{(m-1)}} b_m^{(m-1)} \quad (i = m+1, \dots, n) \quad (4.79)$$

という漸化式で  $\hat{A}$  と  $\vec{b}$  を更新していく事になる。

### 4.13.2 後退代入

この時点では、 $\vec{x}$  の値はまだ求まっていない。

(4.77) 式の係数行列を改めて  $\hat{A}$  とし、その要素を  $a_{ij}$  と表すことにする。また、右辺を  $\vec{b}$  と表し、要素を  $b_i$  と表することにする。 $x_4$  の値はすぐに求めることができる。

$$x_4 = \frac{b_4}{a_{44}} \quad (4.80)$$

$x_4$  の値が求まったので、それを用いてつぎに  $x_3$  は

$$x_3 = \frac{1}{a_{33}}(b_3 - a_{34}x_4) \quad (4.81)$$

同様に順番に

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^4 a_{ij}x_j) \quad (i = 3, \dots, 1) \quad (4.82)$$

と  $\vec{x}$  の値を求めることができる。

ここまででは変数の数  $n = 4$  を例に示したが、変数の数が  $n$  個ある場合には

$$x_n = \frac{b_n}{a_{nn}} \quad (4.83)$$

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^n a_{ij}x_j) \quad (i = n-1, \dots, 1) \quad (4.84)$$

と後退代入することで、連立方程式の解を計算することができる。

課題

つきの連立 1 次方程式の解をガウスの消去法で求めるプログラムを作成して  $x_i$  の値を求めよ。

$$\left( \begin{array}{cccc|c} 4 & -6 & -10 & -2 & -46 \\ -6 & 8 & 21 & -1 & 69 \\ -10 & 21 & -20 & 23 & 64 \\ -2 & -1 & 23 & -18 & -7 \end{array} \right) \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) = \left( \begin{array}{c} -46 \\ 69 \\ 64 \\ -7 \end{array} \right) \quad (4.85)$$

Mtx.dat ファイルの内容

```

1   4
2   4.0  -6.0 -10.0 -2.0 -46.0
3   -6.0   8.0  21.0 -1.0  69.0
4  -10.0  21.0 -20.0  23.0  64.0
5   -2.0  -1.0  23.0 -18.0  -7.0

```

```

1 N=4
2 Original Matrix:
3   4.0000   -6.0000   -10.0000   -2.0000   -46.0000
4   -6.0000    8.0000   21.0000   -1.0000    69.0000
5  -10.0000   21.0000   -20.0000   23.0000    64.0000
6   -2.0000   -1.0000   23.0000   -18.0000   -7.0000
7 Forward elimination:
8 m=1
9   4.0000   -6.0000   -10.0000   -2.0000   -46.0000
10  0.0000   -1.0000    6.0000   -4.0000    0.0000
11  0.0000    6.0000   -45.0000   18.0000   -51.0000
12  0.0000   -4.0000   18.0000   -19.0000   -30.0000
13 m=2
14  4.0000   -6.0000   -10.0000   -2.0000   -46.0000
15  0.0000   -1.0000    6.0000   -4.0000    0.0000
16  0.0000    0.0000   -9.0000   -6.0000   -51.0000
17  0.0000    0.0000   -6.0000   -3.0000   -30.0000
18 m=3
19  4.0000   -6.0000   -10.0000   -2.0000   -46.0000
20  0.0000   -1.0000    6.0000   -4.0000    0.0000
21  0.0000    0.0000   -9.0000   -6.0000   -51.0000
22  0.0000    0.0000    0.0000    1.0000    4.0000
23 Backward substitution:
24 Result of x:
25  1.0000
26  2.0000
27  3.0000
28  4.0000
29 Verification: Ax has to be b
30      Ax          b
31     -46.0000   -46.0000
32      69.0000   69.0000
33      64.0000   64.0000
34     -7.0000   -7.0000

```

図 4.18: ガウスの消去法の実行例

## 4.14 ピボット選択

ガウスの消去法で、ピボット行 ( $m$  行目) の対角要素で割り算をしようとした時、その対角要素がゼロに近いと、オーバーフローをおこして計算が破綻することは容易に想像できる。

そこで、元々の行の順番通りにピボット行にするのではなく、まだゼロになっていない  $i \geq m$  の中から、 $m$  列成分の絶対値が最大の行をピボット行に選ぶ（部分ピボット選択）。これは、連立方程式の順番を入れ替えても、解は変わらないという性質を利用している。

第  $m$  列成分の絶対値が大きいからと言って、実はそれだけでピボット行として適切であるかどうかは分からぬ。各行に定数を掛けても、解は変化しないことを考えると、そのことは理解できる。そこで、 $i \geq m$  の各行に定数を掛けて、その行内の成分の最大値が 1 に成るように規格化する。そのあとで、部分ピボット選択を行う。このことをスケーリングと呼ぶ。

## 4.15 LU 分解

行列  $\hat{A}$  が

$$\hat{A} = \hat{L}\hat{U} \quad (4.86)$$

と書けるとする。ここで、 $\hat{L}$ は下三角行列、 $\hat{U}$ は上三角行列である。 $n = 4$ の場合、

$$\begin{aligned} & \left( \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) \\ &= \left( \begin{array}{cccc} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{array} \right) \left( \begin{array}{cccc} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{array} \right) \quad (4.87) \end{aligned}$$

のように分解できるとする。

$$\hat{A}\vec{x} = \vec{b} \quad (4.88)$$

$$\hat{L}\hat{U}\vec{x} = \vec{b} \quad (4.89)$$

なので、

$$\vec{y} \equiv \hat{U}\vec{x} \quad (4.90)$$

と定義すれば、連立方程式は

$$\hat{L}\vec{y} = \vec{b} \quad (4.91)$$

と書ける。(4.91)式は次のように前進代入で簡単に解ける。

$$y_1 = \frac{b_1}{\alpha_{11}} \quad (4.92)$$

$$y_i = \frac{1}{\alpha_{ii}} \left\{ b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j \right\} \quad (i = 2, 3, \dots, n) \quad (4.93)$$

このように $\vec{y}$ がもとまった後、(4.90)式に後退代入を適用すれば、 $\vec{x}$ を求めることが出来る。

## クラウトのアルゴリズム

ではどのように LU 分解するかが問題である。 $3 \times 3$  の場合の LU 分解の掛け算を実行してみると、

$$\begin{aligned} & \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \\ &= \begin{pmatrix} \alpha_{11}\beta_{11} & \alpha_{11}\beta_{12} & \alpha_{11}\beta_{13} \\ \alpha_{21}\beta_{11} & \alpha_{21}\beta_{12} + \alpha_{22}\beta_{22} & \alpha_{21}\beta_{13} + \alpha_{22}\beta_{23} \\ \alpha_{31}\beta_{11} & \alpha_{31}\beta_{12} + \alpha_{32}\beta_{22} & \alpha_{31}\beta_{13} + \alpha_{32}\beta_{23} + \alpha_{33}\beta_{33} \end{pmatrix} \quad (4.94) \end{aligned}$$

(4.94) 式からも分かる通り、LU 分解は

$$\alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{ij} = a_{ij} \quad (i < j) \quad (4.95)$$

$$\alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{jj} = a_{ij} \quad (i = j) \quad (4.96)$$

$$\alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ij}\beta_{jj} = a_{ij} \quad (i > j) \quad (4.97)$$

という  $n^2$  個の方程式のことである。一方、未知数は  $n^2 + n$  個があるので、まず

$$\alpha_{ii} = 1 \quad (i = 1, \dots, n) \quad (4.98)$$

と置く。つぎに、各  $j = 1, \dots, n$  に対して、

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \quad (i = 1, \dots, j) \quad (4.99)$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left\{ a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right\} \quad (i = j+1, \dots, n) \quad (4.100)$$

を用いて、それぞれの  $i$  について  $\alpha_{ij}$  と  $\beta_{ij}$  を求める。

(4.99) 式では、 $\beta_{ij}$  の値を求めるために、未知の  $\alpha_{ik}, \beta_{kj}$  を用いている様に見えるが、実はすでに求まっている値である。(4.100) 式でも、 $\alpha_{ij}$  の値

を求めるために、未知の  $\alpha_{ik}, \beta_{kj}$  を用いている様にみえるが、実はすでに求まっている値である。

### 課題

つきの連立1次方程式の解をクラウトのアルゴリズムと、前進代入、後退代入で求めるプログラムを作成して  $x_i$  の値を求めよ。

$$\left( \begin{array}{cccc} 4 & -6 & -10 & -2 \\ -6 & 8 & 21 & -1 \\ -10 & 21 & -20 & 23 \\ -2 & -1 & 23 & -18 \end{array} \right) \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) = \left( \begin{array}{c} -46 \\ 69 \\ 64 \\ -7 \end{array} \right) \quad (4.101)$$

## 4.16 逆行列

LU 分解を用いれば、

$$\hat{A}\vec{x} = \vec{b} \quad (4.102)$$

の解を求められることが分かった。

$\hat{A}$  の逆行列  $\hat{A}^{-1}$  は、

$$\hat{A}\hat{A}^{-1} = \hat{I} \quad (4.103)$$

を満たす行列である。ここで、 $\hat{I}$  は対角成分がすべて 1 で、額書く成分がすべて 0 である、単位行列である。

$\vec{b}_i$  ベクトルを第  $i$  番目の成分だけ 1 でそれ以外は 0 であるベクトルだとすると、

$$\hat{I} = [\vec{b}_1, \vec{b}_2, \dots, \vec{b}_n] \quad (4.104)$$

である。したがって、

$$\hat{A}\vec{x}_i = \vec{b}_i \quad (4.105)$$

を解いて、 $\vec{x}_i$  を求めれば、

$$\hat{A}^{-1} = [\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n] \quad (4.106)$$

であることが分かる。

### 課題

下記  $\hat{A}$  行列を LU 分解しなさい。

$$\hat{A} = \begin{pmatrix} 4 & -6 & -10 & -2 \\ -6 & 8 & 21 & -1 \\ -10 & 21 & -20 & 23 \\ -2 & -1 & 23 & -18 \end{pmatrix} \quad (4.107)$$

L 行列、U 行列、y ベクトルをそれぞれ求め、上述の方法で逆行列  $\hat{A}^{-1}$  を求めよ。

## 4.17 行列の固有値と固有ベクトル

行列  $\hat{A}$  に対して、

$$\hat{A}\vec{x} = \lambda\vec{x} \quad (4.108)$$

を満たす値  $\lambda$  のことを  $\hat{A}$  の固有値、また  $\vec{x}$  のことをそれに対する固有ベクトルという。

ここでは詳細な証明などは省略するが、固有値と固有ベクトルを求める方法を簡単に紹介する。詳細は参考文献 [5]などを参照されたい。

固有値は  $\hat{A}$  の次数だけ存在する。たとえば、 $n \times n$  の正方行列に対しては、 $n$  個の固有値が存在する。一般に、固有値は実数となるとは限らない。

絶対値が最大の固有値  $\lambda_1$  を求める方法は簡単である。任意のベクトル  $\vec{v}$  に何度も  $\hat{A}$  を掛けるとそのベクトルは  $\lambda_1$  に対応する固有ベクトル  $\vec{x}_1$  に収束する。

$$\hat{A}^k \vec{v} \rightarrow \vec{x}_1 \quad (k \rightarrow \infty) \quad (4.109)$$

ただし,  $|\lambda_1| > 1$  の時には,  $\vec{x}_1$  の各要素の絶対値がどんどん大きくなるので,  $\vec{x}$  を規格化する必要がある.

$$\vec{x}_1 := \frac{\vec{x}_1}{|\vec{x}_1|} \quad (4.110)$$

この方法をべき乗法という. べき乗法は簡便だが, 絶対値最大の固有値しか求められない.

また一般に大規模行列においては, 固有ベクトルへの収束が速いというわけではない. 高速に絶対値最大の固有値を求めるための Lanczos 法など様々な工夫が存在する.

### 4.17.1 LR 法

$n \times n$  の正方行列  $\hat{A}$  のすべての固有値を同時に求める方法として, LR 法を紹介する.

LU 分解を用いるが, 固有値を求める方法としては歴史的経緯から LR 法と呼ばれることが多い.

$\hat{A}$  を LU 分解する.

$$\hat{A} = \hat{A}_1 = \hat{L}_1 \hat{R}_1 \quad (4.111)$$

ここで,  $\hat{L}_1$  は対角成分が 1 の下三角行列である.  $\hat{R}_1$  は上三角行列である. これらの行列はクラウトのアルゴリズムで求めることができる.

つぎに  $\hat{L}_1$  と  $\hat{R}_1$  を入れ替えて,  $\hat{A}_2$  をつくる.

$$\hat{A}_2 = \hat{R}_1 \hat{L}_1 \quad (4.112)$$

その  $\hat{A}_2$  をまた LU 分解する.

$$\hat{A}_2 = \hat{L}_2 \hat{R}_2 \quad (4.113)$$

これを繰り返すと

$$\hat{A}_{k+1} = \hat{R}_k \hat{L}_k = \hat{L}_{k+1} \hat{R}_{k+1}, \quad k \geq 1 \quad (4.114)$$

と順番に  $\hat{A}_k$  を更新していくことが出来る。

$k$  を大きくしていくと、 $\hat{A}_k$  の対角要素は、 $\hat{A}$  の固有値となる。固有値の絶対値が大きい順番にその対角要素は並ぶ。また  $\hat{A}_k$  行列は上三角行列となる。

このように  $n$  この固有値を一度に求めることができる。

固有ベクトルは逆反復法を用いて求める。

## 4.17.2 逆反復法

固有値  $\lambda$  の近似値  $\sigma$  がわかっているとき、任意のベクトル  $\vec{x}$  に  $(\hat{A} - \sigma \hat{I})^{-1}$  を掛けて  $\vec{x}_1$  をつくると  $\vec{x}_1$  は  $\lambda$  に対する固有ベクトルに近づく。

$\vec{x}_1$  を規格化しさらに  $(\hat{A} - \sigma \hat{I})^{-1}$  を掛けるとさらに固有ベクトルに近づく。これを繰り返すと、固有ベクトルに収束する。

実際は、 $(\hat{A} - \sigma \hat{I})^{-1}$  を求める必要はなく、

$$(\hat{A} - \sigma \hat{I}) \vec{x}_2 = \vec{x}_1 \quad (4.115)$$

を解いて  $\vec{x}_2$  を求めればよい。ここでも LU 分解が活躍する。

LR 法と逆反復法を用いて、固有値と固有ベクトルを求めるプログラム例を下記に示す。

---

```

1 // LU 分解を利用して行列 A の固有値をもとめる。
2 // その後、逆反復法を用いて、固有ベクトルを求める。
3 // 2016 11/24
4
5 #include <stdio.h>
6 #include <math.h>
7
8 #define N_LRM 50 // LR法で固有値もとめる繰り返し回数
9 #define N_Inv 50 // 逆反復法で固有ベクトル求める繰り返し回数
10 #define MtxSiz 10 // 行列の最大サイズ
11 #define DataFile "Mtx.dat" // 行列 A とベクトル b のデータファイル
12
13 int LUD(double a[MtxSiz][MtxSiz], int N);
14 int FwdSub(double b[MtxSiz], int N);
15
16 double alpha[MtxSiz][MtxSiz]; // L 行列
17 double beta[MtxSiz][MtxSiz]; // U 行列
18 double x[MtxSiz]; // 解ベクトル x
19
20 int main(){

```

```

21 // A行列
22 double a[MtxSiz][MtxSiz];
23 double b[MtxSiz]; // bベクトル
24 double aorg[MtxSiz][MtxSiz]; // 元のA行列
25 double borg[MtxSiz]; // 元のbベクトル
26 double eig[MtxSiz];
27 double back[MtxSiz][MtxSiz];
28 double dum;
29 double sum;
30
31 int N;
32 int i,j,k,l;
33
34 FILE* fp;
35
36 fp=fopen(DataFile,"r");
37 fscanf(fp,"%d",&N); // 行列の次元を読み込む
38 printf("N=%d\n",N);
39 printf("Original Matrix:\n");
40 i=1;
41 while(i<=N) {
42     j=1;
43     while(j<=N) {
44         fscanf(fp, "%lf",&dum); // 行列 A の要素を読み込む
45         a[i][j]=dum;
46         aorg[i][j]=dum;
47         printf("%12.4f ",a[i][j]);
48         j++;
49     }
50     fscanf(fp, "%f",&dum); // bベクトルの要素を読み込む
51     b[i]=dum;
52     borg[i]=dum;
53     printf("%12.4f \n",b[i]);
54     i++;
55 }
56 fclose(fp);
57
58 // LR法で固有値を求める
59 l=1 ;
60 while(l<=N_LRM) {
61     // LU分解を実行する関数
62     LUD(a,N);
63
64     // a=UL
65     i=1;
66     while(i<=N) {
67         j=1;
68         while(j<=N) {
69             a[i][j]=0.0;
70             k=1;
71             while(k<=N) {
72                 a[i][j]=a[i][j]+beta[i][k]*alpha[k][j];
73                 k++;
74             }
75             j++;
76         }
77     }
78 }
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96

```

```

77     }
78
79     // a行列の表示
80     printf("l=%d: UL行列\n", l);
81     i=1;
82     while(i<=N) {
83         j=1;
84         while(j<=N) {
85             printf("%12.4f ", a[i][j]);
86             j++;
87         }
88         printf("\n");
89         i++;
90     }
91
92     l++;
93 } // LU法おわり
94
95 // 求めた固有値を eigベクトルに代入
96 i=1;
97 while(i<=N) {
98     eig[i]=a[i][i];
99     i++;
100 }
101
102 // 逆反復法で固有ベクトル xを求める。
103 // 初期ベクトル
104 k=1; // 固有値の番号
105
106 i=1;
107 while(i<=N) {
108     x[i]=0.0;
109     i++;
110 }
111 x[1]=1.0;
112
113 // 逆反復法の繰り返し開始
114 l=1;
115 while(l<=N_INV) {
116     // A-eI 行列をつくる
117     i=1;
118     while(i<=N) {
119         j=1;
120         while(j<=N) {
121             a[i][j]=aorg[i][j];
122             j++;
123         }
124         a[i][i]=a[i][i]-eig[k];
125         i++;
126     }
127
128     // LU分解
129     LUD(a, N);
130
131     i=1;
132     while(i<=N) {

```

```

133         b[i]=x[i];
134         i++;
135     }
136
137     // (A-eI) x=bを解く
138     FwdSub(b,N);
139
140     // xを規格化
141     sum=0.0;
142     i=1;
143     while(i<=N){
144         sum+=x[i]*x[i];
145         i++;
146     }
147     printf("# %15.8eに対する固有ベクトル\n",eig[k]);
148     i=1;
149     while(i<=N){
150         x[i]=x[i]/sqrt(sum);
151         printf("%3d %15.8e\n",i,x[i]);
152         i++;
153     }
154
155     // (Ax)[i]/x[i] と固有値の比較確認
156     printf("# %3d (Ax)[i]/x[i] と固有値 %15.8e の比較確認\n",l,eig[k]);
157     i=1;
158     while(i<=N){
159         b[i]=0.0;
160         j=1;
161         while(j<=N){
162             b[i]+=aorg[i][j]*x[j];
163             j++;
164         }
165         printf("%15.8e \n",b[i]/x[i]);
166         i++;
167     }
168
169     l++;
170 } // 逆反復の繰り返し
171
172 } // main関数の終わり
173
174 int LUD(double a[MtxSiz][MtxSiz], int N){
175     int i,j,k;
176     double sum;
177
178
179     // L行列とU行列をゼロで初期化する。
180     i=1;
181     while(i<=N){
182         j=1;
183         while(j<=N){
184             alpha[i][j]=0.0;
185             beta[i][j]=0.0;
186             j++;
187         }
188     }

```

```
189     i++;
190 }
191
192 // L行列の対角成分を1にする。
193 i=1;
194 while(i<=N) {
195     alpha[i][1]=1.0;
196     i++;
197 }
198
199 // クラウトのアルゴリズムに従って、各成分を求める。
200 j=1;
201 while(j<=N) {
202     i=1;
203     while(i<=j) {
204         k=1; sum=0.0;
205         while(k<=i-1) {
206             sum=sum+alpha[i][k]*beta[k][j];
207             k++;
208         }
209         beta[i][j] = a[i][j] - sum;
210         i++;
211     }
212     i=j+1;
213     while(i<=N) {
214         k=1; sum=0.0;
215         while(k<=j-1) {
216             sum=sum+alpha[i][k]*beta[k][j];
217             k++;
218         }
219         alpha[i][j] = 1.0/beta[j][j]*(a[i][j]-sum);
220         i++;
221     }
222
223     j++;
224 } // クラウトのアルゴリズム終了
225
226 // L行列の出力
227 /*
228 printf("L行列\n");
229 i=1;
230 while(i<=N) {
231     j=1;
232     while(j<=N) {
233         printf("%12.4f ",alpha[i][j]);
234         j++;
235     }
236     printf("\n");
237     i++;
238 }
239 // L行列の出力
240 printf("U行列\n");
241 i=1;
242 while(i<=N) {
243     j=1;
244     while(j<=N) {
```

```

245         printf("%12.4f ",beta[i][j]);
246         j++;
247     }
248     printf("\n");
249     i++;
250 }
251
252 // LUを計算（検算）
253 printf("LU (検算結果)\n");
254 i=1;
255 while(i<=N) {
256     j=1;
257     while(j<=N) {
258         k=1; sum=0.0;
259         while(k<=N) {
260             sum=sum+alpha[i][k]*beta[k][j];
261             k++;
262         }
263         printf("%12.4f ",sum);
264         j++;
265     }
266     printf("\n");
267     i++;
268 }
269 */
270
271 return 0;
272
273 }
274 // LUD 関数終わり
275
276 int FwdSub(double b[MtxSiz], int N){
277
278     int i,j;
279
280     double y[MtxSiz]; // y ベクトル
281     double sum;
282
283     // 前進代入
284     y[1]=b[1]/alpha[1][1];
285     i=2;
286     while(i<=N) {
287         j=1; sum=0.0;
288         while(j<=i-1) {
289             sum=sum+alpha[i][j]*y[j];
290             j++;
291         }
292         y[i]=1.0/alpha[i][i]*(b[i]-sum);
293         i++;
294     }
295     // yベクトルの出力
296     /*
297     printf("yベクトル\n");
298     i=1;
299     while(i<=N) {
300         printf("%12.4f\n",y[i]);
301     }

```

```

301     i++;
302 }
303 */
304
305 // 後退代入
306 x[N]=y[N]/beta[N][N];
307 i=N-1;
308 while(i>=1){
309     j=i+1; sum=0.0;
310     while(j<=N){
311         sum=sum+beta[i][j]*x[j];
312         j++;
313     }
314     x[i]=1.0/beta[i][i]*(y[i]-sum);
315     i--;
316 }
317
318 /*
319 // 解(x)の出力
320 printf("解(x)\n");
321 i=1;
322 while(i<=N){
323     printf("%12.4f\n",x[i]);
324     i++;
325 }
326 */
327
328 return 0;
329
330 }
331 // 前進代入 & 後退代入おわり

```

ここまでで紹介した方法が常に数値的に安定しているとは限らない。場合によっては QR 法 [5] などを使う必要があることに注意されたい。

## 4.18 関数の近似

$y_k = f(x_k)$  を満たす点  $(x_k, y_k)$  の集合があるとき,  $n$  次多項式  $p_n(x)$  で  $f(x)$  を近似する。関数近似には大きく分けて次のふたつの方法がある。

最小二乗法は、すべての点を通るわけではないが、データの概略を表現できる。類似の方法として、他にミニマックス法などもある。

また、補間法を用いると、すべての点を通る関数でデータを表現できる。代表的な補間法としてスプライン補間がある。

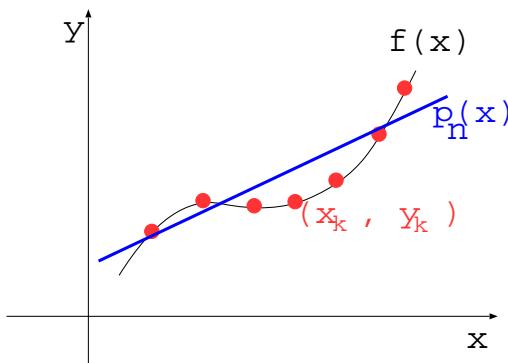


図 4.19: 最小二乗法を用いて、関数  $f(x)$  を  $n$  次多項式  $p_n(x)$  で近似する

#### 4.18.1 メリット

このように多項式を求ることには、おもに次のようなふたつのメリットがある。ひとつめは、最小二乗法を用いてデータを関数で表現すると、記憶データ量が圧縮される。データ  $(x_k, y_k)$  をまるごと記憶するのではなく、多項式として関数表現することで、その係数だけを記憶しておけばよい。たとえば、何千もの観測点データがあるとしても、そのデータが2次関数で近似できるのであれば、パラメータを3つ記憶しておくだけで、そのデータに関する情報は引き出せる。

一方、スプライン補間法ではデータの圧縮効果はない。

2つめは、関数近似には、データに含まれるノイズや誤差などを平滑化する効果がある。平滑化には、移動平均などの様々なフィルターを用いる場合もあるが、多項式関数として表現することにより、比較的少ない計算量で平滑化が行える可能性がある。また、データがセンサー値などの時系列データである場合、時間変数を少し伸ばすことによって簡易的に予測値を求めることが可能である。

これは、最小二乗法とスプライン補間の両者に言える効果である。

おもなメリット

- データの圧縮（最小二乗法）
- データの平滑化，予測

### 4.18.2 最小二乗法

$n$  次多項式  $p_n(x)$  は

$$p_n(x) = c_0 + c_1x + c_2x^2 + \cdots + c_nx^n = \sum_{j=0}^n c_jx^j \quad (4.116)$$

と表される。この多項式は、 $n+1$  個のパラメータ（未定係数  $c_j$ ）を含んでいる。関数近似とは、これらのパラメータを何らかの方法で決める手続きであると言える。

この多項式をデータ集合  $(x_k, y_k)$  に近づけるために、 $n+1$  個のパラメータの関数である、二乗誤差  $Q(\{c_j\})$  を最小化する。

$$Q(c_0, c_1, \dots, c_n) = \sum_{k=1}^m \{p_n(x_k) - y_k\}^2 \quad (4.117)$$

$$= \sum_{k=1}^m \left\{ \sum_{j=0}^n c_j x_k^j - y_k \right\}^2 \quad (4.118)$$

ただし、データは  $m$  個あるとした。つまり、 $k = 1, 2, \dots, m$  である。

$Q(\{c_j\})$  はパラメータ集合  $\{c_j\}$  の関数である。従って、これを  $c_i$  で偏微分して、それぞれがゼロになる条件が、 $Q(\{c_j\})$  が最小になる条件である。

$Q(\{c_j\})$  を  $i = 0, 1, \dots, n$  それぞれに関して  $c_i$  で偏微分する.

$$\begin{aligned}
 \frac{\partial Q}{\partial c_i} &= \frac{\partial}{\partial c_i} \sum_{k=1}^m \left\{ \sum_{j=0}^n c_j x_k^j - y_k \right\}^2 \\
 &= \sum_{k=1}^m \frac{\partial}{\partial c_i} \left\{ \sum_{j=0}^n c_j x_k^j - y_k \right\}^2 \\
 &= \sum_{k=1}^m 2 \left\{ \sum_{j=0}^n c_j x_k^j - y_k \right\} x_k^i \\
 &= 2 \sum_{k=1}^m \sum_{j=0}^n c_j x_k^{i+j} - 2 \sum_{k=1}^m y_k x_k^i \\
 &= 2 \sum_{j=0}^n c_j \left( \sum_{k=1}^m x_k^{i+j} \right) - 2 \sum_{k=1}^m y_k x_k^i
 \end{aligned} \tag{4.119}$$

これが、ゼロにならなければいけないので、

$$\sum_{j=0}^n c_j \left( \sum_{k=1}^m x_k^{i+j} \right) = \sum_{k=1}^m y_k x_k^i \tag{4.120}$$

が成り立たなければならない。一見複雑な式に見えるが、これは  $n+1$  連立 1 次方程式である。

$a_{ij}$  を

$$a_{ij} = \sum_{k=1}^m x_k^{i+j} \tag{4.121}$$

と定義する。また、 $b_i$  を

$$b_i = \sum_{k=1}^m y_k x_k^i \tag{4.122}$$

と定義すると、(4.120) 式は、

$$\begin{aligned}
 \sum_{j=0}^n c_j a_{ij} &= b_i \\
 a_{i0} c_0 + a_{i1} c_1 + \cdots + a_{in} c_n &= b_i
 \end{aligned} \tag{4.123}$$

と書ける。 $i = 0, 1, \dots, n$  に対してそれぞれ式を書くと

$$\begin{aligned} a_{00}c_0 + a_{01}c_1 + \cdots + a_{0n}c_n &= b_0 \\ a_{10}c_0 + a_{11}c_1 + \cdots + a_{1n}c_n &= b_1 \\ &\vdots && \ddots && \vdots \\ a_{n0}c_0 + a_{n1}c_1 + \cdots + a_{nn}c_n &= b_n \end{aligned} \quad (4.124)$$

という連立 1 次方程式となる。

$\hat{A}, \vec{c}, \vec{b}$  をそれぞれ、

$$\hat{A} = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0n} \\ a_{10} & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n0} & a_{n1} & \cdots & a_{nn} \end{pmatrix} \quad (4.125)$$

$$\vec{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix} \quad (4.126)$$

とすれば、(4.124) 式は

$$\hat{A}\vec{c} = \vec{b} \quad (4.127)$$

と表されるので、Gauss の消去法や、LU 分解を用いて  $\vec{c}$  を求めることによって、多項式関数の係数を決めることができる。

最小二乗法の手続きまとめ

1.  $\hat{A}$  行列の各要素を、(4.121) 式で求める。
2.  $\vec{b}$  行列の各要素を、(4.122) 式で求める。
3. 連立 1 次方程式 (4.127) 式を Gauss の消去法や LU 分解で解いて、 $\vec{c}$  を求める。

課題

最小二乗法を用いて図 4.20 に示したデータを近似する多項式を求めよ。1 次, 2 次, 3 次, 4 次, 5 次多項式をそれぞれ用いて近似曲線を求め、それぞれの二乗誤差  $Q(\{c_j\})$  の値を比較せよ。

#	x	y
1	-13.70000	-1.01594
2	-12.10833	-0.58427
3	-10.51667	0.13977
4	-8.92500	1.36046
5	-7.33333	1.73456
6	-5.74167	1.49878
7	-4.15000	1.65006
8	-2.55833	1.77437
9	-0.96667	1.46565
10	0.62500	1.17365
11	2.21667	0.78880
12	3.80833	0.16836
13	5.40000	-0.07467
14	6.99167	-0.19542
15	8.58333	-1.00337
16	10.17500	-1.67055
17	11.76667	-1.34986
18	13.35833	-1.29171
19	14.95000	-2.04990
20	16.54167	-2.02868
21	18.13333	-1.02210
22	19.72500	-0.62330
23	21.31667	-0.56463
24	22.90833	0.57602

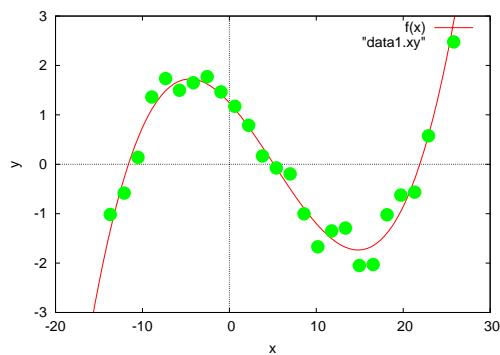


図 4.20: 3 次多項式によって、最小二乗法を用いて 2 次元データの関数近似を行った例。

### 4.18.3 スプライン補間

$n$  個の点  $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$  が与えられた時に、それらの点すべてを通る関数を補間関数という。

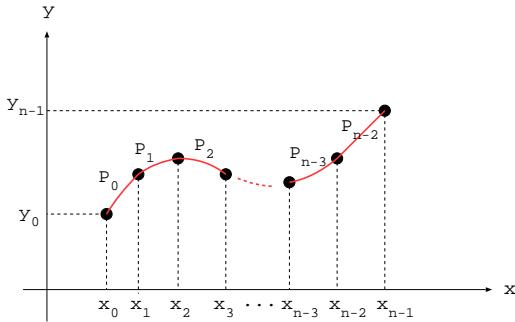


図 4.21: 補間法ではすべての点を通る関数を求める

補間関数を求める方法はいくつか存在するが、ここでは 3 次多項式を用いたスプライン補間について説明する。スプライン (spline) とは自在定規のことである。

$n$  個の点  $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$  から  $n-1$  個の区間  $[x_0, x_1], \dots, [x_{n-2}, x_{n-1}]$  がつくる。スプライン補間では、それぞれの区間  $[x_i, x_{i+1}]$  において、次数が同じ多項式  $P_i(x)$  を用いて補間をおこなう。もちろん、各区間ごとにその多項式内の係数は異なる。

3 次多項式を用いたスプライン補間がよく用いられる（3 次スプライン補間）。それぞれの区間  $[x_i, x_{i+1}]$  における 3 次多項式を

$$P_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad (i = 0, \dots, n-2) \quad (4.128)$$

と書くこととする。従って、 $4(n-1)$  の係数を決める必要がある。

この多項式は両端において  $(x_i, y_i), (x_{i+1}, y_{i+1})$  を通らなければならない

ので,

$$P_i(x_i) = y_i, \quad P_i(x_{i+1}) = y_{i+1} \quad (i = 0, \dots, n-2) \quad (4.129)$$

を満たさなければならない。したがって、 $2(n-1)$  個の係数に関する 1 次式が得られる。具体的には、まず第 1 式から

$$\begin{aligned} P_0(x_0) &= a_0 = y_0 \\ P_1(x_1) &= a_1 = y_1 \\ &\dots \\ P_i(x_i) &= a_i = y_i \\ &\dots \\ P_{n-2}(x_{n-2}) &= a_{n-2} = y_{n-2} \end{aligned} \quad (4.130)$$

という  $n-1$  個の式から、直接  $a_0, \dots, a_{n-2}$  の値がすべて得られる。従つて、あと求めなければならないのは、残りの  $b_i, c_i, d_i$  ( $i = 0, \dots, n-2$ ) の  $3(n-1)$  個である。

つぎに、(4.129) 第 2 式から  $i = 0$  に対して

$$P_0(x_1) = a_0 + b_0(x_1 - x_0) + c_0(x_1 - x_0)^2 + d_0(x_1 - x_0)^3 = y_1$$

すなわち

$$b_0(x_1 - x_0) + c_0(x_1 - x_0)^2 + d_0(x_1 - x_0)^3 = y_1 - a_0 \quad (4.131)$$

である。同様にして、 $i = 1, \dots, k, \dots, n-2$  に対して、

$$\begin{aligned} b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 + d_1(x_2 - x_1)^3 &= y_2 - a_1 \\ &\dots \\ b_k(x_{k+1} - x_k) + c_k(x_{k+1} - x_k)^2 + d_k(x_{k+1} - x_k)^3 &= y_{k+1} - a_k \\ &\dots \\ b_{n-2}(x_{n-1} - x_{n-2}) + c_{n-2}(x_{n-1} - x_{n-2})^2 + d_{n-2}(x_{n-1} - x_{n-2})^3 &= y_{n-1} - a_{n-2} \end{aligned} \quad (4.132)$$

という  $n - 1$  個の係数に関する連立 1 次方程式が得られる。ここで、右辺の  $a_0, \dots, a_{n-2}$  の値はすでに得られていることに注意する必要がある。

つぎに、各繋ぎ目では、スプライン関数は滑らかに繋がらなければならぬことを用いる。すなわち、傾きと曲率が同じでなければならぬ。つまり、 $P_i(x)$  と  $P_{i+1}(x)$  の 1 階微分と、2 階微分がそれぞれの点上で等しくなければならない。

$$\begin{aligned} P'_i(x_{i+1}) &= P'_{i+1}(x_{i+1}), & P''_i(x_{i+1}) &= P''_{i+1}(x_{i+1}) \\ (i &= 0, \dots, n-3) \end{aligned} \quad (4.133)$$

具体的に第 1 式から

$$P'_0(x_1) = P'_1(x_1) \quad (4.134)$$

ここで、

$$P'_i(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \quad (4.135)$$

であるから、

$$b_0 + 2c_0(x_1 - x_0) + 3d_0(x_1 - x_0)^2 - b_1 = 0 \quad (4.136)$$

という係数に関する 1 次方程式が得られる。同様にして、

$$b_1 + 2c_1(x_2 - x_1) + 3d_1(x_2 - x_1)^2 - b_2 = 0 \quad (4.137)$$

…

$$\begin{aligned} b_{n-3} + 2c_{n-3}(x_{n-2} - x_{n-3}) + 3d_{n-3}(x_{n-2} - x_{n-3})^2 - b_{n-2} &= 0 \\ \end{aligned} \quad (4.138)$$

という係数に関する連立 1 次方程式が得られる。(4.136) から (4.138) 式まで合わせて、 $n - 2$  個の連立 1 次方程式が得られた。

つぎに (4.133) 式の第 2 式から  $i = 0$  のとき

$$P''_0(x_1) = P''_1(x_1) \quad (4.139)$$

であるので、ここで

$$P_i''(x) = 2c_i + 6d_i(x - x_i) \quad (4.140)$$

であることを用いると、

$$\begin{aligned} 2c_0 + 6d_0(x_1 - x_0) &= 2c_1 \\ c_0 + 3d_0(x_1 - x_0) &= c_1 \\ c_0 + 3d_0(x_1 - x_0) - c_1 &= 0 \end{aligned} \quad (4.141)$$

同様にして、

$$c_1 + 3d_1(x_2 - x_1) - c_2 = 0 \quad (4.142)$$

...

$$c_{n-3} + 3d_{n-3}(x_{n-2} - x_{n-3}) - c_{n-2} = 0 \quad (4.143)$$

(4.141) から (4.143) 式までの合計  $n - 2$  個の係数に関する連立 1 次方程式が得られる。

ここまで得られた連立方程式の数を合計すると、 $3n - 5$  個である。求めたい係数の数は  $3n - 3$  個であるから、2 個式が足りない。

それらは、両端  $x_0$  と  $x_{n-1}$  におけるスプライン関数の満たす条件によって定める。この条件はいろいろあり得るが、次のように、両端でスプライン関数が曲がっていないという条件を課す場合を、自然スプラインという。

$$P_0''(x_0) = 0, \quad P_{n-2}''(x_{n-1}) = 0 \quad (4.144)$$

すなわち第 1 式から、

$$2c_0 = 0 \quad (4.145)$$

が得られる。つまり、

$$c_0 = 0 \quad (4.146)$$

であるから、求める必要がなく、求める必要があるのは  $3n - 4$  個である。また、第2式から、

$$2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2}) = 0 \quad (4.147)$$

がえられ、連立1次方程式の数は  $3n - 4$  個となるので、必要な係数をすべて LU 分解などを用いて求めることができる。

課題

図 4.22 に示したデータを 3 次スプライン補間により補間曲線をもとめよ。ただし、LU 分解を利用すること。また両端は自然スプラインの条件をもちいること。

---

#	x	y
2	-13.70000	2.84198
3	-9.88000	0.33259
4	-6.06000	-1.50330
5	-2.24000	-1.82648
6	1.58000	-0.69098
7	5.40000	0.62965
8	9.22000	0.98484
9	13.04000	0.53327
10	16.86000	0.15641
11	20.68000	-0.12805

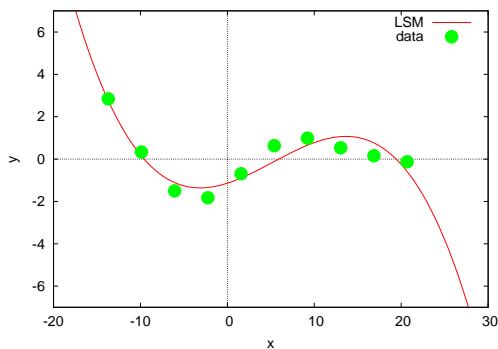


図 4.22: データ例. 図中の曲線は最小二乗法で求めたものである. スプロット曲線と異なり, データ点を通らないことがわかる.

## スプライン補間のアルゴリズム概略

データ点の数は  $n$  である。Spline 曲線を決める係数  $b_i, c_i, d_i (i = 0, \dots, n-2)$  が連立一次方程式の変数ベクトル  $\vec{z}$  にあたる。

$$\vec{z} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ d_0 \\ d_1 \\ \vdots \\ d_{n-2} \end{pmatrix} \quad (4.148)$$

ただし、 $c_0 = 0$  はすでに求まっているので、省略した。したがって、求めたい値は、 $3n - 4$  個あるので、変数ベクトル  $\vec{z}$  の次数も  $3n - 4$  である。係数行列  $\hat{M}$  は  $3n - 4 \times 3n - 4$  である。

$$\hat{M}\vec{z} = \vec{e} \quad (4.149)$$

という行列表現の、 $\hat{M}\vec{e}$  をデータ  $(x_i, y_i)$  から求める。これを解いて、 $\vec{z}$  を求めることにより、スプライン曲線の係数を決定する。

$\hat{M}$  の対角成分にゼロが含まれているとクラウトのアルゴリズムが不安定になる。 $\hat{M}$  の対角成分がゼロにならないように、行の順番を (4.131), (4.132), (4.141), (4.142), (4.143), (4.136), (4.137), (4.138), (4.147) の順序にする。

図 4.23 にアルゴリズムの概略を示す。

```

N = (x,y) データの個数
3N-4 = 行列のサイズ
Function main() is
    | (x,y) データを読み込む
    | a[i]=y[i]; //a[i] の値
    | // M 行列をゼロで初期化する
    | // (4.131) を用いて
    | dx=x[1]-x[0];
    | M[0][0]=dx;
    | M[0][2 * N - 3] = dx3;
    | e[0]=y[1]-a[0];
    | // (4.132) を用いて
    | while 1 ≤ i ≤ N - 2 do
    |   | M[i][i], M[i][i+N-2], M[i][i+2*N-3], e[i] の値を代入
    | end
    | // (4.141) を用いて (c0 = 0)
    | M[N-1][N-1], M[N-1][2*N-3], e[N-1]=0.0
    | // (4.142),(4.143) を用いて
    | while 1 ≤ i ≤ N - 3 do
    |   | M[i+N-1][i+N-2], M[i+N-1][i+N-1], M[i+N-1][i+2*N-3], e[i+N-1]=0.0 の値を代入
    | end
    | // (4.136) を用いて
    | M[2*N-3][2*N-3], M[2*N-3][0], M[2*N-3][1], e[2*N-3]=0.0 の値を代入
    | // (4.137),(4.138) を用いて
    | while 1 ≤ i ≤ N - 3 do
    |   | M[2*N-3+i][2*N-3+i], M[2*N-3+i][i], M[2*N-3+i][i+1], M[2*N-3+i][N+i-2],
    |   | e[2*N-3+i]=0.0 の値を代入
    | end
    | // 自然スプライン (4.147) を用いて
    | i=N-2;
    | M[2*N-3+i][2*N-3+i], M[2*N-3+i][N+i-2], e[2*N-3+i]=0.0
    | // LU 分解, 前進代入, 後退代入 を実行する
end

```

図 4.23: スプライン補間曲線を求めるアルゴリズムの概略

## 4.19 常微分方程式の数値解

常微分方程式の数値的な解法として、ここではルンゲ・クッタ法について説明する。

オイラー法は1次のルンゲ・クッタ法、また、修正オイラー法は2次のルンゲ・クッタ法とみなすこともできる。

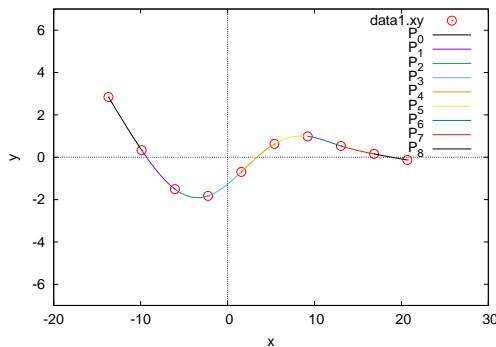


図 4.24: 3 次関数で補間されたスプライン曲線

### 4.19.1 1 階常微分方程式

以下では、変数  $x$  による 1 階微分をダッシュ「,’」を用いて表すことにする。

$$y' = \frac{dy}{dx} \quad (4.150)$$

次の方程式を 1 階常微分方程式という。

$$y' = f(x, y) \quad (4.151)$$

つまり、 $y'$  は  $x$  と  $y$  の関数として表されている場合を考える。この方程式を利用して  $y = f(x)$  の関数の形を求めることを微分方程式を解くという。

どのような微分方程式も必ず解けるというわけではない。例えば下記

の微分方程式は、変数分離法によって解析的に解を求めることができる。

$$y' = \frac{y}{1+x} \quad (4.152)$$

$$\frac{dy}{dx} = \frac{y}{1+x}$$

$$\frac{1}{y} dy = \frac{1}{1+x} dx$$

$$\int \frac{1}{y} dy = \int \frac{1}{1+x} dx$$

$$\log y = \log(1+x) + A$$

$$\log y = \log B(1+x)$$

$$\text{従って } y = B(1+x) \quad (4.153)$$

ともとまる。ただし、 $A, B$  は定数である。この結果から  $y'$  を求めてみると、

$$y' = B \quad (4.154)$$

である。一方、(4.152) 式の右辺から

$$\frac{y}{1+x} = \frac{B(1+x)}{1+x} = B \quad (4.155)$$

となるので、たしかに (4.152) 式が成り立っている。

初期条件として例えば  $y(0) = 1$  とすると、 $B = 1$  となり、

$$y = 1 + x \quad (4.156)$$

という解析的な解が得られる。

## 4.19.2 ルンゲ・クッタ法

修正オイラー法は、2次のルンゲ・クッタ法と呼ばれることもある。以下のルンゲ・クッタ法は、2次のルンゲ・クッタ法と区別するために、4

次のルンゲ・クッタ法と呼ばれる場合もある.

$$x_{i+1} = x_i + h \quad (4.157)$$

$$k_1 = hf(x_i, y_i) \quad (4.158)$$

$$k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \quad (4.159)$$

$$k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \quad (4.160)$$

$$k_4 = hf(x_i + h, y_i + k_3) \quad (4.161)$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (4.162)$$

### 課題 1

以下の微分方程式の解について、オイラー法、修正オイラー法、ルンゲ・クッタ法および解析的な解を比較することで、誤差の蓄積について検討せよ。

$$y' = \frac{2y}{1+x} \quad (4.163)$$

ただし、 $h = 0.5$  と  $h = 0.01$  の 2 つの場合を調べよ。また、初期条件は  $y(0) = 0.5$  とする。

### 課題 2

前節までで扱った、単振り子の振動問題をルンゲ・クッタ法により数値解析し、オイラー法、修正オイラー法およびルンゲ・クッタ法の誤差の蓄積について検討せよ。

## 4.20 微分方程式を数値的に解く

毎日の天気予報や、人が実際に観測できない場所の状態、たとえば地球の内部とか原子炉の中などの状態を予測する方法も基本的には本節で説明する数値解析と同じ方法を用いている。また、最近はさまざま

場所でお目にかかるコンピュータグラフィクス（CG）を使った非常にリアルな物体の動きも、同様な数値解析の結果を利用したものである。

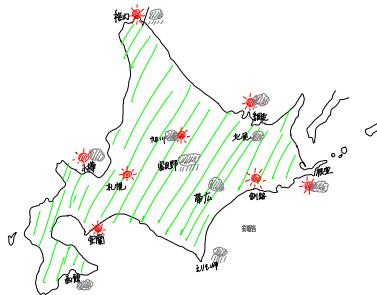


図 4.25: 例えば天気予報も数値解析の結果を利用している。

これらの数値解析は、物理学法則に基づいて行われる。われわれの周りにある物体は、その物理学法則の中でも、古典的な力学法則に従って運動すると考えてよい場合がほとんどである。具体的には運動方程式とよばれる方程式でその運動は記述される。この運動方程式を解いてやって、その解を求めれば、その物体の過去の状態から未来の状態まですべて知ることができる。

ところが、運動方程式は数学的には微分方程式と呼ばれるかたちをしており、多く場合は応用数学の授業で習うように簡単に解を求めることができない。つまり、運動方程式が分かったからといって、すぐに物体の運動を予測することができるとはかぎらない。むしろできない場合がほとんどである。

そこで、大量の数値計算を猛スピードで実行することの得意なコンピューターを使って、その微分方程式から物体の状態を数値的に求める方法（シミュレーション）が考えられた。コンピューターというものの歴史を振り返ってみても、その開発動機はこの物体運動のシミュレーションであった。

微分方程式を数値的に解くためには、様々な高精度な方法が開発され

ている。ここでは、その中でもティラー展開から素直に導かれ、まず、最も基本的で理解しやすいオイラー法について述べる。

オイラー法は、理解しやすい数値解析法であるが、実用的に用いるためには、切り捨て誤差が大きく、すこし物足りない。

そこで、オイラー法の基本的な考え方には、ちょっとした改良を加えた、修正オイラー法についても紹介する。加えられる改良は、わずかな工夫であるし、大規模でテクニカルなものではなく、ごく自然な発想に基づくものであるが、そこから、オイラー法の何千倍もの精度の計算精度を得ることが出来る。

考え方（アルゴリズム）の工夫によって、劇的な効果が得られる良い例であろう。

さらに、さまざまな問題に対して実用的な精度を得るためによく用いられる、ルンゲ・クッタ法について述べる。ルンゲ・クッタ法では要求される計算精度に応じて次数を選ぶことができる。先に述べた修正オイラー法は2次のルンゲ・クッタ法と言うことも出来る。一般的には、より精度の高い4次のルンゲ・クッタ法が用いられることが多い。

## 4.21 振り子運動の数値解析

本節では、一例として、振り子の運動をコンピューターを使った数値解析によって予測するシミュレーションを行う。

振り子の運動なら、基礎的な物理の授業や実験演習などで、よく出てくるモチーフである。その振動数がひもの長さだけに依存しているなど、性質を簡単な式であらわすことができるので、いまさら数値解析するまでもないと感じるかもしれない。

しかし、よく思い出してほしい。その「振り子」は、あくまでも1次元（ひとつの自由度）的な運動で、なおかつ振幅が「非常に小さい」という前提条件の下でのみ考えられていたということを。そのような振り子のことを単振り子とよぶ。本節の中でも、復習の意味もこめて単振り子を最初に扱う。

振り子が3次元的な運動をしたり、振幅が大きくなつたばあいは想定外である。本節では、その想定外を取り除き、任意の振幅の大きさをもち、3次元的な運動をする自由振り子や球面振り子を数値解析のモチーフとする。

### テーマとモチーフについて

テーマというのは、主題のことでの、言葉としてもよく使われるのでその意味するところが理解しやすい。いっぽう、モチーフというのあまり聞きなれないことばかりかもしれない。

美術用語として使ってみる。例えば、りんごの油絵を描くときを思い描いて欲しい。そこで表現したいこと、すなわちテーマは光がつくりだすさまざまな陰影であるとする。そのテーマを具体的に表現するために、モチーフとしてりんごを使っているわけである。

本節で言うと、数値解析（シミュレーション）がテーマであり、振り子がモチーフということになる。

## 4.22 数学的な準備

振り子の運動を数値解析するためには、その運動方程式を考えなければならない。運動方程式とは、数学的には微分方程式のかたちをしている。まず、微分方程式について数学的な説明をする。また、微分方程式を数値的にコンピューターで解析するためには、オイラー法などの手法を利用しなければならない。それらの手法は、数学における泰イラーフィーリング展開を基礎にしているので、泰イラーフィーリング展開に関する説明をする。

### 4.22.1 微分方程式

ある量  $y$  が、変数  $t$  の関数だとする。

$$y = y(t) \tag{4.164}$$

しかし、 $y(t)$  の具体的な関数の形が分からぬときどうすればいいだろうか？まったく手がかりがなければどうしようもないが、 $y$  の  $t$  による 2 階微分  $\ddot{y}$  が<sup>4</sup>  $t, y, \dot{y}$  の関数として分かっている場合を考える。

$$\ddot{y} = f(t, y, \dot{y}) \quad (4.165)$$

この(4.165)式のことを、2階微分方程式と言う。この微分方程式を手がかりにして、 $y(t)$  を数値的に求めていこうというのが、これからやろうとしていることである。振り子の問題の場合、振り子の支点からのふれ角速度  $\theta$  などが、振る舞いを知りたい関数  $y$  に対応する。

### 4.22.2 テイラー展開

微分方程式の解を数値的にもとめるための一つの方法として、後で説明するオイラー法を使う。オイラー法を理解するためには、関数  $y(t)$  の変数  $t$  がわずかに  $h$  だけずれたときに、 $y(t+h)$  が次のように求められることを使う。

$$y(t+h) = y(t) + \dot{y}(t)h + \frac{\ddot{y}(t)}{2!}h^2 + \frac{\ddot{\ddot{y}}(t)}{3!}h^3 + \dots \quad (4.166)$$

この式の右辺のことを、 $y(t+h)$  のテイラー展開と言う。右辺の 2 項目以降が  $t \rightarrow t+h$  という変化による、 $y(t)$  の変化を表している。テイラー展開から分かることは、 $t$  における  $y(t), \dot{y}(t)$  などを知っていれば、 $y(t+h)$  の値を近似的に求めることができるということである。近似的ではなく、完全に求めるためには右辺の … で表した項も含めて無限にたくさんの項を足し合わせなければならない。

## 4.23 オイラー法（微分方程式の数値解法）

テイラー展開(4.166)を利用して、微分方程式(4.165)を数値的に求める。現在では、様々な高精度な方法が編み出されているが、そのなかで

---

<sup>4</sup> $y$  の  $t$  による 2 階微分  $\frac{d^2y}{dt^2}$  を  $\ddot{y}$  と表すことにする。同様に、1 階微分は  $\dot{y}$  で表す。

もっとも基本的なものがオイラー法と呼ばれる方法である。

### 4.23.1 1次近似

もともと  $h \ll 1$  の場合<sup>5</sup> を考えているので、 $h \gg h^2 \gg h^3 \gg \dots$  である。さらに、(4.166) 式の 2 項目以降にある係数の間で

$$\dot{y}(t) \sim \frac{\ddot{y}(t)}{2!} \sim \frac{\ddot{\ddot{y}}(t)}{3!} \sim \dots \quad (4.167)$$

が成り立っていると仮定しよう<sup>6</sup>。すると、

$$\dot{y}(t)h \gg \frac{\ddot{y}(t)}{2!}h^2 \gg \frac{\ddot{\ddot{y}}(t)}{3!}h^3 \gg \dots \quad (4.168)$$

が成り立つ、つまり、(4.166) 式の 3 項目以降を無視して

$$y(t+h) \simeq y(t) + \dot{y}(t)h \quad (4.169)$$

と近似しても妥当だと考えられる<sup>7</sup>。これを  $y(t+h)$  の 1 次近似と呼ぶ。図 4.26 にこの 1 次近似を図示した。 $y(t)$  は、実際は曲線なのであるが、 $t$  から  $t+h$  の間を傾き  $\dot{y}(t)$  の直線で近似したのである。曲線を直線で近似したのであるから、真の値  $y(t+h)$  とその 1 次近似の間には当然誤差が存在していることに注意しなければならない。このことは、あとでさらに詳しく説明する。

### 4.23.2 2階微分方程式から連立 1 階微分方程式へ

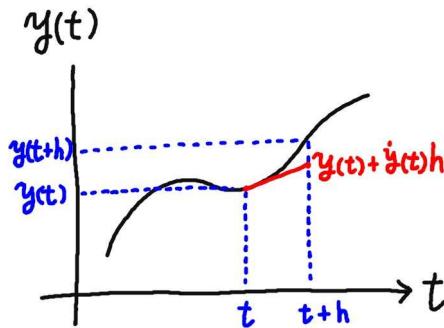
ここまで準備では、2 階の微分方程式 (4.165) を数値的に解くことはまだできない。そのためには、あらたに関数  $y_1(t)$  を

$$y_1(t) \equiv \dot{y}(t) \quad (4.170)$$

<sup>5</sup> $a \ll b$  とは、 $a$  が  $b$  よりも非常に小さいということを意味する。

<sup>6</sup> $a \sim b$  は  $a$  と  $b$  がほぼ等しいということを意味する。ここで「ほぼ等しい」の意味は、「桁違いには違っていない」という程度のかなり緩やかな「ほぼ等しい」である。

<sup>7</sup> $a \simeq b$  は  $a$  を  $b$  で近似できるということを意味する。~とは異なり、ここでは「ほとんど値が等しい」という意味で使うことにする。

図 4.26:  $y(t+h)$  の 1 次近似.

と定義する<sup>8</sup>. これを用いると, (4.165) 式は

$$\dot{y}_1(t) = f(t, y, y_1) \quad (4.171)$$

と書き直すことができる. つまり, 新たに  $y_1$  を導入したことで, 2 階の微分方程式を 1 階の微分方程式 2 つに書き直すことができた. これら 2 つの微分方程式 (4.170), (4.171) 式を合わせて, 連立 1 階微分方程式とよぶ.

### 4.23.3 連立 1 階微分方程式に対するオイラー法

最後に  $y(t), y_1(t)$  に対する連立 1 階微分方程式から, オイラー法の式を求める.  $y(t)$  に関しては, (4.169), (4.170) 式から

$$y(t+h) \simeq y(t) + y_1(t)h \quad (4.172)$$

また,  $y_1(t)$  に関しては, (4.169), (4.171) 式から

$$y_1(t+h) \simeq y_1(t) + f(t, y, y_1)h \quad (4.173)$$

---

<sup>8</sup> $a \equiv b$  は  $a$  を  $b$  で定義するということを意味する.

となる。

$t = t_0$  からスタートすることにする。初期値として、 $y(t_0), y_1(t_0)$  を決めてやれば、(4.172),(4.173) 式を使って、 $t = t_0 + h$  における  $y, y_1$  の値を求めることができる。その  $y, y_1$  を再び (4.172),(4.173) 式の右辺に用いると、 $t + 2h$  における  $y, y_1$  が求められる。これを繰り返していくことによって、 $y, y_1$  の時間発展が求められていく。このアルゴリズムの概略を、PAD 図<sup>9</sup> として、図 4.27 に示した。

$y, y_1$  の値を順番に更新していくだけの単純なアルゴリズムであるが、一つだけ注意を要する点がある。この図中の = の記号は、数式の等号の意味ではなく、C 言語などにおける代入を意味する。したがって、 $y = y + y_1 h$  の行が実行されると、 $y$  の値は、右辺の値によって更新されてしまう。 $y_1 = y_1 + f(t, y, y_1)h$  を実行する際に  $y$  の値を使うのであるが、更新される前の  $y$  を使わなければならない。そこで、一旦  $y_b = y$  を実行して、 $y$  の値を  $y_b$  に退避しておいてから、 $y$  の値を更新する。 $y_1$  の値を更新するときには、 $y_b$  の値を使うことで、問題を回避できる。

## 4.24 单振り子の数値解析

2 階微分方程式はオイラー法を使って数値的に解けることがわかった。あとは、振り子の運動方程式（2 階微分方程式）がわかればよい。つまり、(4.165) 式のなかの  $f(t, y, \dot{y})$  が具体的にどのような表式になるのか求めなければならない。

### 4.24.1 球座標

振り子の運動を考える上では、3 次元空間を  $x$ - 軸、 $y$ - 軸、 $z$ - 軸の座標で考えるよりも、図 4.28 で示した、球座標を用いた方が、便利である。 $\{x, y, z\}$  と  $\{r, \theta, \varphi\}$  の間には

---

<sup>9</sup>Problem Analysis Diagram : フローチャートなどと同様にプログラムのアルゴリズムを図に表す方法

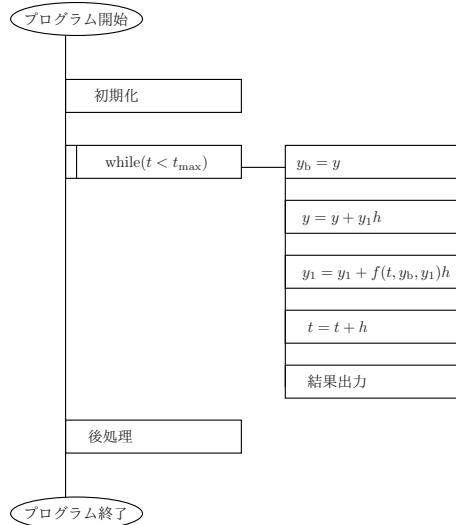


図 4.27: オイラー法のアルゴリズム概略を表す PAD 図。

$$x = r \sin \theta \cos \varphi \quad (4.174)$$

$$y = r \sin \theta \sin \varphi \quad (4.175)$$

$$z = r \cos \theta \quad (4.176)$$

という関係がある。 $\{r, \theta, \varphi\}$  を用いる方が、微分方程式を導くのが容易である。いっぽう、その解を表示するのには、 $\{x, y, z\}$  を用いた方が便利である。

単振り子は重力によって下方を向いているので、 $\theta = \pi$  の周りに振動が起こる。以下のように、 $\pi$  からのずれを  $\theta'$  で表すことにする（図 4.29 参照）。

$$\theta' \equiv \theta - \pi \quad (4.177)$$

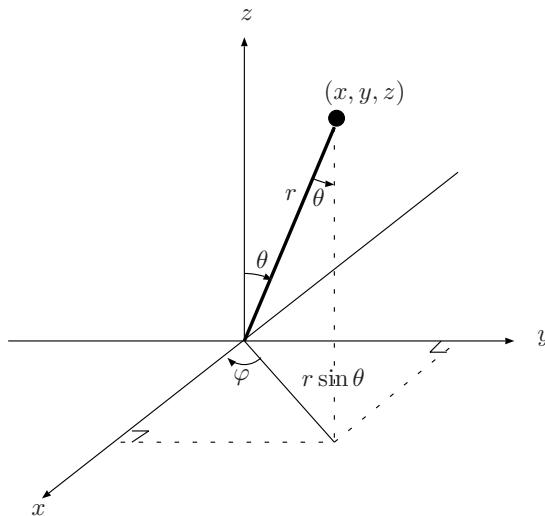


図 4.28: 球座標

#### 4.24.2 单振り子の運動方程式

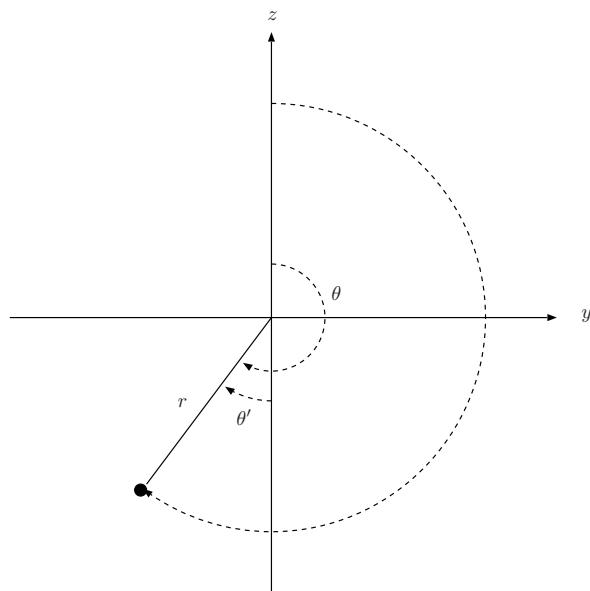
まず、伸び縮みしないひもにつけられた質点が  $yz$ - 平面内だけで振動している振り子（单振り子）を考えよう。時間を  $t$  で表すとすると、 $\theta'$  が変数  $t$  の関数である。

$$\theta' = \theta'(t) \quad (4.178)$$

おもりの質量を  $m$ 、重力加速度を  $g$  とすると、单振り子には図 4.30 に示されたような力が働く。ここで、 $f_T$  はひもがおもりを支点ほうこうに引っ張る張力である。いま、ひもは伸びたり縮んだりしないので、ひも方向には運動しないように、張力はちょうど  $mg \cos \theta'$  と釣り合う。

ひもが伸び縮みしないので、单振り子は円運動する。その円の接線方向の加速度は  $r\ddot{\theta} = r\ddot{\theta}'$  である。ここで、 $\theta' = \theta - \pi$  なので、 $\ddot{\theta} = \ddot{\theta}'$  を使った。

力学の第 2 法則によると、(質量) × (加速度) = 力が成り立たなけれ

図 4.29: 単振り子は  $\theta = \pi$  の周りで振動する

ばいけないので,

$$m \times r\ddot{\theta}' = -mg \sin \theta' \quad (4.179)$$

という 2 階微分方程式が導かれる.

$$y = \theta' \quad (4.180)$$

と考えれば,

$$\ddot{y} = -\frac{g}{r} \sin y \quad (4.181)$$

であるから,

$$f(t, y, y_1) = -\frac{g}{r} \sin y \quad (4.182)$$

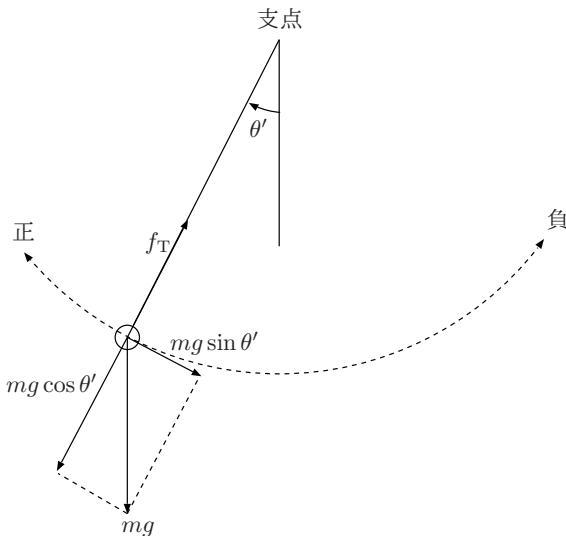


図 4.30: 振り子に働く力

となることが、わかる。 $g, r$  は時間  $t$  に依存しない定数であるので、右辺は  $y$  だけの関数である。

$\theta' \ll 1$  であれば、この微分方程式の解析的な近似解を求めることができる。この場合を次節であつかう。そうでない場合、すなわち  $\theta' \sim 1$  のばあい解析的な近似解を求めることが困難であるので、オイラー法などによって数値的な解を求ることになる。

### 4.24.3 解析的な近似解が得られる場合 ( $\theta' \ll 1$ )

$\theta' \ll 1$  の場合すなわち、振り子の振れ幅がとても小さいとき、

$$\sin \theta' \approx \theta' \quad (4.183)$$

と近似できる。これを、(4.179) 式に用いると

$$\ddot{\theta}' = -\frac{g}{r} \theta' \quad (4.184)$$

(4.179) と異なり、この微分方程式に関しては、解析的な解を簡単に見つけることができる。

$$\theta'(t) = A \sin\left(\sqrt{\frac{g}{r}}t\right) + B \cos\left(\sqrt{\frac{g}{r}}t\right) \quad (4.185)$$

係数  $A, B$  は初期状態によって決まる。いま、 $t = 0$  で、 $\theta'(0) = \theta'_I$ ,  $\dot{\theta}'(0) = 0$  とすると、単振り子の運動は、

$$\theta'(t) = \theta'_I \cos\left(\sqrt{\frac{g}{r}}t\right) \quad (4.186)$$

で記述される。

## 4.25 数値誤差の蓄積とそれを低減させる方法

テイラー展開およびオイラー法の節で説明した通り、展開の次数が高い項は、値が小さいと仮定して切り捨てている（無視している）。したがって、数値解析を進めるにしたがって、それによる誤差が当然蓄積する。その誤差の蓄積はゼロにはできないが、その誤差を小さく抑える修正オイラー法について説明する。

### 4.25.1 オイラー法による打ち切り誤差

オイラー法に限らず、数値解析法では近似を繰り返すことによる打ち切り誤差が存在する。ここでは、その誤差が、時間の微少増加量  $h$  にどのように依存しているのかをみる。

まず、アルゴリズムの計算量を大まかに評価したり、誤差項の評価に使われるランダウの  $O$  記号を用いる。 $O(g(x))$  は  $cg(x)$  未満の量を表す。ここで、 $c$  は定数である。つまり、 $O(g(x))$  は定数  $c$  を除いてほぼ  $g(x)$  に等しい量を意味する。

オイラー法では、テイラー展開

$$y(t+h) = y(t) + \dot{y}(t)h + \frac{\ddot{y}(t)}{2!}h^2 + \frac{\dddot{y}(t)}{3!}h^3 + \dots \quad (4.187)$$

の3項目以降を無視した。したがって、1ステップ計算するたびに  $O(h^2)$  の誤差が  $y$  の値に対して生まれる。時間  $t$  を 0 から始めて、 $T$  まで計算すると、すなわち、

$$0 \leq t \leq T \quad (4.188)$$

の場合、全ステップ数は  $\frac{T}{h}$  であるから、この間に蓄積する打ち切り誤差は

$$O(h^2) \times \frac{T}{h} = T O(h) \quad (4.189)$$

すなわち、オイラー法の打ち切り誤差は、 $h$  に比例して大きくなる。また、それは計算時間  $T$  にも比例して大きくなる。

## 4.25.2 修正オイラー法

オイラー法では、テイラー展開における  $O(h)$  の項までを考慮し、 $O(h^2)$  より高次の項は無視した。図 4.26 に示した通り、この近似法は  $t$  における傾き  $\dot{y}(t)$  を利用して曲線を直線で近似したことに対応している。そのため、打ち切り誤差  $O(h^2)$  が生じていた。

そこで、 $\dot{y}(t)$  のかわりに  $\dot{y}(t)$  と  $\dot{y}(t+h)$  の平均を用いることを考えよう。すなわち、

$$y(t+h) \approx y(t) + \frac{1}{2}(\dot{y}(t) + \dot{y}(t+h))h \quad (4.190)$$

$\dot{y}(t)$  と  $\dot{y}_1(t)$  はそれぞれ、

$$\dot{y}(t) = y_1(t) \quad (4.191)$$

$$\dot{y}_1(t) = -a \sin y(t) \quad (4.192)$$

によって求められる。ただし、簡単のために、

$$a \equiv \frac{g}{r} \quad (4.193)$$

と定義した.

ところが,  $\dot{y}(t+h), \dot{y}_1(t+h)$  はまだ  $y(t+h), y_1(t+h)$  が求められていないので, 計算することができない. そこで,  $y(t+h), y_1(t+h)$  を先にオイラー法で求めておいて, それを修正オイラー法に用いることにする. オイラー法で求めた量と, 修正オイラー法で求めた量を区別するために, オイラー法で求めた  $y, y_1$  を  $\bar{y}, \bar{y}_1$  と表すことにする.

$$\begin{aligned}\bar{y}(t+h) &\simeq y(t) + \dot{y}(t)h \\ &= y(t) + y_1(t)h\end{aligned}\quad (4.194)$$

$$\begin{aligned}\bar{y}_1(t+h) &\simeq y_1(t) + \dot{y}_1(t)h \\ &= y_1(t) - a(\sin y(t))h\end{aligned}\quad (4.195)$$

これらをもちいて,  $\bar{y}(t+h), \bar{y}_1(t+h)$  は

$$\bar{y}(t+h) = \bar{y}_1(t+h) \quad (4.196)$$

$$\bar{y}_1(t+h) = -a \sin \bar{y}(t+h) \quad (4.197)$$

と求められる. これらを, (4.190) 式の,  $\dot{y}(t+h), \dot{y}_1(t+h)$  の代わりに使うと,

$$\begin{aligned}y(t+h) &\simeq y(t) + \frac{h}{2}(\dot{y}(t) + \bar{y}(t+h)) \\ &= y(t) + hy_1(t) - \frac{ah^2}{2} \sin y(t)\end{aligned}\quad (4.198)$$

$$\begin{aligned}y_1(t+h) &\simeq y_1(t) + \frac{h}{2}(\dot{y}_1(t) + \bar{y}_1(t+h)) \\ &= y_1(t) - \frac{ah}{2}\{\sin y(t) + \sin(y(t) + hy_1(t))\}\end{aligned}\quad (4.199)$$

となる. この式を見れば分かるとおり, 右辺は  $O(h^2)$  の項を含んでいる. すなわち, 1 ステップ計算するごとに  $O(h^3)$  の誤差が蓄積され,  $t = 0 \sim T$  の間に蓄積する誤差は,

$$\frac{T}{h} O(h^3) = T O(h^2) \quad (4.200)$$

となる. 仮に  $h = 10^{-3}$  と選んだとすると, 修正オイラー法による打ち切り誤差は, オイラー法の  $\frac{1}{1000}$  となる.

### 4.25.3 自由振り子の数値解析

ここまででは、 $\theta$  方向だけに自由度を持つ振り子、すなわち单振り子を考えてきたが、 $\theta$  に加えて、 $\varphi$  方向にも運動できる振り子のことを球面振り子とよぶ。さらに、 $r$  方向にも自由に運動できる振り子のことを自由振り子とよぶ。ここからは、自由度として  $r, \varphi, \theta$  の三自由度をもつ自由振り子に対する微分方程式を解析力学 [7] におけるラグランジアンを用いて求め、 $r$  方向の運動はしていない、すなわち  $\dot{r} = \ddot{r} = 0$  とおくことによって、球面振り子に関する微分方程式を求める。

#### 自由振り子

解析力学におけるラグランジアン  $L$  とそれが満たす運動方程式（微分方程式）はそれぞれ以下のように表される。

$$L = T - U \quad (4.201)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} = \frac{\partial L}{\partial q} \quad (4.202)$$

ここで、 $T$  は運動エネルギー、 $U$  はポテンシャルエネルギーを表す。また、 $q$  は自由度変数  $r, \theta, \varphi$  のうちいずれかを表し、 $\dot{q}$  はその時間微分を表す。

自由振り子の運動エネルギー  $T$  は

$$T = \frac{1}{2} \{ \dot{r}^2 + (r\dot{\theta}')^2 + (r\dot{\varphi} \sin \theta')^2 \} m \quad (4.203)$$

である。いっぽう、ポテンシャルエネルギー  $U$  は

$$U = -mgr \cos \theta' + \frac{1}{2} k(r - r_0)^2 \quad (4.204)$$

である。ここで、 $k$  はバネ定数、 $r_0$  はバネの自然長（重りがついていないときの長さ）である。これらの (4.203) 式、(4.204) 式を用いると、

$$\frac{\partial L}{\partial \dot{r}} = \frac{\partial T}{\partial \dot{r}} - \frac{\partial U}{\partial r}, \quad \frac{d}{dt} \frac{\partial L}{\partial \dot{r}} = m\ddot{r} \quad (4.205)$$

$$\frac{\partial L}{\partial r} = mr\dot{\theta}'^2 + mr\dot{\varphi}^2 \sin^2 \theta' + mg \cos \theta' - k(r - r_0) \quad (4.206)$$

となる。したがって  $r$  方向の運動方程式は(4.202)式から、

$$m\ddot{r} = mr\dot{\theta}'^2 + mr\dot{\varphi}^2 \sin^2 \theta' + mg \cos \theta' - k(r - r_0) \quad (4.207)$$

が得られる。第1項が  $\theta$  方向の回転による遠心力、第2項が  $\varphi$  方向の回転による遠心力、第3項が重力、最後の第4項がバネによる力である。

つぎに、 $\theta'$  に関して同様に

$$\frac{\partial L}{\partial \dot{\theta}'} = mr^2\dot{\theta}', \quad \frac{d}{dt} \frac{\partial L}{\partial \theta'} = m(2r\dot{r}\dot{\theta}' + r^2\ddot{\theta}') \quad (4.208)$$

$$\frac{\partial L}{\partial \theta'} = mr^2\dot{\varphi}^2 \sin \theta' \cos \theta' - mgr \sin \theta' \quad (4.209)$$

となるので、 $\theta$  方向の運動方程式は

$$m(2r\dot{r}\dot{\theta}' + r^2\ddot{\theta}') = mr^2\dot{\varphi}^2 \sin \theta' \cos \theta' - mgr \sin \theta' \quad (4.210)$$

である。両辺を  $mr$  で割ると、

$$2\dot{r}\dot{\theta}' + r\ddot{\theta}' = r\dot{\varphi}^2 \sin \theta' \cos \theta' - g \sin \theta' \quad (4.211)$$

という微分方程式が得られる。

最後に、 $\varphi$  に関して同様に、

$$\frac{\partial L}{\partial \dot{\varphi}} = mr^2\dot{\varphi} \sin^2 \theta' \quad (4.212)$$

$$\frac{d}{dt} \frac{\partial L}{\partial \varphi} = m(2r\dot{r}\dot{\varphi} \sin^2 \theta' + 2r^2\dot{\theta}'\dot{\varphi} \sin \theta' \cos \theta' + r^2\ddot{\varphi} \sin^2 \theta') \quad (4.213)$$

$$\frac{\partial L}{\partial \varphi} = 0 \quad (4.214)$$

であるから、 $\varphi$  方向の運動方程式は

$$m(2r\dot{r}\dot{\varphi} \sin^2 \theta' + 2r^2\dot{\theta}'\dot{\varphi} \sin \theta' \cos \theta' + r^2\ddot{\varphi} \sin^2 \theta') = 0 \quad (4.215)$$

となり、この式から、

$$2\dot{r}\dot{\varphi} \sin \theta' + 2r\dot{\theta}'\dot{\varphi} \cos \theta' + r\ddot{\varphi} \sin \theta' = 0 \quad (4.216)$$

という微分方程式が得られる。

$r, \theta, \varphi$  の 3 つの変数について得られた微分方程式をまとめると,

$$m\ddot{r} = mr\dot{\theta}'^2 + mr\dot{\varphi}^2 \sin^2 \theta' + mg \cos \theta' - k(r - r_0) \quad (4.217)$$

$$2\dot{r}\dot{\theta}' + r\ddot{\theta}' = r\dot{\varphi}^2 \sin \theta' \cos \theta' - g \sin \theta' \quad (4.218)$$

$$2\dot{r}\dot{\varphi} \sin \theta' + 2r\dot{\theta}'\dot{\varphi} \cos \theta' + r\ddot{\varphi} \sin \theta' = 0 \quad (4.219)$$

となる。

つぎに、これらの微分方程式から単振り子の場合と同様に修正オイラー法の式を得るために、

$$\begin{aligned} y_0 &\equiv \theta', & y_1 &\equiv \dot{\theta}' \\ y_2 &\equiv r - r_0, & y_3 &\equiv \dot{r} \\ y_4 &\equiv \varphi, & y_5 &\equiv \dot{\varphi} \end{aligned} \quad (4.220)$$

と置き換えると、(4.217) 式は、

$$\begin{aligned} m\dot{y}_3 &= m(r_0 + y_2)y_1^2 + m(r_0 + y_2)y_5^2 \sin^2 y_0 + mg \cos y_0 \\ &\quad - ky_2 \end{aligned} \quad (4.221)$$

となる。ここで、

$$\begin{aligned} f_3(y_0, \dots, y_5) &\equiv (r_0 + y_2)y_1^2 + (r_0 + y_2)y_5^2 \sin^2 y_0 + g \cos y_0 \\ &\quad - \frac{k}{m}y_2 \end{aligned} \quad (4.222)$$

と定義すると、

$$\dot{y}_3 = f_3(y_0, \dots, y_5) \quad (4.223)$$

と表すことができる。つぎに、(4.218) 式から

$$2y_3y_1 + (r_0 + y_2)\dot{y}_1 = (r_0 + y_2)y_5^2 \sin y_0 \cos y_0 - g \sin y_0 \quad (4.224)$$

となる。ここで、

$$\begin{aligned} f_1(y_0, \dots, y_5) &\equiv \frac{1}{r_0 + y_2} \{(r_0 + y_2)y_5^2 \sin y_0 \cos y_0 - g \sin y_0 \\ &\quad - 2y_3y_1\} \end{aligned} \quad (4.225)$$

と定義すると,

$$\dot{y}_1 = f_1(y_0, \dots, y_5) \quad (4.226)$$

と表すことができる。さらに、(4.219) 式から、

$$2y_3y_5 \sin y_0 + 2(r_0 + y_2)y_1y_5 \cos y_0 + (r_0 + y_2)\dot{y}_5 \sin y_0 = 0 \quad (4.227)$$

なので、

$$\begin{aligned} f_5(y_0, \dots, y_5) &\equiv \frac{-2}{(r_0 + y_2) \sin y_0} \\ &\times \{y_3y_5 \sin y_0 + (r_0 + y_2)y_1y_5 \cos y_0\} \end{aligned} \quad (4.228)$$

と定義すると、

$$\dot{y}_5 = f_5(y_0, \dots, y_5) \quad (4.229)$$

という式が得られる。

これらの式を用いて、修正オイラー法の式を求める。まず、オイラー法を用いて  $\bar{y}_i(t+h)$  を求める。

$$\bar{y}_0(t+h) = y_0(t) + h\dot{y}_0(t) = y_0(t) + hy_1(t) \quad (4.230)$$

$$\bar{y}_1(t+h) = y_1(t) + h\dot{y}_1(t) = y_1(t) + hf_1 \quad (4.231)$$

$$\bar{y}_2(t+h) = y_2(t) + h\dot{y}_2(t) = y_2(t) + hy_3(t) \quad (4.232)$$

$$\bar{y}_3(t+h) = y_3(t) + h\dot{y}_3(t) = y_3(t) + hf_3 \quad (4.233)$$

$$\bar{y}_4(t+h) = y_4(t) + h\dot{y}_4(t) = y_4(t) + hy_5(t) \quad (4.234)$$

$$\bar{y}_5(t+h) = y_5(t) + h\dot{y}_5(t) = y_5(t) + hf_5 \quad (4.235)$$

つぎに、 $\bar{y}_i(t+h)$  を求める。

$$\bar{y}_0(t+h) = \bar{y}_1(t+h) \quad (4.236)$$

$$\bar{y}_1(t+h) = f_1(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h)) \quad (4.237)$$

$$\bar{y}_2(t+h) = \bar{y}_3(t+h) \quad (4.238)$$

$$\bar{y}_3(t+h) = f_3(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h)) \quad (4.239)$$

$$\bar{y}_4(t+h) = \bar{y}_5(t+h) \quad (4.240)$$

$$\bar{y}_5(t+h) = f_5(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h)) \quad (4.241)$$

$$(4.242)$$

最後に、修正オイラー法の(4.190)式を用いると、

$$y_0(t+h) = y_0(t) + \frac{h}{2}\{y_1(t) + \bar{y}_1(t+h)\} \quad (4.243)$$

$$y_1(t+h) = y_1(t) + \frac{h}{2}\{f_1(y_0(t), \dots, y_5(t)) + f_1(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h))\} \quad (4.244)$$

$$y_2(t+h) = y_2(t) + \frac{h}{2}\{y_3(t) + \bar{y}_3(t+h)\} \quad (4.245)$$

$$y_3(t+h) = y_3(t) + \frac{h}{2}\{f_3(y_0(t), \dots, y_5(t)) + f_3(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h))\} \quad (4.246)$$

$$y_4(t+h) = y_4(t) + \frac{h}{2}\{y_5(t) + \bar{y}_5(t+h)\} \quad (4.247)$$

$$y_5(t+h) = y_5(t) + \frac{h}{2}\{f_5(y_0(t), \dots, y_5(t)) + f_5(\bar{y}_0(t+h), \dots, \bar{y}_5(t+h))\} \quad (4.248)$$

### 球面振り子

$r = r_0, \dot{r} = 0$  の場合を考えれば良いので、

$$y_2 = y_3 = 0 \quad (4.249)$$

である。また、 $\ddot{r} = 0$  なので、

$$\dot{y}_3 = f_3 = 0 \quad (4.250)$$

である。また、 $\theta$  と  $\varphi$  に関して、(4.218) 式から、

$$r_0\ddot{\theta}' = r_0\dot{\varphi}^2 \sin\theta' \cos\theta' - g \sin\theta' \quad (4.251)$$

また、(4.219) 式から、

$$\begin{aligned} 2r_0\dot{\theta}'\dot{\varphi} \cos\theta' + r_0\dot{\varphi} \sin\theta' &= 0 \\ \ddot{\varphi} &= -2 \frac{\cos\theta'}{\sin\theta'} \dot{\theta}'\dot{\varphi} \end{aligned} \quad (4.252)$$

である。これらの式から、

$$f_1(y_0, \dots, y_5) \equiv \frac{1}{r_0} \{r_0 y_5^2 \sin y_0 \cos y_0 - g \sin y_0\} \quad (4.253)$$

$$f_3(y_0, \dots, y_5) \equiv 0 \quad (4.254)$$

$$f_5(y_0, \dots, y_5) \equiv \frac{-2}{\sin y_0} y_1 y_5 \cos y_0 \quad (4.255)$$

を修正オイラー法に用いれば、球面振り子の運動を数値解析できる。

しくみが単純にみえる振り子ひとつとっても、その運動を正確に解析するためにはいろいろと、複雑な道具立てが必要になる。微分方程式、テーラー展開の数学的な知識からはじまって、力学、解析力学の物理学法則や手法が必要であった。

さらに数値解析を行うためには、オイラー法が必要で、打ち切り誤差をできるだけ小さく抑えるために、修正オイラー法を使った。

もちろん、オイラー法や修正オイラー法を実際にコンピューターで実行するためには、アルゴリズムの構成やプログラミング言語の文法やコンパイル法など知識が必要である。

これらすべての事柄を組み合わせて理解し活用できてはじめて、振り子の運動が数値解析可能となる。

# 第5章 時間遅れのある反応行動

回転翼飛行ロボットは回転運動に関して3つの自由度を持っているが、その中の一つだけを取り出して着目する。その運動方程式を求め、ホバリングを目標姿勢とするPI制御について説明する。

小型の飛行ロボットの姿勢変化は比較的短い時間に変化する。そのため、信号伝達やノイズ除去演算による時間遅れが相対的に無視できない[8, 9, 10]。また、同様の理由で、制御の離散性も影響が大きいと考えられる。

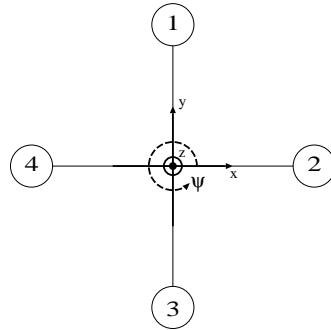
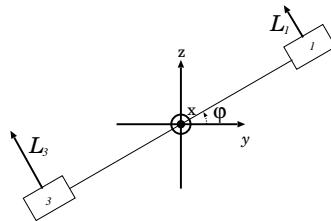
さらに、ローター角速度の緩和時間も、時間遅れや離散時間間隔と同程度の大きさを持つので、これらがすべて考慮されていなければならぬと考えられる。

また、制御に必要なパラメーターの決定には、飛行シミュレーターの存在が不可欠なので、すべてを明示的な数式によって表現しておくことが重要と考える。

## 5.1 回転運動方程式

回転翼飛行ロボットは、図5.1に示したように、4つのローター( $k = 1, 2, 3, 4$ )を持つ。ローター1,3を結ぶ軸を $y$ -軸、ローター2,4を結ぶ軸を $x$ -軸にとる。 $z$ -軸は $xy$ 平面に垂直にとる。いま、飛行ロボットの1軸( $x$ -軸とする)の周りだけで回転運動を考える(図5.2参照)。飛行ロボットの $x$ -軸周りの慣性モーメントを $I_x$ 、また回転角度を $\varphi$ とすると回転運動方程式は

$$I_x \ddot{\varphi} = \tau_x \quad (5.1)$$

図 5.1: 4 回転翼飛行ロボット上の  $xy$  座標軸およびローター ( $k = 1, 2, 3, 4$ ) の配置.図 5.2:  $x$ -軸周りの飛行ロボットの回転角度  $\varphi$  とローター 1,3 による揚力  $L_1, L_3$ 

である。ここで、 $\tau_x$  は 2 つのローターによる揚力差から生じる、 $x$ -軸周りのトルクである。すなわち、 $L_k$  を  $k$  番目のローターによって生じた揚力とすると、

$$\tau_x = r(L_1 - L_3) \quad (5.2)$$

である。ここで、 $r$  は飛行ロボットの中心から、ローターまでの距離である。揚力  $L_k$  は、一般にローター角速度  $\omega_k$  の 2 乗に比例する。すなわち、

$$L_k = a_L \omega_k^2 \quad (5.3)$$

である。図 5.3 に、実際に測定したローター角速度と、その揚力の関係をプロットした。

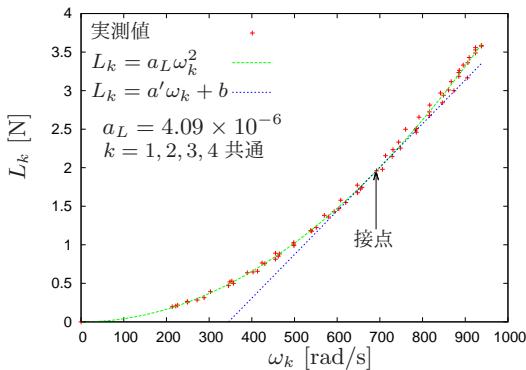


図 5.3: 揚力  $L_k$  は、ローター角速度  $\omega_k$  の 2 次関数である。ローターの番号  $k = 1, 2, 3, 4$  に対して共通。

## 5.2 反応行動としての感覚運動写像

反応行動とは、単純化して言えば、環境情報（の一部）を感知して、それに応じて行動を起こすことを指す。それに伴って環境が変化し、それをまたロボットが感知して…、という風に、ロボットと環境が相互作用のループを続けていく（図 2.1 参照）。

ここで少し誤解が生じるかもしれない。小さなロボットがちょっと行動したぐらいで「環境」が変化するであろうか？という疑問である。例えば、光を好む反応行動を起こすロボットがあるとする。ここで光だけを「環境」とすると、ロボットが移動しても「環境」は変化しない。そうではなくて、ロボットと「環境」全体を含めて環境とみなす。すると、ロボットが光の方に近づく行動を取れば、ロボットが感知する光は変化（増加）するはずである。

飛行ロボットで言えば、ローター回転速度を変化させて、姿勢や位置が変化すれば、ロボット自身が感知する角度や角速度が変化する。それらの情報まで含めて環境情報と考えるわけである。

反応行動をとるための構成要素は、大きく分けると 3 つある。環境を感知する部分、感知した情報に応じて行動を決定する部分、そして行動

を起こす部分の3つである。飛行ロボットにおいて、それぞれの役割を担う要素を、図5.4に示した。

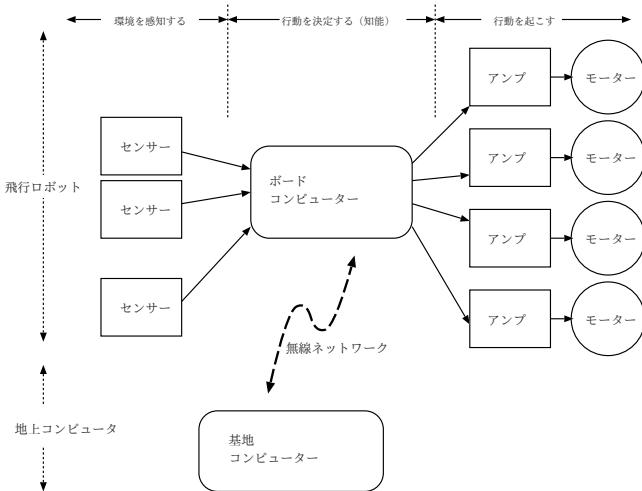


図5.4: 飛行ロボットのシステム構成概略図

環境を感知する部分は、センサーである。人間で言えば、五感の感覚器に相当する部分である。

行動の決定は、ボードコンピューターが主に担う。ただし、生物と異なる点は、その部分が飛行ロボットの身体内部にある必要性は必ずしもないということである。ボードコンピューターは、無線ネットワークを通じて、基地コンピューターと情報のやり取りが可能なので、行動の決定は基地コンピューターが行なって、その結果を行動を起こす部分に渡すこととも可能である。

反応行動の範囲であれば、行動を決定するための手続きに要する計算量が比較的少ないことが予想されるので、この必要はあまりないかもしれない。しかし、計画行動、適応行動といったより複雑な知能が要求される判断においては、ボードコンピューターの計算処理能力では間に合わないかもしれない。基地コンピューターの計算処理能力を動員する必

要があるかもしれないし、またそれが可能である。この点は、生物の知脳と大きく異なり、ロボット知能の可能性を感じる部分の一つである。

行動を起こす部分は、アンプとモーターから成る。地上を走行したり、歩行したりするロボットでは、この部分をどのような構造と機能にするかが大きな比重を占めると考えられる。いっぽう、飛行ロボットでは、この部分は比較的単純な構造と機能であると言える。

ただし、この部分のもつ時間遅れの要素は、十分考慮されなければならない。ローターは、アンプからの信号によって、回転速度が決められるが、その信号の値に応じて瞬間的に（時間遅れおよび緩和時間なしで）回転速度が希望の値に変化するわけではない。

### ローター制御値と角速度の関係

緩和時間がロボットの運動に要する時間に比べて十分に小さいと見なせれば、ローターの角速度は、アンプへの制御値  $\Omega_k$  だけの関数とみなしでも妥当であると考えられるが、実際は、後節 5.4 で述べるように、緩和時間が数十ミリ秒ほどなので、無視できない。したがって、ローターの角速度  $\omega_k$  は、 $\Omega_k$  と時刻  $t$  の関数である。

$$\omega_k = f(\Omega_k, t) \quad (5.4)$$

この関数  $f$  の具体的な形は、ローター、モーターおよびアンプの種類に大きく依存するので、実測によって決定する。まず、十分に時間が経った時のローターの角速度を実測し  $\omega_p$  と表すことにする。すなわち、

$$\omega_p \equiv f(\Omega, \infty) \quad (5.5)$$

と定義する。この  $\omega_p$  が、次節のローターの回転運動における、角速度の目標値である。実測の結果、この  $f(\Omega, \infty)$  は  $\Omega$  の 1 次関数で近似できることが分かっている（図 5.5 参照）。

$$\omega_p = a\Omega + b \quad (5.6)$$

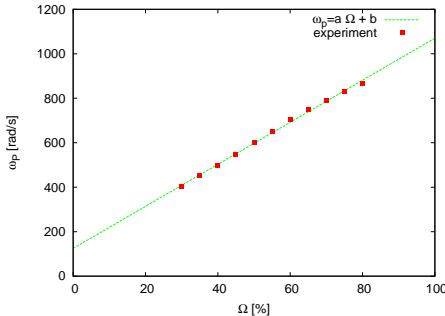


図 5.5:  $f(\Omega, \infty)$  の実測値をつかったフィッティング.  $a = 9.453[\text{rad/s\%}]$ ,  $b = 124.81[\text{rad/s}]$

### 一次関数による感覚運動写像

次に,  $\Omega$  の値を  $\varphi, \dot{\varphi}$  などの値に応じて, 目標姿勢を達成するために決定しなければならない. すなわち,

$$\Omega = \pi(\varphi, \dot{\varphi}, \ddot{\varphi}, \dots) \quad (5.7)$$

$\varphi, \dot{\varphi}, \ddot{\varphi}, \dots$  を感覚,  $\Omega$  をそれに対する応答であるとみなせば, 関数  $\pi$  は, 感覚運動写像と言う事もできる [1]. また,  $\varphi, \dot{\varphi}, \ddot{\varphi}, \dots$  を状態,  $\Omega$  をそれに対する行動とみなせば, 強化学習の用語を用いて  $\pi$  を決定論的方策関数と言う事もできる [6].

感覚運動写像  $\pi$  の最も直感的で基本的な方法は, 反応を感覚の 1 次関数で構成する方法である.

$$\begin{aligned} \Omega_1 &= -K_G(\dot{\varphi} - \dot{\varphi}_a) - K_A(\varphi - \varphi_a) \\ &\quad - K_D(\ddot{\varphi} - \ddot{\varphi}_a) + c(\varphi) \end{aligned} \quad (5.8)$$

$$\begin{aligned} \Omega_3 &= +K_G(\dot{\varphi} - \dot{\varphi}_a) + K_A(\varphi - \varphi_a) \\ &\quad + K_D(\ddot{\varphi} - \ddot{\varphi}_a) + c(\varphi) \end{aligned} \quad (5.9)$$

ここで,  $\varphi_a, \dot{\varphi}_a$  および  $\ddot{\varphi}_a$  は目標姿勢 (aim) を表す. これら 3 つの項におけるゲインパラメーター  $K_G, K_A$  および  $K_D$  の値は, 目標姿勢を達成する

ために調節されなければならない。それらの値は、飛行ロボットの形状や、ローター、モーターの種類などに依存するものであるし、目標姿勢の種類によっても変化するであろう。実験による、試行錯誤でそれらを決定する方法もあるし、限界感度法などの種々の方法も考案されている。遺伝的アルゴリズムや、教師データが準備できる場合であれば、ニューラルネットを用いることも考えられる。

これら、3つの項は、ロボットの傾きや角速度を制御する項であるが、一方、 $c(\varphi)$ は、飛行ロボットを重力に逆らって持ち上げるために必要な項である。この項は、傾きや角速度には影響を与えないが、ゼロにしてしまうと飛行ロボットは飛行状態を維持できないため、必要な項である。

### ホバリング

いま、飛行ロボットのホバリング、すなわち

$$\varphi_a = 0, \quad \dot{\varphi}_a = 0, \quad \ddot{\varphi}_a = 0 \quad (5.10)$$

を目標姿勢とすると、

$$\Omega_1 = -K_G\dot{\varphi} - K_A\varphi - K_D\ddot{\varphi} + c(\varphi) \quad (5.11)$$

$$\Omega_3 = +K_G\dot{\varphi} + K_A\varphi + K_D\ddot{\varphi} + c(\varphi) \quad (5.12)$$

である。さらに、 $K_D = 0$ と固定したPI制御の場合、

$$\Omega_1 = -K_G\dot{\varphi} - K_A\varphi + c(\varphi) \quad (5.13)$$

$$\Omega_3 = +K_G\dot{\varphi} + K_A\varphi + c(\varphi) \quad (5.14)$$

という式でローター操作値  $\Omega$  が決定される。

### 5.3 時間遅れと離散制御

ロボットの回転角度（傾き） $\varphi$ と、その角速度 $\dot{\varphi}$ は時刻 $t$ に依存するので、それを明示的に表すと(5.13), (5.14)式は、

$$\Omega_1(t) = -K_G\dot{\varphi}(t) - K_A\varphi(t) + c(\varphi(t)) \quad (5.15)$$

$$\Omega_3(t) = +K_G\dot{\varphi}(t) + K_A\varphi(t) + c(\varphi(t)) \quad (5.16)$$

と表される。 $\varphi$ は加速度センサー、また $\dot{\varphi}$ はジャイロセンサーで観測する。 $\Omega_i$ を計算してローターが実際に回転を変化させ始める時刻 $t$ は、それらセンサー値の信号伝達やノイズ除去の後なので、センサーでそれらを観測した瞬間よりも、少し遅れているはずである。逆の言い方をすれば、ローターは現在 $t$ より少し前 $t - \delta$ のセンサー情報を元に決められた操作値を受け取ることになる。

$$\begin{aligned} \Omega_1(t) &= -K_G\dot{\varphi}(t - \delta_G) - K_A\varphi(t - \delta_A) \\ &\quad + c(\varphi(t - \delta_A)) \end{aligned} \quad (5.17)$$

$$\begin{aligned} \Omega_3(t) &= +K_G\dot{\varphi}(t - \delta_G) + K_A\varphi(t - \delta_A) \\ &\quad + c(\varphi(t - \delta_A)) \end{aligned} \quad (5.18)$$

ここで、ジャイロセンサーで観測される $\dot{\varphi}$ の時間遅れを $\delta_G$ また、加速度センサーで観測される $\varphi$ の時間遅れを $\delta_A$ と表した。信号伝達による遅れ（むだ時間）は約30[ms]であることが分かっている[8]。また、ローパスフィルターなどによるノイズ除去演算によって、さらに時間遅れは増加する。30個のデータによる単純移動平均によってセンサー値のノイズを低減すると、合計約280[ms]ほどの時間遅れが生じると考えられる[8]。

さらに、ボードコンピューターは常に連続的に操作値をモーターに送り続けているわけではなく、1秒間に30~50回程度の頻度で離散的にモーターを制御している。したがって、その間隔は20~33[ms]である。この値は、先に述べた時間遅れや、後ほど述べる、ローター回転運動の緩和時間とほぼ同等の大きさを持っている。そのためこの制御時刻の離散性

を考慮しなければならない。この離散的な制御の時間間隔を  $\Delta$  とすると、時刻  $t$  ( $j\Delta \leq t < (j+1)\Delta$ ) におけるローター 1,3 の操作値は、

$$\begin{aligned}\Omega_1(t) &= -K_G \dot{\varphi}(j\Delta - \delta_G) - K_A \varphi(j\Delta - \delta_A) \\ &\quad + c(\varphi(j\Delta - \delta_A))\end{aligned}\tag{5.19}$$

$$\begin{aligned}\Omega_3(t) &= +K_G \dot{\varphi}(j\Delta - \delta_G) + K_A \varphi(j\Delta - \delta_A) \\ &\quad + c(\varphi(j\Delta - \delta_A))\end{aligned}\tag{5.20}$$

$$\text{ただし } j = 1, 2, 3, \dots\tag{5.21}$$

である。

## 5.4 ローターの回転運動における緩和時間

ローターおよびモーターの慣性モーメントを一体として  $J$  で表すことにする。この節における議論は、4つのローター全てにおいて共通していることなので、ローターを区別する添字  $i$  を省略して、ローターの角速度を  $\omega$  で表す。また、その目標値を  $\omega_p$  とする。

### 運動方程式

$\omega$  を変化させる角加速度は、 $\omega_p - \omega$  に比例する回転トルクによって与えられると考えられる。すなわち、

$$J\ddot{\omega} = \alpha(\omega_p - \omega)\tag{5.22}$$

という微分方程式にしたがって、ローターの角速度は変化すると考えられる。 $\alpha$  はモーター、ローターおよびアンプの性能を反映したパラメータである。

## 角速度の時間変化

この微分方程式の同次形

$$J\dot{\omega} = -\alpha\omega \quad (5.23)$$

の解は

$$\omega = Ae^{-\frac{\alpha}{J}t} \quad (5.24)$$

である。非同次微分方程式である(5.22)式の解を求めるために、 $A$ を定数ではなく、 $t$ の関数であると考えると、 $\omega(t) = A(t)e^{-\frac{\alpha}{J}t}$ であるから、

$$\begin{aligned}\dot{\omega} &= \dot{A}e^{-\frac{\alpha}{J}t} - \frac{\alpha}{J}Ae^{-\frac{\alpha}{J}t} \\ J\dot{\omega} &= J\dot{A}e^{-\frac{\alpha}{J}t} - \alpha Ae^{-\frac{\alpha}{J}t} \\ J\dot{\omega} &= J\dot{A}e^{-\frac{\alpha}{J}t} - \alpha\omega\end{aligned}\quad (5.25)$$

(5.25)式と(5.22)式を比較すると、

$$\alpha\omega_p = J\dot{A}e^{-\frac{\alpha}{J}t} \quad (5.26)$$

であることがわかる。 $\dot{A}$ を時間積分すると、

$$\begin{aligned}\dot{A} &= \frac{\alpha}{J}\omega_p e^{\frac{\alpha}{J}t} \\ A &= \omega_p e^{\frac{\alpha}{J}t} + B\end{aligned}\quad (5.27)$$

 $B$ は $t$ に依存しない定数である。(5.27)式を(5.24)式に用いると、

$$\begin{aligned}\omega &= (\omega_p e^{\frac{\alpha}{J}t} + B)e^{-\frac{\alpha}{J}t} \\ \omega &= \omega_p + Be^{-\frac{\alpha}{J}t}\end{aligned}\quad (5.28)$$

定数 $B$ は $\omega$ の初期値 $\omega(0)$ によって決まる。

$$\begin{aligned}\omega(0) &= \omega_p + B \\ B &= \omega(0) - \omega_p\end{aligned}\quad (5.29)$$

これを、(5.28) 式に用いると、

$$\omega(t) = \omega_p + (\omega(0) - \omega_p)e^{-\frac{\alpha}{J}t} \quad (5.30)$$

となる。ローターの慣性モーメント  $J$  は、およその値は推測できるが、サイズが 10 数センチで、重量も数グラムなので、2 点吊り法で正確にその値を計測することは、空気抵抗や摩擦の影響が大きく、困難であると考えられる。また、 $\alpha$  の値は、ローター、モーターおよびアンプの性能、あるいはバッテリーの性能や起電力にも依存する可能性があり、先見的にその値を求ることは、これも困難である。

そこで、

$$\tau \equiv \frac{J}{\alpha} \quad (5.31)$$

と  $\tau$  を定義する。これを、(5.30) 式に用いると、 $\omega(t)$  は

$$\omega(t) = \omega_p + (\omega(0) - \omega_p)e^{-\frac{t}{\tau}} \quad (5.32)$$

というかたちで、時間依存すると考えられる。 $\tau$  がローター角速度の時間変化を特徴付ける緩和時間である。

### 角速度実測値からもとめられる緩和時間

図 5.6 に、ローター操作値を 40% から 50% に増やした場合のローター回転速度の実測値を示した。また、その時間変化に関して、(5.32) 式を使って、最小二乗法フィッティングを行った結果も同時に示した。このように、ローター角速度が増加する場合、すなわち、 $\omega_p > \omega(0)$  の場合の緩和時間  $\tau_+$  は約 48.5[ms] であることがわかる。

いっぽう、ローター操作値を 50% から 45% に下げた場合のローター角速度の実測値およびそのフィッティング結果を図 5.7 に示した。ローター角速度が減少する場合、すなわち、 $\omega_p < \omega(0)$  の場合の緩和時間  $\tau_-$  は約 41.4[ms] となり、 $\tau_+$  の値よりも大きい。すなわち、角速度を減速する際には、その緩和時間は加速する場合よりも、6[ms] ほど長くなる傾向が

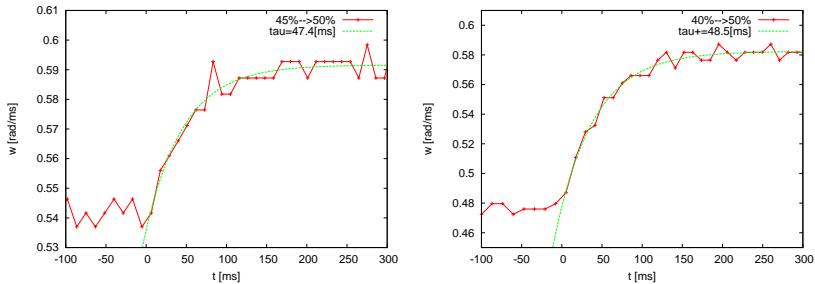


図 5.6: ローター操作値を 45%から 50%に増やした場合のローター回転速度の時間変化.  $\tau_+ = 47.4[\text{ms}]$ . ローター操作値を 40%から 50%に増やした場合のローター回転速度の時間変化.  $\tau_+ = 48.5[\text{ms}]$ .

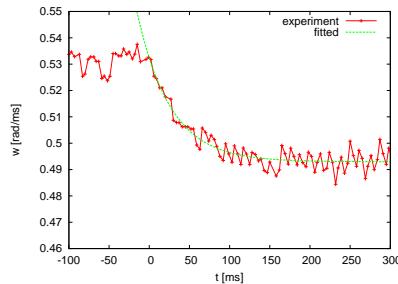


図 5.7: ローター操作値を 50%から 45%に下げた場合のローター回転速度の時間変化.  $\omega_p = 0.493[\text{rad/ms}]$ ,  $\omega(0) = 0.532[\text{rad/ms}]$ ,  $\tau_- = 41.4[\text{ms}]$

ある. つまり, ローターの減速には, 加速よりも時間がかかることがわかる.

## 5.5 時間遅れを考慮した拡張遷移行列

3 章では, 時間遅れがない場合に, 線形近似と行列形式を用いて系の挙動を解析した. 実際の飛行ロボットでは, データ転送時間やアンプの遅延時間など, 時間遅れ  $\delta$  が無視できない. つまり, 現在の状態の変化

率が、少し昔  $t - \delta$  における状態によって決まるというわけである。これを微分方程式で表すと、

$$\frac{d}{dt}\dot{\varphi}(t) = -c_G\dot{\varphi}(t - \delta) - c_A\varphi(t - \delta) \quad (5.33)$$

となる。この式からは、時間遅れがないときのように行列の指標関数を用いて解を容易に求めることができない。

制御理論によれば、フィードバック系のダイナミクスが微分方程式で記述できる場合、それをラプラス変換することによって伝達関数を求める。時間遅れがない場合、伝達関数は有理関数となり、それを逆ラプラス変換することによって、系の振る舞いを明示的に求めることができる。

しかし時間遅れが緩和時間などの時定数と同程度の長さ存在し、無視できない場合、伝達関数は有理関数とはならず、逆ラプラス変換で系の振る舞いを明示的に記述できない。振動応答を求め、ボード線図からその安定性などが議論されるが、時間遅れが無い系に比べて一般に制御が困難であると言われている。その対処法としては、大別すると主に以下の3通りの方法がとられている。

ひとつ目は、伝達関数をパディ近似などで有理関数に近似するというものである。いったん有理関数のかたちに近似された伝達関数は逆ラプラス変換などを通じて詳細に検討可能となる。2つ目は、スミス法などのように、予測器を系に挿入することによって、見かけ上時間遅れをフィードバックループの外に出した形で、予測制御を行うものである。

また、3つ目として、現実の制御システムでよく用いられる方法がある。実際に制御系を動かしてみて、振動の周期などを観測し、その観測データに基づいてゲインを調節するという、限界感度法などが用いられる。しかし、この経験的な方法も時間遅れが存在する制御システムにおいてはその有効性が低いと言われている。だいいち、どのような挙動をするか不明な状態で、飛行ロボットを飛ばしてみて、その様子を観察することは非常に危険なので、いまの場合、この方法を用いることには無理があると言えるであろう。

そこで、ここでは、先に示した行列形式の方法を用いて時間遅れのあ

る制御系を記述し、遷移行列の固有値問題としてその振る舞いを見通すことを試みる。まず、時間遅れがゼロではないが、非常に微小である場合を考える。いずれにせよ、時間遅れが無い場合のように単純な指数関数のかたちでは状態を記述できないので、系の時間発展を遷移行列によるベクトルの漸化式として表すことにする。遷移行列は時間遅れがない場合にすでに現れたが、時間遅れがある場合、遷移行列の次元を拡張するという自然なかたちでその影響を記述できる。遷移行列の次元が増えるわけであるから、その固有値と固有ベクトルを求める手続きは複雑化するわけだが、いったん固有値が求まれば、その性質によって制御系の振る舞いが直接記述されるという利点があると考えられる。

行列形式の具体的な記述に戻ろう。時間遅れ  $\delta$  を含む運動方程式(5.33)に加えて、以下の自明な式

$$\frac{d}{dt}\varphi(t) = \dot{\varphi}(t) \quad (5.34)$$

をまとめて行列形式で書くと

$$\frac{d}{dt} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} + \begin{pmatrix} -c_G & -c_A \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{\varphi}(t-\delta) \\ \varphi(t-\delta) \end{pmatrix} \quad (5.35)$$

ここで、 $\hat{A}_0, \hat{A}_1$  を

$$\hat{A}_0 \equiv \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad (5.36)$$

$$\hat{A}_1 \equiv \begin{pmatrix} -c_G & -c_A \\ 0 & 0 \end{pmatrix} \quad (5.37)$$

と定義すれば、(5.35)式は、

$$\frac{d\vec{s}(t)}{dt} = \hat{A}_0 \vec{s}(t) + \hat{A}_1 \vec{s}(t-\delta) \quad (5.38)$$

となる。

微小時間  $dt = \Delta t$  また、状態ベクトル  $\vec{s}(t)$  の微小変化量  $d\vec{s}(t) = \Delta\vec{s}(t)$  と書くと、

$$\frac{\Delta\vec{s}(t)}{\Delta t} = \hat{A}_0\vec{s}(t) + \hat{A}_1\vec{s}(t - \delta) \quad (5.39)$$

$$\Delta\vec{s}(t) = \hat{A}_0\vec{s}(t)\Delta t + \hat{A}_1\vec{s}(t - \delta)\Delta t \quad (5.40)$$

である。両辺に  $\vec{s}(t)$  を加えると、

$$\vec{s}(t) + \Delta\vec{s}(t) = \vec{s}(t) + \hat{A}_0\vec{s}(t)\Delta t + \hat{A}_1\vec{s}(t - \delta)\Delta t \quad (5.41)$$

となる。ここで、

$$\vec{s}_i = \vec{s}(t) \quad (5.42)$$

$$\vec{s}_{i+1} = \vec{s}(t) + \Delta\vec{s}(t) \quad (5.43)$$

というように、状態ベクトル  $\vec{s}$  が離散的に発展していくとみなす。また、いま時間遅れが微小で  $\delta = \Delta t$  であるとき、

$$\vec{s}(t - \delta) = \vec{s}(t - \Delta t) = \vec{s}_{i-1} \quad (5.44)$$

とみなせる。したがって、時間発展式 (5.41) は、

$$\vec{s}_{i+1} = \vec{s}_i + \hat{A}_0\vec{s}_i\Delta t + \hat{A}_1\vec{s}_{i-1}\Delta t \quad (5.45)$$

$$\begin{pmatrix} \vec{s}_{i+1} \\ \vec{s}_i \end{pmatrix} = \begin{pmatrix} \hat{I} + \hat{A}_0\Delta t & \hat{A}_1\Delta t \\ \hat{I} & \hat{0} \end{pmatrix} \begin{pmatrix} \vec{s}_i \\ \vec{s}_{i-1} \end{pmatrix} \quad (5.46)$$

と表される。 $\hat{A}$  行列が  $\hat{A}_0$  と  $\hat{A}_1$  のふたつの行列にわかれたので、 $\hat{A}$  行列の固有値および固有ベクトルを直接用いた議論がここではできない。

そこで、次のように微小パラメータ  $\varepsilon$  を用いて拡張遷移行列  $\hat{T}(\varepsilon)$  導入する。

$$\begin{aligned} \hat{T}(\varepsilon) &\equiv \begin{pmatrix} \hat{I} + \hat{A}_0\Delta t\varepsilon & \{(1 - \varepsilon)\hat{A}_0 + \hat{A}_1\}\Delta t \\ \hat{I} & \hat{0} \end{pmatrix} \\ &= \begin{pmatrix} \hat{I} & \hat{A}\Delta t \\ \hat{I} & \hat{0} \end{pmatrix} + \begin{pmatrix} \hat{A}_0\Delta t\varepsilon & -\hat{A}_0\Delta t\varepsilon \\ \hat{0} & \hat{0} \end{pmatrix} \end{aligned} \quad (5.47)$$

ここで、 $\hat{I}$  は単位行列、 $\hat{0}$  は要素がすべてゼロのゼロ行列である。ベクトルのベクトルや、行列の行列という見慣れない形式が現れたが、全体としてそれぞれベクトルと行列になるので、この簡略化された表現を用いることにする。

さらに、 $\hat{H}$  を

$$\hat{H} \equiv \begin{pmatrix} \hat{A}_0 \Delta t & -\hat{A}_0 \Delta t \\ \hat{0} & \hat{0} \end{pmatrix} \quad (5.48)$$

と定義する。

## 5.6 フィボナッチ型遷移行列

$\varepsilon = 0$  の場合、漸化式は

$$\begin{pmatrix} \vec{s}_{i+1} \\ \vec{s}_i \end{pmatrix} = \begin{pmatrix} \hat{I} & \hat{A} \Delta t \\ \hat{I} & \hat{0} \end{pmatrix} \begin{pmatrix} \vec{s}_i \\ \vec{s}_{i-1} \end{pmatrix} \quad (5.49)$$

と書き表すことができる。

(5.49) 式内の行列は  $2 \times 2$  行列のように記述されているが、その行列要素が  $2 \times 2$  行列なので、全体として  $4 \times 4$  行列を表す。ベクトルについても同様である。

この漸化式 (5.49) の行列要素がスカラーで、すべて 1 の場合は、フィボナッチ数列という有名な数列を表す漸化式となり、その性質は詳しく知られている。その類似性から、(5.49) 式に現れる行列をフィボナッチ型遷移行列  $\hat{T}_F$  と呼ぶことにしよう。

$$\hat{T}_F \equiv \begin{pmatrix} \hat{I} & \hat{A} \Delta t \\ \hat{I} & \hat{0} \end{pmatrix} \quad (5.50)$$

フィボナッチ型遷移行列  $\hat{T}_F$  と  $\hat{H}$  を用いることで、

$$\hat{T}(\varepsilon) = \hat{T}_F + \varepsilon \hat{H} \quad (5.51)$$

$\hat{T}(1)$  が (5.46) 式の漸化式における遷移行列を表す。また、 $\hat{T}(0)$  がフィボナッチ型の遷移行列を表す。

## 固有値

フィボナッチ型遷移行列の固有値  $\eta$  と右固有ベクトルを求める。右固有ベクトル  $\vec{\nu}$  を

$$\vec{\nu}_F = \begin{pmatrix} \vec{\alpha} \\ \vec{\beta} \end{pmatrix} \quad (5.52)$$

と書くことになると、

$$\begin{pmatrix} \hat{I} & \hat{A}\Delta t \\ \hat{I} & \hat{0} \end{pmatrix} \begin{pmatrix} \vec{\alpha} \\ \vec{\beta} \end{pmatrix} = \eta \begin{pmatrix} \vec{\alpha} \\ \vec{\beta} \end{pmatrix} \quad (5.53)$$

と表すことができる。掛け算を実行してみると、

$$\vec{\alpha} + \Delta t \hat{A} \vec{\beta} = \eta \vec{\alpha} \quad (5.54)$$

$$\vec{\alpha} = \eta \vec{\beta} \quad (5.55)$$

となるが、(5.55) 式を (5.54) 式に代入すると、

$$\eta(1 - \eta) \vec{\beta} + \Delta t \hat{A} \vec{\beta} = \vec{0} \quad (5.56)$$

ここで、行列  $\hat{A}$  の右固有ベクトル  $\vec{\nu}_1, \vec{\nu}_2$  を使って、

$$\vec{\beta} = \beta_1 \vec{\nu}_1 + \beta_2 \vec{\nu}_2 \quad (5.57)$$

と線形結合によって  $\vec{\beta}$  を表すと、

$$\{\eta(1 - \eta) + \lambda_1 \Delta t\} \beta_1 \vec{\nu}_1 + \{\eta(1 - \eta) + \lambda_2 \Delta t\} \beta_2 \vec{\nu}_2 = \vec{0} \quad (5.58)$$

である。 $\vec{\nu}_1, \vec{\nu}_2$  は 1 次独立であるから、任意の  $\beta_1, \beta_2$  に対して、この等式が成り立つためには、

$$\eta(1 - \eta) + \lambda_1 \Delta t = 0 \quad (5.59)$$

$$\eta(1 - \eta) + \lambda_2 \Delta t = 0 \quad (5.60)$$

でなければならない。これらは、2次方程式であるから、解の公式をもちいて簡単に解が得られ、

$$\eta_k = \frac{1}{2} \left( 1 \pm \sqrt{1 + 4\lambda_k \Delta t} \right) \quad (k = 1, 2) \quad (5.61)$$

という4つの固有値が得られる。この固有値の性質を調べれば、飛行ロボットの状態変化、すなわち振動、収束および発散が予測できる。

## 5.7 時間遅れの固有値への繰り込み

感覚運動写像による系の振る舞いを $\hat{A}$ 行列による微分方程式で記述すると、その固有値 $\lambda$ の値から、系の振る舞いがわかる。この微分方程式を、対応する漸化式の形に書き直すと、系の振る舞いは遷移行列を用いて表される。もちろん、遷移行列の固有値と、 $\hat{A}$ 行列の間には関係性があるので、遷移行列の固有値から $\hat{A}$ 行列の固有値は求められる。

まず、微小な時間遅れ $\delta = \Delta t$ があると、それが無い場合から遷移行列がどのように変化するかを調べる。その固有値から $\hat{A}$ 行列の固有値を求めれば、それは微小な時間遅れが繰り込まれた $\lambda$ の値を意味する。この繰り込み変換を $\delta = n\Delta t$ となるまで繰り返せば、その時えられた $\lambda$ の値は、時間遅れ $\delta$ に対応する $\hat{A}$ 行列の固有値を意味する。そのとき、 $\delta = n\Delta t$ の値を保ったまま、 $\Delta t \rightarrow 0, n \rightarrow \infty$ の極限をとる必要がある。

### 微小時間遅れによる $\hat{A}$ 行列固有値の変化

$\delta = \Delta t$ のときのフィボナッチ型遷移行列に対する固有値 $\eta_k$ は、

$$\eta_k = \frac{1}{2} \left( 1 \pm \sqrt{1 + 4\lambda_k \Delta t} \right) \quad (5.62)$$

と求められている。3.12節で述べた様に、遷移行列の固有値 $E_k$ と $\hat{A}$ 行列の固有値の間には、

$$\lambda_k = \frac{E_k - 1}{\Delta t}$$

という関係がある。

いま、 $E_k = \eta_k$  であることに注意すると、

$$\frac{\eta_k - 1}{\Delta t} \quad (5.63)$$

の値が時間遅れ  $\delta$  によってどのように影響を受けるかを調べることによつて、時間遅れの効果を考えよう。この式によると、解  $\eta_k$  の中で

$$\eta = \frac{1}{2} \left( 1 + \sqrt{1 + 4\lambda\Delta t} \right) \quad (5.64)$$

が重要な意味を持つことが予想される。ここでは、 $k = 1, 2$  のどちらの場合でも共通する議論なので、それを省略して示す。

(5.64) 式の形の  $\eta$  をそのまま用いると、考えにくいので、元の特性方程式

$$\eta(1 - \eta) + \lambda\Delta t = 0 \quad (5.65)$$

に戻って、その解が  $\Delta t$  によって 1 からどのように変位するか調べよう。

$\eta$  の  $\Delta t$  による 1 階微分を  $\dot{\eta}$  で表すこととする。この特性方程式 (5.65) の両辺を  $\Delta t$  で微分すると、

$$\begin{aligned} \dot{\eta}(1 - \eta) - \eta\dot{\eta} + \lambda &= 0 \\ \dot{\eta} &= \frac{\lambda}{2\eta - 1} \end{aligned} \quad (5.66)$$

となる。いま、1 からの変位を考えているので、 $\eta \neq 1/2$  と仮定しても良いであろう。

ここで、(5.64) 式をつかうと、

$$2\eta - 1 = \sqrt{1 + 4\lambda\Delta t} \quad (5.67)$$

なので、

$$\dot{\eta} = \frac{\lambda}{\sqrt{1 + 4\lambda\Delta t}} \quad (5.68)$$

のことから、 $\eta$  は

$$\begin{aligned}\eta &\simeq 1 + \dot{\eta}\Delta t \\ &= 1 + \frac{\lambda}{\sqrt{1 + 4\lambda\Delta t}}\Delta t\end{aligned}\quad (5.69)$$

と、求められる。系の振る舞いを特徴づける量を求めるために、これを(5.63)式に用いると、

$$\frac{\eta - 1}{\Delta t} \simeq \frac{\lambda}{\sqrt{1 + 4\lambda\Delta t}} \quad (5.70)$$

となる。

この(5.70)式の左辺の意味は、わずかな時間遅れ  $\Delta t$  によって修正された固有値  $\lambda + \Delta\lambda$  であると解釈できる。すなわち、

$$\lambda + \Delta\lambda = \frac{\lambda}{\sqrt{1 + 4\lambda\Delta t}} \quad (5.71)$$

という  $\lambda$  の時間発展方程式とみなせる。

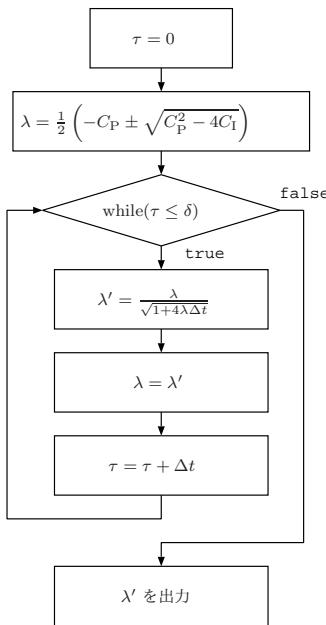
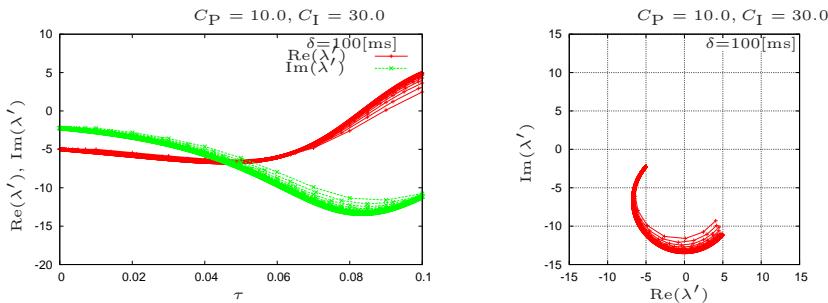
### 時間遅れのある系における $\lambda$

(5.71)式を  $\delta = n\Delta t$  となるまで  $n$  回くり返し用いて得られた  $\lambda$  は、時間遅れ  $\delta$  に対応する  $\lambda$  であると考えられる。

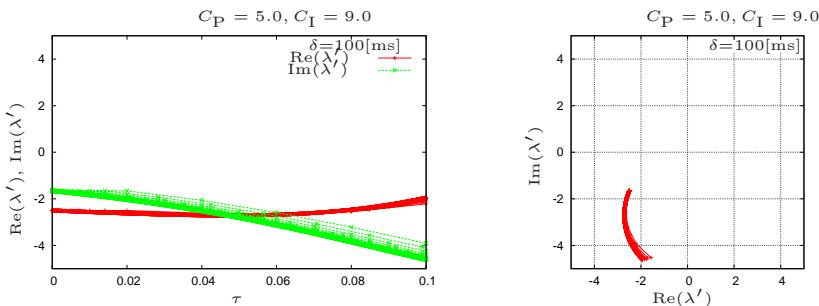
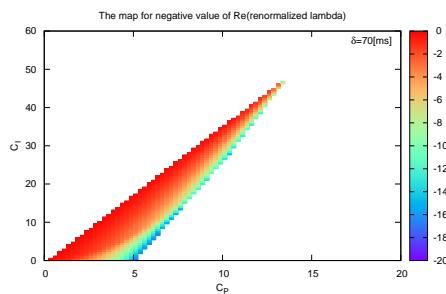
つまり図 5.8 に示した計算を行う。最終的に得られる  $\lambda'$  が時間遅れ  $\delta$  に対応する  $\hat{A}$  行列の固有値に対応すると考えられる。

図 5.9 に様々な  $\Delta t$  の値に対する、この手続きによって求められた  $\lambda'$  について示した。 $\Delta t$  の値を徐々に小さくしていくと、変化の様子は一定の振る舞いに落ち着くので、 $\lambda'$  の  $\Delta t$  依存性は、収束していると考えられる。

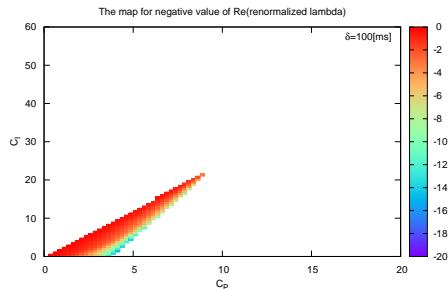
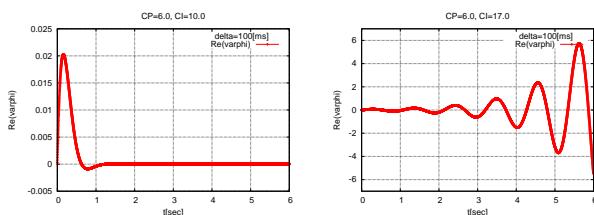
$\tau = 0$  における値が時間遅れが無い場合の  $\lambda$  である。実部、虚部ともに負の値なので、減衰振動をする。いっぽう、 $\tau = \delta$  まで修正された  $\text{Re}(\lambda')$  の値は正である。すなわち、時間遅れの効果によって、減衰振動が発散振動に変化したことを意味する。

図 5.8:  $\hat{A}$  行列の固有値  $\lambda$  に対する繰り込みアルゴリズム図 5.9:  $\lambda$  の時間遅れによる変化。 $\text{Re}(\lambda') > 0$  となる場合の例

さまざまな  $C_P, C_I$  の値に対して同様の計算を行い、 $\text{Re}(\lambda') < 0$  となる領域を図 5.11 に示した。時間遅れがない場合には、ゲイン領域のあらゆ

図 5.10:  $\lambda$  の時間遅れによる変化。 $\text{Re}(\lambda') < 0$  となる場合の例図 5.11: 修正された  $\lambda'$  の実部が負になる領域地図  $\delta = 70[\text{ms}]$ 

る値に対して機体の振る舞いは収束したが、時間遅れが存在することによって、運動が収束するのは非常に限られたゲイン領域であることが分かる。

図 5.12: 修正された  $\lambda'$  の実部が負になる領域地図  $\delta = 100[\text{ms}]$ 図 5.13: 時間遅れが繰り込まれた  $\lambda'$  を用いた状態  $\varphi(t)$



## 第II部

# 実践的な側面



ここでは、実際にロボットをつくる、環境のなかで行動させるため、またそれを観察するために必要な道具となる事柄をまとめた。

LEGO Mindstorm EV3 や BeagleBone Black などのハードウェア的な内容からそれらを利用するためには必要なオペレーティングシステム Debian OS についての内容まで様々な内容を含む。

それぞれについて詳しく正確な記述は、ここでは求めず、あくまでもロボットをつくる動かすために必要な部分をまとめたものであると思って欲しい。

それぞれの章や節は、関連性があるが、比較的独立しているので、マニュアルとして利用することも可能であると思う。



## 第6章 Debian をつかう

Debian とは、ファイル管理やネットワーク環境などコンピューターのオペレーティングシステム機能を提供するフリーソフトウェアパッケージ群のことです。ここでは、Debian GNU/Linux<sup>1</sup> のことを単に Debian と呼ぶことにします。

日常的に利用するデスクトップ環境はもちろん、サーバー機能など、29,000 以上のソフトウェアを提供しており、日本語利用環境<sup>2</sup> も充実しています。

ロボットインテリジェンスの研究に欠かせないボードコンピューターでは、ARM アーキテクチャの CPU が使われることが多いです。Debian は ARM 用のパッケージも提供しているので、i386 や AMD64 アーキテクチャをつかったノート PC などと同じ要領でボードコンピューターにインストールやアップデートなどの管理が可能であるため、さまざまな OS が混在する状況よりも、円滑な作業が可能です。

Debian はフリーな OS であることが特長です。ここでフリーとは、自由という意味です<sup>3</sup>。しかし、Debian は無料で入手できるので、事実上、無料という意味も含まれています。

下記に Debian のインストール DVD のイメージがあるので、ダウンロードしてインストール DVD を作ります。

<http://cdimage.debian.org/debian-cd/current/amd64/iso-dvd/>

インストールの方法には、おおきく 2 通りあります。インストーラーだ

<sup>1</sup><http://www.debian.org/>

<sup>2</sup><http://www.debian.or.jp/>

<sup>3</sup>[http://www.debian.org/social\\_contract#guidelines](http://www.debian.org/social_contract#guidelines)

けが含まれた CD イメージだけをダウンロードして、それをを利用してインストーラーを起動し、パッケージはネットワークからダウンロードしながら Debian をインストールする方法がひとつめです。ネットワークインストール用の CD イメージはサイズが小さくダウンロードが容易です。いっぽう、パッケージも含んだ DVD イメージをダウンロードし、インストール DVD を作り、それをつかって Debian をインストールする方法もあります。DVD-1 にインストーラー機能が含まれています。

ネットワークインストールの方法においては、大量のソフトウェアパッケージをインターネットを通じてダウンロードしなければならないので、インストール時間はネットワークの通信速度に依存します。どのようなパッケージを選択するかによって構築される Debian のサイズは異なりますが、デスクトップ環境やフォントまで含めると 4G バイト程度、BeagleBone Black などのボードコンピューターに CUI 環境を構築すると 1G バイトほどのハードディスク領域を消費するだけで Debian 環境を構築できます。

つまり Debian は、microSD や USB メモリのなかに簡単に構築できてしまします。デフォルトで GUI を含んだ、Debian 以外の巨大な OS を使わず、Debian をロボットインテリジェンス研究に利用する理由のひとつに、この OS のサイズが大きすぎず、自由にパッケージ管理することで、その大きさをコントロールすることができるという点もあります。

### なぜこれほどのOSが無料なのか

Debianをはじめ「フリー」とよばれるソフトウェアをつくっている開発者は、多大な労力かけてそれを作っている。一般には労力に対して金銭で報酬が支払われるが、Debian開発者はそれを求めていないことである。

ソフトウェア開発者は、自分の作ったソフトウェアに誇りをもっており、それが世界中で活用されるということに価値を見出しているのである。ソフトウェアの実態は文字記号の羅列であり、簡単に複製することができる。それが記述されているハードウェア(HDやメモリーなど)が壊れても、壊れていないハードウェアにコピーされたソフトウェアは100%もとのソフトウェアと同じ機能を備えており、損なわれることはない。

Debianを含めたGNUソフトウェアは自由な複製や修正を保証したライセンスの下に管理されている。複製が禁止されたソフトウェアは、それがインストールされているハードウェアが壊れ、その配布元がなくなれば地球上から消滅する。しかし、複製が許されたソフトウェアはだれかが複製するかぎり消滅しない。自らが生み出したものが、いわば「永遠の命」をもった存在になるのである。これ以上の報酬があるだろうか。

ロボット知能の構成論的研究のなかでDebianを活用することは、こういった精神のソフトウェア開発者に対する報酬の一部であると著者は考えている。

## 6.1 パッケージの管理

Debianの大きな特長の一つに、ソフトウェア管理の容易さがあります。一通りDebianのインストールが終了したあとに、新たに別のソフトをイ

ンストールしたり、すでにインストールしてあるソフトを削除したりすることが、コマンドから簡単に実行できます。Debianでは、これらの操作を `dpkg`, `apt-get` また `aptitude` コマンドをつかって行います。

### 6.1.1 既にインストールされているパッケージの確認

たとえば、`g++`に関するパッケージで、すでにインストールされているものを表示するには、

```
# dpkg -l | grep g++
```

と入力します。ここで、プロンプト#は、rootユーザーになって作業をすることを意味します。実行結果例は、たとえば

ii	g++	4:4.7.2-1	amd64	GNU C++ compiler
ii	g++-4.7	4.7.2-5	amd64	GNU C++ compiler

となります。ふたつの`g++`に関するパッケージがインストールされていることが分かります。

### 6.1.2 DVD(CD) ドライブ内のパッケージ認識

DVD(CD) ドライブに Debian のサイトからダウンロードして作ったDVDを挿入します。ドライブを認識させるためには、

```
# apt-cdrom ident
```

その後、以下のコマンドでそれを認識させる必要があります。

```
# apt-cdrom add
```

認識させたいDVDが複数ある場合には、DVDを入れ替えてこのコマンドを繰り返す必要があります。認識されたDVDの情報は、`/etc/apt/sources.list`の中に記述されています。

### 6.1.3 パッケージ情報のアップデート

パッケージのインストールを実行するまえに、情報をアップデートします。

```
# apt-get update
```

### 6.1.4 パッケージの検索

たとえば、gvimに関するパッケージを検索する場合、

```
# apt-cache search gvim
```

と実行します。実行結果例は

```
vim-athena - Vi IMproved - enhanced vi editor - with Athena GUI  
vim-gui-common - Vi IMproved - Common GUI files  
netrik - vi ライクなキーバインディングによるテキストモード WWW ブラウザ  
vim-gnome - Vi IMproved - 強化版 vim エディタ - GNOME2 GUI 付き  
vim-gtk - Vi IMproved - 強化版 vim エディタ - GTK2 GUI 付き
```

です。出力結果が長く、折り返されて複数行に渡っていて見づらい場合もありますが、一番ひだりに表示されているのが、パッケージ名です。ここでは、vim-athena, vim-gui-common, netrik, vim-gnome, vim-gtk の 5 つの gvim に関連するパッケージがあることが分かります。

aptitude を用いる場合、

```
# aptitude search gvim
```

と search オプションを用います。

### 6.1.5 パッケージのインストール

例として、vim-gnome をインストールしてみます。

```
# apt-get install vim-gnome
```

あるいは、

```
# aptitude install vim-gnome
```

とします。aptitude コマンドは、apt-get コマンドと似ていますが、パッケージ間の依存関係の解決能力が改善されています。

### 6.1.6 ネットワークインストール

/etc/apt/sources.list の記述。

```
#deb cdrom:[Debian GNU/Linux testing _Wheezy_ - Official Snapshot \
amd64 DVD Binary-1 20121119-04:55]/ wheezy contrib main
...
deb http://ftp.nara.wide.ad.jp/debian/ wheezy main contrib non-free
```

PROXY の環境変数への設定。

```
# export http_proxy=proxy.muroran-it.ac.jp:8080
```

### 6.1.7 amd64 アーキテクチャで i386 アーキテクチャを使えるようにする

amd64 アーキテクチャで Debian をインストールすると、i386 でしか動かないソフトウェアが起動しない。具体的には、command not found などのメッセージが表示されるだけになる。

ia32-libs をインストールすれば、実行可能になる。

```
# dpkg --add-architecture i386
# apt-get update
# apt-get install ia32-libs
```

## 6.2 エディター vi

ロボットインテリジェンスの研究を行う場合、ボードコンピュータ上のプログラムファイルや、設定ファイルを編集することが必要となる。ボードコンピュータは(Graphical User Interface(GUI)を持たず、メインメモリもPCなどとくらべて大きくないので、viエディタを利用する必要がある。

viはDebian GNU/Linuxの元となったUNIXオペレーティングシステムが開発された当初から標準的に装備されており、使用メモリーも少なく動作も高速である。なおかつ迅速かつ高度な編集機能を十分に備え持っているので、ロボットインテリジェンスの研究に限らず、組み込みシステムなどの開発においてもviエディタが大きな役割を果たす。

### 6.2.1 viの動作設定

viの動作を設定するファイルは、`~/.exrc`である。ただし、`~`は、ホームディレクトリを意味する。たとえば、ユーザー名porcoの場合`/home/porco`のことである。

.exrcファイル内に

```
set nu
```

と書いておくと、起動したときに、自動的に行番号が表示される。

gvimが実行された時の、初期状態は、`~/.gvimrc`ファイルの内容によって指定できる。例えば、

```
set nu
set fileencodings=euc-jp,iso-2022-jp,sjis,utf8
set columns=150
set lines=40
colorscheme torte
syntax on
```

このように指定すると、行番号などが有効となる。

## 6.2.2 vi の使い方

vi は 2 つのモードを持つ。一つは、コマンドモードで、このモード内でカーソル移動や、文字消去、コピー&ペーストなどを行う。もう一つが、インプットモードで、キーボードから入力された文字列がそのままエディタに入力される。

### モードの切り替え

モードの切り替えは、文字”i”と ESC キーで行う。コマンドモードの状態で、”i”を押すと、カーソル位置に文字を入力できる入力モードにモードが切り替わる。いっぽう、入力モードの状態で ESC キーを押すとコマンドモードに戻る。(図 6.1 参照)

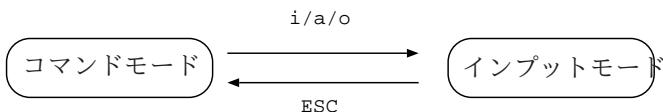


図 6.1: vi(gvim) におけるモードの切り替え

また、カーソル位置から一つだけ後ろの位置に文字入力したい場合は、”i”的かわりに”a”を押す。行末に文字を追加したい場合などに用いる。

入力行を増やす場合には、行末に移動してリターンキーを押してもよいが、コマンドモードで”o”を入力すると、次の行が追加され入力モードとなる。

### 編集コマンド

コマンドモードでよく使う編集コマンドキーを表 6.1 にまとめた。

表 6.1: vi でよく使う編集コマンド

機能	詳細	コマンド
終了	終了 破棄終了	:q :q!
保存	保存 保存終了	:w :wq
カット	一文字カット 一行カット $n$ 行カット	x dd dnd
移動	下に移動 上に移動 右に移動 左に移動 行末に移動 行頭に移動 $n$ 行目に移動	j (↓) k (↑) l (→) h (←) \$ ^ nG
コピー・ペースト	1 行ヤンク (コピー) 現在行から $n$ 行目までヤンク 現在行から $x$ 行分ヤンク ペースト	yy ynG yxy p
検索・置換	文字列検索 全文置換 範囲指定置換 (xxx-yyy 行目)	/文字列 :%s/文字列/置換文字列/gc :xxx,yyy/s/文字列/置換文字列/gc
ファイル間移動	ファイルを開く 最初のファイルに戻る	:e ファイル名 :rew
行連結		J
シェルコマンド		!:!コマンド

一つだけ文字列置換における注意点を挙げておく。置換コマンドの行末にある、 /gc の g は 1 行の中で複数個置換することを意味する。c は置換を毎回確認することを意味する。

これらのコマンドを覚えることで、キーボードのニュートラルポジションから大きく手を移動させずにファイルの編集が可能となる。マウスを使って文字の選択やペーストをおこなう必要はない。またプルダウンメニューから機能を選択したりする必要もない。そういう操作は、直感的なので、少數回ならば生産性の低下は軽微だが、ロボットインテリジェンスの研究のように、ボードコンピュータの GUI がない Debian 上でブ

ログラムなどを編集することが必要な場合、これらのコマンドを習得することが必須である。また、これらのコマンドを一旦使いこなせるようになれば、通常のマウスなどをつかったファイル編集と比べて、短時間で大量の文字列編集が可能となるため、作業効率は飛躍的に高まる。

ここに紹介した機能以外に、viはコマンドのマップなど、驚くほど多機能であり、なおかつ高速に動作する。ロボットインテリジェンスの研究においては、その活用が必須と言えるであろう。

### 6.2.3 GUI 上の vi

viは元々、Character User Interface(CUI)<sup>4</sup>上で利用することを前提としてつくられたエディタであるが、GUIで日本語（全角文字）を編集する場合にはGnome版のgvimが安定している。

```
$ gvim body.tex
```

## 6.3 ネットワーク

TCP/IPプロトコルでDebian PCとボードPCなどをネットワーク接続する。ネットワーク接続では、大まかに言って、物理接続、インターフェイスのドライバ、IPアドレス、名前解決(DNS, /etc/hosts)および経路解決(route)がそれぞれ適切に設定されている必要がある。

ここでは、ネットワークインターフェイスがeth0として認識されている場合について、IPアドレスの設定以降を説明する。

---

<sup>4</sup>コマンドラインインターフェイス(CLI)とも呼ばれる。

```

debian1.tex (~/Labor/2013/Mugui) ((1) of 9) - GVIM
ファイル(F) 編集(E) ツール(T) シンタックス(S) パッファ(B) ウィンドウ(W) ヘルプ(H)
83 一番上に表示されているのが、パッケージ名である。
84 ここでは、vim-athena, vim-gui-common, netrik, vim-gnome, vim-gtk の5つの
85 パッケージがあることがわかる。
86
87 \subsection{パッケージのインストール}
88 例として、vim-gnome をインストールしてみる。
89 \begin{verb+at+}
90 \begin{verb+at+}
91 # apt-get install vim-gnome
92 \end{verb+at+}
93 \end{verb+at+}
94
95
96
97 \section{エディター vi}
98 PC上のDebianはもちろん、飛行ロボットに搭載するボードコンピュータ上の
99 Linuxや、LEGO EV3上Linuxでも利用できるので、viエディタをつかうこと。
100
101 日本語（全角文字）を入力する場合にはGnome版のgvimが設定している。
102 \begin{screen}
103 \begin{verb+at+}
104 \$ gvim body.tex
105 \end{verb+at+}
106 \end{screen}
107
108 \begin{figure}
109 \begin{center}
110 \includegraphics[height=(\fheight,bb=0 0 988 871]{pics/gvimi1.png}
111 \end{center}
112 \end{figure}
113
114 \section{日本語テキスト環境}
115 \subsection{LaTeX}
116 発表会の原稿や卒業論文、修士論文はLaTeXをつかって書くこと。
117 \begin{verb+at+}
118 body.texというLaTeXソースをdviに変換する。
119 \begin{verb+at+}
120 \begin{verb+at+}
121 \$ pdflatex body.tex

```

図 6.2: gvim の実行例.

### 6.3.1 IP アドレス

#### 設定ファイル

/etc/network/interfaces の例.

```

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
    address 172.16.0.xxx
    netmask 255.255.0.0
    network 172.16.0.0
    broadcast 172.16.255.255
    gateway 172.16.0.1
    dns-nameservers 157.19.201.59

```

## インターフェイスの確認

ifconfig コマンドを利用する。

```
# ifconfig
```

## 始動と停止

ネットワークインターフェイス eth0 の停止と始動。以下のコマンドを実行すると、/etc/network/interfaces ファイルの内容が読み込まれて、インターフェイス eth0 の停止と始動が実行され

```
# ifconfig eth0 down  
# ifconfig eth0 up
```

### 6.3.2 名前解決

IP アドレスをすべて記憶しておけば、基本的にネットワークアクセスは可能である。しかし、IP アドレスは数字の羅列なので、記憶しにくい。そこで、IP アドレスとホスト名を対応付けておけば、ホスト名で IP アドレスを指定できるので、利用しやすい。

#### hosts の利用

IP アドレスと、ホスト名の対応表は、/etc/hosts ファイルの中に記述する。/etc/hosts ファイルの例

```
127.0.0.1 localhost
172.16.0.1 maple
172.16.0.2 wiki
172.16.0.18 arm18
172.16.0.42 arm42
172.16.10.18 arm18w
172.16.10.48 tama
172.16.0.50 maple50
172.16.0.51 maple51
172.16.10.51 castle
```

### 通信の確認

通信の確認は, ping コマンドによって行う. 自分の Debian PC から例えば maple にパケットを送って, 戻ってくるまでの時間が msec 単位で表示されれば, 通信に成功している. 何度も繰り返し表示されるが, CTRL+c を入力すると停止する.

```
# ping maple
```

### 結果例.

```
64 bytes from maple (172.16.0.1): icmp_req=1 ttl=63 time=2.19 ms
64 bytes from maple (172.16.0.1): icmp_req=2 ttl=63 time=1.53 ms
64 bytes from maple (172.16.0.1): icmp_req=3 ttl=63 time=1.59 ms
64 bytes from maple (172.16.0.1): icmp_req=4 ttl=63 time=1.73 ms
64 bytes from maple (172.16.0.1): icmp_req=5 ttl=63 time=1.54 ms
^C
```

## DNS, PROXY

ローカルなホストは/etc/hosts の中に列記しておけばよいが, 世界中のホストの IP アドレスをそこの列記するわけにはいかない. URL など自分が知らないホストの IP アドレスはドメインネームサーバに問い合わせせる.

ドメインネームサーバ(DNS)の IP アドレスを /etc/resolv.conf の中に記述する. たとえば DNS が aaa.bbb.ccc.ddd の場合, /etc/resolv.conf のなかに

```
nameserver aaa.bbb.cccddd
```

と記述する。

また、ブラウザなどで指定するプロクシーサーバー(PROXY)は

```
http://proxy.xxxx.ac.jp:8080/
```

のように指定する。最後の8080はポート番号を表す。

### 6.3.3 経路解決

デフォルトルーター(hoge)を追加する。

```
# route add default gw hoge eth0
```

詳しくは、9.4節を参照してください。

### 6.3.4 ネットワークファイルシステム NFS

ネットワークファイルシステム(NFS)を用いると、有線LANや無線LANで接続されたリモートDebianマシン上の外部記憶装置(HDやSDカードなど)や、個々のディレクトリがあたかもローカルなDebianマシン上に存在するかの様に利用することが可能である。

NFSを利用するためには、サーバー側でディレクトリをexportし、そのディレクトリをクライアント側でmountする必要がある。

#### exports

サーバDebianマシンhoge0で、/etc/exportsファイルにNFSクライアントを登録する。

```
/home/honda/work    hogel(rw,sync)  hoge2(rw,sync)
```

NFS サーバに/etc(exports の内容を反映させる。

```
hoge0# /etc/init.d/nfs-kernel-server restart
```

### mount

クライアント側で、

```
hoge1# mount -t nfs hoge0:/home/honda/work /home/honda/work
```

を実行することで、NFS マウントされる。もちろんディレクトリ名はこの例と同じでなくともよい。

## 6.4 kernel の構築とインストール

Linux ボードコンピュータなどに新たなハードウェアを接続したり、こまかいシステムの動作をチューニングするために、最新の kernel をダウンロードしてコンパイルする必要が生じる場合がある。自分の使う Linux にとって必要な kernel 構成を適切に行うためには、相当な慣れと時間を必要とするが、成功した際の果実は、他に得がたいものであるからチャレンジしてみる価値はある。

kernel のソースコードは、<https://www.kernel.org/pub/linux/kernel/> からダウンロードできる。最新のバージョンを用いることが推奨されるが、デバイスのファームウェアとの関係などから、低いバージョンのものを使う必要がある場合もある。上記の URL には過去の kernel ヴァージョンもあるので、状況に適したものを選ぶ。

作業ディレクトリでアーカイブを解凍する。ソースコードは膨大な量で、そこから生成されるバイナリも相当な容量となる。HDD の空きスペースに余裕のあるディレクトリを利用してないと容量を超えてしまう恐れがあるので注意が必要である。

### 6.4.1 構築

kernel は以下の手順でつくる。make menuconfig と make-kpkg を実行するためには必要なパッケージがインストールされていなければならぬが、以下を実行してみてエラーメッセージがでなければ既にそれらがインストールされているということである。必要なパッケージは後述する。

```
# make menuconfig  
...  
# make-kpkg clean  
# make-kpkg --initrd --revision xxxxx kernel_image
```

-initrd オプションは初期 ram disk を使ったブートのために必要である。

最初の make menuconfig で、カーネルのコンフィグレーションを決め。Wireless Network Device など数多くのドライバなどを kernel 自身に組み込むのか、module として外部において、必要な場合にロード (insmod) するのか、あるいは使わないのかを選択できる。

make menuconfig を実行するためには libncurses5-dev がインストールされている必要がある。

```
# aptitude install libncurses5-dev
```

また、make-kpkg を実行するためには kernel-package をインストールする。

```
# aptitude install kernel-package
```

先にも述べたが、kernel の構築にはかなり時間を要する。どれぐらいの時間がかかるかはケースバイケースだが、たとえば kernel 3.18 を Intel Core i7 で make-kpkg すると約 1 時間ほどで終了する。

## 6.4.2 インストール

コンパイルで構築されたカーネルは Debian パッケージとして `linux-xxx.deb` というファイル名で保存されている。xxx の部分には `make-kpkg` 実行時に指定した revision などが記述されている。

保存先は、`make-kpkg` を実行した作業ディレクトリのひとつ上のディレクトリである。`dpkg -i` コマンドを実行して新しい kernel をインストールする。

```
# cd ..
# dpkg -i linux-xxxx.deb
```

複数の OS を管理するブートローダー (GRUB など) が使われている場合には、そのコンフィグレーションも自動的に生成される。システムの再起動時に、ここでインストールした kernel ヴァージョンを選択すれば、自分でチューニングした Linux が起動する。



# 第7章 単純な反応行動

ボードコンピュータやジャイロセンサーなどを用いてロボットをつくるまえに、ここでは LEGO ロボットを用いて、簡単な反応行動のプログラムを作成し、行動実験を行う方法を説明する。

## 7.1 きまぐれロボット

ここでは、レゴ マインドストーム (LEGO MINDSTORMS) という小さな標準ロボットをコンピューター (PC) からプログラミングすることで、動かす。このロボットを以下では省略して LEGO ロボットと呼ぶことにする。

NXT (図 7.1(a) 参照) は WindowsPC 上の Bricx Command Center (BCC) (図 7.1(b) 参照) からプログラムする。

図 7.1: (a) レゴ標準ロボット

(b) 「知能」であるプログラム例

© 2009 KAWAHARA

### 7.1.1 LEGO ロボット

LEGO ロボットには定められた形はなく、パーツをブロックの様に組み合わせることで、さまざまな形のロボットを作ることができる。3つのモーターを用いることで、動きを生み出すことができる。その他に、超音波センサー (UltraSound), 音センサー (Sound), タッチセンサー (Touch), 光センサー (Light) の4つのセンサーを使って、環境を感じることができる（図7.2参照）。

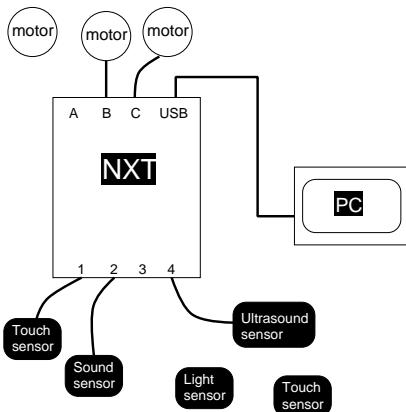


図 7.2: NXT にはモーターをつなぐインターフェイス (A,B,C) と 4 種類のセンサーをつなぐインターフェイス (1,2,3,4) がある。PC と NXT は USB インターフェイスで接続する。

LEGO ロボットはモーターとセンサー以外に小さなコンピューター (NXT) を持っている。ロボットの頭脳部分である。NXT にはモーターを接続するインターフェイスが3つ (A,B,C), センサーを接続するインターフェイスが4つ (1,2,3,4) ある。それらを通じてモーターを制御したり、センサーから情報を受け取ったりする。また、USB インターフェイスを使って後述するプログラムを他のコンピューター (PC) から受け取る。

プログラム自体は、PC で製作し、それを LEGO ロボットに搭載された NXT に転送して実行することによって、ロボットに「知能」を与える。

## 7.1.2 アルゴリズム

ロボットの知能にあたるプログラムはどのような構造をしているのか、直接プログラミング言語で記述するまえに、大まかな流れ（アルゴリズム）を設計する<sup>1</sup>。ここではアルゴリズムの記述に PAD 図 (Problem Analysis Diagram) を使う。図 7.3 に PAD 図で使われる構成要素を例示した。図

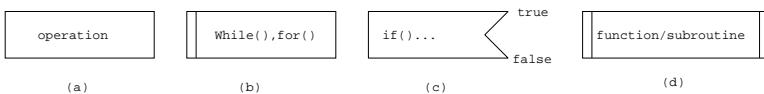


図 7.3: PAD の構成要素. (a) 演算, (b) 繰り返し, (c) 条件分岐, (d) 関数・サブルーチン

7.3(a) は演算、すなわち四則演算など命令実行を表す。図 7.3(b) は繰り返しを表す。図 7.3(c) は条件分岐を表す。図 7.3(d) は関数・サブルーチンを表す。構造化プログラミング [3, 4] の考え方によれば、これらの構成要素だけを使ってすべてのアルゴリズムを構成することが可能である。

例を使って説明しよう。二つのモーター BC を回して 1 秒間前進し、その後、モーター BC を反転させて 1 秒間後退させるアルゴリズムを、図 7.4(a) に示した。このアルゴリズムが実行されると、合計 2 秒間モーターが動作してすべて終了する。一方、図 7.4(b) に、1 秒前進、1 秒後退を無限に繰り返すアルゴリズムを示した。このアルゴリズムが実行されると、強制的にプログラムが終了されるまで、LEGO ロボットは動きつづける。

## 7.1.3 LEGO ロボットを動かす C 言語

前節で示したアルゴリズムを実際に C 言語 (NXC) のプログラムにすることによって、LEGO ロボットを動かしてみる。図 7.5 に図 7.4(a) のアルゴリズムに対応するプログラムを示した。一行目は、task というタイプ

<sup>1</sup> アルゴリズムという概念は、18 世紀、數学者オイラーが太陽と地球、そして月の 3 体問題を調べる際に考案した手続きにはじまったと言われている。

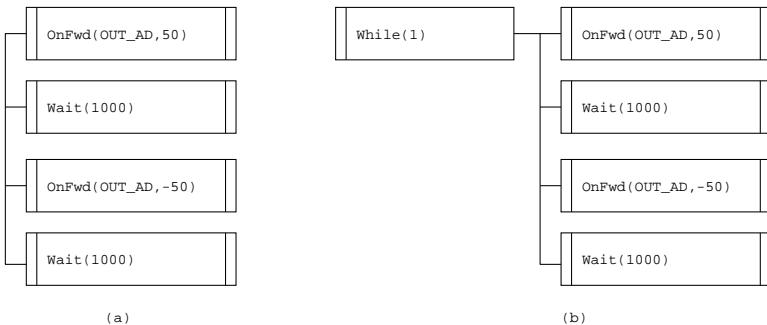


図 7.4: (a)1 秒前進して 1 秒後退するためのアルゴリズム. (b)1 秒前進して 1 秒後退を無限に繰り返すためのアルゴリズム

```
1 task main(){
2     OnFwd(OUT_BC,50);
3     Wait(1000);
4     OnRev(OUT_BC,50);
5     Wait(1000);
6 }
```

図 7.5: 1 秒間前進, 1 秒間後退のプログラム. 図 7.4(a) のアルゴリズムに  
対応

の main 関数がここから始まることを示している。main 関数というのは、プログラム全体のことだと思ってよい。LEGO ロボットを動かすための C 言語では、この main 関数のタイプは常に task を用いる。最後の行の } は main 関数がここで終わることを表しているので、忘れない様に注意すること。OnFwd 関数などの最後についている 50 という数字は、モーターの出力を 50% にすることを指定している。

つぎに、図 7.6 に図 7.4(b) のアルゴリズムに対応するプログラムを示した。while 文によって図 7.4(a) の内容を繰り返す。while 文は while(条件) という形をしており、条件が成り立つ(ture)限り { } でかこわれた内容が繰り返される。

```

1 task main() {
2     while(true) {
3         OnFwd(OUT_BC, 50);
4         Wait(1000);
5         OnRev(OUT_BC, 50);
6         Wait(1000);
7     }
8 }
```

図 7.6: 1 秒間前進、1 秒間後退を繰り返すプログラム。図 7.4(b) のアルゴリズムに対応

この例の場合、つねに `true` なので、無限に繰り返される。プログラムを強制的に終了するには、NXT のオレンジ色のボタンの下のボタンを押す。

### センサーで感じる

ここまでプログラムでは、LEGO ロボットは一定の決まった動きしかしなかった。センサーによって、障害物や音を感じることによって、状況に応じた動きが可能となる。条件文 (`if` 文) によって、センサーの感知した情報を判断する。

### センサーを有効にする

四種類のセンサーをつかえる状態にするための関数を、図 7.7 に示した。この関数をプログラムに一度書き入れるだけで、それぞれのインター

```

SetSensorTouch(IN_1);    // タッチセンサーを 1 番のインターフェイスに
SetSensorSound(IN_2);   // 音センサーを 2 番のインターフェイスに
SetSensorLight(IN_3);   // 光センサーを 2 番のインターフェイスに
SetSensorLowspeed(IN_4); // 超音波センサーを 4 番のインターフェイスに
```

図 7.7: 4 種類のセンサーを有効にする関数

フェイスにつながれたセンサーが使える準備ができる。

### タッチセンサーからの入力

タッチセンサーからの入力値は

$$\text{SENSOR\_1} = \begin{cases} 0 & (\text{タッチセンサーにものが触れていないとき}) \\ 1 & (\text{タッチセンサーにものが触れているとき}) \end{cases} \quad (7.1)$$

となる。図7.6の例で、while(true)の部分をwhile(SENSOR\\_1==0)に変えれば、タッチセンサーに物が触れていない間は、前後運動をつづけ、タッチセンサーに物が触れるとwhileによる繰り返しループが終了する。

### 音センサーからの入力

音センサーからの入力はSENSOR\_2で読み取ることができる。音センサーは感知した音を0から100の値でSENSOR\_2に返してくる。たとえば、図7.6の例でwhile(true)の部分をwhile(SENSOR\_2 < 40)にすればセンサーがレベル40を感じるよりも大きな音がすると、LEGOは前後運動を止めるプログラムになる。

### 超音波センサーからの入力

超音波センサーからの入力はSensorUS(IN\_4)で知ることができる。SensorUS(IN\_4)の値は、超音波センサーから物体までの距離をcm単位で表している。

たとえば、図7.8は、もし超音波センサーから障害物が30cm以内に近づいたら、0.4秒待ったあと、モータB,Cを反転させて、1秒間バックするための条件分岐(if文)である。このif文を上記のループ(while)の中に挿入することによって、ロボットの動きを環境条件によって変えることができる。

```
if(SensorUS(IN_4)<30) {  
    Wait(400);  
    OnRev(OUT_BC, 50);  
    Wait(1000);  
}
```

図 7.8: 超音波センサーからの入力情報で条件分岐する

### 光センサーからの入力

これらのセンサー以外に、もうひとつのタッチセンサーと光センサーが LEGO キットの中には含まれている。LEGO ロボットのには、それらのうちいずれかを追加接続して利用することもできる。光センサーからの入力は図 7.9 のように利用することができる。

```
if(Sensor(IN_3)>40) {  
    ...  
}
```

図 7.9: 光センサーからの入力で条件分岐する

### 確率的な動き

確率的なロボットの動きは、乱数を利用することで、作り出すことができる。Random(1000) という関数は、0~ 1000までの数をランダムに返してくれる。これをを利用して、確率 1/2 で、右回転と左回転をする if 文の例を以下に示す。

### 値を記憶する（変数）

ここまでは、センサーから帰ってくる値をそのまま利用していた。その値は、今現在の環境から得られる値である。たとえば光センサーの場

```

if(Random(1000)>500) {
    OnRev(OUT_B, 50);
    OnFwd(OUT_C, 50);
}
else{
    OnRev(OUT_C, 50);
    OnFwd(OUT_B, 50);
}

```

図 7.10: 確率的に方向を変えるための if 文

合, Sensor(IN\_3) の値は, 今現在の明るさを意味する. 過去の明るさを記憶しておきたい場合どうすれば良いだろうか? 過去の明るさを記憶しておいて, その明るさよりも現在の明るさの方が暗い場合は方向転換するなどの行動が可能である.

変数にセンサーからの値を記憶することによってそれが実現できる. たとえば, light という整数型の変数を使うとすると, 図 7.11 のような使い方が可能である.

```

int light; // light という変数を宣言する. プログラムの中で, 1 度だけでよい.
...
light=Sensor(IN\3); // 光センサーの出力値を light という変数に記憶する.

```

図 7.11: センサーからの値を変数に記憶する

## 昆虫の知能?

センサー入力を読み込んで, それらの値による動きと確率的な動きを簡単に組み合わせる例を示した. これらの非常に単純なアルゴリズムでロボットを動作させても, アリのような昆虫の運動に見えてしまうので不思議である.

NXT では、LEGO 社のファームウェアと Windows OS を用いてプログラミングしました。ここではその次世代である EV3 に Debian OS (ev3dev) を導入しロボット知能のプログラミングを行う方法を説明します。

## 7.2 EV3 上で Debian(ev3dev)

EV3 上で機能する Debian として ev3dev([www.ev3dev.org](http://www.ev3dev.org)) があります。ev3dev は EV3 用に基本的な部分から再構築された linux でなので、各種のハードウェアドライバも、linux kernel の考え方に基づいてつくられているようです。したがって、ファイルの読み書きをするようにそれらをアクセスすることができます。

### 7.2.1 インストールと起動

イメージファイルをダウンロードし、microSD に焼付けて、EV3 の microSD スロットに挿入した後、スイッチを入れると ev3dev が起動します。

<https://github.com/ev3dev/ev3dev/releases> からイメージファイルをダウンロードします。例えば、ev3-ev3dev-jessie-2015-07-08.img.zip をダウンロードし、microSD カードが /dev/sdb というデバイスとして認識されている場合、

```
# unzip ev3-ev3dev-jessie-2015-07-08.img.zip  
# dd if=ev3-ev3dev-jessie-2015-07-08.img of=/dev/sdb
```

と実行するだけで、ev3dev 全体が microSD にインストールされます。Debian 全体なので、解凍にも dd にも数分から十数分の時間を要します。

microSD スロットに ev3dev イメージをコピーした microSD カードを挿入し、電源を入れると、数分の起動処理の後、図 7.12 のような画面が表示されれば正常起動が行われたことがわかります。数分間の起動処理を経たあとオレンジ色の LED が点滅したまま、図 7.12 の画面が表示され

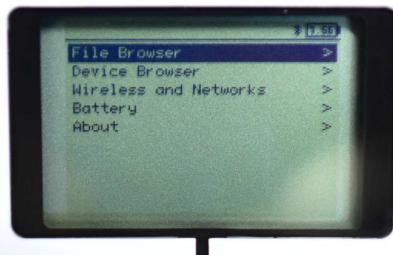


図 7.12: EV3 で microSD から ev3dev を起動した直後の液晶画面

ない場合があります。その場合には、図 7.13 に示した、A と B のボタン

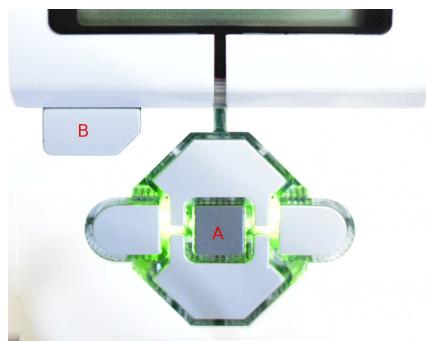


図 7.13: 起動リセットを実行する場合は、図中 A と B のボタンを同時に長押し（数秒）する。

を同時に長押しすると、システムがリセットされ、再起動処理が開始します。

ev3dev の起動が失敗するのは、1 番入力ポートに超音波センサーなどのセンサーを接続してある場合がほとんどです。センサーとの接続ケーブルを抜いた状態で起動すると、ev3dev の起動が途中で中断することは無くなります。

## 7.2.2 停止, 再起動

図 7.13 における B ボタンを押すと, shutdown, reboot などを選択する画面が表示されます。A ボタンの上下にあるボタンでいずれかを選び, A ボタンを押すと, シャットダウン (停止) あるいはリブート (再起動) プロセスが開始されます。

後述のように, USB ネットワークなどで, ev3dev にログインしたあとは, 普通の Linux や Debian のように shutdown -h now のコマンドで, 停止できます。

## 7.2.3 USB 有線 LAN を用いたセットアップ

USB-LAN アダプタ (Buffalo LUA3-U2-ATX) を USB ポートに挿すと, 自動認識されます。

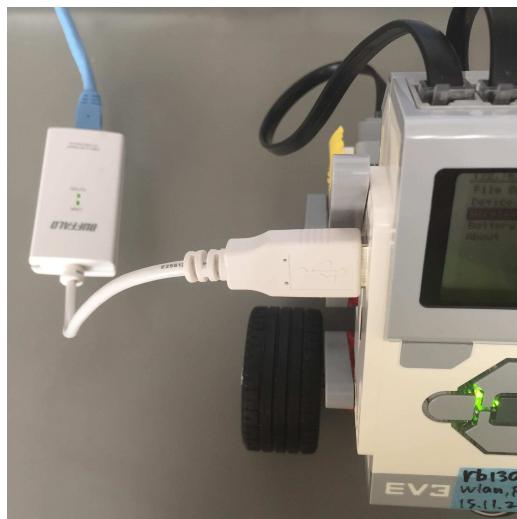


図 7.14: EV3 の USB ポートに有線 LAN を接続する。

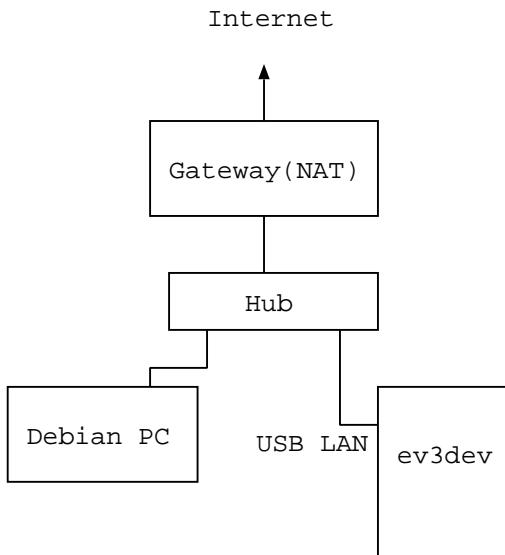


図 7.15: EV3 と Debian PC によるネットワーク構成

Wireless and Networks > All Network Connections > \*Wired

を EV3 の画面で選び IPv4 のタブのなかで、 IP address, Mask, Gateway を入力した後、一番下の apply を選ぶと、有線 LAN にアクセスできるようになります。IP address は 172.16.0.130 などのプライベートアドレスを用います。Mask は 255.255.0.0 とします。Gateway は、利用している環境に応じてルーターや NAT サーバなどのアドレスを入力します。

ssh などで Debian PC からログインした後は、普通の Debian として利用できます。

DebianPC> ssh root@aaa.bbb.cccddd

root の初期パスワードは r00tme になっています。aaa.bbb.cccddd は先に EV3 の画面で設定したプライベート IP アドレス、たとえば 172.16.0.130

などです。

/etc/resolv.conf のなかのアドレスを、たとえば 8.8.8.8<sup>2</sup> に変えると名前解決できるようになります。

```
root@ev3dev:~# vi /etc/resolv.conf
```

```
1 # Generated by Connection Manager  
2 nameserver 8.8.8.8
```

図 7.16: ev3dev における/etc/resolv.conf の例

/etc/apt/sources.list のなかの URL を国内のミラーサーバに変更すると、パッケージのダウンロードなどが高速化されます。apt-get を用いてパッ

```
1 #deb http://ftp.debian.org/debian jessie main contrib non-free  
2 deb http://ftp.nara.wide.ad.jp/debian jessie main contrib non-free  
3  
4 deb http://ev3dev.org/debian jessie main
```

図 7.17: ev3dev における/etc/apt/sources.list の例

ケージ情報をアップデートします。

```
root@ev3dev:~# apt-get update
```

i386 や amd64 の CPU をもつ DebianPC に比べると、時間がかかりますが、それらの DebianPC と同様に apt-get を利用することができます。

gcc,pvm,git をインストールします。

```
root@ev3dev:~# apt-get install gcc  
root@ev3dev:~# apt-get install pvm  
root@ev3dev:~# apt-get install pvm-dev  
root@ev3dev:~# apt-get install git-all  
root@ev3dev:~# apt-get install make  
root@ev3dev:~# apt-get install wireless-tools
```

<sup>2</sup>google public DNS(IPv4)。他に 8.8.4.4 がある

### 7.2.4 USB ネットワーク

EV3 の miniUSB インターフェイスと Debian PC の USB インターフェイスを USB ケーブルで接続すると双方で, usb0 というインターフェイスとして認識されます。

Debian PC 側では, ifconfig コマンドなどを使って, IP アドレスなどを設定すれば, 通常のネットワークインターフェイスと同様に使うことが出来ます (6.3 節を参照)。

EV3 側では, 液晶画面上で USB ネットワークの確認や設定をします。Top メニューから Wireless and Networks > USB > の順序で選択すると, USB ネットワークの状態選択画面が表示されます。CDC (Inactive) の場合, もう一度それを選択して中央の A ボタンを押すと, USB ネットワークが Active になり, IP アドレスも自動的に割り振られて, 液晶画面の左上に表示されます。

自動的に割り振られた IP アドレスを変更したい場合は,

Wireless and Networks > All Network Connections > \*Wired

とメニュー選択すれば, IPv4 ネットワークのアドレスなどを手動で変更できます。

変更しない場合でも, 液晶画面左上に表示された IP アドレスに合わせて, Debian PC 側で IP アドレスを設定すれば, ping や ssh などが利用できる様になります。

### 7.2.5 USB 無線 LAN

例として, NETGEAR の WNA1100 を EV3 の USB ポートに挿入して, 無線 LAN を利用する方法を示します。ファームウェア htc\_9271.fw を [http://wireless.kernel.org/download/htc\\_fw/](http://wireless.kernel.org/download/htc_fw/) から, ダウンロードします。ヴァージョンがいくつかあり, 常に更新されている。基

本的には最新のものを使うとバグや修正されてたり通信速度が改善している可能性があります。

htc\_9271.fw を /lib/firmware に置きます。次に insmod コマンドで各モジュールをロードすると、WNA1100 が認識されます。

```
insmod /lib/modules/2.6.33-rc4/kernel/compat/compat.ko
insmod /lib/modules/2.6.33-rc4/kernel/net/wireless/cfg80211.ko
insmod /lib/modules/2.6.33-rc4/kernel/net/mac80211/mac80211.ko
insmod /lib/modules/2.6.33-rc4/kernel/drivers/net/wireless/ath/ath.ko
insmod /lib/modules/2.6.33-rc4/kernel/drivers/net/wireless/ath/ath9k/\
        ath9k_hw.ko
insmod /lib/modules/2.6.33-rc4/kernel/drivers/net/wireless/ath/ath9k/\
        ath9k_common.ko
insmod /lib/modules/2.6.33-rc4/kernel/drivers/net/wireless/ath/ath9k/\
        ath9k.ko
insmod /lib/modules/2.6.33-rc4/kernel/drivers/net/wireless/ath/ath9k/\
        ath9k_htc.ko
```

起動時に自動的に実行するためには、スクリプトとして保存しておき、/etc/init.d で insserv コマンドを実行して登録します。

### wi-fi ネットワーク設定

EV3 の USB インターフェイスに、例えば NETGEAR WNA1100 を挿入すると、wlanX というインターフェイスとして自動的に認識されます。X の部分は 0, 1, … などの数字です。この順番を変更して、0 に戻す方法については、8.1.3 節を参照してください。

EV3 の液晶画面で

```
Wireless and Networks > Wi-Fi >
```

の順序でメニュー選択すると、周囲にある Wi-Fi 接続ポイントの ESSID が表示されますので、パスワードやキーが分かっている場合はそれらを選択します。IP アドレスは DHCP を利用して自動的に割り振られます。

これは EV3 をクライアントとして利用するだけならば設定が必要ないので便利ですが、EV3 にログインしたり、ホストを指定して PVM でプロセス間通信をしたりする場合には不便です。ev3dev も Debian ですので、

iwconfigなどのコマンドを利用してWi-Fiの設定をAd-Hocモードなどに固定して利用することも可能です。

iwconfigを利用するためには、

```
# dpkg -i libiw30_30-pre9-8_armel.deb  
# dpkg -i wireless-tools_30-pre9-8_armel.deb
```

これらのdebパッケージは、ネットワーク検索によって容易に探すことができますが、

```
https://packages.debian.org/jessie/armel/net
```

の中からリンクをたどることによってダウンロードできます。

ev3devでは、Wi-Fiは自動認識されDHCPでアドレスが取得されるようにデフォルトスクリプトが記述されているようです。無線LAN設定スクリプトをinsservコマンドで起動スクリプトに登録すれば、固定IPを利用するスクリプトが自動実行されますが、デフォルトスクリプトが後から実行されると、DHCP利用モードに戻ってしまいます。/etc/rc2.dなどのディレクトリ内のスクリプトへのリンク名SXXyyyなどで、XXの数字をディレクトリ内で一番大きな数字にリネームしておくと、最後に実行され、固定IPがデフォルトで利用可能になります。

### 7.3 PVM

ev3devにはデフォルトではPVMはインストールされていないので、下記のようにdpkgコマンドを使ってインストールできます。

```
# dpkg -i libpvm3_3.4.5-12.5_armel.deb  
# dpkg -i pvm_3.4.5-12.5_armel.deb
```

これらのパッケージは

```
https://packages.debian.org/jessie/libpvm3
```

などからダウンロードできます。

一般に、`dpkg` はパッケージ間の依存関係を解決してくれませんが、`iwconfig` と PVM を `ev3dev` 上にインストールする際には、依存関係の問題は生じません。

PVM の利用の仕方についてのより詳しい内容は、10 章を参照してください。

## 7.4 ev3c

`ev3c` は `ev3dev` が動いている EV3 上で、C 言語を使ってセンサーやモーターを制御するためのライブラリです。

```
$ git clone https://github.com/theZiz/ev3c.git
```

を実行すると、カレントディレクトリに `ev3c` というディレクトリが作成され、その下に各種ライブラリやテストプログラムがダウンロードされます。

`ev3c` というディレクトリがカレントディレクトリにできるので、そのディレクトリの中で `make` を実行すると、ライブラリやテストプログラムがコンパイルされて作成されます。 `Makefile` の中で、コンパイラは `$(CC)` で参照されているので、クロスコンパイラが `CC` という環境変数に指定されている必要があります。クロスコンパイラの導入方法は後述します。

## 7.5 クロスコンパイルと Makefile

`ev3dev` が動いている EV3 に `ssh` を使ってログインすれば、他の Debian など Linux と同様のシェル環境を利用することができます。デフォルトでは C コンパイラはインストールされていません。また、それをイン

ストールしたとしても、EV3 に用いられている CPU では、PC と比べてコンパイルに非常に時間がかかります。

Debian PC 上で、EV3 用のプログラムをクロスコンパイルして、生成された実行形式ファイルを scp などを使って送ってから実行することで、センサーモーターを利用することができます。

クロスコンパイラを次の URL アドレスからダウンロードします。

```
https://sourcery.mentor.com/GNUToolchain/package4571/public
/arm-none-linux-gnueabi
/arm-2009q1-203-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
```

このクロスコンパイラは i386 環境用なので、amd64 の環境では、ia32-libs をインストールして、i386 のバイナリが実行できる準備が必要です。

上記アーカイブを解凍すると、arm-2009q1/というディレクトリができるので、

```
# cp -rf arm-2009q1/ /usr/local/sbin/
# export CC=/usr/local/sbin/arm-2009q1/bin/arm-none-linux-gnueabi-gcc
```

とすることで、ev3c の Makefile を利用して、make 可能になります。

図 7.18 に示した Makefile を、アーカイブがコンパイルされた ev3c/ が存在するディレクトリ上に置いて make コマンドを実行すると、TG に指

---

```
1 TG=smmmap2
2 CFLAGS = -O3 -lm -lpvm3
3
4 $(TG): $(TG).c ev3c/lib/ev3c.a
5     $(CC) -o $@ $< ev3c/lib/ev3c.a $(CFLAGS)
6
7 clean:
8     rm -rf $(TG)
```

図 7.18: ev3c/ というディレクトリがあるディレクトリでクロスコンパイルするための Makefile

定された実行形式ができるので、それを scp などで ev3 に送れば、実行できます。

また， gcc,pvm,pvm-dev,make がインストールされた ev3dev 上で， この Makefile を使って make を行うこともできます。

Makefile を使うと， 複数のソースコードから実行形式のターゲットをコンパイルして生成できます。 make コマンドは， 複数のソースコードのうち， 更新されたものだけコンパイルを実行します。 つまり， すでに最新のものは再コンパイルしないので， 時間の節約になります。

1 行目で， コンパイル後に作成されるターゲットを指定しています。

2 行目では， コンパイルオプションと， 数学ライブラリおよび PVM ライブラリを指定しています。

4 行目で， ターゲット (\$TG) が \$TG.c と /ev3c/lib/ev3c.a に依存していることを示しています。

5 行目で， このターゲットに対して行われるコンパイルを記述しています。 \$@ は内部マクロで， ターゲット自身を意味します。 \$<sub>i</sub> は依存ファイルの左端のファイルを意味します。 今の場合， smmap2.c になります。 ev3c アーカイブ (ev3c.a) とともにターゲットを生成しているのがわかります。

7， 8 行目は， make clean を実行した際に， ターゲットを削除することを意味します。

### 7.5.1 感覚運動写像と PVM

感覚運動写像による走行ロボット上のプログラム (smmap2.c) を Debian PC 上のプログラム (master2.c) から生成し、ロボットのセンサーで得たセンサー値をリアルタイムで master2 側に PVM で送信する例を示す。

master2.c

---

```

1 #include <stdio.h>
2 #include <pvm3.h>
3
4 #define PROGRAM "/root/smmap2"
5 #define SLAVE "sg154w"
6 #define MAXNHOST 256
7 #define SLEEP_TIME 200000
8 #define OUT_FILE "smmap2.xy"
9
10 #define sTAG 1
11 #define rTAG 2
12
13 int main(){
14     int i;
15     int info;
16     int nhost, narch, infos;
17     int tids[MAXNHOST];
18     int dum;
19     FILE *fp;
20     double rdum[256];
21     struct pvmhostinfo *hostp;
22
23
24     info = pvm_config(&nhost, &narch, &hostp);
25
26     printf("nhost=%d \n",nhost);
27     printf("narch=%d \n",narch);
28     for(i=0;i<nhost;i++)
29         printf("host name[%d]=%s \n",i,hostp[i].hi_name);
30
31     // SLAVE 一つだけ spawn する。
32     pvm_spawn(PROGRAM, (char**)0, 1, SLAVE, 1,tids);
33     printf("tid:%d\n",tids[0]);
34
35
36     fp=fopen(OUT_FILE, "w");
37     i=0;
38     while(1){
39         // SLAVE から送り返されてきたデータを受け取る。
40         if(pvm_recv(tids[0],rTAG)){
41             pvm_upkdouble(rdum,2,1);
42             printf("recv:%lf %lf\n",rdum[0],rdum[1]);
43             fprintf(fp,"%d %lf %lf\n",i,rdum[0],rdum[1]);
44         }
45 }
```

```
46     fflush(fp);
47     i++;
48     usleep(SLEEP_TIME);
49 }
50 fclose(fp);
51 }
52 }
```

## smmap2.c(slave)

```
1  /*
2  2015 11.28 Yasushi Honda
3  cf. https://github.com/theZiz/ev3c
4  */
5
6 #include <stdio.h>
7 #include <math.h>
8 #include "ev3c/ev3c.h"
9 #include <pvm3.h>
10
11 // #define O_FILE "smmap2.xy"
12 #define I_MAX 300
13 #define SLEEP_TIME 200000
14 #define US_MAX 2550 // (mm)
15 #define MOTOR_MAX 500
16 #define B1 300 // (mm)
17 #define A1 0.005
18
19 #define rTAG 1
20 #define sTAG 2
21
22 #define SENSOR_PORT1 2
23 #define SENSOR_PORT2 4
24 #define SENSOR_MODE1 1
25 #define SENSOR_MODE2 1
26
27 int main(){
28     int i;
29     int j;
30     int left,right;
31     double us1d,us2d;
32     //FILE *fp;
33
34     int irecv;
35     int ptid;
36     double dum[256];
37
38     setlinebuf(stdout);
39
40     ptid=pvm_parent();
41
42     printf("こんにちは、超音波センサ値をsendします.\n");
43
44     //Loading all sensors
45     ev3_sensor_ptr sensors = ev3_load_sensors();
46     ev3_sensor_ptr us1,us2;
47
48     us1=ev3_search_sensor_by_port(sensors,SENSOR_PORT1);
49     us2=ev3_search_sensor_by_port(sensors,SENSOR_PORT2);
50     ev3_open_sensor(us1);
51     ev3_open_sensor(us2);
52     ev3_mode_sensor(us1,SENSOR_MODE1);
53     ev3_mode_sensor(us2,SENSOR_MODE2);
54 }
```

```
55     ev3_motor_ptr motors = ev3_load_motors();
56     ev3_reset_motor(motors);
57     ev3_open_motor(motors);
58     ev3_reset_motor(motors->next);
59     ev3_open_motor(motors->next);
60
61     ev3_set_ramp_up_sp(motors, 10);
62     ev3_set_ramp_down_sp(motors, 10);
63     ev3_set_ramp_up_sp(motors->next, 10);
64     ev3_set_ramp_down_sp(motors->next, 10);
65     //fp=fopen(O_FILE, "w");
66     i=0;
67     while(i<I_MAX) {
68         ev3_update_sensor_val(us1);
69         ev3_update_sensor_val(us2);
70         us1d=(double)us1->val_data[0].s32;
71         us2d=(double)us2->val_data[0].s32;
72         right=MOTOR_MAX*(tanh(A1*(us1d-B1))+0.7)*0.5;
73         left=MOTOR_MAX*(tanh(A1*(us2d-B1))+0.7)*0.5;
74         printf("%d/%d \t %lf (-) %lf (-) : %d %d \n",i,I_MAX,us1d,us2d,left,right);
75         //fprintf(fp,"%lf %d \n",us1d,right);
76
77         dum[0]=us1d;
78         dum[1]=us2d;
79         pvm_initsend(0);
80         pvm_pkdouble(dum,2,1);
81         pvm_send(ptid,sTAG);
82
83         /*
84         ev3_set_speed_sp(motors, left%1000);
85         ev3_set_speed_sp(motors->next, right%1000);
86         ev3_command_motor_by_name(motors, "run-forever");
87         ev3_command_motor_by_name(motors->next, "run-forever");
88         */
89
90         i++;
91         usleep(SLEEP_TIME);
92     }
93
94     //fclose(fp);
95
96     ev3_delete_sensors(sensors);
97     ev3_delete_motors(motors);
98     return 0;
99 }
```



# 第8章 ボードコンピュータ

ロボットには、1枚のボードで構成されたボードコンピュータを用いる。多くのボードコンピュータは数センチ四方のサイズで、様々な入力および出力インターフェイスを備えている。ロボットの筐体に組み込んで、感覚運動写像などを行うことに適している。

ボードコンピュータには、Armadillo, BeagleBone Black(BBB), Edison, Raspberry Pi などがある。Armadillo, BBB および Raspberry Pi は、そのCPUにARMアーキテクチャを採用している。一方、EdisonはIntel i386アーキテクチャの流れを汲むAtom採用している。また、LEGO EV3はボードコンピュータではないが、ARM CPUを採用している。

これらのボードコンピュータにLinux OS、特にDebian GNU/Linuxを導入してロボット知能に用いることを考える。DebianはサーバOSあるいはデスクトップOSとして開発されてきた経緯を持つが、それ故に膨大なライブラリやアプリケーションの資産をもっている。それをボードコンピュータに導入することにより、それらの資産を活用すると共に、デスクトップOSとの親和性や運用の容易さが期待される。

## 8.1 BeagleBone Black

BeagleBone Black（以下 BBBと略記）は、AD変換ポートと、PWMポートを複数備えた、ボードコンピュータである<sup>1</sup>。OSとして、Debianをインストールすることも可能であり、感覚行動写像を実現するハードウェアとして、PICやH8を必要としない点でも回路の簡略化や重量の軽

---

<sup>1</sup><http://beagleboard.org/products/beaglebone%20black>

量化が期待できる。また、Debian の中で C 言語を用いてセンサーヤモーターを制御できるので、Armadillo と同様に、複雑な制御プログラムもデスクトップ Linux と同様の環境の下で開発できることが期待される。

### 8.1.1 Debian のインストール

BBB 上に Debian をネットワークインストールする。ネットワークインストールというのは、最初に起動するインストーラーのみをローカルなデバイス (microSD) から起動し、Debian の本体システムはネットワーク経由でサーバーからダウンロードしながらインストールする方法である。このような方法を採用することにより、常に最新のバージョンがインストールできるというメリットがある。

#### 用意するもの

以下のものを用意する。

1. BBB
2. Debian PC と Internet 環境
3. USB-Serial ケーブル (6pin, 3.3V)
4. microSD (1GB 以上)
5. USB-LAN アダプター

BBB 自身は LAN インターフェイスを備えているが、インストーラの不都合上、自動的に認識されない場合があるので、USB インターフェイスに USB-LAN アダプタをつないで利用する。

## 手順

ネットワークインストール手順の流れは以下の通り。

1. Debian PC 上でプロクシ環境変数を設定する。

```
# export http_proxy="http://xxxx.yyyy.zzz:8080/"
```

2. microSD 上に、インストーラーを構築する。

まず、Debian PC 上に、インストーラなどをダウンロードする。

```
# git clone https://github.com/RobertCNelson/netinstall.git  
# cd netinstall
```

つぎに、必要な、ツール類をインストールする。

```
# apt-get install wget dosfstools parted u-boot-tools
```

もし、すでにインストールされていれば、自動的にスルーされる。

microSD を Debian PC に挿入し、`dmesg` コマンドで、microSD に対応するデバイス名を確認する。

```
# dmesg  
...  
[463528.914978] sd 6:0:0:0: [sdb] 15693824 512-byte logical blocks: (8.03 GB/7.48 GiB)  
[463528.915109] sd 6:0:0:0: [sdb] Cache data unavailable  
[463528.915113] sd 6:0:0:0: [sdb] Assuming drive cache: write through  
[463528.915355] sd 6:0:0:0: [sdb] Cache data unavailable  
[463528.915360] sd 6:0:0:0: [sdb] Assuming drive cache: write through  
[463528.916454] sdb:
```

たとえば、上記のようなメッセージが末尾に表示されたとすると、ブロックデバイスは`/dev/sdb`であることが分かる。そのブロックデバイスへ、スクリプトを使ってインストーラを構築する。

```
# ./mk_mmc.sh --mmc /dev/sdb --dtb am335x-boneblack --distro \  
wheezy-armhf --serial-mode linux-firmware
```

いくつか質問されるが、問題なければ”y”を答える。

3. BBB と Debian PC を USB-Serial ケーブルでつなぐ（図8.1参照）.

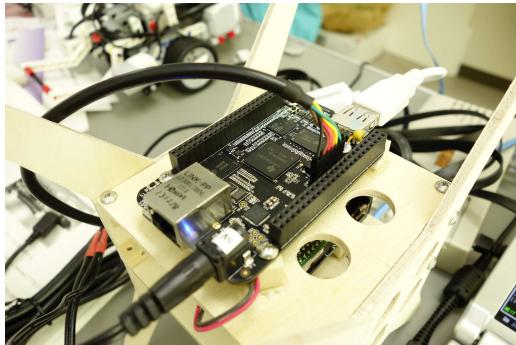


図8.1: BBB にシリアルケーブルを接続する。

ピン配列の向き（リード線の色）に注意する。黒いリード線が BBB 上の黒丸の方に位置するのが正しい向きである。Debian PC 側は USB ポートに接続する。

4. Debian PC 上で minicom （??参照）を実行する。  
通信設定 : 115200bps, 8bit, Non Parity, ハードウェアフロー制御 OFF
5. BBB の USB ポートに、USB-LAN アダプターを接続し Internet アクセス可能な。ハブに接続する。
6. インストーラーが構築された microSD を BBB に挿入し、電源を ON。  
BBB は miniUSB ポートからの電源供給でも起動できるが、インストール中の電力不足が懸念されるので、5V の AC アダプターからの電源供給が推奨される。
7. Debian インストーラーに従う。  
Language は C を選択する。LAN インターフェイスは、eth0, eth1 のふたつ認識されるが、USB-LAN アダプタは eth1 に対応するのでそ

ちらを選ぶ。DHCP サーバが無い場合には、手動で IP アドレスなどを入力する。ディスクのパーティションはインストーラーの推奨する切り方で yes を選択する。

### 8.1.2 Debian のイメージを直接 microSD にコピー

新たに Debian をインストールしてシステムを構築するのもよいが、既存の Debian イメージファイルから、新たな microSD 上にそっくり元と同じ Debian をコピーすることもできる。パッケージはアップデートされないが、こちらの方が、新規構築の手間は省ける。

Debian のインストールされた microSD を、別の Debian PC に挿入する。dmesg などを使ってデバイスを確認して、例えば /dev/sdb と認識されているとする。dd コマンドを使ってイメージファイルを作る。

```
# dd if=/dev/sdb of=xxxx.img
```

あらたな microSD カードを Debian PC に挿入し、

```
# dd if=xxxx.img of=/dev/sdb
```

を実行する。

### 8.1.3 ネットワークデバイスに付けられる番号

この方法で作った microSD を新たな BBB に挿入して電源を入れると、当然ながら元の BBB の情報がそのまま残っている。特にネットワークデバイス eth0, wlan0 などは、ハードウェア情報に基づいて番号付けされるので、新たなネットワークデバイスをもつ BBB では eth1, wlan1 などと割り振られる。これらの情報は /etc/udev/rules.d/70-persistent-net.rules というファイルの中に自動的に記述されている。その中の eth1, wlan1 の部分を eth0, wlan0 に書き換えて、元の eth0, wlan0 の部分は削除すると、再起動後に、新たなハードウェアが eth0, wlan0 として認識される。

### 8.1.4 PIC とのシリアル(UART)通信

PIC と BBB はそれぞれ UART インターフェイスをもっているので、シリアル通信可能である。たとえば、PIC でセンサー値（電圧）を AD コンバートして、その結果を BBB におくるなどする場合に利用できる。BBB 自身も AD コンバートのためのインターフェイスを持っているが、最大電圧が 1.8V であるため、最大電圧が 5V の加速度センサー値などを直接読み取ることができない。一方 16F887 などの PIC は 5V の AD コンバータポートを持っている。ので、それを利用する。

PIC の電源電圧を 5V にすると、UART 出力パルスの最大電圧も 5V となる。ところが、BBB の UART パルス電圧は最大 3.3V であるので、このままだと、直接通信できない。そこで、PIC への入力電圧を 3.3V にすると、UART 出力パルスも 3.3V となる。

3.3V は、BBB から電源用として出ているので、それを利用できる。また、PIC 内の C 言語ソースで、電源が 5V を下回った時のリセット機能を無効にする措置が必要である。

#### BBB のシリアルポート有効化

ケープマネージャの機能を使って、シリアルポートのデバイスマップを生成する。

```
# echo BB-UART1 > /sys/devices/bone_capemgr.*/slots
# chmod 666 /dev/ttyO1
```

#### PIC(16F877) で実行するプログラム (C 言語)

ヘッダファイルのインクルードなど。

```
//通信速度 38.4kbps(10MHz)

#include<pic14/pic16f887.h>

#ifndefdefine ADOSC 0x80 //ADC clock is 32TOSC
#define ACQUI 0x18
#define MAX_AN_PIN 14 //14個のAD変換ポート

//BORを不許可にする
int __at _CONFIG1L __config1 = _HS_OSC & _WDT_OFF & _PWRTE_ON
    & _LVP_OFF & _MCLRE_OFF & _DEBUG_OFF & _BOR_OFF;
int __at _CONFIG2 __config2 = _WRT_OFF;

//global variable
long int adc[MAX_AN_PIN]; //adc( AN0 ~ AN13 )

//prototype of functions
void set_config(void);
void get_send_adc(void);
void int_str(int send_num);
```

### main 関数.

```
void main(void) {
    int i;
    set_config();

    while(1) {
        get_send_adc();
        while(!TRMT);
        TXREG = 0x0A; //break line

        // Loop for decreasing sampling rate down to 50Hz
        i=0;
        while(i<2000) {i++;}
    }
}
```

シリアルポートに文字列を書き出す関数(intToStr())

```
void intToStr(int value){
    int mod, c=0;
    char tmp[8];
    //num to ascii code
    do{
        mod = (value%10)+48;
        value /= 10;
        tmp[c] = mod;
        c++;
    }while( value );
    //send character
    do{
        c--;
        while(!TRMT);
        TXREG = tmp[c];
    }while( c>0 );
}
```

受け取った値(value)を、アスキーコードに変換して、シリアルポートに割り当てられたアドレスに代入している。

AD変換ポートの値を読み込んで、シリアルポートに送る関数(get\_send\_adc())は以下のようなものである。

```

void get_send_adc(void){
    int t;

    //AN0 select
    ADCONO = 0xC3; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[0] = (ADRESH<<8) | ADRESL;
    //AN1 select
    ADCONO = 0xC7; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[1] = (ADRESH<<8) | ADRESL;
    //AN2 select
    ADCONO = 0xCB; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[2] = (ADRESH<<8) | ADRESL;
    //AN3 select
    ADCONO = 0xCF; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[3] = (ADRESH<<8) | ADRESL;
    //AN4 select
    ADCONO = 0xD3; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[4] = (ADRESH<<8) | ADRESL;
    //AN5 select
    ADCONO = 0xD7; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[5] = (ADRESH<<8) | ADRESL;
    //AN6 select
    ADCONO = 0xDB; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[6] = (ADRESH<<8) | ADRESL;
    //AN7 select
    ADCONO = 0xDF; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[7] = (ADRESH<<8) | ADRESL;
    //AN8 select
    ADCONO = 0xE3; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[8] = (ADRESH<<8) | ADRESL;
    //AN9 select
    ADCONO = 0xE7; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[9] = (ADRESH<<8) | ADRESL;
    //AN10 select
    ADCONO = 0xEB; for(t=0; t<ACQUI; t++); GO = 1; while(GO); adc[10] = (ADRESH<<8) | ADRESL;

    while(!TRMT); TXREG = 'a';
    //send an0 value
    while(!TRMT); int_str(adc[0]); while(!TRMT); TXREG = ';';
    //send an1 value
    while(!TRMT); int_str(adc[1]); while(!TRMT); TXREG = ';';
    //send an2 value
    while(!TRMT); int_str(adc[2]); while(!TRMT); TXREG = ';';
    //send an3 value
    while(!TRMT); int_str(adc[3]); while(!TRMT); TXREG = ';';
    //send an4 value
    while(!TRMT); int_str(adc[4]); while(!TRMT); TXREG = ';';
    //send an5 value
    while(!TRMT); int_str(adc[5]); while(!TRMT); TXREG = ';';
    //send an6 value
    while(!TRMT); int_str(adc[6]); while(!TRMT); TXREG = ';';
    //send an7 value
    while(!TRMT); int_str(adc[7]); while(!TRMT); TXREG = ';';
    //send an8 value
    while(!TRMT); int_str(adc[8]); while(!TRMT); TXREG = ';';
    //send an9 value
    while(!TRMT); int_str(adc[9]); while(!TRMT); TXREG = ';';
    //send an10 value
    while(!TRMT); int_str(adc[10]); while(!TRMT); TXREG = ';';
}

```

この例では、11個のポートの値を読み込んで、”a”始まる文字列として、それらの値を,””で区切ってシリアルポートに書き出し(int\_str())でいる。

シリアルポートの設定関数 (set\_config)

```

void set_config(void) {

    //port setting //1:input, 2:output
    TRISA = 0xff; //RA is input
    TRISB = 0xff; //RB is input
    TRISC6 = 0x00; //RC6 is output //serial port
    TRISC7 = 0x01; //RC7 is input //serial port
    TRISD = 0xff; //RD is input
    TRISE = 0xff; //RE is input

    PORTA = 0x00; //init portA
    PORTB = 0x00; //init portB
    PORTC = 0x00; //init portC
    PORTD = 0x00; //init portD
    PORTE = 0x00; //init portE

    BRGH = 1; //highspeed baudrate setting 1:high, 2:low
    SPBRG = 15; //38.4k 10MHz
    //SPBRG = 10; //115200 20MHz
    //SPBRG = 129; //9.6k 20MHz
    SYNC = 0; //non sync
    TXEN = 1; //sending enable

    //RCSTA register
    SPEN = 1; //serial port enable
    RX9 = 0; //no-parity 9bit receive
    SREN = 0; //1byte receive disable
    CREN = 1; //continuary receive enable
    FERR = 0; //no frame error
    OERR = 0; //no over run error
    RX9D = 0; //parity

    //A/D setting
    ADCON1 = 0x00; //AN0-AN7 is no reference
    //ADCON1 = 0x30; //AN0-AN7 is analog AN2 is Vref- AN3 is Vref+
    ADCON0 = 0xC0; //use RC oscillator
    ADON = 1; //start adc module
    ADFM = 1; //A/D right-justified
    ANSEL = 0xff; //AN0-AN7 is analog input.
    ANSELH = 0xff; //AN8-AN13 is analog input
}

}

```

この例では、10MHz のセラロックを利用して 38.kbps の通信速度でシリアル通信する設定となっている。通信速度などを変更する場合は、適宜 SPBRG の値を変更しなければならない。

このプログラムを SDCC でコンパイルしたあと、HEX 形式のファイルを PIC ライターで PIC(16F887) に書き込むと、その PIC は、14 個の AD 変換ポートに読み込んだ 0~5V の電圧値を数値（デジタル）に変換して、

シリアルポートに出力する。

### BBB 側の受信プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <math.h>

// **** serial setting ****
#define SERIAL "/dev/tty01" // For Acceleration sensor
#define SERIAL_SPEED B38400
// *****

// prototype of functions
void    init_serial(void);

//serial setting
struct termios oldtio;

//シリアルポートのファイルディスクリプタ
int serial_fd=0;
fd_set readfs;
```

シリアルポート初期化関数

```

void init_serial(void){

    struct termios newtio; // Set serial port

    serial_fd = open(SERIAL, O_RDWR | O_NOCTTY);
    if(serial_fd < 0){
        perror(SERIAL);
        exit(EXIT_FAILURE);
    }

    tcgetattr(serial_fd, &oldtio);
    bzero(&newtio, sizeof(newtio));

    // BAUDRATE: ポーレートの設定. cfsetispeed と cfsetspeed も使用できる.
    // CRTSCTS : 出力のハードウェアフロー制御 (必要な結線が全てされているケー
    // ブルを使う場合のみ. Serial-HOWTO の 7 章を参照のこと)
    // CS8     : 8n1 (8 ビット, ノンパリティ, ストップビット 1)
    // CLOCAL  : ローカル接続, モデム制御なし
    // CREAD   : 受信文字 (receiving characters) を有効にする.
    newtio.c_cflag = SERIAL_SPEED | CS8 | CLOCAL | CREAD;
    //newtio.c_cflag = SERIAL_SPEED | CRTSCTS | CS8 | CLOCAL | CREAD;

    // IGNPAR  : パリティエラーのデータは無視する
    // ICRNL   : CR を NL に対応させる (これを行わないと, 他のコンピュータで
    //             CR を入力しても, 入力が終りにならない)
    // それ以外の設定では, デバイスは raw モードである (他の入力処理は行わない)
    newtio.c_iflag = IGNPAR;

    //Raw mode output
    newtio.c_oflag = 0;

    // canonical input operation
    newtio.c_lflag = 0;

    // setting for waiting characters by read function
    newtio.c_cc[VTIME] = 0; // wait for a value time * 0.1 sec
    newtio.c_cc[VMIN] = 1;
    // wait until input of character which number is determined by this

    // clear buffer
    tcflush(serial_fd, TCIFLUSH);

    //cfsetspeed(&newtio, SERIAL_SPEED);
    tcsetattr(serial_fd, TCSANOW, &newtio);
}

```

行の先頭から改行まで, 1 文字ずつシリアルポートを読み取るプログラム例.

```
int main(){
    int l;
    char data;
    char buff[256];

    // In order to start at head of line,
    // searching tail of line.
    read(serial_fd, &data, 1);
    while(data!='\n'){
        read(serial_fd, &data, 1);
    }

    l=0;
    read(serial_fd, &data, 1);
    buff[l]=data;
    while(data!='\n'){
        l++;
        buff[l]=data;
        read(serial_fd, &data, 1);
    }
    l++;
    buff[l]=data;
    l++;
    buff[l]='\0';

}
```

## 8.1.5 USB 無線 LAN

### WNA1100

NETGEAR 社の WNA1100 を USB ポートに挿入する。このデバイスを認識するためのモジュールは `ath9k_htc` であるが、ファームウェアのバージョンを適切なものを選ばないと、`wlan0` が起動しない。

[http://wireless.kernel.org/download/htc\\_fw/](http://wireless.kernel.org/download/htc_fw/) から `htc_9271.fw` をダウンロードし /lib/firmware のなかにコピーする。その後、WNA1100 を BBB の USB ポートに挿入すると、自動的にモジュールがロードされる。`lsmod` コマンドで、確認する。

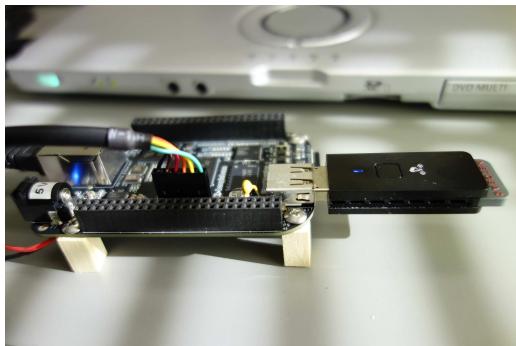


図 8.2: BBB に USB 無線 LAN アダプタとシリアルケーブルを接続する。

```
root@bbb70:/home/honda# lsmod
Module           Size  Used by
nfsd            233548  2
exportfs          3650  1 nfsd
arc4             1659  2
ath9k_htc        76190  0
ath9k_common     3261  1 ath9k_htc
ath9k_hw         391711 2 ath9k_common,ath9k_htc
ath              16416  3 ath9k_common,ath9k_htc,ath9k_hw
mac80211        505859  1 ath9k_htc
cfg80211        424614  3 ath,mac80211,ath9k_htc
rfkill           18347  1 cfg80211
```

と表示されれば正常認識されている。アドレスの設定等は、付録9に従う。

## WLI-UC-GNM

BUFFERLO の小型無線 LAN アダプタ、WLI-UC-GNM は ralink モジュールで自動認識される。ralink モジュールはデフォルトでインストールされているが、ファームウェアを追加でインストールする必要がある。

```
# aptitude install firmware-ralink
# reboot
```

以下に、設定例を示すが、無線 LAN に関する詳細は付録 9 を参照すること。

```
# aptitude install wireless-tools
# iwconfig wlan0 mode managed
# iwconfig wlan0 essid arm18
# iwconfig wlan0 channel 2
# ifconfig wlan0 172.16.10.77
```

### 8.1.6 i2c でセンサーの値を読み取る

i2c（アイ・ツー・シー）とは、周辺機器を CPU などに接続するためのシリアルバス、あるいはその通信方式のことである。組み込みシステムや、携帯電話などでよく使われる。ここでは、BBB とセンサーなどのデバイスを i2c で接続してデータ通信する方法を説明する。

i2c 通信においてデータ通信を行うのは、シリアルデータ (SDA) とシリアルクロック (SCL) と呼ばれる 2 本の通信線である。BBB 側と、センサー側にある SDA ピンどうし、SCL ピンどうしを、お互いに接続する。UART シリアル通信の場合には、TXD と RXD をお互いに接続した。i2c の場合には、SDA と SDA、SCL と SCL を接続するので、注意が必要である。

その他に、電源 (VDD) とアース (GND) が必要である。つまり、i2c 通信を行うためには、BBB とセンサーを、最低 4 本のリード線で結ぶ必要がある。センサーなどには、それ以外のピンが存在するが、さらに他の i2c デバイスをディージーチェーン（数珠つなぎ）接続する際などに用いるものである。

### 8.1.7 PIN 配置

BBB は 3 つの i2c インターフェイスを持っているが、1 つは内部で利用されており、ピンが外部に開放されていない。2 つめの i2c インターフェ

イスとして、P9-19(SCL), P9-20(SDA)がある。このi2cインターフェイスをユーザーは利用できるが、内部で他の機器が接続されている。

3つ目のi2cインターフェイスは、P9-17(SCL), P9-18(SDA)の2ピンが対応している。このインターフェイスには、内部でも他機器などが接続されておらず、すべてのアドレスが利用者に開放されている。これをセンサーなどとの接続に利用する。

## 2 I2C ports

P9			P8		
	1	2		1	2
DGND			DGND		
VDD_3V3	3	4	VDD_3V3		
VDD_5V	5	6	VDD_5V		
SYS_5V	7	8	SYS_5V		
PWR_BUT	9	10	SYS_RESETN		
GPIO_30	11	12	GPIO_60		
GPIO_31	13	14	GPIO_40		
GPIO_48	15	16	GPIO_51		
I2C1_SCL	17	18	I2C1_SDA		
I2C2_SCL	19	20	I2C2_SDA		
I2C2_SCL	21	22	I2C2_SDA		
GPIO_49	23	24	I2C1_SCL		
GPIO_117	25	26	I2C1_SDA		
GPIO_125	27	28	GPIO_123		
GPIO_121	29	30	GPIO_122		
GPIO_120	31	32	VDD_ADC		
AIN4	33	34	GND_ADC		
AIN6	35	36	AIN5		
AIN2	37	38	AIN3		
AIN0	39	40	AIN1		
GPIO_20	41	42	GPIO_7		
DGND	43	44	DGND		
DGND	45	46	DGND		

図8.3: BeagleBone Black のP8, P9ポートピン配列。P9-17, P9-18をi2c-2として利用する。

i2cピンを含む、BBBのP8ポートおよびP9ポートのピン配置を図8.3に示した。SCL, SDAの他に、電源(VDD)とグラウンド(DGND)をi2cセンサーとBBBの間で配線する必要がある。VDDは3.3Vと5VがBBBのP9のピンから供給されているので、センサーに必要な電圧のピンから配線する。たとえば、MPU9150というデバイスの電源電圧は2.4V～3.5Vなので、P9-3(or 4)ピンから供給できる。

## Device Tree

BBB に接続するデバイスなどの周辺機器は Device Tree とよばれる方法によって制御されている。 BBB の起動時にその情報が kernel にわたされていればデバイスファイル (`/dev/i2c-1` など) が利用可能になっている。

起動時に `/dev/i2c-*` が有効になっていないときはどうすればいいだろうか？ そのときには、次節で説明する Cape Manager をつかって Device Tree にデバイスを追加する方法をつかう。

## Cape Manager

Cape Manager というのは、Debian の kernel が起動した後に、i2c ポートを有効化するツールである。kernel 起動後のデフォルト状態で、`/dev/i2c-2` などが有効化されていない場合にはこれを用いる。

Cape Manager をつかって `/dev/i2c-2` を有効にするためには、

```
# echo BB-I2C1 > /sys/devices/bone_capemgr.*/slots
```

を実行する。つまり、所定のディレクトリにある `slots` というファイルに `BB-I2C1` と書き込む。すると、Cape Manager はこのファイルを監視していて、自動的に i2c インターフェイスを有効化する。

`i2cdetect` というコマンドをつかうと、i2c デバイスの状態を確認することができる。これを利用するために `i2c-tools` パッケージをインストールする。

```
# apt-get install i2c-tools
```

一度インストールすれば、起動するたびにインストール必要はない。i2c インターフェイスが利用できる状態になつければ、

```
# i2cdetect -l
```

を実行すると,

i2c-0 i2c	OMAP I2C adapter	I2C adapter
i2c-1 i2c	OMAP I2C adapter	I2C adapter
i2c-2 i2c	OMAP I2C adapter	I2C adapter

と表示される。3つのi2cインターフェイスが存在することがわかる。

さらに, i2cdetectコマンドをつかって,

```
# i2cdetect -r -y 2
```

を実行すると, i2c-2に接続されているデバイスのアドレスが表示される。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
10:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
20:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
30:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
40:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
50:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
60:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	68	
70:	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

この例では, 0x68にデバイスが存在することがわかる。(MPU9150のi2cアドレスは0x68か0x69である)

デフォルトでは, 管理者(root)だけにしかデバイスファイル/dev/i2c-\*に読み書きの許可が与えられていない。そのため, i2dデバイスファイルをアクセスするためには, root権限でプログラムを実行する必要がある。

デバイスファイルをすべてのユーザーがアクセスできる許可を与えるためにには,

```
# chmod a+rwx /dev/i2c-2
```

を実行する。これで, /dev/i2c-2に接続されたi2cデバイスをroot以外のユーザー権限プログラムから読み書きする準備が整った。

### i2c デバイスへのデータの読み書き

MPU9150 というセンサーから加速度値を読み取る場合を例に、 i2c デバイスのアクセス（データ読み書き）の方法を説明する。

先述のように、 BBB の i2c 用の PIN にセンサーを配線すると、 Cape Manager はそれに対応してデバイスファイルを有効にする。 i2c という通信規格は、ひとつの PIN セットに複数の i2c デバイスを接続できる仕様となっている。したがって、デバイスファイルを指定しても、ひとつのセンサーを指定したことにはならない。それぞれのデバイスを区別するのが、アドレスである。上の例では、0x68 であった。接続されている i2c デバイスが 1 つであっても、アドレスは指定しなくてはならない。

さらに、 i2c デバイスは、レジスタというデータ格納領域をもっている。 MPU9150 の例でいうと、加速度センサー値やジャイロセンサー値などが、それぞれ別々のレジスタに格納されている。つまり、データを読み書きするためには、そのレジスタも指定する必要がある。

まとめると、ひとつのデータを i2c デバイスから読み取るためには、デバイスファイル、アドレスおよびレジスタの 3 つが適切に指定される必要がある。

デバイスファイルを開くためには、ファイルディスクリプタでそれを開く。C 言語によってこれを行う例を以下に示す。

```
#include <fcntl.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
...
#define I2CDEVICE "/dev/i2c-2"
...
int fd;
fd=open(I2CDEVICE, O_RDWR);
```

i2c-dev.h と ioctl.h は fd を開くためには必ずしも必要ないが、以下の i2c データのアクセスの中で必要となるので、まとめて示した。fd はアクセスが終了したら close(fd) を実行して閉じなければならない。

つぎに、アクセスしたいデバイスのアドレスは、 ioctl 関数を用いて指定する。

```
#define ADD 0x68
...
ioctl(fd, I2C_SLAVE, ADD);
```

MPU9150 はつねに i2c 通信において slave モードで用いる。これにより BBB 側が master モードとなる。これを一度だけ実行することにより、以下では fd に read, write する際に、アドレス 0x68 のデバイスがアクセスされる。逆に、1 つの i2c インターフェイス（ファイルディスクリプタ）に接続された別のデバイスをアクセスしたい場合には、ioctl 関数でそのアドレスを再設定すればよいことになる。デバイスのアドレスは i2cdetect コマンドで確認することもできるが、デバイスの仕様書などにも記述されているので確認してみるとよい。

ファイルディスクリプタ fd で i2c インターフェイスの open, ioctl 関数で、そのアドレスの指定ができた。最後にデータの読み書きである。read および write 関数をつかう。MPU9150 の場合、レジスタ 0x75 に、MPU9150 自身のアドレス 0x68 が格納されているので、そのレジスタを読み込む例を以下に示す。レジスタを指定してそこからデータを読み込むには、いったんレジスタアドレスそのものを write する。その後 read 関数を実行すると、そのレジスタの値が読み込まれる。

```
unsigned char wbuf[2];
unsigned char rbuf[14];
...
wbuf[0]=0x75;
write(fd,wbuf,1); // 1 バイト分書き込み（レジスタを指定）
read(fd,rbuf,1); // 1 バイト分読み込み（レジスタから値読み込み）
printf("WhoAmI=0x%02x\n",rbuf[0]);
```

WhoAmI=0x68 と標準出力されれば正常に接続されていることが確認できる。配列変数はサイズが 1 バイトの unsigned char 型を用いる。またそのサイズは、ここでは 1 以上であれば十分であるが、後の使用を考えて、2, 14 としている。

つぎにセンサーデバイスを初期化する。MPU9150 の場合、レジスタ 0x6b に 0x00 を、またレジスタ 0x37 に 0x02 を書き込むことによってリ

セットされセンサー値がレジスタにセットされ始めるので、以下を行う必要がある。

```
wbuf[0]=0x6b; wbuf[1]=0x00; write(fd,wbuf,2);  
wbuf[0]=0x37; wbuf[1]=0x02; write(fd,wbuf,2);
```

ここで示したように、書き込みはレジスタの指定と、その値の2バイトをいちどにwriteできる。

加速度センサー値はレジスタ 0x3b から 2 バイトづつ x,y,z の値が格納されているので、それを読み出す例を以下に示す。

```
wbuf[0]=0x3b; write(fd,wbuf,1); read(fd,rbuf,6);  
for(i=0;i<6;i++){ printf("0x%02x ",rbuf[i]); } printf("\n");
```

ジャイロセンサーの値は、この後ろの6バイトに格納されているので、一気に読みだすのであれば、read(fd,rbuf,12) を実行すれば良い。

それぞれの値が上位1バイト、下位1バイトの順で格納されているので、

```
int ax;  
...  
ax=rbuf[0]*256+rbuf[1];  
if(ax>32768){ax=65536-ax;}else{ax=-ax;}
```

というように、角度に対応する値に変換する。

0x3b 以下の 14 バイトのレジスタは随時センサー値が更新格納されるので、再び読みこむことによって現在のセンサー値を知ることが出来る。

## バッファのfsync

i2c デバイスのレジスタに値を書き込んで、その直後に同じレジスタを読みこめば書き込んだ値がそのまま読み取られるはずである。たとえば、加速度センサー値のノイズを平滑化する場合など、ローパスフィルターのレベルをレジスタに書き込んで指定することが出来る。指定したフィルターの強さを確認するためには、その値を再度読みこめばよい。

しかし、それを実行しても異なる値が得られる場合がある。つまり、正

常にレジスタに値が書き込まれていないか、あるいは書き込みが終了していないのである。

これは write 関数による書き込みが、その実行と同時に起こるのではなく、Linux kernel などのバッファに一旦蓄えられ、kernel の判断で実際のレジスタに値が書き込まれることが原因である。この様にバッファを介する方が、即座に書き込みを行うよりも、より効率的に読み書きを行うためである。

即座に書き込みを反映させたい場合には fsync(ファイル記述子) を実行する。これを実行することにより、kernel のバッファ内のデータがデバイスのレジスタに実際に write される。

### 8.1.8 超音波センサー値を i2c で読み取る

srf02 という超音波センサーを用いると、16cm ~ 600cm の範囲で対象物（壁など）からの距離を測定することができる。srf02 からはシリアル通信と i2c 通信によって距離データを読み出すことができるが、ここでは i2c を使った方法を紹介する。

i2c では一本のバス上に 7bit のアドレスで区別されるセンサーデバイスを接続する。つまり、理論上  $2^7 = 128$  個のデバイスが 1 個の i2c ポートに接続可能である。シリアル通信を使うばあい、接続できるセンサーの数はボードコンピュータのシリアルポートの数に制限されてしまう。Beaglebone Black の場合シリアルポートの数は 6 個である。多くのセンサーを接続したい場合には i2c 通信のほうが、大きな可能性を持っているといえる。

i2c プロトコルにおける、アドレスは 8bit で表現されているが、最下位のビットは書き込みか読み込みを区別するために利用されているので、実際のアドレスとして機能するのは、7bit である。srf02 のデータシートを読むと、デフォルトのアドレスは  $(E0)_{16} = (1110\ 0000)_2$  と記述されており 8bit 表記されていることが分かる。srf02 ではアドレスは  $(E0)_{16}, (E2)_{16}, \dots, (FE)_{16}$  まで 16 個のアドレスを選択することが出来るので、1

つの i2c ポートに 16 個まで srf02 を接続可能である。これらのアドレスはすべて偶数である。2 進数で表した場合、最下位のビットは常に 0 であることに注意が必要である。つまり、実際にアドレスとして機能しているのは上位の 7 bit ということである。

linux shell 上で、i2cdetect -r -y コマンドを実行すると、アドレス  $(E0)_{16}$  ではなく、 $(70)_{16}$  が表示される。 $(70)_{16} = (1110\ 000)_2$  であるので、8bit アドレスの使われていない最下位ビットが省略されていることが分かる。linux 内では i2c デバイスのアドレスは 7bit で管理されているようである。このように、マニュアルやデータシートに記載されているアドレスと、linux 上で認識されるアドレスが異なるので、混乱を来さないように注意が必要である。たとえば、 $(FE)_{16}$  とアドレスを変更した場合、i2cdetect コマンドや C 言語プログラムからそのデバイスをアクセスする場合、それを 2 で割った  $(7F)_{16}$  というアドレスを使う必要がある。

もうひとつ、linux shell 上で i2cdetect をする場合に注意する必要がある。デフォルトでは i2cdetect コマンドは  $(03)_{16}, \dots, (77)_{16}$  までしか表示しない。 $(00)_{16}$  から  $(FF)_{16}$  まですべてのアドレスを認識表示するために i2cdetect -r -y -a と -a オプションを付加する必要がある。

## SRF02 のアドレス変更

SRF02 のアドレスを変更するためには、コマンドバッファ  $(00)_{16}$  に  $(A0)_{16}, (AA)_{16}, (A5)_{16}$  につづけて、変更後のアドレスを書き込む。

C 言語プログラムを用いて実際にアドレス変更した例を以下に示す。書き込む先のアドレスは Linux の ioctl を介して行うので、7 ビット表記を用いる。これに対して、書き込むアドレスは、srf02 が保持する値なので 8 ビット表記でなければならないことに注意する必要がある。

```
1 #include <stdio.h>
2 #include <linux/i2c-dev.h>
3 #include <fcntl.h>
4 #include <sys/ioctl.h>
5
6 #define SRF02_DEV "/dev/i2c-1"
7 #define SRF02_ADD 0x71
```

```

8  #define CHANGE_TO_ADD 0xfe
9
10
11 int main() {
12     int fd;
13     unsigned char wbuf[3];
14
15     fd=open(SRF02_DEV, O_RDWR);
16
17     // 0x00に0xA0, 0xAA, 0xA5をつづけて書き込む
18     ioctl(fd, I2C_SLAVE, SRF02_ADD);
19     wbuf[0]=0x00; wbuf[1]=0xA0; write(fd,wbuf,2);
20
21     ioctl(fd, I2C_SLAVE, SRF02_ADD);
22     wbuf[0]=0x00; wbuf[1]=0xAA; write(fd,wbuf,2);
23
24     ioctl(fd, I2C_SLAVE, SRF02_ADD);
25     wbuf[0]=0x00; wbuf[1]=0xA5; write(fd,wbuf,2);
26
27     // 0X00に新しいアドレス値を書き込む
28     ioctl(fd, I2C_SLAVE, SRF02_ADD);
29     wbuf[0]=0x00; wbuf[1]=CHANGE_TO_ADD; write(fd,wbuf,2);
30
31     close(fd);
32 }
```

この例では、アドレスが7ビット表記( $71_{16}$ )から8ビット表記( $fe_{16}$ )に変更されている。

### 8.1.9 BBB の電源

BBB の電源は5VのACアダプターから、あるいは、miniUSBポートから供給できる。長時間、プログラムのデバッグなどを行う場合には、これらの電源をもちいるとよい。いっぽう、走行ロボットや飛行ロボットでは、これらの電源が利用できないので、バッテリーからレギュレータ(DC-DCコンバータ)を通して5V電源を確保する必要がある。

BBB側の配線例を図8.4に示した。赤い線が正(+)側、黒い線が負(-)側のリード線である。BBBも含めた直流電源を用いる電子機器は、電源の正負を間違うと破損するので、注意が必要である。

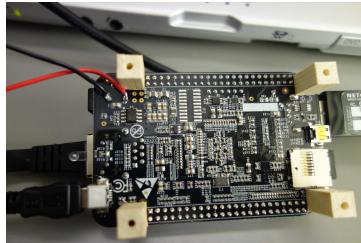


図 8.4: BeagleBone Black の 5V 電源をレギュレータなどからとるための配線

### 8.1.10 PWM でモーターを制御する

感覚運動写像などにおいて、運動を調節するためにはモーターの回転数を出力値に応じて制御しなければならない。モーターの制御は BBB の PWM 機能を用いて行うことが可能である。PWM(Pulse Width Modulation)とは、電圧パルスの電圧の高さを変化させるのではなく、パルスの幅を大きくしたり小さくしたりすることによってモーターの回転数を変化させる方法一般を指す。

ブラシレスモーターの場合、モーターは専用のアンプによって回転数が制御される。アンプに対して PWM パルスを BBB から出力することによって、ブラシレスモーターの回転数を制御できる。

ここでは BBB の PWM 機能を使って、ブラシレスモーターの回転数を制御する方法を説明する [15]。BBB の PWM 機能は、Cape Manager に実装されたデバイスドライバを通じて実現される。具体的には、PWM 制御用の仮想ファイルに値を書き込むことによってパルス幅などを変化させることが出来る。

仮想ファイルへの値の書き込みは、Linux の echo コマンドを使って行うことも可能であるが、ここでは感覚運動写像などのプログラムから PWM 制御を行うことを念頭に、C 言語でその仮想ファイルに値を書き込む方法を説明する。

## BBB による PWM の概略

仮想デバイスファイルへの書き込みなどの機能は参考文献[15]に示されたライブラリ libBBB を用いておこなう。そのため、まず libBBB をインストールしなければならない。この作業は BBB 上の Debian で一度だけ行えば良い。Debian の起動のたびに行う必要はない。

つぎに、Cape Manager を通じて PWM ドライバを有効にする必要がある。これは Debian が再起動されると無効になるので、起動のたびに行う必要がある。

さいごに仮想ファイルへの値の書き込みによってパルスを制御する。ここでは前述のように libBBB を利用して C 言語で PWM 制御を行う例を示す。

## libBBB のインストール

libBBB ライブラリをインストールするためには、make コマンドと gcc をもちいるので、まだインストールされていない場合には、インストールする。

```
# aptitude install make
# aptitude install gcc-4.7
# ln -s /usr/bin/gcc-4.7 /usr/bin/gcc
```

次に libBBB をインストールする。libBBB は参考文献[15]にあるものを自分で入力してもよいが、<http://www.rutles.net/download/393/index.html> からダウンロードすることも出来る。make は libBBB というディレクトリ内でおこなう。ルートディレクトリ(/)下にそれをつくると、Debian のディレクトリ構造が煩雑化するので、作業ディレクトリを作成し、その下で make を行う。

```
# mkdir /home/hogehoge/work
# cd /home/hogehoge/work
... libBBB.tar.gz のダウンロード
# tar xzpvf libBBB.tar.gz
# cd libBBB
# make clean
# make
# make install
```

### Cape Manager による仮想デバイスファイルの生成

以下のコマンドを実行する。

```
# echo am33xx_pwm > /sys/devices/bone_capemgr.*/slots
```

```
# gcc pwm_test.c -lBBB -o pwm_test
```



# 第9章 無線LAN

Debian OS の間の無線ネットワーク通信は、他の UNIX/Linux OS と同様に TCP/IP に基づいたものである。無線 LAN を利用するには、大きく分けてハードウェアの認識、アドレスやキー認証の設定、有線 LAN からのルーティングの変更が必要である。

## 9.1 ツールのインストール

apt-get あるいは aptitude が使える状態であれば、それらを使って wireless-tools パッケージをインストールする。

```
# aptitude install wireless-tools
```

なお、LEGO EV3 の Native Linux では aptitude などが利用できないので、iwconfig と iwlist コマンドのソースコードをダウンロードし、コンパイルしてそれらを作成する必要がある。

ev3dev は Debian ベースの LEGO OS なので、通常の Debian と同じようにインストールや設定が可能である。

## 9.2 設定コマンド

無線 LAN 接続を実現するためには、カーネル・モジュールによるインターフェイスの認識、モード、ESSID、チャンネルおよび暗号化パスワードの設定、さらに IP アドレスの設定がそれぞれ必要である。

インターフェイスの認識は、各ハードウェアに依存する部分もあるので、それぞれのボードコンピュータなどの節を参照してほしい。

ここでは、モード、ESSID、チャンネルなどについて主に説明する。

## モード

無線 LAN には ad-hoc, managed および master の 3 つのモードがある。ad-hoc モードは、2 台の無線 LAN の間で 1 対 1 の通信をするばあいに利用する。

master モードと managed モードを用いると、2 台の間の通信はもちろん可能であるが、3 台以上の環境でも無線 LAN 通信が可能である。ルーターあるいは、アクセスポイントになる無線 LAN を master モードにするが、master モードすなわちアクセスポイントに対応した無線 LAN ハードウェアを用いる必要がある。Armadillo-300 および Armadillo-420+AWL13 は master モードになることができる。

master モードのアクセスポイントにアクセスする子機は managed モードに設定する。一般に 3 台以上で master-managed モードで無線 LAN 環境を利用するよりも、ad-hoc モードで 1 対 1 通信するほうが通信速度が速くなる傾向がある。ロボットと Debian PC との通信など、速度が要求される通信の場合にはどのモードを選択するか考慮が必要である。

以下にインターフェイス wlan0 を ad-hoc モードにする例を示す。

```
# iwconfig wlan0 mode ad-hoc
```

すでに IP アドレスが割り当てられているインターフェイスについてはモードの切り替えができないので、いったんそれを down する必要がある。

```
# ifconfig wlan0 down  
# iwconfig wlan0 mode ad-hoc
```

## IP アドレス

### 1. wlan0 の手動起動例

```
# ifconfig wlan0 172.6.10.xxx
```

### 2. /etc/network/interfaces を使って無線 LAN インターフェイスを起動

```
# ifconfig wlan0 up
```

なお、wlan0 は ath0 など、デバイスに割り当てられるインターフェイス名に依存する。また、wlan0 を停止するには、

```
# ifconfig wlan0 down
```

### 3. アクセスポイントをスキャン

```
# iwlist wlan0 scan | more
```

| more は、長い行数にわたる表示を、1 ページづつに分けて表示することを指定する。スペースキーをおすと、次のページが表示される。たとえば、

```
Cell 02 - Address: 00:80:92:3A:9F:E1
  Channel:2
  Frequency:2.417 GHz (Channel 2)
  Quality=58/70  Signal level=-52 dBm
  Encryption key:off
  ESSID:"arm18"
  Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 6 Mb/s; 9 Mb/s
             11 Mb/s; 12 Mb/s; 18 Mb/s
  Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
  Mode:Master
  ....
```

のような表示が現れれば、無線 LAN インターフェイスは、正常に電波をキャッチしている状態である。6 行目の表示から、ESSID が

arm18 であることが分かる。アクセスしたいルーターに合わせて、以下の essid を指定する。

### チャンネル, ESSID

#### 1. チャンネルの指定

利用する周波数帯を channel を用いて指定する。iwlist コマンドで表示された channel となるだけ重複しないチャンネルを利用する。3 チャンネル以上離れたチャンネルを利用することが推奨されている。

```
# iwconfig wlan0 channel 2
```

#### 2. essid を指定。

```
# iwconfig wlan0 essid arm18
```

### デフォルトルーターの指定

たとえば、arm18w というマシンをルーターに選ぶ場合、詳細は、下記 9.4 を参照すること。

```
# route add default gw arm18w
```

## 9.3 シェルスクリプト

前節で説明したコマンドを実行すれば、アクセスポイントに無線 LAN 接続される。しかし、電波状況が悪い場合等、1回の実行で接続されない場合もある。何度か実行すれば接続されるが、煩雑さは免れない。接続されるまで、つまり、Access Point: にアクセスポイントの MAC アドレ

スが表示されるまで、これらのコマンドを繰り返すスクリプトを以下に示す。

root になって、以下のシェルスクリプトを実行する。このシェルスクリプトは、無線 LAN 接続が確立すれば、自動的に停止する。最大繰り返し回数は、変数 MAX に指定されている。

```
#!/bin/sh
# (c) 2013.12.3 Y.Honda
# name of wireless interface
INTF="wlan0"
# essid for the interface
ESSID="arm18"
# router host name
ROUTER="arm18w"
# mac address of access point
MAC="00:80:92:3A:9F:E1"
# IP address for the interface
ADDR="172.16.10.XX"
# period between commands
PROD="3s"
# max number of iteration
MAX=5

ifconfig $INTF down
echo "mode"
iwconfig $INTF mode managed
sleep $PROD

AP=`iwconfig | grep Mode`
echo $AP
APP=`echo $AP | sed -e 's/ //g' `
CNT=0
while [ `echo $APP |grep Not-Associated` ]
do
    if test $CNT -gt $MAX
    then
        break
    fi

    CNT=`expr $CNT + 1`
    echo "CNT=$CNT"

    ifconfig $INTF $ADDR
    sleep $PROD

    echo "channel"
    iwconfig $INTF channel 2
    sleep $PROD

    echo "essid"
    iwconfig $INTF essid $ESSID
    sleep $PROD

    echo "ap"
    iwconfig $INTF ap $MAC
    sleep $PROD

    AP=`iwconfig | grep Access`
    echo $AP
    APP=`echo $AP | sed -e 's/ //g' `
done
route add default gw $ROUTER $INTF
```

iwconfig コマンドを実行して、以下に示したように、Access Point の MAC アドレスが表示されれば、正常接続が完了したことを示す。

```
eth0      no wireless extensions.

lo       no wireless extensions.

wlan0    IEEE 802.11abgn  ESSID:"arm18"
          Mode:Managed  Frequency:2.417 GHz  Access Point: 00:80:92:3A:9F:E1
          Bit Rate=54 Mb/s  Tx-Power=15 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=58/70  Signal level=-52 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Rx excessive retries:0  Invalid misc:2016  Missed beacon:0
```

## 9.4 経路の追加

ネットワークのアクセス経路の表示や追加は route コマンドを使って行う。

```
# route
```

例えば、以下の様に表示される。

カーネル IP 経路テーブル						
受信先サイト	ゲートウェイ	ネットマスク	フラグ	Metric	Ref	使用数 インタフェース
default	arm18w	0.0.0.0	UG	0	0	0 wlan0
localnet	*	255.255.0.0	U	0	0	0 wlan0

localnet 以外は、すべて wlan0 を通してゲートウェイ arm18w をアクセスする設定になっている。

経路を追加・削除するためには、経路情報うち受信先サイト、ゲートウェイ、メトリック、ネットマスク、インターフェイスを指定する。ルートを追加する場合、

```
# route add -net 受信先サイト gw ゲートウェイ metric メトリック netmask ネットマスク インタフェイス
```

という書式となる。ゲートウェイを省略するとゲートウェイは”\*”となる。受信先サイトには、ネットワークアドレスや、localnet を指定する。ゲー

トウェイには、ホスト名や IP アドレスを指定する。メトリックと言うのは、仮想的な距離を意味し、同じ受信先サイトが複数登録されている場合、メトリックが相対的に小さい方の経路が選択される。ネットマスクは、IP アドレスの中でどこまでネットワークアドレスとして認識し、どこをホストアドレスとして認識するかを指定する。たとえば、255.255.255.0 の場合、最初の 3 バイトがネットワークアドレス、さいごの 1 バイトがホストアドレスを意味することになる。

ネットワーク 172.16.10.xxx をゲートウェイ arm18w を通じてアクセスするには、route コマンドで以下の様に経路を追加する。

```
# route add -net 172.16.10.0 gw arm18w metric 0 netmask 255.255.255.0 \
wlan0
```

route コマンドで、経路を再表示すると、

```
172.16.10.0      arm18w      255.255.255.0    UG        0      0      0      wlan0
```

という行が追加されて表示される。

## 9.5 トラブルシューティング

### 9.5.1 SIOCSIFFLAGS: Operation not possible due to RF-kill

と表示されて、ifconfig などが実行できない場合。

```
# rfkill list
```

を実行して、Wireless LAN の状態を調べます。下記の様に blocked: no となっていれば使える状態です。

```
0: phy0: Wireless LAN
    Soft blocked: no
    Hard blocked: no
3: hci0: Bluetooth
    Soft blocked: no
    Hard blocked: no
```

一方、yes の場合には

```
# rfkill unblock wifi
# rfkill unblock all
```

などとして、ブロックを解除します。Hard blocked: yes の場合には Wireless のスイッチが切れているので、スイッチを入れる必要があります。

### 9.5.2 Network-Manager を使うと DNS が破壊される

GNOME などで Network-Manager を利用すると、無線 LAN の設定などが GUI を用いて切り替えたりできるので便利である。しかし、Network-



図 9.1: GNOME のタスクバーに表示された Network-Manager

Manager は、/etc/network/interfaces と矛盾した IP アドレスなどを使う場合があるので、注意が必要である。

また DNS として Network-Manager 内の情報を使うため、/etc/resolv.conf の情報が削除されることがある。その場合には、resolv.conf を元に戻さなければならない。



# 第10章 並列計算ライブラリ PVM

Paralell Virtual Machine(PVM) は、もともと大規模な学術計算を並列計算するために大学や研究機関が協力して開発したライブラリ群です。その経緯から、Debian と同じくオープンでフリーな存在として定着しており、Debian パッケージとしてもインストールすることができます。

PVM は、TCP/IP などのネットワークの専門知識を必要とせず C 言語から関数を呼び出す形で利用することができます。また、子プロセスからさらに自由にプロセスを生成することができるなど柔軟性にも富んでいます。

ロボット知能の研究では、環境の中で行動するロボットの状態を把握したりロボットに人間側の意志を伝えたりする必要が生じます。それらのデータ通信に PVM を利用することができます。

また、数多くのロボットが協調行動をする場合には、それぞれのロボットを制御するプログラムをそれぞれのロボット自身のなかで実行する必要があります。PVM のもつ spawn 機能を利用すれば、それが簡単に実現できます。

いくつかの pvm 関数を用いることで簡単にデータ通信が可能となること以外に、通信速度が比較的高速に実行出来るというメリットもあります。そのためロボットと、時間的な遅れなしに、リアルタイムにデータ通信することも可能になります。

PVM ライブラリは、非常に多くの機能と関数を持っていますが、ここではデータの送受信を行うための基本的な関数だけを取り上げます。

## 10.1 インストールと開始

PVM を Debian パッケージを利用することで、インストールする方法を説明します。また、PVM は実際にプログラムを実行する前に、PVM に参加するマシンを登録しておく必要があるので、その方法を示します。

### 10.1.1 apt-get によるインストール

PVM に関するパッケージ pvm, libpvm3, pvm-dev をインストールします。

```
# apt-get install pvm
# apt-get install libpvm3
# apt-get install pvm-dev
```

dpkg コマンドを用いて個別にパッケージファイルをインストールする場合と異なり、依存関係を気にする必要がありません。

### 10.1.2 dpkg を用いたインストール

なんらかの理由で、apt-get コマンドによってインストールできない場合には、dpkg コマンドをもちいてパッケージファイルから直接インストールします。以下の 6 つのパッケージをコピーあるいはダウンロードしてインストールします。

1. libtinfo-dev
2. libreadline6-dev
3. libreadline-dev
4. pvm
5. libpvm3

## 6. pvm-dev

具体的な `dpkg` コマンドの使い方は以下の通りです。

```
# dpkg -i 「パッケージ名」.deb
```

「パッケージ名」の部分には上記パッケージ名と、バージョン番号が入ります。

`apt-get` コマンドによるインストールの場合と異なり、`dpkg` コマンドはパッケージの依存関係を自動解決してくれないので、インストールの順序に注意する必要があります。

### 10.1.3 PVM を開始する

PVM の起動と終了、ホストの追加、ホストの確認の方法などについてそれぞれ説明します。

#### 起動と終了

`pvm` コマンドを実行すると、`pvm` プロンプトが起動します。

```
% pvm
```

また、`hosts` ファイルを指定して起動すると、`pvm` に参加するホストを自動的に読み込むことができます。

```
% pvm hosts
```

`hosts` というのは、`pvm` でプロセスを起動するホストの名前をリストアップしたファイル名のことです。べつに名前は `hosts` でなければならぬ訳ではありません。たとえば以下のような内容のファイルであればファイル名は自由です。

```
host1  
host2
```

ここに指定するリモートホストでは ssh が使用できるように設定されていなければなりません。パスワードを聞いてくるので、それに答えます。

```
pvm>
```

というプロンプトが表示されれば正常に始動しています。ssh の認証プロセスを自動化しておけば、パスワード入力の手間を省けます。

```
pvm>quit
```

と入力することによって、pvm プロンプトから抜け出せます。PVM 自体は終了していません。バックグラウンドで pvmd が動いている状態です。

バックグラウンドもふくめて pvmd を完全に止めるには

```
pvm>halt
```

と入力します。

ここで注意することは、/etc/hosts ファイルに記述してある hostname と IP address の値が、実際のホスト名と IP アドレスと必ず一致していなければなりません。PVM はそれらの矛盾を検知するとエラーを生じて起動されません。

## ホストの追加

PVM を開始してから、新たにホスト (host3) を追加することも出来ます。

```
pvm>add host3
```

ssh の場合と同じ、host3 のパスワードが必要となります。ssh で login できる状態になっている必要があります。すなわち、host3 にユーザー登録がされている必要があります。

### ホストの確認

現在 PVM で利用可能なホストの一覧を得るには, conf コマンドを使います.

```
pvm>conf
```

たとえば、下記のような結果になれば、host1 と host2 という 2 つのホストが PVM に参加していることを確認できます。

```
pvm> conf
conf
2 hosts, 2 data formats
      HOST      DTID      ARCH      SPEED      DSIG
    host1      40000  LINUX64      1000  0x00408c41
    host2      80000  LINUXARM     1000  0x00408841
```

### 実行しているプロセスの確認と停止

ps コマンドを用いて書くホストが実行中のプロセスを表示することができます。a オプションですべてのプロセスを表示できます。

```
pvm> ps a
ps a
      HOST      TID      FLAG 0x COMMAND
    host1      40002        4/c -
    host2      80001      6/c,f /tmp/2dovr
    host2      80002      6/c,f /tmp/pixy
    host2      80003      6/c,f /tmp/feeler
```

たとえば、このような結果になった場合、/tmp/2dovr,/tmp/pixy,/tmp/feeler というプロセスが host2 上で実行されています。これらは、pvm\_spawn 関数を用いて生成されたプロセスであるため、その名前が明示されています。

一方、host1 上で実行されているプロセスは、その名前が明示されておらず、- で表されています。これは、host1 のプロンプトで実行したプログラムを表しています。

想定外のプロセスが実行中になっている場合、期待通りのデータ通信が行われないなど、障害が生じるので、以下の reset コマンドを用いて、すべてのプロセスを停止させてから新たなプロセス生成を行うようにします。

```
pvm>reset
```

個別のプロセスを停止させたい場合には、kill コマンドで TID を指定します。

```
pvm>kill 80006
```

この例のように、たとえば 80006 を指定した場合、ps a コマンドで実行中のプロセスを確認すると、

```
pvm> ps a  
ps a  
HOST      TID   FLAG 0x COMMAND  
rock108   40003   4/c -  
bbb140w   80004   6/c,f /tmp/2dovr  
bbb140w   80005   6/c,f /tmp/pixy
```

たしかに、/tmp/feeler というプロセスは停止したことが確認できます。

それ以外のコマンドは

```
pvm>help
```

で調べることができます。

## 10.2 C 言語で PVM を利用する

ここからは、C 言語を用いて pvm を使うための方法を説明します。前節で説明したとおり、PVM が起動している状態で、プログラムの中から pvm 関数を呼び出します。

### 10.2.1 ライブラリのインクルード

プログラムで PVM ライブラリ (API) を利用するためには、 pvm3.h の include が必要です。ソースコードの最初の部分で、

```
#include <pvm3.h>
```

とします。

また、コンパイルの際には、pvm3 ライブラリを指定します。たとえば、hoge.c という C 言語プログラムを作成し、それをコンパイルして hoge という実行ファイルを作成する場合の例は、以下のようになります。

```
% gcc hoge.c -lpvm3 -o hoge
```

### 10.2.2 構成情報の取得

PVM は、複数のマシンを一つのバーチャルマシンとして使うためのライブラリです。その構成状況をプログラム内から調べることによって、それに応じたプロセスの生成などが可能となります。

#### 構成を返す関数

pvm\_config 関数を使います。

```
int info=pvm_config(int *nhost,  
                     int *narch,  
                     struct pvmhostinfo **hostp)
```

それぞれの、引数の意味は以下の通りです。

- nhost

バーチャルマシンにおけるホスト数。

- narch

現在使用中のデータフォーマットの種類。

- hostp

ホスト情報を保持する構造体の配列へのポインタ。この構造体 pvmhostinfo は pvm3.h の中に以下の様に定義されています。

```
struct pvmhostinfo {
    int hi_tid;      /* pvmd tid */
    char *hi_name;  /* host name */
    char *hi_arch;  /* host arch */
    int hi_speed;   /* cpu relative speed */
    int hi_dsig;    /* data signature */
};
```

構造体のそれぞれのメンバーの意味は

- hi\_tid: タスク ID
- hi\_name: 名前
- hi\_arch: アーキテクチャ
- hi\_speed: 相対的速度
- hi\_dsig: 最大パケット長

です。

### プログラム例

PVM の構成情報を調べて、コンソールにそれを表示する部分は以下の例のようになります。

```
...
int info;
int nhost, narch, infos;
struct pvmhostinfo *hostp;

info = pvm_config(&nhost, &narch, &hostp);

printf("nhost=%d \n", nhost);
printf("narch=%d \n", narch);
for(i=0;i < nhost;i++)
    printf("host name[%d]=%s \n", i, hostp[i].hi_name);
...
```

### 10.2.3 プロセスの生成 (spawn)

PVM では、どのプロセスからでも、別のプロセスを生成できます。 pvm\_spawn 関数を使います。英語の `spawn` は、「産卵する」という意味なので、イメージしやすいですね。

具体例を以下に示します。

```
pvm_spawn(PROGRAM,      // 実行ファイルの名前
          (char**)0,    // PROGRAM に渡す引数
          0,           // flag=0:すべてのホストに。flag=1:指定したホストに
          "",          // where, プロセスを生成するホスト。flag=0 の場合は ""
          nprocess,    // 起動する slave の数
          tids);       // 生成されたタスクの ID (配列) が返される
```

`pvm_spawn` 関数が成功すると、`tids` という 1 次元配列に、`spawn` したタスク ID が配列として返って来ます。後ほど、`pvm_send` でデータを送るプロセスを指定する場合に、そこに格納された ID でプロセスを指定します。

#### ホストマシンの指定

このなかで、`flag` の値によってプロセスを生成するホストマシンが決まります。`flag=0` の場合、PVM が全ホストにたいして、自動的にプロセス生成します。`flag=0` の場合には、`where` の値は無視されます。

たとえば、上の例のように、`flag` の値を 0 として、`nprocess` の値を `pvm_config` で調べたホストの数に等しくしておくと、PVM は自動的に各ホストで `PROGRAM` を実行し始めます。

`flag=1` の場合には、次の引数 `where` に指定されたホストでプロセスが生成されます。`where` には、ホスト名を指定します。DNS を用いない場合には、ホスト名は、`/etc/hosts` ファイルに記述されていなければなりません。

### spawn される実行ファイル

spawn される実行ファイルの名前は、PROGRAM で指定します。デフォルトでは、PVM は環境変数 PVM\_ROOT に指定されたディレクトリの元にある実行ファイルを探しに行きます。

PVM\_ROOT に実行プログラムを毎回置くこともできますが、PROGRAM の値として、ディレクトリの絶対パスとファイル名を指定することもできます。

```
#define PROGRAM "/home/taro/slave"
#define N 10
...
int tids[N];
nhost=1;
...
pvm_spawn(PROGRAM, (char**) 0, 1, "host2", nhosts, tids);
...
```

この例では、タスクの数として 10 と固定したので、PVM の構成に加えるホスト数がそれを超えないように注意する必要があります。

flag=1 として、プロセス起動ホストを "host2" と指定し、nhost=1 ので、ひとつだけ /home/taro/slave というプロセスが生成されます。

tids 配列には spawn されたタスクのタスク ID が格納されます。それらのタスクにデータを送る際には、この tids が必要となります。

spawn される実行ファイル、上の例では /home/taro/slave というファイルは spawn をしようとするホスト上に存在し、実行可能な状態でなければなりません。たとえば pvm\_spawn 関数で指定する flag が 0 の場合には、pvm に参加しているすべてのホストでそのプログラムが実行されるので、そのすべてのホストの /home/taro/slave が実行可能な状態にある必要があります。

spawn するホストを指定する場合は、その指定したホストに実行ファイルが存在しなければなりません。

### 10.2.4 標準出力

pvm\_spawn で生成された slave プロセスからの標準出力は、 slave 側の pvm を通して、 master の /tmp/pvml.[uid] の中に書き込まれます。しかし、それぞれのホストの pvm どうしが通信する際に、標準入出力用のバッファの内容が吐き出されて (flush) いなければ、出力はうまく /tmp/pvml.[uid] に反映されない場合があります。

これを解決するためには、 slave 側のプログラムの先頭付近で

```
setlinebuf (stdout);
```

とバッファリングモードを指定しておけば、 printf を実行するたびに、 バッファの内容が吐き出され、 標準出力は /tmp/pvml.[uid] に正確に反映されます。

### 10.2.5 データの送信と受信

PVM を使う場合に限らず、 並列計算・データ処理において、 中心的な役割を果たし、 もっとも頻繁に使用される機能は、 データの送出と受信です。 ごく単純化して言えば、 並列処理とは、 全体の処理を小さなブロックに分割して、 それぞれを別々のマシンで処理し、 結果を集めて出力することと言えます。 あるいは、 次の処理にまわすということの繰り返しであると言えます。

分割して送り、 受け取って処理し、 その結果を送り返すという処理、 つまり、 データ送受信を繰り返すことになります。

PVM では、 配列をパケットにパックして送出し、 受け取り側がそのパケットをアンパック（配列に戻す）という形式でデータ送受信を実現します。 以下では、 具体例に沿って説明します。

## タスク ID(tid)

pvm ではデータの通信をタスク(プロセス)間で行います。そのためには、通信の相手と自分の同定(Identify)が必要です。それを tid と呼ばれる数値で行います。

自分自身のタスク ID を知るためには

```
int mytid;  
mytid=pvm_mytid();
```

という関数を使います。また、自分を生成(spawn)した tid を知るためには、

```
int ptid;  
ptid=pvm_parent();
```

という関数を使います。

parent 側にデータを送り返す場合など、この tid を指定します。

## TAG を指定してデータの送信

データを相手プロセスに送信するためには、`pvm_initsend` 関数、`pvm_pkXXXX` 関数、そして`\pvm_send` 関数をそれぞれ実行しなければなりません。

`pvm_initsend(int encoding)` 関数は、送信バッファをクリアし、データ(メッセージ)のパックの送信バッファを作ります。`encoding` の値はエンコーディング方法を指定する値ですが、デフォルト値 `encoding=0` を用います。この関数は、`pvm_pkXXXX` 関数を呼ぶ前に必ず実行します。そうしないと、送信バッファがクリアされず古いデータの次に、新しいデータが追加されてしまうため、アンパックした際に、正しくデータが読み取れない場合があります。

つぎに、`pvm_pkXXXX()` 関数で、データを送信バッファにパックします。`XXXX` の部分には、`int` や `double` が来ます。たとえば、倍精度浮

動小数点データを送信する場合、

```
pvm_pkdouble(double *data, int nitem, int stride)
```

と、3つの引数を指定します。パックするデータ (double data) は、変数そのものではなく、そのポインタ (&data) で指定します。配列変数の場合、配列名そのものを指定します。これは、受け取る際のアンパックにおいても同様です。pvm\_pk 関数の 2 番めの整数引数は、パックする配列のサイズ (nitem) を指定します。3 番目の引数は、その間隔 (stride) を指定します。配列を nitem だけすべてパックするばあいには、1 を指定します。たとえば、2 を指定すると、1 つおきにデータがパックされます。

pvm\_send 関数では、tids の指定以外に、通信 TAG を指定する必要があります。同じプロセス間で、種類の違う通信が行われる可能性があるので、それらを識別するためです。

下記に整数 int 型と char 型の配列を送信する例を示します。

```
#include <string.h>
#define TAG 1
...
char str[256];

strcpy(str,"最近、寒くなってきましたね");
int i_dum;
for(i=0;i < nprocess;i++) {
    pvm_initsend(0);           //送信バッファなどの初期化
    i_dum=strlen(str);
    pvm_pkint(&i_dum,1,1);    //整数（ポインタ）を1つパック
    pvm_pkbyte(str,strlen(str),1); //文字配列をその長さだけパック
    pvm_send(tids[i],TAG);     //タスク ID 番号 i のプロセスに送信.
}
```

initsend は、send を実行する前に毎回必ず必要なので、注意してください。

## 受信側の例

```
#define TAG 1
...
int ptid;
int my_tid;
ptid=pvm_parent();
mytid=pvm_mytid();
...
pvm_recv(ptid,TAG);           //親タスクからのパケットを受信.
pvm_upkint(&str_len,1,1);    //整数（ポインタ）をアンパック. (配列の長さ)
pvm_upkbyte(str,str_len,1); //文字配列をその長さだけアンパック
str[str_len]='\0';          //文字列末を代入
printf("%d:%s\n",mytid,str);
...
```

受信では、送信とは逆に受信、アンパックという順序でデータを取り出します。

`ptid` で指定されたプロセスから `recv` し、アンパックしてコンソールへ出力しています。コンソール上へ表示された文字は、PVM を開始したホストの `/tmp/pvml.XXXX` に出力されるので、`tail` コマンドなどで確認できます。`XXXX` は PVM を開始したユーザー ID です。

```
$ tail /tmp/pvml.XXXX
```

### 10.2.6 遅延のないデータ送受信を行うために

前節の例では、受信側のソースコードにおいて、`pvm_recv` 関数を用いました。この関数は、受信バッファにデータが到着していない場合には、受信バッファにデータが入るまでプロセスを停止します。つまり、プログラムはそこから先に進みません。このことをブロッキングと呼びます。データが到着するまで、一切の計算を停止させたい場合、すなわち、データの同期をとりたい場合には、これは便利な機能です。

いっぽう、データの同期をとる必要がなく、バッファにデータがない場合には次の計算などの処理に移りたい場合には、困ります。その場合

には、`pvm_nrecv` 関数を用います。

逆に、`pvm_recv` 関数あるいは、`pvm_nrecv` 関数が実行される前に、受信バッファーに大量のデータが蓄積された場合、`recv` 関数は古いデータから順番にデータを取り出します。`pvm` はバッファーにたまっているデータを捨てません。

センサーなどの値を連続的に受け取る場合、受け取り側のプログラムの実行速度が遅いと、送られてきたセンサーデータなどが受信バッファーに大量に蓄積します。この場合、`recv` 関数は古いデータから読み込んでいくので、リアルタイムなセンサー値が読み取れることになります。

これを解決する簡単な方法は、`send` 側のプロセスにおいて `sleep` 関数などを用いて、実行速度を落とします。`recv` 側の実行速度が十分に `send` 側の速度より早ければ、受信バッファーにはつねに最新のデータがたまされます。

### 10.2.7 `spawn` されたプログラムが正常終了しないで残っている場合

各プロセスが何らかの理由で正常に終了しなかった場合には、PVM をリセットしてください。

```
pvm> reset
```

を実行します。プロセスが残っているかどうかは、

```
pvm> ps a
```

で調べることができます。

### 10.2.8 整数型データの通信プログラム例

master 側

```

1 #include <stdio.h>
2 #include <pvm3.h>
3
4 #define PROGRAM "/root/sensor_test1"
5 #define SLAVE "rb143"
6 #define MAXNHOST 256
7
8 #define sTAG 1
9 #define rTAG 2
10
11 int main(){
12     int i;
13     int info;
14     int nhost, narch, infos;
15     int tids[MAXNHOST];
16     int dum;
17     int rdum;
18     struct pvmhostinfo *hostp;
19
20     info = pvm_config(&nhost, &narch, &hostp);
21
22     printf("nhost=%d \n",nhost);
23     printf("narch=%d \n",narch);
24     for(i=0;i<nhost;i++)
25         printf("host name[%d]=%s \n",i,hostp[i].hi_name);
26
27     // SLAVE 一つだけ spawnする.
28     pvm_spawn(PROGRAM, (char**)0, 1, SLAVE, 1,tids);
29     printf("tid:%d\n",tids[0]);
30
31     /*
32     dum=tids[0];
33     pvm_initsend(0); // 初期化は sendの前に必ず必要.
34     pvm_pkint(&dum,1,1);
35     pvm_send(tids[0],sTAG);
36
37     // SLAVEから送り返されてきたデータを受け取る.
38     pvm_recv(tids[0],rTAG);
39     pvm_upkint(&rdum,1,1);
40
41     printf("send:%d recv:%d\n",dum,rdum);
42     */
43 }

```

## slave 側

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define rTAG 1
5 #define sTAG 2
6
7 int main() {

```

```
8  int irecv;
9  int ptid;
10 int dum;
11
12 ptid=pvm_parent();
13
14 printf("こんにちは、 整数をrecvします. \n");
15
16 pvm_recv(ptid,rTAG);
17 pvm_upkint(&irecv,1,1);
18 printf("受け取った数字:%d",irecv);
19
20 dum=irecv;
21 pvm_initsend(0);
22 pvm_pkint(&dum,1,1);
23 pvm_send(ptid,sTAG);
24
25 }
```

## 10.3 トラブルシューティング

PVM の開始やプログラムのコンパイルなどの際によく起こる問題と解決法を簡単に示します。

1. 各マシンで、 /etc/hosts 内のホスト名と /etc/hostname で指定されるホスト名に矛盾が無いこと。矛盾があると、 PVM がフリーズする場合があります。
2. PVM の開始の際、 chmod a+rwx /tmp をしておくことが必要な場合があります。
3. sshd はインストールされている必要があります。環境変数で rsh を指定する場合にはリモートシェルが使える環境が整備されていなければなりません。
4. #include <pvm3.h> をするためには、 pvm-dev がインストールされている必要があります。



# 第11章 グラフ作成(GNUPLOT)

数値データなどをグラフにするには, `gnuplot`<sup>1</sup> を使います。データの時間変化を二次元的なグラフに描画するだけでなく, 変数が2つある場合のデータを3次元的なグラフにしたり, 媒介変数を使って立体的にデータを可視化することも可能です。さまざまなコード例がネット上<sup>2</sup>に例示されており, 目的に応じてそれらを参考にすると役立つでしょう(図11.1参照)。

`gnuplot`では, どのようにグラフを描画するかのコマンドなどを一連のコードとして保存しておいてそれを `load` することによってグラフを作成できます。複雑なコマンドなどをグラフをつくる度に入力する必要はありません。コードファイルとして保存をしておいて, それを `load` するようにならしめましょう。

作成されたグラフを画像として出力して, WEB上などに表示できます。また, EPS形式で保存し, 論文などのLaTeXファイルに読み込むことで, 文章のなかにグラフを示すことができます。本書に用いたグラフもそのようにして作成したものです。

また, 3次元的なグラフを表面表示したり, 同時に等高線を表示することも可能です。3次元的なグラフは複雑な構造になることが多いので、ひとつの角度からそれをながめるだけではそれを容易に理解できない場合があります。Debian<sup>3</sup>デスクトップ上で3D表示されたグラフは, マウスで表示角度を変更しながらその構造を確認できるので, 理解の助けとなります。図11.3に示したコードを `gnuplot` に `load` してみてることを

---

<sup>1</sup><http://www.gnuplot.info/>

<sup>2</sup><http://gnuplot.sourceforge.net/demo/>

<sup>3</sup><https://www.debian.org/index.ja.html>

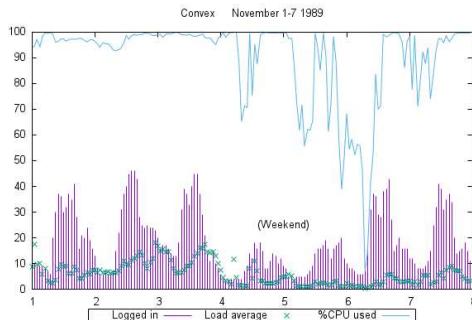


図 11.1: gnuplot で複数のデータをグラフ化する

```

1  # set terminal png transparent nocrop enhanced size 450,320 font "arial,8"
2  # set output 'using.2.png'
3  set key bmargin center horizontal Right noreverse enhanced autotitle box \
4      lt black linewidth 1.000 dashtype solid
5  set style data lines
6  set title "Convex      November 1-7 1989"
7  set xrange [ 1.00000 : 8.00000 ] noreverse nowriteback
8  x = 0.0
9  ## Last datafile plotted: "using.dat"
10 plot 'using.dat' using 3:4 title "Logged in" with impulses, \
11      'using.dat' using 3:5 t "Load average" with points, \
12      'using.dat' using 3:6 "%CPU used" with lines

```

図 11.2: 図 11.1 を gnuplot で描画するためのコード

お勧めします。

## 11.1 gnuplot のインストール

gnuplot は Debian のパッケージになっているので、`apt-get` を使ってインストールします。

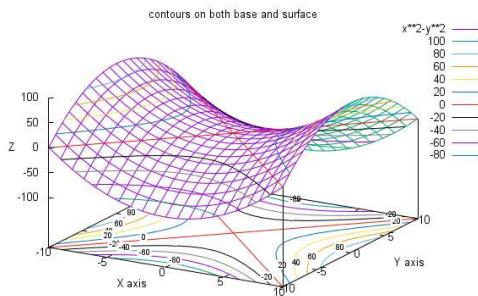


図 11.3: 3 次元構造を可視化する

```

1 # set terminal png transparent nocrop enhanced size 450,320 font "arial,8"
2 # set output 'contours.7.png'
3 set key at screen 1, 0.9, 0 right top vertical Right noreverse enhanced \
4   autotitle nobox
5 set style textbox opaque margins  0.5,  0.5 noborder
6 set view 60, 30, 1, 1.1
7 set samples 20, 20
8 set isosamples 21, 21
9 set contour both
10 set cntrlabel format '%8.3g' font ',7' start 5 interval 20
11 set cntrparam levels auto 10
12 set style data lines
13 set title "contours on both base and surface"
14 set xlabel "X axis"
15 set ylabel "Y axis"
16 set zlabel "Z "
17 set zlabel offset character 1, 0, 0 font "" textcolor lt -1 norotate
18 splot x**2-y**2 with lines, x**2-y**2 with labels boxed notitle

```

図 11.4: 図 11.3 を gnuplot で描画するためのコード

```

# apt-get install gnuplot
# apt-get install gnuplot-x11

```

## 11.2 起動と終了

コンソールから gnuplot を起動すると、対話的にグラフを作成できます。

```
% gnuplot
```

gnuplot プロンプトを終了しコンソール（シェル）にもどるためには。

```
gnuplot> exit
```

を実行します。

## 11.3 データ描画 (2D)

たとえば、以下のようなデータファイル (data.xy) を描画します。

1	0.5	7.8	4.3
2	0.8	6.9	2.1
3	1.1	2.0	3.5
4	1.5	1.9	3.9
5	2.0	0.8	4.2

データを描画するためのコマンドは plot です。

```
gnuplot> plot 'data.xy'
```

このように対話的に plot を用いると、描画結果は 画面上に描画されます。 data.xy ファイル内の 1 列目が横軸、2 列目が縦軸としてグラフが描画されます。

点のサイズが小さいので、大きくしてみましょう。 ポイントサイズを 3 にしてみます。 plot コマンドに ps 3 を追加します。

```
gnuplot> plot 'data.xy' ps 3
```

さらに、点と点を線でつないでみましょう。

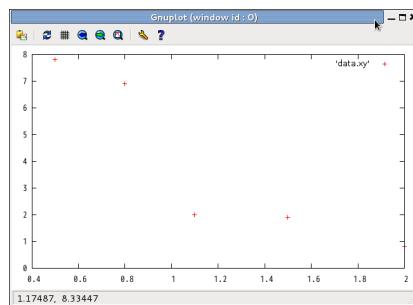


図 11.5: 単純な plot 結果

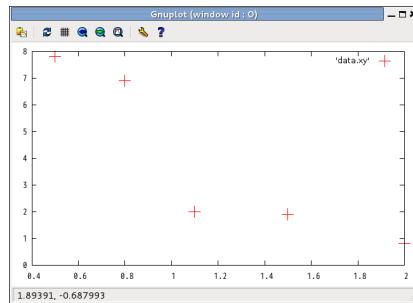


図 11.6: 点のサイズを大きく (3) する

```
gnuplot> plot 'data.xy' ps 3 with lp
```

だいぶ、見やすくなってきました。

つぎに横軸と縦軸にラベルをつきます。

```
gnuplot> set xlabel 'Time(s)'  
gnuplot> set ylabel 'Angle(rad)'  
gnuplot> plot 'data.xy' ps 3 with lp
```

描画範囲を指定するには、`set xrange`, `set yrange` を実行します。

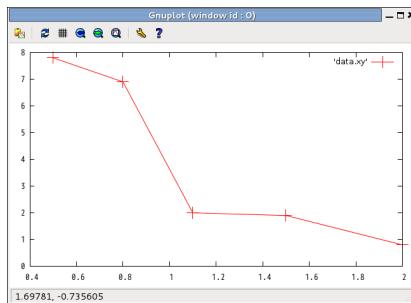


図 11.7: 点と点を線でつなぐ

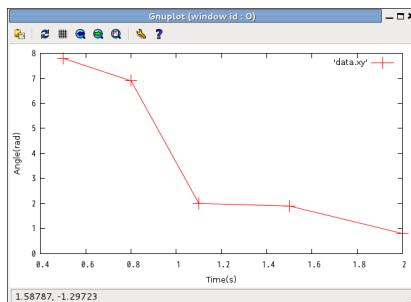


図 11.8: 横軸と縦軸にラベルをつける

```
gnuplot> set xrange [0:3]
gnuplot> set yrange [0:9]
```

`data.xy` の 2 列目も描画すれば、2 つのグラフを同時に一つの図の中に示すことができます。

```
gnuplot> plot 'data.xy' u 1:2 ps 3 w lp, 'data.xy' u 1:3 ps 3 w lp
```

ここで、`with` というオプションを `w` と省略しました。同様に、`u` も実は

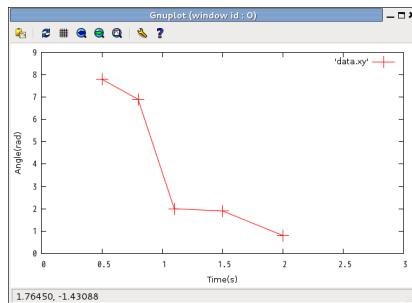


図 11.9: 描画する範囲を指定する。

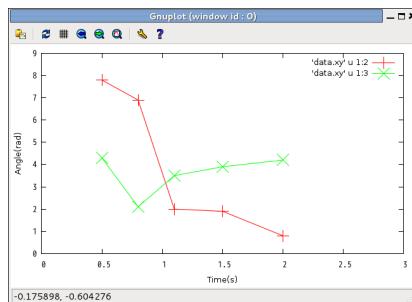


図 11.10: 2 つのデータを同時に描画する。

using の略です。この様にオプションは省略形で短く記述することが出来ます。

条件の異なるデータを比較したい場合には、このように、同時に表示するようにします。

## 11.4 文字の大きさ変更

gnuplot のデフォルトでは、論文掲載用、あるいは発表用としては、文字の大きさが小さい場合があります。set terminal で文字フォントと大き

さを指定します。

```
gnuplot> set terminal wxt 0 font "Helvetica,15"
```

ここでは、`set terminal` の行の中でフォントを指定したので、出力される

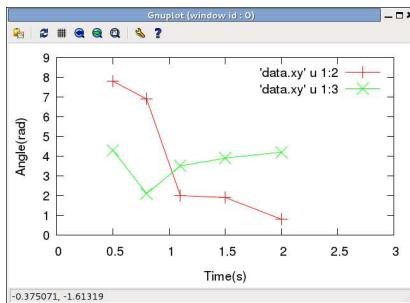


図 11.11: フォントを "Helvetica,15" に指定.

文字すべてにわたって、このフォントとその大きさが摘要されます。一方、`set xlabel font "Helvetica,15"` のように個別の出力項目について、フォントとその大きさを指定することも可能です。

デフォルトでは、データファイル名がグラフタイトルとして表示されるので、グラフの意味に対応して `title` を変更します。

```
plot 'data.xy' u 1:2 ps 3 w lp t "Sound sensor", \
     'data.xy' u 1:3 ps 3 w lp t "Light sensor"
```

`plot` コマンドの行が長くなる場合には、この例の様に、行末に \ をつけると、次の行にその続きをつづけることができます。

## 11.5 制御コードファイル

データを描画する度に、これらの入力を行うのは手間がかかるので、これらの入力をすべてコードファイルとして保存しておくことができます。

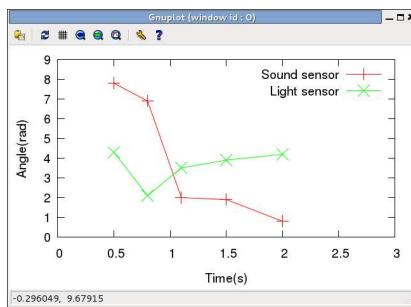


図 11.12: title を付ける

たとえば、vi エディタを使って次の内容のファイル data.gp を作ります。この例では、data.gp というファイルに保存しましたが、もちろん別の

---

```

1 set terminal wxt 0 font "Helvetica,15"
2 set xlabel 'Time(s)'
3 set ylabel 'Angle(rad)'
4 set xrange [0:3]
5 set yrange [0:9]
6 plot 'data.xy' u 1:2 ps 3 w lp t "Sound sensor", \
      'data.xy' u 1:3 ps 3 w lp t "Light sensor"
7

```

---

図 11.13: gnuplot の制御コードファイル例

イル名でもかまいません。いっぽんに、データファイルはどんどん増えていくので、どのようなデータのための gnuplot コードか分かるように、ファイル名を工夫することが必要です。

セーブされた data.gp ファイルを load することによって、グラフに反映させます。

```
gnuplot> load "data.gp"
```

data.gp ファイル内には、先に実行した plot コマンド、set xlabel などの他にも、gnuplot を制御するための様々なパラメータなどを記述することができます。

## 11.6 対数スケール

非常に大きな数や、非常に小さい数を同時に描画したい場合、対数スケールを用います。

```
gnuplot> set logscale <x/y> <基底>
```

## 11.7 EPS 出力

描いたグラフを EPS 形式で出力するためには data.gp ファイル内で set terminal を postscript に変更し、set output を指定します。この様に記述さ

```
1 set terminal postscript eps color enhanced font "Helvetica,20"
2 set output "data.eps"
3 set xlabel 'Time(s)'
4 set ylabel 'Angle(rad)'
5 set xrange [0:3]
6 set yrange [0:9]
7 plot 'data.xy' u 1:2 ps 3 w lp t "Sound sensor", \
8      'data.xy' u 1:3 ps 3 w lp t "Light sensor"
```

図 11.14: グラフを EPS 出力するための gnuplot の制御コードファイル例

れた、data.gp ファイルを先述のように load すると、data.eps というファイルが自動生成されます。

## 11.8 EPS 出力で複数のグラフを **replot** する

terminal が postscript の場合、plot のあと、replot を実行すると、最後の replot されたグラフのみが EPS ファイルに書き込まれ、それ以前のグラフは現れません。それを回避するために、いったん

```
gnuplot> set output '/dev/null'  
gnuplot> plot f(x)  
gnuplot> set output 'file.eps'  
gnuplot> replot g(x)
```

とします。最初のプロットは /dev/null にプロットし、最後の replot の直前で、output ファイルを指定します。

## 11.9 LaTeX 文章へのグラフの組み込み

EPS ファイルができたので、いよいよ LaTeX 文章ファイルにグラフを組み入れてみよう。プリアンブル部で次のように記述します。プリアンブルとは、\documentclass と \begin{document} のあいだの部分のことです。

```
\usepackage[dvips]{graphicx,psfrag}
```

つぎに、LaTeX ファイル内で実際に EPS ファイルを読み込んでグラフを表示したい場所で \includegraphics を用います。

```
\begin{figure}[h]  
  \begin{center}  
    \includegraphics[width=\figw]{data.eps}  
  \end{center}  
  \caption{\label{fig:data_eps}gnuplot から出力された EPS グラフ}  
\end{figure}
```

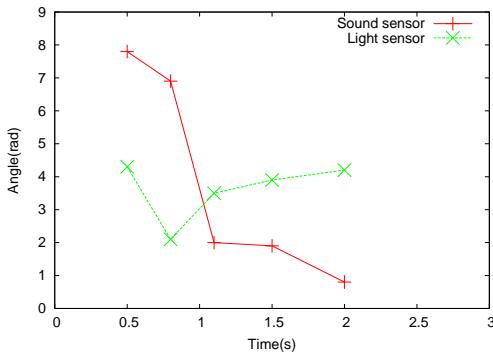


図 11.15: gnuplot から出力された EPS グラフ

## 11.10 EPS ファイル内の文字を日本語に置き換える

```
\begin{figure}[h]
\psfrag{Time(s)}{\footnotesize 時間 (s)}
\psfrag{Angle(rad)}{\footnotesize 角度 (rad)}
\psfrag{Sound sensor}{\tiny 音センサー}
\psfrag{Light sensor}{\tiny 光センサー}
\begin{center}
\includegraphics[width=0.8\textwidth]{data.eps}
\end{center}
\caption{\label{fig:data_psfrag}EPS ファイル内の文字を日本語に置き換える}
\end{figure}
```

この機能を用いるためには、dvipdfmx ではなく、dvips と ps2pdf を使う必要があります。

また、フォントの選択の都合などで pdf ファイルの生成にどうしても dvipdfmx をつかいたい場合には、図の部分だけ図 11.17 の様なファイルを用意して eps ファイルを作つておいてから本文の方にその eps ファイルを読み込むと良いでしょう。

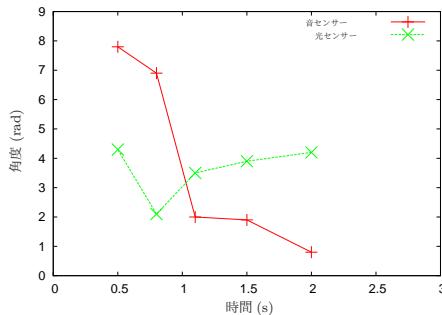


図 11.16: EPS ファイル内の文字を日本語に置き換える

```

1 \documentclass{jarticle}
2 \special{papersize=200.00mm,200.00mm}
3
4 \usepackage[dvips]{graphicx,psfrag}
5 \usepackage{color}
6
7 \newlength{\xadd}
8 \setlength{\xadd}{0mm}
9 \addtolength{\textwidth}{\xadd}
10 \addtolength{\oddsidemargin}{-0.5\xadd}
11 \addtolength{\evensidemargin}{-0.5\xadd}
12
13 \newlength{\figw}
14 \setlength{\figw}{0.9\textwidth}
15
16
17 \begin{document}
18 \pagestyle{empty}
19
20 \psfrag{Time(s)}{\footnotesize 時間 (s)}
21 \psfrag{Angle(rad)}{\footnotesize 角度 (rad)}
22 \psfrag{Sound sensor}{\tiny 音センサー}
23 \psfrag{Light sensor}{\tiny 光センサー}
24 \begin{center}
25   \includegraphics[width=0.8\textwidth]{data.eps}
26 \end{center}
27
28 \end{document}

```

図 11.17: psfrag と ps2pdf コマンドを使って eps ファイルを生成するための LaTeX コード

## 11.11 C 言語ソースコードから直接グラフを作る

`gnuplot` でグラフを描画する作業は、通常は、シェルから `gnuplot` を起動し、その中でコマンドを実行することによって行います。あるいは、`gnuplot` の制御ファイルを作って、それを `load` コマンドで `gnuplot` に読み込むことでグラフを作ります。

ここでは、C 言語のパイプオープン (`popen`) 機能を利用して、シミュレーションなどで作成したデータファイルを、直接 C 言語のソースコードによってグラフにする方法を紹介します。この方法を用いると、シェルから `gnuplot` のコマンドを実行したり、制御ファイルを編集したりする作業が省略できます(図 11.18)。

```
1 #define RESULTS "result_xxxx.xy"
2 // 計算結果を保存するファイル名
3 #define OUTPUTEPS "result_xxxx.eps" // グラフのepsファイル名
4
5 FILE* fp; // 計算結果を出力するためのファイルポインター
6 FILE* gp;
7 // gnuplotをパイプオープンしてコマンドを渡すためのポインター
8
9 fp=fopen(RESULTS,"w"); // ファイルRESULTSを開く
10
11 ...
12 fprintf(fp,"%f %f %f\n", t, res1, res2);
13 // 計算結果をRESULTSに書き込む
14 ...
15
16 fclose(fp); // 計算が終了したら, RESULTSを閉じる.
17 ...
18
19 gp=popen("gnuplot","w"); // プロセスgnuplotを開く.
20 fprintf(gp,"set term post eps enhanced color 'Helvetica,20'\n");
21 // ターミナルタイプをepsにして, 文字の大きさを20ptとする.
22 fprintf(gp,"set output '%s'\n",OUTPUTEPS);
23 // グラフの出力先をOUTPUTEPSにする.
24 fprintf(gp,"set xlabel 'tau'\n");
25 // 横軸のラベル xlabelを指定する.
26 fprintf(gp,"set ylabel 'lambda'\n");
27 // 縦軸のラベル ylabelを指定する.
28 fprintf(gp,"plot '%s' using 1:2 with lp title 'Re(lambdaP)', \
29 '%s' using 1:3 with lp title 'Im(lambdaP)' \
30 \n",RESULTS,RESULTS);
31 // gnuplot の plotコマンドでグラフを描画する.
32 // 複数のグラフを同時に描画する場合には, 複数行に分けて記述した方が
33 // 分かりやすい.
34 pclose(gp); // gnuplotに対するプロセスピンターを閉じる
```

図 11.18: `popen` を使って, C 言語から直接 `gnuplot` を使う例

## 11.12 簡易アニメーション

たとえば、以下のような内容の制御ファイル”anime.gp”を作って、

```

1 xwidth=600
2 ywidth=400
3 set term x11 1 title "Accel" size xwidth,ywidth position 0,0
4 pi=3.141592
5 set yrangle [-pi:pi]
6 plot '< tail -100  opedev9.0.dat' u 1:2 w l t "ax", \
7      '< tail -100  opedev9.0.dat' u 1:3 w l t "ay", \
8      '< tail -100  opedev9.0.dat' u 1:4 w l t "az"
9 set term x11 2 title "Gyro" size xwidth,ywidth position xwidth+10,0
10 set yrangle [*:*]
11 plot '< tail -100  opedev9.0.dat' u 1:5 w l t "gx", \
12      '< tail -100  opedev9.0.dat' u 1:6 w l t "gy", \
13      '< tail -100  opedev9.0.dat' u 1:7 w l t "gz"
14 pause 0.2
15 reread

```

図 11.19: tail コマンドと reread を使って、簡易アニメーションを行うための gnuplot コード

```
% gnuplot anime.gp
```

を実行すると、データファイルの最後の 100 行が常に描画されます。データファイルの行末に、どんどんデータを追加していくけば、リアルタイムにそのデータがグラフに描かれます。

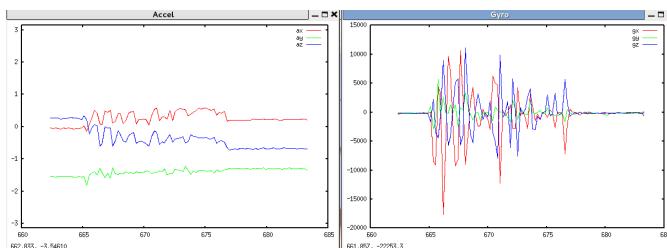


図 11.20: 簡易アニメーションの例

## 11.13 演算結果をプロットする

GNUPLOT では、データファイルの中の値をそのままグラフにするのではなく、それらの値に演算をしたもの、たとえば定数を掛けたものをプロットすることができます。

```
gnuplot> a=0.123
gnuplot> b=3.210
gnuplot> plot "data.xy" using ($1*a):($2*b)
```

ここでは、定数 a,b を定義しておいて、それを 1 列目と 2 列目のデータに掛け算してプロットした例を示した。

## 11.14 データを間引いてプロット

すべてのデータをプロットするのではなく、行やブロックを間引いてグラフを作成することができる。そのためには、`every` オプションを利用する。

### 11.14.1 ブロック

GNUPLOT は 1 行の空白行で区切られたデータのかたまりを一つのブロックとして認識する。つぎのようなデータファイル `gnuplot_sample1.xy` があるとする。

1	0.5	7.8	4.3
2	0.8	6.9	2.1
3	1.1	2.0	3.5
4	1.5	1.9	3.9
5	2.0	0.8	4.2
6			
7	2.2	0.9	5.0
8	2.4	1.0	5.1
9	2.6	1.3	6.0
10	2.8	1.8	6.5
11	3.0	1.5	4.8

6 行目の空白行でブロックが区切られており、1 ~ 5 行目がブロック 0, 7 ~ 11 行目がブロック 1 となる。ここでは、ふたつのブロックの例を示したが、以下空白行で区切られたデータ行が現れるとブロック番号が増えたブロックとして認識される。

以下のような内容の `gnuplot_sample1.gp` というファイルを `gnuplot` で実行する。

```

1 set term post eps enh color "Helvetica,20"
2 set output "gnuplot_sample1.eps"
3 set xlabel "xlabel name"
4 set ylabel "ylabel name"
5 plot "gnuplot_sample1.xy" w lp

```

```
% gnuplot gnuplot_sample1.gp
```

データ間を線で結ぶオプションを用いたが、ブロック間は線で結ばれないで描画される。

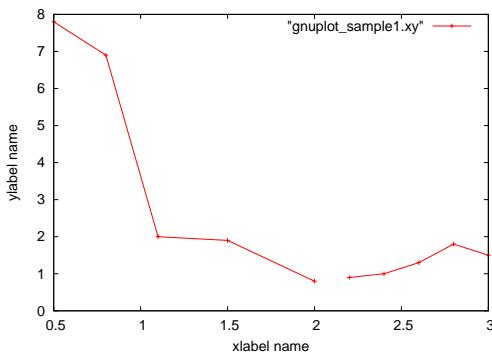


図 11.21: ふたつのブロックにわかれたデータのプロット例

### 11.14.2 `every` オプション

`every` オプションを用いて、行刻みを 2 に指定した

```

1 set term post eps enh color "Helvetica,20"
2 set output "gnuplot_sample2.eps"
3 set xlabel "xlabel name"
4 set ylabel "ylabel name"
5 plot "gnuplot_sample1.xy" every 2 w lp

```

を実行すると、データが2つごとに、すなわち1つ飛ばしで描画される。もちろん、every 2 の部分を every 3 とすれば、3つごとに描画され、

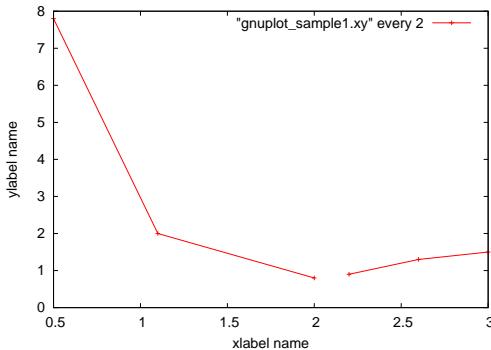


図 11.22: every 2 オプションを用いて、データを2行ごとにプロット

以下同様である。

every オプションでは、行刻み以外に以下のフォーマットで初期行や終了行などを指定できる。

every 行刻み:ブロック刻み:初期行:初期ブロック:終了行:終了ブロック

以下の gnuplot\_sample3.gp というファイルを実行すると、

```

1 set term post eps enh color "Helvetica,20"
2 set output "gnuplot_sample3.eps"
3 set xlabel "xlabel name"
4 set ylabel "ylabel name"
5 set xrange [0.5:3.0]
6 set yrange [0:8]
7 plot "gnuplot_sample1.xy" every 1::1::3 w lp

```

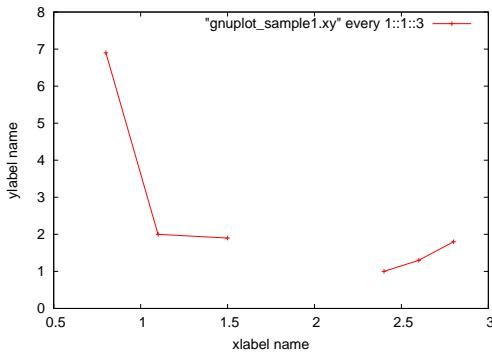


図 11.23: every オプションを用いて、開始行と終了行を指定

各ブロックのなかで、1 ~ 3 行目がプロットされる。行番号は、ブロック番号同様に、0 から始まることに注意する必要がある。

## 11.15 範囲指定してプロット

値の範囲が条件に合う場合だけ、プロットすることができる。usingオプションの中で、例えば

```
plot "gain140820a.dat" u 1:($4<0.1 ? ($4>0.01 ? $2:1/0):1/0)
```

のように値の範囲を指定する。この例では  $0.01 < \text{$4} < 0.1$  の場合だけ y 軸に  $\$2$  の値を用いてプロットする。

$1/0$  の部分には、本来条件に合わない場合にプロットされる値が入る。 $1/0$  は定義されない値なので、なにもプロットされない。

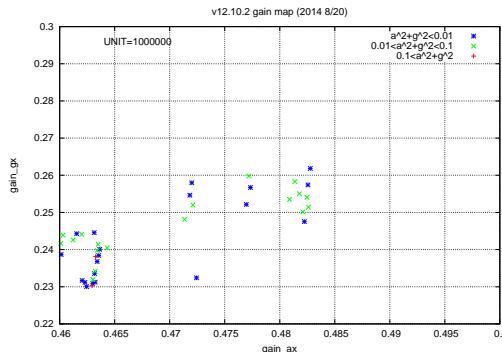


図 11.24: 値の範囲を指定してプロットした例

## 11.16 データを関数で fit する

数値データを関数で近似 (fit) したい場合、たとえば下記のようにします。data1.xy の 1 列目を  $x$ 、2 列目を  $y$  として、3 次関数  $f(x)$  を fit しています。パラメータは 4 つあるので、9 行目の via で指定しています。

この例では行っていませんが、パラメータの初期値を指定しないと、gnuplot が正常にパラメータを決定できず、fit が終了する場合があるので、

```
1 set term post eps enh color font 'Helvetica,20'
2 set output 'data1.eps'
3 set zeroaxis
4 set xlabel "x"
5 set ylabel "y"
6 set xrange [-20:30]
7 set yrange [-3:3]
8 f(x)=a0+a1*x+a2*x*x+a3*x*x*x
9 fit f(x) "data1.xy" u 1:2 via a0,a1,a2,a3
10 plot f(x) , "data1.xy" w p ps 3 pt 7
```

パラメータ初期値を推測できる場合には、それを指定してから fit を実行した方が無難です。

## 第12章 飛行ロボット研究ノート

### 12.1 10.4号機, 飛行に初成功

2014 2/10, 4回転翼飛行ロボットとして飛行に初めて成功する。アンプ Phenix 10A. モーター Hyperion 1709-09.



図 12.1: 4回転翼飛行ロボット(v10.4), 飛行の様子

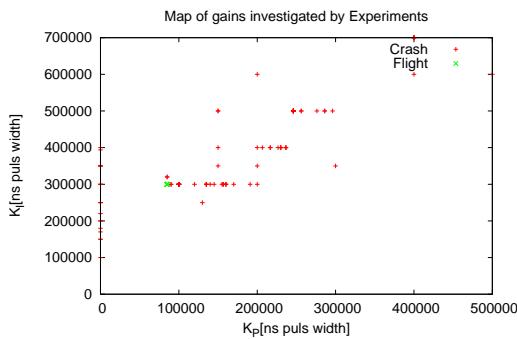


図 12.2: 飛行実験結果 MAP

### 12.1.1 ゲインなどのパラメータ

### 12.1.2 角度, 各加速度データ

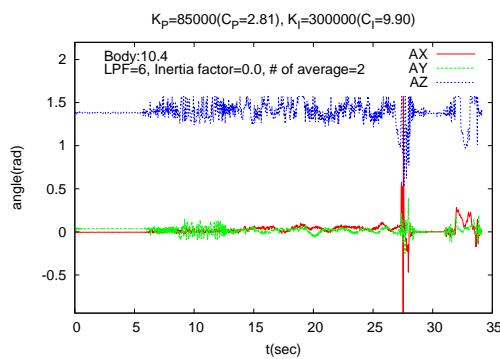


図 12.3: 安定時の角度センサー値

表 12.1: 安定時のゲインなどのパラメータ

名前	値	備考
GAIN_AX	300000.0	$x$ 軸角度ゲイン ( $K_I$ )
GAIN_AY	300000.0	$y$ 軸角度ゲイン ( $K_I$ )
GAIN_GX	85000.0	$x$ 軸角速度ゲイン ( $K_P$ )
GAIN_GY	85000.0	$y$ 軸角速度ゲイン ( $K_P$ )
INERTIA_FACTOR	0.0000	慣性力補正因子
GAIN_X	100000.0	操作機 $x$ 軸ゲイン (不使用)
GAIN_Y	100000.0	操作機 $y$ 軸ゲイン (不使用)
M1_CALIB	0.9890	$m_1$ 出力調整因子
M2_CALIB	0.9980	$m_2$ 出力調整因子
M3_CALIB	0.9850	$m_3$ 出力調整因子
M4_CALIB	0.9990	$m_4$ 出力調整因子
N_AVE	2	移動平均数
LPF	6	ローパスフィルタ (5Hz)

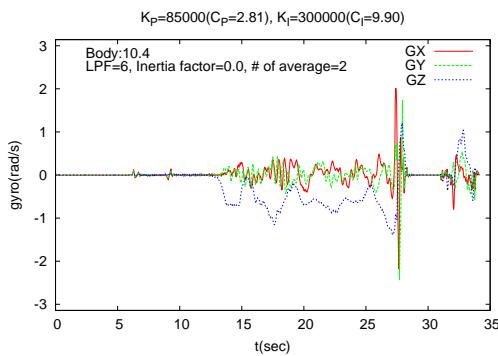


図 12.4: 安定時角速度 (gyro) センサー値

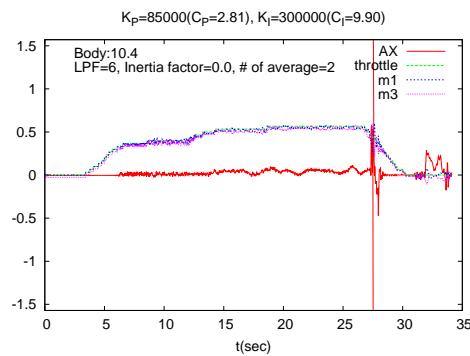


図 12.5: 安定時モーター制御値

表 12.2: 不安定時のゲインなどのパラメータ

名前	値	備考
GAIN_AX	320000.0	$x$ 軸角度ゲイン ( $K_I$ )
GAIN_AY	320000.0	$y$ 軸角度ゲイン ( $K_I$ )
GAIN_GX	85000.0	$x$ 軸角速度ゲイン ( $K_P$ )
GAIN_GY	85000.0	$y$ 軸角速度ゲイン ( $K_P$ )
INERTIA_FACTOR	0.0000	慣性力補正因子
GAIN_X	100000.0	操作機 $x$ 軸ゲイン (不使用)
GAIN_Y	100000.0	操作機 $y$ 軸ゲイン (不使用)
M1_CALIB	0.9800	$m_1$ 出力調整因子
M2_CALIB	0.9900	$m_2$ 出力調整因子
M3_CALIB	0.9850	$m_3$ 出力調整因子
M4_CALIB	0.9990	$m_4$ 出力調整因子
N_AVE	2	移動平均数
LPF	6	ローパスフィルタ (5Hz)

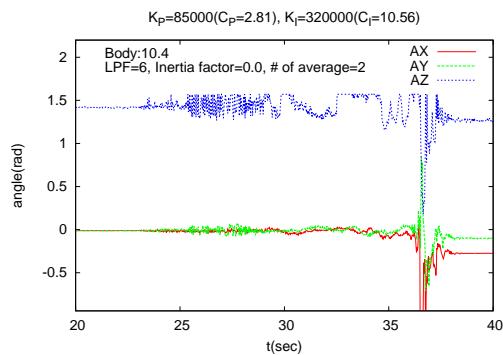


図 12.6: 不安定時の角度センサー値

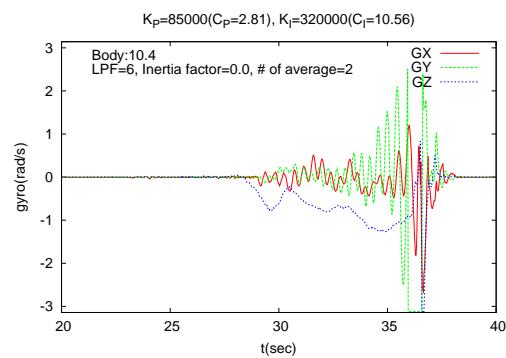


図 12.7: 不安定時角速度(gyro)センサー値

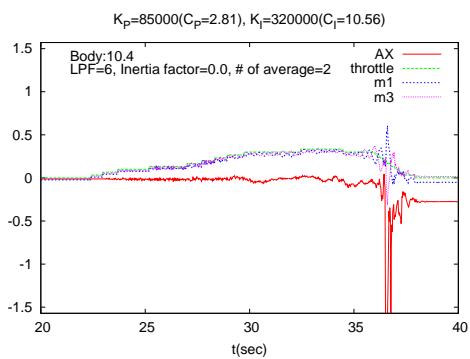


図 12.8: 不安定時モータ一制御値

## 12.2 11号機

2014年5/17 v11.0が完成。モーターにMultistar 2213-935、プロペラGWS 9050x3（3枚タイプ）を使えるように、大型化。アンプMultistar 20A x4。バッテリーRobin 3s 1000mAh 2個使用。



図 12.9: 4回転翼飛行ロボット(v11.0)

分割して持ち運びできるように、改良した。重量約900g。



図 12.10: v11.0号機裏

制作期間、約2ヶ月。

### 12.2.1 初実験 2014 5/17

モーター番号, 回転方向と感覚運動写像関数の調整.

20秒程度の実験を7回行なって, bbb 接続電池 (Robin 3s 1000mAh) が400/1000 消耗. モーター x 2 用の電池は 130/1000 消耗. bbb がかなり電力消費していることが分かる. 図 12.11において, thrott=1 で 100%出力

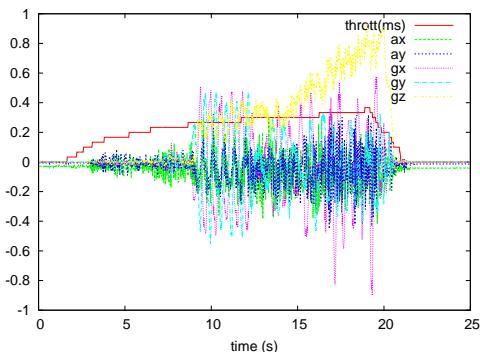


図 12.11: スロットル値と各センサー値の例

に対応する. スロットルは, 27% ほどで浮上が始まる.

8–12秒付近. MPU9150 の DLPF=4, データー平均個数 2 個でセンサー値からノイズ除去. このノイズ除去レベルで機体姿勢を, ノイズに埋もれないで取得できていると考えられる.

12–20秒では, 浮上しようとして, 床に激しくぶつかると同時に, 機体が右回転しているのが分かる.

### 12.2.2 BBB とバッテリー



図 12.12: v11.0 号機に搭載した BBB

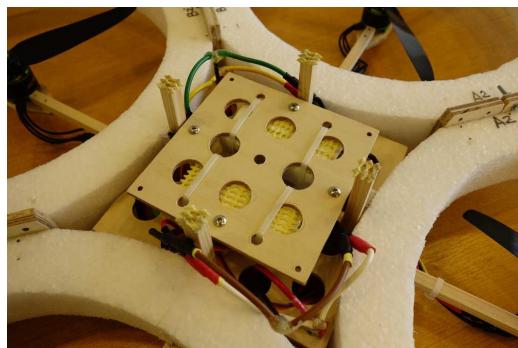


図 12.13: バッテリー搭載は、4mm 厚シナベニアを 3mm ネジ x4 で止める

### 12.2.3 ドーム型プロテクター(v11.2)

このタイプの機体では、水平状態を保てないので、basse と haute を起動時にセンサーのオフセット値を正確に測定できない。そのため、水平



図 12.14: v11.2 号機にドーム型のプロテクタを上下につけた

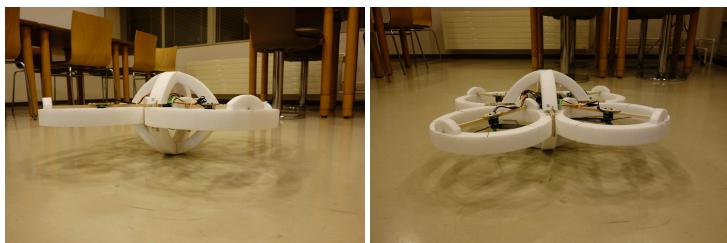


図 12.15: BBB やバッテリーを保護すると同時に、水平状態から可動する  
ので安定ゲインを調べる実験を床のうえで直接行える

状態を作つて、オフセット値を測定し、それ以降はその値を固定して用いることになる。機体が完全水平状態から微妙にずれることにより、横滑りを発生していると考えられる。

次バージョンでは、機体の水平を保てる EPP 脚を付けた。

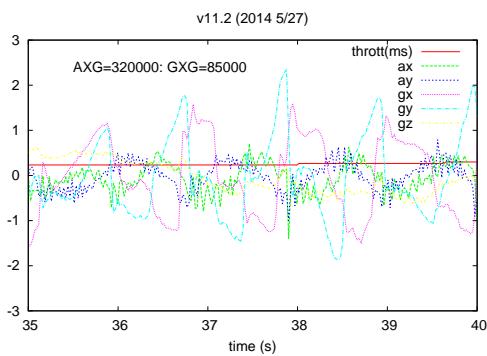


図 12.16: 加速度と角速度の観測結果. 振動の周期から  $\omega$  の値を仮定できそう. ((3.53) 式)

### 12.2.4 ゲインを Self-Avoiding Walk で学習するアルゴリズム

安定飛行をするためには、線形感覚行動写像のゲイン値を適切なものに決めなければならない。試行錯誤で決めていたが、1回のプログラム実行で、ゲインのセットを何組みか試行できるアルゴリズムを考えた。

`basse11.2.2` プログラムからキーボード操作で、ゲイン値の変化量を調節するループを `throttle` を調節するループの外側に加えた。`pvm` を通して `throttle`などを `hautell.2.2` とやり取りするので、`pvm_send` と `pvm_recv` がしっかり対応していないといけない。

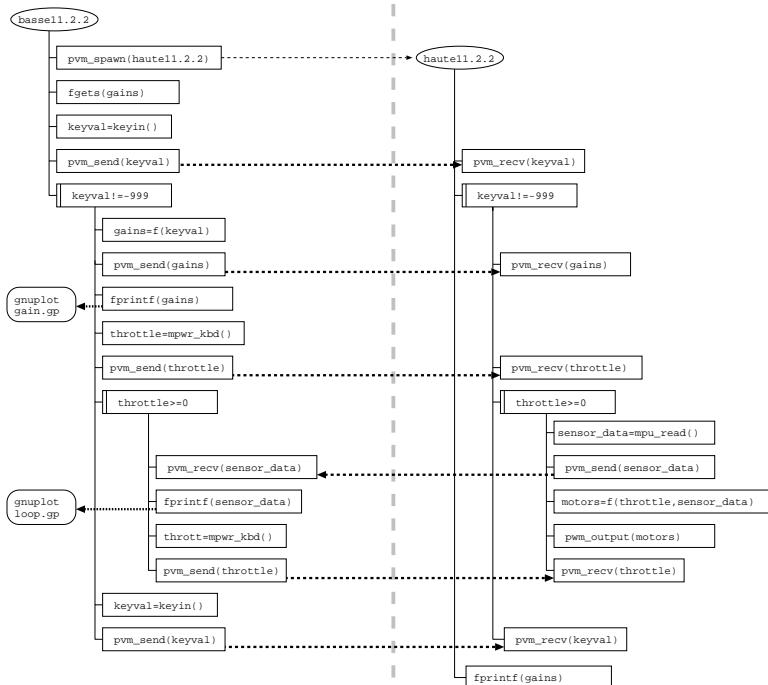


図 12.17: PVM を利用してゲイン値を変更しながら飛行実験するアルゴリズム

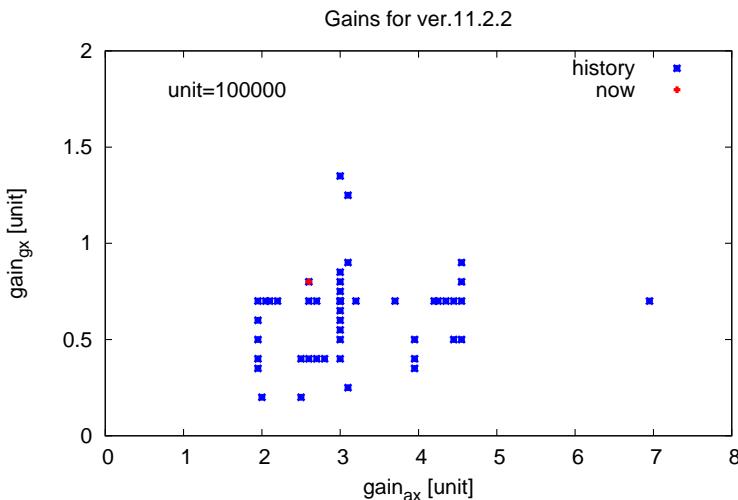


図 12.18: 飛行実験したゲイン (2014.6/2)

## まとめ

モーターに Multistar 2213-935, アンプ Multistar 20A x4, プロペラ GWS 9050x3 (3枚タイプ) あるいは GWS 8x4.5L を用いた。バッテリー Robin 3s 1000mAh 2個使用 あるいは、TURNIGY 3s 1300 30-40c を用いた。試行錯誤の範囲内では、水平安定が得られなかった。

10号機と同じ、アンプ Phenix 10A, モーター Hyperion 1709 に換装すると、10号機と同程度に安定する。このことから、アンプ Multistar 20A x4 モーター Multistar 2213-935 の組み合わせでは、反応遅れが大きすぎると考えられる。機体へのフォトインターラプタ搭載と、アンプおよびモーター接続へのコネクタ利用で、アンプ+モーター+プロペラを変更して反応遅れの本格的計測が必要と思われる。

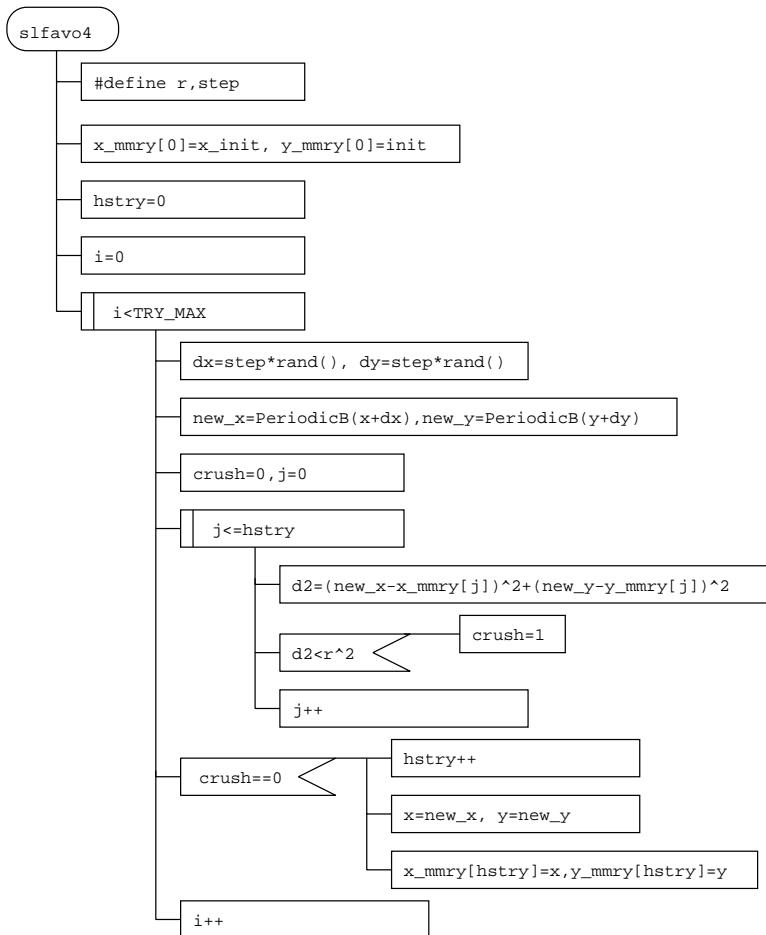


図 12.19: 2 次元 Self Avoiding Walk のアルゴリズム (2014.6/4)

(7/12 追記) Multistar のアンプは PWM のリフレッシュレートがプログラムで可変らしい。最大 499Hz まで上げることが出来るという情報を得る。

アンプの出力信号は、パルス幅が 1msec(出力ゼロ) から 2msec(出力 100%) なので、パルス発生ごとに毎回パルス幅を変更するとしても

500Hz が限界. 499Hz というのはその限界ギリギリまで, リフレッシュレートを上げられるという事を意味する.

Phenix 10A による実測（小松卒論）では 20msec(50Hz) 程度であった. Waypoint 10A アンプに至っては 100msec(10Hz) とリフレッシュレートが低かった. Phenix で飛行成功し, Waypoint で飛行失敗しつづけていた主原因はこの PWM リフレッシュレートの低さが（時間遅れが大きい）であると考えられる.

499Hz が可能であれば, 更に安定領域は広がる可能性がある.

### 十字ナセルアーム (v11.4, 7/2 2014)

アンプ Phenix 10A, モーター Hyperion 1709, プロペラ 7x3.5 に換装し, さらにナセルに付加するモーターマウンタアームを十字にして, 衝突時ナセルの変形を抑えてみる (図 12.20). ナセル内径が 26 cm, プロペラ直径が 18 cm なので, 両端合計で 8 cm のクリアランスがある. 片側 4 cm のクリアランスと十字アームで強化されたナセルの効果で, 墜落時にプロペラがナセルなどと接触して破損する頻度は激減した.



図 12.20: 11.4 号機に搭載された Hyperion 1709 と 7x3.5 ローター

加えて, 外側に向かってナセルアームを上昇させてつけることで, モーター回軸に上反角をつけてみた, 詳細な実験データ観測比較はしていないが, 少なくとも劇的な安定性向上は観測されない.

このパワーユニット構成にしては, 重量および慣性モーメントが大きすぎるとかんがえられ, 飛行中にアンプが突然停止し, 墜落が頻発. プロペラおよびナセルには破損はないが, 上部のドーム状補強アームが破損(図 12.21). この補強アームは交換が容易なので, この部分が差損する

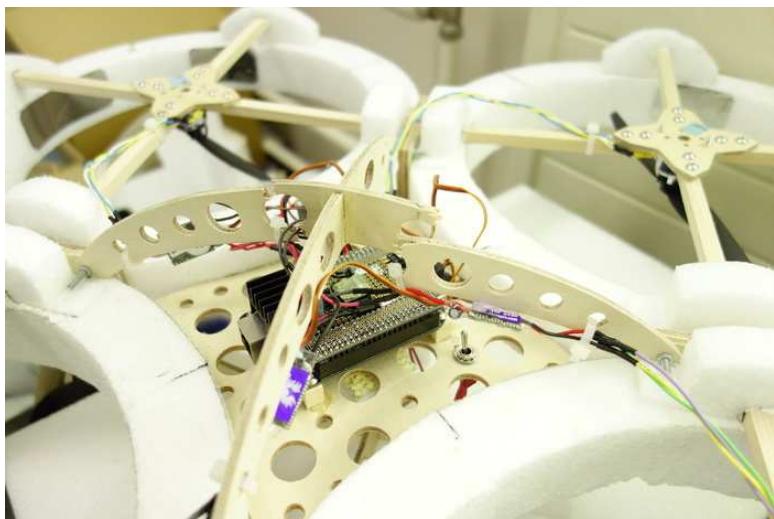


図 12.21: 上部ドーム状補強アームの破損状況

ことによって、他の部分の破損を抑える効果があると考える。

### 12.2.5 飛行成功 (v11.6) 2014.7.7

モーターを Hyperion 2205-34, アンプ Castle TALON 15A, プロペラ GWS 8x4.5(7 インチに切断したものを 10 号機用から流用) に変更(v11.6).

TALON15A は Phenix10A と同程度の時間遅れ(PWM リフレッシュレート)の様で, Phenix10A と同程度に安定する. なおかつ, 15A なので, 電流容量に余裕があり, 飛行中にブレーキがかからない.

プログラム haute10.6.3.c, basse10.6.2.c で飛行成功. 安定感は 10 号機と同程度. Hyperion 1709 モーターと比べて, 重量が増加するが, トルクの余裕と静音性がある.

また, プロペラをゴムオーリングではなく, ネジで止めるタイプを用いたので, 衝撃時に, ナセルやナセルアームにプロペラが衝突する頻度が下がった.

ただし, プロペラは完全固定されているので, 指などに与える衝撃はオーリング固定の場合よりも大きいと予想されるので, 取り扱いを慎重にする必要がある.

表 12.3: 飛行に成功したパラメータ. 図 12.22 参照

GAIN.AX	240000.0	GAIN.AY	240000.0
GAIN.GX	75000.0	GAIN.GY	75000.0
INERTIA.FACTOR	0.000000		
GAIN.X	0.000000	GAIN.Y	0.000000
M1.CALIB	0.995000	M2.CALIB	0.990000
M3.CALIB	0.990000	M4.CALIB	0.999000

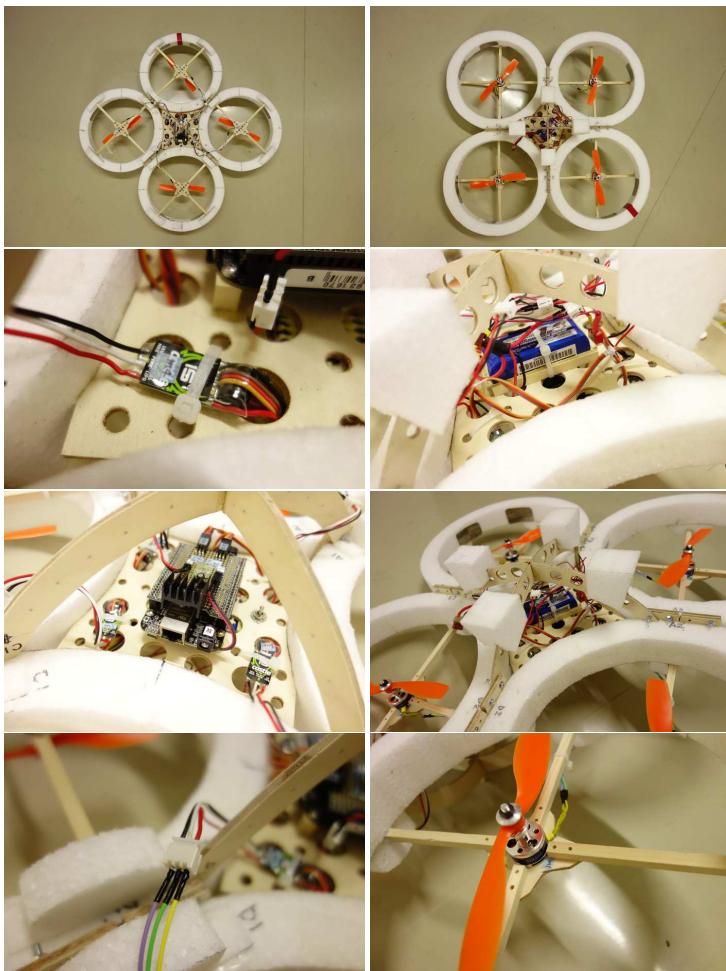


図 12.22: v11.6 の構成

## 12.2.6 上下十字ナセルアームのプル型ローター(v11.8)

2014 7/14 実験

1. アンプ Castle TALON 15A を Castle Link をつかってモード調整する。

デフォルトでは Airplane throttle の Auto throttle range になっている。throttle mode の選択肢から Multi rotor を選択する。Multi rotor mode を選ぶと、throttle range は Fixed, PWM rate などは選択不可の状態になった。

結果、毎回スロットルレンジが固定されているので、実験再現性が改善した。

2. z 軸周りの回転を gyro.gz をつかって止める制御を導入 (haute10.6.4.c).
3. 加速度センサゲインをゼロにして、ジャイロセンサ値だけで飛行実験。フラつきはない。フラつきは加速度センサのノイズが影響している。横へのスライドは止められない。
4. 加速度センサゲインをゼロにして、mpu9150 のローパスフィルタモードをテスト。LPF=5 で必要十分である。(10Hz, Delay=13ms)
5. モーター直下に仮設の脚をつけて水平の精度を上げる。バッテリー保護ドーム下の脚だけでは、モーター直下で 5 mm ほど上下誤差がある。モーターアーム下に脚をつけることによって、誤差 1mm 程度となる。

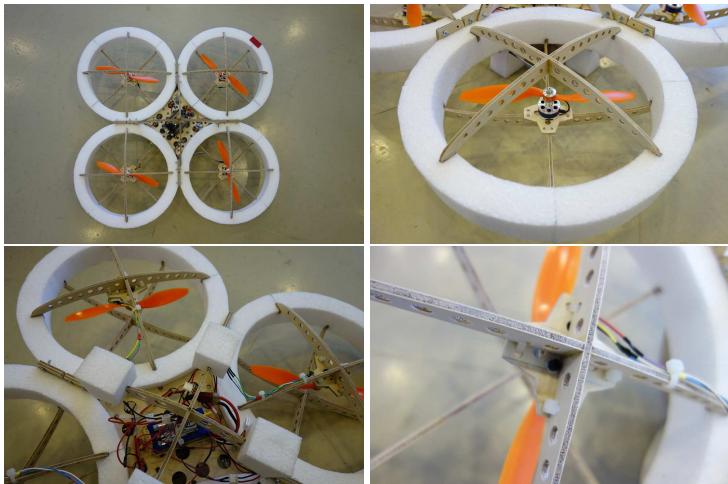


図 12.23: v11.8 の構成。上下十字ナセルアームとプル型ローター

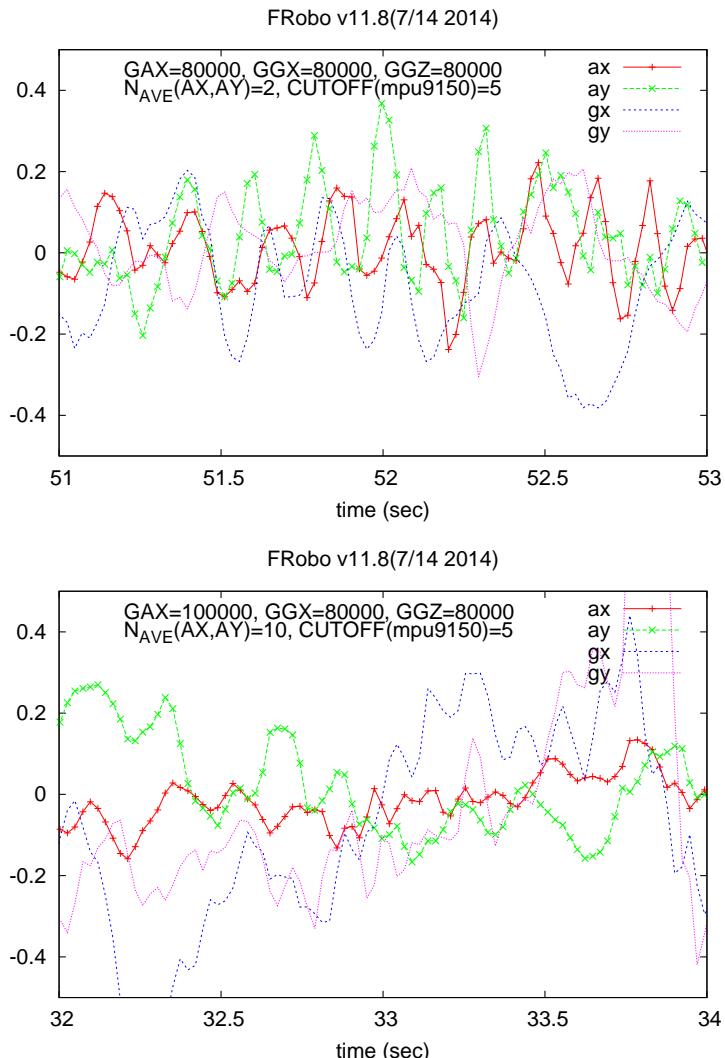


図 12.24: v11.8 のセンサーデータ例 (take3,take4)

### 12.2.7 下部センサーによる制御および制御レートの向上 (2014 7/16)

センサーを下部ドームに設置。慣性力の効果が打ち消されることを期待。ノイズも少しちいさくなったようである。

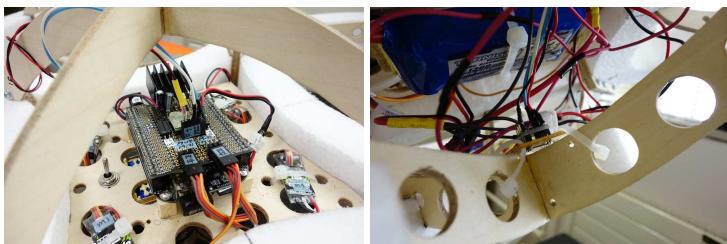


図 12.25: 下部センサーを設置。i2c-2 で値を取得 (address X069)

bbb 上の mpu9150 における外部 SCL, 外部 SDA 端子に下部 mpu9150 の SCL, SDA を接続したが、動作せず。i2c-2 を使うことにする。

また、この実験では、basse10.6.2 の usleep を 10000 とした。つまり、throttle 値の更新レートを 100Hz(10msec ごと) にスピードアップ。これ以上スピードアップすると、BBB の受け取りが滞るのか PVM 遅延が生じる。

haute10.6.5.c で無限ループの毎回 fprintf を実行してみると、PWM 更新レートは 120Hz ほど出ている。ただし、fprintf を 20msec ごと (50Hz) に落とさないと、pvm 遅延が生じる。

飛行が BBB 上部にセンサを設置するより安定する。もっとも安定したと見えた実験結果。数秒間水平を保ったと見えたが、徐々に振動が発散した。図 12.26 参照。

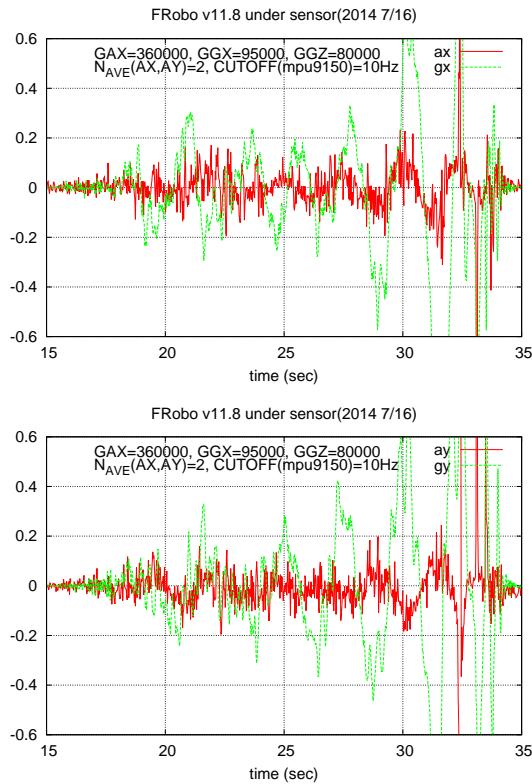


図 12.26: 下部センサーによる制御データ例(take1x,take1y) 飛行時間約 1  
4 秒

### 安定ゲインと、i2c のクロック (7/26 2014)

v11 は障害物が無い限り、安定飛行する。屋外でも実証。安定化のため必要と考えられる要件のまとめ。

1. アンプ (castle Talon/ phenix) を castle link をつかって multi rotor mode にする。スロットルレンジが固定される。PWM rate はいくつなのか不明だが、Phenix 使用時よりも安定するので、50Hz 以上と思われる。
2. i2c のクロックを 100kHz にする。これにより、制御最深ループのレートが 150Hz 程度となる。mpu9150 のデータ読み込みに、意外と時間がかかる。400kHz で試したが、少し不安定。
3. 加速度・ジャイロセンサは下部につける。慣性力効果を相殺する。
4. ゲイン: 加速度センサー 410000 ; ジャイロ (x,y):175000 ; (z)80000. LPF :10Hz(0x05). (プログラムバージョン 11.8.1)

1 1号機は、このまま保存して、実験にはつかわず、デモなどに備える。

i2c のクロックについて補足。haute の更新レート (throttle loop) の中で、最も時間が掛かっているのは i2c で mpu9150 のからセンサー値を読み取る関数の部分。/boot/u-boot/dtbs/am335x-boneblack.dts に am335x-boneblack.dts.i2c-1.400k をコピーして（内容詳細はファイル内を参照）

```
# dtc -I dts -O dtb am335x-boneblack.dts > am335x-boneblack.dtb
```

を実行後リブートすると、/dev/i2c-1 のスピードが 400kHz となり、throttle loop のレートは 300Hz 以上になる。i2c のスピードは、

```
# dmesg | grep i2c
```

で確認できる。i2c-2 は 100kHz で動いている。i2c-2 を使うと throttle loop のレートは 150Hz ほどで、飛行は安定する。i2c を 50kHz で利用すると throttle loop のレートは 80Hz ほどになり、飛行は安定しない。

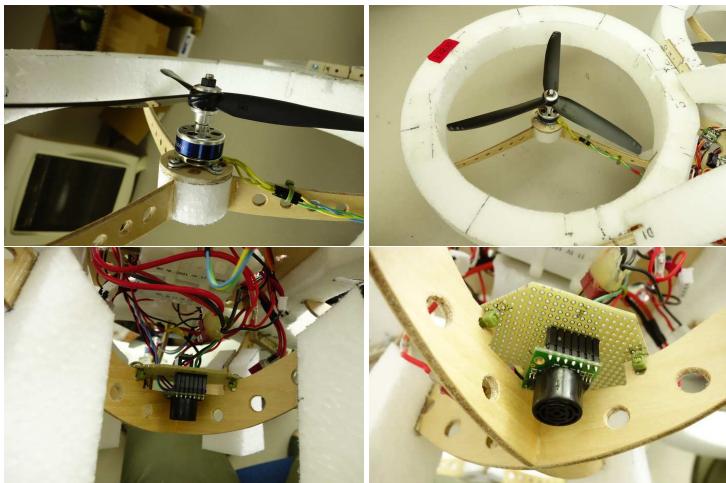
## 12.3 v12(8/21 2014)

11号機と同じナセルを用いて、実験用に12号機を制作。前述の要件を満たせば、屋内3mX3m程度の範囲でも安定する。

マウント部にワッシャー1枚つかって、ローター軸に、わずかに上反角をつけると下部側方の障害物を回避する空力効果がみられる。

12.2.7の要件を満たしていれば、各ゲインはさらに大きくしても安定するようである。

下部ドームにアナログ超音波センサーをつけて、高度を1.5mほどにキープするアルゴリズムを導入(haute12.10.2SAW.c)高度はある程度保持するが、上下振動を2~3回する。速い収束を得るには、高度履歴の差分、つまり微分項を高度制御に導入する必要があると考えられる。



## 12.4 13号機

13号機は12号機と同じ構成および構造で制作した。屋外で実験中に無線LANが途切れ、100メートルほど上昇ご墜落し、大破した。し

かし、BBB やアンプなど主要電子部品は無傷であった。

そのご、アルゴリズムをネットワークが途切れたら現状をキープする  
ように変更した。

## 12.5 8方向測距のための機体(v14.8.0,11/20 2014)

12号機、13号機で水平安定する飛行ロボットの基礎は確立できた。つぎに、自律的に障害物回避する飛行ロボットを目指す。そのためには、障害物までの距離を常に監視している必要がある。

障害物までの距離は超音波センサー(SRF02)を用いて測定する(測距センサー)。SRF02は、I2C通信によって測距データを出力するので、AD変換は必要ない。超音波を出すタイミングや、反射された超音波を観測するタイミングも自由に設定できる。また、比較的小型なので、室内で飛行ロボットが飛行できる程度の大きさの機体であっても、複数を設置することも容易である。

しかし、観測できる範囲はセンサーの前部30~40度ほどの範囲なので、たとえば4つの方向に設置されたSRF02では、360度の全方向にある障害物を感知するには無理がある。そこで、45度づつ方向をずらしてSRF02を機体の周囲に配置することで、周りの障害物を常に検知することを考える。

360度の範囲に渡って測距するためには、8つの超音波センサーが必要となる。12、13号機においてもそれは可能であるが、ナセルが機体を構成することが前提なので、設置がすこし難しい。また、ナセルがあることによって衝突や振動には耐性があるが、機体のサイズが約60cm四方程度、重量も約1kg程度となる。このサイズの飛行ロボットが普通の室内で飛行すると、飛行ロボット自身がつくりだした気流が部屋全体の気流をかき乱し、その影響を飛行ロボット自身が受けて常に不安定な運動が起こる。

そこで、機体を小型化し、なおかつ8つの超音波センサーを周囲に配置しやすい機体を考えた。サイズはローターのプロテクターを含めても40(cm)×40(cm)程度、重量は約300グラムと、12、13号機の約1/3の機体である。



図 12.27: 室内で安定に飛行可能であり、なおかつ 8 方向に測距センサーを配置が容易な機体。写真では、モーターの下部に 4 つの超音波センサーが設置されている。40cm x 40cm, 重量はバッテリー含めて約 300 g. バッテリーは 2 セル 7.4V, 600mAh だが、1 つだと BBB もふくめた機体全體に対して電流供給が不足ぎみで、モーターが破損する傾向があるため、2 個を並列使用。

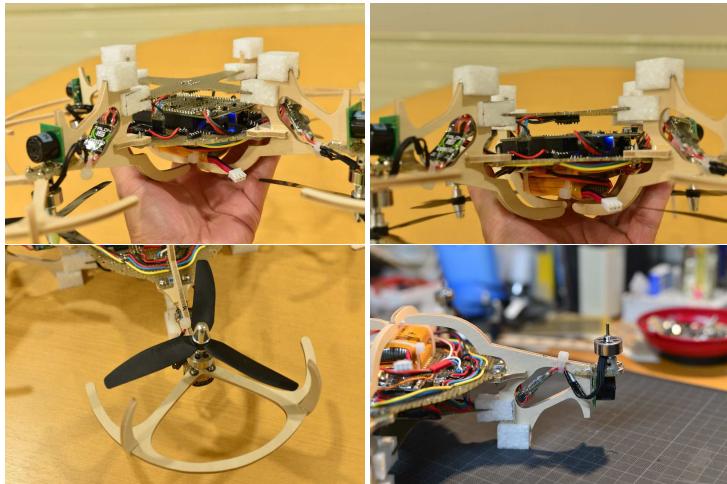


図 12.28: 左上: BBB のピンソケットに直接機体の基板を接続した。右上: 加速度とジャイロセンサーは機体下部（写真上部）に配置する。振動を抑えるために、脚部との接続は EPP を利用。左下: ローターは GWS5030 サイズの 3枚ペラ。プロテクターをモーターマウントに直接ネジ止め。着脱も容易である。右下: モーターはコスマテック CT1811-3800。非常に小型軽量だが、リード線がエナメル線でハンダ付けはしにくい。また電源供給が不安定だと破損するようだ。SRF02 はモーターアーム下部に配置。アームは気流の流れを考えて基板に対して縦に配置した。

## 12.6 15号機



図 12.29: 15.4 号機：超音波センサー (SRF02 を 4 方向). 3 葉の折プロペラ

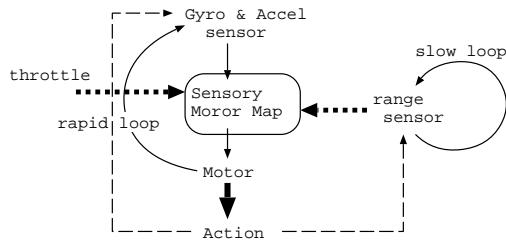


図 12.30: マルチスピード SMM

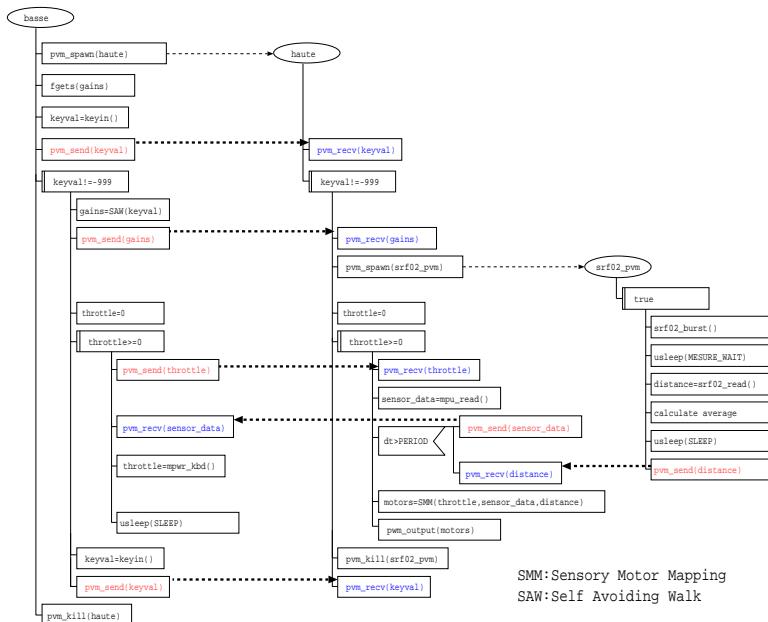


図 12.31: MS-SMM アルゴリズム

## 12.7 16号機

モーターを Hyperion1709-09 に戻し 14号機よりもわずかに大型化した。飛行重量は約 420 g で 11号機の約半分のサイズである。

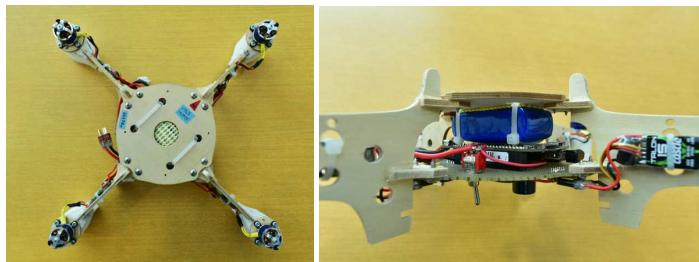


図 12.32: 16.2号機

機体構造を図 12.33 に示した構造に変更した。

アームの先端に EPP を接続し、その上にモーターマウントを接着した。つまり、モーターをアームに直接接続するのではなく EPP を介してマウントしてあるので、加速度センサーへのノイズ低減が期待できる。

アーム下部の脚部と EPP を同じ高さにすることにより、着陸時の衝撃を EPP を受け止めると同時にすべての衝撃を EPP で受け止めるのではなく、4つの脚部が直後に接地することで、衝撃を分散する効果を狙った。また、墜落時には、EPP が破損することで、他の部分の破損を最小化する。

プロペラは、破損が多いので、自作の折ペラとし、先端部（白）は CNC で自作切り出しとした。壁などに接触した際、プロペラが折れ曲がることで、破損を防止する。揚力発生効率は落ち、騒音も少し増えるが、飛行に大きな支障はない。破損した場合にも交換可能である。

特徴まとめは以下の通りである。

1. EPP でモーター振動吸収。
2. EPP 4点およびアーム（シナベニア）4点、合計8点で着地。

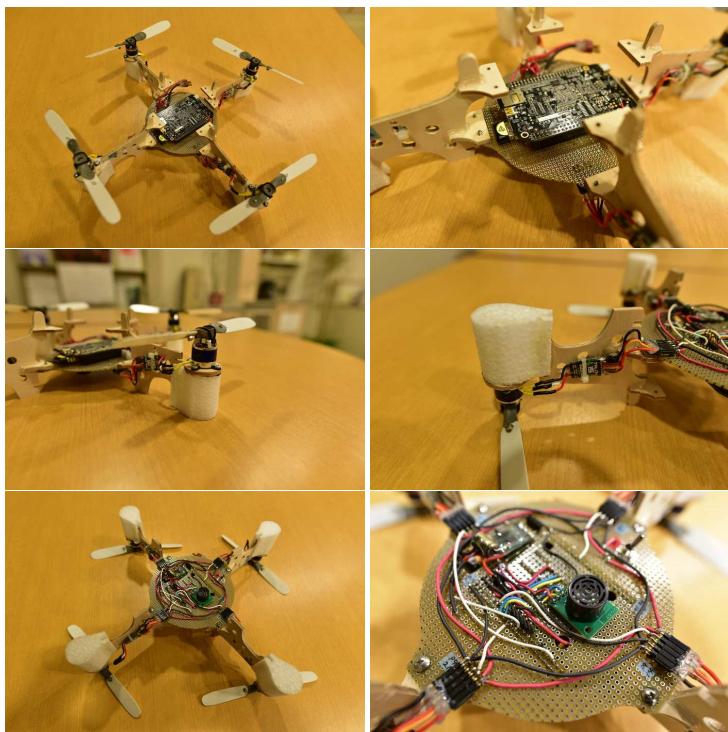


図 12.33: 16.2 号機の構造と部品配置

3. アームとして、シナベニアを縦に使うことで、気流の障害を最小化。
4. 加速度・ジャイロセンサー (MPU9150) は BBB と基板の間にハンダ付けし、露出しない。
5. シリアルポートのピンを基板裏に引き出すことで、シリアルコンソール接続が容易。
6. 基板裏に超音波センサー (SRF02) を直接ハンダ付け。(高度維持制御)  
改善点として、アンプコネクタを基板にハンダ付けする部分を、5mm  
ほど内側に寄せたい。基板とアーム接続部に干渉する。周囲の障害物と

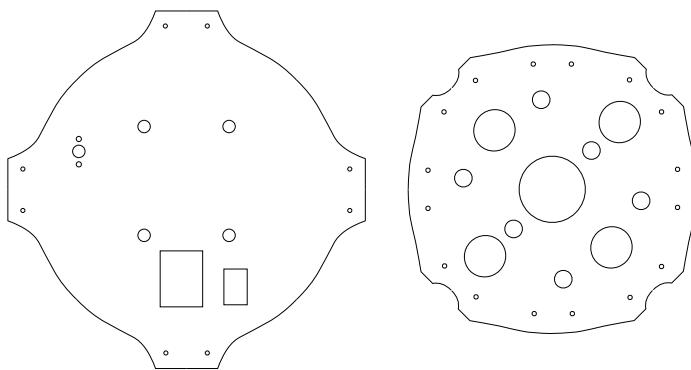


図 12.34: 16.2号機の基板部（ユニバーサル基板）およびバッテリマウント部品（4mm厚シナベニア）。

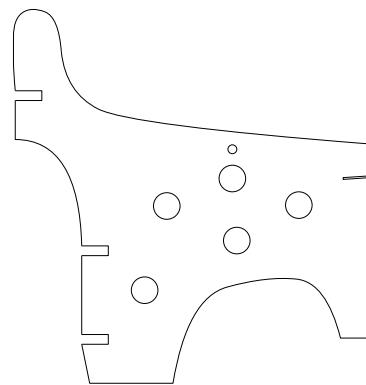


図 12.35: 16.2号機のアーム部品。4mm厚シナベニア

の距離を測定する超音波センサーは、この機体にさらに取り付ける予定のプロテクター部分に設置する予定なので、基板上での DC-DC コンバータや配線は余裕がある。

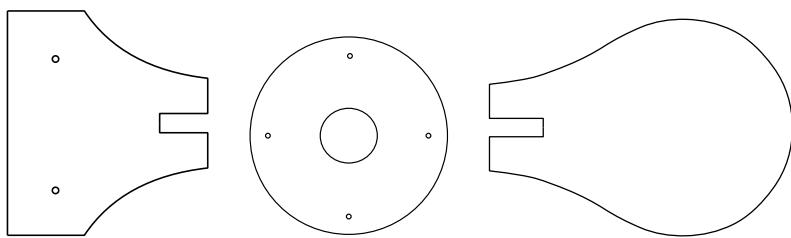


図 12.36: 左から、16.2号機の基板ーアーム結合部品（4mm厚シナベニア）。モーターマウント部品（4mm厚シナベニア）。モーターマウントEPP（40mmのEPP）。

## 付録A シリアルコンソール

ボードコンピュータのシリアルポートと、Debian PC の USB ポートを接続して、コンソールを利用する方法をここに説明する。ネットワークなど、コンソール以外でログインできない場合や、起動ログを詳しく見たい場合など、シリアルコンソールを利用する必要がある。

### A.1 シリアルポートの読み書き許可

Debian PC の USB ポートに USB-シリアル変換ケーブルを接続する。シリアルポートがブロックデバイスファイル/dev/ttyUSB0 として認識されている場合、

```
# chmod 666 /dev/ttyUSB0
```

を実行することによって、一般ユーザー権限で、シリアルポートを利用可能になる。

### A.2 minicom

利用するシリアルポートや通信設定を最初1度だけ行う。この設定は、後ほど変更したり、minicom を起動するたびに行なうこともできるが、よく使う設定は、最初に行って、保存しておくと便利である。

```
# minicom -s
```



図 A.1: minicom -s を実行した画面

矢印キーを使って、「シリアルポート」を選択するとシリアルポートの



図 A.2: シリアルポートを選択した画面

通信速度などが表示されるので、通信相手のボードコンピュータにおける設定と一致していることを確認する。

ハードウェアフロー制御をしていない（多くの場合）場合には”F”を入力して「いいえ」とする。リターンを入力し、最初の選択画面にもどる。これらの設定が後の minicom 起動時にも有効になるように、保存を選択する。



図 A.3: ”F”を入力し、ハードウェアフロー制御「いいえ」を選択



図 A.4: 「”dfl”に設定を保存」を選択し、設定を保存する。

設定が正常に行われると、ボードコンピュータのログインプロンプトが表示される。

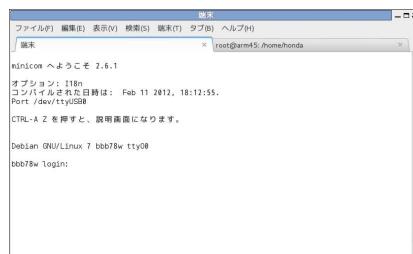


図 A.5: ボードコンピュータのログインプロンプトが表示される。

minicom は CTRL-A X と入力することによっても終了することができる。

## 付録B CNC ノート

CNC(Computer Numerical Control) は、高速回転するスピンドルの位置をコンピューターで数値制御することで、CAD などで設計した図を 3 次元オブジェクトとして削り出すことが出来る。

ここでは、CNC を使ってシナベニアから 2 次元的なパーツを切り出す方法を説明する。

### B.1 2次元データの作成 (tgif)

tgif を使って、2 次元図データを作成する。次に、PyCAM を使って、軌道データを作るために、図データを svg 形式でファイルにプリントアウトする。

### B.2 軌道データの作成 (PyCAM)

CNC は G コードと呼ばれる軌道データファイルで制御される。PyCAM というアプリケーションは svg などの 2 次元データから CNC を制御する G コードデータを自動的に作成する機能を持っている。

1. シェルから pycam と入力して PyCAM をはじめる。
2. svg データファイルを読み込む。
3. Model, Tool と希望するものを選択する。

4. 2次元データから G コードを生成するばあい、Processes で Gravure を選択する必要がある。
5. tasks タブで、Process を Gravure に選んで、Generate Toolpath を実行する。
6. toolpaths タブで、生成された path を選んで、simulate を実行すると、画面上でスピンドルが通る起動を事前に確認できる。
7. Export all を実行すると、ngc ファイル (G コード) が生成される。

### B.3 LinuxCNC

CNC 装置は LinuxCNC というソフトウェアで制御する。

# 付録C 起動時に実行されるシェルスクリプト

自分で作成したスクリプトを、Debian マシンの電源投入時や、reboot 時に自動的に実行されるようにする方法をここに示します。まず、シェルスクリプトをつくり、insserv あるいは update-rc.d コマンドを実行する必要があります。

## C.1 insserv

Debian 6 以降のバージョンでは、insserv コマンドを用います。Debian 6 以前のバージョンについては、update-rc.d を用います。

実行したスクリプト (hoge.sh) を作り /etc/init.d に置いて、insserv コマンドを実行します。

```
# mv hoge.sh /etc/init.d  
# insserv hoge.sh
```

find コマンドを使って、登録されたスクリプトを表示してみると。

```
# find /etc/ -name "*hoge.sh"
```

ランレベルに合わせて登録された結果が表示されます。

```
/etc/rc0.d/K01hoge.sh  
/etc/rc5.d/S16hoge.sh  
/etc/rc6.d/K01hoge.sh  
/etc/rc2.d/S16hoge.sh  
/etc/init.d/hoge.sh
```

ランレベルに応じて、複数のディレクトリにリンクが作成されます。

S ではじまるリンクが起動時に実行され、K ではじまるものが Debian 終了時に実行されます。終了時に特に実行する必要ない場合は、K ではじまるリンクを削除すれば、実行されません。

リンク名の先頭に”S16”などと記号と数字が付加されます。この数字が小さいものから自動的に実行されます。最後に実行したいスクリプトは、この数字をいちばん大きいものに変更しておきます。

たとえば、無線LANのiwconfigなどのスクリプトは、起動の一番最後に実行したほうが確実性が高いようです。

## C.2 update-rc.d

Debian 6 以前のバージョンについては、update-rc.d を用います。実行したスクリプト (hoge.sh) を作り /etc/init.d に置きます。その後、update-rc.d コマンドを実行します。

```
# update-rc.d hostname.sh defaults
```

## 付録D 数式処理 Maxima

行列の固有値と固有ベクトルを求める計算は、 $\hat{A}$  行列のようにその次元が  $2 \times 2$  なら手で紙の上で計算できる。しかし、時間遅れを含む遷移行列のように、その次元が  $4 \times 4$  になると、固有値、固有ベクトルをもとめることはかなりの計算量を要する。それは、不可能とは言わないまでも、計算間違いを犯す可能性も飛躍的に高まる。

数式処理ソフト Maxima<sup>1</sup> を使えば、容易に遷移行列の固有値、固有ベクトルを求めることが出来る。

### D.1 固有値と固有ベクトル

図 D.1 に、フィボナッチ型の遷移行列 ((5.50) 式参照) の定義と、その固有値、固有ベクトルを求めるバッチファイルを示した。/\* … \*/ の部

```
/* フィボナッチ型遷移行列 (4x4) */
TF:matrix([1, 0,-CP*D, -CI*D], \
          [0, 1,      D,      0], \
          [1, 0,      0,      0], \
          [0, 1,      0,      0]);
/* TF の固有値 eigTF と固有ベクトル vectTF を求める。 */
[eigTF,vectTF]:eigenvectors(TF);
```

図 D.1: Maxima で、フィボナッチ型遷移行列  $T_F$  の定義とその固有値、右固有ベクトルを求めるためのバッチファイル (Fibonatch.max)

---

<sup>1</sup>GNU ライセンスの数式処理ソフト (<http://maxima.sourceforge.net/>)

分はコメントを意味し、処理としては無視される。また、空白行も無視されるので、ある程度わかりやすい記述が可能である。ただし、スペースやタブを挿入するとエラーとなるので、注意が必要である。

シェルから Maxima を起動すると、起動コメントが数行表示された後、Maxima のプロンプトが表示される。そこにコマンドを打ち込んでいく形で、処理を進める（図 D.2 参照）。図 D.1 に示したバッチファイルを読

```
% maxima
Maxima 5.27.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) batch("Fibonatch.max");
```

図 D.2: Maxima の起動と、バッチ処理の読み込み

み込むことによって、数式処理が行うことが出来る。もちろん、Maxima のプロンプトに対してバッチと同じ内容をキーボードから打ち込んでも同じ処理が行われるが、ながい入力になるとこのようにバッチファイルを活用する方が効率の面でも、ミスを防ぐという面でも、また修正を加えていくという面においても有効である。

図 D.3 に固有値とそれに対応する右固有ベクトルの出力結果を示した。最初の 4 つの出力が 4 つの固有値としてコンマで区切られて出力されている。次に出力されている [1, 1, 1, 1] は、各固有地の縮退度（重複度）である。たとえば、異なる 2 つの固有ベクトルに対して、その固有値が全く同じならば、縮退度は 2 である。いまフィボナッチ型遷移行列  $T_F$  の縮退度は、すべて 1 であるので、すべての固有ベクトルに対してそれぞれ 1 つづつの異なる固有値が対応している。

それぞれの固有値、固有ベクトルは、変数名に引数を指定することによって参照することができる。例えば、3 番目の固有値は eigTF[1][3]

と指定する（図 D.4, 1 行目参照）。固有値は、固有値の値と、その縮退度がセットで出力されているので、固有値の値を指定するために、一番目の引数の値を 1 にする必要がある。ちなみに縮退度の値を指定するためには、その値を 2 にする。

その結果は図 D.5 の (%o78) の行に示した。2 重根号が含まれた一見複雑な形をしているが、この表式は  $\lambda_1$

$$\lambda_1 = \frac{1}{2} \left( \sqrt{C_p^2 - 4C_I} - C_p \right) \quad (\text{D.1})$$

を含んでいる ((3.41) 式参照)。 $\lambda_1$  は、ここ以外に共通してくり返し登場するので、その展開された表式ではなくて、ひとまとめの記号として  $\lambda_1$  で表したほうが、見通しがよく、理解しやすい。そこで、`ratsubst` コマンドをつかって、置き換えを行った結果が図 D.5 の (%o79) である。これは、確かに本文で示した時間遅れを考慮した遷移行列の固有値の一つ

$$\eta = -\frac{\sqrt{4\Delta t \lambda_1 + 1} - 1}{2} \quad (\text{D.2})$$

となっている ((5.61) 式参照)。

ここで、もう一度図 D.5 の (%o78) と (%o79) を見比べて欲しい。両者は同じものを表している。一つの記号  $L1(\lambda_1)$  を増やして、表式を簡略化しただけである。(%o78) 程度であれば、さほど複雑でもないので、その複雑さが理解や思考の妨げになるということはあまりないかもしれない。しかし、これが図 D.3 の様に、くり返し何度もあらわれて一見複雑な式の形をしているばかり、この置き換えによって見通しがずっと良くなる。Maxima も含めてコンピューターはこの様な置き換えは自動的に行なってくれない。数式処理を繰り返すと、どんどん式は長く展開されていき、煩雑さが増していく。一見単純な置き換えの様に見えるが、どの式の塊をひとかたまりととらえるかという部分に人間にしかない知能の謎が潜んでいるように思える。複雑なものでも、ひとかたまりとしてとらえ、それを使ってまた複雑なもの（式）を構成するという再帰的な作業によって、より複雑なことがらを記述したり理解する知能の鍵になっているかもしれない。

## D.2 右固有ベクトルと左固有ベクトル

右固有ベクトルは Maxima の中で行ベクトルとして表現されている(図 D.3 参照). これら 4 つのベクトルを並べて行列を作り, 転置(transpose)をとった行列は, ちょうど, 右固有ベクトルを列ベクトルにおして, 横に並べて作った変換行列  $\hat{V}$  となる(図 D.6 参照).

この行列  $\hat{V}$  の左から  $\hat{T}_F$  を掛けたもの(図 D.7, 1 行目)は, 各固有ベクトルにそれに対応する固有値を掛けて並べた行列になっているはずである. したがって, それを固有ベクトルで割ると, 固有値が得られるはずである. 図 D.7 の 2 行目に固有ベクトルの 1 番目の要素に関して, その割り算を求めた. ここで, %記号は, 1 行前の処理結果を意味している. `ratsimp(%), algebraic:true` という処理は, そのわり算の結果を, 分母を有理化する形で簡略化する処理を意味している. さらに前述の  $\lambda_1$  による置き換えを行った結果を図 D.8 に示した. たしかに図 D.5 で得られた第 3 番目の固有値と同じ値が確認できる.

非対称行列の左固有ベクトル  $\vec{u}$  は, 右固有ベクトル  $\vec{v}$  と異なる. ここで, 非対称行列である,  $\hat{T}_F$  の左固有ベクトルを求めてみる.

固有値を対角要素に持つ対角行列を次のように  $\hat{E}$  と表そう.

$$\hat{E} \equiv \begin{pmatrix} \eta_1 & 0 & 0 & 0 \\ 0 & \eta_2 & 0 & 0 \\ 0 & 0 & \eta_3 & 0 \\ 0 & 0 & 0 & \eta_4 \end{pmatrix} \quad (\text{D.3})$$

右変換行列  $\hat{V}$  は, 列ベクトルである右固有ベクトルを並べたものである.

$$\hat{V} = \left( \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{pmatrix} \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{pmatrix} \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{pmatrix} \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \\ \vec{v}_4 \end{pmatrix} \right) \quad (\text{D.4})$$

これを用いると,

$$\hat{T}_F \hat{V} = (\eta_1 \vec{v}_1 \ \eta_2 \vec{v}_2 \ \eta_3 \vec{v}_3 \ \eta_4 \vec{v}_4) \quad (\text{D.5})$$

$$= \hat{V} \hat{E} \quad (\text{D.6})$$

が成り立つ。ここで、左から  $\hat{V}$  の逆行列  $\hat{V}^{-1}$  を掛けると、

$$\hat{V}^{-1}\hat{T}_F\hat{V} = \hat{E} \quad (D.7)$$

さらに、

$$\hat{V}^{-1}\hat{T}_F = \hat{E}\hat{V}^{-1} \quad (D.8)$$

となることから、 $\hat{V}^{-1}$  が実は、行ベクトルである左固有ベクトル ${}^t\vec{u}$ を縦に並べたものに等しいことが分かる。

$$\begin{pmatrix} \left( \begin{array}{c} {}^t\vec{u}_1 \\ {}^t\vec{u}_2 \\ {}^t\vec{u}_3 \\ {}^t\vec{u}_4 \end{array} \right) \\ \left( \begin{array}{c} {}^t\vec{u}_1 \\ {}^t\vec{u}_2 \\ {}^t\vec{u}_3 \\ {}^t\vec{u}_4 \end{array} \right) \\ \left( \begin{array}{c} {}^t\vec{u}_1 \\ {}^t\vec{u}_2 \\ {}^t\vec{u}_3 \\ {}^t\vec{u}_4 \end{array} \right) \\ \left( \begin{array}{c} {}^t\vec{u}_1 \\ {}^t\vec{u}_2 \\ {}^t\vec{u}_3 \\ {}^t\vec{u}_4 \end{array} \right) \end{pmatrix} = \hat{V}^{-1} \quad (D.9)$$

となり、(D.7) 式から、この行列が  $\hat{T}_F$  の左変換行列  $\hat{U}$  である。

$$\hat{U} \equiv \hat{V}^{-1} \quad (D.10)$$

対象になっている行列が量子力学におけるハミルトニアンのようにエルミート行列であれば、左変換行列は右変換行列のエルミート共役行列となり、行列の対角化がユニタリー変換となる。しかし、非対称行列の場合には、ここに示したように、ユニタリー変換とはならないので、注意が必要である。

実際に Maxima を用いて逆行列  $\hat{U} = \hat{V}^{-1}$  を求める。左変換行列  $\hat{U}$  と、右変換行列  $\hat{V}$  はお互いに、逆行列の関係にあるはずであるから、かけあわせると、単位行列  $\hat{I}$  となる(図 D.10 参照)。ちなみに、Maxima の ratsimp 関数を用いない場合、このように 0 と 1 だけの簡単化された結果ではなく、煩雑な表式のまま結果が表示されるので、注意が必要である。

### D.3 4つの固有値

Maxima で求められるフィボナッチ型遷移行列の 4 つの固有値は、それぞれ

$$\eta_1 = -\frac{\sqrt{4 \Delta t \lambda_2 + 1} - 1}{2} \quad (\text{D.11})$$

$$\eta_2 = \frac{\sqrt{4 \Delta t \lambda_2 + 1} + 1}{2} \quad (\text{D.12})$$

$$\eta_3 = -\frac{\sqrt{4 \Delta t \lambda_1 + 1} - 1}{2} \quad (\text{D.13})$$

$$\eta_4 = \frac{\sqrt{4 \Delta t \lambda_1 + 1} + 1}{2} \quad (\text{D.14})$$

である。このなかで、系の振る舞いを特徴づける  $\eta = 1$  の近傍の解は、 $\eta_2, \eta_4$  である。

### D.4 遷移行列 $\hat{T}$ の固有値

Maxima を用いて、 $\varepsilon = 1$  の場合の遷移行列  $\hat{T}$  の固有値を求めることは可能である。たとえば、その第 1 番目の固有値は

$$\begin{aligned} \eta_1 &= \left( -\frac{\sqrt{3}i}{2} - \frac{1}{2} \right) \\ &\times \left( \frac{\Delta t \sqrt{27 C_I^2 \Delta t^2 + (4 C_P^3 - 18 C_I C_P) \Delta t - C_P^2 + 4 C_I}}{2 3^{\frac{3}{2}}} \right. \\ &- \left. \frac{27 C_I \Delta t^2 - 9 C_P \Delta t + 2}{54} \right)^{\frac{1}{3}} \\ &- \frac{\left( \frac{\sqrt{3}i}{2} - \frac{1}{2} \right) (3 C_P \Delta t - 1)}{9 \left( \frac{\Delta t \sqrt{27 C_I^2 \Delta t^2 + (4 C_P^3 - 18 C_I C_P) \Delta t - C_P^2 + 4 C_I}}{2 3^{\frac{3}{2}}} - \frac{27 C_I \Delta t^2 - 9 C_P \Delta t + 2}{54} \right)^{\frac{1}{3}}} \\ &+ \frac{2}{3} \end{aligned} \quad (\text{D.15})$$

である。しかし、かなり煩雑な形をしており、フィボナッチ型の遷移行列の固有値  $\lambda$  を用いた表式にすることができない。したがって、 $\lambda$  の繰りこみ変換という形で時間遅れを考慮することは困難である。

## D.5 固有値の1次摂動

2番めの固有値に対する、1次摂動は、

$$\begin{aligned}
 & (\hat{U} \hat{H} \hat{V})_{2,2} \\
 &= -\frac{\Delta t \left( 2\lambda_2^2 \sqrt{4\Delta t \lambda_2 + 1} + 2C_P \lambda_2 \sqrt{4\Delta t \lambda_2 + 1} + C_I \sqrt{4\Delta t \lambda_2 + 1} + 4C_I \Delta t \lambda_2 + C_I \right)}{2(2\lambda_2 + C_P)(4\Delta t \lambda_2 + 1)} \\
 &= -\frac{\Delta t \left( 2\lambda_2^2 \sqrt{4\Delta t \lambda_2 + 1} + 2C_P \lambda_2 \sqrt{4\Delta t \lambda_2 + 1} + C_I \sqrt{4\Delta t \lambda_2 + 1} + C_I(4\Delta t \lambda_2 + 1) \right)}{2(2\lambda_2 + C_P)(4\Delta t \lambda_2 + 1)} \\
 &= -\frac{\Delta t \left( 2\lambda_2^2 + 2C_P \lambda_2 + C_I + C_I \sqrt{4\Delta t \lambda_2 + 1} \right)}{2(2\lambda_2 + C_P) \sqrt{4\Delta t \lambda_2 + 1}} \quad (D.16)
 \end{aligned}$$

$$\begin{aligned}
 & \frac{d(\hat{U} \hat{H} \hat{V})_{2,2}}{d\Delta t} \\
 &= -\frac{C_I S^2 + \{4\Delta t \lambda_2^3 + 2(2C_P \Delta t + 1)\lambda_2^2 + 2(C_I \Delta t + C_P)\lambda_2 + C_I\} \sqrt{S}}{2(2\lambda_2 + C_P) S^2} \quad (D.17)
 \end{aligned}$$

$$S = 4\Delta t \lambda_2 + 1 \quad (D.18)$$

```

1   sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1)
(%o4) [[[- -----,
2
2
sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1)   1
----- + -,
2
2
1   sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1)
-----,
2
2
sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1)   1
----- + -], [1, 1, 1, 1]],
2
2
sqrt(CP - 4 CI) - CP   2
[[[1, -----, - ((sqrt(CP - 4 CI) - CP)
2 CI
2
sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1) + sqrt(CP - 4 CI) - CP)/(4 CI D),
((CP sqrt(CP - 4 CI) - CP + 2 CI) sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1)
+ CP sqrt(CP - 4 CI) - CP + 2 CI)/(4 CI D)],
2
sqrt(CP - 4 CI) - CP   2
[[1, -----, ((sqrt(CP - 4 CI) - CP)
2 CI
2
sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1) - sqrt(CP - 4 CI) + CP)/(4 CI D),
- ((CP sqrt(CP - 4 CI) - CP + 2 CI) sqrt(- 2 sqrt(CP - 4 CI) D - 2 CP D + 1)
- CP sqrt(CP - 4 CI) + CP - 2 CI)/(4 CI D)],
2
sqrt(CP - 4 CI) + CP   2
[[1, -----, ((sqrt(CP - 4 CI) + CP)
2 CI
2
sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1) + sqrt(CP - 4 CI) + CP)/(4 CI D),
- ((CP sqrt(CP - 4 CI) + CP - 2 CI) sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1)
+ CP sqrt(CP - 4 CI) + CP - 2 CI)/(4 CI D)],
2
sqrt(CP - 4 CI) + CP   2
[[1, -----, - ((sqrt(CP - 4 CI) + CP)
2 CI
2
sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1) - sqrt(CP - 4 CI) - CP)/(4 CI D),
((CP sqrt(CP - 4 CI) + CP - 2 CI) sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1)
- CP sqrt(CP - 4 CI) - CP + 2 CI)/(4 CI D)]]]]

```

図 D.3: 固有値と右固有ベクトルの出力結果. 固有値, 固有値の縮退度, 固有ベクトル（行ベクトル）の順に出力される.

```
eigTF[1][3];
eig3:ratsubst(2*L1,sqrt(CP^2-4*CI)-CP,eigTF[1][3]);
```

図 D.4: 3番目の固有値の表示、およびその簡略化置き換え(ratsubst)

```
(%o78)      2
              1   sqrt(2 sqrt(CP - 4 CI) D - 2 CP D + 1)
      - -----
              2                   2
              2
(%i79)      eig3 : ratsubst(2 L1, sqrt(CP - 4 CI) - CP, (eigTF ) )
                           1
                           3
              sqrt(4 D L1 + 1) - 1
      - -----
                           2
(%o79)
```

図 D.5: フィボナッチ型遷移行列の3番目の固有値とそれを ratsubst コマンドで簡略化した結果

```
V:transpose(matrix(vecTF[1][1],vecTF[2][1],vecTF[3][1],vecTF[4][1]));
```

図 D.6: 右固有ベクトルから変換行列  $\hat{V}$  をつくる。

```
/*右固有ベクトル V の確認*/
TF.V;
%[1][3]/V[1][3];
ratsimp(%),algebraic:true;
ratsubst(2*L1,sqrt(CP^2-4*CI)-CP,%);
```

図 D.7: 右固有ベクトルが固有ベクトルになっていることの確認

```
(%i50)      ratsubst(2 L1, sqrt(CP - 4 CI) - CP, %)
              sqrt(4 D L1 + 1) - 1
(%o50)      -----
                           2
```

図 D.8: フィボナッチ型行列とその右固有ベクトルの掛け算から得られた固有値の確認結果.

```
U:V^-1;
/* 逆行列の確認 */
U.V;
ratsimp(%);
```

図 D.9: 左変換行列  $\hat{U}$  を求める.

```
(%i104)      ratsimp(%)
              [ 1  0  0  0 ]
              [           ]
              [ 0  1  0  0 ]
              [           ]
(%o104)      [ 0  0  1  0 ]
              [           ]
              [ 0  0  0  1 ]
```

図 D.10: 左変換行列と右変換行列の積が単位行列となることの確認結果.

## 付録E 分散バージョン管理システム gitlab

gitlab とは、ソースコードなどを開発する際のバージョン管理を WEB 上 (<https://gitlab.com/>) で行うサービスである。従来の中央管理型を始めとした様々ななかたちのワークフローを実現できる。それ故に、理解のしやすい側面があるが、ここでは基本的な利用の仕方を述べる。

gitlab では、基本的な機能は無料で自由に使える。大まかな流れは、

1. ユーザーの登録
2. プロジェクトの作成
3. git add, commit, push
4. git pull

である。この他にも、もちろん多くの機能があるが、ここでは割愛する。ユーザー登録は、最初の sign in の際だけ必要である。ここでは、ローカルな Debian 上にあるプログラムなどを gitlab.com のプロジェクトとして管理する方法を示す。

### ユーザーの登録

<https://gitlab.com> をアクセスすると右上に Sign in ボタンが表示されるので、クリックする。初めての場合は Sign up の項目に必要事項を入力し、gitlab への登録をおこなう。以降は同じユーザー名で Sign in できる。

セキュア通信を行うために鍵の生成と登録を行う。ローカル Debian で以下を実行し公開鍵を表示する。

```
$ ssh-keygen -t rsa -C "$your_email"
$ cat ~/.ssh/id_rsa.pub
```

gitlab の Dashboard 表示の状態で、右上にある Profile setting のアイコン  をクリックする。左端の列に  が現れるので、クリックする。右上の **Add SSH Key** をクリックすると、title と key の入力画面になるので、key の枠に先ほど cat したキーをコピペーストする。ssh- ではじまり、ユーザー名で終わるすべての部分をペーストする必要がある。title も入力する。以上で gitlab とファイルをやり取りするための準備が完了した。

## プロジェクトの作成

Dashboard 右上の  をクリックしてプロジェクトをつくる。あたらしいプロジェクトが作られると以後に必要な作業が gitlab 上に表示されるので、それにしたがう。

```
Git global setup

git config --global user.name "ユーザー名"
git config --global user.email "メールアドレス"

Create a new repository

mkdir プロジェクト名
cd プロジェクト名
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitlab.com:ユーザー名/プロジェクト名.git
```

一連の作業の概略を図 E.1 に示した。gitlab では Working files をいきなり gitlab.com に送ることはしない。作業履歴を History の中に保存してから push を行う。git commit では、かならずコメントをつける必要がある。つまり、編集作業の内容を記録する必要がある。さらに、gitlab では

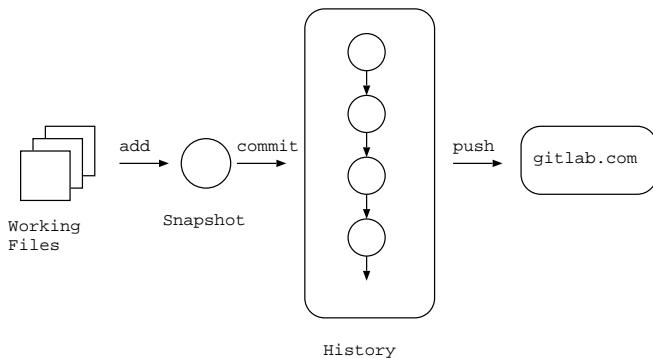


図 E.1: gitlab ワークフローの概略

History に commit するまえの段階に snapshot が存在する。snapshot が存在することにより、作業内容ごとに add をして、その作業に対してコメントを付して履歴を残すことが可能となる。逆に言えば、作業内容を気にせずに working file の編集を行うことが出来る。

### **git add, commit, push**

ローカル Debian 上でファイルを編集などした後、gitlab にプロジェクトを登録するには以下を行う。

```

$ cd プロジェクト名
$ git add -A
$ git commit -m "コメント"
$ git push origin master
    
```

### **git pull**

gitlab.com 上で行った編集をローカル Debian に反映するには、git pull を実行する。

```
$ cd プロジェクト名  
$ git pull origin master
```

gitlab.com 上で編集が行われた際に、git pull を行わずに git push を行うと、履歴に齟齬が生じたとみなされて push が拒否される。その際には、一旦 git pull を行なってから git push を行う。

## 付録F 日本語環境

vi や gvim を使って文章ファイル（例:body.tex）を作って、それを pdf ファイルに変換します。body.tex ファイルがプログラムで、それをコンパイルして pdf ファイルを作るようなイメージです。

### F.1 LaTeX

発表会の原稿や卒業論文、修士論文は LaTeX をつかって書きます。

#### F.1.1 インストール

```
# apt-get install ptex-bin
```

その他、フォントなど deb パッケージ化されているので、apt-get や aptitude でインストールできる。

#### F.1.2 作業の流れ

以下に、作業の流れを箇条書きします。

1. plateX

body.tex という LaTeX ソースを dvi に変換します。

```
$ plateX body.tex
```

## 2. 画像ファイル

pdf の原稿に埋め込む画像の解像度は  $4000 \times 3000$  などの高解像度なものは必要ないので、幅 1000 ピクセルぐらいに縮小します。

```
$ mogrify -resize 1000 hogehoge.jpg
```

大きなサイズのまま、tex -i pdf ファイルと処理をすると、CUP にも無駄な負担がかかり、時間を要します。

画像ファイルは convert コマンドで eps ファイルに変換します。

```
$ convert hogehoge.jpg eps2:hogehoge.eps
```

この例のように、eps をレベル II に指定して圧縮しないと、ファイルサイズが非常に大きくなってしまうので、後々操作に支障を来す場合もあるので、eps2 を指定します。

## 3. dvipdfmx

platex によって生成された dvi ファイルを pdf ファイルに変換します。

```
$ dvipdfmx body.dvi
```

線が細すぎるなどの warning が出る場合には以下を試します。

```
$ dvipdfmx -d 5 body.dvi
```

## 4. dvips, ps2pdf

tgif などで作成した図の中に日本語を組み込むためには dvips, ps2pdf をつかいます。

```
$ dvips hogehoge.dvi  
$ ps2pdf hogehoge.ps
```

### 5. EPS ファイルを組み込む

\includegraphics を使って、EPS 画像を LaTeX に組み込みます。このとき、\psfrag をつかうと、半角文字を日本語などに置き換えることができます。

```
\usepackage[dvips]{graphicx,psfrag}
...
\begin{figure}[h]
\psfrag{porco}{ポルコ}
\includegraphics[width=\textwidth]{hogehoge.eps}
\caption{\label{fig:porco-fig}ポルコの図}
\end{figure}
```

ただし、dvips と ps2pdf を使って PDF ファイルを生成する必要があります。dvipdfmx でこの psfrag 機能を利用することは出来ません。

また、dvips と ps2pdf を用いると、図の位置が適性な位置に表示されない場合があるので、注意が必要です。その場合には、dvips と ps2pdf を用いて図だけの eps ファイルを作り、それを全体の文章ファイルに読み込み、その後で dvipdfmx を用いると位置が適性化されます。

## F.2 日本語入力

ロボット知能とは直接関係ないが、プログラムや README ファイルにコメントを記入する場合にも、日本語が使えると便利です。キーボードから日本語を入力するためには、日本語入力 IME(Mozc や Anthy など)を起動しなければなりません。日本語入力 IME を起動するためには、半角／全角キーあるいはシフト＋スペースキーなどを押します。これらの設定は変更することもできます。

## F.2.1 iBus をインストールする

インプットメソッドインターフェイスとして、iBus をインストールします。

```
# apt-get install ibus
# apt-get install ibus-mozc
```

IME に依存しない一般的な設定などを変更できるようになります。

## F.2.2 日本語入力方法を選択する

半角／全角キーなどを押して、日本語入力モードに入るためには、日本語入力方法が登録されている必要があります。iBus の設定で、日本語入力方法を追加します（図 F.1 参照）。Mozc というのは、Google 日本語



図 F.1: iBus の設定。インプットメソッドとして Mozc や Anthy を追加する。

入力のオープンソース版という位置づけの日本語入力です。辞書として「もしかして」機能など、豊富な辞書や学習機能が特長です。

すでに Mozc が登録されている場合は、この操作は必要ありません。

### F.2.3 Mozc で句読点を変更

科学技術関連のプレゼンテーションや、論文などでは、句読点としてコンマとピリオド「.,」が用いられることが多いです。Mozc でつかう句読点を「.,」に変更します。

日本語入力モードに入ると、デスクトップ上のパネルに Mozc のアイコンが表示されるので、クリックするとプルダウンメニューからプロパティ画面を開けます。

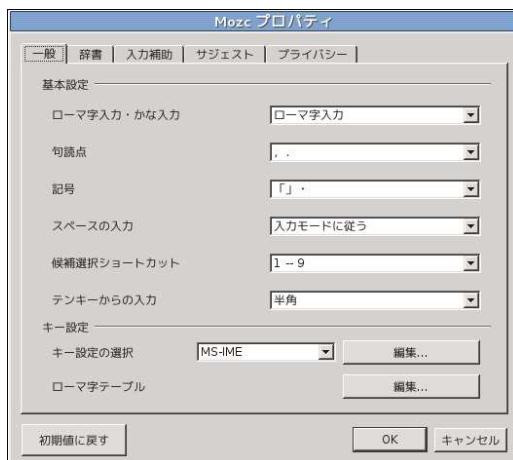


図 F.2: Mozc プロパティ 変更

しかし、句読点を変更しようとすると、Mozc プロパティ画面が突然消えることがあります。これは、uim-mozc と競合していることが原因のようです。以下のように、uim-mozc を削除すると、正常に Mozc のプロパティが変更できるようになります。

```
# aptitude purge uim-mozc
```



## 付録G モーションキャプチャー

V棟1階のVRシアター内にある、モーションキャプチャーシステムの使い方を、ここにまとめる。あくまでも、簡易的なもので、すべての機能を網羅するものではない。このマニュアルを読んで分かることは、システムを起動し、専用のマーカーを付した物体の位置データを取得する方法である。

大まかな作業の流れとしては、

1. 機器接続
2. 作業フォルダ作成（指定）
3. キャリブレーション（2回め以降省略可）
4. マーカーセット作成（指定）
5. 位置データ観測

なかでも、マーカーセットの作成が多少煩雑（バグも潜んでいる可能性大）である。ただし、マーカーセットの作成（指定）が正常に行われていないと、位置データの観測記録ができない。

### G.1 カメラ・ハブ・PCの接続

カメラとPCはLANを通じて接続される。カメラは8台あるので、ハブとカメラの電源システムのスイッチを入れる（電源コンセントを挿す）ハブとPCをLANケーブルでつなぐ。

## G.2 アプリケーション起動

PC の USB ポートにマスタードングルを挿し、Cortex3.0 を起動する。カメラからデータを取得するためには、マスタードングルを挿してある必要がある。取得されたデータを、あとから編集すること（Postprocess）は、Postprocess 用のドングルを挿すことで可能となる。

## G.3 作業フォルダ

マーカセットや取得位置のデータが保存されるフォルダを指定する。複数の作業データが混在すると、混乱を招くので、作業ごとに新しいフォルダを作り、それらを保存することが推奨される。以下が手順概要。

1. Quick File
2. New Folder
3. Set as Working Folder

## G.4 Live mode と Postprocess

Cortex での作業は、Live mode と Postprocess の 2 種類に大別される。Live mode では、実際にマーカの位置をパネル（画面）にリアルタイム表示などが可能である。Postprocess では、マーカセットの登録や、記録されたデータの編集などが可能である。

## G.5 キャリブレーション

マーカ位置を測定するまえに、キャリブレーションを行うことが推奨されている。カメラの位置が固定されており、前回の作業時とまったくずれていなければ、キャリブレーションは必要ないが、カメラは微妙に

ずれるようで、データの分散が大きくなってくると、固定されたカメラシステムでも、再びキャリブレーションを行う必要がある。

1. Connect To
2. play ボタン押す
3. L 字フレーム以外のマーカーが入らないようにする。
4. Calibrate
5. Initial Calibration
6. Next
7. Wand  
すべてのカメラ読み込み領域が緑色になるまで、Wand を振り回す。
8. Finish

## G.6 表示空間の移動

拡大、縮小は表示パネルで、マウスのスクロールボタンを使っておこなう。Alt キーを押しながら、マウスボタンのドラッグをすると、表示空間を移動できる。

## G.7 マーカセットの作成

1. マーカの記録  
Live モードで数秒間データを記録
  - (a) File Name 入力
  - (b) Return
  - (c) 赤丸ボタン押す

(d) もう一度赤丸ボタンで記録終了

## 2. New marker set

ここが、鬼門です。

(a) Load Last ボタン

記録したデータが Postprocess に読み込まれる。

(b) Marker set 横の New タブ

(c) Name 入力

(d) Marker set Type は、Template or Prop.

マーカー間隔が伸び縮みする可能性がある場合や、Link を自由に作りたい場合 Template を選択。いっぽう、マーカー間隔が一定で、ひとかたまりとして認識したい場合は Prop を選択。

Create ボタンを押しても入力窓は閉じないので、× ボタンで閉じる。

(e) Capture: を選び直す。

これを行わないと、Marker の名前入力ボックスが現れない場合がある。原因は謎。

(f) 右側のパネル内でマーカーの個数だけ名前を付ける。

## 3. Quick ID

Quick ID をクリックすると、自動的に順番にマーカーをアイデンティファイできる。File メニューから Save capture を選択して保存。(ここでの保存は必要ないかもしれない)

## 4. Link の作成

Marker set type が Prop. の場合は、自動的にマーカー間にリンクが生成するので、この手順は必要ない。Marker set type が Mars の場合、任意のマーカー間にリンクを作成できる。

(a) 右側の枠内で、Links タグを選ぶ。

- (b) Create Links for Template をクリック
  - (c) 色を選択
  - (d) マーカー間をドラッグして link でつなぐ
5. Create Template

Create Template ボタンをクリックしてから、再度 Save capture する。ここまでやってはじめて、marker set が prop あるいは mars ファイルに保存されて、データ取得時に認識される。

## G.8 データの取得

1. Live モード
2. Marker set の Add/Remove タブ
3. 使うマーカーセットをチェック
4. Connect To
5. Play ボタン
6. Name 入力
7. レコード（赤丸ボタン）
8. File → Save Capture
9. Work Folder 内に保存されて trc ファイルに ascii 形式の 3D データが保存される。



# 付録H pdfで使うフォント

## H.1 dvipdfmx でフォントを指定する方法

latex の文書ファイルなどが置いてあるカレントディレクトリ内に、たとえば次のような motoya.map というフォントマップファイルを作る。左端

---

1	rml	H	NFa1kp.ttc
2	rmlv	V	mika.ttf
3	gbm	H	MTLmr3m.ttf
4	gbmv	V	MTLmr3m.ttf

図 H.1: モトヤフォントをつかった、map ファイルの例

の列は行番号なので、マップファイルそのものには入力する必要はない。

NFa1kp.ttc, MTLmr3m.ttf というのが、TrueType フォントが格納されているファイル名である。それらのフォントファイルは、map ファイルとおなじカレントディレクトリに置く。各原稿のディレクトリにこれらの TrueType フォントファイルを置くと、ディスクの使用量が増えるので、/usr/share/fonts/truetype ディレクトリなどにファイル置いておいて、各ディレクトではそのファイルへのシンボリックリンクを張るとよいであろう。

次に、dvipdfmx コマンドで pdf ファイルを作る際に、例えば次のように

```
$ dvipdfmx -f motoya.map body3nakazawa2up.dvi
```

とフォントマップを指定する。

## H.2 ps2pdfでフォントを指定する方法

ps2pdf コマンドは、ghostscript 付属のポストスクリプトファイルを PDF ファイルに変換するコマンドである。このコマンドが使う日本語フォントの指定は、

/var/lib/ghostscript/fonts/cidfmap というファイルの中でおこなう。

まず、念のためもとの cidfmap ファイルをバックアップする。

```
# cd /var/lib/ghostscript/fonts
# cp cidfmap cidfmap.backup
```

dvips コマンドなどでできた PS ファイルの中身を参照してみれば分かることであるが、たとえば、明朝体のフォントは Ryumin-Light とうフォントを使うことが指定されている。cidfmap ファイルのなかでは、

```
/Ryumin-Light /Japanese-Mincho-Regular
```

となっているので、/Japanese-Mincho-Regular が使われることがわかる。さらに、

```
/Japanese-Mincho-Regular << /FileType /TrueType /Path (/usr/share
/fonts/truetype/motoya/NFalkp.ttc) /SubfontID 0 /CSI [(Japan1) 4]
>> ;
```

と指定されている。実際は 1 行であるが、長いので、折り返して 3 行で記述した。つまり、日本語フォントは、

/usr/share/fonts/truetype/motoya/NFalkp.ttc を使うことが指定されている。

ここに記述されたディレクトリと ttc(あるいは ttf) ファイル名を、別のものに書き換えればそれが ps2pdf を実行した時に使われる、すなわちフォントがそれに代わる。

ここでは、明朝体についてだけ説明したが、ゴシック体についても同様である。

なお、Debian パッケージに含まれるフリーの TrueType フォントに関してはライセンス的に問題がないと考えられるが、ネットワークなどからダウンロードしたフォントを使う場合には、ライセンス条項に注意する必要がある。フリーフォントといっても、個人の利用の範囲であればライセンス上問題ないが、商用に利用する場合には問題がある場合もあるので注意が必要である。



## 関連図書

- [1] 「岩波講座ロボット学4 ロボットインテリジェンス」浅田 稔, 國吉 康夫 著, 岩波書店
- [2] 「オイラーの贈物（人類の至宝  $e^{i\pi} = -1$  を学ぶ）」吉田 武 著, 東海大学出版会
- [3] 「系統的プログラミング／入門」N. Wirth 著, 野下浩平, 篠 捷彦, 武市正人 共訳, 近代科学社
- [4] 「プログラム書法」B.W. Kernighan and P.J. Plauger 著, 木村 泉訳, 共立出版
- [5] 「行列の固有値」F. シャトラン著, 伊理正夫, 伊理由美訳, シュプリンガー・フェアラーク東京
- [6] 「強化学習」Richard S. Sutton and Andrew G. Barto 著, 三上 貞芳, 皆川 雅章 共訳, 森北出版
- [7] 「力学」ランダウ＝リフシツ 理論物理学教程, 広重徹, 水戸巖訳, 東京図書
- [8] 「回転翼飛行ロボットの時間遅れ運動制御」橋本理寛, 本田泰, 第18回交通流のシミュレーションシンポジウム(2012)論文集, 33–36
- [9] 「回転翼飛行ロボットの時間遅れ運動制御シミュレーション」佐藤 宏樹, 橋本理寛, 本田 泰. 第19回交通流のシミュレーションシンポジウム(2013)論文集, 45–48

- [10] 「時間遅れの繰り込みによる感覚行動系の安定性」本田泰, 第19回交通流のシミュレーションシンポジウム(2013)論文集, 53–56
- [11] 「数値計算」川上一郎著, 岩波書店
- [12] 「物理のための数学」和達三樹著, 岩波書店
- [13] 「UNIX ワークステーションによる科学技術計算ハンドブック」戸川隼人著, サイエンス社
- [14] 「ニューメリカルレシピ・イン・シー(C言語による数値計算のレシピ)」ウィリアム・H・プレス／丹慶勝市著, 技術評論社
- [15] 「BeagleBone Blackで遊ぼう！」米田聰, Rutles

# 索引

- .exrc, 191
- .gvimrc, 191
- /etc/hosts, 196
- /etc/network/interfaces, 195
- /etc/resolv.conf, 197
- apt-cache, 189
- apt-cdrom, 188
- apt-get, 189
- BeagleBone Black, 227
- capemanager, 243
- CNC, 345
- convert, 364
- dd, 211, 231
- dmesg, 229
- double 型, 88
- dpkg, 188
- dvipdfmx, 364
- dvips, 364
- EPS 出力, 292
- ev3dev, 211
- exports, 198
- fit, 303
- float 型, 87
- fsync, 248
- gnuplot, 283
- gvim, 194
- i2c, 241
- iBus, 366
- ifconfig, 196, 257
- includegraphics, 365
- insserv, 347
- iwconfig, 255
- iwlist, 257
- kernel, 200
- LEGO, 204
- load, 291, 292
- logscale, 292
- LR 法, 112
- LU 分解, 107
- make-kpkg, 200
- minicom, 341
- mount, 199

- NFS, 198
- NXC, 205
- NXT, 203
- PAD 図, 143, 205
- PID 制御, 162
- ping, 197
- platex, 363
- plot, 286
- proxy, 198
- ps2pdf, 364
- psfrag, 365
- PyCAM, 345
- replot, 292
- route, 258, 261
- Ryumin-Light, 376
- SRF02, 249
- terminal, 290
- title, 290
- unzip, 211
- update-rc.d, 348
- vi, 194
- xlabel, 287
- xrange, 288
- ylabel, 287
- yrange, 288
- アルゴリズム, 205
- アンダーフロー, 92
- インデント, 73
- 打ち切り誤差, 95
- オイラー法, 141
- オーバーフロー, 92
- 階乗, 95
- 感覚運動写像, 162
- 感覚行動写像, 36
- ガウスの消去法, 101
- 逆行列, 110
- 行列の指數関数, 54
- クラウトのアルゴリズム, 109
- ケープマネージャ, 232
- 構造化プログラミング, 70
- 後退代入, 104
- 興奮性結合, 37
- 固有値, 111
- 固有ベクトル, 111
- 誤差, 94
- 最小二乗法, 121
- 修正オイラー法, 150
- 身体性, 31, 35
- 循環小数, 91
- 自由振り子, 151
- 数値誤差, 148
- スプライン補間, 126
- 遷移行列, 66
- 漸化式, 73, 96
- 前進消去, 102

- 相空間, 62
- 走性, 37
- 相対誤差, 94
- 対数スケール, 292
- 単位行列, 66
- ティラー展開, 140
- 二進数, 88
- 二分法, 99
- ニュートン・ラフソン法, 72
- 反復法, 73
- 微分方程式, 140
- ピボット選択, 107
- フィボナッチ遷移行列, 172
- 浮動小数点, 86
- ブロックデバイス, 229
- 丸め誤差, 95
- 有効精度, 95
- ラグランジアン, 151
- ランダウの  $O$  記号, 148
- ルンゲ・クッタ法, 135

Y. Honda  
2017 1/24