特別研究報告書

非線形最適制御および非線形最適状態推定
シミュレーションプログラム自動生成

指導教官
大塚 敏之 講師

報告者
児玉 明

平成 12 年 2 月 25 日

大阪大学 工学部 電子制御機械工学科 機械制御学領域

目 次

要旨

第 1 章　序論 **4**

第 2 章　問題の定式化とアルゴリズム **5**
2.1　一般的な最適制御問題 5
2.2　オイラー・ラグランジュ方程式 5
2.3　Backward Sweep 法 6
2.4　非線形 Receding Horizon 制御 7
2.5　非線形 Receding Horizon 状態推定 11

第 3 章　自動生成システム **15**
3.1　システムの働き 15
3.2　システムの構成 15
3.3　入力ファイル 16
3.4　Mathematica プログラムの実行手順 18
3.5　Splice 18
3.6　CAssign 18
3.7　Optimize 20
3.8　出力ファイル 21

第 4 章　シミュレーションプログラム **22**
4.1　状態フィードバックプログラムの説明 22
4.2　状態推定プログラムの説明 25

第 5 章　結論 **29**

謝辞

付 録 A　入力ファイル **31**
A.1　状態フィードバックの入力ファイル `input.m` 31
A.2　状態推定の入力ファイル `seinput.m` 31

付 録 B　自動生成プログラム **33**

付 録 C 自動生成の元となるファイル **37**
C.1 状態フィードバック用 AutoGen.mc 37

付 録 D main 関数の入ったファイル **40**
D.1 状態フィードバック用 rhmain.c 40

要旨

非線形 receding horizon 制御や非線形 receding horizon 状態推定を行うためのアルゴリズムのプログラミングにおいては複雑な偏微分や行列の演算などを行う必要がある. 本研究では, その繁雑さを解消するために, 偏微分や行列の演算などの数式計算を数式処理ソフトウェアである Mathematica に行わせ, その結果を用いた C 言語のシミュレーションプログラムを自動的に生成するシステムを開発した. 特に, Mathematica のコマンドである `Splice`, `CAssign`, `Optimize` を用いることで, 簡便かつ効率的な自動生成システムとなった. 利用者は自動生成に必要な情報を入力ファイルに与えるだけで良い.

本論文では, 非線形 receding horizon 制御と非線形 receding horizon 状態推定のアルゴリズムの概要と, 自動生成システムの構成と使用法について説明する. また, このシステムにより生成されたシミュレーションプログラムの実行例を示す.

第1章　序論

Receding horizon 制御は最適制御の一種であり, 各時刻においてある有限の評価区間を持つ評価関数を最小にするように制御入力を決定する. 評価区間が時刻の経過と共に移動するので receding horizon という名前が付いた.　一方, receding horizon 状態推定は有限時間過去から現在までの出力と入力から現在の状態を推定するものである. これらの問題に関する理論的研究とともに, 非線形系に対して実時間で実行可能なアルゴリズムの開発が課題となっている.

制御や推定などの動的な最適化問題においての評価関数を最小にする方法として変分法がある [7, 8]. これは評価関数の変分を求めそれが 0 になるようにするもので, その条件としてオイラー・ラグランジュ方程式が得られる. このオイラー・ラグランジュ方程式から 2 点境界値問題が得られ, それを解くことで与えられた制御問題, 状態推定問題を解くことができる. 非線形 2 点境界値問題は通常逐次近似法などで数値的に解かれ, 大きな計算時間が必要となる. しかし, 安定化された連続変形法を用いることで逐次近似を使わずに実時間で問題を解くことができる. この方法を用いてシミュレーションや実験を行った研究がある [3, 4, 5, 6].

非線形 receding horizon 制御や非線形 receding horizon 状態推定問題のアルゴリズムをプログラムする場合, 複雑な偏微分や行列の計算が必要になる. これを人間の手で計算するのは繁雑であり誤りも生じやすい. また, 場合によってはたくさんの数式計算を必要とすることになる. そこで解決策としてなんらかの数式処理言語を使い自動的に式を計算しシミュレーションプログラムのコードを生成するようにすればよい.

このような考えに基づいた自動生成システムとして, Matlab を用いた宇宙ロボットのダイナミクスの計算 [2] や citespacedyn, Maple を用いたロボットの制御シミュレーション [9, 10], Mathematica を用いたロボットの制御 [11] などのためのソフトウェアが公開されている.

本研究では非線形 receding horizon 制御問題と非線形 receding horizon 状態推定問題を対象とし, 与えられた評価関数や状態方程式を Mathematica によって数式処理し C 言語のシミュレーションプログラムを自動的に生成するシステムを開発する. 実際に完成した自動生成システムの構造とその使用例について述べる.

第2章　問題の定式化とアルゴリズム

2.1　一般的な最適制御問題

まず, 通常の有限時間最適制御問題を概観する．制御対象の状態方程式は, 次のように与えられているものとする.

$$\dot{x} = f[x(t), u(t)]$$

ここで $x(t)$ は状態ベクトル, $u(t)$ は制御入力ベクトルを表す. さらに, 初期状態 $x(0) = x_0$ は与えられているとする.

評価関数は

$$J = \varphi[x(T)] + \int_0^T L[x(t), u(t)]dt$$

とする. ただし, $\varphi[x(T)], L[x(t), u(t)]$ は制御目的を反映するように選ばれるものとする. 最適制御問題とは評価関数を最小にするような制御入力 $u(t)$, $(0 \leq t \leq T)$ を求めることである.

2.2　オイラー・ラグランジュ方程式

評価関数を最小にする最適制御は, 状態方程式を拘束条件として評価関数を停留させなければならない. これは, 拡張された汎関数

$$\begin{aligned} \bar{J} &= \varphi[x(T)] + \int_0^T [L[x(t), u(t)] \\ &\quad + \lambda^T(t)\{f[x(t), u(t)] - \dot{x}(t)\}]dt \end{aligned} \tag{2.1}$$

を停留させることにほかならない. $\lambda(t)$ はラグランジュ乗数ベクトルであり, 共状態または随伴変数と呼ばれる.

$$H = L + \lambda^T f \tag{2.2}$$

と置くと,

$$\begin{aligned} \bar{J} &= \varphi[x(T)] \\ &\quad + \int_0^T \{H[x(t), \lambda(t), u(t)] - \lambda^T(t)\dot{x}(t)\}dt \end{aligned} \tag{2.3}$$

となる. H をハミルトニアンと呼ぶ．これを部分積分して,

$$\begin{aligned} \bar{J} &= \varphi[x(T)] - \lambda^T(T)x(T) + \lambda^T(0)x(0) \\ &\quad + \int_0^T \{H[x(t), \lambda(t), u(t)] + \dot{\lambda}^T(t)x(t)\}dt. \end{aligned} \tag{2.4}$$

第 1 変分は,

$$\begin{aligned}\delta\bar{J} &= [(\varphi_x^T - \lambda)^T \delta x]_{t=T} + [\lambda^T \delta x]_{t=0} \\ &\quad + \int_0^T \{(H_x^T - \dot{\lambda})^T \delta x + H_u \delta u\} dt. \end{aligned} \tag{2.5}$$

ここで初期条件固定から $\delta x(0) = 0$ である. 停留条件 $\delta\bar{J} = 0$ より, 最適性の必要条件は,

$$\dot{x} = f \tag{2.6}$$

$$\dot{\lambda} = -H_x^T \tag{2.7}$$

$$H_u = 0 \tag{2.8}$$

$$x(0) = x_0 \tag{2.9}$$

$$\lambda(T) = \varphi_x^T[x(T)] \tag{2.10}$$

式 (2.6)–(2.8) はオイラー・ラグランジュ方程式と呼ばれる. また, 式 (2.6)–(2.10) は, $x(0)$ と $\lambda(T)$ が規定された 2 点境界値問題となる.

2.3 Backward Sweep 法

$H_u = 0$ は u の次元に等しい個数の条件であり, x, λ, u を含む. 与えられた x と λ に対して $H_u = 0$ を満たす u が唯一に決まると仮定すると, u が x と λ の関数として定まる.

$$u(t) = u[x(t), \lambda(t)] = \arg\{H_u[x(t), \lambda(t), u(t)] = 0\} \tag{2.11}$$

ここで, $\arg\{\}$ は括弧中の方程式の解を意味する.

これを使えば x と λ の微分方程式から u を消去できる. そこで, 適当な $\lambda(0)$ を与えれば, $x(0)$ と $\lambda(0)$ を初期条件としてオイラー・ラグランジュ方程式の解が求まる. 当然 $x(T)$ と $\lambda(T)$ も決まる. $x(T)$ と $\lambda(T)$ とは $x(0)$ と $\lambda(0)$ および T の関数とみなせる. したがって,

$$F[\lambda(0), x(0), T] = \lambda(T) - \varphi_x^T[x(T)] \tag{2.12}$$

とおくと, λ の終端条件は

$$F[\lambda(0), x(0), T] = 0 \tag{2.13}$$

という $\lambda(0)$ の非線形代数方程式とみなせる. そして, その解が求まれば 2 点境界値問題が解けたことになる.

つぎに, 初期状態 $x(0)$ が $\delta x(0)$ だけ微小変化した場合を考える. 最適軌道の微小変化を $\delta x(t)$ および $\delta\lambda(t)$ と表すと, オイラー・ラグランジュ方程式から

$$\begin{cases} \delta\dot{x} = f_x \delta x + f_u \delta u \\ \delta\dot{\lambda} = -H_{xx}\delta x - f_x^T \delta\lambda - H_{xu}\delta u \\ H_{ux}\delta x + f_u^T \delta\lambda + H_{uu}\delta u = 0 \end{cases}$$

を得る. これから δu を消去して次のような時変線形微分方程式を得る.

$$\frac{d}{dt}\begin{bmatrix} \delta x \\ \delta\lambda \end{bmatrix} = \begin{bmatrix} A & -B \\ -C & -A^T \end{bmatrix}\begin{bmatrix} \delta x \\ \delta\lambda \end{bmatrix} \tag{2.14}$$

ここで, 行列 A, B, C は

$$\begin{cases} A = f_x - f_u H_{uu}^{-1} H_{ux} \\ B = f_u H_{uu}^{-1} f_u^T \\ C = H_{xx} - H_{xu} H_{uu}^{-1} H_{ux} \end{cases}$$

で与えられ, 最適軌道 $(x(t),\ \lambda(t))$ に沿って決定される. 一方, 終端条件から

$$\delta\lambda(T) = \varphi_{xx}[x(T)]\delta x(T) \tag{2.15}$$

が成り立たねばならない. 与えられた $\delta x(0)$ に対して, 式 $(2.14), (2.15)$ は時変線形 2 点境界値問題になる. これを解くため

$$\delta\lambda(t) = S(t)\delta x(t) \tag{2.16}$$

とおくと, 式 (2.15) から

$$S(T) = \varphi_{xx}[x(T)] \tag{2.17}$$

である. 式 (2.14) に式 (2.17) を代入すると

$$(\dot{S} + A^T S + SA - SBS + C)\delta x = 0. \tag{2.18}$$

これが任意の δx に対して成り立たねばならないから, 次式を得る.

$$\dot{S} = -A^T S - SA + SBS - C \tag{2.19}$$

この微分方程式は式 (2.17) から出発して逆時間で解かれる (backward sweep 法). 結局, $\delta x(0)$ に対する初期共状態の変化は

$$\delta\lambda(0) = S(0)\delta x(0) \tag{2.20}$$

となる.

2.4 非線形 Receding Horizon 制御

2.4.1 問題設定

制御対象は,

$$\dot{x} = f[x(t), u(t)] \tag{2.21}$$

ただし, 状態 $x(t)$ は完全に測定可能とする.

このとき各時刻 t において, 評価関数

$$J = \varphi[x(T)] + \int_t^{t+T} L[x(t'), u(t')]dt' \tag{2.22}$$

を T だけ未来までを評価する. 評価区間が時刻 t を含み時間とともに後退する. 各時刻 t 毎に, 現在の状態 $x(t)$ を出発点として最適制御問題を解き, 最適制御 $u_{opt}[t'; x(t), T](t \leq t' \leq t+T)$ を求める. しかし, 実際の制御入力は $u_{opt}[t; x(t), T]$ のみを使う. ただし, 一般に T は時変である.

Receding horizon 制御の利点は一般に, 閉ループ系の漸近安定性を保証しないが, 近年, 安定性を保証するための条件がいくつか得られている. また, それらの条件が満たされなくても良好な制御が行える場合が多い. しかし, 最適制御問題を各時間毎に解くのは, 通常困難であり, 何らかの工夫が必要である.

2.4.2 問題の変換

時刻 t から τ だけ未来における最適軌道を $x^*(\tau,t)$ と書くことにする．$\lambda^*(\tau,t), u^*(\tau,t)$ も同様とする．すると，receding horizon 制御は，時刻 t に依存する τ 軸上の最適制御問題の族に変換される．
τ 軸上の初期状態は

$$x^*(0,t) = x(t) \tag{2.23}$$

となり，τ 軸上の状態方程式は

$$x^*_\tau(\tau,t) = f[x^*(\tau,t), u^*(\tau,t)] \tag{2.24}$$

となる．また，τ 軸上の評価関数も次のようになる．

$$J = \varphi[x^*(T,t)] + \int_0^T L[x^*(\tau,t), u^*(\tau,t)]d\tau \tag{2.25}$$

この τ 軸上の問題を解き，$u^*(\tau,t)(0 \le \tau \le T)$ を決定する．実際の制御入力は

$$u(t) = u^*(0,t) \tag{2.26}$$

で与えられる．これに対するオイラー・ラグランジュ方程式は

$$\begin{cases} x^*_\tau(\tau,t) = f; & x^*(0,t) = x(t) \\ \lambda^*_\tau(\tau,t) = -H^T_x; & \lambda^*(T,t) = \varphi^T_x[x^*(T,t)] \\ H_u = 0 \end{cases} \tag{2.27}$$

となる．λ^* の終端条件は

$$F[\lambda^*(0,t), x^*(0,t), T] = \lambda^*(T,t) - \varphi^T_x[x^*(T,t)] = 0 \tag{2.28}$$

という非線形代数方程式とみなせる．未知量は，τ 軸上の初期共状態 $\lambda^*(0,t)$ である．$\lambda^*(0,t)$ を $\lambda(t)$ と書くことにする．ここで，$x^*(0,t) = x(t)$ であるから，式 (2.28)

$$F[\lambda(t), x(t), T] = \lambda^*(T,t) - \varphi^T_x[x^*(T,t)] = 0 \tag{2.29}$$

という時刻 t に依存した方程式となる．Receding horizon 制御を実行するには，この非線形代数方程式を $\lambda(t)$ に関して実時間で解かなければならない．

2.4.3 時間と共に変化する解の追跡

共状態量の方程式 (2.29) を各時刻ごとに逐次近似で解くと時間がかかり過ぎる．しかし，状態 $x(t)$ の微小変化に対する共状態 $\lambda(t)$ の変化は backward sweep 法で計算できる．また，状態 $x(t)$ は状態方程式にしたがって滑らかに変化する．

そこで，状態 $x(t)$ の変化に伴う共状態 $\lambda(t)$ の変化を追跡することが実時間で可能だと考えられる．ただし，追跡の過程で誤差の累積を防ぐ工夫と制御開始時の λ を決めるための工夫が必要である．

まず，$F = 0$ を安定化するために $\dot{F} = A_1 F(A_1$：安定行列) が成り立つように $\dot{\lambda}(t)$ を決める．そうすれば誤差を減衰させつつ解を追跡し続けられるはずである．さらに，評価区間

$T=0$ なら $\lambda=\varphi_x^T(x)$ が 2 点境界値問題の自明な解となることを利用して, $T(0)=0$ かつ $t\to\infty$ のとき $T(t)\to T_f$ なる時変の評価区間 $T(t)$ を使い, $\lambda(0)=\varphi_x^T[x(0)]$ を初期条件とする. つまり, 時間をパラメータとする一種の連続変形法である.

なお, 2 点境界値問題を解いて得られるのはあくまでも停留解であり, 局所最適解であることすら事前に保証することはできない. しかし, 実際の応用では停留解が得られれば十分良好な制御が行えることが多い.

2.4.4 解の常微分方程式

条件 $\dot{F}=A_1F(A_1$：安定行列) から, backward sweep 法を使って共状態 $\lambda(t)$ の常微分方程式を導く. まず, $\tau=T$ において,

$$\dot{F}=\lambda_t^*(T,t)+\lambda_\tau^*(T,t)\dot{T}-\varphi_{xx}[x^*(T,t)][x^*(T,t)_t+x^*(T,t)_\tau\dot{T}] \tag{2.30}$$

これが A_1F に等しくなければならないから,

$$\lambda_t^*(T,t)=\varphi_{xx}x_t^*(T,t)+(H_x^T+\varphi_{xx}f)\dot{T}+A_1F \tag{2.31}$$

が $\tau=T$ における λ_t^*, λ_τ^*, $\dot{T}$ の関係となる. 通常の最適制御の場合を参考に,

$$\lambda_t^*(\tau,t)=S^*(\tau,t)x_t^*(\tau,t)+c^*(\tau,t) \tag{2.32}$$

と置くと, $\tau=T$ での境界条件は

$$\begin{cases} S^*(T,t)=\varphi_{xx}|_{\tau=T} \\ c^*(T,t)=(H_x^T+\varphi_{xx}f)_{\tau=T}\dot{T}+A_1F \end{cases} \tag{2.33}$$

となり. また τ 軸上の微分方程式は

$$\begin{cases} S_\tau^*=-A^TS^*-S^*A+S^*BS^*-C \\ c_\tau^*=-(A^T-S^*B)c^* \end{cases} \tag{2.34}$$

となる. 式 (2.32) を $\tau=0$ で考えると $\lambda_t^*(0,t)=\dot{\lambda}, x_t^*(0,t)=\dot{x}(t)$ だから, 共状態 $\lambda(t)$ の常微分方程式

$$\dot{\lambda}(t)=S^*(0,t)\dot{x}(t)+c^*(0,t) \tag{2.35}$$

を得る.

2.4.5 実時間での解法アルゴリズム

以上をまとめると, アルゴリズムは以下のようになる. 制御入力は

$$u(t)=\arg\{H_u[x(t),\lambda(t),u(t)]=0\}$$

で与えられる. ここで $\lambda(t)$ は常微分方程式

$$\begin{aligned} &\dot{\lambda}(t)=S^*(0,t)\dot{x}(t)+c^*(0,t) \\ &\lambda(0)=\varphi_x^T[x(0)] \end{aligned} \tag{2.36}$$

を実時間で解いて得られる. 各時刻毎の $S^*(0,t)$ と $c^*(0,t)$ は, 以下の計算で得られる.

τ 軸上で $x^*(0,t)=x(t)$, $\lambda^*(0,t)=\lambda(t)$ を初期値とし

$$x^*_\tau(\tau,t)=f,\quad \lambda^*_\tau(\tau,t)=-H^T_x \tag{2.37}$$

を順方向で積分し $x^*(T,t)$, $\lambda^*(T,t)$ を求め,

$$\begin{aligned} S^*(T,t) &= \varphi_{xx}|_{\tau=T} \\ c^*(T,t) &= (H^T_x+\varphi_{xx}f)_{\tau=T}\dot{T}+A_1F \end{aligned} \tag{2.38}$$

を計算し, それを初期値とし

$$\begin{aligned} S^*_\tau &= -A^TS^*-S^*A+S^*BS^*-C \\ c^*_\tau &= -(A^T-S^*B)c^* \end{aligned} \tag{2.39}$$

を τ 軸上で逆方向に積分することにより $S^*(0,t)$ と $c^*(0,t)$ を求める.

ここで上の式は逐次近似を含まない常微分方程式となっている. しかも, 導出過程で近似を行っていない. きわめて一般的な制御対象に適用可能である.

2.4.6 時変パラメータを含む場合のアルゴリズム

制御対象に時変パラメータが追加された場合についても考察する. このときの状態方程式は

$$\dot{x}=f[x(t),u(t),p(t)] \tag{2.40}$$

となる. $p(t)$ は時変パラメータを表し, 既知とする.

このとき λ^* の終端条件は

$$F[\lambda(t),x(t),T,t]=\lambda^*(T,t)-\varphi^T_x[x^*(T,t),p(t+T)]=0 \tag{2.41}$$

となる. そして, 次の $x^*_t-x^*_\tau$, $\lambda^*_t-\lambda^*_\tau$ に関する時変線形微分方程式が得られる.

$$\frac{\partial}{\partial\tau}\begin{bmatrix} x^*_t-x^*_\tau \\ \lambda^*_t-\lambda^*_\tau \end{bmatrix}=\begin{bmatrix} A & -B \\ -C & -A^T \end{bmatrix}\begin{bmatrix} x^*_t-x^*_\tau \\ \lambda^*_t-\lambda^*_\tau \end{bmatrix} \tag{2.42}$$

$\dot{F}=A_1F$ とすると, $\tau=T$ において

$$\begin{aligned} \dot{F} &= \lambda^*_t(T,t)-\varphi_{xx}x^*_t(T,t)-\varphi_{xp}p_t(t+T) \\ &\quad +[\lambda^*_\tau(T,t)-\varphi_{xx}x^*_\tau(T,t)-\varphi_{xp}p_\tau(t+T)]\dot{T} \end{aligned}$$

これが A_1F に等しくならなければならないから次式が得られる.

$$\begin{aligned} A_1F &= \lambda^*_t(T,t)-\varphi_{xx}x^*_t(T,t)-\varphi_{xp}p_t(t+T) \\ &\quad +[\lambda^*_\tau(T,t)-\varphi_{xx}x^*_\tau(T,t)-\varphi_{xp}p_\tau(t+T)]\dot{T} \end{aligned}$$

次のように整理する.

$$\lambda^*_t-\lambda^*_\tau=(-\lambda^*_\tau(T,t)+\varphi_{xx}x^*_\tau(T,t)+\varphi_{xp}p_\tau(t+T))(1+\dot{T})+A_1F$$

上の式を参考にして

$$\lambda_t^* - \lambda_\tau^* = S^*(\tau,t)(x_t^* - x_\tau^*) + c^*(\tau,t) \tag{2.43}$$

とおく. このとき $\tau = T$ における境界条件は

$$\begin{cases} S^*(T,t) = \varphi_{xx}|_{\tau=T} \\ c^*(T,t) = (H_x^T + \varphi_{xx} f + \varphi_{xp} p_t)|_{\tau=T}(1+\dot{T}) + A_1 F \end{cases} \tag{2.44}$$

となり, τ 軸上での微分方程式として

$$\begin{cases} S_\tau^* = -A^T S^* - S^* A + S^* B S^* - C \\ c_\tau^* = -(A^T - S^* B)c^* \end{cases} \tag{2.45}$$

が得られる. これより

$$\lambda_t^* = \lambda_\tau^* + S^*(0,t)(x_t^* - x_\tau^*) + c^*(0,t)$$

となり, $\lambda_\tau^*(0,t) = -H_x^T|_{\tau=0}$ と $x_t^*[x(t),u(t),p(t)] = x_\tau^*[x(t),u(t),p(t)] = f[x(t),u(t),p(t)]$ を考慮すると λ の t 軸上での微分方程式

$$\dot{\lambda}(t) = -H_x^T[x(t),\, \lambda(t), u(t), p(t)] + c^*(0,t) \tag{2.46}$$

が得られる. 本論文ではこの微分方程式をアルゴリズムに用いる.

2.5 非線形 Receding Horizon 状態推定

2.5.1 問題設定

まず, 取り扱う非線形システムは次のような状態方程式と出力方程式で表される.

$$\begin{cases} \dot{x}(t) = f[x(t),u(t),w(t)] \\ y(t) = g[x(t),u(t),v(t)] \end{cases} \tag{2.47}$$

ここで, $x(t)$ は状態ベクトルを, $u(t)$ は計測できる入力を, $y(t)$ は計測できる出力を, $v(t)$ は計測できない未知雑音を, $w(t)$ は未知外乱を表すものとする. $x(t)$ は直接計測できないと仮定する. 入力と出力をまとめた時変パラメータベクトル $p(t)$ を次のように定義する.

$$p(t) = \begin{bmatrix} y(t) \\ u(t) \end{bmatrix} \tag{2.48}$$

未知雑音 $v(t)$ は出力方程式から次のように $x(t)$, $p(t)$ の関数として表される.

$$v(t) = h[x(t),p(t)] \tag{2.49}$$

もし, $v(t) = 0$ なら, 状態推定値 $\hat{x}(t)$ から決まる雑音の推定値 $\hat{v}(t) = h[\hat{x}(t),p(t)]$ も 0 となるべきである. また, $\hat{v}(t)$ は推定の残差と見なせるので, なるべく小さくすることが望ましい. 例えば線形システムの場合を考えると $y(t) = Cx(t) + v(t)$ から $\hat{v}(t) = y(t) - C\hat{x}(t)$ となり実際に推定残差となっている. また, モデルの不正確さを表す外乱の推定値 $\hat{w}(t)$ も小さくするのが合理的である.

結局, 状態推定値 $\hat{x}(t)$ を $p(t)$ から決めるためには $\hat{v}(t)$ と $\hat{w}(t)$ を小さくすれば良いということになる. そこで $x(t)$ の最適推定値 $\hat{x}(t)$ を $t-T \leq \tau \leq t$ の評価区間にわたって $\hat{v}(t)$ と $\hat{w}(t)$ を何らかの意味で最小にするように決定する. ただし, $\hat{v}(t)$ は式 (2.49) を使って $\hat{x}(t)$ と $p(t)$ で表されているものとすると, 評価関数の一般形は次のようになる.

$$\begin{aligned} J &= \eta[\hat{x}(t), p(t)] + \varphi[\hat{x}(t-T), p(t-T)] \\ &\quad + \int_{t-T}^{t} L[\hat{x}(t'), p(t'), \hat{w}(t')]dt' \end{aligned} \tag{2.50}$$

この評価関数は有限の過去 $t-T$ から現在の時刻 t までという評価区間を持つ. 一般に評価区間の長さ T は時間によって変化するとする.

2.5.2 問題の変換

Receding–horizon 状態推定問題は時刻 t をパラメータとする τ 軸上の問題に変換できる.

$$\begin{aligned} J &= \eta[x^*(0,t), p(t)] + \varphi[x^*(-T,t), p(t-T)] \\ &\quad + \int_{-T}^{0} L[x^*(\tau,t), p(t+\tau), w^*(\tau,t)]d\tau \end{aligned} \tag{2.51}$$

ただし,

$$x^*_\tau(\tau,t) = f[x^*(\tau,t), p(t+\tau), w^*(\tau,t)] \tag{2.52}$$

推定値 $\hat{x}(t)$,$\hat{w}(t)$ は

$$\hat{x}(t) = x^*(0,t) \tag{2.53}$$

$$\hat{w}(t) = w^*(0,t) \tag{2.54}$$

のように与えるとする. $(\ )^*$ は問題を変換したことを表す.

このときオイラー・ラグランジュ方程式と境界条件は次のようになり 2 点境界値問題を構成する.

$$x^*_\tau(\tau,t) = H^T_\lambda \tag{2.55}$$

$$\lambda^*_\tau(\tau,t) = -H^T_x \tag{2.56}$$

$$H_w = 0 \tag{2.57}$$

$$\lambda^*(0,t) = \eta^T_x[x^*(0,t), p(t)] \tag{2.58}$$

$$\lambda^*(-T,t) = -\varphi^T_x[x^*(-T,t), p(t-T)] \tag{2.59}$$

ここで, $\lambda^*(\tau,t)$ は $x^*(\tau,t)$ に対応する共状態量を表し,

$$H = L + \lambda^{*T} f \tag{2.60}$$

をハミルトニアンとする.

2 点境界値問題は次の非線形代数方程式を満たす $\hat{x}(t) = x^*(0,t)$ を決めることで解くことができる.

$$\begin{aligned} &F[\hat{x}(t), t, T] \\ &\quad = \lambda^*(-T,t) + \varphi^T_x[x^*(-T,t), p(t-T)] = 0 \end{aligned} \tag{2.61}$$

$\lambda^*(-T,t), x^*(-T,t)$ は $\hat{x}(t) = x^*(0,t)$ により変化する.

2.5.3 安定化された連続変形法

式 (2.61) の解 $\hat{x}(t)$ を追跡するために次のような式を考える.

$$\dot{F} = A_s F \tag{2.62}$$

ここで, A_s は安定行列. これを満足するように $\hat{x}(t)$ を決定することにより誤差の累積を防ぎつつ $F = 0$ の解を追跡し続けることができる [3].

2.5.4 Backward–Sweep 法

オイラー・ラグランジュ方程式から次の $x_t^* - x_\tau^*$ と $\lambda_t^* - \lambda_\tau^*$ に関する時変微分方程式を導くことができる.

$$\frac{\partial}{\partial \tau} \begin{bmatrix} x_t^* - x_\tau^* \\ \lambda_t^* - \lambda_\tau^* \end{bmatrix} = \begin{bmatrix} A & -B \\ -C & -A^T \end{bmatrix} \begin{bmatrix} x_t^* - x_\tau^* \\ \lambda_t^* - \lambda_\tau^* \end{bmatrix} \tag{2.63}$$

ここで, A,B,C は次のように表される.

$$\begin{cases} A = f_x - f_w H_{ww}^{-1} H_{wx} \\ B = f_w H_{ww}^{-1} f_w^T \\ C = H_{xx} - H_{xw} H_{ww}^{-1} H_{wx} \end{cases} \tag{2.64}$$

式 (2.63) において $p_t(t+\tau) = p_\tau(t+\tau)$ から $p(t+\tau)$ の偏微分の項は消える.

式 (2.62) の右辺の項は次のような形になる.

$$\begin{aligned} \dot{F} &= [\lambda_t^* - \lambda_\tau^* \dot{T} + \varphi_{xx}(x_t^* - x_\tau^* \dot{T}) \\ &\quad + \varphi_{xp}(p_t - p_\tau \dot{T})]|_{\tau=-T} \end{aligned} \tag{2.65}$$

これを整理して式 (2.62) を書き直すと次の式になる.

$$\begin{aligned} &[\lambda_t^* - \lambda_\tau^* + \varphi_{xx}(x_t^* - x_\tau^*) + (1 - \dot{T}) \\ &\quad \times(-H_x^T + \varphi_{xx} f + \varphi_{xp} p_t)]|_{\tau=-T} = A_s F \end{aligned} \tag{2.66}$$

式 (2.63) の解が終端条件 (2.66) を満たすような $x_t^*(0,t)$ を見つけるため, 次の関係を仮定する.

$$x_t^* - x_\tau^* = \Lambda^*(\tau, t)(\lambda_t^* - \lambda_\tau^*) + r^*(\tau, t) \tag{2.67}$$

このとき式 (2.66) と (2.67) から $\tau = -T$ における行列 Λ とベクトル r の満たすべき境界条件が次のように得られる.

$$\Lambda^*(-T, t) = -\varphi_{xx}^{-1}|_{\tau=-T} \tag{2.68}$$

$$\begin{aligned} r^*(-T, t) &= \varphi_{xx}^{-1}[A_s F - (1 - \dot{T}) \\ &\quad \times(-H_x^T + \varphi_{xx} f + \varphi_{xp} p_t)]|_{\tau=-T} \end{aligned} \tag{2.69}$$

式 (2.63) と式 (2.67) から次の微分方程式が得られる.

$$\Lambda_\tau^* = A\Lambda^* + \Lambda^* A^T + \Lambda^* C \Lambda^* - B \tag{2.70}$$

$$r_\tau^* = (A + \Lambda^* C) r^* \tag{2.71}$$

$\hat{x}(t) = x^*(0,t)$ が決まっていれば, 式 (2.58) から $\lambda^*(0,t)$ を求め, それらを初期値としてオイラー・ラグランジュ方程式の (2.55) と (2.56) を後向きに積分すると $x^*(-T,t), \lambda^*(-T,t)$ が得られる. これらから境界条件 (2.68),(2.69) を計算し, 微分方程式 (2.70) と (2.71) を前向きに積分すると $\tau = 0$ での $\Lambda^*(0,t), r^*(0,t)$ が得られる. また, $\tau = 0$ での λ_t^* の値は式 (2.58) から

$$\lambda_t^* = [\eta_{xx} x_t^* + \eta_{xp} p_t]|_{\tau=0}$$

となる. これらと式 (2.67) から次式が求まり, $\hat{x}(t)$ の時間微分が決まる.

$$\frac{d\hat{x}(t)}{dt} = (I - \Lambda^* \eta_{xx})^{-1} [f + \Lambda^* (H_x^T + \eta_{xp} p_t) + r^*]|_{\tau=0} \tag{2.72}$$

これが $\hat{x}(t)$ の微分方程式である. 外乱の推定値は次の式から求まる.

$$\hat{w}(t) = w^*(0,t) = \arg(H_w|_{\tau=0} = 0) \tag{2.73}$$

式 (2.72) を実時間で積分することにより, 状態推定値 $\hat{x}(t)$ が更新される.

第3章　自動生成システム

前章で説明したアルゴリズムは, 状態方程式やハミルトニアンの偏導関数を含み, そのプログラミングは繁雑かつ誤りを生じやすい. そこで, シミュレーションプログラムの自動生成が有用である. 本章では, receding horizon 制御と receding horizon 状態推定の双方に共通な部分について説明する.

3.1　システムの働き

自動生成する C 言語のシミュレーションプログラムのうち問題に依存する部分と依存しない部分を分けて別々に作っておく. 問題に依存する部分とは数式処理される部分で, この部分を Mathematica を用い自動生成する. 問題に依存しない部分とは配列どうしの計算や数値積分を行う関数を定義した部分と main 関数などがある. 問題に依存しない部分は別のファイルとして先に作っておき, 自動生成されたファイルをコンパイル, リンクするときにインクルードする. これによりシミュレーションプログラムの実行ファイルができあがる.

3.2　システムの構成

自動生成システムは

1. 利用者が定義する入力ファイル (状態フィードバック用 `input.m`, 状態推定用 `seinput.m`)
2. 自動生成を行う Mathematica プログラム (状態フィードバック用 `AutoGen.nb`, 状態推定用 `SeGen.nb`)
3. 自動生成する C のコードの元となるファイル (状態フィードバック用 `AutoGen.mc`, 状態推定用 `SeGen.mc`)

の 3 つからなり, シミュレーションプログラムは

1. 汎用関数を集めたファイル (状態フィードバック用 `rhfunc.c`, 状態推定用 `sefunc.c`)
2. 自動生成されたファイル
3. main 関数の入ったファイル (状態フィードバック用 `rhmain.c`, 状態推定用 `semain.c`)

の 3 つで構成される. 付録として自動生成システムの一部を付けた.

3.3 入力ファイル

利用者が変更するべき部分を, input.m, seinput.m というファイルにひとまとめにしてある (拡張子.m というのは Mathematica のファイルであることを示している). このファイルに必要な情報を Mathematica の書式で書き込んでおけば, 後は自動生成用 Mathematica プログラムを実行することにより, 自動的にシミュレーションプログラムを生成できる.

3.3.1 入力ファイルの形

入力ファイルの例を, recding horizon 制御については付録 A.1 に, recding horizon 状態推定については付録 A.2 にそれぞれ示す. 以下で変数の説明を行う.

まず, 状態ベクトル $x(t)$ や時変パラメータ $p(t)$ などのベクトルの次元を宣言している.

dimx; 状態ベクトル x の次数
dimp; 入力ベクトル u の次数
dimp; 時変パラメータ p の次数
dimw; 外乱ベクトル w の次数
dimy; 出力ベクトル y の次数

次の部分では f, y などの式をそれぞれの変数に入れている.

- fxu は状態方程式を表す.
- fy は $y(t) = g[x(t), u(t), v(t)]$ の式を表す (状態推定のみ).
- L は評価関数の式を表す.
- phi は φ を表す.
- eta は η を表す (状態推定のみ).
- pt は与えられた時変パラメータの形を表す.

次の部分では利用者独自の変数や行列の形の宣言とその値が入っている. これを元にして C のファイルでも値が渡される.

- MyVarNames はユーザー変数の名前を表す.
- MyVarValues はユーザー変数の値を表す.
- MyArrNames はユーザー配列の名前を表す.
- MyArrDims はユーザー配列の大きさを表す.
- MyArrValues はユーザー配列の値を表す.

tsim0 以下の部分はシミュレーションの条件と自動生成の条件が入っている.

- tsim0 はシミュレーションの開始時刻

- tsim はシミュレーションの終了時刻
- tf は評価区間の最大値
- ht は t 軸上での時間のきざみ
- alpha は評価区間の変化の仕方に関するパラメータ
- zeta は安定行列 $A_s = -\zeta I$ を決めるパラメータ
- x0 は初期状態量
- dv は τ 軸の分割数
- dstep 毎にデータをセーブする.
- は外乱の分散を表す (状態推定のみ).
- は外乱の平均を表す (状態推定のみ).
- は雑音の分散を表す (状態推定のみ).
- は雑音の平均を表す (状態推定のみ).
- fnout は自動生成する C のファイル名
- fnhd はデータをセーブするファイル名
- optu は Optimize(後述) するかしないかを表し, 0 以外のとき Optimize する.
- smpu は Mathematica の関数 Simplify を用いて式を簡単化するかどうかを表す. 簡単化すると自動生成するのに必要な処理時間が長くなる.

状態推定での入力ファイルに対する注意

評価区間の長さは $T = T_f(1-e^{-\alpha t})$ で計算される. これの t での微分は $dT/dt = \alpha T_f e^(-\alpha t)$ となる. 開始時刻 tsim0 でこの値が 1 以下でないと開始時刻より前に評価区間が戻ることになり状態推定ができなくなる.

また, $\lambda^*(0,t) = \eta_x^T[x^*(0,t), p(t)]$ と $\lambda^*(-T,t) = -\varphi_x^T[x^*(-T,t), p(t-T)]$ において, 時刻を $t = 0$ とすると $T = 0$ となり

$$\lambda^*(0,0) = \eta_x^T[x^*(0,0), p(0)] = -\varphi_x^T[x^*(0,0), p(0)]$$

が成り立つように φ と η の形を決める必要がある.

3.4 Mathematica プログラムの実行手順

自動生成システムの Mathematica プログラムは以下のような処理をしてシミュレーションプログラムを生成する.

1. 使う関数の定義.
2. いろいろな変数に初期値を与える.
3. 入力ファイルを呼び出す.
4. Optimize コマンドを使うか使わないかをセットする.
5. 使用するベクトルを宣言する.
6. 数式の計算
7. 使用するパッケージ Format2, Optimize2 を呼び出す. オリジナルのパッケージでは Mathematica のバージョンの違いのためかエラーが出る. そのため, Format.m と Optimize.m 中の NProtectedAll を NHoldAll に書き換えたものを Format2.m, Optimize2.m として用いている.
8. コマンド Optimize と CAssign のオプションを設定する.
9. Mathematica 上で一度 Optimize を行うことにより, 置き換えるのに使う配列 o[] の大きさを調べる.
10. Mathematica での配列は添字が 1 から始まるのに対して C 言語での配列は 0 から始まるため, 添字を置換するリストを作る.
11. Splice コマンドの使用により拡張子が .mc のファイルを処理し C のコードを生成する.

Splice, CAssign, Optimize について次に詳しく述べる.

3.5 Splice

Splice[14] は拡張子 .mc がついたファイルの中で<*と*>で括られた部分を Mathematica の入力として読み込み, それを評価した結果を C 言語の形で置き換える. これにより C 言語のソースファイル中に Mathematica での処理結果を取り込むことができる. なお, 拡張子 .mf は Fortran の形に, .mtex は TEX の形に利用できる.

.mc の例を図 3.1 に示す. これは状態フィードバックの AutoGen.mc の一部である.

3.6 CAssign

Format[12] は数式処理結果を Fortran および C で利用するための機能を追加できるパッケージで, その機能の一つにコマンド CAssign がある. CAssign とは Mathematica 上で

```
/*-------------- Control Input -------------- */

void ufunc(double t, double z[], double u1[])
{
    double x[DIMX], lmd[DIMX];
<*If[leux!=0,ou="    double o["<>ToString[leux]<>"];\n",ou=""];
  If[!(FreeQ[uxl,p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
<*CAssign[u1,uxl]*>
}
```

図 3.1: Splice の元となるファイルの例

```
/*-------------- Control Input -------------- */

void ufunc(double t, double z[], double u1[])
{
    double x[DIMX], lmd[DIMX];

    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
    u1[0] = (-0.5*(lm*cos(x[2])*lmd[1] + lmd[2] - lm*lmd[0]*sin(x[2])))/lm;
    u1[1] = (-0.5*(lm*cos(x[2])*lmd[1] - lmd[2] - lm*lmd[0]*sin(x[2])))/lm;
}
```

図 3.2: Splice により生成されたファイルの例

```
/*-------------- Control Input -------------- */

void ufunc(double t, double z[], double u1[])
{
    double x[DIMX], lmd[DIMX];
    double o[6];

    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
    o[0] = 1/lm;
    o[1] = cos(x[2]);
    o[2] = lm*lmd[1]*o[1];
    o[3] = sin(x[2]);
    o[4] = lm*lmd[0]*o[3];
    o[5] = -o[4];
    u1[0] = -0.5*o[0]*(lmd[2] + o[2] + o[5]);
    u1[1] = -0.5*o[0]*(-lmd[2] + o[2] + o[5]);
}
```

図 3.3: Optimize を使用して生成されたファイルの例

のベクトルを C の配列に対応付ける. ただし, Mathematica 上でのベクトルは 1 から始まるので C の配列に合わせるために添字を一つずつ前にずらす必要がある. 例えば, ここでの x[2] は x_3 を表している.

図 3.2 は自動生成したファイルの中で図 3.1 に対応したところである. CAssign によって Mathematica の数式処理結果が格納された配列 uxl が C の配列 u1 に割り当てられている.

3.7 Optimize

コマンド Optimize は Optimize[13] というパッケージに入っていて, これを使うと計算式中に繰り返し現れる表現を o[i] という配列で自動的に置き換えてくれる. これによりシミュレーションの計算時間を短縮することができる. CAssign の実行時に Optimize を用いるよう CAssign のオプションで設定できる. C で配列 o を宣言するとき, その大きさが必要になるが, Optimize の戻り値に配列 o の大きさが含まれるので, Mathematica で例えば以下のようにすると求めることができる.

```
leux=Length[Optimize[uxl][[1]]]
```

図 3.3 と図 3.2 を比べれば Optimize の効果が分かる. 配列 o が宣言されており, また, 複数回現れる表現を配列 o で置き換えている.

3.8 出力ファイル

自動生成されたCのコードをコンパイル, リンクして実行すると, 入力ファイルの中で決めた出力ファイル名に, 状態フィードバックでは c.dat, e.dat, p.dat, x.dat, u.dat, 状態推定では c.dat, e.dat, p.dat, x.dat, w.dat, y.dat をそれぞれ付けたデータファイルができる. 二つに共通する c.dat はコメントとシミュレーションの条件が, e.dat には関数 F の各時刻での値が, p.dat には時変パラメータの値がセーブされる. 状態フィードバックの x.dat には状態量 $x(t)$ と共状態量 $\lambda(t)$ がセーブされ, 状態推定の x.dat には実際の状態量 $x(t)$ と推定された状態量 $\hat{x}(t)$ がセーブされる. u.dat, w.dat, y.dat はそれぞれ入力 $u(t)$, 外乱 $\hat{w}(t)$, 出力 $y(t)$ の値がセーブされる.

第4章　シミュレーションプログラム

4.1　状態フィードバックプログラムの説明

4.1.1　プログラムの動作

状態量 $x(t)$ と共状態量 $\lambda(t)$ を並べたものを z とし, これの値を t 軸でのきざみ毎に計算し値を求める. 評価関数や状態方程式に依存する部分は自動生成されるが, それ以外の部分はあらかじめ rhfunc.c と rhmain.c に用意されている. 以下では各ファイルに含まれる関数について述べる.

4.1.2　関数を集めたファイル rhfunc.c 中の関数

- mmov は配列 a[m][n] の値を他の配列 b[m][n] に渡すための関数
- mmad は配列 a[m][n] と b[m][n] の値を足したものを c[m][n] に渡すための関数
- mmulsc は配列 a[m][n] に k をかけた値を b[m][n] に渡すための関数
- msym は n*(n+1)/2 次のベクトルから n × n 次の対称行列を作るための関数
- defvector は n 次のベクトルを定義する関数
- defmatrix は n × n 次の行列を定義する関数
- locate は n 個のデータを持つ xx という配列に対して, 与えられた x が何番目の区間に含まれるかを調べ, その値を返す関数
- table1 は x が xx の何番目の区間にくるかを locate で求め, それを j としたとき xx[j] と xx[j+1] から x に対応する値を線形補間する関数
- nfrkgex はルンゲ・クッタ・ギルの方法で数値積分する関数 [1, p.355]
- nfadams はアダムス・バシュフォースの方法を予測子, ムルトンの公式を修正子として数値積分する関数 [1, p.360]

4.1.3　自動生成されたファイル中の関数

- pfunc は時変パラメータ $p(t)$ の値を求めるための関数
- phix は φ_x の値を求めるための関数
- initial は初期値 x0 から phix を使い $\lambda(0)$ の値を求め z の初期値を得る関数

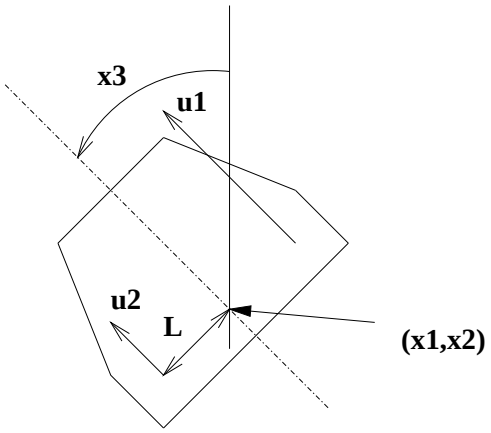

図 4.1: 二輪車両

- ufunc は入力 $u(t)$ の値を求めるための関数
- zpfuncEL は $f, -H_x$ の値を求めるための関数
- erropt は F の値を求めるための関数
- boundary は S^*, c^* の境界値を求めるための関数
- bkfunc は S^*, c^* の τ 軸での微分方程式の値を求めるための関数
- final はシミュレーションの条件をデータファイルにセーブする関数

4.1.4 main 関数の入ったファイル rhmain.c 中の関数

- zpfunct は $x^*(0,t) = x(t)$ と $\lambda^*(0,t) = \lambda(t)$ を初期値として zpfuncEL を τ 軸上で前向きに積分して $x^*(T,t)$, $\lambda^*(T,t)$ を求める. これらから erropt で F の値を, boundary で S^*, c^* の境界値を計算し, bkfunc を後向きに積分して $\tau = 0$ での $S^*(0,t)$, $c^*(0,t)$ を求める. そして, zpfuncEL から dx/dt の値を求め配列 zprime に入れる. また, $d\lambda^*/dt$ の値を求め配列 zprime の後半にその値を入れる.
- savedata はシミュレーションの結果をデータファイルにセーブするための関数
- main 関数は関数 zpfunct を使い dx/dt, $d\lambda^*/dt$ を t 軸で数値積分し $x(t), \lambda(t)$ を得る. そして, シミュレーションの結果を dstep 毎にデータファイルにセーブする.

4.1.5 状態フィードバックのシミュレーション例

シミュレーション例として二輪車両 (図 4.1) の例をあげる. このシステムの状態方程式は次のようになる.

$$\frac{dx}{dt} = G(x)u$$

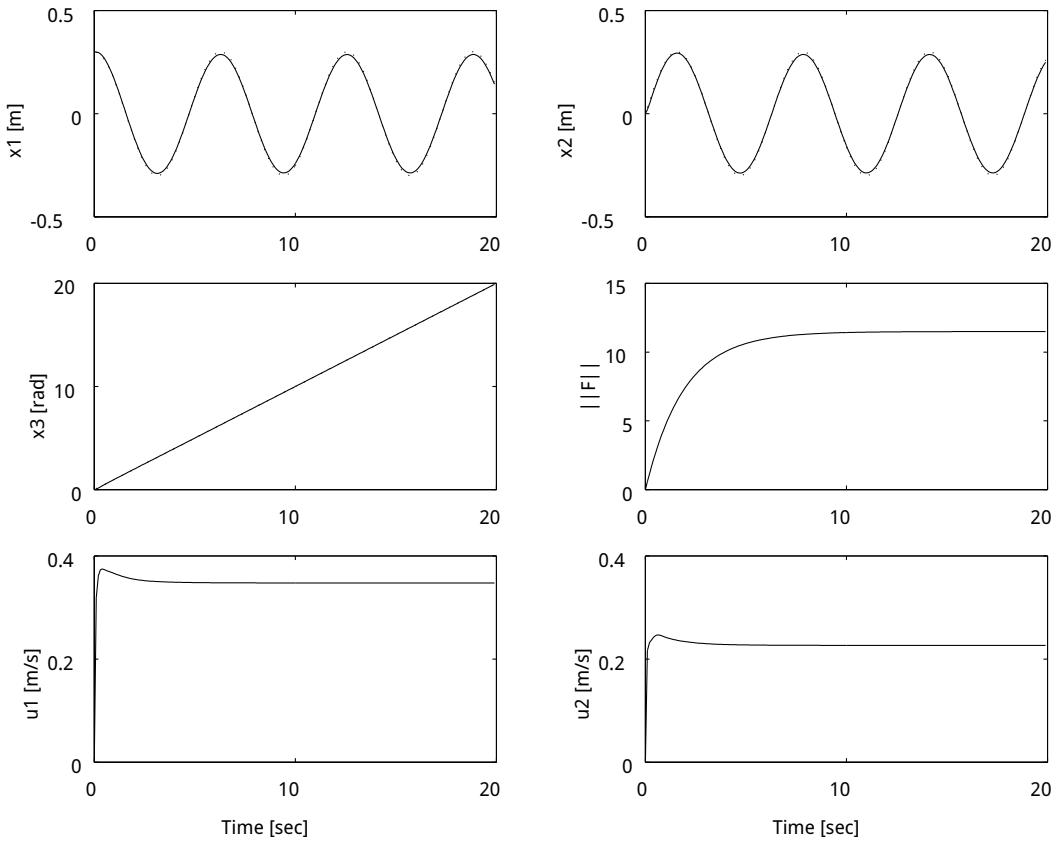

図 4.2: 二輪車両の時間応答

$$G(x) = \frac{1}{2}\begin{bmatrix} -\sin x_3 & -\sin x_3 \\ \cos x_3 & \cos x_3 \\ 1/L & -1/L \end{bmatrix}$$

評価関数を

$$L = \frac{1}{2}\{(p(t) - x(t))^T Q(p(t) - x(t)) + u^T u\}$$

とする. また, 終端でのペナルティを

$$\varphi = \frac{1}{2}(p(t) - x(t))^T S_f (p(t) - x(t))$$

とする. ただし, Q, S_f は対角行列, 時変パラメータ $p(t)$ は目標軌道であり

$$p(t) = \begin{bmatrix} r_0 \cos \omega_0 t \\ r_0 \sin \omega_0 t \\ \omega_0 t \end{bmatrix}$$

で与えられる. 評価区間の長さは $T = T_f(1 - e^{-\alpha t})$ とする.

初期状態 $x_0 = [0.2,\ 0,\ 0]$, 各パラメータは $L = 0.0605\,\mathrm{m}, \omega_0 = 1.0\,\mathrm{rad/s}, R_0 = 0.3\,\mathrm{m}$, . $Q = \mathrm{diag}[10,\ 10,\ 0.1], S_f = \mathrm{diag}[10,\ 10,\ 0.1]$ とした. 安定化の行列は $A_s = -50I$ とす

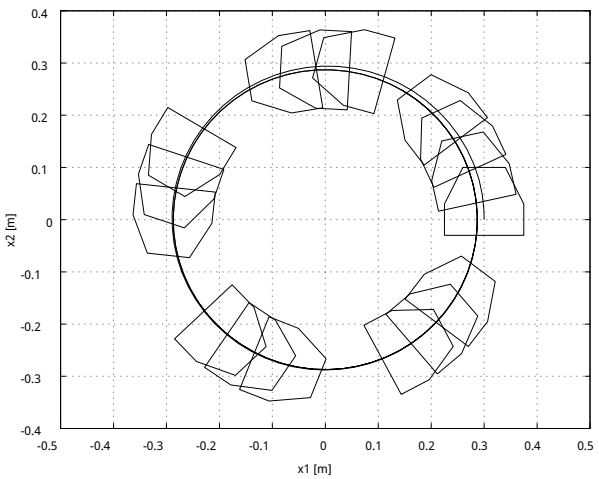

図 4.3: 二輪車両の軌道の軌跡

る. 評価区間の長さは $T_f = 1\,\mathrm{s}$, 評価区間の増加の仕方を決めるパラメータは $\alpha = 0.5$ とする. 数値積分するときの τ 軸の分割数は $20, t$ 軸のきざみは $0.01\,\mathrm{s}$ である.

図 4.2 で状態 x を実線で時変パラメータ p を点線で表し, $u_1,\ u_2$ は制御入力を表す. $\|F\|$ は最適性の誤差のノルムを示す. 図 4.3 は二輪車両の動いた軌跡を表したものである.

このシミュレーションでは 20 秒のシミュレーション時間に対して 4.72 秒で計算ができた. これで実時間での制御が実現できる. なお, 計算は Pentiam III 450MHz の載った Windows マシン上で行った.

4.2 状態推定プログラムの説明

非線形 receding horizon 状態推定のシミュレーションでは実際の状態量 $x(t)$ と推定された状態量 $\hat{x}(t)$ を並べたものを z とし, これの値を t 軸上で計算し求める. 評価関数や状態方程式に依存する部分は自動生成されるが, それ以外の部分はあらかじめ sefunc.c と semain.c に用意されている.

ここでは各ファイルに含まれる関数について述べる.

4.2.1 関数を集めたファイル sefunc.c 中の関数

関数を集めたファイル sefunc.c 中の関数
前節の rhfunc.c に次の関数を付け加えたものになる.

- normal はボックスミューラー法により正規分布する観測雑音と外乱を作るための関数

4.2.2 自動生成されたファイル中の関数

- pfunc は時変パラメータ $p(t)$ の値を求めるための関数

- phix は φ_x の値を求めるための関数
- etax は η_x の値を求めるための関数
- initial は初期値 x0 から z の値を求めるための関数
- wfunc は外乱の推定値 $\hat{w}(t)$ の値を求めるための関数
- yfunc は実際の状態量 $x(t)$ から出力 $y(t)$ の値を求めるための関数
- dyfunc は dy/dt の値を求めるための関数
- zpfuncEL はオイラー・ラグランジェ方程式 $f, -H_x$ の値を求めるための関数
- xpfunc は f から実際の状態量 $x(t)$ の値を求めるための関数
- erropt は F の値を求めるための関数
- boundary は Λ^*, r^* の境界値を求めるための関数
- bkfunc は Λ^*, r^* の τ 軸での微分方程式の値を求めるための関数
- estimate は $d\hat{x}(t)/dt$ の値を求めるための関数
- final はシミュレーションの条件をデータファイルにセーブする関数

4.2.3 main 関数の入ったファイル semain.c 中の関数

- zpfunct は $x^*(0,t) = \hat{x}(t)$ と η_x から $\lambda^*(0,t)$ を求め, それらを初期値として zpfuncEL を後向きに積分して $x^*(-T,t), \lambda^*(-T,t)$ を求める. これらから erropt で F の値を, boundary で Λ^*, r^* の境界値を計算し, bkfunc を前向きに積分して $\tau = 0$ での $\Lambda^*(0,t)$, $r^*(0,t)$ を求める. そして, xfunc から $dx(t)/dt$ の値を estimate から $d\hat{x}(t)/dt$ の値を求める. 最後に zprime に $dx(t)/dt$, $d\hat{x}(t)/dt$ の値を入れる.
- savedata はシミュレーションの結果をデータファイルにセーブするための関数
- main 関数は zpfunct で計算される $\dot{x}(t)$, $d\hat{x}(t)/dt$ を t 軸で数値積分して $x(t)$, $\hat{x}(t)$ を得る. シミュレーションの結果をデータファイルにセーブする. また, そのときの出力 $y(t)$ と推定値 $\hat{x}(t)$ を配列 yt と hat に入れてとっておく.

4.2.4 状態推定のシミュレーション例

シミュレーション例として状態フィードバックと同様に二輪車両の例をあげる. これの状態方程式と出力方程式は次のようになる.

$$\frac{dx}{dt} = G(x)p(t) + w$$

$$y = Cx + v$$

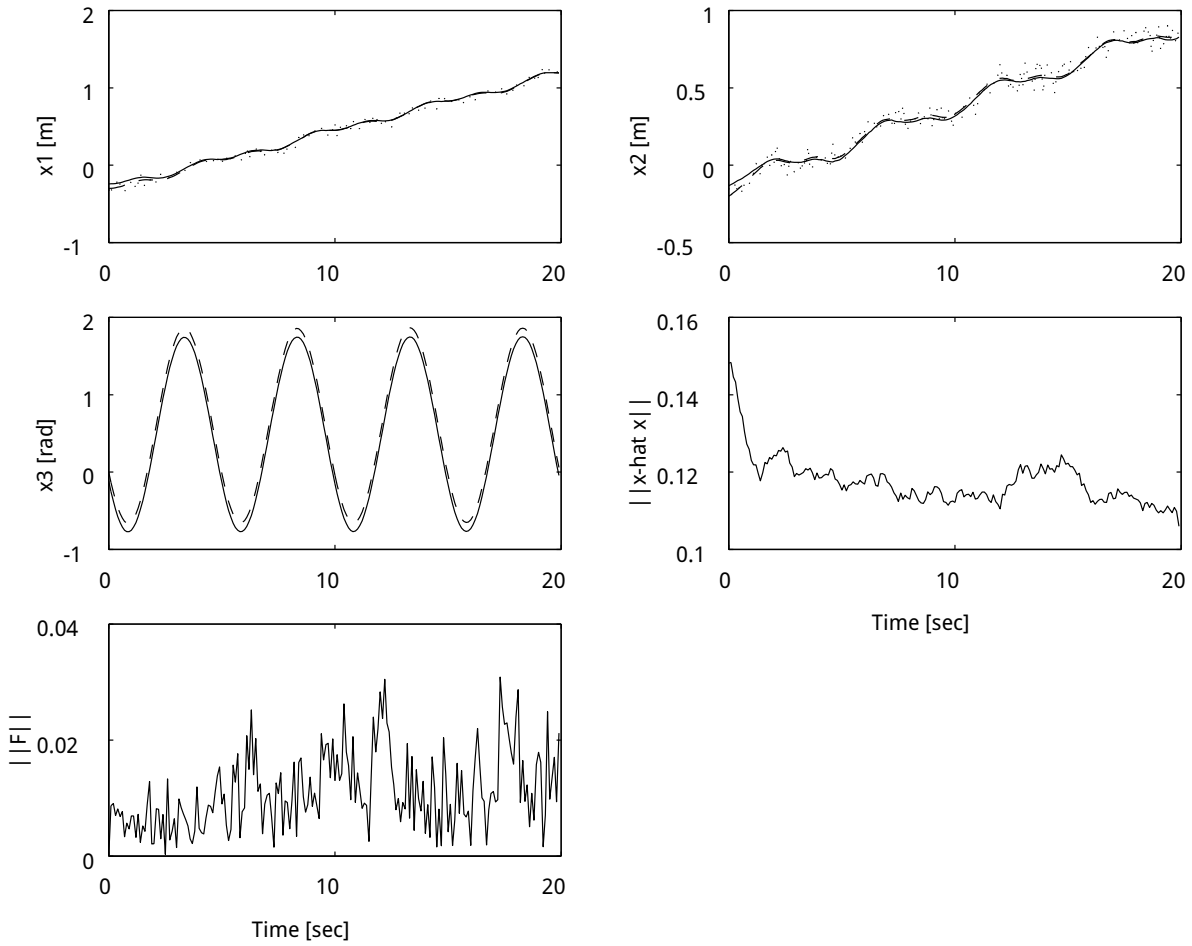

図 4.4: 推定値と実際の状態の初期値が同じ場合の結果

$$G(x) = \frac{1}{2}\begin{bmatrix} -\sin x_3 & -\sin x_3 \\ \cos x_3 & \cos x_3 \\ 1/L & -1/L \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

ここで, $p(t)$ は既知の入力であり,

$$p(t) = \begin{bmatrix} 0.2\sin 0.4\pi t \\ r0\sin(0.4\pi t + 1) \end{bmatrix}$$

評価関数を

$$L = \frac{1}{2}\{(y - C\hat{x})^T Q(y - C\hat{x}) + \hat{w}^T W_f \hat{w}\}$$

とする. また,

$$\eta = 0$$

$$\varphi = \frac{1}{2}\{(\hat{x}(t-T) - x^*(-T,t))^T S_f(\hat{x}(t-T) - x^*(-T,t))\}$$

とする. ただし, Q, W_f, S_f は対角行列である.

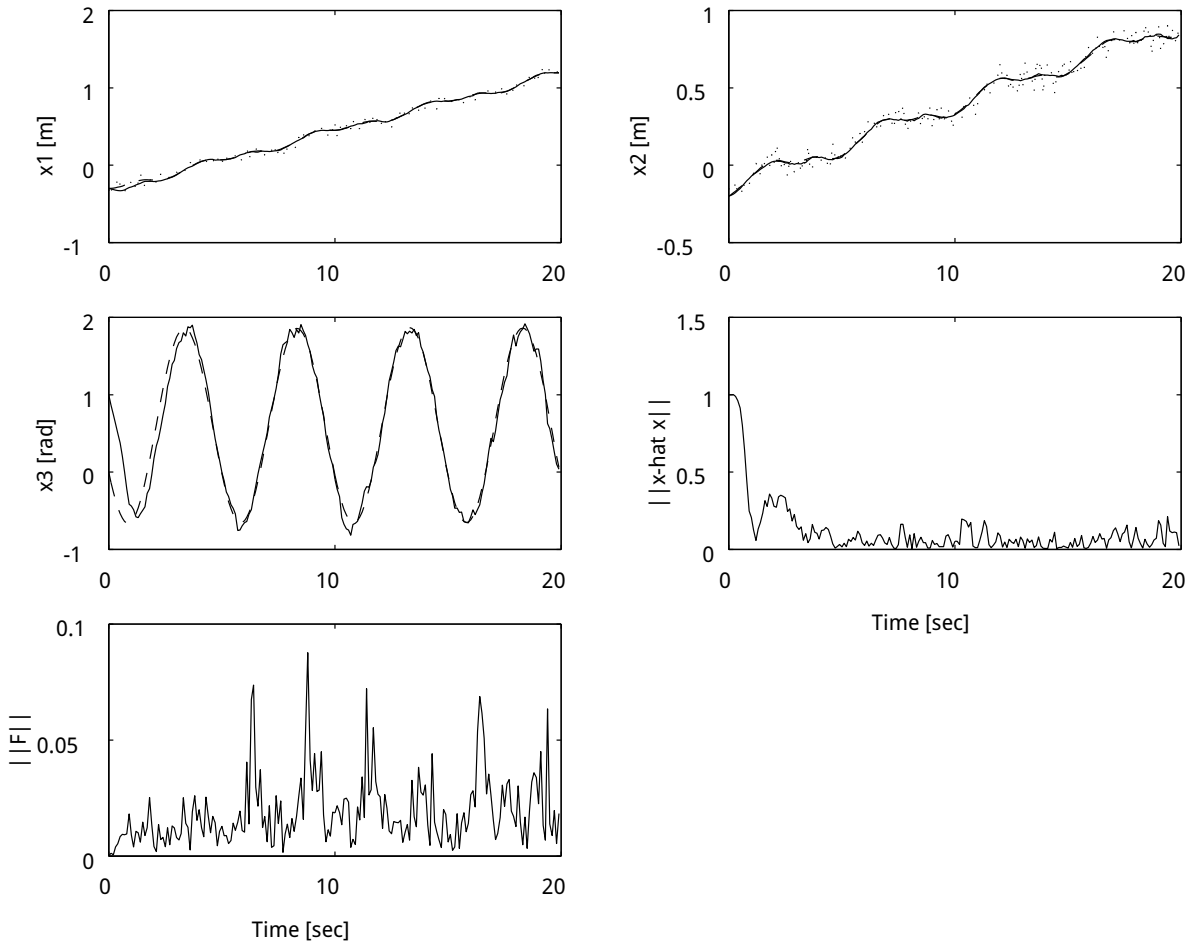

図 4.5: 推定値と実際の状態の初期値がずれた場合の結果

推定値と実際の状態の初期値が同じ場合の結果を図 4.4 に示す. このとき, 初期状態 $x_0 = [-0.3,\ -0.2,\ 0]^T$, 各パラメータは $L = 0.0605\,\mathrm{m}, \omega_0 = 1.26\,\mathrm{rad/s}, R_0 = 0.2\,\mathrm{m}$, $Q = \mathrm{diag}[1,\ 1], S_f = \mathrm{diag}[2,\ 2,\ 2], W_f = \mathrm{diag}[1,\ 1,\ 1]$ とした. 安定化の行列は $A_s = -20I$ とする. 評価区間の長さは $T_f = 1\,\mathrm{s}$, 評価区間の増加の仕方を決めるパラメータは $\alpha = 1$ とする. 数値積分するときの τ 軸の分割数は $50, t$ 軸のきざみは $0.01\,\mathrm{s}$ である. また, 外乱 w と雑音 v は平均 0, 分散 0.05 の正規分布に従うとする.

図 4.4 では状態の推定値 $\hat{x}$ を実線で, 実際の状態の値 x を破線で, 出力 y を点線でそれぞれ表し, 最適性の誤差のノルム $\|F\|$, および推定誤差のノルム $\|x - \hat{x}\|$ も示している.

このシミュレーションでは 20 秒のシミュレーション時間に対して 8.57 秒で計算ができた. これで実時間での状態推定が実現できる.

推定値と実際の状態の初期値を変えた場合の結果を図??に示す. このとき, 実際の状態の初期値を $x_0 = [-0.3,\ -0.2,\ 0]^T$, 推定値の初期値を $\hat{x}_0 = [-0.3,\ -0.2,\ 1]^T$ とする. 各パラメータは $L = 0.0605\,\mathrm{m}, \omega_0 = 1.26\,\mathrm{rad/s}, R_0 = 0.2\,\mathrm{m},\ Q = \mathrm{diag}[1,\ 1], S_f = \mathrm{diag}[2,\ 2,\ 0.01], W_f = \mathrm{diag}[1,\ 1,\ 0.0005]$ とした. 安定化の行列は $A_s = -10I$ とする. 評価区間の長さは $T_f = 1\,\mathrm{s}$, 評価区間の増加の仕方を決めるパラメータは $\alpha = 1$ とする. 数値積分するときの τ 軸の分割数は $50, t$ 軸のきざみは $0.01\,\mathrm{s}$ である. また, 外乱 w と雑音 v は平均 0, 分散 0.05 の

正規分布に従うとする．x_3 の推定値が $1\,\mathrm{rad}(57.3$ 度$)$ ずれたところからスタートしても状態を推定できた．

第5章　結論

本研究では非線形 receding horizon 制御と非線形 receding horizon 状態推定に対して, シミュレーションプログラムの自動生成システムを開発した. このシステムは Mathematica と C 言語が使える環境にあれば, 簡単に扱えるような形になっている. また, 自動生成システムのプログラムの構造は単純であるので, いろいろなものに応用することができる. 例えば, 状態フィードバック制御と状態推定自動生成システムを組み合わせることにより, 出力フィードバック制御や適応制御のシミュレーションプログラムの自動生成システムも作ることができる.

開発した自動生成システムをある程度複雑な問題で用いると, 何十行にもおよぶ数式処理された結果が出力される. これを人間の手で一つのミスもなく行うのは至難の技であり, 開発した自動生成システムがいかに有用であるかがわかる.

なお, 今回開発した自動生成システムを Web 上のいかの URL で公開している.

http://www-newton.mech.eng.osaka-u.ac.jp/~ohtsuka/code/index.htm

謝辞

自動生成システムの開発においてお世話になった大阪大学大学院講師 大塚敏之先生に感謝の意を述べたい. また, 研究に対して御助力頂いた大阪大学大学院教授太田快人先生ならびに大阪大学大学院助手 平田研二先生にも深く感謝している. さらに, いろいろと相談にのってくれた研究室の諸氏にもお礼が言いたい. 最後に大学生活 4 年間を支えてくれた家族に心から感謝している.

付 録A　入力ファイル

A.1　状態フィードバックの入力ファイル input.m

```
dimx=3;
dimu=2;
dimp=3;

fxu={-1/2*Sin[x[3]](u[1]+u[2]),1/2*Cos[x[3]](u[1]+u[2]),(u[1]-u[2])/(2lm)};
qv=Array[q,dimx];
sfv=Array[sf,dimx];
Q=DiagonalMatrix[qv];
Sf=DiagonalMatrix[sfv];
L=(xv-pv).Q.(xv-pv)/2+uv.uv/2;
phi=(xv-pv).Sf.(xv-pv)/2;
pt={r0 Cos[omg0 t],r0 Sin[omg0 t],omg0 t};

MyVarNames={"lm","r0","omg0"};
MyVarValues={0.0605,0.3,1};
MyArrNames={"q","sf"};
MyArrDims={dimx,dimx};
MyArrValues={{1,1,1},{1,1,1}};
tsim0=0;
tsim=20;
tf=1;
ht=0.01;
alpha=0.5;
zeta=50;
x0={0.2,0,0};
dv=20;
dstep=10;
fnout="agrvt2";
fnhd="data\\rvt2";
optu=1;
smpu=3;
```

A.2　状態推定の入力ファイル seinput.m

```
dimx=3;
dimp=2;
dimw=3;
dimy=2;

fxu={-1/2*Sin[x[3]](p[1]+p[2])+w[1],1/2*Cos[x[3]](p[1]+p[2])+w[2],(p[1]-p[2])/(2lm)+w[3]};
fy={x[1],x[2]};
qv=Array[q,dimy];
sfv=Array[sf,dimx];
wfv=Array[wf,dimx];
Q=DiagonalMatrix[qv];
Sf=DiagonalMatrix[sfv];
Wf=DiagonalMatrix[wfv];
L={y[1]-x[1],y[2]-x[2]}.Q.{y[1]-x[1],y[2]-x[2]}/2+wv.Wf.wv/2;
xv=Array[x,dimx];
hxv=Array[hx, dimx];
phi=(hxv-xv).Sf.(hxv-xv)/2;
pt={r0 Sin[omg0 t],r0 Sin[(omg0 t)+1]};
```

```
MyVarNames={"lm","r0","omg0"};
MyVarValues={0.0605,0.2,1.26};
MyArrNames={"q","sf","wf"};
MyArrDims={dimy,dimx,dimx};
MyArrValues={{1,1},{1,1,1},{1,1,1}};

tsim0=0;
tsim=10;
tf=1;
ht=0.01;
alpha=0.5;
zeta=10;
x0={-0.3,-0.2,0};
dv=50;
dstep=10;
st1={0.01,0.01,0.01}
av1={0.0,0.0,0.0}
st2={0.05,0.05}
av2={0.0,0.0}
fnout="esrv11";
fnhd="data\\strv11";
optu=1;
smpu=4;
```

付 録B　自動生成プログラム

■ **Generate Vectors**

```
xv = Array[x, dimx];
lmdv = Array[lmd, dimx];
uv = Array[u, dimu];
pv = Array[p, dimp];
```

■ Generate RHC Algorithm

■ **Euler-Lagrange Equations**

```
H = L + lmdv . fxu;
dlmddt = -Dv[H, xv];
If[smpu != 0, dlmddt = Simplify[dlmddt];];
Hu = Dv[H, uv];
usol = Solve[Table[Hu[i] == 0, {i, dimu}], Table[uv[i], {i, dimu}]];
uxl = Array[u, dimu] /. usol[1];
```

■ **Boudary Conditions for Backward Sweep**

```
phix = Dv[phi, xv];
phix = Simplify[phix];
phix2 = Dv[phix, xv];
dsymphix2 = DeSym[phix2, dimx];
If[smpu != 0, dsymphix2 = Simplify[dsymphix2];]
erropt = lmdv - phix;
zetatemp = zeta;
Clear[zeta];
If[dimp > 0, phixp = Dv[phix, pv];
 If[smu != 0, phixp = Simplify[phixp];];
 cf = (phix2 . fxu + phixp . ∂t pt - dlmddt) onepdtfdt - zeta erropt,
 cf = (phix2 . fxu - dlmddt) onepdtfdt - zeta erropt;];
```

■ **Differential Equations for Bacward Sweep**

```
fx = Dv[fxu, xv];
fu = Dv[fxu, uv];

Hx = Dv[H, xv];
Hu = Dv[H, uv];
Hxx = Dv[Hx, xv];
Hxu = Dv[Hx, uv];
Hux = Transpose[Hxu];
Huu = Dv[Hu, uv];
InvHuu = Inverse[Huu];

As = fx - fu . InvHuu . Hux;
Bs = fu . InvHuu . Transpose[fu];
Cs = Hxx - Hxu . InvHuu . Hux;
```

```
If[smpu >= 2,
As = Simplify[Expand[As]];
Bs = Simplify[Expand[Bs]];
Cs = Simplify[Expand[Cs]];];

S = Sym[Array[s, 1/2 dimx (dimx + 1)], dimx];
cv = Array[c, dimx];

Sdot = -Transpose[As] . S - S . As + S . Bs . S - Cs;
cdot = -(Transpose[As] - S . Bs) . cv;

dsdt = DeSym[Sdot, dimx];
If[smpu >= 3,
dsdt = Simplify[dsdt];
cdot = Simplify[cdot];];

dsdt >> "sdot1.tmp"
cdot >> "cdot1.tmp"

dsdt = << "sdot1.tmp";
cdot = << "cdot1.tmp";
```

■ Generate C Code

■ Load Packages

```
<< FormatFolder`Format2`

<< FormatFolder`Optimize2`
```

■ Setting Options

```
SetOptions[Optimize, OptimizeVariable → {o, Array},
 OptimizeNull -> {List, x, u, lmd, p, s, c, o}, OptimizeCoefficients → True,
 OptimizeProcedure → False];

SetOptions[CAssign,
 AssignOptimize -> ToExpression[optuse], AssignReplace -> { "" -> ""},
 AssignToArray → {""}, AssignPrecision -> 2, AssignBreak → {75, "\\\n"
  }];
```

```
If[smpu >= 2,
As = Simplify[Expand[As]];
Bs = Simplify[Expand[Bs]];
Cs = Simplify[Expand[Cs]];];

S = Sym[Array[s, 1/2 dimx (dimx + 1)], dimx];
cv = Array[c, dimx];

Sdot = -Transpose[As] . S - S . As + S . Bs . S - Cs;
cdot = -(Transpose[As] - S . Bs) . cv;

dsdt = DeSym[Sdot, dimx];
If[smpu >= 3,
dsdt = Simplify[dsdt];
cdot = Simplify[cdot];];

dsdt >> "sdot1.tmp"
cdot >> "cdot1.tmp"

dsdt = << "sdot1.tmp";
cdot = << "cdot1.tmp";
```

■ Generate C Code

■ Load Packages

```
<< FormatFolder`Format2`

<< FormatFolder`Optimize2`
```

■ Setting Options

```
SetOptions[Optimize, OptimizeVariable → {o, Array},
 OptimizeNull -> {List, x, u, lmd, p, s, c, o}, OptimizeCoefficients → True,
 OptimizeProcedure → False];

SetOptions[CAssign,
 AssignOptimize -> ToExpression[optuse], AssignReplace -> { "" -> ""},
 AssignToArray → {""}, AssignPrecision -> 2, AssignBreak → {75, "\\\n"
  }];
```

■ Optimize Dimensions

```
If[optu == 1, lept = Length[Optimize[pt][[1]]];
leph = Length[Optimize[phix][[1]]];
leux = Length[Optimize[uxl][[1]]];
lejo1 = Length[Optimize[Join[fxu, dlmddt]][[1]]];
leer = Length[Optimize[erropt][[1]]];
lejo2 = Length[Optimize[Join[dsymphix2, cf]][[1]]];
lejo3 = Length[Optimize[Join[dsdt, cdot]][[1]]];
leno = Max[{lept, leph, leux, lejo1, leer, lejo2, lejo3}];];
```

■ Conversion of Arrays

```
ArrNames = Join[MyArrNames, {"x", "u", "lmd", "p", "s", "c"}];
ArrDims = Join[MyArrDims, {dimx, dimu, dimx, dimp, 1/2 dimx (dimx + 1), dimx}];
ArrCnv = {};
Do[ArrCnv = Join[ArrCnv, Table[ArrNames[[j]] <> "[" <> ToString[i] <> "]" →
      ArrNames[[j]] <> "[" <> ToString[i - 1] <> "]", {i, ArrDims[[j]]}]],
 {j, Length[ArrNames]}];
ArrCnv2 = ArrCnv;
Do[
 ArrCnv2 = Join[ArrCnv2,
   Table["o[" <> ToString[i] <> "]" → "o[" <> ToString[i - 1] <> "]", {i, leno}]]
]
```

■ Output CProgram

```
outfile = StringJoin[fnout, ".c"];
Splice["agrv16.mc", outfile, FormatType -> OutputForm];
```

付 録C　自動生成の元となるファイル

C.1　状態フィードバック用 AutoGen.mc

```
#include "rhfunc.c"

<*st=""; If[dimp>0, st="#define P_IS_USED\n"]; st*>
/*-------------- Dimensions -------------- */

#define DIMX   <*dimx*>
#define DIMU   <*dimu*>
#define DIMP   <*dimp*>
#define DIMZ   <*2dimx*>
#define DIMRIC <*dimx(dimx+1)/2*>
#define DIMBK  <*dimx(dimx+1)/2+dimx*>
#define DIME   <*dimx*>

/*-------------- Global Variagles -------------- */

double tsim0 = <*tsim0*>;
double tsim = <*tsim*>;
double tf = <*tf*>;
double ht = <*ht*>;
double alpha = <*alpha*>;
double zeta = <*zetatemp*>;
double x0[DIMX] = <*x0*>;
int dv = <*dv*>;
int dstep =   <*dstep*>;
#define FNLEN  <*StringLength[fnhd]+11*>
char fnhd[FNLEN] = <*CForm[fnhd]*>;
char fnmx[FNLEN+6];
char fnmu[FNLEN+6];
char fnme[FNLEN+6];
char fnmc[FNLEN+6];
char fnmp[FNLEN+6];

/*-------------- Global Variables Defined by User -------------- */

<*SetOptions[CAssign,
AssignToArray->ToExpression[ArrNames],
AssignReplace->ArrCnv2,
AssignIndent->"    "];
SetOptions[Optimize,
OptimizeNull->{List,x,u,lmd,p,s,c,o}];
st="";
Do[st=st<>"double  "<>MyVarNames[[i]]<>" = "<>
      ToString[ MyVarValues[[i]] ] <>";\n",{i,Length[MyVarNames]}];
st *>
<* st="";
Do[st=st<>"double  "<>MyArrNames[[i]]<>"["<>ToString[MyArrDims[[i]]]<>
      "] = "<>ToString[ MyArrValues[[i]] ] <>";\n",{i,Length[MyArrNames]}];
st *>
<* ou="";If[dimp>0,ou="/*-------------- Time-Variant Parameters --------------*/\n"<>
"void pfunc(double t, double p1[])\n{\n";
    If[lept!=0,ou=ou<>"    double o["<>ToString[lept]<>"];\n"];
     ou=ou<>ToString[CAssign[p1,pt]]<> "\n}\n"];ou*>
/*-------------- dPhi/dx -------------- */

void phix(double t, double x[], double phx1[])
```

```
{
<*If[leph!=0,ou="    double o["<>ToString[leph]<>"];\n",ou=""];
  If[ !(FreeQ[phix,p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou *>
<*CAssign[phx1,phix]*>
}

/*-------------- Initial Conditions -------------- */

void initial(double t0, double z[])
{
    double lmdf[DIMX];

    mmov(1,DIMX, x0, z);
    phix(t0, x0, lmdf);
    mmov(1,DIMX, lmdf, z+DIMX);
}

/*-------------- Control Input -------------- */

void ufunc(double t, double z[], double u1[])
{
    double x[DIMX], lmd[DIMX];
<*If[leux!=0,ou="    double o["<>ToString[leux]<>"];\n",ou=""];
  If[!(FreeQ[uxl,p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
<*CAssign[u1,uxl]*>
}

/*-------------- Euler-Lagrange Equations -------------- */

void zpfuncEL(double t, double z[], double zprime[])
{
    double x[DIMX], lmd[DIMX], u[DIMU];
<*If[lejo1!=0,ou="    double o["<>ToString[lejo1]<>"];\n",ou=""];
  If[!(FreeQ[Join[fxu,dlmddt],p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    ufunc(t, z, u);
    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
<*CAssign[zprime,Join[fxu,dlmddt]]*>
}

/*-------------- Error in Optimality Condition -------------- */

void erropt(double t, double z[], double termerr[])
{
    double x[DIMX], lmd[DIMX];
<*If[leer!=0,ou="    double o["<>ToString[leer]<>"];\n",ou=""];
  If[\[InvisibleSpace]!(FreeQ[erropt,p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
<*CAssign[termerr,erropt]*>
}

/*-------------- Boundary Conditions for Backward Sweep -------------- */

void boundary(double t, double z[], double bkf[])
{
    double onepdtfdt, x[DIMX], lmd[DIMX], u[DIMU];
<*If[lejo2!=0,ou="    double o["<>ToString[lejo2]<>"];\n",ou=""];
  If[\[InvisibleSpace]!(FreeQ[Join[dsymphix2,cf],p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    ufunc(t, z, u);
    mmov(1,DIMX, z,x);
```

```
    mmov(1,DIMX, z+DIMX,lmd);
    onepdtfdt = 1 + alpha * tf * exp(-alpha * ts);

<*CAssign[bkf,Join[dsymphix2,cf]]*>
}

/*-------------- Differential Equations for Backward Sweep -------------- */

void bkfunc(double t, double bk[], double bkprime[])
{
    double s[DIMRIC], c[DIMX];
    double z[DIMZ], x[DIMX], lmd[DIMX], u[DIMU];
<*If[lejo3!=0,ou="    double o["<>ToString[lejo3]<>"];\n",ou=""];
  If[ !(FreeQ[Join[dsdt,cdot],p[_]]),
      ou=ou<>"    double p[DIMP];\n\n    pfunc(t, p);\n"];ou*>
    mmov(1,DIMRIC, bk,s);
    mmov(1,DIMX, bk+DIMRIC,c);

    table1(dv+1,DIMZ, tau, ztau, t, z);
    mmov(1,DIMX, z,x);
    mmov(1,DIMX, z+DIMX,lmd);
    ufunc(t, z, u);
<*CAssign[bkprime,Join[dsdt,cdot]]*>
}

/*-------------- Save Simulation Conditions -------------- */

void final(FILE *fp)
{
    int i;

    fprintf(fp, "Simulation Result by <*outfile*> \n");
    fprintf(fp, "tsim = %g\n", tsim);
    fprintf(fp, "ht = %g, DSTEP = %d\n", ht, dstep);
    fprintf(fp, "tf = %g, dv = %d, alpha = %g, zeta = %g \n", tf, dv,
alpha, zeta);
<*st="";
      Do[st=st<>"\n    fprintf(fp, \""<>MyVarNames[[i]]<>" = %g\\n\",
"<>MyVarNames[[i]]<>" );",
         {i,Length[MyVarNames]}];st*>
<*st="";
      Do[st=st<>"\n    fprintf(fp, \""<>MyArrNames[[i]]<>" = (%g\",
"<>MyArrNames[[i]]<>"[0] );\n"<>"    for(i=1;
i<"<>ToString[MyArrDims[[i]]]<>"; i++)\n"<>"
       fprintf(fp,\",%g\", "<>MyArrNames[[i]]<>"[i] );\n"<>"    fprintf(fp,
\")\\n\" );\n",{i,Length[MyArrNames]}]
st*>
}

#include "rhmain.c"
```

付 録D　main関数の入ったファイル

D.1　状態フィードバック用rhmain.c

```
/*-------------------------------------------------------
  RHMAIN.C
dv(t)/dt, v=[x(0), lmd(0)]
Continuation in terminal time
ODE: ADAMS/RKG
T.Ohtsuka  '96/12/29~ (rhsf_rv.c)
 '97/08/21~ (rvtr.c)
'97/10/30~'97/10/31  (rhmain.c)
-------------------------------------------------------*/
void zpfunct(double t, double z[], double zprime[])
{
     int i,j;
     double taut, zp3[3][DIMZ];
     double bk[DIMBK], bkp3[3][DIMBK], termerr[DIME];
     double smat[DIMX][DIMX], c[DIMX];
     ts = t;
     zpfuncEL(t, z, zprime);
     tauf = tf * (1. - exp(-alpha * t) );
     htau = tauf / dv;
     mmov(1,DIMZ, z, ztau[0]);
     for(taut = t, i=0; (i < 3)&&(i < dv); taut += htau, i++){
          nfrkgex(zpfuncEL,taut,ztau[i],htau,DIMZ,ztau[i+1],zp3[i]);
          tau[i] = taut;
     }
     for( ; i < dv; taut += htau, i++) {
          nfadams(zpfuncEL,taut,ztau[i],zp3[2],zp3[1],zp3[0],htau,DIMZ,ztau[i+1]);
          tau[i] = taut;
     }
     tau[i] = taut;
     if( (isim%dstep == 0)  && ( (int)((t-tsim0)/ht+0.6) == isim) ){
          erropt(t,ztau[dv], termerr);
          for(i=0; i<DIME; i++) fprintf(fpe, "%g ", (float)termerr[i]);
          fprintf(fpe, "\n");
     }
     boundary(tau[dv],ztau[dv], bk);
     for(i = dv; (i > dv-3)&&(i > 0); i--)
          nfrkgex(bkfunc,tau[i],bk,-htau,DIMBK,bk, bkp3[dv-i]);
     for( ; i > 0; i--)
          nfadams(bkfunc,tau[i],bk,bkp3[2],bkp3[1],bkp3[0],-htau,DIMBK,bk);
     mmov(1,DIMX, bk+DIMRIC, c);
     madd(1,DIMX, zprime+DIMX, c, zprime+DIMX);
}
void savedata(FILE *fpx, FILE *fpu, FILE *fpp, double t, double z[])
{
     int i;
     double u[DIMU], p[DIMP];

     for(i=0; i<DIMZ; i++) fprintf(fpx, "%g ", (float)z[i]);
     fprintf(fpx, "\n");
     ufunc(t,z, u);
     for(i=0; i<DIMU; i++) fprintf(fpu, "%g ", (float)u[i]);
     fprintf(fpu, "\n");
#ifdef P_IS_USED
     pfunc(t, p);
     for(i=0; i<DIMP; i++) fprintf(fpp, "%g ", (float)p[i]);
```

```
     fprintf(fpp, "\n");
#endif
/*
     printf("%g :", (float)t );
     for(i=0; i<DIMZ; i++) printf("%g ", (float)z[i] );
     printf("\n");
     */
}
int main(void)
{
     int nsim;
     double t, z[DIMZ], f3[3][DIMZ];
     FILE *fpx, *fpu, *fpp, *fpc; /* *fpe is a global var. */
     clock_t tstart, tend, tbefore, tafter, ttotal=0;

     nsim = (tsim -tsim0) / ht;
     tau = defvector(dv+1);
     ztau = defmatrix(dv+1, (int)DIMZ);
     strcpy(fnmx, fnhd); strcpy(fnmu, fnhd);
     strcpy(fnme, fnhd); strcpy(fnmp, fnhd);
     strcpy(fnmc, fnhd);

     strcat(fnmx,"x.dat") ;
     strcat(fnmu,"u.dat") ;
     strcat(fnme,"e.dat") ;
     strcat(fnmp,"p.dat") ;
     strcat(fnmc,"c.dat") ;
     fpx = fopen( fnmx, "w");
     fpu = fopen( fnmu, "w");
     fpe = fopen( fnme, "w");
#ifdef P_IS_USED
     fpp = fopen( fnmp, "w");
#else
     fpp = fpx;
#endif
     fpc = fopen( fnmc, "w");

     if( (fpx==NULL)||(fpu==NULL)||(fpe==NULL)||(fpp==NULL)||(fpc==NULL) ){
          printf(" ****** can't open data files \n");
          exit(1);
     }
     initial(tsim0, z);
     printf(" Start  \n");
     tstart = clock();
     for(t=tsim0, isim=0; (isim < 3)&&(isim < nsim); t+=ht, isim++){
          if(isim%dstep == 0 ) savedata(fpx, fpu, fpp, t, z);
          tbefore = clock();
          nfrkgex(zpfunct,t,z,ht,DIMZ, z,f3[isim]);
          tafter = clock();
          ttotal += tafter - tbefore;
     }
     for( ; isim < nsim; t+=ht, isim++) {
          if(isim%dstep == 0 ) savedata(fpx, fpu, fpp, t, z);
          tbefore = clock();
          nfadams(zpfunct,t,z,f3[2],f3[1],f3[0],ht,DIMZ, z);
          tafter = clock();
          ttotal += tafter - tbefore;
     }
     if(isim%dstep == 0 ) savedata(fpx, fpu, fpp, t, z);
     tend = clock();
     printf(" End \n");
     printf("CPU Time (Only for Simulation): %g sec\n", ( (float)(TDIV*ttotal/(CLOCKS_PER_SEC)))/TDIV );
     printf("Start ~ End (Simulation + Data Save) : %g sec\n", ( (float)(TDIV*(tend-tstart)/(CLOCKS_PER_SEC)))/TDIV );
     printf("CLOCKS_PER_SEC = %ld\n", (long)CLOCKS_PER_SEC);
     fclose(fpx);
     fclose(fpu);
     fclose(fpe);
#ifdef P_IS_USED
```

```
     fclose(fpp);
#endif
     final(fpc);
     fclose(fpc);
}
```

参考文献

[1] 赤坂 隆：数値計算, コロナ社

[2] 吉田和哉：The SpaceDyn:ダイナミクス計算のための MATLAB ツールボックス, 計測と制御, Vol38, No2, pp138-143, 1999

[3] T.Ohtsuka: Nonlinear Receding–Horizon State Estimation with Unknown Disturbances, 計測自動制御学会論文集, Vol.35, No.10, pp.1253–1260, 1999.

[4] 大塚敏之：非線形最適フィードバック制御のための実時間最適化手法, 計測と制御, Vol.36, No.11, pp.776–781, 1997.

[5] T.Ohtsuka: Real–time Optimization Algorithm for Nonliner Receding–Horizon Control, Automatica, Vol.33, No.6, pp.1147–1153, 1997.

[6] T.Ohtsuka: Time–Variant Receding–Horizon Control of Nonliner Systems, Journal of Guidance, Control, and Dynamics, Vol.21, No.1, pp.174–176, 1997.

[7] A.E. Bryson Jr. and Y.–C. Ho：Applied Optimal Control, Hemisphere, 1975.

[8] 加藤寛一郎：工学的最適制御, 東京大学出版会, 1988.

[9] 川崎晴久：数式処理 Maple とロボット解析への応用, コロナ社, 1995.

[10] 川崎晴久, 清水年美：ロボット数式処理システム ROSAMII の開発, 日本ロボット学会誌, Vol.17, No.3, pp.408–415, 1999.

[11] R. M. Murray and Z. Li and S. S. Sastry: A Mathematical Introduction to robotic manipulation, CRC press, 1994.

[12] M.Sofroniou: An Efficient Symbolic–Numeric Environment By Extending Mathematica's Format Rules, ftp://ftp.center.osaka-u.ac.jp/MathSouce/Enhancements/Interfacing/Other/0205–254/, 1994

[13] T.Robb: ftp://ftp.center.osaka-u.ac.jp/MathSouyce/Enhancements/Language/0205–939/, 1993.

[14] S.Wolfram：The Mathematica Book, 3rd ed., Wolfram Media/Combridge University Press, 1996.