〔農工研技報　212
13 ～ 28，2012〕

パイプライン非定常流の剛性モデル・閉路解析における オブジェクト指向プログラミングによる数式処理の自動化

田中良和・中田　達・樽屋啓之

目　次

Ⅰ　緒　言…… 13
Ⅱ　状態方程式の導出過程における
　　数式処理の必要性…… 14
Ⅲ　オブジェクト指向プログラミングによる
　　数式処理の実現方法…… 17
Ⅳ　解析例…… 21
Ⅴ　結　言…… 23
参考文献…… 24
Summary…… 28

Ⅰ　緒　言

近年の営農変化に伴って用水需要が変化し，多くの送配水パイプラインシステムは当初の水利用計画と異なる新たな水利用計画に合致した合理的な水管理が求められている。このような用水需要に対して弾力的な水管理方法の改善や施設の改修を実施するために，数値解析によって計画や設計を検討することが必要である。パイプラインシステム内の流量輸送の緩やかな過渡現象を長時間にわたって数値解析するには，バルブやポンプの操作に伴う急激な水理過渡現象を把握するための利用される弾性体モデルによる数値解析はコンピュータの CPU の使用時間が長くなるために不向きである（内藤ら，1983）。このような水理現象の数値解析手法として，鬼塚（1971）は剛性モデル理論による非定常流解析が有効であることを明らかにした。剛性モデル理論による非定常流解析とは，管路系に発生する緩やかな非定常水理現象であるサージング現象を，水の圧縮性と管体の弾性変形を考慮しない剛体水柱理論で近似できる範囲の現象として定式化し，数値解析するものである（鬼塚，1981）。方法として，閉路解析（鬼塚，1977）と接続解析（島田，1991）が知られている。

接続解析は，パイプラインシステムの流量と運動量の連続性を表す 1 階の連立常微分方程式（システム方程式と呼ぶ）から状態方程式を導き，時間積分を行う数値解析手法である。接続解析において導出した状態方程式の数は状態変数よりも多いので，状態方程式の数を状態変数に合わせる方法が，内藤ら（1983）と島田（1991）によって提案されている。

他方，閉路解析は，パイプラインシステムをモデル化したシステムグラフを全域木と補木の情報を得れば，自動的に流量の独立変数と従属変数が判明し，状態方程式が得られる利点がある。ただし，エネルギー基準面との仮想的な経路をパイプラインシステムに追加して閉じたグラフを作成し，閉路情報を利用して状態方程式を導出する作業が必要である。このため，弁の完全閉鎖を扱う問題や調整池の設置の必要性を検討する問題などでは，新たに閉じたグラフを作成し直す必要があり，状態方程式の導出が面倒であった。しかし，①閉路情報を得る作業と②数式処理を行い状態方程式を導出する作業を自動化するように支援すれば，閉路解析は技術者が扱いやすい手法となると期待できる。①については，著者ら（2011）が稿を分けて整理し，全域木探索法と Ternary network flow 法（Doris and Stephan, 1981）を用いて閉路情報を得る方法を提案した。本稿では，②について提案を行う。従来，数式処理は Fortran 言語のような手続き型言語では難しい処理であり，数式処理システム（Reduce, Maxima，および Mathematica など）を利用すれば行うことができるが，数式処理システムを利用した数値シミュレーションは農業土木分野において普及しているとは言い難い。そこで，近年インターネットから数値計算まで広く普及してきたオブジェクト指向プログラミングを利用して，簡易な数式処理システムを構築し，剛性モデル理論による閉路解析を導出する方法を提案する。

水利工学研究領域　水路システム担当
平成 23 年 12 月 14 日受理
キーワード：パイプライン，グラフ理論，状態方程式，非定常流解析，数式処理，オブジェクト指向言語

Ⅱ　状態方程式の導出過程における数式処理の必要性

1　剛体モデル・閉路解析に基づく状態方程式の導出の流れ

剛性モデル・閉路解析では，作成したシステムグラフを全域木と補木の辺に分離できれば，システムグラフ内の閉路と接続関係の情報が明らかになる。これらの情報を利用した流量連続条件とエネルギー連続条件のシステム方程式を数式処理によって変形することにより，状態方程式を得ることができる。この状態方程式を時間積分することによって，状態変数の数値解を得て，非状態変数との関係式から非状態変数の数値解が得られる。導出過程をフローチャートで表すと **Fig.1** の流れになる。

本章では，剛体モデル・閉路解析に基づく状態方程式の導出過程を説明して，この導出過程を自動化するためには，①全域木と補木の辺の分離と②数式処理による式の変形が必要であることを示す。さらに，②の数式処理による式の変形については，どのような式の変形が必要であるかを具体的に整理する。

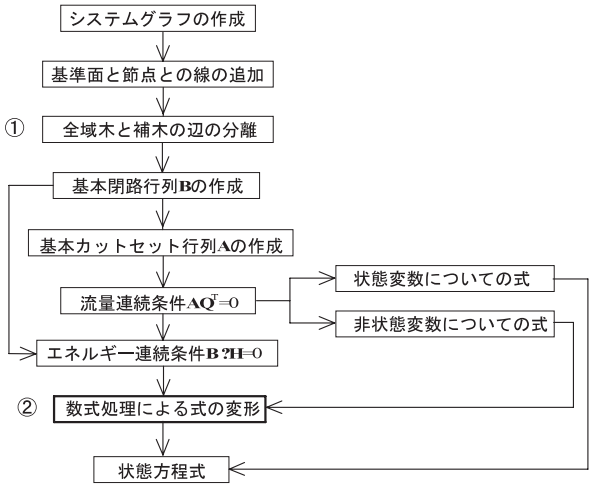

Fig.1　状態方程式の導出過程のフローチャート
Flowchart of derivation process about state equations

a　システムグラフの作成

パイプラインシステムの接続情報は，グラフ理論に基づき，節点とそれらを連結する辺で表す（鬼塚，1998）。1 方向の辺で構成されるグラフを有向グラフ（以下，システムグラフと呼ぶ）である。具体的に **Fig.2** のパイプラインシステムについて考える。

ここで，水頭の基準面を記号 D で表し，既知の水頭境界を H_0，H_5，中間調整池の水頭を h_1，ファームポンドの水頭 h_4，既知の流量境界を q_{10}，その他の q_i を管の流量とする。パイプラインシステムを有向グラフで表したシステムグラフを **Fig.3** に示す。

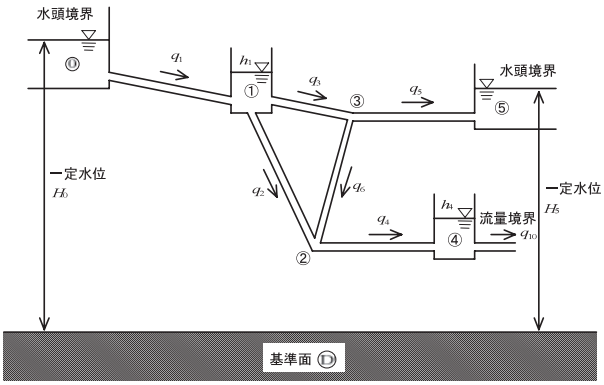

Fig.2　パイプラインシステム例
Example of pipeline system

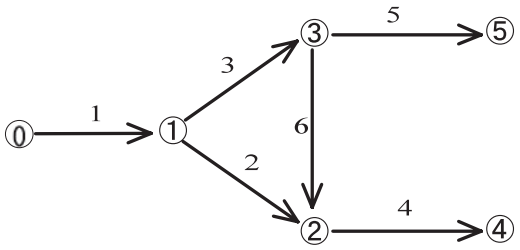

Fig.3　パイプラインシステムのシステムグラフ
System graph of pipeline system

b　基準面と接点とをつなげる辺の追加

閉路解析では，システムグラフ内にある次の 3 つの条件に対して，基準面 D へ辺を追加してパイプラインシステムを複数の閉路（システムグラフ内のある接点を出発して他の接点を経由して，出発した接点に戻った時の経路）で構成する作業が必要である。

①ファームポンドや中間調整池など未知の水頭をもつ水槽

水槽における連続条件を満たすのに，貯留量を流量として計算する必要がある。その流量を表すための辺が必要である。

②既知の流量境界

パイプラインシステムについての流入量や流出量を表すのに，基準面 D との流量のやりとりを表す必要がある。

③既知の水頭境界

既知の水頭の大きさを表すのに，基準面 D との水頭差で表す必要がある。

この 3 つの条件について，基準面 D への辺を追加したシステムグラフを **Fig.4** に示す。

c　全域木と補木の分離

システムグラフは全域木と補木の辺に分解できる。ここで，全域木とは，閉路を 1 つも持たないが，すべての点に接続している辺の組み合わせである。補木は全域木の辺に追加することによって閉路を作ることができる辺の集合である。基準面 D へ接続する辺は，前節の 3 つの条件によって以下のように分類される。

①未知の水頭をもつ水槽から基準面 D への辺は，木

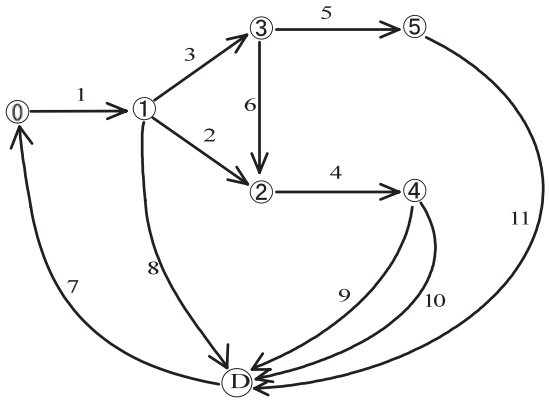

Fig.4　パイプラインシステムに境界条件を追加したシステムグラフ
System graph which added boundary condition to pipeline system

に属する。

②既知の流量境界として基準面 D へつながる辺は，補木に属する。

③既知の水頭境界から基準面 D への辺は，木に属する。

その他の辺の指定はある程度自由度があるが，補木の数は，システムグラフを構成する辺の数を n 本，接点の数を m 個とすると，$(n - m + 1)$ 本でなければならない。著者ら（2011）は，稿を分けてシステムグラフを全域木と補木に分離する方法を提案した。

d　基本閉路行列 B の作成

基本閉路は，補木の辺を 1 本しか含まない閉路である。よって，基本閉路の数は補木の辺の数と等しい。**Fig.4** において，仮に全域木の辺 ｛3，4，7，8，9，11｝，補木の辺 ｛1，2，5，6，10｝ を選択して，行列の成分 b_{ij} の値を次のとおりに分類すると，（1）式の基本閉路行列 **B** を作成できる。

$$b_{ij}=\begin{cases} 1\cdots & \text{辺 } j \text{ が閉路 } i \text{ に含まれ，かつ，同じ向きの場合。} \\ -1\cdots & \text{辺 } j \text{ が閉路 } i \text{ に含まれ，かつ，逆向きの場合。} \\ 0\cdots & \text{辺 } j \text{ が閉路 } i \text{ に含まれない場合。} \end{cases}$$

$$B=\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \cdots\cdots (1)$$

ここで，**B** は行方向（横方向）に全域木の辺を若い番号順に並べ，次に，補木の辺を若い順に並べ替えている。

さらに，**B** の列方向（縦方向）の大きさは基本閉路の個数であり，その並べ方は基本閉路に含まれる補木の辺の若い順である。つまり，B を構成する木の部分を行列 B_t，補木の部分を行列 B_c で表すと，$\mathbf{B}=[B_t,\ B_c]$ で表せる。

e　基本カットセット行列 A の作成

B_t の転置行列をマイナスにした行列 $-B_t^{\mathrm{T}}$ を全域木の辺の個数の大きさの単位行列 I に追加した行列 $\mathbf{A}=[I,\ -B_t^{\mathrm{T}}]$ は，基本カットセット行列である。

$$A=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \quad \cdots\cdots (2)$$

ここで，成分 a_{ij} の値は，以下のような意味を持つ。

$$a_{ij}=\begin{cases} 1\cdots & \text{辺 } j \text{ が基本カットセット } i \text{ に含まれ，かつ，向きが全域木と同じである時。} \\ -1\cdots & \text{辺 } j \text{ が基本カットセット } i \text{ に含まれ，かつ，向きが全域木と反対である時。} \\ 0\cdots & \text{辺 } j \text{ が基本カットセット } i \text{ に含まれていない時。} \end{cases}$$

また，**A** の列方向の順番は **B** と同じであるが，行方向はカットセット内に含まれる全域木が若い順にカットセット（システムグラフを 2 つに分断する辺の集合）を並べている。各カットセットは全域木の辺を 1 本だけ含む。

以下，システムグラフの **A**，**B** を利用した流量連続条件とエネルギー連続条件から状態方程式が導出される。導出過程において，数式処理が必要になる。以下の節では，それらの数式処理方法を詳しく説明する。

f　流量連続条件

パイプラインシステム内の辺の流量ベクトル **Q** は次式で表すことができる。行方向の成分の順番は **B** と同じである。

$$Q=\begin{bmatrix} q_3 & q_4 & q_7 & q_8 & q_9 & q_{11} & q_1 & q_2 & q_5 & q_6 & q_{10} \end{bmatrix} \cdots (3)$$

ただし，水槽 1 や水槽 4 と基準面 D との辺の流量 q_8，q_9 は貯留変化量であるので，未知の水頭の記号 h の上のドットは時間微分を表すと，次式になる。

$$\left.\begin{aligned} q_8 &= A_1\dot{h}_1 \\ q_9 &= A_4\dot{h}_4 \end{aligned}\right\} \quad \cdots\cdots (4)$$

よって，流量ベクトル **Q** は，

$$Q=\begin{bmatrix} q_3 & q_4 & q_7 & A_1\dot{h}_1 & A_4\dot{h}_4 & q_{11} & q_1 & q_2 & q_5 & q_6 & q_{10} \end{bmatrix} \quad \cdots\cdots (5)$$

流量の連続条件 $\mathbf{AQ}^{\mathrm{T}}=0$ として，h_i の時間微分を含んだ項を左辺に残し，その他の項を右辺に移項して整理すると，次式が得られる。

$$\left.\begin{aligned} A_1\dot{h}_1 &= q_1 - q_2 - q_5 - q_6 \\ A_4\dot{h}_4 &= q_2 + q_6 - q_{10} \end{aligned}\right\} \quad \cdots\cdots (6)$$

ここで，水頭 h_1，h_4 のある時刻の状態は時間積分して求める必要があるので，状態変数である。流量 q_1，q_2，q_5，q_6，および q_{10} は，補木に属する辺における状態変数である。

その他，木に属する辺における流量の従属変数を左辺に残し，その他の項を同様に整理すると，流量の従属変数と独立変数の関係式が導かれる。

$$\left.\begin{aligned} q_3 &= q_5 + q_6 \\ q_4 &= q_2 + q_6 \\ q_7 &= q_1 \\ q_{11} &= q_5 \end{aligned}\right\} \quad \cdots\cdots (7)$$

ここで，流量 q_3，q_4，q_7，および q_{11} は，木に属する辺における流量であり，これらは従属変数である。

（6）式を書き直すと一般的な形式は次式になる。

$$\begin{bmatrix} \dot{h}_1 \\ \dot{h}_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{A_1} & 0 \\ 0 & \frac{1}{A_4} \end{bmatrix} \begin{bmatrix} q_1 - q_2 - q_5 - q_6 \\ q_2 + q_6 - q_{10} \end{bmatrix} \cdots\cdots (8)$$

g　エネルギー連続条件

損失水頭ベクトル **ΔH** は次式で表すことができる。ただし，行方向の成分の順番は **B** と同じである。

$$\Delta H = \begin{bmatrix} h_1 - h_3 & h_2 - h_4 & -H_0 & h_1 & h_4 & H_5 & H_0 - h_1 \\ & & & h_1 - h_2 & h_3 - H_5 & h_3 - h_2 & h_4 - 0 \end{bmatrix} \cdots (9)$$

接点 i と接点 j における単一管 k について，管の慣性定数を L_k，管の流量抵抗係数を K_k とすると，運動方程式は次式で表される。

$$L_k\dot{q}_k = h_i - h_j - K_k|q_k|q_k \cdots\cdots (10)$$

$$L_k = \frac{l_k}{A_k g} \cdots\cdots (11)$$

$$K_k = \frac{f_k l_k}{2gD_k A_k^2} \cdots\cdots (12)$$

ここで，l_k は管の延長，A_k は管の断面積，D_k は管の直径，f_k は管の摩擦損失係数である。

よって，損失水頭ベクトルは次式になる。

$$\Delta H^T = \begin{bmatrix} L_3\dot{q}_3 + K_3|q_3|q_3 \\ L_4\dot{q}_4 + K_4|q_4|q_4 \\ -H_0 \\ h_1 \\ h_4 \\ H_5 \\ L_1\dot{q}_1 + K_1|q_1|q_1 \\ L_2\dot{q}_2 + K_2|q_2|q_2 \\ L_5\dot{q}_5 + K_5|q_5|q_5 \\ L_6\dot{q}_6 + K_6|q_6|q_6 \\ h_4 \end{bmatrix} \cdots\cdots (13)$$

エネルギーの連続条件 $\mathbf{B}\Delta\mathbf{H}^{\mathrm{T}} = 0$ を計算すると次式を得る。

$$B\Delta H^T = \begin{bmatrix} -H_0 + h_1 + L_1\dot{q}_1 + K_1|q_1|q_1 \\ L_4\dot{q}_4 + K_4|q_4|q_4 - h_1 + h_4 + L_2\dot{q}_2 + K_2|q_2|q_2 \\ L_3\dot{q}_3 + K_3|q_3|q_3 - h_1 + H_5 + L_5\dot{q}_5 + K_5|q_5|q_5 \\ L_3\dot{q}_3 + K_3|q_3|q_3 + L_4\dot{q}_4 + K_4|q_4|q_4 - h_1 + h_4 + L_6\dot{q}_6 \\ \quad + K_6|q_6|q_6 - h_4 + h_4 \end{bmatrix} = 0 \cdots\cdots (14)$$

（14）式中の流量の従属変数に（7）式を代入すると，以下のシステム方程式が得られる。

$$\begin{bmatrix} -H_0 + h_1 + L_1\dot{q}_1 + K_1|q_1|q_1 \\ L_4(\dot{q}_2 + \dot{q}_6) + K_4|q_2 + q_6|(q_2 + q_6) - h_1 + h_4 + L_2\dot{q}_2 + K_2|q_2|q_2 \\ L_3(\dot{q}_5 + \dot{q}_6) + K_3|q_5 + q_6|(q_5 + q_6) - h_1 + H_5 + L_5\dot{q}_5 + K_5|q_5|q_5 \\ L_3(\dot{q}_5 + \dot{q}_6) + K_3|q_5 + q_6|(q_5 + q_6) + L_4(\dot{q}_2 + \dot{q}_6) + K_4|q_2 \\ \quad + q_6|(q_2 + q_6) - h_1 + h_4 + L_6\dot{q}_6 + K_6|q_6|q_6 - h_4 + h_4 \end{bmatrix} = 0 \cdots\cdots (15)$$

q_i の時間微分を含んだ項の積を展開すると，次式が得られる。

$$\begin{bmatrix} -H_0 + h_1 + L_1\dot{q}_1 + K_1|q_1|q_1 \\ L_4\dot{q}_2 + L_4\dot{q}_6 + K_4|q_2 + q_6|(q_2 + q_6) - h_1 + h_4 + L_2\dot{q}_2 + K_2|q_2|q_2 \\ L_3\dot{q}_5 + L_3\dot{q}_6 + K_3|q_5 + q_6|(q_5 + q_6) - h_1 + H_5 + L_5\dot{q}_5 + K_5|q_5|q_5 \\ L_3\dot{q}_5 + L_3\dot{q}_6 + K_3|q_5 + q_6|(q_5 + q_6) + L_4\dot{q}_2 + L_4\dot{q}_6 + K_4|q_2 \\ \quad - h_4 + h_4 + q_6|(q_2 + q_6) - h_1 + h_4 + L_6\dot{q}_6 + K_6|q_6|q_6 \end{bmatrix} = 0 \cdots\cdots (16)$$

q_i の時間微分で因数分解した項を左辺に残し，その他の項を右辺に移項して整理すると，次式が得られる。

$$L_1\dot{q}_1 = H_0 - h_1 - K_1|q_1|q_1 \cdots\cdots (17)$$

$$(L_2 + L_4)\dot{q}_2 + L_4\dot{q}_6 = h_1 - h_4 - (K_2 + K_4)|q_2|q_2 - K_4|q_6|(q_6) \cdots\cdots (18)$$

$$(L_3 + L_5)\dot{q}_5 + L_3\dot{q}_6 = h_1 - H_5 - (K_3 + K_5)|q_5|q_5 - K_3|q_6|q_6 \cdots\cdots (19)$$

$$L_4\dot{q}_2 + L_3\dot{q}_5 + (L_3 + L_4 + L_6)\dot{q}_6 = h_1 - h_4 - K_4|q_2|q_2 - K_3|q_5|q_5 - (K_3 + K_4 + K_6)|q_6|q_6 \cdots (20)$$

ここで，流量 q_1，q_2，q_5，および q_6 のある時刻の状態は時間積分して求める必要があるので，状態変数である。表記を簡潔にするために（21）式を定義すると，（17）

～（20）式は（22）式に書き換えられる。

$$\left.\begin{aligned} K_1' &= K_1|q_1| \\ K_2' &= K_2|q_2| \\ K_3' &= K_3|q_3| = K_3|q_5 + q_6| \\ K_4' &= K_4|q_4| = K_4|q_2 + q_6| \\ K_5' &= K_5|q_5| \\ K_6' &= K_6|q_6| \end{aligned}\right\} \quad \cdots\cdots(21)$$

$$\begin{bmatrix} L_1 & 0 & 0 & 0 \\ 0 & L_2+L_4 & 0 & L_4 \\ 0 & 0 & L_3+L_5 & L_3 \\ 0 & L_4 & L_3 & L_3+L_4+L_6 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix} =$$

$$\begin{bmatrix} H_0-h_1 \\ h_1-h_4 \\ h_1-H_5 \\ h_1-h_4 \end{bmatrix} - \begin{bmatrix} K_1' & 0 & 0 & 0 \\ 0 & K_2'+K_4' & 0 & K_4' \\ 0 & 0 & K_3'+K_5' & K_3' \\ 0 & K_4' & K_3' & K_3'+K_4'+K_6' \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_5 \\ q_6 \end{bmatrix} \quad \cdots\cdots(22)$$

ここで，左辺の L_i（i=1，2，5，6）で構成される行列を L 行列と呼ぶことにする。

$$L = \begin{bmatrix} L_1 & 0 & 0 & 0 \\ 0 & L_2+L_4 & 0 & L_4 \\ 0 & 0 & L_3+L_5 & L_3 \\ 0 & L_4 & L_3 & L_3+L_4+L_6 \end{bmatrix} \quad \cdots\cdots(23)$$

L 行列の逆行列 L^{-1} を（22）式の両辺にかけると一般的な形式は次式になる。

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix} = L^{-1} \begin{bmatrix} H_0-h_1 \\ h_1-h_4 \\ h_1-H_5 \\ h_1-h_4 \end{bmatrix} - L^{-1} \begin{bmatrix} K_1' & 0 & 0 & 0 \\ 0 & K_2'+K_4' & 0 & K_4' \\ 0 & 0 & K_3'+K_5' & K_3' \\ 0 & K_4' & K_3' & K_3'+K_4'+K_6' \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_5 \\ q_6 \end{bmatrix} \quad \cdots\cdots(24)$$

h　状態方程式

したがって，導出した（8）式と（24）式をまとめた（23）式は，状態方程式である。

$$\left.\begin{aligned} \begin{bmatrix} \dot{h}_1 \\ \dot{h}_4 \end{bmatrix} &= \begin{bmatrix} \dfrac{1}{A_1} & 0 \\ 0 & \dfrac{1}{A_4} \end{bmatrix} \begin{bmatrix} q_1-q_2-q_5-q_6 \\ q_2+q_6-q_{10} \end{bmatrix} \\ \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix} &= L^{-1} \begin{bmatrix} H_0-h_1 \\ h_1-h_4 \\ h_1-H_5 \\ h_1-h_4 \end{bmatrix} - L^{-1} \begin{bmatrix} K_1' & 0 & 0 & 0 \\ 0 & K_2'+K_4' & 0 & K_4' \\ 0 & 0 & K_3'+K_5' & K_3' \\ 0 & K_4' & K_3' & K_3'+K_4'+K_6' \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_5 \\ q_6 \end{bmatrix} \end{aligned}\right\} \quad \cdots\cdots(25)$$

この状態方程式を時間積分することによって，未知の水頭 h_1，h_4 と流量の独立変数 q_1，q_2，q_5 および q_6 が求まる。これらの流量を（7）式に代入することによって，q_3，q_4，q_7 および q_{11} が求まる。よって，パイプラインシステムの状態を求めることができる。

2　導出過程に必要な数式処理

（25）式は，**Fig.2** のパイプラインシステム例に対して得られた状態方程式であり，他の例を解く場合は新たに状態方程式を導出する必要がある。

1 節の状態方程式を導出する過程において，状態変数は，未知の水頭をもつ自由境界から基準面 D への辺における水頭と補木に属する辺の流量であることが分かる。また，数式処理による式の変形が必要な箇所は，1 節 g のエネルギー連続条件から状態方程式を導出する箇所であり，数式処理として，代入，移項，掛け算の展開，および因数分解が必要であることが分かる。以下，それぞれの数式処理について具体的に説明を行う。

a　代入

流量連続条件から得られた従属変数の関係式（7）式をエネルギー連続条件（14）式に代入して，従属変数を消去する際に必要な処理である。例えば，（14）式中の $K_3|q_3|q_3$ の項に，（7）式中の q_3=q_5+q_6 を代入して $K_3|q_5$+$q_6|(q_5$+$q_6)$ を得る。

b　移項

流量連続条件とエネルギー連続条件から状態方程式を得る際に，状態変数を含んだ項のみを左辺に残し，その他の項は右辺に移項する処理が必要である。例えば，(17) 式中の $-K_1|q_1|q_1$ は（16）式中では左辺にあった項を右辺へ移項した項である。

c　掛け算の展開

流量連続条件から得られた従属変数の関係式（7）式をエネルギー連続条件から得られた（14）式に代入した後に，掛け算の展開を行う。例えば (15) 式中の $L_4(\dot{q}_2+\dot{q}_3)$ は掛け算を展開して，$L_4\dot{q}_2+L_4\dot{q}_3$ になる。

d　因数分解

（15）式に掛け算の展開を行った後に，状態変数の係数をひとまとめにして，（16）式を整理するために，状態変数が同じものがあるか式中の項を検索して，同じ状態変数があった場合は因数分解を行う。例えば，（16）式中では $L_2\dot{q}_2+L_4\dot{q}_2$ は，（18）式中の $(L_2+L_4)\dot{q}_2$ に因数分解される。

Ⅲ　オブジェクト指向プログラミングによる数式処理の実現方法

Ⅱ章で説明したエネルギー連続条件から状態方程式を導出過程における式の変形は，従来，解析する者が手作

業で行うか，もしくは数式処理システムを用いて自動化を行う必要があった。手作業では剛性モデルの数値解析手法における手順が自動化できないため，解析対象となるパイプラインシステムの問題に応じて個別に状態方程式 (25) 式を導出する労力を解析者に強いることになり，その過程において数式の間違いが生じやすい問題がある。数式処理の自動化は便利であるが，数式処理システムは製品が高価であったり，フリーソフトウェアであっても利用者が少ないため，農業土木分野の技術者に普及

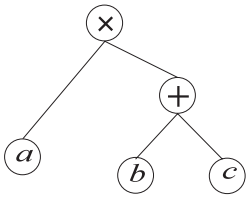

Fig.5 構文木による数式 $a(b+c)$ の表現
Mathematical expression $a(b+c)$ using by the syntax tree

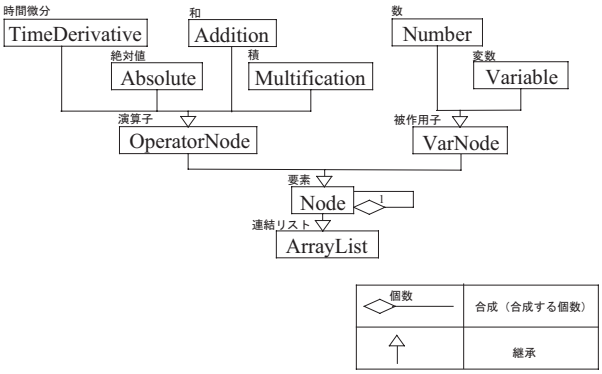

Fig.6 構文木のクラス構造
Class structure of the syntax tree

Table 1 構文木を構成するクラスの主要なフィールドとメソッド
Capital fields and methods defined in classes which constitute the syntax tree

クラス名	フィールドとメソッド	内容
ArrayList	add(Node A)	要素 A の参照を連結リストに追加するメソッド。
	remove(Node A)	要素 A の参照を連結リストから取り除くメソッド。
	size()	要素の数を返すメソッド。
	set(Node A, Node B)	要素 A の参照を要素 B に置き換えるメソッド。
Node	OPERATOR	演算子を指定する整数フラグのフィールド
	VARIABLE	変数を指定する整数フラグのフィールド
	NUMBER	定数を指定する整数フラグのフィールド
	getParent()	親要素を返すメソッド。
	setParent(OperatorNode Parent)	Parent を親要素として設定するメソッド。
OperatorNode	MULT	掛け算の演算子×を指定する整数フラグのフィールド
	ADD	足し算の演算子＋を指定する整数フラグのフィールド
	TDER	時間微分の演算子 d/dt を指定する整数フラグのフィールド
	ABS	絶対値の演算子 abs を指定する整数フラグのフィールド
	getOperator(int A)	指定した演算子 A のインスタンスを作成して返すメソッド。
	replace(Node A, Node B, boolean C)	要素 A に要素 B を代入するメソッド。つまり，要素 A を要素 B で置き換える。C は再帰的に行うかどうかのフラグ。
	levelize()	可換な演算子を平滑化するメソッド。例えば, $a+(b+c)$ を $a+b+c$ とする。
	facrotBy(Node A)	要素 A によって因数分解を行うメソッド。
	expandAll()	すべての子要素の expand() を呼び出して, 掛け算を展開するメソッド。
	eval()	演算子を評価するメソッド。
	equals(Object O)	同じ参照であるかどうか判定するメソッド。
TimeDerivative	expand()	子要素の expand() を呼び出すメソッド。
	eval()	子要素を評価するメソッド。
Addition	expand()	子要素の expand() を呼び出すメソッド。
	eval()	足し算を実行するメソッド。
Absolute	expand()	子要素の expand() を呼び出すメソッド。
	eval()	絶対値を評価するメソッド。
Multification	expand()	掛け算を展開する。例えば，$a(b+c)$ を $ab+ac$ とするメソッド。
	eval()	積の演算を実行するメソッド。
VaNode	setValue()	値を設定するメソッド。
	equals(Object O)	同じ参照であるかどうか判定するメソッド。
Variable	getValue()	関数で定義された値を返すメソッド。
Number	getValue()	数の値を返すメソッド。

していないようである。本章では，より一般的に広く普及しているオブジェクト指向プログラミングに基づいて数式処理を支援するプログラムについて提案する。

1　オブジェクト指向プログラミング

オブジェクト指向プログラミングにおける「オブジェクト」とは，現実世界の「もの」の静的な構造を，計算機内へ比喩を用いて模写した知識表現形式である（たとえば，落水・東田，1998）。知識表現形式を計算機のメモリー上に動的に生成した実体（以下，インスタンスと呼ぶ）は内部記憶を持つ。インスタンス同士は互いの内部記憶を参照することによって状態を共有する。インスタンスに対しメッセージを送信すると，知識表現形式の構造から適切な操作関数を選択し実行する。その副作用として内部記憶が変更され，状態が遷移し処理が進行する。オブジェクト指向の知識表現形式は「クラス」と呼ばれる。クラスの構造は，いくつかの属性変数とその操作関数である。さらに，他クラスを属性変数として内包したり（この機能を「合成」と呼ぶ），他のクラスの属性変数や操作関数をそのままあるいは変更して再利用（この機能を「継承」と呼ぶ）して構成される。また，Java 言語はゴーベッジコレクションと呼ばれるメモリー管理機能が備わっているために，プログラミング初心者が起こしやすいメモリーリークの心配がなく，インターネットから数値計算まで幅広い用途に利用されている。本稿では，プログラミング言語として Java 言語を採用した。

2　数式処理システムのモデル化

簡易な数式処理システムをオブジェクト指向プログラミング言語によって構築する。そのために，一般的な数式処理システムの基本的な仕組みをモデル化する必要がある（猪股・益崎，1994）。例えば，数式処理システム Maxima は関数型プログラミング言語 Common Lisp によって実装されている。Common Lisp における式の記述方法はすべて前置表記である（ポール・グレアム，2002）。前置表記とは，例えば（2 + 3）という足し算の式を（+ 2 3）のように，演算子 + を式の先頭に置き，その後に引数として被作用子を置く表記方法である。Common Lisp のプログラムは式で構成されており，式を実行して値を得ることを「評価する」という。Common Lisp では演算子 + は引数を足しあわせる手続きを行う「関数」としてプログラミング言語処理系にあらかじめ用意されている。評価のルールは，引数が左から右への順に関数に渡され，式全体として値を返す仕組みである。逆に，この評価ルールを停止するための特殊なオペレータ quote も備えている。例えば，（+ 2 3）を評価して値 5 を得るのではなく，記号 +，2 および 3 が並んだリストとして数式を表現するには，特殊オペレータ quote による前置表記の式を作り，（quote（+ 2 3））とする。数式処理システムでは，評価ルールを停止する仕組みを利用して数式を記号が並んだリストのデータとして扱い，数式の値が必要なときに式の評価を行っている。数式処理システムの前置表記の式は，オブジェクト指向プログラミング言語では，構文木のクラスを作成して実現する。構文木の構造はグラフの全域木と同じ考え方で，木のルートには演算子を割り当て，その子には演算子か被作用子を割り当てる。ただし，被作用子は子を持つことが許されない。例えば，$a(b+c)$ という数式は，前置表記による式は（× a(+ b c)）となり，構文木は **Fig.5** のように

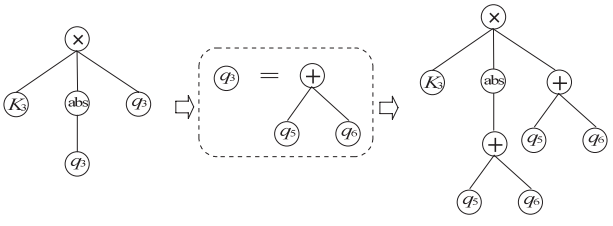

Fig.7　構文木による代入の表現
Expression of substitution using by the syntax tree

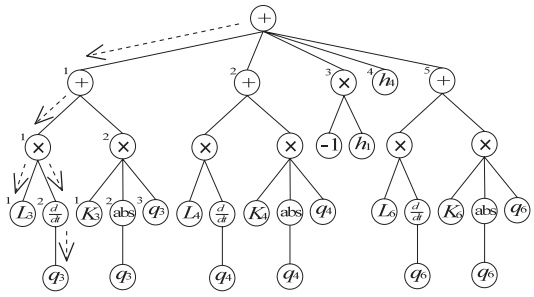

Fig.8　構文木における代入の探索手順
Search procedure of the substitution in the syntax tree

```
public void replace(VarNode key, Node node, boolean recursive) {
    for (int i = 0; i < size(); i++) {                          //子の個数だけ以下の手順を繰り返す。
        if (get(i).getType() == OPERATOR && recursive)          // i 番目の子が演算子であれば、
            ((OperatorNode) get(i)).replace(key, node, recursive); //その子について、replaceメソッドを再帰的に適用する。
        if (get(i).equals(key))                                 // i 番目の子が、探していたインスタンスkeyであれば、
            set(i, node);                                       // i 番目の子インスタンスの参照先をnodeインスタンスへ置き替える。
    }
}
```

Fig.9　メソッド replace における代入の手続き
Procedure of substitution in replace method

表現される。

数式処理をオブジェクト指向で表現するためには，構文木のデータ構造をクラスとして作成するとともに，演算子や被作用子をクラスとして作成する必要がある。

構文木の木構造を表すために，子は 1 つの親を属性変数として持ち，親は複数の子を属性変数として持つ必要がある。構文木を構成する節点を Node クラスとして定義する。1 つの Node クラスのインスタンスを親として持つために自分自身と同じ Node クラスの属性変数を合成する。また，複数の Node クラスのインスタンスを子として持つ。そのために自分自身と同じ Node クラスのインスタンスを複数個格納することができる連結リストの ArrayList クラスを継承する。

構文木を構成する節点は，演算子か被作用子のどちらかであるので, 演算子クラスは OperatorNode クラスとし，被作用子は VarNode クラスとして，Node クラスを継承するようにモデリングした。

剛性モデルに基づく数値解析における状態方程式の導出過程に現れる演算子は，時間微分，絶対値，足し算（引き算を含む），および掛け算のみである。よって，それぞれ TimeDerivative，Absolute，Addition，および Multification クラスとしてモデリングした。これらは演算子 OperatorNode を継承している。

単なる数や変数はそれぞれ Number，Variable クラスとし，被作用子のクラス VarNode を継承するようにモデリングした。変数 Variable クラスは流量境界条件における流量を関数によって定義して与えるために必要なクラスである。

構文木のクラス構造を **Fig.6** に図示する。構文木を構成する各クラスの主な操作関数を **Table 1** に整理する。

① 代入

代入の数式処理を（14）式中の項 $K_3|q_3|q_3$ について説明する。従属変数を独立変数で表した（7）式中の $q_3=q_5+q_6$ を代入すると $K_3|q_5+q_6|(q_5+q_6)$ を得る。その際に，絶対値の演算子を abs とすると，$K_3|q_3|q_3$ の構文木は（× K_3（abs q_3）q_3）である。（7）式 $q_3=q_5+q_6$ は，q_3 のインスタンスをキー，構文木（+ q_5　q_6）のインスタンスを値として格納したマップ構造によって表現する。q_3 に（+ q_5　q_6）を代入する操作は，構文木（× K_3（abs q_3）q_3）における q_3 へ参照先を q_3 のインスタンスから構文木（+ q_5　q_6）のインスタンスへ置き換える操作である（**Fig.7**）。

（14）式中の 4 番目の式の左辺は **Fig.8** に示す構文木で表される。（7）式のマップ構造から q_3 のインスタンスをキーとして取り出し，q_3 のインスタンスが無いかどうか再帰的に探索する。その手順は，構文木のルートである最上階演算子＋の子を左から右へ（番号の若い）順番に取り出す。その子が q_3 のインスタンスであれば，参照先を構文木（+ q_5　q_6）のインスタンスに置き換える。最初に見つかる q_3 のインスタンスは **Fig.8** 中の点線の矢印で示した手順で探索されたものである。仮に，子から q_3 のインスタンスが見つからなければ，その下の子を取り出し，その子に q_3 のインスタンスがあるかを探索し，これがあれば，構文木（+ q_5　q_6）のインスタンスに置き換えることを再帰的に繰り返す。最末端まで繰り返しても，q_3 のインスタンスがなければ，親に戻り，右側の（次の番号の）子について，同様に探索する。最も右側の（番号が最後の）子まで探索したら，終了である。

この代入の操作は，OperatorNode クラスに replace() というメソッドを定義して実現した。**Fig.9** に replace() メソッドの手続きを記した。

② 移項

移項の数式処理を，（16）式中の項 $K_1|q_1|q_1$ を左辺から右辺へ移項する操作に着目して説明する。構文木（× K_1（abs q_1）q_1）を，**Fig.10** に示すように－ 1 の数インスタンスをルートの演算子×の子として追加して構文木（×（-1）K_1（abs q_1）q_1）に変形する操作とする。

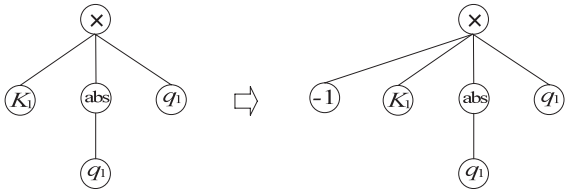

Fig.10 構文木による移項の表現
Expression of transposition using by the syntax tree

移項の手続きのコードを **Fig.11** に記した。（16）式中の流量の時間微分項を除いた全ての項を **Fig.11** のコード中のインスタンス nd として，構文木（×（-1）nd）を作成する手続きである。

③ 掛け算の展開

掛け算の展開は，演算子×のインスタンスの子に演算子＋のインスタンスがあった場合には，新たに作成した演算子＋のインスタンスをルートとして構文木を作り替える操作である。よって，演算子×のインスタンスの子

```
OperatorNode add1 = OperatorNode.getOperator(OperatorNode.ADD);   //右辺の項として演算子＋のインスタンスを作成する。
OperatorNode o = OperatorNode.getOperator(OperatorNode.MULT);     //演算子×のインスタンスを作成する。
o.add(new Number(-1));                                            //×のインスタンスoに-1の数インスタンスを追加する。
o.add(nd);                                                        //右辺へ移項したいインスタンスndを×のインスタンスoに追加する。
add1.add(o);                                                      //×のインスタンスoを演算子＋のインスタンスadd1に追加する。
```

Fig.11 移項の手続き
Procedure of transposition

が演算子＋以外のインスタンスであれば，この変化は起こらない。（15）式中の $L_4(\dot{q}_2+\dot{q}_3)$ を $L_4\dot{q}_2+L_4\dot{q}_3$ に展開することを例に説明すると，構文木（× L_4（+(d/dt q_2)（d/dt q_3)））は，ルートに演算子×があり，その子に演算子＋があるので，構文木を（+（× L_4（d/dt q_2)）（× L_4（d/dt q_3)））に作り替える（**Fig.12**）。

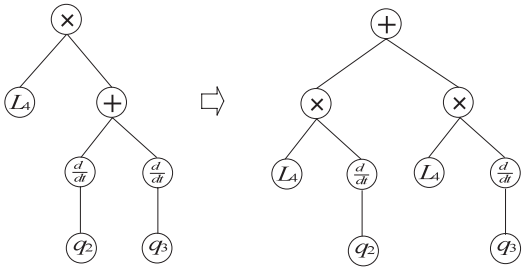

Fig.12　構文木による掛け算の展開の表現
Expression of expanding multiplication using by the syntax tree

コードでは **Fig.13** のように実装される。まず，OperatorNode クラスにおいて定義された expandAll() メソッドでは，すべての子インスタンスに対して expand() メソッドを呼び出す仕組みにする。掛け算の演算子を表す Multiplication クラスでは，expand() メソッドが定義されており，子インスタンスが演算子＋のインスタンスであるならば，新たに作成した演算子＋のインスタンスをルートとして構文木を作り替えている（**Fig.14**）。

ただし，式 $a(b+c)$ を $ab+ac$ と変形するような単純な掛け算の展開しか行わなず，$(a+c)(b+c)$ のような掛け合わせる被作用子が（）で括られているような複雑な掛け算は対象外とした。その他の足し算，絶対値および時間微分を表す演算子クラス Addition クラス，Absolute クラスおよび TimeDerivative クラスでは，単に子インスタンスの expand() メソッドを呼び出すだけである（**Fig.15**）。

④　因数分解

因数分解は，演算子＋のインスタンスをルートとした構文木を演算子×のインスタンスがルートになる構文木に作り替える操作である。(16)式中では $L_2\dot{q}_2+L_4\dot{q}_2$ を（18）式中の $(L_2+L_4)\dot{q}_2$ に因数分解することを例に説明する。**Fig.16** に示すように，構文木（+（× L_2（d/dt q_2)）（× L_4（d/dt q_2)））は，最上層に演算子×のインスタンスを，また，その子に演算子＋のインスタンスを作成して構文木（×(d/dt q_2)(+ L_2　L_4)）に作り替える。

因数分解の手続きのコードを **Fig.17** に記した。引数として渡された要素 pe を項の中に含んでいるかを検索してあれば，新たに演算子＋のインスタンスを作成して，要素 pe を取り除いた項を子の参照として追加する。この項と要素 pe の参照を，新たに作成した演算子×のインスタンスの子と追加すれば，要素 pe による因数分解後の式ができる。

3　エネルギー連続条件から状態方程式を導出する手続きの実装

本章 2 節で説明した種々の数式処理（代入，移項，掛け算の展開および因数分解）を利用すれば，Ⅱ章 1 節 g においてエネルギー連続条件から状態方程式を導出する手続きを行うことができる。その手続きを整理して，フローチャートにすると **Fig.18** になる。そのフローチャートを processEnergyCont() メソッドとして実装したコードが **Fig.19** である。コード中では，（14）式の左辺を配列 energyContEq に格納し，数式処理した後に得られた状態方程式は，その右辺を連結リスト eqFromEnergyContCond に，左辺を連結リスト eqFromEnergyContCondLHS に格納するようにした。マップ nonStateToStateVarMap は，（7）式の流量に関する非状態変数と状態変数の関係式を，非状態変数のインスタンスをキーに，状態変数のインスタンスを値にしたマップ構造である。コード中に出てくる配列 edges はパイプラインシステムをグラフ構造で表現した時の辺をインスタンスとして格納したものである。

Ⅳ　解析例

Ⅲ章において提案した数式処理システムをオブジェクト指向プログラミング言語 Java にて実装した。そのプログラムを利用して **Fig.2** に示したパイプラインシステムについて，状態方程式を導出した例を記す。ここで，**Fig.2** のパイプラインシステムにおける水槽と管路の諸元は **Table 2** の通りとした。ただし，流量 q_{10} は水需要の変動に伴う取り出し流量として，流量が時間 t に依存

```
public void expandAll() {
    boolean doneSomething = true;
    while (doneSomething) {
        doneSomething = false;
        for (int i = size() - 1; i >= 0; i--) {              //子の数だけ以下の手順を繰り返す。
            if (get(i).getType() == OPERATOR)                //子が演算子インスタンスであれば、
                if (((OperatorNode) get(i)).expand())        //演算子インスタンスのexpandメソッドを呼び出す。
                    doneSomething = true;
        }
    }
}
```

Fig.13　OperatorNode クラスのメソッド expandAll における手続き
Procedure of expandAll method in OperatorNode class

```
public boolean expand() {
    boolean bb = false;
    for (int i = 0; i < size(); i++) {                    //子の数だけ以下の手順を繰り返す。
        if (get(i).getType() == OPERATOR)                 //子が演算子インスタンスであれば、
            if (((OperatorNode) get(i)).expand())         //演算子インスタンスのexpandメソッドを呼び出す。
                bb = true;
        for (int j = i + 1; j < size(); j++) {            //隣の子に以下の手順を繰り返す。
            if (get(i) == get(j))
                continue;
            if (get(j).getType() == OPERATOR)             //子が演算子インスタンスであれば、
                if (((OperatorNode) get(j)).expand()) //演算子インスタンスのexpandメソッドを呼び出す。
                    bb = true;
            if (get(j).getType() == OPERATOR
                && ((OperatorNode) get(j)).getOperatorType() == ADD
                && get(i).getType() != OPERATOR
                || get(i).getType() == OPERATOR
                && ((OperatorNode) get(i)).getOperatorType() == ADD
                && get(j).getType() != OPERATOR) {
                 OperatorNode add = OperatorNode.getOperator(OperatorNode.ADD);   //足し算の演算子
                 VarNode v = null;                                                 //インスタンスを作成する。
                 OperatorNode o = null;
                 if (get(j).getType() == OPERATOR) {
                     v = (VarNode) get(i);
                     o = (OperatorNode) get(j);
                 } else {
                     v = (VarNode) get(j);
                     o = (OperatorNode) get(i);
                 }
                 for (Node o0 : o) {          //演算子インスタンスの場合、子の数だけ以下の手順を繰り返す。
                     OperatorNode mul = getOperator(OperatorNode.MULT);   //掛け算の演算子インスタンスを作る。
                     mul.add(v);                                          //参照vを掛け算演算子の子に追加する。
                     mul.add(o0);                                         //参照oの子を掛け算演算子の子に追加する。
                     add.add(mul);                                        //足し算の演算子の子に掛け算の演算子を追加する。
                 }
                 bb = true;
            }
        }
    }
    return bb;
}
```

隣同士の一つが演算子でなくもう片方が足し算の演算子インスタンスであった場合以下の手順を行う。

隣同士のインスタンスの参照をそれぞれ、vとoとして得る。

Fig.14 Multiplication クラスのメソッド expand における手続き
Procedure of expand method in Multiplication class

```
public boolean expand() {
    boolean bb = false;
    for (int i = 0; i < size(); i++) {                    //子の数だけ以下の手順を繰り返す。
        if (get(i).getType() == OPERATOR)                 //子が演算子インスタンスであれば、
            if (((OperatorNode) get(i)).expand())         //演算子インスタンスのexpandメソッドを呼び出す。
                bb = true;
    }
    return bb;
}
```

Fig.15 Addition，Absolute および TimeDerivative クラスのメソッド expand における手続き
Procedure of expandmethod in Addition, Absolute and Multiplication class

する式で与えた。入力データでは，補木と全域木を指定して，**Table 2** の諸元を与えた（**Fig.20**）。入力データの上から 2 行目までは，Ⅱ章 1 節 c の条件に基づいて記述したグラフ構造に関する情報である。1 行目は節点数が 7 個であることを記している。2 行目は線の指定である。一本の辺のデータは始点と終点の番号および補木か木かを指定するフラグの並びで表している。全域木の辺の場合が 1 で，補木の辺の場合が 2 としている。合計 11 本の辺のデータをカンマ（,）で区切って番号順に並べている。それ以降の行は，パイプラインシステム内の水槽と管路の諸元を記したものである。

入力データの最終行から数えて 7 行目から最終行まで

は，流量境界 q_{10} を表すために追加した辺 10 の情報である。流量境界 q_{10} は水需要変動 Q_i を三角関数で表している。例えば，(26) 式のように表すことができる。入力データでは，q_i を 0.5 m^3/s として，この水需要変動 Q_i の式を FunctionImpl0 という名前のクラスに実装していることを表記している。

$$Q_i = q_i\left(1 + 0.5\sin(\omega t)\right) \quad \cdots\cdots (26)$$

プログラムによって出力された結果は，**Fig.21** のとおりである。数式処理されて，流量連続条件から (6) 式と (7) 式を導出し，エネルギー連続条件（14）式から状態方程式（25）式が導出されていることが確認できる。導出された状態方程式のインスタンスは，ルンゲ・クッタ法などの常微分方程式を時間積分を定義したクラスのインスタンスに渡され，演算子クラス OperatorNode のインスタンスが eval() メソッドを呼び出されることによって，数式の評価が行われ，数値解が得られることになる。この例のように，簡易な数式処理システムを実装すれば，グラフ構造の情報と水槽と管路の諸元を入力することによって，状態方程式が自動的に求まる。これら一連の数値解析の作業を行うためには，その他にも，データ入力と出力を実装する必要があるが，紙面の都合上説明しない。

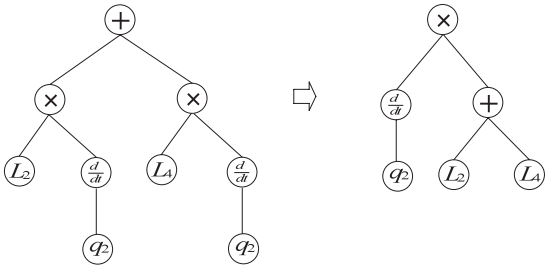

Fig.16　構文木による因数分解の表現
Expression of factorization using by the syntax tree

V　結　言

本稿を整理すると，以下のように要約できる。

①剛性モデルに基づく接続解析では，状態変数の数が状態方程式よりも多く冗長であった。冗長さを取り除くために，補助的な入力データが必要であった。しかし，剛性モデルに基づく閉路解析では，システムグラフを全域木と補木とに分離できれば，状態変数がどれか判明し，状態変数と同じ数だけ状態方程式が得られることを説明した。

②剛性モデルに基づく状態方程式は，パイプラインシステムの接続関係や境界条件の配置によって異なるので，他の問題を解く場合には，新たに状態方程式を導出する必要がある。一般に広く普及しているオブジェクト指向プログラム言語 Java を用いて簡易な数式処理システムを実装して，状態方程式の導出を自動化した。

```
public boolean factorBy(Node pe) {
    boolean doneSomething = false;
    OperatorNode aa = OperatorNode.getOperator(OperatorNode.ADD);          //新たに足し算のインスタンスを作る。
    for (int i = size() - 1; i >= 0; i--) {                                //子の数だけ以下の手順を繰り返す。
        Node p = get(i);                                                   // i 番目の子インスタンスをpとする。
        if (p.getType() == Node.OPERATOR                                   ┐
          && ((OperatorNode) p).getOperatorType() == OperatorNode.MULT     ├ 子pが掛け算の演算子インスタンスであり、
          && ((OperatorNode) p).contains(pe)) {                            ┘ 引数で渡された要素peを含んでいる場合、
            if (((OperatorNode) p).size() == 1)                              以下の手順を繰り返す。
                aa.add(new Number(1));
            else {
                p.remove(pe);                                              //要素peの参照を子pから取り除く。
                aa.add(p);                                                 //足し算インスタンスへ子pの参照を追加する。
            }
            this.remove(p);                                                // i 番目の子pの参照を取り除く。
            doneSomething = true;
        } else if (p.equals(pe)) {                                         //要素peが子pのインスタンスと同じものであれば、
            aa.add(new Number(1));                                         //足し算インスタンスに-1の数インスタンスを追加する。
            this.remove(p);                                                // 自分自身から i 番目の子pの参照を取り除く。
            doneSomething = true;
        }
    }
    if (aa.size() > 0) {
        OperatorNode mult = OperatorNode.getOperator(OperatorNode.MULT);   //新たに掛け算のインスタンスを作る。
        mult.add(pe);                                                      //掛け算インスタンスへ要素peの参照を追加する。
        mult.add(aa);                                                      //掛け算インスタンスへ足し算インスタンスの参照を追加する。
        this.add(mult);                                                    //掛け算インスタンスの参照を自分自身に追加して付け替える。
    }
    return doneSomething;
}
```

Fig.17　OperationNode クラスのメソッド factorBy における手続き
Procedure of factorBy method inOperationNode class

③著者ら（2011）が示した①に対する閉路解決方法と本稿で提案した②の数式処理の手順を一般化することによって，剛性モデルによる閉路解析の数値解析手法を自動化することが可能となった。これによって，弁の完全閉鎖を扱う問題や調整池の設置の必要性を検討する問題などの管路の接続関係を修正する必要がある場合に，新たに閉じたグラフを作成し直すれば，①閉路情報を得る作業と②数式処理を行い状態方程式を導出する作業を自動化することができるため，容易に技術者が解析する可能性が期待できる。

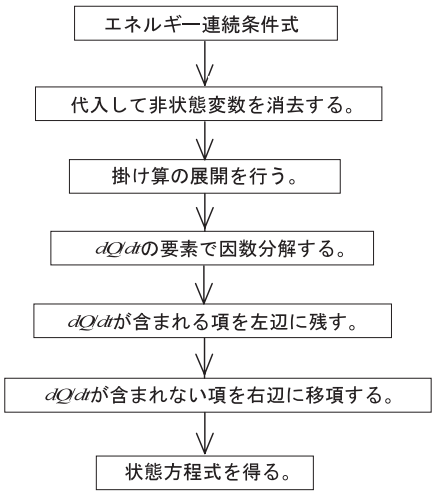

Fig.18　エネルギー連続条件から状態方程式を導出する手続きのフローチャート
Flowchart which deduces state equation from energy condition of continuity

参考文献

1）猪股俊光，益崎真治（1994）：Scheme による記号処理入門，57-68，森北出版，東京
2）Doris R. Ryan and Stephen Chen（1981）：A Comparison of Three Algorithms for Finding Fundamental Cycles in a Directed Graph，Networks，11，1-12
3）内藤克美，小山潤，岩崎和己（1983）：管路系の“ゆるやかな過渡水理現象”解析のための汎用プログラムの開発，農土誌，51，3，215-227
4）落水浩一郎，東田雅宏（1998）：オブジェクトモデリング，新版，4-32，アジソンウェスレイ，東京
5）鬼塚宏太郎（1977）：状態空間解析による枝わかれ管路のサージング減衰特性の評価，土木学会論文報告集，262，79-88
6）鬼塚宏太郎（1981）：農業用水幹線パイプラインの非定常解析に関する研究，東京農工大学農学部学術報告，22，1-54
7）鬼塚宏太郎（1998）：送配水システム解析入門，61-85，技報堂出版，東京
8）ポール・グレアム，（訳）久野雅樹，須賀哲夫（2002）：ANSI Common Lisp，6-10，ピアソン・エデュケーション，東京
9）島田正志（1991）：パイプライン非定常流の剛性モデル・接続行列法における状態変数の決定法，農土論集，156，17-22
10）田中良和，中田達，樽屋啓之（2012）：パイプライン非定常流の剛性モデル・閉路解析における全域木と補木の辺の決定支援方法，農工研技報，212，1-12


```
private void processEnergyCont() {
        eqFromEnergyContCond = new ArrayList<OperatorNode>();
        eqFromEnergyContCondLHS = new ArrayList<OperatorNode>();
        Object[] key = nonStateToStateVarMap.keySet().toArray();
        for (int i = 0; i < energyContEq.length; i++) {
            for (int j = 0; j < key.length; j++)
                energyContEq[i].replace((VarNode) key[j],
                        nonStateToStateVarMap.get(key[j]), true);
            energyContEq[i].levelize();
            energyContEq[i].reduceMul(true);
            energyContEq[i].expandAll();
            energyContEq[i].levelize();
            energyContEq[i].reduceMul(true);
            boolean containsQdot = false;
            for (int j = 0; j < edges.length; j++) {
                if (edges[j].isStateVariable() && edges[j].getQdot() != null) {
                    if (energyContEq[i].factorBy(edges[j].getQdot()))
                        containsQdot = true;
                }
            }
```

状態方程式の右辺と左辺を格納する連結リストの作成

非状態変数に状態変数の関係式を代入する。

掛け算の展開を行う。

dQ/dtの要素で因数分解する。

Fig.19(a)　エネルギー連続条件から状態方程式を導出する手続きのコード（前半）
Code of procedure which deduces state equation from energy condition of continuity (anterior half)

```
                if (containsQdot) {
                    OperatorNode add0 = OperatorNode.getOperator(OperatorNode.ADD);
                    OperatorNode add1 = OperatorNode.getOperator(OperatorNode.ADD);
                    loopk: for (int k = 0; k < energyContEq[i].size(); k++) {
                        Node nd = energyContEq[i].get(k);
                        boolean lhs = false;
                        if (nd.getType() == Node.OPERATOR) {
                            OperatorNode on = (OperatorNode) nd;
                            for (int j = 0; j < edges.length; j++) {
                                boolean contains = edges[j].isStateVariable()
                                        && edges[j].getQdot() != null
                                        && on.contains(true, edges[j].getQdot());
                                if (contains) {
                                    add0.add(nd);
                                    lhs = true;
                                    continue loopk;
                                }
                            }
                        } else {
                            for (int j = 0; j < edges.length; j++) {
                                if (edges[j].isStateVariable()
                                        && edges[j].getQdot() != null
                                        && nd.equals(edges[j].getQdot())) {
                                    add0.add(nd);
                                    lhs = true;
                                    break;
                                }
                            }
                        }
                        if (!lhs) {
                            OperatorNode o = OperatorNode
                                    .getOperator(OperatorNode.MULT);
                            o.add(new Number(-1));
                            o.add(nd);
                            add1.add(o);
                            add1.levelize();
                            add1.reduceMul(true);
                        }
                    }
                    eqFromEnergyContCondLHS.add(add0);
                    eqFromEnergyContCond.add(add1);
                }
            }

        }
```

dQ/dtが含まれる項を左辺に残す。

dQ/dtが含まれない項を右辺に移項する。

状態方程式を得る。

Fig.19(b)　エネルギー連続条件から状態方程式を導出する手続きのコード（後半）
Code of procedure which deduces state equation from energy condition of continuity (last half)

Table 2 パイプラインシステム（**Fig.2**）の水槽と管路の諸元
Dimension of tanks and pipes in pipeline system described in **Fig.2**

管路番号	口径（m）	長さ（m）	摩擦損失係数	水槽番号	水面積（m^2）	定常水位（m）	初期水位（m）
1	1.2	2000	0.0268	0	5000	60	60
2	0.8	1000	0.0226	1	5000	無し	50
3	0.5	1000	0.0221	4	4500	無し	30
4	0.8	4000	0.0226	5	5000	45	45
5	0.5	4000	0.0226				
6	0.5	1000	0.0221				

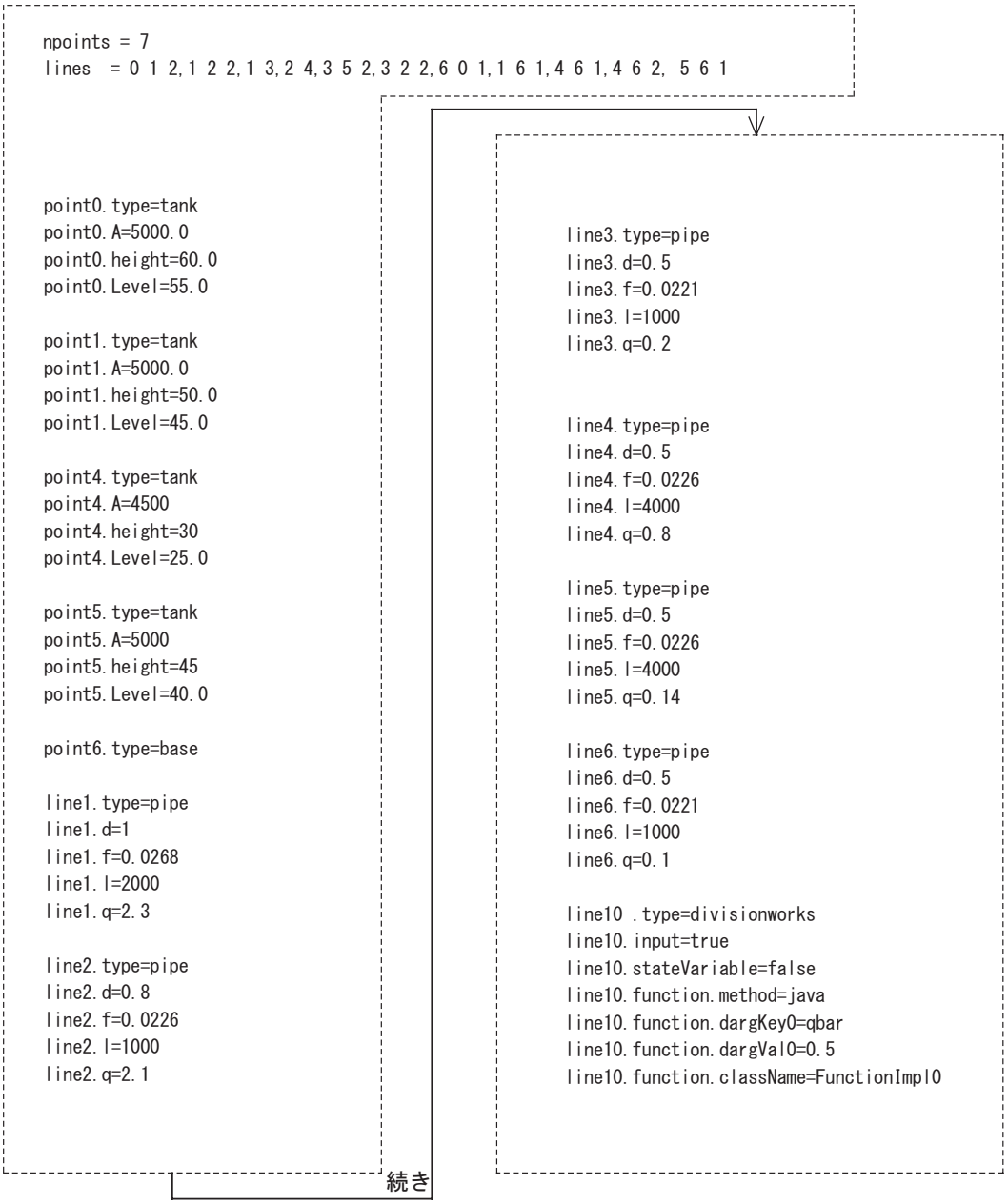

Fig.20　パイプラインシステムの水槽と管路の諸元（**Table 2**）を入力するデータの書式
Format of data which inputed the dimension of tank and pipe line in pipeline system listed at **Table 2**

```
--- 各辺の流量ベクトルと損失水頭ベクトル ---
流量ベクトル 3 : q3
流量ベクトル 4 : q4
流量ベクトル 7 : q7
流量ベクトル 8 : A1 x dH1/dt
流量ベクトル 9 : A4 x dH4/dt
流量ベクトル 11 : A5 x dH5/dt
流量ベクトル 1 : q1
流量ベクトル 2 : q2
流量ベクトル 5 : q5
流量ベクトル 6 : q6
流量ベクトル 10 : q10

損失水頭ベクトル : 3 : (L3 x dq3/dt + K3 x |q3| x q3)
損失水頭ベクトル : 4 : (L4 x dq4/dt + K4 x |q4| x q4)
損失水頭ベクトル : 7 : (0 + -1.0 x H0)
損失水頭ベクトル : 8 : (H1 + -1.0 x 0)
損失水頭ベクトル : 9 : (H4 + -1.0 x 0)
損失水頭ベクトル : 11 : (H5 + -1.0 x 0)
損失水頭ベクトル : 1 : (L1 x dq1/dt + K1 x |q1| x q1)
損失水頭ベクトル : 2 : (L2 x dq2/dt + K2 x |q2| x q2)
損失水頭ベクトル : 5 : (L5 x dq5/dt + K5 x |q5| x q5)
損失水頭ベクトル : 6 : (L6 x dq6/dt + K6 x |q6| x q6)
損失水頭ベクトル : 10 : (H4 + -1.0 x 0)

--- 状態方程式の構築 ---
流量連続条件式 0 : (q3 + -1.0 x q5 + -1.0 x q6) = 0
流量連続条件式 1 : (q4 + -1.0 x q2 + -1.0 x q6) = 0
流量連続条件式 2 : (q7 + -1.0 x q1) = 0
流量連続条件式 3 : (A1 x dH1/dt + -1.0 x q1 + q2 + q5 + q6) = 0
流量連続条件式 4 : (A4 x dH4/dt + -1.0 x q2 + -1.0 x q6 + q10) = 0
流量連続条件式 5 : (A5 x dH5/dt + -1.0 x q5) = 0
    ↓
非状態変数
q7 = 1.0 x q1
q4 = (1.0 x q2 + 1.0 x q6)                                   (7)式
q3 = (1.0 x q5 + 1.0 x q6)
状態変数
A1 x dH1/dt = (1.0 x q1 + -1.0 x q2 + -1.0 x q5 + -1.0 x q6)   (6)式
A4 x dH4/dt = (1.0 x q2 + 1.0 x q6 + -1.0 x q10)

--- A行列---
A1 = 5000.0
A4 = 4500.0

--- A行列の逆行列 ---
A^-1(0) = 2.0E-4
A^-1(1) = 2.2222222222222223E-4

エネルギー連続条件式 0 : ((0 + -1.0 x H0) + (H1 + -1.0 x 0) + (L1 x dq1/dt + K1 x |q1| x q1)) = 0
エネルギー連続条件式 1 : ((L4 x dq4/dt + K4 x |q4| x q4) + -1.0 x (H1 + -1.0 x 0) + (H4 + -1.0 x 0) + (L2 x dq2/dt + K2 x |q2| x q2)) = 0
エネルギー連続条件式 2 : ((L3 x dq3/dt + K3 x |q3| x q3) + -1.0 x (H1 + -1.0 x 0) + (H5 + -1.0 x 0) + (L5 x dq5/dt + K5 x |q5| x q5)) = 0
エネルギー連続条件式 3 : ((L3 x dq3/dt + K3 x |q3| x q3) + (L4 x dq4/dt + K4 x |q4| x q4) + -1.0 x (H1 + -1.0 x 0) + (H4 + -1.0 x 0) + (L6 x dq6/dt + K6 x |q6| x q6)) = 0   (14)式
エネルギー連続条件式 4 : (-1.0 x (H4 + -1.0 x 0) + (H4 + -1.0 x 0)) = 0

--- 非状態変数を状態変数で展開 ---
(dq2/dt x (L4 x 1.0 + L2) + dq6/dt x L4 x 1.0)
 = (-1.0 x K2 x |q2| x q2 + -1.0 x H4 + 1.0 x 0 + -1.0 x (K4 x 1.0 x q2 + K4 x 1.0 x q6) x |(1.0 x q2 + 1.0 x q6)| + 1.0 x H1 + -1.0 x 0)
(dq5/dt x (L3 x 1.0 + L5) + dq6/dt x L3 x 1.0)
 = (-1.0 x K5 x |q5| x q5 + -1.0 x H5 + 1.0 x 0 + -1.0 x (K3 x 1.0 x q5 + K3 x 1.0 x q6) x |(1.0 x q5 + 1.0 x q6)| + 1.0 x H1 + -1.0 x 0)

--- L行列 ---
L(0,0) = (L4 x 1.0 + L2) = 2281.7621943276827
L(0,1) = 0.0 x 0.0 = 0.0
L(0,2) = L4 x 1.0 = 2078.7584403839387
L(1,0) = 0.0 x 0.0 = 0.0
L(1,1) = (L3 x 1.0 + L5) = 2598.4480504799235
L(1,2) = L3 x 1.0 = 519.6896100959847
L(2,0) = L4 x 1.0 = 2078.7584403839387
L(2,1) = L3 x 1.0 = 519.6896100959847
L(2,2) = (L4 x 1.0 + L3 x 1.0 + L6) = 3118.1376605759083

--- 最終的に得られた状態方程式 ---
dH1/dt = (q1 + -1.0 x q2 + -1.0 x q5 + -1.0 x q6) x A^-1(0)
dH4/dt = (q2 + q6 + -1.0 x q10) x A^-1(1)
dq2/dt =
(K2*-1.0*L^-1(0,0) x |q2| x q2 + -1.0*L^-1(0,0) x H4 + -1.0*L^-1(0,0) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q2 + 1.0 x q6)| + L^-1(0,0) x H1 + K5*-1.0*L^-1(0,1) x |q5| x q5 + -1.0*L^-1(0,1) x H5 +
-1.0*L^-1(0,1) x (1.0*K3 x q5 + 1.0*K3 x q6) x |(1.0 x q5 + 1.0 x q6)| + L^-1(0,1) x H1 + K6*-1.0*L^-1(0,2) x |q6| x q6 + -1.0*L^-1(0,2) x H4 + -1.0*L^-1(0,2) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q
2 + 1.0 x q6)| + -1.0*L^-1(0,2) x (K3 x q5 + K3 x q6) x |(q5 + q6)| + L^-1(0,2) x H1)
dq5/dt =
(K2*-1.0*L^-1(1,0) x |q2| x q2 + -1.0*L^-1(1,0) x H4 + -1.0*L^-1(1,0) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q2 + 1.0 x q6)| + L^-1(1,0) x H1 + K5*-1.0*L^-1(1,1) x |q5| x q5 + -1.0*L^-1(1,1) x H5 +
-1.0*L^-1(1,1) x (1.0*K3 x q5 + 1.0*K3 x q6) x |(1.0 x q5 + 1.0 x q6)| + L^-1(1,1) x H1 + K6*-1.0*L^-1(1,2) x |q6| x q6 + -1.0*L^-1(1,2) x H4 + -1.0*L^-1(1,2) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q
2 + 1.0 x q6)| + -1.0*L^-1(1,2) x (K3 x q5 + K3 x q6) x |(q5 + q6)| + L^-1(1,2) x H1)
dq6/dt =
(K2*-1.0*L^-1(2,0) x |q2| x q2 + -1.0*L^-1(2,0) x H4 + -1.0*L^-1(2,0) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q2 + 1.0 x q6)| + L^-1(2,0) x H1 + K5*-1.0*L^-1(2,1) x |q5| x q5 + -1.0*L^-1(2,1) x H5 +
-1.0*L^-1(2,1) x (1.0*K3 x q5 + 1.0*K3 x q6) x |(1.0 x q5 + 1.0 x q6)| + L^-1(2,1) x H1 + K6*-1.0*L^-1(2,2) x |q6| x q6 + -1.0*L^-1(2,2) x H4 + -1.0*L^-1(2,2) x (1.0*K4 x q2 + 1.0*K4 x q6) x |(1.0 x q
2 + 1.0 x q6)| + -1.0*L^-1(2,2) x (K3 x q5 + K3 x q6) x |(q5 + q6)| + L^-1(2,2) x H1)                                   (25)式
```

Fig.21　構築した数式処理システムによる出力結果
Output result by constructed formula manipulation system

Automation Method of State Equations by Object Oriented Programming Using Rigid Water Column Model of Analyzing Slow Transients in Pipelines

TANAKA Yoshikazu, NAKADA Toru and TARUYA Hiroyuki

Summary

The aspect of a water demand changes with agricultural business change in recent years, and the rational water management which coincides with the water-use plan has been required. Numerical fluid analysis technique is an effective technique it examines the plan which carries out improvement of the elastic method of the water management and improvement of facilities. In rigid water column model-closed circuit analysis for slow transients in pipeline, it is advantageous that independent variable and dependent variable of the flow rate are automatically got, if to separate system graph to spanning tree and cotree is possible, and shortly obtains the state equation. However, there was a problem in which to transform those mathematical expressions was very troublesome. Then, the simple formula manipulation system was constructed according to the object oriented programming language. By the benefit of this program, it would be possible to automatically deduce the state equation, and it would be possible that the numerical analysis solution was smoothly obtained. The possibility of spreading through the engineer can be expected, since the difficulty of the mathematical work for utilizing rigid water column model-closed circuit analysis for slow transients in pipeline is removed.

keywords: pipeline system, graph theory, state equation, unsteady flow analysis, formula manipulation, object oriented programming