数理解析研究所講究録
915巻 1995年 157-166

システムのモデリングのための言語 SIMPLE

数理システム　山下浩 (Hiroshi Yamashita)
数理システム　田辺隆人 (Takahito Tanabe)
数理システム　富永純 (Jun Tominaga)
数理システム　逸見宣博 (Nobuhiro Henmi)

1　モデリング言語のもつべき性質

本稿は与えられたシステムを記述して解析するためのモデリング言語SIMPLE の解説を目的とする．
システムの記述のための言語が備えるべき性質として，次のようなものが考えられる．

- 数学的関係が自然な形で記述できる．
- 機能 (behavior) 記述ができる．
- 大規模モデルを簡便に記述できる．
- モジュール化，階層化記述ができる．
- 全体の記述が平易簡便にできる．
- システムの解析法に関してユーザが意識する必要がない．
- 新しい解析プログラム（ソルバ）との結合が簡単に出来る．

上に述べたこと，あるいはシステムの記述自身についてはそれぞれ説明が必要であるが，以下の本言語の機能の解説によって自ずと明らかになることと思われるので，ここでは詳しく述べない．

本言語はシステムの記述をなるべく簡単に行い，実際のシミュレータやソルバ等につなげて所要の解析を行うことを目的とする．利用分野としては

- 連続系のシミュレーション
- 離散系のシミュレーション
- 偏微分方程式の有限要素解析
- システムの最適化
- あるいはこれらの混合システム

などを想定している．このような言語が使用できる事によって，各種の解析が簡便に行われ数学的手法が広く利用されることが期待される．

本システムは，C++の機能（とくに演算子関数）を利用することによって上に述べたような記述を可能にするものである．以下では，主として例題を記述することによって本言語の機能を説明する．

2 小規模な問題

2.1 連立方程式

次のような非線形連立方程式の記述を例にとる.

$$\begin{aligned} x_1 - 2\sin(x_2) &= 0, \\ e^{x_1} x_2 &= 3. \end{aligned}$$

ここで, 未知変数は x_1, x_2である. この方程式を記述するには次のようにする.

```
Variable x1, x2;       // 変数x1, x2の定義
x1 - 2*sin(x2) == 0;   // 第1方程式の記述
    exp(x1)*x2 == 3;   // 第2方程式の記述
```

すなわち, この場合のシステムの記述は変数の定義と式の記述から成る. そして, システムの情報に関してはそれ以上の記述は必要ない.(もちろん, C++の関数として意味あるものにするためには前後に若干の記述が必要である. これについては後で述べる.)

上の例で, 演算子==は等式を意味している. このような連立の非線形方程式を解くには, 通常 Newton 法が使用される. その場合にソルバはまず問題の変数や関数の数を知る必要がある. そして, その後変数の値を何らかの値に設定して, その変数値に対応した関数値や微係数の値を必要とする. これらのソルバに必要な機能は本システムによって提供される. これらの機能はC++の演算子関数の機能と, それらが生成した計算グラフ上での自動微分の手法を利用して提供される.

したがって, 上の例ではこの記述部分が実行されるとSIMPLE は変数が 2 個, 方程式が 2 個あることを認識しそれぞれの関数を計算するためのグラフを作成し, ソルバから問い合わせがあったとき通常の自動微分の手法により所定の量を計算して答えることになる.

システムのみたす法則を記述するには, 方程式以外に不等式も使用出来る. 不等式は演算子<=, >=の両側に式を置くことによって示される. たとえば,

```
x1 <= 2*sin(x2);
```

のように書く事が出来る.

方程式に名前を付けて参照するためには

```
Equation f1, f2;           // 方程式f1, f2の定義
f1 = x1 - 2*sin(x2) == 0;  // 第1方程式をf1に代入
f2 =     exp(x1)*x2 == 3;  // 第2方程式をf2に代入
```

のようにすればよい. プログラムの中で使用された名前は計算結果の出力に使用することができる. Equation に対する=演算子は右辺にある等式全体を左辺のオブジェクトに代入することを意味する. したがって, 以後オブジェクトf1, f2 の内容はそれぞれ代入された方程式を示す.

式の途中結果を保持するためには, Expression というオブジェクトを定義して

```
Variable x1, x2;
Expression e;
e = 2*sin(x2);
  x1 - e  == 0;
exp(x1)*x2 == 3;
```

のようにすればよい．この例では，変数e に式2*sin(x2) が代入され，以後e に別のものが代入されるまではその式の内容を保持する．

また，次のようにパラメータ（定数）を用いることもできる．

```
Variable x1, x2;
Parameter p;
p = 2;
x1 - p*sin(x2) == 0;
     exp(x1)*x2 == 3;
```

Parameter には数字，文字列，Parameter などを代入することが出来る．

Variable 型のオブジェクトに直接にも間接にも依存しない式を定数式という．定数式に関してはその値をチェックする事が出来る．たとえば，

```
Parameter cost(name = "コスト");
check(cost > 0);
```

と書くと，データファイルにコストのデータが存在するときは，それを読み込み（Parameter のコンストラクタの機能），次の式によってその値が正であるかどうかチェックされる．

また，式は自分自身が，定数式であるか，1 次式であるか，2 次式であるか，あるいはそれ以外の一般の非線形式であるかを常に知っている．

以上に述べた

```
Variable  Expression  Parameter  Equation
```

がSIMPLE の基本的なクラスである．大規模モデルを記述するためには，後に述べる集合と要素の概念が必要となるが小規模モデルには上述のクラスで十分に実用的なシステム記述が可能となる．

2.2 数理計画問題

次の線形計画問題を考える．

$$\begin{array}{ll} \text{最小化} & 25x_1 + 30x_2, \\ \text{条　件} & x_1/150 + x_2/200 \leq 50, \\ & 0 \leq x_1 \leq 5000,\ 0 \leq x_2 \leq 4000. \end{array}$$

この問題は以下のように記述される．

```
Variable x1, x2;
minimize(25*x1 + 30*x2);
x1/150 + x2/200 <= 50;
0 <= x1 <= 5000;
0 <= x2 <= 4000;
```

各種の量に名前を付けるときは

```
Variable x1, x2;
Objective cost;
cost = 25*x1 + 30*x2;
minimize(cost);
```

```
Constraint c1, b1, b2;
c1 = x1/150 + x2/200 <= 50;
b1 = 0 <= x1 <= 5000;
b2 = 0 <= x2 <= 4000;
```

のようにする[1]．最大化問題のときは

```
maximize(cost);
```

と書けばよい．

3 大規模モデルのために

3.1 集合とその要素

小規模の問題に対しては，以上のような機能で十分であるが，大規模モデルに対しては実用的ではない．大規模モデルの特徴として，同じ種類のものの繰り返しパターンがあげられる．(そもそも，このような性質のない大規模モデルの取り扱いは絶望的である)．そこで，SIMPLE では「集合とその要素」という概念を導入して，繰り返しパターンを簡便に記述する．

集合S とその要素i の定義は

```
Set S;
Element i(set = S);
```

によってなされる．集合は互いに異なる要素からなる集まりと考える．通常，S の要素の実際の値（文字列あるいは数）は，定義されたときに（もし，データがあれば）データファイルから読み込まれる．プログラムの中で代入操作によって定義する事も可能である：

```
Set Years;
Years = "1990 1991 1992 1993";
```

空集合は""で表わされる．

次の方程式

$$x_i + y_i = 3.0, \quad i \in S$$

は

```
Set S;
Element i(set = S);
Variable x(index = S), y(index = S);
x[i] + y[i] == 3.0;
```

と記述される．Variable x とy の宣言は，それらがS の要素を添え字（インデックス）として持つことを示す[2]．一般にインデックスを含む数学的表式を表現するときに採用される書き方になるべく沿った記述の方法を可能にしたいというのがこの言語の目的の一つである．要素を含む式はその要素の値が特に指定されていなければすべての要素にわたって成立しているものと解釈される．したがって，通常の場合は特別なiterator を使用する必要はない．

[1] Objective はExpression の，Constraint はEquation の別名である．

[2] Variable x(index = i), y(index = i) のようにSet の要素を使用して等価な宣言も出来る．

たとえば，Sとは異なる集合Tに対して

$$x_i + y_i = 3.0, \quad i \in T$$

が成立することを記述するためには

```
x[i] + y[i] == 3.0, i < T;
```

あるいは

```
i < T;
x[i] + y[i] == 3.0;
```

と書けば良い．演算子<は$\in$の意味に使用される．また，演算子>は$\notin$の意味に使用される．上の第一の例では，要素i が集合T に属するという条件はその行にコンマ（,）で結ばれて記述された内容にのみ適用される．第二の例では条件式i < T は別な行に書かれていて，その場合にはこの条件は次にi を含む別な条件式が現れるまで有効となる．コンマで区切って条件を表わす記述法が通常の数式表現に似ていることに注意されたい．

3.2 集合と要素の演算

上で述べたように，SIMPLE では集合やその要素の間の関係を通常の数学的表現で利用する程度に自由に扱える．集合の種類には

```
Set  OrderedSet  CyclicSet  Sequence
```

などがあり，これらの間に色々な演算が用意されている．

集合同士の演算には

和：S | T　積：S & T　差：S - T　直積：S * T

などがあり，集合を含む条件式には

等価関係：S == T　非等価関係：S != T　包含関係：S < T

などがある．

また，添字付き集合を使用することも出来る．これは，集合Sの要素iに対して，集合$T_i, i \in S$が存在するときに使用する．このようなオブジェクトは

```
Set S;      Element i(set = S);
Set T(index = S);  Element j(set = T[i]);
```

によって定義される．最後の定義はT_iの要素$j \in T_i$を定義している．これは，数学的には集合の射像を記述するものであるが，現在のところSIMPLE では集合を要素として持つ集合は定義していないので，単に添字付き集合と考える．

集合が他の集合の部分集合として定義されるときは

```
Set S(subset(T));
```

と定義することによって，常にこの関係がみたされる．(すなわち，S にある要素が加えられると，それは自動的にT に加えられる．

また，具体的な関係による集合の定義，たとえば

$$T = \{i \in S | p_i > 0\}$$

のような定義は

```
Set S; Element i(set = S);
Parameter p(index = S);
Set T = setof(i, p[i] > 0);
```

のように書く事が出来る.

3.3 階層構造

系の中に階層構造（システムの中のサブシステム）が存在する場合を考える. 同じ種類のサブシステムは, ある集合の異なる要素によって表わされるとする. そして, そのサブシステムの記述はサブシステム集合の要素を引数とする関数呼び出しによってなされる.

```
Set S, T;
Element i(set = S), j(set = T);
Variable x(index = S), y(index = T);
...
//サブシステムの記述
sub1(i, x[i]);
sub2(j, y[i]);
```

上の記述で, $i \in S$と$j \in T$というサブシステムが存在することが記述される. そして, サブシステムの側では, たとえば

```
void sub1(DummyElement i, Variable& x) { サブシステムの記述 }
```

のように受ける. Element i をDummyElement i で受けることがサブシステムの記述に入ったことを意味する. もちろん, それ以下の記述は親システムと同様である.

4 機能記述（behavior model）のために

システムの任意の動作を記述するためには, 通常のプログラミング的な機能も必要となる. SIMPLE のクラスオブジェクト以外の変数には通常のC++のプログラムが有効である. SIMPLE のクラスオブジェクトに対しては以下のような記述をする.

1. 集合の要素にわたる iterator：
   この場合はたとえば

   ```
   OrderedSet S; Element i(set = S);
   for(i = S.first(); i < S; i = S.next(i)) { ... }
   ```

   のように, iにSの要素を一つ一つ代入して行く機能を利用する. 同様に, while ループを利用することも出来る.

2. Variable に依存する条件式による分岐：
   この場合は

   ```
   a = ifelse(条件式, 式, 式);
   ```

のように，分岐を処理するための関数を利用し，動的な実行を前もって制御して計算グラフを作成する．たとえば，

```
a[i] = ifelse(x[i] >= 0.0, x[i]*x[i], 0.0);
```

のように使用される．ifelse 関数で一連の手続きを記述するときは，コンマ（,）演算子で連結するか，関数呼び出しを引数に書けば良い．

また，定数式になるような論理式に対しては通常のif 文やifelse 文を使用できる．

5 線形計画問題の例題

大規模問題になりうる例として次のような線形計画問題（diet problem）を考える．

集合 Fは料理の集合（例えば，ハンバーグ，天ぷら等），

集合 Nは栄養素の集合（例えば，ビタミン A，ビタミン C 等），

それぞれの食品の単位量あたりの価格を $c_i, i \in F$，

それぞれの栄養素がそれぞれの食品の単位量に含まれる量を $a_{ji}, j \in N, i \in F$，

それぞれの栄養素の最低摂取量，および最大摂取量を $(N_{min})_j, (N_{max})_j, j \in N$，

それぞれの食品の最低および最大購入量を $(F_{min})_i, (F_{max})_i, i \in F$，

それぞれの食品の購入量（未知数）を $x_i, i \in F$，

とすると，最小化したい目的関数はトータルの購入金額

$$\sum_{i \in F} c_i x_i$$

で，制約条件は次のようになる．栄養素の摂取量に対しては

$$(N_{min})_j \leq \sum_{i \in F} a_{ji} x_i \leq (N_{max})_j, \quad j \in N$$

それぞれの食品の購入量に対しては

$$(F_{min})_i \leq x_i \leq (F_{max})_i, \quad i \in F$$

となる．したがって，問題は

$$\begin{aligned} &\text{最小化} \quad \sum_{i \in F} c_i x_i, \\ &\text{条　件} \quad (N_{min})_j \leq \sum_{i \in F} a_{ji} x_i \leq (N_{max})_j, \quad j \in N, \\ &\qquad\quad (F_{min})_i \leq x_i \leq (F_{max})_i, \quad i \in F \end{aligned}$$

となる．

上の問題は次のように記述される．

```
Set Food, Nutrition(name = "栄養素");
Element i(set = Food), j(set = Nutrition);
Parameter cost(index = Food, name = "単価"); check(cost[i] > 0);
          //パラメータの値に関するチェック
Parameter Fmin(index = Food, name = "最低購入量"),
          Fmax(index = Food, name = "最大購入量"),
          Nmin(index = Nutrition, name = "最低摂取量"),
          Nmax(Nutrition, name = "最大摂取量");
          0 <= Fmin[i] <= Fmax[i];
          0 <= Nmin[j] <= Nmax[j];
Parameter a(index = (Nutrition, Food), name = "栄養");  check(a[j, i] >= 0);
        //a が Nutrition と Food の直積集合上で定義されていることを示す
Variable x(index = Food);
    Fmin[i] <= x[i] <= Fmax[i];     //変数の上下限制約を指定
    x[i] = Fmin[i];                 //変数の初期値を指定
Objective totalCost;
totalCost = sum(cost[i]*x[i], i);      //添え字 i について和を取る
    minimize(totalCost);            //問題は最小化問題であることを指定
Constraint diet(Nutrition);
    diet[j] =
        Nmin[j] <= sum(a[j,i]*x[i], i) <= Nmax[j];
```

上の問題のためのデータファイルの内容は，たとえば次のようになる.(データの値自身に意味はない.)

```
単価 = 天ぷら 500 ハンバーグ 300 焼き魚 250 コロッケ 200;
最低購入量.default = 0;
最低購入量 = コロッケ 200;
最大購入量 = 天ぷら 1000 ハンバーグ 1000 焼き魚 1000 コロッケ 1000;
最低摂取量 = VitaminA 10000 VitaminB 500 VitaminC 10000 Calcium 5000;
最大摂取量 = VitaminA 20000 VitaminB 5000 VitaminC 100000 Calcium 20000;
栄養.default = 0;
栄養 = [VitaminA *] 天ぷら 1000 コロッケ 200
       [VitaminB *] ハンバーグ 200 天ぷら 2000
       [VitaminC *] 天ぷら 1000
       [Calcium  *] ハンバーグ 1000 焼き魚 5000;
```

Set Food の内容は，その集合の要素を添え字としてもつParameter のデータによって間接的に示されることに注意されたい.

この例題で分かるように，モデル記述は一般的規則を記述すればよく，具体的な要素はデータとして与えられる．したがって，特定の目的に対しては 1 度システム記述をすれば，後はデータを更新するだけで色々な場合が解析できる.

また，データとは無関係にモデルを精密化する作業を進める事も可能である.

6 有限差分法の例題

次のポアッソン方程式を2次元空間上の有界な領域Ωで考える.

$$\nabla(\varepsilon(x,y)\nabla\phi(x,y)) = \rho(x,y), \quad (x,y) \in \Omega,$$

ここで, 未知量は電位ϕである. 電流密度$\rho(x,y)$はΩ上で与えられている. また, Ωの境界の電極部では電位ϕの値が与えられている. 上式を有限差分近似によって離散化すると以下のような方程式が得られる.

$$\sum_{j \in Nb(i)} \varepsilon_{ij} \frac{l_{ij}}{z_{ij}} (\phi_j - \phi_i) = \rho_i S_i, \quad i \in P,$$

$$\phi_i = \phi_{ei}, \quad i \in P_e$$

ここで,

PはΩの点の集合,

P_eはΩの境界の電極の点の集合

$Nb(i)$は点iの隣接点の集合,

S_iは点iに関するコントロールヴォリュームの面積,

z_{ij}は辺$i-j$の長さ,

l_{ij}は辺$i-j$に直交するフラックス辺の長さ,

である.

上の問題を記述するためには, 以下のようなプログラムとなる.

```
Set points(name = "points"), electrode(subset(points),name = "electrode");
Element i(set = points);
Parameter x(index = points, name = "x"), y(index = points, name = "y"),
rho(index = points, name = "rho"), phib(index = electrode, name = "phib");

Set neibours(index = points);
Element j(set = neibours[i]);
Parameter area(index = points);
    calArea(i, points, neibours[i], area[i]);
Parameter epsilon(index = (i, neibours[i]));
Parameter z(index = (i, neibours[i]));
    z[i,j] = sqrt(pow((x[i]-x[j]),2)+pow((y[i]-y[j]),2));
Parameter l(index = (i, j));
    calL(i, neibours[i], l[i, j]);
Variable phi(index = points),

//Poisson equation
Equation poisson(index = points);
poisson[i] =
```

```
    sum(epsilon[i,j]*l[i,j]*(phi[i]-phi[j])/z[i,j], j) == rho[i]*area[i],
          i > electrode;
// boundary condition
phi[i] == phib[i], i < electrode;
```

7 モデル記述とその利用

以上のようなC++のプログラムを書くことによって対象とするシステムの記述を行う．それを，どのように利用するかを以下で簡単に説明する．

まず，C++のプログラムの任意の部分でたとえば次のような宣言を行う．

```
System A(&myModel);
```

ここで，Systemというのは SIMPLE のクラスで A がそのインスタンスである．myModel は関数名であり，その中に上述のようなシステムの記述がなされている．以後，myModel において記述されたシステムの内容は A に問い合わせることによってユーザあるいはソルバが知ることが出来る．ソルバはインスタンス A に各種の量（関数値，微分値その他）を問い合わせすることによって，実行を進める．

また，モデルの中で使用されている特定の変数に関して問い合わせをしたいときは，それを呼出側とモデル記述関数の両者に知られている大域変数として定義する．そして，その変数名を引数としてユーティリティ関数を呼び出す．

8 その他の問題

以上の概略の解説では述べなかったことを記しておく．これらの中には将来の問題も含まれる．

- 時間変数のあつかい，時間による微分のあつかい．
- 解法の指定や，シミュレーションの指定のしかた．
- 離散的なシミュレーションの方法の詳細．
- ユーザのカスタマイズの方法．
- グラフィカルなユーザインタフェースの可能性．
- モデル記述からシステムの可解性の判定をしたり，構造を整理（階層化，グループ化，説明変数の選定など）したりする機能．その他感度分析などの機能．