

数理学のためのモデリング言語 SIMPLE

山下浩 田辺隆人 逸見宣博
(株) 数理システム

SIMPLE は数理学のさまざまな分野に現れる数理的モデルを効率的に記述するためのモデリング言語である。本言語は、C++上のクラスライブラリとして実現されている。ユーザはシステムの情報を記述していくだけで、全システムの計算グラフが形成され、自動微分の技法によって、1階微分や2階微分が自動的に計算され、Newton法などのソルバと結合することによって各種計算が簡便に行われる。

SIMPLE - A Modeling Language for Mathematical Science

Hiroshi Yamashita, Takahito Tanabe, Nobuhiro Henmi
Mathematical Systems Institute, Inc.

SIMPLE is an algebraic modeling language for describing mathematical models which appear in various fields. The language is implemented as a class library in C++. Users need simply describe information about their systems. Computational graphs are automatically created by executing the program, and first and second derivatives are computed automatically by using the technique of automatic differentiation. Then the necessary computations are done by solvers as those which use Newton's method that receive necessary information from the system.

1 はじめに

SIMPLE 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。

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) とのリンクが簡単に出来る。

SIMPLE は、これらの性質をみたくモデリング言語の構築を目的として設計された。SIMPLE の利用分野としては

- 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。

SIMPLE 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。

2 SIMPLE の使用法

本節では、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。

2.1 モデルの記述される関数

モデルは適当な名前のファイル (例えば model.cc) に以下のように記述される。

```
void ufun() {  
    . . .  
}
```

ここで、ufun はあらかじめ決められた関数名である。その中に、たとえば以下の例題のような記述がなされる。

2.2 小規模な問題

2.2.1 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SIMPLEで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
```

この場合のシステムの記述は変数の宣言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。¹

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セットして、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。

システムの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、

$$x_1 \leq 2 * \sin(x_2);$$

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。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

¹以下に述べるように、データファイルからデータを読み込むときや、計算結果を出力するときには名前を参照するためには、オブジェクトの宣言の際にVariable x(name = "変数");のように、名前も登録する。

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

のようにする。この例では、変数eに式2*sin(x2)が代入され、以後e1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 = "費用");
check(cost > 0);
```

と書くと、データファイルにcost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。

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である。

2.2.2 数理計画問題

次の線形計画問題を考える。

$$\begin{aligned}\text{最小化} & \quad 25x_1 + 30x_2, \\ \text{条件} & \quad x_1/150 + x_2/200 \leq 50, \\ & \quad 0 \leq x_1 \leq 5000, 0 \leq x_2 \leq 4000.\end{aligned}$$

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。²最大化問題のときは

```
maximize(cost);
```

と書く。

2.3 大規模モデルのために

2.3.1 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が絶望的である)。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、宣言されたときにデータファイルから読み込まれるが、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 Cities = "Tokyo Osaka Kyoto";
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。³一般にインデックスを含む数学的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。

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;
```

と書ける。演算子 $<$ は \in の意味で使用される。⁴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。

2.3.2 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に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
```

²ObjectiveはExpressionの、ConstraintはEquationの別名である。

³Variable $x(\text{index} = i)$, $y(\text{index} = i)$ のようにSetの要素を使用して等価な宣言も出来る。

⁴ $i \notin T$ は $i > T$ と記述される。

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。最後の宣言は 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 では要素を集合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。

2.4 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 = first(S); i < S; i = 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。

2.5 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.

となる.

したがって、問題は

$$\text{最小化 } \sum_{i \in F} c_i x_i,$$

$$\text{条件 } (N_{\min})_j \leq \sum_{i \in F} a_{ji} x_i \leq (N_{\max})_j, \quad j \in N, \quad \text{上の問題はSIMPLEで次のように記述される.}$$

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

```

Set F(name = 'food'), N(name = 'nutrition');
Element i(set = F), j(set = N);
Parameter c(index = F, name = 'cost'); check(cost[i] > 0);
//パラメータの値に関するチェック
Parameter Fmin(index = F), Fmax(index = F),
          Nmin(index = N), Nmax(index = N);
0 <= Fmin[i] <= Fmax[i];
0 <= Nmin[j] <= Nmax[j];
Parameter a(index = (N, F)); check(a[j, i] >= 0);
//a が N と F の直積集合上で定義されていることを示す
Variable x(index = F);
Fmin[i] <= x[i] <= Fmax[i]; //変数の上下制限約を指定
x[i] = Fmin[i]; //変数の初期値を指定
Objective totalCost;
totalCost = sum(c[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(index = N);
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.)

```

cost = 天ぷら 500 ハンバーグ 300 焼き魚 250 コロッケ 200;
Fmin.default = 0;
Fmin = コロッケ 200;
Fmax = 天ぷら 1000 ハンバーグ 1000 焼き魚 1000 コロッケ 1000;
Nmin = VitaminA 10000 VitaminB 500 VitaminC 10000 Calcium 5000;
Nmax = VitaminA 20000 VitaminB 5000 VitaminC 100000 Calcium 20000;
a.default = 0;
a = [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 F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。これらのデータはそれぞれの変数が宣言された時点で代入される。

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。

この例題で分かるように、モデル記述は一般的規則を 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。

ここで、

2.6 有限差分法の例題

次のポアソン方程式を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。

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

P は Ω の点の集合、

P_c 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ボリュームの面積、

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SIMPLEで記述するためには、以下のようなプログラムとなる。

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

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

//Poisson equation
Equation poisson(index = P);
poisson[i] =
  sum(epsilon[i,j]*l[i,j]*(phi[i]-phi[j])/z[i,j], j)
  == rho[i]*area[i], i > Pe;

// boundary condition
phi[i] == phib[i], i < Pe;
```

3 参考文献

- [1] 伊理正夫, 久保田光一, 高速自動微分法 (I), 応用数理, 1, pp17-35, 1991
- [2] 田辺隆人, 山下浩, 逸見宣博, 富永純, モデリング言語SIMPLEにおける集合の概念と大規模システムへの応用, 第24回数値解析シンポジウム講演予稿集, pp115-118, 1995
- [3] (株) 数理システム, SIMPLE0.9 マニュアル, 1995