数式処理 *Bulletin of JSSAC(2012)*
Vol. 18, No. 2, pp. 161 - 171

特集: 数式処理システム 〜いま使うならこれだ〜

気軽に使える本格的数式処理システム Sage

竹本 浩*

有限会社プライマリウェイヴ

1 はじめに

Sage は，Mathematica のような数式処理を行うオープンソースのソフトウェアです．Sage の歴史はまだ浅く，ウィリアム・スタイン氏 (William Stein) によって 2005 年 2 月に開発がスタートし，2006 年 2 月の UCSD Sage Days 1 で Sage 1.0 が公開され，最新のバージョンは 4.7.2 です[1)]．Sage の特徴を挙げると，次のようなものが挙げられます．

- ブラウザーで使える
- 柔軟な図化機能
- Maxima, R 等の既存ツールとの連携

2 ブラウザーで気軽に使える

Sage の最大の特徴は，Firefox や Internet Explorer 等のブラウザーから Sage Notebook Server にアクセスして，気軽に数式処理を実行することが出来ることです[2)]．ノートブックは，Sage での一連の計算を記録したノートであり，計算に関する説明文を挿入したり，値を変更して再計算することができます．

2.1 ノートブックの作成

Sage のノートブックを体験するには，Sage の開発サイトでアカウントを作成し，ワークシートを作成するのが最も簡単な方法です[3)]．開発サイトへのログインが完了すると図 1 のようなノートブック画面になります．

2.2 ワークシートの作成とセルの操作

ノートブック画面で New Worksheet をクリックすると新しいワークシートが作成されます．ワークシートで式を評価するには，セルと呼ばれるテキストエリアを利用します．セルの基本操

*take@pwv.co.jp

1) William Stein の 2007 年 UW CSE コロキアムの資料から引用.

2) Linux, Mac OSX, Windows にダウンロードして使うこともできます.

3) Sage 公開サーバの URL：http://www.sagenb.org/,
筆者のサーバにも公開サーバ（http://www.pwv.co.jp:8000/）を置いています.

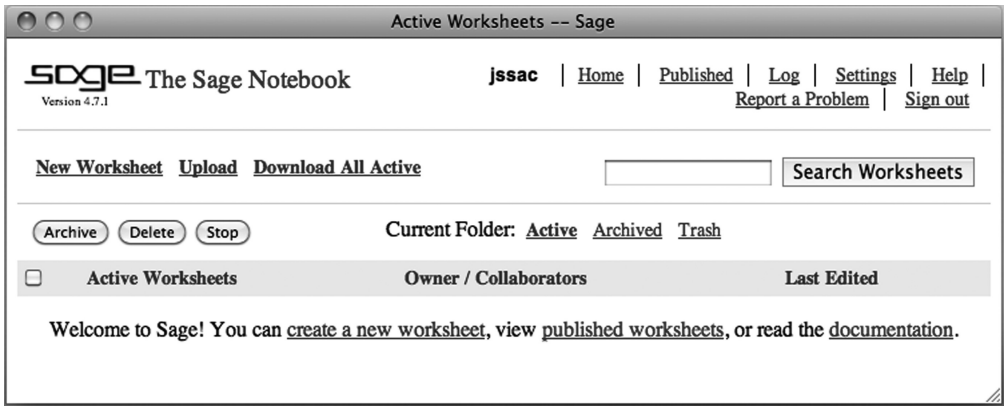

図 1: 初期ログイン時のノートブック画面

作は，以下のように行います．

セルの評価

セルに記述した式を評価するには，シフトキーとリターンキーを同時に押す（Shift-Return と記す）方法と evaluate をクリックする方法があります．

セルの追加

セルの上下にマウスを移動すると青い帯が表示されます．この青い帯をクリックするとセルが追加されます．

セルの削除

セル内のテキストをすべて削除し，もう一度バックスペースキーを押すとセルが削除されます．

2.3 ヘルプ

Sage の関数の使い方は，関数名のあとに?を付けて Shift-Return でヘルプ画面が表示されます．例として，abs 関数のヘルプを表示させるには，次のように入力します（結果は，図 2 参照）．

```
abs?
```

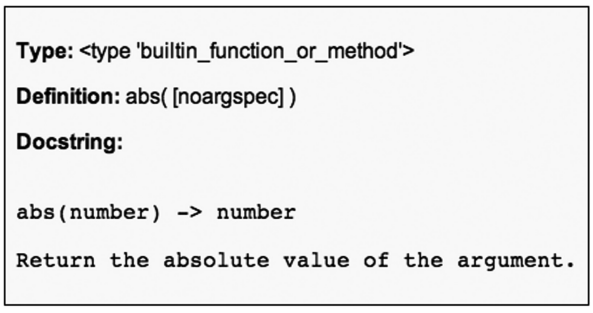

図 2: abs 関数のヘルプ画面

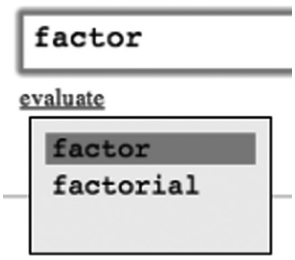

図 3: fac に対する補完結果

2.4 補完機能

関数名や Python の変数名は，タブキーで補完することができます．例として，`fac` を入力した後にタブキーを押すと，自動的に `factor` まで入力が補完され，`factor` と `factorial` の 2 つの候補が表示されています（図 3）．

2.5 説明文の挿入

ワークシートには，説明文や図，数式を入力するためのツールが提供されています．セルの上下にマウスを移動すると青い帯が表示されますので，シフトキーを押しながらクリックすると図 4 のようなエディタ画面が表示されます．説明文には，LaTeX 形式で数式が書けるので，計算手法の解説などが簡単に挿入でき，ワークシートの活用に役立ちます．

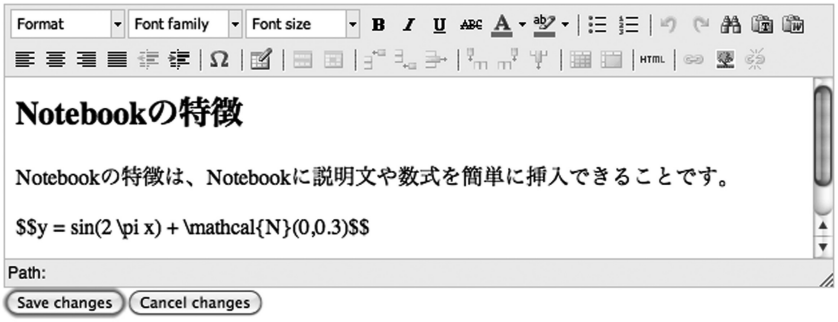

図 4: エディタ画面の編集例

2.6 Python 文法

Sage では，Python のインタプリタを使用しているため，Python の文法でコマンドを記述します．Python ではシャープ (#) 以下がコメント文となり，セミコロン (;) で複数の式を 1 行にまとめて記述することができます．

Sage 固有の使い方としては，変数への代入結果を表示するためにセミコロンの後に変数名を追加します（図 5 の第 1 セル）． Sage は数式処理システムですので，結果は数値ではなく数式を評価した値となります（図 5 の第 2 セル）．数値として表示するには，`N` 関数を使用します（図 5 の第 3 セル）．

```
# 代入の結果は値を返さないので、;で区切って変数名を書いて変数の値を表示します。
a = 1; a
```

```
1
```

```
a = 1/2 + 1/3; a
```

```
5/6
```

```
N(a)
```

```
0.833333333333333
```

図 5: Sage 固有の使い方

2.7 変数と関数の定義

Sage の数式において，文字を変数として認識させるには，変数を var 関数で宣言しなくてはなりません．変数の宣言は var の引数に文字列で使用する変数名をスペースで区切って行います（図 6 の第 1 セル）．また，Sage で関数を定義するには，以下の 3 つの方法があります．

シンボリック関数：	f(x) =式の形式で定義します．
Python 関数：	Python の def を使って関数を定義します．
lambda 関数：	Python の lambda 式を使って関数を定義します．

例として，3 乗根をシンボリック関数，Python 関数，lamba 関数で定義し，その結果を表示してみます（図 6 の第 3 セル以降）．

```
# 変数yの定義
u, v = var('u v')
```

```
type(u)
```

```
    <type 'sage.symbolic.expression.Expression'>
```

```
# 関数3乗根の定義
# シンボリック関数の例
symCubicRoot(x) = x^(1/3)
# Python関数の例
def pyCubicRoot(x):
    return x^(1/3)
# lambda関数の例
lamCubicRoot = lambda x : x^(1/3)
# 関数の呼び出しと結果
print symCubicRoot(2), pyCubicRoot(2), lamCubicRoot(2)
```

```
    2^(1/3) 2^(1/3) 2^(1/3)
```

図 6: Sage 変数と関数の定義例

2.8 よく使われる定数

Sage でよく使われる定数を表 1 に示します．

	円周率	自然対数の底	虚数単位	無限大
慣用的な表現	π	e	i	∞
Sage への入力	pi	e	I (大文字の i)	oo (o が 2 つ)

表 1: Sage でよく使われる定数

2.9 グラフの重ね書きも簡単

Sage のとても便利な機能にグラフの重ね書きがあります．例として，Sin 曲線 + ノイズのデータと元の Sin 曲線（正弦関数）を重ね書きする処理を図 7（第 1 セル）に示します．行っている処理は次の通りです．

1. Sin 曲線のプロット結果を変数 `sin_plt` に代入
2. データのリストプロット結果を変数 `data_plt` に代入
3. `sin_plt` と `data_plt` の和に対して，`show` メッセージを送って重ね書きを実行

```
# グラフの重ね書き
X = [random() for i in range(100)]
t = [(sin(2*pi*x) + gauss(0, 0.3)).n() for x in X]
# グラフ変数に使用するxを定義
var('x')
sin_plt = plot(sin(2*pi*x),[x, 0, 1], rgbcolor='green')
data_plt = list_plot(zip(X, t))
(data_plt + sin_plt).show()
```

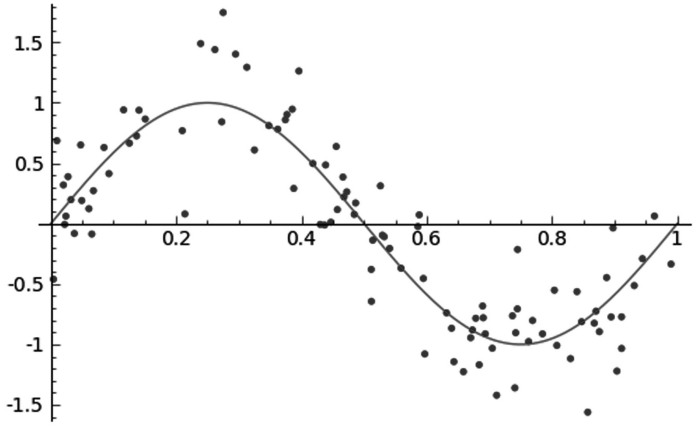

図 7: Sin 曲線 + ノイズのデータ（100 個）と元の Sin 曲線の重ね書き例

2.10 インタラクティブな処理

様々な値に対する結果をインタラクティブに提供するために，`@interact` コマンドが用意されており，`@interact` に続いて関数定義 `def_(引数)` を記述することで利用可能です．引数には，次のようなものの中からユーザが指定することができます．

- 引数に `slider(リスト)` またはリストを代入するとスライドバー
- 引数に `selector(リスト)` を代入するとプルダウンメニュー
- 引数に値を代入するとテキストフィールド

図 8 に Sin 曲線のフィッティングにおいて，多項式の次数 M をスライドバーの操作で 0 から 9 まで選択可能にした例を示します（選択可能な値をリストで指定しています）．

2.11 サンプルワークシートのダウンロードと再利用の方法

本稿で紹介したワークシートや参考文献 [1] で使用したワークシート[4)]は，自由にダウンロードできます．

[4)] http://www.pwv.co.jp:8000/home/pub/16/ にワークシートを公開しています．

```
# PRMLの図1.3
@interact
def _(M= slider(range(10))):
    # 計画行列Φ
    Phi = matrix([[ _phi(x,j) for j in range(0, (M+1))] for x in X.list()])
    Phi_t = Phi.transpose()
    # ムーア__ペンローズの疑似逆行列
    Phi_dag = (Phi_t * Phi).inverse() * Phi_t;
    # 平均の重み
    Wml = Phi_dag * t
    f = lambda x : sum(Wml[i]*x^i for i in range(0, (M+1)))
    y_plt = plot(f, [x, 0, 1],  rgbcolor='red')
    (y_plt + data_plt + sin_plt).show(ymin=-1.5, ymax=1.5)
```

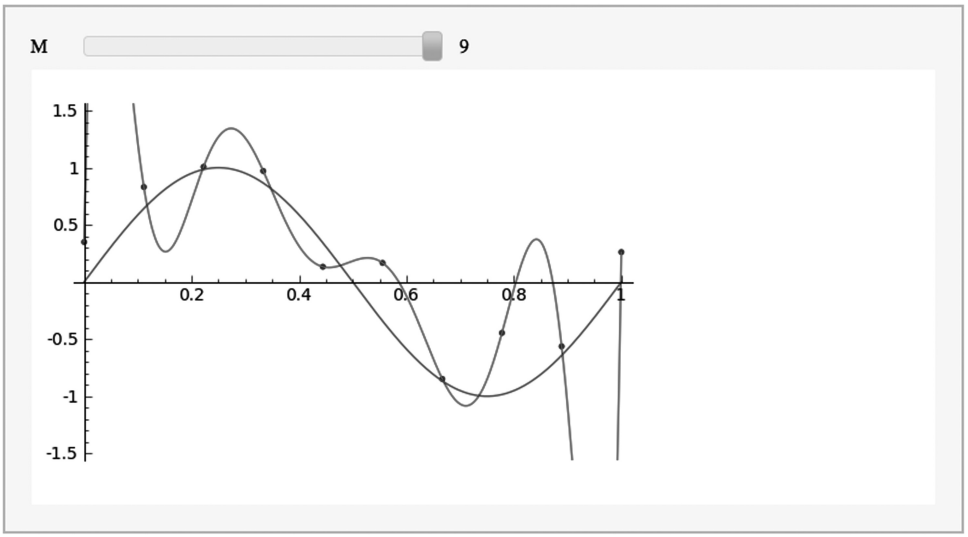

図 8: Sin 曲線のフィッティング問題で多項式の次数 M をスライドバーで与えた例

各公開ワークシートを表示し，先頭にある Download を押すと，ダウンロードが開始します．ダウンロードしたワークシートは，ユーザのノートブックで Upload をクリックし，ダウンロードしたファイルを指定することで，簡単にアップロードできます．自分のノートブックでワークシートの計算を再現したり，値を変えて評価することができますので，是非試してみて下さい．

3 数式処理システムらしい解法

共役勾配法を例に Sage の数式処理システムらしい解法を紹介します．Sage の数式処理機能と Python の記述力を合わせると，とてもスマートに共役勾配法で 2 次関数の極値を求めることができます．

例として，以下のような 2 次形式の関数を考えます．

$$f(x) = \frac{3}{2}x_1^2 + x_1x_2 + x_2^2 - 6x_1 - 7x_2 \tag{1}$$

初期値 x_0 が反復計算により極値に達するためには，勾配 ∇f からある程度接線方向 t にずれた

共役勾配 d_n 方向に進む必要があります．ここでは，d の初期値を $d_0 = -\nabla f(x_0)$ とします．

$$d_n = -\nabla f(x_n) + \beta_n d_{n-1}, \quad \beta_n = \frac{(\nabla f(x_n))^T \nabla f(x_n)}{(\nabla f(x_{n-1}))^T \nabla f(x_{n-1})} \tag{2}$$

x_{n+1} は刻み値 α_n による次の反復公式で更新されます．ここで H は，$f(x)$ のヘッセ行列です．

$$x_{n+1} = x_n + \alpha_n d_n, \quad \alpha_n = -\frac{d_n^T \nabla f(x_n)}{d_n^T H d_n} \tag{3}$$

3.1 Sage での実装

このように定義される共役勾配法を，Sage を使って実装していきます．ベクトル `v` を変数 `x1`, `x2` で定義し，ベクトル `v` 使って関数 `f` を定義します．

```
# 変数定義
vars = var('x1 x2')
v = vector([x1, x2])
```

```
# f を定義
def f(v):
  return 3/2 * v[0]^2 + v[0]*v[1] + v[1]^2 - 6*v[0] - 7*v[1]
```

次に，∇f を計算する関数 `nabla_f` を定義します．ここでは，あらかじめ関数 `f` の各変数で偏微分した結果を `dfs` に保存した上で，その結果を利用して引数ベクトル `vx` の値を代入した結果を返すように定義しています．

```
# f を偏微分したリスト
dfs = [diff(f(v), x_i) for x_i in v]
# ▽ f を定義　(dfs に vx の要素の値を適応した結果を返す)
def nabla_f(vx):
  # ベクトル vx の各要素の値を v の要素に対応づける
  s = dict(zip(v, vx))
  # ベクトルの各要素の偏微分の結果に s を適応させる
  return vector([df.subs(s) for df in dfs])
```

ヘッセ行列も Sage の数式機能と Python のリスト内包を使えば, 簡単に求めることができます．最初に `diff` コマンドで `f(v)` を変数 `x_i`, `x_j` でそれぞれ偏微分した 2 重リストを作成し，`matrix` 関数で行列型に変換します．ここでは，変換結果をきれいな行列形式で出力して確認するために，`jsmath(H)` で画面に表示しています．

```
# ヘッセ行列
H = matrix([[diff(diff(f(v),x_i), x_j) for x_i in v] for x_j in v])
print jsmath(H)
```

$$\begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$$

α_n の定義も式 3 の通りです．

```
# α_n の定義
def alpha_n(x, d):
  return -d.dot_product(nabla_f(x)) / (d * H * d)
```

すべての準備が整ったので，共役勾配法を使って極値を計算してみます．

```
eps = 0.001
x0 = vector([2, 1])
d = - nabla_f(vx=x0)
x = x0
k = 1
while (true):
  o_nabla_f_sqr = nabla_f(x).dot_product(nabla_f(x))
  o_x = x
  x += alpha_n(x, d)*d
  if ((x - o_x).norm() < eps):
    break
  beta = nabla_f(x).dot_product(nabla_f(x)) / o_nabla_f_sqr
  d = -nabla_f(x) + beta*d
  if (d.norm() == 0): # 0 除算への対応
    break
  k += 1
print "x=", x
print "k=", k
```

結果は，次のようになります．

```
x= (1, 3)
k= 2
```

比較のため，Sage の最適化機能で同じ問題を解かせますと，これらの結果は一致します．

```
# 同様の処理を Sage の機能を使って計算してみる
g = 3/2*x1^2 + x1*x2 + x2^2 - 6*x1 -7*x2
minimize(g, [2, 1], algorithm="cg")
    Optimization terminated successfully.
     Current function value: -13.500000
     Iterations: 2
     Function evaluations: 5
     Gradient evaluations: 5
(1.0, 3.0)
```

求まった解をプロットすると，図 9 のようになります．

```
# 関数と解をプロット
p3d = plot3d(g, [x1, -1, 4], [x2, -1, 4])
pt = point([1, 3, f(x)], color='red')
(p3d+pt).show()
```

このように数式処理システムの特徴を活かすように Sage でプログラムを記述すると，数式での表現と実際の計算手続きがきれいに対応するように記述することが可能です．

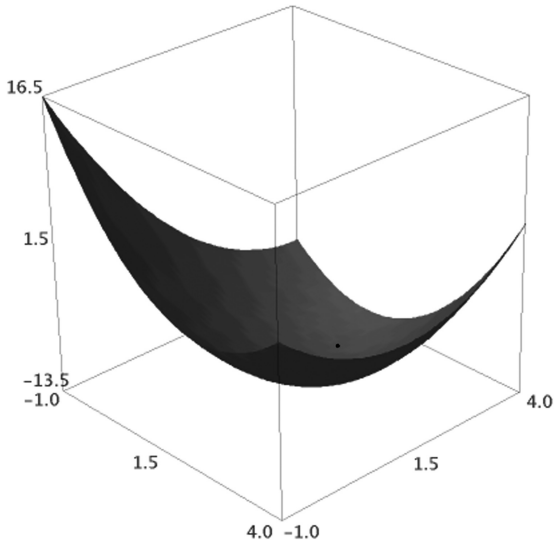

図 9: 関数 f と求めた解をプロットした図

4 既存ツールとの連携

Sage ではすべての処理を Sage 単独で行うのではなく，既存のツールと連携するためのインタフェースを提供しています．例として，R での主成分分析の結果を Sage に渡す方法を紹介します[5)]．

Oil Flow のデータを使って主成分分析をします．データは，あらかじめノートブックにアップロードしておきます．ここでは，データがファイル `DataTrn.txt` に，ラベルがファイル `DataTrnLbls.txt` に保存されていると仮定しています．なお，`DATA` は，Sage においてアップロードされたフォルダを表す定数です．

R で主成分分析を計算させるには，R とのインタフェース関数 `r` を使って R のコマンドを記述します．以下のコードは，データを読み込み，主成分分析を `result` 変数に代入します．

[5)] http://www.pwv.co.jp:8000/home/pub/14/ にワークシートを公開しています．

```
# R でデータを読み込み PCA を計算
fileName = DATA + 'DataTrn.txt'
oilflow = r("oilflow <- read.table('%s')" %fileName)
result = r("result <- prcomp(oilflow)")
```

R のグラフを Sage で表示するには，一度ファイルに出力する必要があります（図 10 参照）．出力された画像ファイルを表示するために，html 関数で pca.pdf を表示する HTML を記述しています．

```
#  ラベルの読み込み
fileName = DATA + 'DataTrnLbls.txt'
labels = r("oilflow.labels <- read.table('%s')" %fileName)
# プロットファイル名の設定
filename = DATA+'pca.pdf'
r.pdf(file='"%s"' %filename)
# 結果のプロット
r("col <- colSums(t(oilflow.labels) * c(4,3,2))")
r("pch <- colSums(t(oilflow.labels) * c(3,1,4))")
r("plot(result$x[,1:2], col=col, pch=pch, xlim=c(-3,3), ylim=c(-3,3))")
r.dev_off()
# 式を変えたときにはブラウザーで再読込が必要
html('<img src="pca.pdf">')
```

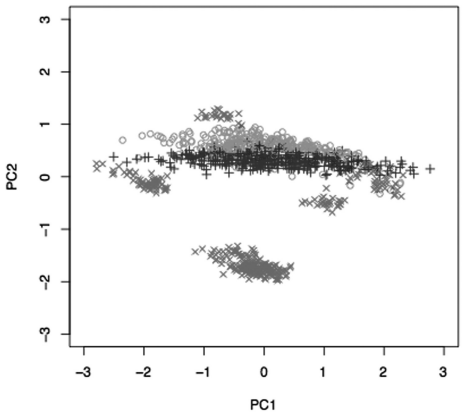

図 10: 主成分分析の結果を Sage で表示

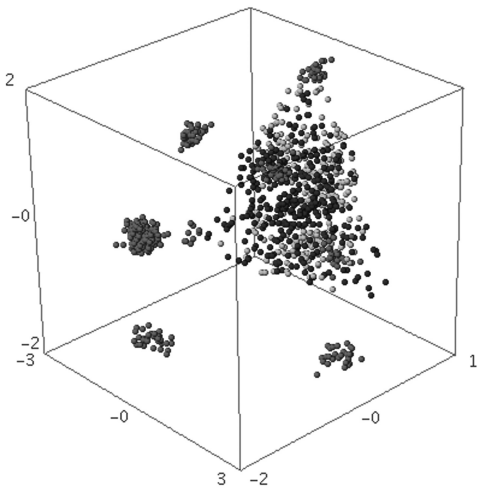

図 11: R で読み込んだデータを Sage で表示

4.1 R のデータを Sage に渡す

R のデータを Sage に渡すために Sageobj 関数と_save_メソッドが提供されています．例として，R で読み込んだデータセットを Sage の 3 次元プロットを使ってプロットしてみます（結果は，図 11 参照）．

```
#R のデータセットを Sage の形式に変換すると
#'DATA' ディクショナリに列単位で V1, V2 のようにセットされる
lb = Sageobj(labels)
# これを zip でまとめて使う
lbs = zip(lb['DATA']['V1'],lb['DATA']['V2'],lb['DATA']['V3'])
# point3d でプロット
N = len(lbs)
plt = Graphics()
for n in range(N):
  [x, y, z] = rs[n]
  if lbs[n][0] == 1:
    plt += point3d([x, y, z], rgbcolor='blue')
  elif lbs[n][1] == 1:
    plt += point3d([x, y, z], rgbcolor='green')
  else:
    plt += point3d([x, y, z], rgbcolor='red')
plt.show()
```

5 Sage の今後

このように柔軟で便利な機能をもつ Sage ですが，日本国内ではまだまだ普及していません．本稿を通じて，一人でも多くの方に Sage を使って頂ければ幸いです．今後のユーザ会などの支援団体の創設や解説書の出版が待たれます．沼田泰英氏が Sage のレファレンスカードを日本語に翻訳して公開されています[6]．印刷して傍らに置いておくと便利です．

最後に，本稿の査読にご協力頂きました (株) 構造計画研究所の鈴木 由宇氏，本稿の LaTeX への変換にご協力頂きました神戸大学の長坂 耕作准教授に感謝申し上げます．

参 考 文 献

[1] 竹本浩: OSS 数式処理システム「Sage」，Software Design，2010 年 6 月号，pp. 89–94.

[6] http://www.stat.t.u-tokyo.ac.jp/~numata/nora/sage-doc/