容易になった非線形最適化：
NAGルーチンと併せてAMPLモデリング言語を使用するためのチュートリアル

Jan Fiala*
The Numerical Algorithm Group, Ltd.
February 9, 2011

概要

最適化、あるいは一般的に言うオペレーションズ・リサーチは今日では私たちの生活において重要な役割を担っています。評判の高い金融会社であっても、あるいは数学を学ぶ学生であっても、おそらく何らかの最適化ルーチンを使用したことがあります。1940年代半ばに線形計画法が取り入れられてから最適化の分野は急速に変化しました。より強力なコンピュータにより、さらに現実的で複雑なモデルを高性能のアルゴリズムを使用して考えることができるようになりました。線形計画問題への入力が相対的に簡単であるのに対し、一般の非線形計画の場合問題への入力は細心の注意を要する作業です。それに対処する一つの方法は、問題の記述のために専門の言語を導入することです。このチュートリアルでは、E04UFF とE04UGF という名前の二つのNAGソルバーを備えたAMPLと呼ばれる特殊な言語に重点を置いています。

AMPL（A Mathematical Programming Language：数理計画言語 [3, 8] ）はパッケージ化されたいくつかの優れた機能を持っており、かなりの好評を得ています。この言語は共通の数学表記を使用しているため、理解しやすいです。さらに自動微分パッケージ（正確な導関数を計算する手法）を含んでおり、そのため導関数のコーディングを完全に回避することができます。これは問題を速く解決したいときには最適です。デモや教育に最適であり、また数学のモデルの迅速なプロトタイピングにも最適です。従って、コーディングではなく最適化そのものに集中することができます。また、問題の設定とソルバーの設定両方のインターフェースを統一しており、いくつかの問題でユーザのソルバーをテストできます。あるいは同じ問題をいくつかのソルバーを使ってそれほど苦労せずに解くことができます。NAG のよくチューニングされたソルバーと併せてモデリング言語の機能を提供しているので、AMPL とNAG ソフトウェアの接続は自然であるように思われます。

このチュートリアルは以下のようにまとめられています。初めの２つのセクションではAMPLとAMPL言語で書かれた簡単な例を紹介しています。 ３つ目のセクションでは計算用のプラットフォームをどのように準備したらいいか専門的な内容に重点を置いています。ソルバーの呼び出しや任意パラメータの設定はセクション４とセクション５で述べられています。次のセクションで

---

*jan@nag.co.uk

はセクション７で使用されるAMPL言語のより拡張した要素について述べています。セクション７では架空の問題とそのモデルの開発が示されています。残りの章では、AMPLソルバーの構築についての概要と結びの言葉を述べています。
注： 本稿で述べられている全てのファイルは NAG Web サイト [14] からダウンロードできますので、実際に取得してご自分でテストいただけます。

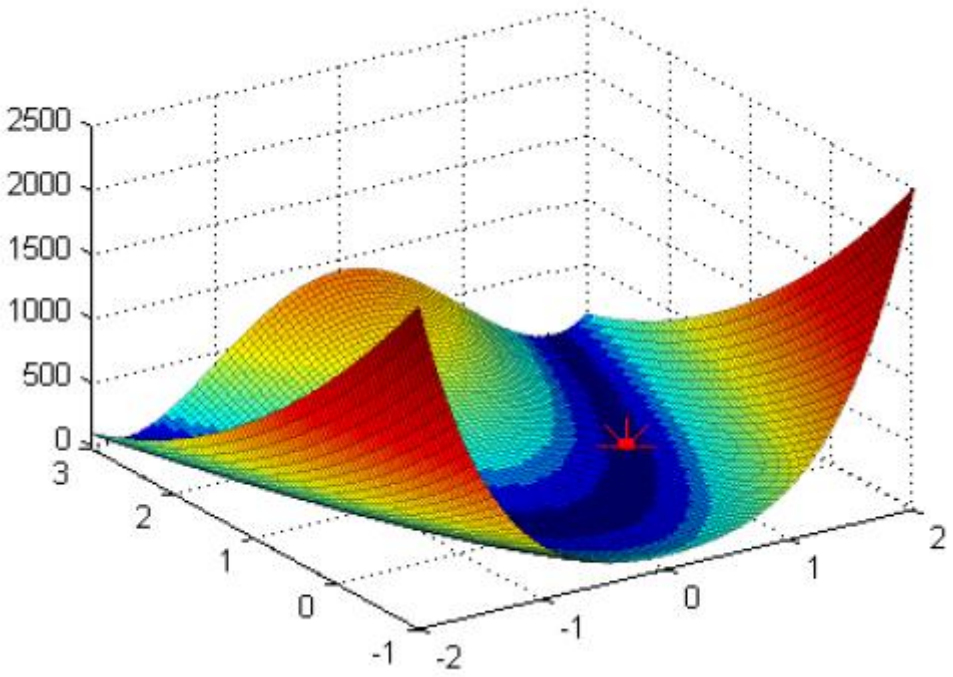
1 AMPLの紹介

前述のとおり、AMPLはモデリング言語、すなわち様々な最適化問題を記述できる特別なコンピュータ言語です。２，３例をあげると線形計画法、整数計画法、非線形計画法などの最適化問題があげられます。この言語の構文はかなり直感的に理解することができます。最適化問題はプログラマでない人でもわかりやすい共通の数学的構造に基づく代数式表記法で書かれます。
AMPLはまたソフトウェアの一つで、AMPL言語にモデルを読み込み、それらを翻訳し互換性のあるソルバーに受け渡す事実上のプラットフォームです。ソルバーは結果をAMPLに返しますのでユーザは結果を表示することができ、さらに分析することができます。このようにいつくかの最適化問題に対する入力と様々なソルバーに対する入力を統一することができます。
このような問題を記述するための環境はAMPLだけではありません。線形計画法や二次計画法についてはいくつもの形式があります。しかしながら、一般の数理計画問題の形式は著しく少ないです。おそらく最もよく知られているAMPLの競合相手は GAMS 言語[9]です。
AMPL は1990年代初めにベル研究所から始まりました。それ以降大規模に開発されてきました。今日では、AMPLはデータや複数の目的関数の定義を入力するためのデータベースアクセスや、AMPL言語の構成要素を拡張する一般ユーザ関数を作成するためのデータベースアクセスを提供しています。私たちは非線形計画法（NLP）（場合によっては大規模）の基本のみに重点を置きます。しかし、もしご興味がありましたら、NAGのSチャプター（例えばベッセル関数）からブラックボックス関数をAMPLに組み込む方法や他のテーマもご説明致します。

2 最初のAMPLモデルの記述

AMPL言語は自然な“数学的”言語に非常に似ていますので、特別に何か学習する必要はなくすぐにAMPLモデルを記述することができます。Rosenbrockのバナナ関数[15]を解きたいと仮定します。問題は以下のように記述することができます。

$$\min_{x,y\in\mathbb{R}} (1-x)^2 + 100(y-x^2)^2$$



大域的最小点は図で示されているように点（1, 1）に位置します。AMPLで問題を解くために、まずAMPLモデルを作成する必要があります。これは純粋なテキストファイルで、ここでは rosenbrock.mod という名前で、以下の内容を持ちます：

rosenbrock.mod

```
var x;
var y;

minimize ros_func:
   100*(y - x^2)^2 + (1-x)^2;

let x:=-1.2;
let y:=1;
```

表記法は簡単で、一目瞭然です。最初の２行は変数名を宣言し、minimize は最適化の処理と目的関数（ここではros_funcと呼びます）を定義します。最後の２行は任意で、開始点を示しています。各行はセミコロンで終わる必要がありますのでご注意下さい。後で他のモデルで制約条件、変数の配列及びパラメータやデータファイルを定義しますが、まずRosenbrockの問題を解きましょう。

3　準備－ AMPLプラットフォームの準備

計算を行えるようAMPLの環境を整えるには２つのことが必要となります。一つめは、AMPL実行フ

ファイル、すなわちモデルを読み込み、ソルバーを実行するためのプラットフォームです。このプログラムは市販の製品です。まだお持ちでない場合は、AMPL Student Editionという名前のテスト限定版（最大300 個の変数と300 個の制約条件までを持つモデル）をダウンロードするか、あるいは制限のないトライアル版を入手する必要があります。テストが目的の場合は、前者で十分です。MS Windows、Linux、MacOS、Solarisなどのいくつかのアーキテクチャ用に[1] からダウンロードすることができます。それはgzipで圧縮された実行ファイルとしてダウンロードされます。例えば、MS Windows版のダイレクトリンクは[2]です。コマンドgzip -d ampl.exe.gz あるいは他の類似のコマンドを使用して解凍し作業用ディレクトリに保存して下さい。

MS Windows ユーザは [11] からgzipプログラムをダウンロードすることができます。あるいは7-zip [17]のようなGUIで製品を使用することができます。

２つめに必要なことは、AMPL入力用のインターフェースを用いてコンパイルされたソルバーです。私たちはこのようなサポートを持つ２つのNAGルーチン、E04UF [6] と E04UG [7]を用意しました。E04UFは密非線形最適化問題に適しており、E04UGは広域的スパース非線形最適化問題に適しています。

両者ともNAG webサイト [14]からダウンロードできます。MS Windowsと Linux向けの実行ファイルとしてダウンロードされ、便利の良いよう既にコンパイル済みであり、すぐに使用できます。

注：これらはデモでの使用のみを対象としたもので、商業的利用を対象としていません。

これらはAMPL Student Editionと同じ制約があります。たとえば、300個の変数より大きいモデル、あるいは300個の制約条件より大きいモデルを解くことはできません。もし別のアーキテクチャや制限のないバージョンを希望する場合、ソースコードを含めていますので、ユーザのFortran NAGライブラリや C NAGライブラリを用いてこのようなバージョンをコンパイルすることができます。あるいは、どのNAG library をお持ちか私たちにお知らせいただければ、できる限りの対応をさせていただきます。ユーザ独自のNAG-AMPLソルバーを構築されたい場合は、AMPLインターフェースを中心に扱うセクション9を参照下さい。

AMPL実行ファイルを用意し現在作業中のディレクトリにソルバーをコピーする準備ができましたら２つのソルバー両方ともコピーして下さい。コマンドプロンプトから引数なしでそれらを実行することで簡単にテストすることができます。以下のメッセージに類似したメッセージが表示されます：

```
C:¥honza¥math¥nag¥ampl>e04uf
E04UF, NAG Fortran Library Mark 22
**********************************************************
***    This routine is intended for demonstration only.          ***
***    Commercial use is not permited.                           ***
***    A limited version to problems with up to                   ***
***    300 variables and 300 constraints.                        ***
**********************************************************
```

Licence: valid (demonstration only)

Based on library:
*** Start of NAG Library implementation details ***

Implementation title: NAG Fortran Library for Win32 Applications (DLL)
Precision: FORTRAN double precision
Product Code: FLDLL224ML
Mark: 22.2 (self-contained)

*** End of NAG Library implementation details ***
No test problem set, terminating.
Call `e04uf -?` for help.

4　AMPLでのモデルの読み込み、ソルバーの呼び出し、結果の確認

これで、最初のAMPLモデル、rosenbrock.mod を解く用意ができました。わかりやすくするためにその他の全てのファイルと同じディレクトリにコピーし、AMPL インタープリターを開始するためにコマンドラインからamplを呼び出します。AMPLコマンドプロンプトが表示されます。

ampl:

AMPL はユーザ入力の準備ができていますが、各命令文の終わりにセミコロンをつけるのを忘れないようにして下さい。もし忘れてしまった場合は、プロンプトは以下に変わり、次の入力（前の行の続き）を要求します。

ampl?

';'を入力するだけでよく、AMPLは続行します。
代表的なAMPLセッションは以下のようになります：

```
ampl: model "rosenbrock.mod";
ampl: option solver e04uf;
ampl: solve;
ampl: display x, y;
ampl: display ros_func;
```

これによりモデルがメモリに読み込まれ、ソルバーe04uf に問題を解くよう指示します。変数と最終的な目的関数が表示されます。
もしAMPLが現在のディレクトリにファイルrosenbrock.mod を見つけることができない場合は、以下のようなエラーがレポートされます：

```
ampl: model "missing_model.mod";
Can't find file "missing_model.mod"
context: >>> model "missing_model.mod" <<< ;
```

このエラーが起きた場合は、Windows ユーザと Linux/Unix ユーザはそれぞれ次のコマンドを使用して現在のディレクトリの内容を確認して下さい。

```
ampl: shell 'dir';
ampl: shell 'ls';
```

ファイルの一貫性や構文がチェックされ、エラーは全てレポートされます。代表的な問題としては、未定義の変数、関数名やキーワードのスペルミス、セミコロンのつけ忘れ、角括弧、丸括弧やその他よくあるプログラミングのミスなどがあります。AMPLは大文字と小文字の区別をしますので、Fortranユーザは特に注意が必要です。
二つ目のコマンドはソルバーを選択しており、この場合はe04uf を選択しています。ソルバーが実行される時になると、AMPLが現在の作業ディレクトリもしくはPATH変数で指定された場所にあるその名前の実行ファイルを呼び出すことをこのコマンドは意味しています。作業ディレクトリにソルバーが置かれておりユーザがセクション３で示唆している通りにテストを行っている限り、ソルバーは問題なく実行します。
Linux ユーザは、以下のように相対パスを含めてソルバーのフルネームを指定する必要がある場合があります：

```
ampl: option solver './e04uf.exe';
```

solve; とコマンド入力すると様々なことが起きます。現在メモリに格納されているモデルは、バイナリファイルに変換されます。そしてユーザのソルバーが呼び出されバイナリファイルを使用してフェッチされます。ソルバーの全ての出力はAMPLの画面に転送されますので、ユーザは通常計算の進捗を見ることができます。ソルバーが結果を計算すると、AMPLに読み込まれる結果ファイルを生成します。
結果をみる場合は、display コマンドを使用します。パラメータとして変数名、関数名、あるいは数式を指定することができます。
AMPL は効率を良くするためにモデルの変数を入れ替えたり、あるいは変数を除外する場合がありますのでご注意下さい(設定解除が可能な presolveフェーズについてはAMPLヘルプを参照)。
もしソルバーが変数の最終値を画面に出力する場合は、モデルファイルで指定した順番とは異なる順番で出力される場合があります。そのためソルバーから自動的に行われる出力を信頼せずに常にdisplayコマンドを使用する必要があります。
この段階で、ユーザは新しいモデルを読み込むためにメモリから現在のモデルを削除しなければならない場合があります。もしこれを行わなかった場合、変数や関数の定義が混ざり合いおそら

く衝突が起こる可能性があります。これを行うコマンドは以下です：

ampl: reset;

もしくは以下のコマンドでセッションを終わらせてコマンドプロンプトに戻ります。

ampl: quit;

もしメモリに読み込まれたモデルがどんな内容か確認する必要がある場合、show;コマンドを使用します。

ampl: show;

これはモデルの全要素を表示します。各要素がどのように定義されているか見たい場合も同じコマンドを使用します。例えば、以下のとおりです。

ampl: show ros_func;

モデルを解く場合は、以下をタイプします。

ampl: solve;

決定変数の現在の値は開始点となりますので、変更したい場合は以下のように行います。

ampl: let x:=10;
ampl: let y:=-5;

5 選択ソルバーの設定

多くの最適化ソルバーはユーザがアルゴリズムをチューニングできるように任意の設定パラメータを備えています。提供される NAGソルバー E04UF と E04UG は確実にこの機能を活用しています。幸いAMPL はそれらに対処する便利な方法を提供しています。一般的なオプション設定コマンドは以下です。

ampl: option solver_options 'keyword1=option keyword2=option ...';

ここでは solver_options は特定のソルバーのオプション設定の文字列につけられた名前です。ソルバー e04uf の場合は、このような文字列は 04uf_options と呼び、e04ugの場合は、e04ug_options と呼びます。文字列の名前の次にキーワードのリストが続きます。それらの値はアポストロフィで囲まれます。
ソルバー E04UF は以下の実装されたオプションを持ちます。（ε はマシンの精度を表します。[6])を参照）：

feasibility	実現可能性許容値 （デフォルト $\sqrt{\varepsilon}$）
maxit	主要反復限界値（デフォルト 0、自動）
optimality	最適性許容値（デフォルト$\varepsilon^{0.8}$）
precision	関数の精度 （デフォルト$\varepsilon^{0.9}$）
print	印刷レベル （デフォルト 10）

唯一の引数として'-='を指定してソルバーを呼び出すと必ずこのようなリストが表示されます。
AMPL環境では以下のようになります：

ampl: shell 'e04uf -=';

設定についての詳細な説明はルーチンの正式なドキュメント[6, 7]で記述されています。
緩い精度の許容値を用いて、なおかつ画面への出力を制限して前述のモデルを解きたいとします。
solveコマンドを呼び出す前に以下を呼び出します。

ampl: option e04uf_options 'print=0 precision=1e-3';

このように簡単にオプション設定を調整することができ、再度何もコンパイルをすることなくモデルを再計算できます。特にソルバーのパラメータの感度を確認する場合、テストを行うのに非常に便利です。
前述のAMPLセッションの詳細なリストは以下です。

```
ampl: model "rosenbrock.mod";
ampl: option solver e04uf;
ampl: option e04uf_options 'print=0 precision=1e-3';
ampl: solve;
E04UF, NAG Fortran Library Mark 22:
************************************************************
***   This routine is intended for demonstration only.     ***
***   Commercial use is not permited.                      ***
***   A limited version to problems with up to             ***
***   300 variables and 300 constraints.                   ***
************************************************************
print=0
precision=1e-3
Calls to E04UEF
------------------
```

Verify Level = -1

Function Precision = 1.000000e-03

Print Level = 0

Result OK

converged: optimal solution found

ampl: display x, y;

x = 1.00015

y = 1.00028

ampl: display ros_func;

ros_func = 4.79927e-08

ampl: quit

6 AMPLモデルの改善

このセクションでは、ユーザがより現実的なモデルを作成するのを可能にする、AMPL 言語の他の要素について紹介したいと思います。後ほど、AMPLの最大の強み（新規モデルの開発）を実証するために架空の問題を使用します。

AMPL言語を理論的に長々と紹介するのではなく、それらを飛ばして例を使用して重要な点を全てお見せします。Hock-Schittkowskiのテストスイート [12]からテスト問題番号72を選びました。以下の制約つき最適化問題とそのAMPLモデルを参照下さい。

$$\min_{x \in \mathbb{R}^4} \quad 1 + \sum_{j=1}^{4} x_j$$

$$\text{subject to } \sum_{j=1}^{4} \frac{a_{i,j}}{x_j} \leq b_j, i = 1.2$$

$$x_j >= 0.001, \quad j = 1,\ldots,4 \qquad (1)$$

$$x_1 \leq 400000$$

$$x_2 \leq 300000$$

$$x_3 \leq 200000$$

$$x_4 \leq 100000$$

where

$$\mathrm{A} = \begin{pmatrix} 4 & 2.25 & 1 & 0.25 \\ 0.16 & 0.36 & 0.64 & 0.64 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.0401 \\ 0.010085 \end{pmatrix}$$

hs072.mod

```
# Hock-Schittkowski test problem #72
#
# W. Hock, K. Schittkowski, Test Examples for Nonlinear Programming
# Codes, Lecture Notes in Economics and Mathematical Systems,
# Springer, Vol. 187, 1981

var x {1..4} >= 0.001;

param a {1..2, 1..4};
param b {1..2};

minimize obj: 1 + sum {j in 1..4} x[j];
subject to constr {i in 1..2}: sum {j in 1..4} a[i,j]/x[j]<=b[i];
subject to upperb {j in 1..4}: x[j] <= (5-j)*1e5;

let x[1] := 1;
let x[2] := 1;
let x[3] := 1;
let x[4] := 1;

data;
param a:     1        2        3        4 :=
      1        4     2.25        1    0.25
      2     0.16    0.36     0.64    0.64   ;
param b :=
      1    0.0401
      2    0.010085 ;
```

AMPL言語には一度に紹介された多くの特徴（配列、定数、制約など）がありますが、それは直感的に理解できるものでなければなりません。ここでは詳しくそれらを見てみます。
より複雑なモデルで処理する場合、コメントが特に役に立ちます。モデルの中で#文字で始まる行は全てコメントとしてみなされますので、無視されます。
スカラー（決定）変数はrosenbrock.modで既に紹介しました。それらは以下のような変数名が後に続くキーワード変数により宣言されていることを思い出して下さい。

```
var my_var;
```

もしお望みならば、スカラー変数を変数の（おそらく多次元）配列に変換する一連のインデックスを定義する波括弧を加えることができます。例えば以下です。

```
var my_var{1..2, -1..1};
```

これはmy_var[1,-1]、my_var[1,0]、my var[1,1]、my var[2,-1]などと呼ばれる６個の変数の二次元配列を宣言します。角括弧が配列の一つの特定の要素しか参照しないのに対し、AMPLの波括弧が一連のインデックスを定義するのは何度も目にします。
私たちの場合、

```
var x {1..4} >= 0.001;
```

この行は変数x[1]からx[4]までを下限の指定と併せて宣言しています。全ての変数に対する矩形制約を一度で示唆する簡単な方法です。このような限度値は後で別の制約にも含めることができます；それはユーザの好みによります。
配列と変数のインデックスは数式のいくつかを単純化するのに大いに役立ちます。全ての変数を加算したいとします。x[1]+x[2]+x[3]+x[4] と記述するか、あるいは単に以下を記述します：

```
sum {j in 1..4} x[j]
```

同様に乗算の命令文もあります：

```
prod {j in 1..4} x[j]
```

決定変数に加えて、定数（モデルのパラメータ）も宣言することができます。構文は変数と同じで、varの代わりにキーワード paramを使用するだけであとは変わりません。パラメータは特に２通りで役に立ちます。一つめは、和や積などのいくつかの数式を単純化します。二つめは、同じ問題の型のモデルを再利用することが望ましい場合があります。もし数字を式に含まれないようにし、パラメータがその代わりに使用される場合、モデルの数値部分（AMPLではデータブロックと呼びます）を交換するのは簡単で、残りの部分は問題なく機能します。
パラメータはモデルを解く前に初期化される必要があります。それらは宣言のところですぐに、あるいは後でデータブロックのところで関連づけされています。前者はパラメータが直接計算できる場合、例えばインデックスや他のパラメータに基づいて計算できる場合、役に立ちます。以下の例はヒルベルト行列がどのように作成することができるかを表しています：

```
param n;
param H {i in 1..n, j in 1..n} := 1/(i+j-1);
```

n がデータブロックで指定される場合は必ず、行列Hは自動的に希望する大きさに更新されます。これは以下のAMPLセッションで表されています。

```
ampl: param n;
ampl: param H {i in 1..n, j in 1..n} := 1/(i+j-1);
ampl: data;
ampl data: param n:=2;
ampl data: display H;

ampl: update data n;
ampl: data;
ampl data: param n:=4;
ampl data: display H;
```

その他の全ての場合で､全ての要素をリストに載せることでデータブロックにパラメータを割り当てることができます｡そのやり方について様々なフォーマットがあります｡詳細は参考文献[8]を参照下さい。以下の２つの例は同じ行列を定義しています。最初の例は hs072.mod にコーディングされているように表形式のデータレコードを使用しており､もう一方の例は全てゼロでない要素を明示し、それぞれの値はインデックスが先に表示されています：

```
data;
param a:   1      2      3      4 :=
       1   4   2.25      1   0.25
       2 0.16   0.36   0.64   0.64
```

```
data;
param a:=
 1  1    4
 1  2    2.25
 1  3    1
 1  4    0.25
 2  1    0.16
 2  2    0.36
 2  3    0.64
 2  4    0.64
 ;
```

データブロックはdata;文で始まり、モデルの残りの部分と併せて（hs072.mod参照）一つのファイルあるいは様々な問題の交換データをさらに簡単にする別のファイル（セクション７参照）に置くことができます。

目的関数は、関数名と関数定義が後に続くminimize（あるいはmaximize）キーワードによって与えられます。同じ表記法が制約に対しても同様に使用されます。制約は subject to（あるいは単にs.t.)キーワード、制約名と制約それ自体によって表されます。波括弧の制約を組み合わせて一連の制約を一度に指定することが可能です。

上記のAMPL言語の全ての構文はAMPLモデルの大部分を理解するのに十分です。モデル hs072.mod をよく見ると、４つの決定変数、１つの目的関数、２つの非線形制約及び８個の簡単な限度値があります。最適解は[12]でレポートされているように目的関数値727:67937を持つ x = (193:4071; 179:5475; 185:0186; 168:7062)です。
問題は Rosenbrock モデルと全く同じ方法で呼び出すことができます：

```
ampl: reset;
ampl: model 'hs072.mod';
ampl: option solver 'e04uf';
ampl: solve;
ampl: display x;
ampl: display obj;
```

E04UFはレポートされた値と一致しています。
このようなより複雑なモデルの双対変数、スラックや限度値に興味をお持ちかもしれません。上限、下限、双対変数やスラック変数を表示するには変数名や制約名の後にそれぞれ.ub、.lb、.dual や.slack を単に付け足します。例えば、前述の問題の結果は以下のようになります：

```
ampl: display x.lb,  x,  x.ub,  x.slack;
:    x.lb       x          x.ub     x.slack    :=
1   0.001   193.407    4e+05    193.406
2   0.001   179.547    3e+05    179.546
3   0.001   185.018    2e+05    185.017
4   0.001   168.707    1e+05    168.706
;
ampl: display constr.lb,  constr,  constr.ub,  constr.dual;
:     constr.lb    constr    constr.ub    constr.dual   :=
1    -Infinity    -7692.94    0.0401      -7692.94
2    -Infinity    -41466.8    0.010085   -41466.8
;
```

7　AMPLケーススタディー非線形データフィッティング

AMPLが最もサポートする部分が新しいモデルのプロトタイピングであるのは特に驚くことではありません。モデルを現実にさらに近づくよう調整するのに新しい制約の追加が必要となるため、あるいは単に新しいデータを得るために、開発の際に様々な変更が必要となります。以下の架空の問題で実証します。
物理学の実験があり、測定データが凸放物線関数の動きをすることを知っていると仮定して下さ

い。さらにこの関数は正値の入力に対して増加していると予想します。
私たちはこのような未知の商a、b、c を持つ二次関数の関係を公式化することができます。これらの商は、凸性を与えるというa に対する付加制約と、x > 0の場合に増加する作用を制限するというb に対する付加制約を持ちます。私たちは曲線フィッティング、例えば二乗誤差の合計の最小化に最小二乗法を用います。問題は以下のように記述することができます：

$$\min_{a,b,c\in\mathbb{R}} \quad \sum_{i=1}^{n}(f(x_i)-y_i)^2$$

$$\text{subject to} \quad a \geq 0$$
$$b \geq 0$$
$$\text{where}$$
$$f(x) = ax^2 + bx + c$$

組み合わせ［xi, yi］は測定点です。データとして以下の８点を８個の適切な測定値と併せて使用しましょう：

x_i	y_i
0.5	3.9802
1.0	7.3611
1.5	6.7004
2.0	11.0118
2.5	14.3323
3.0	22.3791
3.5	27.2975
4.0	36.0287

この問題は、モデルとは別にデータを格納することが有利になる代表的な例です。新しい測定結果を受け取る場合、モデルに触れることなくデータを簡単に交換することができます。AMPL言語のモデルとデータファイルは以下のように記述することができます：

lsfit1.mod

```
param n >= 1;
        # number of measurements
param xpts{1..n};
        # measurement points (x_i)
param D{1..n};
        # Data to fit y_i
var a;
var b;
var c;

minimize obj:
    sum{i in 1..n}
        (a*xpts[i]^2+b*xpts[i]+c-D[i])^2;
subject to constr1:    a>=0;
subject to constr2:    b>=0;
```

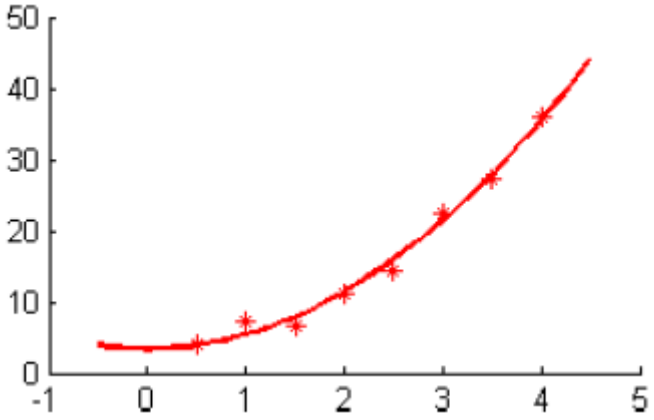
lsfit1a.dat

```
data;
param n := 8;
param xpts :=
    1   0.5
    2   1.0
    3   1.5
    4   2.0
    5   2.5
    6   3.0
    7   3.5
    8   4.0 ;
param D :=
    1    3.9802
    2    7.3611
    3    6.7004
    4   11.0118
    5   14.3323
    6   22.3791
    7   27.2975
    8   36.0287 ;
```

データファイルを読み込むために一般的なコマンドの順番でコマンドdata filename;を付け加える必要があります。 入力全体は以下のようになります：

```
ampl: reset;
ampl: model 'lsfit1.mod';
ampl: data 'lsfit1a.dat';
ampl: option solver 'e04uf';
ampl: solve;
ampl: display a,b,c;
ampl: display obj;
```

計算された商は、a = 2.00052、b = 0、c = 3.38308で、合計誤差（目的関数）は9.615132です。フィッテングの図を参照下さい。



ここで、結果を確認したいと仮定します。実験をあと２回繰り返すことができます。以下の表で要約されているように、独立したデータセットをさらに２つ得ることができます：

Points xi	Set 1	Set 2	Set 3
0.5	3.9802	5.4776	3.0925
1.0	7.3611	8.1040	4.0135
1.5	6.7004	8.4392	6.1188
2.0	11.0118	10.748	9.9346
2.5	14.3323	14.8901	13.6188
3.0	22.3791	19.3112	20.0661
3.5	27.2975	23.7691	27.8293
4.0	36.0287	32.2135	37.9609

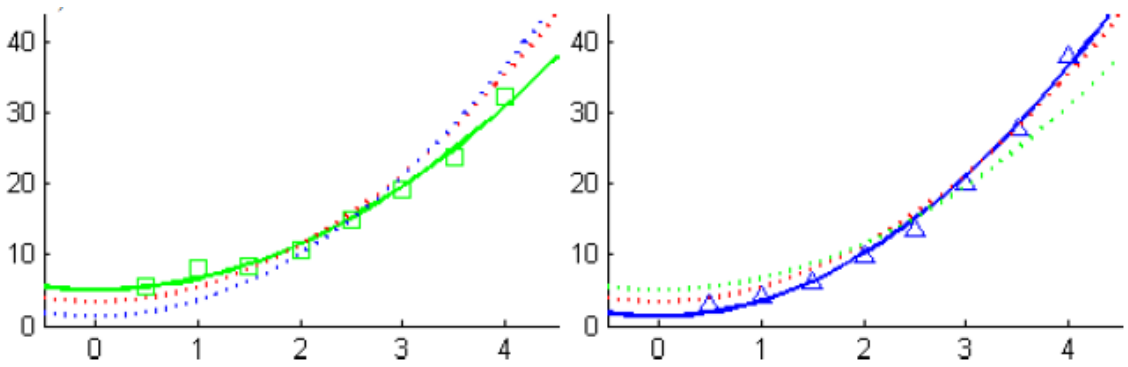
新しいデータであと２つデータファイルlsfit1b.dat と lsfit1c.dat を作成し、AMPLセッションを続けましょう：

```
ampl: reset data;
ampl: data lsfit1b.dat;
ampl: solve;
ampl: display a,b,c;
ampl: display obj;
ampl: reset data;
ampl: data lsfit1c.dat;
ampl: solve;
ampl: display a,b,c;
ampl: display obj;
```

データを変更したい場合、モデル全体を再読み込みする必要はありません。reset data; コマンドを使用するだけです。結果は以下の表のように要約されます：

Parameter	Set 1	Set 2	Set 3
a	2.00052	1.61829	2.19829
b	0	0	0
c	3.38308	5.05247	1.31522
obj	9.615132	5.72935	7.27976

予想したとおり、それぞれのデータセットは他とは異なる個別の解をもたらしました。（グラフを参照、点線は他のデータセットの結果を反映）



私たちの新しい課題は、一つの最終的な曲線を得るように全てのデータセットを一貫して処理する新しいモデルです。全てのデータ点を簡単に一つにまとめて、一つの大きなデータセットを作ることができました。しかし私たちの場合これは必ずしも最善の解ではありません。それぞれの実験のデータを独立させておきたいとします。そして各データセットの最大誤差（いわゆる最悪の事態）を最小化したいとします。
これは以下のように記述することができます。

$$\min_{a,b,c\in\mathbb{R}} \max_{k=1,2,3} \sum_{i=1}^{n} \left(f(x_i) - y_i^k\right)^2$$

$$\text{subject to} \quad a \geq 0$$
$$b \geq 0$$

ここではy_i^k（k = 1, 2, 3 ）はそれぞれのデータセットの測定データです。最大誤差である特別な変数alpha を使用してAMPLでモデル化することができます：

lsfit2.mod

```
param n >= 1;                    # number of measurement points
param m >= 1;                     # number of data sets
param xpts{1..n};                # measurement points (x_i)
param D{1..n,1..m};               # Data to fit y^k_i

var a;
var b;
var c;
var alpha;              # maximal error of the fit for a data set

minimize obj: alpha;
subject to err {j in 1..m}:
    sum{i in 1..n} (a*xpts[i]^2 + b*xpts[i] +c - D[i,j])^2 <= alpha;
subject to constr1: a>=0;
subject to constr2: b>=0;
```

lsfit2.dat

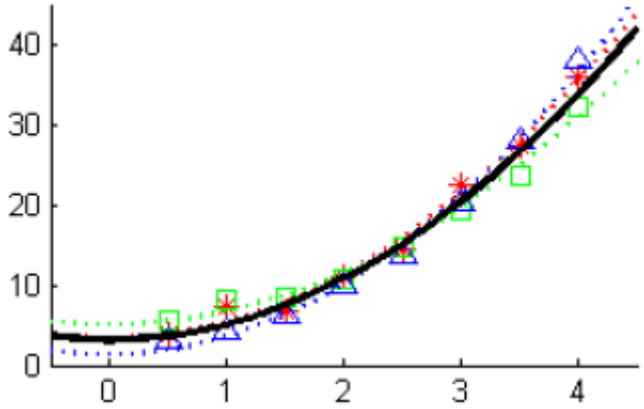
```
data;
param n := 8;
param m := 3;
param xpts :=
 1    0.5
 2    1.0
 3    1.5
 4    2.0
 5    2.5
 6    3.0
 7    3.5
 8    4.0 ;
param D:        1          2          3    :=
 1          3.9802     5.4776     3.0925
 2          7.3611     8.1040     4.0135
 3          6.7004     8.4392     6.1188
 4         11.0118     10.748     9.9346
 5         14.3323     14.8901    13.6188
 6         22.3791     19.3112    20.0661
 7         27.2975     23.7691    27.8293
```

```
8          36.0287     32.2135    37.9609 ;
```

AMPL セッションを続行し問題を解くには、いつものようにタイプします：

```
ampl: reset;
ampl: model lsfit2.mod;
ampl: data lsfit2.dat;
ampl: solve;
ampl: display a,b,c;
ampl: display alpha;
ampl: display {i in 1..m}: err[i]+alpha;
```

最後のコマンドは各データセットの実際のフィッティング誤差を示しています。計算された商は a = 1.91428、b = 0、c = 3.14526 で合計誤差は（alpha）25.2803 、各フィッティング誤差（err[i]+alpha）は 25.2803、24.7906、24.7699です。



この小さな例では非線形最小二乗法の最善の方法を表していませんが、私たちの目的は一つのことを明らかにすることだけでした。すなわち多量にコーディングせずにモデリング言語で書かれたモデルを調整することが実に簡単であることを示すことでした。もしモデルが一般的なプログラミング言語で書かれた場合、同じ変更をするにはもっと多くの労力が必要となります。特に新しい目的関数や制約を組み込むのに多くの労力が必要となります。

AMPL ではユーザは単にコマンドをタイプし、NAGルーチンに残り（結果の出力）を実行させるだけです。重要なのは結果です。

8　その他のモデル

AMPL言語は事実上非線形計画問題のソルバーの中の標準のインターフェースです。そのためAMPLで書かれたベンチマークのための様々なテストスイートを見つけることができます。先に記述したHock-Schittkowski テスト集全体は、Professor Robert J. Vanderbei[13]のwebページ上にAMPL書式で掲載されています。

COPS（a large-scale Constrained Optimization Problem Set：大規模制約つき最適化問題セット）という名前の別のベンチマークスイートは[5]からダウンロードすることができます。COPSは人口動態、形状最適化あるいは流体力学といった分野から生じるNLPの22個の難しいテストケースから構成されています。これら全てはAMPLモデルとして得ることができます。これはAMPL言語の順応性をよく実証しています；また創造性を高めるためにも見る価値があります。

9 テクニカルノート

これまでユーザの視点、すなわちAMPLでのモデルの書き方や解き方に重点を置いてきましたが、AMPLと互換性のあるこのようなソルバーの構築の仕方について記述するのを避けてきました。詳細にそれを説明することはこのテキストの範囲をはるかに超えています。与えられたソースコードからご自分のAMPLソルバーをコンパイルできるレベルまでしか説明していません。ご自分でAMPL インターフェースを開発することに興味がありましたら、詳細は[10] を参照下さい。
セクション 4で暗に示しているように、AMPLプラットフォームはあらゆるソルバーとファイルを経由してやり取りをしています。ユーザのモデルファイルはバイナリファイルに変換され、ソルバーはコマンドラインの引数として任意パラメータ（文字列 solver_options）と共にそのモデルファイルの名前を取得します。一旦最適化の実行が終了すると、ソルバーはAMPLによって読み込まれる結果を含む別ファイルを生成します。
もしAMPLと互換性のあるソルバーを作成したい場合は、３つの主な要素が必要となります：

- ソルバー自体；ユーザの問題を解く機能をもつルーチン
- 入力されたバイナリファイルを読み込み、格納されているモデルを処理し、最後には逆に結果ファイルを生成できる一連のルーチン
- 両者をつなぐメインルーチン

私たちの場合、ソルバーはユーザがインストールしたライブラリによって E04UFF と E04UGF（NAG Fortran ライブラリ） あるいは E04UCC と E04UGC（NAG C ライブラリ）のどちらかになります。
AMPL の開発者は入力、出力、モデルの機能評価に対処するAMPL ソルバーライブラリ（ASL）と呼ばれるライブラリを提供します。ユーザは[4]から無料のソースコードをダウンロードすることができます。最終的に、ファイル e04uff_ampl.c、e04ugf_ampl.c、e04ucc_ampl.c 及び e04ugc_ampl.c はASLとNAGライブラリとの間に中間層を作成します。それらはNAG webサイト[14]からご利用いただけます。
ユーザはASLをダウンロードして構築する必要があります。ユーザのプラットフォーム用の正しいmakefileを選択するためのスクリプト configure とconfigurehere があります。ライブラリを構築するためにreadme ファイルの内容とmakefileの指示に従って下さい。もしご希望でしたら小さなサンプルプログラム[16]を使用してASLをテストすることができます。そのテストではASLビルドが成功したかどうか結果を示します。

NAG ライブラリ、ASLビルドと提供ルーチンの一つを併せてコンパイルすることのできるmakefileがいくつかあります。 例えば、Makefile.fldll224ml.cl はNAG Fortran ライブラリFLDLL224ML (Mark 22, Windows DLL 32-bit)と Microsoft C++ コンパイラ (cl)を対象としています。 makefileを使用する前に、NAGライブラリとASLのパスを定義しているファイル中の該当する行を編集する必要があります。例えば以下のような行です。

NAG_LIBDIR = C:¥math¥NAG¥FL22¥fldll224ml
AMPL_LIBDIR = C:¥honza¥math¥ampl¥amplsolver¥sys.cl

もしご希望ならば、他の部分（例えばコンパイラフラグ）も同様に編集することができますが、そのまま動くはずです。以下のテーブルはテスト済みのシステムを要約したものです。

NAG Library code	System	Compilers
CLDLL084ZL	Win32	cl
CLL6A08DGL	Linux64	gcc
CLL6A09DHL	Linux64	gcc
CLLUX08DCL	Linux32	icc
CLLUX08DGL	Linux32	gcc
CLW3209DAL	Win32	cl, gcc
FLDLL214AL	Win32	cl, gcc
FLDLL224ML	Win32	cl, gcc
FLL3A21DFL	Linux32	gcc
FLL3A22DFL	Linux32	gcc
FLL6A22DFL	Linux64	gcc
FLL6I22DCL	Linux64	gcc, icc
FLLUX21DGL	Linux32	gcc
FLLUX22DCL	Linux32	gcc

ユーザのNAG ライブラリ／システムが対応されていない場合は､ユーザ独自のものを記述できるようにする一般のmakefileが２つあります。必要であれば、NAGライブラリと併せて提供されている、実装の詳細が書かれているユーザノートをご確認下さい。
ソルバーを構築するために、makefileのコメントに記述されているとおりにmake（あるいはnmake）を呼び出して下さい。たとえば以下のようになります：

C:¥honza¥math¥nag¥ampl¥src>nmake -f Makefile.fldll224ml.cl all

Microsoft (R) Program Maintenance Utility Version 8.00.50727.42

```
cl /O2 /nologo /I"C:¥honza¥math¥ampl¥amplsolver¥sys.cl" /DUSE_STDCALL
e04uff_ampl.c "C:¥honza¥math¥ampl¥amplsolver¥sys.cl¥amplsolv.lib"
"C:¥math¥NAG¥FL22¥fldll224ml¥lib¥FLDLL224M_nag.lib" /Fe04uf.exe
e04uff_ampl.c
Creating library e04uf.lib and object e04uf.exp

cl /O2 /nologo /I"C:¥honza¥math¥ampl¥amplsolver¥sys.cl" /DUSE_STDCALL
e04ugf_ampl.c "C:¥honza¥math¥ampl¥amplsolver¥sys.cl¥amplsolv.lib"
"C:¥math¥NAG¥FL22¥fldll224ml¥lib¥FLDLL224M_nag.lib" /Fe04ug.exe
e04ugf_ampl.c
Creating library e04ug.lib and object e04ug.exp
```

AMPL インターフェースを用いてユーザ独自のソルバーを構築すると、セクション３で示唆されている通りにユーザはテストする必要があります。あるいはmakefileの内蔵テストを用いてテストする必要があります：

```
C:¥honza¥math¥nag¥ampl¥src>nmake -f Makefile.fldll224ml.cl test

Microsoft (R) Program Maintenance Utility Version 8.00.50727.42
Copyright (C) Microsoft Corporation. All rights reserved.
############################################################
# Mini test of e04uf.exe
# Test1: Routine identification & licence check
#      the result should start with:
#          E04UF, NAG Fortran Library
#          Licence: valid
# Test2: Simple test problem, Hock-Schittkowski #72
#      an approximate results are:
#          x = [193.4, 179.5, 185.0, 168.7]
#          objective function = 727.679
############################################################
########################## TEST 1 ##########################
                e04uf.exe
E04UF, NAG Fortran Library
Licence: valid
...
```

何か問題がある場合は、ビルド処理がエラー無しで完了し、NAGライブラリがPATH変数（Windowsの場合）あるいはLD LIBRARY PATH（Linuxの場合）で指定されたパスに存在しているかどうか確

認して下さい。ビルドはNAGライブラリアプリケーションの他のビルドと類似している必要があります。
残念ながら私たちはASLライブラリに関するサポートを提供することができませんが、ユーザのライブラリバージョンに固有のインターフェースの構築を支援することができる場合があります。遠慮なくお問い合わせ下さい。

10 結び

AMPL と NAG 最適化ルーチンのお試しセッションを楽しんでいただけたことと思います。
AMPL言語の基本についてほとんどを取り上げましたが、ご自分で試していかにAMPL（あるいはモデリング言語）が便利かを知っていただくのに十分だったと思います。
モデルの多様性は、おそらく様々なソルバーや問題に渡って統一されたインターフェースと共にAMPLの大きな強みです。これにより開発の最初のフェーズにおいてかなりの時間を省くことができます。結果を得るのに多量にコーディングする必要がないので、デモを行ったり教育したりするのに便利です。AMPLは全ての解決法にはならないのは明らかですが、たとえNAGソルバーをソフトウェアの大部分に組み込みたい場合でもAMPLは数学モデルの初期のプロトタイピングやテストを行うのに優れています。
皆様のご意見をぜひお聞かせ下さい。ご意見やご経験をお聞かせいただければ嬉しく思います。
NAG Webサイトから全てのファイルをダウンロードできることをお忘れにならないようお願い致します。何か問題に直面されましたか？AMPLや他のモデリング言語を使用されていますか？ライブラリに含まれるルーチンのAMPLインターフェースをご覧になりたいですか？ご回答いただけましたら幸いに存じます。

参考文献

[1] AMPL executables (Student Edition)
http://netlib.sandia.gov/ampl/student/index.html

[2] AMPL executable for MS Windows (Student Edition)
http://netlib.sandia.gov/ampl/student/mswin/ampl.exe.gz

[3] AMPL main website
http://www.ampl.com/

[4] ASL { AMPL Solver Library, source codes
http://netlib.sandia.gov/ampl/solvers/index.html

[5] COPS, a large-scale Constrained Optimization Problem Set
http://www.mcs.anl.gov/~more/cops/

[6] E04UFF, NAG Fortran Library Manual, Mark 22
http://www.nag.co.uk/numeric/FL/nagdoc fl22/pdf/E04/e04uff.pdf

[7] E04UGF, NAG Fortran Library Manual, Mark 22

http://www.nag.co.uk/numeric/FL/nagdoc fl22/pdf/E04/e04ugf.pdf

[8] Fourer R., Gay D. M., Kernighan B. W.: AMPL: A Modeling Language for Mathematical Programming, 2nd. ed, 2002. Brooks/Cole Publishing Company.

[9] GAMS main website
http://www.gams.com/

[10] Gay D. M.: Hooking Your Solver to AMPL, Technical report, 1997. Bell Laboratories, Murray Hill, NJ.
http://www.ampl.com/REFS/hooking2.pdf

[11] gzip website and Windows binary
http://www.gzip.org/
ftp://tug.ctan.org/tex-archive/tools/zip/info-zip/MSDOS/gzip124.exe

[12] Hock W., Schittkowski K.: Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems, No 187, 1981. Springer.
Schittkowski K.: More Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems, No 282, 1987. Springer.
http://www.math.uni-bayreuth.de/~kschittkowski/tpnp08.htm

[13] Hock-Schittkowski test collection as AMPL models
http://www.princeton.edu/~rvdb/ampl/nlmodels/hs/

[14] NAG Technical Reports website
http://www.nag.co.uk/doc/techrep/index.asp

[15] Rosenbrock function, Wikipedia.
http://en.wikipedia.org/wiki/Rosenbrock function

[16] Simple examples for ASL, source codes
http://netlib.sandia.gov/ampl/solvers/examples/index.html

[17] 7-zip website
http://www.7-zip.org/