Les Cahiers du GERAD ISSN: 0711–2440

The Lightning AMPL Tutorial

A Guide for Nonlinear Optimization Users

D. Orban

G–2009–66

October 2009

Les textes publiés dans la série des rapports de recherche HEC n'engagent que la responsabilité de leurs auteurs. La publication de ces rapports de recherche bénéficie d'une subvention du Fonds québécois de la recherche sur la nature et les technologies.

The Lightning AMPL Tutorial

A Guide for Nonlinear Optimization Users

Dominique Orban

GERAD & École Polytechnique de Montréal
C.P. 6079, Succ. Centre-ville
Montréal (Québec) Canada, H3C 3A7

dominique.orban@gerad.ca

October 2009

Les Cahiers du GERAD

G–2009–66

Copyright © 2009 GERAD

Abstract

This intentionally short tutorial is an introduction to the main features of AMPL that are relevant to nonlinear optimization model authoring. Pointers are given to further documentation and resources for more advanced features. The author estimates that reading this document and looking at the examples carefully should not take more than an hour, downloading and installing the software should only take a few minutes. With this document at hand, the user can be up and running in just over an hour.

Résumé

Ce bref tutorial se veut une introduction aux aspects du langage AMPL pertinents à la modélisation de problèmes d'optimisation non-linéaire. On donne ensuite des pointeurs vers des sources d'information plus approfondie. L'auteur estime que la lecture de ce document et l'étude des exemples ne devrait pas prendre plus d'une heure. L'installation des logiciels ne devrait prendre que quelques minutes. En un peu plus d'une heure, l'utilisateur sera capable d'écrire ses propres modèles et de les résoudre.

1 Introduction

AMPL is a modeling language for mathematical programming. It allows users to describe optimization problems. In this document, we concentrate on continuous optimization problems such as

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h(x) = 0, \\ & c_{\text{L}} \leq c(x) \leq c_{\text{U}}, \\ & \ell \leq x \leq u, \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $h : \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued function of equality constraints, $c : \mathbb{R}^n \to \mathbb{R}^p$ is a vector-valued function of inequality constraints, $c_{\text{L}}, c_{\text{U}} \in \mathbb{R}^p$ are the lower and upper bounds for the inequality constraints and $\ell, u \in \mathbb{R}^n$ are the lower and upper bounds on the variables. In this condensed notation, inequalities are understood componentwise. Note that some or all components of c_{L} and of ℓ may be equal to $-\infty$ and some or all components of c_{U} and of u may be equal to $+\infty$. A bound taking an infinite value means that the corresponding constraint is absent.

The salient advantages of a modeling language such as AMPL are that

(a) it provides an intuitive and extensive syntax to describe problems much as we would write them on paper,
(b) it provides facilities to automatically compute derivatives for use by solvers. Hence, derivatives need not be coded by hand and users may concentrate on the modeling task.

In the AMPL modeling language, a problem such as (1) may be represented in an intuitive manner as *two text files*. The user writes the two text files using his or her favorite text editor. Whichever text editor the user choses is in the end irrelevant, but the typing of the model can be more pleasant if an appropriate editor is chosen. For example, some editors will allow the user to have expressions indented automatically or keywords to be highlighted in color. We recommend the two following editors which are available on UNIX, LINUX, MAC OS/X and WINDOWS platforms:

(a) The EMACS editor, www.gnu.org/software/emacs,
(b) The VIM editor, www.vim.org.

Optional AMPL-specific syntax highlighting and indentation extensions are available for both editors at the address www.mgi.polymtl.ca/dominique.orban/software.html.

Finally, we note that the description (1) of our problem is convenient since in AMPL, general constraints and bounds on the variables are usually specified separately. This makes sense because solvers typically treat them differently.

2 Obtaining AMPL

AMPL is commercial software. However, the AMPL authors generously realease a free-of-charge *student version* of AMPL. The student version of the software has a single restriction: the problem size is limited to 300 variables and 300 constraints. In terms of problem (1), this means that n cannot exceed 300 and the total of m, the number of effective inequality constraints—i.e., the number of inequalities for which the lower or upper bound is finite—and the number of effective bound constraints cannot exceed 300. If both the lower and upper bounds of a bound constraint on a variable are finite, they count as two constraints.

AMPL is a *command-line utility*. This means that there is no fancy graphical user interface and you will have to type commands at the command prompt. UNIX, LINUX and MACOS/X users will open a terminal window and WINDOWS users will start the *command* utility, or, as a more convenient alternative, can download the *scrolling window* utility `sw.exe` from www.ampl.com, which offers cut-and-paste facilities, font selection and a larger buffer size.

The student version of AMPL may be downloaded for various flavors of UNIX, including LINUX and MACOS/X and for WINDOWS from the main website www.ampl.com. You will be downloading a compressed executable. Follow the instructions given on the website for local installation. On the WINDOWS platform, be aware that your browser may already uncompress the file as it is being downloaded.

If you desire, a link on the AMPL website will also let you download student versions of some solvers that are compatible with AMPL. In this document, we will use the student version of AMPL to write and debug small-scale models locally. We will use one of the precompiled solvers to solve a small-scale model locally. When the time comes to solve larger-dimensional versions of our models, we will make use of the NEOS Server for Optimization neos.mcs.anl.gov.

3 How Does it Work?

Once the AMPL executable is installed and available, we call it from the command line and we are presented with the AMPL prompt. Valid AMPL commands may be entered at the AMPL prompt. For example:

```
ampl: var x;
ampl: let x := 3.14;
ampl: display x;
x = 3.14

ampl: exit;
```

In the previous example, we declare a variable called `x` by means of the keyword `var`, we assign a value to it using `let` (note the assignment operator `:=`) and we ask AMPL to display the value of this variable using `display`. We then terminate our session with the `exit` command. Note that *all* commands end with a semi-colon '`;`'.

Of course, it would be very tedious to type all commands at the prompt over and over again, so AMPL also accepts a *commands file* as argument. A *commands file* is a text file that can be thought of as a *script* or *batch file*, i.e., a list of commands given in the same order in which we would type them at the prompt. The previous example can be reproduced by creating a text file called `example1.ampl` containing

```
var x;
let x:= 3.14;
display x;
exit;
```

Lauching AMPL with this file as argument then produces the same output as in the previous example.

4 Problem Description

For the purpose of an example, suppose we are modeling an optimization problem in which we seek the natural configuration of N electrons constrained to lie on the surface of a conducting ellipsoid with half-axes r_x, r_y and r_z. This problem is a variation of a problem described in [SK97] and in [DMM04].

Physicits know that the electrons will stabilize when the Coulomb potential is minimal. In $\mathbb{R}^3$, we represent the position of each electron by a triple (x_i, y_i, z_i), $i = 1, \ldots, N$. The Coulomb potential is given by

$$U(x, y, z) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2\right]^{-1/2},$$

up to a scalar factor that will not affect the solution. In this function, $x \in \mathbb{R}^N$ is the vector whose components are the x_i and we used similar definitions for the vectors y and z. The potential U is our objective function to minimize. The electrons being constrained to lie on the surface of an ellipsoid, their coordinates must satisfy

$$\frac{x_i^2}{r_x^2} + \frac{y_i^2}{r_y^2} + \frac{z_i^2}{r_z^2} = 1, \qquad i = 1, \ldots, N.$$

In this problem, there are $3N$ variables and N constraints. There are only general equality constraints; there are no inequality constraints or bounds. The values of N, r_x, r_y and r_z are parameters and may be changed by the user to provide a different instance of the same model problem. Changing N changes the total number of variables and constraints. Changing r_x, r_y or r_z merely changes the geometry of the problem.

As mentioned in §1, the problem is described using two text files:

(a) a *model file*, `electrons.mod`, whose purpose is to represent the *structure* of the problem, i.e., the description (1),

(b) a *data file*, `electrons.dat`, whose purpose is to give values to all constants and parameters in the model—such as N and r—and to specify an initial guess for the solution, i.e., an initial electronic configuration.

Ultimately, the name of these files is unimportant. However, it is good practice to name them as above.

The model file for this problem could be written as in Listing 1 and the data file as in Listing 2. Note that in the listings, indentation only serves the purpose of easing readability. It is not required by the syntax. As the reader will immediately guess, comments in AMPL start with a hash sign '`#`'. This model declares the variables `x`, `y` and `z`, each of them a vector indexed from `1` through `N`.

Listing 1: Model File for the Electrons Problem

```
model;        # Opens up a model file. This directive must be present

param N  > 0, integer; # Number of electrons: value is given in data file
param rx > 0;  # Half axis in x: value is given in data file
param ry > 0;  # Half axis in y: value is given in data file
param rz > 0;  # Half axis in z: value is given in data file

var x {1..N};  # x-coordinates of the electrons
var y {1..N};  # y-coordinates of the electrons
var z {1..N};  # z-coordinates of the electrons

minimize CoulombPotential:   # This is a name given to the objective function
  sum{i in 1..N-1}
    sum{j in i+1..N}
      1/sqrt( (x[i]-x[j])^2 + (y[i]-y[j])^2 + (z[i]-z[j])^2 );

subject to EllipsoidConstraint {i in 1..N}: # Name given to group of constraints
  x[i]^2/rx^2 + y[i]^2/ry^2 + z[i]^2/rz^2 = 1;
```

Listing 2: Data File for the Electrons Problem

```
data;           # Opens up a data file: must be present

param N := 10;  # Actual number of electrons (note the ':=')
param rx := 3;  # Actual half axis in x of the ellipsoid
param ry := 2;  # Actual half axis in y of the ellipsoid
param rz := 1;  # Actual half axis in z of the ellipsoid

# Distribute the electrons randomly on the ellipsoid
option randseed '12345';   # Initialize a random seed

param pi;
let pi := acos(-1.0);  # A 'param' cannot have operations in it. A 'let' can.

param theta {i in 1..N} := 2 * pi * Uniform01();  # 0 <= theta <= 2pi
param phi   {i in 1..N} :=     pi * Uniform01();  # 0 <=  phi  <=  pi

# Assign initial guess: this is an ellipsoid in spherical coordinates
let {i in 1..N} x[i] := rx * cos(theta[i]) * sin(phi[i]);
let {i in 1..N} y[i] := ry * sin(theta[i]) * sin(phi[i]);
let {i in 1..N} z[i] := rz * cos(phi[i]);
```

Why separate the model from the data ? The reasons are readability and flexibility. Whether we want a small-scale or a large-scale instance of a problem, to change some boundary conditions or the half-axes of the ellipsoid in the electrons example, it suffices to adjust some parameters in the data file. With a single model file and several data files, we obtain a family of related problems.

In AMPL, the usual trigonometric functions are available, as are the square root, the powers, logarithms and the sum operator. They all have the expected name and calling sequence. In the data file, we made use of the built-in function `Uniform01()` which returns a random number, according to a uniform law, between 0 and 1. Using the spherical coordinates, we ensured that our starting point satisfied the constraints. Had we not specified an initial configuration, AMPL would have set all variables `x`, `y` and `z` to zero. While most optimization solvers do not require that the initial guess be anywhere close to a local solution, let alone satisfy the constraints, it is generally a good idea to put our knowledge of the problem to good use and specify a reasonable initial guess. This initial guess might help a solver identify a local solution more efficiently, or at all!

With the following commands, we obtain some information about the objective and constraint values at the starting point.

Listing 3: Querying the Objective and Constraint Values at the Initial Point

```
ampl: model electrons.mod;              # Load model file
ampl: data electrons.dat;               # Load data file
ampl: display CoulombPotential;         # Objective value at starting point
CoulombPotential = 138.979

ampl: display EllipsoidConstraint[N]; # Last constraint value at starting point
EllipsoidConstraint[N] = 0
```

The constraint value is zero, which means that it is satisfied; AMPL internally rearranged the expression of the constraint so the right-hand side is zero. Note that `display EllipsoidConstraint;` would have output the whole vector of constraints. We can obtain formatted output using C-like format strings and the `printf` command, and even redirect output to a file:

```
ampl: printf {i in 1..N} '%9.6f  %9.6f  %9.6f\n', x[i], y[i], z[i];

 0.585913   1.613986   0.221845
 0.909399  -0.750201   0.539111
 0.270759  -0.166082   2.877176
 0.184721   0.703949  -2.752804
 0.249890  -0.459935  -2.821707
-0.357302  -1.186707  -2.163886
 0.142948  -0.603632  -2.827765
 0.229000  -0.365430   2.868374
 0.069781  -0.488146  -2.901729
-0.281886   0.343167  -2.831942
ampl: printf {i in 1..N} '%9.6f  %9.6f  %9.6f\n', x[i], y[i], z[i] > coords.txt;
```

The file `coords.txt` now contains the output of the first `printf`, which may be used by, say, a plotting program to represent the electronic configuration.

The model above does not contain any bounds on the variables or general inequality constraints. It is however very easy to incorporate some. Suppose we are now interested in solving the same problem but only on the section of the ellipsoid described by the bounds $-r_z/2 \leq z_i \leq r_z/2$ for all $i = 1, \ldots, N$. We might simply add to our model file the lines of Listing 4. More general inequality constraints can be added in the same way. Note that inequality constraints can be one-sided or two-sided.

Listing 4: Model File for the Electrons Problem with Bound Constraints

```
subject to RangeConstraint {i in 1..N}:
  -0.5 * rz <= z[i] <= 0.5 * rz;
```

What did we learn from this example?

(a) variables are declared with the keyword '`var`' in the model file,

(b) parameters are declared with the keyword '`param` in the model file,

(c) variables are given a value in the data file using '`let`',

(d) parameters are given a value in the data file using '`param`', except if the value involves an arithmetic or mathematical operation, in which case we can use '`let`',

(e) objectives and constraints must be named,
(f) most usual operators and functions are available in AMPL.

By far, the most common error in a model is to forget semi-colons.

5 Solving a Problem Locally

Once we have written our model and data files, how can we go about finding a solution to the problem? The easy answer is to use an existing optimization solver, one that is already able to accept models in the AMPL language. On the AMPL website, a number of precompiled binaries are available for download. In our case, we need a solver able to process nonlinear problems with constraints. We select, for instance, the SNOPT solver [GMS05], download the binary and place it in our environment's search path. The commands file in Listing 5 shows how to solve the problem with SNOPT.

Listing 5: Solving the Electrons Problem with SNOPT

```
model electrons.mod;
data electrons.dat;
option solver snopt; # Must be the name of the executable
solve;
printf {i in 1..N} '%9.6f  %9.6f  %9.6f\n', x[i], y[i], z[i] > snopt-sol.coords;
```

Executing this commands file, we receive the message

```
SNOPT 7.2-8 : Optimal solution found.
104 iterations, objective 17.3441059
Nonlin evals: obj = 87, grad = 86, constrs = 87, Jac = 86.
```

While this is not much information, it does give us the final value of the Coulomb potential, 17.3441059, and an idea of the amount of work performed by solver. We could get a far more detailed output by passing an option to SNOPT telling it to display information as it proceeds. This is done by changing Listing 5 to Listing 6.

Listing 6: Passing Options to SNOPT

```
model electrons.mod;
data electrons.dat;
option solver snopt; # Must be the name of the executable
option snopt_options "outlev=2";
solve;
printf {i in 1..N} '%9.6f  %9.6f  %9.6f\n', x[i], y[i], z[i] > snopt-sol.coords;
```

We now receive the detailed output (truncated for readability)

```
SNOPT 7.2-8 : outlev=2

 SNMEMB EXIT 100 -- finished successfully
 SNMEMB INFO 104 -- memory requirements estimated


 Nonlinear constraints      10     Linear constraints       1
 Nonlinear variables        30     Linear variables         0
 Jacobian  variables        30     Objective variables     30
 Total constraints          11     Total variables         30



 The user has defined      30   out of      30   constraint gradients.
 The user has defined      30   out of      30   objective  gradients.


 Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
     0     14              1  1.3E-01  1.5E-01  4.6913243E+01    14          r
     1      1  3.1E-01      2  5.8E-02  7.8E-01  1.1854748E+02    14 3.6E+02 n rl
     2      2  1.0E+00      3  8.0E-02  9.1E-01  1.9685776E+01    13 3.1E+01 s
     3      2  1.0E+00      4  1.3E-02  3.7E-01  2.0728865E+01    14 3.2E+01

    ...
```

```
   80      1  1.0E+00     84 (4.1E-09) 4.9E-05  1.7344106E+01    12 3.0E+01
   81      1  1.0E+00     85 (7.9E-10) 1.5E-05  1.7344106E+01    12 3.0E+01
   82      1  1.0E+00     86 (1.0E-10)(1.7E-06) 1.7344106E+01    12 3.0E+01

 SNOPTB EXIT   0 -- finished successfully
 SNOPTB INFO   1 -- optimality conditions satisfied

 Problem name                 at20213
 No. of iterations                104   Objective value      1.7344105903E+01
 No. of major iterations           82   Linear objective     0.0000000000E+00
 Penalty parameter           2.993E+01   Nonlinear objective  1.7344105903E+01
 No. of calls to funobj            87   No. of calls to funcon             87
 No. of superbasics                12   No. of basic nonlinears            10
 No. of degenerate steps            0   Percentage                        .00
 Max x                     17 2.0E+00   Max pi                    11 1.0E+00
 Max Primal infeas          0 0.0E+00   Max Dual infeas           27 3.3E-06
 Nonlinear constraint violn    6.0E-10

 Solution not printed

 Time for MPS input                              .00 seconds
 Time for solving problem                        .02 seconds
 Time for solution output                        .00 seconds
 Time for constraint functions                   .00 seconds
 Time for objective function                     .02 seconds
SNOPT 7.2-8 : Optimal solution found.
104 iterations, objective 17.3441059
Nonlin evals: obj = 87, grad = 86, constrs = 87, Jac = 86.
```

The detailed output shows the progress of the solver and, by inspection, might point to a source of difficulties should we encounter any. Most solvers have numerous options and the user should refer to the solver's documentation for a complete description. Typically, a list of available options and a short description of their purpose can be obtained by calling the executable for the solver from the command line with the argument "`-=`" (a *minus* sign followed by an *equal* sign). For instance, "`snopt -=`" displays all options recognized by SNOPT.

Our commands file finally outputs the final solution—i.e., after the `solve` command—to a file for plotting, as we did in the previous section. For illustration, we increased the value of N to 41. The final electronic configuration is given in Fig. 1.

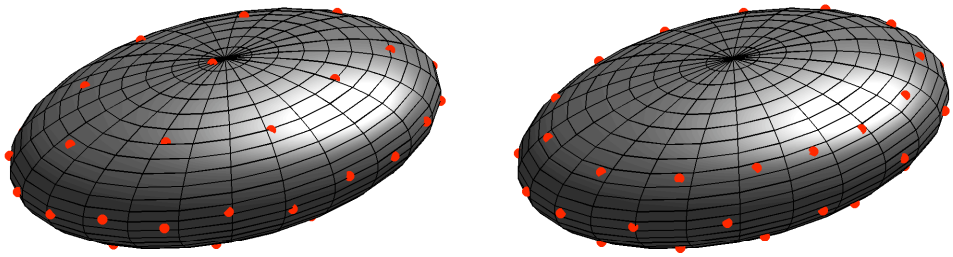

Figure 1: Final Electronic Configuration. The red dots indicate the final position of the electrons as determined by SNOPT. The leftmost plot shows the solution to the problem with equality constraints only while the rightmost plot includes the bound constraints on z. Note how the electrons are confined to the region of the ellipsoid specified in our constraints in the second plot.

6 Solving a Problem Online

Since the student version of AMPL is size-limited, it may be used to solve small-scale problems locally and, importantly, to debug the model and data files. Indeed, upon loading the problem in an AMPL session, as in Listing 3, AMPL will report any syntax errors. But once the model has been debugged, what should we do in order to solve a large-scale instance of it? Of course, a possibility is to obtain the full version of AMPL. This might also mean that we must install the solver of interest locally—a task which is not always simple and often

requires installing many dependencies. Another possibility is to use the NEOS Server for Optimization, a free-of-charge online central server to which we can send a request to solve a given problem modeled in AMPL with any of the solvers available on the server. More information may be found on the NEOS website at neos.mcs.anl.gov. In particular, the list of available solvers is found at neos.mcs.anl.gov/neos/solvers, most of which accept input in the form of AMPL model and data files. Upon clicking on '[AMPL input]' next to the name of any solver, we arrive at the *solver page*. At the bottom of the page is a form which we can use to upload our model, data, and commands files.

The Kestrel tool is a transparent communication channel between the user's local machine and the NEOS Server that acts as a local solver. It forwards an AMPL model to the NEOS Server and thus allows to solve problems without installing solvers locally. It does however require that a full version of AMPL be installed locally because the AMPL model must be decoded before it is sent to a remote solver. With Kestrel, the notion of a *solver* differs slightly from that in the previous section in that, to the eyes of AMPL, the solver is always Kestrel itself. Here, a *solver* is just a program that performs a task with an AMPL model as argument. If we wish to solve our optimization problem with a remote installation of SNOPT, we specify `snopt` as an option to Kestrel. Our commands file takes the form of Listing 7.

Listing 7: Solving the Electrons Problem with Kestrel and a Remote Installation of SNOPT

```
model electrons.mod;
data electrons.dat;
option solver kestrel;                  # Must always be the Kestrel executable
option kestrel_options "solver=snopt";  # Optimization solver is an option to Kestrel
solve;
# After 'solve' has returned, everything happens on the local machine
printf {i in 1..N} '%9.6f  %9.6f  %9.6f\n', x[i], y[i], z[i] > snopt-sol.coords;
```

If it is not possible to obtain the full version of AMPL, a similar effect may be achieved from our local command line by means of the Python interface to the NEOS Server. The PyNeos tool is a gateway to the NEOS Server written in Python, and can be obtained from www.gerad.ca/∼orban/pyneos. A typical call to PyNeos has the form

```
python pyneos.py -m electrons.mod -d electrons.dat -c electrons.ampl -k nco -s snopt
```

The option `-s snopt` specifies the SNOPT solver, which is found in the `nco` category—nonlinear constrained optimization. Upon calling PyNeos with the `--help` command-line option, a list of accepted options is displayed. Those options may be used, e.g., to retrieve the list of available solvers and solver categories.

The main apparent difference between Kestrel and PyNeos is that with PyNeos the user will not be able to output data to disk in the commands file. That is because all commands in the commands file will be executed on a remote machine and the NEOS Server will not return output files. Only standard output—i.e., to the screen—will be returned. All data must thus be output to the screen. The output returned by NEOS must subsequently be processed to extract the data.

It is good practice to debug a model locally and to submit it to NEOS only once all syntax errors have been eliminated. Keep in mind that NEOS is a free service and that users submitting buggy models are merely cluttering up the queue and penalizing the other users.

We end this tutorial with a few statistics. Between January 1, 2009 and September 30, 2009, problems submitted to the NEOS Server were by far mostly in AMPL format with 109231 submissions out of a total 168038 submissions. The next most popular modeling language accounts for less than half of the number of AMPL submissions. By far most requests concerned the nonlinear constrained optimization category with 48879 requests, all modeling languages together. The next most popular category was that of mixed-integer linear programs with 33485 submissions. The most popular submission interface was the Web interface with 128642 submissions followed by the XML-RPC interface, i.e., that used by PyNeos with 25622 submissions, and by the Kestrel interface with 12554 submissions.

7 Where to Find More Information?

The AMPL modeling language has many more features than those described in this document and is regularly updated. Extensive information is available in the literature and on the Internet as well as numerous examples. We enumerate a few below.

(a) The AMPL book [FGK02],
(b) The AMPL website www.ampl.com,
(c) The first chapter of the AMPL book available free of charge www.ampl.com/BOOK/contents.html,
(d) The AMPL discussion group groups.google.com/group/ampl,
(e) Example models from older papers at www.netlib.org/ampl/models,
(f) Robert Vanderbei's fascinating website and numerous AMPL models at www.princeton.edu/∼rvdb/ampl/nlmodels

Note

If you found this tutorial useful and/or if you think crucial information is missing, please do not hesitate to contact the author at dominique.orban@gerad.ca.

The author is not affiliated in any way with any company that commercializes the AMPL software.

References

[DMM04] E. D. Dolan, J. J. Moré, and T. S. Munson. Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Argonne National Laboratory, 2004.

[FGK02] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks / Cole Publishing Company, 2nd edition, 2002.

[GMS05] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithms for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[SK97] E. G. Saaf and A. Kuijlaars. Distributing many points on the sphere. *Mathematics Intelligencer*, 19:5–11, 1997.