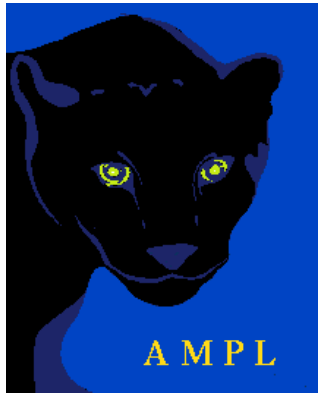
# AMPL

## A Modeling Language for Mathematical Programming




Ladislav Lukšan, Ctirad Matonoha, Jan Vlček

Institute of Computer Science AS CR, Prague

PANM 15, 6. - 11. června 2010, Dolní Maxov


# Outline

1. Introduction to AMPL
2. Maximizing profit
3. Minimizing costs
4. Transportation model
5. AMPL Solvers
6. Rosenbrock function
7. Conclusion


# 1. Introduction to AMPL


# Mathematical Programming

**Programming** – was in use by 1940 to describe the planning or scheduling of activities within a large optimization

**Variable** – programmers found that they could represent the amount or level of each activity as a variable whose value was to be determined

**Constraints** – the restrictions inherent in the planning or scheduling problem as a set of equations or inequalities involving the variables

**Objective** – a function of the variables, such as cost or profit, that could be used to decide whether one one solution was better than another

**Mathematical Programming** – to describe the minimization or maximization of an objective function of many variables, subject to constraints on the variables


# Linear Programming

One special case of mathematical programming

**Linear Program** – all the costs, requirements and other quantities of interest are terms strictly proportional to the levels of the activities, or sums of such terms
Mathematical terminology: the objective is a linear function, and the constraints are linear equations and inequalities

**Linear Programming** – the process of setting up such a problem and solving it

- It is particularly important because a wide variety of problems can be modeled as linear programs
- There are fast and reliable methods for solving linear programs even with thousands of variables and constraints
- The ideas of linear programming are also important for analyzing and solving mathematical programming problems that are not linear


# Large-scale Optimization

The linearity assumption is sometimes too unrealistic

**Nonlinear Program** – if instead some smooth nonlinear functions of the variables are used in the objective or constraints

- Solving such a problem is harder, though in practice not impossibly so
- Computational methods for solving nonlinear programs in many variables were developed in recent decades
- After the success of methods for linear programming

**Large-scale Optimization** – the field of mathematical programming described above

**Integer Programming** – if some variables must take on whole number, or integral, values (the assumptions of linear programming break down, in general becomes much harder)


# The algorithmic method

The full sequence of events is more like this:

1. Formulate a model, the abstract system of variables, objectives, and constraints that represent the general form of the problem to be solved
2. Collect data that define a specific problem instance
3. Generate a specific objective function and constraint equations from the model and data
4. Solve the problem instance by running a program, or solver, to apply an algorithm that finds optimal values of the variables
5. Analyze the results
6. Refine the model and data as necessary, and repeat


# Modeling languages

There are many differences between the form in which human modelers understand a problem and the form in which solver algorithms work with it. Conversion from the *modeler's form* to the *algorithm's form* is consequently a time-consuming, costly, and often error-prone procedure.

A modeling language for mathematical programming

- is designed to express the modeler's form in a way that can serve as direct input to a computer system
- the translation to the algorithm's form can then be performed entirely by computer, without the intermediate stage of computer programming
- it can help to make mathematical programming more economical and reliable
- it is particularly advantageous for development of new models and for documentation of models that are subject to change


# Algebraic modeling language

An *algebraic* modeling language is a popular variety based on the use of traditional mathematical notation to describe objective and constraint functions. An algebraic language provides computer-readable equivalents of notations such as

$$x_j + y_j, \sum_{j=1}^{n} a_{ij} x_j, \; x_j \geq 0, \text{ and } j \in \mathcal{S}$$

that would be familiar to anyone who has studied algebra or calculus.

Advantages of algebraic modeling languages:

- familiarity
- applicability to a particularly wide variety of linear, nonlinear and integer programming models


# About AMPL

- AMPL is an algebraic modeling language for mathematical programming problems subject to constraints
- It was designed and implemented by
  Robert Fourer, David M. Gay, Brian W. Kernigham
  around 1985, and has been evolving ever since
- It is notable for the similarity of its arithmetic expressions to customary algebraic notation
- It offers an interactive command environment for setting up and solving mathematical programming problems
- Its flexible interface enables several solvers to be available at once so a user can switch among solvers and select options that may improve solver performance


# 2. Maximizing profit


# Maximizing profit 1

A steel company must decide how to allocate next week's time on a rolling mill. The mill takes unfinished slabs of steel as input, and can produce either of two semi-finished products, which we will call bands and coils.

- The mill's two products come off the rolling line at different rates:
  Tons per hour: Bands 200, Coils 140
- They also have different profitabilities:
  Profit per ton: Bands $25, Coils $30
- The following weekly production amounts are the most that can be justified in light of the current booked orders:
  Maximum tons: Bands 6.000, Coils 4.000

The question facing the company is as follows:

If 40 hours of production time are available this week, how many tons of bands and how many tons of coils should be produced to bring in the greatest profit?


# Maximizing profit 2

Variables:

- $x_B$ – the number of tons of bands to be produced
- $x_C$ – the number of tons of coils to be produced

The total hours to produce all these tons is given by

(hours to make a ton of bands)*$x_B$ + (hours to make a ton of coils)*$x_C$

It cannot exceed the 40 hours available, so we have a constraint:

$$x_B/200 + x_C/140 \leq 40.$$

There are also production limits:

$$0 \leq x_B \leq 6000, \quad 0 \leq x_C \leq 4000$$

By analogy with the formula for total hours, the total profit must be

(profit per ton of bands)*$x_B$ + (profit per ton of coils)*$x_C$

That is, our objective is to maximize

$$25x_B + 30x_C.$$


# Maximizing profit 3

We have the following linear program:

$$\begin{aligned} \text{Maximize} \quad & 25x_B + 30x_C \\ \text{Subject to} \quad & x_B/200 + x_C/140 \leq 40 \\ & 0 \leq x_B \leq 6000 \\ & 0 \leq x_C \leq 4000 \end{aligned}$$

The solution is

- 6000 tons of bands
- 1400 tons of coils
- profit is $192000


# Maximizing profit 4

Solving this linear program with AMPL can be as simple as typing AMPL's description of the linear program into a file *prod.mod*

```
var XB;
var XC;
maximize Profit:  25 * XB + 30 * XC;
subject to Time:  (1/200) * XB + (1/140) * XC <= 40;
subject to B_limit:  0 <= XB <= 6000;
subject to C_limit:  0 <= XB <= 4000;
```

and the typing a few AMPL commands:

```
ampl:   model prod.mod;
ampl:   solve;
MINOS 5.5:  optimal solution found.
2 iterations, objective 192000
ampl:   display XB, XC;
XB = 6000
XC = 1400
ampl:   quit;
```


# Maximizing profit 5

An improved model: Steel production model *steel.mod*

```
set PROD; # products
param rate {PROD} > 0; # tons produced per hour
param avail >= 0; # hours available in week
param profit {PROD}; # profit per ton
param market {PROD} >= 0; # limit on tons sold in week
var Make {p in PROD} >= 0, <= market[p];
# tons produced
maximize Total_Profit:
   sum {p in PROD} profit[p] * Make[p];
# objective:  total profit from all products
subject to Time:
   sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
# constraints:  total of hours used by all products
# may not exceed hours available
```


# Maximizing profit 6

Data for steel production model *steel1.dat*

```
set PROD := bands coils;
param:     rate  profit  market  :=
    bands   200     25     6000
    coils   140     30     4000   ;
param avail := 40;
```


# Maximizing profit 7

Then a solution can be found and displayed by typing just a few statements:

```
ampl:   model steel.mod;
ampl:   data steel1.dat;
ampl:   solve;
MINOS 5.5:  optimal solution found.
2 iterations, objective 192000
ampl:   display Make;
Make[*] :=
bands 6000
coils 1400
;
```


# Maximizing profit 8

Suppose that we add another product, steel plate. The model stays the same, but in the data we have to add *plate* to the list of members for the set *PROD*, and we have to add a line of parameter values for *plate*:

```
set PROD := bands coils plate;
param:     rate  profit  market  :=
    bands   200     25     6000
    coils   140     30     4000
    plate   160     29     3500   ;
param avail := 40;
```

We put this version of the data in a file *steel2.dat* and use AMPL to get the solution:


# Maximizing profit 9

```
ampl:   model steel.mod;
ampl:   data steel2.dat;
ampl:   solve;
MINOS 5.5:  optimal solution found.
2 iterations, objective 196400
ampl:   display Make;
Make[*] :=
bands 6000
coils 0
plate 1600
;
```

Profits have increased compared to the two-variable version, but now it is best to produce no coils at all! In reality, a whole product line cannot be shut down solely to increase weekly profits. The simplest way to reflect this in the model is to add lower bounds on the production amounts.


# Maximizing profit 10

Lower bounds on production – *steel3.mod*

```
set PROD; # products
param rate {PROD} > 0; # tons produced per hour
param avail >= 0; # hours available in week
param profit {PROD}; # profit per ton
param commit {PROD} >= 0;
# lower limit on tons sold in week
param market {PROD} >= 0;
# upper limit on tons sold in week
var Make {p in PROD} >= commit[p], <= market[p];
# tons produced
maximize Total_Profit:
   sum {p in PROD} profit[p] * Make[p];
# objective:  total profit from all products
subject to Time:
   sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
# constraints:  total of hours used by all products
# may not exceed hours available
```


# Maximizing profit 11

Data for lower bounds on production *steel3.dat*

```
set PROD := bands coils plate;
param:     rate  profit  commit  market  :=
    bands   200     25     1000    6000
    coils   140     30      500    4000
    plate   160     29      750    3500   ;
param avail := 40;
```


# Maximizing profit 12

After these changes are made, we can run AMPL again to get a more realistic solution:

```
ampl:   model steel3.mod;
ampl:   data steel3.dat;
ampl:   solve;
MINOS 5.5:  optimal solution found.
2 iterations, objective 194828.5714
ampl:   display commit, Make, market;
:       commit    Make    market  :=
bands    1000     6000     6000
coils     500      500     4000
plate     750   1028.57    3500
;
```

It is most profitable to produce bands up to the market limit, and then to produce plate with the remaining available time.


# 3. Minimizing costs


# Minimizing costs 1

Consider the problem or choosing prepared foods to meet certain nutritional requirements. Suppose that precooked dinners of the following kinds are available for the following prices per package. These dinners provide the following percentage, per package, of the minimum daily requirements for vitamins A, B1, B2 and C.

| | | | A | B1 | B2 | C |
|---|---|---|---|---|---|---|
| BEEF | beef | $3.19 | 60% | 10% | 15% | 20% |
| CHK | chicken | $2.59 | 8% | 20% | 20% | 0% |
| FISH | fish | $2.29 | 8% | 15% | 10% | 10% |
| HAM | ham | $2.89 | 40% | 35% | 10% | 40% |
| CHS | cheese | $1.89 | 15% | 15% | 15% | 35% |
| MTL | meat loaf | $1.99 | 70% | 15% | 15% | 30% |
| SPG | spaghetti | $1.99 | 25% | 25% | 15% | 50% |
| TUR | turkey | $2.49 | 60% | 15% | 10% | 20% |


# Minimizing costs 2

The problem is to find the cheapest combination of packages that will meet a week's requirements – that is, at least 700% of the daily requirements for each nutrient.
The total cost of the diet will be

$$\begin{aligned} 3.19x_{BEEF} + 2.59x_{CHK} + 2.29x_{FISH} + 2.89x_{HAM} \quad & + \\ 1.89x_{CHS} + 1.99x_{MTL} + 1.99x_{SPG} + 2.49x_{TUR} \quad & \rightarrow \quad \min \end{aligned}$$

The total percentage of the vitamin A requirement is given by a similar formula

$$\begin{aligned} 60x_{BEEF} + 8x_{CHK} + 8x_{FISH} + 40x_{HAM} \quad & + \\ 15x_{CHS} + 70x_{MTL} + 25x_{SPG} + 60x_{TUR} \quad & \geq \quad 700 \end{aligned}$$

The same is for vitamins B1, B2, C.
We can specify a file *diet.mod*.


# Minimizing costs 3

```
var Xbeef >= 0; var Xchk >= 0; var Xfish >= 0;
var Xham >= 0; var Xchs >= 0; var Xmtl >= 0;
var Xspg >= 0; var Xtur >= 0;
minimize cost:
   3.19*Xbeef + 2.59*Xchk + 2.29*Xfish + 2.89*Xham +
   1.89*Xchs + 1.99*Xmtl + 1.99*Xspg + 2.49*Xtur ;
subject to A:
   60*Xbeef + 8*Xchk + 8*Xfish + 40*Xham +
   15*Xchs + 70*Xmtl + 25*Xspg + 60*Xtur >= 700 ;
subject to B1:
   ...
subject to B2:
   ...
subject to C:
   ...
```


# Minimizing costs 4

Again a few AMPL commands then suffice to read the file:

```
ampl:  model diet.mod;
ampl:  solve;
MINOS 5.5:  optimal solution found.
6 iterations, objective 88.2
ampl:  display Xbeef, Xchk, Xfish, Xham, Xchs, Xmtl, Xspg, Xtur;
Xbeef = 0
Xchk = 0
Xfish = 0
Xham = 0
Xchs = 46.6667
Xmtl = -3.69159e-18
Xspg = -4.05347e-16
Xtur = 0
```


# Minimizing costs 5

The optimal solution is found quickly, but it is hardly what we might have hoped for. The cost is minimized by a monotonous diet of $46\frac{2}{3}$ packages of cheese. This neatly provides

$$15\% \cdot 46.6667 = 700\%$$

of the requirement for vitamins A, B1, and B2, and a lot more vitamin C than necessary; the cost is only

$$\$1.89 \cdot 46.6667 = \$88.20$$

Better solution would be generated by requiring the amount of each vitamin to equal $700\%$ exactly. The solution is approximately

- 19.5 packages of chicken
- 16.3 packages of cheese
- 4.3 of meat loaf

But since equalities are more restrictive than inequalities, the cost goes up to $\$89.99$.


# Minimizing costs 6

A general model – file *diet.mod*

```
set NUTR;
set FOOD;
param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];
param amt {NUTR, FOOD} >= 0;
var Buy {j in FOOD} >= f_min[j], <= f_max[j];
minimize Total_Cost:
   sum {j in FOOD} cost[j]*Buy[j];
subject to Diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD}
   amt[i,j]*Buy[j] <= n_max[i];
```

Now we specify appropriate data and solve the problem.


# Minimizing costs 7

Data for diet model – *diet1.dat*

```
set NUTR := A B1 B2 C;
set FOOD := BEEF CHK FISH HAM CHS MTL SPG TUR;
param:    cost  f_min  f_max  :=
    BEEF  3.19    0     100
    CHK   2.59    0     100
    FISH  2.29    0     100
    HAM   2.89    0     100
    CHS   1.89    0     100
    MTL   1.99    0     100
    SPG   1.99    0     100
    TUR   2.49    0     100   ;
```


# Minimizing costs 8

```
param:   n_min   n_max   :=
    A      700    10000
    B1     700    10000
    B2     700    10000
    C      700    10000  ;
param amt (tr):
            A   B1   B2    C  :=
    BEEF   60   10   15   20
    CHK     8   20   20    0
    FISH    8   15   10   10
    HAM    40   35   10   40
    CHS    15   15   15   35
    MTL    70   15   15   30
    SPG    25   25   15   50
    TUR    60   15   10   20  ;
```


# Minimizing costs 9

Then AMPL is used as follows to read these files and to solve the LP:

```
ampl:   model diet.mod;
ampl:   data diet1.dat;
ampl:   solve;
MINOS 5.5:  optimal solution found.
6 iterations, objective 88.2
ampl:   display Buy;
Buy [*] :=
 BEEF   0
  CHK   0
 FISH   0
  HAM   0
  CHS   46.6667
  MTL   -3.69159e-18
  SPG   -4.05347e-16
  TUR   0
;
```


# Minimizing costs 10

Now suppose that we want to make the following enhancements. The weekly diet must contain between 2 and 10 packages of each food. The amount of sodium and calories in each package is also given; total sodium must not exceed 40000 mg, and total calories must be between 16000 and 24000. Data for enhanced diet model – *diet2.dat*

```
set NUTR := A B1 B2 C NA CAL;
set FOOD := BEEF CHK FISH HAM CHS MTL SPG TUR;
param:    cost  f_min  f_max  :=
    BEEF  3.19    2      10
    CHK   2.59    2      10
    FISH  2.29    2      10
    HAM   2.89    2      10
    CHS   1.89    2      10
    MTL   1.99    2      10
    SPG   1.99    2      10
    TUR   2.49    2      10
```


# Minimizing costs 11

```
param:   n_min   n_max   :=
   A       700   20000
   B1      700   20000
   B2      700   20000
   C       700   20000
   NA        0   40000
   CAL   16000   24000  ;
```


# Minimizing costs 12

```
param amt (tr):
            A   B1   B2    C     NA   CAL   :=
    BEEF   60   10   15   20    938   295
    CHK     8   20   20    0   2180   770
    FISH    8   15   10   10    945   440
    HAM    40   35   10   40    278   430
    CHS    15   15   15   35   1182   315
    MTL    70   15   15   30    896   700
    SPG    25   25   15   50   1329   370
    TUR    60   15   10   20   1397   450  ;
```


# Minimizing costs 13

We can run AMPL again:

```
ampl:   model diet.mod;
ampl:   data diet2.dat;
ampl:   solve;
MINOS 5.5:  infeasible problem.
9 iterations
```

The message *infeasible problem* tells us that we have constrained the diet too tightly; there is no way that all of the restrictions can be satisfied. We can look for the source of the infeasibility by displaying some values associated with the solution.


# Minimizing costs 14

```
ampl:   display Diet.lb, Diet.body, Diet.ub;
 :     Diet.lb  Diet.body  Diet.ub  :=
 A         700   1993.00      20000
 B1        700   841.091      20000
 B2        700   601.091      20000
 C         700   1272.55      20000
 CAL     16000   17222.9      24000
 NA          0    40000       40000
```

We can see that the diet returned by the solver does not supply enough vitamin B2, while the amount of sodium has reached its upper bound. There are two obvious choices: we could require less B2 or we could allow more sodium. If we try the latter, a feasible solution becomes possible:


# Minimizing costs 15

```
ampl:   let n_max["NA"] := 50000;
ampl:   solve;
MINOS 5.5:  optimal solution found.
5 iterations, objective 118.0594032
ampl:   display Buy;
Buy [*] :=
 BEEF  5.36061
  CHK  2
 FISH  2
  HAM  10
  CHS  10
  MTL  10
  SPG  9.30605
  TUR  2
;
```


# Minimizing costs 16

One still disappointing aspect of the solution is the need to buy 5.36061 packages of beef and 9.30605 of spaghetti. How can we find the best possible solution in terms of whole packages? Rounding optimal values to whole numbers is not so easy to do in a feasible way. Using AMPL we can observe that it will violate the sodium limit:

```
ampl:   let Buy["BEEF"] := 6;
ampl:   let Buy["SPG"] := 10;
ampl:   display Diet.lb, Diet.body, Diet.ub;
:     Diet.lb  Diet.body  Diet.ub  :=
A         700      2012      20000
B1        700      1060      20000
B2        700       720      20000
C         700      1730      20000
CAL     16000     20240      24000
NA          0     51522      50000
```


# Minimizing costs 17

AMPL does provide for putting the integrality restriction directly into the declaration of the variables:

```
var Buy {j in FOOD} integer >= f_min[j], <= f_max[j];
```

This will only help if we use a solver that can deal with problems whose variables must be integers – e.g. CPLEX.

```
ampl:   reset;
ampl:   model dieti.mod;
ampl:   data diet2a.dat;
ampl:   option solver cplex;
ampl:   solve;
CPLEX 8.0.0:  optimal integer solution; objective 119.3
11 MIP simplex iterations
1 branch-and-bound nodes
```


# Minimizing costs 18

```
ampl:  display Buy;
Buy [*] :=
 BEEF  9
  CHK  2
 FISH  2
  HAM  8
  CHS  10
  MTL  10
  SPG  7
  TUR  2
;
```


# 4. Transportation model


# Transportation model 1

A single good is to be shipped from several origins to several destinations at minimum overall cost. This problem gives rise to the simplest kind of linear program for minimum-cost flows. AMPL offers convenient features for describing network flow models that specify network structure directly.

Suppose that we have decided to produce steel coils at three mill locations, in the following amounts:

| | | |
|---|---|---|
| GARY | Gary, Indiana | 1400 |
| CLEV | Cleveland, Ohio | 2600 |
| PITT | Pittsburgh, Pennsylvania | 2900 |


# Transportation model 2

The total of 6900 tons must be shipped in various amounts to meet orders at seven locations of automobile factories:

| | | |
|---|---|---|
| FRA | Framingham, Massachusetts | 900 |
| DET | Detroit, Michigan | 1200 |
| LAN | Lansing, Michigan | 600 |
| WIN | Windsor, Ontario | 400 |
| STL | St. Louis, Missouri | 1700 |
| FRE | Fremont, California | 1100 |
| LAF | Lafayette, Indiana | 1000 |


# Transportation model 3

We now have an optimization problem: What is the least expensive plan for shipping the coils from mills to plants?
To answer the question, we need a table of shipping costs per ton:

| | GARY | CLEV | PITT |
|---|---|---|---|
| FRA | 39 | 27 | 24 |
| DET | 14 | 9 | 14 |
| LAN | 11 | 12 | 17 |
| WIN | 14 | 9 | 13 |
| STL | 16 | 26 | 28 |
| FRE | 82 | 95 | 99 |
| LAF | 8 | 17 | 02 |


# Transportation model 4

Let **GARY:FRA** be the number of tons to be shipped from *GARY* to *FRA*, and similarly for other city pairs. Then the objective can be written as follows:

```
Minimize
        39*GARY:FRA + 27*CLEV:FRA + 24*PITT:FRA +
        14*GARY:DET + 9*CLEV:DET + 14*PITT:DET +
        11*GARY:LAN + 12*CLEV:LAN + 17*PITT:LAN +
        14*GARY:WIN + 9*CLEV:WIN + 13*PITT:WIN +
        16*GARY:STL + 26*CLEV:STL + 28*PITT:STL +
        82*GARY:FRE + 95*CLEV:FRE + 99*PITT:FRE +
        8*GARY:LAF + 17*CLEV:LAF + 20*PITT:LAF
```

There are 21 decision variables in all.


# Transportation model 5

We need to add constraint that the sum of the shipments from *GARY* to the seven factories is equal to the production level of 1400:

```
GARY:FRA + GARY:DET + GARY:LAN + GARY:WIN +
GARY:STL + GARY:FRE + GARY:LAF = 1400
```

and analogously for the other two mills.
There also have to be constraints like these at the factories, to ensure that the amounts shipped equal the amounts ordered:

```
GARY:FRA + CLEV:FRA + PITT:FRA = 900
```

and similarly for the other six factories.
We have ten constraints in all. If we add the requirement that all variables be nonnegative, we have a complete linear program for the transportation problem.


# Transportation model 6

A general transportation model – *transp.mod*

```
set ORIG; # origins
set DEST; # destinations
param supply {ORIG} >= 0;
# amounts available at origins
param demand {DEST} >= 0;
# amounts required at destinations
check:  sum {i in ORIG}
   supply[i] = sum {j in DEST} demand[j];
# test to issue an error message if it is violated
param cost {ORIG, DEST} >= 0; # shipment costs per unit
var Trans {ORIG, DEST} >= 0; # units to be shipped
minimize Total_Cost:
   sum {i in ORIG, j in DEST} cost[i,j] * Trans[i,j];
subject to Supply {i in ORIG}:
   sum {j in DEST} Trans[i,j] = supply[i];
subject to Demand {j in DEST}:
   sum {i in ORIG} Trans[i,j] = demand[j];
```


# Transportation model 7

Data for transportation model – *transp.dat*

```
# defines set "ORIG" and param "supply"
param:  ORIG:  supply  :=
         GARY    1400
         CLEV    2600
         PITT    2900   ;
# defines set "DEST" and param "demand"
param:  DEST:  demand  :=
         FRA     900
         DET    1200
         LAN     600
         WIN     400
         STL    1700
         FRE    1100
         LAF    1000   ;
```


# Transportation model 8

```
param cost:
          FRA   DET   LAN   WIN   STL   FRE   LAF   :=
    GARY   39    14    11    14    16    82     8
    CLEV   27     9    12     9    26    95    17
    PITT   24    14    17    13    28    99    20   ;
```

Now we can solve the linear program and examine the output.


# Transportation model 9

```
ampl:   model transp.mod;
ampl:   data transp.dat;
ampl:   solve;
CPLEX 8.0.0:  optimal solution; objective 196200
12 dual simplex iterations (0 in phase I)
ampl:   display Trans;
Trans [*,*] (tr)
 :     CLEV  GARY  PITT  :=
 DET   1200     0     0
 FRA      0     0   900
 FRE      0  1100     0
 LAF    400   300   300
 LAN    600     0     0
 STL      0     0  1700
 WIN    400     0     0
;
```


# Transportation model 10

We see that most destinations are supplied from a single mill, but *CLEV, GARY* and *PITT* all to *LAF*.
It is instructive to compare this solution with another solver, SNOPT:


# Transportation model 11

```
ampl:   option solver snopt;
ampl:   solve;
SNOPT 6.1-1:  optimal solution found.
15 iterations, objective 196200
ampl:   display Trans;
Trans [*,*] (tr)
 :     CLEV  GARY  PITT  :=
 DET   1200     0     0
 FRA      0     0   900
 FRE      0  1100     0
 LAF    400     0   600
 LAN    600     0     0
 STL      0   300  1400
 WIN    400     0     0
;
```


# Transportation model 12

The minimum cost is still 196200, but it is achieved in a different way. Alternative optimal solutions such as these are often exhibited by transportation problems, particularly when the coefficients in the objective function are round numbers.


# 5. Solvers


# MINOS

- Bruce A. Murtagh, Michael A. Saunders (Stanford Business Software)
- For mathematical programs that are nonlinear in the objective but linear in the constraints, it employs a reduced gradient approach, which can be viewed as a generalization of the simplex algorithm.
- To deal with nonlinear constraints, MINOS further generalizes its algorithm by means of a projected Lagrangian approach.
- http://www.sbsi-sol-optimize.com/asp/sol_product_minos.htm


# CPLEX

- David M. Gay (IBM)
- Designed to solve integer, mixed-integer, linear programming, and quadratic problems, including problems with quadratic constraints possibly involving integer variables.
- It does not solve general (non-QP) nonlinear programs.
- http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/


# SNOPT

- Philip E. Gill, Walter Murray, Michael A. Saunders (Stanford Business Software)
- It minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs.
- It uses a sequential quadratic programming (SQP) algorithm. Search directions are obtained from QP subproblems that minimize a quadratic model of the Lagrangian function subject to linearized constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point.
- http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm


# LOQO

- Robert J. Vanderbei (Princeton university)
- A system for solving smooth constrained optimization problems.
- The problems can be linear or nonlinear, convex or nonconvex, constrained or unconstrained. The only real restriction is that the functions defining the problem be smooth.
- It is based on an infeasible, primal-dual, interior-point method applied to a sequence of quadratic approximations to the given problem.
- http://www.princeton.edu/~rvdb/


# IPOPT

- Carl Laird, Andreas Waechter (The COmputational INfrastructure for Operations Research)
- A software package for large-scale nonlinear optimization.
- It implements an interior point line search filter method that aims to find a local solution.
- https://projects.coin-or.org/Ipopt


# KNITRO

- Jorge Nocedal et al. (Ziena Optimization, inc.)
- A solver for nonlinear optimization.
- It is designed for large problems with dimensions running into the hundred thousands. It is effective for solving linear, quadratic, and nonlinear smooth optimization problems, both convex and nonconvex. It is also effective for nonlinear regression, problems with complementarity constraints (MPCCs or MPECs), and mixed-integer programming (MIPs), particular convex mixed integer, nonlinear problems (MINLP).
- It provides 3 state-of-the-art algorithms/solvers for solving problems: Interior-point direct algorithm, Interior-point CG algorithm, Active set algorithm.
- http://www.ziena.com/knitro.htm


# DONLP2

- Peter Spellucci (TU Darmstadt)
- Minimization of a (in general nonlinear) differentiable real function subject to (in general nonlinear) inequality and equality constraints.
- The method implemented is a sequential equality constrained quadratic programming method (with an active set technique) with an alternative usage of a fully regularized mixed constrained subproblem in case of nonregular constraints (i.e. linear dependent gradients in the "working set").
- http://www.mathematik.tu-darmstadt.de/fbereiche/numerik/staff/spellucci/DONLP2/


# NPSOL

- Philip E. Gill, Walter Murray, Michael A. Saunders, Margaret H. Wright (Stanford Business Software)
- Software package for solving constrained optimization problems (nonlinear programs).
- It employs a dense SQP algorithm and is especially effective for nonlinear problems whose functions and gradients are expensive to evaluate.
- http://www.sbsi-sol-optimize.com/asp/sol_product_npsol.htm


# PENNON

- Michal Kočvara, Michael Stingl (University of Erlangen)
- Optimization problems with nonlinear objective subject to nonlinear inequalities and equalities as constraints.
- The algorithm is based on a choice of penalty/barrier function that penalizes the inequality constraints and combines ideas of the (exterior) penalty and (interior) barrier methods with the Augmented Lagrangian method.
- http://www2.am.uni-erlangen.de/~kocvara/pennon/


# TRON

- Chih-Jen Lin, Jorge Moré (Argonne National Laboratory)
- A trust region Newton method for the solution of large bound-constrained optimization problems.
- It uses a gradient projection method to generate a Cauchy step, a preconditioned conjugate gradient method with an incomplete Cholesky factorization to generate a direction, and a projected search to compute the step.
- http://www.mcs.anl.gov/~more/tron/index.html


# 6. Rosenbrock function


# Rosenbrock function 1

$$\min_{x \in \mathcal{R}^n} \sum_{i=2}^{n} \left( 100(x_{i-1}^2 - x_i)^2 + (x_{i-1} - 1)^2 \right)$$

where $n = 100$ and initial

$$x^{(0)} = (-1.2, 1.0, -1.2, 1.0, \ldots)^T$$


# Rosenbrock function 2

AMPL code, file *Rosf.mod*:

```
param n := 100;
var x {1..n};
minimize f:
   sum {i in 2..n}
   (100*(x[i-1]^2-x[i])^2 + (x[i-1]-1)^2)
;
let {i in 1..n} x[i] :=
   ( if i mod 2 = 1 then -1.2 else 1.0 );
#option solver minos / loqo / snopt / knitroampl;
option log_file "E:\Ampl\Rosf.out";
solve;
display f;
display x;
```


# Rosenbrock function 3

Typing
`ampl:` **model Rosf.mod;**

yields the results:

| solver | objective | iterations | obj | grad |
|---|---|---|---|---|
| MINOS | 3.986623854 | 647 | 1553 | 1552 |
| LOQO | 3.986623854 | 167 | 352 | - |
| KNITRO | 3.9866238543011 3e+000 | 156 | 182 | 157 |
| SNOPT | 5.453988203e-12 | 867 | 691 | 690 |


# Rosenbrock function 4 - MINOS

```
MINOS 5.5:  optimal solution found.
647 iterations, objective 3.986623854
Nonlin evals:  obj = 1553, grad = 1552.
f = 3.98662
x [*] :=
 1  -0.993286  21  1  41  1  61  1  81  1
 2   0.996651  22  1  42  1  62  1  82  1
 3   0.998330  23  1  43  1  63  1  83  1
 4   0.999168  24  1  44  1  64  1  84  1
 5   0.999585  25  1  45  1  65  1  85  1
...
```


# Rosenbrock function 5 - LOQO

```
LOQO 6.01:  optimal solution (167 iterations, 352
evaluations)
primal objective 3.986623854
dual objective 3.98662378
f = 3.98662
x [*] :=
 1  -0.993286  21  1  41  1  61  1  81  1
 2   0.996651  22  1  42  1  62  1  82  1
 3   0.998330  23  1  43  1  63  1  83  1
 4   0.999168  24  1  44  1  64  1  84  1
 5   0.999585  25  1  45  1  65  1  85  1
...
```


# Rosenbrock function 6 - KNITRO

```
KNITRO 5.2.0:
================================================================
    Academic Ziena License (NOT FOR COMMERCIAL USE)
                        KNITRO 5.2.0
                  Ziena Optimization, Inc.
                   website:  www.ziena.com
                   email:  info@ziena.com
================================================================
The problem is identified as unconstrained.
...
EXIT: Locally optimal solution found.
```

# Rosenbrock function 7 - KNITRO

```
Final Statistics
----------------
 Final objective value               =   3.98662385430113e+000
 Final feas.  error (abs/rel)        =   0.00e+000 / 0.00e+000
 Final opt.  error (abs/rel)         =   6.65e-008 / 6.65e-008
 # of iterations                     =             156
 # of CG iterations                  =              33
 # of function evaluations           =             182
 # of gradient evaluations           =             157
 # of Hessian evaluations            =             156
 Total program time (secs)           =  0.033 (0.031 CPU time)
 Time spent in evals.  (secs)        =           0.012
================================================================
```


# Rosenbrock function 8 - KNITRO

```
Locally optimal solution.
objective 3.986623854; feasibility error 0
156 major iterations; 182 function evaluations
f = 3.98662
x [*] :=
 1  -0.993286  21  1  41  1  61  1  81  1
 2   0.996651  22  1  42  1  62  1  82  1
 3   0.998330  23  1  43  1  63  1  83  1
 4   0.999168  24  1  44  1  64  1  84  1
 5   0.999585  25  1  45  1  65  1  85  1
...
```


# Rosenbrock function 9 - SNOPT

```
SNOPT 6.1-1:  Optimal solution found.
867 iterations, objective 5.453988203e-12
Nonlin evals:  obj = 691, grad = 690.
f = 5.45399e-12
x [*] :=
 1  1  21  1  41  1  61  1  81  1
 2  1  22  1  42  1  62  1  82  1
 3  1  23  1  43  1  63  1  83  1
 4  1  24  1  44  1  64  1  84  1
 5  1  25  1  45  1  65  1  85  1
...
```


# 7. Conclusion


# Conclusion

- AMPL is a language for large-scale optimization and mathematical programming problems in production, distribution, blending, scheduling, and many other applications.
- Combining familiar algebraic notation and a powerful interactive command environment, it makes it easy to create models, use a wide variety of solvers, and examine solutions.
- Though flexible and convenient for rapid prototyping and development of models, it also offers the speed and generality needed for repeated large-scale production runs.
- Free versions of AMPL and several solvers for experiment, evaluation, and education that run on Windows, Unix/Linux, and Mac OS X can be downloaded from

  www.ampl.com

  This web site also lists vendors of the commercial version of AMPL and a variety of solvers.