

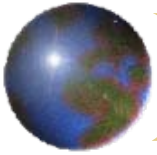
Improving Solver Performance on Tough Optimization Models by Removing Nonlinearities

www.lindo.com

© LINDO Systems

2013

Keywords: Linearization, Piecewise linear, Log transformation, Scaling, Second order cone, Conic program, Bivariate piecewise linear, Global optimization;

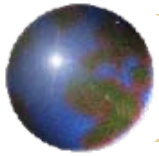


Reducing Solution Time of My Optimization Model

Motivation: We recently received a nonlinear model that was taking **forty+ hours** to solve to optimality.

User: Can you reduce solution times?

Result: Applying a fairly automated technique called linearization, the model was converted to a linear integer model that could be solved in under **fourteen minutes**.



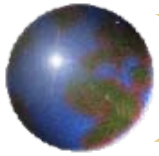
Guidelines / Tricks for (Re)Formulating Nonlinear Models

Q: What is the best way to formulate a nonlinear model?

A: Don't.

Formulate it as a Linear Program (perhaps larger & with integer variables).
Linear programs are an order of magnitude easier to solve
than general nonlinear programs of comparable size.

Let's pursue this, . . .



Reducing Solution Time of My Optimization Model

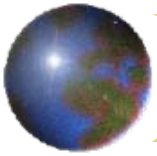
Basic Idea:

Reformulate your model, making it simpler, though perhaps somewhat larger, so it can be solved by a faster solver.

Model solve speed, from fastest to slowest:

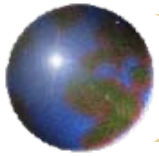
- 1) LP, For vectors a, x : $ax \leq b$;
- 2) QP, Quadratic – convex, LP, and $x'Qx \leq b$;
- 3) Second order cone/Conic: QP, and $x'Qx - u*v \leq -b$; $u, v \geq 0$;
where Q is a positive definite matrix,
- 4) Semi-definite programs: Conic, and X positive definite,
helpful when Q is not positive definite,
- 5) Nonlinear – convex,
- 6) Nonlinear – nonconvex, Need global solver.

For each of the above, adding integer variables makes it an order of magnitude more difficult, especially if not LP.



Model Simplifications, Summary of Useful Tools

- 1) Substitute out fixed variables,
- 2) Disregard extraneous variables and associated nonlinearities,
- 3) Exploit common expressions,
- 4) Linearization of piecewise linear functions such as
 $ABS(x)$, and $MAX(x, y)$,
- 5) Aggregation of interchangeable variables,
- 6) Use the Prayer Algorithm, i.e., speculatively
disregard complicating constraints and pray they are satisfied.
- 7) Variable scaling and change of variable.
- 8) Transform convex polynomial programs to SOC/Conic.
- 9) Transform Geometric programs to (almost) linear programs.
- 10) Transformation to piecewise linear programs.
- 11) Disjunctive/Scenario formulations for integer programs.
- 12) Exploit Semi-Definite Program constraints,
e.g., to choose a correlation matrix.
- 13) Global optimization.



Linearization: Eliminate Fixed Variables

If you have the set of constraints:

$$x_1 = 2;$$

$$x_2/x_1 = 3;$$

$$x_2 * x_3 + x_1 * x_4 \leq 7;$$

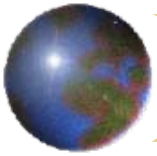
! These look . . . ;

! nonlinear;

We say that x_1 and x_2 are fixed variables and can be substituted out before hand so the above is equivalent to the linear constraint:

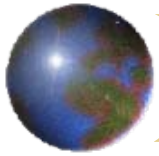
$$6 * x_3 + 2 * x_4 \leq 7;$$

Most optimization solvers will do such substitutions automatically.



Linearization: Eliminate Fixed Variables

```
MODEL: ! Markowitz Value-at-Risk Portfolio Model(PORTVAR);
SETS:
  STOCKS: X, RET;
  COVMAT(STOCKS, STOCKS): VARIANCE;
ENDSETS
DATA:
  STOCKS = ATT GMC USX;
!Covariance matrix and expected returns;
  VARIANCE = .01080754 .01240721 .01307513
             .01240721 .05839170 .05542639
             .01307513 .05542639 .09422681 ;
  RET = 1.0890833 1.213667 1.234583 ;
  STARTW = 1.0; ! How much we start with;
  RHO = .05; ! Choose a Risk tolerance, must be < .5;
ENDDATA
!-----;
! Get the s.d. corresponding to this risk threshold;
  RHO = @PSN( Z); ! @PSN is Normal cdf, left tail probability;
  @FREE( Z);
! Maximize value not at risk, e.g., the probability that
  the portfolio value is < ARET + Z*SD is <= RHO;
  Max = ARET + Z * SD;
! Use exactly 100% of the starting budget;
  @SUM( STOCKS: X) = STARTW;
! The average return;
  ARET = @SUM( STOCKS: X * RET) ;
! Compute the standard deviation of the portfolio;
  @SUM( COVMAT(I, J): X(I) * X(J) * VARIANCE(I, J)) - SD^2 <= 0; ! Conic constraint;
END
```



Eliminate Fixed Variables, Identifying Conic Constraints

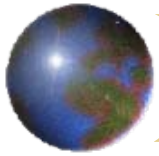
LINGO returns with the message:

```
Model is a second-order cone.  
Global optimal solution found.  
Objective value:      0.9257590
```

Notice that the fixed variable, **Z**, implied by

```
RHO = @PSN( Z) ;
```

was automatically substituted out, so the problem was recognized as a second order cone type problem and efficiently solved by a second order cone algorithm.



Linearization: Eliminate Extraneous Variables

Suppose you have the apparently nonlinear model:

$$\text{Min} = 20 * x_1 + 30 * x_2 ;$$

$$x_1 + x_2 \leq 4 ;$$

$$3 * x_1 + 4 * x_2 \leq 14 ;$$

$$x_3 = x_1 * x_1 ; \quad ! \text{ These not only look nonlinear . . . ;}$$

$$x_4 = x_3 + x_2 * x_2 ; \quad ! \quad \text{they are;}$$

$$x_1, x_2 \geq 0 ;$$

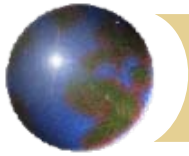
We say that x_3 and x_4 are extraneous or reporting variables.

The user may be interested in their values, however, the constraints that define them do not affect the optimal solution.

These constraints can be removed (temporarily), leaving a linear program.

Once we have a solution in x_3 and x_4 , the deleted constraints can be restored so we can compute and report the values for x_3 and x_4 .

This is a variation of the primal algorithm.



Linearization: Avoid Rationality

Sometimes it is natural to think in terms of ratios.

General rule: Don't. Usually it is good to multiply by denominator.

Example 1: We want a mix of x and y to have 10% protein.

We might have a constraint in natural form:

$$(.16*x + .07*y)/(x+y) = .10;$$

This is nonlinear. It is better to write the linear constraint:

$$.16*x + .07*y = .10*(x+y); \quad ! \text{ LINDO Global will do this};$$

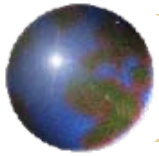
Example 2: More generally we may have a constraint:

$$r = y/x;$$

It is usually better to write it as

$$r*x = y;$$

This avoids possibilities of dividing by zero. Computers have trouble representing ∞ but no trouble with 0.



Linearization: Don't be Rational, off on a Tangent

Sometimes it is not so obvious we are introducing a potential divide by 0.

Suppose one of our expression contains $\tan(x)$.

Now $\tan(x)$ is undefined for $x = \pi/2, 3*\pi/2, \dots$

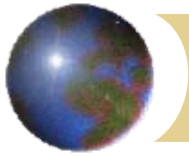
Recall that $\tan(x) = \sin(x)/\cos(x)$.

Thus, given

$$r = \tan(x) = \sin(x)/\cos(x),$$

It may be better to avoid the potential divide by zero by replacing this by:

$$r*\cos(x) = \sin(x);$$

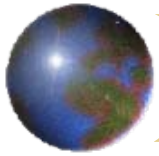


Linearization Using Integer Variables

Replace a (piecewise linear) nonlinear function or expression by linear constraints, and perhaps a collection of additional binary integer variables and such that the modified model is mathematically equivalent to the original.

Some such transformations done automatically by the LINDO API (and LINGO and What'sBest!) include:

<u>Functions</u>	<u>Operators</u>	<u>Expressions</u>
$ABS(x)$	$<$	$x*y$ (where at least one of x and y is a binary 0/1 variable)
$MAX(x, y)$	\leq	$u*v = 0$
$MIN(x, y)$	$<>$	
$IF(b, x, y)$	$=$	
$b \text{ AND } c$	$<$	
$b \text{ OR } c$	\geq	
$NOT \ b$		



Global Optimization with IF(, ,) Function

A small text book example:

A B C D

1 EOQ Inventory with Quantity Discount

2 All Units Case, C and M, Chapter 7

3 Parameters

4 120000 = D = demand/year

5 100 = K = setup cost

6 0.2 = i = interest charge

7 Discount schedule

8 Breakpoint Cost/unit at or above this level

9 0 3

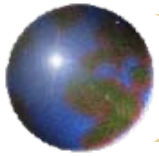
10 5000 2.96

11 10000 2.92

12 10000 = Q = amount to order

*13 Total cost/year = \$354,520.00 = (K*D/Q) + (i*Q/2 + D) * IF(Q < A10, B9, IF(Q < A11, B10, B11))*

The IF statement can be linearized so that a global optimum can be computed automatically using *What'sBest!*.



IF(, ,) Function and its Usefulness

IF(, ,) is convenient for representing quantity discount price schedules, using nested IF's.

A customer example:

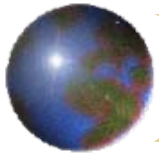
7 discount levels,

13 suppliers,

361 SKU's

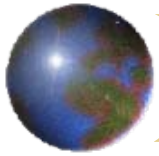
Resulted in model with

4646 rows and 7790 variables.



The sheet as it came from the user....

*=**IF**(D3<'Rebate Structure'!\$A\$3,0,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$4,'Rebate Structure'!D3*'Rebate Calculation'!D3,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$5,'Rebate Structure'!D4*'Rebate Calculation'!D3,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$6,'Rebate Structure'!D5*'Rebate Calculation'!D3,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$7,'Rebate Structure'!D6*'Rebate Calculation'!D3,**IF**(D3<'Rebate Structure'!\$A\$8,'Rebate Structure'!D7*'Rebate Calculation'!D3,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$9,'Rebate Structure'!D8*'Rebate Calculation'!D3,**IF**('Rebate Calculation'!D3<'Rebate Structure'!\$A\$10,'Rebate Structure'!D9*'Rebate Calculation'!D3)))))))))*



Linearization, Methodology, Max, Min, Abs

Some functions can be recognized and linearized exactly.

Let δ be a 0/1 variable. $M =$ a big number.

Given:

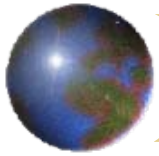
a) $r = \max(x,y);$

The Linearization is to replace $r = \max(x,y)$, by:

$$\begin{aligned} x \leq r \leq x + \delta M; & \quad ! \delta = 0 \text{ implies } y \leq x; \\ y \leq r \leq y + (1 - \delta)M; & \quad ! \delta = 1 \text{ implies } x \leq y; \end{aligned}$$

b) $r = \text{abs}(x) = \max(x, -x);$

c) $r = \min(x,y) = -\max(-x, -y);$



Linearization continued, Products and IF().

d) $r = \delta y$; !Replace by the following linear constraints;

$$y - (1 - \delta) M \leq r \leq y + (1 - \delta) M; \quad ! \delta = 1 \text{ implies } r = y;$$

$$r \leq \delta M; \quad ! \delta = 0 \text{ implies } r = 0;$$

e) $x y = 0$; (Complementarity)

$$-(1 - \delta) M \leq x \leq (1 - \delta) M;$$

$$-\delta M \leq y \leq \delta M;$$

!Replace by ;

$$! \delta = 1 \text{ implies } x = 0;$$

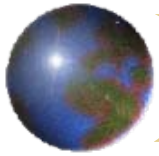
$$! \delta = 0 \text{ implies } y = 0;$$

f) $r = IF(\delta, x, y)$;

!Replace by ;

$$x - (1 - \delta) M \leq r \leq x + (1 - \delta) M; \quad ! \delta = 1 \text{ implies } r = x;$$

$$y - \delta M \leq r \leq y + \delta M; \quad ! \delta = 0 \text{ implies } r = y;$$



Linearization continued (Product with General Integer).

g) $r = x * y$; where,

$x = 0, 1, 2, \dots, U$; i.e., a general integer variable.

Introduce binary variables, $\delta_1, \delta_2, \dots$, and add constraints:

$$x = \delta_1 + 2 * \delta_2 + 4 * \delta_3 + 8 * \delta_4 + \dots;$$

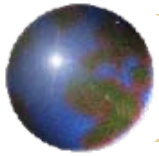
$$r = \rho_1 + 2 * \rho_2 + 4 * \rho_3 + 8 * \rho_4 + \dots;$$

Apply the previous linearization of the products $\rho_i = \delta_i * y$, i.e.,

$$y - (1 - \delta_i) M \leq \rho_i \leq y + (1 - \delta_i) M; \quad ! \delta_i = 1 \text{ implies } \rho_i = y;$$

$$\rho_i \leq \delta_i M; \quad ! \delta_i = 0 \text{ implies } \rho_i = 0.$$

E.g., $\delta_1 = \delta_3 = 1, \delta_2 = \delta_4 = 0$, corresponds to $x = 5, r = 5 * y$;



Linearization continued, Function of Integer Variable.

h) $r = f(x)$, where,

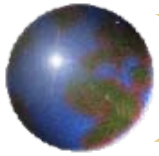
$$x = 0, 1, 2, \dots, U;$$

Introduce binary variables, $\delta_0, \delta_1, \delta_2, \dots, \delta_U$, and add constraints:

$$x = \delta_1 + 2 * \delta_2 + 3 * \delta_3 + 4 * \delta_4 + \dots + U * \delta_U;$$

$$r = \delta_0 * f(0) + \delta_1 * f(1) + \delta_2 * f(2) + \delta_3 * f(3) + \delta_4 * f(4) + \dots + \delta_U * f(U)$$

$$1 = \delta_0 + \delta_1 + \delta_2 + \delta_3 + \delta_4 + \dots + \delta_U;$$



Linearization with a Single Complicating Variable

Some models have the feature that there is a

Single variable, e.g., a batch size or cycle length,
such that if fixed,

then the model is linear. (E.g., benchmarking with Data Envelopment Analysis)

Slightly more specifically, there is a variable $s (> 0)$, so model can be written:

Constraints for i in set A : $s(a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \dots) = b_i$;

Constraints for i in set B : $a_{i1}x_1 + a_{i2}x_2 + \dots = b_i$;

Linearization:

1) Multiply or scale the constraints in set B by s , giving:

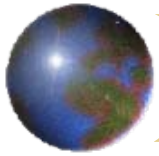
$$a_{i1}x_1s + a_{i2}x_2s + \dots - b_i s = 0;$$

2) Do a change of variable, $y_j = x_j s$, giving the linear set:

$$A: a_{i0}s + a_{i1}y_1 + a_{i2}y_2 + \dots = b_i;$$

$$B: a_{i1}y_1 + a_{i2}y_2 + \dots - b_i s = 0;$$

3) Solve LP, then do (nonlinear) post processing to original variables with $x_j = y_j/s$;



Linearization of Ratio or Fractional Objective, cont.

The idea also works for fractions, e.g.

set $A: (a_{i0} + a_{i1}x_1 + a_{i2}x_2 + \dots)/t = b_i; (t > 0)$ Define $s = 1/t$.

Rescale all constraints by $s = 1/t$;

Original

Scale by $s = 1/t$

Change of variable $y_i = x_i s$

Min $(c_1x_1 + c_2x_2 + \dots) / t$;

Min $c_1x_1s + c_2x_2s + \dots$;

Min $(c_1y_1 + c_2y_2 + \dots)$;

s.t.

s.t. .

s.t.

$a_{11}x_1 + a_{12}x_2 + \dots \geq b_1$;

$a_{11}x_1s + a_{12}x_2s + \dots \geq b_1s$;

$a_{11}y_1 + a_{12}y_2 + \dots - b_1s \geq 0$;

etc. **etc.** etc.

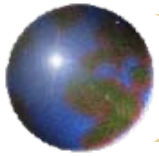
$t + \dots \geq d$;

$ts + \dots \geq ds$;

$1 + \dots \geq ds$;

**Need last constraint
to avoid divide by 0.**

**The above model
is linear.**



Linearization of Multiple Ratios is Sometime Possible

A firm wants to allocate personnel to several locations* to several different task types**.

x_{ij} = people at location i assigned task j ;

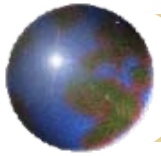
Nonlinear complication: for administrative reasons we have the constraint for any two locations i and k : $x_{ij}/x_{ij+1} = x_{kj}/x_{kj+1}$;

	Task type				
<u>Locn</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	
1	3	12	9	6	(= 30)
2	1	4	3	2	(= 10)
3	2	8	6	4	(= 20)

This looks like a nonlinear constraint, however, for a given location i , $\sum_j x_{ij} = t_i$, where t_i is a given constant.

* call center locations;

** call types, e.g., tech support, sales, etc.



Linearization of Multiple Ratios is Sometime Possible-II

How to linearize. Define the variable :

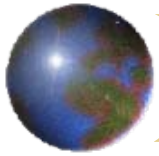
f_j = fraction of staff allocated to task j in every location;

	Task type				
<u>Locn</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	
1	6	3	12	9	(= 30)
2	2	1	4	3	(= 10)
3	4	2	8	6	(= 20)

Replace $x_{ij}/x_{ij+1} = x_{kj}/x_{kj+1}$ by the linear constraints:

$$x_{ij} = f_j * k_i;$$

$$\sum_j f_j = 1;$$



Linearization in Quadratic Models

Sometimes variable scaling can be used to convert an apparently nonlinear model to a quadratic model. Consider the well known Sharpe Ratio maximization model:

MODEL:

```
R0 = 1.05; ! The risk free rate;
```

```
! Maximize the Sharpe ratio;
```

```
MAX = RDIF/SD;
```

```
! Expected return of the portfolio;
```

```
RP = 1.089083*ATT + 1.213667*GMC + 1.234583*USX;
```

```
RDIF = RP - R0; ! Excess expected return;
```

```
! Variance of the portfolio;
```

```
SD*SD >=
```

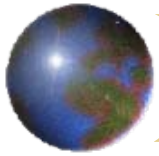
```
    .01080754 * ATT * ATT + .01240721 * ATT * GMC + .01307513 * ATT * USX  
+ .01240721 * GMC * ATT + .05839170 * GMC * GMC + .05542639 * GMC * USX  
+ .01307513 * USX * ATT + .05542639 * USX * GMC + .09422681 * USX * USX;
```

```
! Use exactly 100% of the starting budget;
```

```
[BUD] ATT + GMC + USX = 1;
```

```
END
```

This is a quadratic(SOC/Conic) model except for the ratio in **MAX** = RDIF/SD;



Linearization in Quadratic Models, II

There is just one variable, $1/SD$, that makes it general nonlinear...;

MODEL:

! Converting an apparently general nonlinear model with a ratio, to a quadratic model. We want to MAX = RDIF/SD;

! Steps in the conversion:

1) Define: $SC = 1/SD$, so $SC*SD = 1$.

2) Scale, i.e., multiply, various constraints by either SC or SC*SC.

3) Do a change of variable $XSC = X*SC$, giving the quadratic model;

R0 = 1.05; ! The risk free rate;

! Maximize the Sharpe ratio;

MAX = RDIFSC; ! = RDIF*SC;

! Expected return of the portfolio;

RPSC = 1.089083*ATTSC + 1.213667*GMCSC + 1.234583*USXSC;

RDIFSC = RPSC - R0*SC; ! Excess expected return;

! Variance of the portfolio;

1 >= ! (SD*SC) * (SD*SC) >= ...;

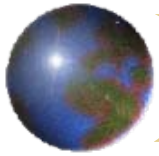
.01080754 * ATTSC * ATTSC + .01240721 * ATTSC * GMCSC + .01307513 * ATTSC * USXSC
+.01240721 * GMCSC * ATTSC + .05839170 * GMCSC * GMCSC + .05542639 * GMCSC * USXSC
+.01307513 * USXSC * ATTSC + .05542639 * USXSC * GMCSC + .09422681 * USXSC *

USXSC;

! Use exactly 100% of the starting budget;

[BUD] ATTSC + GMCSC + USXSC = 1*SC;

END



Linearization in Quadratic Models, III

Solution to first version

Global optimal solution found.

Objective value: 0.6933179

Model Class: NLP

Variable	Value
RDIF	0.151793
SD	0.218937
RP	1.201793
ATT	0.131855
GMC	0.650475
USX	0.217670

Solution to second version

Global optimal solution found.

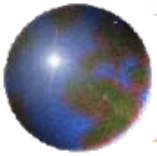
Objective value: 0.6933179

Model Class: CONIC

Variable	Value
RDIFSC	0.693318
SC	4.567532
RPSC	5.489226
ATTSC	0.602254
GMCS	2.971065
USXSC	0.994212

Post-processing the solution to the second version, notice that

$$\begin{aligned}
 0.151793 &= 0.693318 / 4.567532 \\
 0.218937 &= 1 / 4.567532 \\
 1.201793 &= 5.489226 / 4.567532 \\
 0.131855 &= 0.602254 / 4.567532 \\
 0.650475 &= 2.971065 / 4.567532 \\
 0.217670 &= 0.994212 / 4.567532
 \end{aligned}$$



Geometric Program/Logarithmic Transformations

Example:

! Box shape design Geometric Program in LINGO.

Ref: Boyd, Kim, Vandenberghe, and Hassibi,
"A tutorial on Geometric Programming" ;

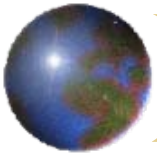
```
Max = h*w*d;      ! Maximize volume;
w*d <= 90;        ! Floor area limit;
.8 <= h/w; h/w <= 1.1; ! Aesthetic limits;
1.1 <= d/w; d/w <= 1.2;

2*(h*w + h*d) <= 150; ! Wall area not too great;
```

This is an NLP and it is not convex.

The objective and all constraints but the last are called monomials.

The last constraint is called a posynomial
(positively weighted sum of monomials).



Geometric Program/Logarithmic Transformations

```
! In log transformation form,  $t_h = \log(h)$ , etc.;
! Max =  $h*w*d$ ;    ! Maximize volume;
Max =  $t_h + t_w + t_d$ ;

!     $w*d \leq 90$ ;    ! Floor area limit;
     $t_w + t_d \leq @LOG(90)$ ;

! Aesthetic limits;
!     $.8 \leq h/w$ ; ! $h/w \leq 1.1$ ;
     $@LOG(.8) \leq t_h - t_w$ ;  $t_h - t_w \leq @LOG(1.1)$ ;

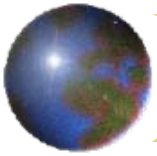
!     $1.1 \leq d/w$ ; ! $d/w \leq 1.2$ ;
     $@LOG(1.1) \leq t_d - t_w$ ;  $t_d - t_w \leq @LOG(1.2)$ ;

!     $2*(h*w + h*d) \leq 150$ ; ! Wall area not too great;
     $@EXP(@LOG(2) + t_h + t_w) + @EXP(@LOG(2) + t_h + t_d) \leq 150$ ;

! With the understanding that
     $t_h = @LOG(h)$  or;  $h = @EXP(t_h)$ ;
    !  $t_w = @LOG(w)$  or;  $w = @EXP(t_w)$ ;
    !  $t_d = @LOG(d)$  or;  $d = @EXP(t_d)$ ;
```

The objective and first constraints are linear.

The last constraint is convex, so a local optimum is a global optimum.



Logarithmic Transformation and Scaling

! Chemical equilibrium problem of Peters, Hayes and Hieftje. Calculate concentrations of various components of phosphoric acid, H_3PO_4 , with a pH of 8 and total phosphate concentration of .10. The equilibrium equations in obvious form have the following poorly scaled form:

```
! H2P * H/ H3P = .0075;
! HP*H/H2P = 6.2 * 10^-8);
! H*P/HP = 4.8*10^-13;
! H = 10 ^-8;
! H3P + H2P + HP + P = .1;
```

! The very small numbers make this a difficult problem to solve with default tolerances.

Doing a log transformation helps the scaling, + makes it almost linear;

```
LH2P + LH - LH3P = @LOG( .0075);
LHP + LH - LH2P = @LOG( 6.2 * 10^-8);
LH + LP - LHP = @LOG( 4.8 * 10^-13);
LH = @LOG( 10 ^-8);
```

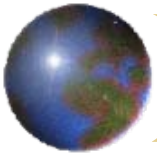
! Convert back to original variables to represent the last (posynomial) constraint;

```
H = @EXP( LH);
P = @EXP( LP);
HP = @EXP( LHP);
H2P = @EXP( LH2P);
H3P = @EXP( LH3P);
H3P + H2P + HP + P = .1;
```

! Must unconstrain log variables if original variable could be fractional, so log could be < 0;

```
@FREE( LH2P); @FREE( LH); @FREE( LH3P);
@FREE( LHP); @FREE( LP);
```

! Solution should be: LH2P= -4.2767, LH= -18.4207,
LH3P= -17.8045, LHP= -2.4522, LP= -12.3965;



Geometric Program Summary

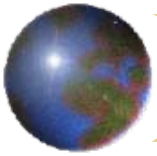
A Pure Geometric program, has the features,

Every constraint is either:

- 1) a monomial, or
- 2) a " \leq " constraint with the LHS a posynomial, i.e., a positive weighted sum of monomials, and

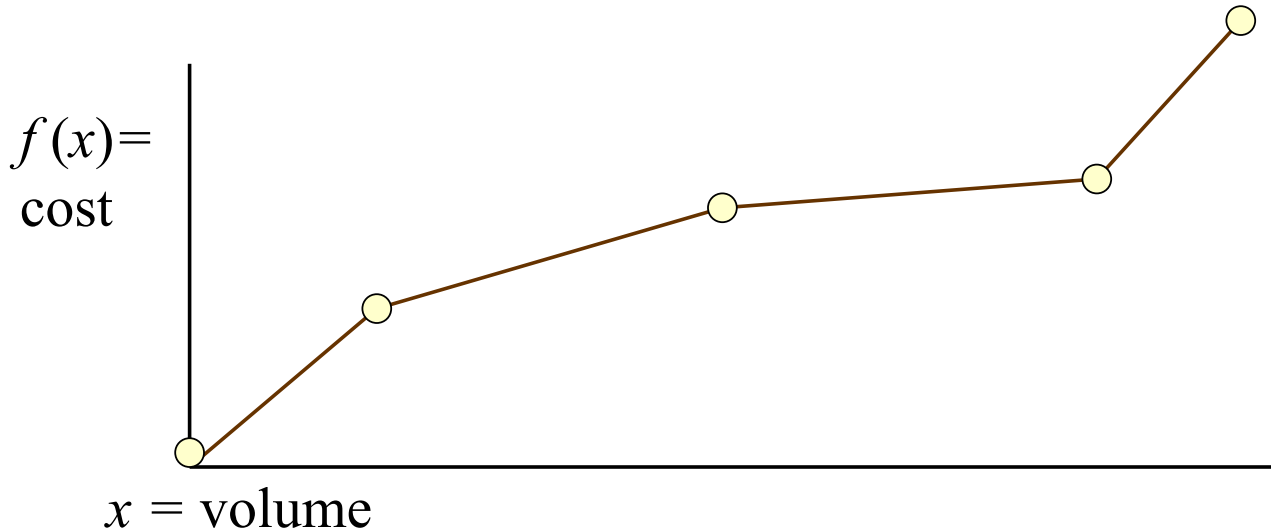
The objective is either:

- 1) a monomial, or
- 2) Minimize a posynomial, i.e., a positive weighted sum of monomials;



Piecewise Linear Functions of One Variable

In some situations a cost function has no elegant mathematical form. Perhaps the most obvious example is the quantity discount schedule from a supplier. How to represent it?



Basic idea: $w_i =$ weight on point i ,

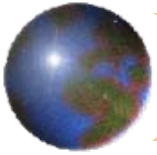
$$x = w_i x_i + w_{i+1} x_{i+1};$$

$$f(x) = w_i f(x_i) + w_{i+1} f(x_{i+1});$$

$$1 = w_i + w_{i+1}; \quad w_i \geq 0;$$

Tricky part:

Must choose 2 adjacent points.

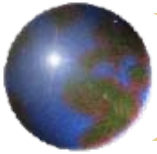


Piecewise Linear, Choosing 2 Adjacent Points, SOS2

A standard feature in Math Programming systems for enforcing the “choose 2 adjacent points” condition is the SOS2 (Special Ordered Set, type 2) feature.

LINGO 12 and later versions have a “qualifier”, SOS2,
What'sBest! 10 and later have a “qualifier” WBSOS2(),

to specify SOS2 ordered sets.



Piecewise Linear, SOS2 in LINGO

MODEL:

! Demonstrates the SOS2 feature of LINGO for representing arbitrary, piecewise-linear cost curves.

We have two suppliers. Each with his own piecewise-linear cost curve.

How much should we buy from each to buy a given total amount? ;

SETS:

! Two suppliers with the price schedules;

POINT1: W1, VOLP1, COSTP1;

POINT2: W2, VOLP2, COSTP2;

ENDSETS

DATA:

! Supplier 1;

! Volume at each breakpoint;

VOLP1 = 0 5 12 20;

! Total cost at each breakpoint;

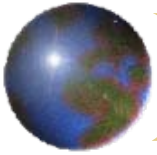
COSTP1 = 0 8 35 55;

! Supplier 2, ditto..;

VOLP2 = 0 4 12 19 24;

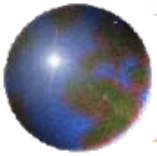
COSTP2 = 0 10 36 50 51;

ENDDATA



Piecewise Linear with SOS2

```
MIN = COST1 + COST2;  
  ! Volume we need;  
    X1 + X2 >= 18;  
  
! Supplier 1.;! calculate weighted cost;  
COST1 = @SUM( POINT1( i): COSTP1( i) * W1( i));  
  
! Calculate weighted volume;  
X1 = @SUM( POINT1( i): VOLP1( i) * W1( i));  
  
! Weights must sum to 1;  
  1 = @SUM( POINT1( i): W1( i));  
  
! Weights must satisfy SOS2 condition:  
  at most 2 weights > 0, must be adjacent;  
@FOR( POINT1( i): @SOS2( 'SOS2_1', W1( i)));  
  
! Same for Supplier 2;  
COST2 = @SUM( POINT2( i): COSTP2( i) * W2( i));  
  X2 = @SUM( POINT2( i): VOLP2( i) * W2( i));  
  1 = @SUM( POINT2( i): W2( i) );  
@FOR( POINT2( i): @SOS2( 'SOS2_2', W2( i)));  
END
```



Piecewise Linear with SOS2, Solution

Global optimal solution found.

Objective value: 46.00000

Variable	Value
COST1	8.000000
COST2	38.000000
X1	5.000000
X2	13.000000
W1 (1)	0.000000
W1 (2)	1.000000
W2 (3)	0.857143
W2 (4)	0.142857

Recall:

! Volume at each breakpoint;

VOLP1 = 0 5 12 20;

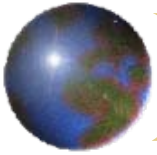
! Total cost at each breakpoint;

COSTP1 = 0 8 35 55;

! Supplier 2, ditto..;

VOLP2 = 0 4 12 19 24;

COSTP2 = 0 10 36 50 51;



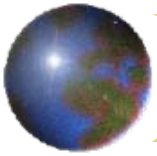
Piecewise Linear, SOS2 with Unbounded Final Interval

```
MODEL:      ! (PieceLinUbd);
! Demonstrates the SOS2 feature of LINGO for representing
arbitrary, piecewise-linear cost curves, where the
final interval has no upper bound.
We have two suppliers. Each with his
own piecewise-linear cost curve.
The highest segment for each supplier has unbounded range.
How much should we buy from each to
buy a given total amount? ;
```

```
SETS:
! Two suppliers with the price schedules;
POINT1: W1, VOLP1, COSTP1;
POINT2: W2, VOLP2, COSTP2;
ENDSETS
```

```
DATA:
! Supplier 1;
! Volume at each breakpoint, except at last point
it is the cost per additional unit for unlimited amounts;
VOLP1 = 0 5 12 20 1 ;
! Total cost at each breakpoint;
COSTP1 = 0 8 35 55 2.10;

! Supplier 2, ditto..;
VOLP2 = 0 4 12 19 24 1;
COSTP2 = 0 10 36 50 51 2.20;
ENDDATA
```



Piecewise Linear, SOS2 with Unbounded Final Interval

```
! Get number of points + final ray for each supplier;
  N1 = @SIZE(POINT1);
  N2 = @SIZE(POINT2);

MIN = COST1 + COST2;
! Volume we need;
  X1 + X2 >= 40;

! Supplier 1..; ! calculate weighted cost;
COST1 = @SUM( POINT1( i): COSTP1( i) * W1( i));

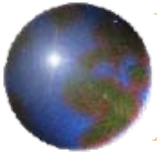
! Calculate weighted volume;
  X1 = @SUM( POINT1( i): VOLP1( i) * W1( i));

! Weights must sum to 1, but final ray is not included;
  1 = @SUM( POINT1( i) | i #LT# N1: W1( i));

! Weights must satisfy SOS2 condition:
  at most 2 weights > 0, must be adjacent;
@FOR( POINT1( i): @SOS2( 'SOS2_1', W1( i)));

! Same for Supplier 2;
COST2 = @SUM( POINT2( i): COSTP2( i) * W2( i));
  X2 = @SUM( POINT2( i): VOLP2( i) * W2( i));
  1 = @SUM( POINT2( i) | i #LT# N2: W2( i) );
@FOR( POINT2( i): @SOS2( 'SOS2_2', W2( i)));

END
```



Piecewise Linear, SOS2 with Unbounded Final Interval

Global optimal solution found.

Objective value: 83.20000

Variable	Value
COST1	8.000000
COST2	75.20000
X1	5.000000
X2	35.00000
W1(2)	1.000000
W2(5)	1.000000
W2(6)	11.00000

Recall:

! Volume at each breakpoint;

VOLP1 = 0 5 12 20 1;

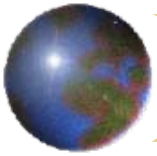
! Total cost at each breakpoint;

COSTP1 = 0 8 35 55 2.10;

! Supplier 2, ditto..;

VOLP2 = 0 4 12 19 24 1;

COSTP2 = 0 10 36 50 51 2.20;



Piecewise Linear Approximations to Multivariate Functions

Typical applications:

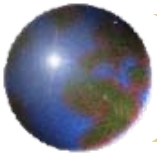
Hydro-electric power generation

Flow through a pipe,

Pooling problems in petroleum industry,

Tank models, saturation and flow rate in chemical industry,

Quantity discounts in purchasing.

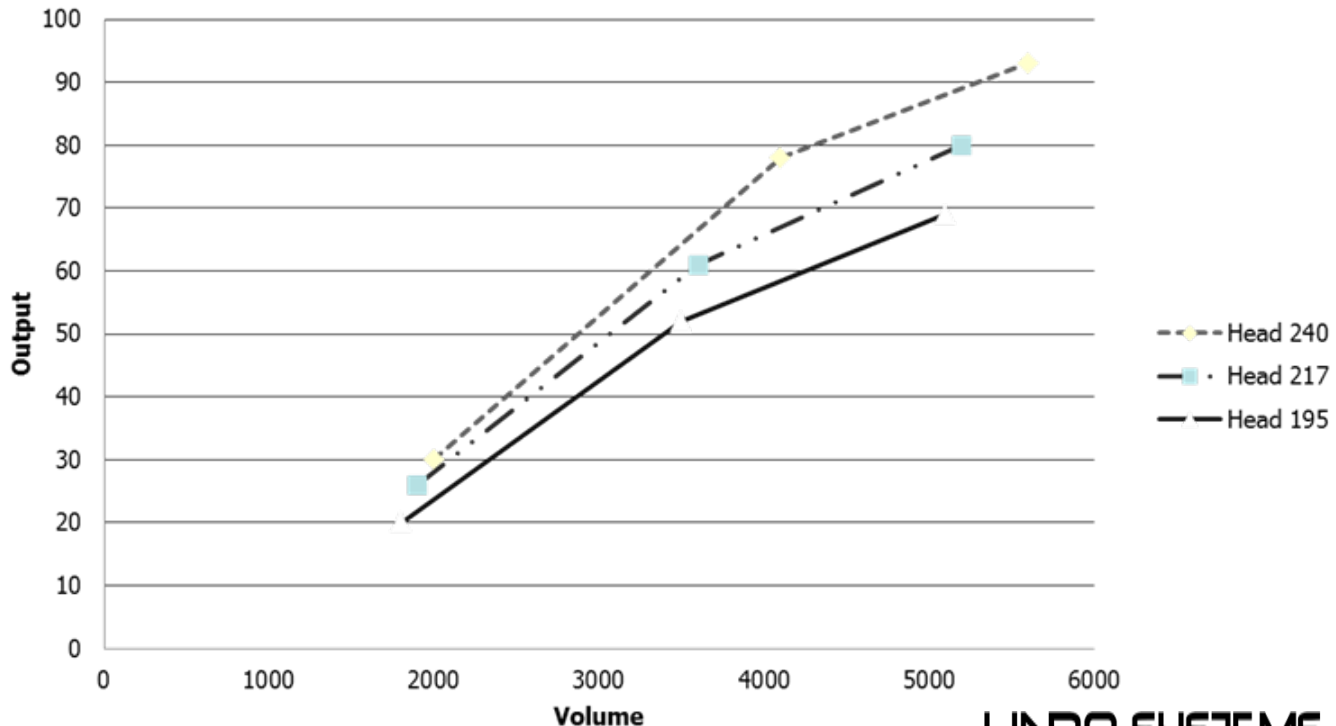


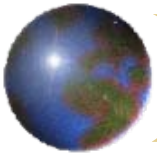
Piecewise Linear Approximations to Multivariate Functions

Can we extend the piecewise linear interpolation method to functions of 2 variables?

Example: power output from a hydro generator is a function of two variables: 1) head or pressure, and 2) flow volume.

Output vs. Head & Volume





Piecewise Linear Approximations to Multivariate Functions

Suppose we have a function of two variables:

$output = f(p, v)$, e.g., pressure and volume for water through a hydro generator.

We can construct a piecewise linear approximation to this function if we

choose n points, $(pbar_i, vbar_i)$ for $i = 1, 2, \dots, n$, e.g., corner points of the 8 triangles in the figure below, and introduce the n ($3 \times 3 = 9$) nonnegative variables, w_i and add the constraints:

$$1 = \sum_i w_i,$$

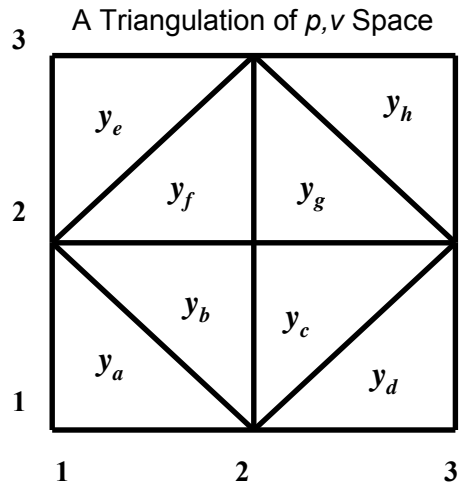
$$output = \sum_i w_i * f(pbar_i, vbar_i),$$

$$p = \sum_i w_i * pbar_i,$$

$$v = \sum_i w_i * vbar_i.$$

If we are lucky, e.g., $f(p, v)$ is convex in the appropriate way then:

- a) at most three of the w_i will be nonzero, and
- b) the nonzero w_i will correspond to adjacent points, that is, corner points of a triangle containing no other points.



How can we enforce the single triangle requirement?

Define a 0/1 variable for each triangle. Add constraints:

Must choose exactly one triangle:

$$y_a + y_b + \dots + y_h = 1;$$

If we put any weight on w_{ij} , we must choose one of its triangles:

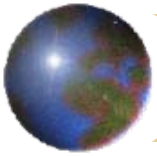
$$w_{11} \leq y_a;$$

$$w_{12} \leq y_a + y_b + y_c + y_d;$$

$$w_{13} \leq y_d;$$

$$w_{21} \leq y_a + y_b + y_f + y_e;$$

etc.



Piecewise Linear Approximations to Multivariate Functions, Revisited...

Looks like we need one 0/1 variable for each triangle. This could be a lot of 0/1 variables. Can we do any better?

Yes, can get by with approximately $\log_2(\text{number of triangles})$ 0/1 variables..

Use a binary partition approach. See figure below. Define:

$y_H = 1$ if choose triangle above horizontal split (red line),

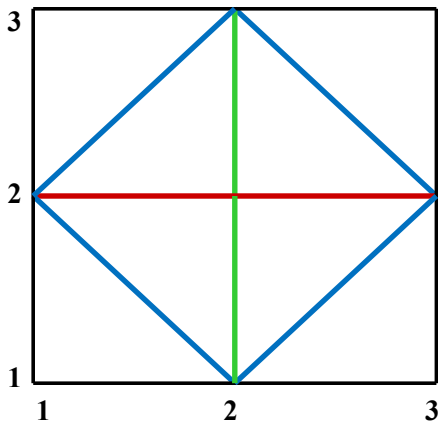
$y_V = 1$ if choose triangle to right of vertical split (green line),

$y_C = 1$ if choose triangle within center diamond (blue diamond), else 0;

Still require: $w_{11} + w_{12} + w_{13} + w_{21} + w_{22} + w_{23} + w_{31} + w_{32} + w_{33} = 1$;

If we put any weight on w_{ij} , then the relevant partition variables must take the correct 0 or 1 value:

A Triangulation of p, v Space



$$w_{31} + w_{32} + w_{33} \leq y_H;$$

$$w_{11} + w_{21} + w_{31} \leq 1 - y_V;$$

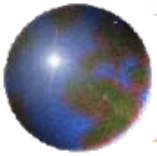
$$w_{13} + w_{23} + w_{33} \leq y_V;$$

$$w_{22} \leq y_C;$$

$$w_{11} + w_{13} + w_{31} + w_{33} \leq 1 - y_C;$$

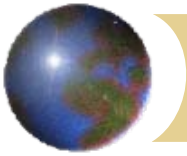
$$w_{11} + w_{12} + w_{13} \leq 1 - y_H;$$

Summary: 0/1 vars: $8 \rightarrow 3$; Constraints: $9 \rightarrow 6$;



Piecewise Linear Bivariate Functions, in LINGO

```
!Piecewise linear approximation of f(h,v);
SETS:
  HSET; ! The horizontal dimension;
  VSET; ! The vertical dimension;
  HXV( VSET, HSET): HV, VV, FV, WGT;
ENDSETS
DATA:
  HSET = 1..3;
  VSET = 1..3;
! Matrix of Data points for head(pressure), volume
  and power output(function value) for a hydro-electric generator;
  HV,  VV,  FV =
  195 1800 20 ! Row 1;
  195 3500 52
  195 5100 69
  217 1900 26 ! Row 2;
  217 3600 61
  217 5200 80
  240 2000 30 ! Row 3;
  240 4100 78
  240 5600 93 ;
ENDDATA
```



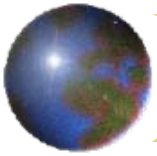
Piecewise Linear Bivariate Functions, in LINGO - II

```
! Weights must form a convex combination;
[CNVX] @SUM( HXV(i,j): WGT(i,j) ) = 1;
! Compute associated horizontal or X value;
[COMPX] @SUM( HXV(i,j): HV(i,j)* WGT(i,j) ) = HA;
! Compute associated vertical or Y value;
[COMPY] @SUM( HXV(i,j): VV(i,j)* WGT(i,j) ) = VA;
! Compute associated function value;
[COMPFV] @SUM( HXV(i,j): FV(i,j)* WGT(i,j) ) = FA;

! Choice must be binary, all or nothing;
@BIN(YH); @BIN(YV); @BIN(YC);

! Implications of setting YV;
WGT(1,3) + WGT(2,3) + WGT(3,3) <= YV;
WGT(1,1) + WGT(2,1) + WGT(3,1) <= 1-YV;
! Implications of setting YH;
WGT(3,1) + WGT(3,2) + WGT(3,3) <= YH;
WGT(1,1) + WGT(1,2) + WGT(1,3) <= 1-YH;
! Implications of setting YC;
WGT(2,2) <= YC;
WGT(1,1) + WGT(3,1) + WGT(1,3) + WGT(3,3) <= 1-YC;

! Define an arbitrary optimization problem to illustrate the method;
! An arbitrary tradeoff cost between head and volume;
[ARBOBJ] Min = VA + 15*HA;
! We need this much power;
[ARB1] FA >= 75;
! Arbitrarily put some constraints on Head;
[ARB2] HA <= 229;
[ARB3] HA >= 227;
```



Piecewise Linear Bivariate Functions, Solution

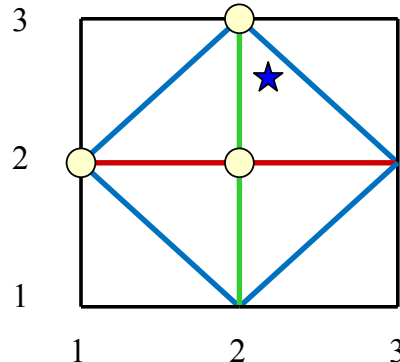
Global optimal solution found.

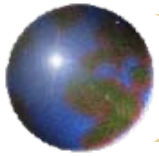
Objective value: 7727.906

Variable	Value
HA	229.000000
VA	4292.906000
FA	75.000000
YH	1.000000
YV	1.000000
YC	1.000000
WGT(2, 2)	0.208238
WGT(2, 3)	0.270023
WGT(3, 2)	0.521739

Recall:

HV,	VV,	FV =	
195	1800	20	! Row 1;
195	3500	52	
195	5100	69	
217	1900	26	! Row 2;
217	3600	61	
217	5200	80	
240	2000	30	! Row 3;
240	4100	78	
240	5600	93	





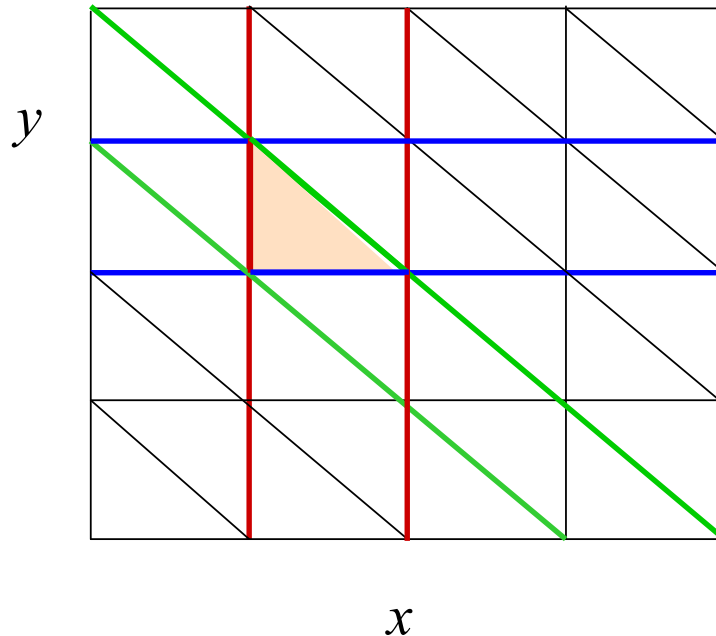
Piecewise Linear Bivariate Functions Using SOS2

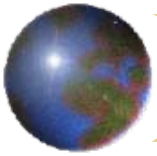
You might find it easier to use the SOS2 feature.

Basic idea: Given relation $r = f(x,y)$, use linear approximation within each triangle.

A unique triangle can be identified by choosing
2 adjacent vertical,
2 adjacent horizontal, and
2 adjacent diagonal lines.

Why use triangles
rather than, say,
rectangles?





Piecewise Linear Bivariate Functions Using SOS2 - I

!Piecewise linear approximation of $f(h,v)$;

SETS:

HSET; ! The horizontal dimension;

VSET; ! The vertical dimension;

DSET: WD; ! The diagonal dimension;

HXV(VSET, HSET): HV, VV, FV, WGT;

ENDSETS

DATA:

HSET = 1..3;

VSET = 1..3;

DSET = 1..5; ! There are $n*n - 1$ diagonal lines;

! Matrix of Data points for head(pressure), volume
and power output(function value) for a hydro-electric generator;

HV, VV, FV =

195 1800 20 ! Row 1;

195 3500 52

195 5100 69

217 1900 26 ! Row 2;

217 3600 61

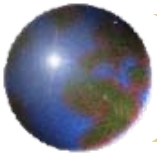
217 5200 80

240 2000 30 ! Row 3;

240 4100 78

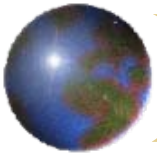
240 5600 93 ;

ENDDATA



Piecewise Linear Bivariate Functions Using SOS2 - II

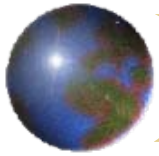
```
! Weights must form a convex combination;
! Horizontal;
  @SUM( HSET(k) : WH(k) ) = 1;
! Vertical;
  @SUM( VSET(k) : WV(k) ) = 1;
! Diagonal;
  @SUM( DSET(k) : WD(k) ) = 1;
! Declare the grid weights to be SOS2 sets, i.e., at most
  2 can be > 0 and they must be adjacent;
  @FOR( HSET(k) : @SOS2( 'SH2', WH(k) ) );
  @FOR( VSET(k) : @SOS2( 'SV2', WV(k) ) );
  @FOR( DSET(k) : @SOS2( 'SD2', WD(k) ) );
! Tie the grid line weights to the point weights;
  @FOR( HSET(k) : WH(k) = @SUM( VSET(j) : WGT(k,j) ) );
  @FOR( VSET(k) : WV(k) = @SUM( HSET(i) : WGT(i,k) ) );
  @FOR( DSET(k) : WD(k) =
    @SUM( HXV(i,j) | i+j-1 #EQ# k : WGT(i,j) ) );
```

Piecewise Linear Bivariate Functions Using SOS2 - III

```
! Compute associated horizontal or X value;
[COMPX]  @SUM( HXV(i,j): HV(i,j)* WGT(i,j)) = HA;
! Compute associated vertical or Y value;
[COMPY]  @SUM( HXV(i,j): VV(i,j)* WGT(i,j)) = VA;
! Compute associated function value;
[COMPFV] @SUM( HXV(i,j): FV(i,j)* WGT(i,j)) = FA;

! Define an arbitrary optimization problem to illustrate the method;
! An arbitrary tradeoff cost between head and volume;
[ARBOBJ]  Min = VA + 15*HA;
! We need this much power;
[ARB1]    FA >= 75;
! Arbitrarily put some constraints on Head;
[ARB2]    HA <= 229;
[ARB3]    HA >= 227;
```

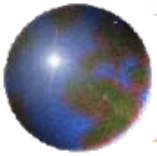


Piecewise Linear Bivariate Functions Using SOS2 - IV

Global optimal solution found.

Objective value: 7727.906

Variable	Value
HA	229.000000
VA	4292.906000
FA	75.000000
WH(2)	0.478261
WH(3)	0.521739
WV(2)	0.729977
WV(3)	0.270023
WD(3)	0.208238
WD(4)	0.791762
WGT(2, 2)	0.208238
WGT(2, 3)	0.270023
WGT(3, 2)	0.521739



Piecewise Linear Bivariate Functions Using Two SOS2

An alternative approach to representing a piecewise linear function of two variables, $output = f(p, v)$, is to partition the (p, v) space into a grid of rectangles, and then use two SOS2 sets to choose an interval on the p coordinate, and an interval on the v coordinate. The formulation is:

$$1 = \sum_i wp_i, \quad ! \text{ Must choose a value for } p \text{ coordinate;}$$

$$1 = \sum_j wv_j, \quad ! \text{ Must choose a value for } v \text{ coordinate;}$$

$$p = \sum_i wp_i * pbar_i,$$

$$v = \sum_j wv_j * vbar_j,$$

For each i : ! Weights on points must be consistent with weights on lines;

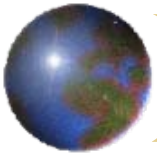
$$wp_i = \sum_j wpv_{ij},$$

For each j ;

$$wv_j = \sum_i wpv_{ij},$$

$$output = \sum_{ij} wpv_{ij} * f(pbar_i, vbar_j),$$

The weights $wp_i \geq 0$ and $wv_j \geq 0$ must be SOS2 sets, i.e., at most two weights nonzero and they must be adjacent.



Piecewise Linear Bivariate Functions Using Two SOS2 - II

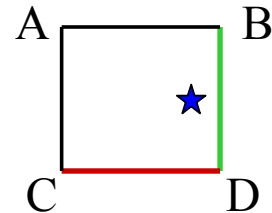
The Advantages and Disadvantages of the “Double SOS2” formulation are:

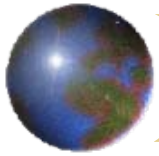
- + Exploits the SOS2 capability of most software, typically resulting in good/smart branching in the search tree.
- The number of implicit binary variables, effectively the $w p_i$ and $w v_j$, increase with the square root of the number of piecewise linear sections, rather than with the log of the number of sections.
- The representation of a point within a rectangle is not unique, whereas the representation of a point within a triangle is unique.

E.g., the star can be represented as either a convex combination of

A, B, D or of

B, C, D. Thus, the function value or output value is not uniquely defined. The optimizer will choose the more favorable one.





Linear Approximations of Cross-Product, $x_i * x_j$ Terms

Suppose our model contains the product:

$$x_1 * x_2, \quad (\text{This happens for example in petroleum pooling problems.})$$

An alternate representation is to add the linear constraints:

$$y_1 = (x_1 + x_2)/2$$

$$y_2 = (x_1 - x_2)/2.$$

Then, replace every instance of $x_1 * x_2$ by the term $y_1^2 - y_2^2$. That is, the claim is:

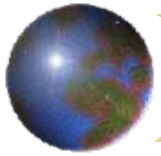
$$x_1 * x_2 = y_1^2 - y_2^2.$$

The justification is observed by noting:

$$\begin{aligned} y_1^2 - y_2^2 &= (x_1^2 + 2 * x_1 * x_2 + x_2^2)/4 - (x_1^2 - 2 * x_1 * x_2 + x_2^2)/4 \\ &= 4 * x_1 * x_2 / 4 = x_1 * x_2. \end{aligned}$$

A standard SOS2 linear approximation can be applied to each of y_1^2 and y_2^2 .

This example suggests that, any time you have a product of two variables, you can add two new variables to the model and replace the product term by a sum of two squared variables. If you have n original variables, you could have up to $n(n-1)/2$ cross product terms. This suggests that you might need up to $n(n-1)$ new variables to get rid of all cross product terms. In fact, sometimes the above ideas can be generalized, using various factorization techniques such as Cholesky and others, so only at most n new variables are needed.



Linear Approximations of Cross-Product, $x_i^*x_j$ Terms

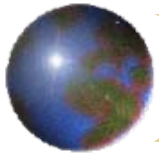
If we have:

- a) the constraint $x'Qx \leq r$,
 - b) matrix Q is positive definite, i.e., $x'Qx$ is a convex function,
 - c) a fast solver that handles convex quadratic constraints,
- then life is good.

But suppose (b) does not hold. In this case, you can easily find a nonnegative diagonal matrix d such that the matrix $(Q+d)$ is positive definite and so:

$$x'Qx = x'(Q+d)x - x'dx = x'(Q+d)x - r;$$

If we have a solver that can efficiently solve models with convex quadratic expression as well as integer variables, then we can approximate $r = \sum_i x_i d_{ii} x_i$ with at most n SOS2 sets, if Q is n by n .



Linearizing by Changing Our Metric or Variables

Example: Computing Quality of a Blend

We have two ingredients,

one with a density of 0.7 g/cc and the other with a density of 0.9 g/cc.

If we mix together one gram of each, is the density 8 g/cc?

Clearly, the mix has a weight of 2 grams. Its volume in cc's is $1/0.7 + 1/0.9$.

Thus, its density is $2/(1/0.7 + 1/0.9) = 0.7875$ g/cc.

This is less than the 0.8 we would predict if we took the arithmetic average.

If we define:

X_i = units of feature i in the mix,

t = target lower limit on density desired.

Then, we can write the density constraint for our little example as:

$$(X_1 + X_2)/(X_1/0.7 + X_2/0.9) \geq t,$$

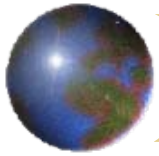
or as a linear constraint:

$$(X_1 + X_2)*(1/t) \geq X_1*(1/0.7) + X_2*(1/0.9),$$

(i.e., a harmonic mean constraint).

The constraint became linear by changing the measure from g/cc to cc/g.

Similar example occurs in computing average fleet miles/gallon to meet U.S. CAFE (Corporate Average Fuel Economy) standards.



Linearization via Changing How We Measure Things

One can generalize the idea just discussed by introducing a transformation $f(q)$. The function $f()$ “linearizes” the quality. Many of the quality measures used in practice were chosen somewhat arbitrarily (e.g., why is the freezing point of water 32 degrees on the Fahrenheit scale?). So, even though a standardly used quality measure does not “blend linearly”, perhaps we can find a transformation that does. Such linearizations are common in industry. Some examples:

1. Rigby, Lasdon, and Waren (1995) use this idea when calculating the Reid vapor pressure of a blended gasoline at Texaco. If r_i is the Reid vapor pressure of component i of the blend, they use the transformation:

$$f(r_i) = r_i^{1.25}$$

For example, if component 1 has a vapor pressure of 80, component 2 has a vapor pressure of 100, r_i is the amount used of component i , and we want a blend with a vapor pressure of at least 90, the constraint could be written:

$$80^{1.25} * x_1 + 100^{1.25} * x_2 \geq 90^{1.25} * (x_1 + x_2),$$

or

$$239.26 x_1 + 316.23 x_2 \geq 277.21 (x_1 + x_2).$$

2. The flashpoint of a chemical is the lowest temperature at which it will catch fire. Typical jet fuel has a flashpoint of around 100 degrees F. Typical heating oil has a flashpoint of at least 130 degrees F. The jet fuel used in the supersonic SR-71 jet aircraft had a flashpoint of several hundred degrees F. If p_i is the flashpoint of component i , then the transformation:

$$f(p_i) = 10^{42} (p_i + 460)^{-14.286}$$

will approximately linearize the flashpoint. Notice that $f(p_i)$ is a decreasing function of p_i , so a higher flashpoint means a lower $f(p_i)$ value.

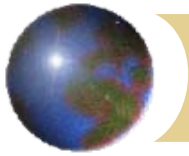
For example, if component 1 has a flashpoint of 100, component 2 has a flashpoint of 140, x_i is the amount used of component i , and we want a blend with a flashpoint of at least 130, the constraint would be written:

$$10^{42} (100 + 460)^{-14.286} * x_1 + 10^{42} (140 + 460)^{-14.286} * x_2 \leq$$

$$10^{42} (130 + 460)^{-14.286} * (x_1 + x_2),$$

or

$$548.76 x_1 + 204.8 x_2 \leq 260 (x_1 + x_2).$$



Linearizing by Changing Metric- II

3. The American Petroleum Institute likes to measure the lightness of a material in “API gravity”, see Dantzig and Thapa (1997). API gravity does not blend linearly. However, the specific gravity, defined by:

$$sg = 141.5 / (API\ gravity + 131.5)$$

does blend linearly. Note, the specific gravity of a material is the weight in grams of one cubic centimeter of material. Water has an API gravity of 10.

4. The viscosity of a liquid is a measure, in units of centistokes, of the time it takes a standard volume of liquid, at 122 degrees Fahrenheit, to flow through a hole of a certain diameter. The higher the viscosity, the less easily the liquid flows. If v_i is the viscosity of component i , then the transformation:

$$f(v_i) = \ln(\ln(v_i + .08))$$

will approximately linearize the viscosity.

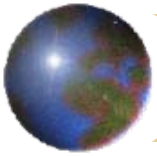
For example, if component 1 has a viscosity of 5, component 2 has a viscosity of 25, x_i is the amount used of component i , and we want a blend with a viscosity of at most 20, the constraint would be written:

$$\ln(\ln(5 + .08)) * x_1 + \ln(\ln(25 + .08)) * x_2 \leq$$

$$\ln(\ln(20 + .08)) * (x_1 + x_2),$$

or

$$.4857 x_1 + 1.17 x_2 \leq 1.0985(x_1 + x_2).$$



Linearizing by Changing Metric- - III

5. In the transmissivity of light through a glass fiber of length x_i , or the financial growth of an investment over a period of length x_i , or in the probability of no failures in a number of trials x_i , one may have constraints of the form: $a_1^{x_1} a_2^{x_2} \dots a_n^{x_n} \geq a_0$. This can be linearized by taking logarithms (e.g., $\ln(a_1) * x_1 + \ln(a_2) * x_2 + \dots + \ln(a_n) * x_n \geq \ln(a_0)$).

For example, if we expect stocks to have a long term growth rate of 10% per year, we expect less risky bonds to have a long term growth rate of 6% per year, we want an overall growth of 40% over five years, and x_1 and x_2 are the number of years we invest in stocks and bonds respectively over a five year period, then we want the constraint:

$$(1.10)^{x_1} (1.06)^{x_2} \geq 1.40.$$

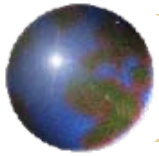
Linearizing, this becomes:

$$\ln(1.10) x_1 + \ln(1.06) x_2 \geq \ln(1.40), \text{ or}$$

$$.09531 x_1 + .05827 x_2 \geq .3364,$$

$$x_1 + x_2 = 5.$$

The preceding examples apply the transformation to each quality individually. One could extend the idea even further by allowing a “matrix” transformation to several qualities together.



Second Order Cone Constraints: Quadratics + more

The pair of constraints:

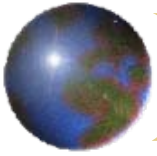
$$S1) \quad x_1^2 + x_2^2 + \dots + x_n^2 - z^2 \leq 0;$$

$$S2) \quad z \geq 0;$$

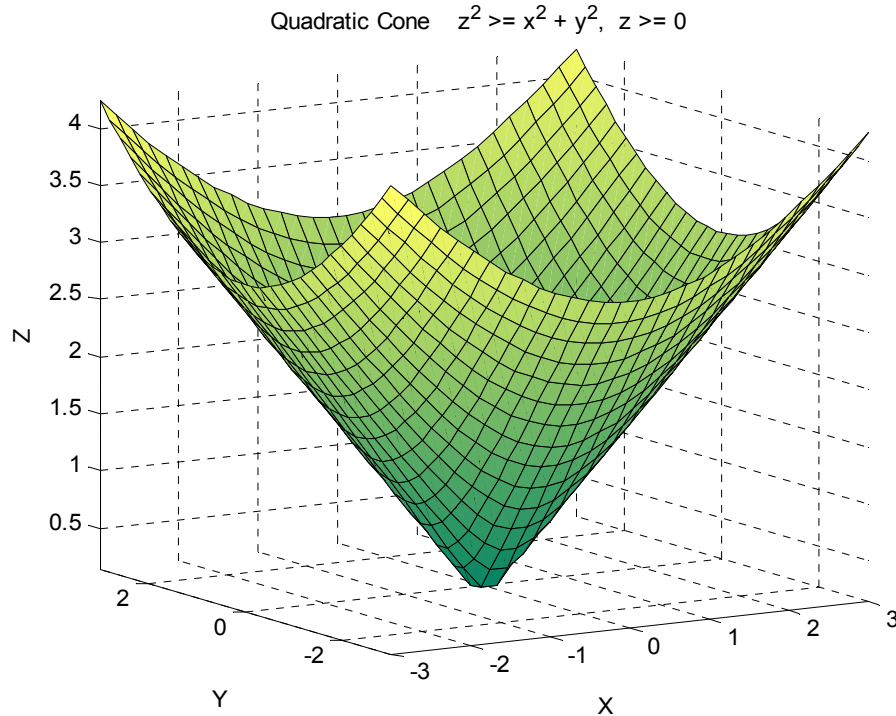
describe a convex set and
is called a second order cone.

Note that constraint S1 by itself is not a convex set.
A traditional “quadratic” solver may refuse to solve a model that contains S1 with the message something like the model contains a nonconvex constraint.

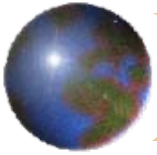
More recent Second Order Cone Program (SOCP) solvers will tolerate S1, S2, and in fact solve quickly.



Second Order Cone Constraints, Graphically



If we drop the constraint $z \geq 0$, then the feasible region would consist of two cones, the second one a reflection of the first, below the first. The feasible region would not be convex.



Second Order Cone Constraints, Generality

SOC constraints are surprisingly general, if we are allowed to complement the SOC with appropriate linear constraints. Some important examples are:

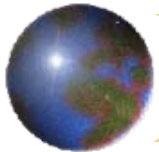
Rotated SOC constraints:

The pair of constraints:

$$S1') \quad x_1^2 + x_2^2 + \dots x_n^2 - y*z \leq 0;$$

$$S2') \quad y, z \geq 0;$$

can be shown to be equivalent to the original S1, S2. Note, some low level solvers may require S1' to be written: $x_1^2 + x_2^2 + \dots x_n^2 - 2*y*z \leq 0;$



Converting Models to SOC/Conic Form

Conic constraints are more general than perhaps is superficially obvious. Generally, any constraint of the form:

$$|x|^p \leq t;$$

for p rational and either $p \geq 1$, or $p \leq 0$, $x \geq 0$, can be represented by conic constraints by appropriate transformations.

For example, in **financial portfolio** models where one wants to take into account transaction costs and the **“market impact” of a transaction**, i.e., how the price of a commodity is affected by how much we buy or sell of that commodity, we may wish to represent constraints such as:

$$x^{1.5} \leq t; \quad x \geq 0;$$

This is enforced with the two conic constraints:

$$x^2 \leq 2st;$$

$$s^2 \leq 2xr;$$

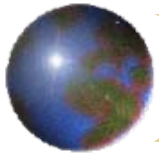
and the linear constraints:

$$r = 1/8;$$

$$s \geq 0;$$

Notice that $s = (x^{0.5})/2$ will satisfy the second constraint so the first constraint is

$$x^2 \leq 2[(x^{0.5})/2] t; \quad \text{or} \quad x^{1.5} \leq t;$$



Converting Models to SOCP Form - II

As another illustration of this generality, consider a constraint set of the form:

$$r \geq (a + bx)/(c+dx);$$

$$c+dx \geq 0;$$

Expressions such as this arise for example in modeling **traffic delay or congestion** as a function of traffic volume through a congested facility or transportation link. A constraint such as the above can be put into SOCP form if $a - bc/d \geq 0$. To do this define:

$2y = c+dx$, then $x = (2y-c)/d$, and $r \geq (a + bx)/(c+dx) = (a + bx)/(2y) = (a - bc/d)/(2y) + b/d$.

Thus, the constraint is convex if $y \geq 0$ and $a - bc/d \geq 0$.

If we define $u = (r-b/d)$, then $r - b/d \geq (a - bc/d)/(2y)$ is equivalent to the cone constraint:

$$2yu \geq a-bc/d.$$

Summarizing, given $a - bc/d \geq 0$, we can replace:

$$r \geq (a + bx)/(c+dx);$$

$$c+dx \geq 0;$$

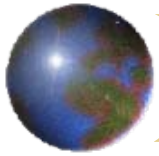
by the SOCP set of constraints:

$$2y = c+dx;$$

$$r = u + b/d;$$

$$2yu \geq a-bc/d;$$

$$y \geq 0;$$



Converting Models to SOCP Form, Value at Risk Portfolios

Although perhaps not immediately obvious, a SOCP is at least as general as a quadratic program. In a quadratic program one typically wants to either minimize a quadratic expression, written as $x'Qx$, or constrain $x'Qx$ from above. A related example is in **Value-At-Risk analysis**, where one may have models of the form:

$$\text{Minimize } k*\sigma - \mu;$$

subject to

$$\sigma^2 \geq x'Qx;$$

$$\mu = r'x;$$

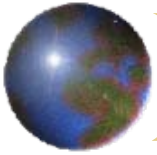
If the Q matrix is positive definite, then $x'Qx$ is convex and SOCP can be applied as outlined below. An easy way to a) check for positive definiteness, and b) put the model into a SOCP form is to compute a Cholesky Decomposition or “square root” of the Q matrix. In matrix notation we can write:

$$\sigma^2 \geq x'Qx = xLL'x'.$$

Here, L is a lower triangular matrix which we can think of as the square root of Q . So the SOCP form is:

$$\sigma^2 \geq yy';$$

$$y = xL;$$



SOC Constraints and IP, Numerical Illustration, Quadratic Plant Location

SUBMODEL SQFL:

! Simple plant location with quadratic shipping costs,
and min ship quantities;

! Parameters:

$C(i)$ = fixed cost of opening facility i ,

$Q(i,j)$ = distance from i to customer j ,

MU = minimum fraction of j 's demand that must satisfied
from i , if anything is shipped from i to j ,

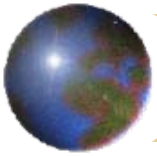
Variables:

$X(i,j)$ = fraction of customer j 's demand
received from facility i ,

$ZF(i)$ = 1 if facility i is opened, else 0,

$Z(i,j)$ = 1 if anything shipped from i to j , else 0,

;



A Quadratic Integer Program ==> SOC IP

! The model, the simplest version;

```
MIN = @SUM( FACILITY(i): C(i)*ZF(i))  
      + @SUM( FXC(i,j): Q(i,j)*Y(i,j));
```

! Each customer j must be served;

```
@FOR( CUST(j):  
      @SUM( FACILITY(i): X(i,j)) = 1;  
      );
```

! If ship anything from i to j ...;

```
@FOR( FXC(i,j):  
  
      ! must turn on shipment variable;  
      X(i,j) <= Z(i,j);  
  
      ! must ship minimum amount;  
      X(i,j) >= MU*Z(i,j);  
  
      ! i must be open;  
      Z(i,j) <= ZF(i);
```

! Force $Y(i,j) \geq$ cost of serving j from i;

```
      X(i,j)*X(i,j) - Y(i,j) <= 0;           ! <== QP;
```

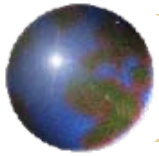
! Optional, tighten above into an SOC constraint;

```
!      X(i,j)*X(i,j) - Y(i,j)*Z(i,j) <= 0;           ! <== SOC;  
      );
```

! Facility is either completely open or close;

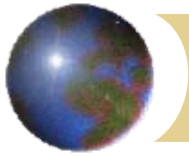
```
@FOR( FACILITY(i) : @BIN( ZF(i)));  
@FOR( FXC(i,j) : @BIN( Z(i,j)));
```

ENDSUBMODEL



Computational Results for an Example

<u>Model formulation</u>	<u>Continuous relaxation</u>	<u>IP Optimal objective</u>	<u>Seconds to NLIP optimum</u>
QP	162.9	317.1	>600
SOC	316.9	317.1	20



Aggregation of Variables

Basic Idea/Hope:

Do not distinguish between variables,
- aggregate similar ones if you can still have a precise model.

Illustration from a Multi-Commodity Flow Network:

You want to model the shipping network for UPS, FedEx, or DHL, etc.
Suppose 1000 cities in your network, each both an origin & destination.
You must keep track of each type of shipment so it gets to correct destination.

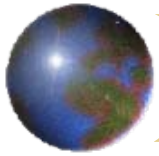
Question: Do we need $1000*1000 = 1,000,000$ variables?

Key observation: Once a shipment enters the network,
we only need to know its destination. We do not need to remember its origin.
Instead of needing variables:

x_{jkfd} = volume on link from j to k of goods from f destined for d , only need:

x_{jkd} = volume on link from j to k of goods destined for d .

If the network has, say 700 (j, k) links, this reduces the variables from
700,000,000 to 700,000.



Disjunctive Approach for Formulating Integer Programs

Situation: We have to choose among two or more alternatives and we want to figure out which is best. If we disregard the alternatives, our variables are simply called x_1, x_2, \dots, x_n . We call the conditions that must hold if alternative s is chosen, disjunction or scenario s . Without much loss of generality, assume all variables ≥ 0 . The scenario/disjunctive approach to formulating a discrete decision problem proceeds as follows:

For each scenario s :

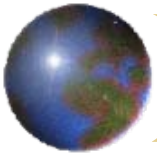
- 1) Write all the constraints that must hold if scenario s is chosen.
- 2) For all variables in these constraints add a subscript s , to distinguish them from equivalent variables in other scenarios. So x_j in scenario s becomes x_{sj} .
- 3) Add a 0/1 variable, y_s , to the model with the interpretation that $y_s = 1$ if scenario s is chosen, else 0.
- 4) Multiply the RHS constant term of each constraint in scenario s by y_s .
- 5) For each variable x_{sj} that appears in any of the scenario s constraints, add the constraint:
$$x_{sj} \leq M * y_s$$
, where M is a suitably large positive constant. The purpose of this step is to force all variables in scenario s to be 0 if scenario s is not chosen.

Finally, we tie all the scenarios together with:

$$1 = \sum_s y_s, \text{ i.e., we must choose one scenario;}$$

For each variable x_j , add the constraint:

$$x_j = \sum_s x_{sj}, \text{ so } x_j \text{ takes on the value appropriate to which scenario was chosen.}$$



Disjunctive Approach, Example

Our vendor gives us the following “all-units” quantity discount schedule:

We pay \$50 if we buy anything in a period, plus
\$2.00/unit if quantity < 100,
\$1.90/unit if quantity ≥ 100 but < 1000,
\$1.80/unit if ≤ 1000 but ≤ 5000 .

Let x denote the amount we decide to purchase. The possible scenarios are:

1) $x = 0$; 2) $1 \leq x \leq 99$, 3) $100 \leq x \leq 999$, 4) $1000 \leq x \leq 5000$.

Applying the scenario or disjunctive formulation approach,

For segment/scenario 1, $y_1 = 1$, is chosen, then

$$cost_1 = 0;$$

$$x_1 = 0;$$

If segment/scenario 2, $y_2 = 1$, is chosen, then

$$cost_2 = 50*y_2 + 2.00*x_2,$$

$$x_2 \geq 1*y_2;$$

$$x_2 \leq 99*y_2;$$

If segment/scenario 3, $y_3 = 1$, is chosen, then

$$cost_3 = 50*y_3 + 1.90*x_3,$$

$$x_3 \geq 100*y_3;$$

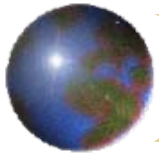
$$x_3 \leq 999*y_3;$$

If segment/scenario 4, $y_4 = 1$, is chosen, then

$$cost_4 = 50*y_4 + 2*x_4,$$

$$x_4 \geq 1*y_4;$$

$$x_4 \leq 99*y_4;$$



Disjunctive Approach, Example

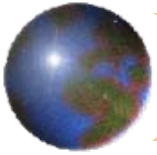
Now tie the all the scenarios together:

$$y_1 + y_2 + y_3 + y_4 = 1; \quad ! \text{ Must choose 1;}$$

$$x_1 + x_2 + x_3 + x_4 = x; \quad ! \text{ The actual quantity;}$$

$$\text{cost}_1 + \text{cost}_2 + \text{cost}_3 + \text{cost}_4 = \text{cost}; \quad ! \text{ Actual cost;}$$

$$y_1, y_2, y_3, y_4 = 0 \text{ or } 1;$$



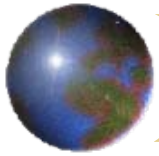
Global Solvers & Arbitrary Nonconvex NLP's

We have run out of tricks.

We are confronting an
unavoidably nonlinear, possibly nonconvex problem.
What to do?

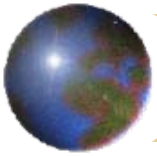
Global solver technology is now well advanced.

Can get a guaranteed global optimal solution.



Real World Applications for Global Solvers

- Real world models tend to be nonlinear\ nonconvex\ nonsmooth\ noncontinuous, have integer variables.
- New improvements shrink convex relaxation gap and locate good quality solutions fast
- Industrial experience in solving large scale spreadsheet nonconvex models with up to 10k variables:
 - - Energy Production Scheduling
 - - Chemical Process Design: Flowsheets in Spreadsheets
 - - Global Supply Chain Management of Chemicals

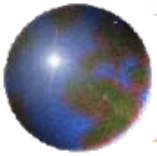


Relax and Branch Methodology

Basic ideas: convex relaxation, interval analysis constraint propagation, bound tightening, algebraic reformulation,.

Uses the “Prayer” algorithm, based on two ideas

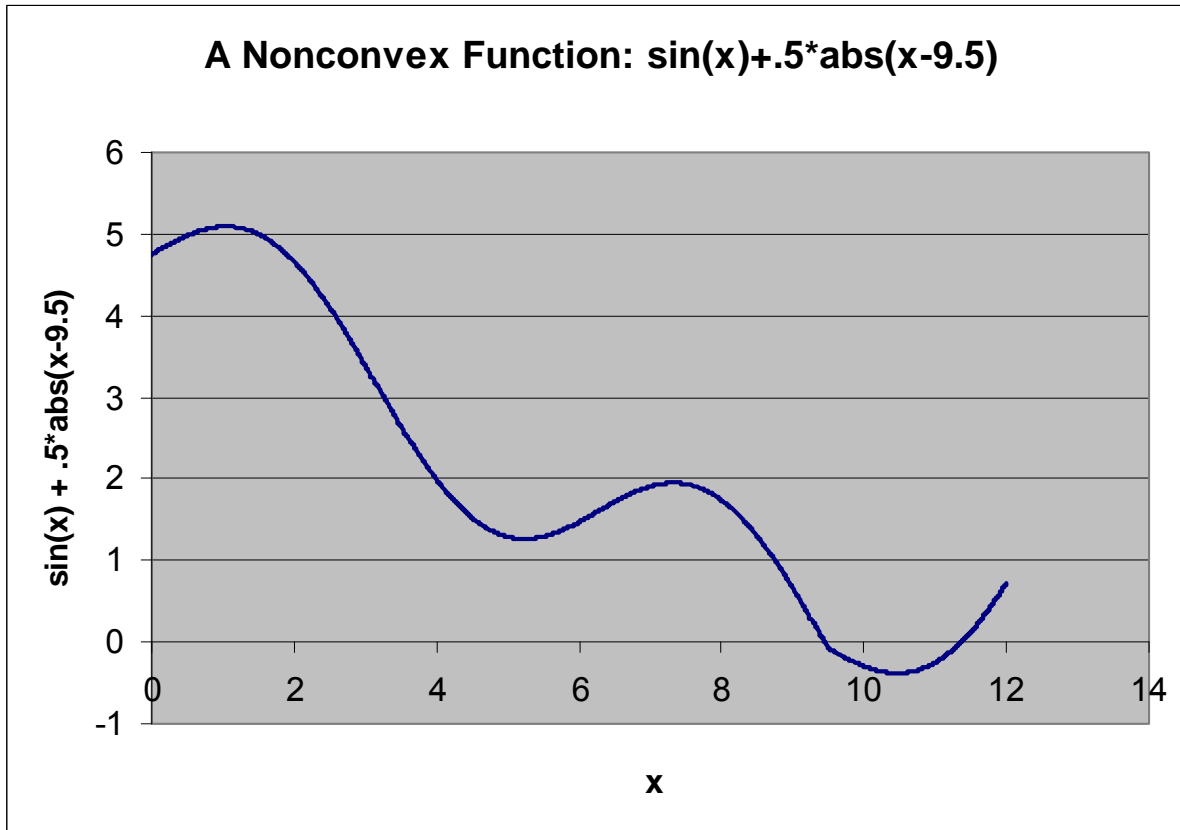
- 1) For each(arbitrary) nonlinear function, given current bounds on variables, automatically generate a convex relaxation of the function. Solve the relaxed convexified model.
- 2) Solve relaxed problem and pray that solution is feasible to the original model, else branch, i.e., partition the feasible region into two subregions. Calculate new implied bounds on the variables for each subproblem. Go back to (1).

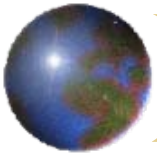


Creating a Convex Relaxation/Bound

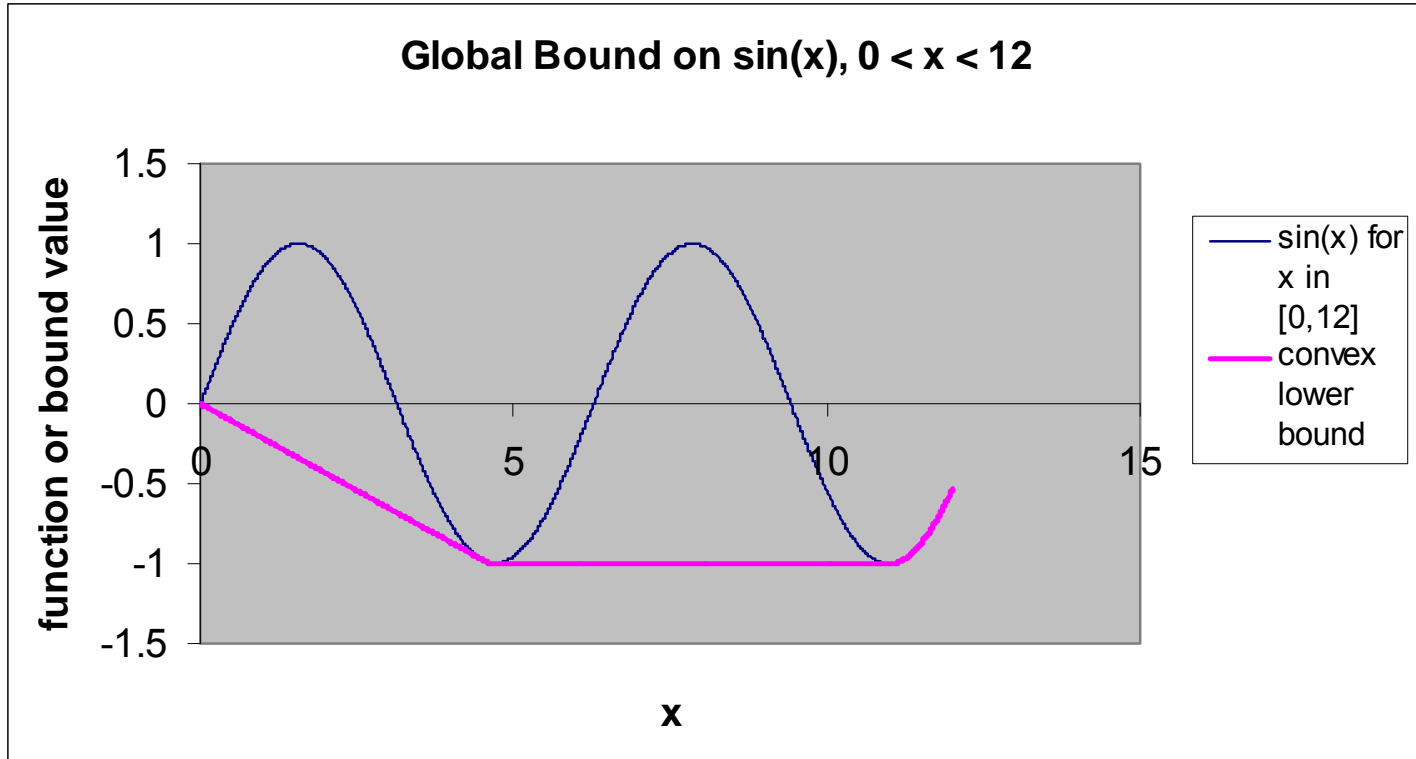
Example: $\text{Min} = \sin(x) + .5*\text{abs}(x-9.5);$

s.t. $0 \leq x \leq 12;$

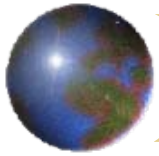




Bounding a Nonconvex Function



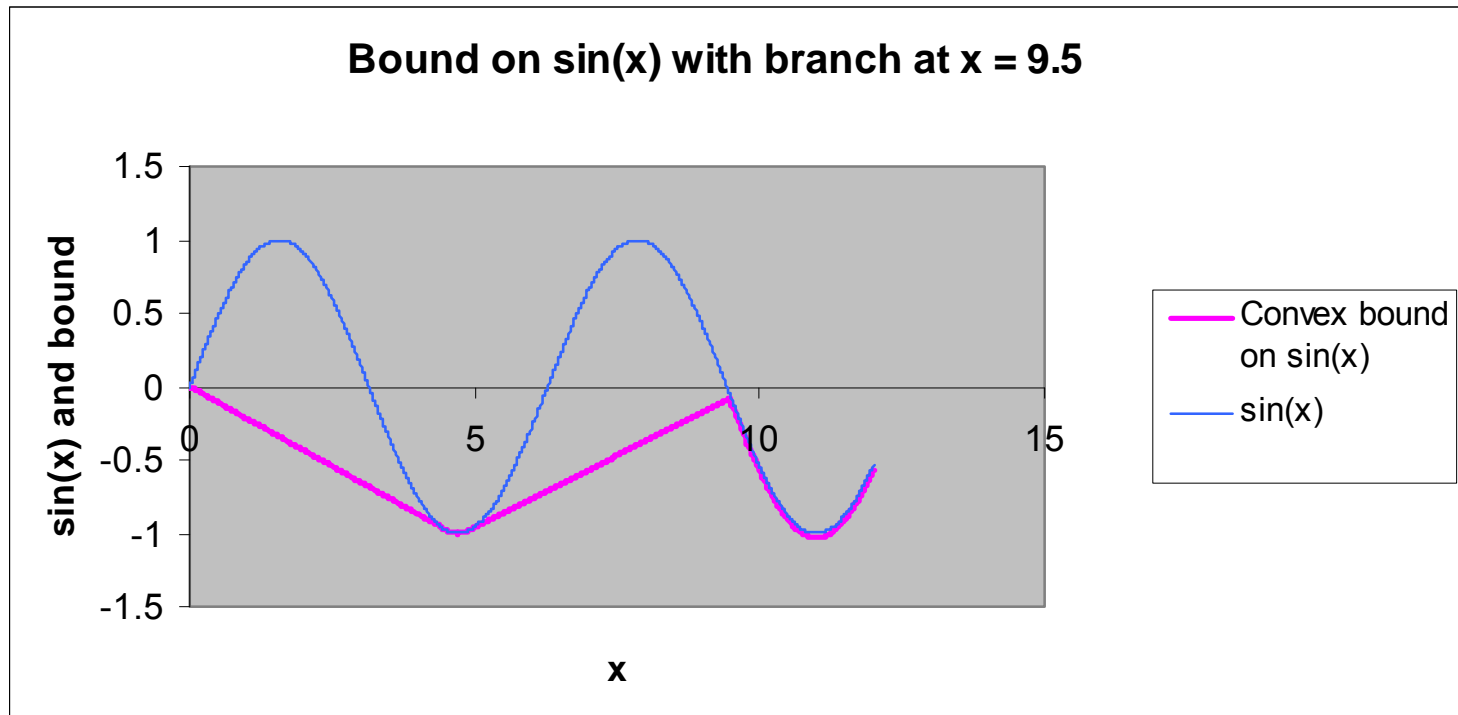
We replace $\sin(\)$ by its convex bound. Solve, get $x = 9.5$.



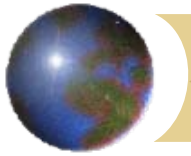
Branching

We branch on $x \leq 9.5$ vs. $x \geq 9.5$ and re-bound.

The branch $x \geq 9.5$ is convex with feasible solution $x = 10.47197$.



Bound discards $x \leq 9.5$ case; Case ≥ 9.5 is convex, we are done.



Global Solver Overview

LINDO API contains a global solver that finds a **mathematically guaranteed global optimum** to an arbitrary optimization problem;

Global solver fully supports all common math functions:

Continuous and smooth:

$x+y$, $x-y$, $x*y$, $\log(x)$, $\exp(x)$, \sqrt{x} , $\sin(x)$, $\cos(x)$, $\arccos(x)$, $\arcsin(x)$

Smooth, not quite continuous:

$\tan(x)$, x/y , x^y , $\text{floor}(x)$, $\text{mod}(x,y)$, $\text{sign}(x)$, $\text{int}(x)$,

Continuous, not quite smooth:

$\text{abs}(x)$, $\text{max}(x,y)$, $\text{min}(x,y)$,

Logical:

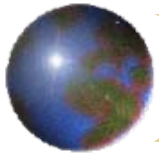
$\text{IF}(x,y,z)$, AND , OR , [where x is a logical expression]

Statistical:

$\text{normsdist}(u)$, $\text{psl}(z)$, [Normal distribution, linear loss]

Vector functions(for speed):

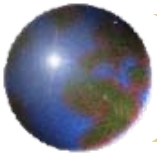
$\text{sum}(x_i)$, $\text{sumif}(x_i, y, z_i)$, $\text{vlookup}(x, y_i, z_i)$



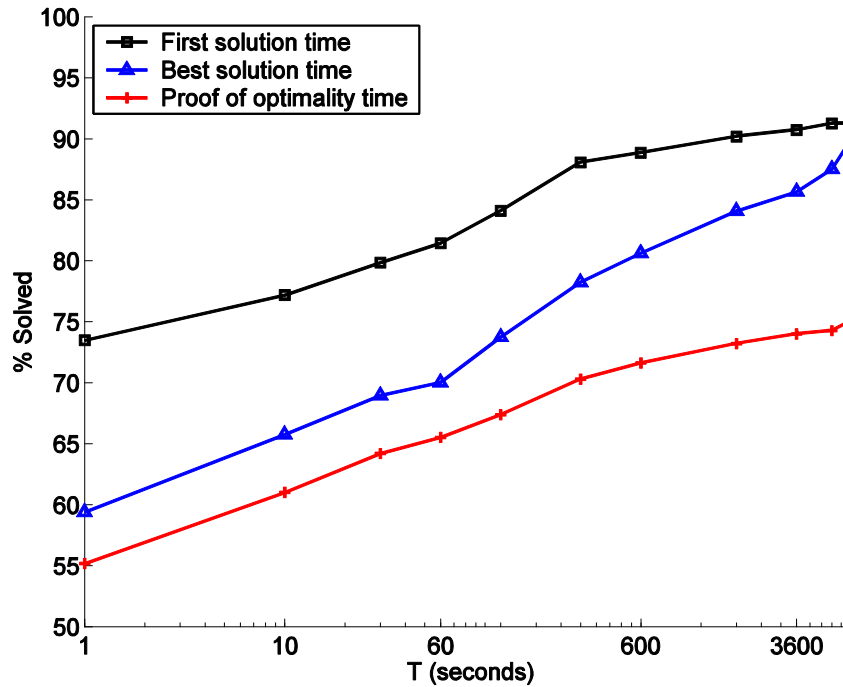
Global Solver Benefits

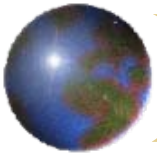
The real benefit of the Global Solver is that
on difficult problems
it finds better solutions faster.

A nice bonus is that if given enough time it will also prove optimality.



Performance on Continuous NLP's Globallib

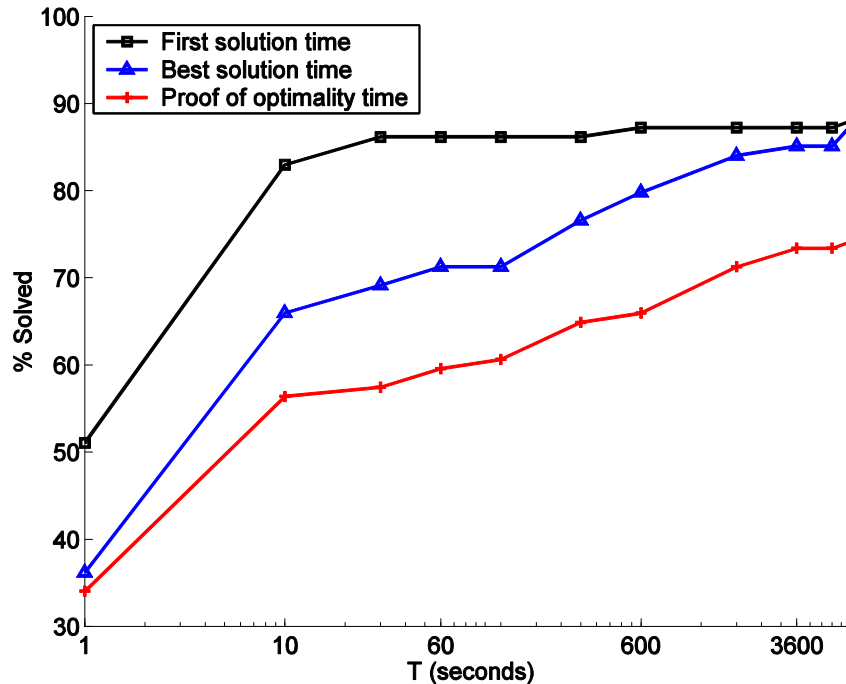


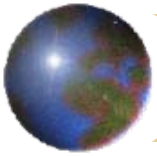


Math Programs with Equilibrium Constraints, MPECLIB

Optimization models for which a significant number of the constraints are of the form $x*y = 0$;

E.g., $price_i * surplus_i = 0$; or $opportunity_cost_i * usage_i = 0$;





Performance on NLP Integer Programs, MINLPLIB

