Energy Innovation Summit
Modeling Meets Optimization

# Solving MINLPs with Gurobi

**Jaromił Najman**
Senior Developer

# Why nonlinear?

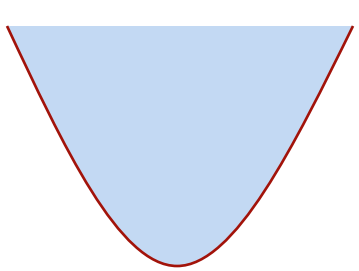
Many engineering applications require the use of formulas defined by nonlinear terms

- Finance
  - Risk $x^2$ - quadratic convex
- Pooling
  - Mixing products $x \cdot y$ – quadratic non-convex
- ACOPF
  - Power definitions $V_i V_j \sin(\theta)$ – nonlinear non-convex
- Gas networks
  - Pipe flow $|x| \cdot x, x^3$ - nonlinear non-convex
- Machine learning
  - Activation function $\frac{1}{1+e^{-x}}$ - nonlinear non-convex
- and more…

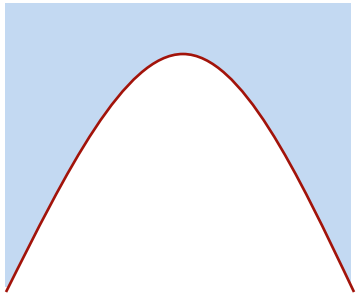
 

# Convex Quadratics

- Gurobi supports **convex** quadratic terms since version 5
  - After presolve remaining Q constraints and objective have to be convex



$x^T Q x \leq b$

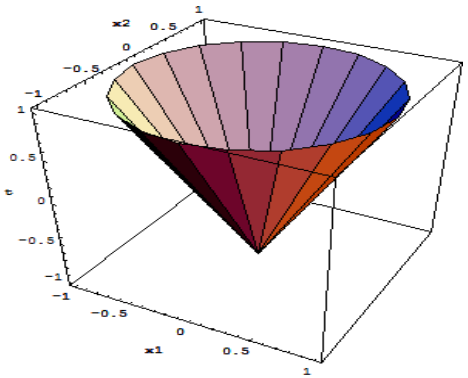


convex
$-z + x^2 \leq 0$

non-convex
$-z - x^2 \leq 0$

- If $Q$ is positive semi-definite (PSD) then $x^T Q x \leq b$ is convex
  - $Q$ is PSD if and only if $x^T Q x \geq 0$ for all $x$
- But $x^T Q x \leq b$ can also be convex in certain other cases, e.g., second order cones (SOCs)

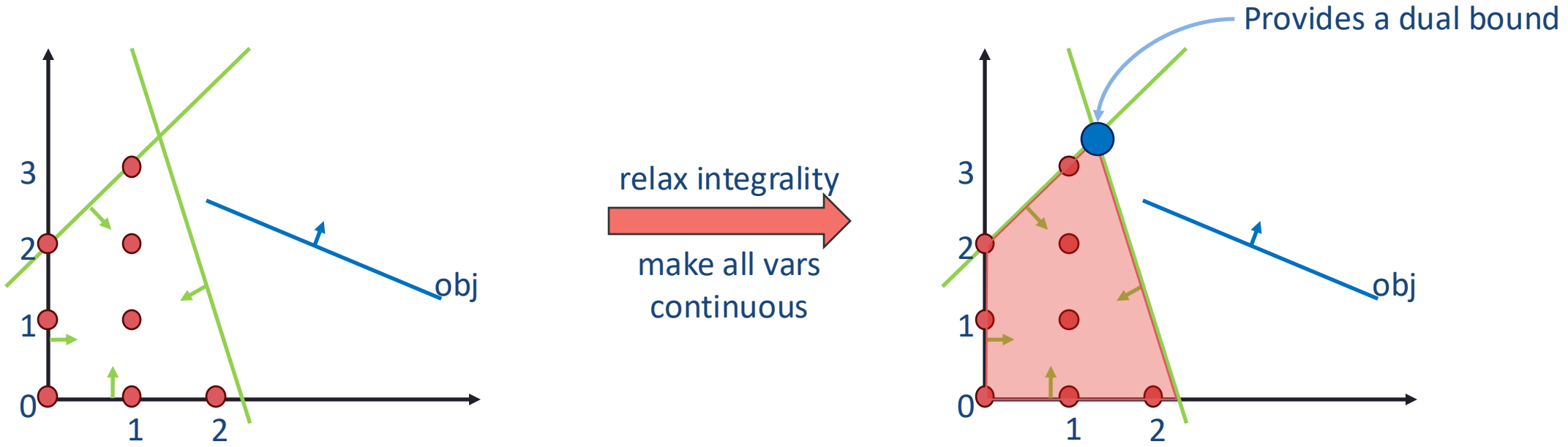
$$x_1^2 + \cdots + x_n^2 - z^2 \leq 0$$



# Non-convex Quadratics

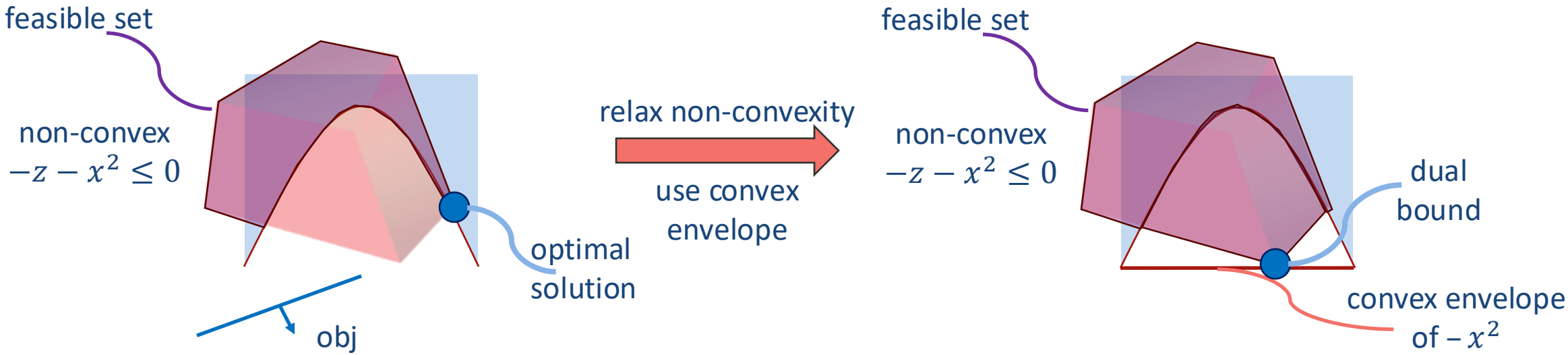
- Introduced support for non-convex quadratic problems with version 9
  - (MI)Q(C)Ps
- Extended the classical Integer Branch-and-Bound algorithm to deal with continuous variables
  - Spatial Branch-And-Bound
  - Outer-approximation



# Non-convex Quadratics

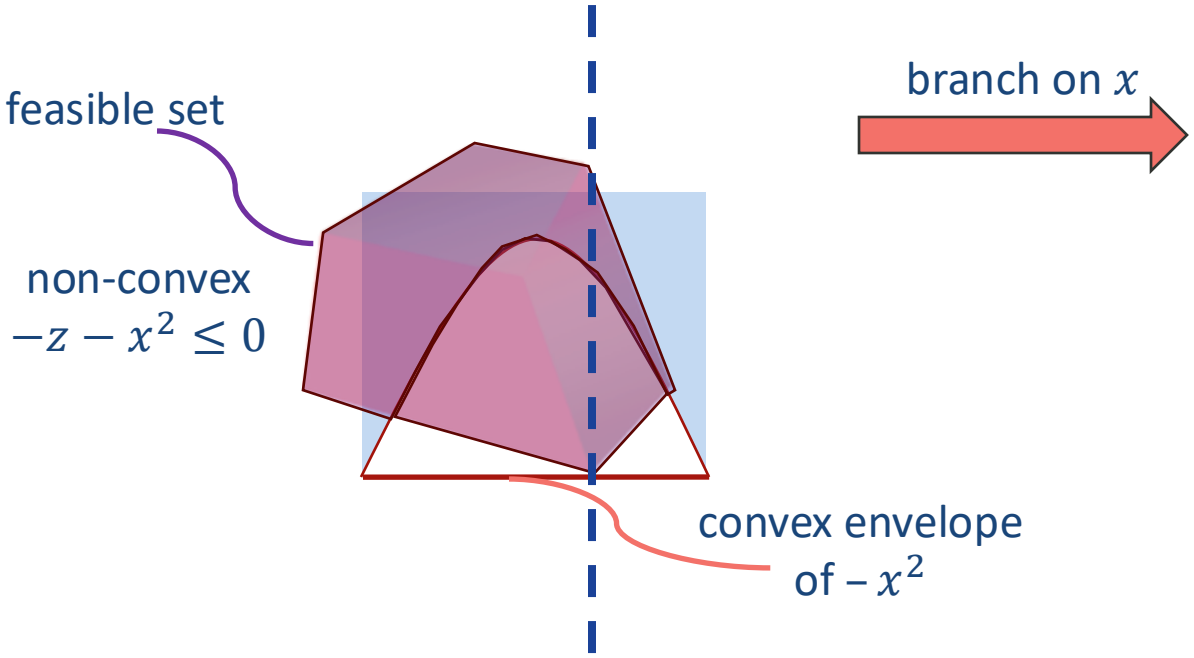
- Introduced support for non-convex quadratic problems
  - (MI)Q(C)Ps
- Extended the classical Integer Branch-and-Bound algorithm to deal with continuous variables
  - Spatial Branch-And-Bound
  - Outer-approximation

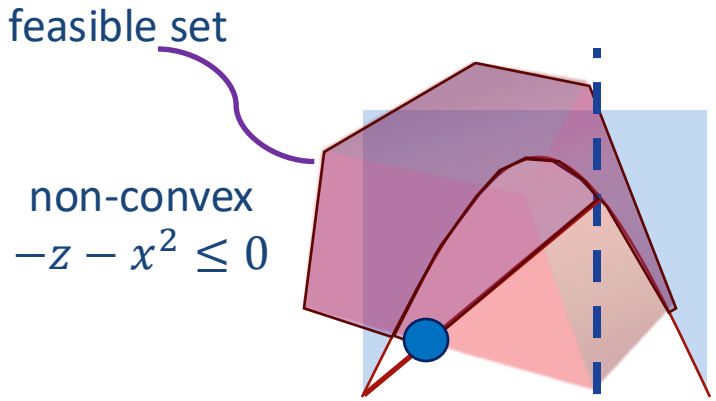


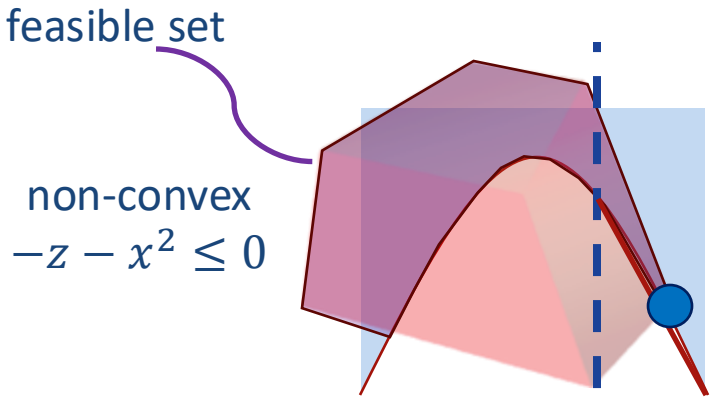

 

# Non-convex Quadratics

- But how can we improve the relaxation?
  - For MIPs we branch on discrete variables
- Branch on *continuous* variables

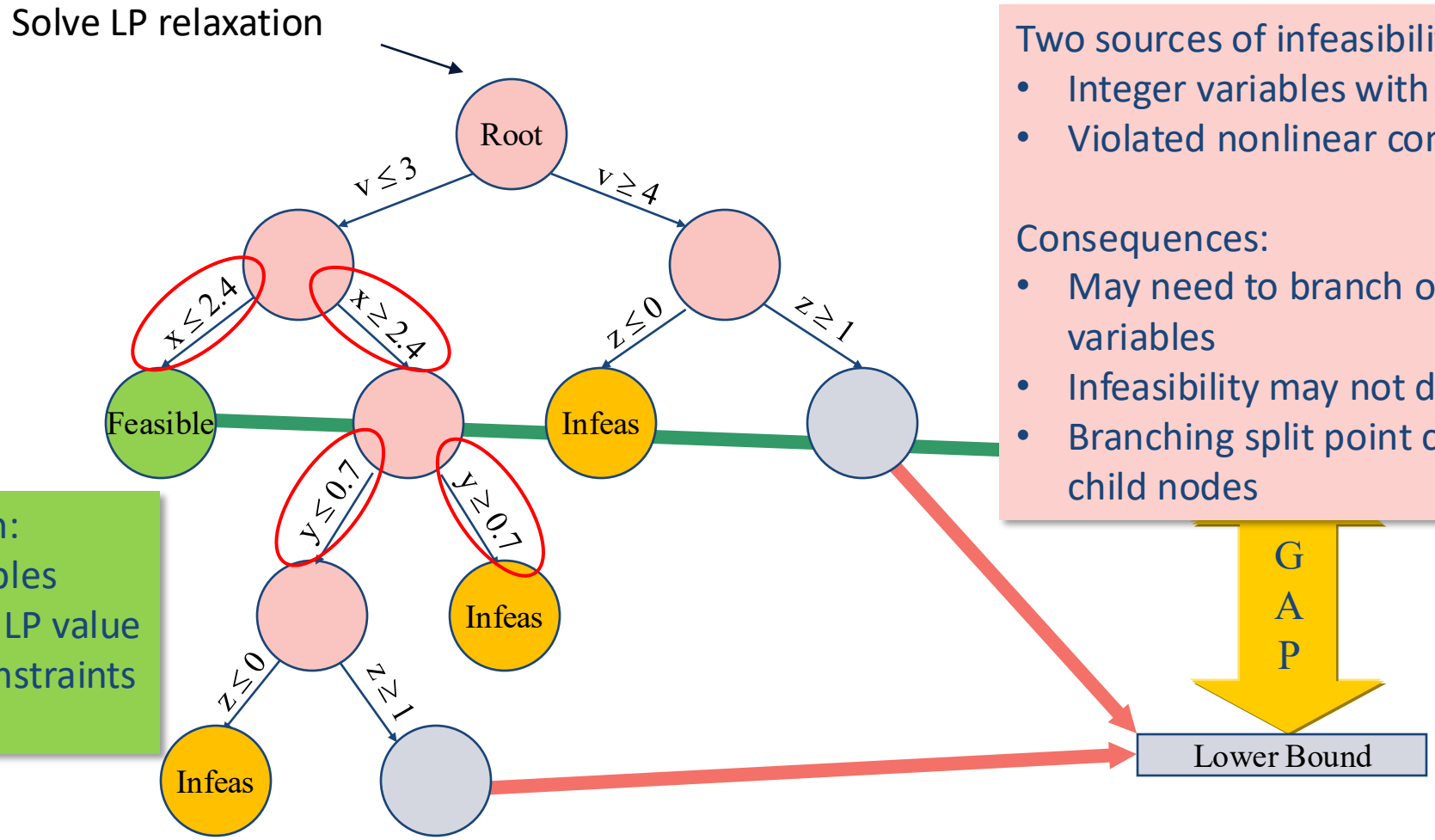


# Spatial Branch and Bound




Two sources of infeasibility:

- Integer variables with fractional LP value
- Violated nonlinear constraints

Consequences:

- May need to branch on continuous variables
- Infeasibility may not directly be resolved
- Branching split point contained in both child nodes

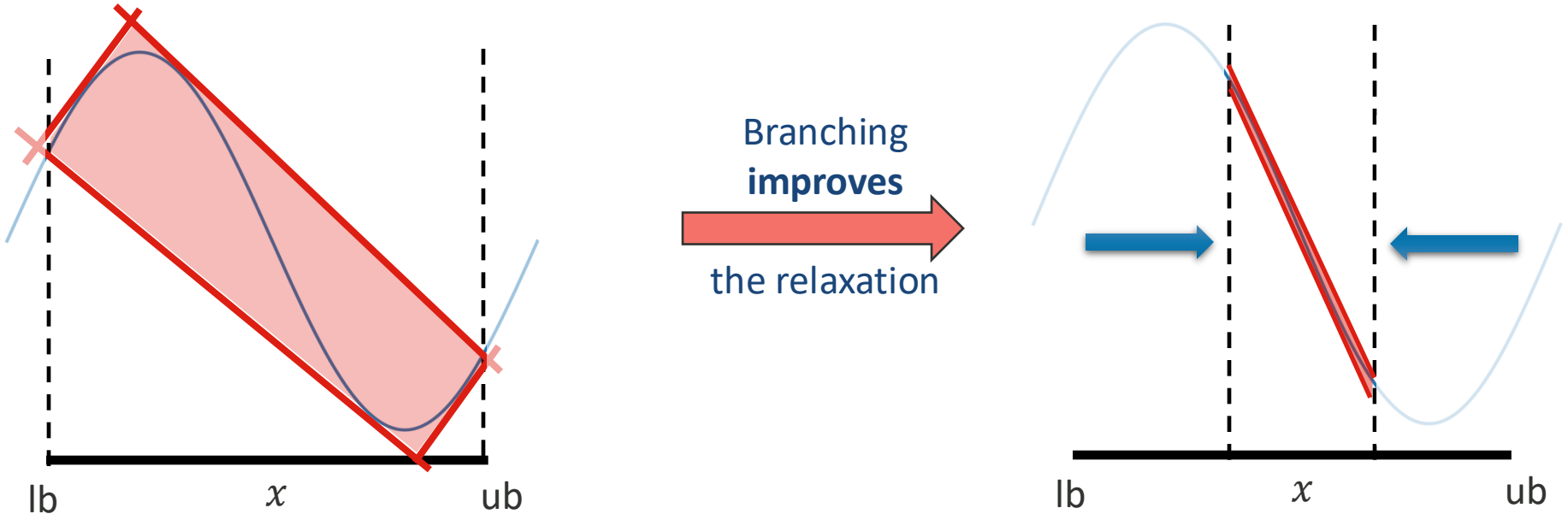
Feasible solution:

- Integer variables have integral LP value
- Nonlinear constraints are satisfied

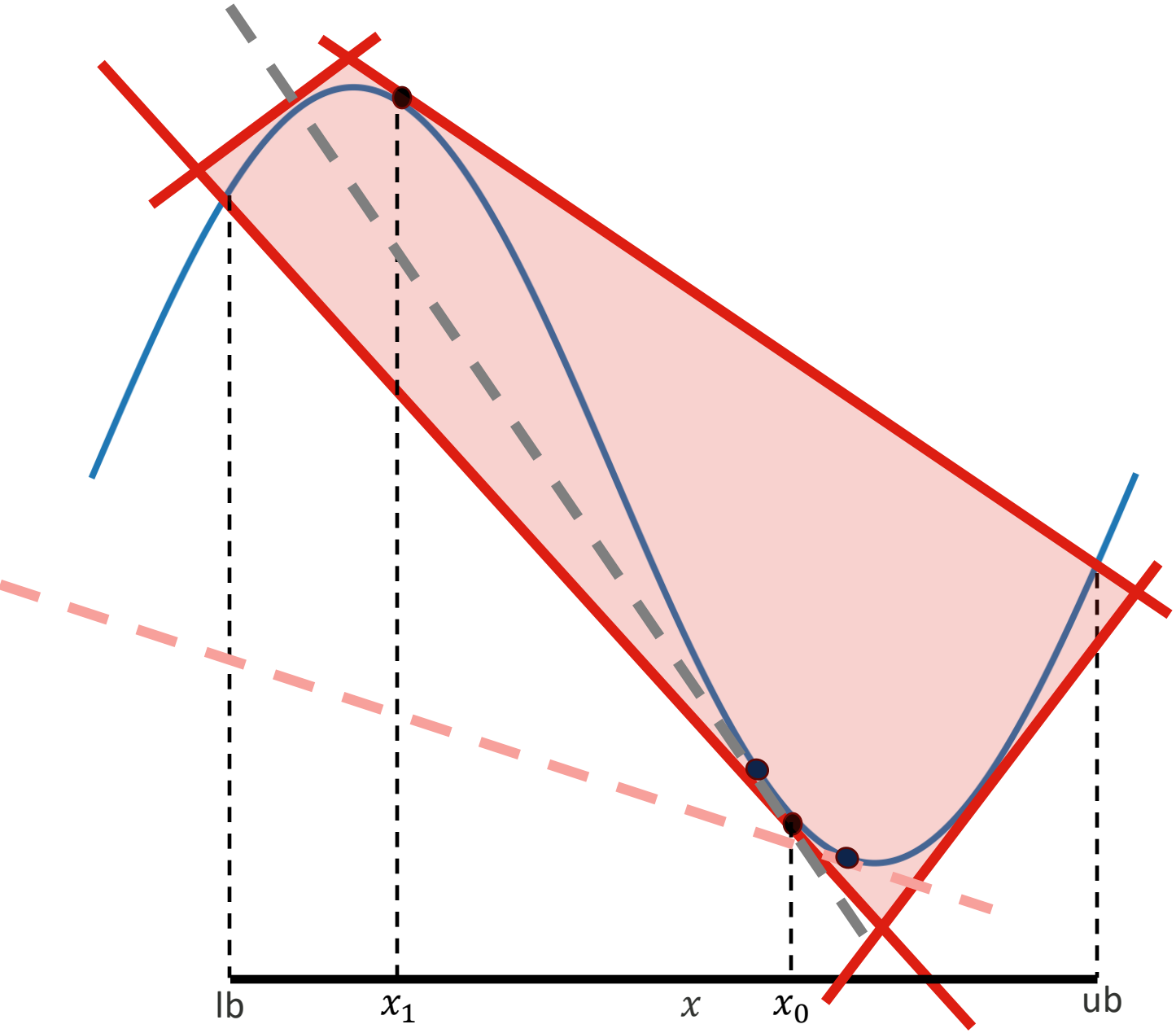
 

# Nonlinear functions

- Handle nonlinear functions via outer approximation
  - Compute envelopes of nonlinear terms other than quadratic terms $x^2$ and $x \cdot y$






GUROBI OPTIMIZATION

# Relaxation of non-convex functions

- $\sin$ is neither convex nor concave on the domain of $x$…
- Lower envelope
  - Compute leftmost solution $x_0$ to $\frac{d}{dx}\sin(x) = \frac{\sin(x) - \sin(\text{lb})}{x - \text{lb}}$
  - Computed $x_0$ defines one tangent
  - Remaining part is convex, use some tangent(s)
- Upper envelope
  - Compute rightmost solution $x_1$ to $\frac{d}{dx}\sin(x) = \frac{\sin(\text{ub}) - \sin(x)}{\text{ub} - x}$
  - Computed $x_1$ defines one tangent
  - Remaining part is concave, use some tangent(s)

# Modeling Nonlinear Functions

- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$
  - Use method `addConstr` or `addGenConstrNL`

```
import gurobipy as gp
from gurobipy import nlfunc

x1 = model.addVar(name="x1")
x2 = model.addVar(name="x2")
y  = model.addVar(name="y")

# y = sin(x1) * x2 + sin(x1)/exp(x2)
model.addConstr(y == gp.nlfunc.sin(x1) * x2
                   + gp.nlfunc.sin(x1) / gp.nlfunc.exp(x2),
                name="nlconstr")

model.addGenConstrNL(y, gp.nlfunc.sin(x1) * x2
                        + gp.nlfunc.sin(x1) / gp.nlfunc.exp(x2),
                     name="nlconstr")
```

**Straightforward and convenient!**

# Modeling Nonlinear Functions (A Bad Alternative)

- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$
  - Use methods `addGenConstrExp(), addGenConstrSin()`

```
x1 = model.addVar(name="x1")
x2 = model.addVar(name="x2")
y  = model.addVar(name="y")

sinx1 = model.addVar(name="sinx1")
# sinx1 = sin(x1)
model.addGenConstrSin(x1, sinx1, name="sinx1")

expx2 = model.addVar(name="expx2")
# expx2 = exp(x2)
model.addGenConstrExp(x2, expx2, name="expx2")

div = model.addVar(name="div")
# div = sin(x1)/exp(x2) <=> div * exp(x2) = sin(x1)
model.addConstr(div * expx2 == sinx1)

model.addConstr(y == sinx1 * x2 + div, name="nlconstr")
```

Requires introduction of multiple auxiliary variables

**This has significant disadvantages!**

# Why is the Aggregated Form Better?

- Convenience
  - no need to model composite nonlinear constraint as a set of linear, quadratic, and univariate nonlinear constraints
- Feasibility
  - In the disaggregated version the composite function may have been violated!
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$
- Disaggregated Gurobi model: $u = \sin(x_1), w = \exp(x_2), v \cdot u = w, y = u \cdot x_2 + v$
- One solution:
  - $x_1 = \frac{\pi}{2}$
  - $x_2 = 0$
  - $u = 1$
  - $w = 1$
  - $v = 1$
  - $y = 1$

Different solution:

$x_1 = \frac{\pi}{2}$

$x_2 = 0$

$u = 0.999999$

$w = 1.000000999998$

$v = 1.000002$

$y = 1.000002$

Violation in composite constraint

$1 + 2 \cdot 10^{-6} \neq 1$

# Version 12 (2024) – Limitations

- Nonlinear constraints **must** be equalities and have an optimization variable defining them

- How to model inequalities $f(x) \leq const$?
  - Introduce a free auxiliary variable $z$
  - $f(x) = z, z \leq const$

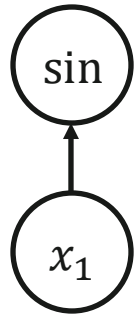
- How to model inequalities $f(x) \leq y$?
  - Introduce a free auxiliary variable $z$
  - $f(x) - y = z, z \leq 0$

- How to model a nonlinear objective $\min f(x)$?
  - Introduce a free auxiliary variable $z$
  - $\min z$ , s. t. $f(x) = z$

# Version 12 (2024) – Nonlinear Non-Python APIs

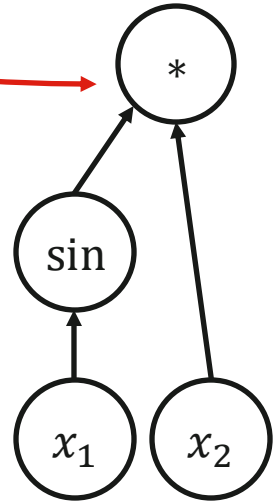
- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
  - But what about other APIs like Java, C, C++, C#?
- Use a so-called expression tree to represent a nonlinear expression
  - What does it look like?
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$



# Version 12 (2024) – Nonlinear APIs

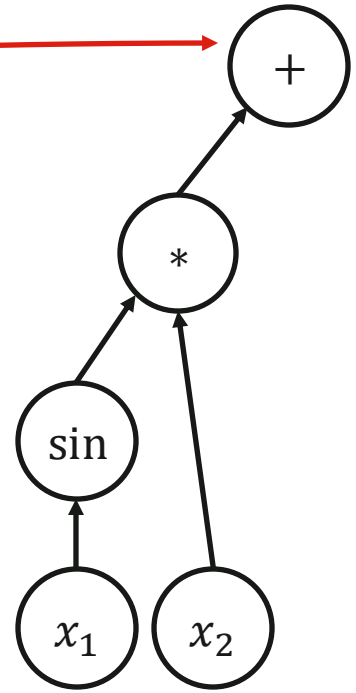
- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
  - But what about other APIs like Java, C, C++, C#?
- Use a so-called expression tree to represent a nonlinear expression
  - How does this look like?
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$



# Version 12 (2024) – Nonlinear APIs

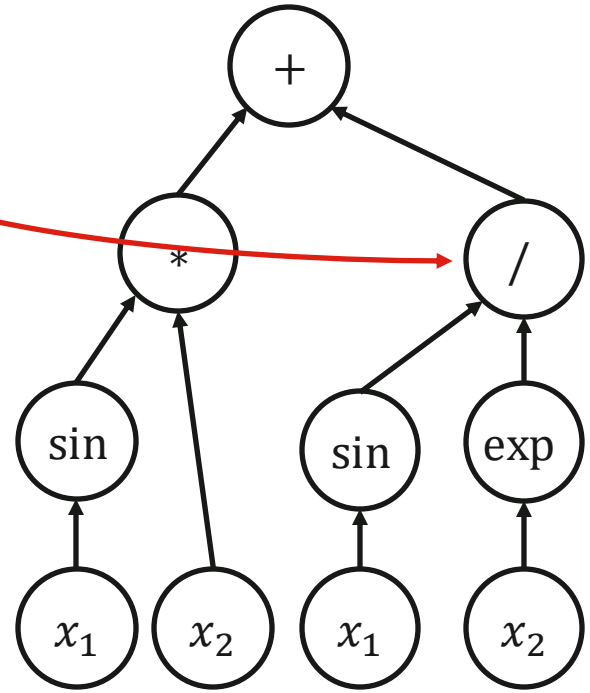
- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
  - But what about other APIs like Java, C, C++, C#?
- Use a so-called expression tree to represent a nonlinear expression
  - How does this look like?
- $y = \sin x_1 \cdot x_2$ $+$ $\sin x_1$ / $\exp x_2$



# Version 12 (2024) – Nonlinear APIs

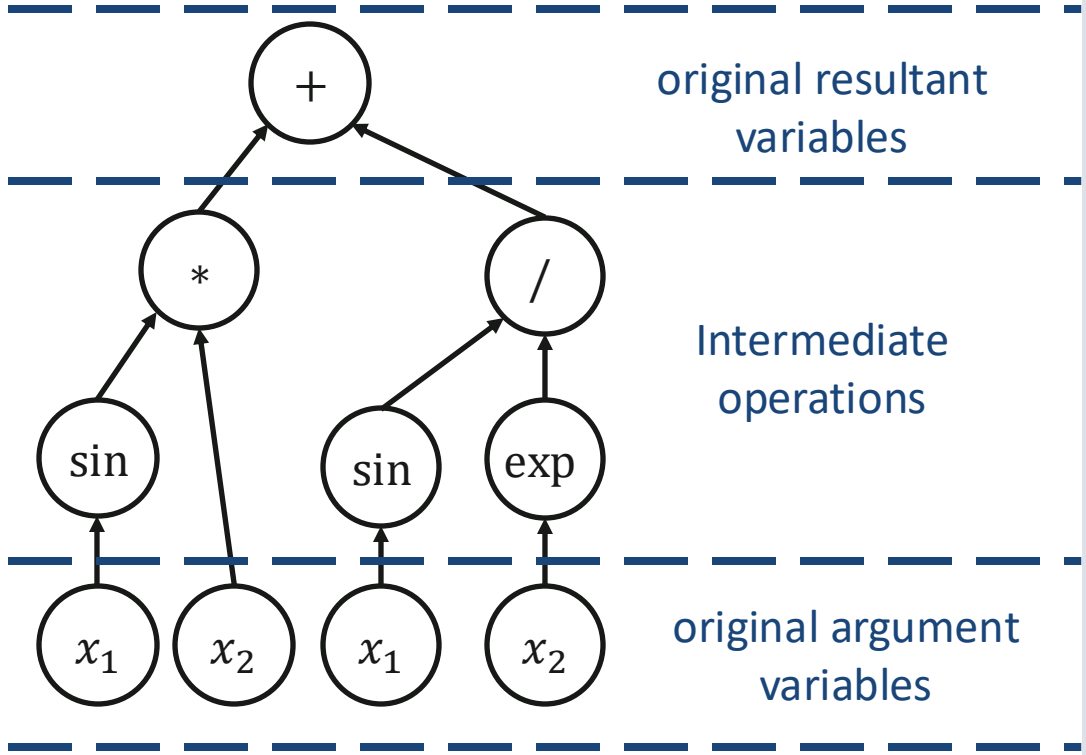
- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
  - But what about other APIs like Java, C, C++, C#?
- Use a so-called expression tree to represent a nonlinear expression
  - How does this look like?
- $y = \sin x_1 \cdot x_2 + \dfrac{\sin x_1}{\exp x_2}$



# Version 12 (2024) – Nonlinear APIs

- We have a new `NLExpr` object in `gurobipy` to handle composite nonlinear expressions
  - But what about other APIs like Java, C, C++, C#?
- Use a so-called expression tree to represent a nonlinear expression
  - How does this look like?
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$



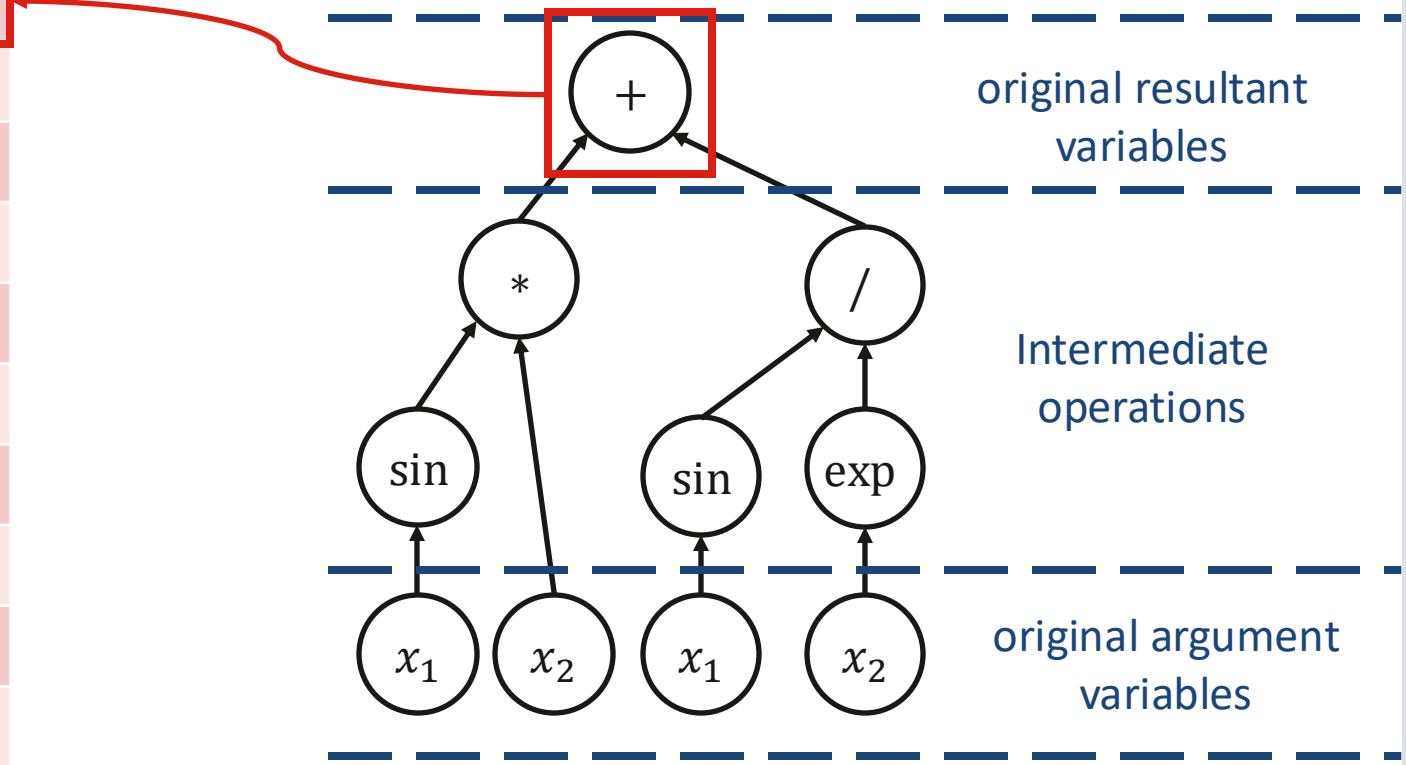

 

# Version 12 (2024) – Nonlinear APIs

We define an expression tree by 3 arrays

- Array size = #nodes in the tree

| index | parent | data | operation |
|---|---|---|---|
| 0 | -1 | - | $+$ |
| 1 | 0 | - | $*$ |
| 2 | 1 | - | sin |
| 3 | 2 | $x_1$ | $var$ |
| 4 | 1 | $x_2$ | $var$ |
| 5 | 0 | - | $/$ |
| 6 | 5 | - | sin |
| 7 | 6 | $x_1$ | $var$ |
| 8 | 5 | - | exp |
| 9 | 8 | $x_2$ | $var$ |



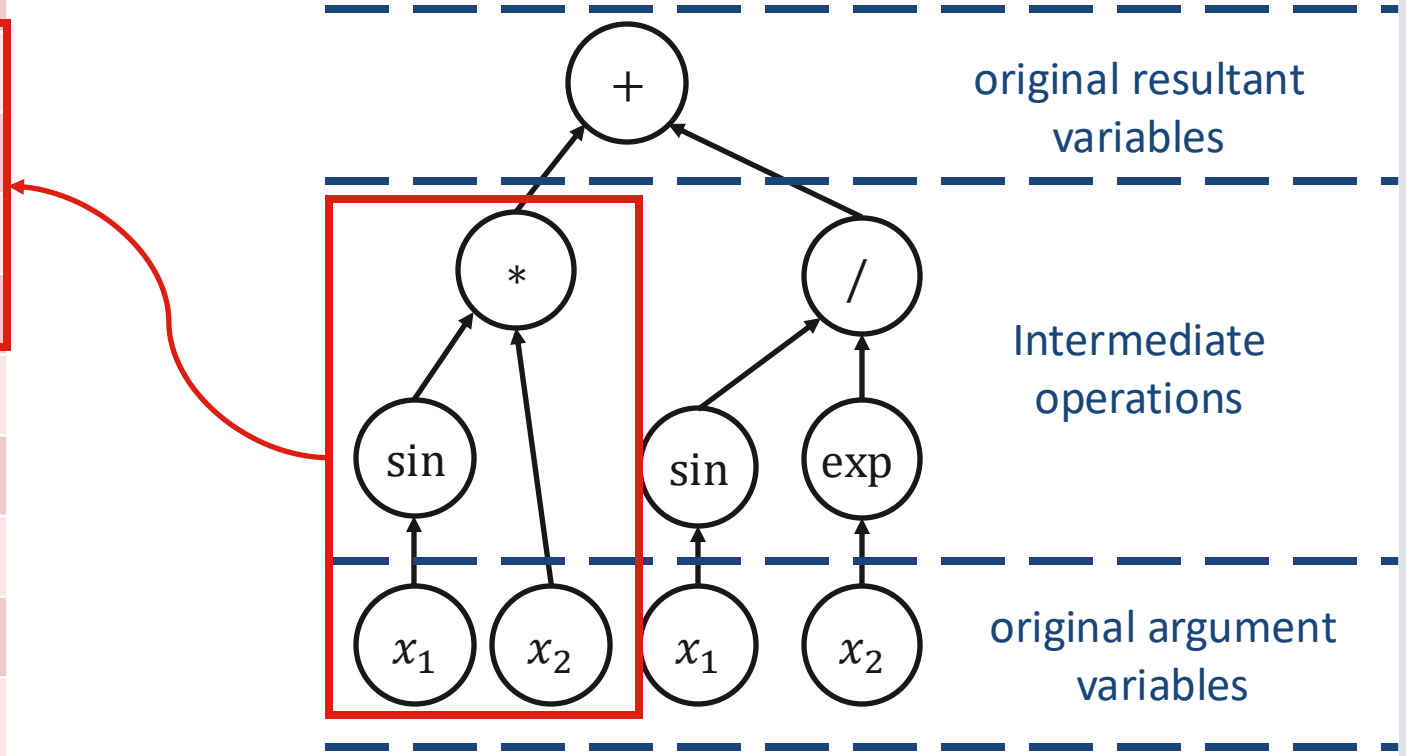

 

# Version 12 (2024) – Nonlinear APIs

We define an expression tree by 3 arrays

- Array size = #nodes in the tree

| index | parent | data | operation |
|---|---|---|---|
| 0 | -1 | - | $+$ |
| 1 | 0 | - | $*$ |
| 2 | 1 | - | sin |
| 3 | 2 | $x_1$ | $var$ |
| 4 | 1 | $x_2$ | $var$ |
| 5 | 0 | - | $/$ |
| 6 | 5 | - | sin |
| 7 | 6 | $x_1$ | $var$ |
| 8 | 5 | - | exp |
| 9 | 8 | $x_2$ | $var$ |

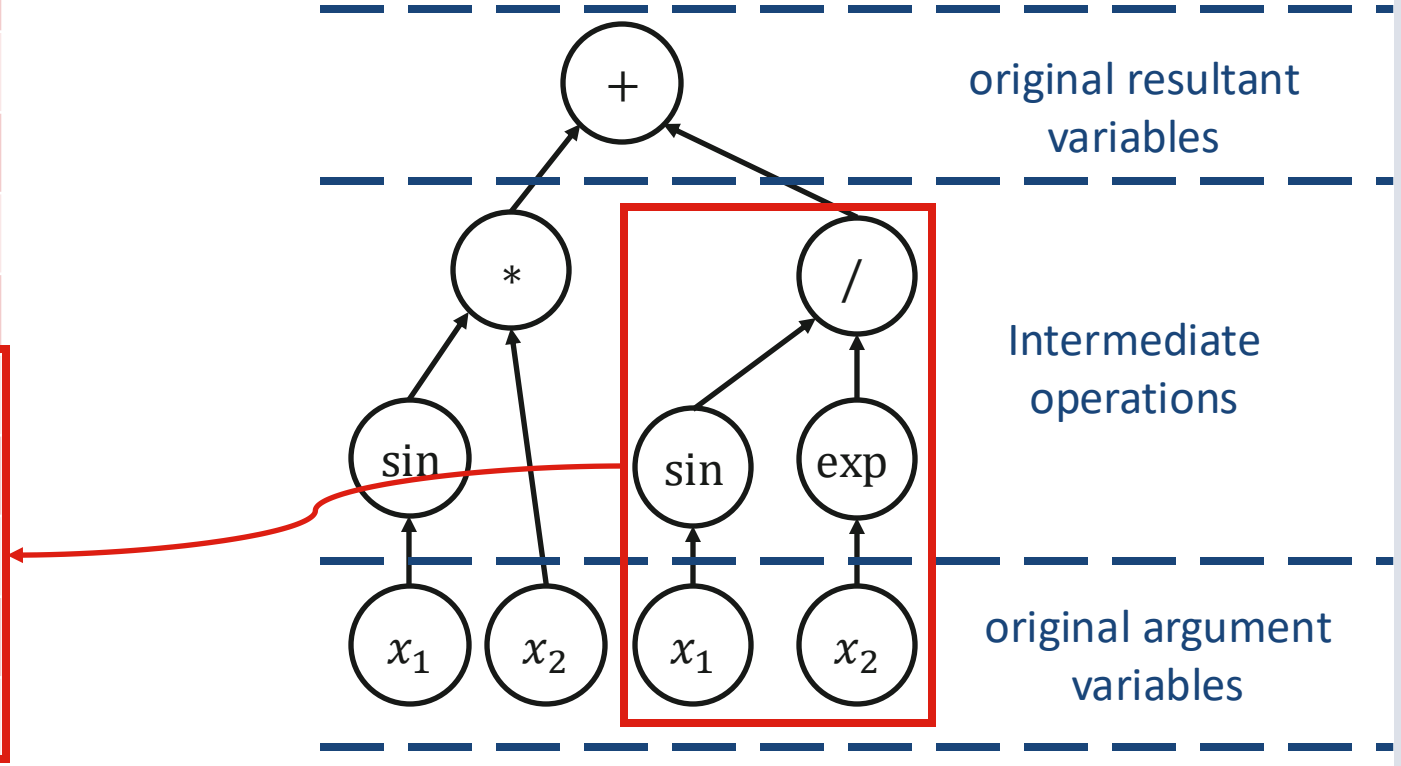


# Version 12 (2024) – Nonlinear APIs

We define an expression tree by 3 arrays

- Array size = #nodes in the tree

| index | parent | data | operation |
|---|---|---|---|
| 0 | -1 | - | $+$ |
| 1 | 0 | - | $*$ |
| 2 | 1 | - | sin |
| 3 | 2 | $x_1$ | $var$ |
| 4 | 1 | $x_2$ | $var$ |
| 5 | 0 | - | $/$ |
| 6 | 5 | - | sin |
| 7 | 6 | $x_1$ | $var$ |
| 8 | 5 | - | exp |
| 9 | 8 | $x_2$ | $var$ |



# Version 13 (2025) – New Nonlinear Operators

In v13 we add two new nonlinear operators

- The hyperbolic tangent function $\mathbf{tanh}(x)$ often used in Neural Networks as an activation function
- The so-called signed power function $\mathbf{signpower}(x, a)$ used in gas network optimization

Why?

- They can be described by already supported nonlinear operators
  - $\mathbf{exp}(x)$ , $\mathbf{log}(x)$ , $\frac{x}{y}$, …
- But in a global optimization algorithm, this is inefficient
  - You can buy yellow and blue paint and mix your own green paint…
  - … or you can just buy green paint directly instead

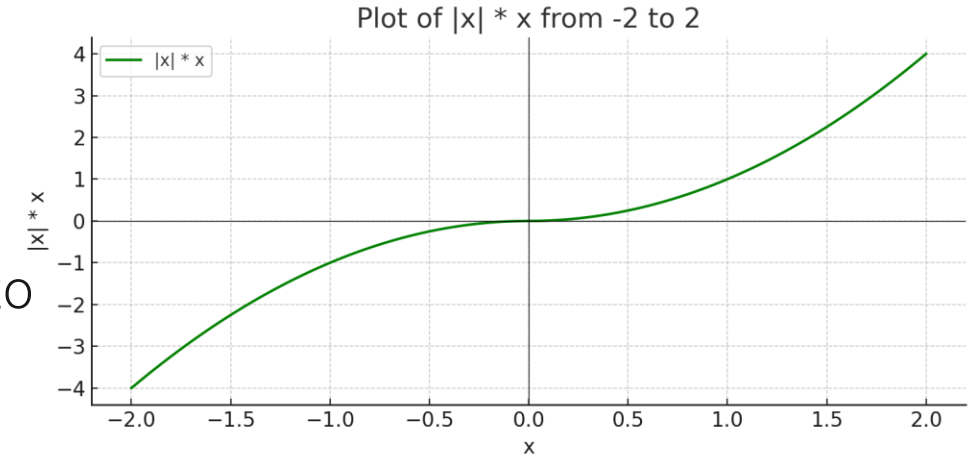
# Version 13 (2025) – Signed Power

The signed power function is defined as

- $\mathbf{signpower}(x, a) = \mathbf{sign}(x) \cdot |x|^a, \mathrm{a} \in \mathbb{R}_{\geq 1}$

It is most often used for $a = 2$ which is then simplified to

- $\mathbf{signpower}(x, 2) = |x| \cdot x$




In v12 (today) one could „translate“ $\mathbf{signpower}(x, 2)$

- Use the general function $|x|$ and the product $|x| \cdot x$
- Use $\sqrt{x^2} \cdot x$ to write $|x| \cdot x$

Both do not work well

In v13 we handle $\mathbf{signpower}(x, a)$ directly

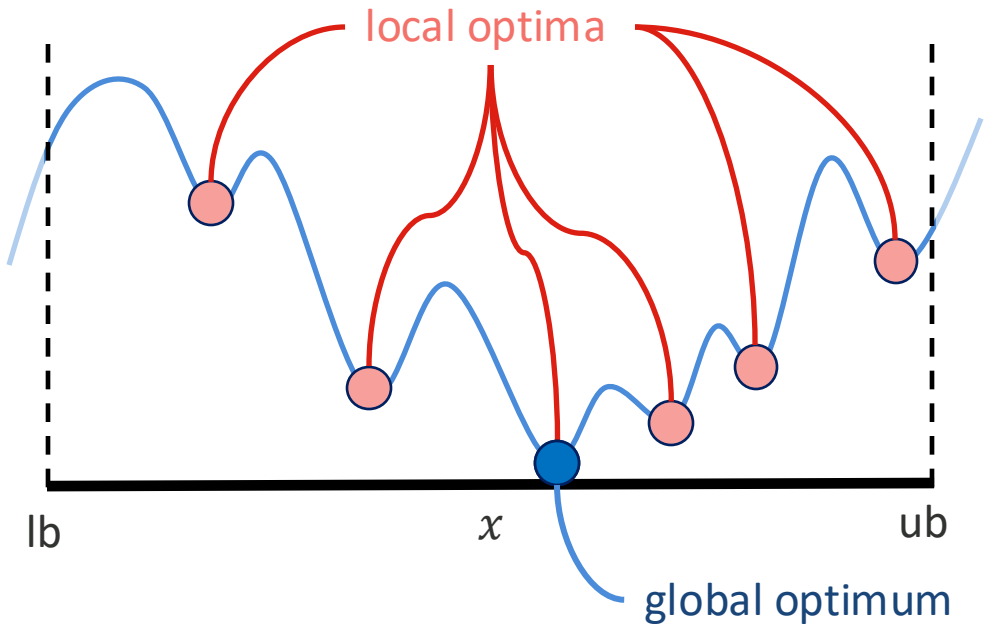
- Positive exponent $a \in \mathbb{R}_{\geq 1}$ allows for modelling of the Darcy-Weisbach and Hazen-Williams equation

# Version 13 (2025) - Global vs Local optimality

- The Branch and Bound algorithm can prove global optimality for a MINLP
- Gurobi's default behavior is to
  - find a globally optimal solution (primal bound)…
  - …and provide a proof of global optimality (dual bound)

- In many applications a "good" feasible point is often enough
  - A good dual bound is can help classifying a given feasible point
- As of today Gurobi's local optimization abilities for MINLPs are limited
- We are working on a dedicated local solver
- We expect a significant improvement in finding feasible points for (MI)NLPs in v13



# Summary of Nonlinearities in Gurobi

GUROBI OPTIMIZATION

## Convex

- Quadratic objective: $\min c^T x + x^T Q x$ with $Q \succcurlyeq 0$
- Quadratic constraints: $a^T x + x^T Q x \leq b$ describing a convex region
  - $Q \succcurlyeq 0$ is only one of many recognized cases

## Non-convex

- Discrete objects: $x \in \mathbb{Z}$, SOS constraints
- Bilinear terms: $z = x \cdot y$
  - Non-convex quadratic constraints: $a^T x + x^T Q x \leq b$
- Nonlinear functions
  - $y = f(x), x \in \mathbb{R}^n$
  - $y = f(x)$ with univariate $f$ given as $\exp, \log, \sin$

GUROBI OPTIMIZATION

# Questions?

Jaromił Najman
Senior Developer

najman@gurobi.com



© 2025 Gurobi Optimization, LLC. Confidential, All Rights Reserved | 26

# Non-Convex QP, QCP, MIQP, MIQCP

GUROBI OPTIMIZATION

- Prior Gurobi versions could deal with two types of non-convexity
  - Integer variables
  - SOS constraints
- Gurobi 9.0 can deal with a third type of non-convexity
  - Bilinear constraints
- These non-convexities are treated by
  - Cutting planes
  - Branching

- Translation of non-convex quadratic constraints into bilinear constraints

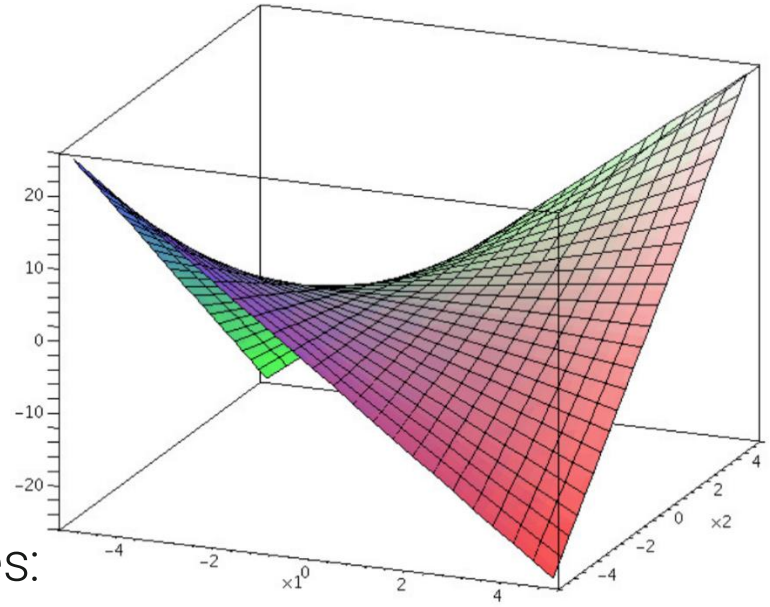
$$3x_1^2 - 7x_1x_2 + 2x_1x_3 - x_2^2 + 3x_2x_3 - 5x_3^2 = 12 \qquad \text{(non-convex Q constraint)}$$

$$\Rightarrow \quad z_{11} := x_1^2, z_{12} := x_1x_2, z_{13} := x_1x_3, z_{22} := x_2^2, z_{23} := x_2x_3, z_{33} := x_3^2 \qquad \text{(6 bilinear constraints)}$$

$$3z_{11} - 7z_{12} + 2z_{13} - z_{22} + 3z_{23} - 5z_{33} = 12 \qquad \text{(linear constraint)}$$

# LP Relaxation of Bilinear Constraints

- Mixed product case: $-z + xy = 0$



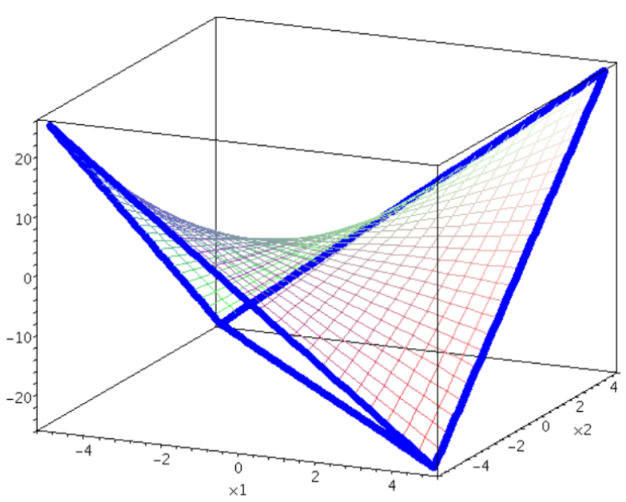

- McCormick lower and upper envelopes:

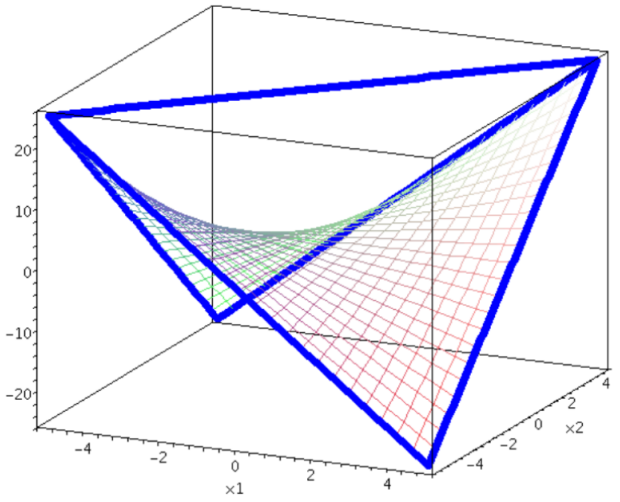
$$-z + l_x y + l_y x \leq l_x l_y$$
$$-z + u_x y + u_y x \leq u_x u_y$$

$$-z + u_x y + l_y x \geq u_x l_y$$
$$-z + l_x y + u_y x \geq l_x u_y$$

coefficients depend on local bounds

pictures from Costa and Liberti: "Relaxations of multilinear convex envelopes: dual is better than primal"



# NonConvex Parameter

- `NonConvex`:
  - `-1`: automatic
  - `0`: return error if original model has nonconvex Q objective or constraints
  - `1`: return error if presolved model has nonconvex Q that cannot be linearized
  - `2`: accept nonconvex Q by using a bilinear transformation
- Gurobi 10.0 default (-1): equivalent to 1
- Gurobi 11.0 default (-1): essentially equivalent to 2

- Change can break user code or be at least surprising to users!
- Often, nonconvexity is still a sign of an error in model or data
  - Users now may want to explicitly set `NonConvex=1`

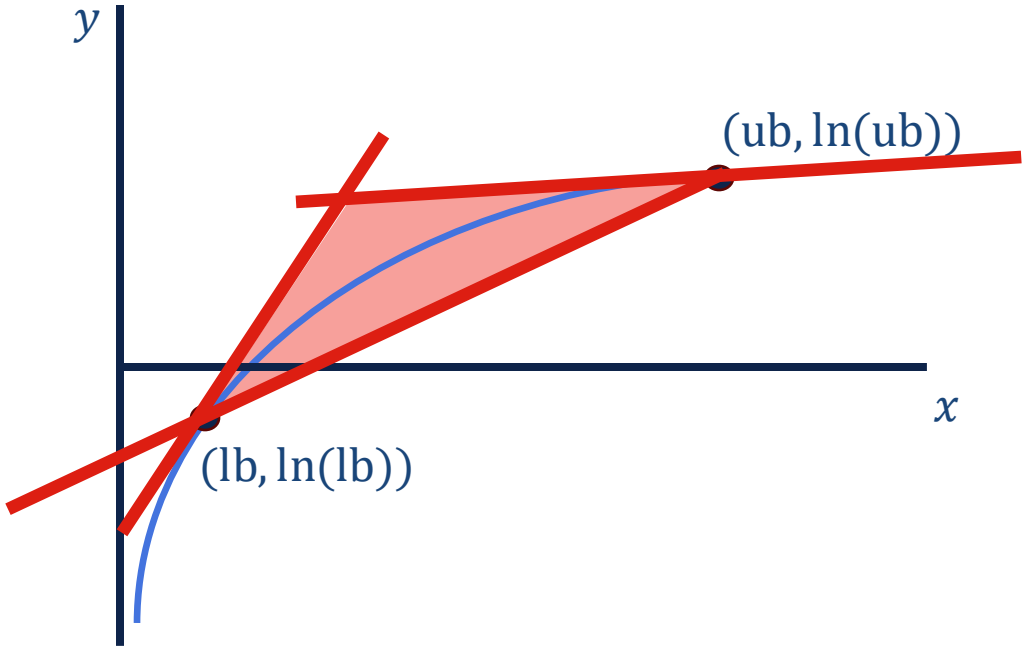
# FuncNonlinear Parameter and Attributes

- Existing general function constraints attributes to control PWL approximation:
  - `FuncPieces`
  - `FuncPieceError`
  - `FuncPieceLength`
  - `FuncPieceRatio`
- Default behavior of `FuncPieces` is now to use relative error approach
  - Was mainly restricting total number of pieces in Gurobi 10.0
- New `FuncNonlinear` attribute to switch between PWL and outer approximation:
  - `-1`: behavior defined by FuncNonlinear parameter (default)
  - `0`: use static PWL approximation
  - `1`: use dynamic outer approximation
- New `FuncNonlinear` parameter to control default (-1) of attributes:
  - `0`: use static PWL approximation (default)
  - `1`: use dynamic outer approximation

# Relaxation of Nonlinear Functions

- Some convex envelopes are easier to approximate than others
  - Example: $y = \ln(x)$, a concave function, $l \leq x \leq u$
  - Lower envelope is given by secant through $\ln(l)$ and $\ln(u)$
  - Upper envelope is constructed by tangents



# Version 11 (2023) - Limitations

- Need to model composite nonlinear constraint as a set of
  - linear,
  - quadratic, and
  - univariate nonlinear constraints
- Each individual constraint is subject to feasibility tolerance
- Result could be that composite constraint is violated much more
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$
- Gurobi model: $u = \sin(x_1), w = \exp(x_2), v \cdot u = w, y = u \cdot x_2 + v$
- One solution:
  - $x_1 = \frac{\pi}{2}$
  - $x_2 = 0$
  - $u = 1$
  - $w = 1$
  - $v = 1$
  - $y = 1$

Different solution:

$x_1 = \frac{\pi}{2}$

$x_2 = 0$

$u = 0.999999$

$w = 1.000000999998$

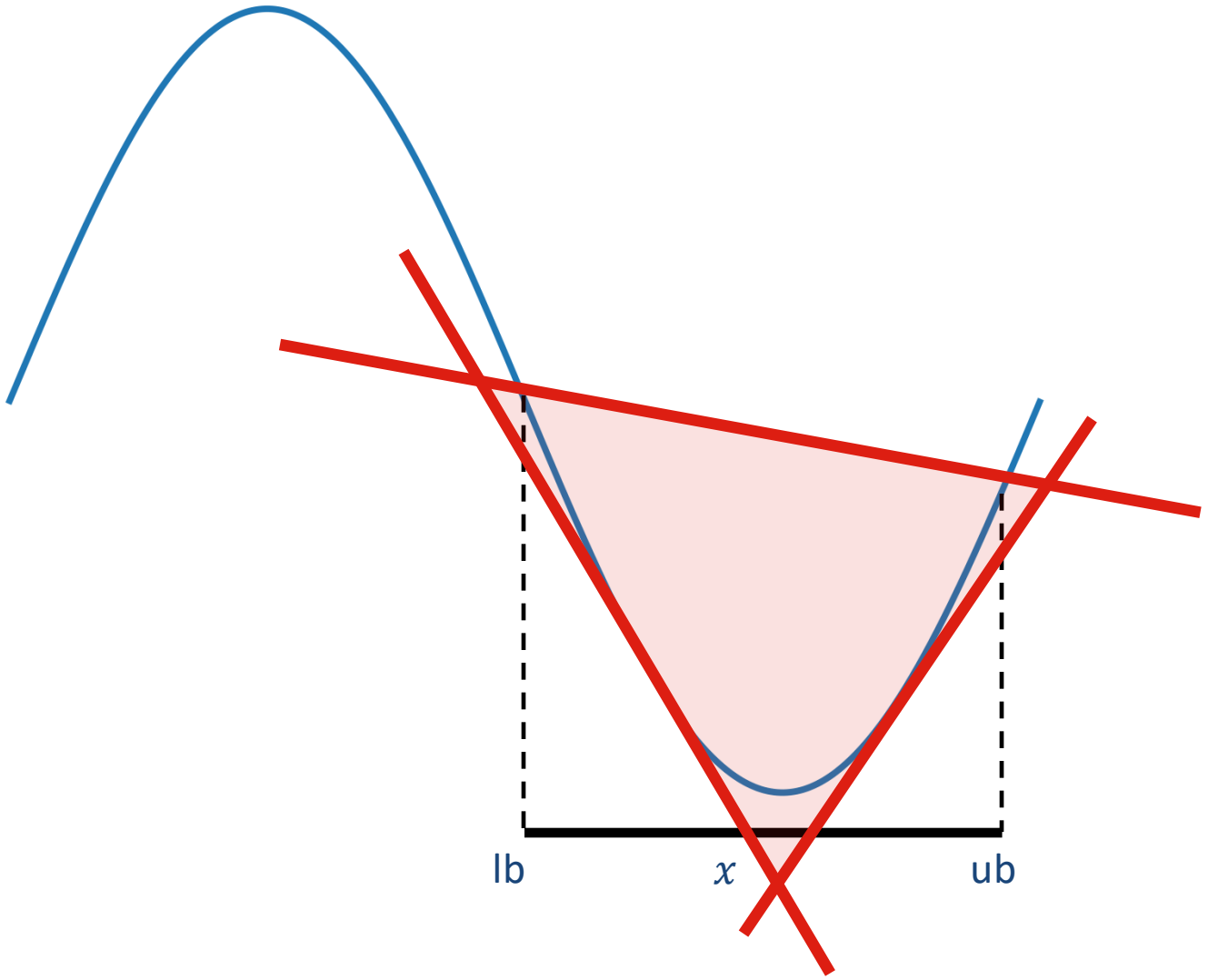
$v = 1.000002$

$y = 1.000002$

Violation in composite constraint

$1 + 2 \cdot 10^{-6} \neq 1$




GUROBI OPTIMIZATION

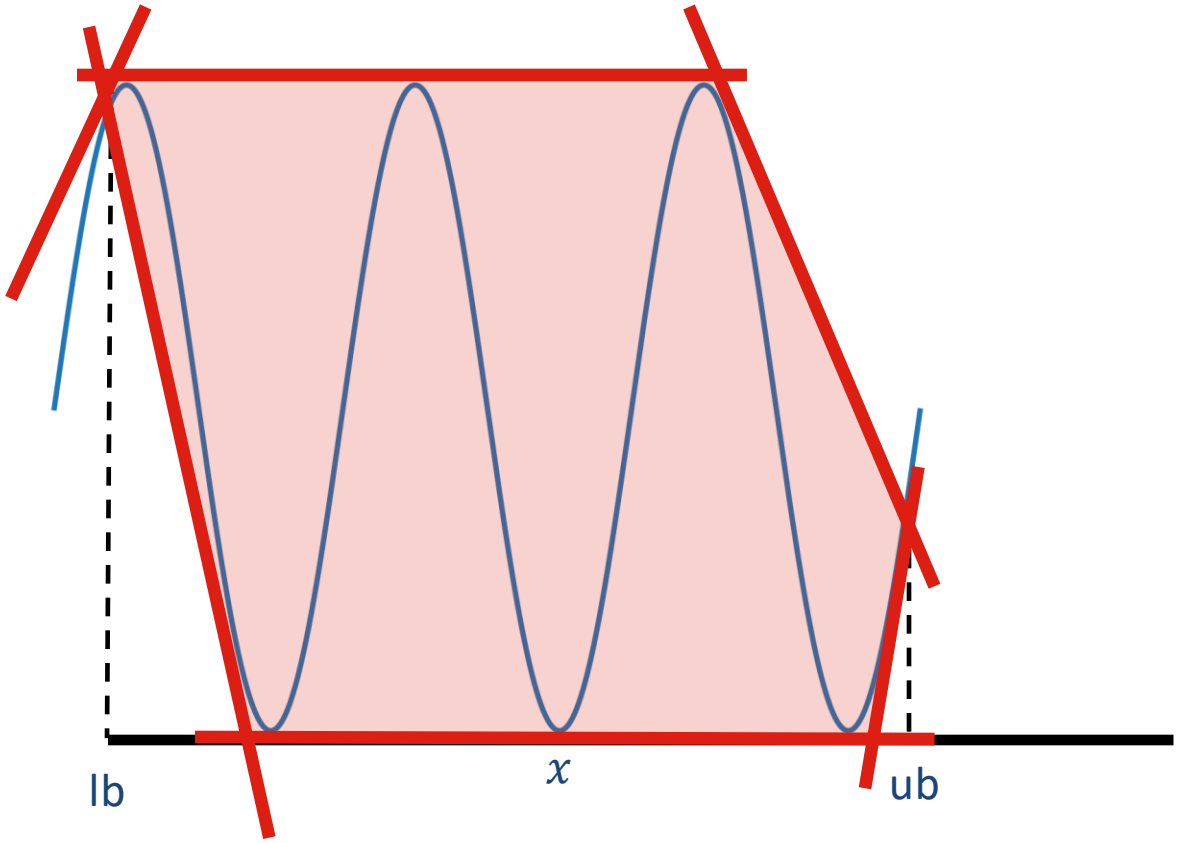
# Extend to More Complex Functions

If sin is convex within the bounds of $x$ …

- Upper envelope is given by secant through $f(\mathrm{lb})$ and $f(\mathrm{ub})$
- Lower envelope constructed by tangents to $\sin$
- Resulting hyperplanes added to LP
- Shaded in red: Relaxation of $y = f(x)$

- Similar if $\sin$ is *concave* on the domain of $x$
- Adding more tangents at various points improves the relaxation




GUROBI OPTIMIZATION

# “Large” Domains

- Not much to get from the relaxation if domain of $x$ is large
- Branching on $x$ tightens the relaxation quickly!
- Tighter initial bounds will speed up performance

# Version 13 (2025) – TANH

The hyperbolic tangent function $\tanh(x)$ is defined as

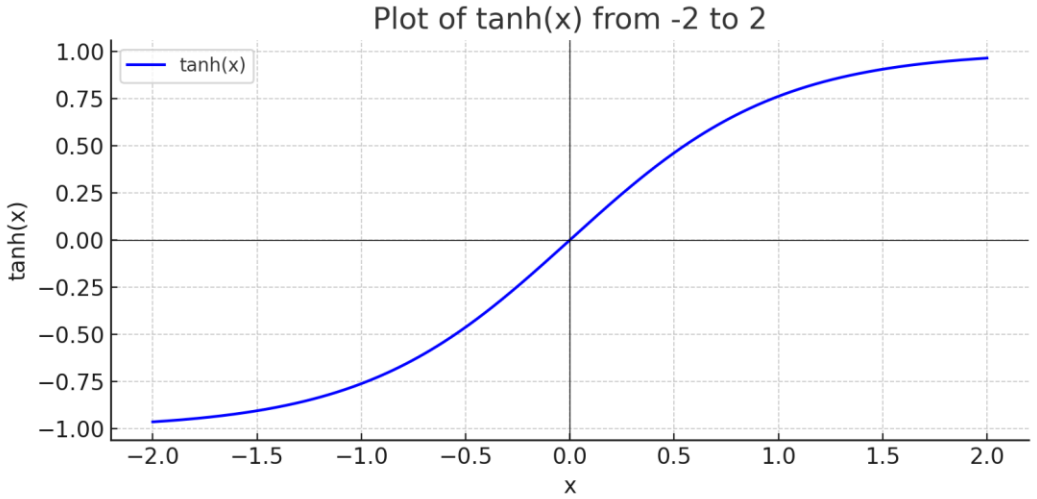
- $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

The construction of a relaxation of the $\exp(x)$ formulation would require the introduction of multiple auxiliary variables given as

$$
\begin{aligned}
z_1 &= \exp(x) \\
z_2 &= \exp(-x) \\
z_3 &= z_1 - z_2 \\
z_4 &= z_1 + z_2 \\
z_5 &= \frac{z_3}{z_4}
\end{aligned}
$$

The above disaggregated formulation does not only increase the dimensionality of the model, but also hides many numerical traps

- We handle $\tanh(x)$ directly instead



# Version 11 - Limitations

- Need to model composite nonlinear constraint as a set of
  - linear
  - quadratic
  - and univariate nonlinear constraints
- Each individual constraint is subject to feasibility tolerance
- Result could be that composite constraint is violated much more
- $y = \sin x_1 \cdot x_2 + \frac{\sin x_1}{\exp x_2}$
- Gurobi model: $u = \sin(x_1), w = \exp(x_2), v \cdot u = w, y = u \cdot x_2 + v$
- One solution:
  - $x_1 = \frac{\pi}{2}$
  - $x_2 = 0$
  - $u = 1$
  - $w = 1$
  - $v = 1$
  - $y = 1$

Different solution:

$x_1 = \frac{\pi}{2}$

$x_2 = 0$

$u = 0.999999$

$w = 1.000000999998$

$v = 1.000002$

$y = 1.000002$

Violation in composite constraint

$1 + 2 \cdot 10^{-6} \neq 1$