Computer Methods (MAE 3403)

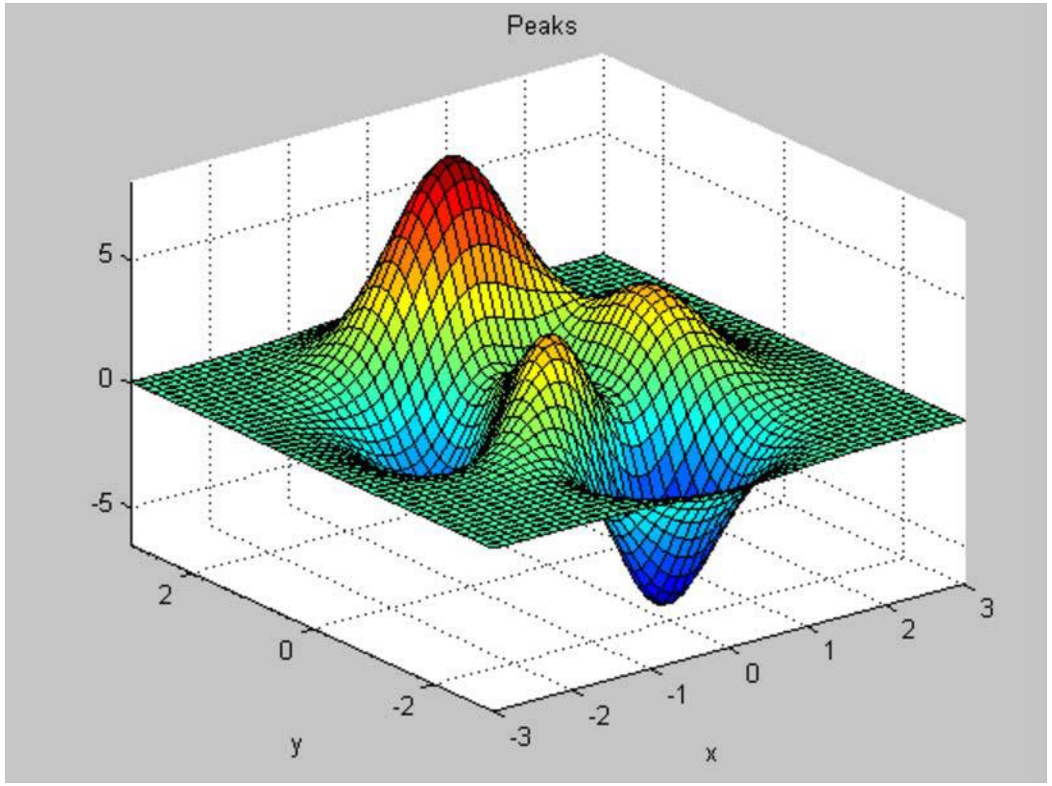
Optimization

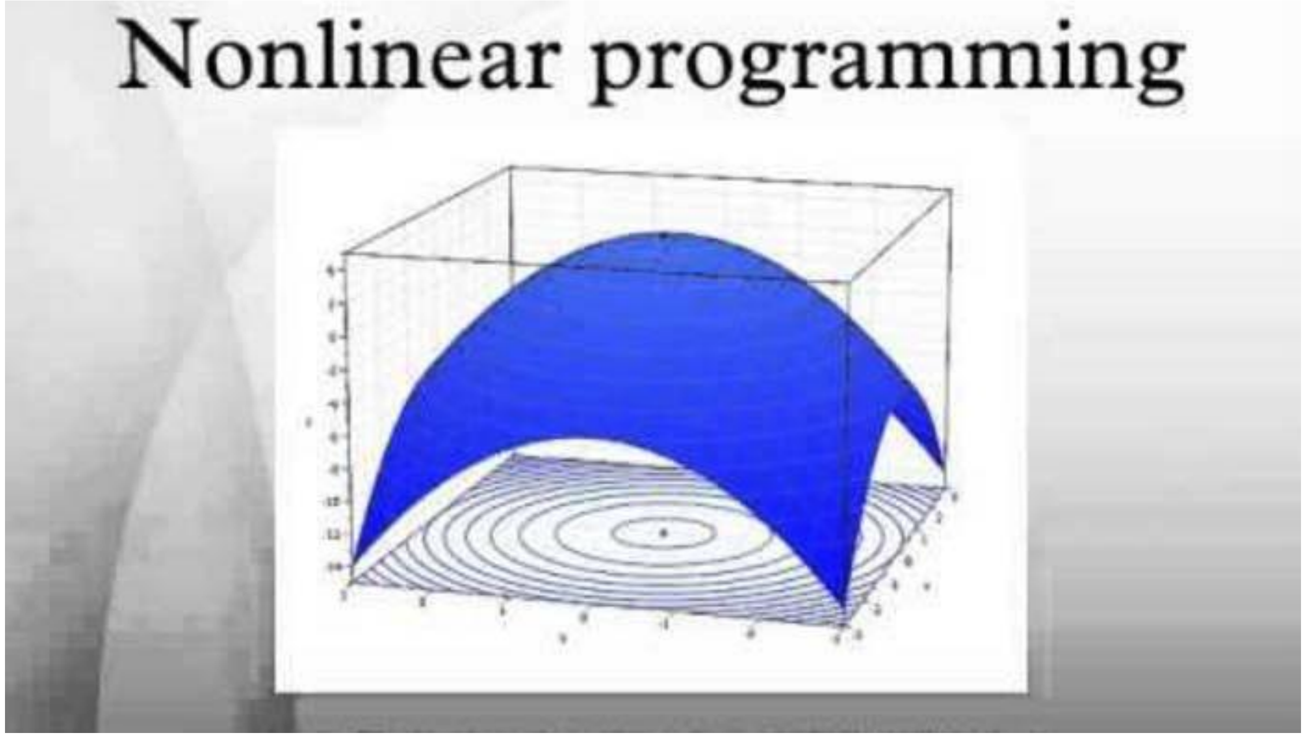
Optimization: VERY IMPORTANT

- End the semester in a high note
- Optimization is a critical component in many, many engineering problems
- It is also an active research area, particularly with the advancement of ML/AI technologies

Optimization

- *Find the Location of the Maximum or Minimum Value of a function*





Engineering applications

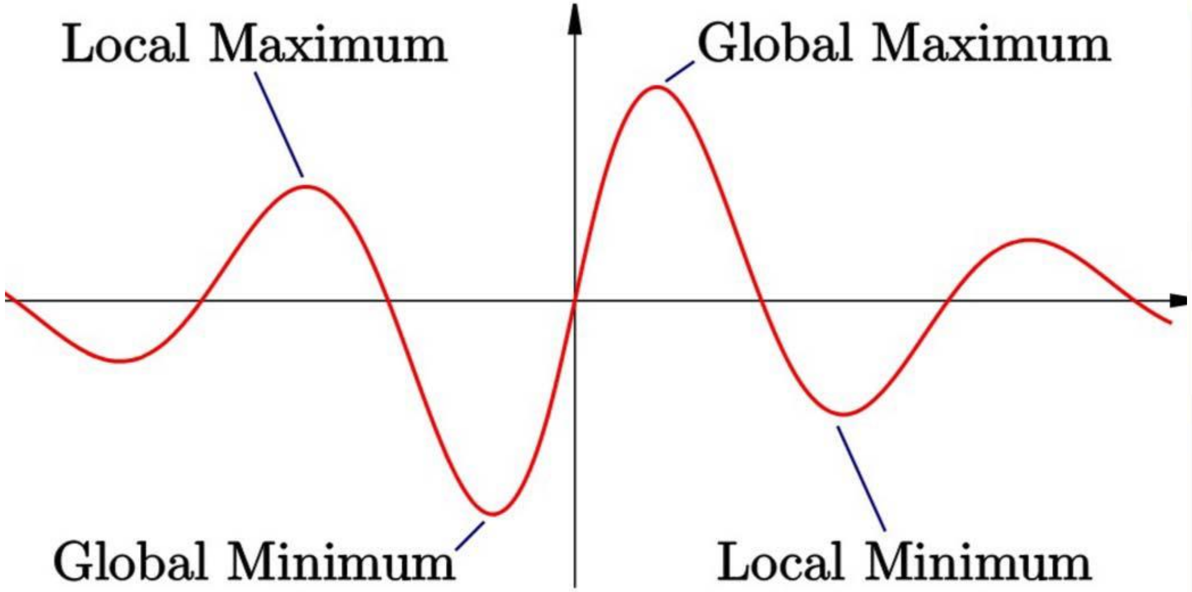
- Choose the values of design parameters or operating parameters to achieve maximum (GOOD) or minimum (BAD) subject to constraints
- GOOD and BAD for:
  - Designs of airplanes, race cars
  - Performance of thermal systems
  - Collisions between vehicles
  - …

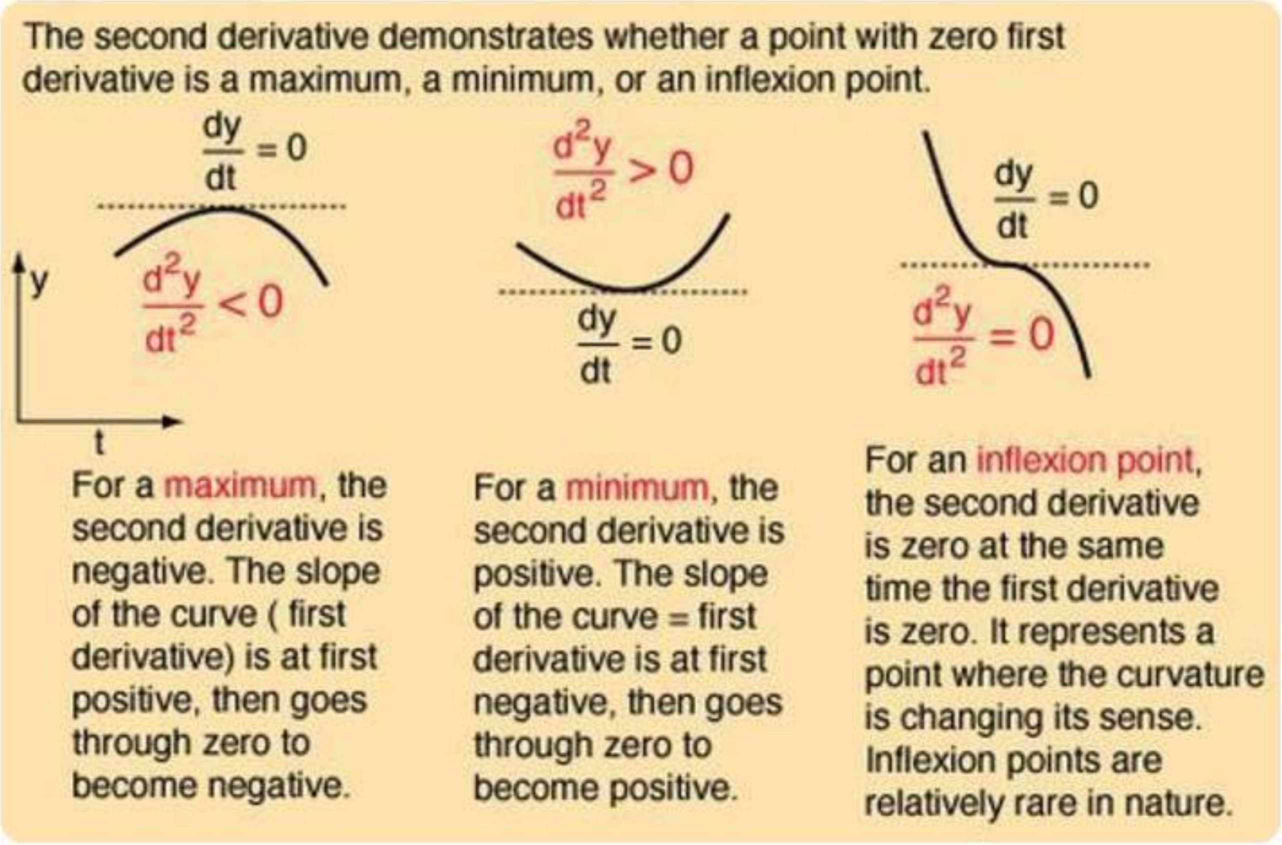
Minimize or Maximize

- To unify the syntax, we consider only minimization problems.

- Convert maximization problems to minimization:
  - maximize **f(x)** subject to constraints on **x** = minimize **–f(x)** subject to the same set of constraints on **x**

One variable optimization





- First order derivative
- Second order derivative: Hessian

Typical form of an optimization problem

$$\min f(\boldsymbol{x})$$

subject to $g_i(\boldsymbol{x}) \geq 0, i = 1, 2, \ldots$
$h_j(\boldsymbol{x}) = 0, j = 1, 2, \ldots$
$\boldsymbol{LB} \leq \boldsymbol{x} \leq \boldsymbol{UB}$

f(**x**): objective function to be minimized, **a scalar function**
x: decision variable, **an array of** design parameters
$g_i(\boldsymbol{x}) \geq 0$: 1 inequality constraint.
$h_j(\boldsymbol{x}) = 0$: 1 equality constraint.
$\boldsymbol{LB}$, $\boldsymbol{UB}$: lower and upper bounds of each element of $\boldsymbol{x}$

What if inequality constraints are in the form of $\bar{\boldsymbol{g}}_i(x) \leq 0$?
Let $g_i(x) = -\bar{\boldsymbol{g}}_i(x)$. Then $g_i(x) \geq 0$ is equivalent to $\bar{g}_i(x) \leq 0$

scipy.optimize.minimize

- fun: f(x), objective function to be minimized
- x0: some initial guess of an optimal decision variable x
- args: extra parameters passed to fun
- method: choice of solvers

scipy.optimize.

minimize

`minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)`

Minimization of scalar function of one or more variables. [source]

Parameters:

fun : *callable*

The objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is a 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where `n` is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

method : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'COBYQA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)
- custom - a callable object, see below for description.

continued

- bounds: (min, max) of each element of x
  - x must be greater than 0
  - x must be within a range
- constraints:
  - inequality constraints
  - equality constraints

bounds : *sequence or* `Bounds`*, optional*

Bounds on variables for Nelder-Mead, L-BFGS-B, TNC, SLSQP, Powell, trust-constr, COBYLA, and COBYQA methods. There are two ways to specify the bounds:

1. Instance of `Bounds` class.
2. Sequence of `(min, max)` pairs for each element in *x*. None is used to specify no bound.

constraints : *{Constraint, dict} or List of {Constraint, dict}, optional*

Constraints definition. Only for COBYLA, COBYQA, SLSQP and trust-constr.

Constraints for 'trust-constr' and 'cobyqa' are defined as a single object or a list of objects specifying constraints to the optimization problem. Available constraints are:

- `LinearConstraint`
- `NonlinearConstraint`

Constraints for COBYLA, SLSQP are defined as a list of dictionaries. Each dictionary with fields:

type : *str*

Constraint type: 'eq' for equality, 'ineq' for inequality.

fun : *callable*

The function defining the constraint.

jac : *callable, optional*

The Jacobian of *fun* (only for SLSQP).

args : *sequence, optional*

Extra arguments to be passed to the function and Jacobian.

Equality constraint means that the constraint function result is to be zero whereas inequality means that it is to be non-negative. Note that COBYLA only supports inequality constraints.

tol : *float, optional*

Tolerance for termination. When *tol* is specified, the selected minimization algorithm sets some relevant solver-specific tolerance(s) equal to *tol*. For detailed control, use solver-specific options.

options : *dict, optional*

A dictionary of solver options. All methods except *TNC* accept the following generic options:

maxiter : *int*

scipy.optimize.minimize returns

res: *OptimizeResult*

The optimization result represented as an OptimizeResult object.

Important attributes are:
x the solution array,
success a Boolean flag indicating if the optimizer exited successfully,
message which describes the cause of the termination.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

Rosenbrock function of N variables

Nelder-Mead Simplex algorithm (`method='Nelder-Mead'`)

In the example below, the `minimize` routine is used with the *Nelder-Mead* simplex algorithm (selected through the `method` parameter):

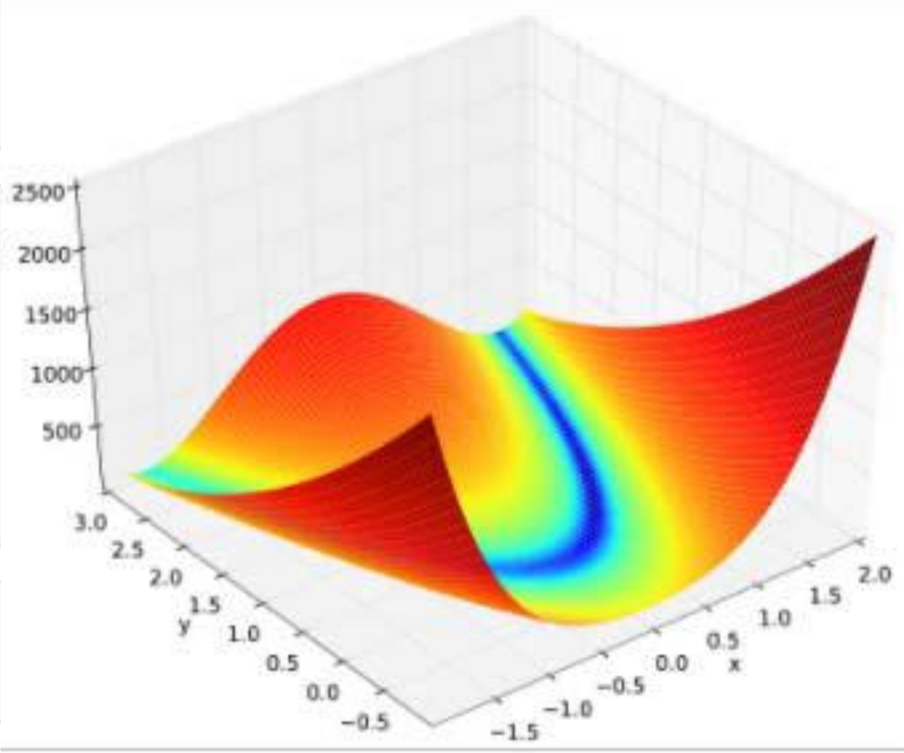
$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$$

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

```
>>> import numpy as np
>>> from scipy.optimize import minimize
```

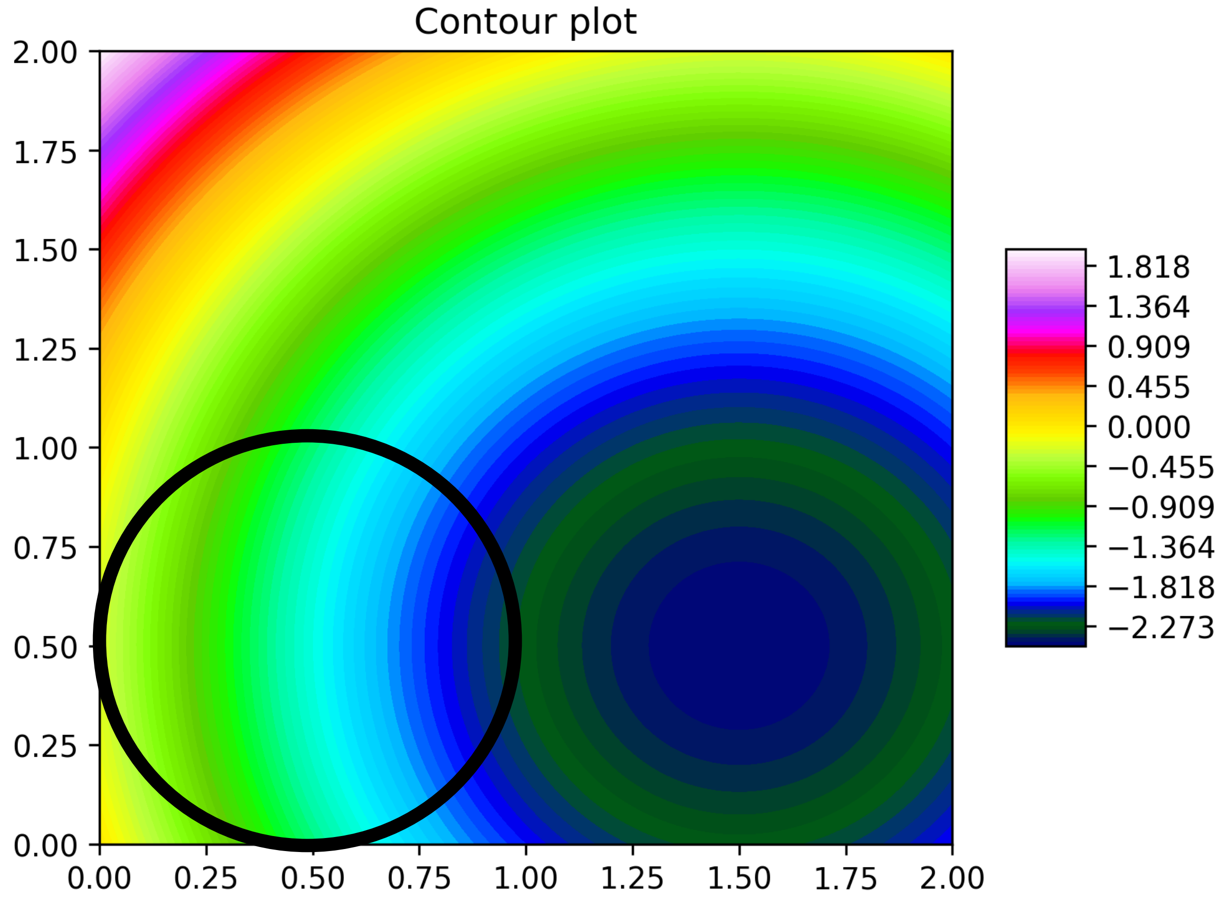
```
>>> def rosen(x):
...     """The Rosenbrock function"""
...     return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
```

```
>>> x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
>>> res = minimize(rosen, x0, method='nelder-mead',
...                options={'xatol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 339
         Function evaluations: 571
```




```
>>> print(res.x)
[1. 1. 1. 1. 1.]
```

With constraints



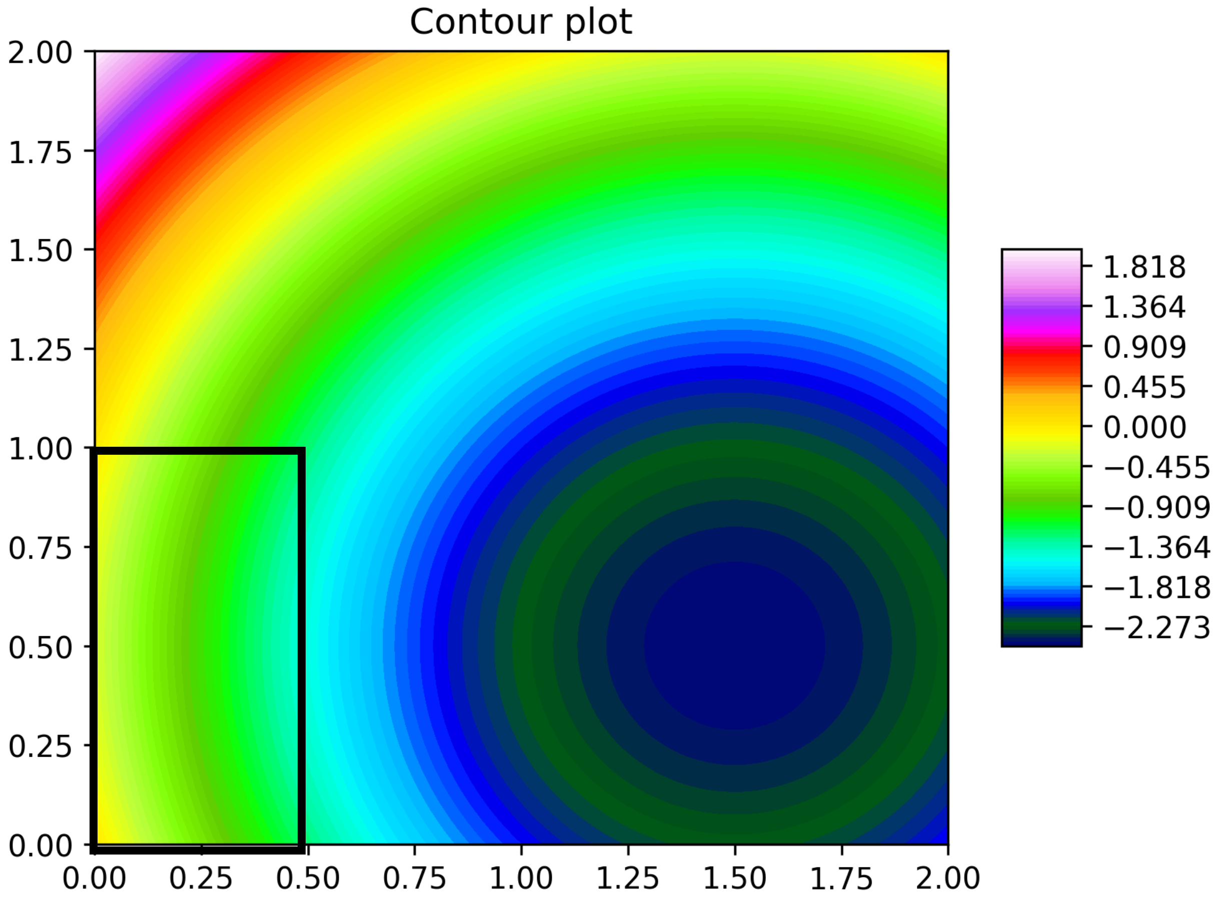
- min $z(x,y)=y^2-y+x^2-3x$ subject to $(x-0.5)^2 + (y-0.5)^2 <= 0.5^2$
- Convert the constraint to

  $0.5^2 - (x-0.5)^2 - (y-0.5)^2 >= 0$

```
cons = ({'type': 'ineq', \
'fun': lambda x:  0.25 - (x[0] - 0.5) ** 2 - (x[1] - 0.5) ** 2})
```

- **DO NOT set a solver**

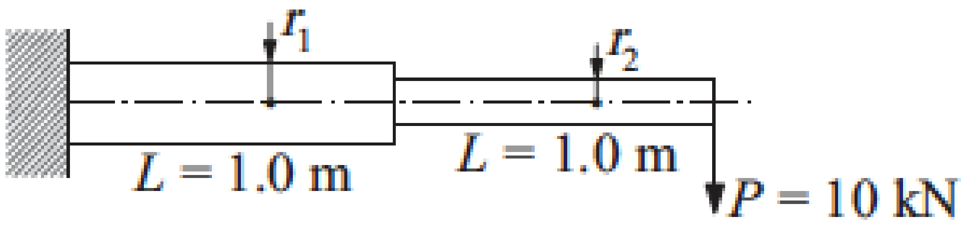
With bounds



- Write it in the “bounds” argument
- min $z(x,y)=y^2-y+x^2-3x$
  subject to 0<=x<=0.5, 0<=y<=1
- bounds = [(xlb, xub), (ylb, yub)]

Another example

- min $(x-1)^2 + (y-2.5)^2$
- subject to
  - $x - 2y+2 >=0$,
  - $-x - 2y + 6>=0$,
  - $-x+2y + 2>=0$
  - $x > 0, y > 0$



Final example

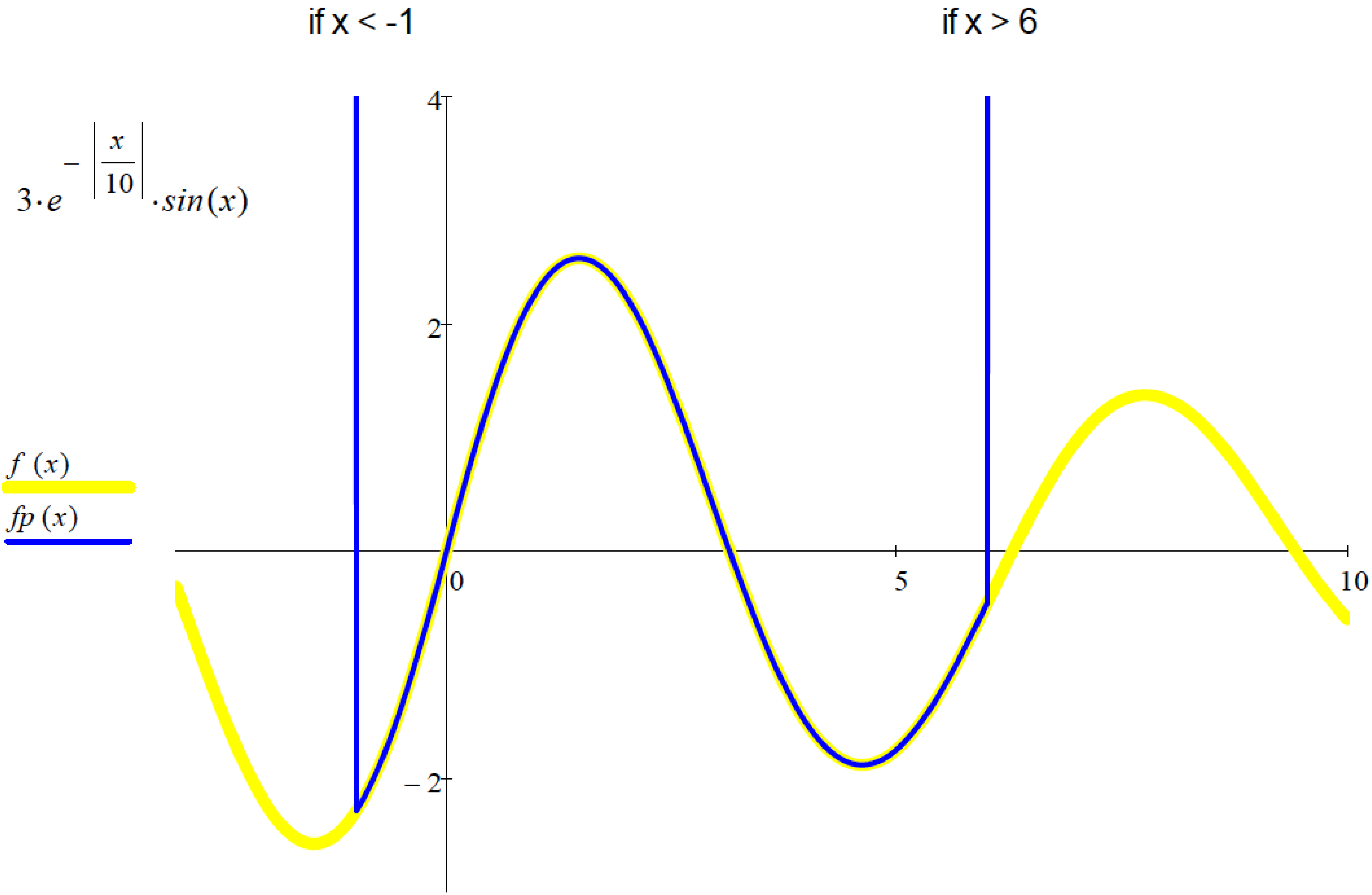
- The cantilever beam of the circular cross section is to have the smallest volume subject to constraints:
  - $\sigma_1 \leq 180$ Mpa, $\sigma_1 = \frac{8\mathrm{PL}}{\pi r_1^3}$
  - $\sigma_2 \leq 180$ Mpa, $\sigma_2 = \frac{4\mathrm{PL}}{\pi r_2^3}$
  - $\delta \leq 25$ mm, $\delta = \frac{\mathrm{PL}^3}{3\pi E}(\frac{7}{r_1^4} + \frac{1}{r_2^4})$
  - E = 200 Gpa. P = 10 kN. Determine r_1 and r_2.

Formulation

- Minimize the volume $V = \pi r_1^2 L + \pi r_2^2 L = L\pi(r_1^2 + r_2^2)$
  - How about just $\min\ (r_1^2 + r_2^2)$
- Constraints: 1 Mpa = 1 Newton/mm^2, r_1, r_2 in mm
  - $\frac{8\text{PL}}{\pi r_1^3} < 180\ MPa \Rightarrow \frac{8 * 10{,}000 * 1{,}000}{\pi r_1^3} < 180$
  - $\frac{4\text{PL}}{\pi r_2^3} < 180\ \text{Mpa} \Rightarrow \frac{4 * 10{,}000 * 1{,}000}{\pi r_2^3} < 180$
  - $\frac{\text{PL}^3}{3\pi E}\left(\frac{7}{r_1^4} + \frac{1}{r_2^4}\right) < 25\ mm \Rightarrow \frac{10{,}000 * 1{,}000^3}{3\pi * 200{,}000}\left(\frac{7}{r_1^4} + \frac{1}{r_2^4}\right) < 25$
  - Bounds: r_1, $r_2 > 0$

Constraints and bounds can also be handled via penalty functions

$$f(x) = 3 \cdot e^{-\left|\frac{x}{10}\right|} \cdot \sin(x)$$



Constraints and Penalty Functions - for Unconstrained MINIMIZERs

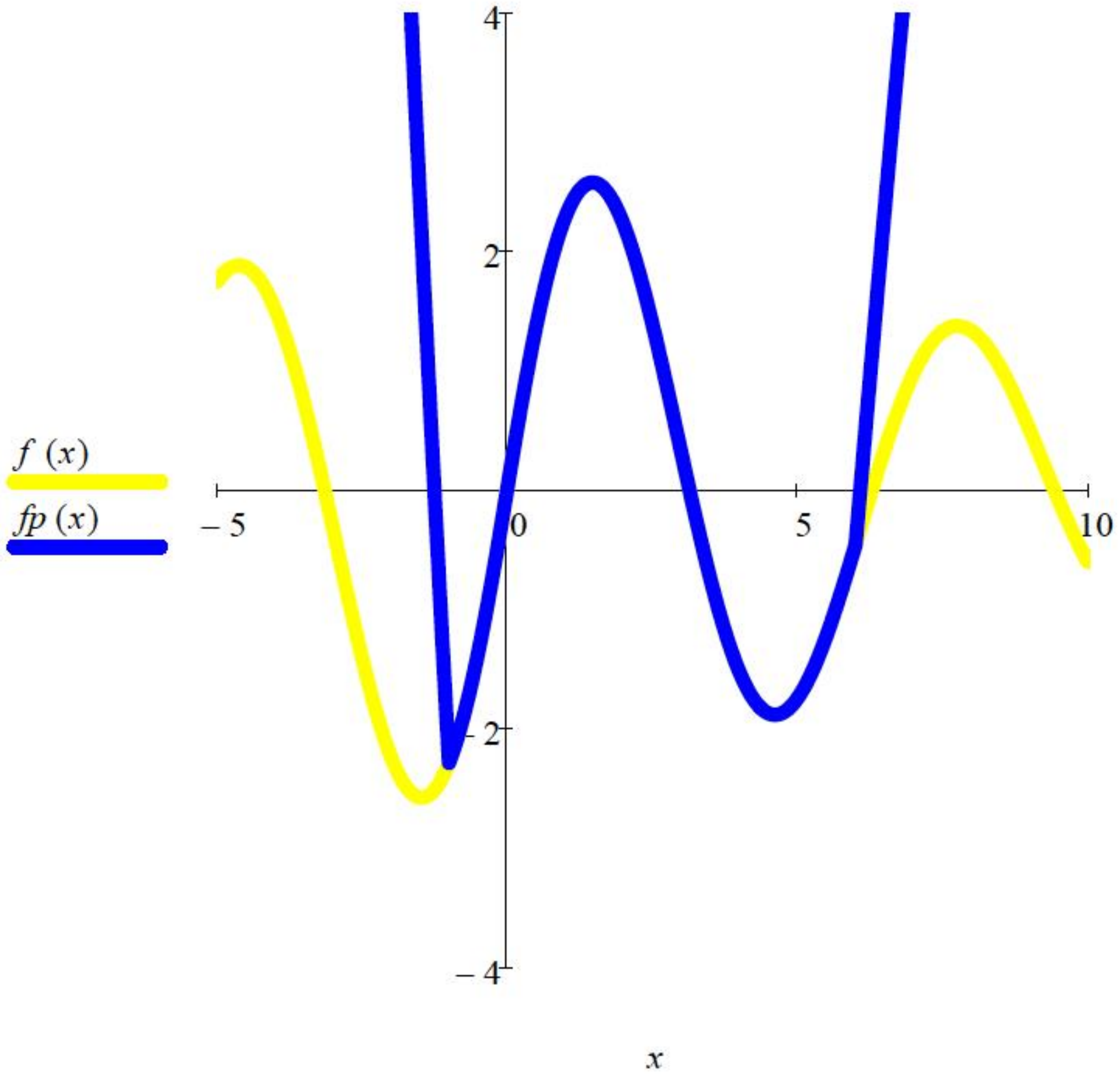
if x < -1

$$p = 10\cdot(-1 - x)$$

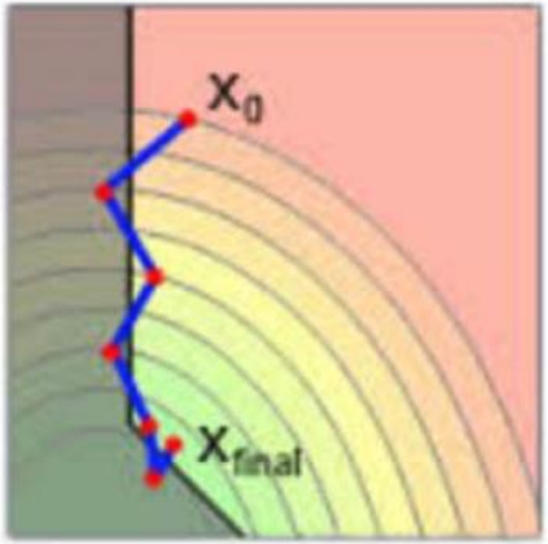
if x > 6

$$p = 4\cdot(x - 6)$$

$$f(x) = 3\cdot e^{-\left|\frac{x}{10}\right|}\cdot \sin(x)$$



Usually slopes around 10^6



Penalty method