# Week 9: We can optimize that!

## Amath 301

**TA Session**

# Today

0. Cookbook optimization
1. Optimization suites in Matlab and Python
2. Techniques for ugly functions

UW is very strong in optimization/applied optimization:

Math 407/408/409, Math/Amath 514/515/516/518

EE 445/447/550/556/578, CSE 421/521/535/541

# What's an optimization problem?

All optimization problems look like this:

$$\min_{x \in X} f(x)$$

Sometimes we can get $x^* = \arg\max_{x \in X} f(x)$ if we know $x^*$ is unique.

Two pieces to work out:

1. A domain $X$ of states to search over. Can be discrete, an interval, the real line, or much more complicated. Sometimes called the *Feasible Region*.
2. A function $f$ called the "objective function. It's what we want to optimize.

Issues?
The optimal solution $x^*$ may not exist or may not be unique. Mathematical analysis is needed to **prove** a unqiue optimal solution exists.

## Sometimes the work is done for us already

Common domains: $[0, 1]$, $[-1, 1]$, $\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{Z}$ (harder) but not always.

Some useful facts/results:

1. If $f$ is continuous and $X$ is a closed interval, then at least one optimal solution exists.
2. If $f$ is convex over $X$ and $X$ is a closed interval or $\mathbb{R}$, then a **unique** optimal value exists.

For twice-differentiable functions, it suffices to show that $f''(x) > 0$ for all $x \in X$ to claim convexity. To prove properties using convexity, we often restrict the domain.

## The reality

Many optimization problems can be solved over an entire domain. Sometimes, you need more structure to your solution.

Constraints: Additional equations you require your solution to satisfy. These come from your problem domain (e.g. conservation of mass, energy, etc).

Examples:

- $x_1 + x_2 + x_3 = 1$ Conservation laws can be enforced
- $x_1 + ex_2 - x_3^2 = -\pi$ Do not need to be "nice"
- $x_1 > 0$ Positivity or non-negativity can be enforced
- $0 \leq x \leq 1$ ($x$ lies in $[0, 1]$)

# Why just $\min$? Why not $\max$ too?

$$\min_{x \in X} f(x) \quad \text{vs} \quad \max_{x \in X} f(x)$$

# Optimization and curve fitting?

Fitting a curve is as easy (or as hard) as solving an optimization problem.

- Curve fitting - minimize an error term.
  - you choose (or are told) the form of the error term
  - You choose (or are told) the form of the curve

Example:

$$E(a) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(f(x_j; a) - y_j\right)^2}$$

where $f$ is the model you want to fit.

# Let's write some code.

In code:

```
def sample_model(x, params):
    """ A vectorized model"""
    return params[0] * np.sin(params[1] * x)

def E2error(x,xdata, ydata, model):
    """ Given parameters `x`, some data, and a model, return the l2 error"""
    n = xdata.size  # how many points do you have? - 1d only

    # Using your parameters and a model, compute the predictions
    preds = model(xdata, x)
    error = np.sqrt(np.sum((preds - ydata) ** 2)/n)
    return error

init_guess = [1,1]
opt_params = fmin(lambda x: E2error(x, xdata, ydatam sample_model), init_guess)
```

In code:

```
function preds = simple_model(x, params):
    preds = params(1) * np.sin(params(2)) .* x);
end

function error = E2error(x,xdata, ydata, model):
    % Given parameters `x`, some data, and a model, return the l2 error
    n = length(xdata);  % how many points do you have? - 1d only

    # Using your parameters and a model, compute the predictions
    preds = model(xdata, x);  % call the model
    error = sqrt(sum((preds - ydata) .^ 2)./n);  % error calculation
end
```

```
init_guess = [1,1];
opt_params = fminsearch(@(x) (@E2error(x, xdata, ydata, @simple_model)), init_guess)
```

# Optimization tools

Use `fmin` or `fminsearch` for HW7.

"Traditional" optimization solvers require 2,3, or 4 pieces of information usually:

1. Initial guess
2. Function $f$ to optimize
3. Region to search over
4. Derivative information (Jacobian or Hessian matrices) - matrices of derivatives for multi-valued functions

Always need 1. and 2., sometimes 3., and sometimes 4.
Sometimes the initial guess is easy - ODE/PDE problems, or 0. Others (ML, data sci), not so easy!

## Python

The more complicated your method or problem, the more code you write.

- `Scipy.optimize` module (Start here for your problems). Modern API for new code
    - (local) `minimize` function offers 14 different algorithms
    - (local) `minimize_scalar` offers 3 methods for single variable problems.
    - (global) `shgo`, `basinhopping`, `brute` (force!), `dual_annealing`

## Matlab

- (local) Optimization toolbox:
    - `fminsearch` Unconstrained problems (5 algorithm choices)
    - `fmincon` Constrained problems (5 algorithm choices)
- Global optimization toolbox:
    - `particleswarm` Video
    - `simulannealbnd` Simulated annealing Video, Demo

## What to do if these techniques don't work?

Two options:

1. Deep learning. Add a few more million parameters and you can fit just about any data.
2. Monte Carlo Methods. A broad class of stochastic boardering on statistical methods that leverage the randomness to find meaningful patterns. The computational sibling to computational Bayesian statistics.

- Major application: Generate samples from "interesting" distributions (data-driven).

To speed these up, get a larger (super) computer.

# Deep learning and Machine learning

## Python

- `sklearn` (`conda install scikit-learn -y`) A more machine learning focused library
- `TensorFlow` and `PyTorch` - Deep learning libraries. Now feature numpy-like APIs
  - Interfaces to C++ and CUDA code that handle the actual computation

## Matlab

- Statistics and Machine Learning Toolbox
- Deep learning toolbox

## How does deep learning work?

Gradients. The backprop algorithm is just fancy chain rule.

These calculations are handled for you by Tensorflow/PyTorch/Matlab.