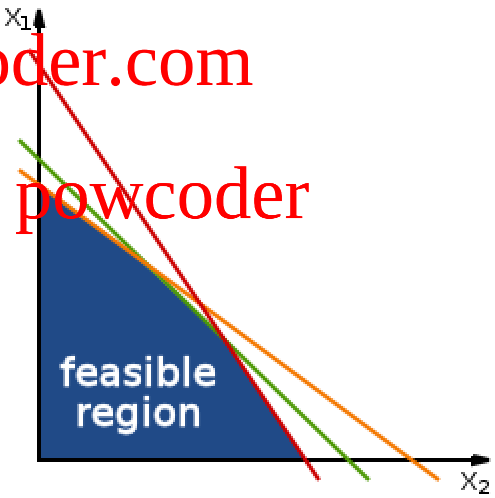
Introduction to algorithms and programming



ME 564/SYS 564
Wed Sep 12, 2018
Steven Hoffenson

Goal of Week 3: To learn some basic derivative-free algorithms and get experience using Excel Solver and MATLAB for optimization

Teams

Team I	Team 1	Team A	Gold Team
Siyi	Jack	Remy	Marta
Junlin	Nick O.	Nick D.	Alkim
Siyuan	Joe	Yu	Candace
			Raif

	Team I	Team 1	Team A	Gold Team
Common interests	Human-powered washing machine	Racing shell RC airplane? 3D printer?	Front-line logistics RC airplane? Rifle cartridge? 3D printer?	Management system Car (autonomous)? Power system?

Take some time to do re-introductions, discuss interests and competencies in these topics, discuss team norms/meetings.

Then, work on your decomposition and modeling strategy. What are the objectives, constraints, and variables for the system and subsystems, and how will you evaluate the impact of inputs on outputs? This is not something that we will explicitly cover in this class.

Recap: Optimization formulation

Objective function

Parameters

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & f(\mathbf{x}, \mathbf{p}) \\ \text{subject to} & \mathbf{g}(\mathbf{x}, \mathbf{p}) \leq 0 \\ & \mathbf{h}(\mathbf{x}, \mathbf{p}) = 0 \end{array}$$

Variables

Constraints

A ***solution*** should tell us the optimal value of the objective, f^*, as well as the optimizers—the values x^* that achieve f^*

"negative null" form

Take a few minutes to discuss in your teams how you will model your *f*'s, *g*'s, and *h*'s

Recap: Explore the problem space

- Does a solution exist? (feasibility)
- Is the problem well-bounded?
- Are the constraints active?
- Are the functions monotonic?
- Can the formulation be simplified?

Answering these questions can help detect formulation errors and save time (and sometimes solve the problem!)

Recap: How to optimize

(Weeks 1-2, 4, 9-12)

1. **Formulate** the problem
   a) Define system boundaries
   b) Develop analytical models
   c) Explore/reduce the problem space
   d) Formalize optimization problem

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}, \mathbf{p}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}, \mathbf{p}) \leq 0 \\ & \mathbf{h}(\mathbf{x}, \mathbf{p}) = 0 \end{aligned}$$

TODAY

(Weeks 3, 5-8, 12)

2. **Solve** the problem
   a) Choose the right approach/algorithm
   b) Solve (by hand, code, or software)
   c) Interpret the results
   d) Iterate if needed

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{H}(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_0)$$

Today’s topics

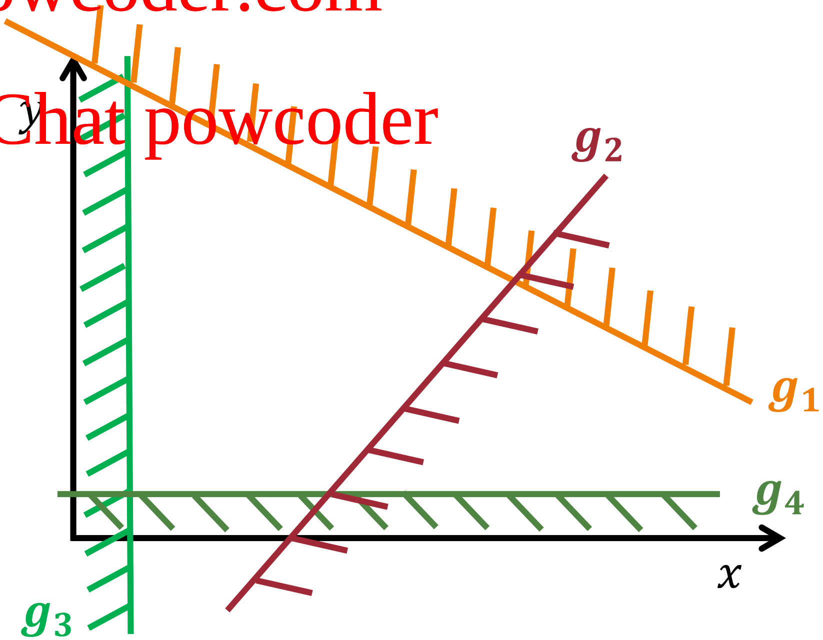
- Linear programming
  - Properties
  - Three ways to solve
  - Practice using Excel
- Derivative free algorithms
  - Coordinate/pattern search
  - Space-filling search
  - Downhill random search
- MATLAB programming exercise

Linear programming

If f, g, and h are **all linear functions**, we have a linear programming problem

$$
\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x},\mathbf{p}) \\
\text{subject to} \quad & \mathbf{g}(\mathbf{x},\mathbf{p}) \leq 0 \\
& \mathbf{h}(\mathbf{x},\mathbf{p}) = 0
\end{aligned}
$$

$$
\begin{aligned}
\min_{\{x,y\}\in R^2} \quad & f = -2x - y \\
\text{s.t.} \quad & g_1 : x + 2y - 8 \leq 0 \\
& g_2 : x - y - 1.5 \leq 0 \\
& g_3 : -2x + 1 \leq 0 \\
& g_4 : -2y + 1 \leq 0
\end{aligned}
$$



Method 1: Graphing

$$\min_{\{x,y\}\in R^2} \quad f = -2x - y$$

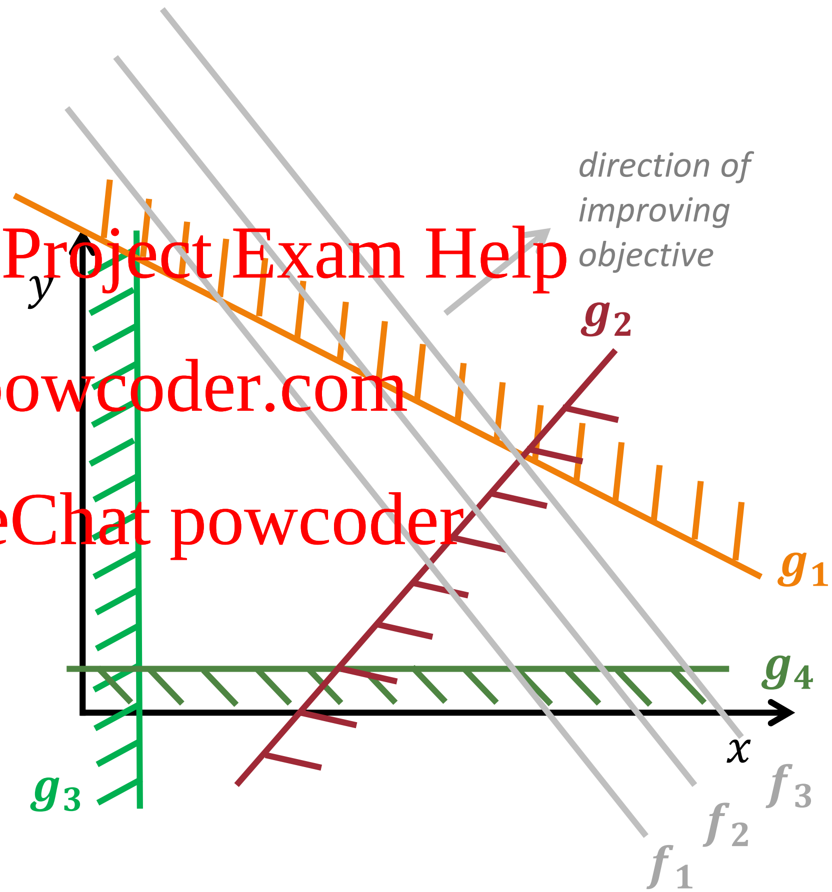
s.t.

$$g_1 : x + 2y - 8 \leq 0$$

$$g_2 : x - y - 1.5 \leq 0$$

$$g_3 : -2x + 1 \leq 0$$

$$g_4 : -2y + 1 \leq 0$$

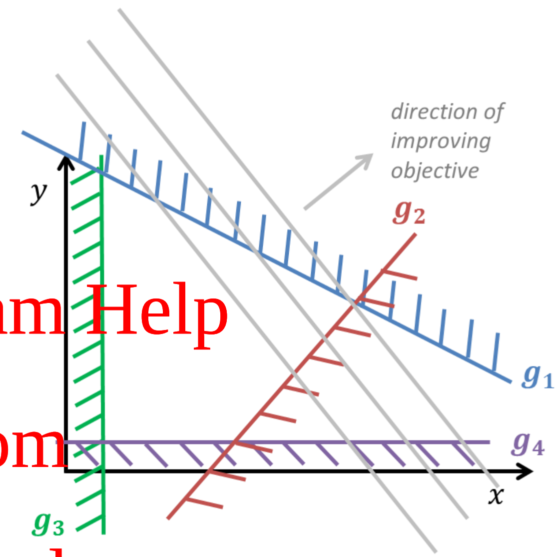


We can see that the optimizer is the intersection of g_1 and g_2, so we just need to solve 2 equations of 2 unknowns

This is challenging for 3 variables and impossible for 4+

Method 2: Monotonicity Analysis

$$\min_{x,y}\ f(x^-, y^-) = -2x - y$$
$$\text{s.t.}\quad g_1(x^+, y^+) = x + 2y - 8 \le 0$$
$$g_2(x^+, y^-) = x - y - 1.5 \le 0$$
$$g_3(x^-) = -2x + 1 \le 0$$
$$g_4(y^-) = -2y + 1 \le 0$$



From MP1, we know g_1 must be active!

$$y = 4 - 0.5x$$

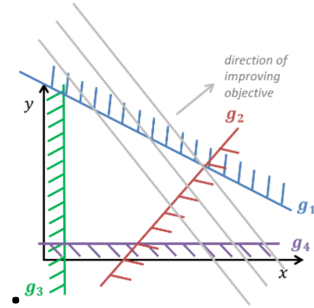
$$\min_{x,y}\ f(x^-) = -1.5x - 4$$
$$\text{s.t.}\quad g_2(x^+) = 1.5x - 5.5 \le 0$$
$$g_3(x^-) = -2x + 1 \le 0$$
$$g_4(x^+) = x - 7 \le 0$$

Which dominates?
$x \le 3.67$ or $x \le 7$

Because $x \le 3.67$ is stronger, g_2 is the active one

So, $x^* = 11/3$
$y^* = 13/6$
$f^* = -19/2$

Method 3: LP algorithm



Note: The Simplex Algorithm is a standard method in linear algebra to do this using slack variables and pivot tables; this is an equivalent matrix algorithm:

1. Start with a point at the vertex of the feasible design space (an **x** that lies at the intersection of *n* constraints, where *n* is the number of variables); these active constraints form the *active set*
2. Choose a direction to move for improvement, and move to an adjacent, better extreme point; this results in swapping one constraint in the active set
3. Repeat (2) until no more improvement can be found

Linear program properties

c is a vector of parameters in the objective function

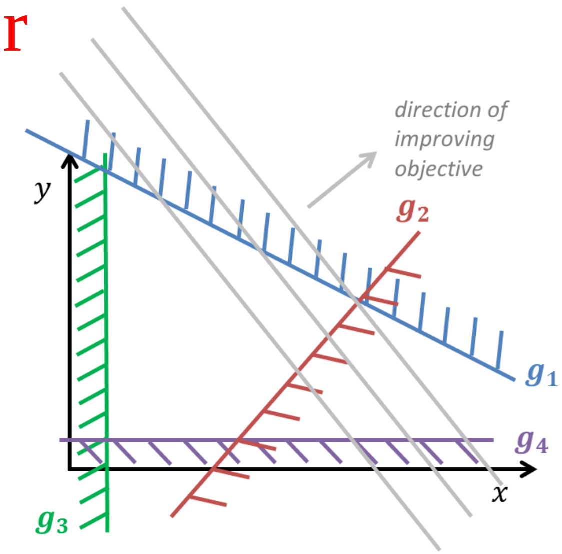
$$\min_{\boldsymbol{x}} \quad \boldsymbol{c}^T \boldsymbol{x}$$

$$\text{s.t.} \quad \boldsymbol{h} = \boldsymbol{A}_1 \boldsymbol{x} - \boldsymbol{b}_1 = 0$$

$$\boldsymbol{g} = \boldsymbol{A}_2 \boldsymbol{x} - \boldsymbol{b}_2 \leq 0$$

$\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{b}_1, \boldsymbol{b}_2$ are matrices & vectors of parameters in the constraints

1. All objectives and constraints are monotonic
2. Solutions will always be at vertices of the feasible space, or along an entire edge/face/hyperplane section



Exercise: Excel Solver

Together, we will do the example from before:

$$\min_{\{x,y\}\in R^2} \quad f = -2x - y$$

s.t.

$$g_1 : x + 2y - 8 \leq 0$$

$$g_2 : x - y - 1.5 \leq 0$$

$$g_3 : -2x + 1 \leq 0$$

$$g_4 : -2y + 1 \leq 0$$

Then, try it out on your own on the below:

$$\max_{\mathbf{x}} \quad f(\mathbf{x}) = x_1 - x_2$$

s.t.

$$g_1(\mathbf{x}) = 2x_1 + 3x_2 - 10 \leq 0$$

$$g_2(\mathbf{x}) = -x_1 - x_2 + 1 \leq 0$$

$$g_3(\mathbf{x}) = -2x_1 + 7x_2 - 8 \leq 0$$

Derivative-free search algorithms

"Smart" ways to search the design space based on heuristics or intuitive rules

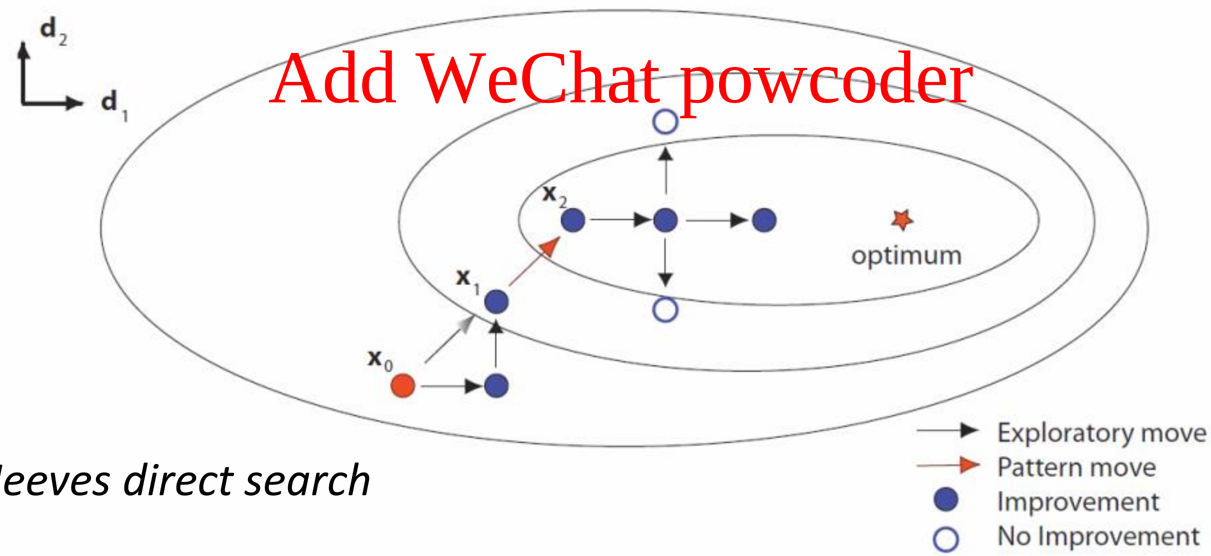
Derivative-free search

- Pattern search (Hooke-Jeeves, MADS, GPS)
- Space-filling search (DIRECT)
- Downhill random search (Simulated Annealing)

Note: Section 7.2 also includes Genetic Algorithm and Particle Swarm Optimization – we will cover those in a few weeks

Coordinate search

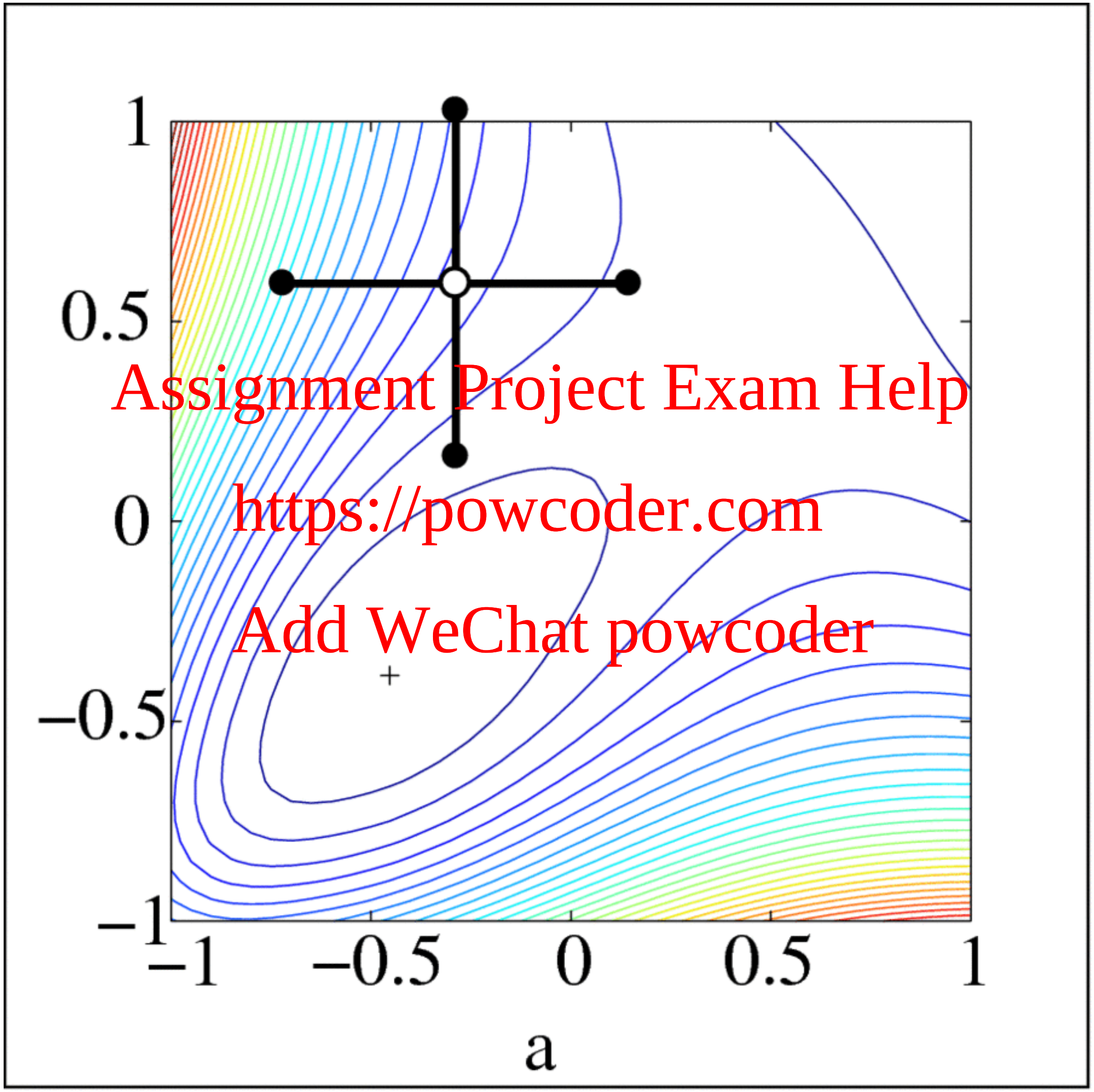
1. Pick a start point x_0
2. Test points some distance d away from the current point in many directions within the variable space
3. When a better point is found, move there and repeat (2)
4. If no better point is found, decrease d and go back to (2)
5. When d is small enough, quit



Hooke-Jeeves direct search

Kramer, O., Echeverría Ciaurri, D., and Koziel, S. (2011) "Derivative-Free Optimization", in S. Koziel and X.S. Yang (Eds.) *Computational Optimization, Methods and Algorithms*, Series: Studies in Computational Intelligence, Springer, pp. 61-83, 2011.

Coordinate search in action



Generalized Pattern Search (GPS)

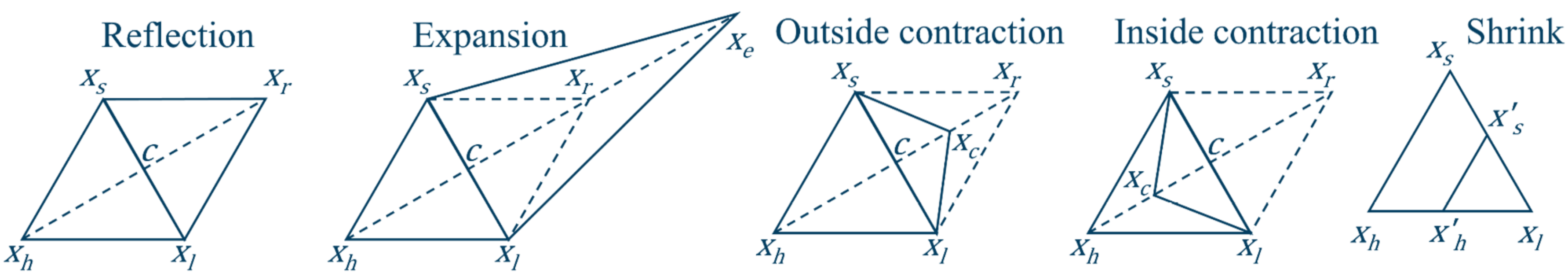
This is an extension of Hooke-Jeeves to include a global search step

1. Pick a starting point and mesh/step size
2. Global search: Create mesh over design space and evaluate the objective at some points in mesh
3. Local poll: Search in some pre-defined set of directions from current best point
4. If no improvements found, reduce mesh/step size and repeat
5. Continue until mesh spacing small enough or max iterations met

Nelder-Meade algorithm

Applies simplex concepts to non-LP problems:

1. Define a *simplex* (an n+1-sided, n-dimensional “polygon”), and evaluate the objective at each vertex
2. Identify the best side, and transform the simplex around that side: Reflect, expand, or contract, depending on how good the reflection point is compared to the others
3. Stop when the vertices are close enough (in the design or objective space) or when you’ve done too many iterations



when $f_l \le f_r < f_s$	when $f_r < f_l$ and $f_e < f_r$	when $f_s \le f_r < f_h$ and $f_c \le f_r$	when $f_h \le f_r$ and $f_c < f_h$	otherwise

Pattern/direct search – Pros/cons

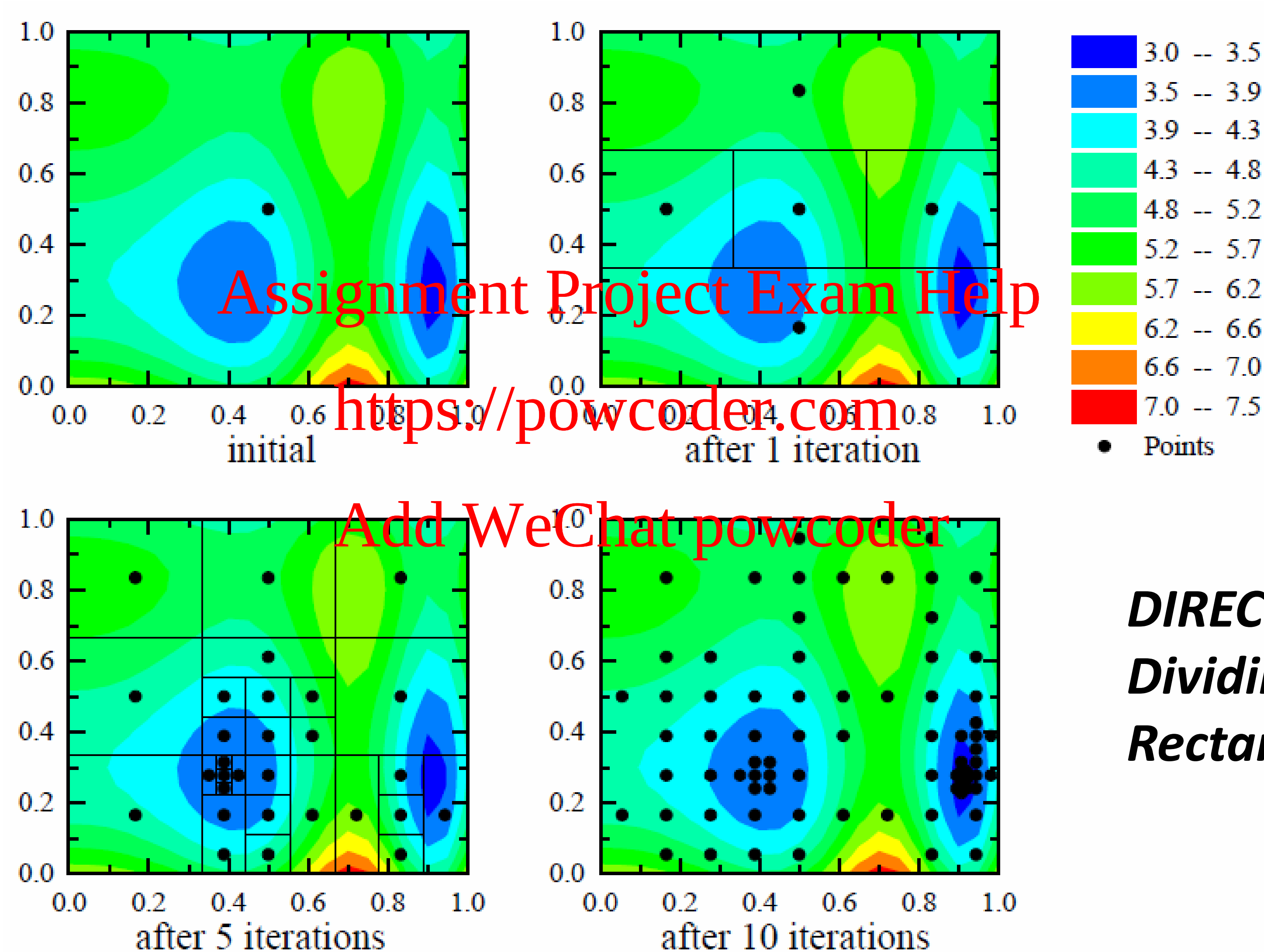
Advantages

- Conceptually straightforward
- Deterministic (repeatable)
- Tend to converge on a local optimum
- No need for derivatives or other problem information

Disadvantages

- Must identify a feasible starting point or points
- Dimensionality: Cost increases with many variables
- Slow local convergence
- No global exploration
- Not particularly efficient, since you may be evaluating functions in obviously “bad” directions

Space-filling: DIRECT

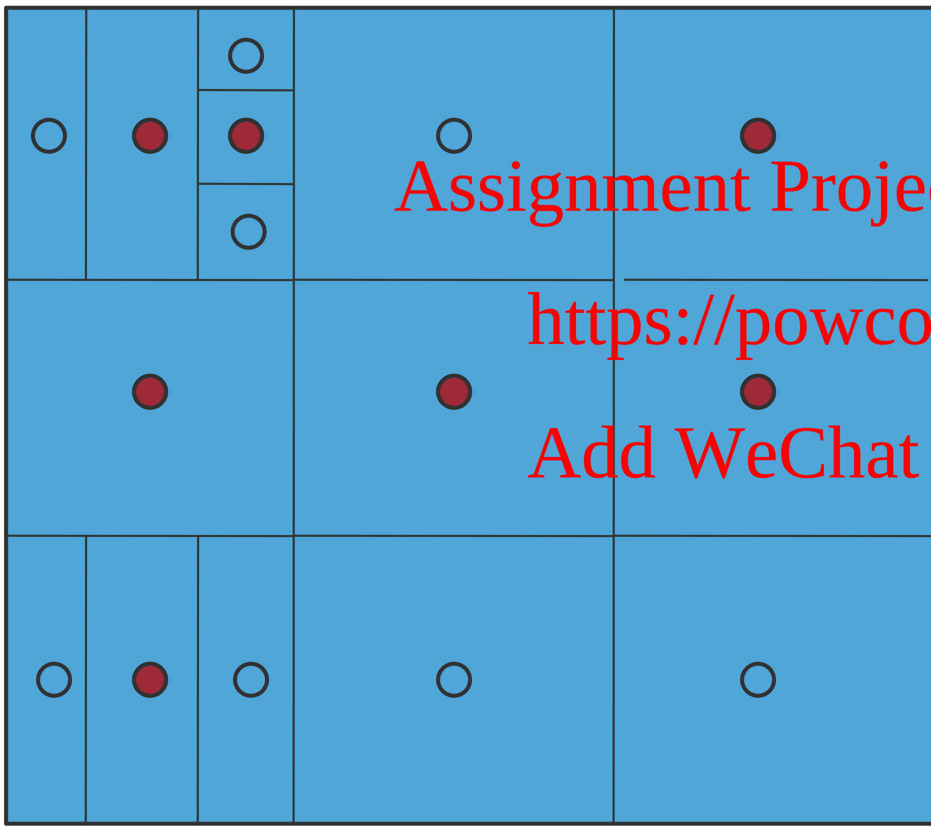


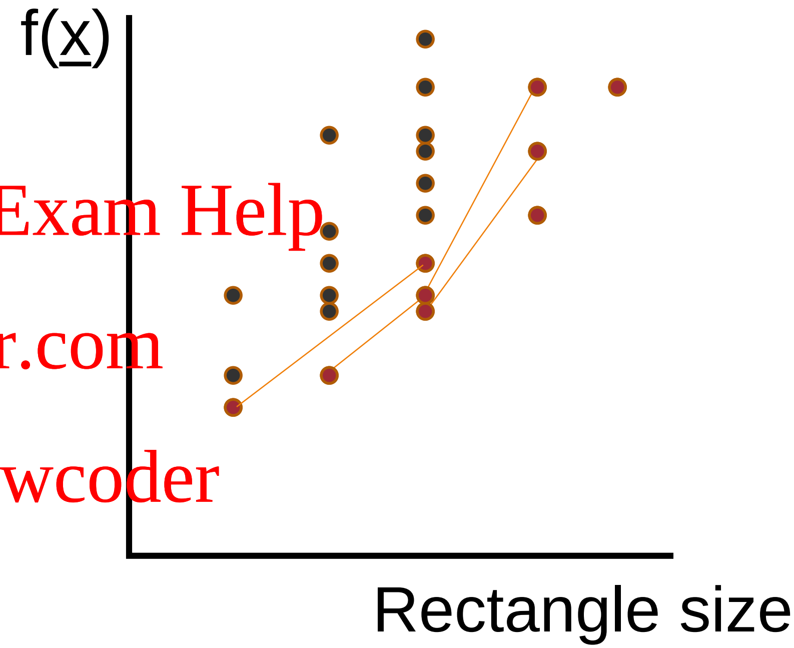
DIRECT = Dividing Rectangles

http://hosting.cs.vt.edu/hpcs2008/HPCS08DIRECTtut.pdf

DIRECT with Two Variables

Iteration **4**





Each iteration selects rectangles to further divide based on:

1. Function value (low)
2. Rectangle size (large)
3. Constraint violation (low)

figures from John Whitefoot, 2010 lecture

DIRECT – Pros/Cons

Advantages

- Systematic search balances local and global search
- Can be re-started where you left off
- Deterministic (repeatable)
- No parameters to tune
- Can handle integers and equality constraints
- Very robust

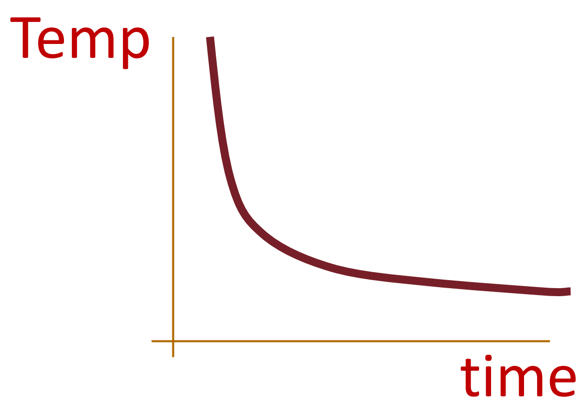
Disadvantages

- Dimensionality: Struggles with many (e.g., >10) variables
- Slow local convergence

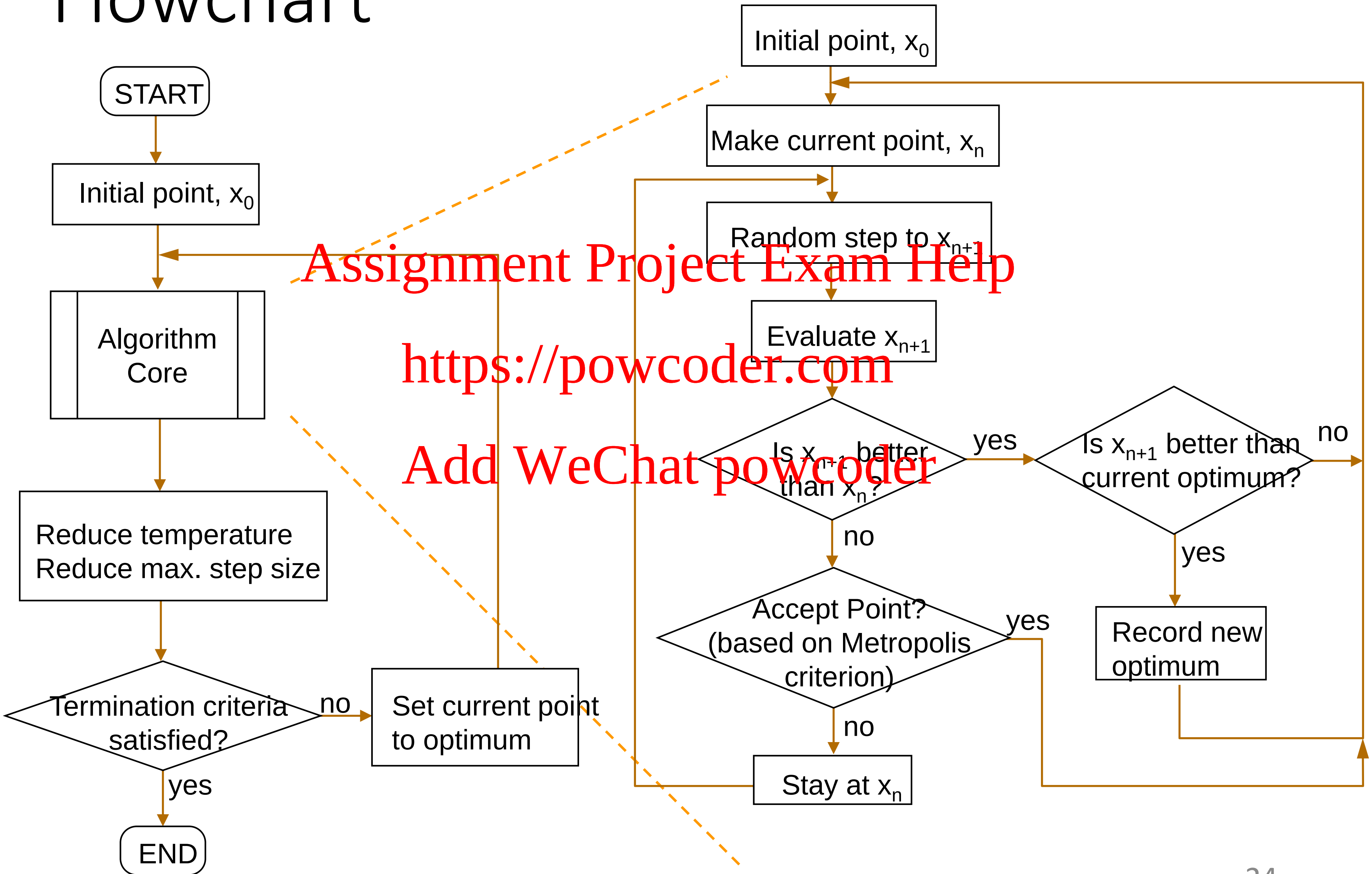
Simulated Annealing (SA) overview

- Based on cooling metals: Seeking lowest energy state
- Performs random search with some probability of accepting worse point (to search globally)
- Probability of accepting worse point based on Metropolis criterion, which decreases over time

$$p(\Delta f) = \exp\left(\frac{-\Delta f}{T}\right)$$



Simulated Annealing Algorithm Flowchart



Simulated Annealing – Pros/Cons

Advantages

- Doesn’t need to systematically cover the space – can be more efficient for high-dimension problems

Disadvantages

- Highly dependent on starting point

- Doesn’t always cover the design space (local)
- Random search not very “smart”

  - Can repeat areas already searched
  - Can require many function evaluations
- Many parameters to tune that influence result
  - Penalty function weights
  - Temperature cooling schedule

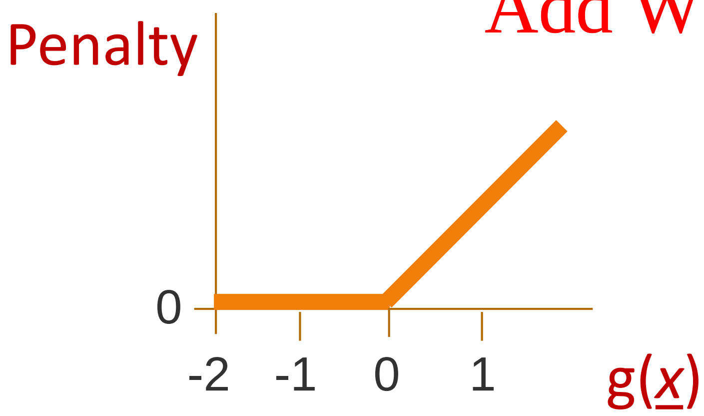
Constraint handling

Penalty functions are an easy way to add constraints

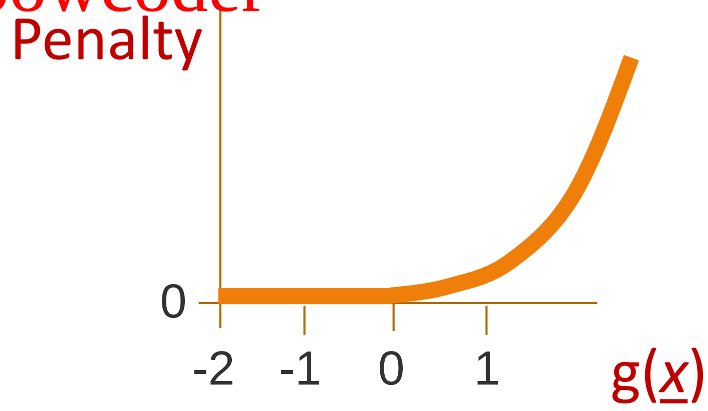
Penalty Functions for Constraints

When algorithm doesn't specifically handle constraints, can add them to the objective via a penalty function

$$\min\ f_P(\bar{x}, Penalty) = f(\bar{x}) + \sum_{i=1}^{m} w_i \cdot \left(\max(0, g_i(\bar{x}))\right)^2$$



Linear



Quadratic

Summary

- **Linear programs** are special cases
  - All functions **monotonic**
  - Solutions must lie on **boundary** of design space
  - **Simplex algorithm** is efficient
- **Derivative-free algorithms** for nonlinear problems are **straightforward** and **robust**, but may **take longer** and converge on **local optima**
  - Coordinate search
  - Nelder-Meade
  - Space-filling DIRECT
  - Simulated Annealing

Acknowledgements

- Much of this material came from Chapters 5.8, 7.1, and 7.3 of the textbook, *Principles of Optimal Design*
- Some of the slides and examples came from Dr. John Whitefoot while he was at the University of Michigan

MATLAB Programming: Simulated Annealing code

- Based on cooling metals: Seeking lowest energy state
- Performs random search with some probability of accepting worse point (to search globally)
- Probability of accepting worse point based on Metropolis criterion, which decreases over time

We will write a code that does this in class

$$p(\Delta f) = \exp\left(\frac{-\Delta f}{T}\right)$$