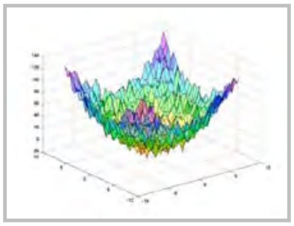
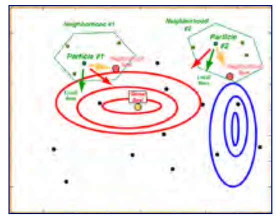


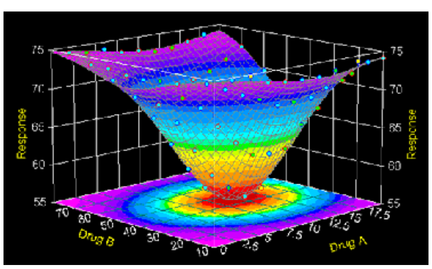
Numerical Optimization

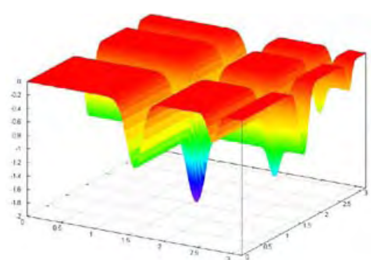
Robert Stengel
Robotics and Intelligent Systems MAE 345, Princeton University, 2015

- Gradient search
- Gradient-free search
  - Grid-based search
  - Random search
  - Downhill simplex method
- Monte Carlo evaluation
- Simulated annealing
- Genetic algorithms
- Particle swarm optimization

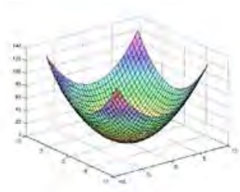


Numerical Optimization





- Previous examples with simple cost function, *J*, could be evaluated analytically
- What if *J* is too complicated to find an analytical solution for the minimum?
- … or *J* has multiple minima?
- Use numerical optimization to find *local* and/or *global* solutions



Two Approaches to Numerical Minimization

1) Slope and curvature of surface

- a) Evaluate gradient , $\partial J/\partial u$, and search for zero
- b) Evaluate Hessian, $\partial^2 J/\partial u^2$, and search for positive value

$$\left(\frac{\partial J}{\partial \mathbf{u}}\right)_o = \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_0} = \textit{starting guess}$$

$$\left(\frac{\partial J}{\partial \mathbf{u}}\right)_n = \left(\frac{\partial J}{\partial \mathbf{u}}\right)_{n-1} + \Delta\left(\frac{\partial J}{\partial \mathbf{u}}\right)_n = \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_n} \textit{ such that } \left|\frac{\partial J}{\partial \mathbf{u}}\right|_n < \left|\frac{\partial J}{\partial \mathbf{u}}\right|_{n-1}$$

… until gradient is close enough to zero

2) Evaluate cost, J , and search for smallest value

$$J_o = J(\mathbf{u}_o) = \textit{starting guess}$$

$$J_1 = J_o + \Delta J_1(\mathbf{u}_o + \Delta\mathbf{u}_1) \textit{ such that } \quad J_1 < J_o$$

$$J_2 = J_1 + \Delta J_2(\mathbf{u}_1 + \Delta\mathbf{u}_2) \textit{ such that } \quad J_2 < J_1$$

Stop when difference between J_n and J_{n-1} is negligible

Gradient/Hessian Search to Minimize a Quadratic Function

Cost function, gradient, and Hessian matrix

$$J = \frac{1}{2}(\mathbf{u}-\mathbf{u}^*)^T \mathbf{R}(\mathbf{u}-\mathbf{u}^*), \quad \mathbf{R} > \mathbf{0}$$

$$= \frac{1}{2}\left(\mathbf{u}^T\mathbf{R}\mathbf{u} - \mathbf{u}^T\mathbf{R}\mathbf{u}^* - \mathbf{u}^{*T}\mathbf{R}\mathbf{u} + \mathbf{u}^{*T}\mathbf{R}\mathbf{u}^*\right)$$

$$\frac{\partial J}{\partial \mathbf{u}} = (\mathbf{u}-\mathbf{u}^*)^T\mathbf{R} = \mathbf{0} \textit{ when } \mathbf{u} = \mathbf{u}^*$$

$$\frac{\partial^2 J}{\partial \mathbf{u}^2} = \mathbf{R} = \textit{symmetric constant} > \mathbf{0}$$

Guess a starting value of u, u_o

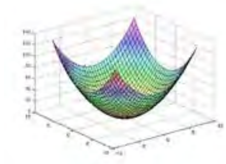
$$\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o} = (\mathbf{u}_o-\mathbf{u}^*)^T\mathbf{R} = (\mathbf{u}_o-\mathbf{u}^*)^T \left.\frac{\partial^2 J}{\partial \mathbf{u}^2}\right|_{\mathbf{u}=\mathbf{u}_o}$$

$$(\mathbf{u}_o-\mathbf{u}^*)^T = \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o}\mathbf{R}^{-1} \quad (\textit{row})$$

Solve for u*

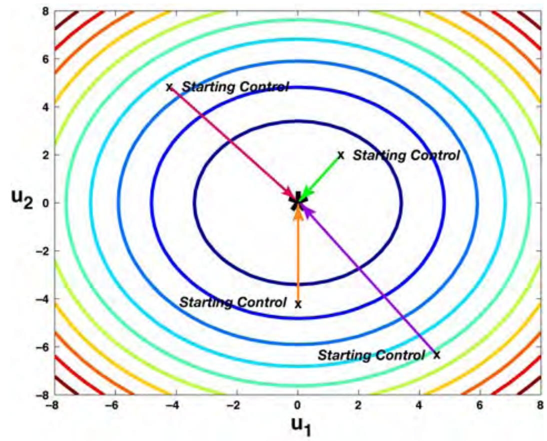
$$\mathbf{u}^* = \mathbf{u}_o - \mathbf{R}^{-1}\left[\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o}\right]^T \quad (\textit{column})$$

Optimal Value of Quadratic Function Found in a One Step



$$\mathbf{u}^* = \mathbf{u}_o - \mathbf{R}^{-1}\left[\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o}\right]^T$$

$$= \mathbf{u}_o - \left[\left.\frac{\partial^2 J}{\partial \mathbf{u}^2}\right|_{\mathbf{u}=\mathbf{u}_o}\right]^{-1}\left[\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o}\right]^T$$



- **Gradient** establishes general search direction
- **Hessian** fine-tunes direction and tells exactly how far to go

Numerical Example

- Cost function and derivatives

$$J = \frac{1}{2}\left\{\left[\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix}\right]^T \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}\left[\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix}\right]\right\}$$

$$\left(\frac{\partial J}{\partial \mathbf{u}}\right)^T = \left[\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix}\right]^T \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}; \quad \mathbf{R} = \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}$$

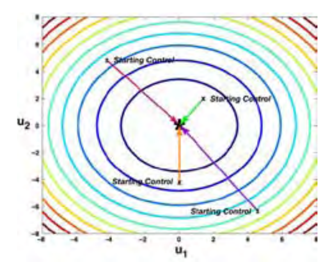
- First guess at optimal control

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}_0 = \begin{pmatrix} 4 \\ 7 \end{pmatrix}$$

- Derivatives at starting point

$$\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_0} = \left[\begin{pmatrix} 4 \\ 7 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix}\right]^T \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix} = \begin{pmatrix} 11 \\ 42 \end{pmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}$$



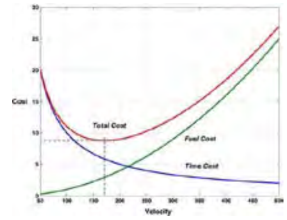
- Solution from starting point

$$\mathbf{u}^* = \mathbf{u}_o - \mathbf{R}^{-1}\left[\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_o}\right]^T$$

$$\mathbf{u}^* = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}^* = \begin{pmatrix} 4 \\ 7 \end{pmatrix} - \begin{bmatrix} 9/5 & -2/5 \\ -2/5 & 1/5 \end{bmatrix}\begin{pmatrix} 11 \\ 42 \end{pmatrix}$$

$$= \begin{pmatrix} 4 \\ 7 \end{pmatrix} - \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Newton-Raphson Iteration



- Many cost functions are not quadratic
- However, the surface is well-approximated by a quadratic in the vicinity of the optimum, u*

$$J(\mathbf{u}^*+\Delta\mathbf{u}) \approx J(\mathbf{u}^*) + \Delta J(\mathbf{u}^*) + \Delta^2 J(\mathbf{u}^*) + \ldots$$

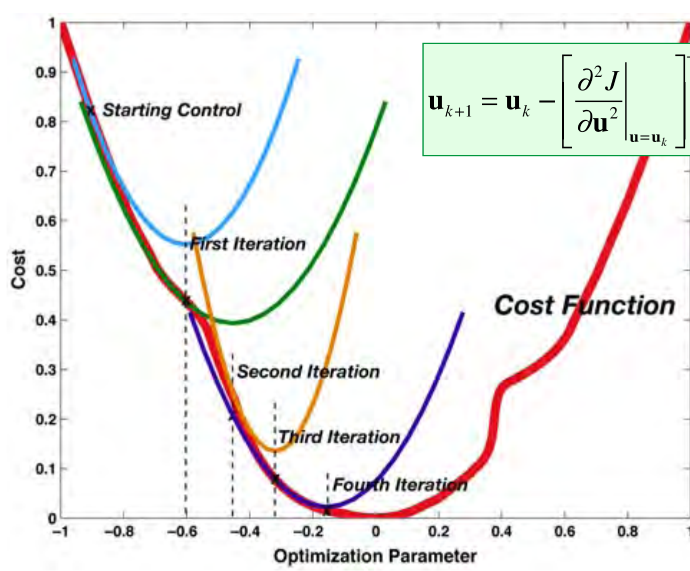
$$\Delta J(\mathbf{u}^*) = \Delta\mathbf{u}^T \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}^*} = 0$$

$$\Delta^2 J(\mathbf{u}^*) = \Delta\mathbf{u}^T \left[\left.\frac{\partial^2 J}{\partial \mathbf{u}^2}\right|_{\mathbf{u}=\mathbf{u}^*}\right] \Delta\mathbf{u} \geq 0$$

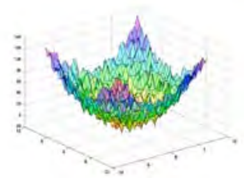
Optimal solution requires multiple steps

Newton-Raphson Iteration

Newton-Raphson algorithm is an iterative search using both the gradient and the Hessian matrix



$$\mathbf{u}_{k+1} = \mathbf{u}_k - \left[\left.\frac{\partial^2 J}{\partial \mathbf{u}^2}\right|_{\mathbf{u}=\mathbf{u}_k}\right]^{-1} \left[\left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_k}\right]^T$$



Difficulties with Newton-Raphson Iteration

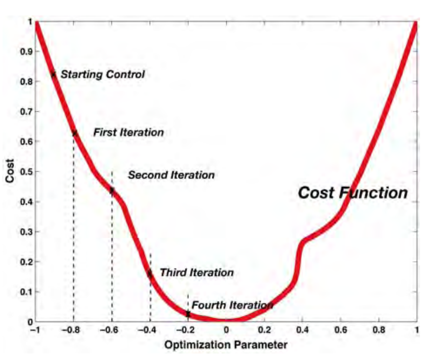
$$\mathbf{u}_{k+1} = \mathbf{u}_k - \left[\boxed{\left. \frac{\partial^2 J}{\partial \mathbf{u}^2} \right|_{\mathbf{u}=\mathbf{u}_k}} \right]^{-1} \left[\left. \frac{\partial J}{\partial \mathbf{u}} \right|_{\mathbf{u}=\mathbf{u}_k} \right]^T$$

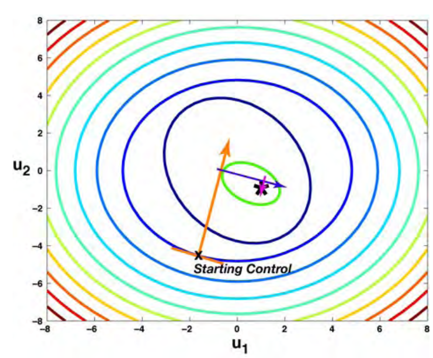
- **Good when close to the optimum, but …**
- **Hessian matrix (i.e., the curvature) may be**
  - **Hard to estimate, e.g., large effects of small errors**
  - **Locally misleading, e.g., wrong curvature**
- **Gradient searches focus on local minima**

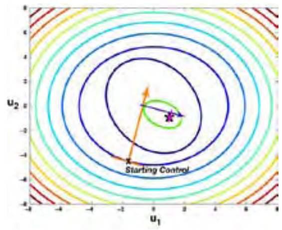
Steepest-Descent Algorithm Multiplies Gradient by a Scalar Constant

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \varepsilon \left[\left. \frac{\partial J}{\partial \mathbf{u}} \right|_{\mathbf{u}=\mathbf{u}_k} \right]^T$$

- **Replace Hessian matrix by a scalar constant**
- **Gradient is orthogonal to equal-cost contours**



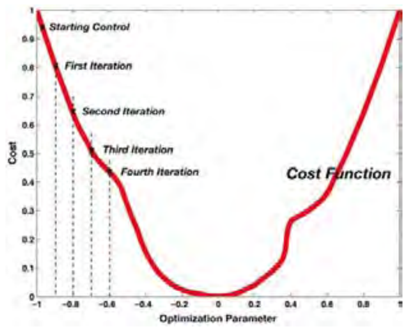




Choice of Steepest-Descent Constant

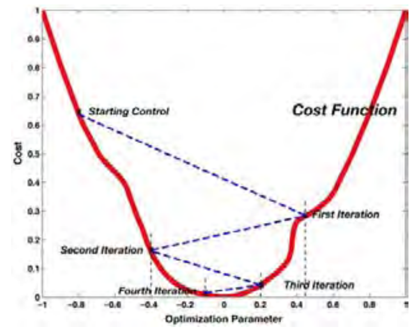
If gain is too small

Convergence is slow



If gain is too large

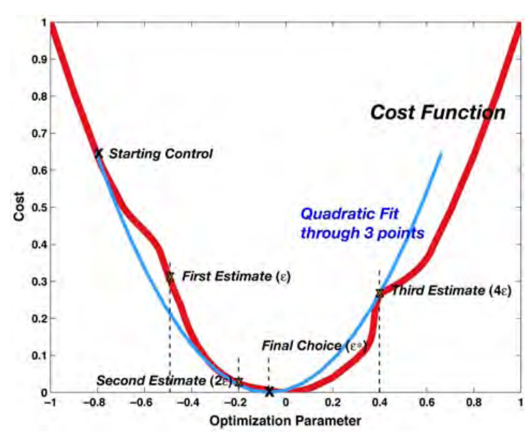
Convergence oscillates or may fail

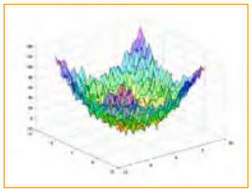


Solution: Make gain adaptive

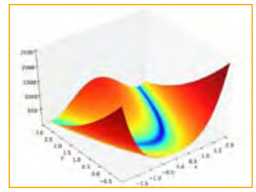
Optimal Steepest-Descent Constant

- **Use optimal gain on each iteration**
- **Find optimal step size by evaluating cost, *J*, for intermediate solutions (with same *dJ/d*u)**
- **Adjustment rule (partial) for**
  - **Starting estimate, J_o**
  - **1st estimate, J_1, using ε**
  - **2nd estimate, J_2, using 2ε**
  - **If $J_2 > J_1$**
    - **Quadratic fit through 3 points to find best ε and use for next iteration**
  - **Else**
    - **3rd estimate using 4ε**
    - **etc.**





Gradient Search Issues



Steepest Descent

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \varepsilon \left[\left. \frac{\partial J}{\partial \mathbf{u}} \right|_{\mathbf{u}=\mathbf{u}_k} \right]^T$$

Newton Raphson

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \left[\left. \frac{\partial^2 J}{\partial \mathbf{u}^2} \right|_{\mathbf{u}=\mathbf{u}_k} \right]^{-1} \left[\left. \frac{\partial J}{\partial \mathbf{u}} \right|_{\mathbf{u}=\mathbf{u}_k} \right]^T$$

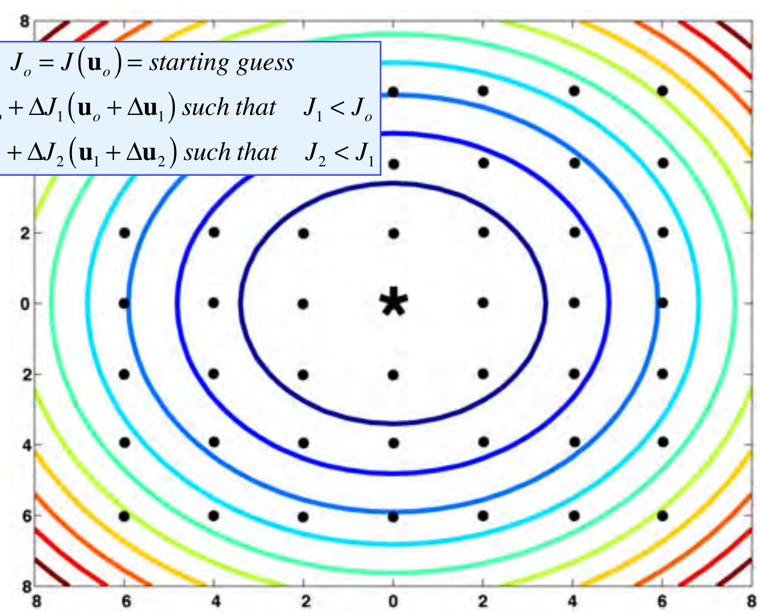
- **Need to evaluate gradient (and possibly Hessian matrix)**
- **Not global: gradient searches focus on local minima**
- **Convergence may be difficult with "noisy" or complex cost functions**

Gradient-Free Search: Grid-Based Search

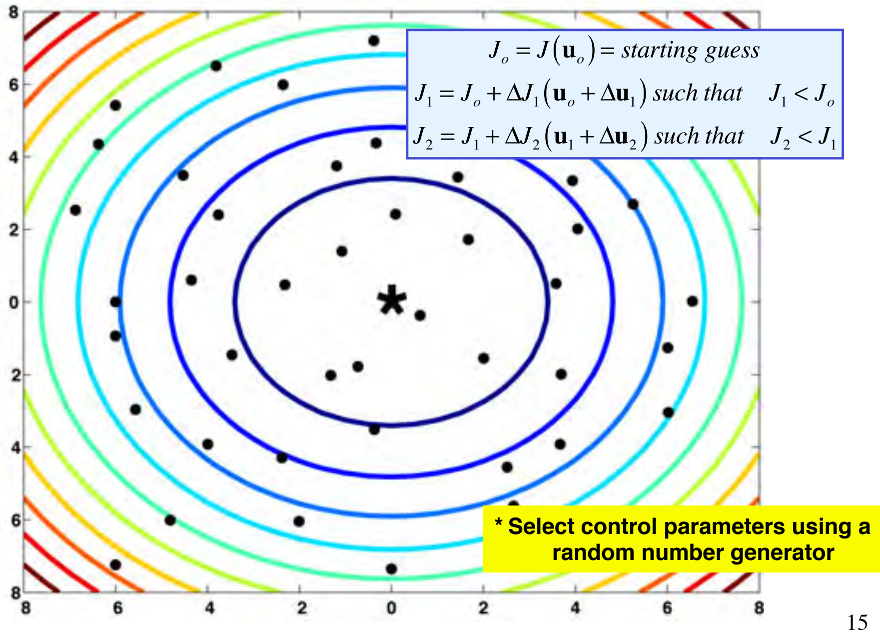
$$J_o = J(\mathbf{u}_o) = \textit{starting guess}$$

$$J_1 = J_o + \Delta J_1(\mathbf{u}_o + \Delta \mathbf{u}_1) \textit{ such that } \quad J_1 < J_o$$

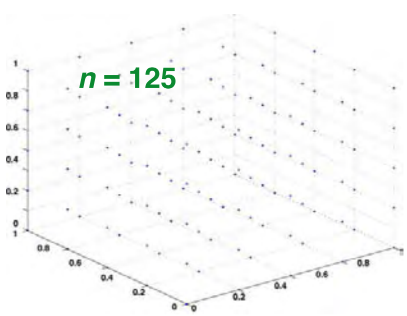
$$J_2 = J_1 + \Delta J_2(\mathbf{u}_1 + \Delta \mathbf{u}_2) \textit{ such that } \quad J_2 < J_1$$

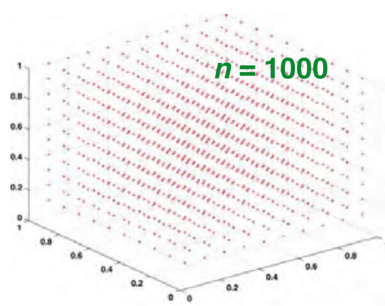


Gradient-Free Search: Random Search



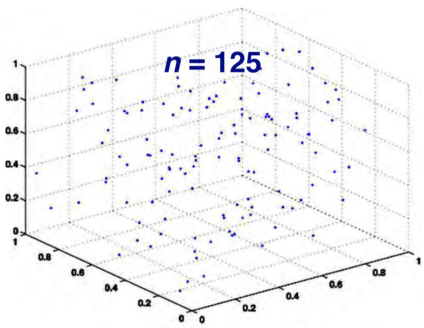
Three-Parameter Grid Search

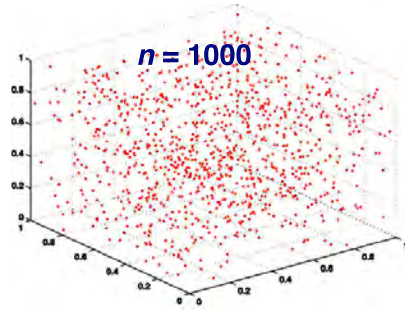




- Regular spacing
- Fixed resolution
- Trials grow as m^n, where
  - n = Number of parameters
  - m = Resolution

Three-Parameter Random Field Search

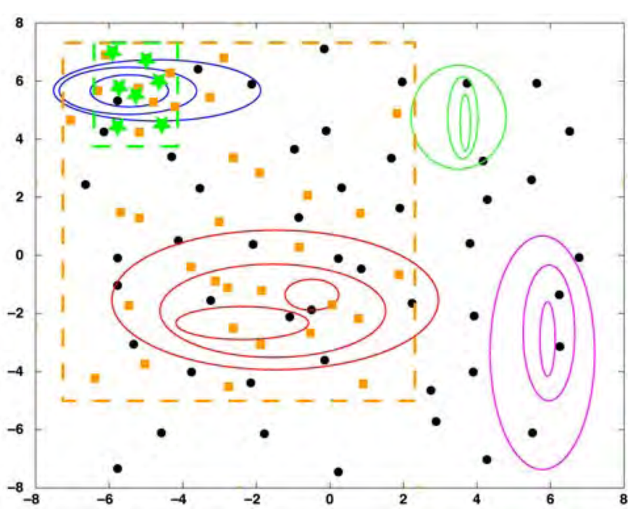




Variable spacing and resolution
Arbitrary number of trials
Random space-filling

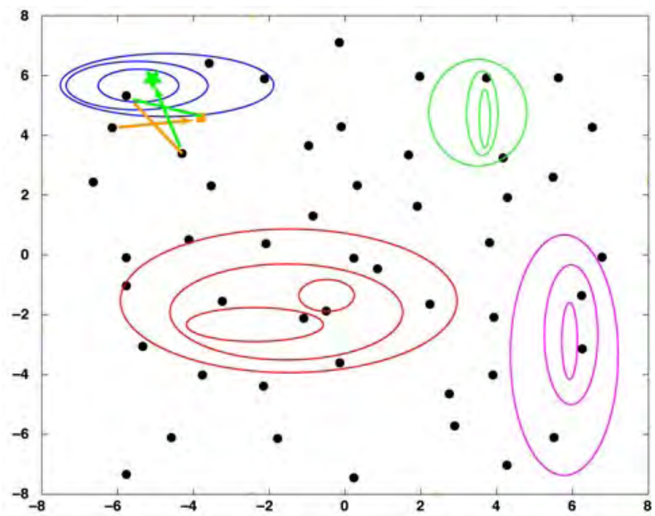
Directed (Structured) Search for Minimum Cost

Continuation of grid-based or random search
Localize areas of low cost
Increase sampling density in those areas



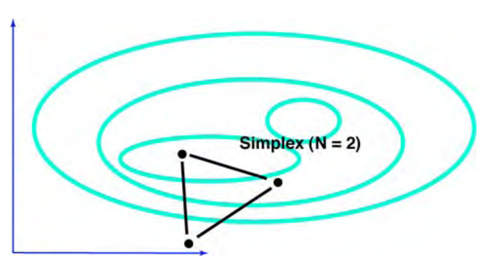
Directed (Structured) Search for Minimum Cost

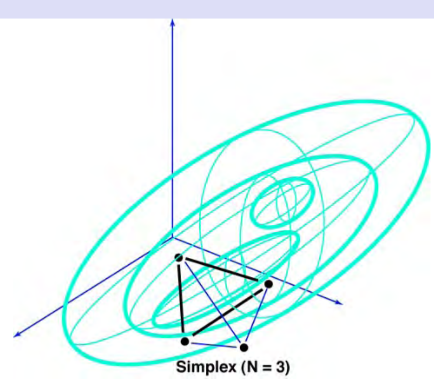
- Interpolate or extrapolate from one or more starting points



Downhill Simplex Search (Nelder-Mead Algorithm)

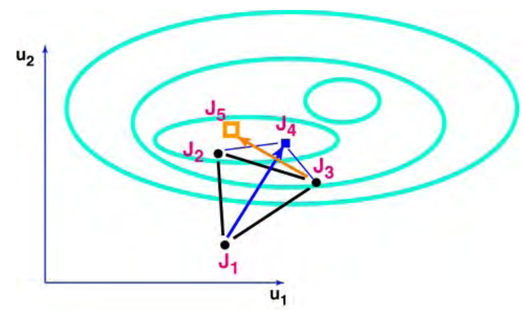
- <u>Simplex</u>: *N*-dimensional figure in control space defined by
  - *N* + 1 vertices
  - (*N* + 1) *N* / 2 straight edges between vertices





Search Procedure for Downhill Simplex Method

- Select starting set of vertices
- Evaluate cost at each vertex
- Determine vertex with largest cost (e.g., J_1 at right)



- Project search from this vertex through middle of opposite face (or edge for $N = 2$)
- Evaluate cost at new vertex (e.g., J_4 at right)
- Drop J_1 vertex, and form simplex with new vertex
- Repeat until cost is small

Humanoid Walker optimized via Nelder-Mead
http://www.youtube.com/watch?v=BcYPLR_j5dg

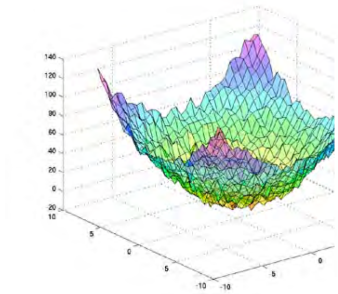
Monte Carlo Evaluation of Systems and Cost Functions

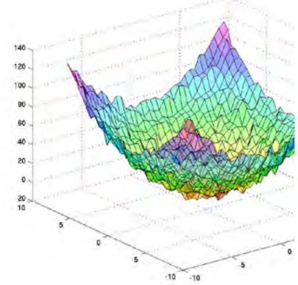
- Multiple evaluations of a function with uncertain parameters using
  - Random number generators, and
  - Assumed or measured statistics of parameters
- Not an exhaustive evaluation of all parameters

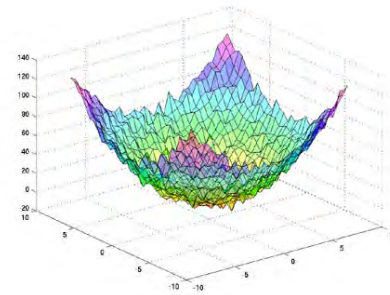
Monte Carlo Evaluation of Systems and Cost Functions

- **Example:** 2-D quadratic function with added Gaussian noise
- Each trial generates a different result $(\sigma_z = 4)$

```
[X,Y] = meshgrid(-8:.5:8);
   Z   =   X.^2 + Y.^2;
   Z1  =   Z + 4*randn(33);
   surf(X,Y,Z1)
   colormap hsv
   alpha(.4)
```



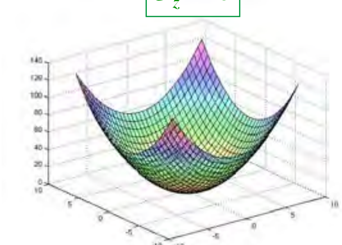




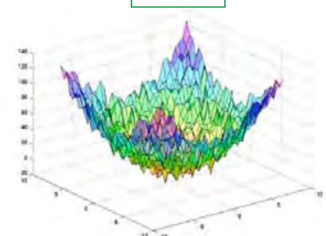

```
[X,Y] = meshgrid(-8:.5:8);
   Z   =   X.^2 + Y.^2;
   Z1  =   Z + 4*randn(33);
   surf(X,Y,Z1)
   colormap hsv
   alpha(.4)
```

Effect of Increasing Noise on Cost Function

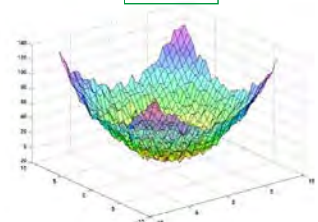
$\sigma_z = 0$



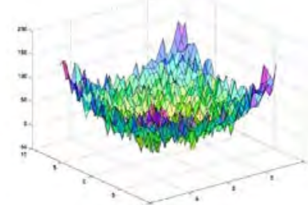
$\sigma_z = 8$



$\sigma_z = 4$

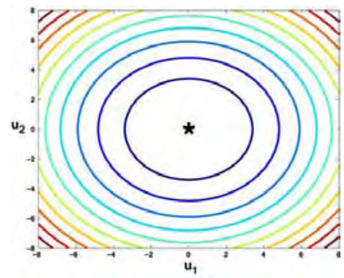


$\sigma_z = 16$

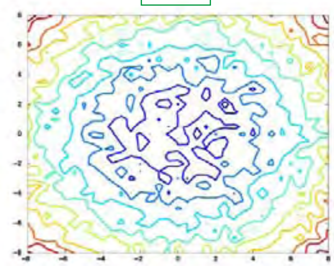


Iso-Cost Contours Lose Structure with Increasing Noise

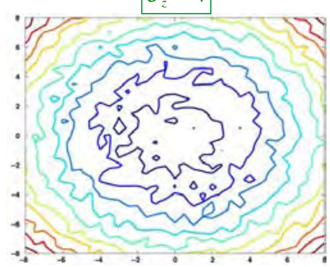
$\sigma_z = 0$



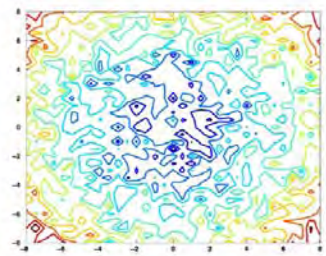
$\sigma_z = 8$



$\sigma_z = 4$

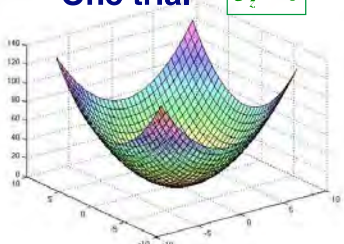


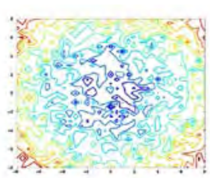
$\sigma_z = 16$

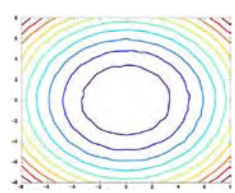


Effect of Averaging on Noisy Cost Function

- **One trial** $\sigma_z = 0$

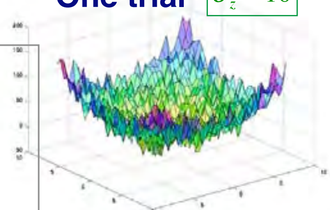




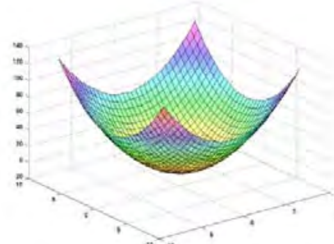


```
[X,Y] = meshgrid(-8:.5:8);
   Z   =   X.^2 + Y.^2;
   Z1  =   Z + 16*randn(33);
   Z2  =   Z1;
%   Averaged Z1
   for k = 1:1000
      Z1  =   Z + 16*randn(33);
      Z2  =   Z1 * (1/(k + 1)) + Z2 * (k/(k+1));
   end
   figure
   surf(X,Y,Z2)
   colormap hsv
   alpha(.4)
```

- **One trial** $\sigma_z = 16$



- **1,000 trials** $\sigma_z = 16$



Estimating the Probability of Coin Flips

- **Single coin**
  - **Exhaustive search: Correct answer in 2 trials**
  - **Random search (20,000 trials)**

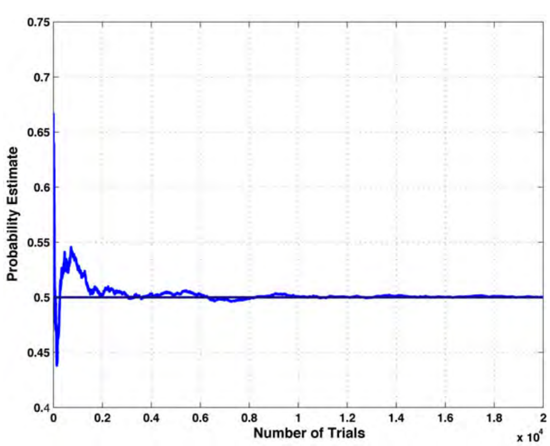
- **21 coins**
  - **Exhaustive search: Correct answer in $n^m = 2^{21}$ = 2,097,152 trials**
  - **Random search (20,000 trials)**

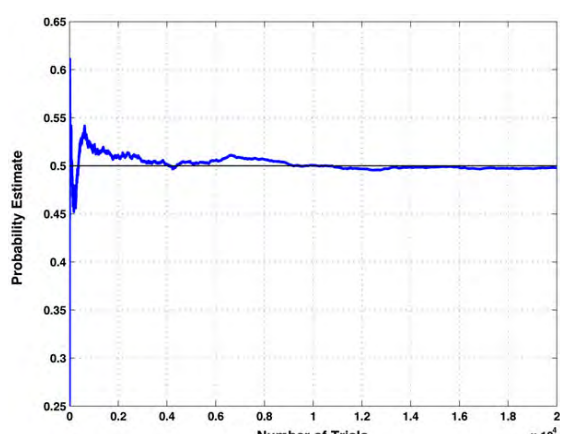
```
%   Single coin
   x       =  [];
   prob  =  round(rand);
   for k = 1:20000
      prob    =  round(rand) * (1/(k + 1)) + prob * (k/(k+1));
      x         =  [x prob];
   end
   figure
   plot(x), grid

%   21 coins
   y       =  [];
   prob  =  round(rand);
   for k = 1:20000
      for j = 1:21
         coin(j) =  round(rand);
      end
      score  =  sum(coin);
      if score > 10
         result  =  1;
      else   result  =  0;
      end
      prob =   result * (1/(k + 1)) + prob * (k/(k+1));
      y       =  [y prob];
   end
   figure
   plot(y), grid
```


Random Search Excels When There are Many Uncertain Parameters

- **Single coin**
  - **Exhaustive search: Correct answer in 2 trials**
  - **Random search (20,000 trials)**

- **21 coins**
  - **Exhaustive search: Correct answer in $n^m = 2^{21}$ = 2,097,152 trials**
  - **Random search (20,000 trials)**





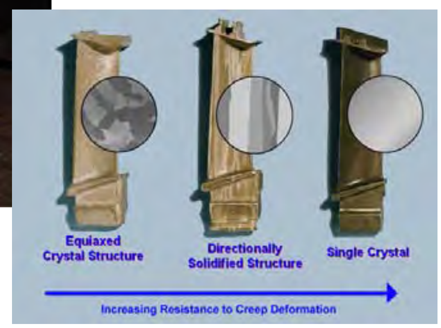
Physical Annealing

- Produce a strong, hard object made of crystalline material
  - High temperature allows molecules to redistribute to relieve stress, remove dislocations
  - Gradual cooling allows large, strong crystals to form
  - Low temperature “working” produces desired crystal structure and shape

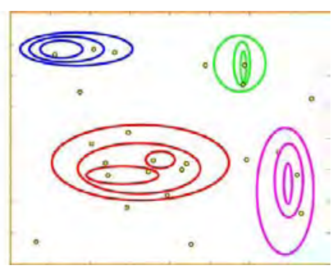
Turbojet Engine Turbines

Turbine Blade Casting

Single-Crystal Turbine Blade



Simulated Annealing Algorithm

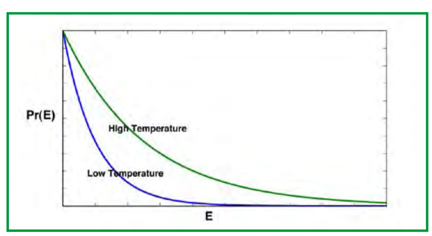


- **Goal:** Find global minimum among local minima
- **Approach:** Randomized search, with convergence that emulates *physical annealing*
  - Evaluate cost, J_k
  - Accept if $J_k < J_{k-1}$
  - Accept with probability $\Pr(E)$ if $J_k > J_{k-1}$
- Probability distribution of energy state, E *(Boltzmann Distribution)*

$$\Pr(E) \propto e^{-E/kT}$$

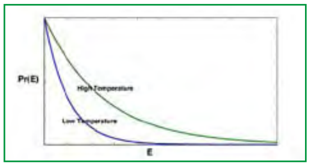
k: Boltzmann's constant

T: Temperature



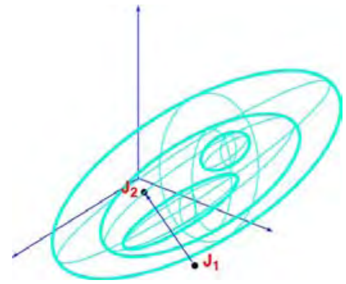
- Algorithm’s “cooling schedule” accepts many bad guesses at first, fewer as iteration number, k, increases

General Simulated Annealing Algorithm
http://www.mathworks.com/matlabcentral/fileexchange/10548-general-simulated-annealing-algorithm



Application of Annealing Principle to Search

- **If cost decreases ($J_2 < J_1$), always accept new point**
- **If cost increases ($J_2 > J_1$), accept new point with probability proportional to *Boltzmann factor***

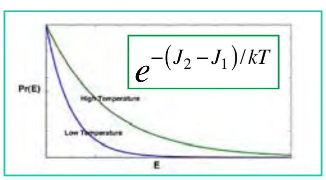


$$\boxed{e^{-(J_2 - J_1)/kT}}$$

- **Occasional diversion from convergent path intended to prevent entrapment by a local minimum**
- **As search progresses, decrease kT, making probability of accepting a cost increase smaller**

Realistic Bird Flight Animation by SA
http://www.youtube.com/watch?v=SoM1nS3uSrY

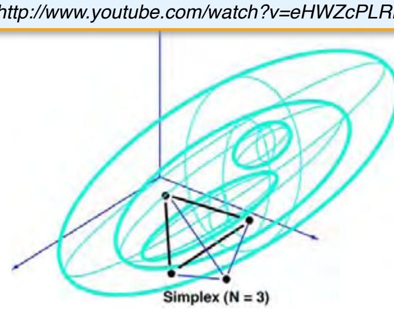
SA Face Morphing
http://www.youtube.com/watch?v=SP3nQKnzexs



Combination of Simulated Annealing with Downhill Simplex Method

- **Introduce random "wobble" to simplex search**
  - **Add random components to costs evaluated at vertices**
  - **Project new vertex as before based on modified costs**
  - **With large T, this becomes a random search**
  - **Decrease random components on a "cooling" schedule**

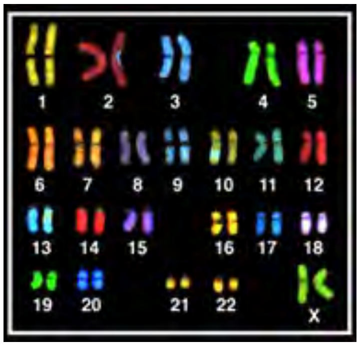
SA Mona Lisa
http://www.youtube.com/watch?v=eHWZcPLRMjQ

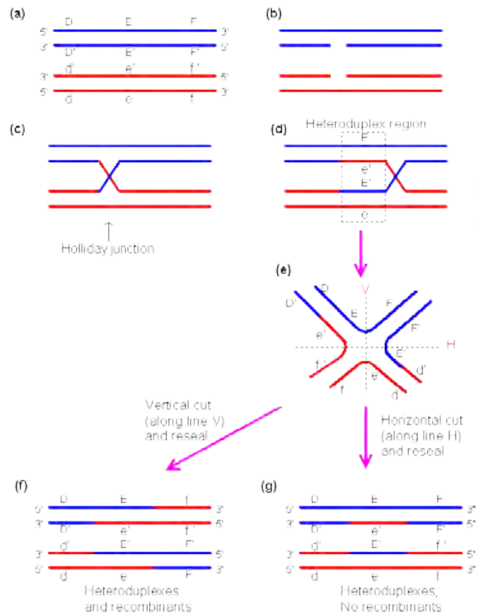


- **Same annealing strategy as before**
  - **If cost decreases ($J_2 < J_1$), always accept new point**
  - **If cost increases ($J_2 > J_1$), accept new point probabilistically**
  - **As search progresses, decrease T**

$$\begin{aligned}
J_{1_{SA}} &= J_1 + \Delta J_1(rng) \\
J_{2_{SA}} &= J_2 + \Delta J_2(rng) \\
J_{3_{SA}} &= J_3 + \Delta J_3(rng) \\
\ldots &= \ldots
\end{aligned}$$

Genetic Coding, Recombination, and Mutation





Broad Characteristics of Genetic Algorithms

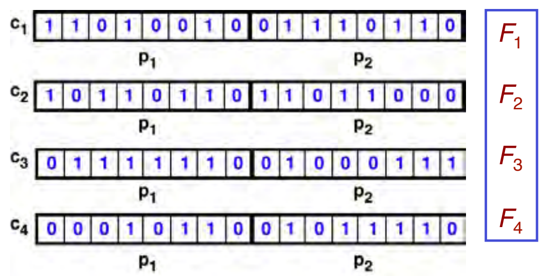
- Search based on the coding of a parameter set, not the parameters themselves
- Search evolves from a population of points
- "Blind" search, i.e., without gradient
- Probabilistic transitions from one control state to another
- Control parameters assembled as genes of a single chromosome strand (Example: four 4-bit parameters)

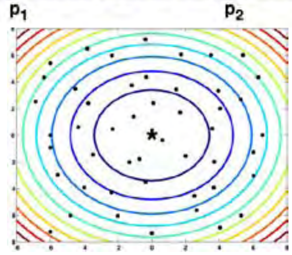
1	1	0	1	0	0	1	0	0	1	1	1	0	1	1	0
p_1				p_2				p_3				p_4			

Progression of a Genetic Algorithm

Most fit chromosome evolves from a sequence of reproduction, crossover, and mutation

- Initialize algorithm with N (even) random chromosomes, c_n (two 8-bit genes or parameters in example)
- Evaluate fitness, F_n, of each chromosome
- Compute total fitness, F_{total}, of chromosome population



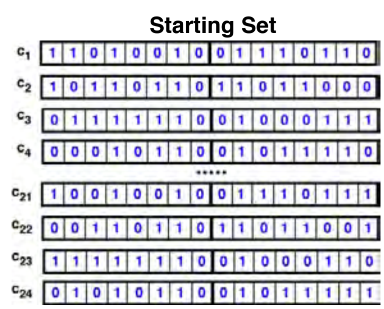


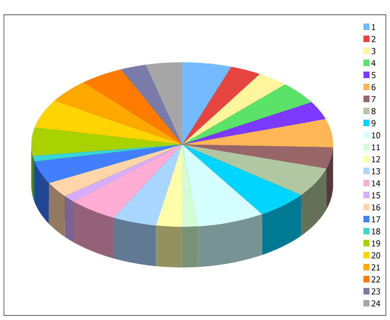
$$F_{total} = \sum_{n=1}^{N} F_n$$

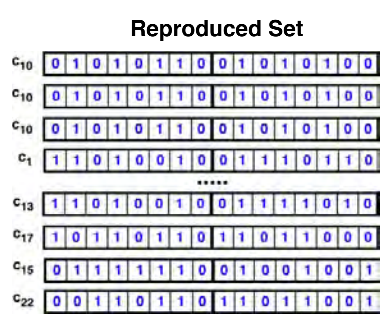
Bigger F is better

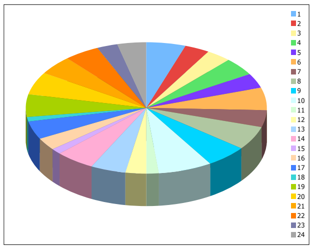
Genetic Algorithm: Reproduction

- Reproduce N additional copies of the N originals with probabilistic weighting based on relative fitness, F_n/F_{total}, of originals (Survival of the fittest)
- Roulette wheel selection:
  - $\Pr(c_n) = F_n/F_{total}$
  - Multiple copies of most-fit chromosomes
  - No copies of least-fit chromosomes



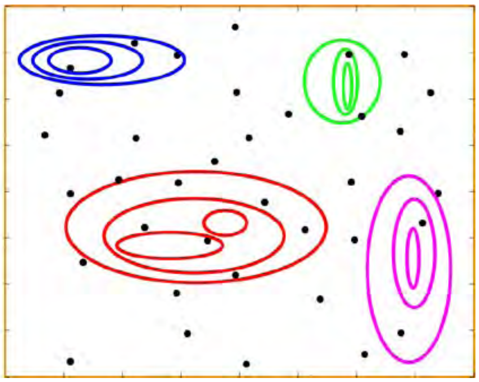




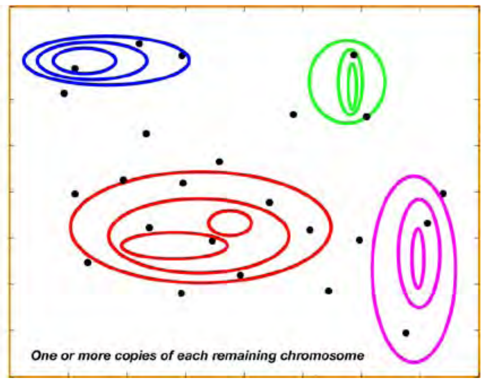


Reproduction Eliminates Least Fit Chromosomes Probabilistically

Starting Set

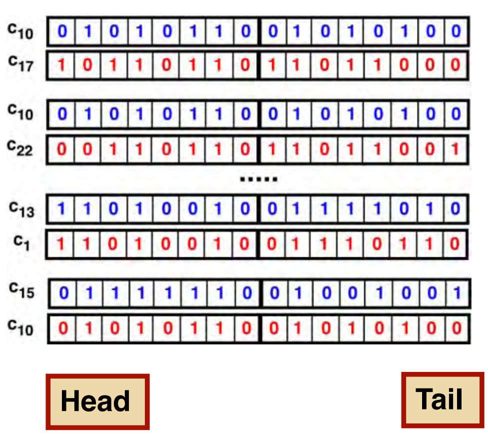


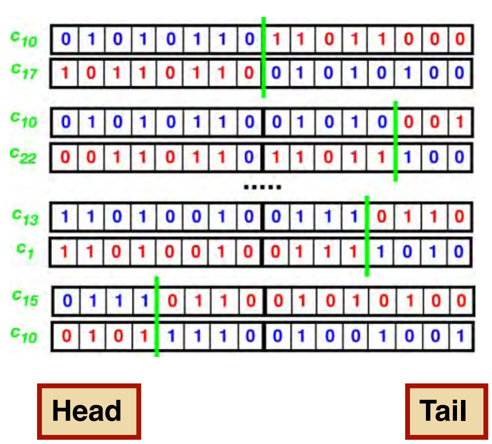
Reproduced Set



Genetic Algorithm: Crossover

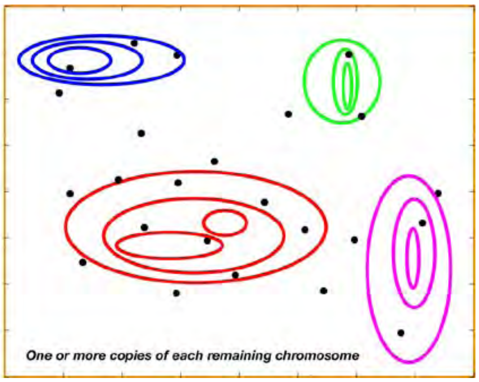
- Arrange *N* new chromosomes in *N*/2 pairs chosen at random
- Interchange tails that are cut at random locations



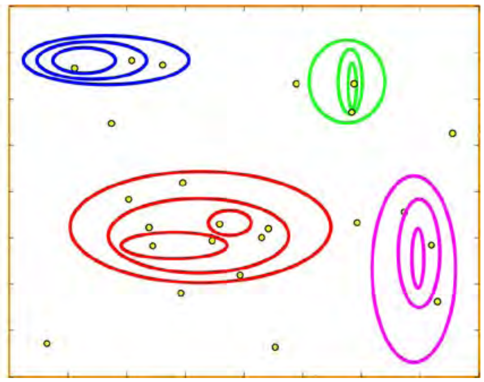


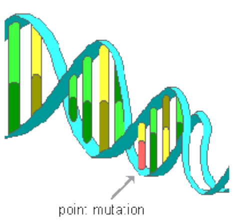
Crossover Creates New Chromosome Population Containing Old Gene Sequences

Reproduced Set



Crossover Set

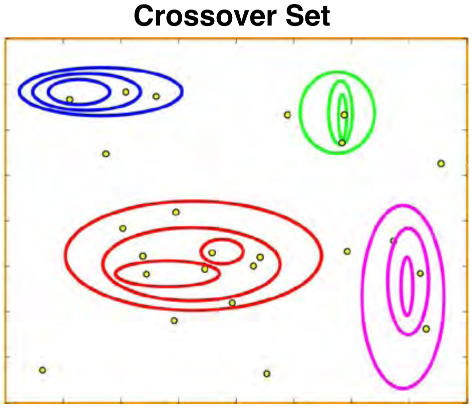




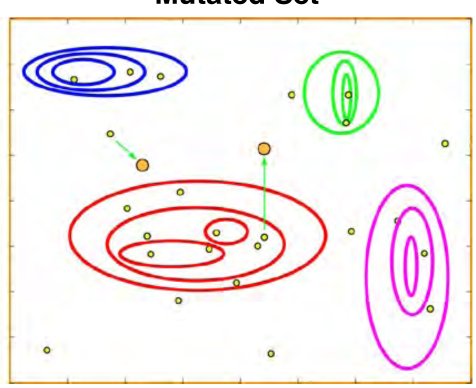
Genetic Algorithm: Mutation

Flip a bit, 0 -> 1 or 1 -> 0, at random every 1,000 to 5,000 bits

Crossover Set



Mutated Set



1	1	0	1	0	0	1	0	0	1	1	1	0	1	1	0

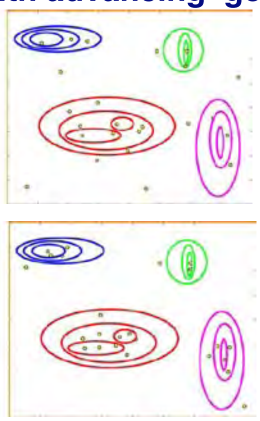
p_1 p_2

1	1	0	1	0	0	1	0	1	1	1	1	0	1	1	0

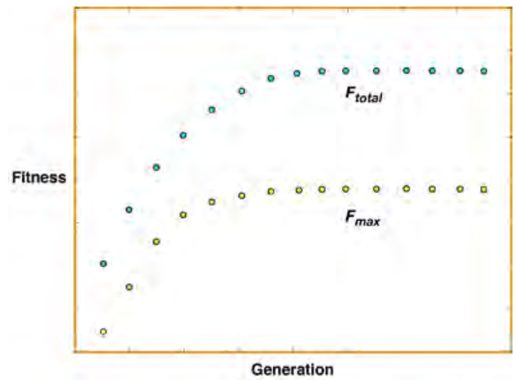
p_1 p_2

Create New Generations By Reproduction, Crossover, and Mutation Until Solution Converges

Chromosomes narrow in on best values with advancing generations



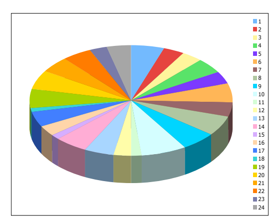
F_{max} and F_{total} increase with advancing generations



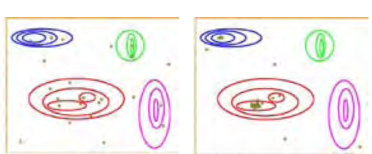
Open Genetic Algorithm Toolbox
http://www.mathworks.com/matlabcentral/fileexchange/37998-open-genetic-algorithm-toolbox

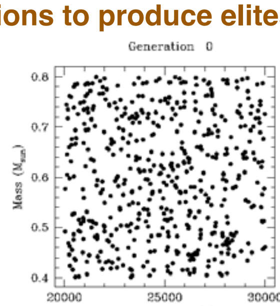
Comments on GA

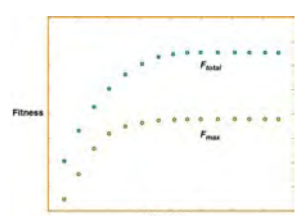
GA Mona Lisa
http://www.youtube.com/watch?v=rGt3iMAJVT8



- Short, fit genes tend to survive crossover
- Random location of crossover
  - produces large and small variations in genes
  - interchanges genes in chromosomes
- Multiple copies of best genes evolve
- Alternative implementations
  - Real numbers rather than binary numbers
  - Retention of "elite" chromosomes
  - Clustering in "fit" regions to produce elites

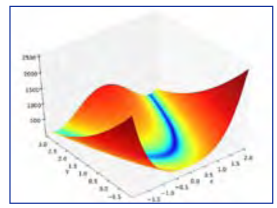






Particle Swarm Optimization

- Converse of the GA: Uses multiple cost evaluations to guide parameter search directly
- Stochastic, population-based algorithm
- Search for optimizing parameters modeled on social behavior of groups that possess cognitive consistency
- Particles = Parameter vectors
- Particles have position and velocity
- Projection of own best (Local best)
- Knowledge of swarm's best
  - Neighborhood best
  - Global best



Peregrine Falcon Hunting Starlings in Rome
https://www.youtube.com/watch?v=V-mCuFYfJdI

Particle Swarm Optimization

$$\text{Find } \min_{\mathbf{u}} J(\mathbf{u}) = J^*(\mathbf{u}^*)$$

$$\textbf{Jargon:} \quad \text{argmin} J(\mathbf{u}) = \mathbf{u}^*$$

$$i.e., \text{ argument of } J \text{ that minimizes } J$$

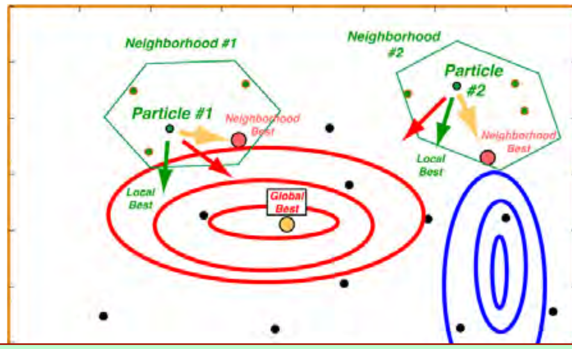
Recursive algorithm to find best particle or configuration of particles

$$\mathbf{u}: \text{ Parameter vector} \sim \textbf{"Position"} \text{ of the particles}$$

$$\mathbf{v}: \textbf{"Velocity"} \text{ of } \mathbf{u}$$

$$\dim(\mathbf{u}) = \dim(\mathbf{v}) = \text{ Number of particles}$$

Particle Swarm Optimization



- **Local best: RNG, downhill simplex, or SA step for each particle**
- **Neighborhood best: argmin of closest *n* neighboring points**
- **Global best: argmin of all particles**

$$\mathbf{u}_k = \mathbf{u}_{k-1} + a\mathbf{v}_{k-1}$$

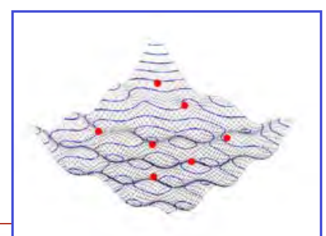
$$\mathbf{v}_k = b\mathbf{v}_{k-1} + c\left(\mathbf{u}_{best_{local_{k-1}}} - \mathbf{u}_{k-1}\right) + d\left(\mathbf{u}_{best_{neighborhood_{k-1}}} - \mathbf{u}_{k-1}\right) + e\left(\mathbf{u}_{best_{global_{k-1}}} - \mathbf{u}_{k-1}\right)$$

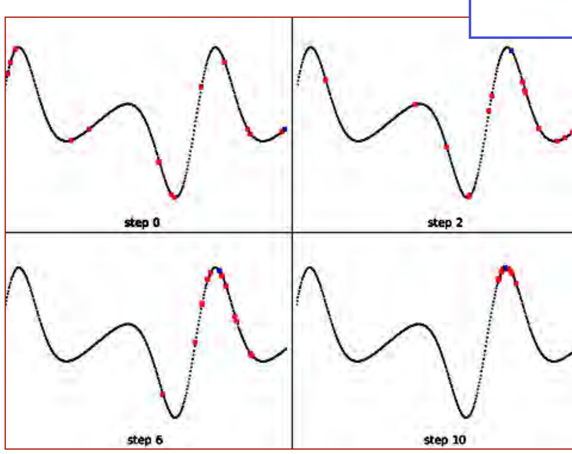
$\mathbf{u}_0$: Starting value from random number generator

$\mathbf{v}_0$: Zero

a, b, c, d : Search tuning parameters

Particle Swarm Optimization



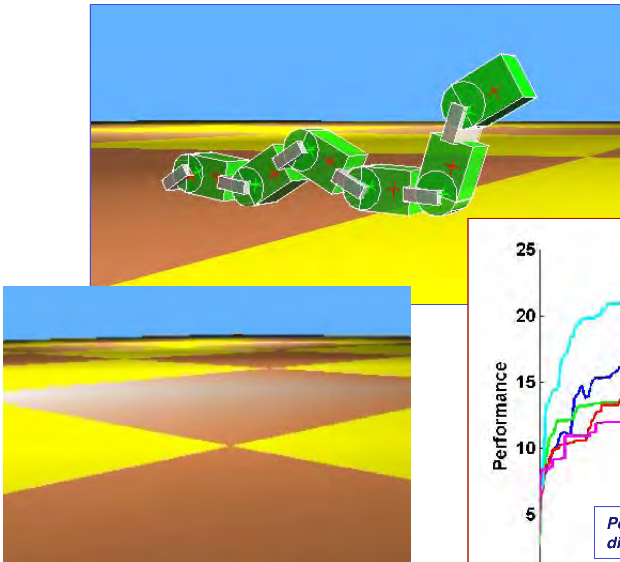


Particle Swarm Toolboxes

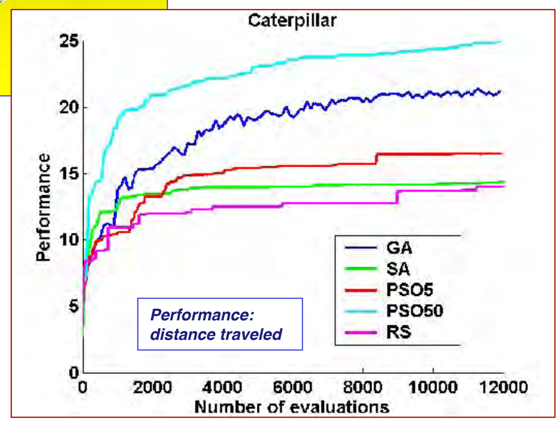
http://www.mathworks.com/matlabcentral/fileexchange/7506

http://www.mathworks.com/matlabcentral/fileexchange/25986-another-particle-swarm-toolbox

Comparison of Algorithms in Caterpillar Gait-Training Example



Y. Bourquin, U. Sussex, BIRG



Next Time:
Dynamic Optimal Control

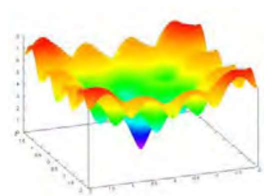
Supplemental Material

Cost Function and Gradient Searches

- Evaluate *J* and search for smallest value

$$
\begin{aligned}
J_o &= J(\mathbf{u}_o) = \textit{starting guess} \\
J_1 &= J_o + \Delta J_1(\mathbf{u}_o + \Delta \mathbf{u}_1) \ \textit{such that} \quad J_1 < J_o \\
J_2 &= J_1 + \Delta J_2(\mathbf{u}_1 + \Delta \mathbf{u}_2) \ \textit{such that} \quad J_2 < J_1
\end{aligned}
$$

Stop when difference between J_n and J_{n-1} is negligible



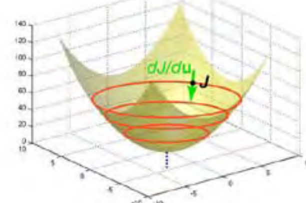
- ***J*** **is a scalar**
- ***J*** **provides no search direction**

- Evaluate $\partial J/\partial$u and search for zero

$$
\left(\frac{\partial J}{\partial \mathbf{u}}\right)_o = \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_0} = \textit{starting guess}
$$

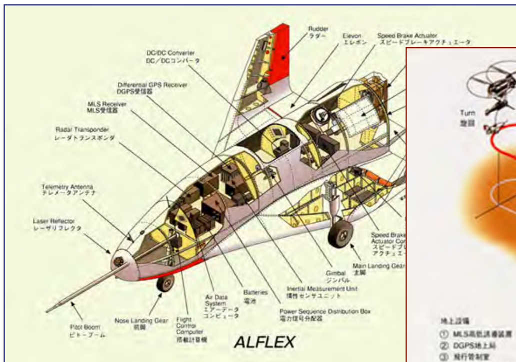
$$
\left(\frac{\partial J}{\partial \mathbf{u}}\right)_n = \left(\frac{\partial J}{\partial \mathbf{u}}\right)_{n-1} + \Delta\left(\frac{\partial J}{\partial \mathbf{u}}\right)_n = \left.\frac{\partial J}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{u}_n} \ \textit{such that} \quad \left|\frac{\partial J}{\partial \mathbf{u}}\right|_n < \left|\frac{\partial J}{\partial \mathbf{u}}\right|_{n-1}
$$

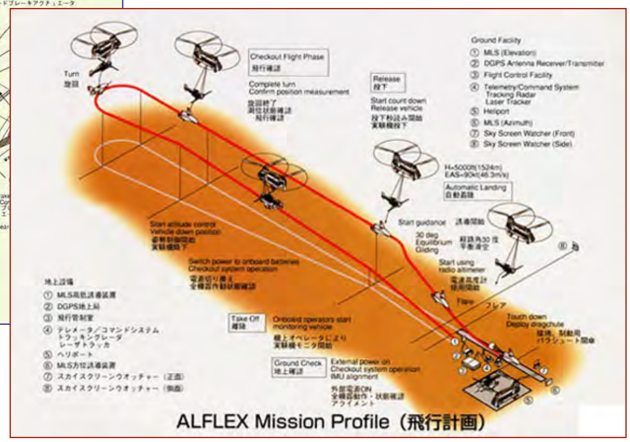
… until gradient is close enough to zero



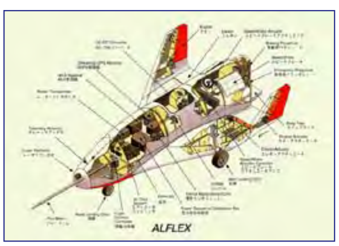
- **$\partial J/\partial$u is a vector**
- **$\partial J/\partial$u indicates feasible search direction**

Comparison of SA, DS, and GA in Designing a PID Controller: ALFLEX Reentry Test Vehicle





Motoda, Stengel, and Miyazawa, 2002



Parameter Uncertainties and Touchdown Requirements for ALFLEX Reentry Test Vehicle

Table 4 Uncertain parameters for ALFLEX model

Category	Number of parameters
Mass parameters	5
Aerodynamics	27
Actuator dynamics	9
Sensor dynamics and error	38
Atmospheric condition	6
Initial condition and error at release	18

Table 5 Requirements of ALFLEX touchdown performance

Touchdown states	Requirement
Position,[a] m	$X > 0$, $\|Y\| < 18$
Velocity, m/s	$V_G < 62$, $\dot{Z} < 3$
Attitude, deg	$\Theta < 23$, $\|\Phi\| < 10$, $\|\Psi\| < 8$
Side slip, deg	$\|\beta_G\| < 8$

[a]Runway coordination; the origin is at the runway threshold, the X axis is directed along the runway centerline, and the Z axis is directed downward.

ALFLEX Pitch Attitude Control Logic

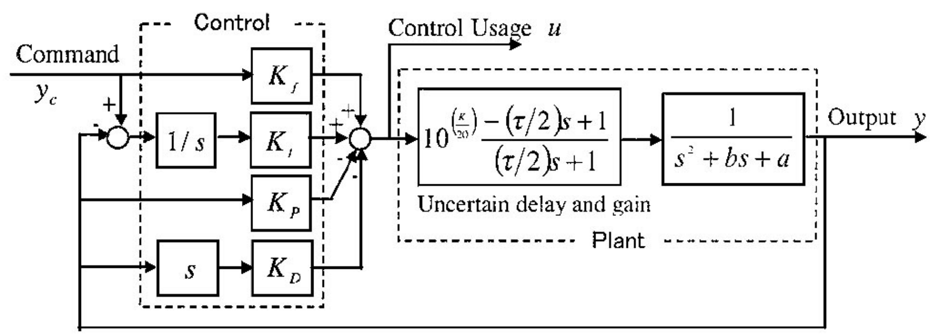


Fig. 5 Simplified model of pitch attitude control.

Comparison of SA, DS, and GA in Designing a PID Controller

Table 2 Comparison of three optimization methods

Parameter	Simulated annealing	Downhill–simplex	Genetic algorithm
Best design vector $\boldsymbol{d}^*$			
K_f	0.866	2.95	0.423
K_P	3.88	4.33	4.11
K_I	1.04	2.24	1.08
K_D	3.05	3.31	3.18
Total simulation number	31,998	13,604	121,552
Number of evaluated design vectors	66	51	745

Table 3 Results of 10,000 Monte Carlo evaluations using optimized design parameters

Method	Cost function J	[Confidence interval]
Simulated annealing	0.0135	[0.012, 0.016]
Downhill–simplex	0.0278	[0.025, 0.031]
Genetic algorithm	0.0133	[0.012, 0.015]

Genetic Algorithm Applications

GA Mona Lisa, 2
http://www.youtube.com/watch?v=A8x4Lyj33Ro&NR=1

Learning Network Weights for a Flapping Wing Neural-Controller
http://www.youtube.com/watch?v=BfY4jRtcE4c&feature=related

Virtual Creature Evolution
http://www.youtube.com/watch?v=oquKOVfzGfk&NR=1

Evolution of Locomotion
http://www.youtube.com/watch?v=STkfUZtR-Vs&feature=related

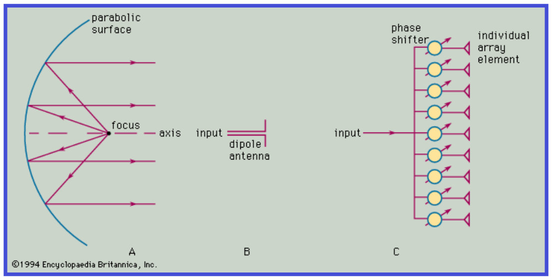
Examples of Particle Swarm Optimization

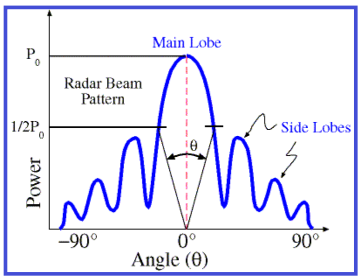
Robot Swarm Animation
http://www.youtube.com/watch?v=RLIA1EKfSys

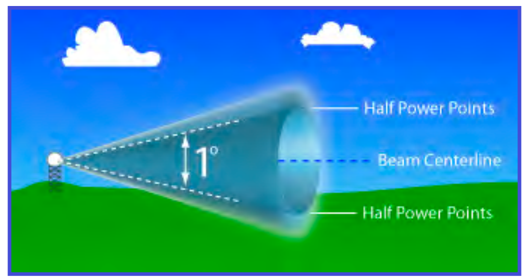
Swarm-Bots Finding a Path and Retrieving Object
http://www.youtube.com/watch?v=Xs_Y22N1r_A

Learning Robot Control System Gains
http://www.youtube.com/watch?v=itf8NHF1bS0&feature=related

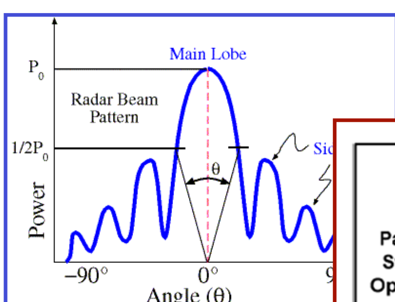
Parabolic and Phased-Array Radar Antenna Patterns

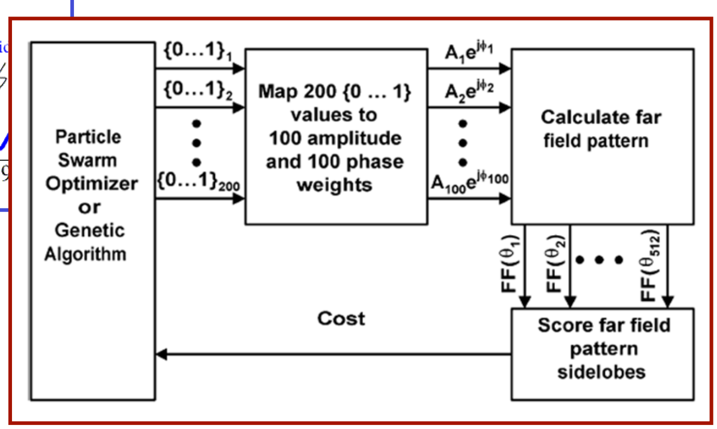






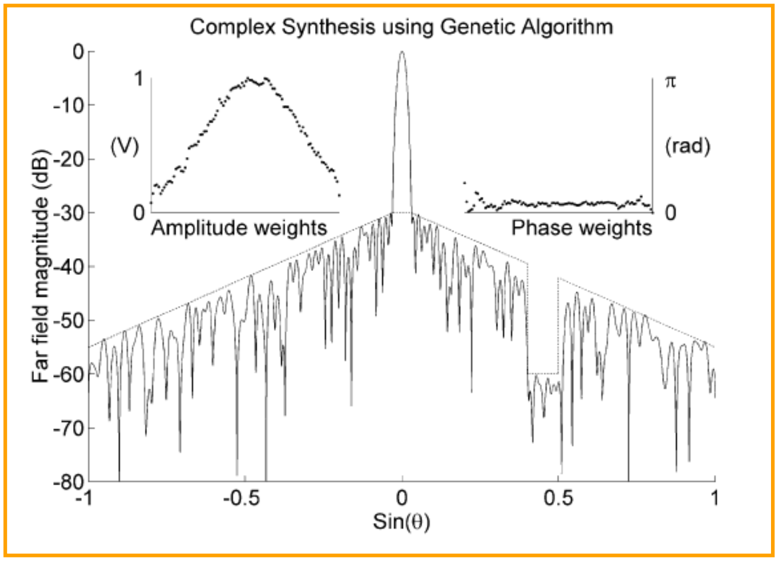
Phased-Array Antenna Design Using Genetic Algorithm or Particle Swarm Optimization



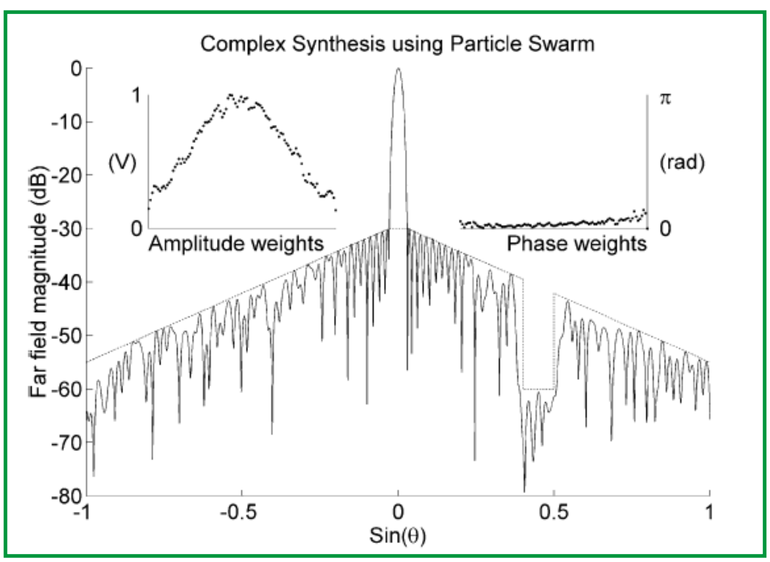


Boeringer and Warner, 2004

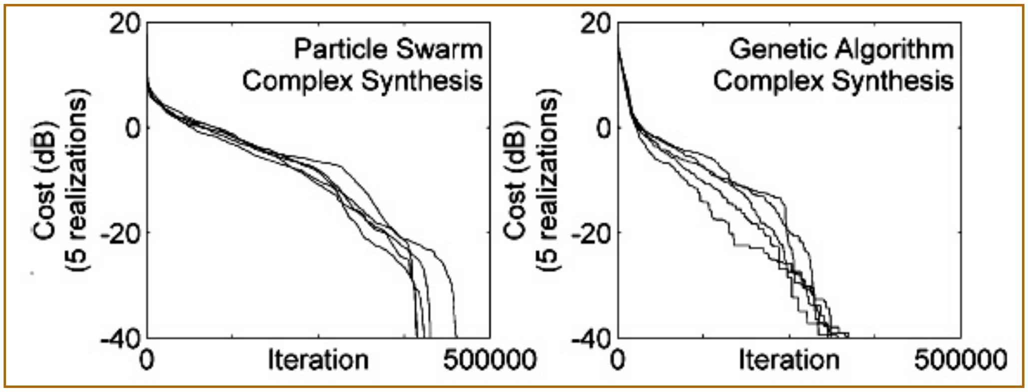
Phased-Array Antenna Design Using Genetic Algorithm



Phased-Array Antenna Design Using Particle Swarm Optimization



Comparison of Phased-Array Antenna Designs



Summary of Gradient-Free Optimization Algorithms

- Grid search
  - Uniform coverage of search space
- Random Search
  - Arbitrary placement of test parameters
- Downhill Simplex Method
  - Robust search of difficult cost function topology
- Simulated Annealing
  - Structured random search with convergence feature
- Genetic Algorithm
  - Coding of the parameter set
- Particle Swarm Optimization
  - Intuitively appealing, efficient heuristic