Numerical Optimization

Robert Stengel
Robotics and Intelligent Systems MAE 345, Princeton University, 2017

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

Copyright 2017 by Robert Stengel. All rights reserved. For educational use only.
http://www.princeton.edu/~stengel/MAE345.html

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

\[
\frac{\partial J}{\partial u} = \left. \frac{\partial J}{\partial u} \right|_{u=u_o} = \text{starting guess}
\]

\[
\left( \frac{\partial J}{\partial u} \right)_{n+1} = \left. \left( \frac{\partial J}{\partial u} \right)_{n+1} + \Delta \left( \frac{\partial J}{\partial u} \right)_{n+1} \right|_{u=u_o} \text{ such that } \left| \frac{\partial J}{\partial u} \right|_{n+1} < \left| \frac{\partial J}{\partial u} \right|_{n+1}
\]

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

\[
J_o = J(u_o) = \text{starting guess}
\]

\[
J_1 = J_o + \Delta J_1 (u_o + \Delta u) \text{ such that } J_1 < J_o
\]

\[
J_2 = J_1 + \Delta J_2 (u_1 + \Delta u_2) \text{ such that } J_2 < J_1
\]

Gradient/Hessian Search to Minimize a Quadratic Function

Cost function, gradient, and Hessian matrix

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

\[
= \frac{1}{2} (u^T R u - u^T R u^* - u^* R u + u^* R u^*)
\]

\[
\frac{\partial J}{\partial u} = (u - u^*)^T R = 0 \text{ when } u = u^*
\]

\[
\frac{\partial^2 J}{\partial u^2} = R = \text{symmetric constant > 0}
\]

Guess a starting value of $u$, $u_o$

\[
\left. \frac{\partial J}{\partial u} \right|_{u=u_o} = (u_o - u^*)^T R = (u_o - u^*)^T R \left( \frac{\partial^2 J}{\partial u^2} \right)_{u=u_o}
\]

\[
(u_o - u^*)^T = \left. \frac{\partial J}{\partial u} \right|_{u=u_o} R^{-1} \quad \text{(row)}
\]

Solve for $u^*$

\[
u^* = u_o - R^{-1} \left[ \left. \frac{\partial J}{\partial u} \right|_{u=u_o} \right]^T \quad \text{(column)}
\]
Optimal Value of Quadratic Function Found in a One Step

\[ u^* = u_o - R^{-1} \left[ \frac{\partial J}{\partial u_{u=u_o}} \right]^T \]

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

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

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

- Cost function and derivatives
- First guess at optimal control
- Derivatives at starting point

\[ u^* = u_o - R^{-1} \left[ \frac{\partial J}{\partial u_{u=u_o}} \right]^T \]

| \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix} |

\[ \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 9/5 \\ -2/5 \end{bmatrix}; \quad \begin{bmatrix} 9/5 \\ -2/5 \end{bmatrix} = \begin{bmatrix} 11/42 \end{bmatrix} \]

Solution from starting point

Numerical Example
Newton-Raphson Iteration

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

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

Optimal solution requires multiple steps

Newton-Raphson Iteration

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

\[
u_{k+1} = u_k - \left[ \frac{\partial^2 J}{\partial u^2}_{|u=u_k} \right]^{-1} \left[ \frac{\partial J}{\partial u}_{|u=u_k} \right]^{T}
\]
Difficulties with Newton-Raphson Iteration

\[
\mathbf{u}_{k+1} = \mathbf{u}_k - \nabla J |_{u=u_k} \cdot \mathbf{u}^T \\
\text{• Good when close to the optimum, but ...} \\
\text{• Hessian matrix (i.e., the curvature) may be} \\
  \quad \text{Hard to estimate, e.g., large effects of small errors} \\
  \quad \text{Locally misleading, e.g., wrong curvature} \\
\text{• Gradient searches focus on local minima}
\]

Steepest-Descent Algorithm Multiplies Gradient by a Scalar Constant

\[
\mathbf{u}_{k+1} = \mathbf{u}_k - \epsilon \left( \frac{\partial J}{\partial \mathbf{u}} \bigg|_{u=u_k} \right)^T \\
\text{• Replace Hessian matrix by a scalar constant} \\
\text{• Gradient is orthogonal to equal-cost contours}
\]
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 Gain

Find optimal gain by evaluating cost, $J$, for intermediate solutions (with same $\partial J/\partial u$)

Adjustment rule for $\epsilon$

- Starting estimate, $J_0$
- First estimate, $J_1$, using $\epsilon$
- Second estimate, $J_2$, using $2\epsilon$

If $J_2 > J_1$
- Quadratic fit through three points to find $\epsilon^*$
- Else, third estimate, $J_3$, using $4\epsilon$
- ...

Use optimal gain, $\epsilon^*$, on each major iteration
Gradient Search

Issues

- 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

\[
\mathbf{u}_{k+1} = \mathbf{u}_k - \varepsilon \left[ \frac{\partial J}{\partial \mathbf{u}} \right]_u
\]

\[
\mathbf{u}_{k+1} = \mathbf{u}_k - \left( \frac{\partial^2 J}{\partial \mathbf{u}^2} \right)_{u_k}^{-1} \left[ \frac{\partial J}{\partial \mathbf{u}} \right]_{u_k}^T
\]

Gradient-Free Search:

Grid-Based Search

\[
J_0 = J(\mathbf{u}_0) = \text{starting guess}
\]

\[
J_1 = J_0 + \Delta J_1(\mathbf{u}_0 + \Delta \mathbf{u}_0) \text{ such that } J_1 < J_0
\]

\[
J_2 = J_1 + \Delta J_2(\mathbf{u}_1 + \Delta \mathbf{u}_1) \text{ such that } J_2 < J_1
\]
Gradient-Free Search:  
Random Search

\[ J_o = J(u_o) = \text{starting guess} \]
\[ J_1 = J_o + \Delta J_o (u_o + \Delta u_o) \text{ such that } J_1 < J_o \]
\[ J_2 = J_1 + \Delta J_1 (u_1 + \Delta u_1) \text{ such that } J_2 < J_1 \]

*Select control parameters using a random number generator

Three-Parameter Grid Search

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

\( n = 125 \)
\( n = 1000 \)
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)

- **Simplex**: $N$-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 $N = 2$)
  - Reflection [equal distance along direction]
  - Expansion [longer distance along direction]
  - Contraction [shorter distance along direction]
  - Shrink [replace all but best point with points contracted toward best point]

- Evaluate cost at new vertex (e.g., $J_2$ at right)
- Drop $J_1$ vertex, and form simplex with new vertex
- Repeat until cost is “small enough” (termination)
- MATLAB implementation: `fminsearch`

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

Example (from Wikipedia):
Evaluation of $\pi$ from percentage of points that fall within the unit circle (30,000 trials)
Monte Carlo Evaluation of Systems and Cost Functions

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

\[
\begin{align*}
[X,Y] &= \text{meshgrid(-8:.5:8);} \\
Z &= X.^2 + Y.^2; \\
Z1 &= Z + 4*\text{randn(33);} \\
surf(X,Y,Z1) \\
\text{colormap hsv} \\
\text{alpha(.4)}
\end{align*}
\]

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

\[
\begin{align*}
\sigma_z = 0 & & \sigma_z = 8 \\
\sigma_z = 4 & & \sigma_z = 16
\end{align*}
\]

Effect of Averaging on Noisy Cost Function

- **One trial** \( \sigma_z = 0 \)

- **One trial** \( \sigma_z = 16 \)

- **1,000 trials** \( \sigma_z = 16 \)

\[
\begin{align*}
[X,Y] &= \text{meshgrid}(-8:.5:8); \\
Z &= X.^2 + Y.^2; \\
Z1 &= Z + 16*\text{randn}(33); \\
Z2 &= Z1; \\
\text{% Averaged } Z1 \\
\text{for } k = 1:1000 \\
&= Z + 16*\text{randn}(33); \\
Z2 &= Z1 * (1/(k+1)) + Z2 * (k/(k+1)); \\
end \\
\text{figure} \\
\text{surf}(X,Y,Z2) \\
\text{colormap hsv} \\
\text{alpha(.4)}
\end{align*}
\]
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 \( n^n = 2^{21} = 2,097,152 \) trials
  - Random search (20,000 trials)

```matlab
% 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 \( n^n = 2^{21} = 2,097,152 \) 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” (e.g., squeezing, bending, drawing, shearing, and hammering) produces desired crystal structure and shape

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
- MATLAB implementation: **simulannealbnd** (Global Optimization Toolbox)

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
\[ e^{-(J_2 - J_1)/kT} \]
- 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
- 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 $T$

\[
J_{1,SA} = J_1 + \Delta J_1 (rng) \\
J_{2,SA} = J_2 + \Delta J_2 (rng) \\
J_{3,SA} = J_3 + \Delta J_3 (rng) \\
... = ...
\]
Genetic Coding: Replication, Recombination, and Mutation of Chromosomes

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 (using random number generator)
- Control parameters assembled as genes of a single chromosome strand (Example: four 4-bit parameters = four “genes”)
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, \( F_{\text{total}} \), of chromosome population

\[
F_{\text{total}} = \sum_{n=1}^{N} F_n
\]

Bigger \( F \) is better

Genetic Algorithm: Reproduction

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

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

Genetic Algorithm: Mutation

Flip a bit, 0 -> 1 or 1 -> 0, at random every 1,000 to 5,000 bits
Create New Generations By
Reproduction, Crossover, and Mutation
Until Solution Converges

Chromosomes narrow in on best values with advancing generations

$F_{\text{max}}$ and $F_{\text{total}}$ increase with advancing generations

MATLAB implementation: \texttt{ga}
(\textit{Global Optimization Toolbox})

Comments on GA

- 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

GA Mona Lisa
http://www.youtube.com/watch?v=rGt3lMAJVT8

https://en.wikipedia.org/wiki/Genetic_algorithm
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 Murmuration of Starlings in Rome**
[https://www.youtube.com/watch?v=V-mCuFYfJdl](https://www.youtube.com/watch?v=V-mCuFYfJdl)

---

**Particle Swarm Optimization**

\[
\text{Find } \min_u J(u) = J^*(u^*)
\]

**Jargon:** \(\text{argmin}_u J(u) = u^*\)

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

**Recursive algorithm to find best particle or configuration of particles**

\(\mathbf{u}: \) Parameter vector \~ \text{"Position"} of the particles

\(\mathbf{v}: \) \text{"Velocity"} of \(\mathbf{u}\)

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

[https://en.wikipedia.org/wiki/Particle_swarm_optimization](https://en.wikipedia.org/wiki/Particle_swarm_optimization)
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

\[
\begin{align*}
    \mathbf{u}_k &= \mathbf{u}_{k-1} + a\mathbf{v}_{k-1} \\
    \mathbf{v}_k &= b\mathbf{v}_{k-1} + c(\mathbf{u}_{\text{best}_k} - \mathbf{u}_{k-1}) + d(\mathbf{u}_{\text{best}_{\text{neighborhood}_k}} - \mathbf{u}_{k-1}) + e(\mathbf{u}_{\text{best}_{\text{global}_k}} - \mathbf{u}_{k-1})
\end{align*}
\]

- $\mathbf{u}_0$: Starting value from random number generator
- $\mathbf{v}_0$: Zero
- $a, b, c, d$: Search tuning parameters

MATLAB implementation: `particleswarm` *(Global Optimization Toolbox)*
Comparison of Algorithms in Caterpillar Gait-Training Example

Next Time:
Dynamic Optimal Control
Rosenbrock Function

Typical test function for numerical optimization algorithms

\[ J(u_1, u_2) = (1 - u_1)^2 + 100 (u_2 - u_1^2)^2 \]

Wolfram Alpha
Minimize[(1-u1)^2+100 (u2 - u1^2)^2, u1, u2]
Cost Function and Gradient Searches

- Evaluate $J$ and search for smallest value
  \[ J_0 = J(u_0) = \text{starting guess} \]
  \[ J_1 = J_0 + \Delta J_1(u_0 + \Delta u_1) \text{ such that } J_1 < J_0 \]
  \[ J_2 = J_1 + \Delta J_2(u_1 + \Delta u_2) \text{ such that } J_2 < J_1 \]

- $J$ is a scalar
- $J$ provides no search direction
- Evaluate $\partial J/\partial u$ and search for zero
  \[ \frac{\partial J}{\partial u} = \text{starting guess} \]
  \[ \frac{\partial J}{\partial u} = \frac{\partial J}{\partial u} \bigg|_{u_0} + \Delta \left( \frac{\partial J}{\partial u} \bigg|_{u_0} \right) \text{ such that } \left| \frac{\partial J}{\partial u} \bigg|_{u_n} \right| < \left| \frac{\partial J}{\partial u} \bigg|_{u_{n-1}} \right| \]

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

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

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Number of parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass parameters</td>
<td>5</td>
</tr>
<tr>
<td>Aerodynamics</td>
<td>27</td>
</tr>
<tr>
<td>Actuator dynamics</td>
<td>9</td>
</tr>
<tr>
<td>Sensor dynamics and error</td>
<td>38</td>
</tr>
<tr>
<td>Atmospheric condition</td>
<td>6</td>
</tr>
<tr>
<td>Initial condition and error at release</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 5 Requirements of ALFLEX touchdown performance

<table>
<thead>
<tr>
<th>Touchdown states</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position, m</td>
<td>$X &gt; 0,</td>
</tr>
<tr>
<td>Velocity, m/s</td>
<td>$V_C &lt; 62, Z &lt; 3$</td>
</tr>
<tr>
<td>Attitude, deg</td>
<td>$\Theta &lt; 23,</td>
</tr>
<tr>
<td>Side slip, deg</td>
<td>$</td>
</tr>
</tbody>
</table>

*Runway coordinate; 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

![Simplified model of pitch attitude control.](image)

Fig. 5 Simplified model of pitch attitude control.
Comparison of SA, DS, and GA in Designing a PID Controller

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Simulated annealing</th>
<th>Downhill-simplex</th>
<th>Genetic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best design vector $d^*$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$K_f$</td>
<td>0.866</td>
<td>2.95</td>
<td>0.423</td>
</tr>
<tr>
<td>$K_p$</td>
<td>3.88</td>
<td>4.33</td>
<td>4.11</td>
</tr>
<tr>
<td>$K_i$</td>
<td>1.04</td>
<td>2.24</td>
<td>1.08</td>
</tr>
<tr>
<td>$K_d$</td>
<td>3.05</td>
<td>3.31</td>
<td>3.18</td>
</tr>
<tr>
<td>Total simulation number</td>
<td>31,998</td>
<td>13,604</td>
<td>121,552</td>
</tr>
<tr>
<td>Number of evaluated design vectors</td>
<td>66</td>
<td>51</td>
<td>745</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Cost function $J$</th>
<th>Confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated annealing</td>
<td>0.0135</td>
<td>[0.012, 0.016]</td>
</tr>
<tr>
<td>Downhill-simplex</td>
<td>0.0278</td>
<td>[0.025, 0.031]</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>0.0133</td>
<td>[0.012, 0.015]</td>
</tr>
</tbody>
</table>

Genetic Algorithm Applications

- GA Mona Lisa, 2
  [Video](http://www.youtube.com/watch?v=A8x4Lyj33Ro&NR=1)

- Learning Network Weights for a Flapping Wing Neural-Controller
  [Video](http://www.youtube.com/watch?v=Bl74jRtcE4c&feature=related)

- Virtual Creature Evolution
  [Video](http://www.youtube.com/watch?v=oquKOV1zGfk&NR=1)

- Evolution of Locomotion
  [Video](http://www.youtube.com/watch?v=STkfUZiR-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 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