Numerical Optimization

Joachim Vandekerckhove

Motivating Problems for Numerical Optimization

- **Function Minimization:** Find the minimum of a function $f(x)$ over a given range of x.

Motivating Problems for Numerical Optimization

- **Function Minimization:** Find the minimum of a function $f(x)$ over a given range of x.
- **Parameter Estimation:** Given a set of observations, estimate the parameters of a model that best fit the data.

Motivating Problems for Numerical Optimization

- **Function Minimization:** Find the minimum of a function $f(x)$ over a given range of x.
- **Parameter Estimation:** Given a set of observations, estimate the parameters of a model that best fit the data.
- **Machine Learning:** Train a model to predict outcomes based on input features by minimizing a loss function.

Motivating Problems for Numerical Optimization

- **Function Minimization:** Find the minimum of a function $f(x)$ over a given range of x.
- **Parameter Estimation:** Given a set of observations, estimate the parameters of a model that best fit the data.
- **Machine Learning:** Train a model to predict outcomes based on input features by minimizing a loss function.
- **Procedure Optimization:** Find the conditions where a certain procedure works best.

Numerical optimization in cognitive modeling

- Cognitive models have parameters that need to be estimated

Numerical optimization in cognitive modeling

- Cognitive models have parameters that need to be estimated
- Estimation means minimizing the loss $\ell(x, \theta)$...

$$\hat{\theta} \quad = \quad \arg\min_{\theta} \left(\ell(x; \theta)\right)$$

Numerical optimization in cognitive modeling

- Cognitive models have parameters that need to be estimated
- Estimation means minimizing the loss $\ell(x, \theta)$...

$$\hat{\theta} = \arg\min_{\theta} (\ell(x; \theta))$$

- ... or maximizing the likelihood $f(x|\theta)$:

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} (f(x|\theta)) \\ &= \arg\min_{\theta} (-\log(f(x|\theta))) \end{aligned}$$

Numerical optimization in cognitive modeling

- Cognitive models have parameters that need to be estimated
- Estimation means minimizing the loss $\ell(x, \theta)$...

$$\hat{\theta} = \arg\min_{\theta} (\ell(x; \theta))$$

- ... or maximizing the likelihood $f(x|\theta)$:

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} (f(x|\theta)) \\ &= \arg\min_{\theta} (-\log(f(x|\theta))) \end{aligned}$$

- To-be-optimized function is called target or objective function

Numerical optimization in cognitive modeling

Optimization is sometimes possible analytically but most often done using numerical methods:

1. Guess θ through some method

Numerical optimization in cognitive modeling

Optimization is sometimes possible analytically but most often done using numerical methods:

1. Guess θ through some method
2. Compute objective function

Numerical optimization in cognitive modeling

Optimization is sometimes possible analytically but most often done using numerical methods:

1. Guess θ through some method
2. Compute objective function
3. Check if the current guess is best yet

Numerical optimization in cognitive modeling

Optimization is sometimes possible analytically but most often done using numerical methods:

1. Guess θ through some method
2. Compute objective function
3. Check if the current guess is best yet
4. Repeat until good enough

Numerical optimization in cognitive modeling

Algorithm: Abstract Numerical Minimization

Input : objective function $\ell(\theta)$

Output: the learned parameters θ

1. Initial guess parameters θ
2. **while** $\ell(\theta)$ is too high **do**
   Propose new parameters: $\theta \leftarrow \theta + \text{update}$;
3. **return** θ;

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
    - Derivatives can be approximated using finite differences:

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$
    2. Guess θ_1 nearby and compute $\ell(x; \theta_1)$

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$
    2. Guess θ_1 nearby and compute $\ell(x; \theta_1)$
    3. If $\ell(x; \theta_1) < \ell(x; \theta_0)$, move further past θ_1, otherwise move back beyond θ_0

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$
    2. Guess θ_1 nearby and compute $\ell(x; \theta_1)$
    3. If $\ell(x; \theta_1) < \ell(x; \theta_0)$, move further past θ_1, otherwise move back beyond θ_0
    4. Decrease step size a litte

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$
    2. Guess θ_1 nearby and compute $\ell(x; \theta_1)$
    3. If $\ell(x; \theta_1) < \ell(x; \theta_0)$, move further past θ_1, otherwise move back beyond θ_0
    4. Decrease step size a litte
    5. Repeat until good enough

Numerical optimization in cognitive modeling

- Choice of numerical method depends on the problem and available resources, there's no free lunch
- Gradient-based optimization methods use derivatives to search for the minimum
  - Derivatives can be approximated using finite differences:
    1. Guess θ_0 through some method and compute $\ell(x; \theta_0)$
    2. Guess θ_1 nearby and compute $\ell(x; \theta_1)$
    3. If $\ell(x; \theta_1) < \ell(x; \theta_0)$, move further past θ_1, otherwise move back beyond θ_0
    4. Decrease step size a litte
    5. Repeat until good enough
- Other optimization methods include genetic algorithms, simulated annealing, particle swarm optimization, and the Nelder-Mead simplex

Gradient-based optimization

Gradient-based optimization

Gradient-based optimization

- Gradient descent works by iteratively adjusting the model's parameters in the direction of steepest descent of the objective function.

Gradient-based optimization

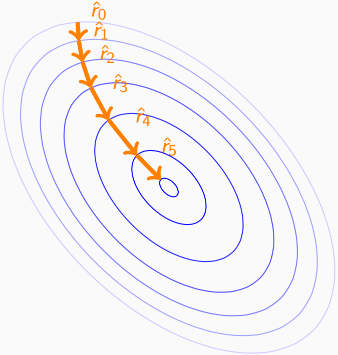
- Gradient descent works by iteratively adjusting the model's parameters in the direction of steepest descent of the objective function.
- Newton's method uses the Hessian matrix (i.e., the local curvature) to adjust the parameters.

Gradient-based optimization

- Gradient descent works by iteratively adjusting the model's parameters in the direction of steepest descent of the objective function.
- Newton's method uses the Hessian matrix (i.e., the local curvature) to adjust the parameters.
- Quasi-Newton methods approximate the Hessian matrix using a limited amount of information.

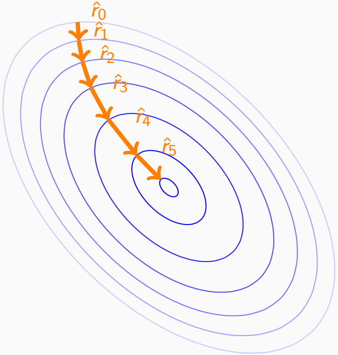
Gradient-based optimization

- Gradient descent works by iteratively adjusting the model's parameters in the direction of steepest descent of the objective function.
- Newton's method uses the Hessian matrix (i.e., the local curvature) to adjust the parameters.
- Quasi-Newton methods approximate the Hessian matrix using a limited amount of information.



Gradient-based optimization

- Gradient descent works by iteratively adjusting the model's parameters in the direction of steepest descent of the objective function.
- Newton's method uses the Hessian matrix (i.e., the local curvature) to adjust the parameters.
- Quasi-Newton methods approximate the Hessian matrix using a limited amount of information.



  - Some nice visualizations: https://www.benfrederickson.com/numerical-optimization/

Simulated Annealing

- Simulated annealing is a powerful global optimization method that is inspired by the process of annealing in metallurgy.

Simulated Annealing

- Simulated annealing is a powerful global optimization method that is inspired by the process of annealing in metallurgy.
- The process is governed by a temperature setting that determines the probability of accepting a step in the wrong direction.

Simulated Annealing

- Simulated annealing is a powerful global optimization method that is inspired by the process of annealing in metallurgy.
- The process is governed by a temperature setting that determines the probability of accepting a step in the wrong direction.
- A hot algorithm will move randomly, with little regard for the objective function value. A cold algorithm performs steepest ascent.

Simulated Annealing

- Simulated annealing is a powerful global optimization method that is inspired by the process of annealing in metallurgy.
- The process is governed by a temperature setting that determines the probability of accepting a step in the wrong direction.
- A hot algorithm will move randomly, with little regard for the objective function value. A cold algorithm performs steepest ascent.
- A slow cooling schedule allows the algorithm to explore many possible peaks before climbing to the top.

Simulated Annealing

Algorithm 2: Simulated Annealing Algorithm

Input: Initial guess s, initial temperature T; objective $\ell(\theta)$

Output: Best parameter set found during search

while stopping condition not met **do**

  Generate new guess s' by random perturbation to s;

  Calculate loss difference $\Delta\ell = \ell(s') - \ell(s)$;

  if $\Delta E < 0$ **then**

    Accept s' as the new current state s;

  else

    Accept s' with probability $\exp(-\Delta\ell/T)$;

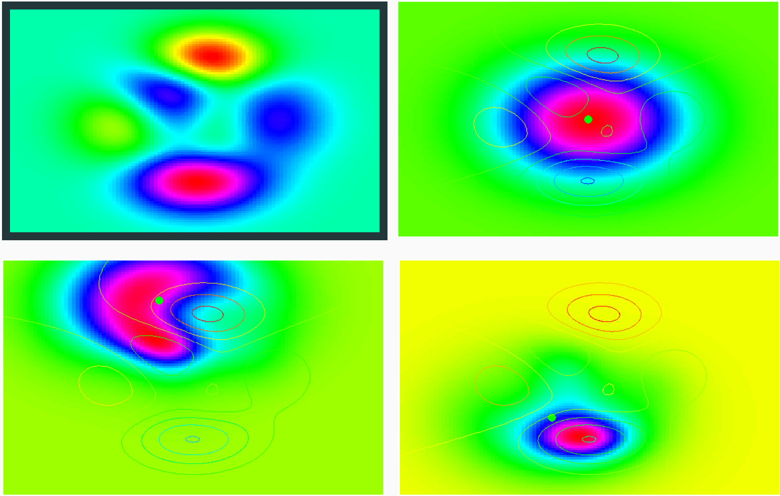
  end

  Decrease temperature T;

end

return Best state found during search

Simulated Annealing



Constrained vs. Unconstrained Optimization

- Two main types of optimization problems: constrained and unconstrained

Constrained vs. Unconstrained Optimization

- Two main types of optimization problems: constrained and unconstrained
- In an unconstrained optimization problem, the goal is to minimize (or maximize) a function subject to no constraints

Constrained vs. Unconstrained Optimization

- Two main types of optimization problems: constrained and unconstrained
- In an unconstrained optimization problem, the goal is to minimize (or maximize) a function subject to no constraints
- In a constrained optimization problem, the goal is to minimize (or maximize) a function subject to one or more constraints

Constrained vs. Unconstrained Optimization

- Two main types of optimization problems: constrained and unconstrained
- In an unconstrained optimization problem, the goal is to minimize (or maximize) a function subject to no constraints
- In a constrained optimization problem, the goal is to minimize (or maximize) a function subject to one or more constraints
- Constrained optimization often requires the use of specialized optimization algorithms and techniques to find the global minimum (or maximum)

Constrained vs. Unconstrained Optimization

Constraints can be that some linear transformation A of the parameters must be less than some constant b, and/or that some linear transformation A_{eq} of the parameters must be equal to some constant b_{eq}, and/or that they must fall in bounds (l_b, u_b), and/or that some are integers.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f^\top x \\ \text{subject to} \quad & \left\{ \begin{array}{c} Ax \le b \text{ (MATLAB)} \\ Ax \ge b \text{ (python)} \\ A_{eq}x = b_{eq} \\ l_b \le x \le u_b \\ x(\texttt{intflag}) \text{ are integers} \end{array} \right\} \end{aligned}$$

Unconstrained Example: Production Environment

- Imagine a company wants to optimize its production process by finding the optimal temperature and pressure settings for the production equipment.

Unconstrained Example: Production Environment

- Imagine a company wants to optimize its production process by finding the optimal temperature and pressure settings for the production equipment.
- The cost of production is given by:

$$\min_{x} \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

Unconstrained Example: Production Environment

- Imagine a company wants to optimize its production process by finding the optimal temperature and pressure settings for the production equipment.
- The cost of production is given by:

$$\min_{x} \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

- What are the optimal settings to run the production process?

Unconstrained Example: Parameter Estimation

- Assume we have some data $D \leftrightarrow \{5, -7, 3, 0, -11\}$

Unconstrained Example: Parameter Estimation

- Assume we have some data $D \leftrightarrow \{5, -7, 3, 0, -11\}$
- We want to describe D with a single number such that the squared distance to each data point is minimal:

$$\min_x \quad f(x) = \sum_{i=1}^{5} (x - d_i)^2$$

Unconstrained Example: Parameter Estimation

- Assume we have some data $D \leftrightarrow \{5, -7, 3, 0, -11\}$
- We want to describe D with a single number such that the squared distance to each data point is minimal:

$$\min_{x} \quad f(x) = \sum_{i=1}^{5} (x - d_i)^2$$

- What is the best x?

Unconstrained Example: Signal Detection Theory

- Assume we have some data $D \leftrightarrow \{\text{Hits} = 4, \text{Misses} = 1, \text{False alarms} = 2, \text{Correct rejections} = 3\}$

Unconstrained Example: Signal Detection Theory

- Assume we have some data $D \leftrightarrow \{\text{Hits} = 4, \text{Misses} = 1, \text{False alarms} = 2, \text{Correct rejections} = 3\}$
- Which $(d', c) \leftrightarrow x$ make D most likely:

$$(\max_x) \quad f(x) \quad = \quad \binom{5}{4} \theta_H^5 (1-\theta_H)^1 \times \binom{5}{2} \theta_F^2 (1-\theta_F)^3$$

$$\text{with} \begin{cases} \theta_H & = & 1 - \Phi\left(-\frac{1}{2}d' + c\right) \\ \theta_F & = & 1 - \Phi\left(\frac{1}{2}d' + c\right) \end{cases}$$

Unconstrained Example: Signal Detection Theory

- Assume we have some data $D \leftrightarrow \{\text{Hits} = 4, \text{Misses} = 1, \text{False alarms} = 2, \text{Correct rejections} = 3\}$
- Which $(d', c) \leftrightarrow x$ make D most likely:

$$
\begin{aligned}
(\max_x) \quad f(x) \quad &= \quad \binom{5}{4}\theta_H^5 (1-\theta_H)^1 \times \binom{5}{2}\theta_F^2 (1-\theta_F)^3 \\
&\qquad \text{with} \begin{cases} \theta_H &= \quad 1 - \Phi\left(-\frac{1}{2}d' + c\right) \\ \theta_F &= \quad 1 - \Phi\left(\frac{1}{2}d' + c\right) \end{cases}
\end{aligned}
$$

Unconstrained Example: Signal Detection Theory

- Assume we have some data $D \leftrightarrow \{\text{Hits} = 4, \text{Misses} = 1, \text{False alarms} = 2, \text{Correct rejections} = 3\}$
- Which $(d', c) \leftrightarrow x$ make D most likely:

$$
\begin{aligned}
(\max_x) \quad f(x) &= \binom{5}{4} \theta_H^5 (1 - \theta_H)^1 \times \binom{5}{2} \theta_F^2 (1 - \theta_F)^3 \\
&\quad \text{with} \begin{cases} \theta_H = 1 - \Phi\left(-\frac{1}{2} d' + c\right) \\ \theta_F = 1 - \Phi\left(\frac{1}{2} d' + c\right) \end{cases} \\
\Rightarrow (\min_x) \quad f(x) &= -\log\left[\theta_H^5 (1 - \theta_H)^1 \times \theta_F^2 (1 - \theta_F)^3\right] \\
&= -5 \log(\theta_H) - \log(1 - \theta_H) \\
&\quad - 2 \log(\theta_F) - 3 \log(1 - \theta_F)
\end{aligned}
$$

MATLAB

Unconstrained Optimization using `fminsearch`

$$\min_{x} \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

- Set up the problem as an anonymous function:

```
% Define the objective function
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
```

Unconstrained Optimization using `fminsearch`

$$\min_{x} \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

- Set up the problem as an anonymous function:

```
% Define the objective function
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
```

- Minimize using `fminsearch`:

```
% Solve the optimization problem
start = [0, 0];
x = fminsearch(fcn, start);
```

Unconstrained Optimization using `fminsearch`

$$\min_{x} \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

- Set up the problem as an anonymous function:

```
% Define the objective function
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
```

- Minimize using `fminsearch`:

```
% Solve the optimization problem
start = [0, 0];
x = fminsearch(fcn, start);
```

- The solution is $x_1 = 2$ and $x_2 = 0.5$.

Unconstrained Optimization using `fminunc`

```
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
x0 = [0.5, 1.5];
options = optimoptions('fminunc', 'Display', 'iter', ...
                       'Algorithm', 'quasi-newton', ...
                       'MaxIterations', 1000);
[x, fval, exitflag, output] = fminunc(fcn, x0, options);
```

- `Display=iter` shows optimization progress at each iteration

Unconstrained Optimization using fminunc

```
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
x0 = [0.5, 1.5];
options = optimoptions('fminunc', 'Display', 'iter', ...
                       'Algorithm', 'quasi-newton', ...
                       'MaxIterations', 1000);
[x, fval, exitflag, output] = fminunc(fcn, x0, options);
```

- `Display=iter` shows optimization progress at each iteration
- `Algorithm` specifies the optimization algorithm to use

Unconstrained Optimization using fminunc

```
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
x0 = [0.5, 1.5];
options = optimoptions('fminunc', 'Display', 'iter', ...
                       'Algorithm', 'quasi-newton', ...
                       'MaxIterations', 1000);
[x, fval, exitflag, output] = fminunc(fcn, x0, options);
```

- `Display=iter` shows optimization progress at each iteration
- `Algorithm` specifies the optimization algorithm to use
- `MaxIterations` sets the maximum number of iterations

Unconstrained Optimization using `fminunc`

```
fcn = @(x) exp( (x(1)-2).^2 + (x(2)-.5).^2);
x0 = [0.5, 1.5];
options = optimoptions('fminunc', 'Display', 'iter', ...
                       'Algorithm', 'quasi-newton', ...
                       'MaxIterations', 1000);
[x, fval, exitflag, output] = fminunc(fcn, x0, options);
```

- `Display=iter` shows optimization progress at each iteration
- `Algorithm` specifies the optimization algorithm to use
- `MaxIterations` sets the maximum number of iterations
- Output variables `x`, `fval`, `exitflag`, and `output` contain the optimization results and information

fminunc vs. fmincon

Using fminunc:

```
fcn = @(x)sum((x-[5, -7, 3, 0, -11]).^2);
x0 = 0.5;
[x, fval, exitflag, output] = fminunc(fcn, x0);
```

Enforce $x \geq 0$ with fmincon:

```
fcn = @(x)sum((x-[5, -7, 3, 0, -11]).^2);
x0 = 0.5;
[x, fval, exitflag, output] = fmincon(fcn, x0, -1, 0);
```

Note that $A = -1$ and $b = 0$, so we enforce that $-x \leq 0$

MATLAB Optimization Toolbox

MATLAB has a lot of numerical optimization features.

As of September 2022, the MATLAB Optimization Toolbox User's Guide is 1,584 pages long. It is an excellent review of modern numerical optimization methods.

Python

Unconstrained Optimization using `scipy.optimize.minimize`

$$\min_x \quad f(x) = e^{(x_1-2)^2+(x_2-0.5)^2}$$

```
import numpy as np
import scipy.optimize as optim

# Define the objective function
def fcn(x):
    return np.exp((x[0]-2)**2 + (x[1]-0.5)**2)

# Set the initial point and solve the optimization problem
start = [0, 0]
x = optim.minimize(fcn, start, method='Nelder-Mead')

# The solution is x1=2 and x2=0.5
print(f"The solution is x1={x.x[0]} and x2={x.x[1]}")
```

The solution is $x_1 = 2$ and $x_2 = 0.5$.

Unconstrained Optimization using `scipy.optimize.minimize`

```python
import numpy as np
import scipy.optimize as optim

# Define the objective function
def fcn(x):
    return np.exp((x[0]-2)**2 + (x[1]-0.5)**2)

# Set the initial point and solve the optimization problem
x0 = [0.5, 1.5]
options = {'disp': True, 'maxiter': 1000}
x = optim.minimize(fcn, x0, method='BFGS', options=options)
```

- `'disp'`:`True` shows optimization progress at each iteration

Unconstrained Optimization using `scipy.optimize.minimize`

```
import numpy as np
import scipy.optimize as optim

# Define the objective function
def fcn(x):
    return np.exp((x[0]-2)**2 + (x[1]-0.5)**2)

# Set the initial point and solve the optimization problem
x0 = [0.5, 1.5]
options = {'disp': True, 'maxiter': 1000}
x = optim.minimize(fcn, x0, method='BFGS', options=options)
```

- `'disp':True` shows optimization progress at each iteration
- `'maxiter'` sets the maximum number of iterations

Unconstrained Optimization using `scipy.optimize.minimize`

```
import numpy as np
import scipy.optimize as optim

# Define the objective function
def fcn(x):
    return np.exp((x[0]-2)**2 + (x[1]-0.5)**2)

# Set the initial point and solve the optimization problem
x0 = [0.5, 1.5]
options = {'disp': True, 'maxiter': 1000}
x = optim.minimize(fcn, x0, method='BFGS', options=options)
```

- `'disp':True` shows optimization progress at each iteration
- `'maxiter'` sets the maximum number of iterations
- `method='BFGS'` selects a Quasi-Newton optimization algorithm

Unconstrained vs. constrained

```
import numpy as np
import scipy.optimize as optim

# Define the objective function
def fcn(x):
    return np.sum((x - [5, -7, 3, 0, -11])**2)

# Set the initial point
x0 = 0.5
```

Unconstrained:

```
x = optim.minimize(fcn, x0)
```

Enforce $x \geq 0$ with `LinearConstraint`:

```
x = minimize(fcn, x0,
    constraints=optim.LinearConstraint(1, 0))
```

Note that $A = 1$ and $b = 0$, so we enforce that $x \geq 0$