ORF522 – Linear and Nonlinear Optimization

23. The role of optimization
In the lecture you mentioned "sampling" from the parameter space and get its label of strategy. Does this mean that every time you do this, you have to solve a strong branching problem? Is this how we get the so-called "expert labels" or the y's in our classification problem? This sounds like more work than solving the problem directly using strong branching?
Today’s lecture
The role of optimization

• Geometry of optimization problems
• Solving optimization problems
• What’s left out there?
• The role of optimization
Basic use of optimization

Optimal decisions

Variables  Objective  Constraints

Decisions

Mathematical language

The algorithm computes them for you
Most optimization problems cannot be solved
Geometry of optimization problems
Linear optimization

minimize \( c^T x \)
subject to \( Ax \leq b \)

Optimal point properties

- Extreme points are optimal
- Need to search only between extreme points
Nonlinear optimization

minimize \[ f(x) \]
subject to \[ x \in C \]

Optimal point properties

- Any feasible point could be optimal
- Can have many locally optimal points
Fermat’s optimality conditions

minimize \( f(x) \)
subject to \( x \in C \)

\[ \nabla f(x) = 0 \]
\[ \mathcal{N}(x) \subseteq \partial f(x) \]

Stationarity conditions

Differentiable \( f \) convex \( C \)

\[ 0 \in \partial f(x) + \mathcal{N}_C(x) \rightarrow -\nabla f(x) \in \mathcal{N}_C(x) \]

Properties

- Convex optimization (necessary and sufficient)
- Nonconvex optimization (necessary)
KKT optimality conditions

minimize \( f(x) \)

subject to \( g_i(x) \leq 0, \quad i = 1, \ldots, m \)

\[ \nabla f(x^*) + \sum_{i=1}^{m} y_i^* \nabla g_i(x^*) = 0 \quad \text{stationarity} \]

\[ y^* \geq 0 \quad \text{dual feasibility} \]

\[ g_i(x^*) \leq 0, \quad i = 1, \ldots, m \quad \text{primal feasibility} \]

\[ y_i^* g_i(x^*) = 0, \quad i = 1, \ldots, m \quad \text{complementary slackness} \]
KKT optimality conditions

minimize \( f(x) \)
subject to \( g_i(x) \leq 0, \quad i = 1, \ldots, m \)

\[
\nabla f(x^*) + \sum_{i=1}^{m} y_i^* \nabla g_i(x^*) = 0
\]

stationarity

\( y^* \geq 0 \)

\( g_i(x^*) \leq 0, \quad i = 1, \ldots, m \)

dual feasibility

primal feasibility

\( y_i^* g_i(x^*) = 0, \quad i = 1, \ldots, m \)

complementary slackness

Remarks

- Require Slater’s conditions or constraint qualifications (LICQ)
- Can be derived from Fermat’s optimality
- Necessary and sufficient for convex problems
- Only necessary for nonconvex problems
KKT optimality conditions

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \leq 0, \quad i = 1, \ldots, m \\
\n\end{align*}
\]

\[
\nabla f(x^*) + \sum_{i=1}^{m} y_i^* \nabla g_i(x^*) = 0
\]

- Stationarity
- Dual feasibility
- Primal feasibility
- Complementary slackness

Remarks
- Require Slater’s conditions or constraint qualifications (LICQ)
- Can be derived from Fermat’s optimality
- Necessary and sufficient for convex problems
- Only necessary for nonconvex problems

In practice
Search for KKT points
Certifying optimality

Dual function

\[ g(y) \]

Properties

- Lower bound: \( g(y) \leq f(x), \quad \forall x, y \)
- Always convex

Strong duality

\[ g(y^*) \leq f(x^*) \]

- Linear optimization (unless primal and dual infeasible)
- Convex optimization (if Slater’s condition holds)
Certifying optimality

**Dual function**

\[ g(y) \]

**Properties**

- Lower bound: \( g(y) \leq f(x), \quad \forall x, y \)
- Always convex

**Strong duality**

\[ g(y^*) \leq f(x^*) \]

- Linear optimization (unless primal and dual infeasible)
- Convex optimization (if Slater’s condition holds)

**Optimality gap**

- Convex optimization without strong duality
- Nonconvex optimization

**It works as a suboptimality certificate**

Solving optimization problems
Classical vs modern view

Classical view

- **Linear optimization**
  (zero curvature) is easy

- **Nonlinear optimization**
  (nonzero curvature) is hard
Classical vs modern view

Classical view

• Linear optimization (zero curvature) is easy
• Nonlinear optimization (nonzero curvature) is hard

Correct view

• Convex optimization (nonnegative curvature) is easy
• Nonconvex optimization (negative curvature) is hard
Classical vs modern view

**Classical view**
- Linear optimization (zero curvature) is easy
- **Nonlinear optimization** (nonzero curvature) is hard

**Correct view**
- Convex optimization (nonnegative curvature) is easy
- **Nonconvex optimization** (negative curvature) is hard

The classical view is wrong
Numerical linear algebra

The core of optimization algorithms is linear systems solution

\[ Ax = b \]

Direct method

1. Factor \( A = A_1 A_2 \ldots A_k \) in “simple” matrices \( O(n^3) \)
2. Compute \( x = A_k^{-1} \ldots A_1^{-1} b \) by solving \( k \) “easy” linear systems \( O(n^2) \)
Numerical linear algebra

The core of optimization algorithms is linear systems solution

\[ Ax = b \]

Direct method

1. Factor \( A = A_1 A_2 \ldots A_k \) in “simple” matrices \( O(n^3) \)
2. Compute \( x = A_k^{-1} \ldots A_1^{-1} b \) by solving \( k \) “easy” linear systems \( O(n^2) \)

Main benefit
factorization can be reused
with different right-hand sides \( b \)
Numerical linear algebra

The core of optimization algorithms is linear systems solution

\[ Ax = b \]

Direct method

1. Factor \( A = A_1 A_2 \ldots A_k \) in “simple” matrices \( O(n^3) \)
2. Compute \( x = A_k^{-1} \ldots A_1^{-1} b \) by solving \( k \) “easy” linear systems \( O(n^2) \)

Main benefit

factorization can be reused
with different right-hand sides \( b \)

You never invert \( A \)
Solving convex problems

Simplex methods

- Tailored to LPs
- Exponential worst-case performance
- Up to 10,000 variables

Cheap iterations (rank-1 updates)
Solving convex problems

**Simplex methods**
- Tailored to LPs
- Exponential worst-case performance
- Up to 10,000 variables

Cheap iterations
(rank-1 updates)

**Second-order methods**
(e.g., interior-point)
- Up to ~10,000 variables
- Polynomial worst-case complexity

Expensive iterations
(matrix factorizations)
Solving convex problems

**Simplex methods**
- Tailored to LPs
- Exponential worst-case performance
- Up to 10,000 variables

Cheap iterations (rank-1 updates)

**Second-order methods**
(e.g., interior-point)
- Up to ~10,000 variables
- Polynomial worst-case complexity

Expensive iterations (matrix factorizations)

**First-order methods**
- Up to 1B variables
- Several convergence rates

Cheap iterations (matrix prefactored)
Convex optimization solvers

Remarks

• **No babysitting/**initialization required

• Very **reliable** and **efficient**

• Can solve problems in **milliseconds** on embedded platforms

• **Simplex** and **interior-point** solvers are **almost a technology**

• **First-order** methods are more **sensitive to data scaling** but work in **huge dimensions**
First-order methods for large-scale convex optimization

- Gradient/subgradient method
- Forward-backward splitting (proximal algorithms)
- Accelerated forward-backward splitting
- Douglas-Rachford splitting (ADMM)
- Interior-point methods (not covered for convex)
First-order methods for large-scale convex optimization

- Gradient/subgradient method
- Forward-backward splitting (proximal algorithms)
- Accelerated forward-backward splitting
- Douglas-Rachford splitting (ADMM)
- Interior-point methods (not covered for convex)

Large-scale systems
- start with feasible method with cheapest per-iteration cost
- if too many iterations, transverse down the list
Methods for nonconvex optimization

Convex optimization algorithms: global and typically fast

Nonconvex optimization algorithms: must give up one, global or fast

- **Local methods: fast but not global**
  Need not find a global (or even feasible) solution. They cannot certify global optimality because KKT conditions are not sufficient.

- **Global methods: global but often slow**
  They find a global solution and certify it.
What’s left out there?
What we did not cover in nonlinear optimization

**Second-order methods:** High accuracy on small/medium-scale data
- Newton’s method
- Quasi-Newton (BFGS, L-BFGS)
- Interior-point methods for nonlinear optimization (IPOPT)
What we did not cover in nonlinear optimization

**Second-order methods:** High accuracy on small/medium-scale data

- Newton’s method
- Quasi-Newton (BFGS, L-BFGS)
- Interior-point methods for nonlinear optimization (IPOPT)

**Stochastic gradient methods**

- Stochastic gradient descent
- Variance reduction methods
- Deep learning optimizers

Covered in

- COS512/ELE539: Optimization for Machine Learning
- ELE522: Large-Scale Optimization for Data Science
What we did not cover in nonlinear optimization

**Second-order methods:** High accuracy on small/medium-scale data
- Newton’s method
- Quasi-Newton (BFGS, L-BFGS)
- Interior-point methods for nonlinear optimization (IPOPT)

**Stochastic gradient methods**
- Stochastic gradient descent
- Variance reduction methods
- Deep learning optimizers

**Optimization in data science**
- Compressed sensing
- Low-rank matrix recovery
- Many more…

Covered in
- COS512/ELE539: Optimization for Machine Learning
- ELE522: Large-Scale Optimization for Data Science
- ELE520: Mathematics of Data Science
What we did not cover in convex optimization?

More in details on convex analysis

Conic optimization

- Second-order cone programming
- Semidefinite programming
- Sum-of-squares optimization

Convex relaxations of NP-hard problems

Covered in ORF523: Convex and Conic Optimization
The role of optimization
Optimization as a surrogate for real goal

Very often, optimization is not the actual goal

The goal usually comes from practical implementation (new data, real dynamics, etc.)

Real goal is usually encoded (approximated) in cost/constraints
Optimization problems are just models

“All models are wrong, some are useful.”

— George Box
Optimization problems are just models

“All models are wrong, some are useful.”
— George Box

Implications

• Problem formulation does not need to be “accurate”
• Objective function and constraints “guide” the optimizer
• The model includes parameters to tune

We often do not need to solve most problems to extreme accuracy
Portfolio

Optimization problem
maximize \( \mu^T x - \gamma x^T \Sigma x \)
subject to \( 1^T x = 1 \)
\( x \geq 0 \)

Goal
Optimize backtesting performance
Portfolio

Optimization problem

maximize \( \mu^T x - \gamma x^T \Sigma x \)

subject to

\( 1^T x = 1 \)

\( x \geq 0 \)

Goal
Optimize backtesting performance

Uncertain returns

\( p_t \) random variable:

mean \( \mu \), covariance \( \Sigma \)

Backtesting performance
(sum over all past realizations)

- Total returns
- Cumulative risk (quadratic term)
Control

Optimization problem
(control policy)

\[ \phi(\bar{x}) = \min \sum_{t=0}^{T-1} \ell(x_t, u_t) \]

subject to

\[ x_{t+1} = f(x_t, u_t) \]

\[ x_0 = \bar{x} \]

\[ x_t \in \mathcal{X}, \quad u_t \in \mathcal{U} \]

Goal:
Optimize closed-loop performance

Real dynamics
\[ x_{t+1} = f(x_t, u_t, w_t) \]

Control input
\[ u_t = \phi(x_t) \]

Closed-loop performance
\[ J = \sum_{t=0}^{\infty} \ell(x_t, u_t) \]
Quadcopter control
Low accuracy works well

Quadcopter example
Linearized dynamics
\[ x_{t+1} = Ax_t + Bu_t + w_t \]
\[ x_t \in \mathbb{R}^{12}, \quad u_t \in \mathbb{R}^4 \]

Input and state constraints
\[ x_t \in [x, \bar{x}], \quad u_t \in [u, \bar{u}] \]
Quadcopter control
Low accuracy works well

Quadcopter example

Linearized dynamics

\[ x_{t+1} = Ax_t + Bu_t + w_t \]

\( x_t \in \mathbb{R}^{12}, \quad u_t \in \mathbb{R}^4 \)

Goal: track trajectory

minimize

\[ \sum_t \| x_t - x_t^{\text{des}} \|_2^2 + \gamma \| u_t \|_2^2 \]

Input and state constraints

\( x_t \in [x, \bar{x}], \quad u_t \in [u, \bar{u}] \)
Quadcopter control
Low accuracy works well

Quadcopter example
Linearized dynamics
\[ x_{t+1} = Ax_t + Bu_t + w_t \]
\[ x_t \in \mathbb{R}^{12}, \quad u_t \in \mathbb{R}^4 \]

Input and state constraints
\[ x_t \in [x, \bar{x}], \quad u_t \in [u, \bar{u}] \]

Goal: track trajectory
\[ \text{minimize } \sum_t \|x_t - x_t^{\text{des}}\|_2^2 + \gamma \|u_t\|_2^2 \]

Closed loop simulation
Simulated dynamics
\[ x_{t+1} = Ax_t + Bu_t + w_t \]

random variable (nonlinearities, disturbances, etc.)
Quadcopter control

Closed-loop behavior with OSQP solver

- Low accuracy: $\epsilon = 0.1$
- High accuracy: $\epsilon = 0.0004$

Altitude reference tracking
Model fitting

Training data

\[ \mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N \]

Optimization problem

\[ \text{minimize } f_{\text{train}}(w) = \sum_{(x_i,y_i) \in \mathcal{D}_{\text{train}}} \ell(y_i, h_w(x_i)) \]
Model fitting

Training data

\[ \mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N \]

Optimization problem

\[
\min_w \quad f_{\text{train}}(w) = \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \ell(y_i, h_w(x_i))
\]

Goal

Optimize test performance

Test data
(unknown)

\[ \mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^N \]

Test performance

\[
\sum_{(x_i, y_i) \in \mathcal{D}_{\text{test}}} \ell(y_i, h_w(x_i))
\]
Model fitting
Support vector machine (linear classification)

Given a set of points \( \{v_1, \ldots, v_N\} \) with binary labels \( s_i \in \{-1, 1\} \), find hyperplane that strictly separates the two classes

\[
\begin{align*}
 a^T v_i + b &> 0 \quad \text{if} \quad s_i = 1 \\
 a^T v_i + b &< 0 \quad \text{if} \quad s_i = -1
\end{align*}
\]

(homogeneous)

Equivalent to

\[
 s_i v_i^T x \geq 1 
\]

\[
 v_i = (v_i, 1) \\
 x = (a, b)
\]

minimize \( \sum_{i=1}^{N} \max\{0, 1 - s_i v_i^T x\} + \gamma/2 \|x\|_2^2 \)

quadratic term
(interpretation: maximum margin)
Consensus SVM

Operator splitting form

minimize
subject to

$$f = \sum_{i=1}^{N} \max\{0, 1 - s_i \nu_i^T x\} + \frac{\gamma}{2} \|z\|_2^2$$
$$x = z$$
Consensus SVM

Operator splitting form

minimize

subject to

\[ f \]

\[ g \]

split across workers \( j \) with samples \( D_j \)

Worker loss

\[ f_j(x) = \sum_{j \in D_j} \max\{0, 1 - s_j \nu_j^T x\} \]
Consensus SVM

Operator splitting form

\[
\begin{align*}
\text{minimize} & \quad f \left( \sum_{i=1}^{N} \max\{0, 1 - s_i \nu_i^T x\} + \gamma/2 \|z\|^2 \right) \\
\text{subject to} & \quad x = z
\end{align*}
\]

split across workers \(j\)
with samples \(D_j\)

Worker loss

\[
\begin{align*}
f_j(x) &= \sum_{j} \max\{0, 1 - s_j \nu_j^T x\}
\end{align*}
\]

Distributed model fitting ADMM

\[
\begin{align*}
x_j^{k+1} &= \text{prox}_{\lambda f_j}(z^k - u_j^k) \\
z^{k+1} &= \frac{N/\lambda}{1/\gamma + N/\lambda} (\bar{x}^{k+1} + \bar{u}^{k+1}) \\
u_j^{k+1} &= u_j^k + x_j^{k+1} - z^{k+1}
\end{align*}
\]

Local SVM \(\text{QP}\)

Averaging

Local update
Consensus SVM
Linear classification

Dashed lines are local workers’ hyperplanes

Optimal consensus hyperplane on test set after ~10 iterations
Conclusions

In ORF522, we learned to:

• **Model decision-making problems** across different disciplines as mathematical optimization problems.

• **Apply the most appropriate optimization tools** when faced with a concrete problem.

• **Implement** optimization algorithms and **prove** their convergence.

• **Understand** the limitations of optimization
Optimization cannot solve all our problems
It is just a mathematical model

But it can help us making better decisions

Thank you!

Bartolomeo Stellato