Numerical Optimization in DROP

v7.14 12 October 2025

Convex Optimization - Introduction and Overview

Motivation, Background, and Setup

1. Mathematical Formulation and Framework: Convex Optimization is an important class of constrained optimization problems that subsumes linear and quadratic programming. Such problems are specified in the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$x \in S$$

where the feasible set

$$S \subseteq \mathbb{R}^n$$

is a *convex* closed subset of $\mathbb{R}^n$ and f is a *convex function* (Hauser (2012)).

2. Local Minima as Global Minima: Convex optimization problems are important because all local minima are also guaranteed to be globally minimal, although this value may not be reached necessarily at a unique point.

Convex Sets and Convex Hull

1. Definition of a Convex Set: A convex set S satisfies the following property. For any two points x and y in S the point $(1-u)\cdot x + u\cdot y$ for

$$u \in [0,1]$$

lies in S as well. In other words, the line segment between any two points in S must also lie in S.
2. Definition of a Convex Hull: The smallest convex set containing another set A is called the *convex Hull* of $C(A)$.
3. Definition of a Convex Function: A function of a scalar field f is convex if it satisfies the following property:

$$f\big((1-u)\cdot x + u\cdot y\big) \leq (1-u)\cdot f(x) + u\cdot f(y)$$

for all x and y and any

$$u \in [0,1]$$

In other words, the function value between any two points x and y lies below the line of f connecting $f(x)$ and $f(y)$. An equivalent definition is that on the line segment between x and y the area over the graph of f forms a convex set.
4. Minima of a Convex Function: Convex functions are always somewhat "bowl shaped", have no local maxima (unless constant), and have no more than one local minimum value. If a local minimum exists, then it is also a global minimum. Hence gradient descent and Newton methods (with line search) are guaranteed to produce the global minimum when applied to such functions.

Properties of Convex Sets/Functions

1. Connection Property: Convex sets are connected.
2. Intersection Property: The intersection of any number of convex sets is also a convex set.
3. Axiomatic Convex Sets: The empty set, the whole space, any linear sub-spaces, and half-spaces are also convex.
4. Convex Set Dimensionality Reduction: If a convex set of S is of a lower dimension than its surrounding space $\mathbb{R}^n$ then S lies on a linear sub-space of $\mathbb{R}^n$.
5. Typical Convex Set Closure Properties: Some common convex functions include e^x, $-\log x$, x^k where k is an even number. Convex functions are closed under addition, the max operator, monotonic transformations of the x variable, and scaling by a non-negative constant.
6. Convex Set from Ellipsoidal Constraints: Constraints of the form

$$x^T A x \leq c$$

   produce a closed convex set (an ellipsoid) if A is positive semi-definite, and c is a non-negative number.
7. Transformation onto Semi-definite Programs: The set of positive semi-definite matrices is a closed convex set. Optimization problems with these constraints are known as *semi-definite programs* (SDPs). A surprising number of problems, including LPs and QPs, can be transformed into SDPs.

Convex Optimization Problems

1. General Form of Convex Optimization: An optimization problem in general

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$g_i(x) \leq 0, i = 1, \cdots, m$$

and

$$h_j(x) = 0, j = 1, \cdots, p$$

is convex as long as f is convex, all of the g_i for

$$i = 1, \cdots, m$$

are convex, and all of the h_j for

$$j = 1, \cdots, p$$

are linear.

2. Elimination of the Equality Constraints: In convex optimization we will typically eliminate the equalities before optimizing, either by converting them into two inequalities

$$h_j(x) \leq 0$$

and

$$-h_j(x) \leq 0$$

or by performing a null-space transformation. Hence the h_j's can be dropped from typical treatments.

References

- Hauser (2012): Convex Optimization and Interior Point Methods

Newton's Method in Optimization

Method

1. Iterative Root Finding vs Optimization: In calculus Newton's method is an iterative method for finding the roots of a differentiable function f, i.e., solutions to the equation

$$f(x) = 0$$

In optimization the Newton's method is applied to the derivative f' of a twice-differentiable function f to find the roots of the derivative, i.e., solutions to

$$f'(x) = 0$$

also known as the stationary points of f.

2. The Stationary Point of f: In the $1D$ problem the Newton's method attempts to construct a sequence x_n from an initial guess x_0 that converges towards some value x^* satisfying

$$f(x^*) = 0$$

This x^* is a stationary point of f (Newton's Method in Optimization (Wiki)).

3. Taylor Expansion of f around x_n: One wishes to find Δx such that $f(x_n + \Delta x)$ is a maximum. Thus, one seeks to solve the equation that sets the derivative of the last expression with respect to Δx equal to zero:

$$0 = \frac{\partial}{\partial \Delta x}\left\{f(x_n) + f'(x_n)\Delta x + \frac{1}{2}f''(x_n)[\Delta x]^2\right\} = f'(x_n) + f''(x_n)\Delta x$$

4. Conditions for approximating the Stationary Point: For the value of

$$\Delta x = -\frac{f'(x_n)}{f''(x_n)}$$

which is a solution to the above equation, typically

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

will be closer to the stationary point x^*. Provided that $f(x)$ is a twice-differentiable function and other technical conditions are satisfied, the sequence $x_1, \cdots, x_n$ will converge to the point x^* satisfying

$$f(x^*) = 0$$

5. Geometric Interpretation of the Newton's Method: The geometric interpretation of the Newton's method is that at each iteration one approximates $f(\vec{x})$ by a quadratic function $\vec{x}_n$ and then takes a step towards a maximum/minimum of that quadratic function. In higher dimensions, this stationary point may also be a saddle point. Of course, if $f(\vec{x})$ happens to *be* a quadratic function then the exact extremum is found in one step.

Higher Dimensions

1. Determination of the Iteration Increment: The above iteration scheme can be generalized to several dimensions by replacing the derivative with a gradient $\vec{\nabla} f(\vec{x})$

and the reciprocal of the second derivative with the inverse of the Hessian matrix $\mathbb{H}f(\vec{x})$. One thus obtains the iteration scheme

$$\vec{x}_{n+1} = \vec{x}_n + [\mathbb{H}f(\vec{x})]^{-1}\vec{\nabla}f(\vec{x}), n \geq 0$$

2. Adjustment to the Iteration Increment: Often Newton's method is modified to include a small step size

$$\gamma \in (0, 1)$$

instead of

$$\gamma = 1$$

to result in

$$\vec{x}_{n+1} = \vec{x}_n + \gamma[\mathbb{H}f(\vec{x})]^{-1}\vec{\nabla}f(\vec{x})$$

This is often done to ensure that the Wolfe conditions are satisfied at each step

$$\vec{x}_n \longrightarrow \vec{x}_{n+1}$$

of the iteration.

3. Newton's Method - Speed of Convergence: Where applicable Newton's method converges much faster towards the local maximum or minimum than gradient descent. In fact, every local minimum has a neighborhood $\mathbb{N}$ such that if one starts with

$$x_0 \in \mathbb{N}$$

Newton's method with step size

$$\gamma = 1$$

converges quadratically. The only requirements are that the Hessian be invertible, and it be a Lipschitz-continuous function of $\vec{x}$ in that neighborhood.

4. By-passing Explicit Hessian Computation: Finding the inverse of the Hessian in high dimensions can be an expensive exercise. In such cases, instead of directly inverting the Hessian it may be better to calculate the vector

$$\Delta\vec{x}_n = \vec{x}_{n+1} - \vec{x}_n$$

as a solution to the system of linear equations

$$[\mathbb{H}f(\vec{x}_n)]\vec{\nabla}f(\vec{x}_{n+1}) = -\vec{\nabla}f(\vec{x}_n)$$

This may be solved by various factorizations or approximately – and to great accuracy – using iterative methods.

5. Caveats of the Matrix Methods: Many of these methods are only applicable to certain types of equations – e.g., the Cholesky factorization and the conjugate gradient method will work only if $\mathbb{H}f(\vec{x}_n)$ is a positive definite matrix. While this may seem like a limitation, it is often a useful indicator of something gone wrong. For instance, if a maximization problem is being considered and $\mathbb{H}f(\vec{x}_n)$ is not positive definite, the iterations end up converging to a saddle point and not to a minimum.
6. Stable Variants of these Methods: On the other hand if constrained optimization is being considered – for example using Lagrange multipliers – the problem may become one of saddle point finding, in which case the Hessian will be symmetric indefinite and a solution for $\vec{x}_{n+1}$ needs to be done with approaches that will work for such situations, such as the LDL^T variant of the Cholesky factorization or the conjugate gradient method.

7. Techniques of Quasi-Newton Methods: There are exist various quasi-Newton methods where the approximation for the Hessian – or its direct inverse – is built up from the changes in the gradient.
8. Challenges with non-invertible Hessians: If a Hessian is close to a non-invertible matrix the inverted Hessian can be numerically unstable and the solution may diverge. In this case certain workarounds have been tried in the past, each of which have had varied success in differing setups.
9. Converting Hessian to Positive Definite: For example, one can modify the Hessian by adding a correction matrix B_n so as to make $\mathbb{H}f(\vec{x}_n) + B_n$ positive definite. On approach is to diagonalize $\mathbb{H}f(\vec{x}_n)$ and choose B_n so that $\mathbb{H}f(\vec{x}_n) + B_n$ has the same eigenvectors as $\mathbb{H}f(\vec{x}_n)$ but with each negative eigenvalue replaced by

$$\epsilon > 0$$

10. Approach used by Levenberg-Marquardt: An approach exploited by the Levenberg-Marquardt algorithm – which uses an approximate Hessian – is to add a scaled identity matrix $\mu\mathbb{I}$ to the Hessian, with the scale adjusted at every iteration as needed.
11. Comparison with Gradient Descent Approach: For large μ and small Hessian, the iterations behave like gradient descent with a step size $\frac{1}{\mu}$. This results in slower but more reliable convergence when the Hessian doesn't provide useful information.

Wolf Conditions

1. Motivation, Rationale, and Primary Purpose: In the unconstrained minimization problem setting, the **Wolf Conditions** are a set of inequalities for performing *inexact* line search, especially in quasi-Newton methods (Wolf (1969, 1971)).
2. Problem Setup - Function Extremization: In these methods the idea is to find

$$\min_{\vec{x}} f(\vec{x})$$

for some smooth

$$f: \mathbb{R}^n \longrightarrow \mathbb{R}$$

Each step often involves approximately solving the sub-problem

$$\min_{\alpha} f(\vec{x}_k + \alpha \vec{p}_k)$$

where $\vec{x}_k$ is the current best guess

$$\vec{p}_k \in \mathbb{R}^n$$

is a search direction, and

$$\alpha \in \mathbb{R}$$

is the step length.

3. Application of the Wolfe Conditions: The inexact line search provides an efficient way of computing an acceptable step length α that reduces the objective function sufficiently rather than minimizing the objective function over

$$\alpha \in \mathbb{R}^+$$

exactly. A line search algorithm can use the Wolfe conditions as a requirement for any guessed α before finding a new search direction $\vec{p}_k$.

Armijo Rule and Curvature Condition

1. The Inequalities of Wolfe Condition: A step length α_k is said to satisfy the *Wolfe conditions*, restricted to the direction $\vec{p}_k$, if the following inequalities hold:

$$f(\vec{x}_k + \alpha_k \vec{p}_k) \leq f(\vec{x}_k) + c_1 \alpha_k \vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)$$

$$\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k + \alpha_k \vec{p}_k) \geq c_2 \vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)$$

with

$$0 < c_1 < c_2 < 1$$

In examining the second condition once recalls that to ensure $\vec{p}_k$ is a descent direction one has

$$\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k) < 0$$

2. Choices for c_1 and c_2: c_1 is usually chosen quite small while c_2 is much larger. Nocedal and Wright (2006) provide example values of

$$c_1 = 10^{-4}$$

and

$$c_2 = 0.9$$

for Newton or quasi-Newton methods, and

$$c_2 = 0.1$$

for the non-linear conjugate gradient method. The first inequality is known as the **Armijo Rule** (Armijo (1966)) and ensures that the step length α_k decreases f sufficiently. The second inequality is called the **Curvature Condition**; it ensures that the slope has reduced sufficiently.

3. Strong Wolfe Conditions on Curvature: Denoting by ϕ a univariate function restricted to the direction $\vec{p}_k$ as

$$\phi(\alpha) = f(\vec{x}_k + \alpha_k \vec{p}_k)$$

there are situations where the Wolfe conditions can result in a value for the step length that is not close to a minimizer of ϕ. If one modifies the curvature condition to

$$\left|\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k + \alpha_k \vec{p}_k)\right| \leq c_2 \left|\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)\right|$$

then the curvature condition along with the Armijo rule form the so-called **strong Wolfe conditions**, and force α_k to lie close to a critical point of ϕ.

Rationale for the Wolfe Conditions

1. Convergence of the Gradient to Zero: The principal reason for imposing the Wolfe conditions in an optimization algorithm where

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$$

is to ensure the convergence of the gradient to zero. In particular if the cosine of the angle between $\vec{p}_k$ and the gradient

$$\cos\theta = \frac{\vec{\nabla} f(\vec{x}_k) \cdot \vec{p}_k}{\left\|\vec{\nabla} f(\vec{x}_k)\right\| \|\vec{p}_k\|}$$

is bounded away from zero, and Armijo rule and curvature conditions (modified) hold,

$$\vec{\nabla} f(\vec{x}_k) \longrightarrow 0$$

2. Positive Definiteness of Update Matrix: An additional motivation, in the case of a quasi-Newton method, is that if

$$\vec{p}_k = {B_k}^{-1} \vec{\nabla} f(\vec{x}_k)$$

where the matrix B_k is updated by the BFGS or the DFP formula, then if B_k is positive, the (modified) curvature condition implies that B_{k+1} is also positive definite.

References

- Armijo, L. (1966): Minimization of Functions having Lipschitz Continuous First Derivatives *Pacific Journal of Mathematics* **16 (1)** 1-3
- Nocedal, J., and S. Wright (2006): *Numerical Optimization 2nd Edition* **Springer**
- Wikipedia (2019): Newton's Method in Optimization
- Wikipedia (2019): Wolfe Conditions
- Wolfe, P. (1969): Convergence Conditions for Ascent Methods *SIAM Review* **11 (2)** 226-235
- Wolfe, P. (1971): Convergence Conditions for Ascent Methods II: Some Corrections *SIAM Review* **13 (2)** 185-188

Constrained Optimization

Constrained Optimization – Definition and Description

1. Constrained Optimization Definition – Variable Constraints: In mathematical optimization **constrained optimization** is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables.
2. Constrained Optimization Definition - Objective Function: The objective function is either a cost function or an energy function that is to be minimized, or a reward function or a utility function that has to be maximized.
3. Hard vs Soft Variable Constraints: Constraints can either be *hard constraints* that set conditions on variables required to be satisfied, or *soft constraints* that have some variable values that are penalized in the objective function if – and based on the extent that – the conditions on the variables are not satisfied (Constrained Optimization (Wiki)).

General Form

1. Constrained Optimization Problem – Mathematical Specification: A general constrained minimization problem may be written as follows:

$$\min_{x} f(x)$$

subject to

$$g_i(x) = c_i$$

for

$$i = 1, \cdots, n$$

equality constraints and

$$h_j(x) \geq d_j$$

for

$$j = 1, \cdots, m$$

inequality constraints where

$$g_i(x) = c_i$$

for

$$i = 1, \cdots, n$$

and

$$h_j(x) \geq d_j$$

for

$$j = 1, \cdots, m$$

are the constraints to be satisfied; these are called the hard constraints.

2. Constrained Optimizers - Soft Objective Function: In some problems, often called *constraint optimization problems*, the objective function is a sum of the cost functions, each of which penalized the extent if any to which a soft constraint – a constraint that is preferred but not required to be satisfied - is violated.

Solution Methods

1. Adaptation from Unconstrained Algorithms: Many unconstrained optimization algorithms can be adapted to the constrained case, often via the use of a penalty method. However, the search steps taken by the unconstrained method may be unacceptable for the constrained problem, leading to a lack of convergence. This is referred to as the Maratos effect (Sun and Yua (2010)).
2. Equality Constraints Lagrange Multiplier Technique: If the constrained problem has only equality constraints the method of Lagrange multipliers can be used to convert it to an unconstrained problem whose number of variables is the original number of variables plus the number of constraints.
3. Equality Constraints - Cross Variable Substitution: Alternatively, if the constraints are all equality constraints and linear, they can be solved for some of the variables in terms of the others, and the former can be substituted out of the objective function, leaving an unconstrained problem in a smaller number of variables.
4. Inequality Constraints - Conditions on Variables: With inequality constraints the problem can be characterized in terms of Geometric Optimality Conditions, Fritz-John Conditions, and Karush-Kuhn-Tucker Conditions in which simple problems may be directly solvable.
5. Linear Programming – Simplex/Interior Point: If the objective function and all of the hard constraints are linear, then the problem is a linear programming one. This can be solved by the Simplex method, which usually works in polynomial time in the

problem size but is not guaranteed to, or by interior point methods, which are guaranteed to work in polynomial time.

6. Quadratic Programming - Objective Function Constraints: If all the hard constraints are linear but the objective function is quadratic the problem is a quadratic programming problem.
7. Quadratic Programming Convex Objective Function: Quadratic Programming problems can be solved by the ellipsoid method in polynomial time if the objective function is convex; otherwise the problem is NP hard.

Constraint Optimization: Branch and Bound

1. Idea behind Branch and Bound: Constraint optimization can be solved by branch-and-bound algorithms. These are back-tracking algorithms that store the cost of the best solution found during execution and use it eventually to avoid part of the search.
2. Back Tracking vs. Solution Extension: More precisely whenever the algorithm encounters a partial solution that cannot be extended to form a solution with better cost than the best stored cost, the algorithm back tracks instead of trying to extend this solution.
3. Efficient of Back Tracking Algorithms: Assuming that the cost is to be minimized the efficiency of these algorithms depends on how the cost can be obtained from extending a partial solution is evaluated. Indeed, if the algorithms can back track from a partial solution, part of the search can be skipped. The lower the estimated cost the better the algorithm, as a lower estimated cost is more likely to be lower than the best cost of the solution found so far.
4. Algorithm Lower and Upper Bounds: On the other hand, this estimated cost cannot be lower than the effective cost obtained by extending the solution, as otherwise the algorithm could backtrack while a solution better than the best found so far exists. As a result, the algorithm requires an upper bound on the cost that can be obtained by extending a partial solution and this upper bound should be as small as possible.

5. Hansen's Branch-and-Bound Variation: A variation of the above method called Hansen's method uses interval methods (Leader (2004)). It inherently implements rectangular constraints.

Branch-and-Bound: First-Choice Bounding Conditions

1. Separately treating each Soft Constraint: One way of evaluating this upper bound for a partial solution is to consider each soft constraint separately. For each soft constraint the maximal possible value for any assignment to the unassigned variables is assumed. The sum of these variables is an upper bound because the soft constraints cannot assume a higher value.
2. Exact Nature of Upper Bound: It is exact because the maximal nature of soft constraints may derive from different evaluations: a soft constraint may be maximal for

$$x = a$$

while another soft constraint is maximal for

$$x = b$$

Branch-and-Bound Russian Doll Search

1. Branch-and-Bound Sub-problems: This method runs a branch-and-bound algorithm on n problems where n is the number of variables (Verfaillie, Lemaitre, and Schiex (1996)). Each such sub-problem is the sub-problem obtained by dropping the sequence $x_1, \cdots, x_i$ from the original problem along with the constraints containing them.

2. Sub-problem Cost Upper Bound: After the problem on variables $x_{i+1}, \cdots, x_n$ is solved its optimal cost can be used as an upper bound while solving the other problems.
3. Assigned/Unassigned Variables Sub-problem: In particular the cost of estimating a solution having $x_{i+1}, \cdots, x_n$ is added to the cost that derives from the evaluated variables. Virtually this corresponds to ignoring the evaluate variables and solving the problem on the unassigned ones, except that the latter problem has been already solved.
4. Sub-problem Total Cost Update: More precisely the cost of the constraints containing both the assigned and the unassigned variables is estimated as above, or using another arbitrary method; the cost of soft constraints using unassigned variables is instead estimated using the optimal solution of the corresponding problem, which is already known at this point.
5. Similarities with Dynamic Programming Approach: Like Dynamic Programming Russian Doll Search solves the sub-problems in order to solve the whole problem.
6. Differences with Dynamic Programming Approach: However, whereas Dynamic Programming directly combines the results obtained on the sub-problems to get the result of the whole program, the Russian Doll Search only uses them as bounds during the search.

Branch-and-Bound – Bucket Elimination

1. Elimination of a Specified Variable: The bucket elimination algorithm can be used for constraint optimization. A given variable can be removed from the problem by replacing all the soft constraints containing it with a new soft constraint.
2. Removed Variable Constraint-Cost Expression: The cost of the constraint expression is estimated assuming the maximal objective function value for each of the removed variable. Formally if x is the variable to be removed, $C_1, \cdots, C_n$ are the constraints containing it, and $y_1, \cdots, y_m$ are the variables containing x, the new soft constraint is defined by

$$C(y_1 = a_1, \cdots, y_n = a_n) = \max_a \sum_i C_i(x = a, y_1 = a_1, \cdots, y_n = a_n)$$

3. Bucket of all Variable Constraints: Bucket elimination works with an arbitrary ordering of the variables. Every variable is associated with a bucket of constraints; the bucket of a variable contains all the constraints having the variable has the highest in the order.
4. Order of the Bucket Elimination: Bucket elimination proceeds from the last variable to the first. For each variable, all constraints of the bucket are replaced as above to remove the variable. The resulting constraint is then placed in the appropriate bucket.

References

- Leader, J. J. (2004): *Numerical Analysis and Scientific Computation* **Addison Wesley**
- Sun, W., and Y. X. Yua (2010): *Optimization Theory and Methods: Non-linear Programming* **Springer**
- Verfaillie, G., M. Lemaitre, and T. Schiex (1996): Russian Doll Search for solving Constraint Optimization Problems *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference* **Portland** 181-187
- Wikipedia (2019): Constrained Optimization

Lagrange Multipliers

Motivation, Definition, and Problem Formulation

1. Purpose behind the Lagrange Methodology: In mathematical optimization the **method of Lagrange multipliers** (Lagrange (1811)) is a strategy for finding the local maxima and minima subject to equality constraints (Lagrange Multiplier (Wiki)).
2. Optimization Problem Mathematical Setup: Consider the optimization problem of maximizing $f(x, y)$ subject to

$$g(x, y) = 0$$

   A new variable λ is introduced (this is called the Lagrange multiplier) and the Lagrange function (or Lagrangian) defined by

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

   is studied. Here the term λ may be either added or subtracted.
3. Lagrange Multipliers as Stationary Points: If $f(x_0, y_0)$ is the maximum of $f(x, y)$ for the original constrained problem, then there exists a λ_0 such that (x_0, y_0, λ_0) is a stationary point for the Lagrangian function (stationary points are those where the partial derivatives of $\mathcal{L}$ are zero).
4. Necessary Conditions for Constrained Optimality: However not all stationary points yield a solution to the original problem. Thus, the method of Lagrange multipliers yields necessary conditions for optimality in constrained problems (Hiriart-Urruty and Lemarechal (1993), Bertsekas (1999), Vapnyarskii (2001), Lemarechal (2001), Lasdon (2002)).

5. Sufficient Conditions for Constrained Optimality: Sufficient conditions for a maximum or a minimum also exist. Sufficient conditions for a constrained local maximum or a minimum can be stated in terms of a sequence of principal minors, i.e., determinants of the upper-left-justified sub-matrices, of the bordered Hessian matrix of the second derivatives of the Lagrangian expression (Chiang (1984)).

Introduction, Background, and Overview

1. Advantage of the Lagrange Multiplier Formulation: One of the most common problems in calculus is that of finding the extrema of a function, but it is often difficult to find a closed form for the function being extremized. Such difficulties often arise when one wishes to extremize the function subject to fixed outside constraints or conditions. The method of Lagrange multipliers is a powerful tool for solving this class of problems without the need to explicitly solve for the conditions and use them to eliminate the extra variables.
2. Intuition behind the Lagrange Multiplier Methodology: Consider the 2D problem introduced above: maximize $f(x, y)$ subject to

$$g(x, y) = 0$$

The method of Lagrange multipliers relies on the intuition that at the maximum $f(x, y)$ cannot be increasing in the direction of any neighboring point where

$$g(x, y) = 0$$

If it did, one could walk along

$$g(x, y) = 0$$

to get higher, meaning that the starting point wasn't actually the maximum.

3. Function Realization/Constraint Value Contours: The contours of f given by

$$f(x,y) = d$$

for various values of d as well as the contours of g given by

$$g(x,y) = 0$$

may be visualized as being parallel/tangential to each other.

4. Invariance of $f(x,y)$ along the Constraint: Suppose one walks along the contour with

$$g(x,y) = 0$$

One is interested in funding points along $f(x,y)$ that do not change along the walk since these points may be maxima. There are two ways this can happen. First one could be walking along the contour line of $f(x,y)$ since by definition $f(x,y)$ does not change along the contour line. This would mean that the contour lines of $f(x,y)$ and $g(x,y)$ are parallel here. The second possibility is that one has reached a "level" part of $f(x,y)$ meaning that $f(x,y)$ cannot change in any direction.

5. Constraint/Objective Function Gradient Scaling: To check for the first possibility, one notices that the gradient of a function is perpendicular to its contour lines, and the contour lines of $f(x,y)$ and $g(x,y)$ are parallel if and only if the gradients of $f(x,y)$ and $g(x,y)$ are parallel. Thus, one wants points (x,y) where

$$g(x,y) = 0$$

and

$$\vec{\nabla}_{x,y} f(x,y) = -\lambda \vec{\nabla}_{x,y} g(x,y)$$

for some λ where

$$\vec{\nabla}_{x,y} f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\vec{\nabla}_{x,y} g(x,y) = \left[\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right]$$

are the respective gradients. The constant λ is required because although the two gradient vectors are parallel, the magnitudes of the gradient vectors are generally not equal. This constant is called the Lagrange multiplier (the negative sign is a convention).

6. The "Level" Plateau of f: This method also addresses the second possibility; if f is level then its gradient is zero, and setting

$$\lambda = 0$$

is a solution regardless of g.

7. Formulation of the Lagrangian Function: To incorporate these conditions into one equation one introduces an auxiliary function

$$\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda g(x,y)$$

and solves for

$$\vec{\nabla}_{x,y,\lambda} \mathcal{L}(x,y,\lambda) = 0$$

This is the method of Lagrange multipliers. Note that

$$\vec{\nabla}_{\lambda}\mathcal{L}(x,y,\lambda)=0$$

implies

$$g(x,y)=0$$

To summarize

$$\vec{\nabla}_{x,y,\lambda}\mathcal{L}(x,y,\lambda)=0$$

results in

$$\vec{\nabla}_{x,y}f(x,y)=-\lambda\vec{\nabla}_{x,y}g(x,y)$$

and

$$g(x,y)=0$$

The constrained extrema of f are the *critical points* but they are not necessarily the *local extrema* of $\mathcal{L}$.

8. Alternate Formulations for the Lagrangian: One may re-formulate the Lagrangian as a Hamiltonian, in which case the solutions are the local minima for the Hamiltonian. This is done in optimal control theory in the form of Pontryagin's minimum principle.
9. Non-Extremum Solutions to Lagrangian: The fact that the solutions to the Lagrangian are not necessarily the extrema also poses difficulties for numerical optimization. This can be addressed by computing the *magnitude* of the gradient, as the zeroes of the magnitude are necessarily the local minima, as illustrated later in the part on numerical optimization.

Handling Multiple Constraints

1. Notation Used for Multiple Constraints: Throughout this section the independent variables are denoted $x_1, \cdots, x_N$ and, as a group denoted as

$$p = (x_1, \cdots, x_N)$$

Also, the function being analyzed will be denoted $f(p)$ and the constraints will be represented by the equations

$$g_1(p) = 0$$
$$\vdots$$
$$g_M(p) = 0$$

2. Basic Idea behind the Constraints: The basic idea remains essentially the same: if we consider only satisfy the constraints (i.e., are *in* the constraints) then a point $\big(p, f(p)\big)$ is a stationary point (i.e., a point in a *flat* region) of f if and only if the constraints at that point do not allow movement in a direction where f changes value. Once the stationary points are located further tests are needed to see if it is a minimum, a maximum, or just a stationary point that is neither.
3. Multi-dimensional Invariance of f: Consider the level set of f at $\big(p, f(p)\big)$. Let the set of vectors $\{v_L\}$ contain the directions in which one can move and still remain in the same level set, the directions where the value of f does not change (i.e., the change equals zero). For every vector v in $\{v_L\}$ the following relation must hold:

$$\frac{\partial f}{\partial x_1} v_1 + \cdots + \frac{\partial f}{\partial x_N} v_N = 0$$

where the notation v_k above refers to the k^{th} component of the vector v.

4. Invocation of the Gradient of f: The equation above can be re-written in a more compact geometric form that helps the intuition:

$$\left[\frac{\partial f}{\partial x_1} \quad \cdots \quad \frac{\partial f}{\partial x_N}\right] \cdot \begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} = 0$$

i.e.

$$\vec{\nabla} f^T \cdot \vec{v} = 0$$

This makes it clear that if one is at p then *all* directions from this point that do *not* change the value of f *must be perpendicular to* $\vec{\nabla} f(p)$ (the gradient of f at p).

5. Usage of the Constraint Gradient: Each constraint limits the directions that one can move from a particular point and still satisfy the constraint. One uses the same set of procedures above to look for a set of vectors $\{v_C\}$ that contain the directions in which can move and still satisfy the constraint. As above, for every vector v in $\{v_C\}$ the following relation must hold:

$$\frac{\partial g}{\partial x_1} v_1 + \cdots + \frac{\partial g}{\partial x_N} v_N = 0$$

i.e.

$$\vec{\nabla} g^T \cdot \vec{v} = 0$$

From this it can be seen that at point p all directions from this point that will satisfy the constraint must be proportional to $\vec{\nabla} g(p)$.

6. Lagrange Multiplier Method Formal Definition: A point on f is a constrained stationary point if and only if the direction that changes f violates at least one of the constraints, i.e., it has no "component" in the legal space perpendicular to $\vec{\nabla} g(p)$.

Mathematically this means that the gradient of f at this constrained stationary point is perpendicular to the space spanned by the set of vectors $\{v_C\}$ which in turn is perpendicular to the gradients of the constraints g.

7. Single Constraint f/g Gradients: For a single constraint the above statement says that at stationary points the direction that changes f is the same as that violates the constraint. To determine if two vectors are in the same direction, note that if two vectors start from the same point then open vector can always reach the other by changing its length and/or flipping or the opposite way along the same direction line. This requires that

$$\vec{\nabla} f(p) = \lambda \vec{\nabla} g(p)$$

implying that

$$\vec{\nabla} f(p) - \lambda \vec{\nabla} g(p) = 0$$

8. Single Constraint Case - Compact Formulation: On adding another simultaneous equation to guarantee that this test is performed only at the point that satisfies the constraint two simultaneous equations result, which when solved identify all the constrained stationary points.

$$g(p) = 0$$

implies that p satisfies the constraint, thus

$$\vec{\nabla} f(p) - \lambda \vec{\nabla} g(p) = 0$$

is a stationary point.

9. Single Constraint Case Expanded Formulation: Fully expanded, there are simultaneous equations that need to be solved for $N + 1$ variables, which are $x_1, \cdots, x_N$ and λ:

$$g(x_1, \cdots, x_N) = 0$$

$$\frac{\partial f(x_1, \cdots, x_N)}{\partial x_1} - \lambda \frac{\partial g(x_1, \cdots, x_N)}{\partial x_1} = 0$$
$$\vdots$$
$$\frac{\partial f(x_1, \cdots, x_N)}{\partial x_N} - \lambda \frac{\partial g(x_1, \cdots, x_N)}{\partial x_N} = 0$$

10. Multiple Constraints and their Gradients: For more than one constraint a similar reasoning applies. Each gradient function g has a space of allowable directions at p – the space of vectors perpendicular to $\vec{\nabla} g(p)$. The set of directions allowed by all constraints is thus the space of directions perpendicular to all of the constraint gradients. Denoting the space of allowable moves by A and the span of gradients by S, using the discussion above

$$A = S^{\perp}$$

the space of vectors perpendicular to every element in S.

11. Multiple Constraints Case Gradient Formulation: If p is an optimum any element not perpendicular to $\vec{\nabla} f(p)$ is not an allowable direction. One can show that this implies

$$\vec{\nabla} f(p) \in A^{\perp} = S$$

Thus, there are scalars $\lambda_1, \cdots, \lambda_M$ such that

$$\vec{\nabla} f(p) = \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p)$$

implies

$$\vec{\nabla} f(p) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p) = 0$$

As before the simultaneous equations are added to guarantee that this test is performed only at a point that satisfies every constraint, and the resulting equations, when satisfied, identify all the constrained stationary points.

$$g_1(p) = \cdots = g_M(p) = 0$$

implies that p satisfies all constraints; further

$$\vec{\nabla} f(p) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p) = 0$$

indicates that p is a stationary point.

12. Multiple Gradient Case Lagrangian Formulation: The method is complete now from the standpoint of finding the stationary points, but these equations can be condensed into a more succinct and elegant form. The equations above look like partial derivatives of a larger scalar function $\mathcal{L}$ that takes all the $x_1, \cdots, x_N$ and all the $\lambda_1, \cdots, \lambda_M$ as inputs. Setting every equation to zero is exactly what one would have to do to solve for the *unconstrained* stationary points of the larger function. Finally, the larger function $\mathcal{L}$ with partial derivatives that are exactly the ones that we require can be constructed very simply as

$$\mathcal{L}(x_1, \cdots, x_N, \lambda_1, \cdots, \lambda_M) = f(x_1, \cdots, x_N) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(x_1, \cdots, x_N)$$

Solving the above equation for its *unconstrained* stationary points generates exactly the same stationary points as solving for the *constrained* stationary points of f under the constraints $g_1, \cdots, g_M$

13. The Method of Lagrange Multipliers: In Lagrange's honor the above function is called a *Lagrangian*, the scalar multipliers $\lambda_1, \cdots, \lambda_M$ are called *Lagrange Multipliers*, and the method itself is called *The Method of Lagrange Multipliers*.
14. The Karush-Kuhn-Tucker Generalization: The method of Lagrange multipliers is generalized by the Karush-Kuhn-Tucker conditions, which also takes into account inequality constraints of the form

$$h(\vec{x}) \leq c$$

Modern Formulation via Differentiable Manifolds

1. Local Extrema of $\mathbb{R}^d \to \mathbb{R}^1$: Funding the local maxima of a function

$$f: \mathbb{U} \to \mathbb{R}$$

where $\mathbb{U}$ is an open subset of $\mathbb{R}^d$ is done by finding all points $x \in \mathbb{U}$ such that

$$\mathbb{D}_x f = 0$$

and then checking whether all of the eigenvalues of the Hessian $\mathbb{H}_x f$ are negative. Setting

$$\mathbb{D}_x f = 0$$

is a non-linear problem and, in general, arbitrarily difficult. After funding the critical points checking for the eigenvalues is a linear problem and thus easy.

2. Extrema under the Level-Set Constraint: When

$$g: \mathbb{R}^d \to \mathbb{R}^1$$

is a smooth function such that

$$\mathbb{D}_x g \neq 0$$

for all x in the level set

$$g(x) = c$$

then $g^{-1}(c)$ becomes an $n-1$ dimensional smooth manifold $\mathbb{M}$ by the level set theorem. Funding local maxima is by definition a local problem, so it can be done on the local charts of $\mathbb{M}$ after funding a diffeomorphism

$$\varphi: \mathbb{V} \to \mathbb{R}^{n-1}$$

from an open subset of

$$\mathbb{V} \subset \mathbb{M}$$

onto an open subset

$$\mathbb{U} \subset \mathbb{R}^{n-1}$$

and thus, one can apply the algorithm in the previous part to the function

$$f' = f \circ \varphi^{-1} \colon \mathbb{U} \to \mathbb{R}$$

3. Approach of Lagrange Multiplier Method: While the above idea sounds good it is difficult to compute φ^{-1} in practice. *The entire method of Lagrange multipliers reduces to skipping that step and finding the zeroes of* $\mathbb{D}_x f'$ *directly.* It follows from the construction in the level set theorem that $\mathbb{D}_x \varphi^{-1}$ is the inclusion map

$$\ker \mathbb{D}_{\varphi^{-1}(x)} g \subseteq \mathbb{R}^n$$

Therefore

$$0 = \mathbb{D}_x f' = \mathbb{D}_x (f \circ \varphi^{-1}) = \mathbb{D}_{\varphi^{-1}(x)} f \circ \mathbb{D}_x \varphi^{-1}$$

if and only if

$$\ker \mathbb{D}_y g \subseteq \ker \mathbb{D}_y f$$

from writing y for $\varphi^{-1}(x)$

4. Existence of the Linear Map: By the first isomorphism theorem this is true if and only if there exists a linear map

$$\mathcal{L} \colon \mathbb{R} \to \mathbb{R}$$

such that

$$\mathcal{L} \circ \mathbb{D}_y g = \mathbb{D}_y f$$

As a linear map one must have that

$$\mathcal{L}(x) = \lambda x$$

for a fixed

$$\lambda \in \mathbb{R}$$

Therefore, funding the critical point of f' is equivalent to solving the system of equations

$$\lambda \mathbb{D}_y g = \mathbb{D}_y f$$

$$g(y) = c$$

in the variables

$$y \in \mathbb{R}^{n-1}$$

and

$$\lambda \in \mathbb{R}$$

This is in general a non-linear system of n equations and n unknowns.

5. Multiple Constraints Surjective Linear Map: In the case of general constraints one works with

$$g: \mathbb{R}^n \to \mathbb{R}^m$$

and replaces the condition

$$\mathbb{D}_x g \neq 0$$

for all

$$x \in g^{-1}(c)$$

with the requirement that $\mathbb{D}_x g$ be surjective at all such points. In this case $\mathcal{L}$ will be a linear map $\mathbb{R}^m \to \mathbb{R}$ i.e., a row vector with m entries.

Interpretation of the Lagrange Multipliers

1. As a Rate of Change Quantity: Often the Lagrange multipliers have an interpretation as some quantity of interest. For example, if the Lagrangian expression is

$$\begin{aligned}\mathcal{L}(x_1, \cdots, x_N, \lambda_1, \cdots, \lambda_M) \\ &= f(x_1, \cdots, x_N) + \lambda_1[c_1 - g_1(x_1, \cdots, x_N)] + \cdots \\ &+ \lambda_M[c_M - g(x_1, \cdots, x_N)]\end{aligned}$$

then

$$\frac{\partial \mathcal{L}}{\partial c_k} = \lambda_k$$

So λ_k is the rate of change of the quantity being optimized as a function of the constraint variable.

2. Examples of Lagrange Multiplier Roles: As examples, in Lagrangian mechanics the equations of motion are derived by finding stationary points of action, i.e., the time integral of the difference between the potential and the kinetic energies. Thus, the force on a particle due to a scalar potential

$$\vec{F} = -\vec{\nabla}V$$

can be interpreted as a Lagrange multiplier determining the change in action – transfer of potential to kinetic energy – following a variation in the particle's constrained trajectory. In control theory this formulated instead as co-state equations.

3. Marginal Effect of the Constraint: Moreover, by the control theorem the optimal value of a Lagrange multiplier has an interpretation as the marginal effect of the corresponding constraint constant upon on the optimal attainable value of the original objective function. Denoting the optimum with an asterisk it can be shown that

$$\frac{\partial f\left(x_1{}^*(c_1,\cdots,c_M),\cdots,x_N{}^*(c_1,\cdots,c_M)\right)}{\partial c_k} = \lambda_k{}^*$$

4. Lagrange Multiplier Usage in Economics: For example, in economics the optimal profit to a player is calculated subject to a constrained space of actions, where the Lagrange multiplier is the change in the optimal value of the objective function (profit) due to the relaxation of a given constraint – e.g., through a change in the income. In such a context λ^* is the marginal cost of the constraint, and is referred to as the shadow price.

Lagrange Application: Maximal Information Entropy

1. Information Entropy Maximization – Problem Setup: Suppose on wishes to find the discrete probability distribution on the points $\{p_1,\cdots,p_n\}$ with maximal information entropy. This is the same as saying that one wishes to find the least biased probability on the points $\{p_1,\cdots,p_n\}$. In other words, one wishes to maximize the Shannon entropy equation

$$f(p_1,\cdots,p_n) = -\sum_{j=1}^{n} p_j \log p_j$$

2. Cumulative Discrete Probability Normalization Constraint: For this to be a probability distribution the sum of the probabilities p_j at each point x_i must equal 1, so the constraint becomes

$$g(p_1,\cdots,p_n) = \sum_{j=1}^{n} p_j \log p_j$$

3. Application of the Lagrange Multipliers: Using Lagrange multipliers to find the point of maximum entropy $\vec{p}^*$ across all discrete probability distributions $\vec{p}$ on $\{x_1,\cdots,x_n\}$ requires that

$$\frac{\partial}{\partial \vec{p}}[f + \lambda(g-1)]\Big|_{\vec{p}=\vec{p}^*} = 0$$

which gives a system of n equations with indices

$$k = 1,\cdots,n$$

such that

$$\frac{\partial}{\partial p_k}\left[-\sum_{j=1}^{n} p_j \log p_j + \lambda\left(\sum_{j=1}^{n} p_j - 1\right)\right]\Bigg|_{\vec{p}=\vec{p}^*} = 0$$

4. Solution to the Discrete Probability: Carrying out the differentiation on these n equations one gets

$$-[1 + \log p_k{}^*] + \lambda = 0$$

This shows that all $p_k{}^*$ are the same because they depend only on λ. By using the constraint

$$\sum_{j=1}^{n} p_j = 1$$

one finds that

$$p_k{}^* = \frac{1}{n}$$

Hence the uniform distribution is the distribution with the greatest entropy, using distributions on n points.

Lagrange Application: Numerical Optimization Techniques

1. Local Minima vs. Saddle Points: The critical points of the Lagrangian occur at saddle points rather than the local minima or maxima (Heath (2005)). Unfortunately, many numerical optimization techniques such as hill climbing, gradient descent, some of the quasi-Newton methods, among others, are designed to find local minima (or maxima) and not the saddle points.
2. Conversion to an Extremization Problem: For this reason one must modify the formulation to ensure that this is a minimization problem – for example by extremizing the square of the gradient of the Lagrangian – or use an optimization technique that finds stationary points and not necessarily extrema, e.g., such as Newton's method without an extremum seeking line search.

3. "Saddle" Critical Points - An Example: As a simple example consider the problem of finding the value of x that minimizes the function

$$f(x) = x^2$$

constrained such that

$$x^2 = 1$$

Clearly this problem is pathological because there are only two values that satisfy the constraint, but it is useful for illustration purposes because the corresponding unconstrained function can be visualized in three dimensions.

4. Application of the Lagrange Multipliers: Using Lagrange multipliers this problem can be converted into an unconstrained optimization problem

$$\mathcal{L}(x, \lambda) = x^2 + \lambda(x^2 - 1)$$

The two critical points occur at the saddle points where

$$x = +1$$

and

$$x = -1$$

5. Transformation to Local Extremum Problem: In order to solve this problem with a numerical optimization technique one must first transform this problem such that the critical points occur at the local minima. This is done by computing the magnitude of the gradient of the unconstrained optimization problem.

6. Objective Variate/Constraint Multiplier Jacobian: First one computes the partial derivative of the unconstrained problem with respect to each variable;

$$\frac{\partial \mathcal{L}}{\partial x} = 2x + 2\lambda x$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = x^2 - 1$$

If the target function is not easily differentiable the differential with respect to each variable can be approximated using the divided differences technique, i.e.

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial x} \approx \frac{\mathcal{L}(x+\epsilon,\lambda) - \mathcal{L}(x-\epsilon,\lambda)}{2\epsilon}$$

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial \lambda} \approx \frac{\mathcal{L}(x,\lambda+\epsilon) - \mathcal{L}(x,\lambda-\epsilon)}{2\epsilon}$$

where ϵ is a small value.

7. Computing the Magnitude of the Gradient: Next one computes the magnitude of the gradient, which is the square root of the sum of squares of the partial derivatives:

$$h(x,\lambda) = \sqrt{(2x + 2\lambda x)^2 + (x^2 - 1)^2}$$

$$\approx \sqrt{\left[\frac{\mathcal{L}(x+\epsilon,\lambda) - \mathcal{L}(x-\epsilon,\lambda)}{2\epsilon}\right]^2 + \left[\frac{\mathcal{L}(x,\lambda+\epsilon) - \mathcal{L}(x,\lambda-\epsilon)}{2\epsilon}\right]^2}$$

Since the magnitude is always non-negative optimizing over the squared magnitude is equivalent to optimizing over the magnitude. Thus, the square root may be omitted from these equations with no expected differences in the results of the optimization.

8. Critical Points as Local Extrema: The critical points of h occur at

$$x = +1$$

and

$$x = -1$$

just as in $\mathcal{L}$. However, the critical points in h occur at local minima, so the numerical optimization techniques can be used to find them.

Lagrange Multipliers – Common Practice Applications

1. Lagrange Multipliers Applied in Economics: Constrained optimization plays a central role in economics. For example, the choice problem for a customer is represented as one of maximizing a utility function subject to a budget constraint. The Lagrange multiplier has an interpretation as the shadow price associated with that constraint – in this instance the marginal utility of the income. Other examples include profit maximization for a firm, along with various macro-economic applications.
2. Lagrange Multipliers in Control Theory: In optimal control theory the Lagrange multipliers are represented as co-state variables and are re-formulated as the minimization of the Hamiltonian, e.g., they are the basis behind the Pontryagin's minimum principle.
3. Lagrange Multipliers in Non-linear Programming: The Lagrange multiplier method has several generalizations. In non-linear programming there are several multiplier rules, e.g., the Caratheodery-John Multiplier Rule and the Convex Multiplier Rule for inequality constraints (Pourciau (1980)).

References

- Bertsekas, D. P. (1999): *Nonlinear Programming 2nd Edition* **Athena Scientific** Cambridge MA
- Chiang, A. C. (1984): *Fundamental Methods of Mathematical Economics 3rd Edition* **McGraw-Hill**
- Heath, M. T. (2005): *Scientific Computing – An Introductory Survey* **McGraw-Hill**
- Hiriart-Urruty, J. B., and C. Lemarechal (1993): XII Abstract Duality for Practitioners, in: *Convex Analysis and Minimization Algorithms, Volume II: Advanced Theory and Bundle Methods* **Springer-Verlag** Berlin
- Lagrange, J. L. (1811): *Mecanique Analytique*
- Lasdon, L. S. (2002): *Optimization Theory for Large Systems* **Dover Publications** Mineola NY
- Lemarechal, C. (1993): Lagrangian Relaxation, in: *Computational Combinatorial Optimization: Papers from the Spring School held in Schloss Dagstuhl (Editors: M. Junger and D. Naddef)* **Springer-Verlag** Berlin
- Pourciau, B. H. (1980): Modern Multiplier Rules *American Mathematical Monthly* **87 (6)** 433-452
- Vapnyarskii, I. B. (2001): Lagrange Multiplier, in: *Encyclopedia of Mathematics (editor: M. Hazewinkel)* **Springer**
- Wikipedia (2019): Lagrange Multiplier

Spline Optimizer

Constrained Optimization using Lagrangian

1. Base Set up: Use the Lagrangian objective function to optimize a multi-variate function $L(x, y)$ to incorporate the constraint

$$g(x, y) = c$$

as

$$\Lambda(x, y, z) = L(x, y) + \lambda[g(x, y) - c]$$

Here λ is called the Lagrange multiplier, and use one Lagrange multiplier per constraint.

2. Optimize (Maximize/Minimize) the Lagrangian: Optimize (i.e., maximize/minimize) the Lagrangian with respect to x, y, and λ – thus

$$\frac{\partial \Lambda(x, y, z)}{\partial x} = 0$$

$$\frac{\partial \Lambda(x, y, z)}{\partial y} = 0$$

and

$$\frac{\partial \Lambda(x, y, z)}{\partial \lambda} = 0$$

Notice that

$$\frac{\partial \Lambda(x,y,z)}{\partial \lambda} = 0$$

automatically implies the validity of the constraint

$$g(x,y) = c$$

thereby accommodating it in a natural way.

- Further

$$\frac{\partial^2 \Lambda(x,y,z)}{\partial \lambda^2} = 0$$

always, since $\Lambda(x,y,z)$ is linear in λ. Further, since the constraint is true by the optimizer construction, there should be no explicit dependence on λ.

3. Comparison with unconstrained optimization:
   - Unconstrained Optimization results in

$$\frac{\partial L(x,y)}{\partial x} = 0$$

and

$$\frac{\partial L(x,y)}{\partial y} = 0$$

Constrained Optimization results in

$$\frac{\partial \Lambda(x,y,z)}{\partial x} = \frac{\partial L(x,y)}{\partial x} + \lambda \frac{\partial g(x,y)}{\partial x}$$

and

$$\frac{\partial \Lambda(x,y,z)}{\partial y} = \frac{\partial L(x,y)}{\partial y} + \lambda \frac{\partial g(x,y)}{\partial y}$$

$$\lambda = 0$$

automatically reduces the constrained case to the unconstrained.

- Advantage of the Lagrange Multiplier Incorporation into Optimization => This ends up converting the constrained formulation space over on to the unconstrained, thereby providing with as many equations as the number of optimization unknowns, and one equation each per every constraint.
- Drawback of the Lagrange Multiplier Optimization Incorporation => While it lets you passively move to the unknowns' space from the constrained formulation space, this may end up impacting the application of certain boundary conditions, such as financial boundary, natural boundary conditions, Not-A-Knot boundary condition, etc.

4. Constraints of inequality $g(x,y) < c$ or $g(x,y) > c$: Solutions to these constraints exist either inside the unconstrained set, in which case the unconstrained equations can be solved, or they don't, in which case convert the inequality to an equality and give it the Lagrange multiplier treatment.
5. Comparison with Convex Optimization: Convex optimization is predicated on the presence of at least one minimum/maximum in the zone of interest. One example is variance/covariance constrained optimization, where both the variance/covariance and the constraints are both quadratic. Eigenization (just another type of constrained variance/covariance optimization) is another.
6. Penalizing Optimizer as a Constrained Optimization Setup: Naively put

$$\Lambda = Good - Bad$$

where, from a calibration point of view, "Good" refers to closeness of fit, and "Bad" to the curvature/smoothness penalty. Thus, constrained optimization here corresponds to "maximize the Good, and minimize the Bad".

- De-coupling "Good/Bad" from closeness of fit and curvature/smoothness penalty, respectively, can lead to alternate optimization framework formulations in finance, along with its insights.

Least Squares Optimizer

1. Least Squares Optimization Formulation:

$$\mu_i(x_i) = y_i$$

$$\hat{\mu}_i(x_i) = \sum_{j=1}^{n} a_j B_{ij}(x_i)$$

$$S = \sum_{i=1}^{m} [\mu_i(x_i) - \hat{\mu}_i(x_i)]^2 = \sum_{i=1}^{m} \left[y_i - \sum_{j=1}^{n} a_j B_{ij}(x_i) \right]^2$$

$$= \sum_{i=1}^{m} \left\{ {y_i}^2 - 2y_i \sum_{j=1}^{n} a_j B_{ij}(x_i) + \left[\sum_{j=1}^{n} a_j B_{ij}(x_i) \right]^2 \right\}$$

$$\frac{\partial S}{\partial a_j} = 2 \left[\sum_{i=1}^{m} a_j B_{ij}(x_i) - y_j \right] \left[\sum_{i=1}^{m} a_j B_{ij}(x_i) \right]$$

2. Least Squares Matrix Formulation:

$$Y = [y_1, \cdots, y_m]^T$$

$$B_i(\vec{x}) = \left[B_{i,1}(\vec{x}), \cdots, B_{i,m}(\vec{x})\right]^T$$

$$a = [a_1, \cdots, a_m]^T$$

Then

$$\left[\frac{\partial \vec{S}}{\partial A}\right] = 2AB(\vec{x})B^T(\vec{x})A - 2YB(\vec{x})$$

As expected,

$$y_i = \sum_{j=1}^{n} a_j B_{ij}(x_i)$$

is the optimized least squares tight fit.

Karush-Kuhn-Tucker Conditions

Introduction, Overview, Purpose, and Motivation

1. KKT Conditions - Necessity and Scope: In mathematical optimization the Karush-Kuhn-Tucker (KKT) Conditions – also known as Kuhn-Tucker Conditions – are the first order necessary conditions for a solution in non-linear programming to be optimal provided some regularity conditions are satisfied (Karush-Kuhn-Tucker Conditions (Wiki)).
2. KKT Conditions vs. Lagrange Multipliers: Allowing for inequality constraints the KKT approach to non-linear programming generalizes the method of Lagrange multipliers, which only allows for equality constraints.
3. Mathematical Foundation behind Optimization Algorithms: The system of equations corresponding to the KKT conditions is usually not solved directly except in a few special cases where a closed-form solution may be derived analytically. In general, many optimization algorithms can be interpreted as methods for numerically solving the KKT systems of equations (Boyd and van den Berghe (2009)).
4. KKT Conditions - Historical Attribution Revision: The KKT conditions were originally named after Harold W. Kuhn and Albert W. Tucker who first published the conditions in 1951 (Kuhn and Tucker (1951)). Later scholars discovered that then necessary conditions for this problem had been stated by William Karush in his master's thesis (Karush (1939), Kjeldsen (2000)).

Necessary Conditions for Optimization Problems

1. KKT Statement - Non-linear Optimization Problem: Consider the following non-linear optimization problem: Maximize $f(x)$ subject to

$$g_i(x) \leq 0$$

and

$$h_j(x) = 0$$

where x is the optimization variable, f is the *objective* or the *utility* function, $g_i(i = 1, \cdots, m)$ are the inequality constraint functions, and $h_j(j = 1, \cdots, l)$ are the equality constraint functions. The numbers of the inequality and the equality constraints are denoted m and l respectively.

2. KKT Statement - The Necessary Condition: Suppose that the objective function

$$f: \mathbb{R}^n \to \mathbb{R}$$

and the constraint functions

$$g_i: \mathbb{R}^n \to \mathbb{R}$$

and

$$h_j: \mathbb{R}^n \to \mathbb{R}$$

are continuously differentiable at the point x^*. If x^* is a local optimum that satisfies some regularity conditions below, there exist constants $\mu_i(i = 1, \cdots, m)$ and $\lambda_j(j = 1, \cdots, l)$ called KKT multipliers that have the following properties.

3. KKT Statement - Optimal Stationary Conditions: For maximizing $f(x)$

$$\vec{\nabla} f(x^*) = \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

For minimizing $f(x)$

$$-\vec{\nabla} f(x^*) = \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

4. KKT Statement - Primal Feasibility Conditions:

$$g_i(x^*) \leq 0$$

for all

$$i = 1, \cdots, m$$

and

$$h_j(x^*) = 0$$

for all

$$j = 1, \cdots, l$$

5. KKT Statement - Dual Feasibility Conditions:

$$\mu_i \geq 0$$

for all

$$i = 1, \cdots, m$$

6. KKT Statement – Complementary Slackness Conditions:

$$\mu_i g_i(x^*) = 0$$

for all

$$i = 1, \cdots, m$$

7. Reduction to the Lagrange Criterion: In the particular case

$$m = 0$$

where there are no inequality constraints the KKT conditions turn into the Lagrange conditions and the KKT multipliers become the Lagrange multipliers.

8. Non-differentiable Version of KKT: If some of the functions are non-differentiable sub-differentiable versions of the KKT conditions are available (Eustaquio, Karas, and Ribeiro (2008)).

Regularity Conditions or Constraint Qualifications

1. The KKT Necessary Regularity Conditions: In order for a minimum point x^* to satisfy the above KKT conditions the problem should satisfy some regularity conditions. The most common ones are listed below.
2. Linear Constraint Qualification: If g_i and h_j are affine functions the no other condition is needed to be satisfied.

3. Linear Independence Constraint Qualification (LICQ): The gradients of the active inequality constraints and the gradients of the equality constraints are linearly independent at x^*.
4. Mangasarian-Fromovitz Constraint Qualification (MFCQ): The gradients of the active inequality constraints and the gradients of the equality constraints are positive-linearly independent at x^*.
5. Constant Rank Constraint Qualification (CRCQ): For each subset of the gradients of the active inequality constraints and the gradients of the active equality constraints the rank at the vicinity of x^* is constant.
6. Constant Positive Linear Dependence Constraint Qualification (CPLD): For each subset of the gradients of the active inequality constraints and the gradients of the equality constraints, if it is positive-linearly dependent at x^*, then it is positive-linearly dependent at the vicinity of x^*.
7. Quasi-Normality Constraint Qualification (QNCQ): If the gradients of the active inequality constraints and the gradients of the active equality constraints are positive-linearly dependent at x^* with the associated multipliers λ_i for equalities and μ_j for inequalities then there is no sequence

$$x_k \to x^*$$

such that

$$\lambda_i \neq 0$$

implies

$$\lambda_i h_i(x_k) > 0$$

AND

$$\mu_j \neq 0$$

implies

$$\mu_j g_j(x_k) > 0$$

8. Slater Condition: For a convex problem there exists a point x such that

$$h_j(x) = 0$$

and

$$g_i(x) < 0$$

9. Positive Linear Dependency Condition Definition: The set of vectors $\{v_1, \cdots, v_n\}$ is positive linearly dependent if there exists

$$a_1 \geq 0, \cdots, a_n \geq 0$$

not all zero such that

$$a_1 v_1 + \cdots + a_n v_n = 0$$

10. Strength Order of Constraint Qualifications: It can be shown that

$$LICQ \Longrightarrow MFCQ \Longrightarrow CPLD \Longrightarrow QNCQ$$

and

$$LICQ \Longrightarrow CRCQ \Longrightarrow CPLD \Longrightarrow QNCQ$$

– the converses are not true – although MFCQ is not equivalent to CRCQ (Ruszczynski (2006)). In practice weaker constraint qualifications are preferred since they provide stronger optimality conditions.

Sufficient Conditions

1. The Second Order Sufficiency Conditions: In some cases, the necessary conditions are also sufficient for optimality. In general, the necessary conditions are not sufficient for optimality and more information is needed, such as the Second Order Sufficiency Conditions (SOSC). For smooth functions SOSC involve the second derivatives, which explains its name.
2. When Necessary Conditions are Sufficient: The necessary conditions are sufficient for optimality of the objective function f of a maximization problem is a concave function, the inequality constraints g_i are continuously differentiable convex functions, and the equality constraints h_j are affine functions.
3. The Type 1 Invex Functions: The broader class of functions in which the KKT conditions guarantee global optimality are the so-called Type 1 **invex functions** (Martin (1985), Hanson (1999)).
4. Second Order Sufficiency Conditions Formulation: For smooth non-linear optimization problems the second order sufficiency conditions are given as follows. Consider x^*, λ^*, and μ^* that find a local minimum using the Karush-Kuhn-Tucker conditions above. With μ^* such that the strict complementary condition is held at x^*, i.e.

$$\mu > 0$$

for all

$$s \neq 0$$

such that

$$\left[\frac{\partial g(x^*)}{\partial x}, \frac{\partial h(x^*)}{\partial x}\right] s = 0$$

the following equation must hold:

$$s^T \nabla_{xx}{}^2 \mathcal{L}(x^*, \lambda^*, \mu^*) s > 0$$

If the above condition is strictly met, the function is a strict constrained local minimum.

KKT Conditions Application - Economics

1. KKT Models as Theoretical Tools: Often in mathematical economics the KKT approach is used in theoretical models in order to obtain qualitative results.
2. Minimum Profit Constraint Revenue Maximization: Consider a firm that maximizes its sales revenue subject to a minimum profit constraint. Letting Q be the quantity of output produced – this is to be chosen - $R(Q)$ be the sales revenue with a positive first derivative and with a zero value at zero output, $C(Q)$ be the production cost with a positive first derivative and with a non-negative value at zero output, and G_{min} be the positive minimal acceptable level of profit, then the problem is a meaningful one if the revenue function levels off so it is eventually less steep than the cost function.
3. Application of the KKT Condition: The problem can be expressed in the following minimization form: Minimize $-R(Q)$ subject to

$$G_{min} \leq R(Q) - C(Q)$$

$$Q \geq 0$$

and the KKT conditions are

$$Q\left[\frac{\partial R}{\partial Q}(1+\mu) - \mu\frac{\partial C}{\partial Q}\right] = 0$$

$$R(Q) - C(Q) - G_{min} \geq 0$$

$$\mu \geq 0$$

$$\mu[R(Q) - C(Q) - G_{min}] = 0$$

4. Revenue Growth vs. Cost Growth: Since

$$Q = 0$$

would violate the minimum profit constraint

$$Q > 0$$

must hold, and hence the third condition implies that the first condition holds with equality. Solving that equality gives

$$\frac{\partial R}{\partial Q} = \frac{\mu}{1+\mu}\frac{\partial C}{\partial Q}$$

5. Impact of the Minimum Profit Constraint: Because it was given that $\frac{\partial R}{\partial Q}$ and $\frac{\partial C}{\partial Q}$ are strictly positive, the inequality along with the non-negativity condition on μ is

positive and so the revenue-maximizing firm operates at a level of output at which the marginal revenue $\frac{\partial R}{\partial Q}$ is less than the marginal cost $\frac{\partial C}{\partial Q}$ - a result that is of interest because it contrasts the behavior of a profit maximizing firm which operates at a level at which they are equal.

KKT Conditions Application – Value Function

1. Equality/Inequality Function Space Constraints: Reconsidering the optimization problem as a maximization problem with constant inequality constraints $v_1, \cdots, v_n$ Maximize $f(x)$ subject to

$$g_i(x) \leq a_i$$

$$h_j(x) = 0$$

2. Definition of the Value Function: The value function is defined as

$$V(a_1, \cdots, a_n) = \sup_x f(x)$$

subject to

$$g_i(x) \leq a_i$$

$$h_j(x) = 0$$

$$j \in \{1, \cdots, l\}$$

$$i \in \{1, \cdots, m\}$$

So, the domain of V is

$$a \in \mathbb{R}^m$$

for some

$$x \in X$$

$$g_i(x) \leq a_i$$

$$i \in \{1, \cdots, m\}$$

3. Interpretation of a_i and μ_i: Give this definition each coefficient μ_i is the rate at which the value of the function increases as a_i increases. Thus, if each a_i is interpreteed as a resource constraint the coefficients indicate how much increasing the resource increases the optimum value of f. This interpretation is especially important in economics and is used, for example, in utility maximization problems.

Generalizations

1. KKT Extensions – Fritz John Conditions: With an extra constant multiplier μ_0 which may be zero, in front of $\vec{\nabla} f(x^*)$ the KKT stationary conditions turn into

$$\mu_0 \vec{\nabla} f(x^*) + \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

which are called the Fritz John conditions.

2. KKT Class as FONC Support: The KKT conditions belong to the wider class of the First Order Necessary Conditions (FONC) which allow for non-smooth functions using sub-derivatives.

References

- Boyd, S., and L. van den Berghe (2004): *Convex Optimization* **Cambridge University Press**
- Eustaquio, R., E. Karas, and A. Ribeiro (2008): *Constraint Qualification under Non-linear Programming* Technical Report **Federal University of Parana**
- Hanson, M. A. (1999): Invexity and the Kuhn-Tucker Theorem *Journal of Mathematical Analysis and Applications* **236 (2)** 594-604
- Karush, W. (1939): *Minima of Functions of Several Variables with Inequalities as Side Constraints* Master's Thesis **University of Chicago** Chicago Illinois
- Kjeldsen, T. H. (2000): A Contextualized Analysis of the Kuhn-Tucker Theorem in Non-linear Programming: The Impact of World War II *Historia Mathematica* **27 (4)** 331-361
- Martin, D. H. (1985): The Essence of Invexity *Journal of Optimization Theory and Applications* **47 (1)** 65-76
- Ruszczynski, A. (2006): *Nonlinear Optimization* **Princeton University Press** Princeton NJ
- Wikipedia (2019): Karush Kuhn Tucker Conditions

Interior Point Method

Motivation, Background, and Literature Survey

1. Definition of Interior Point Methods: Interior Point Methods (also known as *barrier methods*) are a class of algorithms that solves linear and non-linear convex optimization problem (Interior Point Method (Wiki)).
3. John von Neumann's Early Approach: John von Neumann suggested an interior point method of linear programming that was neither a polynomial time method not an efficient method in practice (Dantzig and Thapa (2003)). In fact, it turned out to be slower in practice than the simplex method which is not a polynomial time method.
4. Karmarkar's Extension to the Simplex Method: Karmarkar (1984) developed a method for linear programming called the Karmarkar's algorithm which runs in provably polynomial time, and is also very efficient in practice. It enabled solutions to linear programming problems that were beyond the capabilities of the simplex method.
5. Traversal across the Feasible Region: Contrary to the Simplex method Karmarkar's algorithm reaches the best solution by traversing the interior of the feasible region. The method can be generalized to convex programming based on a self-concordant barrier function used to encode the convex set.
6. Transformation of the Convex Function: Any convex optimization problem can be transformed into minimizing (or maximizing) a linear function over a convex set by converting to an epigraph form (Boyd and van den Berghe (2004)). The idea of encoding the feasible set using a barrier and designing barrier methods was studied by Anthony V. Fiacco, Garth P. McCormick, and others in the early '60s. These ideas were mainly developed for general non-linear programming, but they were later abandoned due to the presence of more competitive methods for this class of problems (e.g., sequential quadratic programming).

7. Barriers to Encode Convex Set: Yuri Nesterov and Arkadi Nemirovskii came up with a special class of such barriers that can be used to encode any convex set. They provide guarantees that the number of iterations of the algorithm is bounded by a polynomial in the dimension and as well for the accuracy of the solution (Wright (2004)).
8. Interior Point Methods with Barriers: Karmarkar's breakthrough re-vitalized the study of interior point methods and barrier problems showing that it was possible to create an algorithm for linear programming characterized by polynomial time complexity, and, moreover, that was competitive with the simplex method. Already Khachiyan's ellipsoid method was a polynomial time algorithm; however, it was too slow to be of practical interest.
9. Interior Point Method Sub-classes: The class of primal dual path-following interior point methods is considered the most successful. Mehrotra's predictor-corrector algorithm provides the basis for most implementations of this class of methods (Mehrotra (1992), Potra and Wright (2000)).

Interior Point Methodology and Algorithm

1. Principle, Concept, and Approach Methodology: The main idea behind interior point methods is to iterate inside of the feasible set and progressively approach the boundary – if the minimum does lie on the boundary. This is done by transforming the original problem into a sequence of unconstrained optimization problems in which the objective function uses a *barrier function* that goes to ∞ at the boundaries of the feasible region. By reducing the strength of the barrier at each subsequent optimization the sequence of minima approach arbitrarily closely toward the minimum of the original problem.
2. Objective Function with Logarithmic Barriers: For the inequality constrained problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$g_i(x) \leq 0, i = 1, \cdots, m$$

f is modified to obtain a *barrier function*

$$f_\alpha(x) = f(x) - \alpha \sum_{i=1}^{m} \log g_i(x)$$

3. Impact of the Barrier Strength Parameter: Since the logarithm goes to $-\infty$ as its argument approaches 0 the modified barrier function becomes arbitrarily large as x approaches the boundaries of the feasible region. The parameter α gives the barrier "strength". Its significance is that as α approaches 0 the optimum of $f_\alpha(x)$ approaches the optimum of $f(x)$. More precisely if f^* optimizes $f(x)$ and

$$x_\alpha{}^* \equiv \underset{x}{\arg min}\, f_\alpha(x)$$

then

$$\underset{\alpha \to 0}{Limit}\, f(x_\alpha{}^*) = f^*$$

4. Gradient of the Barrier Function: Newton's method is employed to find the minimum of $f_\alpha(x)$ The gradient is

$$\nabla f_\alpha(x) = \nabla f(x) - \alpha \sum_{i=1}^{m} \frac{\nabla g_i(x)}{g_i(x)}$$

5. Numerical Stability Close to the Boundary: One concern is that as $x_\alpha{}^*$ proceeds closer and closer to the boundary the numerical stability of the unconstrained optimization problem becomes worse and worse because the denominator approaches zero. Thus, it would be very challenging to apply the gradient descent to a small tolerance.
6. Use of KKT Type Multipliers: Therefore, another set of KKT multiplier like variables

$$\vec{\lambda} = (\lambda_1, \cdots, \lambda_m)$$

with

$$\lambda_i \equiv \frac{\alpha}{g_i(x)}$$

is introduced. So, in the $\left(x, \vec{\lambda}\right)$ space, one seeks the root of the equation

$$\nabla_x f_\alpha\left(x, \vec{\lambda}\right) = \nabla f(x) - \sum_{i=1}^{m} \lambda_i \nabla g_i(x) = 0$$

subject to the equality

$$\lambda_i g_i(x) = \alpha$$

for

$$i = 1, \cdots, m$$

7. Stability of the Modified Solution: This set of equations is far more numerically stable than

$$\nabla f_\alpha(x) = \nabla f(x) - \alpha \sum_{i=1}^{m} \frac{\nabla g_i(x)}{g_i(x)}$$

near the boundary. Note the similarity between these equations and the KKT equations. In fact, if α were 0 one gets the KKT equations except with the equalities stripped away.

8. Objective/Constraint Function - Partial Derivatives: To find a root $\left(x_\alpha{}^*, \vec{\lambda}_\alpha{}^*\right)$ where both of the functions are satisfied the Newton's method is applied. To do so one needs the partial derivatives of the above functions.

$$\nabla_{xx}{}^2 f_\alpha\left(x, \vec{\lambda}\right) = \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$$

$$\nabla_{\vec{\lambda}} \nabla_x f_\alpha(x) = -\sum_{i=1}^{m} \nabla g_i(x)$$

$$\nabla_x(\lambda_i g_i(x) - \alpha) = \lambda_i \nabla g_i(x)$$

$$\frac{\partial}{\partial \lambda_i}[\lambda_i g_i(x) - \alpha] = g_i(x)$$

for

$$i = 1, \cdots, m$$

9. Variate/Constraint Multipliers newton Increment: At the current iterate $\left(x_t, \vec{\lambda}_t\right)$ the Newton step $\left(\Delta x_t, \Delta\vec{\lambda}_t\right)$ is derived from the following system of equations.

$$\begin{bmatrix} \mathcal{H} & -\mathcal{G} \\ Diagonal(\vec{\lambda}_t) \cdot \mathcal{G}^T & Diagonal(\vec{g}) \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta \vec{\lambda}_t \end{bmatrix} = \begin{bmatrix} -\nabla f(x_t) + \mathcal{G} \cdot \vec{\lambda}_t \\ \alpha \mathbb{I} - Diagonal(\vec{g}) \cdot \vec{\lambda}_t \end{bmatrix}$$

where $\mathcal{H}$ denotes the Hessian

$$\nabla_{xx}{}^2 f_\alpha(x, \vec{\lambda}) = \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$$

evaluated at $(x_t, \vec{\lambda}_t)$, $\vec{g}$ denotes the $m \times 1$ vector of g_i's, $\mathcal{G}$ denotes the $n \times m$ matrix whose i^{th} column is $\nabla g_i(x_t)$, $Diagonal(\vec{v})$ produces a square matrix whose diagonal is the vector $\vec{v}$, and $\mathbb{I}$ denotes the $m \times 1$ vector of all 1's.

10. Variate Retention inside Feasible Region: Solving this system of equations provides a search direction that is then update via line search to avoid divergence. Note that to keep x drifting out of the feasible set one must enforce for all i the condition

$$g_i(x) \geq 0$$

or equivalently

$$\lambda_i \geq 0$$

during the line search.

11. Progressive Reduction of the Barrier Strength: Once the unconstrained search has converged α may be reduced (e.g., by multiplication by a small number) and the optimization can begin again. There is a trade-off in the convergence thresholds for each unconstrained search; too small and the algorithm must perform a lot of work even when α is high; but too large and the sequence of unconstrained optima does not converge quickly. A more balanced approach is to choose the threshold proportional to α.

12. Application to Non-Convex Functions: The formulation above did not explicitly require that the problem be convex. Interior point methods can certainly be used in general non-convex problems, but like any local optimization problem they are not guaranteed to a minimum. Furthermore, computing the Hessian matrix is quite often expensive.
13. Use in Linear/Quadratic Problems: They do, however, work quiet well in convex problems. For example, in quadratic programming the Hessian is constant, and in linear programming the Hessian is zero.
14. Initialization at a Feasible Point: The interior point method must be initialized at an interior point, or else the barrier function is undefined. To find the initial feasible point x the following optimization may be used.

$$\underset{x \in \mathbb{R}^n, s_1, \cdots, s_m}{max} \sum_{i=1}^{m} s_i$$

such that

$$g_i(x) - s_i \geq 0, i = 1, \cdots, ms_i \geq 0$$

If the problem is feasible then the optimal s_i will all be 0. It is easy to find an initial set of s_i for any given x simply by setting

$$s_i = \min(g_i(x), 0)$$

References

- Boyd, S., and L. van den Berghe (2004): *Convex Optimization* **Cambridge University Press**

- Dantzig, G. B., and M. N. Thapa (2003): *Linear Programming 2 – Theory and Extensions* **Springer-Verlag**
- Karmarkar, N. (1984): A New Polynomial-Time Algorithm for Linear Programming *Combinatorica* **4 (4)** 373-395
- Mehrotra, S. (1992): On the Implementation of the Primal-Dual Interior Point Method *SIAM Journal on Optimization* **2 (4)** 575-601
- Potra, F. A., and S. J. Wright (2000): Interior Point Methods *Journal of Computational and Applied Mathematics* **124 (1-2)** 281-302
- Wikipedia (2019): Interior Point Method
- Wright, M. H. (2004): The Interior-Point Revolution in Optimization; History, Recent Developments, and Lasting Consequences *Bulletin of the American Mathematical Society* **42 (1)** 39-56

Portfolio Selection with Cardinality and Bound Constraints

Synopsis

1. Portfolio Selection Problem – Formulation/Solution: Tadonki and Vial (2004) formulate and solve the portfolio selection problem with transaction costs and bound constraints on both the number of selected assets and the range of the corresponding investments.
2. Objective Function Risk/Return Tradeoff: The problem is to find a set of assets and the corresponding investment levels that suit the trade-off between risk and reward.
3. Combinatorial Mixed Integer Quadratic Problem: The underlying model results in a mixed integer quadratic problem. Tadonki and Vial (2004) show that the bound constraints on the investment yields a difficult combinatorial problem.
4. Feasibility Handling through Virtual Variable: To overcome this feasibility problem. Tadonki and Vial (2004) add a virtual variable that will remain selected if and only if the problem is infeasible.
5. Optimal Solution Greedy Heuristic Approximation: They propose a powerful greedy heuristic that provides a good approximation of the optimal solution in an affordable computation time.
6. Branch and Bound Global Solution: The algorithm for finding a global solution is a standard branch and bound processing.
7. Combining Heuristics with Cutting Planes: Tadonki and Vial (2004) use a cutting plane routine to solve the generic problem, and the heuristic to find potential upper bounds.
8. Test Runs on Benchmark Problems: Experimental results on a set of benchmark problems shows that the algorithm is quite efficient.

Introduction

1. Mean-Variance Portfolio Selection Formulation: The classical approach to the portfolio selection problem (Constantinides and Malliaris (1995)) is the Markowitz mean-variance formulation (Markowitz (1952, 1959)). The mean and the variance of a portfolio's return represent the benefit and the risk associated with the investment.
2. Standard Quadratic Convex Programming Problem: The approach boils down to a simple convex quadratic programming problem that is easily solved by standardized toolkits (CPLEX (2019), MOSEK (2019)). Various other aspects of the problem have also been studied by a number of authors (Mossin (1968), Samuelson (1969), Hakansson (1971), Zariphopoulou (1992), Li and Ng (2000), Zhou and Li (2000)).
3. Extensions to the Standard Framework: The problem can be considered with special properties; bounds on the total investment, fixed amount of investment, pre-determined set of assets, and lower/upper bounds on risk/rewards.
4. Accommodation of Discrete Portfolio Selection: In addition, in order to capture the realism of portfolio selection, a number of discrete constraints have been considered by different authors (Jobst, Horniman, Lucas, and Mitra (2001)), in particular, the *cardinality constraint*, which enables to take into account the expected number of assets in the portfolio.
5. Bounds on the Number of Assets: Jobst, Horniman, Lucas, and Mitra (2001) have studied the case where the number of assets to be selected is fixed. This chapter considers an upper bound instead.
6. Approaches for Handling Cardinality Constraints: For investors, the cardinality constraint is important for monitoring and control purposes. Chang, Meade, Beasley, and Sharaiha (1999) have proposed heuristics – genetic algorithm, tabu search, and simulated annealing – to solve for the cardinality constraint.
7. Lower Bounds on Invested Assets: Investors also use the so-called *buy-in thresholds*, which specifies the minimum investment level, and therefore eliminates small trades in the selection.

8. Upper Bounds on Invested Assets: This chapter also considers *buy-in limit*, which specifies the maximum investment level on an asset (Bienstock (1996), Lee and Mitchell (1997)). It is obvious that bounding the investments has an impact on the number of selected assets.
9. The Round-Lot Investment Constraint: Another interesting constraint is the round-lot constraint, which are discrete number is the assets taken as the basic units of the selection. The size of the portfolio is the integer linear combination of the round-lots. The feasibility problem coming from the round-lot constraints has been established to be NP-complete (Mansini and Speranza (1999)).
10. Incorporating Fixed Transaction Cost Constraint: This chapter extends the base mean-variance portfolio model by considering three special constraints. First, a transaction cost is assumed on each item. Prior treatments of these costs include Adcock and Meade (1994) and Young (1998).
11. Total Number of Assets Bound: Second, a bound on the total number of assets in the portfolio is imposed.
12. Lower and Upper Asset Bounds: Third, a lower and an upper limit is considered on the total number of items in the portfolio.
13. Mixed Integer Quadratic Programming Problem: The resulting mathematical problem is a quadratic mixed integer program.
14. Feasibility Handling via Virtual Barrier: The restriction on the investment levels yields a feasibility problem, which is another issue to consider. The standard approach to overcome this problem is to add a free virtual with a NULL reward and a high transaction cost.
15. Heuristics Approach for the Solution: Tadonki and Vial (2004) develop a heuristic that gives a good approximation of the problem in its generic form. The heuristic is used at different level trees of the branch-and-bound process to obtain upper bounds.
16. Analytic Center Cutting Plane Solver: The node problem is solved using an analytic center cutting planes solver named PROXACCPM (Gondzio, du Merle, Sarkissian, and Vial (1996)).

17. Organization of the Chapter: The chapter is organized as follows. The first section presents the formulation of the problem as it is considered in this treatment. This is followed by a complexity analysis of the problem.
18. Bender Decomposition and Greedy Heuristic: The Bender decomposition approach is presented in the next section, followed by the Tadonki and Vial (2004) greedy heuristic.
19. Cutting Plane Algorithm and Performance: The next section describes the cutting plane algorithm and its performance on the generic problem.
20. Chvatal-Gomory Cuts and Variants: This is followed by a presentation of the Chvatal-Gomory cuts and some variants.
21. Branching Rule and Node Selection: The penultimate section describes the branching rule and the node selection used in Tadonki and Vial (2004).
22. Discussion of Results and Conclusions: Computational Results are then presented, and the chapter is concluded in the final section.

Problem Formulation

1. Covariance Matrix and the Returns Vector: Let

$$V \in \mathbb{R}^{n \times n}$$

be the variance-covariance matrix among the n assets, and let

$$r \in \mathbb{R}^{n}$$

be the vector of the expected returns of the assets.

2. Asset Weights as the Decision Vector: The decision variable

$$x \in \mathbb{R}^{n}$$

represents the fractional share of each asset in the portfolio. By definition, x belongs to the simplex, that is,

$$e^T x = 1$$

where

$$e = (1, \cdots, 1)$$

and

$$x \geq 0$$

3. Lower and Upper Bounds Vector: The lower and the upper bounds on the investment are given by the positive vectors

$$d, f \in \mathbb{R}^n$$

$$0 \leq d$$

$$f \leq 1$$

D and F denote the diagonal matrix with the main diagonal d and f, repectively.

4. Number of Assets Bound and the Transaction Cost: Finally, the fixed relative transaction cost is given by the positive vector

$$h \in \mathbb{R}^n$$

and

$$p \leq n$$

is a positive integer to bound the number of assets in the portfolio.

5. Objective Function and Constrains: To complete the formulation of the problem, the binary variables

$$y \in \{0,1\}^n$$

are introduced. The problem is:

$$\min_{x,y} \frac{1}{2} x^T V x - \mu r^T x + h^T y$$

$$e^T x = 1$$

$$Dy \leq x \leq Fy$$

$$e^T y \leq p$$

$$y \in \{0,1\}^n$$

6. Solving for the Optimal Allocation: The purpose of this chapter is to provide an efficient routine to solve the portfolio allocation problem based on the above formulation setup. The focus will be on a *Bender decomposition* approach together with a branch and bound technique.

Analysis of the Problem

1. Definition of the Optimization Parameters: For a given non-NULL vector

$$x \in \mathbb{R}^n$$

and two vectors

$$u, v \in \mathbb{R}^n$$

such that

$$u \leq v$$

$\varphi(x)$ - respectively $\psi(x, u, v)$ - is defined as the set of indices where the corresponding component value of x is non-NULL, i.e., x is between that of u and v:

$$\varphi(x) = \{i \in (1, \cdots, n)\ s.t.\ x_i \neq 0\}$$

$$\psi(x, u, v) = \{i \in (1, \cdots, n)\ s.t.\ u_i \leq x_i \leq v_i\}$$

2. Feasible Asset Subset - Lemma Statement: Given two vectors

$$u, v \in \mathbb{R}^n$$

and an integer

$$1 \leq p$$

there is a vector

$$x \in \mathbb{R}^n$$

such that

$$e^T x = 1$$

$$|\psi(x,u,v)| \leq p$$

$$\psi(x,u,v) = \varphi(x)$$

if and only if there is a subset

$$I \subset (1,\cdots,n)$$

with

$$|I| \leq p$$

such that

$$\sum_{i \in I} u_i \leq 1 \leq \sum_{i \in I} v_i$$

3. Feasible Asset subset Lemma Proof: Let

$$x \in \mathbb{R}^n$$

satisfying

$$e^T x = 1$$

$$|\psi(x,u,v)| \leq p$$

$$\psi(x,u,v) = \varphi(x)$$

It is easy to see that for

$$I = \varphi(x)$$

the relation

$$\sum_{i \in I} u_i \leq 1 \leq \sum_{i \in I} v_i$$

holds.

4. Proof of the Inverse Lemma: Let

$$I \subset (1, \cdots, n)$$

with

$$|I| \leq p$$

such that the relation

$$\sum_{i \in I} u_i \leq 1 \leq \sum_{i \in I} v_i$$

holds. The function τ defined by

$$\begin{matrix}\mathbb{R}^n \cap [u,v] \to \\ x \end{matrix} \begin{matrix}\mathbb{R}^n \\ \to \tau(x) = \sum_{i \in I} x_i\end{matrix}$$

is continuous and contains the value 1 inside its range, thus there is a vector

$$w \in [u,v]$$

such that

$$\tau(w) = 1$$

5. Recovery of x from w: The required vector x is then obtained as follows:

$$x_i = \begin{cases} w_i : i \in I \\ 0 : i \notin I \end{cases}$$

6. Condition for Portfolio Selection Feasibility: The above lemma provides a condition for the feasibility of the portfolio selection problem. However, the underlying combinatorial problem is NP-hard. This can be stated as follows.
7. Optimal Vector Determination Complexity - Inputs: An integer n, another integer p,

$$1 \leq p \leq n$$

and two vectors

$$u, v \in \mathbb{R}^n$$

$$0 < u$$

$$v < 1$$

such that

$$u \leq v$$

8. Optimal Vector Determination Complexity - Problem: The problem statement is: Is there a subset

$$I \subset (1, \cdots, n)$$

with

$$|I| \leq p$$

such that

$$\sum_{i \in I} u_i \leq 1 \leq \sum_{i \in I} v_i$$

9. NP-Complete Nature of the Subset Problem: To show that the problem is NP-hard, simply consider the case where

$$u = v$$

Here one obtains an instance which is equivalent to the *subset sum problem*, which is known to be NP-complete (Garey and Johnson (1979)).

10. Case where the Original Problem is Feasible: Consequently, one adds an artificial variable with any investment restriction, and which does not provide any profit. This ensures that the problem continues to be feasible.

11. Case where the Original Problem is Infeasible: However, of the considered case $u = v$ is infeasible, one seeks an exhaustive – and hence exponential – exploration process ending with the artificial point as the best solution found.
12. Problem Re-formulation with Augmented Variables: Technically, if V, μ, h, d and f are original parameters, the problem may be re-formulated using the following augmented variables: $\begin{pmatrix} V & 0 \\ 0 & \max(diagonal\, V) \end{pmatrix}$, $\begin{pmatrix} r \\ 0 \end{pmatrix}$, $\begin{pmatrix} h \\ \max h \end{pmatrix}$, $\begin{pmatrix} d \\ 0 \end{pmatrix}$, and $\begin{pmatrix} f \\ 1 \end{pmatrix}$. The rest of the chapter assumes that there is at least one variable with restriction on its investment.

Bender's Decomposition

1. Alternate Route to Computing Bounds: Bender's decomposition offers an alternative route to computing the bounds.
2. Convex Quadratic Program without Cardinality: Let $Q(y)$ be the value of the following convex quadratic program:

$$Q(y) = \underset{x,y}{min}\ \frac{1}{2}x^T V x - \mu r^T x + h^T y$$

   The above quadratic program can be easily computed using standard approaches. Moreover, the dual variables obtained from the computation of $Q(y)$ can be used to provide a valid sub-gradient at y, as will be seen down.
3. Convex Quadratic Program with Cardinality: The above problem is difficult in two respects. First, the component $Q(y)$ is defined implicitly. It can only be approximated by the supporting hyper-planes, but this introduces a continuous variable

$$\varsigma \geq Q(y)$$

4. Complexity of Mixed-Integer Programming: Secondly, the outer approximation obtained by adding convexity cuts – the supporting hyper-planes – yields a mixed integer programming problem.
5. Relaxation of Binary to Linear Constraints: A possible course of action is to relax the binary constraints and solve the linear problem. If the solution of the relaxation is boolean in y, then the solution is globally optimal. However, we may not expect that it will be so in general.
6. Branch and Bound Heuristic Solutions: Consequently, exploration techniques like *branch* and *bound* should be considered. For this purpose, the next section provides a heuristic that results in a feasible approximation to the problem.

A Greedy Heuristic

1. Cardinality Constrained Quadratic Convex Problem: A key goal here is to provide a heuristic which results in a feasible solution to

$$\min_{x,y} \left(\frac{1}{2} x^T V x - \mu r^T x + h^T y \mid e^T x = 1, Dy \leq x \leq Fy, e^T y \leq p, y \in \{0,1\}^n \right)$$

2. Upper Constrained Quadratic Convex Problem: Let x be the solution to

$$\min_{x} \left(\frac{1}{2} x^T V x - \mu r^T x + p h^T x \mid e^T x = 1, 0 \leq x \leq f \right)$$

3. Lower Bound Adhering/Violating Allocations: Let J, K be the two sets of

$$I \subset (1, \cdots, n)$$

defined by

$$J = \{i \in I : x_i < d_i\}$$

$$K = \{i \in I : d_i \leq x_i \leq f_i\}$$

The following two exclusive cases are considered.

4. Target Optimal Allocation Cardinality Match: Considering the case

$$|K| \leq p$$

one defines $\tilde{x}$ by

$$\tilde{x}_i = \begin{cases} i \in K : x_i \\ i \in J : 0 \end{cases}$$

5. Target Optimal Allocation Cardinality Excess: Consider the case

$$|K| > p$$

Let

$$L \subset K$$

such that

$$|L| = p$$

and

$$\min_{i \in L} x_i \geq \max_{i \notin L} x_i$$

The subset L can be obtained by a s straight forward greedy algorithm which selects the index of the maximum value as a new candidate.

6. Candidate Subset Selection by Greedy Algorithm: One defines $\tilde{x}$ by

$$\tilde{x}_i = \begin{cases} i \in K : x_i \\ i \in J : 0 \end{cases}$$

Also define

$$\Delta = \sum_{i \in I} (x_i - \tilde{x}_i)$$

7. Allocation Gap Quadratic Convex Problem: If

$$\Delta > 0$$

the following problem is considered:

$$\min_{z} \left(\frac{1}{2} z^T V z - \mu r^T z + p h^T z \mid e^T z = \Delta, 0 \leq z \leq f - \tilde{x} \right)$$

8. Iterative Reduction of the Departure Metric: If z is a solution of the above optimization, then one considers $\tilde{x} + z$ as the approximation of

$$\min_{x,y} \left(\frac{1}{2} x^T V x - \mu r^T x + h^T y \mid e^T x = 1, Dy \leq x \leq Fy, e^T y \leq p, y \in \{0,1\}^n \right)$$

One then applies the process repeatedly until

$$\Delta = 0$$

or there is no more improvement. In the latter case, one finds the component

$$i \in K$$

(or L) such that

$$d_j \leq \tilde{x}_j + \Delta \leq f_j$$

which gives the minimum value for

$$\min\left(\frac{1}{2}[\tilde{x}_j + \Delta]^T V[\tilde{x}_j + \Delta] - \mu r^T[\tilde{x}_j + \Delta] + ph^T[\tilde{x}_j + \Delta]\right)$$

The solution is then obtained by $\tilde{x}_j$ to $\tilde{x}_j + \Delta$.

9. Allocation Fraction and Asset Index: At the end of the iterative process the solution x is retrieved from $\tilde{x}$. After computing the vector $\tilde{x}$ as described above, y is defined by

$$y_j = \begin{cases} 0 & \tilde{x}_j = 0 \\ 1 & \tilde{x}_j \neq 0 \end{cases}$$

Thus, it is clear that $(\tilde{x}, y)$ is a feasible approximation to

$$\min_{x,y}\left(\frac{1}{2}x^T V x - \mu r^T x + h^T y \mid e^T x = 1, Dy \leq x \leq Fy, e^T y \leq p, y \in \{0,1\}^n\right)$$

10. Applying the Heuristic to Fixed Bounds: The heuristic can also be applied when some components have a fixed value. In that case, one just needs to set the corresponding components of d and f to the same fixed value.
11. Performance of the Heuristic Greedy Algorithm: The table below displays the performance of the heuristic with a 2.5 GHz processor. As can be seen, the running time is globally acceptable. Moreover, it appears that the performance is better with a

larger value of p. The intuitive reason for this is that selecting more items – the non-NULL component of x – imparts a more significant reduction of the gap between current solution and what can be expected at further steps.

12. Heuristic Greedy Algorithm Performance Table: Source: Tadonki and Vial (2004)

n	p	Selected	Iterations	Time (seconds)
50	5	5	15	2.80
50	10	7	4	0.78
100	10	10	10	2.07
100	20	12	6	1.20
200	10	10	27	8.16
200	30	18	14	4.51
300	10	10	31	15.21
300	25	23	14	5.97
400	10	10	23	14.20
400	30	30	3	3.16
500	10	10	22	23.36
500	50	27	11	8.78

Cutting Planes Algorithm and PROXACCPM – Concept and Tool

1. NDO Oracle and Cutting Plane: The non-differentiable optimization (NDO) is based on the notions of *Oracle* and *Cutting Plane* (Kelley (1960)).
2. Hyperplane Supports for Polyhedral Domains: The main idea is to use hyperplanes that are linear supports of the objective function, to build a workable, polyhedral approximation of the function. This approximation is generally refined to approach an optimal solution.

3. Localization Set and the Query Point: The domain of the polyhedral approximation is called the *localization set*. The mechanism that supports the linear supports of the objective function at the so-called *query points* is fully problem dependent.
4. Generation of the Cutting Plane: In an NDO algorithm, the previous mechanism is assimilated into a *black box* and is usually named the *Oracle*. The Oracle output is one or more *cutting planes*, and the problem that is built around the generated cutting planes is called the *master problem*.
5. Steps involved in the Cutting Plane Algorithm: The cutting plane algorithm can be summarized as follows:
    a. Get an initial upper bound and a lower bound for the optimal solution.
    b. Find a point in the current localization set and the associated lower bound.
    c. Query the Oracle at the generated point. The Oracle generates one or more cuts for both the objective function and the sub-gradient. If the point is feasible, then the corresponding function value is used as a potential upper bound for the optimal value. If the point is infeasible, the cut is added as a domain constraint.
    d. Update the upper and the lower bounds and add the new cuts.
6. The Iteration Convergence/Termination Criterion: These steps are repeated until the gap between the upper bound and the lower bound falls below a prescribed optimality tolerance.

PROXACCPM Performance on the Generic Problem

1. PROXACCPM – Analytic Cutting Plane Implementation: PROXACCPM (Goffin, Haurie, and Vial (1992), Gondzio, du Merle, Sarkissian, and Vial (1996)) is a convex optimization solver based on the analytic center cutting plane algorithm. The notion of analytic center (Sonnevend (1986)) is the key of the query point selection procedure.

2. Convex Optimization under Cardinality Constraint: Recall that the problem solved is the following relaxation:

$$\min_{x,y} \left(\frac{1}{2} x^T V x - \mu r^T x + h^T y \mid e^T x = 1, Dy \leq x \leq Fy, e^T y \leq p, 0 \leq y \leq 1 \right)$$

3. PROXACCPM Objective Function and Constraint Set: The PROXACCPM objective is defined by

$$\mathcal{F}(y) = \min_{x,y} \left(\frac{1}{2} x^T V x - \mu r^T x + h^T y \mid e^T x = 1, Dy \leq x \leq Fy \right)$$

4. Query Value and the Dual: The value of $\mathcal{F}$ is obtained by solving the corresponding quadratic programming problem, which also adds dual variables λ_l and λ_u for the inequalities

$$x \geq Dy$$

and

$$x \leq Fy$$

respectively.

5. Sub-gradient at the Point y: One sub-gradient of $\mathcal{F}$ at the point y is given by

$$\mathcal{G}(y) = h + \lambda_l \otimes d - \lambda_u \otimes f$$

where $\otimes$ is a point-to-point product of two vectors.

6. Performance on Sub-problem Solving: Source: Tadonki and Vial (2004).

n	p	Iterations	Total (seconds)	Method (seconds)	Oracle (seconds)
50	5	56	8.54	0.63	7.81
50	10	58	8.76	0.67	8.03
100	10	88	15.55	1.28	14.09
100	20	90	15.65	1.30	14.20
200	10	93	27.04	1.71	25.12
200	30	88	25.04	1.56	23.25
300	10	86	39.24	1.93	37.10
300	25	89	40.26	2.11	37.99
400	10	96	71.42	2.93	68.07
400	30	90	65.56	2.50	62.74
500	10	89	98.52	3.61	94.52
500	50	144	165.98	8.75	156.45

Chvatal-Gomory Cuts and Variants

1. Types of Cutting Planes used: The use of linear inequalities as cutting planes is now well-established and proven to be a powerful approach. In this direction, a number of authors have proposed several types of cuts (Nemhauser and Wolsey (1998), Letchford and Lodi (2002)) using different techniques – polyhedra, algorithmic, algebraic, etc.
2. Chvatal-Gomory Family of Cuts: This section considers the Chvatal-Gomory cuts (Chvatal (1973)), derived from the original fractional Gomory cuts (Gomory (1958)) where the use of the slack variables is dropped. In addition, the strengthening procedure proposed by Letchford and Lodi (2002) to improve the impact of cuts is also considered.

Chvatal-Gomory Cuts

1. Standard Integer Linear Program Setup: This section considers the following standard integer linear program (ILP) setup (Letchford and Lodi (2002)):

$$\min\{c^T x : Ax \leq b, x \in \mathbb{Z}_+^n\}$$

where

$$A \in \mathbb{Z}^{m \times n}$$

$$b \in \mathbb{Z}^m$$

and

$$c \in \mathbb{R}_+^n$$

2. Polyhedra Associated with the ILP: This ILP is associated with the two polyhedra

$$P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$$

and

$$P_I = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$$

which represent the feasible region of the relaxation and the corresponding convex hull, respectively.

3. Fractional Part of $\mathbb{R}$:For any $\alpha \in \mathbb{R}$ the *fractional part* of α is denoted by

$$f(\alpha) = \alpha - \lfloor \alpha \rfloor$$

where $\lfloor \alpha \rfloor$ is the lower integer part of α. This is used to define the *Chvatal-Gomory* cut as follows.

4. Definition of the Chvatal-Gomory Cut: Given an ILP of the form above, and any vector

$$\lambda \in \mathbb{R}_+^m$$

the inequality

$$\lfloor \lambda^T A \rfloor x \leq \lfloor \lambda^T b \rfloor$$

is valid for P_I (Chvatal (1973)). Some variants of the Chvatal-Gomory cut are shown below.

5. Strengthening of the Chvatal-Gomory Cut: If

$$f(\lambda^T b) < \frac{1}{2}$$

and t is positive integer such that

$$\frac{1}{2} \leq t f(\lambda^T b) < 1$$

then replacing λ by $f(\lambda t)$ yields another stronger or equivalent CG cut (Chvatal (1973)).

6. Cut Strengthening - Definitions and Notations: The following notations are used to state the next variant:

$$N = \{1, \cdots, n\}$$

$$a_0 = \lambda^T b$$

and for

$$i \in N$$

$$a_i = \lambda^T (Ae_i)$$

where a_i is the i^{th} column vector of the canonical basis of $\mathbb{R}^n$.

7. Alternate Expression for CG Cut: The valid inequality

$$\sum_{i \in N} (\lambda^T A) x_i \leq \lambda^T b$$

can be written as

$$\sum_{i \in N} a_i x_i \leq a_0$$

and the CG cut

$$\lfloor \lambda^T A \rfloor x \leq \lfloor \lambda^T b \rfloor$$

can also be written as

$$\sum_{i \in N} \lfloor a_i \rfloor x_i \leq \lfloor a_0 \rfloor$$

8. First Burdet and Johnson Strengthener: The inequality

$$\sum_{i \in N} \left[\lfloor a_i \rfloor + \max \left\{ 0, \frac{f(a_i) - f(a_0)}{1 - f(a_0)} \right\} \right] x_i \leq \lfloor a_0 \rfloor$$

is valid and dominates the CG cut

$$\lfloor \lambda^T A \rfloor x \leq \lfloor \lambda^T b \rfloor$$

(Burdet and Johnson (1977)). Thus, one can consider $f(\lambda t)$ instead of λ and then obtain the strengthened inequality of the above form.

9. Second Burdet and Johnson Strengthener: Let g be any non-decreasing super-additive function such that

$$g(0) = 0$$

The cut

$$\sum_{i \in N} g(a_i) x_i \leq g(a_0)$$

is satisfied by all non-negative integer solutions to

$$\sum_{i \in N} a_i x_i \leq a_0$$

10. Letchford and Lodi Cut Strengthener: The cut proposed by Letchford and Lodi (2002) can stated as follows. Consider the inequality

$$\sum_{i \in N} a_i x_i \leq a_0$$

Suppose that

$$f(a_0) > 0$$

and let

$$k \geq 1$$

be a unique integer such that

$$\frac{1}{k+1} \leq f(a_0) < \frac{1}{k}$$

The set

$$N = \{1, \cdots, n\}$$

is partitioned into $N_0, \cdots, N_k$ as follows. Let

$$N_0 = \{i \in N : f(a_i) \leq f(a_0)\}$$

and, for

$$p = 1, \cdots, k$$

let

$$N_p = \left\{ i \in N : f(a_0) + (p-1)\frac{1 - f(a_0)}{k} \leq f(a_0) + p\frac{1 - f(a_0)}{k} \right\}$$

The inequality

$$\sum_{i\in N_0}(k+1)\lfloor a_i\rfloor x_i+\sum_{p=1}^{k}\sum_{i\in N_p}[(k+1)\lfloor a_i\rfloor+p]x_i\le(k+1)\lfloor a_0\rfloor$$

is valid for P_I and dominates the CG cut

$$\sum_{i\in N}\lfloor a_i\rfloor x_i\le\lfloor a_0\rfloor$$

(Letchford and Lodi (2002)).

Deriving the Cuts for the Setup

1. Assuming Rational D and F: For their problem, Tadonki and Vial (2004) assume that D and F are rational from a computation point of view. Thus, there are two integers α and β such that αD and βF are rational.
2. Chvatal Gomory Cut Coefficients: Thus, the constraint polyhedron is

$$P=\{y\in[0,1]^n: e^Ty\le p;\ (\alpha d^T)y\le\alpha;\ (-\beta f^T)y\le-\beta\}$$

Tadonki and Vial (2004) consider the following coefficients:

$$\lambda_1=1$$

$$\lambda_2\in\left\{\frac{1}{u},u\in(\alpha d_i,d_i>0,i=1,\cdots,n)\right\}$$

and

$$\lambda_3 \in \left\{\frac{1}{v}, v \in (\beta f_i, f_i > 0, i = 1, \cdots, n)\right\}$$

3. Computing the CG Cut Variants: For each of these coefficients, Tadonki and Vial (2004) compute the corresponding CG cuts and the variants. It may be noted that, since the variables are binary, the cut

$$u^T x \leq b$$

should satisfy

$$u^T e < b$$

in order to be useful.

Branching Rule and Node Selection

1. Closest Integer Based Component Branching: Tadonki and Vial (2004) use a natural branching rule based on the selection of the component whose value is closest to an integer binary value.
2. Extensively Pruned Child Node Selection: For node selection, two ways are considered. First, since there is a cardinality constraint, it follows that by giving priority to the branch corresponding to the value 1 one gets pruned nodes much earlier.
3. Child Nodes with Value 1: Thus, each time a branching component is selected, the child node with a value 1 at the reference component is explored. This approach leads to a deep exploration of the branching tree.
4. Best Relaxation Value Node Selection: Also considered was the common approach based on the selection of the node with the best relaxation value.

Computational Results

1. Setup of the Computational Runs: This section reports the results of Tadonki and Vial (2004) on a 2.5 GHz processor with 2 GB of memory, in a MATLAB environment. The quadratic programming sb-problem was solved by MOSEK (2019).
2. Random vs Beasley Data Sets: The first group of data has been generated randomly using a normal-distribution based generator – the *rand* routine of MATLAB. The second group is a collection of five data sets drawn from Hang Seng, DAX, FTSE, S&P, and NIKKEI indexes (Chang, Meade, Beasley, and Sharaiha (2000)).
3. Global Performance on Randomly Generated Data Sets: Source: Tadonki and Vial (2004)

n	p	Nodes	Time (sec)
30	10	37	267
30	15	14	122
30	20	22	226
50	5	25	155
50	10	160	878
100	10	13	95
100	20	527	2734
200	10	8	167
200	20	715	5639
300	10	74	3200
300	20	287	6300
400	30	33	551
500	50	429	8922

4. Global Performance on Beasley Collection: Source – Tadonki and Vial (2004)

Set	n	p	Nodes	Time (sec)
Hang Seng	31	10	28	14.0
DAX	85	10	3	14.5
FTSE	89	10	3	28.0
S&P	98	10	5	40.8
NIKKEI	225	10	92	2837.5

5. Data Distribution vs. Selection Count: As can be seen in the above tables, the algorithm solves for the portfolio selection problem quiet efficiently. The performance depends both on the data distribution and on the bound on the number of selections $\langle p \rangle$.
6. Impact of the Cardinality Constraint Number: Intuitively, the performance should be better with the lower values of the cardinality bound, since the search space is reduced. However, a larger cardinality bound can also result in a surprisingly good performance because of direct solution coming from the relaxation. Thus, there is no absolute rule that enables the prediction of the performance without taking into account the distribution of the data.
7. Shortcoming of the Simulation Environment: From a technical point of view, Tadonki and Vial (2004) ran a set of MATLAB routines, i.e., the cutting plane solver, the heuristic, and the master branch-and-bound program. It is clear that better timings would have been obtained of they were using a compiled version of the software.
8. Branch Node Selection Rules Impact: Regarding the branching rules, it was observed that selecting the nodes according to the relaxation values was by far the best way.
9. Performance Impact of the Heuristic: On each instance, the number of explored nodes affords a computationally affordable processing. This is particularly due to the power of the heuristic, as can be seen in the table below.
10. Qualitative Performance of the Heuristic on the Beasley Collection: Source – Tadonki and Vial (2004)

Set	n	p	Optimal	Heuristic
Hang Seng	31	10	-0.0047	-0.0041
DAX	85	10	-0.0064	-0.0056
FTSE	89	10	-0.0049	-0.0042
S&P	98	10	-0.0067	-0.0061
NIKKEI	225	10	-0.0043	-0.0031

Conclusion

1. Portfolio Selection Problem Algorithmic Solution: This chapter presented an algorithm for solving the portfolio selection problem using the analytic cutting center plane method together with a branch and bound scheme.
2. Asset Count and Investment Bounds: It has considered a very general formulation of the problem including bounds on the number of selected assets and their investment limits.
3. Heuristic Estimation and Branch Selection: It has proposed a heuristic that provides an estimation of the solution and the corresponding selection pattern.
4. Computational Speed and Memory Assumption: Computational results show that the algorithm and the related heuristic solve the problem efficiently both in terms of computational time and memory space.

References

- Adcock, C. J., and N. Meade (1994): A Simple Algorithm to incorporate Transaction Costs in Quadratic Optimization *European Journal of Operations Research* **79 (1)** 85-94

- Bienstock, D. (1996): Computational Study of a Family of Mixed-Integer Quadratic Programming Problems *Mathematical Programming* **74 (2)** 121-140
- Burdet, C. A., and E. L. Johnson (1977): A Sub-additive Approach to Solve Linear Integer Programs *Annals of Discrete Mathematics* **1** 117-143
- Constantinides, G. M., and A. G. Malliaris (1995): Portfolio Theory *Handbook in Operations Research and Management Science* **9** 1-30
- Chang, T., J., N. Meade, J. E. Beasley, and Y. M. Sharaiha (2000): Heuristics for Cardinality Constrained Portfolio Optimization *Computers and Operations Research* **27 (13)** 1271-1302
- Chvatal, V. (1973): Edmonds Polytopes in a Hierarchy of Combinatorial Problems *Discrete Mathematics* **4 (4)** 305-337
- CPLEX (2019): https://en.wikipedia.org/wiki/CPLEX
- Garey, M. R., and D. S. Johnson (1979): *Computer and Intractability – A Guide to the Theory of NP-Completeness* **W H Freeman** New York
- Goffin, J. L. Haurie, A. Laurie, and J. P. Vial (1992): Decomposition and Non-differentiable Optimization with a Projective Algorithm *Management Science* **38 (2)** 284-302
- Gomory, R. E. (1958): Outline of an Algorithm for Integer Solutions to Linear Programs *Bulletin of the American Mathematical Society* **64 (5)** 275-278
- Gondzio, J., O. du Merle, R. Sarkissian, and J. P. Vial (1996): ACCPM – A library for Convex Optimization Based on Analytic Center Cutting Plane Method *European Journal of Operation Research* **94 (1)** 206-211
- Hakansson, N. H. (1971): Multi-period Mean-variance Analysis: Toward a General Theory of Portfolio Choice *Journal of Finance* **26 (4)** 857-884
- Jobst, N. J., M. D. Horniman, C. A. Lucas, and G. Mitra (2001): Computational Aspects of Alternative Portfolio Selection Models in the Presence of Discrete Asset Choice Constraints *Quantitative Finance* **1 (5)** 1-13
- Kelley, J. E. (1960): The Cutting Plane Method for Solving Convex Problems *Journal for the Society of the Industrial and Applied Mathematics* **8 (4)** 703-712

- Lee, E. K., and J. E. Mitchell (1997): Computational Experience of an Interior-Point SQP Algorithm in a Parallel Branch-and-Bound Framework, in: *High Performance Optimization, Applied Optimization* (*Frenk, H., K. Roos, T. Terlaky, S. Zhang editors*) **33 Springer** Boston, MA
- Letchford, A. N. and A. Lodi (2002): Strengthening Chvatal-Gomory Cuts and Gomory Fractional Cuts *Operations Research Letters* **30 (2)** 74-82
- Li, D., and W. L. Ng (2000): Optimal Dynamic Portfolio Selection: Multi-period Mean-variance Formulation *Mathematical Finance* **10 (3)** 387-406
- Mansini, R., and M. G. Speranza (1999): Heuristic Algorithms for the Portfolio Selection Problem with Minimum Transaction Lots *European Journal of Operations Research* **114 (2)** 219-233
- Markowitz, H. (1952): Portfolio Selection *Journal of Finance* **7 (1)** 77-91
- Markowitz, H. (1959): *Portfolio Selection: Efficient Diversification of Investments* **John Wiley and Sons** New York
- MOSEK (2019): https://en.wikipedia.org/wiki/MOSEK
- Mossin, C. (1968): Optimal Multi-period Portfolio Policies *Journal of Business* **41 (2)** 215-229
- Nemhauser, G. L., and L. A. Wolsey (1998): *Integer and Combinatorial Optimization* **John Wiley and Sons** New York
- Samuelson, P. A. (1969): Life-time Portfolio Selection by Dynamic Stochastic Programming *Review of Economics and Statistics* **51 (3)** 239-246
- Sonnevend, G. (1986): An *Analytic Center* for Polyhedrons and New Classes of Global Algorithms for Linear (Smooth, Convex) Programming, in: *System Modeling and Optimization, Lecture Notes in Control and Information Sciences, Prekopa, A., B. Szelezsaan, and B. Strazicky editors* **84 Springer** Berlin
- Tadonki, C., and J. P. Vial (2004): Portfolio Selection with Cardinality and Bound Constraints
- Young, M. R. (1998): A Minimax Portfolio Selection Rule with Linear Programming Solution *Management Science* **44 (5)** 673-683

- Zariphopoulou, T. (1992): Investment-Consumption Models with Transaction fees and Markov-Chain Parameters *SIAM Journal of Control and Optimization* **30 (3)** 613-636
- Zhou, X. Y., and D. Li (2000): Continuous-Time Mean-variance Portfolio Selection: A Stochastic LQ Framework *Applied Mathematics and Optimization* **42 (1)** 19-33

Simplex Algorithm

Introduction

1. Objective of the Simplex Algorithm: Dantzig's simplex algorithm – or the simplex method – is a popular algorithm for linear programming (Murty (1983), Wikipedia (2020)).
2. Use of Simplicial Cone Polytopes: The name of the algorithm is derived from the concept of the simplex and was suggested by T. S. Motzkin (Murty (1983)). Simplices are not actually used in the method, but one interpretation of it is that it operates on simplicial *cones*, and these become proper simplices with an additional constraint (Murty (1983), Strang (1987), Stone and Tovey (1991a, 1991b)). The simplicial cones in question are the corners, i.e., the neighborhoods of the vertices, of a geometric object called a polytope. The shape of this polytope is defined by the constraints applied to the objective function.

Overview

1. Canonical Form of the Simplex Setup: The simplex algorithm operates on linear programs in the canonical form

$$\min \boldsymbol{c^T x}$$

subject to

$$\boldsymbol{Ax} \leq \boldsymbol{b}$$

and

$$\boldsymbol{x} \geq \boldsymbol{0}$$

with

$$\boldsymbol{x} = (x_1, \cdots, x_n)$$

the variables of the problem;

$$\boldsymbol{c} = (c_1, \cdots, c_n)$$

the coefficients of the objective function; $\boldsymbol{A}$ a $n \times p$ matrix, and

$$\boldsymbol{b} = (b_1, \cdots, b_p)$$

non-negative constraints, i.e.,

$$b_j \geq 0 \; \forall \, j$$

There is a straightforward process to convert any linear program into the standard form, so using this form of linear program results in no loss of generality.

2. Feasible Extreme Point in the Polytope: In geometric terms, the feasible region defined by all values of $\boldsymbol{x}$ such that

$$\boldsymbol{Ax} \leq \boldsymbol{b}$$

and

$$x_i \geq 0 \; \forall \, i$$

is a possibly unbounded convex polytope. An extreme point or vertex of this polytope is known as *basic feasible solution* (BFS).

3. Function Optimum lies on the Polytope: It can be shown that for a linear program in the standard form, if the objective function has a maximum value in the feasible

region, then it has this value in at least one of the extreme points (Murty (1983)). This in itself reducers the problem to a finite computation since there are a finite number of extreme points, but the number of extreme points is unmanageably large in all but the smallest linear programs (Murthy (1983)).

4. Guaranteed Navigation toward the Optimum: It can also be shown that, if an extreme point is not a maximum point of the objective function, then there is an edge containing the point so that the objective function is strictly increasing on the edge moving away from the point (Murthy (1983)).
5. Impact of Finite, Unbounded Edges: If the edge is finite, then the edge connects to another extreme point where the objective function has a greater value, otherwise the objective function is unbounded above on the edge and the linear program has no solution.
6. Progressive Navigation towards the Optimum: The simplex algorithm applies this insight by walking along the edges of the polytope to extreme points with greater and greater objective values. This continues until the maximum value is reached, or an unbounded edge is visited, concluding that the problem has no solution.
7. Guaranteed Termination of the Algorithm: The algorithm always terminates because the number of vertices in the polytope is finite; moreover, since the jump between the vertices are always in the same direction – that of the objective function optimum, it is hoped that the number of vertices visited will be small (Murty (1983)).
8. Locating the Basic Feasible Starting Point: The solution to a linear program is accomplished in two steps. In the first step, known as Phase I, a starting extreme point is found. Depending upon the nature of the program this may be trivial, but in general it can be solved by applying the simplex algorithm to a modified version of the original program. The possible results of Phase I are that either a basic feasible solution is found or that the feasible region is empty. In the latter case the program is called *infeasible*.
9. Progressive Traversal along the Polytope: In the second step, called Phase II, the simplex algorithm is applied using the basic feasible solution found in Phase I as the starting point. The possible results from Phase II are either an optimum basic feasible

solution, or an infinite edge on which the objective function is unbounded above (Nering and Tucker (1993), Dantzig and Thapa (1997), van der Bei (2008)).

Standard Form

1. Step #1 – Bound Variable Gap: The transformation of the linear program to one in standard form may be accomplished as follows (Murty (1983)). First, for each variable with a lower bound other than zero, a new variable that represents the difference between the variable and its bound is introduced. The original variable can then be eliminated by substitution. For example, given the constraint

$$x_1 \geq 5$$

a new variable y_1 is introduced with

$$y_1 = x_1 - 5$$

$$x_1 = y_1 + 5$$

The second equation may be used to eliminate x_1 from the linear program. In this way, all lower bound constraints may be changed to non-negativity constraints.

2. Constraint Gap - Surplus/Slack Variables: Second, for each remaining inequality constraint, a new variable called a *slack* variable is introduced to change the constraint to an equality constraint. This variable represents the difference between the two sides of the inequality and is assumed to be non-negative. For example, the inequalities

$$x_2 + 2x_3 \leq 3$$

$$-x_4 + 3x_5 \geq 2$$

are replaced with

$$x_2 + 2x_3 + s_1 = 0$$

$$-x_4 + 3x_5 - s_2 = 2$$

$$s_1, s_2 \geq 0$$

It is much easier to perform algebraic manipulations on inequalities of this form. In inequalities where $\geq$ appears such as the second one, some authors refer to the variable as a *surplus* variable.

3. Elimination of the Unrestricted Variable: Third, each unrestricted variable is eliminated from the program. This can be done in two ways; one is solving for the variables in one of the equations in which it appears and then eliminating the variable by substitution. The other is to replace the variable with the difference of two restricted variables. For example, if z_1 is unrestricted then write

$$z_1 = {z_1}^+ - {z_1}^-$$

and

$${z_1}^+, {z_1}^- \geq 0$$

This equation may be used to eliminate z_1 from the linear program.

4. Re-cast to the Canonical Form: When the process is complete, the feasible region will be in the form

$$\boldsymbol{Ax = b}$$

and

$$x_i \geq 0 \; \forall \; i$$

It is also useful to note that the rank of $\boldsymbol{A}$ is the number of rows. This results in no loss of generality, since otherwise either the system

$$\boldsymbol{Ax} = \boldsymbol{b}$$

has redundant equations which can be dropped, or the system is inconsistent and the linear program has no solution (Murty (1983)).

Simplex Tableau

1. Standard Form of the Simplex Tableau: A linear program in standard form can be represented as a *tableau* of the form $\begin{bmatrix} 1 & -\boldsymbol{c^T} & 0 \\ 0 & \boldsymbol{A} & \boldsymbol{b} \end{bmatrix}$
2. Rows Defining Objectives and Constraints: The first row defines the objective function and the remaining rows define the constraints. The zero in the first column represents the zero vector of the same dimension as vector $\boldsymbol{b}$ - though different authors use different conventions as to the precise layout.
3. Transforming $\boldsymbol{A}$ to Identity Matrix: If the columns of $\boldsymbol{A}$ can be arranged so that it contains the identity matrix of order p - the number of rows of $\boldsymbol{A}$ - then the tableau is said to be in the *canonical form* (Murty (1983)). The variables corresponding to the

columns of the identity matrix are called the *basic variables* while the remaining variables are called *non-basic* or *free variables*.

4. Extracting the Basic Feasible Solution: If the values of the non-basic variables are set to zero, then the values of the basic variables are easily obtained as entries in $\boldsymbol{b}$ and this solution is a basic feasible solution. The algebraic interpretation here is that the coefficients of linear equation represented by each row are either 0, 1, or some other number. Each row will have one column with a value 1, $p - 1$ columns with coefficient 0, and the remaining columns with some other coefficients – these other variables represent the non-basic variables. By setting the values of the non-basic variables to 0, it is ensured that in each row the value represented by a 1 in its column is equal to its $\boldsymbol{b}$ value at that row.
5. Inverse of the Non-zero Variables Matrix: Conversely, given a basic feasible solution, the columns corresponding to the non-zero variables can be expanded to a non-singular matrix. If the corresponding tableau is multiplied by the inverse of this matrix, then the result is a tableau in a canonical form (Murty (1983)).
6. Split-up of the Canonical Tableau: Let $\begin{bmatrix} 1 & -\boldsymbol{c_B}^T & -\boldsymbol{c_D}^T & 0 \\ 0 & \boldsymbol{I} & \boldsymbol{D} & \boldsymbol{b} \end{bmatrix}$ be a tableau in the canonical form. Additional row operations can be applied to remove the coefficients $\boldsymbol{c_B}^T$ from the objective function.
7. Pricing Out Relative Cost Coefficients: The above process is called *pricing out* and results in a canonical tableau $\begin{bmatrix} 1 & 0 & -\tilde{\boldsymbol{c}}_{\boldsymbol{D}}^T & z_B \\ 0 & \boldsymbol{I} & \boldsymbol{D} & \boldsymbol{b} \end{bmatrix}$ where z_B is the value of the objective function at the corresponding basic feasible solution. The updated coefficients, also called the *relative cost coefficients*, are the rates of change of the objective function with respect to the non-basic variables (Nering and Tucker (1993)).

Pivot Operations

1. Stepping across Adjacent Feasible Solutions: The geometric operation of moving from one basic feasible solution to an adjacent basic feasible solution is implemented as a *pivot operation*.
2. Making the Pivot Element Coefficient Unity: First, a non-zero *pivot element* is selected in a basic column. The row containing this element is multiplied by its reciprocal to change this element to 1, and then multiples of the row are added to the other rows to change the other entries in the column to 0. The result is that, if the pivot element is in the row r, then the column becomes the r^{th} column of the identity matrix.
3. Entering and Leaving Basic Variables: The variable for this column now is a basic variable, replacing the variable which corresponded to the r^{th} column pf the identity matrix before the operation. In effect, the variable corresponding to the pivot column enters the set of basic variables and is called the *entering variable*, and the variable being replaced leaves the set of basic variables and is called the *leaving variable*. The tableau is still in the canonical form, but with the set of basic variables changed by one element (Nering and Tucker (1993), Dantzig and Thapa (1997)).

The Algorithm

Let a linear program be given by a canonical tableau. The simplex algorithm proceeds by performing successive pivot operations each of gives a improved basic solution, the choice of the pivot element at each step is largely determined by the requirement that this pivot improves the solution.

Entering Variable Selection

1. Increase/Decrease of the Objective Function: Since the entering variable will, in general, increase from 0 to a positive number, the value of objective function will decrease if the derivative of the objective function with respect to this variable is negative. Equivalently, the value of the objective function is decreased if the pivot column is selected so that the corresponding entry in the objective row of the tableau is positive.
2. Tie-Breaking of Multiple Entering Variables: If there is more than one column such that the entry in the objective row is positive, then the choice of which one to add to the set of basic variables is somewhat arbitrary, and several *entering variable choice rules* (Murty (1983)) such as the Devex algorithm (Harris (1973)) have been developed.
3. Switching Optimum to Minimum/Maximum: By changing the entering variable choice rule so that it selects a column where the entry in the objective row is negative, the algorithm is changed so that it finds the maximum of the objective function rather than the minimum.

Leaving Variable Selection

1. Rationale behind the Pivot Row Selection: Once the pivot column has been selected, the choice of the pivot row is largely determined by the requirement that the resulting

solution be feasible. First, only positive entries in the pivot column are considered since this guarantees that the value of the entering variable will be non-negative.

2. Detecting an Unbounded Objective Function: If there are no positive entries in the pivot column, then the entering variable can take any non-negative value with the solution remaining feasible. In this case the objective function is unbounded from below and there is no minimum.
3. Minimum Ratio Test Based Row Selection: Next, the pivot row must be selected so that all other basic variables remain positive. A calculation shows that this occurs when the resulting value of the entering variable is at a minimum. In other words, if the pivot column is c, then the pivot row is chosen so that b_r/a_{rc} is minimum over all r so that

$$a_{rc} > 0$$

This is called a *minimum ratio test* (Murthy (1983)). If there is more than one row for which the minimum is achieved, then a *dropping variable choice rule* (Murty (1983)) can be used to make a determination.

Example #1

1. Sample Objective and Constraint Set: Consider the linear program:

$$\min Z = -2x - 3y - 4z$$

subject to

$$3x + 2y + z \leq 10$$

$$2x + 5y + 3z \leq 15$$

$$x, y, z \geq 0$$

2. Canonical Tableau using Slack Variables: With the addition of the slack variables s and t, this is represented by the canonical tableau $\begin{bmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$ where columns 5 and 6 represent the basic variables s and t, and the corresponding basic feasible solution is

$$x = y = z = 0$$

$$s = 10$$

$$t = 15$$

3. Choice of Pivot Row/Pivot Operations: Columns 2, 3, and 4 can be selected as pivot columns, so for this example column 4 is selected. The values of z resulting from the above choice of rows 2 and 3 are

$$\frac{10}{1} = 10$$

and

$$\frac{15}{3} = 5$$

respectively. Of these the minimum is 5 so row 3 must be the pivot row. Performing the pivot operation produces $\begin{bmatrix} 3 & -2 & -11 & 0 & 0 & -4 & -60 \\ 0 & 7 & 1 & 0 & 3 & -1 & 15 \\ 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$

4. Adjacent Basic Feasible Solution: Now columns 4 and 5 represent the basic variables z and s, and the corresponding basic feasible solution is

$$x = y = t = 0$$

$$z = 5$$

$$s = 5$$

5. Objective Function with No Positive Entries: For the next step, there are no positive entries in the objective row, and in fact

$$Z = \frac{-60 + 2x + 11y + 4t}{3} = -20 + \frac{2x + 11y + 4t}{3}$$

so the minimum value of Z is -20.

Finding an Initial Canonical Tableau

1. Transformation to the Canonical Form: In general, a linear program will not be given in a canonical form and an equivalent canonical tableau must be found before the simplex algorithm can start. This can be accomplished by the introduction of *artificial variables*. Columns of identity matrix can be added as column vectors for these variables.
2. Case of Negative $\boldsymbol{b}$ Values: If the $\boldsymbol{b}$ value for a constraint equation is negative, the equation must be negated before adding the identity matrix columns. This does not change the set of feasible solutions or the optimal solution, and it ensures that the slack variables will constitute an initial feasible solution.
3. Artificial Variables for Phase I: The new tableau is in the canonical form, but it is not equivalent to the original problem. So, a new objective function, equal to the sum of the artificial variables, is introduced and the simplex algorithm is applied to find the minimum; the modified linear program is called *Phase I* problem (Murty (1983)).

4. Zero Lower Bound Guarantees Termination: The simplex algorithm applied to the Phase I problem must terminate with a minimum value for the new objective function, since, being the sum of non-negative variables, its value is bounded below by zero.
5. Elimination of Variables at Zero Minimum: If the minimum is zero, then the artificial variables can be eliminated from the resulting canonical tableau producing a canonical tableau equivalent to the original problem.
6. Detecting Lack of Feasible Solution: If the minimum is positive, then there is no feasible solution for the Phase I problem where all the artificial variables are zero. This implies that the feasible region for the original problem is empty, and so the original problem has no solution (Nering and Tucker (1993), Dantzig and Thapa (1997), Padberg (1999)).

Example #2

1. Linear Program with Equality Constraints: Consider the linear program

$$\min Z = -2x - 3y - 4z$$

subject to

$$3x + 2y + z = 10$$

$$2x + 5y + 3z = 15$$

$$x, y, z \geq 0$$

2. Non-Canonical Tableau Representation: This is represented by the non-canonical tableau $\begin{bmatrix} 1 & 2 & 2 & 4 & 0 \\ 0 & 3 & 3 & 1 & 10 \\ 0 & 2 & 5 & 3 & 15 \end{bmatrix}$
3. Artificial Variables corresponding to Equality Constraints: Introducing the artificial variables u and v and the objective function

$$W = u + v$$

gives a new tableau $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$ This equation defining the original objective function is retained in anticipation of Phase II.

4. Addition of Equality Constraint Terms: By construction u and v are both non-basic variables since they are part of the critical identity matrix. However, the objective function W currently assumes that u and v are both 0. In order to adjust the objective function to be the correct value

$$u = 10$$

and

$$v = 15$$

the third and the fourth rows are added to the first row giving

$$\begin{bmatrix} 1 & 0 & 5 & 7 & 4 & 0 & 0 & 25 \\ 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 1 & 0 & 10 \\ 0 & 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$$

5. Performing the First Pivot Operations: Selecting column 5 as a pivot column makes the prior row to be row 4, and the updated tableau is

$$\begin{bmatrix} 3 & 0 & 7 & 1 & 0 & 0 & -4 & 15 \\ 0 & 3 & -2 & -11 & 0 & 0 & -4 & -60 \\ 0 & 0 & 7 & 1 & 0 & 3 & -1 & 15 \\ 0 & 0 & 2 & 5 & 3 & 0 & 1 & 15 \end{bmatrix}$$

6. Performing the Second Pivot Operations: Column 3 is selected as the next pivot column, for which row 3 must be the pivot, and one gets

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 7 & 0 & -25 & 0 & 2 & -10 & -130 \\ 0 & 0 & 7 & 1 & 0 & 3 & -1 & 15 \\ 0 & 0 & 0 & 11 & 7 & 2 & 3 & 25 \end{bmatrix}$$

7. Canonical Tableau after dropping Artificial Variables: The artificial variables are now set to zero, and they may be dropped giving a canonical tableau equivalent to the original problem: $\begin{bmatrix} 7 & 0 & -25 & 0 & -130 \\ 0 & 7 & 1 & 0 & 15 \\ 0 & 0 & 11 & 7 & 25 \end{bmatrix}$ Fortunately, this is already optimal and the optimum value for the original linear program is $-\frac{130}{7}$.

Advanced Topics – Implementation

1. Shortcomings of the Standard Simplex Algorithm: The tableau form used above to describe the algorithm lends itself to an immediate implementation in which the tableau is maintained as a rectangular $(m + 1)$ by $(m + 1 + n)$ array. It is straightforward to avoid storing m explicit columns of the identity matrix that will occur within the tableau by virtue of $\boldsymbol{b}$ being a subset of the columns of $[\boldsymbol{A}, \boldsymbol{I}]$. This implementation is referred to as the *standard simple algorithm*. The storage and overhead requirements are such that the standard simplex method is a prohibitively expensive approach to solving large linear programming problems.
2. Minimizing the Step-wise Data Requirements: In each simplex iteration, the only data required are the first row of the tableau, the pivotal column of the tableau corresponding to the entering variable, and the right-hand side. The latter can be updated using the pivotal column and the first row of the tableau can be updated using the pivotal column corresponding to the leaving variable.
3. Motivation behind the Revised Simplex Algorithm: Both the pivotal column and the pivotal row may be computed directly using the solutions of the linear systems of equations involving the matrix $\boldsymbol{b}$ and a matrix-vector product using $\boldsymbol{A}$. These observations motivate the *revised simplex algorithm* for which implementations are distinguished by their invertible representations of $\boldsymbol{b}$.
4. Exploiting the Sparseness of $\boldsymbol{A}/\boldsymbol{b}$ Matrices: In large linear programs $\boldsymbol{A}$ is typically a large sparse matrix, and when the resulting sparsity of $\boldsymbol{b}$ is exploited when maintaining its invertible representation, the resulting simplex algorithm is much more efficient than the standard simplex. Commercial solvers are based on the revised simplex algorithm (Maros and Mitra (1996), Padberg (1999), Alevras and Padberg (2001), Dantzig and Thapa (2003), Maros (2003)).

Degeneracy and Stalling

1. Convergence using Strictly Positive Basic Variables: If the values of all the basic variables are strictly positive, then a pivot must result in the improvement of the objective value. When this is always the case no set of basic variables occurs twice and the simplex algorithm terminates after a finite number of steps.
2. Origins of Degeneracy and Stalling: Basic feasible solutions where at least one set of basic variables is zero are called *degenerate* and may result in pivots for which there is no improvement in the objective value. In this case, there is no actual change in the solution, but only a change in the basic set of variables. When several such pivots occur in succession, there is no improvement; in large industrial applications, degeneracy is common and such *stalling* is noticeable.
3. Cycling of the Basic Variables: Worse than stalling is the possibility that the same set of basic variables occurs twice, in which case, determining the pivot rules of the simplex algorithm will produce an infinite loop, or *cycle*. While degeneracy is the rule in practice and stalling is common, cycling is rare in practice. A discussion of an example of practical cycling occurs in Padberg (1999).
4. Rules/Algorithms to Prevent Cycling: Bland's rule prevents cycling and thus guarantees that the simplex algorithm always terminates (Bland (1977), Murty (1983), Padberg (1999)). Another pivoting algorithm – criss-cross algorithm – never cycles on linear programs. Specifically, there are abstract optimization problems, called *oriented matroid* programs, on which the Bland's rule cycles incorrectly, while the criss-cross algorithm terminates correctly.
5. History Based Stalling/Cycling Prevention: History based pivot rules such as Zadeh's rule or Cunningham's rule also try to circumvent the issue of stalling and cycling by keeping track of how often particular variables are being used, and then favor such variables that have been used least often.

Efficiency

1. Worst-case Complexity of Dantzig's Simplex: The simplex method is remarkably efficient in practice and is a great improvement over earlier methods such as Fourier-Motzkin elimination. However, Klee and Minty (1972) gave an example of the Klee-Minty cube, showing that the worst-case complexity of the simplex method as formulated by Dantzig is exponential time.
2. Simplex Variants Worst-Case Complexity: Since then, for almost every variation of the method, it has been shown that there is a family of linear programs on which it performs badly. It is an open question whether there is a version with polynomial time, although sub-exponential pivot rules are known (Hansen and Zwick (2015)).
3. NP-mighty Variant of Simplex: It has been proved that a particular variant of the simplex method is NP-mighty, i.e., it can be used to solve, with polynomial overhead, any problem in NP implicitly during the algorithm's execution.
4. Determining Entering Variables and Iteration Count: Moreover, determining whether a given variable even enters the basis during the algorithm's execution on a given input, and determining the number of iterations required for solving a given problem are both NP-hard (Disser and Skutella (2018)). Computing the output of certain other pivot rules is already known to be PSPACE-complete (Adler, Christos, and Rubinstein (2014), Fearnly and Savani (2015)).
5. Developing Alternate Measures o Complexity: Analyzing and quantifying the observation that the simplex is efficient in practice despite its exponential worst-case complexity has led to the development of other measures of complexity.
6. Polynomial-Time Average-Case Complexity: The simplex algorithm has polynomial-time average-case complexity under various probability distributions, with the precise average-case performance of the simplex algorithm depending upon the choice of the

probability distribution for the random matrices (Borgwardt (1987), Schrijver (1998)).

7. Baire Category Theory Based Approaches: Another approach to studying *typical phenomena* uses the Baire category theory from general topology, and to show that, topologically, *most* matrices can be solved by the simplex algorithm in a polynomial number of steps.
8. Smoothened Analysis of Simplex Performance: Another method to analyze the performance of the simplex algorithm studies the behavior of worst-case scenarios under small perturbations, i.e., are the worst-case scenarios stable under a small change in the sense of structural stability, or do they become tractable? This area of research, called *smoothened analysis*, was specifically introduced to study the simplex method. As demonstrated by Spielman and Teng (2001), the running time of the simplex method on input with noise is polynomial on the number of variables and the magnitude of the perturbations.

Other Algorithms

Other algorithms for solving linear programming problems exist. Another basis-exchange pivoting algorithm is the criss-cross algorithm (Terlaky and Zhang (1993), Fukuda and Terlaky (1997)). There are polynomial-time algorithms for linear programming that use interior-point methods; these include Khachiyan's ellipsoidal algorithm, Karmarkar's projective algorithm, and path following algorithms (van der Bei (2008)).

Linear-Fractional Programming

Linear-fractional programming (LFP) is a generalization of linear programming (LP). In LP the objective function is a linear function, while the objective function of a linear-fractional program is a ratio of two linear functions. In other words, the linear program is a fractional linear program in which the denominator is the constant function having the value on everywhere. A linear-fractional program can be solved by a variant of the simplex algorithm (Murty (1983), Craven (1988), Mathis and Mathis (1995), Kruk and Wolkowicz (1999)) or by the criss-cross algorithm (Illes, Szirmai, and Terlaky (1999)).

References

- Adler, I., P. Christos, and A. Rubinstein (2014): On Simplex Pivoting Rules and Complexity Theory **arXiV**
- Alevras, D. and M. W. Padberg (2001): *Linear Optimization and Extensions; Problems and Extensions* **Springer-Verlag**
- Bland, R. G. (1977): New Finite Pivoting Rules for the Simplex Method *Mathematics of Operations Research* **2 (2)** 103-107
- Borgwardt, K. H. (1987): The Simple Method – A Probabilistic Analysis *Algorithms and Combinatorics* **1 Springer-Verlag** Berlin
- Craven, B. D. (1988): Fractional Programming *Sigma Series in Applied Mathematics* **4 Heldermann Verlag** Berlin
- Dantzig, G. B., and M. N. Thapa (1997): *Linear Programming I: Introduction* **Springer-Verlag**
- Dantzig, G. B., and M. N. Thapa (2003): *Linear Programming II: Theory and Extensions* **Springer-Verlag**
- Disser, Y., and M. Skutella (2018): The Simplex Algorithm is NP-mighty **arXiV**

- Fearnly, J., and R. Savani (2015): The Complexity of the Simplex Method **arXiV**
- Fukuda, K., and T. Terlaky (1997): Criss-cross Methods: A Fresh View on Pivot Algorithms *Mathematical Programming Series B* **79 (1-3)** 369-395
- Hansen, T., and U. Zwick (2015): An Improved Version of the Random-Facet Pivoting Rule for the Simplex Algorithm *STOC '15: Proceedings of the 47th Annual ACM Symposium on Theory of Computing* 209-218
- Harris, P. M. J. (1973): Pivot Selection Methods of the Devex LP Code *Mathematical Programming* **5** 1-28
- Illes, T., A. Szirmai, and T. Terlaky (1999): The Finite Criss-cross Method for Hyperbolic Programming *European Journal of Operations Research* **114 (1)** 198-214
- Klee, V., and G. J. Minty (1972): How good is the Simplex Algorithm? in: *Inequalities III: Proceedings of the 3rd Symposium on Inequalities, editors: Shisha Oved* **Academic Press** New York-London
- Kruk, S., and H. Wolkowicz (1999): Pseudo-linear Programming *SIAM Review* **41 (4)** 795-805
- Maros, I., and G. Mitra (1996): Simplex Algorithms, in: *Advances in Linear and Integer Programming, editor: J E Beasley* **Oxford Science** 1-46
- Maros, I. (2003): Computational Techniques of the Simplex Method *International Research in Operations Research and Management Science* **61 Kluwer Academic Publishers** Boston MA
- Mathis, F. H., and L. J. Mathis (1995): A Non-linear Programming Algorithm for Hospital Management *SIAM Review* **37 (2)** 230-234
- Murty, K. G. (1983): *Linear Programming* **John Wiley and Sons** New York
- Nering, E. D., and A. W. Tucker (1993): *Linear Programs and Related Problems* **Academic Press**
- Padberg, M. W. (1999): *Linear Optimization and Extensions 2nd Edition* **Springer-Verlag**
- Schrijver, A. (1998): *Theory of Linear and Integer Programming* **John Wiley and Sons**

- Spielman, D., and S. H. Teng (2001): Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time **arXiV**
- Stone, R. E., and C. A. Tovey (1991a): The Simplex and the Projective Scaling Algorithms as Iteratively Re-weighted Least Squares Methods *SIAM Review* **33 (2)** 220-237
- Stone, R. E., and C. A. Tovey (1991b): Erratum: The Simplex and the Projective Scaling Algorithms as Iteratively Re-weighted Least Squares Methods *SIAM Review* **33 (3)** 461
- Strang, G. (1987): Karmarkar's Algorithm and its Place in Applied Mathematics *Mathematical Intelligencer* **9 (2)** 4-10
- Terlaky, T., and S. Z. Zhang (1993): Pivot Rules for Linear Programming: A Survey on Recent Theoretical Developments *Annals of Operations Research* **46-47 (1)** 203-233
- van der Bei, R. J. (2008): Linear Programming: Foundations and Extensions 3rd Edition *International Series in Operations Research and Management Science* **114 Springer-Verlag**
- Wikipedia (2020): Simplex Algorithm

Practical Guide to the Simplex Method of Linear Programming

Basic Steps of Simplex Algorithm – Write the Linear Programming Problem in Standard Form

1. What is Linear Programming: Linear Programming – or *linear optimization* – refers to the problem of optimizing a linear *objective function* of several variables subject to a set of linear equality or inequality constraints.
2. Standard Form of Linear Programming: Every linear programming problem can be written in the following *standard form*: Minimize

$$\zeta = \boldsymbol{c}^T \boldsymbol{x}$$

subject to

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{x} \geq \boldsymbol{0}$$

3. Vector Spaces of the Variables: Here

$$\boldsymbol{x} \in \mathbb{R}^n$$

is a vector of unknowns,

$$\boldsymbol{A} \in \mathbb{R}^{m \times n}$$

with typically m much larger than n, the coefficient vector of the objective function, and the expression

$$\boldsymbol{x} \geq \mathbf{0}$$

signifies

$$x_i \geq 0$$

for

$$i = 1, \cdots, n$$

(Oliver (2020)).

4. Assumption on the Rank of $\boldsymbol{A}$: For simplicity, it is assumed that

$$Rank\ \boldsymbol{A} = m$$

i.e., that the rows of $\boldsymbol{A}$ are linearly independent.

5. Conversion into the Standard Form: Turning the problem into the standard form involves the following steps.
6. Maximization as Minimization Plus Standard Inequalities: Turn the maximization into minimization, and write inequalities in standard order. Multiply expressions, where appropriate, by -1.
7. Introduction of the Slack Variables: Introduce *slack variables* to turn inequality constraints into equality constraints with non-negative unknowns. Any inequality of the form

$$a_1 x_1 + \cdots + a_n x_n \leq c$$

can be replaced with

$$a_1 x_1 + \cdots + a_n x_n + s = c$$

and

$$s \geq 0$$

8. Replacement of the Unconstrained Variables: Replace variables that are not sign-constrained by differences. Any real number x can be written as the difference of non-negative numbers

$$x = u - v$$

with

$$u, v \geq 0$$

Basic Steps of the Algorithm – Write the Coefficients of the Problem into a Simplex Tableau

1. Gaussian Matrix of the Coefficients: The coefficients of the linear system are collected into an augmented matrix as known from Gaussian elimination for systems of linear equations; the coefficients of the objective function are written in a row with a zero on the right-hand column.
2. Basic and Non-basic Variables: In the following steps, the variables will be divided into m *basic* and $n - m$ *non-basic* variables.

3. Pivots Chosen from Basic Variables: The tableau will be acted upon by the rules of Gaussian elimination, where the pivots are always chosen from the columns corresponding to the basic variables.
4. Initial Set of Basic Variables: The choice of the initial set of basic variables corresponds to a point in the feasible region of the linear programming problem – although such a choice may not be obvious.

Basic Steps of the Simplex Algorithm – Gaussian Elimination

1. Reduction of Basic Variable Columns: For a given set of basic variables, Gaussian elimination is used to reduce the columns to a permutation of the identity matrix.
2. Zeroing of the Non-basic Coefficients: This amount to solving

$$\boldsymbol{Ax = b}$$

   in such a way that the values for the basic variables are explicitly given by the entries in the right-hand column of the fully reduced matrix.
3. Elimination of the Objective Pivot Coefficients: In addition, the coefficients of the objective function for each pivot are eliminated.
4. Basic Variables must be Non-negative: The solution expressed by the tableau is only admissible if all the basic variables are non-negative, i.e., if the right-hand column of the reduced tableau is free of negative entries.
5. Appropriate Choice of Initial Variables: At the initial stage, however, negative entries may come up; this indicates that different initial basic variables should have been chosen.
6. Guaranteeing the Initial Feasible Tableau: At later stages, the selection rules for the basic variables will guarantee that an initial feasible tableau will remain feasible throughout the process.

Basic Steps of the Simplex Algorithm – Choose New Basic Variables

1. Choice of the Entering Variable: If, at this stage, the objective function row has at least one negative entry, the cost can be lowered by making the corresponding variable basic. The new basic variable is called the *entering variable*.
2. Choice of the Leaving Variable: Correspondingly, one formerly basic variable has then to become non-basic, and this variable is called the *leaving variable*. This leads to the following standard selection rules.
3. Entering Variable Selection Rule #1: The *entering variable* shall correspond to the column which has the most negative entry in the cost function row.
4. Entering Variable Selection Rule #2: If all cost function coefficients are non-negative, the cost cannot be lowered and one has reached an optimum. The algorithm then terminates.
5. Leaving Variable Selection Rule #1: Once the entering variable is determined, the *leaving variable* shall be chosen as follows. For each row, the ratio of the right-hand coefficient to the corresponding coefficient in the entering variable column is computed.
6. Leaving Variable Selection Rule #2: The row with the smallest finite positive ratio is then selected. The leaving variable is then determined by the column that currently owns the pivot in the row.
7. Leaving Variable Selection Rule #3: If all the coefficients in the entering variable column are non-positive, the cost can be lowered indefinitely, i.e., the linear programming problem does not have a finite solution. The algorithm then also terminates.
8. Locating the Entering/Leaving Variables: If the entering and the leaving variables can be found, one goes to the Gaussian elimination step and iterates.
9. Choice of the Most Negative Coefficient: The choice of the most negative coefficient in the selection of the entering variable is only a heuristic for choosing a direction fast decrease of the objective function.

10. Retaining the Basic Variable Feasibility: The leaving variable selection rule ensures that the new set of basic variables remains feasible.

Basic Steps of the Simplex Algorithm – Read off the Solution

1. Optimal Basic/Non-Basic Variable Values: The solution represented by the final tableau has all non-basic variables set to zero, while the values for the basic variables can be read off the right-hand columns.
2. Optimal Value of the Objective: The right-most value of the objective function row gives the negative value of the optimal objective function.
3. Verification of Equality/Non equality Constraints: As a final check the solution just satisfy the equality and the inequality constraints, as can be verified by a direct computation.

Initialization

1. Feasible Set of Initial Variables: For several problems, it may not be obvious which set of variables form a *feasible* initial set of basic variables.
2. Trial-and-error Combinatorial Complexity: For large problems, a trial-and-error approach in prohibitively expensive due to the rapid growth of $\binom{m}{n}$, the number of possibilities for choosing m basic variables out of a total of n variables, as n and m become large.
3. Introducing Large Valued Artificial Variables: This problem can be overcome by adding a set of m artificial variables which form a trivial set of basic variables and which are penalized by a large coefficient ω in the objective function.
4. Artificial Variables will become Non-basic: This penalty will cause the artificial variables to become non-basic as the algorithm proceeds.

5. Choosing the Value for ω: When using the computer to perform the simplex algorithm numerically, ω should be chosen large – one or two orders of magnitude larger than any other coefficients in the problem – but not too large – to avoid loss of significant digits in floating point arithmetic.
6. Artificial Variables Staying Basic: If not all artificial variables become non-basic, ω must be increased. If this happens for any value of ω, the feasible region is empty.
7. ω Limited to Artificial Variable Columns: In the final tableau, ω can only appear in artificial variable columns.

References

- Oliver, M. (2020): Practical Guide to the Simplex Method of Linear Programming

Optimal Control

Introduction

1. Optimal Control Theory: Optimal Control Theory is a branch of control theory that deals with finding a control for a dynamical system over a period of time such that an objective function is optimized (Ross (2015), Wikipedia (2025)).
2. Incorporating OR Problems in the Framework: A dynamical system may also be introduced to embed operations research problems within the framework of the optimal control theory (Ross, Karpenko, and Proulx (2016), Ross, Proulx, and Karpenko (2020)).
3. Extension to the Calculus of Variations: Optimal control is an extension to the calculation of variations, and is a mathematical optimization method for deriving control policies (Sargent (2000)).

General Approach

1. Attainment of the Optimality Criterion: Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved.
2. State/Control Variables Cost Function: A control problem includes a cost functional that is a function of the state and control variables.
3. Definition of an Optimal Control: An *optimal control* is a set of differential equations describing the paths of the control variables that minimize the cost function.
4. Pontryagin's Principle and HJB Approach: The optimal control can be derived using Pontryagin's maximum principle – a necessary condition also known as Pontryagin's

minimum principle or simply Pontryagin's principle (Ross (2015)), or by scaling the Hamilton-Jacobi-Bellman equation – a sufficient condition.

5. Optimization of a Continuous-time Cost Functional: A more abstract framework goes as follows (Ross (2015)). Minimize the continuous-time cost functional

$$J\big(\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f\big) := E\big(\boldsymbol{x}(t_0), t_0, \boldsymbol{x}\big(t_f\big), t_f\big) + \int_{t_0}^{t_f} F(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt$$

subject to the first-order dynamic constraints – the *state equation*

$$\dot{\boldsymbol{x}}(\cdot) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$$

the algebraic *path constraints*

$$h(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq 0$$

where $\boldsymbol{x}(t)$ is the *state*, $\boldsymbol{u}(t)$ is the *control*, t is the independent variable – generally speaking, time - t_0 is the initial time, and t_f is the terminal time.

6. End point and Running Costs: The terms E and F are called the endpoint cost and the running cost. In the calculus of variations, E and F are referred to as the Mayer term and the *Lagrangian*, respectively.
7. Active Nature of the Constraints: Furthermore, it is noted that the path constraints are, in general, *inequality* constraints, and thus may not be active, i.e., equal to zero, at the optimal solution.
8. Generation of Locally Optimal Solutions: It is also noted that the optimal control problem as stated above may have multiple solutions, i.e., the solution may not be unique. Thus, it is most often the case that any solution $\left[\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_0^*, t_f^*\right]$ to the optimal control is *locally minimized*.

Linear Quadratic Control

1. Linear Quadratic Optimal Control Problem: A special case of the general nonlinear optimal control problem given in the previous section is the *linear quadratic* – LQ – optimal control problem.
2. Statement of the LQ Problem: Minimize the *quadratic* continuous-time cost functional

$$J = \frac{1}{2}\boldsymbol{x}^T(t_f)\boldsymbol{S}_f\boldsymbol{x}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}[\boldsymbol{x}^T(t)\boldsymbol{Q}(t)\boldsymbol{x}(t) + \boldsymbol{u}^T(t)\boldsymbol{R}(t)\boldsymbol{u}(t)]dt$$

   subject to the *linear* first-order dynamic constraints

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t)$$

   and the initial condition

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

3. Linear Quadratic Regulator: A particular form of the LQ problem that arises in many control system problems is that of the *linear quadratic regulator* – LQR – where all of the matrices, i.e., A, B, Q, and R – are *constant*, the initial time is arbitrarily set to 0, and the terminal time is taken in the limit of

$$t_f \to \infty$$

   This last assumption is what is known as *infinite horizon*.
4. Formulation of the LQR Problem: Minimize the infinite horizon quadratic continuous-time cost functional

$$J = \frac{1}{2}\int_{t_0}^{t_f} [\boldsymbol{x}^T(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^T(t)\boldsymbol{R}\boldsymbol{u}(t)]dt$$

subject to the *linear time-invariant* first-order dynamic constraints

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$$

and the initial conditions

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

5. Final Horizon $\boldsymbol{Q}$ and $\boldsymbol{R}$: In the finite-horizon case, the matrices are restricted in that $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive semi-definite and positive definite, respectively.
6. Infinite-Horizon $\boldsymbol{Q}$ and $\boldsymbol{R}$ #1: In the infinite-horizon case, however, the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are not only positive semi-definite and positive definite, respectively, but are also *constant*.
7. Infinite-Horizon $\boldsymbol{Q}$ and $\boldsymbol{R}$ #2: The additional restrictions on $\boldsymbol{Q}$ and $\boldsymbol{R}$ in the infinite-horizon case are enforced to ensure that the cost functional remains positive.
8. Restrictions on $\boldsymbol{A}$ and $\boldsymbol{B}$: Furthermore, in order to ensure that the cost function is *unbounded*, the additional restriction is imposed that the pair $(\boldsymbol{A}, \boldsymbol{B})$ is *controllable*.
9. Intuition behind LQ/LQR Cost Functionals: Note that the LQ or the LQR cost functionals can be thought of physically as attempting to minimize the *control energy* – measured as a quadratic form.
10. Final State under LQR: The infinite-horizon problem, i.e., LQR, may seem overly restrictive and essentially useless because it assumes that the operation is driving the system to zero-state and hence driving the output of the system to zero.
11. Solving for the Non-zero Output: However, the problem of driving the output to a desired non-zero level can be solved *after* the zero output one is. In fact, it can be proved that this secondary LQR problem can be solved in a straightforward manner.

12. Form of Optimal Control Feedback: It has been shown in the optimal control theory that the LQ – or LQR – optimal control has the feedback from

$$\boldsymbol{u}(t) = -\boldsymbol{K}(t)\boldsymbol{x}(t)$$

where $\boldsymbol{K}(t)$ is a properly dimensioned matrix, given as

$$\boldsymbol{K}(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^{T}\boldsymbol{S}(t)$$

and $\boldsymbol{S}(t)$ is the solution to the differential Riccati equation.

13. Differential Riccati Equation: The differential Riccati equation is given as

$$\dot{\boldsymbol{S}}(t) = -\boldsymbol{S}(t)\boldsymbol{A} - \boldsymbol{A}^{T}\boldsymbol{S}(t) + \boldsymbol{S}(t)\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{T}\boldsymbol{S}(t) - \boldsymbol{Q}$$

14. Finite Horizon LQ Problem: For the finite horizon LQ problem, the Riccati equation is integrated backward in time using the terminal boundary condition

$$\boldsymbol{S}(t_f) = \boldsymbol{S}_f$$

15. Algebraic Riccate Equation ARE: For the infinite horizon LQR problem, the differential Riccati equation is replaced with the algebraic Riccati equation ARE given as

$$\boldsymbol{0} = -\boldsymbol{S}\boldsymbol{A} - \boldsymbol{A}^{T}\boldsymbol{S} + \boldsymbol{S}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{T}\boldsymbol{S} - \boldsymbol{Q}$$

16. Constant $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}$: ARE arises from the infinite horizon problem, and from the fact that A, B, Q, and R are constant.
17. PD/PSD Solution to the Riccati Equation: In general, there are multiple solutions to the algebraic Riccati equation and the *positive-definite* – or positive semi-definite – solution is the one that is used to compute the feedback gain.

Numerical Methods for Optimal Control

1. Numerical Techniques for Optimal Solutions: Optimal control problems are generally nonlinear and therefore, generally do not have analytic solutions, e.g., like the linear-quadratic optimal control problem. As a result, it is necessary to employ numerical methods to solve optimal control problems.
2. The Indirect Method: In an indirect method, the calculus of variations is employed to obtain the first-order optimality conditions.
3. Multi-point Boundary Value Problem: These conditions result in a two-point – or, in the case of a complex problem, a multi-point – boundary-value problem. This boundary-value problem actually has a special structure because it arises from taking the derivative of a Hamiltonian.
4. Resulting Dynamic Hamiltonian System: Thus, the resulting dynamical system is a Hamiltonian system of the form (Issac (2015))

$$\dot{x} = \frac{\partial H}{\partial \lambda}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x}$$

where

$$H = F + \lambda^T f - \mu^T h$$

is the *augmented Hamiltonian* and in an indirect method, the boundary-value problem is solved – using the appropriate boundary or *transversality* conditions.

5. Advantage of the Indirect Method: The beauty of using an indirect method is that the state and adjoint, i.e., λ, are solved for and the resulting solution is readily verified to be an external trajectory.
6. Drawback of the Indirect Method: The disadvantage of indirect methods that the boundary-value problem is often extremely difficult to solve – particularly for problem that span large time intervals or problems with interior point constraints.
7. Direct Method: In a direct method, the state or the control, or both, are approximated using an appropriate function approximation, e.g., polynomial approximation or piece-wise constant parameterization. Simultaneously, the cost functional is approximated as a *cost function*.
8. Nonlinear Optimization Problem: Then, the coefficients of the function approximations are treated as optimization variables and the problem is *transcribed* to a nonlinear optimization problem of the form: Minimize $F(z)$ subject to the algebraic constraints

$$g(z) = 0$$

$$h(z) \leq 0$$

9. Size of the Optimization Problems: Depending upon the type of direct method employed, the size of the nonlinear optimization problem, e.g., as in a direct shooting or quasi-linearization method, moderate, e.g., pseudo-spectral optimal control (Ross and Karpenko (2012)), or maybe quite large, e.g., a direct collocation method (Betts (2010)).
10. Solution to the Nonlinear Optimization: In the latter case, i.e., a collocation method. The nonlinear optimization problem maybe literally thousands to tens of thousands of variables and constraints.
11. Sparse Nature of the NLP: Given the size of the NLP's arising from a direct method, it may appear somewhat counter-intuitive that solving the nonlinear optimization problem is easier than solving the boundary-value problem.

12. Sparse Nature of the NLP: The reason for the relative ease of computation, particularly of a direct collocation method, is that the NLP is *sparse* and many well-known software programs exist, e.g., SNOPT (Gill, Murray, and Saunders (2007)) to solve large sparse NLPs.

Discrete-time Optimal Control

1. Discrete Time Systems and Solutions: The treatment thus far has shown continuous-time systems and control solutions. In fact, as optimal control solutions are now often implemented computationally, contemporary control theory is now primarily concerned with discrete time solutions and systems.
2. Theory of Consistent Approximations: The Theory of Consistent Approximations (Polak (1993), Ross (2005)) provides conditions under which solutions to a series of increasingly accurate discretized optimal control problem converge to the solution of the original, continuous-time problem.
3. Situations where Convergence does not Occur: Not all discretization methods have this property, even seemingly obvious ones (Fahroo and Ross (2008)). For instance, using a variable step-size routine to integrate the problem's dynamic equation may generate a gradient which does not converge to zero – or point in the right direction – as the solution is approached. The direct method RIOTS – http://www.schwartz-home.com/RIOTS - is based on the Theory of Consistent Approximation.

Examples

1. Solution to the Co-state: A common solution strategy in many optimal control problems is to solve for the co-state – sometimes called the shadow price - $\lambda(t)$.
2. Incremental Change in the Co-state: The co-state summarizes in one number the marginal value of expanding or contracting the state variable next turn.

3. Marginal Value of $\lambda(t)$: The marginal value is not only the gains accessing to it next turn but associated with the duration of the program.
4. Analytical Solution for $\lambda(t)$: It is nice when $\lambda(t)$ can be solved analytically, but usually, the most one can do is describe it sufficiently well that the intuition can grasp the character of the solution and an equation solver can solve numerically for the value.
5. Turn-t Optimal Value for $\lambda(t)$: Having obtained $\lambda(t)$, the turn-t optimal value for the control can be solved as a differential equation conditional on the knowledge of $\lambda(t)$.
6. Numerical Solutions for $\lambda(t)$: Usually, the strategy is to solve the thresholds and regions that characterize the optimal control and a numerical solver to isolate the actual choice value in time.

Examples – Finite Time

1. Problem of a Mine Owner: Consider the problem of a mine owner who must decide at what rate to extract ore from their mine. They own rights to the ore from date 0 to date T.
2. Time-dependent Amount of Ore $x(t)$: At date 0 there is x_0 ore in the ground, and the time-dependent amount of ore $x(t)$ left in the ground declines at the rate of $u(t)$ that the mine owner extracts it.
3. Extraction Cost of the Ore: The mine owner extracts ore at $\frac{u^2(t)}{x(t)}$ – the cost of extraction increasing with the square of the extraction and the inverse of the amount of ore left – and sells ore at a constant price p.
4. No "Scrap Value": Anyone left in the ground at time T cannot be sold and has no value – there is no scrap value.
5. Optimal Trajectory of Extraction: The owner chooses the rate of extraction varying with time $u(t)$ to maximize profits over the period od ownership with no time discounting.

6. Discrete-time Version - Profit: The manager maximizes profit P:

$$\mathrm{P} = \sum_{t=0}^{T-1} \left(p u_t - \frac{u_t^2}{x_t} \right)$$

subject to the law of motion for the state variables

$$x_{t+1} - x_t = -u_t$$

7. Discrete-time Version - Hamiltonian: Form the Hamiltonian and differentiate:

$$\mathrm{H} = p u_t - \frac{u_t^2}{x_t} - \lambda_{t+1} u_t$$

$$\frac{\partial \mathrm{H}}{\partial u_t} = p - \lambda_{t+1} - 2\frac{u_t}{x_t} = 0$$

$$\lambda_{t+1} - \lambda_t = -\frac{\partial \mathrm{H}}{\partial t} = -\left(\frac{u_t}{x_t}\right)^2$$

8. Discrete-time Version - Terminal λ: As the mine owner does not value the ore remaining at time T

$$\lambda_T = 0$$

9. Discrete-time Version - λ_t and x_t: Using the above, it is easy to solve for the x_t and λ_t series

$$\lambda_t = \lambda_{t+1} + \frac{(p - \lambda_{t+1})^2}{4}$$

$$x_{t+1} = x_t \frac{2 - p + \lambda_{t+1}}{2}$$

and using the initial and time-T conditions, the x_t series can be solved explicitly, giving u_t.

10. Continuous-time Version - Profit: The manager maximizes the profit P:

$$\mathrm{P} = \int_0^T \left[pu(t) - \frac{u^2(t)}{x(t)} \right] dt$$

where the state variable $x(t)$ evolves as follows:

$$\dot{x}(t) = -u(t)$$

11. Continuous-time Version - Hamiltonian: Form the Hamiltonian and differentiate:

$$\mathrm{H} = pu(t) - \frac{u^2(t)}{x(t)} - \lambda(t)u(t)$$

$$\frac{\partial \mathrm{H}}{\partial u} = p - \lambda(t) - 2\frac{u(t)}{x(t)} = 0$$

$$\dot{\lambda}(t) = -\frac{\partial \mathrm{H}}{\partial x} = -\frac{u^2(t)}{x(t)}$$

12. Continuous -time Version - Terminal λ: As the mine owner does not value the ore remaining at time T

$$\lambda(T) = 0$$

13. Continuous -time Version - $\lambda(t)$ and $x(t)$: Using the above, it is easy to solve for the differential equations governing u_t and $\lambda(t)$ series

$$\dot{\lambda}(t) = -\frac{[p - \lambda(t)]^2}{4}$$

$$u(t) = x(t)\frac{p - \lambda(t)}{2}$$

and using the initial and time-T conditions, the functions can be solved to yield, giving

$$x(t) = \frac{(4 - pt + pT)^2}{(4 + pT)^2}x_0$$

References

- Betts, J. T. (2010): *Practical Control for Optimal Control using Nonlinear Programming 2nd Edition* **SIAM Press** Philadelphia PA
- Fahroo, F., and I. M. Ross (2008): Convergence of the Co-states does not imply Convergence of the Control *Journal of Guidance, Control, and Dynamics* **31 (5)** 1492-1497
- Polak, E. (1993): On the Use of Consistent Approximations in the Solution of Semi-infinite Optimization and Optimal Control Problems *Mathematical Programming* **62** 393-415
- Ross, I. M. (2005): A Roadmap for Optimal Control: The Right Way to Commute *Annals of the New York Academy of Sciences* **1065 (1)** 210-231
- Ross, I. M., and M. Karpenko (2012): A Review of Pseudo-spectral Optimal Control: From Theory to Flight *Annual Reviews in Control* **36 (2)** 182-197

- Ross, I. M. (2015): A Primer on Pontryagin's Principle on Optimal Control **Collegiate Publishers** San Francisco CA
- Ross, I. M., M. Karpenko, and R. J. Proulx (2016): A Non-smooth Calculus for Solving some Graph-Theoretic Control Problems *10th IFAC Symposium on Nonlinear Control Systems NOLCOS 2016* **49 (18)** 462-467
- Ross, I. M., R. J. Proulx., and M. Karpenko (2020): An Optimal Control Theory for the Traveling Salesman Problem and its Variants **arXiv**
- Sargent, R. W. H. (2000): Optimal Control *Journal of Computational and Applied Mathematics* **124 (1-2)** 361-371
- Wikipedia (2025): Optimal Control

Hamilton-Jacobi-Bellman Equation

Introduction

1. Hamilton-Jacobi-Bellman HJB Equation: The Hamilton-Jacobi-Bellman HJB equation is a nonlinear partial differential equation that provides the necessary and the sufficient control for optimality of a control with respect to a loss function (Kirk (1970), Wikipedia (2025)).
2. Optimal Control from Value Function: Its solution is the value function of the optimal control problem which, once known, can be used to obtain the optimal control by taking the maximizer – or minimizer – of the Hamiltonian involved in the HJB equation (Yong and Zhu (1999), Naidu (2003)).
3. Wide Applicability of the HJB Method: While classical variational problems, such as the brachistochrone problem, can be solved using the Hamilton-Jacobi-Bellman equation (Kemajou-Brown (2016)), the method can be applied to a broader spectrum of problems.
4. Generalization to Stochastic Systems: Further, it can be generalized to stochastic systems, in which case the HJB equation is a second-order elliptic partial differential equation (Chang (2004)).
5. Viscosity Solution: Instead, the notion of a viscosity solution is required, in which conventional derivatives are replaced by set-valued derivatives (Bardi and Capuzzo-Dolcetta (1997)).

Optimal Control Problem

1. Deterministic Optimal Control Problem: Consider the following problem in deterministic optimal control over the time-period $[0, T]$:

$$V(x_0, 0) = \min_u \left\{ \int_0^T C(x_t, u_t)dt + D(x_T) \right\}$$

where $C(\cdot)$ is the scalar cost rate function and $D(\cdot)$ is a function that provides the bequest value at the final state, x_t is the system state vector, x_0 is assumed to be given, and u_t for

$$0 \leq t \leq T$$

is the control vector that needs to be determined. Thus, $V(x, t)$ is the value function.

2. Constraint on the State Evolution: The system is also subject to

$$\dot{x}_t = F(x_t, u_t)$$

where $F(\cdot)$ gives the vector determining the physical evolution of the state vector over time.

The Partial Differential Equation

1. The Hamilton-Jacobi-Bellman Partial Differential Equation: For the simple system above, the HJB PDE is

$$\frac{\partial V(x_t, t)}{\partial t} = \min_u \left\{ \frac{\partial V(x_t, t)}{\partial x} \cdot F(x_t, u_t) + C(x_t, u_t) \right\}$$

subject to the terminal value condition

$$V(x_t, T) = D(x_t)$$

2. Bellman-Value Function: As before, the unknown scalar function $V(x_t, t)$ in the above PDE is the Bellman value function, which represents the cost incurred from starting in the state x_t at time t and controlling the system optimally from then until time T.

Deriving the Equation

1. Bellman's Principle of Optimality: Intuitively, the HJB equation can be derived as follows. If $V(x_t, t)$ is the optimal cost-to-go function – also called the *value function* – then, by Bellman's principle of optimality going from t to $t + \Delta t$, one has

$$V(x_t, t) = \min_u \left\{ V(x_{t+\Delta t}, t + \Delta t) + \int_t^{t+\Delta t} C(x_s, u_s) ds \right\}$$

2. Taylor Expansion of the First Term: The Taylor expansion of the first-term on the right-hand side is

$$V(x_{t+\Delta t}, t + \Delta t) = V(x_t, t) + \frac{\partial V(x_t, t)}{\partial t} \cdot \Delta t + \frac{\partial V(x_t, t)}{\partial x} \cdot \dot{x}_t \cdot \Delta t + \mathcal{O}(\Delta t)$$

where $\mathcal{O}(\Delta t)$ denotes the terms in the Taylor expansion of higher order.

3. Derivative in the Limit: Subtracting $V(x_t, t)$ from both sides, and dividing by Δt, and taking the limit as Δt approaches zero, one obtains the HJB equation above.

Solving the Equation

1. Solution Backwards in Time: The HJB equation is usually solved backwards in time, starting from

$$t = T$$

and ending at

$$t = 0$$

(Lewis, Vrabie, and Syrmos (2012)).

2. Necessary and Sufficient Conditions: When solved over the whole of state space and $V(x_t)$ is continuously differentiable, the HJB equation is a necessary and a sufficient condition for an optimum when the terminal cost is unconstrained (Bertsekas (2005)). If one can solve for $V(x_t)$, there exists a control u that achieves the minimum cost.
3. Non-smooth Value Function: In the general case, the HJB equation does not have a classical, smooth solution. Several notions of generalized solutions have ben developed to cover such situations, including viscosity solutions (Bardi and Capuzzo-Dolcetta (1997)), minimax solution, and others.
4. Approximate Dynamic Programming: Approximate dynamic programming has been introduced by Bertsekas and Tsitsiklis (1996) with the use of use of artificial neural networks – multilayer perceptrons – for approximating the Bellman function in general.
5. Reducing the Impact of Dimensionality: This is an effective mitigation strategy for reducing the impact of dimensionality by replacing the memorization of the complete function mapping for the whole space domain with the memorization of the sole neural network parameters.
6. Continuous-time Approximate DP Approach: In particular, for continuous-time systems, an approximate dynamic programming approach that combines both policy iterations and neural networks was introduced (Abu-Khalaf and Lewis (2005)).

7. Discrete-time Approximate DP Approach: In discrete-time, an approach to solve the equation combining value iterations and neural networks was introduced (Al-Tamimi, Lewis, and Au-Khalaf (2008)).
8. Sum of Squares Optimization: Alternatively, it has been shown the sum-of-squares optimization can yield an approximate polynomial solution to the Hamilton-Jacobi-Bellman equation arbitrarily well with respect to the $\mathcal{L}^1$ norm (Jones and Peet (2020)).

Extension to Stochastic Problems

1. Generalization to Stochastic Control Problems: The idea of solving a control problem by applying the Bellman's principle of optimality and working out backwards in time to optimizing strategy can be generalized to stochastic control problems.
2. The Stochastic Process to Optimize: Consider similar as above

$$\min_{U} \mathbb{E}\left[\int_0^T C(t, X_t, U_t)dt + D(X_T)\right]$$

now with

$$(X_t)_{t\in[0,T]}$$

the stochastic process to optimize and

$$(U_t)_{t\in[0,T]}$$

the steering.

3. Stochastic HJB Equation: By first using Bellman and then expanding $V(X_t, t)$ with Ito's rule, one finds the stochastic HJB equation

$$\min_u \{\mathcal{A}V(X_t, t) + C(X_t, U_t, t)\} = 0$$

where $\mathcal{A}$ represents the stochastic differentiation operator, and subject to the terminal condition

$$V(X_t, T) = D(X_t)$$

4. Intermediate Solution Step for $V(X_t, T)$: In this case, a solution $V(X_t, T)$ of the latter does not necessarily solve for the primal problem; it is only a candidate and further verification is required.

Extension to Stochastic Problems – Application to LCQ-Control

1. Linear Dynamics with Quadratic Cost: As an example, this section looks at a system with linear stochastic dynamics and quadratic cost.
2. HJB Equation and Optimal Action: If the system dynamics is given by

$$\Delta X_t = (aX_t + bU_t) \cdot \Delta t + \sigma . \Delta W_t$$

and the cost accumulates at the rate

$$C(X_t, U_t) = \frac{1}{2} r_t U_t^2 + \frac{1}{2} q_t X_t^2$$

the HJB equation is given by

$$-\frac{\partial V(X_t,t)}{\partial t}=\frac{1}{2}q_t X_t^2+\frac{\partial V(X_t,t)}{\partial X_t}aX_t-\frac{b^2}{2r_t}\left[\frac{\partial V(X_t,t)}{\partial X_t}\right]^2+\frac{1}{2}\sigma^2\frac{\partial^2 V(X_t,t)}{\partial X_t^2}$$

with optimal action given by

$$u_t=-\frac{b}{r_t}\frac{\partial V(X_t,t)}{\partial X_t}$$

3. Quadratic Form for Value Function: Assuming a quadratic form for the value function, one obtains the usual Riccati equation for the Hessian of the value function as is usual for Linear Quadratic Gaussian Control.

References

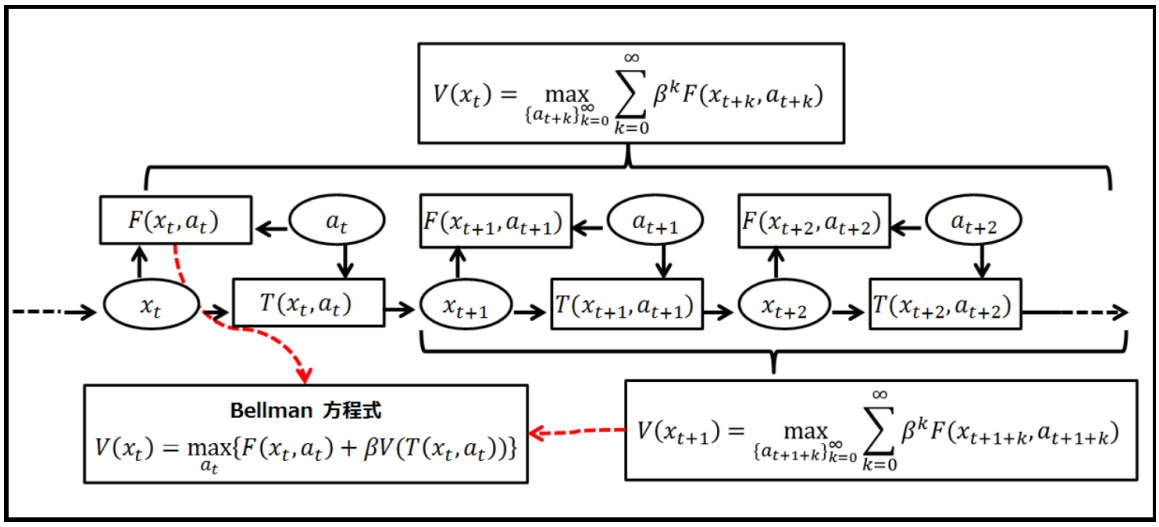
- Abu-Khalaf, M., and F. L. Lewis (2005): Nearly Optimal Control Laws for Nonlinear Systems with Saturating Actuators using a Neural Network HJB Approach *Automatica* **41 (5)** 779-791
- Al-Tamima, A., F. L. Lewis, and M. Abu-Khalaf (2008): Discrete-time Nonlinear HJB Solution using Approximate Dynamic Programming: Convergence Proof *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* **38 (4)** 943-949
- Bardi, M., and Capuzzo-Dolcetta (1997): *Optimal Control and Viscosity Solutions of the Hamilton-Jacobi-Bellman Equations* **Cambridge University Press** Cambridge UK
- Bertsekas, D. P. (2005): *Dynamic Programming and Optimal Control* **Athena Scientific** Nashua NH
- Bertsekas, D. P. and J. N. Tsitsiklis (1996): *Neuro-dynamic Programming* **Athena Scientific** Nashua NH
- Chang, F. R. (2004): *Stochastic Optimization in Continuous Time* **Birkhauser** Boston MA

- Jones, M., and M. Peet (2020): Polynomial Approximations of Value Functions and Nonlinear Control Design with Performance Bounds **arXiv**
- Kemajou-Brown, I. (2016): Brief Theory of Optimal Control Theory and some recent Developments, in: *Probability of Algebraic and Geometric Structures (editors: Budzban, G., H. R. Hughes, and H. Schurz) Contemporary Mathematics* **668** 119-130
- Kirk, D. E. (1970): *Optimal Control Theory: An Introduction* **Prentice-Hall** Englewood Cliffs NJ
- Lewis, F. L., D. Vrabie, and V. L. Syrmos (2012): *Optimal Control 3rd Edition*
- Naidu, D. S. (2003): *Optimal Control Systems* **CRC Press** Boca Raton FL
- Wikipedia (2025): Hamilton-Jacobi-Bellman Equation
- Yong, J., and X. Y. Zhou (1999): *Dynamic Programming and HJB Equations* **Springer** New York NY

Bellman Equation

Overview

1. Objective of the Bellman Equation: *Bellman equation* is a technique in dynamic programming which breaks an optimization problem into a sequence of simpler sub-problems, as Bellman's "principle of optimality" prescribes (Kirk (1970), Wikipedia (2025)). It is a necessary condition for optimality (Dixit (1990)).
2. Value of a Decision Problem: The *value* of a decision problem at a certain point is written in terms of the pay-off from some initial choices and the *value* of the remaining decision problem that results from these initial choices.



3. Algebraic Structures with Total Ordering: The equation applies to algebraic structures with a total ordering; for algebraic structures with a partial ordering, the generic Bellman's equation can be used (Szczesniak and Wozna-Szczesniak (2023)).
4. Discrete Bellman using State Augmentation: In discrete time any multi-stage optimization can be solved by analyzing the appropriate Bellman equation. The

appropriate Bellman equation can be found by state variables – state augmentation (Jones and Peet (2020)).

5. Curse of Dimensionality: However, the resulting augmented-state multi-stage optimization problem has a higher dimensional state space than the original multi-stage optimization problem – an issue that can potentially render the augmented problem intractable due to the *curse of dimensionality*.
6. Backward Separability in Multi-stage Optimization: Alternatively, it has been shown that if the cost function of the multi-stage problem satisfies a "backward separable" structure, then the Bellman equation can be found without state augmentation (Jones and Peet (2021)).

Analytical Concepts in Dynamic Programming

1. Objective Function of Optimization Problem: Any optimization problem has an objective: minimizing time travel, minimizing cost, maximizing profits, maximizing utility, etc. The mathematical function that describes this objective is called the *objective function* (Bertsekas (2005)).
2. Decomposition of the Multi-period Problem: Dynamic programming breaks a multi-period planning problem into simpler steps at different points in time. Therefore, it requires keeping track of how the decision situation is evolving over time.
3. The State Vector: The information about the current situation that is needed to make a correct decision is call the *state* (Bellman (1957), Dreyfus (2002)).
4. State Variable Vector - Example: For example, to decide how much to consume and spend at each point in time, people would need to know – among other things – their initial wealth. Therefore, the wealth W would be one of their state variables, but there would probably be others.
5. Choice of the Control Variables: The variables chosen at any given point in time are often called the *control variables*. For instance, given their current wealth, people may decide how much to consume now.

6. Transition to the Adjacent State: Choosing the control variables now may be equivalent to choosing the next state; more generally, the next state is affected by other factors in addition to the current control.
7. State Vector Evolution - Example: For example, in the simplest case, today's wealth – the state – and the consumption – the control – might determine exactly tomorrow's wealth – the new state – though typically other factors will affect tomorrow's wealth too.
8. Optimality using Dynamic Programming: The dynamic programming approach describes an optimal plan by finding the rule that tells what the optimal control should be, given any possible value of the state.
9. The Policy Function Example: For example, if the consumption c depends only on the wealth W, one would seek a rule $c(W)$ that gives consumption as a function of wealth. Such a rule that determines the controls as a function of the state is called a *policy function* (Bellman (1957)).
10. Optimal Decision Rule: Finally, by definition, the optimal decision rule is the one that achieves the best possible value of the objective.
11. Utility based Optimal Value Extraction: For example, if someone chooses consumption, given wealth, in order to maximize happiness – assuming happiness H can be represented by a mathematical optimization and is something defined using wealth – then each level of wealth will be associated with some highest level of happiness $H(W)$.
12. Definition of the Value Function: The best possible value of the objective, written as a function of the state, is called the *value function*.
13. Bellman Equation - Formal Definition: Bellman showed that a dynamic optimization problem in discrete time can be stated in a discrete step-by-step form known as backward induction by writing down the relationship between the value function in one period and the value function in the next period. The relationship between the two functions is called the *Bellman equation*.
14. Relation between the Sequential Value Functions: In this approach, the optimal policy in the last time is specified in advance as a function of the state variable's value at

that time, and the resulting optimal value of the objective function is thus expressed in terms of that value of the state variable.

15. Iteration of the Value Function Backwards: Next, the next-to-last period's optimization involves maximizing the sum of that period's specific objective function and the optimal value of the future objective function, giving that period's optimal policy contingent upon the value of the state variable as of the next-to-last period decision.
16. Retracking the Initial Value: This logic continues recursively back in time, until the first period decision rule is derived, as a function of initial state variable value, by optimizing the sum of the first period-specific objective function and the value of the second period's value function, which gives the value for all the future periods.
17. Guarantee of Optimal Execution: Thus, each period's decision is made by explicitly acknowledging that all future decisions will be optimally made.

Derivation – A Dynamic Decision Problem

1. Initial State and Time: Let x_t be the state at time t. For a decision that begins at time 0, the initial state is given as x_0.
2. Universe of State-dependent Action: At any time, the set of possible actions depends on the current state; this is expressed as

$$a_t \in \mathcal{T}(x_t)$$

where a particular action t represents particular values for one or more control variables, and $\mathcal{T}(x_t)$ is the set of actions available to be taken at state x_t.

3. Migration to a Subsequent State: It is also assumed that the state changes to a new state $T(x, a)$ from x when the action a is taken, and that the current payoff in taking action a in state x is $F(x, a)$.

4. Impatience as a Discount Factor: Finally, one assumes impatience, represented by a discount factor

$$0 < \beta < 1$$

5. Infinite-Horizon Decision Problem: Under these assumptions, an infinite-horizon decision problem takes the form

$$V(x_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, a_t)$$

subject to the constraints

$$a_t \in \Gamma(x_t)$$

$$x_{t+1} = T(x_t, a_t)$$

$$\forall t = 0, 1, 2, \cdots$$

6. Optimal Value for the Value Function: Notice that one has defined $V(x_0)$ to denote the optimal value that one can obtain by maximizing this objective function subject to the constraints. This function is the *value function*.
7. Function of the Initial State: It is a function of the initial state variable x_0, since the best value obtainable depends on the initial situation.

Derivation - Bellman's Principle of Optimality

1. Breaking Down the Decision Problem: The dynamic programming method breaks this decision problem into smaller sub-problems.

2. Principle of Optimality: An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision (Bellman (1952, 1957), Dreyfus (2002)).
3. Treating the First Decision Separately: As suggested by the *principle of optimality*, one considers the first decision separately, setting aside all future decisions – starting afresh from time 1 with new state x_1.
4. Recasting the Infinite-decision Problem: Collecting the future decisions in brackets on the right, the above infinite-horizon decision problem is equivalent to

$$\max_{a_0} \left\{ F(x_0, a_0) + \beta \left[\max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} F(x_t, a_t) : \; a_t \in \mathcal{T}(x_t), x_{t+1} = T(x_t, a_t), \forall t \geq 1 \right] \right\}$$

subject to the constraints

$$a_0 \in \mathcal{T}(x_0)$$

$$x_1 = T(x_0, a_0)$$

5. Optimal x_1 after First Step: Here one chooses a_0, with the awareness that this choice will cause the time 1 state to be

$$x_1 = T(x_0, a_0)$$

That new state will then affect the decision problem from time 1 on.

6. Future Decision Problem: The entire future decision problem appears inside the square brackets on the right.

Derivation – The Bellman Equation

1. Recursive Definition of the Value Equation: The problem above can be re-written as a recursive definition of the value function:

$$V(x_0) = \max_{a_0} \{F(x_0, a_0) + \beta V(x_1)\}$$

subject to the constraints

$$a_0 \in \Gamma(x_0)$$

$$x_1 = T(x_0, a_0)$$

2. Simplification of the Bellman Equation: The above may be simplified even further if the time sub-scripts are dropped and the value of the next state is plugged in:

$$V(x) = \max_{a \in \Gamma(x)} \{F(x, a) + \beta V(T(x, a))\}$$

3. Functional Equation for Value Function: The Bellman equation is classified as a functional equation, because solving it means finding the unknown function V, which is the *value function*.
4. Optimal Action Policy Function: By calculating the value function, one will also find the function $a(x)$ that describes the optimal cost as a function of the state; this is called the *proxy function*.

Derivation – In a Stochastic Problem

1. Specific Example from Economics: This section considers an infinitely-lived consumer with initial wealth endowment a_0 at period 0.
2. Consumption Based Utility Function: The consumer has an instantaneous utility function $u(c)$ where c denotes the consumption and discounts the next period utility at the rate

$$0 < \beta < 1$$

   What is not consumed in period t carries over to the next period with interest rate r.
3. Consumer's Utility Maximization Problem: Then the consumer's utility maximization problem is to choose a consumption plan $\{c_t\}$ that solves

$$\max \sum_{t=0}^{\infty} \beta^t u(c_t)$$

   subject to

$$a_{t+1} = (1 + r)(a_t - c_t)$$

$$c_t \geq 0$$

$$\lim_{t \to \infty} a_t \geq 0$$

4. Constraints Bounding the Bellman Equation: The first constraint is the capital accumulation/law of motion specified by the problem, while the second constraint is

the transversality condition that the consumer does not carry debt at the end of their life. The Bellman equation is

$$V(x) = \max_{0 \le c \le a} \{u(c) + \beta V((1+r)(a-c))\}$$

5. Sequence Problems using Hamiltonian Equations: Alternatively, one can treat the sequence problem directly using, for example, the Hamiltonian equations.
6. Stochastic Interest Rates: If the interest rate varies from period to period, the consumer is faced with a stochastic optimization problem.
7. Markov Process Governing the Interest Rates: Let the interest rate r follow a Markov process with transition probability function $\mathbb{Q}(r, \mathbb{P}_r)$ where $\mathbb{P}_r$ denotes the probability measure governing the distribution of interest rate next period if the current period interest rate is r.
8. Choice of the Current Consumption: In this model the consumer decides their current period consumption after the current period interest rate is announced.
9. Optimizing the Utility over $\{c_t\}$: Rather than simply choose a single sequence $\{c_t\}$, the consumer must choose a sequence $\{c_t\}$ for each possible realization of $\{r_t\}$ in such a way that their lifetime expected utility is maximized

$$\max_{\{c_t\}_{t=0}^{\infty}} \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t u(c_t)\right]$$

10. Probability Measure for the Expectation: The expectations $\mathbb{E}$ is taken on the appropriate probability measure given by $\mathbb{Q}$ on the sequence of r's. Because r is governed by a Markov process, dynamic programming simplifies the problem significantly.
11. The Corresponding Bellman Equation: Then the Bellman equation is

$$V(a,r) = \max_{0 \le c \le a} \left\{u(c) + \beta \int V((1+r)(a-c), r')\mathbb{Q}(r, \mathbb{P}_r)\right\}$$

Under some reasonable assumptions, the resulting optimal policy function $g(a, r)$ is measurable.

12. Ex-post Stochastic Sequential Optimization Problem: For a general stochastic sequential optimization problem with Markovian shocks and where their agent is faced with their decision *ex-post*, the Bellman equation takes a very similar form

$$V(x, z) = \max_{c \in \mathcal{T}(x, z)} \left\{ F(x, c, z) + \beta \int V(T(x, c), z') d\mathbb{P}_z[z'] \right\}$$

Solution Methods

1. Method of Undetermined Coefficients: Also known as 'guess and verify', this can be used to solve certain infinite-horizon, autonomous Bellman equations (Ljungqvist and Sargent (2004)).
2. Reduction to the Euler Form: By calculating the first-order conditions associated with the Bellman equation, then using the envelope theorem to eliminate the derivatives of a value function, it is possible to obtain a system of difference equations or differential equations called the 'Euler equations' (Miao (2014)).
3. Standard Techniques for Euler Equations: Standard techniques for the solutions of difference or differential equations can then be used to calculate the dynamics of the state variables and the control variables of the optimization problem.

Markov Decision Process Example

1. Bellman-based Recursion for Expected Rewards: In Markov decision processes, a Bellman equation is a recursion for expected rewards.

2. Expected Reward for Prescribed Policy Action: For example, the expected reward for being in a state s and following a fixed policy π has the Bellman form

$$V_{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s'} P\big(s' \mid s, \pi(s)\big) V_{\pi}(s')$$

This equation describes the expected reward for taking the action prescribed by the policy π.

3. Bellman Optimality Equation: The equation for the optimal policy is referred to as the *Bellman optimality equation*:

$$V_{\pi^*}(s) = \underset{a}{max} \left\{ R(s,a) + \gamma \sum_{s'} P(s' \mid s, a) V_{\pi^*}(s') \right\}$$

where π^* is the optimal policy and V_{π^*} refers to the value function at the optimal policy. The equation above describes the reward for taking action giving the highest expected return.

References

- Bellman, R. E. (1952): On the Theory of Dynamic Programming *Proceedings of the National Academy of Sciences USA* **38 (8)** 716-719
- Bellman, R. E. (1957): *Dynamic Programming* **Dover** Garden City NY
- Bertsekas, D. P. (2005): *Dynamic Programming and Optimal Control* **Athena Scientific** Nashua NH
- Dirk, D. (1970): *Optimal Control Theory: An Introduction* **Prentice-Hall** Hoboken NJ
- Dixit, A. K. (1990): *Optimization in Economic Theory* **Oxford University Press** Oxford UK

- Dreyfus, S. (2002): Richard Bellman on the Birth of Dynamic Programming **50 (1)** 48-51
- Jones, M., and M. M. Peet (2020): Extensions of the Dynamic Programming Framework: Battery Scheduling, Demand Charges, and Renewable Integration *IEEE Transactions on Automatic Control* **66 (4)** 1602-1617
- Jones, M., and M. M. Peet (2021): A Generalization of Bellman's Equations to Applications of Path Planning, Obstacle Avoidance, and Invariance Set Estimation *Automatica* **127** 109510
- Ljungqvist, L., and T. J. Sargent (2004): *Recursive Macroeconomic Theory 2nd Edition* **MIT Press** Cambridge MA
- Miao, J. (2014): *Economic Dynamics in Discrete Time* **MIT Press** Cambridge MA
- Szczesniak, I., and Wozna-Szczesniak, B. (2023): Generic Dijkstra: Correctness and Tractability **arXiv**
- Wikipedia (2025): Bellman Equation

Understanding Exact Dynamic Programming Through Bellman Operators

Value Functions as Vectors

1. State and Action Spaces: Assume the state space $\mathcal{S}$ consists of n states $\{s_1, \cdots, s_n\}$, and the action space $\mathcal{A}$ consists of m actions $\{a_1, \cdots, a_m\}$. Extensions to continuous spaces is obvious.
2. Stochastic and Deterministic Policies: The stochastic policy is denoted as $\pi(a \mid s)$ - probability of a given s – deterministic policy is denoted

$$\pi(s) = a$$

3. Value Function over State Space: Consider n-dimensions $\mathbb{R}^n$, each dimension corresponding to a state in $\mathcal{S}$. The value function is

$$v : \mathcal{S} \to \mathbb{R}$$

   is a vector over this space, with coordinates $[v(s_1), \cdots, v(s_n)]$
4. Value Function for a Policy π:

$$v_\pi : \mathcal{S} \to \mathbb{R}$$

5. Optimal Value Function: Optimal value function denoted as

$$v_* : \mathcal{S} \to \mathbb{R}$$

such that for any

$$s \in \mathcal{S}$$

$$v_*(s) = \max_{\pi} v_\pi(s)$$

Expected Reward and State Transition Probabilities

1. $\mathcal{R}_{s,a}$: The expected reward upon action a in state s
2. $\mathcal{P}_{s,s',a}$: Probability of transition

$$s \to s'$$

upon action a

3. R_π and P_π - #1:

$$R_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \cdot \mathcal{R}_{s,a}$$

$$\mathcal{R}_\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(a \mid s) \cdot \mathcal{R}_{s,s',a}$$

4. R_π and P_π - #2: R_π is the vector $[R_\pi(s_1), \cdots, R_\pi(s_n)]$ P_π is the matrix $P_\pi(s_i, s_{i'})$

$$1 \leq i$$

$$i' \leq n$$

5. γ: MDP Discount Factor

Bellman Operators B_π and B_*

1. Transformation of Value Function Vectors: This section defines operators that transform one value function vector to another.
2. Bellman Policy Operator B_π: B_π operator – for a policy π – on a value function vector v is

$$B_\pi v = R_\pi + \gamma P_\pi \cdot v$$

3. Linearity of the B_π Operator: B_π is a linear operator with fixed point v_π, i.e.

$$B_\pi v_\pi = v_\pi$$

4. Bellman Optimality Operator B_*: B_* operator on a value function v is

$$(B_* v)(s) = \max_a \left\{ \mathcal{R}_{s,a} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s',a} \cdot v(s') \right\}$$

5. Nonlinearity of the B_* Operator: B_* is a nonlinear operator with a fixed point v_*, i.e.

$$B_* v_* = v_*$$

6. Deterministic Greedy Policy Mapping Function: Define function G mapping a value function v to a deterministic *greedy* policy $G(v)$ as

$$G(v)(s) = \arg\max_a \left\{ \mathcal{R}_{s,a} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s',a} \cdot v(s') \right\}$$

7. $G(v)$ as the Optimal Policy:

$$B_{G(v)}v = B_*v$$

for any value function v, i.e., policy $G(v)$ achieves the max in B_*.

Contraction and Monotonicity of Operators

1. L_∞ γ-Contractible Operators: Both B_π and B_* are γ-contraction operators in L_∞ norm, i.e., for any two value functions v_1 and v_2

$$\|B_\pi v_1 - B_\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

$$\|B_* v_1 - B_* v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

2. Determination of the Fixed Point: Therefore, the Contraction Mapping Theorems are used to determine the fixed points.
3. Notation for Vector Comparison: The notation

$$v_1 \leq v_2$$

for any two value functions v_1 and v_2 means

$$v_1(s) \leq v_2(s)$$

for all

$$s \in \mathcal{S}$$

4. Monotonicity of B_π and B_*: Both B_π and B_* are monotonic, i.e., for any two value functions v_1 and v_2

$$v_1 \leq v_2 \Longrightarrow B_\pi v_1 \leq B_\pi v_2$$

$$v_1 \leq v_2 \Longrightarrow B_* v_1 \leq B_* v_2$$

Policy Evaluation

1. Bellman Expectation Equation: B_π satisfies the conditions of Contraction Mapping Theorem, and has a unique fixed point v_π, i.e.

$$B_\pi v_\pi = v_\pi$$

2. Policy Evaluation Algorithm: Starting with any value function v and repeatedly applying B_π, on reaches v_π:

$$\lim_{N \to \infty} B_{\pi,N} v = v_\pi$$

for any value function v

Policy Improvement

1. k^{th} Policy Iteration: Let π_k and v_{π_k} denote the policy and the value function for the policy in iteration k of the Policy Iteration.
2. Deterministic Greedy Policy Improvement Step:

$$\pi_{k+1} = G\left(V_{\pi_k}\right)$$

3. Expression for $B_* v_k$: Earlier, it was defined that

$$B_* v = B_{G(v)} v$$

for any value function v. Therefore

$$B_* v_{\pi_k} = B_{G\left(v_{\pi_k}\right)} v_{\pi_k} = B_{\pi_{k+1}} v_{\pi_k}$$

4. Relation between $B_{\pi_{k+1}} v_{\pi_k}$ and v_{π_k}: It is also established earlier that

$$B_* v \geq B_\pi v$$

for all π, v

$$B_* v_{\pi_k} \geq B_{\pi_k} v_{\pi_k} = v_{\pi_k}$$

Therefore, using the previous two observations

$$B_{\pi_{k+1}} v_{\pi_k} \geq v_{\pi_k}$$

5. Monotonicity of $B_{\pi_{k+1}}$: Monotonicity of $B_{\pi_{k+1}}$ implies

$$B_{\pi_{k+1},N} v_{\pi_k} \geq \cdots \geq B_{\pi_{k+1},2} v_{\pi_k} \geq B_{\pi_{k+1}} v_{\pi_k} \geq v_{\pi_k}$$

$$v_{\pi_{k+1}} = \underset{N \to \infty}{limit} B_{\pi_{k+1},N} v_{\pi_k} \geq v_{\pi_k}$$

Policy Iteration

1. When Equality becomes Inequality: It was shown that in iteration $k+1$ of Policy Iteration

$$v_{\pi_{k+1}} \geq v_{\pi_k}$$

If

$$v_{\pi_{k+1}} = v_{\pi_k}$$

the above inequalities hold as equalities.

2. Unique Fixed Point v_*: The above means that

$$B_* v_{\pi_k} = v_{\pi_k}$$

But B_* has a unique fixed point v_*, so

$$v_{\pi_k} = v_*$$

3. Termination Criterion for Policy Iteration: Thus, at each iteration, the policy iteration either strictly improves the value function or achieves the optimal value function v_*.

Value Iteration

1. Bellman Optimality Function: B_* satisfies the conditions of the Contraction Mapping Theorem. Further, B_* has a unique fixed point v_*, i.e.

$$B_* v_* = v_*$$

2. Value Iteration Algorithm: Starting with a value function v and repeatedly applying B_*, one reaches v_*:

$$\lim_{N \to \infty} B_{*,N} v = v_*$$

Greedy Policy from an Optimal Value Function is an Optimal Policy

1. Deterministic Greedy Policy – Recap: It was seen earlier that

$$B_{G(v)} v = B_* v$$

for any function v. Therefore

$$B_{G(v_*)} v_* = B_* v_*$$

2. Fixed Point of B_*: But v_* is the fixed point of B_*, i.e.

$$B_* v_* = v_*$$

Therefore

$$B_{G(v_*)} v_* = v_*$$

3. Unique Fixed Point of $B_{G(v_*)}$: But $B_{G(v_*)}$ has a unique fixed point $v_{G(v_*)}$. Therefore

$$v_* = v_{G(v_*)}$$

4. $G(v_*)$ as the Optimal Deterministic Policy: The above simply states that following the deterministic greedy function $G(v_*)$ - created from the optimal value function v_* - in fact achieves the optimal value function v_*. Therefore, $G(v_*)$ is an optimal deterministic policy.