SFWR ENG 4O03

Kemal Ahmed

Dr. Deza

Fall 2015

Contents

[Linear 1](#_Toc436247936)

[Converting constraints to equalities 2](#_Toc436247937)

[e.g.) 2](#_Toc436247938)

[Graphical Method 2](#_Toc436247939)

[Simplex Method: Maximization 2](#_Toc436247940)

[Simplex: Minimization 3](#_Toc436247941)

[e.g.) 3](#_Toc436247942)

[Phase Simplex 3](#_Toc436247943)

[Phase I 3](#_Toc436247944)

[Phase II 4](#_Toc436247945)

[Bland’s Rule 4](#_Toc436247946)

[Algorithms 4](#_Toc436247947)

[Constraint Graph 4](#_Toc436247948)

[Maximum Flow 4](#_Toc436247949)

[Ford-Fulkerson algorithm 4](#_Toc436247950)

# Linear

**Linear Program**: an optimization problem in which the objective function is linear and each constraint is a linear inequality or equality

**Decision variables**: describe our choices that are under our control

**Objective function**: describes a criterion that we wish to max/minimize; doesn’t have an in/equality

e.g. max 40x + 30y

**Integer linear program**: a linear program that only deals with integers

**Constraints**: describe the limitations that restrict our choices for our decision variables, always *inequalities*.

**Free**: no constraints

**Basic variable**: the variables corresponding to the identity matrix, usually have to be set to 0

**Non-basic variable**: …not basic variables

## Converting constraints to equalities

**Slack variable**: basic variable greater than constraint, added to turn inequalities into equalities

**Surplus variable**: equation variable less than constraint, subtracted

**Hyperplane**: a hyperplane in Rx is a shape in Rx–1, e.g. line in R2

**Optimal Solution**: either a maximum or minimum of the objective function based on constraints

**Basic Solution**: a solution which has as many slack variables as basic variables

**Basic Feasible Solution**: all basic variables are non-negative

* Unique
* obtained by setting the non-basic variables to 0

**Standard form**: when you take inequalities and use slack variables to turn them into equalities.

* Note: all variables need to be ≥ 0.
* All remaining constraints are expressed as equality constraints.

### e.g.)

2x1 + 4x2 – x3 – x4 ≥ 1

2x1 + 4x2 – x3 – x4 + s = 1

## Graphical Method

1. Sketch the region corresponding to the system of constraints. The points inside or on the boundary of the region are the *feasible solutions*.
2. Find the vertices of the region.
3. Test the objective function at each of the vertices and select the values of the variables that optimize the objective function. For a bounded region, both a minimum and maximum value will exist. For an unbounded region, if an optimal solution exists, then it will occur at a vertex.

## Simplex Method: Maximization

**Simplex Method**: useful for solving linear optimization problems cheaply

* Cannot be done with **strict inequalities**, i.e. when there is no possibility of being equal
* Can only work if your objective function is in *standard form*

**Simplex Tableau**: visual representation of stuff

* The *basic variables* can be identified if they have a column with one row of 1 and the rest of the rows are 0’s. The value of the variable is at the row with the 1.
* The objective row is going to identify the constants for the new equation. You should see 0’s in the columns that are non-basic.
* The first column (if used) is only an indicator of the existence of the variable you’re trying to min/maximize, i.e. 0’s for all rows, except for the objective function
* RHS must be ≥ 0

Process:

1. You’ll have as many slack variables as you have constraint equations.
2. Find the column with the smallest coefficient (< 0) in the objective function. That column is called the **pivot column**. The **entering variable** is the variable with the smallest coefficient.
3. **Minimum test**: find the row with the smallest **departing variable** or **exiting variable**, i.e. RHS/xpivot. That row is called the **pivot row**. xpivot must be ≥ 0
4. The intersection of the pivot row & column is called the **pivot point**.
5. If your pivot point ≠ 1, divide your row out by the value of your point
6. Use row operations, i.e. Gauss-Jordan to make the other elements in the pivot column 0.
7. Go to step 2, until objective function is all ≥ 0.

## Simplex: Minimization

To minimize a function, we just oppositize the problem so we can use the maximization technique on it. You’ll see. Just remember that we minimize [w] & maximize [z] AND minimize is (vars ≥ 0), while maximize is (vars ≤ 0). I’ll explain using an example:

### e.g.)

w = 0.12x1 + 0.15x2

60x1 + 60x2 ≥ 300

12x1 + 6x2 ≥ 36

10x1 + 30x2 ≥ 90

1. Ignore slack variables for now. Make a matrix with just the variables you have.

|  |  |  |  |
| --- | --- | --- | --- |
| w | x1 | x2 |  |
| 0 | 60 | 60 | 300 |
| 0 | 12 | 6 | 36 |
| 0 | 10 | 30 | 90 |
| 1 | –0.12 | –0.15 | 0 |

1. Find the transpose of this matrix

|  |  |  |  |
| --- | --- | --- | --- |
| 60 | 12 | 10 | –0.12 |
| 60 | 6 | 30 | –0.15 |
| 300 | 36 | 90 | 0 |

This gives us:

z = 300y1 + 36y2 + 90y3

60y1 + 12y2 + 10y3 ≤ 0.12

60y1 + 6y2 + 30y3 ≤ 0.15

300y1 + 36y2 + 90y3 ≤ 0

Notice how the x’s are now y’s? Yeah I know you did. Well now, since you turned this into a maximization problem, what are you waiting for? [Go to the maximization section](#_Simplex_Method:_Maximization)!

## Phase Simplex

This is useful for when you have a mix of constraints that are maximum and minimum constraints.

**Artificial Variable** [y]: since you can’t have negative variables (x1, x2 ≥ 0), you can’t just use a regular slack variable

### Phase I

1. Replace all negative slack variables with artificial variables
2. Replace objective function with w = –Σyi
3. Isolate your artificial variables in your constraint equations,
   1. e.g. 2x1 + x2 − x3 − x4 + y2 = 10 => y2 = 10 – 2x1 – x2 + x3 + x4
4. Replace your y’s in your objective function with the isolated artificial variables, then move the RHS’s to the new RHS
   1. e.g. for x1 + x2 − x3 − x4 + y2 = 10 & −x2 + x4 + y3 = 10, w – 2x1 + x3 = –20
5. Treat as [maximization](#_Simplex_Method:_Maximization).

### Phase II

Oh no!

## Bland’s Rule

**Bland’s Rule**: a way of guaranteeing that you don’t repeat going over the same variables (a cycle) by picking the smallest (or most negative) number

# Algorithms

See [SFWR ENG 2C03 Summary](https://drive.google.com/file/d/0BxW61uJyyN8TcDRiOWFuR1BpNjg/view).

Bellman-Ford vs Dijkstra’s:

Dijkstra’s omits the possibility that past nodes can be improved. Bellman-Ford makes sure that old nodes have been covered. If you have already looked at a node, but the minimum path to the node changes, you have to re-look at the node as well as all nodes connected to it.

# Constraint Graph

How do these work?

a – b ≤ db-a

# Maximum Flow

## Ford-Fulkerson algorithm

G(V,E)

Incoming flow = outgoing flow for each vertex

In the end, you’re good when back edges are 0 and most forward edges are full