Project 1 : Simplex Method Animation
Subject: Linear Programming - Math Modeling

Application Overview

The simplex method is one of the most popular methods to solve linear programming problems. These types of problems consist of optimizing a linear objective function subject to a set of linear constraints. In this application, we animate the simplex method to solve a user-defined linear programming problem.

Model Definition and Assumptions

The simplex method, developed by George Dantzig in 1947, maintains a basic feasible solution at every step. Given a basic feasible solution, the method first applies the optimality criteria to test the optimality of the current solution. If it does not fulfill this condition, then the algorithm performs a pivot operation to obtain another basis structure with a lower or the same cost. The simplex method repeats this process until the current basic feasible solution satisfies the optimality criteria.

We will now describe the simplex algorithm using a numerical example. Let’s consider the following linear programming problem in which we seek to maximize a value written in terms of four variables. This objective is limited by three constraints and a non-negative variable requirement.

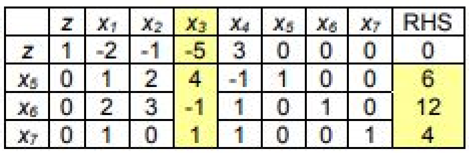
Maximize: $z = 2x_1 + x_2 + 5x_3 - 3x_4$

Subject to:
$x_1 + 2x_2 + 4x_3 - x_4 \le 6$
$2x_1 + 3x_2 - x_3 + x_4 \le 12$
$x_1 + x_3 + x_4 \le 4$
$x_1, x_2, x_3, x_4 \ge 0$

The problem must first be modified to canonical form before the simplex method can be applied. The addition of slack variables and the transformation to canonical form restates the problem as follows:

Maximize: $z = 2x_1 + x_2 + 5x_3 - 3x_4 + 0x_5 + 0x_6 + 0x_7$
subject to:
$x_1 + 2x_2 + 4x_3 - x_4 + x_5 = 6$
$2x_1 + 3x_2 - x_3 + x_4 + x_6 = 12$
$x_1 + x_3 + x_4 + x_7 = 4$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \ge 0$

Note that there are now seven variables and three constraints and that adding the slack variables has not changed the value of the objective function or constraints. The coefficients of all of the variables in the objective function and constraints can now be written as a matrix, or tableau. Here is a representation of the initial tableau:

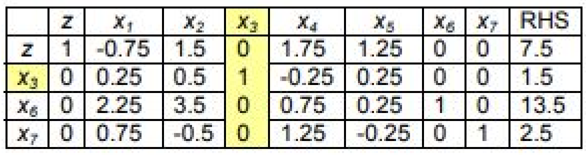


	z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
z	1	-2	-1	-5	3	0	0	0	0
x_5	0	1	2	4	-1	1	0	0	6
x_6	0	2	3	-1	1	0	1	0	12
x_7	0	1	0	1	1	0	0	1	4

The coefficient values of the constraints’ slack variables form an identity matrix. This tableau is used to perform the pivot operations for each iteration of the simplex method. These operations identify a nonbasic entering variable that has the largest negative value (for maximization problems) or the largest positive value (for minimization problems) in the objective function.

The application then compares the ratios of the corresponding column coefficients to the RHS column coefficients to find the basic variable with the minimum ratio. This variable is the leaving variable. Then, the application performs pivot operations to make this column have 0 and 1 coefficient values (to become part of the identity matrix) such that the 1 coefficient value is in the row of the leaving variable. The leaving and entering variables are then switched to complete the iteration.

For example, the first iteration of the simplex method for the problem defined above selects x3 as the entering variable since it has the largest negative coefficient in the objective function (-5). It then compares the ratios of 6/4, -12/1, and 4/1; excluding the negative ratio, the 6/4 ratio can be declared as the minimum ratio, thus identifying x5 as the leaving variable. Pivot operations is then performed and the variables are switched to yield the resulting tableau for the first iteration:



	z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
z	1	-0.75	1.5	0	1.75	1.25	0	0	7.5
x_3	0	0.25	0.5	1	-0.25	0.25	0	0	1.5
x_6	0	2.25	3.5	0	0.75	0.25	1	0	13.5
x_7	0	0.75	-0.5	0	1.25	-0.25	0	1	2.5

Input

- Initial tableau
- Number of variables (after transformation to canonical form)
- Number of constraints
- Maximization or minimization objective
- Entering variable for each iteration

Task:

- Implementation of logic to generate following output.
    - Leaving variable for selected entering variable
    - Tableau for each iteration
    - Objective function value for each iteration
    - Final tableau
    - Chart of change in objective function over all iterations
    - Table of entering variable, leaving variable, minimum ratio, reduced cost, and objective function value for each iteration.
    - Populate the final results on screen.

Note: User interface code and further details will be provided during the course.

Reference: Winston, Wayne L., and Jeffrey B. Goldberg. *Operations research: applications and algorithms*. Vol. 3. Belmont: Thomson Brooks/Cole, 2004.

Project 2 : Projectile Motion
Subject: Linear Programming - Math Modeling

Application Overview

Projectile motion is an important topic in physics. In this application, we allow a user to visualize and analyze the motion of a projectile in three projectile situations: when an object is shot from cannon; when it is dropped from a plane; and when it is thrown off a cliff. This tool can be used by instructors while teaching projectile motion or by students in order to better understand this subject.

Model Definition and Assumptions

We use standard physics equations to calculate the projectile motion. Given the initial condition of the object, we display its motion and plot its velocity and acceleration as a function of time until it hits the ground. The values that define the object's initial condition are the initial height, initial velocity, and initial angle. The user provides these initial values. For the projectile calculations, we will also use values for time, a time increment for animation, x-position (or distance) and y-position (or height), and x-velocity, and y-velocity. Additionally, we will employ a constant for the gravitational force, which we set at 9.81. We assume that, initially, the x-position and y-position are zero.

With the following equation, we calculate the total time that the projectile motion requires:

$$t = (v_0 * \sin(\theta_0) + \sqrt{((v_0 * \sin(\theta_0))^2 + 2 * g * y_0)}\ ) / g$$

[Total Time = (Initial Velocity * Sin(Initial Angle) + √((Initial Velocity * (Sin(Initial Angle)))^2 + 2 * Gravitational Force * Initial Height)) / Gravitational Force]

We calculate the total distance traveled as follows:

$$d = v_0 * \cos(\theta_0) * t$$

[Total Distance = Initial Velocity * (Cos(Initial Angle)) * Total Time]

We calculate the initial x-velocity and y-velocity with the following two equations:

$$vx_0 = v_0 * \cos(\theta_0)$$

[Initial X-Velocity = Initial Velocity * (Cos(Initial Angle))]

$$vy_0 = v0 * \sin(\theta_0)$$

[Initial Y- Velocity = Initial Velocity * (Sin(Initial Angle))]

We calculate the highest point of the projectile motion and the time at which this maximum y-position is reached as follows: $tymax = vy_0 / g$

[Time at Max Y-Position = Initial Y-Velocity / Gravitational Force]

$y_{max} = y_0 + v_0 * \sin(\theta_0) * ty_{max} - (g/2) * (ty_{max})^2$

[Max Y-Position = Initial Height + Initial Velocity * (Sin(Initial Angle)) * Time at Max Y-Position – (Gravitational Force / 2) * (Time at Max Y-Position)^2]

As we animate the projectile motion, we determine its x-position, y-position, velocity, and angle for each time increment. We do so using the following equations:

$$d = v_0 * \cos(\theta_0) * t$$

[X-Position = Initial Velocity * (Cos(Initial Angle)) * Time]

$y = y_0 + v_0 * \sin(\theta_0) * t - g/2$

[Y-Position = Initial Height + Initial Velocity * (Sin(Initial Angle)) * Time – Gravitational Force / 2]

$v = \sqrt{(v_x^2 + v_y^2)}$

[Velocity = √(X-Velocity^2 + Y-Velocity^2)]

$\theta = \arctan(v_y / v_x)$

[Angle = Arctan(Y-Velocity / X-Velocity)]

We assume that the projectile motion stops at the ground, at a y-position equal to zero.

Input

The input for this application is the initial height, initial velocity, and initial angle for whichever projectile scenario the user chooses.

- Projectile scenario (cannon, plane, or cliff)
- Initial height
- Initial velocity
- Initial angle

Task:

- Implementation of logic to generate following output.
    - Chart of projectile motion
    - X-position, y-position, velocity, and angle for each time increment
    - Total time for projectile motion
    - Highest y-position reached
    - Populate the final results on screen.

Note: User interface code and further details will be provided during the course.

Reference:

Halliday, David, Robert Resnick, and Jearl Walker. *Fundamentals of Physics, Chapters 33-37*. John Wiley & Sons, 2010.

Project 3: Sales Force Allocation
Subject: Statistical Model

Application Overview

This application seeks to determine the optimal allocation of sales force in order to maximize profits. In this application, we consider a customer-call scenario in which a manager is trying to determine how many calls should be made to each customer in the upcoming period. We employ optimization to solve the problem, and a trade off curve option allows users to observe optimized results for various upper bound values on the total number of calls to make.

Model Definition and Assumptions

We begin by prompting the user to enter historical data about the sales force allocation. We assume that this historical data is known and includes the number of calls, unit sales, revenue per sale, and cost per call for all customers. This historical data is used to find s-curve parameters to estimate future profit. In marketing research, the s-curve is considered a more accurate model than the power curve. It is modeled as follows:

$$R = a + \frac{(b-a)S^c}{d+S^c}$$

The Solver finds for each customer the optimal s-curve parameters that minimize the sum of squared errors in the estimated profit and the calculated profit. Given the historical number of calls made to a customer, a variation of this value is found for 0.10, 0.50, 1.50, and 10 times the historical value. Aside from the historical number of calls, which has a corresponding historical unit sales value, the user is asked to estimate the unit sales for the variations of the number of calls. From each of these estimates, an estimated profit value is found. A calculated profit value is then determined using the s-curve parameters and the s-curve model provided above. The Solver is then used to find the best s-curve parameters that minimize the sum of squared errors between these two profit values

Input

The user is prompted for various inputs throughout this application:

- Historical input = number of calls, unit sales, revenue per sale, and cost per call for each customer.
- Estimate input = expected unit sales for variations on the historical number of calls.
- Optimization input = lower and upper bounds on the number of calls that can be made to each customer, up

Task:

- Implementation of logic to generate following output.
- There are two main outputs for this application:
    - Number of calls to make to each customer in order to maximize profits.
    - Trade-off curve of upper bound on total number of calls versus the total overall revenue
- Populate the final results on screen.

Note: User interface code and further details will be provided during the course.

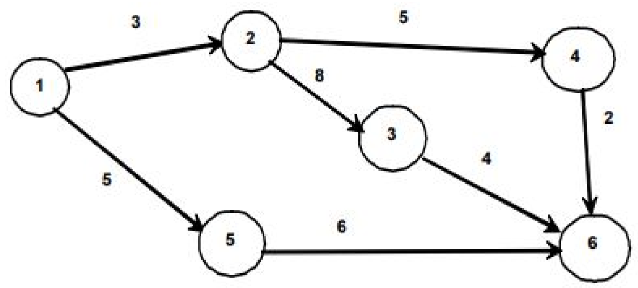
Reference:

Fleisher, Craig S., and Babette E. Bensoussan. *Strategic and competitive analysis: methods and techniques for analyzing business competition*. Upper Saddle River, NJ: Prentice Hall, 2003.

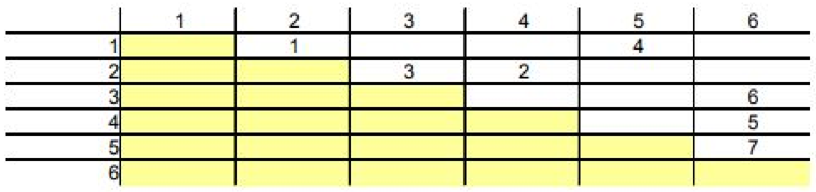
Project 4: Critical Path Method
Subject: Statistical Model

Application Overview
The critical path method is a quantitative technique to manage projects. This technique models a project as a network of several activities. Each activity has an associated activity time that denotes the length of the time needed to complete the activity. Activities also have precedence relationships between them; in other words, an activity can be started only when other activities are completed. The critical path method (CPM) seeks to determine which path, or set, of activities is critical to completing the project. Related to the critical path method, the time-cost tradeoff problem entails crashing, or reducing, the activity times to a crash time at a given crash cost. Depending on the structure of the activities network, there may be several different ways in which the activities can be crashed in order to reduce the overall project time at a minimal cost. As activities are crashed and the project time is reduced, the total project cost is increased, which creates the time-cost tradeoff. The time-cost tradeoff problem is to determine for a desired overall project time which activities should be crashed in order to minimize the total crash cost, or total project cost. In this application, the user can either find the critical path of the project or create a graph of the time-cost tradeoff from crashing activities. For the critical path method option, the user creates a project network, provides the activity times and costs, provides a precedence matrix of the events, and then views the critical path of the project with the total project completion time. For the time-cost tradeoff option, the user creates a project network, provides the activity times and costs along with the crash times and crash costs, provides a precedence matrix of the activities, and then views the time-cost tradeoff as the project is crashed for various project times.

Model Definition and Assumptions
For the CPM option, the user provides information about the activities. Here, we assume that the project network is represented by nodes as events and arcs as activities that connect those events. For example, in the network pictured below, there are six events and seven activities. The numbers written above each activity and each arc are the activity times.



To create the precedence matrix for this network representation, we create a matrix with the number of rows and number of columns equal to the number of events. Then, if any event in row i precedes an event in column j, then the activity that connects these two events is written in cell (i, j). The table below is the precedence matrix corresponding to the network above.



	1	2	3	4	5	6
1		1			4	
2			3	2		
3						6
4						5
5						7
6						

The CPM then computes the early start time (the earliest time the activity can start) and the late start time (the latest time the activity can start) for each activity. Early start times are calculated by analyzing the network activities from the beginning to the end of the project. This analysis is performed using the precedence matrix; the matrix is scanned over each row i = 1 to the number of events and over each column j = 1 to the number of events. The following If, Then statement calculates the early start times based on the activities found in the precedence matrix. [Here CurrentAct is the activity listed in a cell (i, j).]

```
If EarlyTime(j) < (EarlyTime(i) + ActDuration(CurrentAct)) Then
    EarlyTime(j) = (EarlyTime(i) + ActDuration(CurrentAct))
End If
```

The application calculates the late times by analyzing the network activities from the end back to the beginning of the project. This analysis is performed using the precedence matrix; the matrix is scanned over each row i = number of events backwards to 1 and over each column j = number of events backwards to 1. The following If, Then statement calculates the late start times based on the activities found in the precedence matrix. [Here CurrentAct is again the activity listed in a cell (i, j).]

```
If LateTime(i) = 0 or LateTime(i) > (LateTime(j) - ActDuration(CurrentAct)) Then
    LateTime(i) = (LateTime(j) - ActDuration(CurrentAct))
End If
```

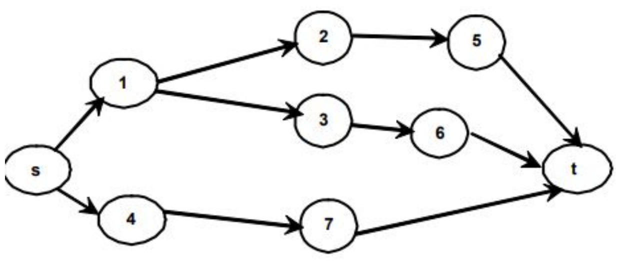
From these times, the free float and total float for each activity are computed using the precedence matrix. If there is an activity in a cell (i, j), then the following calculations are performed:

```
FreeFloat(CurrentAct) = EarlyTime(j) - EarlyTime(i) - ActDuration(CurrentAct)
TotalFloat(CurrentAct) = LateTime(j) - EarlyTime(i) - ActDuration(CurrentAct)
```

The CPM can now determine the critical path of the project. Any activity with a total float time of zero is on the critical path. The total project time and the total project cost are updated based on these critical activities.

```
If TotalFloat(i) = 0 Then
    ProjDur = ProjDur + ActDuration(i)
    ProjCost = ProjCost + NormCost(i)
End If
```

For the time-cost tradeoff option, the user provides the project network and activity information including the crash times and crash costs for each activity. Here, we assume that the project network is represented by nodes as activities, and arcs represent the ordering of those activities. For example, in the network pictured below, there are seven activities. The nodes s and t are dummy sources and sink nodes respectively; they represent the beginning and end of the project.



To create the precedence matrix for this network representation, we develop a matrix with the number of rows and columns equal to the number of activities + 2. (The additional two activities are for the dummy s and t nodes.) Then, if any activity in row i precedes an activity in column j, the number 1 is written in cell (i, j). The table below is the precedence matrix that corresponds to the network above.

	s	1	2	3	4	5	6	7	t
s		1			1				
1			1	1					
2						1			
3							1		
4								1	
5									1
6									1
7									1
t									

The application then solves a linear programming model to find the best selection of activities that should be crashed to minimize the total project cost for a desired project time. We solve this problem iteratively for various desired project times. The linear programming problem is prepared in a hidden worksheet and solved by the Solver. The parts of the model and the formulation for this problem are below. (Figure 1 presents the hidden worksheet with the prepared model.)

Decision Variables:
Start time per activity, u(i)
Crash time per activity, B(i)

Constraints:

- Crash times should be less than or equal to the difference between the activity time and the crash time provided in the activity table.
  B(i) ≤ ActDur(i) – CrashDur(i)
- The total project time must be less than or equal to the desired project time.
- Start time of end – Start time of beginning <= desired project time
  u(t) – u(s) ≤ P
- Using the precedence matrix for every cell (i, j) with a value of 1, the start time of activity j should be greater than or equal to the start time of activity i plus the activity time for i minus the crash time for i.
  start time(j) >= start time(i) + ActDur(i) – CrashTime (i) u(j) ≥ u(i) + ActDur(i) – B(i)

Objective Function:
Minimize total project cost = SUMPRODUCT(decision variables, crash slopes)
(crash slope (i) = ABS((CrashCost(i) – NormCost(i)) / (ActDur(i) - CrashDur(i))))

Model Formulation: Minimize $\sum_i B_i S_i$, S_i = crash slopes or cost per time unit crashed
Subject to:
$B_i \leq A_i - C_i$,
A_i = activity time and C_i = crash time
$u_t - u_s \leq P$
$u_j \geq u_i + A_i - B_i$
$B_i \geq 0$
$u_i \geq 0$

Input
The input for the CPM option is the following:

- Project network with nodes = events and arcs = activities
- Activity times and costs
- Precedence matrix of the events

The input for the time-cost tradeoff option is the following:

- Project network with nodes = activities
- Activity times and costs along with the crash times and crash costs
- Precedence matrix of the activities

Task:

- Implementation of logic to generate following output.
    - The output for the CPM option is the following:
        - Critical path of the project
        - Total project completion time
        - Event early start times and late start times
        - Activity free float and total float times
    - The output for the time-cost tradeoff option is the following:
        - Time-cost tradeoff graph for various project times and costs
        - Populate the final results on screen.

Note: User interface code and further details will be provided during the course.

Reference:
Winston, Wayne L., and Jeffrey B. Goldberg. *Operations research: applications and algorithms*. Vol. 3. Belmont: Thomson Brooks/Cole, 2004.

Project 5: Stochastic Customer Forecasting
Subject: Machine Learning

Application Overview
This application is to forecast stochastic customer arrival while considering a set of factors that influence the forecast. We are looking at a bank scenario in which we are trying to forecast the number of daily customers that arrive at the bank. We consider the following factors that affect this forecast: the month, the day of the week, and such special factors as holidays, periodic days (such as paydays), and ranges of dates (such as Christmas week). Since this stochastic forecasting, also known as regression forecasting, requires up to 15 independent variables, we use a non linear programming model and the Solver as alternatives in order to estimate the coefficients of these factors.

Model Definition and Assumptions

This application determines a constant factor, a factor for each month, a factor for each weekday, and special factors such that the sum of the square errors between the predicted number of customers and the historical number of customers is minimized. To forecast the number of customers with these factors, we use the following equation: Predicted Number of Customers = Constant Factor + Month Factor + Weekday Factor + Sum(Special Factors)

Input
The input for this application is the following:
- Historical data = the number of customers on each workday for an entire calendar year
- Special factors = the holidays, periodic days, or a range of dates with a biased number of customers

Task:
- Implementation of logic to generate following output.
  - A constant factor, month factors, weekday factors, and special factors
  - A predicted number of customers
  - The square error between the predicted number of customers and the historical data
  - A chart of predicted and historical values
  - Populate the final results on screen.

Note: User interface code and further details will be provided during the course.

Reference:
Annis, David H. "Dyadic data analysis." (2007): 371-371.