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.

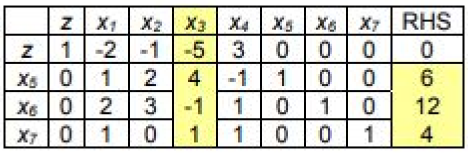
Maximize: $z = 2x_1 + x_2 + 5x_3 - 3x_4$

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

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:
$x_1 + 2x_2 + 4x_3 - x_4 + x_5 = 6$
$2x_1 + 3x_2 - x_3 + x_4 + x_6 = 12$
$x_1 + x_3 + x_4 + x_7 = 4$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0$

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:

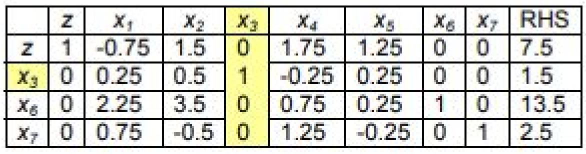


	z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
z	1	-2	-1	-5	3	0	0	0	0
x_5	0	1	2	4	-1	1	0	0	6
x_6	0	2	3	-1	1	0	1	0	12
x_7	0	1	0	1	1	0	0	1	4

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.

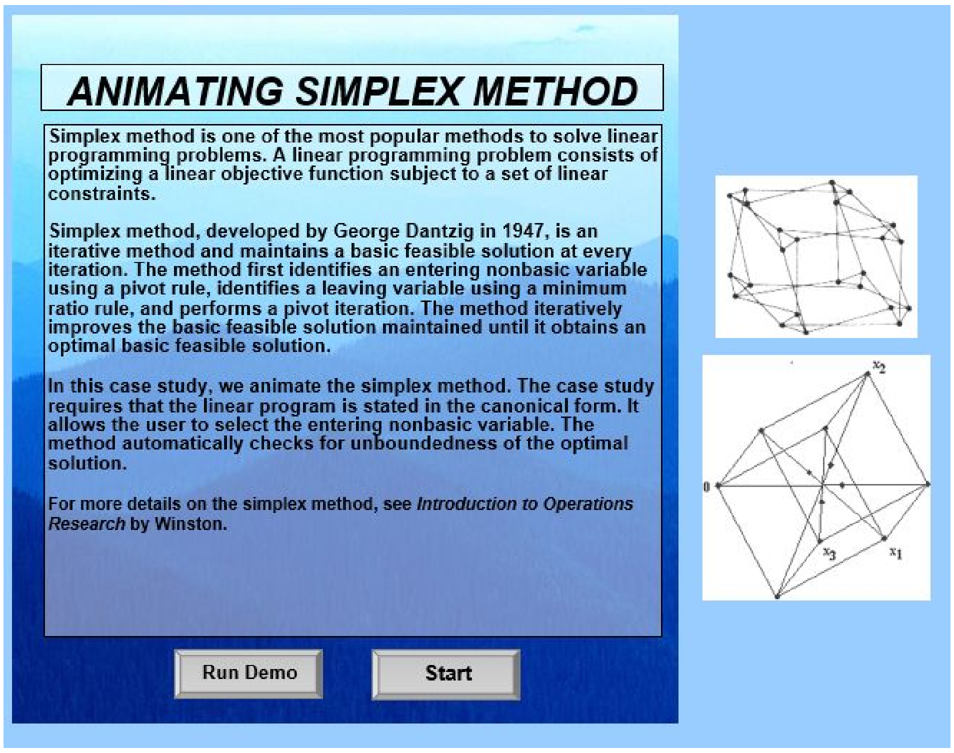
Output

- 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.

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

User Interface

This application requires 4 screens: the welcome screen, the input screen, the example screen and the report screen. The welcome screen contains the title, the description of the application, and the “Run Demo” and “Start” buttons. (See Figure 1.) The “Run Demo” button takes the user to the input screen and creates the initial tableau from the example screen. (See Figure 2.) The “Start” button takes the user to the input screen to create his or her own initial tableau.


ANIMATING SIMPLEX METHOD

Simplex method is one of the most popular methods to solve linear programming problems. A linear programming problem consists of optimizing a linear objective function subject to a set of linear constraints.

Simplex method, developed by George Dantzig in 1947, is an iterative method and maintains a basic feasible solution at every iteration. The method first identifies an entering nonbasic variable using a pivot rule, identifies a leaving variable using a minimum ratio rule, and performs a pivot iteration. The method iteratively improves the basic feasible solution maintained until it obtains an optimal basic feasible solution.

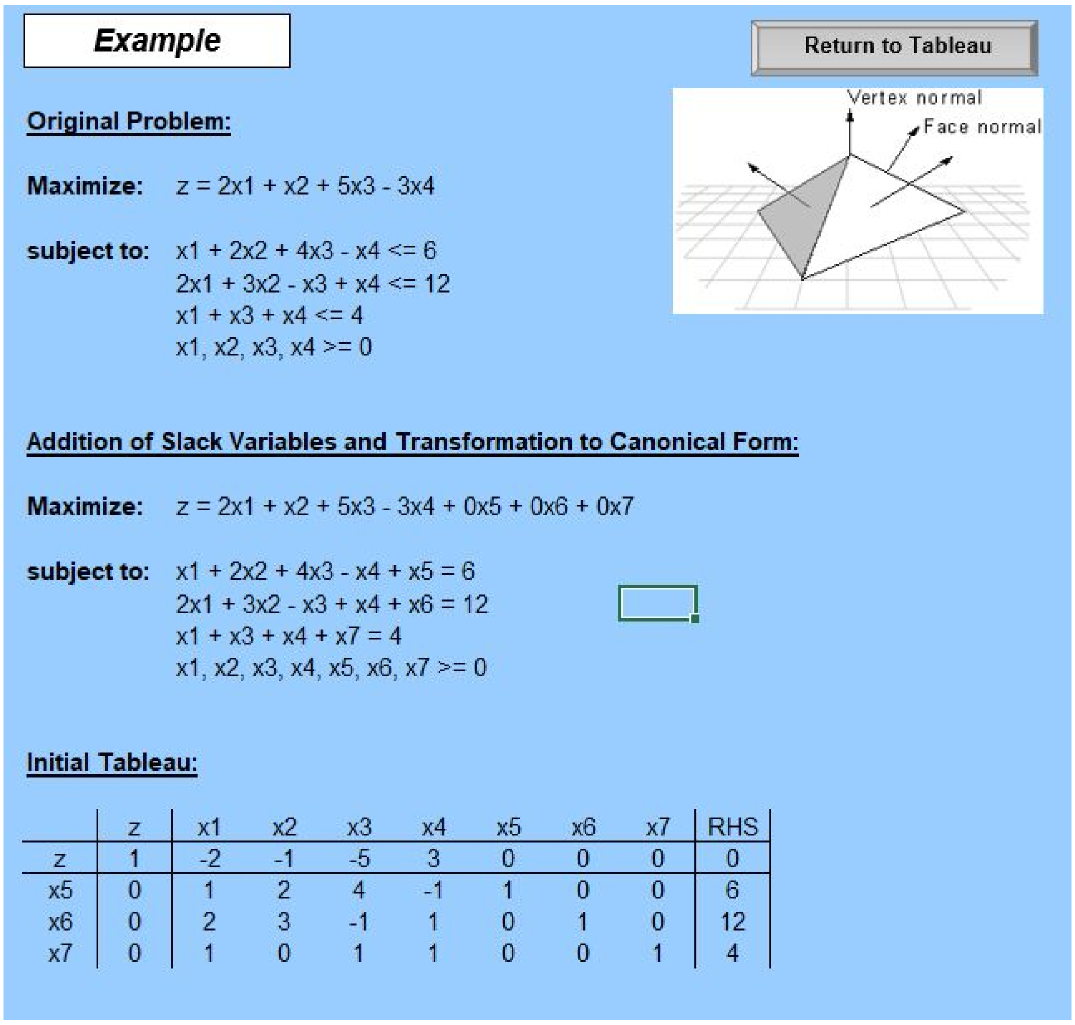
In this case study, we animate the simplex method. The case study requires that the linear program is stated in the canonical form. It allows the user to select the entering nonbasic variable. The method automatically checks for unboundedness of the optimal solution.

For more details on the simplex method, see *Introduction to Operations Research* by Winston.

Figure 1 Welcome Screen

The input screen instructs the user how to create the initial tableau and animate the simplex method iterations. A navigational form, which is always available to the user on the input screen, provides several options for performing the iterations. The user can also view the example screen for guidance in constructing the initial tableau by clicking the “See Example” button. The example screen contains an example of a linear programming problem.It reveals how to transform a problem into canonical form by adding slack variables. It also provides the initial tableau for this problem, which is the same initial tableau used for the demo option.

For this application’s user interface, we use navigational buttons and two user forms. The input screen contains a navigational form that is always shown. It is a dynamic floating form with several different buttons appearing as options, which are made available to the user. When the user first arrives at the input screen, the navigational form appears, as shown in Figure 4(a). Here, the main two options available to the user are “Show Each Iteration” and “Show Final Solution.” When these buttons are clicked, the user’s initial tableau is checked and a second form appears.



Original Problem:

Maximize: z = 2x1 + x2 + 5x3 - 3x4

subject to: x1 + 2x2 + 4x3 - x4 <= 6
2x1 + 3x2 - x3 + x4 <= 12
x1 + x3 + x4 <= 4
x1, x2, x3, x4 >= 0

Addition of Slack Variables and Transformation to Canonical Form:

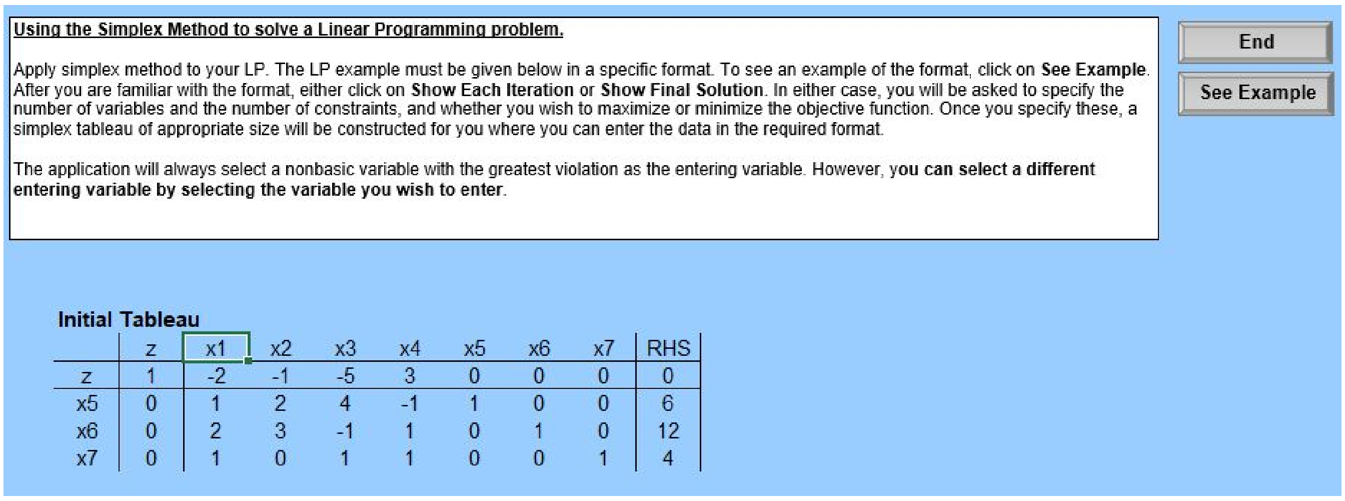
Maximize: z = 2x1 + x2 + 5x3 - 3x4 + 0x5 + 0x6 + 0x7

subject to: x1 + 2x2 + 4x3 - x4 + x5 = 6
2x1 + 3x2 - x3 + x4 + x6 = 12
x1 + x3 + x4 + x7 = 4
x1, x2, x3, x4, x5, x6, x7 >= 0

Initial Tableau:

	z	x1	x2	x3	x4	x5	x6	x7	RHS
z	1	-2	-1	-5	3	0	0	0	0
x5	0	1	2	4	-1	1	0	0	6
x6	0	2	3	-1	1	0	1	0	12
x7	0	1	0	1	1	0	0	1	4

Figure 2 Example Screen



Using the Simplex Method to solve a Linear Programming problem.

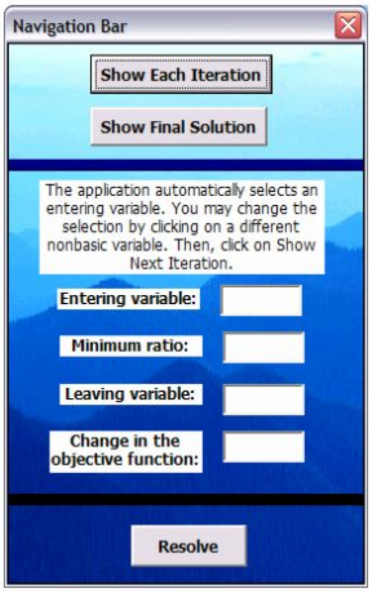
Apply simplex method to your LP. The LP example must be given below in a specific format. To see an example of the format, click on **See Example**. After you are familiar with the format, either click on **Show Each Iteration** or **Show Final Solution**. In either case, you will be asked to specify the number of variables and the number of constraints, and whether you wish to maximize or minimize the objective function. Once you specify these, a simplex tableau of appropriate size will be constructed for you where you can enter the data in the required format.

The application will always select a nonbasic variable with the greatest violation as the entering variable. However, **you can select a different entering variable by selecting the variable you wish to enter**.

Initial Tableau

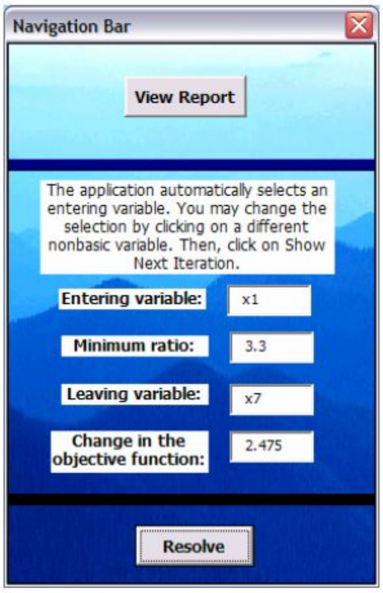
	z	x1	x2	x3	x4	x5	x6	x7	RHS
z	1	-2	-1	-5	3	0	0	0	0
x5	0	1	2	4	-1	1	0	0	6
x6	0	2	3	-1	1	0	1	0	12
x7	0	1	0	1	1	0	0	1	4

Figure 3 Input screen during the simplex method animation.



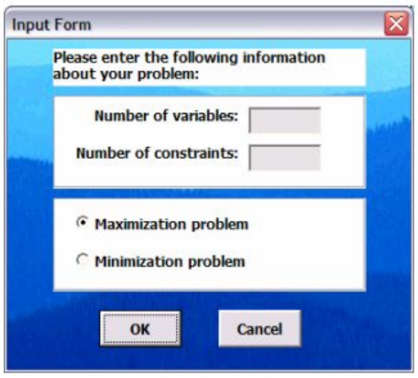
The application automatically selects an entering variable. You may change the selection by clicking on a different nonbasic variable. Then, click on Show Next Iteration.

Entering variable:	
Minimum ratio:	
Leaving variable:	
Change in the objective function:	



The application automatically selects an entering variable. You may change the selection by clicking on a different nonbasic variable. Then, click on Show Next Iteration.

Entering variable:	x1
Minimum ratio:	3.3
Leaving variable:	x7
Change in the objective function:	2.475



Please enter the following information about your problem:

Number of variables:	
Number of constraints:	

[✓ Maximization problem]
Minimization problem

Figure 4(a) 4(b) on left, and 5 on right

This second form is the input form. (See Figure 5.) Here, the user inputs the number of variables (including the slack variables) and the number of constraints in the problem. He or she also specifies if the problem has a maximization or minimization objective.

If the user selects to view each iteration, then the navigational form changes to display a “Show Next Iteration” button. As the iterations are performed, the user can select an entering variable by clicking on the variable name on the current tableau; the corresponding minimum ratio and the leaving variable values are displayed on the navigational form along with the change in the objective function. (See Figure 5.)

When the optimal solution is found, the navigational form changes again to display a “View Report” button. This button takes the user to the report screen, that displays a summary report of the iterations performed. A table lists the entering variable, the leaving variable, the minimum ratio, the reduced cost, and the objective function value for each iteration. The “End” button on the input screen and the report screen takes the user back to the welcome screen. The “See Example” button takes the user to the example screen, and the “Return to Tableau” button on the example screen and on the report screen returns the user to the input screen.

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

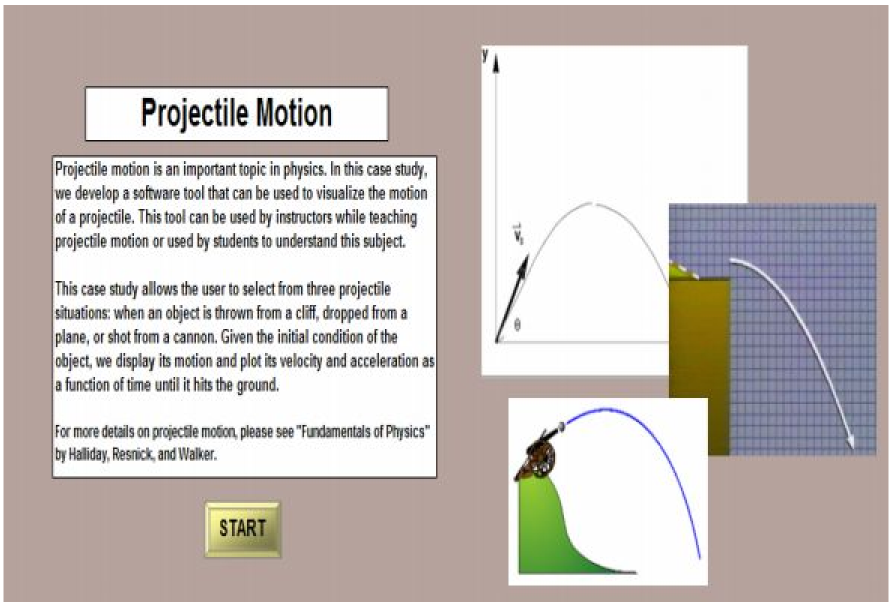
Output

The output for this application is the path of the projectile motion.

- Chart of projectile motion
- X-position, y-position, velocity, and angle for each time increment
- Total time for projectile motion
- Highest y-position reached

User Interface

We use two screens in this application: the welcome screen and the projectile screen. The welcome screen contains the title and the description of the application, as well as an image. (See Figure 1.) The "Start" button on the welcome screen brings the user to the projectile screen.



Projectile Motion

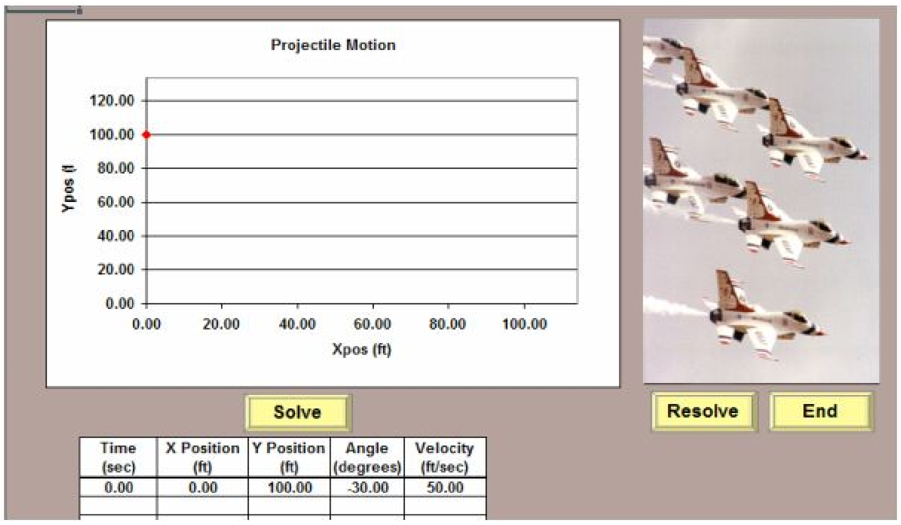
Projectile motion is an important topic in physics. In this case study, we develop a software tool that can be used to visualize the motion of a projectile. This tool can be used by instructors while teaching projectile motion or used by students to understand this subject.

This case study allows the user to select from three projectile situations: when an object is thrown from a cliff, dropped from a plane, or shot from a cannon. 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.

For more details on projectile motion, please see "Fundamentals of Physics" by Halliday, Resnick, and Walker.

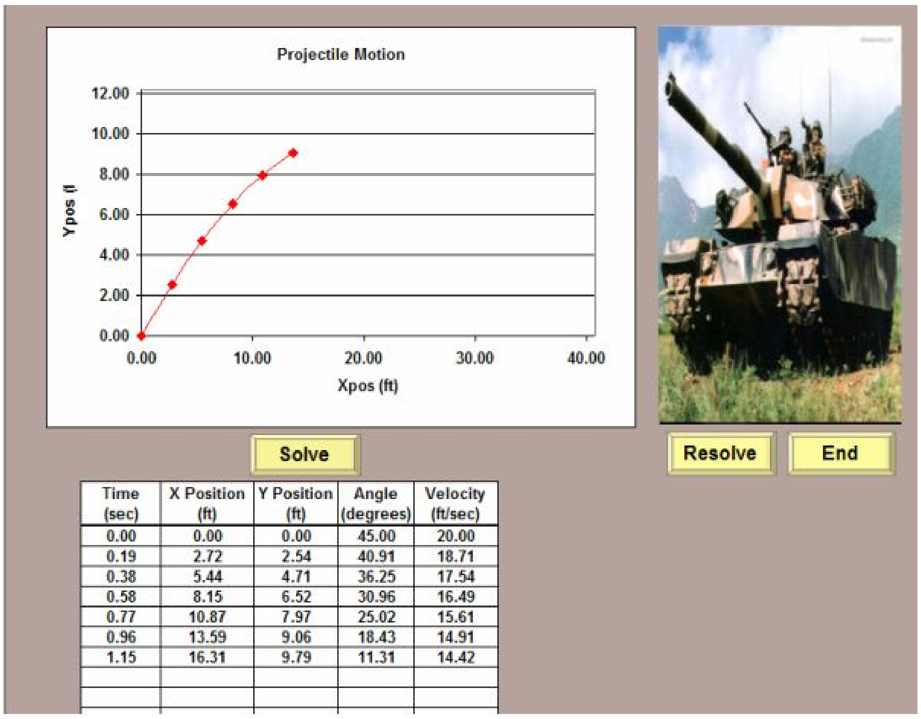
Figure 1 Welcome Screen

The projectile screen includes a chart to animate the projectile motion and a table to display the values of the x-position, y-position, velocity, and angle for each time increment. (See Figure 2.) The picture next to the chart corresponds to the projectile scenario, and the "Solve" button begins the projectile animation. The "Re-solve" button allows the user to change the projectile scenario and/or input values, and, finally, the "End" button brings the user back to the welcome screen. Figures 4, 5, 6 exhibit the final projectile motions for the cannon, plane, and cliff scenarios, respectively. (These particular projectile motions were calculated using default input values.)



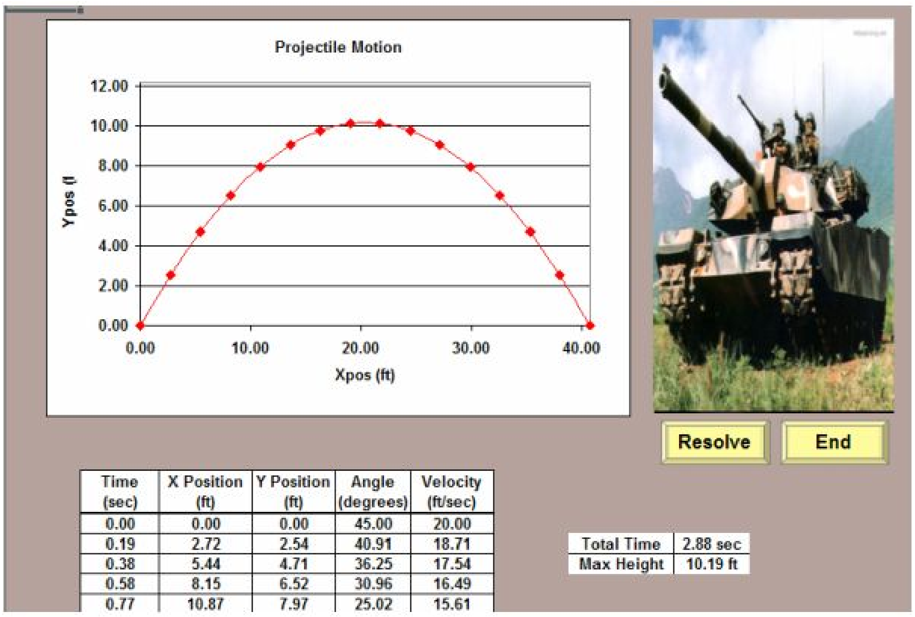
Time (sec)	X Position (ft)	Y Position (ft)	Angle (degrees)	Velocity (ft/sec)
0.00	0.00	100.00	-30.00	50.00

Figure 2 Projectile Screen



Time (sec)	X Position (ft)	Y Position (ft)	Angle (degrees)	Velocity (ft/sec)
0.00	0.00	0.00	45.00	20.00
0.19	2.72	2.54	40.91	18.71
0.38	5.44	4.71	36.25	17.54
0.58	8.15	6.52	30.96	16.49
0.77	10.87	7.97	25.02	15.61
0.96	13.59	9.06	18.43	14.91
1.15	16.31	9.79	11.31	14.42

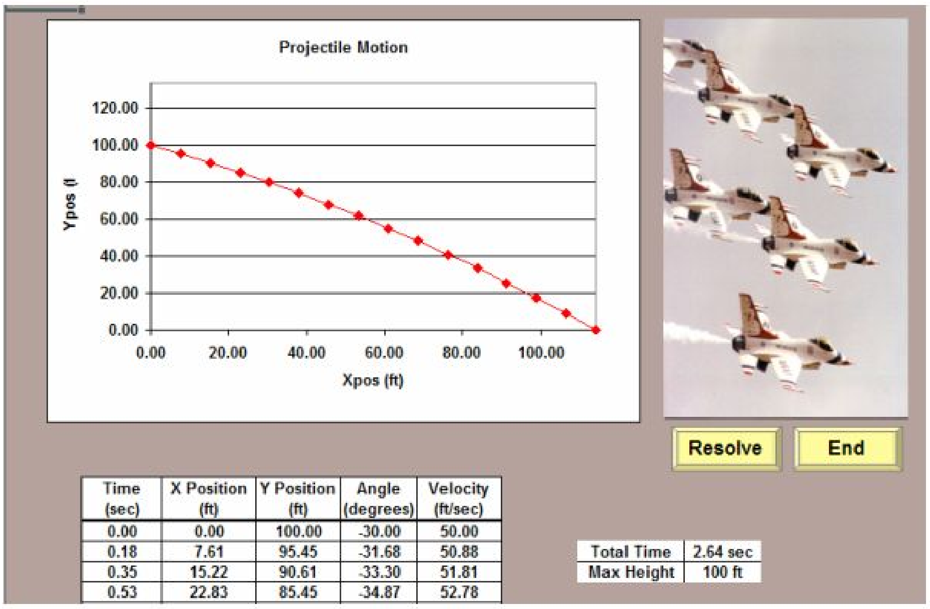
Figure 3 Animating the projectile motion



Time (sec)	X Position (ft)	Y Position (ft)	Angle (degrees)	Velocity (ft/sec)
0.00	0.00	0.00	45.00	20.00
0.19	2.72	2.54	40.91	18.71
0.38	5.44	4.71	36.25	17.54
0.58	8.15	6.52	30.96	16.49
0.77	10.87	7.97	25.02	15.61

Total Time	2.88 sec
Max Height	10.19 ft

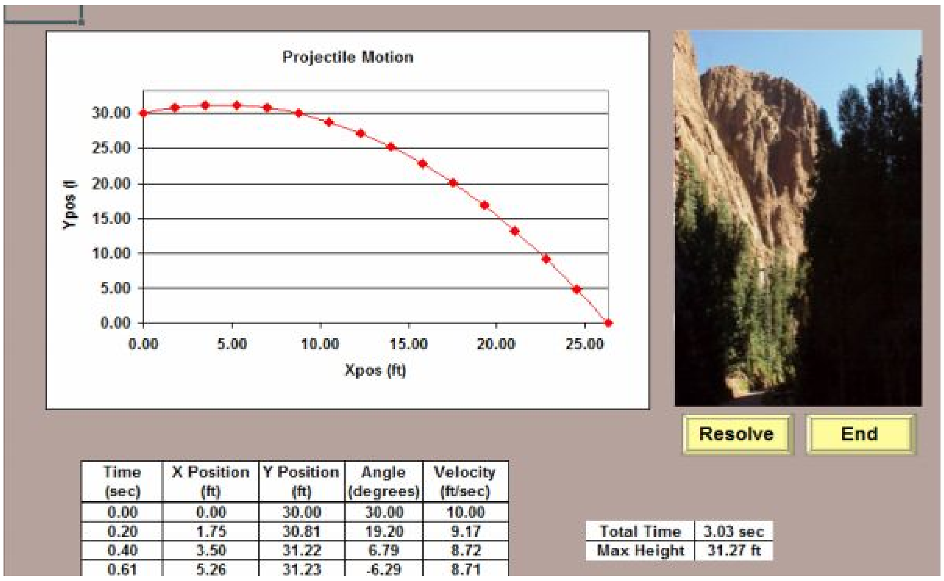
Figure 4 : The projectile motion of an object being shot from cannon.



Time (sec)	X Position (ft)	Y Position (ft)	Angle (degrees)	Velocity (ft/sec)
0.00	0.00	100.00	-30.00	50.00
0.18	7.61	95.45	-31.68	50.88
0.35	15.22	90.61	-33.30	51.81
0.53	22.83	85.45	-34.87	52.78

Total Time	2.64 sec
Max Height	100 ft

Figure 5 The projectile motion of an object being dropped from a plane.



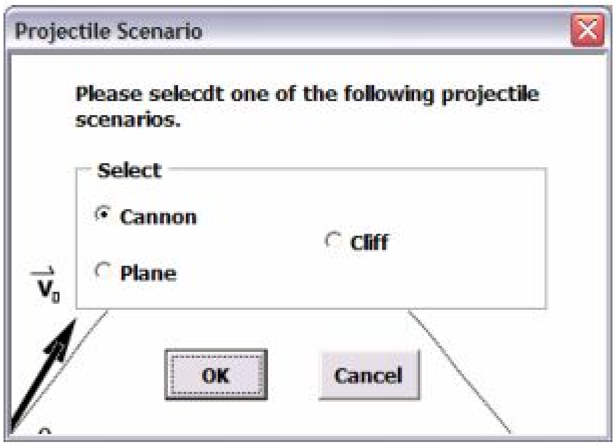
Time (sec)	X Position (ft)	Y Position (ft)	Angle (degrees)	Velocity (ft/sec)
0.00	0.00	30.00	30.00	10.00
0.20	1.75	30.81	19.20	9.17
0.40	3.50	31.22	6.79	8.72
0.61	5.26	31.23	-6.29	8.71

Total Time	3.03 sec
Max Height	31.27 ft

Figure 6 The projectile motion of an object being thrown off a cliff.

For this application's user interface, we use navigational and functional buttons as well as two user forms. On the welcome sheet, the "Start" button brings the user to the projectile sheet. The user is then shown a user form for selecting which projectile scenario he or she wants to analyze. (See Figure 7.) The user can select one of three option buttons: "Cannon," "Plane," and "Cliff."

Once the user has selected a projectile scenario, the input form then appears. (See Figure 8) This form has three text boxes to receive the values for the initial height, initial velocity, and initial angle. We use labels to clarify the units assumed for each value, and we also provide the user with default values. These default values differ depending on the specific projectile scenario. Figures 8, 9, 10 reveal the default values for the cannon, plane, and cliff projectile scenarios, respectively.


Projectile Scenario

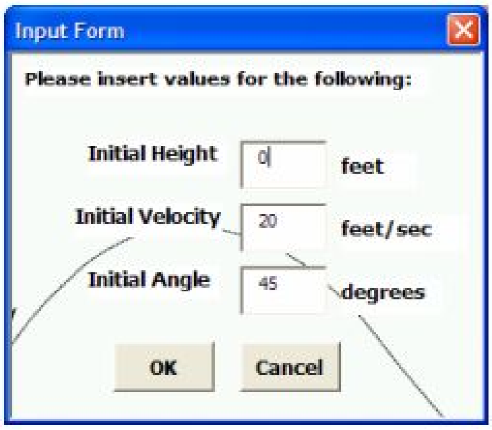
Please selecdt one of the following projectile scenarios.

Select

(•) Cannon
( ) Cliff
( ) Plane

OK Cancel

Figure 7 The “Projectile Scenario” form.


Input Form

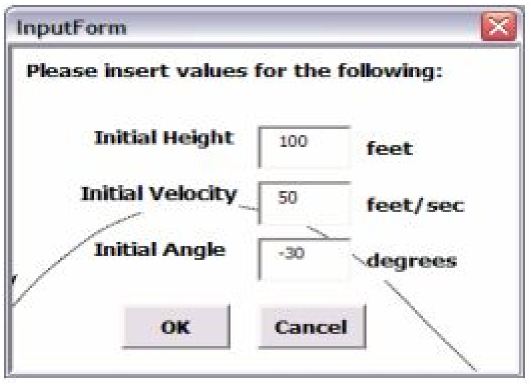
Please insert values for the following:

Initial Height	0	feet
Initial Velocity	20	feet/sec
Initial Angle	45	degrees

OK Cancel

Figure 8 The input form with the default values for the cannon projectile.

The projectile sheet additionally has three buttons: “Solve,” a functional button that begins the projectile animation; “Re-solve,” also a functional button that allows the user to change the projectile scenario and/or input values and re-perform the projectile calculations and animation; and “End,” a navigational button that brings the user back to the welcome sheet.

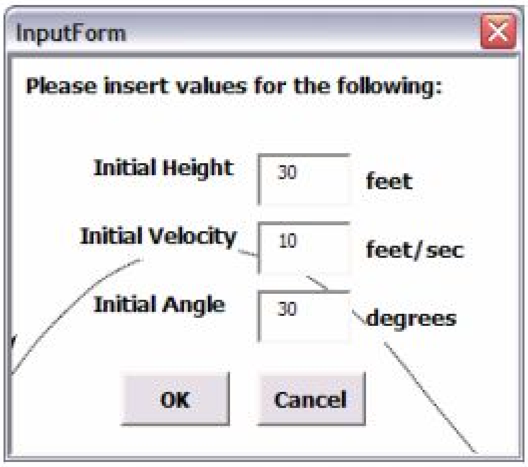

InputForm

Please insert values for the following:

Initial Height	100	feet
Initial Velocity	50	feet/sec
Initial Angle	-30	degrees

OK Cancel

Figure 9 The input form with the default values for the plane projectile.


InputForm

Please insert values for the following:

Initial Height	30	feet
Initial Velocity	10	feet/sec
Initial Angle	30	degrees

Figure 10 The input form with the default values for the cliff projectile.

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

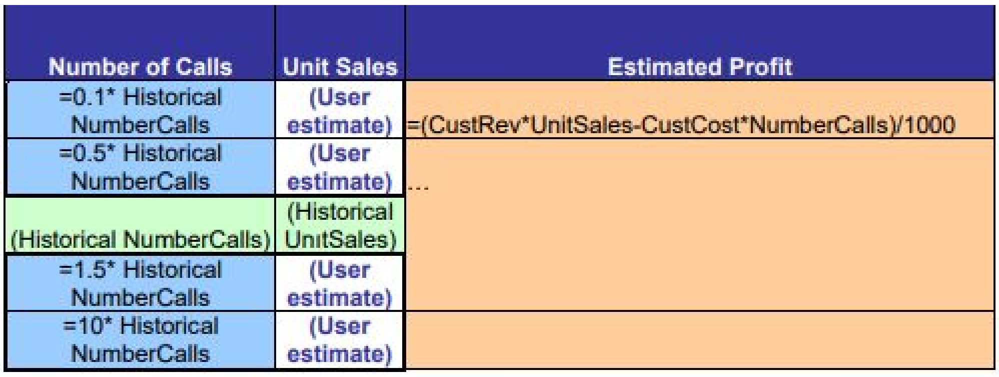
This application seeks to determine the optimal allocation of sales force in order to maximize profits. In this application, we consider a customer-call scenario in which a manager is trying to determine how many calls should be made to each customer in the upcoming period. We employ optimization to 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

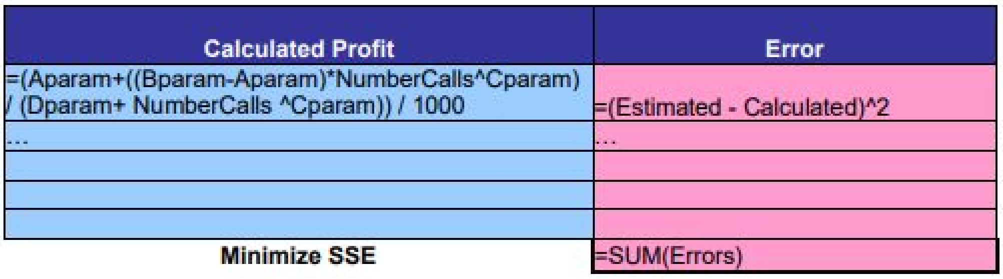
We begin by prompting the user to enter historical data about the sales force allocation. We assume that this historical data is known and includes the number of calls, unit sales, revenue per sale, and cost per call for all customers. This historical data is used to find s-curve parameters to estimate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

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

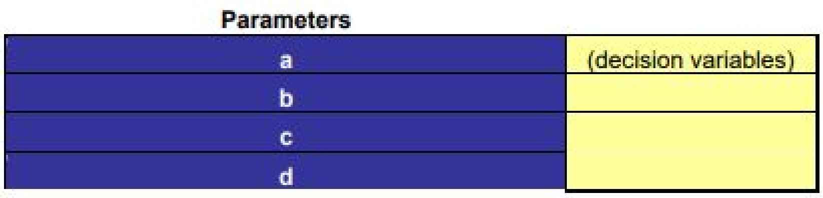
The Solver finds for each customer the optimal s-curve parameters that minimize the sum of squared errors in the estimated profit and the calculated profit. Given the historical number of calls made to a customer, a variation of this value is found for 0.10, 0.50, 1.50, and 10 times the historical value. 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



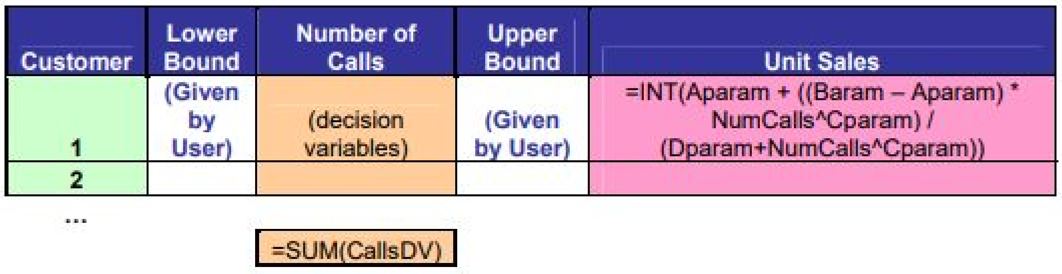
Number of Calls	Unit Sales	Estimated Profit
=0.1* Historical NumberCalls	(User estimate)	=(CustRev*UnitSales-CustCost*NumberCalls)/1000
=0.5* Historical NumberCalls	(User estimate)	...
(Historical NumberCalls)	(Historical UnitSales)	
=1.5* Historical NumberCalls	(User estimate)	
=10* Historical NumberCalls	(User estimate)	



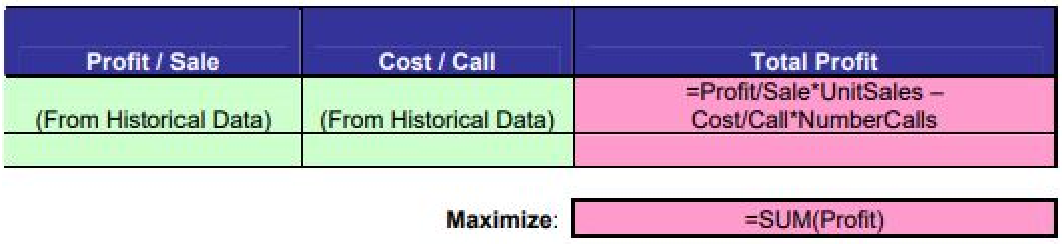
Calculated Profit	Error
=(Aparam+((Bparam-Aparam)*NumberCalls^Cparam) / (Dparam+ NumberCalls ^Cparam)) / 1000	=(Estimated - Calculated)^2
...	...
Minimize SSE	=SUM(Errors)



Parameters	
a	(decision variables)
b	
c	
d	



Customer	Lower Bound	Number of Calls	Upper Bound	Unit Sales
1	(Given by User)	(decision variables)	(Given by User)	=INT(Aparam + ((Baram – Aparam) * NumCalls^Cparam) / (Dparam+NumCalls^Cparam))
2				
...				
		=SUM(CallsDV)		



Profit / Sale	Cost / Call	Total Profit
(From Historical Data)	(From Historical Data)	=Profit/Sale*UnitSales – Cost/Call*NumberCalls

Maximize:	=SUM(Profit)

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

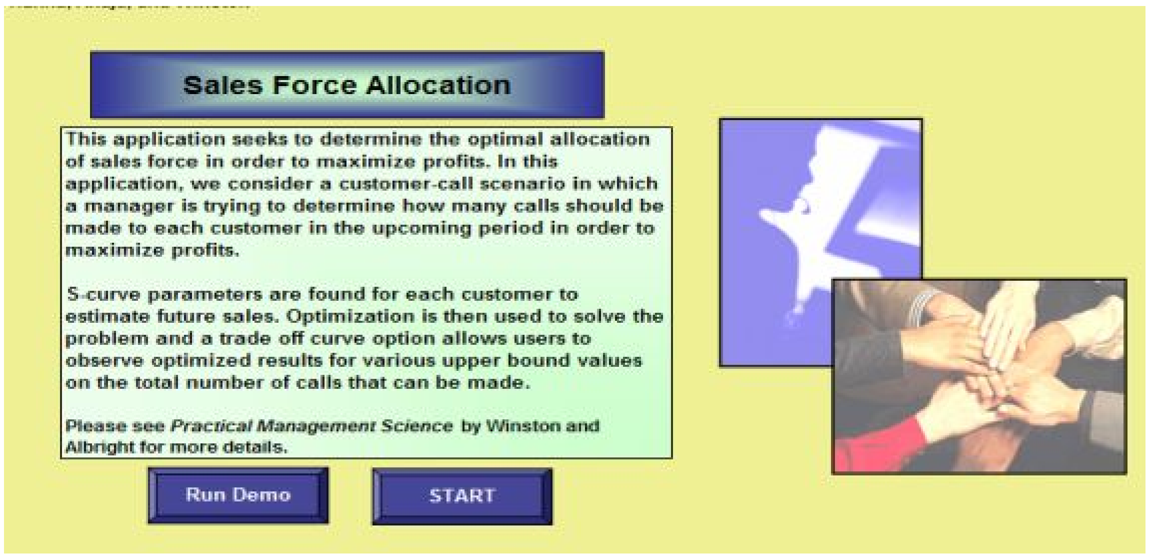
Output

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

User Interface

This application requires five screens: the welcome screen, the historical data screen, the s-curve parameters screen, the optimization screen, and the trade-off curve screen. The welcome screen contains the title, the description of the application, and the “Run Demo” and “Start” buttons. (See Figure 1.) The “Run Demo” button copies some demo historical data and then takes the user to the historical data sheet. The “Start” button on the welcome screen prompts the user for some input and then displays the historical data screen.



Sales Force Allocation

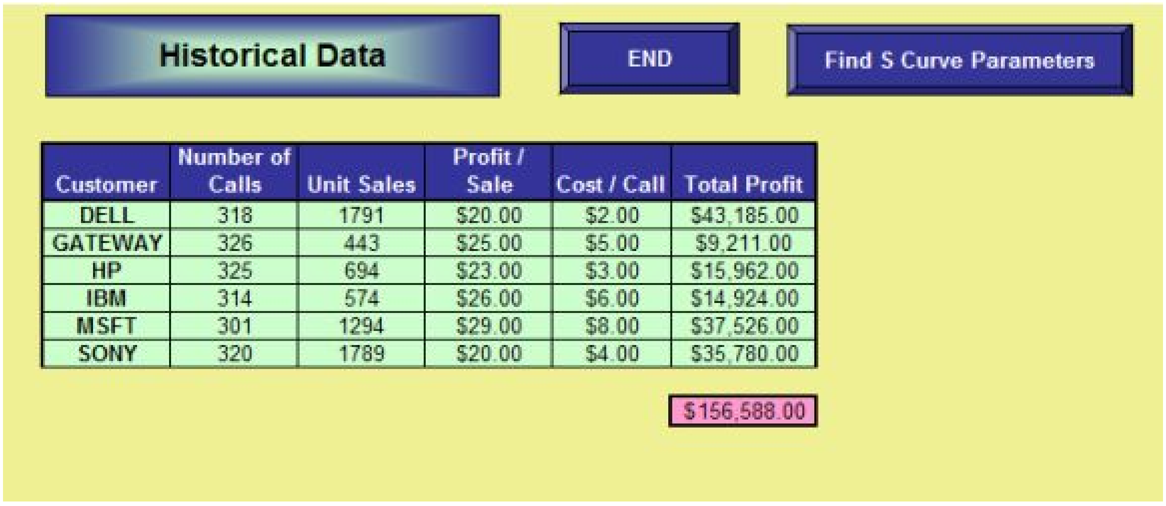
This application seeks to determine the optimal allocation of sales force in order to maximize profits. In this application, we consider a customer-call scenario in which a manager is trying to determine how many calls should be made to each customer in the upcoming period in order to maximize profits.

S-curve parameters are found for each customer to estimate future sales. Optimization is then used 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 that can be made.

Please see *Practical Management Science* by Winston and Albright for more details.

Figure 1 The welcome screen

The historical data screen stores the input for the historical number of calls made to each customer, the unit sales achieved, the profit per unit sale, and the cost per call made. The total profit is calculated per customer with a simple formula of unit sales * profit per sale – number of calls * cost per call. The total profit overall customers is also calculated as the sum of the total profit per customer. Figure 2 presents the historical data screen with the demo data.

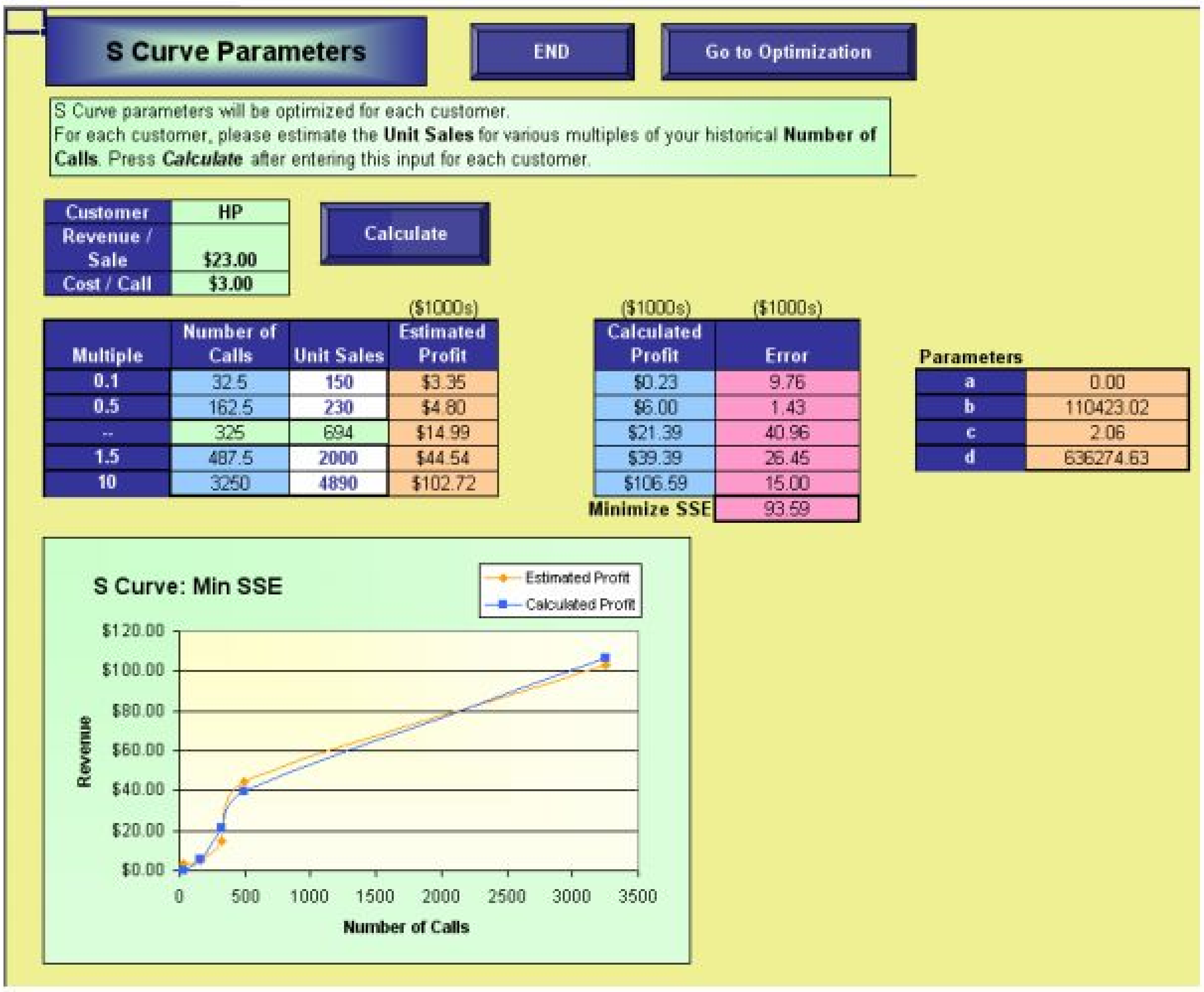


Customer	Number of Calls	Unit Sales	Profit / Sale	Cost / Call	Total Profit
DELL	318	1791	$20.00	$2.00	$43,185.00
GATEWAY	326	443	$25.00	$5.00	$9,211.00
HP	325	694	$23.00	$3.00	$15,962.00
IBM	314	574	$26.00	$6.00	$14,924.00
MSFT	301	1294	$29.00	$8.00	$37,526.00
SONY	320	1789	$20.00	$4.00	$35,780.00

$156,588.00

Figure 2 The historical data screen.

The “Find S Curve Parameters” button takes the user to the s-curve parameters screen, which is where the user optimizes the s-curve parameters for each customer. (See Figure 3.) For each customer, one at a time, the historical data is referenced as input on this screen. The number of calls and unit sales in the historical data are found in the middle row of the main calculation table of this screen. Using the historical number of calls, the application calculates variations of .10, .50, 1.5, and 10 times the historical value. The user is then asked to estimate the unit sales for these variations in the number of calls. With the user’s values, the application calculates an estimated profit for each scenario. The Solver then finds the a, b, c, and d parameter values for the s-curve such that the mean squared error between the calculated profit from these parameters and the estimated profit from the user’s input are minimized. A chart comparing the estimated and calculated profits is shown. Next, the user clicks the “Calculate” button to optimize these s-curve parameters for each customer after he or she has entered the sales estimates. When all the customer’s parameters have been optimized, the user can press the “Go to Optimization” button to proceed to the optimization screen.



S Curve parameters will be optimized for each customer.
For each customer, please estimate the **Unit Sales** for various multiples of your historical **Number of Calls**. Press ***Calculate*** after entering this input for each customer.

Customer	HP
Revenue / Sale	$23.00
Cost / Call	$3.00

Multiple	Number of Calls	Unit Sales	Estimated Profit ($1000s)
0.1	32.5	150	$3.35
0.5	162.5	230	$4.80
--	325	694	$14.99
1.5	487.5	2000	$44.54
10	3250	4890	$102.72

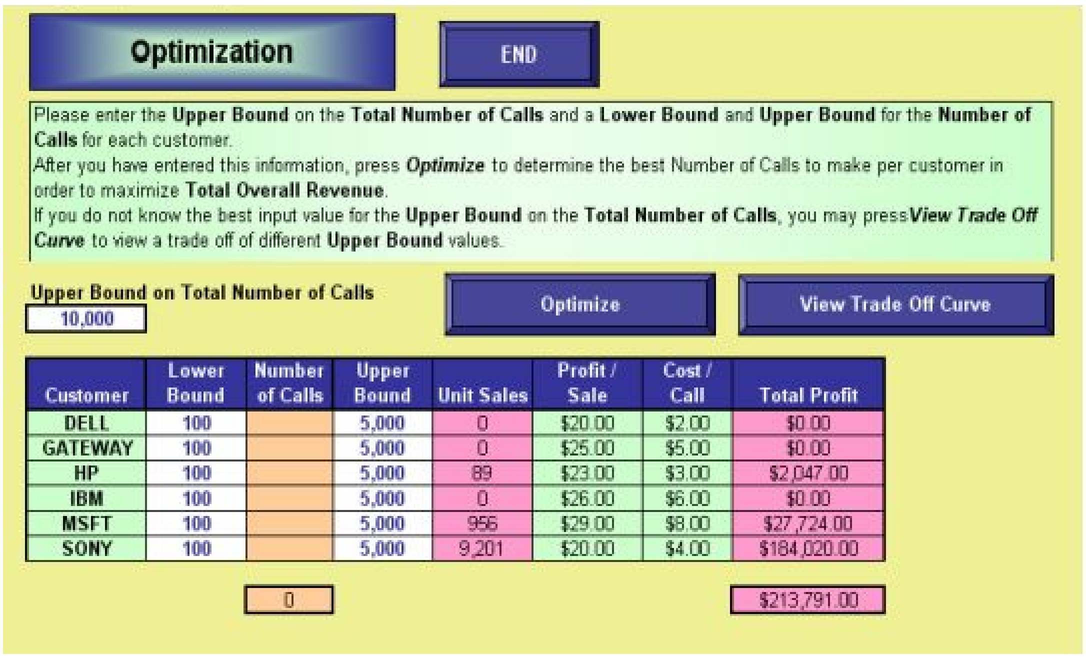
Calculated Profit ($1000s)	Error ($1000s)
$0.23	9.76
$6.00	1.43
$21.39	40.96
$39.39	26.45
$106.59	15.00
Minimize SSE	93.59

Parameters

a	0.00
b	110423.02
c	2.06
d	636274.63

Figure 3 The s-curve parameters sheet.

The main model of the application is solved on the optimization sheet. (See Figure 4.) The user provides a lower bound and an upper bound on the number of calls that can be made to each customer; these are the decision variables. Using the optimized s-curve parameters and these decision variables, the unit sales are calculated. From these unit sales, the profit per sale, and the cost per call input, the total profit per customer is also calculated. The objective function is to maximize the sum of these total profit values for all the customers. When the user presses the "Optimize" button, the model is solved. The user can also specify an upper bound constraint on the total number of calls for all the customers. By selecting the "View Trade Off Curve" button, the user can view the results of the different values of this upper bound for the total number of calls, which are listed on the trade-off curve sheet.



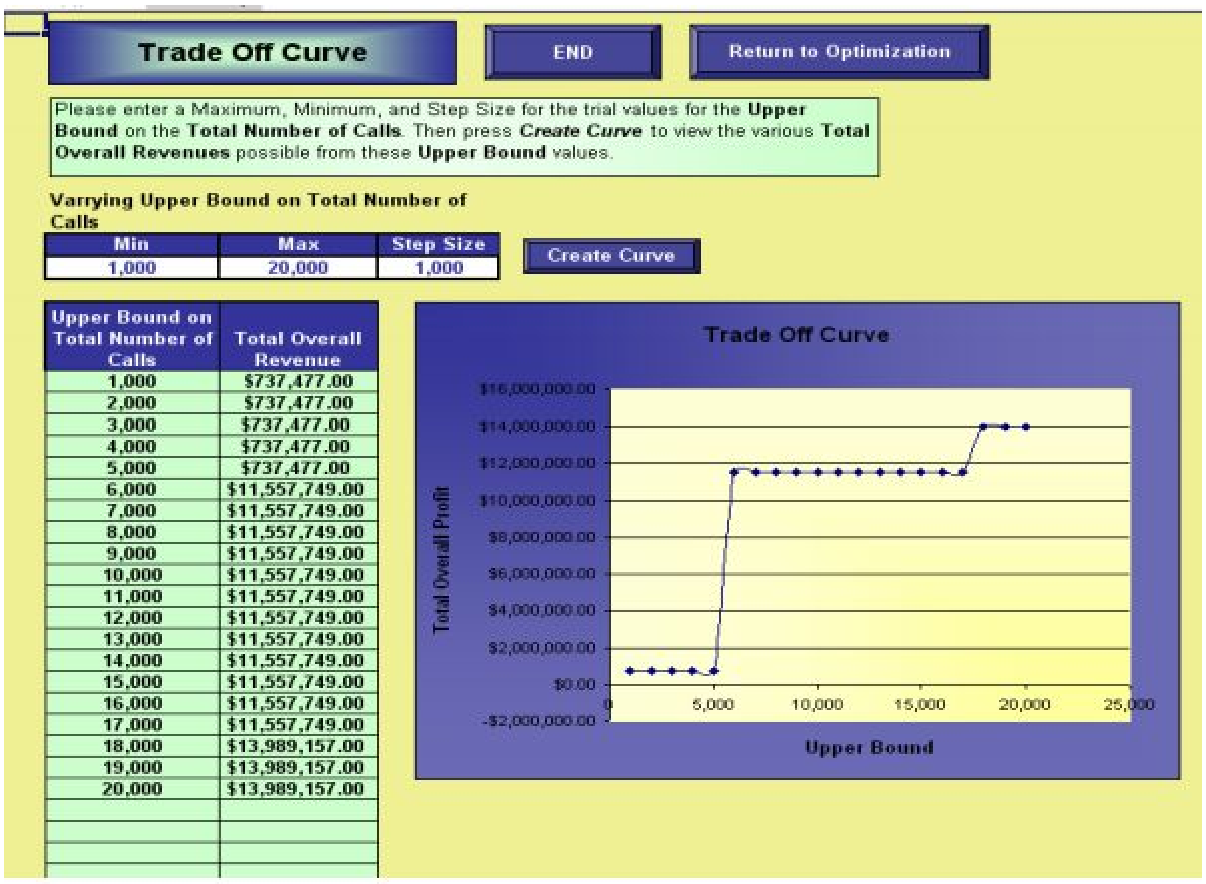
Please enter the **Upper Bound** on the **Total Number of Calls** and a **Lower Bound** and **Upper Bound** for the **Number of Calls** for each customer.
After you have entered this information, press ***Optimize*** to determine the best Number of Calls to make per customer in order to maximize **Total Overall Revenue**.
If you do not know the best input value for the **Upper Bound** on the **Total Number of Calls**, you may press ***View Trade Off Curve*** to view a trade off of different **Upper Bound** values.

Upper Bound on Total Number of Calls
10,000

Customer	Lower Bound	Number of Calls	Upper Bound	Unit Sales	Profit / Sale	Cost / Call	Total Profit
DELL	100		5,000	0	$20.00	$2.00	$0.00
GATEWAY	100		5,000	0	$25.00	$5.00	$0.00
HP	100		5,000	89	$23.00	$3.00	$2,047.00
IBM	100		5,000	0	$26.00	$6.00	$0.00
MSFT	100		5,000	956	$29.00	$8.00	$27,724.00
SONY	100		5,000	9,201	$20.00	$4.00	$184,020.00
		0					$213,791.00

Figure 4 The optimization sheet.

The trade-off curve sheet presents the change in the overall revenue as the user changes the upper bound on the total number of calls to be made. (See Figure 5) The user varies these values by specifying a range and a step size for the trial upper bound values. Once this information has been entered, the user can press the "Create Curve" button to view the trade-off curve for these values. He or she can return to the optimization sheet by pressing the "Return to Optimization" button.



Please enter a Maximum, Minimum, and Step Size for the trial values for the **Upper Bound** on the **Total Number of Calls**. Then press ***Create Curve*** to view the various **Total Overall Revenues** possible from these **Upper Bound** values.

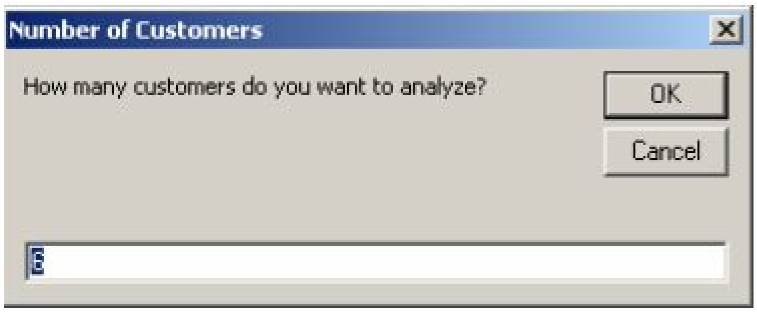
Varrying Upper Bound on Total Number of Calls

Min	Max	Step Size
1,000	20,000	1,000

Upper Bound on Total Number of Calls	Total Overall Revenue
1,000	$737,477.00
2,000	$737,477.00
3,000	$737,477.00
4,000	$737,477.00
5,000	$737,477.00
6,000	$11,557,749.00
7,000	$11,557,749.00
8,000	$11,557,749.00
9,000	$11,557,749.00
10,000	$11,557,749.00
11,000	$11,557,749.00
12,000	$11,557,749.00
13,000	$11,557,749.00
14,000	$11,557,749.00
15,000	$11,557,749.00
16,000	$11,557,749.00
17,000	$11,557,749.00
18,000	$13,989,157.00
19,000	$13,989,157.00
20,000	$13,989,157.00

Figure 5 The trade-off curve sheet.

For this application's user interface, we use navigational and functional buttons, input boxes and message boxes, and a userform. Pressing the "Start" button on the welcome sheet displays an input box for the number of customers to analyze. (See Figure 6.) The user form then ascertains from the user how the historical data will be provided. (See Figure 7.) The user has two options: import a text file, for which he or she will be prompted to select a file to import; or enter the data manually, for which he or she will proceed directly to the historical data sheet. This form therefore requires two option buttons.

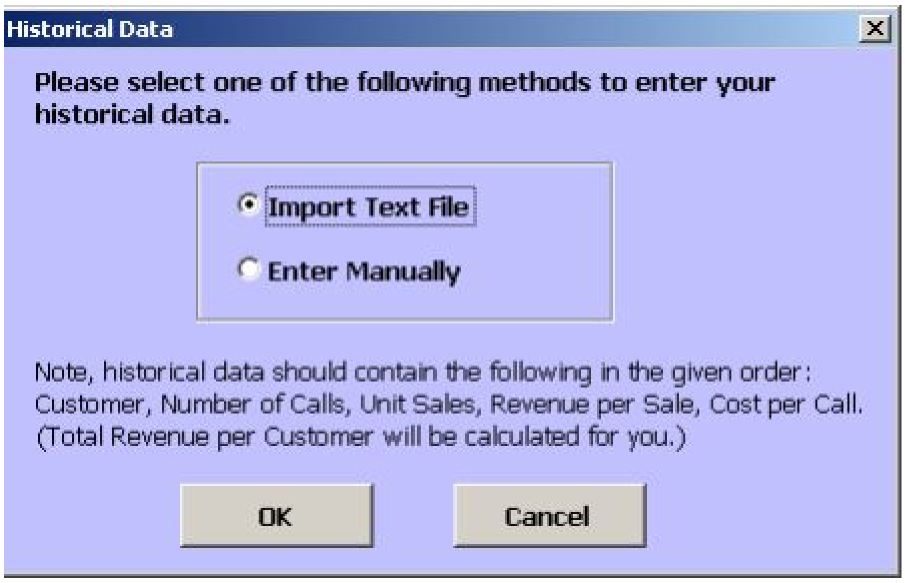


Number of Customers

How many customers do you want to analyze?

3

Figure.6 The input box for the number of customers.


Historical Data

Please select one of the following methods to enter your historical data.

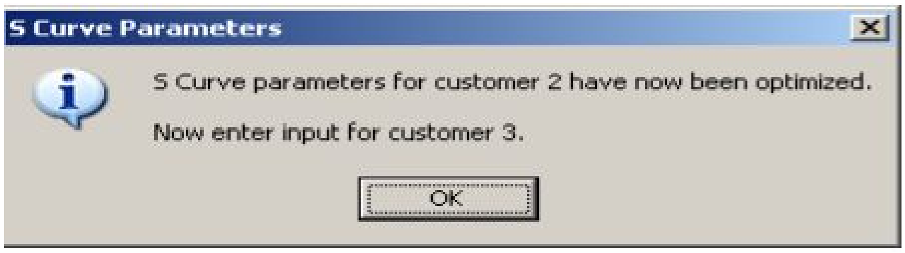
[✓ Import Text File]

Enter Manually

Note, historical data should contain the following in the given order: Customer, Number of Calls, Unit Sales, Revenue per Sale, Cost per Call. (Total Revenue per Customer will be calculated for you.)

Figure 7 The historical data form.

We include several message boxes on the s-curve parameters sheet to communicate with the user. As each customer’s s-curve parameters are optimized, we display a message box to the user to declare that one customer’s parameters are optimized, and so he or she should now enter the input for the next customer. (See Figure 8.)

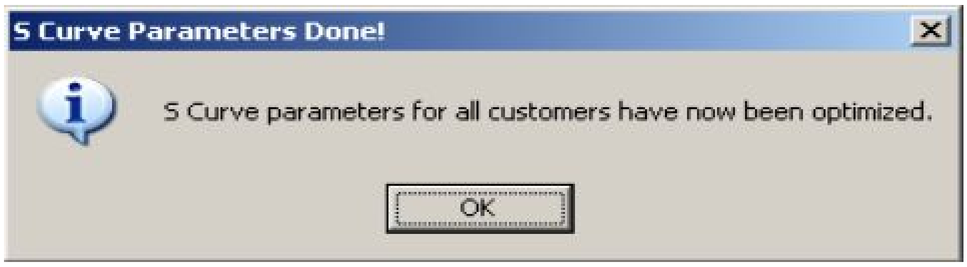

S Curve Parameters

S Curve parameters for customer 2 have now been optimized.

Now enter input for customer 3.

Figure 8 The s-curve input box between customers.

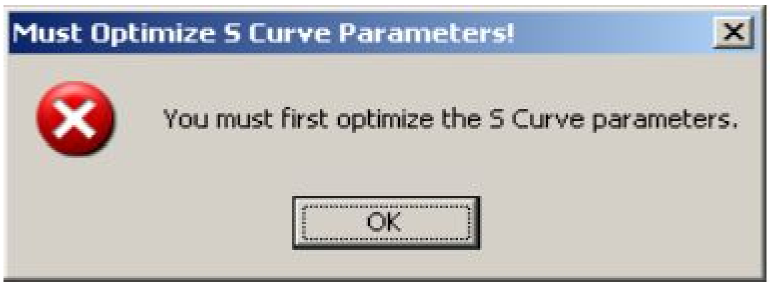
When all of the customers’ s-curve parameters have been optimized, a message box appears to inform the user that this sheet is completed. (See Figure 9.)


S Curve Parameters Done!

S Curve parameters for all customers have now been optimized.

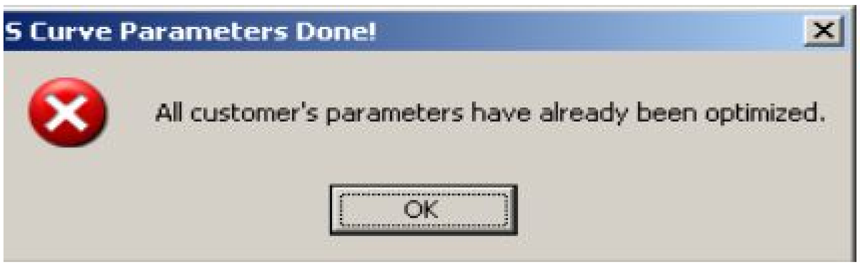
Figure 9 The s-curve input box when all the customers have been optimized.

We also use message boxes here for two error checks. If the user tries to go to the optimization sheet before optimizing all the customers’ s-curve parameters, a message box appears with an error message. (See Figure 10.) If the user continues pressing the “Calculate” button to optimize the customer’s s-curve parameters after all the customers’ parameters have been optimized, then another error message appears. (See Figure 11.)


Must Optimize S Curve Parameters!

You must first optimize the S Curve parameters.

Figure 10 Error checking to finish the s-curve parameters before going to the optimization sheet.


S Curve Parameters Done!

All customer's parameters have already been optimized.

Figure 11 Error checking to prevent further s-curve optimization after is has been completed.

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.

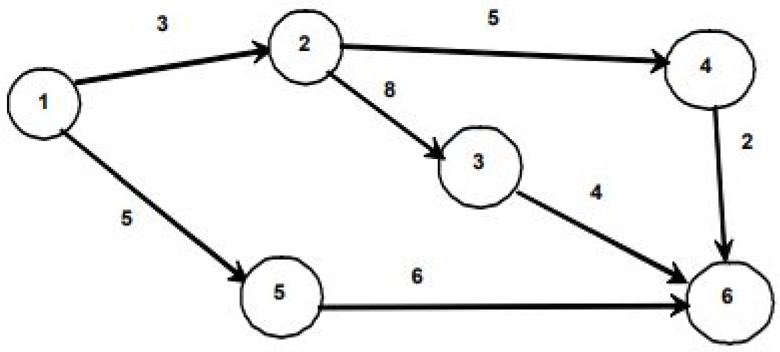
Project 4: Critical Path Method
Subject: Statistical Model

Application Overview

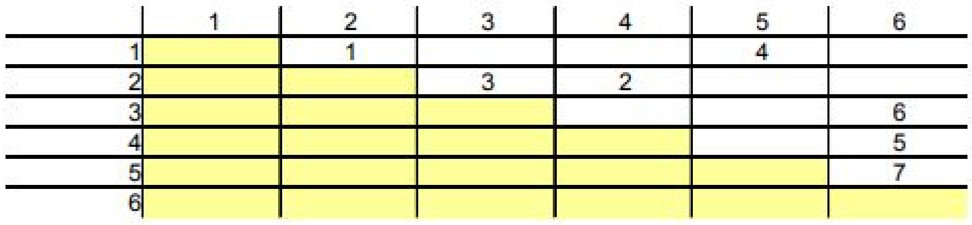
The critical path method is a quantitative technique to manage projects. This technique models a project as a network of several activities. Each activity has an associated activity time that denotes the length of the time needed to complete the activity. Activities also have precedence relationships 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:

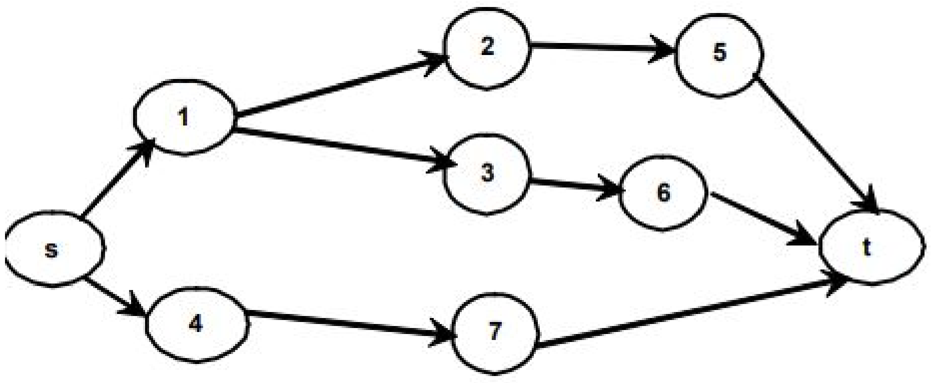
```
FreeFloat(CurrentAct) = EarlyTime(j) - EarlyTime(i) - ActDuration(CurrentAct)
TotalFloat(CurrentAct) = LateTime(j) - EarlyTime(i) - ActDuration(CurrentAct)
```

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)
```

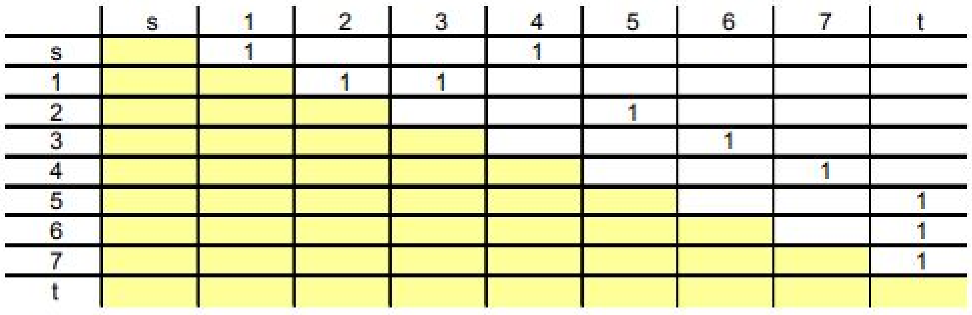
```
        ProjCost = ProjCost + NormCost(i)
    End If
```

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.



$$\begin{tikzcd}
 & & 2 \arrow[r] & 5 \arrow[dr] & \\
 & 1 \arrow[ur] \arrow[r] & 3 \arrow[r] & 6 \arrow[r] & t \\
s \arrow[ur] \arrow[r] & 4 \arrow[r] & 7 \arrow[urr] & &
\end{tikzcd}$$

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)

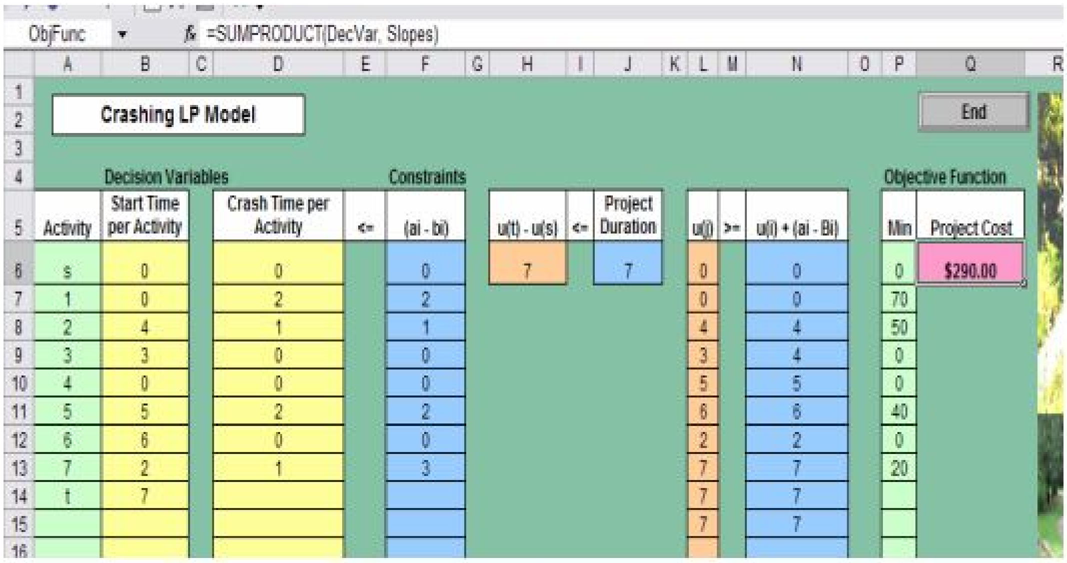
Constraints:

- 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:
$B_i \le A_i - C_i$,
A_i = activity time and C_i = crash time
$u_t - u_s \le P$
$u_j \ge u_i + A_i - B_i$
$B_i \ge 0$
$u_i \ge 0$



ObjFunc =SUMPRODUCT(DecVar, Slopes)

Crashing LP Model

End

	Decision Variables			Constraints							Objective Function	
Activity	Start Time per Activity	Crash Time per Activity	<=	(ai - bi)	u(t) - u(s)	<=	Project Duration	u(j)	>=	u(i) + (ai - Bi)	Min	Project Cost
s	0	0		0	7		7	0		0	0	$290.00
1	0	2		2				0		0	70	
2	4	1		1				4		4	50	
3	3	0		0				3		4	0	
4	0	0		0				5		5	0	
5	5	2		2				6		6	40	
6	6	0		0				2		2	0	
7	2	1		3				7		7	20	
t	7							7		7		
								7		7		

Figure 1 The hidden optimization sheet.

The time-cost tradeoff graph shows the project cost achieved from the crashing model for each iterative project time. In this graph, the total project cost includes the sum of the normal activity costs. The resulting graph should be a piece-wise linear convex function. For more details on the critical path method or time-cost tradeoff problem, please see Introduction to Operations Research by Winston.

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

Output

The output for the CPM option is the following:

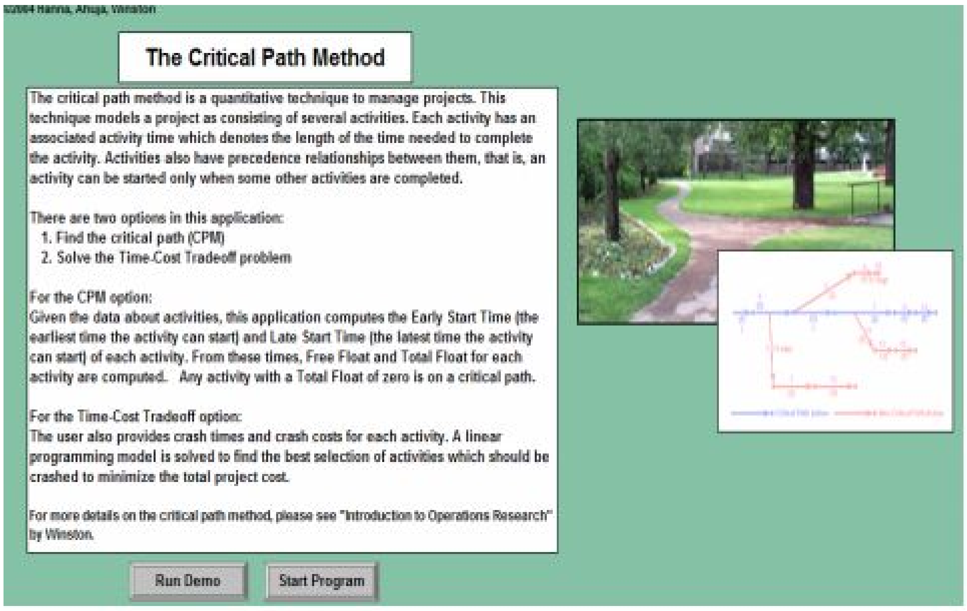
- Critical path of the project
- Total project completion time
- Event early start times and late start times
- Activity free float and total float times

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

- Time-cost tradeoff graph for various project times and costs

User Interface

This application requires seven worksheets: the welcome sheet, the network sheet, the activity table sheet, the precedence matrix sheet, the CPM output sheet, the time-cost tradeoff output sheet, and the hidden crash LP model sheet. (Note: we also have created an example sheet, which stores demo data and is available for the user to view if extra help is needed.) The welcome sheet contains the title, the description of the application, and the “Run Demo” and “Start” buttons. (See Figure 2.) “Run Demo” and “Start” both display an option form and take the user to the network sheet.



©2004 Hanna, Ahuja, Winston

The Critical Path Method

The critical path method is a quantitative technique to manage projects. This technique models a project as consisting of several activities. Each activity has an associated activity time which denotes the length of the time needed to complete the activity. Activities also have precedence relationships between them, that is, an activity can be started only when some other activities are completed.

There are two options in this application:
1. Find the critical path (CPM)
2. Solve the Time-Cost Tradeoff problem

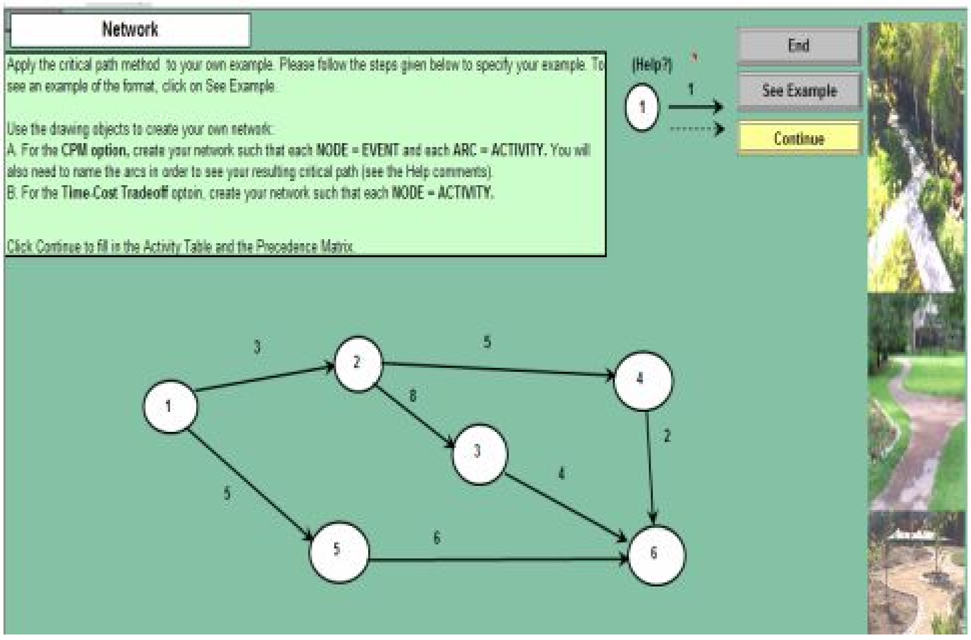
For the CPM option:
Given the data about activities, this application computes the Early Start Time (the earliest time the activity can start) and Late Start Time (the latest time the activity can start) of each activity. From these times, Free Float and Total Float for each activity are computed. Any activity with a Total Float of zero is on a critical path.

For the Time-Cost Tradeoff option:
The user also provides crash times and crash costs for each activity. A linear programming model is solved to find the best selection of activities which should be crashed to minimize the total project cost.

For more details on the critical path method, please see "Introduction to Operations Research" by Winston.

Figure 2 The welcome sheet.

On the network sheet, the user creates the project network with the sample drawing objects provided (the circle with the text box, the arrow with the text box, and the dashed arrow). (See Figure 3.) For the CPM option, the user creates the network such that each node = event and each arc = activity. [See Figure 3 (a).] The user also needs to name the arcs in order to view the resulting critical path. Instructions on how to name the arcs are provided in a comment box of the cell with the text “Help?” next to the buttons on the sheet. For the time-cost tradeoff option, the user creates the network such that each node = activity. [See Figure 3 (b).] The buttons on the sheet are: “End,” which takes the user back to the welcome sheet; “See Example,” which takes the user to an example sheet; and “Continue,” which takes the user to the activity table sheet.



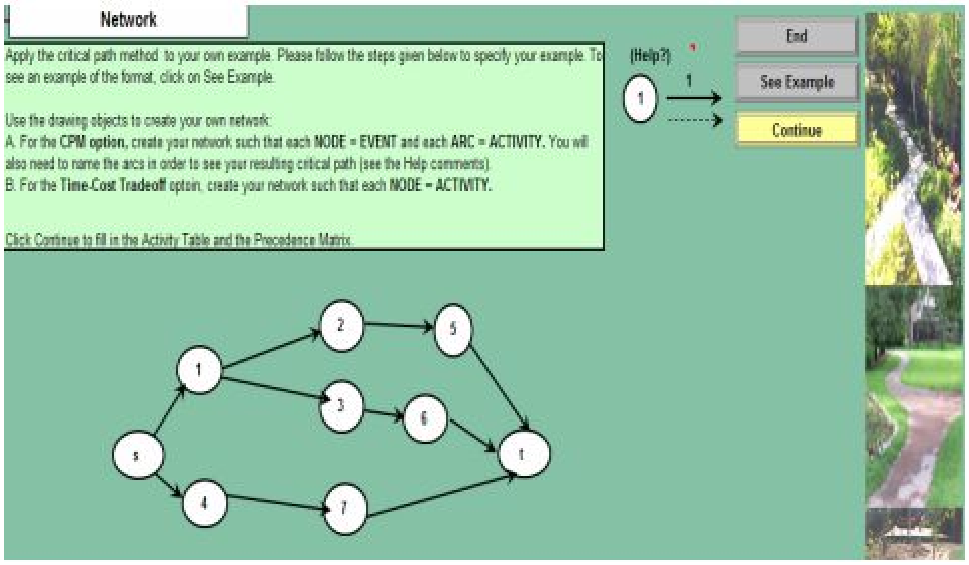
Network

Apply the critical path method to your own example. Please follow the steps given below to specify your example. To see an example of the format, click on See Example.

Use the drawing objects to create your own network:
A. For the **CPM option,** create your network such that each **NODE = EVENT** and each **ARC = ACTIVITY.** You will also need to name the arcs in order to see your resulting critical path (see the Help comments).
B. For the **Time-Cost Tradeoff** option, create your network such that each **NODE = ACTIVITY.**

Click Continue to fill in the Activity Table and the Precedence Matrix.

Figure 3 (a)



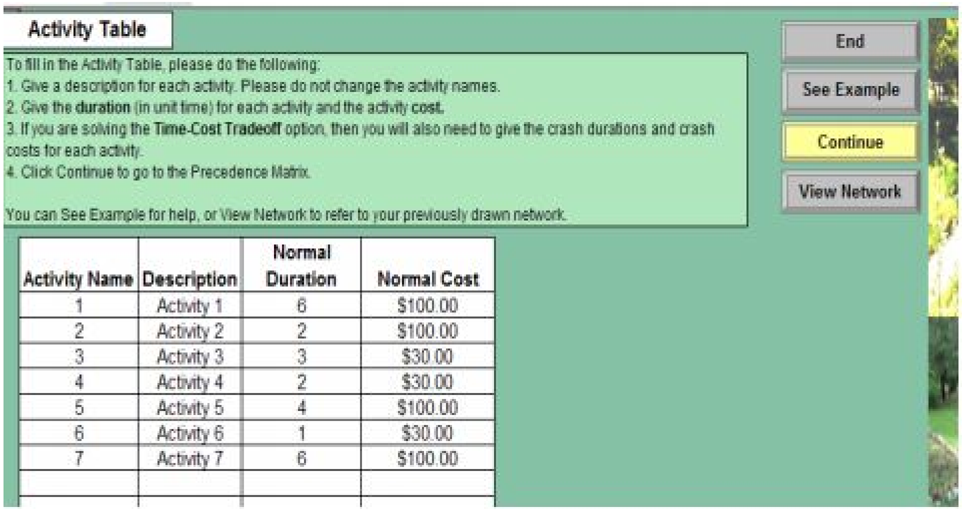
Apply the critical path method to your own example. Please follow the steps given below to specify your example. To see an example of the format, click on See Example.

Use the drawing objects to create your own network:
A. For the **CPM option,** create your network such that each **NODE = EVENT** and each **ARC = ACTIVITY.** You will also need to name the arcs in order to see your resulting critical path (see the Help comments).
B. For the **Time-Cost Tradeoff** optoin, create your network such that each **NODE = ACTIVITY.**

Click Continue to fill in the Activity Table and the Precedence Matrix.

Figure.3 (b)
Figure 3 The network sheet.

On the activity table sheet, the user provides information about each activity in the project. (See Figure 4.) An activity number is automatically entered in the table; the user then describes for each activity and inputs the normal duration, or time, of the activity as well as the normal cost of the activity. [See Figure 4 (a).]
For the time-cost tradeoff option, the user also provides the crash duration and crash cost for each activity. [See Figure 4 (b).] The same buttons found on the network sheet are available here; this time the “Continue” button takes the user to the precedence matrix sheet. There is also a “View Network” button, which takes the user back to the network sheet; this may be useful if the user needs to check the project network to find the corresponding activities times.

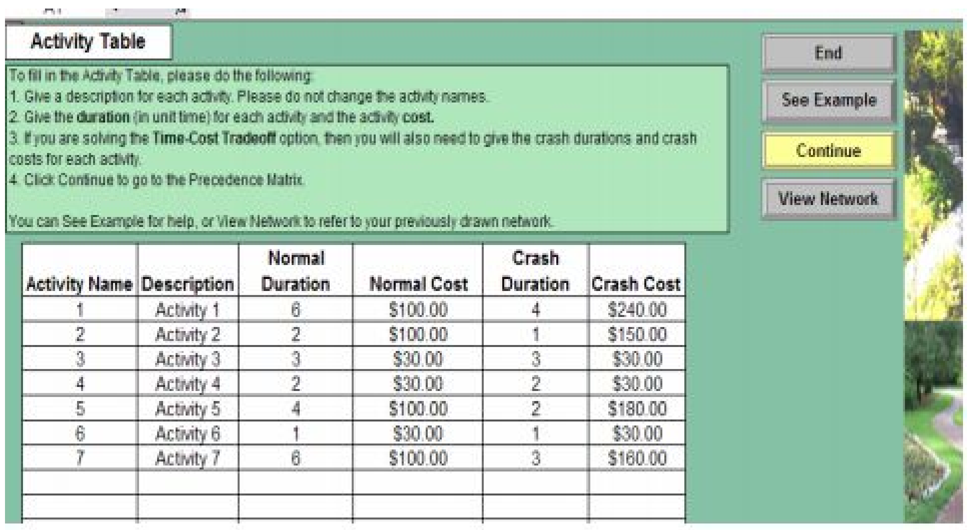


To fill in the Activity Table, please do the following:
1. Give a description for each activity. Please do not change the activity names.
2. Give the **duration** (in unit time) for each activity and the activity **cost.**
3. If you are solving the **Time-Cost Tradeoff** option, then you will also need to give the crash durations and crash costs for each activity.
4. Click Continue to go to the Precedence Matrix.

You can See Example for help, or View Network to refer to your previously drawn network.

Activity Name	Description	Normal Duration	Normal Cost
1	Activity 1	6	$100.00
2	Activity 2	2	$100.00
3	Activity 3	3	$30.00
4	Activity 4	2	$30.00
5	Activity 5	4	$100.00
6	Activity 6	1	$30.00
7	Activity 7	6	$100.00

Figure 4 (a)



Activity Table

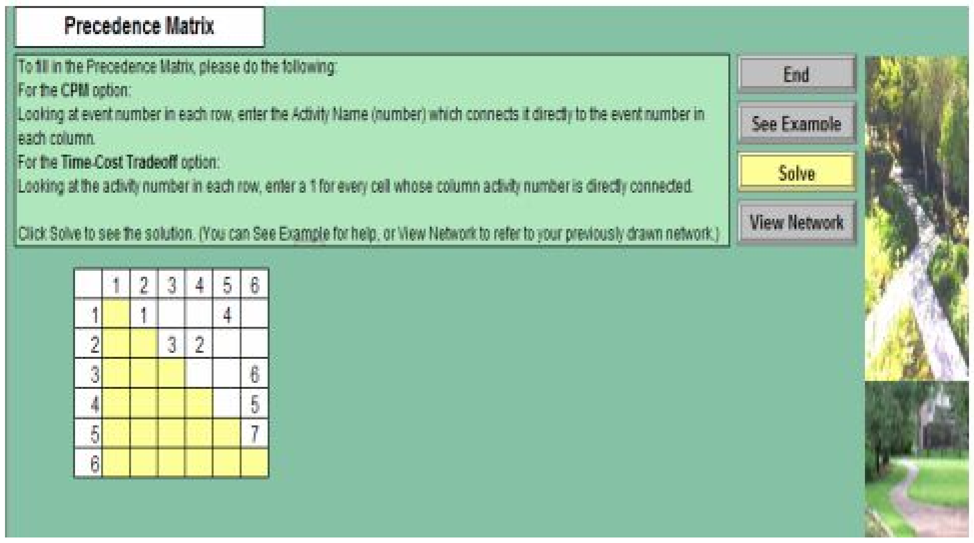
To fill in the Activity Table, please do the following:
1. Give a description for each activity. Please do not change the activity names.
2. Give the **duration** (in unit time) for each activity and the activity **cost.**
3. If you are solving the **Time-Cost Tradeoff** option, then you will also need to give the crash durations and crash costs for each activity.
4. Click Continue to go to the Precedence Matrix.

You can See Example for help, or View Network to refer to your previously drawn network.

Activity Name	Description	Normal Duration	Normal Cost	Crash Duration	Crash Cost
1	Activity 1	6	$100.00	4	$240.00
2	Activity 2	2	$100.00	1	$150.00
3	Activity 3	3	$30.00	3	$30.00
4	Activity 4	2	$30.00	2	$30.00
5	Activity 5	4	$100.00	2	$180.00
6	Activity 6	1	$30.00	1	$30.00
7	Activity 7	6	$100.00	3	$160.00

Figure 4 (b)
Figure 4 The activity table sheet.

On the precedence matrix sheet, the user completes the project's precedence matrix. (See Figure 5.) Depending on which option the user has selected to solve, the corresponding precedence matrix format is automatically created. In other words, for the CPM option, a precedence matrix is created with the number of rows and columns = number of events. [See Figure 5(a).] Here, cell values equal the activity number in the precedence relationship between the row and column events. For the time-cost tradeoff option, a precedence matrix is created with the number of rows and columns = number of activities plus two. [See Figure 5 (b).] Here, cell values equal 1 if precedence exists between the row and column activities. The "End," "See Example," and View Network" buttons are also on this sheet. The user can also press the "Solve" button to solve the problem. He or she is then taken to the corresponding output sheet.



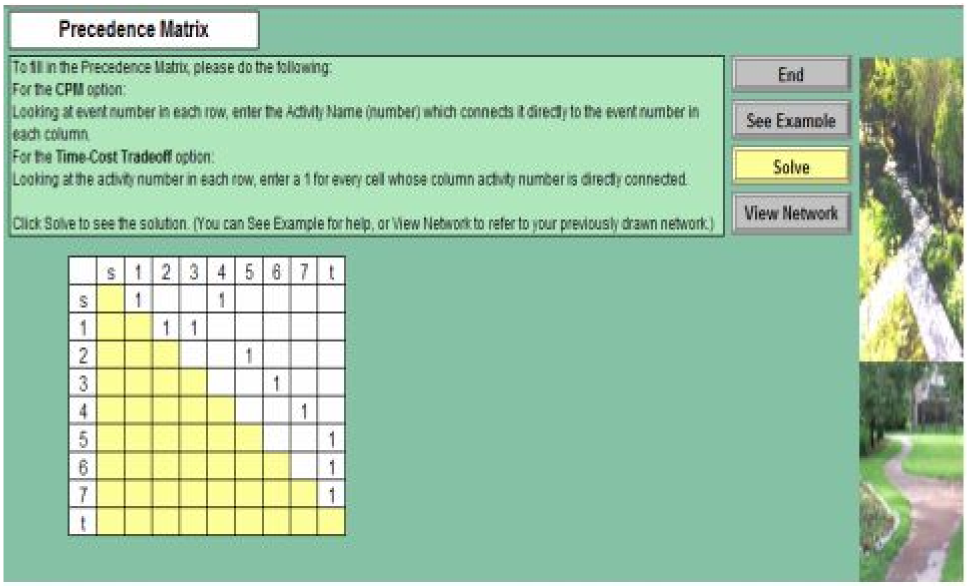
Precedence Matrix

To fill in the Precedence Matrix, please do the following:
For the **CPM** option:
Looking at event number in each row, enter the Activity Name (number) which connects it directly to the event number in each column.
For the **Time-Cost Tradeoff** option:
Looking at the activity number in each row, enter a 1 for every cell whose column activity number is directly connected.

Click Solve to see the solution. (You can See Example for help, or View Network to refer to your previously drawn network.)

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

Figure CS12.5 (a)



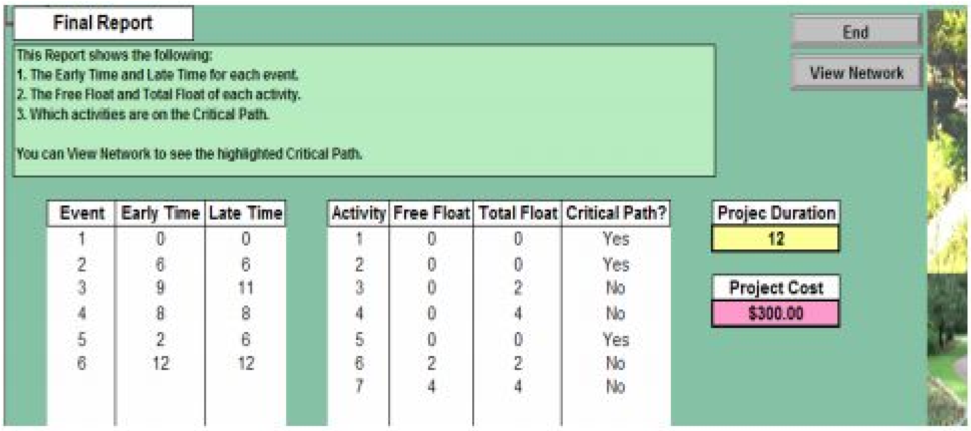
To fill in the Precedence Matrix, please do the following:
For the CPM option:
Looking at event number in each row, enter the Activity Name (number) which connects it directly to the event number in each column.
For the **Time-Cost Tradeoff** option:
Looking at the activity number in each row, enter a 1 for every cell whose column activity number is directly connected.

Click Solve to see the solution. (You can See Example for help, or View Network to refer to your previously drawn network.)

	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									

Figure 5 (b)
Figure 5 The precedence matrix sheet.

The output sheets show the solution to the selected problem. (See Figure 6.) For the CPM option, the report presents the early time and late time for each event, the free float and total float time of each activity, and which activities are on the critical path. [See Figure 6 (a).] The total project duration and total project cost are also provided. The user can press the “View Network” button to return to the network sheet and see the critical path highlighted on the network he or she drew. (See Figure 7.) For the timecost tradeoff option, the report presents the time-cost tradeoff graph, the corresponding desired project durations, the achieved project durations, and the project costs for each time-crashing iteration. [See Figure 6 (b).]



This Report shows the following:
1. The Early Time and Late Time for each event.
2. The Free Float and Total Float of each activity.
3. Which activities are on the Critical Path.

You can View Network to see the highlighted Critical Path.

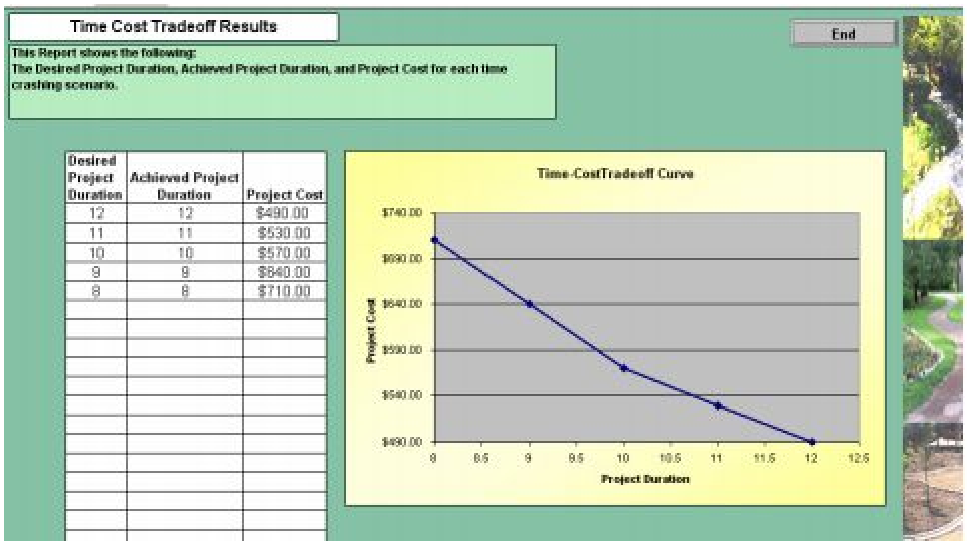
Event	Early Time	Late Time
1	0	0
2	6	6
3	9	11
4	8	8
5	2	6
6	12	12

Activity	Free Float	Total Float	Critical Path?
1	0	0	Yes
2	0	0	Yes
3	0	2	No
4	0	4	No
5	0	0	Yes
6	2	2	No
7	4	4	No

Projec Duration
12

Project Cost
$300.00

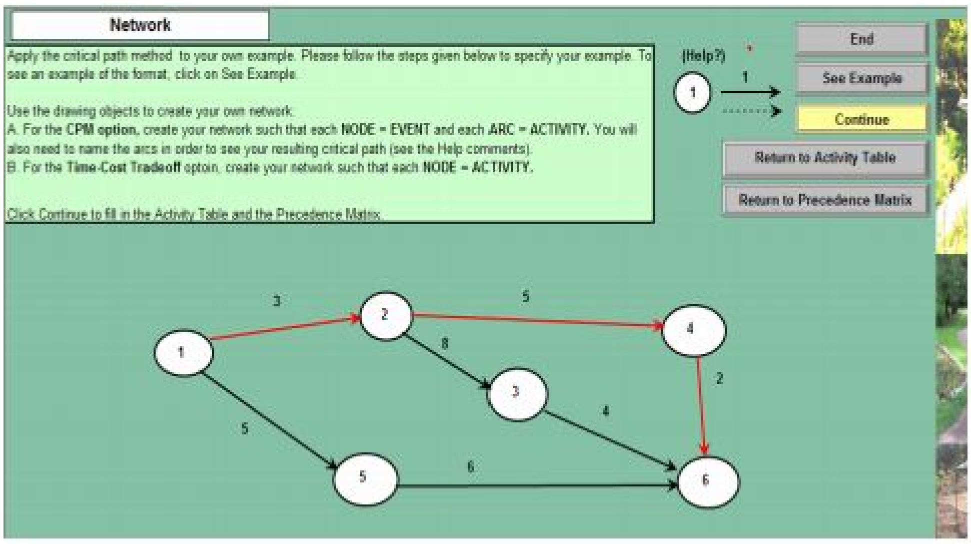
Figure 6 (a)



This Report shows the following:
The Desired Project Duration, Achieved Project Duration, and Project Cost for each time crashing scenario.

Desired Project Duration	Achieved Project Duration	Project Cost
12	12	$490.00
11	11	$530.00
10	10	$570.00
9	9	$640.00
8	8	$710.00

Figure 6 (b)
Figure 6 The output sheets.



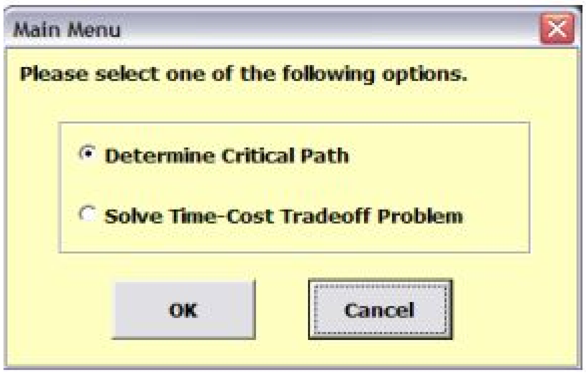
Apply the critical path method to your own example. Please follow the steps given below to specify your example. To see an example of the format, click on See Example.

Use the drawing objects to create your own network.
A. For the **CPM option,** create your network such that each **NODE = EVENT** and each **ARC = ACTIVITY.** You will also need to name the arcs in order to see your resulting critical path (see the Help comments).
B. For the **Time-Cost Tradeoff** optoin, create your network such that each **NODE = ACTIVITY.**

Click Continue to fill in the Activity Table and the Precedence Matrix.

Figure 7 The updated network sheet for the CPM option.

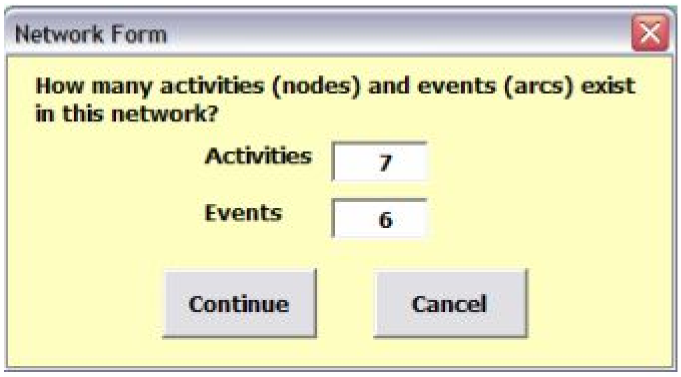
For this application's user interface, we use navigational and functional buttons and two user forms. When the user presses the "Run Demo" or "Start" button on the welcome sheet, the main options form appears. (See Figure 8.) On this form, the user selects whether to solve the CPM problem or the time-cost tradeoff problem. This form requires one frame and two option buttons. After the user creates the project network on the network sheet and presses the "Continue" button, the following form appears before the activity table sheet. (See Figure CS12.9.) This form, the network form, prompts the user for the number of events and activities represented in the project network. This form requires two text boxes.



Please select one of the following options.

[✓ Determine Critical Path]
Solve Time-Cost Tradeoff Problem

Figure 8 The main menu form.


Network Form

How many activities (nodes) and events (arcs) exist in this network?

Activities 7

Events 6

Figure 9 The network input form

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

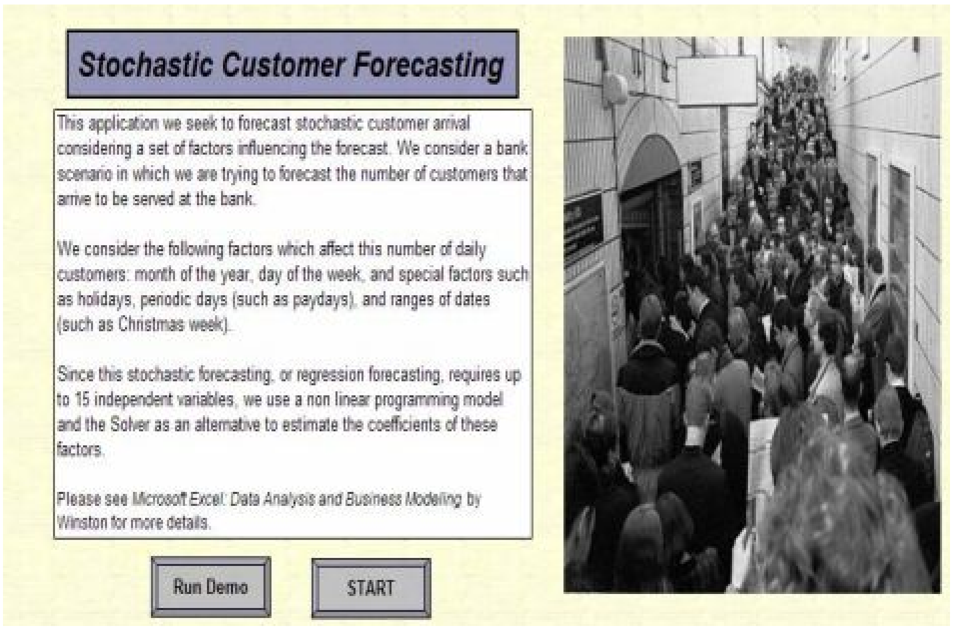
Output

The output for this application is the following:

- 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

User Interface

This application requires four sheets: the welcome sheet, the historical data sheet, the solutions sheet, and the solutions chart sheet. The welcome sheet contains the title, the description of the application, and the “Run Demo” and “Start” buttons. (See Figure 1.) The “Run Demo” button copies the demo data to the historical data sheet and takes the user to this sheet. The “Start” button prompts the user for some input and then takes him or her to the historical data sheet.



This application we seek to forecast stochastic customer arrival considering a set of factors influencing the forecast. We consider a bank scenario in which we are trying to forecast the number of customers that arrive to be served at the bank.

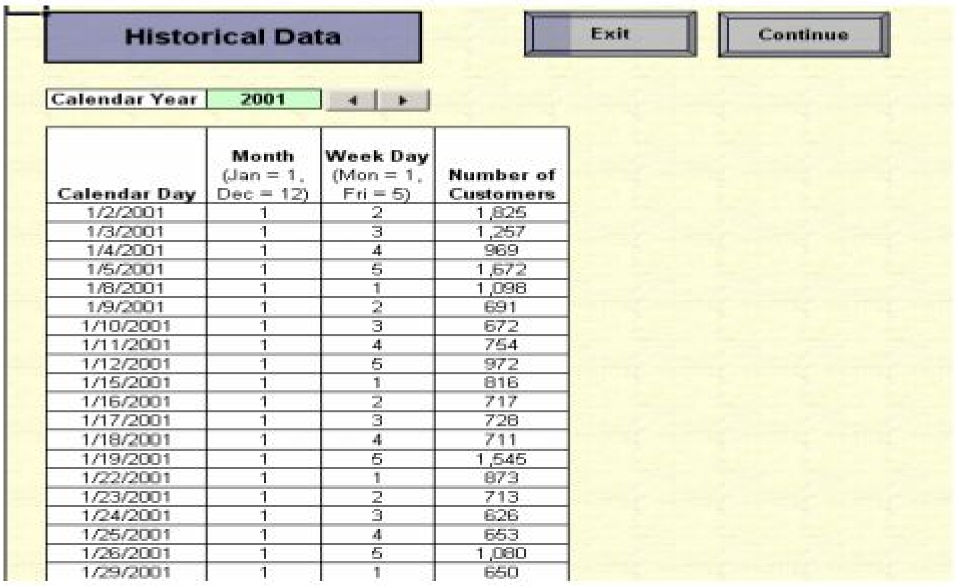
We consider the following factors which affect this number of daily customers: month of the year, day of the week, and special factors such as holidays, periodic days (such as paydays), and ranges of dates (such as Christmas week).

Since this stochastic forecasting, or regression forecasting, requires up to 15 independent variables, we use a non linear programming model and the Solver as an alternative to estimate the coefficients of these factors.

Please see *Microsoft Excel: Data Analysis and Business Modeling* by Winston for more details.

Figure 1 The welcome sheet.

The historical data sheet contains the historical number of customers for every workday for an entire calendar year. (See Figure 2.) The user can change the calendar year using the spin button at the top of the sheet; the calendar day, month, and weekday values are automatically updated for the selected year. The user can also import this information as a text file. Once the historical data has been entered, the user presses the “Continue” button to proceed to the solution sheet.

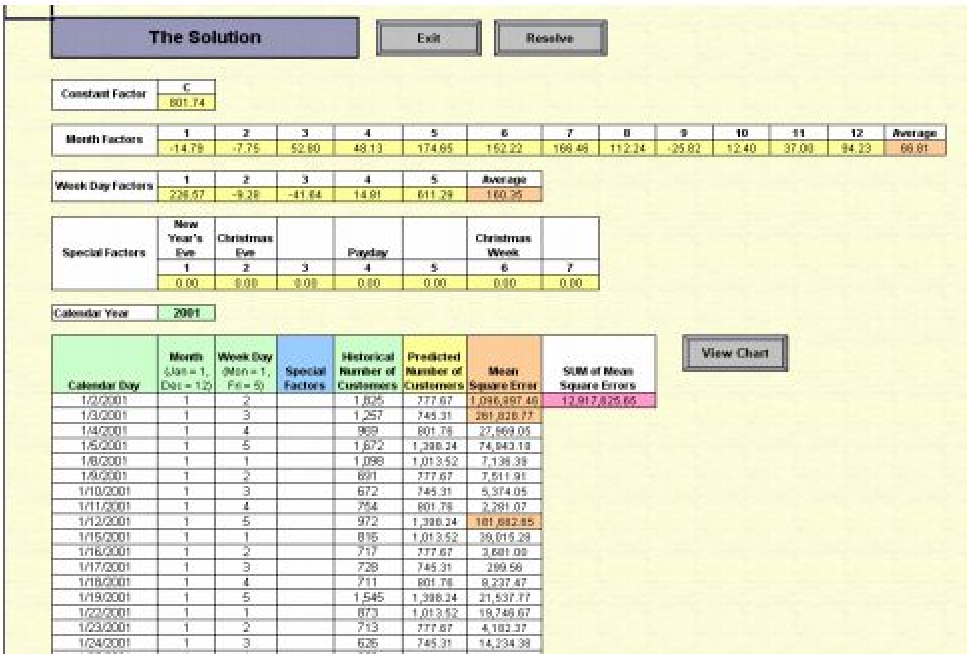


Calendar Year 2001

Calendar Day	Month (Jan = 1, Dec = 12)	Week Day (Mon = 1, Fri = 5)	Number of Customers
1/2/2001	1	2	1,825
1/3/2001	1	3	1,257
1/4/2001	1	4	969
1/5/2001	1	5	1,672
1/8/2001	1	1	1,098
1/9/2001	1	2	691
1/10/2001	1	3	672
1/11/2001	1	4	754
1/12/2001	1	5	972
1/15/2001	1	1	816
1/16/2001	1	2	717
1/17/2001	1	3	728
1/18/2001	1	4	711
1/19/2001	1	5	1,545
1/22/2001	1	1	873
1/23/2001	1	2	713
1/24/2001	1	3	626
1/25/2001	1	4	653
1/26/2001	1	5	1,080
1/29/2001	1	1	650

Figure 2 The historical data sheet.

The solution sheet contains the model and the results of the model for the factors in the customer forecasting. (See Figure 3.) A constant factor, a factor for each month, a factor for each weekday, and special factors are determined such that the sum of the mean square errors between the predicted number of customers and the historical number of customers is minimized. Outliers in the mean square error column are highlighted to help the user determine if another special factor should be considered. The user can press the “View Chart” button to view the solution chart sheet.



The Solution

Constant Factor	C
	801.74

Month Factors	1	2	3	4	5	6	7	8	9	10	11	12	Average
	-14.79	-7.75	52.80	48.13	174.65	152.22	166.46	112.24	-25.82	12.40	37.00	94.23	66.81

Week Day Factors	1	2	3	4	5	Average
	226.57	-9.28	-41.64	14.81	611.29	160.35

Special Factors	New Year's Eve	Christmas Eve		Payday		Christmas Week	
	1	2	3	4	5	6	7
	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Calendar Year	2001

Calendar Day	Month (Jan = 1, Dec = 12)	Week Day (Mon = 1, Fri = 5)	Special Factors	Historical Number of Customers	Predicted Number of Customers	Mean Square Error	SUM of Mean Square Errors
1/2/2001	1	2		1,825	777.67	1,096,897.46	12,917,826.65
1/3/2001	1	3		1,257	745.31	261,828.77	
1/4/2001	1	4		969	801.76	27,969.05	
1/5/2001	1	5		1,672	1,398.24	74,943.18	
1/8/2001	1	1		1,098	1,013.52	7,136.38	
1/9/2001	1	2		691	777.67	7,511.91	
1/10/2001	1	3		672	745.31	5,374.05	
1/11/2001	1	4		754	801.76	2,281.07	
1/12/2001	1	5		972	1,398.24	181,682.85	
1/15/2001	1	1		816	1,013.52	39,015.28	
1/16/2001	1	2		717	777.67	3,681.00	
1/17/2001	1	3		728	745.31	299.56	
1/18/2001	1	4		711	801.76	8,237.47	
1/19/2001	1	5		1,545	1,398.24	21,537.77	
1/22/2001	1	1		873	1,013.52	19,746.67	
1/23/2001	1	2		713	777.67	4,182.37	
1/24/2001	1	3		626	745.31	14,234.38	

Figure 3 The solution sheet.

The solution chart sheet graphs the predicted number of customers with the historical number of customers. (See Figure 4.) The user presses the “Return to Results” button to return to the solutions sheet.

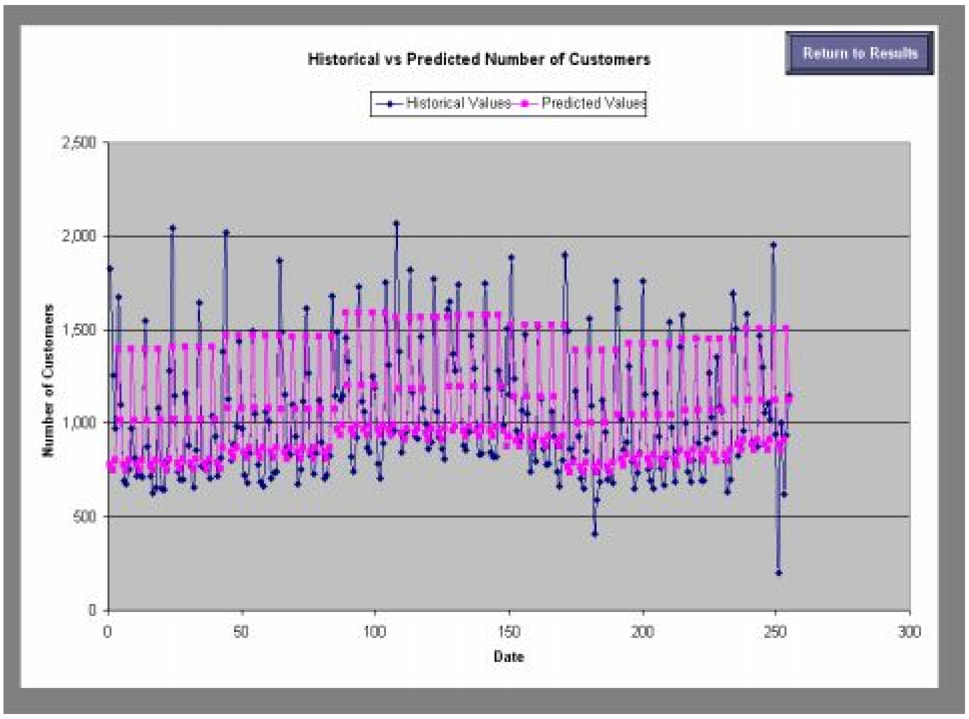
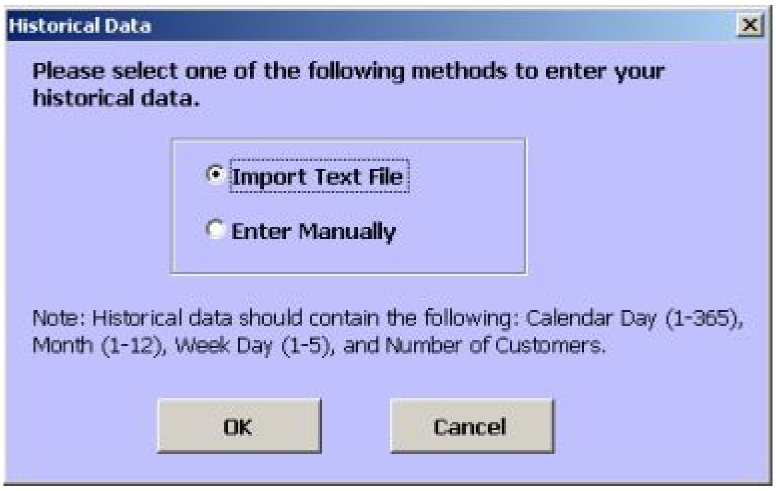


Figure 4 The solution chart sheet.

For this application’s user interface, we use navigational buttons, controls on the worksheet, and two user forms. By pressing the “Start” button on the welcome sheet, the user is shown a historical data user form. (See Figure 5.) This form, which requires two option buttons and a frame, determines if the user will import data from a text file or enter the historical number of customers manually.


Historical Data

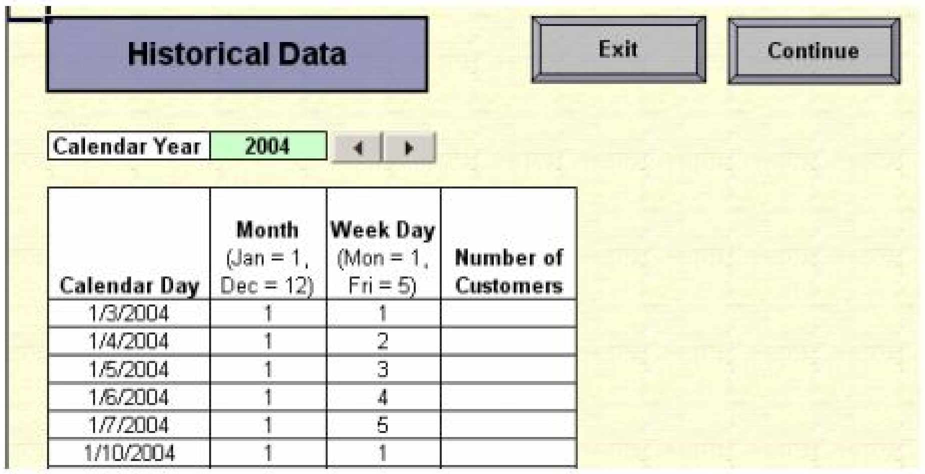
Please select one of the following methods to enter your historical data.

[✓ Import Text File]
Enter Manually

Note: Historical data should contain the following: Calendar Day (1-365), Month (1-12), Week Day (1-5), and Number of Customers.

Figure 5 The historical data form.

The user then views the historical data sheet where there is a spin button control that allows the user to change the calendar year to study. (See Figure 6.) When this spin button is changed, the linked cell is updated, and the calendar days, the month, and the weekday values are automatically updated.

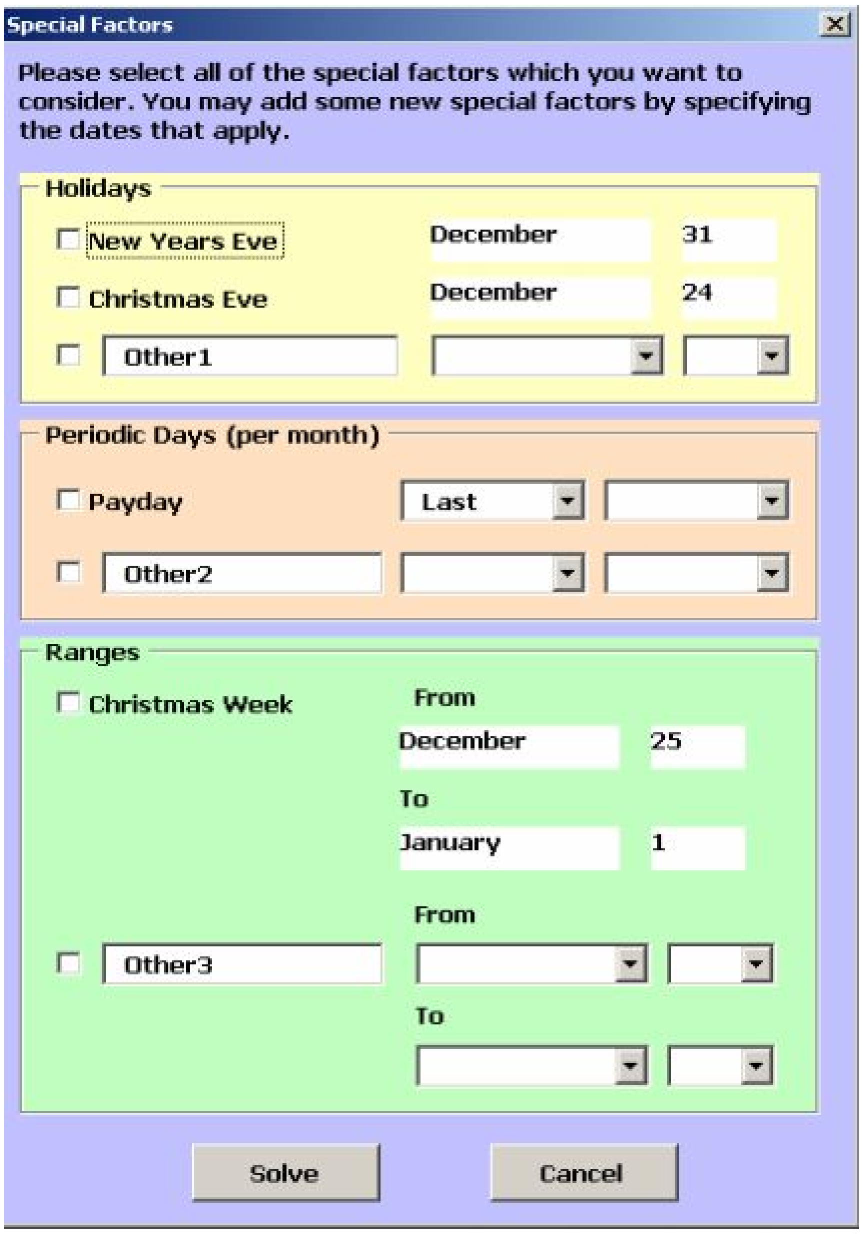


Calendar Year	2004

Calendar Day	Month (Jan = 1, Dec = 12)	Week Day (Mon = 1, Fri = 5)	Number of Customers
1/3/2004	1	1	
1/4/2004	1	2	
1/5/2004	1	3	
1/6/2004	1	4	
1/7/2004	1	5	
1/10/2004	1	1	

Figure.6 The historical data year spin button.

When the user presses the “Continue” button on this sheet, he or she views the special factors user form. (See Figure 7) This form prompts the user to select any special factors that may bias his or her historical number of customers. There are three categories of special factors: holidays, periodic days (per month), and ranges of dates. For each category, some default days are provided as well as a blank day in which the user can create his or her own special factors. The user can select all or none of these and then click the “Solve” button to solve the model and proceed to the solution sheet. Frames group these categories of special factors, and check boxes are provided for each special factor selection. Additionally, combo boxes are provided for the months, days, and periodic type and day.


Special Factors

Please select all of the special factors which you want to consider. You may add some new special factors by specifying the dates that apply.

Holidays

☐	New Years Eve	December	31
☐	Christmas Eve	December	24
☐	Other1		

Periodic Days (per month)

☐	Payday	Last	
☐	Other2		

Ranges

☐ Christmas Week

From: December 25

To: January 1

☐ Other3

From:

To:

Solve

Cancel

Figure 7 The special factors form.

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