15.053x

OpenSolver

(http://opensolver.org/)

Table of Contents

Note that there is an Excel file that accompanies this tutorial; each worksheet tab in the Excel corresponds to each example problem

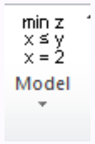
Introduction to OpenSolver (1 of 2)

- **OpenSolver** is an open source linear and integer optimizer for Microsoft Excel, which extends Excel's built-in Solver with a more powerful Linear Programming solver. (http://opensolver.org/)
- Compared to Excel's built-in Solver
  - Modeling steps are very similar.
  - No restriction on number of decision variables (***used for solving large problem***)
  - Faster solution time by using excellent solvers: CBC or Gurobi (with proper setups)
  - Better user-interface and modeling support. (saving models and quick solve)
  - Nonlinear programming is also supported with additional setup of external solvers.
- Similar to Excel Solver, **OpenSolver** is an Add-In program that you will need to download and load in Excel. Detailed installation instructions are available at http://opensolver.org/installing-opensolver/.
- In a word, **OpenSolver** is a natural upgrade from Excel Solver if you want to solve larger and more complex problem with spreadsheet.

Introduction to OpenSolver (2 of 2)

- Using OpenSolver to solve an LP is quite similar to using Excel Solver. A major difference is that OpenSolver separates the *modeling* and *solution* process.

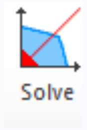
Modeling Process:



– Press the “Model” button

1. Identify the cell that contains the value of your objective function as the ***Objective Cell***
2. Identify the decision variables that can be varied, called ***Variable Cells***
3. Identify the constraints and enter them into the program to tell **SOLVER** how to solve the problem
4. Choose a solver you want to use by clicking the “Solver Engine” tab. (CBC is the default solver)

Solution Process:



1. Press the “Solve” button

– At this point, the optimal solution to our problem will be placed on the spreadsheet, with its value in the target cell.

Example 1

Diet Problem: Set-Up (1 of 7)

Problem Statement

- Consider the problem of finding a minimum cost diet.
- There are four different types of food: Brownies, Ice Cream, Cola, and Cheese Cake, with nutrition values and cost per unit as follows:

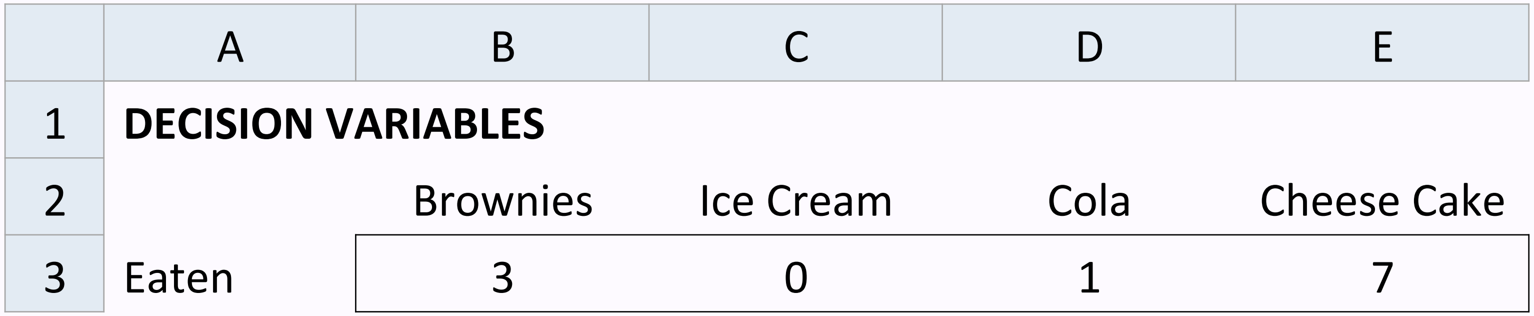
	Brownies	Ice Cream	Cola	Cheese Cake
Calories	400	200	150	500
Chocolate	3	2	0	0
Sugar	2	2	4	4
Fat	2	4	1	5
Cost	\$0.50	\$0.20	\$0.30	\$0.80

Task:

- Find a minimum-cost diet that contains
  - at least 500 calories
  - at least 6 grams of chocolate
  - at least 10 grams of sugar
  - at least 8 grams of fat.

Diet Problem: Set-Up (2 of 7)

- First, we must format our spreadsheet correctly to be entered into OpenSolver
- Identify the decision variables (variable cells)
  - To begin we enter heading for each type of food in B2:E2
  - The range B3:E3 will be the variable cells. In order to help see if our Excel formulae are correct, we will input trial values for the amount of each food eaten (any values will work, but at least one should be positive)
  - In the example shown below, we indicate that we are considering eating 3 brownies, 0 scoops of chocolate ice cream, 1 bottle of cola, and 7 pieces of pineapple cheesecake:



	A	B	C	D	E
1	**DECISION VARIABLES**				
2		Brownies	Ice Cream	Cola	Cheese Cake
3	Eaten	3	0	1	7

Diet Problem: Set-Up (3 of 7)

- Write and enter the formula for objective function in the objective cell.
  - To see if the diet is optimal, we must determine its cost as well as the calories, chocolate, sugar, and fat it provides
  - In the range B7:E7 we reference the number of units, and in B8:E8 we input the per-unit cost for each available food
    - We can compute the cost of the diet in cell B10 with the formula
      = B7*B8 + C7*C8 + D7*D8 + E7*E8
      …But it is usually preferable to enter the formula using "SUMPRODUCT".
      = **SUMPRODUCT (B7:E7, B8:E8)**
      …And this is much easier to understand for anyone reading the spreadsheet
  - The ***=SUMPRODUCT*** function requires two ranges as inputs
    - The first cell in range 1 is multiplied by the first cell in range 2, then the second cell in range 1 is multiplied by the second cell in range 2, and so on
    - All of these products are then added
    - Thus, in cell B10 the "=SUMPRODUCT" function computes total cost as
      3*50 + 0*20 + 1*30 + 0*80 = 180 cents.

Example 1

Diet Problem: Set-Up (4 of 7)

- Now, the spreadsheet should look like:

	A	B	C	D	E
1	**DECISION VARIABLES**				
2		Brownies	Ice Cream	Cola	Cheese Cake
3	Eaten	3	0	1	0
4					
5	**OBJECTIVE FUNCTION**				
6		Brownies	Ice Cream	Cola	Cheese Cake
7	Eaten	=B3	=C3	=D3	=E3
8	Cost	50	20	30	80
9					
10	Total	180	= SUMPRODUCT (B7:E7, B8:E8)		

Diet Problem: Set-Up (5 of 7)

- Finally, we must enter the constraints (for calories, chocolate, sugar, and fat)
  - To begin, we input the table in Excel that defines how many calories and units of chocolate, sugar, and fat are in each type of dessert
    - We can use this information to calculate total amounts based on the quantities of different decision variables
  - Next, take the =SUMPRODUCT of the number of items with the calories in each to calculate total calories in our dessert selection

    = **SUMPRODUCT (B7:E7, B14:E14)**
  - Finally, indicate the limitations highlighted in the problem
    - Add a >= or <= to identify maximum versus minimum constraints in Column G, and use Column H to indicate those limits:

Example 1

Diet Problem: Set-Up (6 of 7)

- The formulas will look like:

	A	B	C	D	E	F	G	H
13		Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
14	Calories	400	200	150	500	=SUMPRODUCT($B$7:$E$7,B14:E14)	>=	500
15	Chocolate	3	2	0	0	=SUMPRODUCT($B$7:$E$7,B15:E15)	>=	6
16	Sugar	2	2	4	4	=SUMPRODUCT($B$7:$E$7,B16:E16)	>=	10
17	Fat	2	4	1	5	=SUMPRODUCT($B$7:$E$7,B17:E17)	>=	8

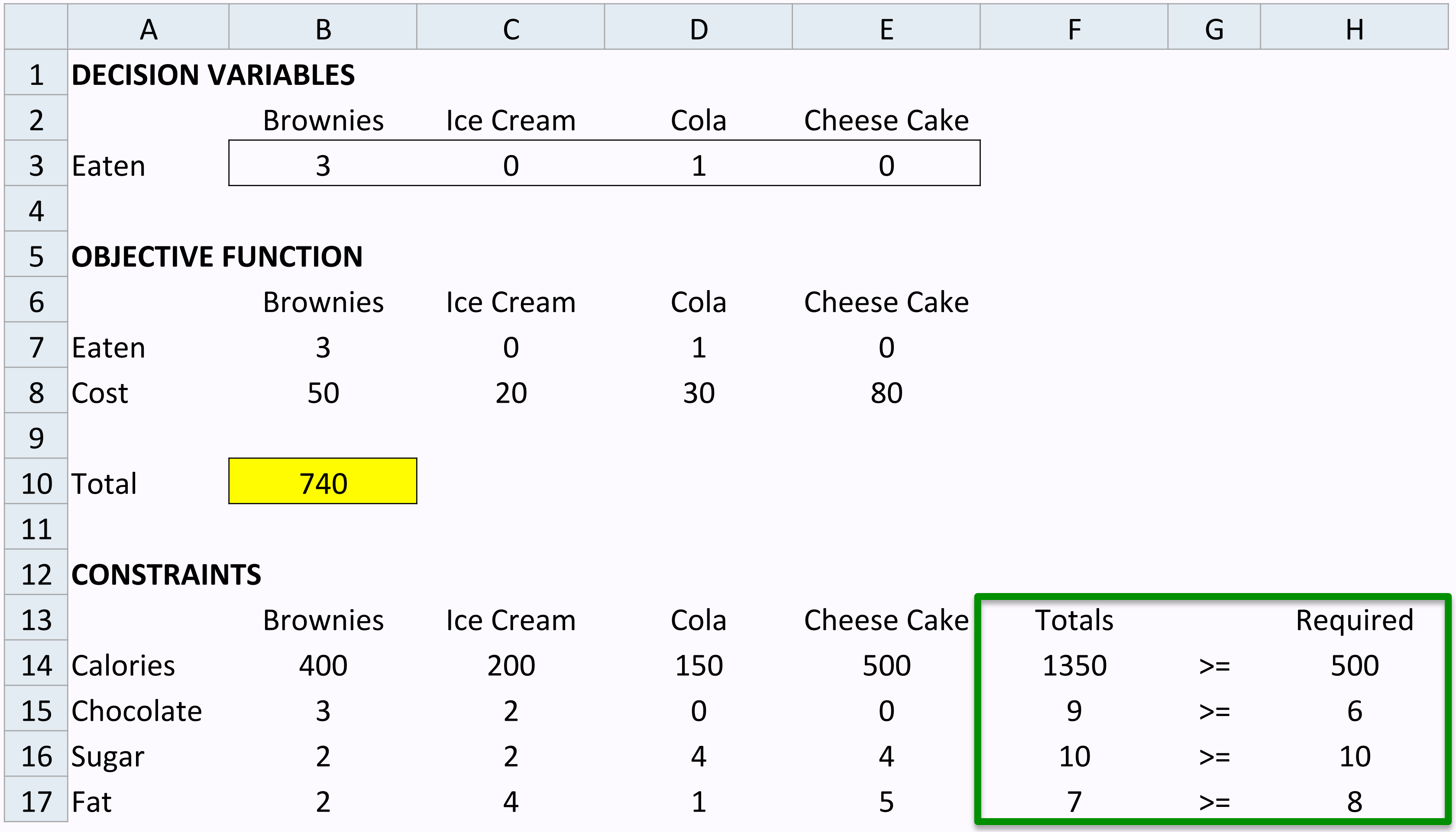
- The constraint values that will show up on your screen look like:

	A	B	C	D	E	F	G	H
13		Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
14	Calories	400	200	150	500	1350	>=	500
15	Chocolate	3	2	0	0	9	>=	6
16	Sugar	2	2	4	4	10	>=	10
17	Fat	2	4	1	5	7	>=	8

Example 1

Diet Problem: Set-Up (7 of 7)

- The complete LP to be entered into SOLVER now looks like:

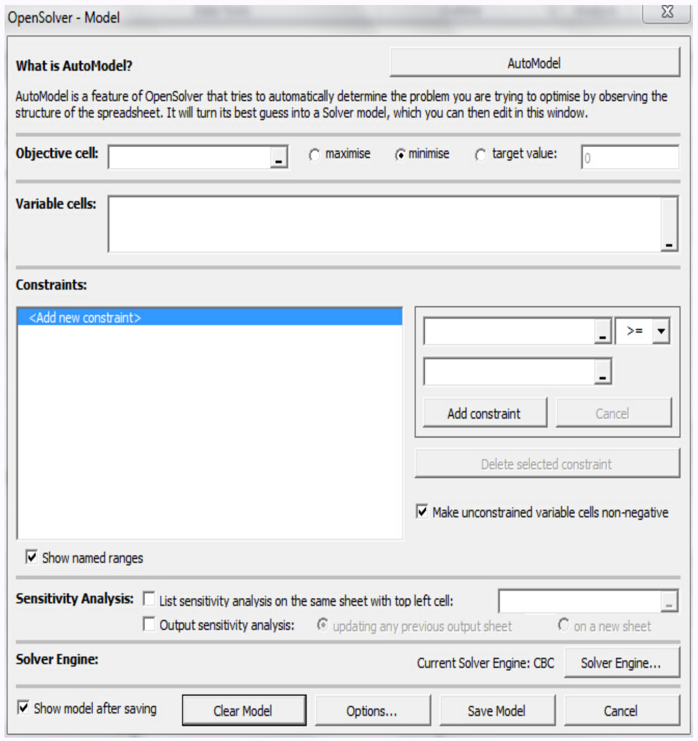


	A	B	C	D	E	F	G	H
1	**DECISION VARIABLES**							
2		Brownies	Ice Cream	Cola	Cheese Cake			
3	Eaten	3	0	1	0			
4								
5	**OBJECTIVE FUNCTION**							
6		Brownies	Ice Cream	Cola	Cheese Cake			
7	Eaten	3	0	1	0			
8	Cost	50	20	30	80			
9								
10	Total	740						
11								
12	**CONSTRAINTS**							
13		Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
14	Calories	400	200	150	500	1350	>=	500
15	Chocolate	3	2	0	0	9	>=	6
16	Sugar	2	2	4	4	10	>=	10
17	Fat	2	4	1	5	7	>=	8

Diet Problem: Dialog Box (1 of 6)

- Now, we need to enter the LP into OpenSolver (click on "Data" > "OpenSolver" > "Model" to get this box)

- We need to fill in each of the components of the Parameters Dialog Box



OpenSolver - Model

What is AutoModel?

AutoModel is a feature of OpenSolver that tries to automatically determine the problem you are trying to optimise by observing the structure of the spreadsheet. It will turn its best guess into a Solver model, which you can then edit in this window.

Objective cell: [ ] maximise [✓ minimise] target value: 0

Variable cells:

Constraints:

<Add new constraint>

[✓] Make unconstrained variable cells non-negative

[✓] Show named ranges

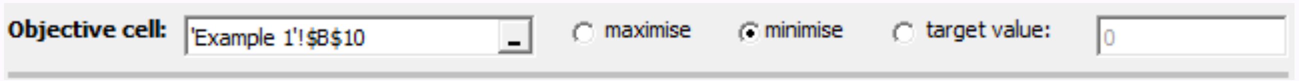
Sensitivity Analysis: [ ] List sensitivity analysis on the same sheet with top left cell:

[ ] Output sensitivity analysis: [✓ updating any previous output sheet] on a new sheet

Solver Engine: Current Solver Engine: CBC

[✓] Show model after saving

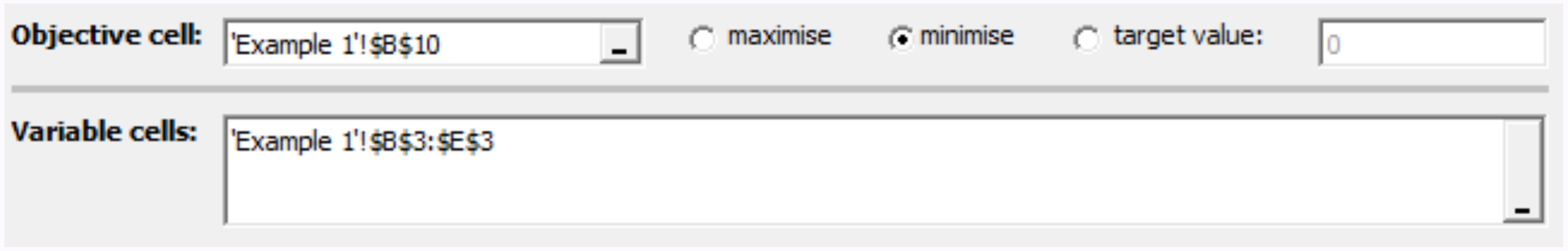
1. Identify the cell that contains the value of your objective function as the ***Objective Cell***
   - Fill in the "Objective Cell" box by clicking on the cell in our spreadsheet that calculates our objective function (in this case, B10)
   - Use the buttons to identify the type of problem we are solving; a "Maximize" or "Minimize" (here we want to minimize total cost, so select "Minimize")



Objective cell: 'Example 1'!$B$10 maximise [✓ minimise] target value: 0

Diet Problem: Dialog Box (2 of 6)

2. Identify the decision variables that can be varied, called “Variable Cells”
   - Click into the “Variable Cells” box
   - Select the decision variable cells of our LP (which are B3:E3)



Objective cell: 'Example 1'!$B$10

( ) maximise [✓ minimise] ( ) target value: 0

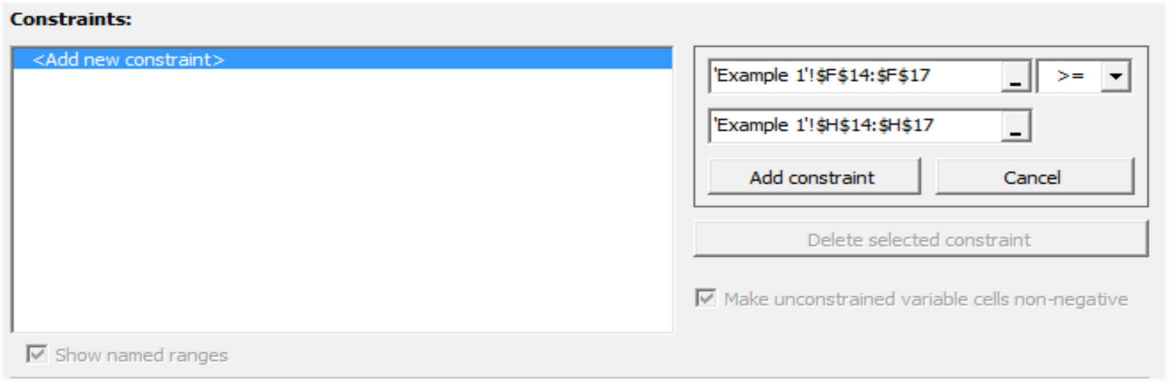
Variable cells: 'Example 1'!$B$3:$E$3

   - OpenSolver now knows that it can change the number of brownies, scoops of ice cream, sodas, and pieces of cheese cake to reach an optimal solution

   - Note: If decision variable cells are not contiguous, one can add one group of variables after another by pressing CTRL key. (This technique is not needed in Excel Solver, but is required in OpenSolver.)

Diet Problem: Dialog Box (3 of 6)

3. Identify the constraints and enter them into the program
   - In the right part of the Constraints tab, we can add constraints by specifying the cell(s) of left hand side, the cell(s) right hand side and direction of the inequality (equality)
     - For this problem, we would click on Cells F14 to F17, which computed the total calories, chocolate, sugar and fat from all our desserts
   - There are several options for constraint type: <=, >=, =, int (integer), bin (binary), or alldiff (all different). (alldiff requires that all of the decision variables take on different values.)
     - After adjusting the constraint type to be greater than or equal to (>=), click on the cells referencing the minimum quantity permitted (Cell H14 to H17)
     - *Note:* Instead of a reference, we can also enter a specific number
     - The set of constraints that was just created is as follows:

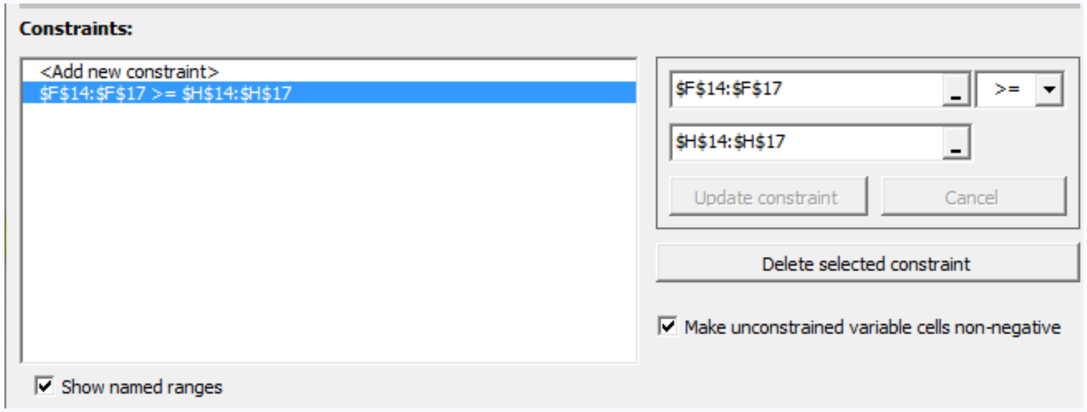

Constraints:
<Add new constraint>
'Example 1'!$F$14:$F$17 >=
'Example 1'!$H$14:$H$17
Add constraint
Cancel
Delete selected constraint
Make unconstrained variable cells non-negative
Show named ranges

     - Then press "Add constraint" button to include this set of constraints.

Example 1

Diet Problem: Dialog Box (4 of 6)

- This single set of constraints corresponds to four different constraints of the linear program.



Constraints:

<Add new constraint>
$F$14:$F$17 >= $H$14:$H$17

$F$14:$F$17 >=
$H$14:$H$17

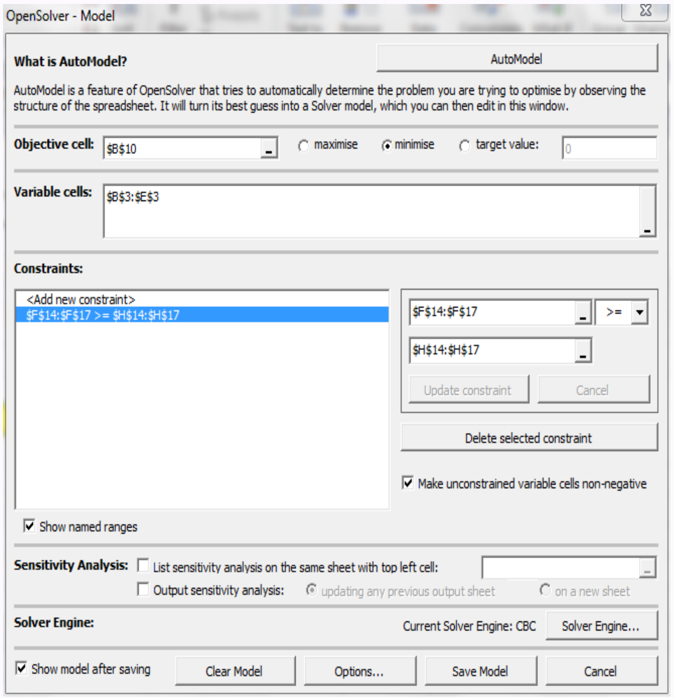
☑ Make unconstrained variable cells non-negative

☑ Show named ranges

- The “Update constraint” button allows you to modify a constraint already entered and “Delete select constraint” allows you to delete a previously entered constraint

Diet Problem: Dialog Box (5 of 6)

- The final OpenSolver Parameters Dialog Box:
  - *Note:* the checked box titled "Make Unconstrained Variables Cells Non-Negative" allows us to capture non-negativity constraints (all variables will be constrained to be >= 0)
  - The default solver is CBC. You can change the solver that is used by clicking "Solver Engine"
- Finally, click "Save Model" to finish the modeling process. **(Do not forget this important step.)**
- The decision variable is boxed with pink rectangle objective cell is boxed with yellow rectangle, constraints are boxed with blue rectangles
- Click Show/Hide model to turn on/off the display



OpenSolver - Model

What is AutoModel? AutoModel

AutoModel is a feature of OpenSolver that tries to automatically determine the problem you are trying to optimise by observing the structure of the spreadsheet. It will turn its best guess into a Solver model, which you can then edit in this window.

Objective cell: $B$10 maximise [✓ minimise] target value: 0

Variable cells: $B$3:$E$3

Constraints:

<Add new constraint>
$F$14:$F$17 >= $H$14:$H$17

$F$14:$F$17 >= $H$14:$H$17

Update constraint Cancel

Delete selected constraint

☑ Make unconstrained variable cells non-negative

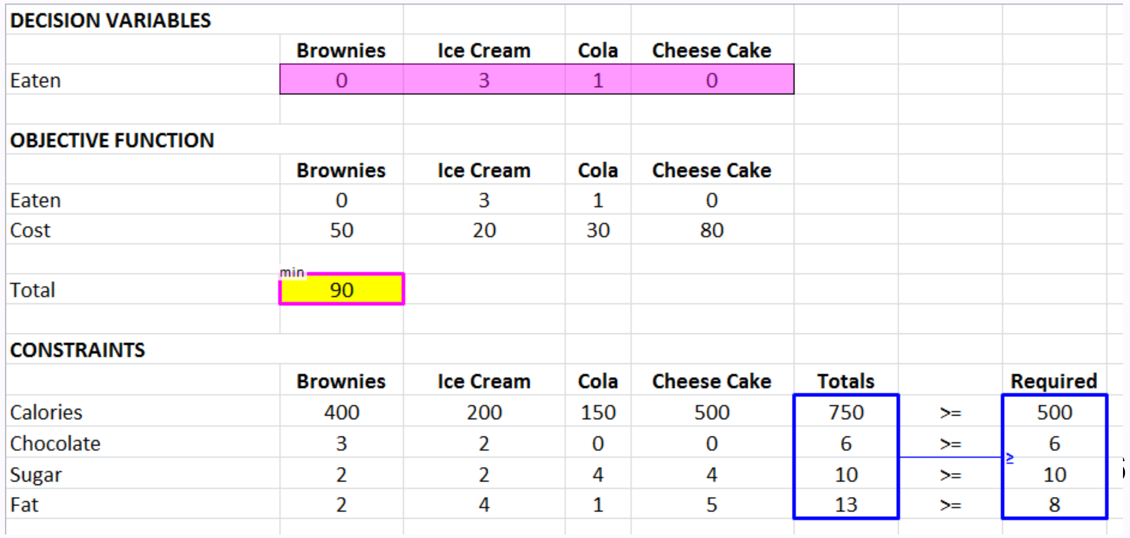
☑ Show named ranges

Sensitivity Analysis: ☐ List sensitivity analysis on the same sheet with top left cell:

☐ Output sensitivity analysis: ● updating any previous output sheet ○ on a new sheet

Solver Engine: Current Solver Engine: CBC Solver Engine...

☑ Show model after saving Clear Model Options... Save Model Cancel



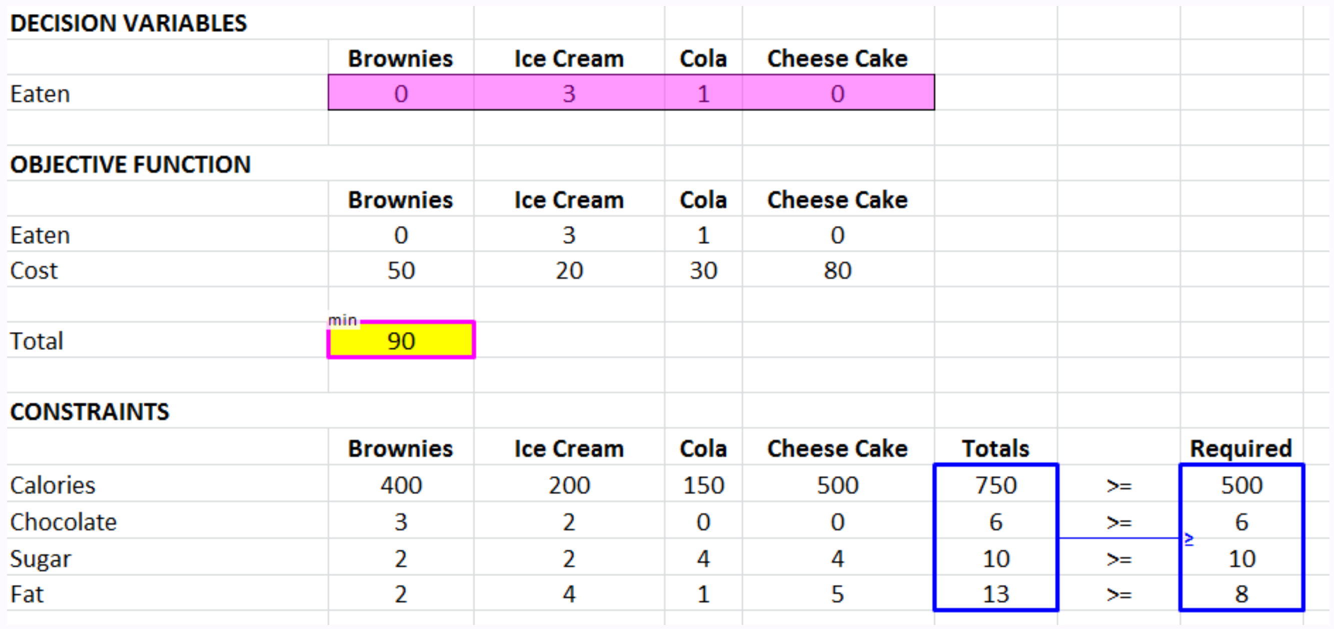
DECISION VARIABLES				
	Brownies	Ice Cream	Cola	Cheese Cake
Eaten	0	3	1	0

OBJECTIVE FUNCTION				
	Brownies	Ice Cream	Cola	Cheese Cake
Eaten	0	3	1	0
Cost	50	20	30	80
	min			
Total	90			

CONSTRAINTS							
	Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
Calories	400	200	150	500	750	>=	500
Chocolate	3	2	0	0	6	>=	6
Sugar	2	2	4	4	10	>=	10
Fat	2	4	1	5	13	>=	8

Diet Problem: Dialog Box (6 of 6)

- Then press the “solve” button
  - The optimal solution and value is then displayed on the corresponding cells



DECISION VARIABLES

	Brownies	Ice Cream	Cola	Cheese Cake
Eaten	0	3	1	0

OBJECTIVE FUNCTION

	Brownies	Ice Cream	Cola	Cheese Cake
Eaten	0	3	1	0
Cost	50	20	30	80
Total	90			

CONSTRAINTS

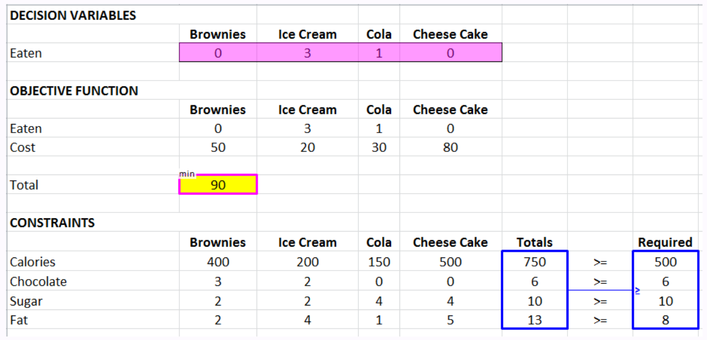
	Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
Calories	400	200	150	500	750	>=	500
Chocolate	3	2	0	0	6	>=	6
Sugar	2	2	4	4	10	>=	10
Fat	2	4	1	5	13	>=	8

Diet Problem: Quick Solve (1 of 2)

- *Quick Solve*
  - "Quick Solve" is an advanced feature in OpenSolver. Often we want to solve an instance many times while only changing the data a little. This can be time consuming if the model is large because OpenSolver will solve the problem from scratch each time. To remedy this, OpenSolver allows you to define "parameter" cells, which are the cells you will be changing between successive solves. OpenSolver then analyses your spreadsheet (which it does just once) to build the model. Every time you solve the problem, it remembers the current solution. The next time you solve the problem (after changing the data), it will use the old solution as an "advanced start". This speeds up the algorithm a lot.
  - Let's now illustrate this function using our diet problem. Suppose we are going to find a least cost food combination to satisfy the following requirement
    - Find a minimum-cost diet that contains
      - at least 500 (1000, 1500) calories
      - at least 6 grams of chocolate
      - at least 10 grams of sugar
      - at least 8 grams of fat.
    - We need to change the minimum calories requirement three times, that's where "Quick Solve" comes in. (Note: we are illustrating how Quick Solve is used. But you would never use it in practice for such a small problem. You would not have much of an advantage if the solver takes less than a few seconds.)

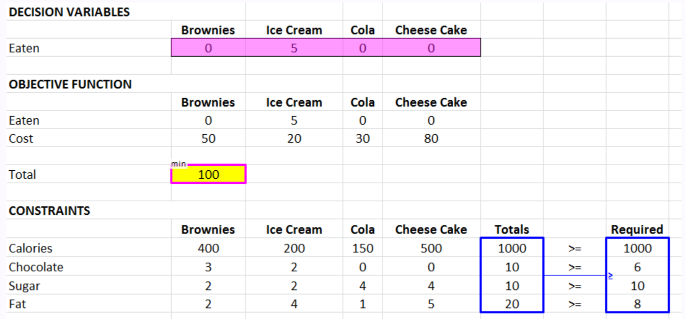
Diet Problem: Quick Solve (2 of 2)

- To set up "Quick Solve", we need to first select the cell we need to change during each successive solving.
  - Click "OpenSolver->Set Quick Solve Parameters…". In our diet problem, we choose cell H14 as our quick solve parameter. (***Note*: this cell can only be the right hand side of a constraint**)
  - Then click "OpenSolver->Initialize Quick Solve"
- Now we are ready to do the successive solving process.
  - Click "Quick Solve" in the OpenSolver tab, we get the optimal solution



DECISION VARIABLES							
	Brownies	Ice Cream	Cola	Cheese Cake			
Eaten	0	3	1	0			
OBJECTIVE FUNCTION							
	Brownies	Ice Cream	Cola	Cheese Cake			
Eaten	0	3	1	0			
Cost	50	20	30	80			
Total	min 90						
CONSTRAINTS							
	Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
Calories	400	200	150	500	750	>=	500
Chocolate	3	2	0	0	6	>=	6
Sugar	2	2	4	4	10	>=	10
Fat	2	4	1	5	13	>=	8

  - Then change H14 to 1000, click "Quick Solve" again, we get the optimal solution for the revised model.



DECISION VARIABLES							
	Brownies	Ice Cream	Cola	Cheese Cake			
Eaten	0	5	0	0			
OBJECTIVE FUNCTION							
	Brownies	Ice Cream	Cola	Cheese Cake			
Eaten	0	5	0	0			
Cost	50	20	30	80			
Total	min 100						
CONSTRAINTS							
	Brownies	Ice Cream	Cola	Cheese Cake	Totals		Required
Calories	400	200	150	500	1000	>=	1000
Chocolate	3	2	0	0	10	>=	6
Sugar	2	2	4	4	10	>=	10
Fat	2	4	1	5	20	>=	8

Example 2

Knapsack Problem: (1 of 3)

Problem Statement

- We have **n** objects and a knapsack.
- Each object **i** has positive weight $\mathbf{w_i}$ and positive value $\mathbf{v_i}$.
- The knapsack can hold a weight of most **C**.

Object	**1**	**2**	**3**	**4**	**5**
weight	1	2	15	6	28
value	1	6	18	22	7

Task

- We wish to select a set of objects to put in the knapsack that weighs less than C and has maximum total value.
- In this example, the capacity of the knapsack C is 11.

Example 2

Knapsack Problem: (2 of 3)

Formulation of the general knapsack problem (with n items)

- Decision variables
  x_j: 1 if we choose to include item "j" in the bag, 0 otherwise.

- Objective function
  Maximize the utility of the items that will be in the bag:

$$\text{Max} \sum_{j=1}^{n} v_j \, x_j$$

- Constraints

$$\sum_{j=1}^{n} w_j x_j \leq C$$

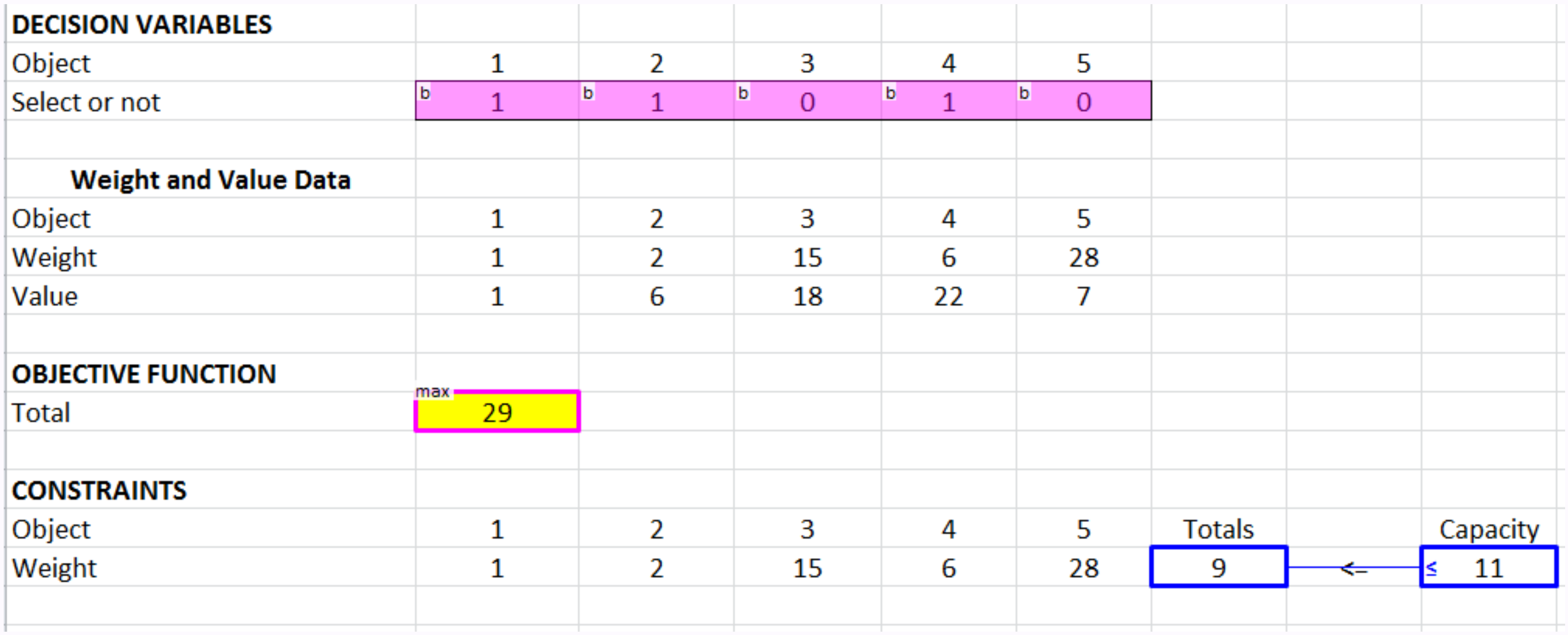
$$x_j \in \{0,1\} \quad \forall j \in \{1, \dots, n\}$$

Task:

- Solve this in OpenSolver on your own!
  Hint: put additional constraints to make x as binary variables

Knapsack Problem: (3 of 3)

Solution:



DECISION VARIABLES								
Object	1	2	3	4	5			
Select or not	b 1	b 1	b 0	b 1	b 0			
Weight and Value Data								
Object	1	2	3	4	5			
Weight	1	2	15	6	28			
Value	1	6	18	22	7			
OBJECTIVE FUNCTION								
Total	max 29							
CONSTRAINTS								
Object	1	2	3	4	5	Totals		Capacity
Weight	1	2	15	6	28	9	<=	≤ 11