MATH2605 Operational Research and Monte-Carlo Simulations Portfolio

Task 1:

Using the information provided in the question, we can rewrite the data in a tabled format. This makes the data easier to understand at a glance, as well as making it easier to decide our variables and constraints.

Factory	Cost (£1000's)	Production (Std)	Production (SUV)
A	210	50	37
B	182	0	63
C	170	51	0

Decision Variables

Deciding on our decision variables is straightforward. We want inequalities that deal with the total money spent over a time period, and we know the total money spent per day on each factory. If we make our decision variables the number of days that these factories operate, we can easily make our inequalities as we want them.

Let $X = Number\ of\ days\ in\ September\ for\ Factory\ A\ to\ operate$

Let $Y = Number\ of\ days\ in\ September\ for\ Factory\ B\ to\ operate$

Let $Z = Number\ of\ days\ in\ September\ for\ Factory\ C\ to\ operate$

Constraints

The constraints are also straightforward. We know that factories cannot be open for a negative number of days, and we also know that they cannot be open for more than 30 days (The number of days in September). We also know how many of each type of car we plan on producing, as well as how many each factory is able to produce. That gives us the following constraints:

$$X \geq 0;\ Y \geq 0;\ Z \geq 0$$

$$X \leq 30;\ Y \leq 30; Z \leq 30$$

$$50X + 51Z \geq 1050$$

$$37X + 63Y \geq 1500$$

Objective Function

As a company, we aim to spend as little as possible. Our total cost will be the cost per factory each day, multiplied by the number of days each factory is open. Hence the objective function is to **Minimise** the following:

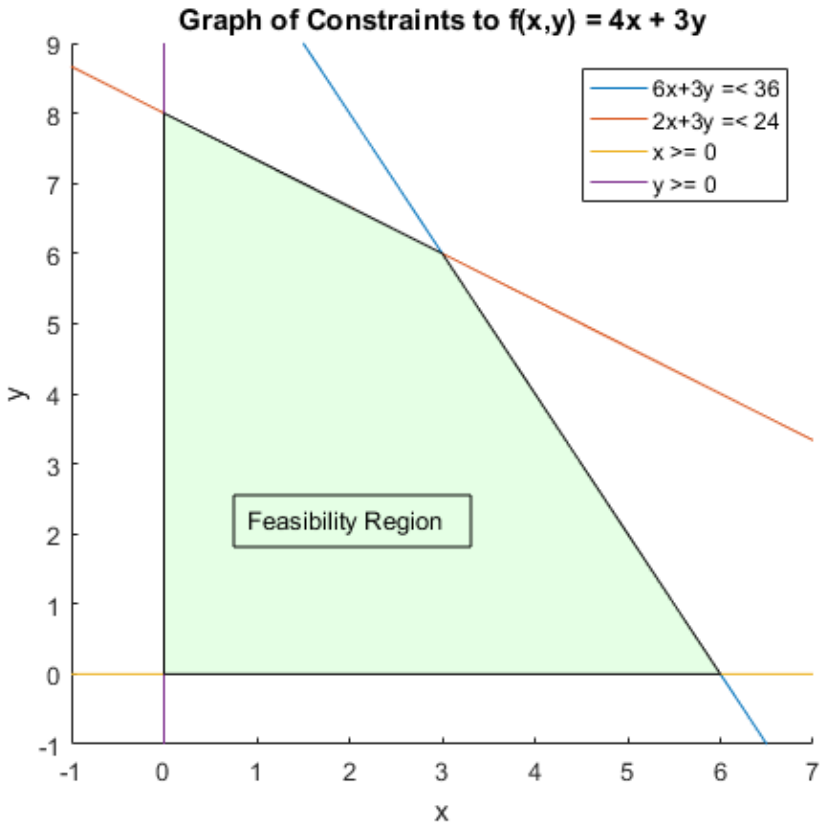
$$f(X, Y, Z) = 210X + 182Y + 170Z$$

Task 2

Plotting the feasibility region.

```
% Variable Declaration
x = -20:20;
y1 = (36 - 6*x)/3;
y2 = (24 - 2*x)/3;
x1 = x*0;
y3 = x*0;

%Plot block
f1 = figure('Name','Feasibility Region','Position',[50,50,500,450]); hold
on;
plot(x, y1, x, y2, x, x1, y3, x);
hold on;
title('Graph of Constraints to f(x,y) = 4x + 3y');
xlabel('x');
ylabel('y');
patch('Faces',[1,2,3,4],'Vertices',[0, 0; 0, 8; 3, 6; 6, 0],'FaceColor',
'g', 'FaceAlpha',0.1)
legend('6x+3y =< 36', '2x+3y =< 24', 'x >= 0', 'y >= 0');
xlim([-1 7]);
ylim([-1 9]);
annotation('textbox',[.3,.2,.2,.2],'String','Feasibility
Region','FitBoxToText','on');
```

Output:



Computation of observed corners:

We take the values of x, y at the corners of the feasibility region, i.e., when two or more lines intersect. Then, by substituting those values for x, y the in the objective function, we find feasible solutions to the problem and their function value. We know that the maximum exists on these corners and hence by choosing the largest resulting function value, we also deduce the corresponding solutions at the maximum.

Admittedly the following loop is too much code for a small set of calculations such as we have, however the following loop could be implemented into problems with a much larger number of corners to evaluate, saving the time required to perform many calculations for the time to simply enter data.

```
cornerY = [6, 8, 0];
cornerX = [3, 0, 6];
objFunc = zeros(length(cornerY));
for i= 1:length(cornerX)
    objFunc(i) = 4*cornerX(i) + 3*cornerY(i);
    fprintf('\nWhen:          The objective function gives:\n x = %i
4x + 3y = %i\n y = %i', cornerX(i), objFunc(i), cornerY(i));
end
 maxIndex = find(max(objFunc));
 fprintf('\n\nHence the solutions are:\n x = %i\n y = %i\n\nAnd the
maximum value: \n %i\n', cornerX(maxIndex), cornerY(maxIndex),
objFunc(maxIndex));
```

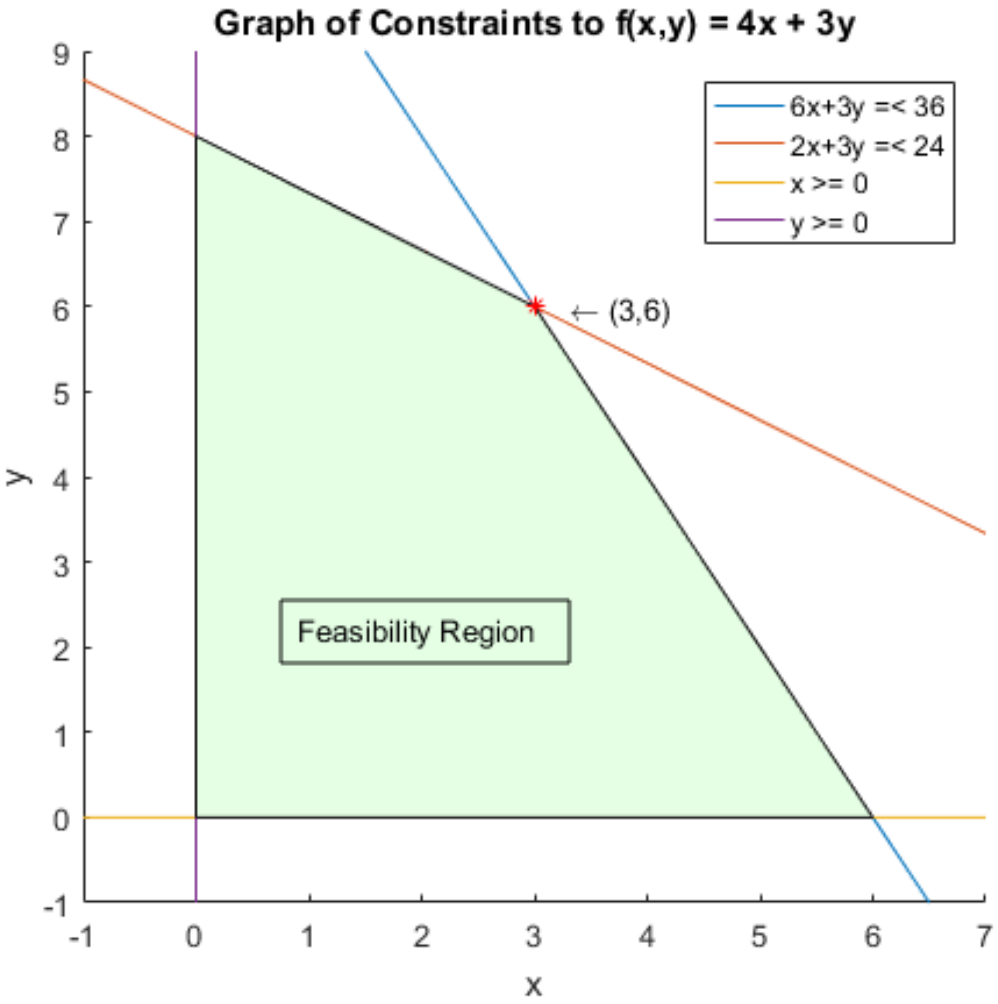
Output:

```
When:          The objective function gives:
 x = 3          4x + 3y = 30
 y = 6
When:          The objective function gives:
 x = 0          4x + 3y = 24
 y = 8
When:          The objective function gives:
 x = 6          4x + 3y = 24
 y = 0

Hence the solutions are:
 x = 3
 y = 6


And the maximum value:

 30
```

Annotating Plot

Having observed the plot and considered the corners of the feasibility region, we mark the corner at which we found the maximum of the objective function.

```
text(3,6, '    \leftarrow (3,6)');
plot(3,6,'r*');
```



Task 3:

Objective Function (Minimise):

$$f(x_1, x_2) = 10x_1 + 2x_2$$

We are given the following inequalities:

$$x_1 + x_2 \geq 1$$

$$x_2 - 5x_1 \leq 0$$

$$x_1 - x_2 \geq -1$$

$$x_{1,2} \geq 0$$

In order to minimise this objective function using the method described in section 2.3 of the course notes, we must first turn the inequalities into equations. We will do this by adding the slack variables $s_{1,2,3}$ into our inequalities. Note, that if the inequality uses $\leq$, then we must instead take the slack variable away from it, to ensure that the equation still holds true when all our slack variables are positive.

Therefore:

$$x_1 + x_2 - s_1 = 1$$

$$x_2 - 5x_1 + s_2 = 0$$

$$x_1 - x_2 - s_3 = -1$$

$$x_{1,2} \geq 0$$

$$s_{1,2,3} \geq 0$$

Now we create a table that sets pairs of variables equal to zero, and work out the values of the other variables based on this.

X1	X2	X3	X4	X5	Feasible?	F = 10X1 + 2X2
0	0	*	*	*		
0	*	0	*	*		
0	*	*	0	*		
0	*	*	*	0		
*	0	0	*	*		
*	0	*	0	*		
*	0	*	*	0		
*	*	0	0	*		
*	*	0	*	0		
*	*	*	0	0		

For example, the first line has $x_{1,2} = 0$. Using our equations above, we can then show:

$$0 + 0 - s_1 = 1$$

$$0 - 5(0) + s_2 = 0$$

$$0 - 0 - s_3 = -1$$

Giving us that, for this pair being equal to 0, we find:

$$s_1 = -1$$

$$s_2 = 0$$

$$s_3 = 1$$

Because we have defined $x_{1,2} \geq 0$ & $s_{1,2,3} \geq 0$ we know that this solution isn't feasible, and hence we don't need to work out the value of the function with these values.

We can do this for each row in our table, which gives us the values below:

X1	X2	S1	S2	S3	Feasible?	F = 10X1 + 2X2
0	0	-1	0	1	NO	
0	1	0	-1	0	NO	
0	0	-1	0	1	NO	
0	1	0	-1	0	NO	
1	0	0	5	2	YES	10
0	0	-1	0	1	NO	
-1	0	-2	-5	0	NO	
1/6	5/6	0	0	2/6	YES	20/6 = 10/3
0	1	0	-1	0	NO	
1/4	5/4	1/2	0	0	YES	5

We can see form this table that we only have 3 feasible solutions that fit our inequalities and function. Because we want to minimise our function, we look for the values which will give us the smallest answer when plugged in.

From the table, we can see that the minimum is found when:

$$x_1 = \frac{1}{6}, x_2 = \frac{5}{6} \Rightarrow f(x_1, x_2) = \frac{10}{3}$$

Task 4

Defining the problem:

```
Objective:
  Maximise

Objective function:
  f(x1,x2,x3,x4) = (3*x1) + (4*x2) + x3 + (5*x4)

Domain:
  x1 ≥ 0
  x2 ≥ 0
  x3 ≥ 0
  x4 ≥ 0

Constraints:
  x1 + x2 + (4*x3) ≤ 20
 (2*x2) + (5*x3) ≤ 40
 (2*x1) + (6*x3) + (4*x4) ≤ 50
  x2 + x3 + (3*x4) ≤ 30
```

Variable Declaration

```
% The objective function
f = [-3, -4, -1, -5];

% Matrix containing vectors of decision variable coefficients derived from
% the LHS of the constraining inequalities.
A = [1,1,4,0; % <-- e.g. values taken from LHS of x1 + x2 + (4*x3) <= 20
    0,2,5,0;
    2,0,6,4;
    0,1,1,3;
    -1,0,0,0;
    0,-1,0,0;
    0,0,-1,0;
    0,0,0,-1];

% Vector containing RHS of constraining inequalities
b = [20; % <-- e.g. value taken from RHS of x1 + x2 + (4 * x3) <= 20
    40;50;30;0;0;0;0];
```

Running the function

```
[x, fval] = linprog(f, A, b);

% Solutions
fprintf('Solutions:\n   x1 = %4.2f\n   x2 = %4.2f\n   x3 = %4.2f\n   x4 =
%4.2f\n\n   Maximum Value of Objective Function = %4.2f',
x(1),x(2),x(3),x(4), -fval);
```

Output:

```
Optimization terminated.
Solutions:
   x1 = 11.00
   x2 = 9.00
   x3 = 0.00
   x4 = 7.00

   Maximum Value of Objective Function = 104.00
```

Task 5

Defining the problem:

```
Problem:

   Objective:
     Maximise

   Objective function:
     f(x1,x2) = -x1 - (1/3)*x2

   Domain:
     x1 ≥ 0
     x2 ≥ 0

   Constraints:
     x1 + x2 ≤ 2
     x1 + (1/4)*x2 ≤ 1
     x1 - x2 ≤ 2
     -(1/4)*x1 - x2 ≤ 1
     -x1 - x2 ≤ -1
     -x1 + x2 ≤ 2
```

Variable Declaration

```
% The objective function
f = [-1,-1/3];

% Matrix containing vectors of decision variable coefficients derived from
% the LHS of the constraining inequalities.
A = [1 1; % <-- e.g. values taken from LHS of x1 + x2 <= 2
    1 1/4; 1 -1; -1/4 -1; -1 -1; -1 1];

% Vector containing RHS of constraining inequalities
b = [2;% <-- e.g. value taken from RHS of x1 + x2 <= 2
    1; 2; 1; -1; 2];

% Variables required by the linprog() function to run, but whose vectors
% are in this case empty.

% Matrix containing vectors of decision variable coefficients derived from
% the LHS of the constraining equalities. (In this case, an empty vector)
Aeq = [];
```

```
% Vector containing RHS of constraining equalities (Again an empty
vector).
beq = [];

%Upper and lower bounds, respectively.
lb = [];
ub = [];

% Starting value of x, unrequired by the interior-point linprog algorithm
% as X0 is defined by the algorithm, yet declared as function fails to run
% without the parameter.
x0 = [];
```

Running the function

```
x =
linprog(f,A,b,Aeq,beq,lb,ub,x0,optimoptions('linprog','Algorithm','interio
r-point'));

% Solutions
fprintf('Solutions:\nx1 = %1.7f\nx2 = %1.7f\n', x(1), x(2))
```

Output:

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the selected value of the function
tolerance,
and constraints are satisfied to within the selected value of the
constraint
tolerance.

Solutions:
x1 = 0.6666667
x2 = 1.3333333
```

Discussion of Interior Point Algorithms

The interior point methods are algorithms that are useful in solving linear and nonlinear convex optimisation problems that contain inequalities as constraints, there are two such methods; the primal-dual IP and barrier methods.

The key difference between IPM and the simplex method is that rather than testing the boundary or vertices of the feasible region, an IPM starts in the interior of the feasibility region and travels towards the boundary before finding the optimal solution. This has several benefits over the simplex method, namely that for larger, more complex problems the IPM is much faster. This largely comes from the fact that the number of iterations required for IPM is independent from the size of the problem, and the fact that IPM has polynomial time complexity, that is that the time taken to complete will always be bounded from above by some polynomial.

An example of an IPM is the barrier method, which involves creating a function that implicitly includes the inequalities of the problem into the objective function as a sum of a function applied to each constraint such that all the functions are twice differentiable and that when the value of the constraining function approaches 0 the resulting value of the logarithm approaches infinity. This then restricts the objective functions value from crossing the boundary of the feasible region. Examples of these barrier functions, $B(x)$, are:

$$-\sum_{i=1}^{m} \ln(-g_i(x)) \qquad and \qquad -\sum_{i=1}^{m} \frac{1}{g_i(x)}$$

As the solutions to optimisation problems generally lie on the boundary itself, the summation is also multiplied by a term that gradually decreases to zero in order to also gradually remove the “barrier”. We can then utilise Newton’s method to iterate though a series of points satisfying any equality constraints, called a Central Path.

Task 6

Variable Declaration

```
lam = 0.5;
k = 3;
the = k/lam;
x = gamrnd(3,1/0.5,[10000,1]);
M = mean(x);
```

Comparison of Expected and Resulting Mean Values.

```
fprintf('The average mean of the randomly generated numbers of the erlang
distribution:\n      %f\nThe theoretical mean:\n      k/\x03BB = %f\nThe
difference between the theoretical mean and the exact mean:\n      %f\n',
M, the, abs(M-the));
```

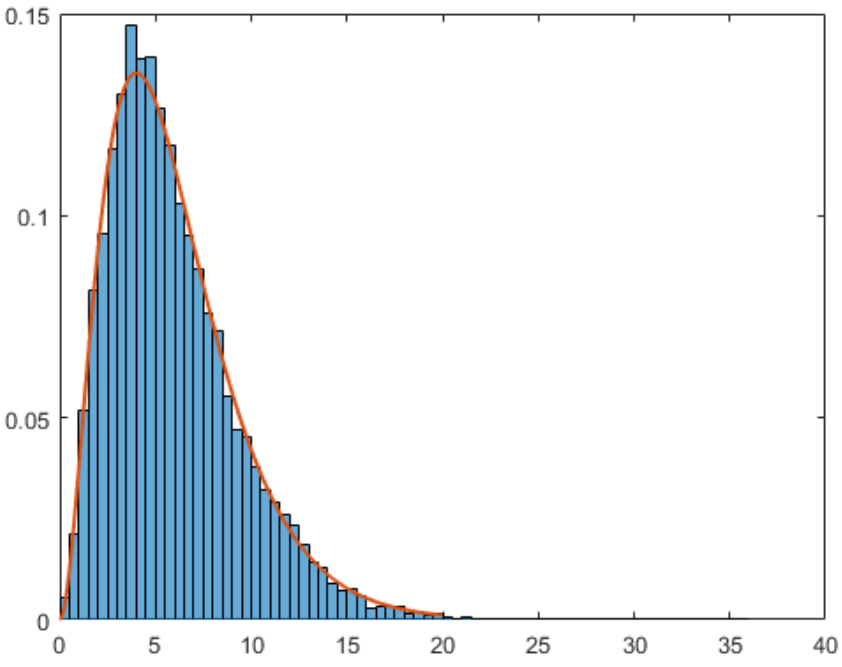
Output:

```
The average mean of the randomly generated numbers of the erlang
distribution:
      5.962437
The theoretical mean:
      k/λ = 6.000000
The difference between the theoretical mean and the exact mean:
      0.037563
```

Plot of Solutions

```
histogram(x, 'Normalization', 'PDF')
hold on
y = 0:0.1:20; f = ((lam^k).*(y.^(k-1)).*exp(-lam*y))/factorial(k-1);
plot(y,f, 'LineWidth',1.5)
```

Output:



Task 7

Common Variable Declaration:

```
fun = @(u1,A,X) cos(2*pi*u1+dot(A,X));
X = zeros(4,1);
```

Use of Monte Carlo Loop as a Function

For clarity, I have omitted declaring the Monte Carlo loop in this code as a function. Typical utilisation however would dictate that the loop be defined in a separate function file or (as of version R2016b) at the end of this file.

Consideration #1

```
% Variables and Container:
nSample = 10^6 ;
x = zeros(nSample , 1) ;
A = [1,1,1,1];
u1 = 1 ;

% Monte Carlo Loop
for i=1:nSample
  for j=1:4
      X(j)=rand();
  end
  x(i) = fun(u1,A,X);
end

% Error Estimation:
err = std(x) / sqrt(nSample) ;
```

Consideration #1 Results:

Output:

```
Consideration #1
(nsample=1000000)
u1 = 1.0
A = [1.0,  1.0,  1.0,  1.0]
Monte Carlo estimate = -0.351187 +/- 0.000462
Exact result = -0.351764
```

Consideration #2

```
% Variables and Container:
nSample = 4*10^6 ;
x = zeros(nSample , 1) ;
% A = [1,1,1,1];    !!Unnecessary!! Kept for clarity
% u1 = 1 ;          !!Unnecessary!! Kept for clarity

% Monte Carlo Loop
for i=1:nSample
  for j=1:4
      X(j)=rand();
  end
  x(i) = fun(u1,A,X);
end

% Error Estimation:
err = std(x) / sqrt(nSample) ;
```

Consideration #2 Results:

Output:

```
Consideration #2
(nsample=4000000)
u1 = 1.0
A = [1.0,  1.0,  1.0,  1.0]
Monte Carlo estimate = -0.351637 +/- 0.000231
Exact result = -0.351764
```

Conclusion of Consideration # 2

The error is defined as the standard deviation of the resulting values of the integral divided by the square root of the number of iterations made. In this case, the number of iterations were 4 times than that in the first consideration, hence the resulting error would be $\sqrt{4} = 2$ times smaller, as is the case in our results.

Consideration #3

```
% Variables and Container:
A = [.1,.1,.1,.1];
u1 = .1 ;
err = [];
meanr = [];
NSample = [];
for k=1:3:4
    nSample = k*10^6 ;
    x = zeros(nSample , 1) ;
%   Monte Carlo Loop
    for i=1:nSample
        for j=1:4
            X(j)=rand();
        end
        x(i) = fun(u1,A,X);
    end
    % Error Estimation:
    err = [err, std(x) / sqrt(nSample)] ;
    meanr = [meanr, mean(x)];
    NSample = [NSample,nSample];
end
```

Consideration #3 Results :

Output:

```
Consideration #3a
(nsample=1000000)
u1 = 1.0
A = [0.1,  0.1,  0.1,  0.1]
Monte Carlo estimate = 0.978435 +/- 0.000012
Exact result = 0.978434
Consideration #3b
(nsample=4000000)
u1 = 1.0
A = [0.1,  0.1,  0.1,  0.1]
Monte Carlo estimate = 0.978436 +/- 0.000006
Exact result = 0.978434
```

Conclusion of Consideration #3

Once again, to interpret our error estimations we refer to the computation of the error. As we discussed, as the denominator (√# of iterations) gets larger, the size of the error decreases. In a similar vein, as the numerator (Defined by std(x)) gets smaller, so too will the error. In this case, as we have decreased the size of the value of each integral by 10, we are then evaluating over a 10 x 4 = 40 times smaller range, the error has then also decreased by 40 times.

Task 8

```
% Common Variables and Container Variable
myfun = @(x) (x./sin(x)).^4;
Nsample = 10000 ;
Ncopy = 10 ;
xxx = zeros(Ncopy,1) ;
I = -((pi^3)/12)+2*pi*log(2)+((pi^3)*log(2)/3)-
(3/2)*pi*1.2020569031595942853997381615114499907649862923;

% *Using Matlab Integral Function*
I0 = integral(myfun, 0, pi/2);

% Matlab Integral Function Results
fprintf('Matlab Integral = %1.16f\n Exact Value = %1.16f\n\n' , I0,I) ;

% Using Standard Monte Carlo Method
x = zeros(Nsample , 1) ;
for i=1:Nsample
    rx = pi/2*rand() ;
    x(i) = pi/2*myfun(rx) ;
end

% Basic Monte Carlo Results:
meanr = mean(x) ;
err = std(x) / sqrt(Nsample) ;
fprintf('Monte Carlo estimate (nsample=%d) = %1.16f +/- %1.16f \n Exact
Value = %1.16f\n\n' ,Nsample, meanr , err, I) ;

% *Using Monte Carlo 2-Stratified*
for ic=1:Ncopy
    r1 = pi/4 * rand(Nsample/2 , 1) ;
    r2 = pi/4 + pi/4 * rand(Nsample/2 , 1) ;
    r1f = myfun(r1) ;
    r2f = myfun(r2) ;
    xxx(ic) = pi/4*(mean(r1f) + mean(r2f)) ;
end


% Monte Carlo 2-Stratified Results:
xm = mean(xxx) ; xerr = std(xxx)/sqrt(Nsample) ;
fprintf('2-Stratified MC estimate (nsample=%d, Ncopy=%d) = %1.16f+/-
%1.16f \n Exact Value = %1.16f\n\n' , Nsample, Ncopy, xm , xerr, I) ;
```

```
% *Using Monte Carlo 4-Stratified*
for ic=1:Ncopy
    r1 = pi/8 * rand(Nsample/4 , 1) ;
    r2 = pi/8 + pi/8 * rand(Nsample/4 , 1) ;
    r3 = pi/4 + pi/8 *rand(Nsample/4, 1) ;
    r4 = 3*pi/8 + pi/8 *rand(Nsample/4, 1) ;
    r1f = myfun(r1) ;
    r2f = myfun(r2) ;
    r3f = myfun(r3) ;
    r4f = myfun(r4) ;
    xxx(ic) = pi/8*(mean(r1f) + mean(r2f)+mean(r3f) + mean(r4f)) ;
end

% Monte Carla 4-Stratified Results:
xm = mean(xxx) ; xerr = std(xxx)/sqrt(Nsample) ;
fprintf('4-Stratified MC estimate (nsample=%d, Ncopy=%d) = %1.16f+/-
%1.16f \n Exact Value = %1.16f\n\n' , Nsample, Ncopy, xm , xerr, I) ;
```

Output:

```
Matlab Integral = 3.2707271732080669
 Exact Value = 3.2707271732080683

Monte Carlo estimate (nsample=10000) = 3.2760369563267506 +/-
0.0203132268504575
 Exact Value = 3.2707271732080683

2-Stratified MC estimate (nsample=10000, Ncopy=10) = 3.2755289272599115
+/- 0.0001133342195576
 Exact Value = 3.2707271732080683

4-Stratified MC estimate (nsample=10000, Ncopy=10) = 3.2701268985021898
+/- 0.0000552537682412
 Exact Value = 3.2707271732080683
```

Conclusion

Clearly, we see a reduction in error as stratified sampling techniques are used, the results indicate that the error is further reduced as the integral is split into a larger number of sub-intervals.

Task 9

Importing Data from csv file.

Matlab returns a warning that variable names were changed during the import of the .cvs file, this is likely because the source data sheet contains special characters (spaces) in the column headers.

```
gReviewRatigns = readtable('google_review_ratings.csv');
% Importing specific columns from table.
% Variables used to store names for later debugging
column1 = 'Category11';
column2 = 'Category22';
pubAverage = gReviewRatigns{:,{column1}} ;
cafeAverage = gReviewRatigns{:,{column2}} ;
```

OUTPUT:

```
Warning: Variable names were modified to make them valid MATLAB
identifiers.
```

Cleaning Data

As there may be data in the .csv file that causes errors or is not within the range expected (1 to 5), we perform a little preliminary analysis and remove erroneous data. For the sake of the end user (and debugging), a warning is given when data is removed.

```
pubAverage = str2double(pubAverage);
count = 0;
for i = 1:length(pubAverage)
    if isnan(pubAverage(i-count))
        pubAverage(i-count) = [];
        count=count+1;
    end
end
if count > 0
    fprintf('Source .csv file contains unreadable data.\nEntry cleanup
performed on %s\n%i element removed.\n\n', column1, count)
end
count = 0;
for i = 1:length(pubAverage)
    if pubAverage(i-count)<1 || pubAverage(i-count)>5
        pubAverage(i-count) = [];
```

```
        count=count+1;
    end
end
if count > 0
    fprintf('Source .csv file contains erroneous data.\nEntry cleanup
performed on %s\n%i element removed.\n\n', column1, count)
end
count = 0;
for i = 1:length(cafeAverage)
    if isnan(cafeAverage(i-count))
        cafeAverage(i-count) = [];
        count=count+1;
    end
end
if count > 0
    fprintf('Source .csv file contains unreadable data.\nEntry cleanup
performed on %s\n%i element removed.\n\n', column2, count)
end
count = 0;
for i = 1:length(cafeAverage)
    if cafeAverage(i-count)<1 || cafeAverage(i-count)>5
        cafeAverage(i-count) = [];
        count=count+1;
    end
end
if count > 0
    fprintf('Source .csv file contains erroneous data.\nEntry cleanup
performed on %s\n%i element removed.\n\n', column2, count)
end
```

OUTPUT:

```
Source .csv file contains unreadable data.
Entry cleanup performed on Category11
1 element removed.

Source .csv file contains erroneous data.
Entry cleanup performed on Category11
452 element removed.

Source .csv file contains erroneous data.
Entry cleanup performed on Category22
2618 element removed.
```

Bootstrap analysis of the IQ median data.

The bootstrap method is a resampling technique that allows us to get multiple theoretical samples when we only have 1 to start with.

```
pubData = pubAverage;
cafeData = cafeAverage;
pubnData = length(pubData);
cafenData = length(cafeData);
pubmt = median(pubData);
pubmm = mean(pubData);
cafemt = median(cafeData) ;
cafemm = mean(cafeData) ;
fprintf('Pub Data:                                    Cafe Data: \n');
fprintf('Number of Pub data items = %d          Number of Cafe data items
= %d\n' , pubnData, cafenData) ;
fprintf('Median of Pub data = %f            Median of Cafe data = %f\n' ,
pubmt, cafemt ) ;
fprintf('Mean of Pub data = %f              Mean of Cafe data = %f\n\n' ,
pubmm, cafemm )  ;

% Using the bootstrap method we can generate 1000 samples, each the same
% size of the original sample
nboot = 1000 ;
fprintf('Number of bootstrap samples %d \n' , nboot) ;
store = zeros(1 , nboot) ;
bdata = zeros(1,pubnData) ;

%%Pub Bootstrap
for j = 1:nboot
    %  create bootstrap sample
    r = randi([1 pubnData],1,pubnData);
    for i = 1:pubnData
       bdata(i) = pubData( r(i)  ) ;
    end
    % For each of the bootstrap samples, we calculate the medians
    store(j) =  median(bdata);
end
%Then we sort these medians
pubSort = sort(store);
pubStd = std(store);

%%Cafe Bootstrap
bdata = zeros(1,cafenData) ;
for j = 1:nboot
```

```
    %  create bootstrap sample
    r = randi([1 cafenData],1,cafenData);
    for i = 1:cafenData
       bdata(i) = cafeData( r(i)  ) ;
    end
    store(j) =  median(bdata);
end
cafeSort = sort(store);
cafeStd = std(store);

fprintf('Pub Median SD = %f                Cafe Median SD = %f \n\n' ,
pubStd, cafeStd )
```

OUTPUT:

```
Pub Data:                                  Cafe Data:
Number of Pub data items = 5003            Number of Cafe data items = 2838
Median of Pub data = 2.150000              Median of Cafe data = 2.015000
Mean of Pub data = 2.701275                Mean of Cafe data = 2.762040

Number of bootstrap samples 1000
Pub Median SD = 0.034127                   Cafe Median SD = 0.049722
```

Confidence Intervals

Having calculated the median and sorted the bootstrap samples, we can then find the upper and lower percentiles, which then gives us our Confidence Intervals.

```
alpha = 5;
upperPer = 100-(alpha/2);
lowerPer = alpha/2;
pubStore = sort(pubStore);
cafeStore = sort(cafeStore);

pubUpper = prctile(pubSort, upperPer);
pubLower = prctile(pubSort, lowerPer);
cafeUpper = prctile(cafeSort, upperPer);
cafeLower = prctile(cafeSort, lowerPer);

fprintf('Comparison of Medians using bootstrap method with 95%% Confidence
Intervals: \n') ;
fprintf('Pub Median Upper Bound = %f        Cafe Median Upper Bound = %f
```

```
\n' ,  pubUpper, cafeUpper );
fprintf('Pub Median Lower Bound = %f        Cafe Median Lower Bound = %f
\n\n' ,  pubLower, cafeLower );
```

OUTPUT:

```
Comparison of Medians using bootstrap method with 95% Confidence
Intervals:
Pub Median Upper Bound = 2.190000        Cafe Median Upper Bound =
2.070000
Pub Median Lower Bound = 2.075000        Cafe Median Lower Bound =
1.900000
```

Conclusion

For this example we are looking at the median rating for pubs and cafes in the local area.

We have a sample median of 2.15 and 2.015 respectively.

We also find the pub median CI = [2.075, 2.190], and the cafe median Ci = [1.90, 2.07]

It would therefore be our suggestion that you were to build slightly more pubs than cafes, as we can be fairly confident that the average rating for pubs is slightly higher than that of the average rating for cafes.

Task 10

```
%Common Variables
num_iterations = 10000;
```

Standard Game Variant

```
% Debugging
%ind_game = input('Would you like to see individual game results? (Give
boolean response)\n');
% Variable Declaration
AWin = 0;
BWin = 0;
GameLengths = zeros(num_iterations,1);
clc;
%Monte Carlo Loop
for i=1:num_iterations
    Astake = 2000 ;
    Bstake = 2000 ;
    bet_per_game = 50 ;
    count = 0 ;
    while( Astake > 0 && Bstake > 0 )
        rr = rand()  ;
        if(  rr < 0.49 )
            Astake = Astake + bet_per_game ;
            Bstake = Bstake - bet_per_game  ;
        else
            Astake = Astake - bet_per_game  ;
            Bstake = Bstake + bet_per_game  ;
        end
    count = count +  1 ;
    end
    % Debugging Continued
    %if (ind_game == 1)
    %    fprintf('Player stake = £%d | Bank stake = £%d | Length of game =
%d bets\n', Astake, Bstake,count)
    %end
    if (Bstake <= 0)
        AWin = AWin +1;
    else
        BWin = BWin +1;
    end
    GameLengths(i) = count;
end
Prob_A_Win = AWin/num_iterations*100;
```

```
Prob_B_Win = BWin/num_iterations*100;
mean_length = sum(GameLengths)/num_iterations;
error = std(GameLengths)/sqrt(num_iterations);
fprintf('\n\nThe probabilities of either player winning the game is as
follows:\n\n   Player Wins (Player A) - %3.2f%%\n   Bank Wins (Player B)
- %3.2f%%\n\nThe mean length of each game is:\n\n   %1.0f \xOOB1 %1.0f
bets\n\n',Prob_A_Win,Prob_B_Win,mean_length,error)
```

Output:

```
The probabilities of either player winning the game is as follows:

   Player Wins (Player A) - 16.61%
   Bank Wins (Player B)   - 83.39%

The mean length of each game is:

   1323 ± 10 bets
```

Kelly Criterion Game Variant - Monte Carlo Evaluation

```
% Debugging
%Cont = input('Press enter to continue')
%ind_game = input('Would you like to see individual game results? (Give
boolean response)\n');
% Variable Declaration
AWin = 0;
BWin = 0;
GameLengths = zeros(num_iterations,1);
clc;
%Monte Carlo Loop
for i=1:num_iterations
    Astake = 2000 ;
    Bstake = 2000 ;
    count = 0 ;
    bet_per_game = 0;
    % Complicated while loop, but will also check that each player can
    % uphold bet without going into negative finances.
    while((Astake > 0 && Bstake > 0) && (Astake >= bet_per_game && Bstake
>= bet_per_game))
        rr = rand()  ;
        bet_per_game = Bstake*(0.51*(1+1)-1)/1 ;
        if(  rr < 0.49 )
            Astake = Astake + bet_per_game ;
            Bstake = Bstake - bet_per_game  ;
        else
            Astake = Astake - bet_per_game  ;
```

```
            Bstake = Bstake + bet_per_game  ;
        end
    count = count +  1 ;
    end
    % Debugging Continued
    %if (ind_game == 1)
    %    fprintf('Player stake = £%f | Bank stake = £%f | Length of game =
%d bets\n', Astake, Bstake,count)
    %end
    if (Bstake < Astake)
        AWin = AWin +1;
    else
        BWin = BWin +1;
    end
    GameLengths(i) = count;
end
```

Kelly Criterion Results Results

```
Prob_A_Win = AWin/num_iterations*100;
Prob_B_Win = BWin/num_iterations*100;
mean_length = sum(GameLengths)/num_iterations;
error = std(GameLengths)/sqrt(num_iterations);
fprintf('\n\nThe probabilities of either player winning the game is as
follows:\n\n   Player Wins (Player A) - %3.2f%%\n   Bank Wins (Player B)
- %3.2f%%\n\nThe mean length of each game is:\n\n   %1.0f \xOOB1 %1.0f
bets\n\n',Prob_A_Win,Prob_B_Win,mean_length,error)
```

Output:

```
The probabilities of either player winning the game is as follows:

   Player Wins (Player A) - 0.00%
   Bank Wins (Player B)   - 100.00%

The mean length of each game is:

   3339 ± 55 bets
```

Conclusion

It would be my reccomendation that the bank impliments the Kelly criterion as it both increases the average length of the game (hence player retention) and also the percentage chance of the bank winning the game (higher profits).

As for the player implementing the Kelly criterion, I am unsure of what the question refers to, so I will consider both scenarios:

Case 1: The Kelly Criterion is in regards to the bank's stake is not reccomended as previously discussed.

Case 2: The Kelly criterion is in regards to the players stake. This may be reccomended as it would increase the length of the game dramatically, although the stake would more often than not reduce to a very low value due to the larger likelyhood of the bank winning a game. I would recommend this then for a player interested in a longer game time but not for a player interested in profit.

Task 11

Simplex by hand

We are asked to maximise the following function, with the following constraints:

Objective function:

$$f = 2x_1 + x_2 + x_3$$

Constraints:

$$3x_1 + 2x_2 - x_3 \leq 6$$
$$x_1 + x_2 + x_3 \leq 4$$
$$2x_1 - x_2 + 2x_3 \leq 4$$

We must first introduce the slack variables $x_{4,5,6}$, in order to turn our inequalities into equations. This gives us:

$$3x_1 + 2x_2 - x_3 + x_4 = 6$$
$$x_1 + x_2 + x_3 + x_5 = 4$$
$$2x_1 - x_2 + 2x_3 + x_6 = 4$$

We can now check for a BFS by plugging $x_{1,2,3}$ = 0. This gives:

$$x_4 = 6$$
$$x_5 = 4$$
$$x_6 = 4$$

First Run through

This BFS is not optimal as all of these are positive. We therefore look back to our objective function and note that x_1 has the largest coefficient, making it our leaving variable for the simplex method. $x_{2,3}$ are therefore our leading variables. We will therefore choose to rewrite our constraints so that x_1 has coefficient of 1 in all equations via the ratio test. This gives us:

$$① \quad x_1 + \frac{2}{3}x_2 - \frac{1}{3}x_3 + \frac{1}{3}x_4 = 2$$
$$② \quad x_1 + x_2 + x_3 + x_5 = 4$$
$$③ \quad x_1 - \frac{1}{2}x_2 + x_3 + \frac{1}{2}x_6 = 2$$

Here we note that both ① & ③ have $RHS = 2$. Due to the fact that our objective function has coefficients of 1 & 2, and equation ③ has coefficients of 1 & $\frac{1}{2}$, we will chose ③ as the equation to rewrite in terms of x_1

$$④ \quad x_1 = 2 + \frac{1}{2}x_2 - x_3 - \frac{1}{2}x_6$$

Subbing ④ back into our objective function as well as ① & ② gives us:

$$f = 4 + 2x_2 - x_3 - x_6$$

$$① \quad \frac{7}{2}x_2 - 4x_3 + x_4 - \frac{3}{2}x_6 = 0$$

$$② \quad \frac{3}{2}x_2 + x_5 - \frac{1}{2}x_6 = 2$$

$$③ \quad 2x_1 - x_2 + 2x_3 + x_6 = 4$$

Because we do not have all negative x_i in our Objective function, we must iterate the method again.

Run Through 2

x_2 has largest coefficient so becomes the leaving variable. We can therefore ratio test the equations with reference to x_2, giving:

$$① \quad x_2 - \frac{8}{7}x_3 + \frac{2}{7}x_4 - \frac{3}{2}x_6 = 0$$

$$② \quad x_2 + \frac{2}{3}x_5 - 3x_6 = \frac{4}{3}$$

$$③ \quad 2x_1 - x_2 + 2x_3 + x_6 = 4$$

Equation ①, has the smallest RHS. We therefore rewrite it in terms of x_2, giving:

$$④ \quad x_2 = \frac{8}{7}x_3 - \frac{2}{7}x_4 + \frac{3}{7}x_6$$

Subbing ④ back into our objective function as well as ② & ③ gives us:

$$f = 4 + \frac{9}{7}x_3 - \frac{4}{7}x_4 - \frac{1}{7}x_6$$

$$① \quad \frac{7}{2}x_2 - 4x_3 + x_4 - \frac{3}{2}x_6 = 0$$

$$② \quad \frac{12}{7}x_3 - \frac{3}{7}x_4 + x_5 + \frac{1}{7}x_6 = 2$$

$$③ \quad 2x_1 + \frac{6}{7}x_3 + \frac{2}{7}x_4 + \frac{4}{7}x_6 = 4$$

Because we do not have all negative x_i in our Objective function, we must iterate the method again.

Run Through 3

x_3 has largest coefficient so becomes the leaving variable. We have used equations ① & ③ , so we now rewrite equation ② in terms of x_3, giving:

$$④ \quad x_3 = \frac{1}{4}x_4 - \frac{7}{12}x_5 - \frac{1}{12}x_6 + \frac{7}{6}$$

Subbing ④ back into our objective function as well as ① & ③ gives us:

$$f = \frac{31}{6} - \frac{89}{20}x_4 - \frac{7}{12}x_5 - \frac{19}{84}x_6$$

$$① \quad \frac{7}{2}x_2 + \frac{7}{3}x_5 - \frac{7}{6}x_6 = \frac{14}{3}$$

$$② \quad \frac{12}{7}x_3 - \frac{3}{7}x_4 + x_5 + \frac{1}{7}x_6 = 2$$

$$③ \quad 4x_1 - \frac{3}{7}x_4 + x_5 + \frac{1}{7}x + 6 = 2$$

Because we do have all negative x_i in our Objective function, we know that in order to maximise the function we want $x_{4.5.6}$ to be as small as possible, so we can set them equal to 0.

From equations ①, ②, ③, this gives us the answer:

$$x_1 = \frac{3}{2}, x_2 = \frac{4}{3}, x_3 = \frac{7}{6}$$

Plugging this back into our objective function gives us the maximum value:

$$f\left(\frac{3}{2}, \frac{4}{3}, \frac{7}{6}\right) = \frac{11}{2}$$

Task 12

```
% Variable Declaration
fprintf('Problem:\n   Objective:\n     Maximise\n   Objective Function:\n
f(x1,x2,x3,x4) = x3 + x4\n   Domain:\n     x1,x2,x3,x4 \x2265 0\n
x3,x4 \x2264 32\n   Constraints:\n     (0.1 * x1) + (0.2 * x2) \x2264
1000\n     (0.002 * x1) + (0.001 * x2) - x3 = 0\n     (0.001 * x1) +
(0.003 * x2) - x4 = 0\n\n');
f = -[0, 0, 1, 1];
intcon = 1:4;
A = [0.1,0.2,0,0];
b = 1000;
Aeq = [0.002,0.001,-1,0;0.001,0.003,0,-1];
Beq = [0;0];
lb = zeros(4,1);
ub = [Inf;Inf;32;32];

% In-Built Matlab Function
[x,fval]=intlinprog(f,intcon,A,b,Aeq,Beq,lb,ub);
```

Output:

```
Problem:
   Objective:
     Maximise
   Objective Function:
     f(x1,x2,x3,x4) = x3 + x4
   Domain:
     x1,x2,x3,x4 ≥ 0
     x3,x4 ≤ 32
   Constraints:
     (0.1 * x1) + (0.2 * x2) ≤ 1000
     (0.002 * x1) + (0.001 * x2) - x3 = 0
     (0.001 * x1) + (0.003 * x2) - x4 = 0

LP:                Optimal objective value is -30.000000.


Optimal solution found.
Intlinprog stopped at the root node because the objective value is within
a gap
tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (the
default
value). The intcon variables are integer within tolerance,
options.IntegerTolerance = 1e-05 (the default value).
```

Solutions

```
fprintf('Solutions:\n   x1 = %4.2f\n   x2 = %4.2f\n   x3 = %4.2f\n   x4 = %4.2f\n\n   f = %4.2f', x(1),x(2),x(3),x(4),-fval);
```

Output:

```
Solutions:
   x1 = 10000.00
   x2 = 0.00
   x3 = 20.00
   x4 = 10.00

   f = 30.00
```

Conclusion

The solutions show that by investing all advertisement into YouTube would result in 30 students overall; 10 mathematics students and 20 english students.

The linear programming model has found the maximum number of students using the smallest amount of the budget. Clearly, the results indicate that the at the current budget the maximum is found when the entire budget is spent, yet that maximum is not in fact the maximum number of students which the school can support. It is clear then that the advertising budget is in fact insufficient and my suggestion would therefore be to increase the advertising budget.

Task 13

Problem:
  Objective:
    Minimize
  Objective Function:
    f(x1,x2,...,x16) = (145 * x1) + (122 * x2) + (130 * x3) + (95 * x4) + (80 * x5) + (63 * x6) + (85 * x7) + (48 * x8) + (121 * x9) + (107 * x10) + (93 * x11) + (69 * x12) + (118 * x13) + (83 * x14) + (116 * x15) + (80 * x16)
  Domain:
    x1, x2, ..., x16 ≥ 0
    x1, x2, ..., x16 ≤ 1
  Constraints:
    x11 + x12 + x13 + x14 = 1
    x21 + x22 + x23 + x24 = 1
    x31 + x32 + x33 + x34 = 1
    x41 + x42 + x43 + x44 = 1
    x11 + x21 + x31 + x41 = 1
    x12 + x22 + x32 + x42 = 1
    x13 + x23 + x33 + x43 = 1
    x14 + x24 + x34 + x44 = 1

```
% Variable Declaration
f = [145,122,130,95,80,63,85,48,121,107,93,69,118,83,116,80];

A = [];
b = [];
Aeq = [1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0;
       0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0;
       0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0;
       0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1;
       1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0;
       0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0;
       0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0;
       0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1];
beq = [1;1;1;1;1;1;1;1];
intcon = 1:16;
lb = zeros(16,1);
ub = ones(16,1);

% In-Built Matlab Function
[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
```

Solutions

```
fprintf('Solutions:\n  x11 = %1.0f\n  x12 = %1.0f\n  x13 = %1.0f\n  x14 =
%1.0f\n  x21 = %1.0f\n  x22 = %1.0f       The minimum function value is:\n
x23 = %1.0f                %5.3f\n  x24 = %1.0f\n  x31 = %1.0f\n  x32 =
%1.0f\n  x33 = %1.0f\n  x34 = %1.0f\n  x41 = %1.0f\n  x42 = %1.0f\n  x43 =
%1.0f\n  x44 =
%1.0f\n',x(1),x(2),x(3),x(4),x(5),x(6),x(7),fval,x(8),x(9),x(10),x(11),x(1
2),x(13),x(14),x(15),x(16))
```

Output:

```
Solutions:
  x11 = 0
  x12 = 0
  x13 = 0
  x14 = 1
  x21 = 1
  x22 = 0       The minimum function value is:
  x23 = 0                 351.000
  x24 = 0
  x31 = 0
  x32 = 0
  x33 = 1
  x34 = 0
  x41 = 0
  x42 = 1
  x43 = 0
  x44 = 0
```

Conclusion

Hence, the work should be assigned as follows:

Part	Team Member
1	Member 2 (Gavin)
2	Member 4 (Ruth)
3	Member 3 (Bob)
4	Member 1 (Sarah)

Task 14

Problem:

Objective:
Maximize

Objective Function:
f(x,y) = 14x + 18y

Domain:
x,y >= 0
x,y <= 1

Constraints:
-x+3y <= 6

Call for Package Functionality

```
> with(Optimization):
```

Variable Declaration

```
> obj := 14*x+18*y:
  constr := {-x+3*y <= 6, 7*x+y <= 35, x>=0, y>=0}:
  opts := assume = {nonnegative,integer}, maximize:
```

Maple Function

```
> LPSolve(obj,constr,opts);
```

$$[110, [x = 4, y = 3]] \quad \textbf{(1)}$$

Hence, the maximum value of f(x,y) is found when x = 4, y = 3, giving:

f (4, 3) = 110

Task 15

```
%The following inputs are for ease of use for the end user, the date of
%the order whose minimisation is required is to be inputted and the values
%used to create a dynamic SQL query.
%For ease of publishing, these input variables have been commented out and
%their values assumed in a later declaration.

%queryYear = input('Please enter the week date of data required (yyyy):');
%queryMonth = input('Please enter the week date of data required (mm):');
%queryDay = input('Please enter the week date of data required (dd):');
%queryDate = strcat('SELECT * FROM keyboard_record where week_date="',
num2str(queryYear),'-',num2str(queryMonth,'%02d'),'-
',num2str(queryDay,'%02d'),'"');

%This line would be ommitted for it's following commented out line.
sqlquery = 'SELECT * FROM keyboard_record where week_date="2019-04-01"';
%sqlquery = queryDate;

conn = sqlite('keyboard_orders.db','readonly');
results = fetch(conn,sqlquery);
close(conn)
x = length(results)-2;
order = zeros(x,1);
for i = 1:x
    order(i) = double(cell2mat(results(i+2)));
end

% Matlab wants to minimize
f = [ 14 13 11 12 13 13 ]
```

Output:

```
f =

    14    13    11    12    13    13
```

Inequalities

```
% Representation : [x1,x2,x3,y1,y2,y3]
% x1,x2,x3  => Factory in China working on orders from company 1,2,3
% respectively
% y1,y2,y3 => Factory in Korea working on orders from company 1,2,3
% respectively
```

Dynamic Arrays

As we are attempting to create a Matlab script more efficient by removing "by hand" entries, we must consider that as the manufacturer expands, it is likely that more companies may wish to make orders. To increase the codes efficiency and avoid needing to manually edit the code to adapt to these extra orders we instead looked at trying to create dynamically sized and self-defined arrays.

Usually, this process can become very complicated and very often is difficult to conceptualise. Fortunately, in this case, the arrays are defined by easy to code patterns.

```
% "A" matrix is essentially two initial rows:
% The first represents the China Factory working for company i (i=1,2,...)
% The second represents the Korea Factory working for company i
(i=1,2,...)
%
% The remaining rows are essentialy a negative identity matrix
representing
% the fact that the production for each company in regards to either
% factory is an integer value.
A1 = zeros(2,x*2);
for i = 1:x
    A1(1,i) = 1;
    A1(2,i+x) = 1;
end
A2 = -eye(x*2);
A = vertcat(A1,A2);

% "b" matrix is nice and simple, the first two elements simply a
% representation of the maximum production of the China and Korea
factories
% respectively.
b = zeros(1,x*2+2);
b(1,1) = 1200;
b(1,2) = 1000;

% "Aeq" is another simple matrix, essentially two identity matrices
% concatenated horizontally, representing the production in both factories
% simulateously for each company respectively.
Aeq = [eye(x) eye(x)];

% "beq" is a matrix whose elements represent the number of orders from
each
% company respectively.
beq = zeros(x,1);
for i = 1:x
    beq(i) = order(i);
end
```

Solve the linear programming with integer decision variables.

Tell Matlab that variables 1 to 6 are integers

```
intcon = 1:6 ;

[xx,ffval] = intlinprog(f,intcon,A,b, Aeq, beq )
```

Output:

```
LP:                Optimal objective value is 22914.000000.


Optimal solution found.

Intlinprog stopped at the root node because the objective value is within
a gap
tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (the
default
value). The intcon variables are integer within tolerance,
options.IntegerTolerance = 1e-05 (the default value).


xx =

     0
   158
   750
   390
   610
     0


ffval =

       22914
```

Conclusion

Hence, the production for each factory is as follows:

	China Factory Production (Units)	Korea Factory Production (Units)
Company 1	0	390
Company 2	158	610
Company 3	750	0