Indian Institute of Space Science and Technology

Thiruvananthapuram

Ambuj Nayan
SC20B007

Optimization Assignment 1

October 1, 2023

Department of Aerospace Engineering

It is a Non Linear Problem

```
In [ ]: using JuMP
        import Ipopt
```

```
In [ ]: model = Model(Ipopt.Optimizer) # Using Non Linear solver
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
```

```
In [ ]: @variable(model, x, lower_bound = 0, upper_bound = 80) # Fixing minimum and maxi
```

$$x$$

```
In [ ]: h = 50
```

```
50
```

```
In [ ]: g = 9.81
```

```
9.81
```

```
In [ ]: v = 90
```

```
90
```

Time to reach peak height :

$$t_1 = \frac{v * sin(\theta)}{g}$$

```
In [ ]: t1 = @NLexpression(model, v * sind(x) / g)
```

subexpression[1]: (90.0 * sind(x)) / 9.81

Time to reach ground from peak height:

$$t_2 = \sqrt{\frac{2H}{g}}$$

where H is the peak height

$$H = h + \frac{v^2 * sin^2(\theta)}{2g}$$

So

$$t_2 = \sqrt{\frac{2h}{g} + (\frac{v * sin(\theta)}{g})^2}$$

In [ ]:
```
t2 = @NLexpression(model, sqrt(2 * h / g + (v * sind(x) / g)^2))
```

```
subexpression[2]: sqrt((2.0 * 50.0) / 9.81 + ((90.0 * sind(x)) / 9.81) ^ {2.0})
```

Thus total time will be:

$$t_1 + t_2$$

In [ ]:
```
total_time = @NLexpression(model, t1 + t2)
```

```
subexpression[3]: subexpression_{1} + subexpression_{2}
```

In [ ]:
```
total_range = @NLexpression(model, total_time * v * cosd(x))
```

```
subexpression[4]: subexpression_{3} * 90.0 * cosd(x)
```

In [ ]:
```
@NLobjective(model, Max, total_range)
```

In [ ]:
```
@show model
```

```
model = A JuMP Model
Maximization problem with:
Variable: 1
Objective function type: Nonlinear
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 1 constraint
`VariableRef`-in-`MathOptInterface.LessThan{Float64}`: 1 constraint
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
Names registered in the model: x
A JuMP Model
Maximization problem with:
Variable: 1
Objective function type: Nonlinear
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 1 constraint
`VariableRef`-in-`MathOptInterface.LessThan{Float64}`: 1 constraint
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
Names registered in the model: x
```

In [ ]:
```
optimize!(model)
```

```
This is Ipopt version 3.14.13, running with linear solver MUMPS 5.6.1.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        1

Total number of variables............................:        1
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        1
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  2.8749203e+02 0.00e+00 1.44e+01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.9105335e+02 0.00e+00 1.36e+01  -1.0 2.46e-01    -  6.36e-02 1.00e+00f  1
   2  8.4789227e+02 0.00e+00 5.98e+00  -1.0 3.56e+01   0.0 9.79e-03 1.00e+00f  1
   3  8.7423639e+02 0.00e+00 3.72e-01  -1.0 7.74e+00    -  1.00e+00 1.00e+00f  1
   4  8.7425946e+02 0.00e+00 1.17e-04  -1.0 2.19e-01    -  1.00e+00 1.00e+00f  1
   5  8.7425946e+02 0.00e+00 1.33e-09  -2.5 7.21e-04    -  1.00e+00 1.00e+00f  1
   6  8.7425946e+02 0.00e+00 3.57e-13  -3.8 1.18e-05    -  1.00e+00 1.00e+00f  1
   7  8.7425946e+02 0.00e+00 9.95e-16  -5.7 6.60e-07    -  1.00e+00 1.00e+00f  1
   8  8.7425946e+02 0.00e+00 2.45e-15  -8.6 8.18e-09    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  -8.7425945913405087e+02    8.7425945913405087e+02
Dual infeasibility......:   2.4455532738497097e-15    2.4455532738497097e-15
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5059039712143006e-09    2.5059039712143006e-09
Overall NLP error.......:   2.5059039712143006e-09    2.5059039712143006e-09


Number of objective function evaluations             = 9
Number of objective gradient evaluations             = 9
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 8
Total seconds in IPOPT                               = 0.004

EXIT: Optimal Solution Found.
```

In [ ]:

```
@show value.(x) # Gives out value of x for which range is maximum.
```

```
value.(x) = 43.363373916696226
43.363373916696226
```

In [ ]:

```
@show objective_value(model) # Gives out maximum range.
```

```
objective_value(model) = 874.2594591340509
874.2594591340509
```

2

October 2, 2023

[ ]:

```
using JuMP
using CPLEX
```

[ ]:

```
m = 10
```

```
10
```

[ ]:

```
n = 30
```

```
30
```

[ ]:

```
p = 20
```

```
20
```

[ ]:

```
Q = rand(10:50, m, n)
```

```
10×30 Matrix{Int64}:
 44  11  40  46  33  40  43  30  32  …  49  22  50  40  36  30  48  37  12
 37  36  27  38  30  27  39  15  24     12  10  16  19  27  22  47  45  11
 49  21  15  45  14  39  35  45  14     12  12  49  48  48  44  25  22  17
 18  34  39  48  11  27  23  30  43     46  47  25  22  47  40  44  39  37
 13  32  27  14  12  30  40  27  25     41  39  13  38  30  34  41  30  30
 32  48  31  32  22  39  40  21  19  …  50  26  33  30  15  49  35  24  49
 48  35  26  44  21  35  27  46  46     12  32  29  41  40  38  34  36  46
 20  26  11  45  13  11  49  35  12     11  16  28  33  23  30  35  24  34
 36  19  41  45  44  23  46  15  34     43  15  10  26  41  15  16  44  32
 17  27  46  45  21  16  42  32  10     17  21  34  23  10  32  34  41  45
```

[ ]:

```
C = rand(60:100, p, n)
```

```
20×30 Matrix{Int64}:
 63  100  83  76  64   93  100  75  75  …  90   67   64   74  96  87   86  87
 81   71  82  71  89   80   62  94  60     61   69   70   95  97  91   62  82
 80   81  89  96  80   97   68  62  92     88   61   95   86  63  68   95  67
 94   74  99  69  73   96   78  77  82     75  100   87   63  97  71   76  96
 68   88  97  75  68   81   61  95  80     72   80  100   68  91  66   65  68
 63   68  73  72  64   87  100  65  82  …  83   77   75   72  89  63   72  71
 97   81  68  75  61   62   69  92  84     74   61   99   85  93  68   70  63
 92   61  97  74  91   97   78  80  62     87   97   89   61  94  86   95  64
 97   64  71  83  78   73   64  91  62     70   92   89   96  95  76   74  84
 97   74  80  66  98   77   62  69  94     92   98   83   95  98  74   62  63
 70   88  99  93  79   85   88  93  93  …  98   91   82   91  85  65  100  96
 62   88  90  80  68   86   98  72  88     78   74   97   95  60  73   96  85
 60   65  80  63  92   82   97  98  99     69   71   80   78  78  99  100  98
 75   63  65  99  88   99   75  86  87     81   96   73   91  71  90   92  94
 88   66  98  72  75   85   70  75  88     94   95   76   61  82  96   86  95
 65   87  71  94  95   91   86  89  91  …  83   98   86   99  95  92   67  85
 76   88  63  62  98   60   69  92  83     61   99   67  100  89  77   61  96
 61   90  88  87  69   99   69  85  85     65   79   95   64  73  98   92  63
 76   80  65  75  74  100   78  78  79     74   95   90   88  63  80   86  97
 83   81  98  94  65   74   83  87  69     71  100   65   78  99  71   69  63
```

```
[ ]: model = Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

```
[ ]: @variable(model, x[i=1:m, j =1:p], lower_bound = 0, Bin) # Binary variable
```

```
10×20 Matrix{VariableRef}:
 x[1,1]   x[1,2]   x[1,3]   x[1,4]   x[1,5]   …  x[1,18]   x[1,19]   x[1,20]
 x[2,1]   x[2,2]   x[2,3]   x[2,4]   x[2,5]      x[2,18]   x[2,19]   x[2,20]
 x[3,1]   x[3,2]   x[3,3]   x[3,4]   x[3,5]      x[3,18]   x[3,19]   x[3,20]
 x[4,1]   x[4,2]   x[4,3]   x[4,4]   x[4,5]      x[4,18]   x[4,19]   x[4,20]
 x[5,1]   x[5,2]   x[5,3]   x[5,4]   x[5,5]      x[5,18]   x[5,19]   x[5,20]
 x[6,1]   x[6,2]   x[6,3]   x[6,4]   x[6,5]   …  x[6,18]   x[6,19]   x[6,20]
 x[7,1]   x[7,2]   x[7,3]   x[7,4]   x[7,5]      x[7,18]   x[7,19]   x[7,20]
 x[8,1]   x[8,2]   x[8,3]   x[8,4]   x[8,5]      x[8,18]   x[8,19]   x[8,20]
 x[9,1]   x[9,2]   x[9,3]   x[9,4]   x[9,5]      x[9,18]   x[9,19]   x[9,20]
 x[10,1]  x[10,2]  x[10,3]  x[10,4]  x[10,5]     x[10,18]  x[10,19]  x[10,20]
```

```
[ ]: @constraint(model, sum(x[:, j] for j in 1:p) .== 1) # Ensuring 10 plants be␣
      ↪established
```

```
10-element Vector{ConstraintRef{Model, MathOptInterface.
 ↪ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},␣
 ↪MathOptInterface.EqualTo{Float64}}, ScalarShape}}:
 x[1,1] + x[1,2] + x[1,3] + x[1,4] + x[1,5] + x[1,6] + x[1,7] + x[1,8] + x[1,9]␣
 ↪+ x[1,10] + x[1,11] + x[1,12] + x[1,13] + x[1,14] + x[1,15] + x[1,16] +␣
 ↪x[1,17] + x[1,18] + x[1,19] + x[1,20] == 1
 x[2,1] + x[2,2] + x[2,3] + x[2,4] + x[2,5] + x[2,6] + x[2,7] + x[2,8] + x[2,9]␣
 ↪+ x[2,10] + x[2,11] + x[2,12] + x[2,13] + x[2,14] + x[2,15] + x[2,16] +␣
 ↪x[2,17] + x[2,18] + x[2,19] + x[2,20] == 1
 x[3,1] + x[3,2] + x[3,3] + x[3,4] + x[3,5] + x[3,6] + x[3,7] + x[3,8] + x[3,9]␣
 ↪+ x[3,10] + x[3,11] + x[3,12] + x[3,13] + x[3,14] + x[3,15] + x[3,16] +␣
 ↪x[3,17] + x[3,18] + x[3,19] + x[3,20] == 1
 x[4,1] + x[4,2] + x[4,3] + x[4,4] + x[4,5] + x[4,6] + x[4,7] + x[4,8] + x[4,9]␣
 ↪+ x[4,10] + x[4,11] + x[4,12] + x[4,13] + x[4,14] + x[4,15] + x[4,16] +␣
 ↪x[4,17] + x[4,18] + x[4,19] + x[4,20] == 1
 x[5,1] + x[5,2] + x[5,3] + x[5,4] + x[5,5] + x[5,6] + x[5,7] + x[5,8] + x[5,9]␣
 ↪+ x[5,10] + x[5,11] + x[5,12] + x[5,13] + x[5,14] + x[5,15] + x[5,16] +␣
 ↪x[5,17] + x[5,18] + x[5,19] + x[5,20] == 1
 x[6,1] + x[6,2] + x[6,3] + x[6,4] + x[6,5] + x[6,6] + x[6,7] + x[6,8] + x[6,9]␣
 ↪+ x[6,10] + x[6,11] + x[6,12] + x[6,13] + x[6,14] + x[6,15] + x[6,16] +␣
 ↪x[6,17] + x[6,18] + x[6,19] + x[6,20] == 1
 x[7,1] + x[7,2] + x[7,3] + x[7,4] + x[7,5] + x[7,6] + x[7,7] + x[7,8] + x[7,9]␣
 ↪+ x[7,10] + x[7,11] + x[7,12] + x[7,13] + x[7,14] + x[7,15] + x[7,16] +␣
 ↪x[7,17] + x[7,18] + x[7,19] + x[7,20] == 1
 x[8,1] + x[8,2] + x[8,3] + x[8,4] + x[8,5] + x[8,6] + x[8,7] + x[8,8] + x[8,9]␣
 ↪+ x[8,10] + x[8,11] + x[8,12] + x[8,13] + x[8,14] + x[8,15] + x[8,16] +␣
 ↪x[8,17] + x[8,18] + x[8,19] + x[8,20] == 1
 x[9,1] + x[9,2] + x[9,3] + x[9,4] + x[9,5] + x[9,6] + x[9,7] + x[9,8] + x[9,9]␣
 ↪+ x[9,10] + x[9,11] + x[9,12] + x[9,13] + x[9,14] + x[9,15] + x[9,16] +␣
 ↪x[9,17] + x[9,18] + x[9,19] + x[9,20] == 1
 x[10,1] + x[10,2] + x[10,3] + x[10,4] + x[10,5] + x[10,6] + x[10,7] + x[10,8] +␣
 ↪x[10,9] + x[10,10] + x[10,11] + x[10,12] + x[10,13] + x[10,14] + x[10,15] +␣
 ↪x[10,16] + x[10,17] + x[10,18] + x[10,19] + x[10,20] == 1
```

```
[ ]: @constraint(model, sum(x[i, :] for i in 1:m) .<= 1) # One place can't have more␣
      ↪than 1 plant
```

```
20-element Vector{ConstraintRef{Model, MathOptInterface.
 ↪ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},␣
 ↪MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 x[1,1] + x[2,1] + x[3,1] + x[4,1] + x[5,1] + x[6,1] + x[7,1] + x[8,1] + x[9,1]␣
 ↪+ x[10,1] <= 1
 x[1,2] + x[2,2] + x[3,2] + x[4,2] + x[5,2] + x[6,2] + x[7,2] + x[8,2] + x[9,2]␣
 ↪+ x[10,2] <= 1
 x[1,3] + x[2,3] + x[3,3] + x[4,3] + x[5,3] + x[6,3] + x[7,3] + x[8,3] + x[9,3]␣
 ↪+ x[10,3] <= 1
 x[1,4] + x[2,4] + x[3,4] + x[4,4] + x[5,4] + x[6,4] + x[7,4] + x[8,4] + x[9,4]␣
 ↪+ x[10,4] <= 1
 x[1,5] + x[2,5] + x[3,5] + x[4,5] + x[5,5] + x[6,5] + x[7,5] + x[8,5] + x[9,5]␣
 ↪+ x[10,5] <= 1
 x[1,6] + x[2,6] + x[3,6] + x[4,6] + x[5,6] + x[6,6] + x[7,6] + x[8,6] + x[9,6]␣
 ↪+ x[10,6] <= 1
 x[1,7] + x[2,7] + x[3,7] + x[4,7] + x[5,7] + x[6,7] + x[7,7] + x[8,7] + x[9,7]␣
 ↪+ x[10,7] <= 1
 x[1,8] + x[2,8] + x[3,8] + x[4,8] + x[5,8] + x[6,8] + x[7,8] + x[8,8] + x[9,8]␣
 ↪+ x[10,8] <= 1
 x[1,9] + x[2,9] + x[3,9] + x[4,9] + x[5,9] + x[6,9] + x[7,9] + x[8,9] + x[9,9]␣
 ↪+ x[10,9] <= 1
 x[1,10] + x[2,10] + x[3,10] + x[4,10] + x[5,10] + x[6,10] + x[7,10] + x[8,10] +␣
 ↪x[9,10] + x[10,10] <= 1
 x[1,11] + x[2,11] + x[3,11] + x[4,11] + x[5,11] + x[6,11] + x[7,11] + x[8,11] +␣
 ↪x[9,11] + x[10,11] <= 1
 x[1,12] + x[2,12] + x[3,12] + x[4,12] + x[5,12] + x[6,12] + x[7,12] + x[8,12] +␣
 ↪x[9,12] + x[10,12] <= 1
 x[1,13] + x[2,13] + x[3,13] + x[4,13] + x[5,13] + x[6,13] + x[7,13] + x[8,13] +␣
 ↪x[9,13] + x[10,13] <= 1
 x[1,14] + x[2,14] + x[3,14] + x[4,14] + x[5,14] + x[6,14] + x[7,14] + x[8,14] +␣
 ↪x[9,14] + x[10,14] <= 1
 x[1,15] + x[2,15] + x[3,15] + x[4,15] + x[5,15] + x[6,15] + x[7,15] + x[8,15] +␣
 ↪x[9,15] + x[10,15] <= 1
 x[1,16] + x[2,16] + x[3,16] + x[4,16] + x[5,16] + x[6,16] + x[7,16] + x[8,16] +␣
 ↪x[9,16] + x[10,16] <= 1
 x[1,17] + x[2,17] + x[3,17] + x[4,17] + x[5,17] + x[6,17] + x[7,17] + x[8,17] +␣
 ↪x[9,17] + x[10,17] <= 1
 x[1,18] + x[2,18] + x[3,18] + x[4,18] + x[5,18] + x[6,18] + x[7,18] + x[8,18] +␣
 ↪x[9,18] + x[10,18] <= 1
 x[1,19] + x[2,19] + x[3,19] + x[4,19] + x[5,19] + x[6,19] + x[7,19] + x[8,19] +␣
 ↪x[9,19] + x[10,19] <= 1
 x[1,20] + x[2,20] + x[3,20] + x[4,20] + x[5,20] + x[6,20] + x[7,20] + x[8,20] +␣
 ↪x[9,20] + x[10,20] <= 1
```

```
[ ]: cost = sum(x[i, k] * C[k, j] * Q[i, j] for i in 1:m for j in 1:n for k in 1:p)
```

$$79025x_{1,1}+75279x_{1,2}+79614x_{1,3}+80530x_{1,4}+76856x_{1,5}+74016x_{1,6}+75501x_{1,7}+78629x_{1,8}+78341x_{1,9}+78042x_{1,10}+$$

```
[ ]: @objective(model, Min, cost)
```

$$79025x_{1,1}+75279x_{1,2}+79614x_{1,3}+80530x_{1,4}+76856x_{1,5}+74016x_{1,6}+75501x_{1,7}+78629x_{1,8}+78341x_{1,9}+78042x_{1,10}+$$

```
[ ]: optimize!(model)
```

```
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Found incumbent of value 0.000000 after 0.02 sec. (0.01 ticks)

Root node processing (before b&c):
  Real time             =    0.02 sec. (0.01 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
```

```
                             ------------
Total (root+branch&cut) =    0.02 sec. (0.01 ticks)
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Found incumbent of value 716973.000000 after 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Reduced MIP has 30 rows, 200 columns, and 400 nonzeros.
Reduced MIP has 200 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.03 sec. (0.23 ticks)
Probing time = 0.00 sec. (0.27 ticks)
Tried aggregator 1 time.
Reduced MIP has 30 rows, 200 columns, and 400 nonzeros.
Reduced MIP has 200 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.23 ticks)
Probing time = 0.00 sec. (0.27 ticks)
Clique table members: 30.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.02 sec. (0.16 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                        716973.0000        0.0000           100.00%
*     0+    0                        706224.0000        0.0000           100.00%
*     0     0      integral     0    698289.0000   698289.0000       23    0.00%
Elapsed time = 0.08 sec. (1.46 ticks, tree = 0.00 MB, solutions = 3)
```

[ ]:
```
@show value.(x)
```

```
value.(x) = [-0.0 -0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 1.0 -0.0 -0.0 -0.0; -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 -0.0 1.0 -0.0; -0.0 -0.0 -0.0 -0.0 1.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0; -0.0 1.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0; -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 0.0 -0.0 1.0; -0.0 0.0 -0.0 -0.0 -0.0 -0.0 1.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -0.0; -0.0 -0.0 -0.0 -0.0 0.0 1.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0; -0.0 -0.0
-0.0 -0.0 -0.0 -0.0 -0.0 1.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0; -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
-0.0 -0.0 -0.0 1.0 -0.0 -0.0; -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 1.0
-0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 -0.0]

10×20 Matrix{Float64}:
 -0.0  -0.0  -0.0  -0.0  -0.0   0.0  …  -0.0  -0.0   1.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0   0.0  -0.0   1.0  -0.0
 -0.0  -0.0  -0.0  -0.0   1.0  -0.0     -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0   1.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0  -0.0   0.0  -0.0   1.0
 -0.0   0.0  -0.0  -0.0  -0.0  -0.0  …  -0.0  -0.0   0.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0   0.0   1.0     -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0  -0.0   1.0  -0.0  -0.0
 -0.0  -0.0  -0.0  -0.0  -0.0  -0.0     -0.0  -0.0  -0.0  -0.0   0.0  -0.0
```

```
[ ]: @show objective_value(model)
```

```
objective_value(model) = 698289.0
```

```
698289.0
```

3

October 1, 2023

```
[ ]: using JuMP
```

```
[ ]: using CPLEX
```

```
[ ]: model=Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

Our Decision Variable would be amount of fruits supplied by growers to each plant. For example $x_{1,2}$ denotes fruits supplied by first grower to plant B.

```
[ ]: @variable(model, x[i=1:3,j=1:2],lower_bound=0) # Fixing minimum value of the␣
       ↪decision variable to 0
```

```
3×2 Matrix{VariableRef}:
 x[1,1]  x[1,2]
 x[2,1]  x[2,2]
 x[3,1]  x[3,2]
```

```
[ ]: @constraint(model, sum(x[1,j] for j=1:2) == 200)
```

$$x_{1,1} + x_{1,2} = 200$$

```
[ ]: @constraint(model, sum(x[2,j] for j=1:2) == 310)
```

$$x_{2,1} + x_{2,2} = 310$$

```
[ ]: @constraint(model, sum(x[3,j] for j=1:2) == 420)
```

$$x_{3,1} + x_{3,2} = 420$$

```
[ ]: @constraint(model, sum(x[i,1] for i=1:3) <= 460)
```

$$x_{1,1} + x_{2,1} + x_{3,1} \leq 460$$

```
[ ]: @constraint(model, sum(x[i,2] for i=1:3) <= 560)
```

$$x_{1,2} + x_{2,2} + x_{3,2} \leq 560$$

```
[ ]: BC=[1100,1000,900]
```

```
3-element Vector{Int64}:
 1100
 1000
  900
```

```
[ ]: buying_cost=sum(BC[i] * sum(x[i,j] for j=1:2) for i=1:3)
```

$$1100x_{1,1} + 1100x_{1,2} + 1000x_{2,1} + 1000x_{2,2} + 900x_{3,1} + 900x_{3,2}$$

```
[ ]: SC=[[3000,3500],[2000,2500],[6000,4000]]
```

```
3-element Vector{Vector{Int64}}:
 [3000, 3500]
 [2000, 2500]
 [6000, 4000]
```

```
[ ]: shipping_cost= sum(sum(SC[i][j] * x[i, j] for j = 1:2) for i = 1:3)
```

$$3000x_{1,1} + 3500x_{1,2} + 2000x_{2,1} + 2500x_{2,2} + 6000x_{3,1} + 4000x_{3,2}$$

```
[ ]: canning_cost = 26000*sum(x[i,1] for i=1:3) + 21000 * sum(x[i,2] for i=1:3)
```

$$26000x_{1,1} + 26000x_{2,1} + 26000x_{3,1} + 21000x_{1,2} + 21000x_{2,2} + 21000x_{3,2}$$

```
[ ]: selling_price = 50000 * sum(sum(x[i, j] for j = 1:2) for i = 1:3)
```

$$50000x_{1,1} + 50000x_{1,2} + 50000x_{2,1} + 50000x_{2,2} + 50000x_{3,1} + 50000x_{3,2}$$

```
[ ]: profit=selling_price-buying_cost-shipping_cost-canning_cost
```

$$19900x_{1,1} + 24400x_{1,2} + 21000x_{2,1} + 25500x_{2,2} + 17100x_{3,1} + 24100x_{3,2}$$

```
[ ]: @objective(model,Max,profit)
```

$$19900x_{1,1} + 24400x_{1,2} + 21000x_{2,1} + 25500x_{2,2} + 17100x_{3,1} + 24100x_{3,2}$$

```
[ ]: @show model
```

```
A JuMP Model
Maximization problem with:
Variables: 6
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 3 constraints
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 6 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: x
```

```
model = A JuMP Model
Maximization problem with:
Variables: 6
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 3 constraints
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 6 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: x
```

```
[ ]: optimize!(model)
```

```
CPLEX Error  3003: Not a mixed-integer problem.
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 3 columns.
Aggregator did 3 substitutions.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.00 ticks)
```

```
[ ]: @show value.(x) # Most optimal distibution
```

```
value.(x) = [60.0 140.0; 310.0 0.0; 0.0 420.0]
```

```
3×2 Matrix{Float64}:
  60.0  140.0
 310.0    0.0
   0.0  420.0
```

```
[ ]: @show objective_value(model)
```

```
objective_value(model) = 2.1242e7
```

```
2.1242e7
```

4

October 1, 2023

```
[ ]: using JuMP
```

```
[ ]: using CPLEX
```

```
[ ]: model=Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

```
[ ]: n = 2
```

```
2
```

```
[ ]: K = 6
```

```
6
```

x_1 is amount of Acid A produced and x_2 is amount of acid B produced.

```
[ ]: @variable(model, x[1:2],lower_bound=0)
```

```
2-element Vector{VariableRef}:
 x[1]
 x[2]
```

```
[ ]: Time=[3 4; 3 2]
```

```
2×2 Matrix{Int64}:
 3  4
 3  2
```

1 Operation 1

```
[ ]: @constraint(model, sum(x.*Time[:,1]) <= 20)
```

$$3x_1 + 3x_2 \leq 20$$

2 Operation 2

[ ]:
```julia
@constraint(model, sum(x.*Time[:,2]) <= 18)
```

$$4x_1 + 2x_2 \leq 18$$

[ ]:
```julia
@variable(model, c_sold,lower_bound=0)
```

$$c\_sold$$

[ ]:
```julia
@variable(model, c_destroyed,lower_bound=0)
```

$$c\_destroyed$$

[ ]:
```julia
@constraint(model, c_sold <= K) # Limiting amount of C sold
```

$$c\_sold \leq 6$$

$c_{sold} + c_{destroyed} = c_{produced} = n * x_2$

[ ]:
```julia
@constraint(model, c_sold + c_destroyed == n*x[2])
```

$$-2x_2 + c\_sold + c\_destroyed = 0$$

[ ]:
```julia
P=[80 60 20 15]
```

```
1×4 Matrix{Int64}:
 80  60  20  15
```

[ ]:
```julia
total_profit = sum(P[1:2].*x) + P[3]*c_sold - P[4] * c_destroyed
```

$$80x_1 + 60x_2 + 20c\_sold - 15c\_destroyed$$

[ ]:
```julia
@objective(model,Max,total_profit)
```

$$80x_1 + 60x_2 + 20c\_sold - 15c\_destroyed$$

[ ]:
```julia
@show model
```

```
model = A JuMP Model
Maximization problem with:
Variables: 4
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 1 constraint
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 3 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 4 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: c_destroyed, c_sold, x
```

```
A JuMP Model
Maximization problem with:
Variables: 4
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 1 constraint
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 3 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 4 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: c_destroyed, c_sold, x
```

[ ]:

```
optimize!(model)
```

```
CPLEX Error  3003: Not a mixed-integer problem.
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 0 columns.
Aggregator did 1 substitutions.
Reduced LP has 2 rows, 3 columns, and 6 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)

Iteration log . . .
Iteration:     1   Dual infeasibility =             0.000000
```

Amount of A and B produced

[ ]:

```
@show value.(x)
```

```
value.(x) = [3.0, 3.0]
```

```
2-element Vector{Float64}:
 3.0
 3.0
```

Amount of C produced

[ ]:

```
@show value.(n*x[2])
```

```
value.(n * x[2]) = 6.0
```

```
6.0
```

Amount of C sold

```
[ ]: @show value.(c_sold)
```

```
value.(c_sold) = 6.0
```

```
6.0
```

Total Profit

```
[ ]: @show objective_value(model)
```

```
objective_value(model) = 540.0
```

```
540.0
```

5

October 1, 2023

[ ]:
```
using JuMP
```

[ ]:
```
using CPLEX
```

[ ]:
```
model=Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

[ ]:
```
d = 500
```

```
500
```

[ ]:
```
a = [6 5 4] # Walking speed of students
```

```
1×3 Matrix{Int64}:
 6  5  4
```

[ ]:
```
b = [14 15 16]  # Biking speed of students
```

```
1×3 Matrix{Int64}:
 14  15  16
```

[ ]:
```
@variable(model, x[1:3],lower_bound=0) # Distance for which each student will␣
 ↪use bicycle
```

```
3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
```

[ ]:
```
@constraint(model, sum(x) == d)
```

$$x_1 + x_2 + x_3 = 500$$

[ ]:
```
t= [(x[i] / b[i] + (d-x[i])/a[i]) for i in 1:3] # Time array is defined
```

```
3-element Vector{AffExpr}:
 -0.09523809523809523 x[1] + 83.33333333333333
 -0.13333333333333336 x[2] + 100
 -0.1875 x[3] + 125
```

[ ]:

```julia
# Define a variable to represent the maximum of t
@variable(model, max_t)
```

$$max\_t$$

[ ]:

```julia
# Add constraints to ensure that max_t is greater than or equal to all t values
for i in 1:3
    @constraint(model, max_t >= t[i])    # max_t will be greater than or equal to than the time taken by the last student.
end
```

[ ]:

```julia
@objective(model,Min, max_t)
```

$$max\_t$$

[ ]:

```julia
@show model
```

```
model = A JuMP Model
Minimization problem with:
Variables: 4
Objective function type: VariableRef
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 1 constraint
`AffExpr`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: max_t, x
```

```
A JuMP Model
Minimization problem with:
Variables: 4
Objective function type: VariableRef
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 1 constraint
`AffExpr`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: max_t, x
```

[ ]:

```julia
optimize!(model)
```

```
CPLEX Error  3003: Not a mixed-integer problem.
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time = 0.00 sec. (0.00 ticks)
Initializing dual steep norms . . .

Iteration log . . .
Iteration:     1   Scaled dual infeas =             0.133332
Iteration:     3   Dual objective     =            71.428571
```

[ ]:
```julia
@show value.(x)
```

```
value.(x) = [68.75000000000011, 174.10714285714272, 257.14285714285717]
```

```
3-element Vector{Float64}:
  68.75000000000011
 174.10714285714272
 257.14285714285717
```

[ ]:
```julia
@show objective_value(model) # Minimum time
```

```
objective_value(model) = 76.78571428571428
```

```
76.78571428571428
```

6

October 1, 2023

1 It is a Non Linear Problem

[ ]:
```
using JuMP
import Ipopt
```

[ ]:
```
model = Model(Ipopt.Optimizer) # Using Non Linear solver
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
```

[ ]:
```
@variable(model, l, lower_bound = 0)
```

$$l$$

[ ]:
```
@variable(model, b, lower_bound = 0)
```

$$b$$

[ ]:
```
@variable(model, h, lower_bound = 0)
```

$$h$$

[ ]:
```
corner_waste_cost = 4 * h^2
```

$$4h^2$$

[ ]:
```
welding_cost = 2 * h
```

$$2h$$

[ ]:
```
@constraint(model, l + 2 * h == 22)
```

$$l + 2h = 22$$

[ ]:

```julia
@constraint(model, b + 2 * h == 17)
```

$$b + 2h = 17$$

[ ]:

```julia
total_profit = @NLexpression(model, 8* l * b * h - corner_waste_cost - welding_cost)
```

subexpression[1]: (8.0 * l * b * h - h * h * 4.0) - 2.0 * h

[ ]:

```julia
@NLobjective(model, Max, total_profit)
```

[ ]:

```julia
@show model
```

```
model = A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: Nonlinear
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
Names registered in the model: b, h, l
```

```
A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: Nonlinear
`AffExpr`-in-`MathOptInterface.EqualTo{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Ipopt
Names registered in the model: b, h, l
```

[ ]:

```julia
optimize!(model)
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

This is Ipopt version 3.14.13, running with linear solver MUMPS 5.6.1.
```

```
Number of nonzeros in equality constraint Jacobian…:        4
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian…:        6

Total number of variables…:        3
                     variables with only lower bounds:        3
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints…:        2
Total number of inequality constraints…:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -2.0391979e-02 2.20e+01 1.13e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -4.1810080e+00 1.99e+01 7.69e+01  -1.0 8.54e+00    -  1.17e-03 9.28e-02h  1
   2 -2.6539272e+02 3.55e-15 3.76e+02  -1.0 7.64e+00    -  1.09e-02 1.00e+00h  1
   3  1.1296492e+03 3.55e-15 2.55e+02  -1.0 2.89e+00   2.0 6.54e-01 1.00e+00f  1
   4  3.5562631e+03 0.00e+00 6.87e+02  -1.0 1.04e+01   1.5 2.26e-01 1.00e+00f  1
   5  4.1765848e+03 0.00e+00 4.06e+01  -1.0 2.25e+00    -  1.00e+00 1.00e+00f  1
   6  4.1874980e+03 0.00e+00 9.96e-01  -1.0 3.53e-01    -  1.00e+00 1.00e+00f  1
   7  4.1875048e+03 0.00e+00 6.64e-04  -1.7 9.11e-03    -  1.00e+00 1.00e+00f  1
   8  4.1875048e+03 0.00e+00 1.78e-09  -3.8 5.85e-06    -  1.00e+00 1.00e+00f  1
   9  4.1875048e+03 0.00e+00 1.85e-11  -5.7 2.88e-09    -  1.00e+00 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  4.1875048e+03 0.00e+00 3.53e-13  -8.6 3.61e-11    -  1.00e+00 1.00e+00f  1

Number of Iterations…: 10

                                   (scaled)                 (unscaled)
Objective…:  -4.1875047654921573e+03    4.1875047654921573e+03
Dual infeasibility…:   3.5295945402507846e-13    3.5295945402507846e-13
Constraint violation…:   0.0000000000000000e+00    0.0000000000000000e+00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity…:   2.5059035703024915e-09    2.5059035703024915e-09
Overall NLP error…:   2.5059035703024915e-09    2.5059035703024915e-09


Number of objective function evaluations             = 11
Number of objective gradient evaluations             = 11
Number of equality constraint evaluations            = 11
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 1
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 10
Total seconds in IPOPT                               = 1.358
```

```
EXIT: Optimal Solution Found.
```

[ ]:
```julia
@show value.(l)
```

```
value.(l) = 15.742373793241692
```

```
15.742373793241692
```

[ ]:
```julia
@show value.(b)
```

```
value.(b) = 10.742373793241692
```

```
10.742373793241692
```

[ ]:
```julia
@show value.(h)
```

```
value.(h) = 3.1288131033791533
```

```
3.1288131033791533
```

[ ]:
```julia
@show objective_value(model) # Max Profit
```

```
objective_value(model) = 4187.504765492157
```

```
4187.504765492157
```

7

October 1, 2023

[ ]:

```
using JuMP
```

[ ]:

```
using CPLEX
```

[ ]:

```
model=Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

[ ]:

```
@variable(model, x,lower_bound=0)
```

$$x$$

[ ]:

```
@variable(model, y,lower_bound=0)
```

$$y$$

[ ]:

```
@variable(model, z,lower_bound=0)
```

$$z$$

[ ]:

```
@constraint(model, x+y+z <= 20)
```

$$x + y + z \leq 20$$

[ ]:

```
@constraint(model, 10*x+12*y+8*z<=2000)
```

$$10x + 12y + 8z \leq 2000$$

[ ]:

```
fertilizer_cost=10*(200*x+300*y+100*z)
```

$$2000x + 3000y + 1000z$$

[ ]:
```
labour_cost= 40*(10*x+12*y+8*z)
```

$$400x + 480y + 320z$$

[ ]:
```
selling_price = 20*400*x+15*600*y+25*200*z
```

$$8000x + 9000y + 5000z$$

[ ]:
```
profit=selling_price-fertilizer_cost-labour_cost # Defining profit
```

$$5600x + 5520y + 3680z$$

[ ]:
```
@objective(model,Max,profit)
```

$$5600x + 5520y + 3680z$$

[ ]:
```
@show model
```

```
model = A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: x, y, z

A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: AffExpr
`AffExpr`-in-`MathOptInterface.LessThan{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
Names registered in the model: x, y, z
```

[ ]:
```
optimize!(model)
```

```
CPLEX Error  3003: Not a mixed-integer problem.
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 3 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.00 ticks)
```

[ ]:

```julia
@show value(x)
```

```
value(x) = 20.0
```

```
20.0
```

[ ]:

```julia
@show value(y)
```

```
value(y) = 0.0
```

```
0.0
```

[ ]:

```julia
@show value(z)
```

```
value(z) = 0.0
```

```
0.0
```

[ ]:

```julia
@show objective_value(model) # Maximum profit
```

```
objective_value(model) = 112000.0
```

```
112000.0
```

8

October 1, 2023

```
[ ]: using JuMP
     using CPLEX
```

```
[ ]: model = Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

```
[ ]: demand = [1500, 2100, 1800, 1950]
```

```
4-element Vector{Int64}:
 1500
 2100
 1800
 1950
```

```
[ ]: regular_hrs = 480
     max_overtime = 80
     production_rate = 160
     max_subcontract = 500
     regular_pay = 50
     overtime_pay = 55
     subcontracting_cost = 9000
     backlog_cost = 1200
     inventory_holding_cost = 50
     hiring_cost = 1000
     firing_cost = 1200
```

```
1200
```

```
[ ]: num_init_workers = 600
     inventory_start = 200
     inventory_end = 300
```

```
300
```

```
[ ]: @variable(model, total_overtime_hrs[1:4], lower_bound = 0, Int)
```

```
4-element Vector{VariableRef}:
 total_overtime_hrs[1]
 total_overtime_hrs[2]
 total_overtime_hrs[3]
 total_overtime_hrs[4]
```

```
[ ]: @variable(model, w[1:4], lower_bound = 0, Int) # Number of workers in each quarter
```

```
4-element Vector{VariableRef}:
 w[1]
 w[2]
 w[3]
 w[4]
```

```
[ ]: @variable(model, w_h[1:4], lower_bound = 0, Int) # Number of workers being hired in each quarter
```

```
4-element Vector{VariableRef}:
 w_h[1]
 w_h[2]
 w_h[3]
 w_h[4]
```

```
[ ]: @variable(model, w_f[1:4], lower_bound = 0, Int) # Number of workers being fired in each quarter
```

```
4-element Vector{VariableRef}:
 w_f[1]
 w_f[2]
 w_f[3]
 w_f[4]
```

```
[ ]: @variable(model, production[1:4], lower_bound = 0, Int)
```

```
4-element Vector{VariableRef}:
 production[1]
 production[2]
 production[3]
 production[4]
```

```
[ ]: @variable(model, subcontract[1:4], lower_bound = 0, Int)
```

```
4-element Vector{VariableRef}:
 subcontract[1]
 subcontract[2]
 subcontract[3]
 subcontract[4]
```

```
[ ]: @variable(model, inventory[1:4], lower_bound = 0, Int)
```

```
4-element Vector{VariableRef}:
 inventory[1]
 inventory[2]
 inventory[3]
 inventory[4]
```

```
[ ]: @variable(model, backlog[1:4], lower_bound = 0, Int)
```

```
4-element Vector{VariableRef}:
 backlog[1]
 backlog[2]
 backlog[3]
 backlog[4]
```

```
[ ]: @constraint(model, w[1]== num_init_workers + w_h[1] - w_f[1]) # # relating␣
      ↪number of workers to workers being hired and fired
```

$$w_1 - w\_h_1 + w\_f_1 = 600$$

```
[ ]: @constraint(model, production[1] + inventory_start + subcontract[1] ==␣
      ↪demand[1] + inventory[1] + backlog[1]) # Relating inventory with production␣
      ↪subcontracting and backlog
```

$$production_1 + subcontract_1 - inventory_1 - backlog_1 = 1300$$

```
[ ]: @constraint(model, w[2:4]== w[1:3] + w_h[2:4] - w_f[2:4]) # relating number of␣
      ↪workers to workers being hired and fired
```

$$[-w_1+w_2-w\_h_2+w\_f_2, -w_2+w_3-w\_h_3+w\_f_3, -w_3+w_4-w\_h_4+w\_f_4] \in \text{MathOptInterface.Zeros(3)}$$

```
[ ]: @constraint(model,production[2:4] + inventory[1:3] + subcontract[2:4] ==␣
      ↪demand[2:4] + inventory[2:4] + backlog[1:3] + backlog[2:4]) # Relating␣
      ↪inventory with production subcontracting and backlog
```

$$[production_2+subcontract_2+inventory_1-inventory_2-backlog_1-backlog_2-2100, production_3+subcontract_3+inve$$

```
[ ]: @constraint(model, total_overtime_hrs <= max_overtime * w[1:4]) # Max overtime␣
      ↪constraint
```

$$[total\_overtime\_hrs_1-80w_1, total\_overtime\_hrs_2-80w_2, total\_overtime\_hrs_3-80w_3, total\_overtime\_hrs_4-8$$

```
[ ]: @constraint(model, production <= w[1:4] * regular_hrs / production_rate +␣
     ↪total_overtime_hrs / production_rate) # Max production constraint
```

$[-0.00625 total\_overtime\_hrs_1 - 3w_1 + production_1, -0.00625 total\_overtime\_hrs_2 - 3w_2 + production_2, -0.00625t$

```
[ ]: @constraint(model, subcontract[1:4] .<= max_subcontract) # Max subcontracted␣
     ↪amount constraint
```

```
4-element Vector{ConstraintRef{Model, MathOptInterface.
 ↪ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},␣
 ↪MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 subcontract[1] <= 500
 subcontract[2] <= 500
 subcontract[3] <= 500
 subcontract[4] <= 500
```

```
[ ]: @constraint(model, inventory[4] == inventory_end) # final inventory should be␣
     ↪300 vaccum cleaners
```

$$inventory_4 = 300$$

```
[ ]: total_cost = sum(w .* regular_hrs .* regular_pay) + sum(hiring_cost .* w_h) +␣
     ↪sum(firing_cost .* w_f) + sum(inventory_holding_cost .* inventory) +␣
     ↪sum(backlog_cost .* backlog) + sum(subcontracting_cost .* subcontract) +␣
     ↪sum(overtime_pay .* total_overtime_hrs)
```

$24000w_1 + 24000w_2 + 24000w_3 + 24000w_4 + 1000w\_h_1 + 1000w\_h_2 + 1000w\_h_3 + 1000w\_h_4 + 1200w\_f_1 + 1200w\_f_2 + 1$

```
[ ]: @objective(model, Min, total_cost)
```

$24000w_1 + 24000w_2 + 24000w_3 + 24000w_4 + 1000w\_h_1 + 1000w\_h_2 + 1000w\_h_3 + 1000w\_h_4 + 1200w\_f_1 + 1200w\_f_2 + 1$

```
[ ]: optimize!(model)
```

```
Lift and project cuts applied:  1

Root node processing (before b&c):
  Real time             =    0.05 sec. (0.45 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.05 sec. (0.45 ticks)
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Tried aggregator 2 times.
MIP Presolve eliminated 9 rows and 2 columns.
MIP Presolve added 5 rows and 5 columns.
Aggregator did 1 substitutions.
Reduced MIP has 19 rows, 34 columns, and 66 nonzeros.
Reduced MIP has 0 binaries, 34 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.05 ticks)
Found incumbent of value 7.3939200e+07 after 0.00 sec. (0.12 ticks)
Tried aggregator 1 time.
Detecting symmetries...
MIP Presolve eliminated 5 rows and 5 columns.
MIP Presolve added 5 rows and 5 columns.
Reduced MIP has 19 rows, 34 columns, and 66 nonzeros.
Reduced MIP has 0 binaries, 34 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.04 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.00 sec. (0.05 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                       7.39392e+07   15000.0000            99.98%
*     0+    0                       7.38201e+07   15000.0000            99.98%
      0     0   5.96996e+07    10   7.38201e+07   5.96996e+07       10   19.13%
*     0+    0                       5.97157e+07   5.96996e+07             0.03%
*     0+    0                       5.97156e+07   5.96996e+07             0.03%
      0     0   5.96996e+07    10   5.97156e+07      Fract: 1       11    0.03%
*     0+    0                       5.97005e+07   5.96996e+07             0.00%
```

```
[ ]: @show value.(w) # Number of workers in each quarter
```

```
value.(w) = [620.0000000001232, 621.0, 621.0, 621.0]

4-element Vector{Float64}:
 620.0000000001232
 621.0
 621.0
 621.0
```

```
[ ]: @show value.(inventory)
```

```
value.(inventory) = [560.0000000003697, 323.00000000036965, 387.0, 300.0]

4-element Vector{Float64}:
 560.0000000003697
 323.00000000036965
 387.0
 300.0
```

```
[ ]: @show value.(backlog)
```

```
value.(backlog) = [-0.0, -0.0, -0.0, 0.0]

4-element Vector{Float64}:
 -0.0
 -0.0
 -0.0
  0.0
```

```
[ ]: @show value.(w_h)
```

```
value.(w_h) = [20.000000000123226, 0.9999999998767743, -0.0, -0.0]

4-element Vector{Float64}:
 20.000000000123226
  0.9999999998767743
 -0.0
 -0.0
```

```
[ ]: @show value.(w_f)
```

```
value.(w_f) = [0.0, 0.0, 0.0, 0.0]

4-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
```

```
[ ]: @show value.(subcontract)
```

```
value.(subcontract) = [0.0, -0.0, 0.9999999996303232, 0.0]

4-element Vector{Float64}:
  0.0
 -0.0
  0.9999999996303232
  0.0
```

```
[ ]: @show value.(production)
```

```
value.(production) = [1860.0000000003697, 1863.0, 1863.0, 1863.0]

4-element Vector{Float64}:
 1860.0000000003697
 1863.0
 1863.0
 1863.0
```

[ ]:

```
@show value.(total_overtime_hrs)
```

```
value.(total_overtime_hrs) = [0.0, 0.0, 0.0, 0.0]
```

```
4-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
```

[ ]:

```
@show objective_value(model)
```

```
objective_value(model) = 5.970049999999967e7
```

```
5.970049999999967e7
```

9

October 2, 2023

[ ]:
```
using JuMP
using CPLEX
```

[ ]:
```
order_details = [14 5 200; 31 10 350; 36 15 400; 45 5 500]
```

```
4×3 Matrix{Int64}:
 14   5  200
 31  10  350
 36  15  400
 45   5  500
```

[ ]:
```
scrap_price = 5
```

```
5
```

[ ]:
```
manufacturing_cost = 700
```

```
700
```

[ ]:
```
model = Model(CPLEX.Optimizer)
```

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: CPLEX
```

[ ]:
```
@variable(model, x[i=1:4, j =1:10], lower_bound = 0, Int) # Amount of orders of each type to be cut from available 10 rolls
```

```
4×10 Matrix{VariableRef}:
 x[1,1]  x[1,2]  x[1,3]  x[1,4]  x[1,5]  …  x[1,7]  x[1,8]  x[1,9]  x[1,10]
 x[2,1]  x[2,2]  x[2,3]  x[2,4]  x[2,5]     x[2,7]  x[2,8]  x[2,9]  x[2,10]
 x[3,1]  x[3,2]  x[3,3]  x[3,4]  x[3,5]     x[3,7]  x[3,8]  x[3,9]  x[3,10]
 x[4,1]  x[4,2]  x[4,3]  x[4,4]  x[4,5]     x[4,7]  x[4,8]  x[4,9]  x[4,10]
```

[ ]:
```
@constraint(model, sum(x[:, j] for j in 1:10) <= order_details[:, 2]) # Constraint to make sure that production of any particular type do not exceedits demand
```

$[x_{1,1}+x_{1,2}+x_{1,3}+x_{1,4}+x_{1,5}+x_{1,6}+x_{1,7}+x_{1,8}+x_{1,9}+x_{1,10}-5, x_{2,1}+x_{2,2}+x_{2,3}+x_{2,4}+x_{2,5}+x_{2,6}+x_{2,7}+x_{2,8}+x_{2,9}+x_{2,10}$

[ ]:
```
@constraint(model, sum((x .*order_details[:, 1])[i,:] for i in 1:4) .<= 100) # Roll length contraint
```

```
10-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 14 x[1,1] + 31 x[2,1] + 36 x[3,1] + 45 x[4,1] <= 100
 14 x[1,2] + 31 x[2,2] + 36 x[3,2] + 45 x[4,2] <= 100
 14 x[1,3] + 31 x[2,3] + 36 x[3,3] + 45 x[4,3] <= 100
 14 x[1,4] + 31 x[2,4] + 36 x[3,4] + 45 x[4,4] <= 100
 14 x[1,5] + 31 x[2,5] + 36 x[3,5] + 45 x[4,5] <= 100
 14 x[1,6] + 31 x[2,6] + 36 x[3,6] + 45 x[4,6] <= 100
 14 x[1,7] + 31 x[2,7] + 36 x[3,7] + 45 x[4,7] <= 100
 14 x[1,8] + 31 x[2,8] + 36 x[3,8] + 45 x[4,8] <= 100
 14 x[1,9] + 31 x[2,9] + 36 x[3,9] + 45 x[4,9] <= 100
 14 x[1,10] + 31 x[2,10] + 36 x[3,10] + 45 x[4,10] <= 100
```

[ ]:
```
scrap = 100 .- sum((x .*order_details[:, 1])[i,:] for i in 1:4)
```

```
10-element Vector{AffExpr}:
 -14 x[1,1] - 31 x[2,1] - 36 x[3,1] - 45 x[4,1] + 100
 -14 x[1,2] - 31 x[2,2] - 36 x[3,2] - 45 x[4,2] + 100
 -14 x[1,3] - 31 x[2,3] - 36 x[3,3] - 45 x[4,3] + 100
 -14 x[1,4] - 31 x[2,4] - 36 x[3,4] - 45 x[4,4] + 100
 -14 x[1,5] - 31 x[2,5] - 36 x[3,5] - 45 x[4,5] + 100
 -14 x[1,6] - 31 x[2,6] - 36 x[3,6] - 45 x[4,6] + 100
 -14 x[1,7] - 31 x[2,7] - 36 x[3,7] - 45 x[4,7] + 100
 -14 x[1,8] - 31 x[2,8] - 36 x[3,8] - 45 x[4,8] + 100
 -14 x[1,9] - 31 x[2,9] - 36 x[3,9] - 45 x[4,9] + 100
 -14 x[1,10] - 31 x[2,10] - 36 x[3,10] - 45 x[4,10] + 100
```

[ ]:
```
profit = sum(sum((x .* order_details[:, 3])[i,:] for i in 1:4)) + sum(scrap * scrap_price) - manufacturing_cost*10
```

$130x_{1,1}+195x_{2,1}+220x_{3,1}+275x_{4,1}+130x_{1,2}+195x_{2,2}+220x_{3,2}+275x_{4,2}+130x_{1,3}+195x_{2,3}+220x_{3,3}+275x_{4,3}+130x_{1,4}$

[ ]:
```
@objective(model, Max, profit)
```

$130x_{1,1}+195x_{2,1}+220x_{3,1}+275x_{4,1}+130x_{1,2}+195x_{2,2}+220x_{3,2}+275x_{4,2}+130x_{1,3}+195x_{2,3}+220x_{3,3}+275x_{4,3}+130x_{1,4}$

[ ]:
```
optimize!(model)
```

```
Mixed integer rounding cuts applied:  41
Zero-half cuts applied:  2

Root node processing (before b&c):
  Real time             =    0.00 sec. (0.01 ticks)
Parallel b&c, 8 threads:
  Real time             =    9.36 sec. (2016.17 ticks)
  Sync time (average)   =    1.29 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    9.36 sec. (2016.17 ticks)
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
Found incumbent of value -2000.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 14 rows, 40 columns, and 80 nonzeros.
Reduced MIP has 0 binaries, 40 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.04 ticks)
Tried aggregator 1 time.
Detecting symmetries…
Reduced MIP has 14 rows, 40 columns, and 80 nonzeros.
Reduced MIP has 0 binaries, 40 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.07 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.00 sec. (0.06 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                       -2000.0000    20250.0000             ---
*     0+    0                        4175.0000    20250.0000           385.03%
      0     0     4388.8889    10     4175.0000     4388.8889       28    5.12%
      0     0     4388.8889    18     4175.0000      Cuts: 13       52    5.12%
      0     0     4388.8889    19     4175.0000      Cuts: 16       79    5.12%
      0     2     4388.8889    19     4175.0000     4388.8889       79    5.12%
Elapsed time = 0.06 sec. (1.11 ticks, tree = 0.02 MB, solutions = 2)
```

[ ]:
```
@show value.(x) # Amount of rolls of each type to be cut to maximise profit on␣
 ↪each available roll
```

```
value.(x) = [1.0 0.0 2.0 0.0 2.0 0.0 0.0 0.0 0.0 0.0; 0.0 2.0 0.0 2.0 0.0 2.0
0.0 2.0 0.0 2.0; 1.0 1.0 2.0 1.0 2.0 1.0 0.0 1.0 0.0 1.0; 1.0 0.0 0.0 0.0 0.0
0.0 2.0 0.0 2.0 0.0]
```

```
4×10 Matrix{Float64}:
 1.0  0.0  2.0  0.0  2.0  0.0  0.0  0.0  0.0  0.0
 0.0  2.0  0.0  2.0  0.0  2.0  0.0  2.0  0.0  2.0
 1.0  1.0  2.0  1.0  2.0  1.0  0.0  1.0  0.0  1.0
 1.0  0.0  0.0  0.0  0.0  0.0  2.0  0.0  2.0  0.0
```

[ ]:

```
@show objective_value(model)
```

```
objective_value(model) = 4175.0
```

```
4175.0
```