Set-up / Environment Configuration

In [1]:

```
!pip install gurobipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Collecting gurobipy
  Downloading gurobipy-10.0.0-cp38-cp38-manylinux2014_x86_64.whl (12.8 MB)
     |████████████████████████████████| 12.8 MB 23.3 MB/s
Installing collected packages: gurobipy
Successfully installed gurobipy-10.0.0
```

In [2]:

```
from google.colab import drive
import os
drive.mount('/content/drive', force_remount=True)
os.chdir(os.path.join(os.getcwd(), 'drive','MyDrive', 'Colab Notebooks', 'gurobi'))
```

```
Mounted at /content/drive
```

In [3]:

```
import gurobipy as gp
from gurobipy import GRB
with open('gurobi.lic', 'r') as f:
    lic = f.readlines()

WLSACCESSID = lic[-3].replace('\n', '').replace('WLSACCESSID=', '')
WLSSECRET = lic[-2].replace('\n', '').replace('WLSSECRET=', '')
LICENSEID = int( lic[-1].replace('\n', '').replace('LICENSEID=', '') )

e = gp.Env(empty=True)
e.setParam('WLSACCESSID', WLSACCESSID)
e.setParam('WLSSECRET', WLSSECRET)
e.setParam('LICENSEID', LICENSEID)
e.start()
```

```
Set parameter WLSAccessID
Set parameter WLSSecret
Set parameter LicenseID to value 889498
Academic license - for non-commercial use only - registered to klt45@cornell.edu
```

Out[3]:

```
<gurobipy.Env, Parameter changes: WLSAccessID=(user-defined), WLSSecret=(user-defined), Li
censeID=889498>
```

Problem Statement

m machines
for each machine i:

- d_i is the number of work cycles to be executed on machine i
- t_i is the time it takes to execute a work cycle on machine
- a_i is the noise dosage that one work cycle on machine i inflict on worker
- u_i is the maximum number of work cycles any worker can execute on machine i

H is the maximum number of hours a worker can work
z is the upper bound on how much noise dosage any schedule can have

Output: find an assignment of work cycles to workers such that the maximum noise dosage inflicted on any worker is minimized

In [4]:
```
import numpy as np
n = 36
m = 25
H = 80
z = 145

arr = np.array([
1,      2,      41,     33,     43,
2,      2,      33,     93,     29,
3,      4,      9,   57, 10,
4,      6,      9,      59,     17,
5,      2,      6,      36,     23,
6,      6,      41,     86,     49,
7,      4,      12,     14,     30,
8,      3,      17,     97,     53,
9,      3,      29,     17,     39,
10,     4,      17,     40,     21,
11,     2,      17,     71,     39,
12,     4,      26,     30,     57,
13,     3,      37,     46,     21,
14,     3,      12,     63,     37,
15,     2,      16,     22,     11,
16,     6,      57,     72,     33,
17,     5,      10,     24,     37,
18,     3,      39,     91,     27,
19,     3,      42,     87,     54,
20,     3,      35,     95,     46,
21,     6,      35,     51,     45,
22,     2,      24,     18,     54,
23,     4,      20,     94,     27,
24,     3,      24,     47,     29,
25,     5,      58,     21,     50,
]).reshape(25,5).T

d, t, alpha, u = arr[1], arr[2], arr[3], arr[4]
```

Primal Problem

Let s be the schedules for workers of how many work cycles they execute on each machine

Decision Variables/Parameters:

x_s is the number of workers that work according to schedule s

w_{si} be the number of work cycles on machine i in schedule s

Objective:

$$\min \sum_s x_s$$

Constraint:

$$\sum_s w_{si} x_s \geq d_i \ \forall \text{ machines } i$$

In [55]:

```python
import gurobipy as gp
from gurobipy import GRB
def Primal(m, d, S, w, debug=False):
    """
    m: number of machines
    d: number of work cycles to be executed on each machine
    S: number of schedules
    w: number of work cycles on each machine in each schedule
    """
    model = gp.Model("Primal", env=e)
    if not debug:
        model.setParam('OutputFlag', False)
    x = model.addVars(list(range(1, S+1)), vtype=GRB.INTEGER, name="xs")
    model.setObjective(x.sum(), GRB.MINIMIZE)
    for i in range(m):
        model.addConstr(gp.quicksum(w[s-1][i] * x[s] for s in range(1,S+1)) >= d[i])
    model.optimize()
    for v in model.getVars():
        if debug:
            print(v.varName, '=', v.x)
    return model.objVal

S = 25
w = np.eye(S)# 25 x 25 --> 1 on diagonals
Primal(m, d, S, w, debug=True)
```

```
Gurobi Optimizer version 10.0.0 build v10.0.0rc2 (linux64)

CPU model: AMD EPYC 7B12, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to klt45@cornell.edu
Optimize a model with 25 rows, 25 columns and 25 nonzeros
Model fingerprint: 0xfcfea8a1
Variable types: 0 continuous, 25 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 6e+00]
Found heuristic solution: objective 90.0000000
Presolve removed 25 rows and 25 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.06 seconds (0.00 work units)
Thread count was 1 (of 2 available processors)

Solution count 1: 90

Optimal solution found (tolerance 1.00e-04)
Best objective 9.000000000000e+01, best bound 9.000000000000e+01, gap 0.0000%
xs[1] = 2.0
xs[2] = 2.0
xs[3] = 4.0
xs[4] = 6.0
xs[5] = 2.0
xs[6] = 6.0
xs[7] = 4.0
xs[8] = 3.0
xs[9] = 3.0
xs[10] = 4.0
xs[11] = 2.0
xs[12] = 4.0
```

```
xs[13] = 3.0
xs[14] = 3.0
xs[15] = 2.0
xs[16] = 6.0
xs[17] = 5.0
xs[18] = 3.0
xs[19] = 3.0
xs[20] = 3.0
xs[21] = 6.0
xs[22] = 2.0
xs[23] = 4.0
xs[24] = 3.0
xs[25] = 5.0
```

Out[55]:

```
90.0
```

In [51]:

```python
import pandas as pd
pd.DataFrame(w, columns=[f'machine {i}' for i in range(1, m+1)], index=[f'schedule {i}' fo
```

Out[51]:

	machine 1	machine 2	machine 3	machine 4	machine 5	machine 6	machine 7	machine 8	machine 9	machine 10
schedule 1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 5	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
schedule 6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
schedule 7	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
schedule 8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
schedule 9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
schedule 10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
schedule 11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	machine 1	machine 2	machine 3	machine 4	machine 5	machine 6	machine 7	machine 8	machine 9	machine 10
schedule 17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 21	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 22	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
schedule 25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

25 rows × 25 columns

Dual Problem

Objective:

$$max \sum_i d_i y_i$$

Constraints:

$$\sum_i w_{si} y_i \leq 1 \; \forall \text{ schedules } s$$

$$y_i \geq 0 \; \forall \text{ machines } i$$

In [54]:

```
def Dual(m, d, S, w, debug=False):
    """
    m: number of machines
    d: number of work cycles to be executed on each machine
    S: number of schedules
    w: number of work cycles on each machine in each schedule
    """
    model = gp.Model("Dual")
    if not debug:
        model.setParam('OutputFlag', False)
    y = model.addVars(list(range(1, m+1)), vtype=GRB.CONTINUOUS, name="yi")
    model.setObjective(gp.quicksum(d[i-1] * y[i] for i in range(1, m+1)), GRB.MAXIMIZE)
    for s in range(S):
        model.addConstr(gp.quicksum(w[s][i-1] * y[i] for i in range(1, m+1)) <= 1)
    model.optimize()
    yi_star = []
```

```python
    for v in model.getVars():
        if debug:
            print(v.varName, '=', v.x)
        yi_star.append(v.x)
    return yi_star

Dual(m, d, S, w, debug=True) #  yi_star is a vector
```

```
Gurobi Optimizer version 10.0.0 build v10.0.0rc2 (linux64)

CPU model: AMD EPYC 7B12, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 25 rows, 25 columns and 25 nonzeros
Model fingerprint: 0x7d2cf07a
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+00, 6e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 25 rows and 25 columns
Presolve time: 0.01s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.0000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  9.000000000e+01
yi[1] = 1.0
yi[2] = 1.0
yi[3] = 1.0
yi[4] = 1.0
yi[5] = 1.0
yi[6] = 1.0
yi[7] = 1.0
yi[8] = 1.0
yi[9] = 1.0
yi[10] = 1.0
yi[11] = 1.0
yi[12] = 1.0
yi[13] = 1.0
yi[14] = 1.0
yi[15] = 1.0
yi[16] = 1.0
yi[17] = 1.0
yi[18] = 1.0
yi[19] = 1.0
yi[20] = 1.0
yi[21] = 1.0
yi[22] = 1.0
yi[23] = 1.0
yi[24] = 1.0
yi[25] = 1.0
```

Out[54]:

```
[1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
```

```
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0]
```

Integer Linear Program to see if constraint is violated by dual solution

- Solving the primal with only a subset of the variables is the same as solving the dual with only a subset of the constraints
- We get an optimal dual solution for the relaxation that only has a subset of the constraints
- We call this optimal dual solution y_i^*
- Now we want to check whether the constraints we satisfied for all schedules, and if not produce the schedule for which it is violated
- y_i^* now input to this problem

Decision variables:
w_i = number of work cycles on machine i

Objective:

$$\max \sum_i w_i y_i^*$$

Constraints:

1. $\sum_i a_i w_i \leq z$ <-- noise dosage for this schedule

1. $w_i \leq u_i \; \forall$ machines i

1. $\sum_i t_i w_i \leq H$ <-- time in hours for this schedule

2. $w_i \geq 0 \; \forall$ machines i

If objective value of an optimal solution ≤ 1, then we do not have any violated constraints in the dual -- means y_i^* is an optimal solution even with all constraints.

Otherwise, we found a schedule that corresponds to a violated constraint in dual

In [68]:

```
def subProblem(m, d, S, z, H, t, alpha, u, yi_star, debug=False):
    """
    m: number of machines
    d: number of work cycles to be executed on each machine
    S: number of schedules
    z: upper bound on noise dosage
```

```
    H: upper bound on time in hours
    d: number of work cycles on each machine in each schedule
    t: time in hours for each machine in each schedule
    alpha: noise dosage for each machine in each schedule
    u: upper bound on number of work cycles for each machine
    yi_star: optimal dual solution for the relaxation that only has a subset of the constr
    """
    model = gp.Model("subProblem", env=e)
    if not debug:
        model.setParam('OutputFlag', False)
    w = model.addVars(list(range(1, m+1)), vtype=GRB.INTEGER, name="wi") # de
    model.setObjective(gp.quicksum(w[i] * yi_star[i-1] for i in range(1, m+1)), GRB.MAXIMI
    model.addConstr(gp.quicksum(alpha[i-1] * w[i] for i in range(1, m+1)) <= z)
    for i in range(1, m+1):
        model.addConstr(w[i] <= u[i-1])
    model.addConstr(gp.quicksum(t[i-1] * w[i] for i in range(1, m+1)) <= H)
    model.optimize()
    column = []
    for v in model.getVars():
        if debug:
            print(v.varName, '=', v.x)
        column.append(v.x)
    return model.objVal, column

obj, column = subProblem(m, d, S, z, H, t, alpha, u, yi_star, debug=True)
column
```

```
Gurobi Optimizer version 10.0.0 build v10.0.0rc2 (linux64)

CPU model: AMD EPYC 7B12, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to klt45@cornell.edu
Optimize a model with 27 rows, 25 columns and 75 nonzeros
Model fingerprint: 0x75f3a76f
Variable types: 0 continuous, 25 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 1e+02]
Found heuristic solution: objective 2.0000000
Presolve removed 25 rows and 22 columns
Presolve time: 0.00s
Presolved: 2 rows, 3 columns, 6 nonzeros
Found heuristic solution: objective 4.0000000
Variable types: 0 continuous, 3 integer (0 binary)

Root relaxation: objective 7.540230e+00, 3 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    7.54023    0    2    4.00000    7.54023  88.5%     -    0s
H    0     0                       7.0000000    7.54023  7.72%     -    0s
     0     0    7.54023    0    2    7.00000    7.54023  7.72%     -    0s

Explored 1 nodes (3 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 3: 7 4 2

Optimal solution found (tolerance 1.00e-04)
Best objective 7.000000000000e+00, best bound 7.000000000000e+00, gap 0.0000%
wi[1] = -0.0
```

```
wi[2] = -0.0
wi[3] = -0.0
wi[4] = -0.0
wi[5] = 1.0
wi[6] = -0.0
wi[7] = 6.0
wi[8] = -0.0
wi[9] = -0.0
wi[10] = -0.0
wi[11] = -0.0
wi[12] = -0.0
wi[13] = -0.0
wi[14] = -0.0
wi[15] = -0.0
wi[16] = -0.0
wi[17] = -0.0
wi[18] = -0.0
wi[19] = -0.0
wi[20] = -0.0
wi[21] = -0.0
wi[22] = -0.0
wi[23] = -0.0
wi[24] = -0.0
wi[25] = -0.0
```

Out[68]:

```
[-0.0,
 -0.0,
 -0.0,
 -0.0,
 1.0,
 -0.0,
 6.0,
 -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]
```

Putting it together - Takes ~ 3 seconds to solve, and the optimal primal objective is 37

In [69]:

```
def NoiseDosage(n, m, H, z, d, t, alpha, u):
    S = m # initialize with as many schedules as you have machines
    w = np.eye(S)# 25 x 25 --> 1 on diagonals
    objective = float('inf')
    while objective > 1:
        yi_star = Dual(m, d, S, w) #  yi_star is a vector
        objective, column = subProblem(m, d, S, z, H, t, alpha, u, yi_star)
        S += 1
        w = np.vstack( (w, np.array(column)) ) # add column
    return Primal(m, d, S, w, debug=True)#w, S, objective
```

```
NoiseDosage(n, m, H, z, d, t, alpha, u)
```

```
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Academic license - for non-commercial use only - registered to klt45@cornell.edu
Gurobi Optimizer version 10.0.0 build v10.0.0rc2 (linux64)

CPU model: AMD EPYC 7B12, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to klt45@cornell.edu
Optimize a model with 25 rows, 67 columns and 121 nonzeros
Model fingerprint: 0xbc144c59
Variable types: 0 continuous, 67 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 6e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 6e+00]
Found heuristic solution: objective 47.0000000
Presolve removed 3 rows and 28 columns
Presolve time: 0.00s
Presolved: 22 rows, 39 columns, 91 nonzeros
```

```
Variable types: 0 continuous, 39 integer (3 binary)

Root relaxation: objective 3.616667e+01, 34 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   36.16667    0   15   47.00000   36.16667  23.0%     -    0s
H    0     0                      38.0000000   36.16667  4.82%     -    0s
H    0     0                      37.0000000   36.16667  2.25%     -    0s
     0     0   36.16667    0   15   37.00000   36.16667  2.25%     -    0s

Explored 1 nodes (38 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 2 (of 2 available processors)

Solution count 3: 37 38 47

Optimal solution found (tolerance 1.00e-04)
Best objective 3.700000000000e+01, best bound 3.700000000000e+01, gap 0.0000%
xs[1] = -0.0
xs[2] = -0.0
xs[3] = -0.0
xs[4] = -0.0
xs[5] = -0.0
xs[6] = -0.0
xs[7] = -0.0
xs[8] = -0.0
xs[9] = -0.0
xs[10] = -0.0
xs[11] = -0.0
xs[12] = -0.0
xs[13] = -0.0
xs[14] = -0.0
xs[15] = -0.0
xs[16] = -0.0
xs[17] = -0.0
xs[18] = -0.0
xs[19] = -0.0
xs[20] = -0.0
xs[21] = -0.0
xs[22] = -0.0
xs[23] = -0.0
xs[24] = -0.0
xs[25] = -0.0
xs[26] = 0.0
xs[27] = 1.0
xs[28] = 0.0
xs[29] = 0.0
xs[30] = 2.0
xs[31] = 0.0
xs[32] = 0.0
xs[33] = 0.0
xs[34] = 0.0
xs[35] = 1.0
xs[36] = 3.0
xs[37] = 0.0
xs[38] = 0.0
xs[39] = 0.0
xs[40] = 0.0
xs[41] = 1.0
xs[42] = 1.0
xs[43] = 3.0
xs[44] = 2.0
xs[45] = 0.0
xs[46] = 0.0
xs[47] = 4.0
```

```
xs[48] = 1.0
xs[49] = 3.0
xs[50] = 1.0
xs[51] = 0.0
xs[52] = 1.0
xs[53] = 1.0
xs[54] = 2.0
xs[55] = 0.0
xs[56] = 0.0
xs[57] = 3.0
xs[58] = 0.0
xs[59] = 0.0
xs[60] = 3.0
xs[61] = 0.0
xs[62] = 2.0
xs[63] = 0.0
xs[64] = 0.0
xs[65] = 1.0
xs[66] = 0.0
xs[67] = 1.0
```

Out[69]:

```
37.0
```