UNITCOMMITMENT+2

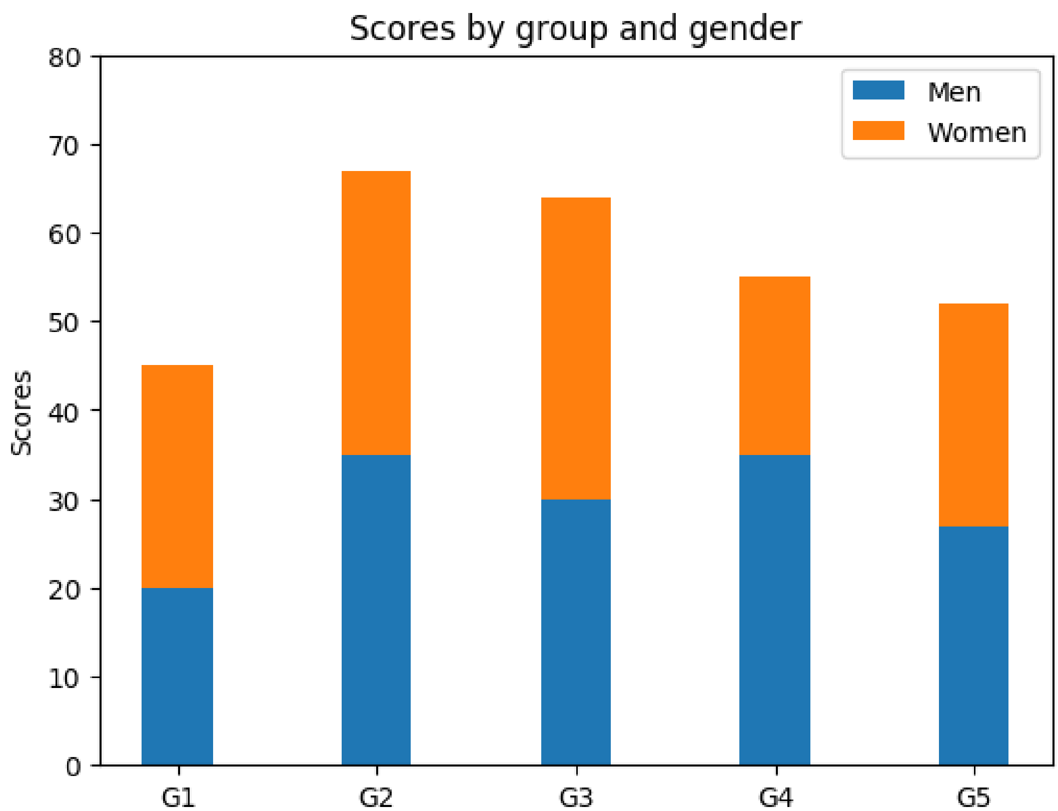
April 24, 2021

[44]:
```
A=matrix([[10,6],[20,18]])
x,y,B= var('x,y,B')
for x in range(20):
    for y in range(20):
        if ((A[0][0]*x+A[0][1]*y)<=300 & (A[1][0]*x+A[1][1]*y)>200):
            B=[5*x+4*y]
            for i in range(len(B)):
                C=[{B[i]:(x,y)}]
                #print(C)

l = [22, 13, 45, 50, 98, 69, 43, 44, 1]
[x+1 if x >= 45 else x+5 for x in l]
```

[44]:
```
[27, 18, 46, 51, 99, 70, 48, 49, 6]
```

[45]:
```
 histogram([ [1,1,1,1,2,2,2,3,3,3], [4,4,4,4,3,3,3,2,2,2] ], stacked=True,␣
 ↪color=['blue', 'red'])
import numpy as np
import matplotlib.pyplot as plt



N = 5
menMeans = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd = (2, 3, 4, 1, 2)
womenStd = (3, 5, 2, 3, 3)
ind = np.arange(N)    # the x locations for the groups
width = 0.35       # the width of the bars: can also be len(x) sequence

p1 = plt.bar(ind, menMeans, width)#, yerr=menStd)
p2 = plt.bar(ind, womenMeans, width,
             bottom=menMeans)#, yerr=womenStd)

plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.yticks(np.arange(0, 81, 10))
```

```
plt.legend((p1[0], p2[0]), ('Men', 'Women'))

plt.show()
```



0.1 MIXED INTEGER LINEAR PROGRAMMING

[46]:

```
UNITCOMIT=matrix([[3000,400,440,2,50,2,400,40,75,5200,43.536,50],
                  [2500,345,365,2,70,3,350,50,74,4700,42.985,44],
                  [2670,180,220,2,78.00,2,205,40,65,1320,50.5,36],
                  [3132,61,210,2,52,3,197,40,56,1291,59,29],
                  [2020,165,165,2,54.25,2,155,30,67,1280,37.146,43],
                  [1940,158,158,2,39.00,2,150,60.0,70.0,1105,44.84,35],
                  [2980,0,90,2,17.40,2,98,34.0,60.0,560,40.222,33],
                  [2544,32,87,3,15.20,2,76,55.0,80.0,554,46.522,40],
                  [3220,0,20,2,4.0,3,20,40,40,300,56.33,43],
                  [1560,9,12,2,2.4,2,12,22,22,250,57.642,23]])
UNITCOMIT.change_ring(CDF).round(1)
```

```
[46]: [3000.0  400.0  440.0    2.0   50.0    2.0  400.0   40.0   75.0 5200.0   43.5
50.0]
 [2500.0  345.0  365.0    2.0   70.0    3.0  350.0   50.0   74.0 4700.0   43.0
44.0]
 [2670.0  180.0  220.0    2.0   78.0    2.0  205.0   40.0   65.0 1320.0   50.5
36.0]
 [3132.0   61.0  210.0    2.0   52.0    3.0  197.0   40.0   56.0 1291.0   59.0
29.0]
 [2020.0  165.0  165.0    2.0   54.2    2.0  155.0   30.0   67.0 1280.0   37.1
43.0]
 [1940.0  158.0  158.0    2.0   39.0    2.0  150.0   60.0   70.0 1105.0   44.8
35.0]
 [2980.0    0.0   90.0    2.0   17.4    2.0   98.0   34.0   60.0  560.0   40.2
33.0]
 [2544.0   32.0   87.0    3.0   15.2    2.0   76.0   55.0   80.0  554.0   46.5
40.0]
 [3220.0    0.0   20.0    2.0    4.0    3.0   20.0   40.0   40.0  300.0   56.3
43.0]
 [1560.0    9.0   12.0    2.0    2.4    2.0   12.0   22.0   22.0  250.0   57.6
23.0]
```

```
[47]: M=sum([UNITCOMIT[n][1] for n in range(10)])
M
```

```
[47]: 1350.00000000000
```

```
[48]: table(UNITCOMIT.change_ring(CDF).round(1))
```

```
[48]: <repr(<sage.misc.table.table at 0x7fe0091d5650>) failed: TypeError: object of
type 'sage.matrix.matrix_complex_double_dense.Matrix_complex_double_dense' has
no len()>
```

```
[49]: UNITCOMITTMENT=[['unit','energy','fixed-cost','initial','max-gen','min-downtime','min-gen','mi
                ['coal1','coal',208,400,440,2,50,3,400,40,75,5200,43],
                ['coal2','coal',117,345,365,2,70,2,350,50,74,4700,43],
                ['gas1','gas',174,180,220,3,78,3,205,40,65,1320,50],
                ['gas2','gas',172,60,210,1,52,2,197,40,56,1291,59],
                ['gas3','gas',95,165,165,2,54,2,155,30,67,1280,37],
                ['gas4','gas',144,98,158,2,39,2,150,60,70,1105,44],
                ['diesel1','diesel',54,70,90,2,17,3,98,34,60,560,40],
                ['diesel2','diesel',55,76,87,2,15,3,76,20,35,554,41],
                ['diesel3','diesel',80,0,20,2,3,1,20,18,60,300,116],
                ['diesel4','diesel',16,0,12,2,2,1,12,5,12,250,77]]
table(UNITCOMITTMENT)
```

```
[49]:   unit      energy    fixed-cost    initial    max-gen    min-downtime    min-gen
min-uptime    operating-max-gen    ramp-down    ranp-up    start-cost    variable-
```

```
cost
  coal1     coal    208         400       440       2              50        3
400                 40          75        5200       43
  coal2     coal    117         345       365       2              70        2
350                 50          74        4700       43
  gas1      gas     174         180       220       3              78        3
205                 40          65        1320       50
  gas2      gas     172          60       210       1              52        2
197                 40          56        1291       59
  gas3      gas     95          165       165       2              54        2
155                 30          67        1280       37
  gas4      gas     144          98       158       2              39        2
150                 60          70        1105       44
  diesel1   diesel  54           70        90       2              17        3
98                  34          60        560        40
  diesel2   diesel  55           76        87       2              15        3
76                  20          35        554        41
  diesel3   diesel  80            0        20       2              3         1
20                  18          60        300        116
  diesel4   diesel  16            0        12       2              2         1
12                  5           12        250        77
```

[50]:
```
raw_demand= [1259.0, 1439.0, 1289.0, 1211.0, 1433.0, 1287.0, 1285.0, 1227.0,␣
↪1269.0, 1158.0, 1277.0, 1417.0, 1294.0, 1396.0, 1414.0, 1386.0,
              1302.0, 1215.0, 1433.0, 1354.0, 1436.0, 1285.0, 1392.0, 1172.0,␣
↪1446.0, 1367.0, 1243.0, 1275.0, 1363.0, 1258, 1394.0, 1345.0,
              1217.0, 1432.0, 1431.0, 1356.0, 1360.0, 1364.0, 1286.0, 1440.0,␣
↪1440.0, 1313.0, 1389.0, 1385.0, 1265.0, 1442.0, 1435.0, 1432.0,
              1280.0, 1411.0, 1440.0, 1258.0, 1333.0, 1293.0, 1193.0, 1440.0,␣
↪1306.0, 1264.0, 1244.0, 1368.0, 1437.0, 1236.0, 1354.0, 1356.0,
              1383.0, 1350.0, 1354.0, 1329.0, 1427.0, 1163.0, 1339.0, 1351.0,␣
↪1174.0, 1235.0, 1439.0, 1235.0, 1245.0, 1262.0, 1362.0, 1184.0,
              1207.0, 1359.0, 1443.0, 1205.0, 1192.0, 1364.0, 1233.0, 1281.0,␣
↪1295.0, 1357.0, 1191.0, 1329.0, 1294.0, 1334.0, 1265.0, 1207.0,
              1365.0, 1432.0, 1199.0, 1191.0, 1411.0, 1294.0, 1244.0, 1256.0,␣
↪1257.0, 1224.0, 1277.0, 1246.0, 1243.0, 1194.0, 1389.0, 1366.0,
              1282.0, 1221.0, 1255.0, 1417.0, 1358.0, 1264.0, 1205.0, 1254.0,␣
↪1276.0, 1435.0, 1335.0, 1355.0, 1337.0, 1197.0, 1423.0, 1194.0,
              1310.0, 1255.0, 1300.0, 1388.0, 1385.0, 1255.0, 1434.0, 1232.0,␣
↪1402.0, 1435.0, 1160.0, 1193.0, 1422.0, 1235.0, 1219.0, 1410.0,
              1363.0, 1361.0, 1437.0, 1407.0, 1164.0, 1392.0, 1408.0, 1196.0,␣
↪1430.0, 1264.0, 1289.0, 1434.0, 1216.0, 1340.0, 1327.0, 1230.0,
              1362.0, 1360.0, 1448.0, 1220.0, 1435.0, 1425.0, 1413.0, 1279.0,␣
↪1269.0, 1162.0, 1437.0, 1441.0, 1433.0, 1307.0, 1436.0, 1357.0,
              1437.0, 1308.0, 1207.0, 1420.0, 1338.0, 1311.0, 1328.0, 1417.0,␣
↪1394.0, 1336.0, 1160.0, 1231.0, 1422.0, 1294.0, 1434.0, 1289.0]
```

[51]:
```
import random
demand=[]
demand+=[raw_demand[i]+50 for i in range(len(raw_demand))]#random.
↪randint(150,300)
for i in range(5):
 print(demand[i])
```

```
1309.00000000000
1489.00000000000
1339.00000000000
1261.00000000000
1483.00000000000
```

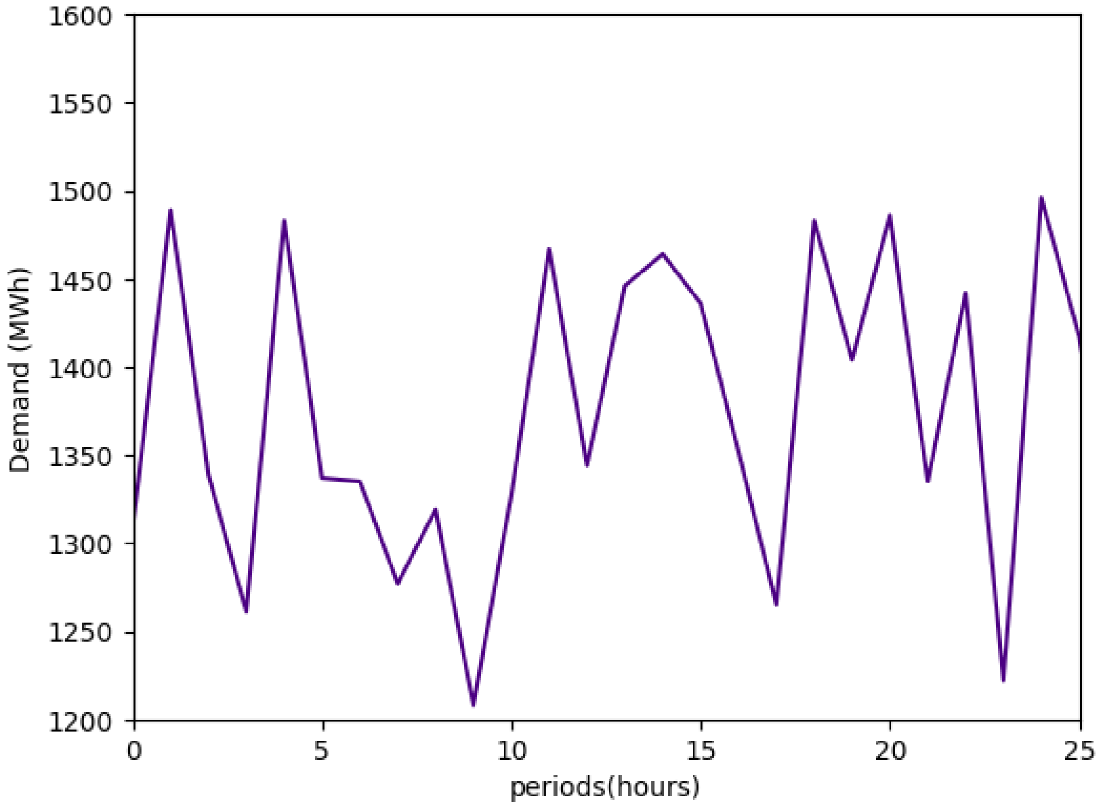
[52]:
```
import random
spanning=[]
spanning.append(41)
spanning1=[0+random.randint(30,50) for i in range(0,len(raw_demand)-1)]#random.
↪randint(150,300)
for i in range(len(raw_demand)-1):
    spanning.append(spanning1[i])
len(spanning)
```

[52]: 192

[53]:
```
demand[0]+spanning[0]
```

[53]: 1350.00000000000

[54]:
```
import matplotlib.pyplot as plt
plt.plot(demand, color='indigo')
plt.ylabel('Demand (MWh)')
plt.xlabel('periods(hours)')
plt.axis([0, 25, 1200,1600])
plt.show()
```



[55]:

```
N=10
T=50
K=N*T
R = PolynomialRing(RR, 'Gen', K)
Gen = R.gens()
#tim = S.gens()
Gen=[SR(Gen[i]) for i in srange(0,K)];
R = PolynomialRing(RR, 'gen', 20)
S = PolynomialRing(RR, 'tim', 20)
gen = R.gens()
tim = S.gens()
gen_i_time_j=[SR(gen[i])*SR(tim[j]) for i in srange(0,10) for j in srange(0,10)␣
↪];
len(gen_i_time_j)
R = PolynomialRing(RR, 'Gen', K)
Gen = R.gens()
tim = S.gens()
Gen=[SR(Gen[k]) for k in srange(0,K)];
p = MixedIntegerLinearProgram(maximization=False, solver = "GLPK")#Variables x␣
↪and y ve assume only real and non-negative values.
v = p.new_variable(real=True, nonnegative=True)
```

```python
w = p.new_variable(binary=True, nonnegative=True)
t =p.new_variable(binary=True, nonnegative=True)
s=p.new_variable(binary=True, nonnegative=True)
h = p.new_variable(integer=True, nonnegative=True)

Var=[v[Gen[k]] for k in range(K)]
var1=[h[Gen[k]] for k in range(K)]
indic_su=[w[Gen[k]] for k in range(K)]
indic_sd=[t[Gen[k]] for k in range(K)]
indic_on_off=[s[Gen[k]] for k in range(K)]
gencostt=[sum([UNITCOMIT[n][10]*Var[T*n+t] for n in range(N)]) for t in range(T)]
startcostt=[sum([UNITCOMIT[n][9]*indic_su[T*n+t] for n in range(N)]) for t in range(T)]
shutcostt=[sum([UNITCOMIT[n][0]*indic_sd[T*n+t] for n in range(N)]) for t in range(T)]
z1=sum(gencostt)
z2=sum(startcostt)
z3=sum(shutcostt)
z=z1+z2+z3
show(z)
#OBJECTIVE FUNCTION

p.set_objective(z)
tgent=[sum([Var[T*n+t]for n in range(N) ]) for t in range(T)]



#CONNSTRAINTS

#Initial condition
[p.add_constraint(Var[T*n]==UNITCOMIT[n][1]) for n in range(N)]
#power balance constraints
[p.add_constraint(tgent[t] == demand[t]+spanning[t])for t in range(T)]
#[p.add_constraint(tgent[j] >= demand[j]) for j in range(J)]
#Power generation at each time period should be within the allowable limit
[p.add_constraint(Var[T*n+t]>= (UNITCOMIT[n][4])*indic_on_off[T*n+t]) for n in range(N) for t in range(T)]
[p.add_constraint(Var[T*n+t]<= (UNITCOMIT[n][2])*indic_on_off[T*n+t]) for n in range(N) for t in range(T)]
#Ramp up and ramp down constraints
[p.add_constraint(Var[T*n+t+1]-Var[T*n+t]<= UNITCOMIT[n][7]) for n in range(N) for t in range(T-1)]
[p.add_constraint(Var[T*n+t+1]-Var[T*n+t]>= (-1)*UNITCOMIT[n][4]) for n in range(N) for t in range(T-1)]
#sum_indic_on_off=[sum([indic_on_off[10*i+j] for i in range(I)]) for j in range(J)]
```

```
#sum_indic_on_off1=[sum([indic_on_off[10*i+j] for i in range(I)]) for j in␣
↪range(7,J)]
#show(sum_indic_on_off1)
#[p.add_constraint(sum_indic_on_off[i]<=9) for i in range(10)]
#[p.add_constraint(sum([indic_on_off[i]])>=1) for i in range(8*J,10*J)]

#[p.
↪add_constraint(indic_su[J*i+j]==(indic_on_off[J*i+j]-indic_on_off[J*i+j-1]))␣
↪for i in range(I) for j in range(1,J)]
[p.add_constraint(indic_on_off[T*8+T-7]==1)]
[p.add_constraint(indic_on_off[T*3+T-4]==1)]


#print(indic_on_off[10*0])





#Minimum uptime and minimum down time constraints
#for n in range(N):
    #for t in range(1,T-2):
            #if (indic_on_off[T*n+t]-indic_on_off[T*n+t-1]==1):
                #[p.
↪add_constraint(indic_on_off[T*n+t]+indic_on_off[T*n+t+1]==2)]
           #elif (indic_on_off[T*n+t]-indic_on_off[T*n+t-1]==-1):
                #p.
↪add_constraint(indic_on_off[T*n+t]+indic_on_off[T*n+t+1]+indic_on_off[T*n+t+2]==0)

    #[p.add_constraint(indic_su[i]==1) if Var[i]-Var[i-1]==Var[i] else  p.
↪add_constraint(indic_su[i]==0)
#for i in range(100)]
for  i in range(1,len(Var)):
    if (Var[i]-Var[i-1]==Var[i]):
        p.add_constraint(indic_su[i]==1)
    else :
        p.add_constraint(indic_su[i]==0)



Set1=[Var[T*n+t] for n in range(N) for t in range(T)]
Set2=[indic_su[T*n+t] for n in range(N) for t in range(T)]
Set3=[indic_on_off[T*n+t] for n in range(N) for t in range(T)]
Set4=[indic_sd[T*n+t] for n in range(N) for t in range(T)]
Z1=(p.solve())
#Z2=(p.solve(2))
#Z3=(p.solve(3))
Val1=[p.get_values(Set1[k]) for k in range(K)]
```

```
Val2=[p.get_values(Set2[k]) for k in range(K)]
Val3=[p.get_values(Set3[k]) for k in range(K)]
Val4=[p.get_values(Set4[k]) for k in range(K)]
D=[[Val1[T*n+t]for n in range(N)] for t in range(T)]

for n in range(N):
    Val2[T*n]=0
for n in range(0,N):
    for t in range(1,T):
        if (Val1[T*n+t]-Val1[T*n+t-1]==Val1[T*n+t] and Val1[T*n+t]>0):
            Val2[T*n+t]=1
        else:
            Val2[T*n+t]=0
for n in range(0,N):
    for t in range(1,T):
        if (Val1[T*n+t]-Val1[T*n+t-1]==(-1*Val1[T*n+t-1]) and Val1[T*n+t-1]>0):
            Val4[T*n+t]=1
        else:
            Val4[T*n+t]=0

#startcostt=sum([sum([UNITCOMIT[i][9]*Val2[10*i+j] for i in range(I)]) for j in␣
↪range(J)])
#shutcostt=sum([sum([UNITCOMIT[i][9]*Val4[10*i+j] for i in range(I)]) for j in␣
↪range(J)])
#Minimum uptime
Fn=[min([T,t+UNITCOMIT[n][5]])for n in range(N) for t in range(T)]
for n in range(1,N):
    for t in range(1,T-1):
        if (Val2[T*n+t]==1):
            s=t
            #print(s)
            [p.add_constraint(indic_on_off[T*n+s]==1) for s in␣
↪range(t,Fn[T*n+t])]
#Minimum downtime
Hn=[min([T,t+UNITCOMIT[n][3]])for n in range(N) for t in range(T)]
#print(Hn)
for n in range(N):
    for t in range(T-1):
        if (Val4[T*n+t]==1):
            l=t
            [p.add_constraint(indic_on_off[T*n+l]==0) for l in␣
↪range(t,Hn[T*n+t])]
#show(Z1)
#show(Z2)
#show(Z3)
```

```
Z=Z1
show(Z)

Schedule=matrix(D)
show(Schedule)
E=[[Val2[T*n+t]for n in range(N)] for t in range(T)]
show(matrix(E))
F=matrix([[Val3[T*n+t]for n in range(N)] for t in range(T)])
show(F)
G=matrix([[Val4[T*n+t]for n in range(N)] for t in range(T)])
show(G)
gencost=[sum([UNITCOMIT[n][10]*Val1[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
startcost=[sum([UNITCOMIT[n][9]*Val2[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
shutcost=[sum([UNITCOMIT[n][0]*Val4[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
operationcost=[gencost[i]+startcost[i]+shutcost[i] for i in range(10) ]
AVTIME=[sum([Val3[T*n+t] for t in range(T)]) for n in range(N)]
AVOUT=[(1/AVTIME[n])*sum([Val1[T*n+t] for t in range(T)]) for n in range(N)]
```

```
43.536*x_0 + 43.536*x_1 + 43.536*x_2 + 43.536*x_3 + 43.536*x_4 + 43.536*x_5 + 43.
↪536*x_6 + 43.536*x_7 + 43.536*x_8 + 43.536*x_9 + 43.536*x_10 + 43.536*x_11 +␣
↪43.536*x_12 + 43.536*x_13 + 43.536*x_14 + 43.536*x_15 + 43.536*x_16 + 43.
↪536*x_17 + 43.536*x_18 + 43.536*x_19 + 43.536*x_20 + 43.536*x_21 + 43.536*x_22␣
↪+ 43.536*x_23 + 43.536*x_24 + 43.536*x_25 + 43.536*x_26 + 43.536*x_27 + 43.
↪536*x_28 + 43.536*x_29 + 43.536*x_30 + 43.536*x_31 + 43.536*x_32 + 43.536*x_33␣
↪+ 43.536*x_34 + 43.536*x_35 + 43.536*x_36 + 43.536*x_37 + 43.536*x_38 + 43.
↪536*x_39 + 43.536*x_40 + 43.536*x_41 + 43.536*x_42 + 43.536*x_43 + 43.536*x_44␣
↪+ 43.536*x_45 + 43.536*x_46 + 43.536*x_47 + 43.536*x_48 + 43.536*x_49 + 42.
↪985*x_50 + 42.985*x_51 + 42.985*x_52 + 42.985*x_53 + 42.985*x_54 + 42.985*x_55␣
↪+ 42.985*x_56 + 42.985*x_57 + 42.985*x_58 + 42.985*x_59 + 42.985*x_60 + 42.
↪985*x_61 + 42.985*x_62 + 42.985*x_63 + 42.985*x_64 + 42.985*x_65 + 42.985*x_66␣
↪+ 42.985*x_67 + 42.985*x_68 + 42.985*x_69 + 42.985*x_70 + 42.985*x_71 + 42.
↪985*x_72 + 42.985*x_73 + 42.985*x_74 + 42.985*x_75 + 42.985*x_76 + 42.985*x_77␣
↪+ 42.985*x_78 + 42.985*x_79 + 42.985*x_80 + 42.985*x_81 + 42.985*x_82 + 42.
↪985*x_83 + 42.985*x_84 + 42.985*x_85 + 42.985*x_86 + 42.985*x_87 + 42.985*x_88␣
↪+ 42.985*x_89 + 42.985*x_90 + 42.985*x_91 + 42.985*x_92 + 42.985*x_93 + 42.
↪985*x_94 + 42.985*x_95 + 42.985*x_96 + 42.985*x_97 + 42.985*x_98 + 42.985*x_99␣
↪+ 50.5*x_100 + 50.5*x_101 + 50.5*x_102 + 50.5*x_103 + 50.5*x_104 + 50.5*x_105␣
↪+ 50.5*x_106 + 50.5*x_107 + 50.5*x_108 + 50.5*x_109 + 50.5*x_110 + 50.5*x_111␣
↪+ 50.5*x_112 + 50.5*x_113 + 50.5*x_114 + 50.5*x_115 + 50.5*x_116 + 50.5*x_117␣
↪+ 50.5*x_118 + 50.5*x_119 + 50.5*x_120 + 50.5*x_121 + 50.5*x_122 + 50.5*x_123␣
↪+ 50.5*x_124 + 50.5*x_125 + 50.5*x_126 + 50.5*x_127 + 50.5*x_128 + 50.5*x_129␣
↪+ 50.5*x_130 + 50.5*x_131 + 50.5*x_132 + 50.5*x_133 + 50.5*x_134 + 50.5*x_135␣
↪+ 50.5*x_136 + 50.5*x_137 + 50.5*x_138 + 50.5*x_139 + 50.5*x_140 + 50.5*x_141␣
↪+ 50.5*x_142 + 50.5*x_143 + 50.5*x_144 + 50.5*x_145 + 50.5*x_146 + 50.5*x_147␣
↪+ 50.5*x_148 + 50.5*x_149 + 59*x_150 + 59*x_151 + 59*x_152 + 59*x_153 +␣
↪59*x_154 + 59*x_155 + 59*x_156 + 59*x_157 + 59*x_158 + 59*x_159 + 59*x_160 +␣
↪59*x_161 + 59*x_162 + 59*x_163 + 59*x_164 + 59*x_165 + 59*x_166 + 59*x_167 +␣
↪59*x_168 + 59*x_169 + 59*x_170 + 59*x_171 + 59*x_172 + 59*x_173 + 59*x_174 +␣
↪59*x_175 + 59*x_176 + 59*x_177 + 59*x_178 + 59*x_179 + 59*x_180 + 59*x_181 +␣
↪59*x_182 + 59*x_183 + 59*x_184 + 59*x_185 + 59*x_186 + 59*x_187 + 59*x_188 +␣
↪59*x_189 + 59*x_190 + 59*x_191 + 59*x_192 + 59*x_193 + 59*x_194 + 59*x_195 +␣
↪59*x_196 + 59*x_197 + 59*x_198 + 59*x_199 + 37.146*x_200 + 37.146*x_201 + 37.
↪146*x_202 + 37.146*x_203 + 37.146*x_204 + 37.146*x_205 + 37.146*x_206 + 37.
↪146*x_207 + 37.146*x_208 + 37.146*x_209 + 37.146*x_210 + 37.146*x_211 + 37.
↪146*x_212 + 37.146*x_213 + 37.146*x_214 + 37.146*x_215 + 37.146*x_216 + 37.
↪146*x_217 + 37.146*x_218 + 37.146*x_219 + 37.146*x_220 + 37.146*x_221 + 37.
↪146*x_222 + 37.146*x_223 + 37.146*x_224 + 37.146*x_225 + 37.146*x_226 + 37.
↪146*x_227 + 37.146*x_228 + 37.146*x_229 + 37.146*x_230 + 37.146*x_231 + 37.
↪146*x_232 + 37.146*x_233 + 37.146*x_234 + 37.146*x_235 + 37.146*x_236 + 37.
↪146*x_237 + 37.146*x_238 + 37.146*x_239 + 37.146*x_240 + 37.146*x_241 + 37.
↪146*x_242 + 37.146*x_243 + 37.146*x_244 + 37.146*x_245 + 37.146*x_246 + 37.
↪146*x_247 + 37.146*x_248 + 37.146*x_249 + 44.84*x_250 + 44.84*x_251 + 44.
↪84*x_252 + 44.84*x_253 + 44.84*x_254 + 44.84*x_255 + 44.84*x_256 + 44.84*x_257␣
↪+ 44.84*x_258 + 44.84*x_259 + 44.84*x_260 + 44.84*x_261 + 44.84*x_262 + 44.
↪84*x_263 + 44.84*x_264 + 44.84*x_265 + 44.84*x_266 + 44.84*x_267 + 44.84*x_268␣
↪+ 44.84*x_269 + 44.84*x_270 + 44.84*x_271 + 44.84*x_272 + 44.84*x_273 + 44.
↪84*x_274 + 44.84*x_275 + 44.84*x_276 + 44.84*x_277 + 44.84*x_278 + 44.84*x_279␣
↪+ 44.84*x_280 + 44.84*x_281 + 44.84*x_282 + 44.84*x_283 + 44.84*x_284 + 44.
↪84*x_285 + 44.84*x_286 + 44.84*x_287 + 44.84*x_288 + 44.84*x_289 + 44.84*x_290␣
↪+ 44.84*x_291 + 44.84*x_292 + 44.84*x_293 + 44.84*x_294 + 44.84*x_295 + 44.
↪84*x_296 + 44.84*x_297 + 44.84*x_298 + 44.84*x_299 + 40.222*x_300 + 40.
```

```
3962087.8189999885
```

```
50 x 10 dense matrix over Real Double Field (use the '.str()' method to see the␣
↪entries)
```

```
50 x 10 dense matrix over Integer Ring (use the '.str()' method to see the␣
↪entries)
```

```
50 x 10 dense matrix over Real Double Field (use the '.str()' method to see the␣
↪entries)
```

```
50 x 10 dense matrix over Real Double Field (use the '.str()' method to see the␣
↪entries)
```

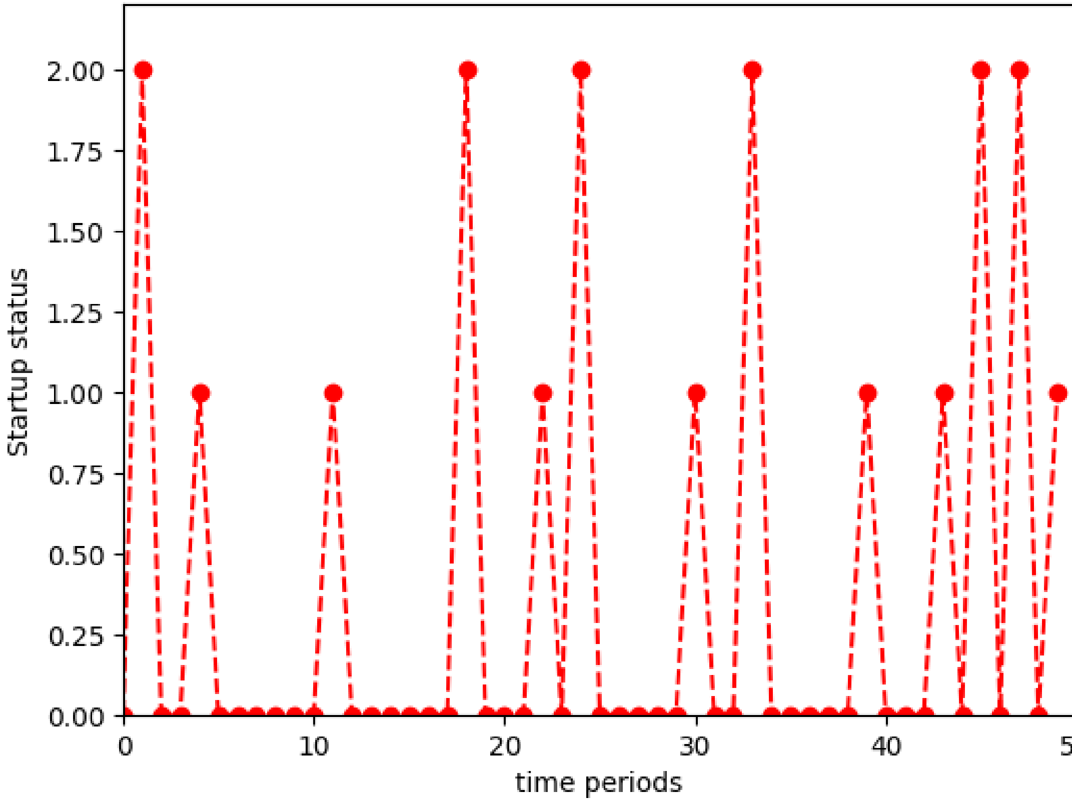
[56]:

```
D=[[Val1[T*n+t]for t in range(T)] for n in range(N)]
k=[[Val1[T*n+t]for n in range(N)] for t in range(T)]
show(k[0][0])
bars = [sum([k[t][n] for n in range(1,N)]) for t in range(T)]
```

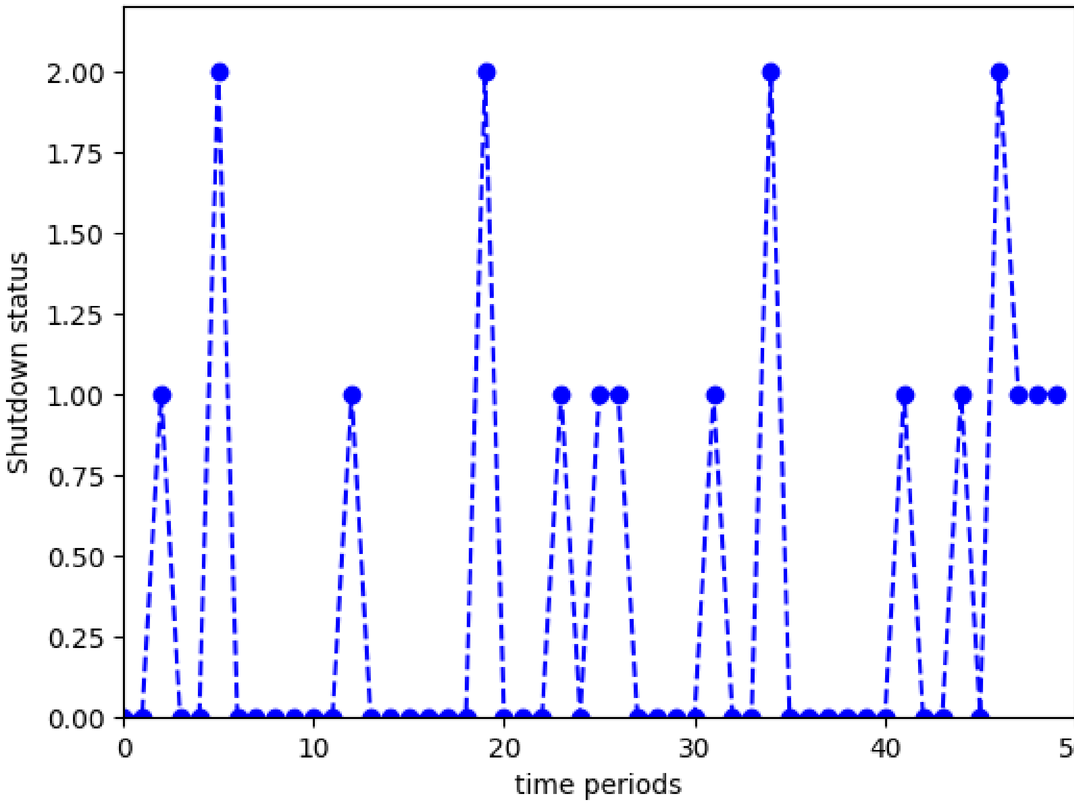
```
400.0
```

[57]:

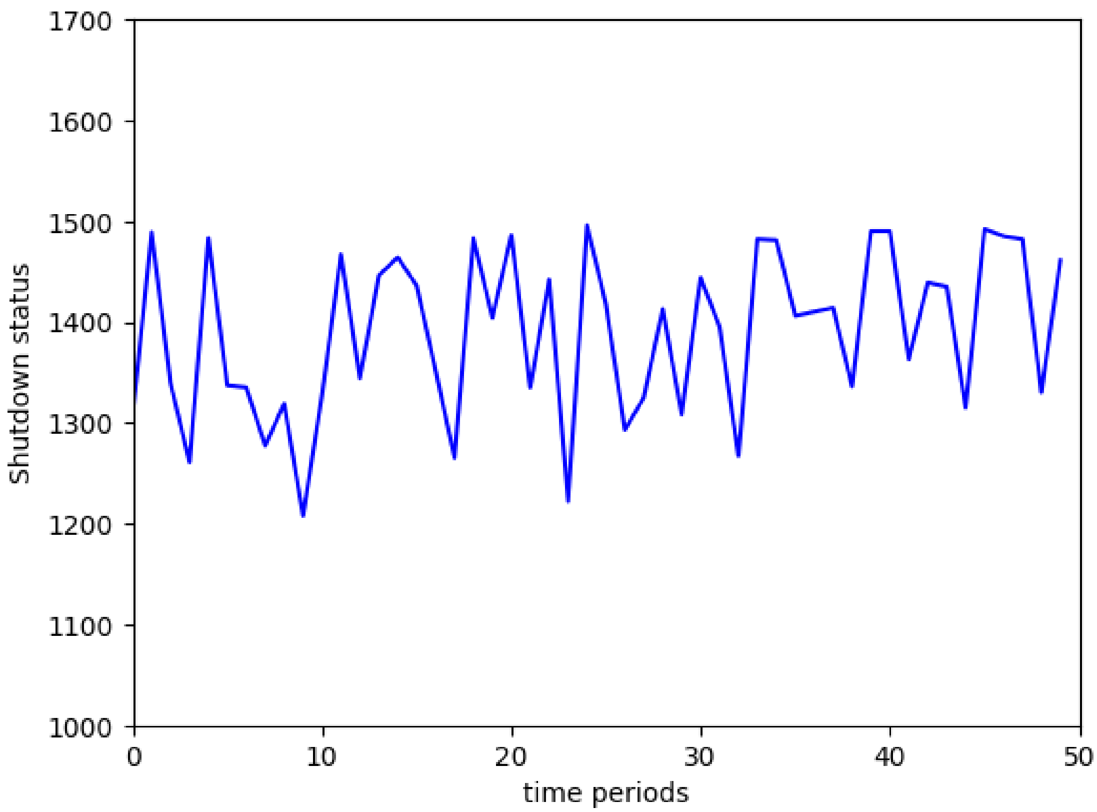
```
startup=[sum(E[t]) for t in range(T)]
import matplotlib.pyplot as plt
a=plt.plot(startup,'bo', color='red')
b=plt.plot(startup,'r--', color='red')
plt.ylabel('Startup status')
plt.xlabel('time periods')
plt.axis([0,T, 0,2.2])
plt.show()
```




```
[58]: shutdown=[sum(G[t]) for t in range(T)]
      #shutdown[44]=1
      #shutdown[46]=0
      #l=[startup[i] for i in range(45,50)]
      #show(l)
      import matplotlib.pyplot as plt
      plt.plot(shutdown, 'bo', color='blue')
      plt.plot(shutdown, 'r--', color='blue')
      plt.ylabel('Shutdown status')
      plt.xlabel('time periods')
      plt.axis([0,T, 0,2.2])
      plt.show()
```



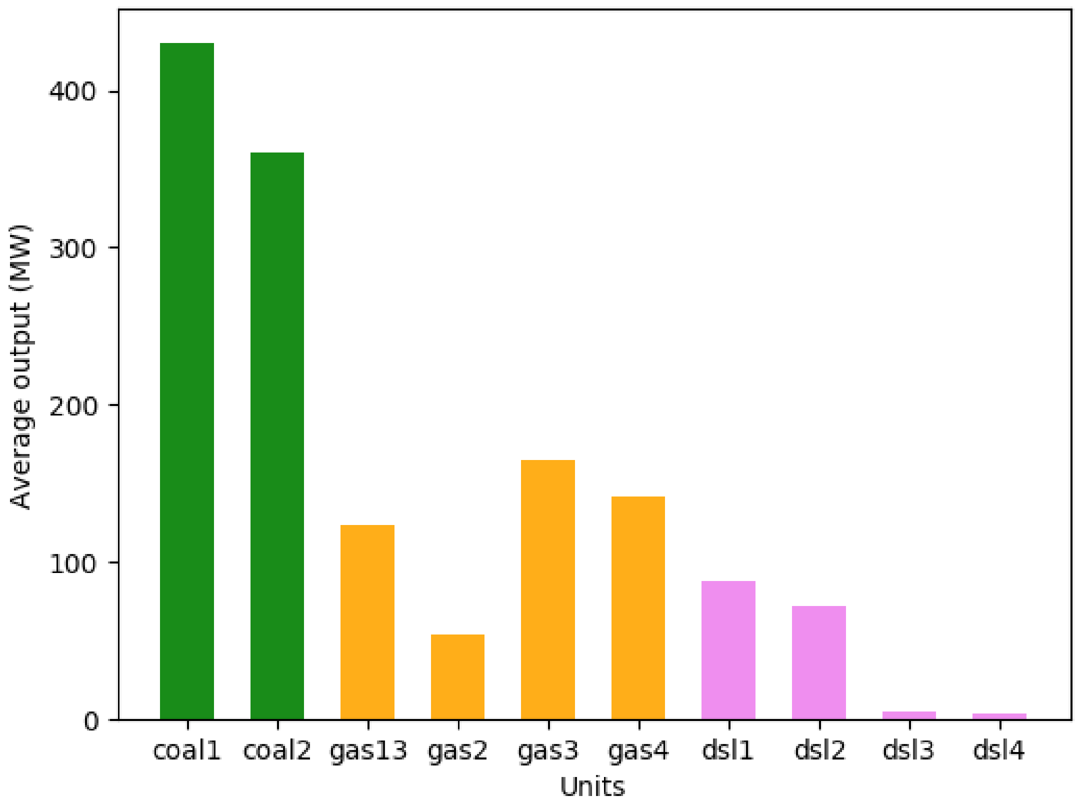
[59]:
```
d=[demand[i] for i in range(T)]
plt.plot(d, color='blue')
plt.ylabel('Shutdown status')
plt.xlabel('time periods')
plt.axis([0,T, 1000,1700])
plt.show()
```



[60]:
```
P=AVOUT
show(P)
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('coal1', 'coal2', 'gas13', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4',)
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.9,width=0.6,␣
 ↪color=['green','green','orange','orange','orange','orange','violet','violet','violet','viol
plt.xticks(y_pos, objects)
plt.ylabel('Average output (MW)')
plt.xlabel('Units')
plt.title('')
plt.show()
```

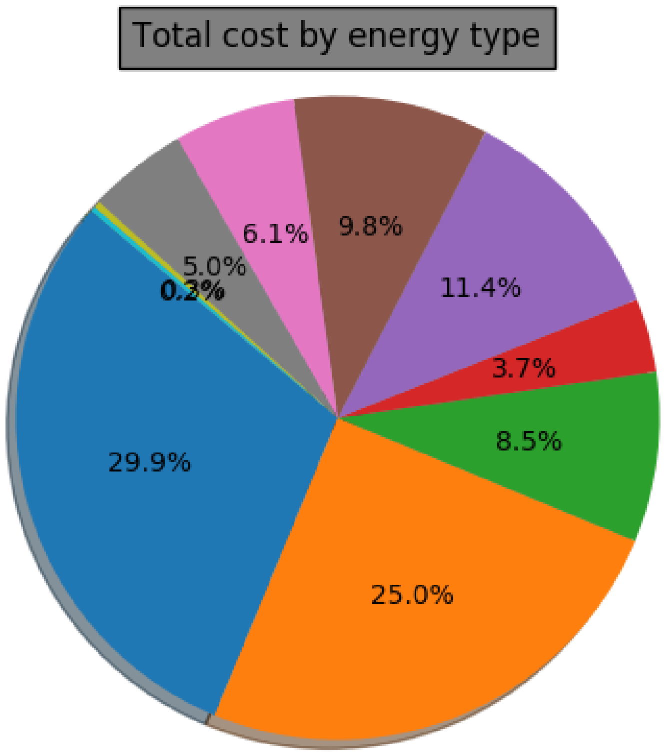
```
[430.43199999999996,
 360.42800000000005,
```

```
123.052,
53.34893617021277,
164.48,
140.956,
88.0530612244898,
72.02000000000001,
4.933333333333336,
3.5999999999999988]
```



[61]:
```
import matplotlib.pyplot as plt
coal=sum([gencost[i]+startcost[i]+shutcost[i] for i in range(0,2)])
gas =sum([gencost[i]+startcost[i]+shutcost[i] for i in range(2,6)])
diesel =sum([gencost[i]+startcost[i]+shutcost[i] for i in range(6,10)])
cost=AVOUT
# Data to plot
labels = 'coal', 'gas', 'diesel'
#cost=[coal,gas,diesel]
#colors = ['gold', 'yellowgreen', 'lightcoral']
explode = ( 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0)  # explode 1st slice

# Plot
```

```
plt.pie(cost, explode=explode,  autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Total cost by energy type', bbox={'facecolor':'0.5', 'pad':5})

plt.axis('equal')
plt.show()
show(cost)
```

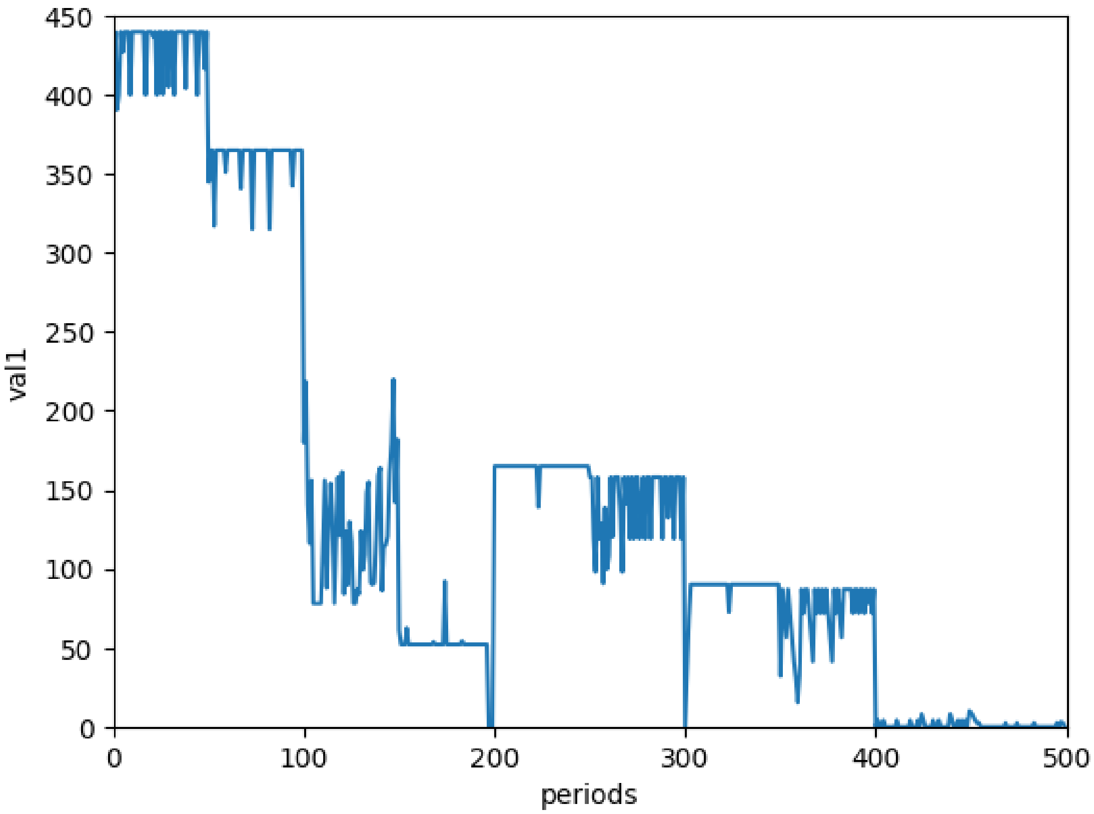



```
[430.43199999999996,
 360.42800000000005,
 123.052,
 53.34893617021277,
 164.48,
 140.956,
 88.0530612244898,
 72.02000000000001,
 4.933333333333336,
 3.5999999999999988]
```

[62]:

```
# import matplotlib.pyplot as plt
plt.plot(Val1)
plt.ylabel('val1')
plt.xlabel('periods')
```

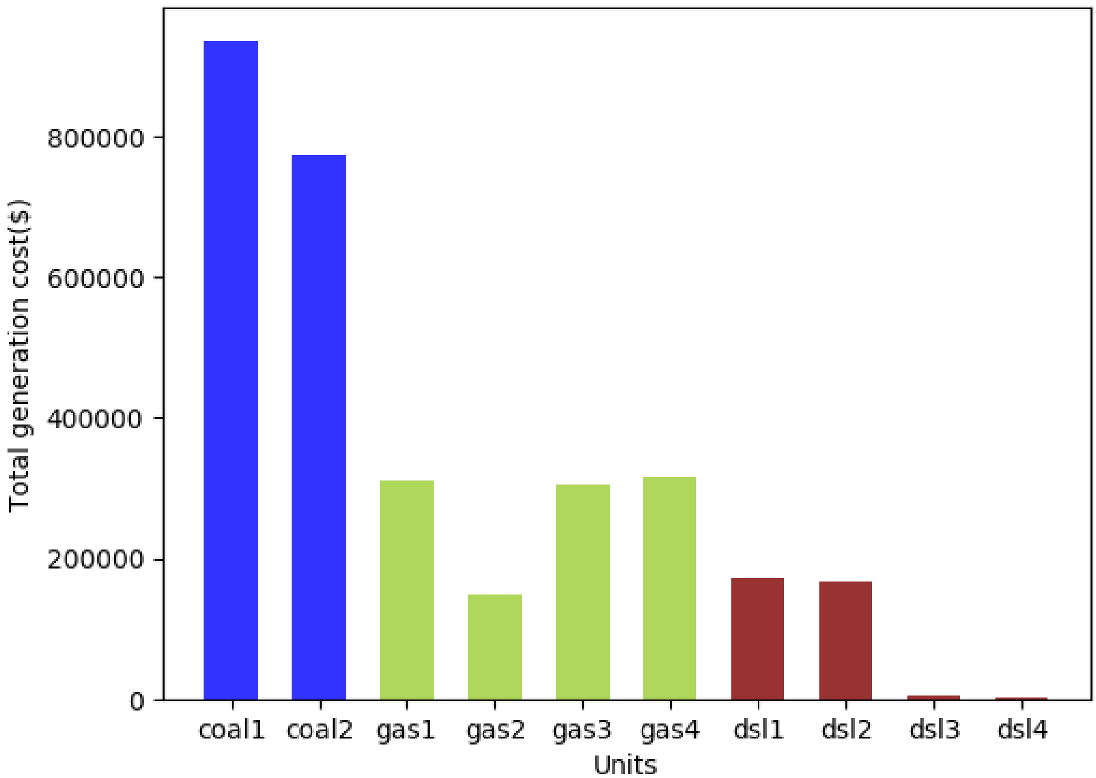
```
plt.axis([0,K, 0,450])
plt.show()
```



[63]:

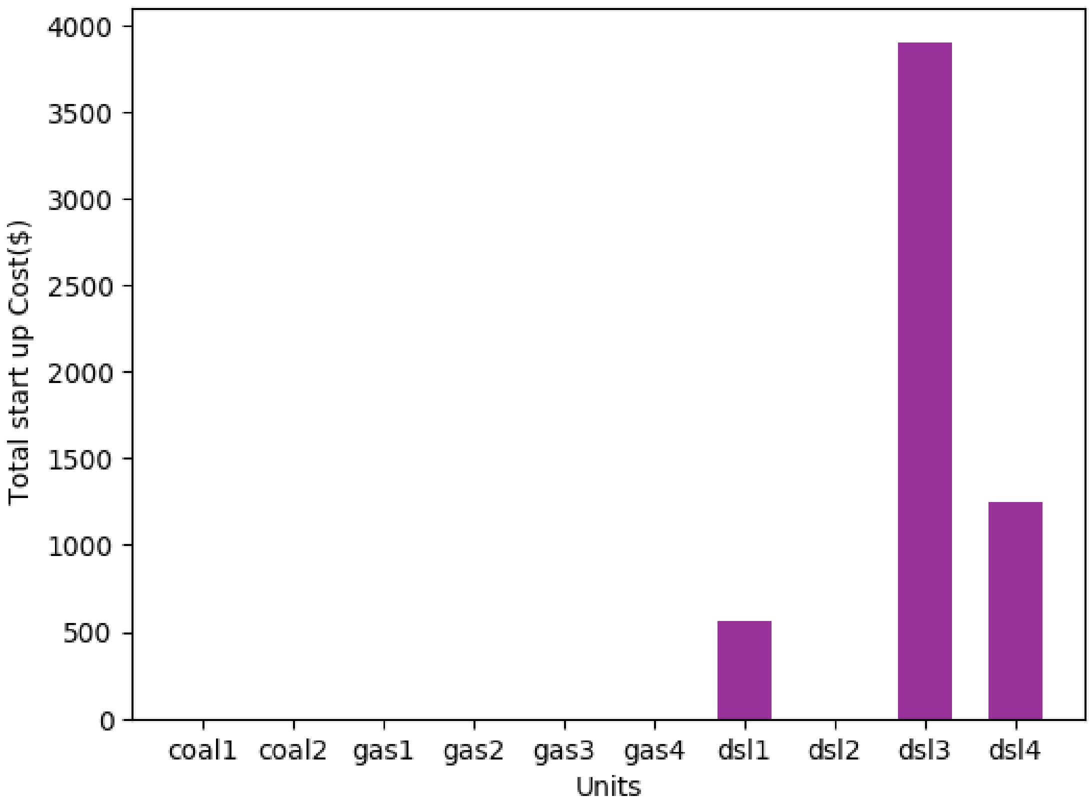
```
#P=[sum([Val1[J*i+j] for j in range(J)]) for i in range(I)]
gencost=[sum([UNITCOMIT[n][10]*Val1[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
startcost=[sum([UNITCOMIT[n][9]*Val2[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
shutcost=[sum([UNITCOMIT[n][0]*Val4[T*n+t] for t in range(T)]) for n in␣
 ↪range(N)]
P=gencost
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance = P
```

```
plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
 ↪color=['blue','blue','yellowgreen','yellowgreen','yellowgreen','yellowgreen','maroon','maro
plt.xticks(y_pos, objects)
plt.ylabel('Total generation cost($)')
plt.xlabel('Units')
plt.title('')
plt.show()
```



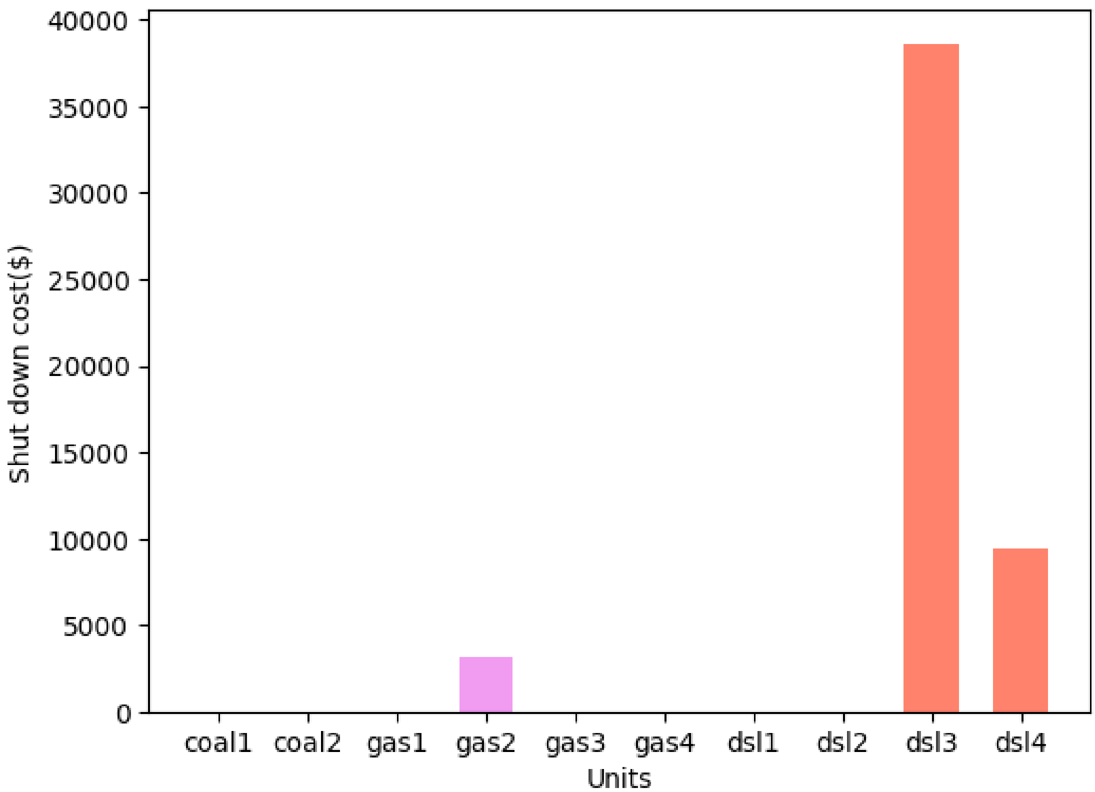
[64]:
```
#P=[sum([Val1[J*i+j] for j in range(J)]) for i in range(I)]
P=startcost
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance = P
barWidth = 0.2
```

```
plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
 ↪color=['sienna','sienna','green','green','green','green','purple','purple','purple','purple
plt.xticks(y_pos, objects)
plt.ylabel(' Total start up Cost($)')
plt.xlabel('Units')
plt.title('')
plt.show()
```



[65]:
```
P=shutcost
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
shut=[0,0,0,1,0,0,0,0,1,7]
objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
 ↪color=['black','black','violet','violet','violet','violet','tomato','tomato','tomato','toma
```

```
plt.xticks(y_pos, objects)
plt.ylabel('Shut down cost($)')
plt.xlabel('Units')
plt.title('')
plt.show()
```



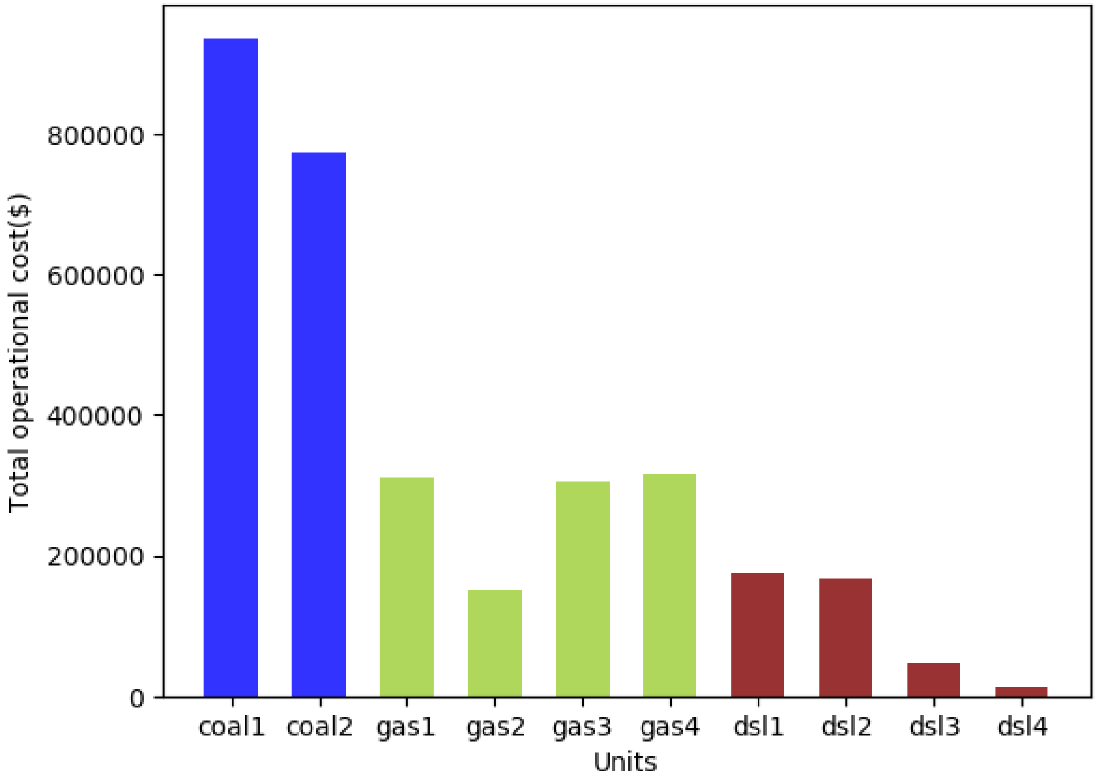
[ ]:

```

```

[66]:

```
#P=[sum([Val1[J*i+j] for j in range(J)]) for i in range(I)]
P=operationcost
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
 ↪color=['blue','blue','yellowgreen','yellowgreen','yellowgreen','yellowgreen','maroon','maro
```

```
plt.xticks(y_pos, objects)
plt.ylabel('Total operational cost($)')
plt.xlabel('Units')
plt.title('')
plt.show()
```

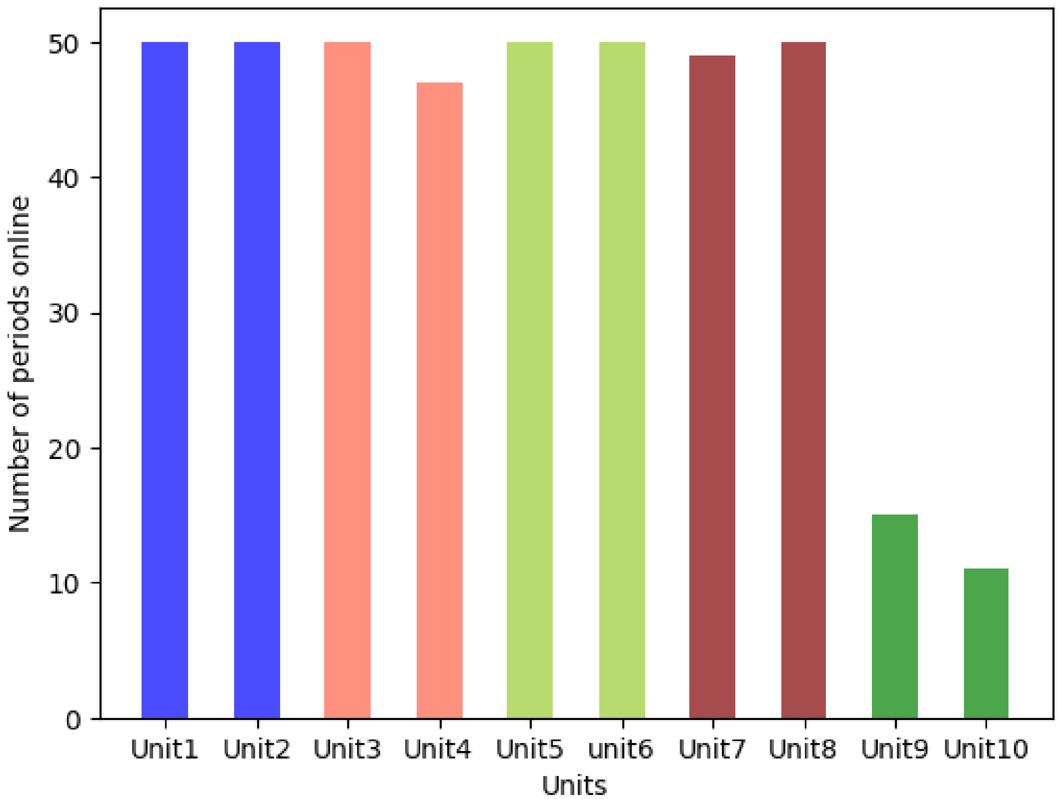


[67]:

```
P=[sum([Val3[T*n+t] for t in range(T)]) for n in range(N)]
show(P)
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('Unit1', 'Unit2', 'Unit3', 'Unit4', 'Unit5', 'unit6','Unit7',␣
↪'Unit8', 'Unit9', 'Unit10',)
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.7,width=0.5,␣
↪color=['blue','blue','tomato','tomato','yellowgreen','yellowgreen','maroon','maroon','green
plt.xticks(y_pos, objects)
plt.ylabel('Number of periods online')
```

```
plt.xlabel('Units')
plt.title('')
plt.show()
```

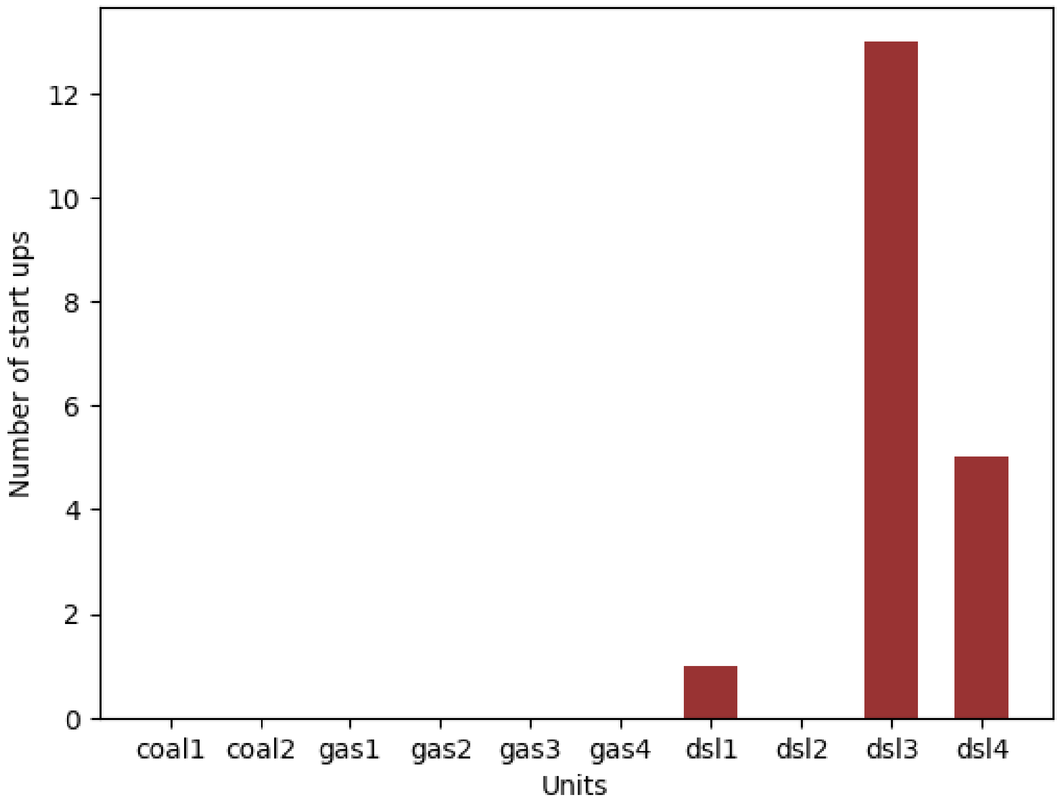
```
[50.0, 50.0, 50.0, 47.0, 50.0, 50.0, 49.0, 50.0, 15.0, 11.0]
```



[ ]:

```
[68]: P=[sum([Val2[T*n+t] for t in range(T)]) for n in range(N)]
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
 ↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
 ↪color=['blue','blue','yellowgreen','yellowgreen','yellowgreen','yellowgreen','maroon','maro
plt.xticks(y_pos, objects)
```

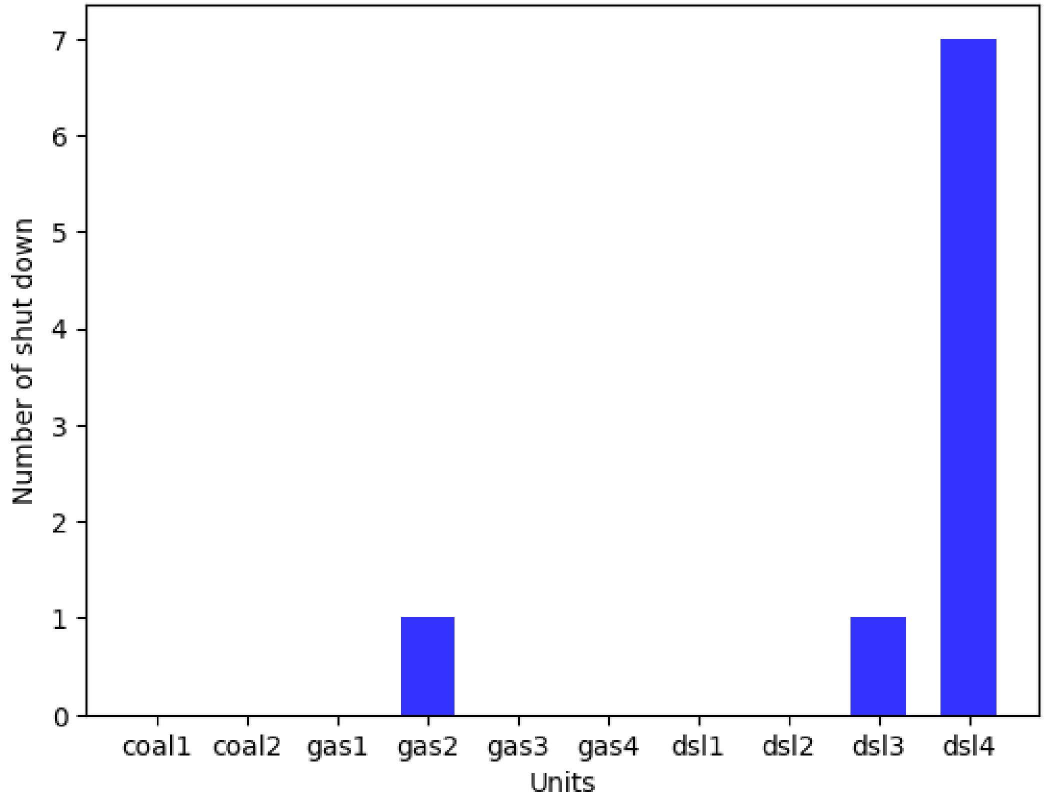
```
plt.ylabel('Number of start ups')
plt.xlabel('Units')
plt.title('')
plt.show()
```



[69]:

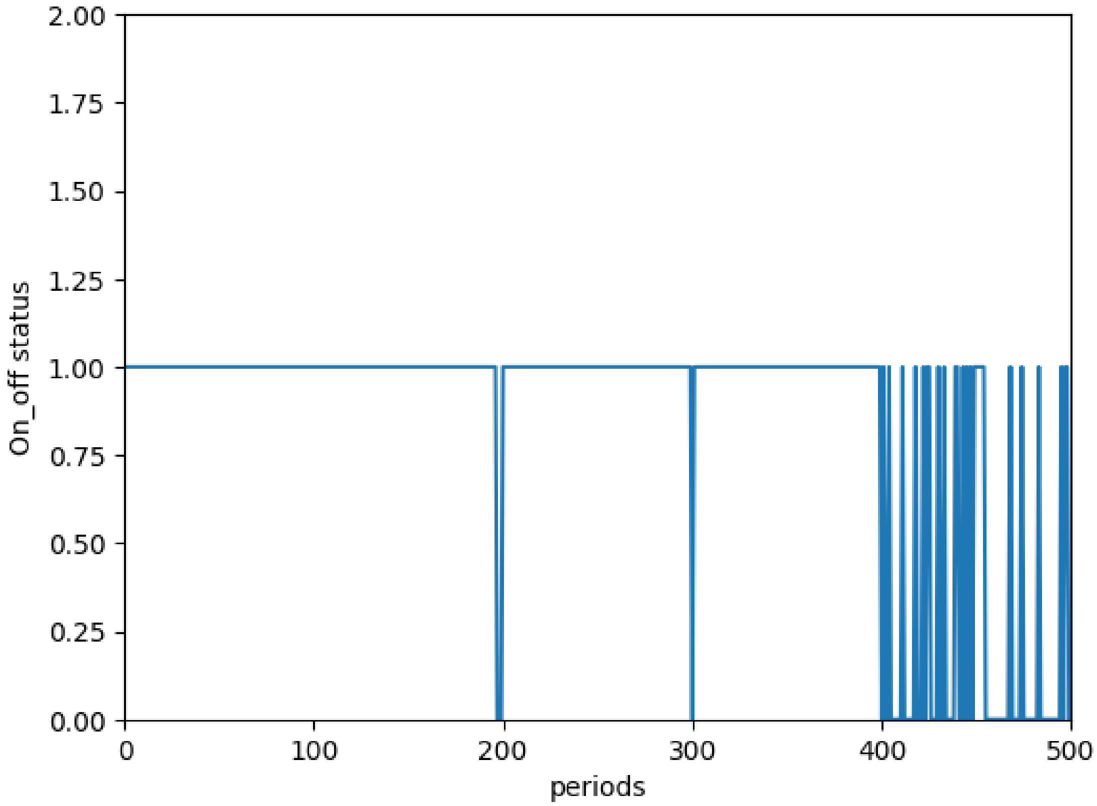
```
#P=[sum([Val4[T*n+t] for t in range(T)]) for n in range(N)]
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
objects = ('coal1', 'coal2', 'gas1', 'gas2', 'gas3', 'gas4','dsl1', 'dsl2',␣
↪'dsl3', 'dsl4')
y_pos = np.arange(len(objects))
performance =shut

plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.6,␣
↪color=['blue','blue','yellowgreen','BLUE','yellowgreen','yellowgreen','maroon','blue','blue
plt.xticks(y_pos, objects)
plt.ylabel('Number of shut down')
plt.xlabel('Units')
plt.title('')
```

```
plt.show()
```



[70]:

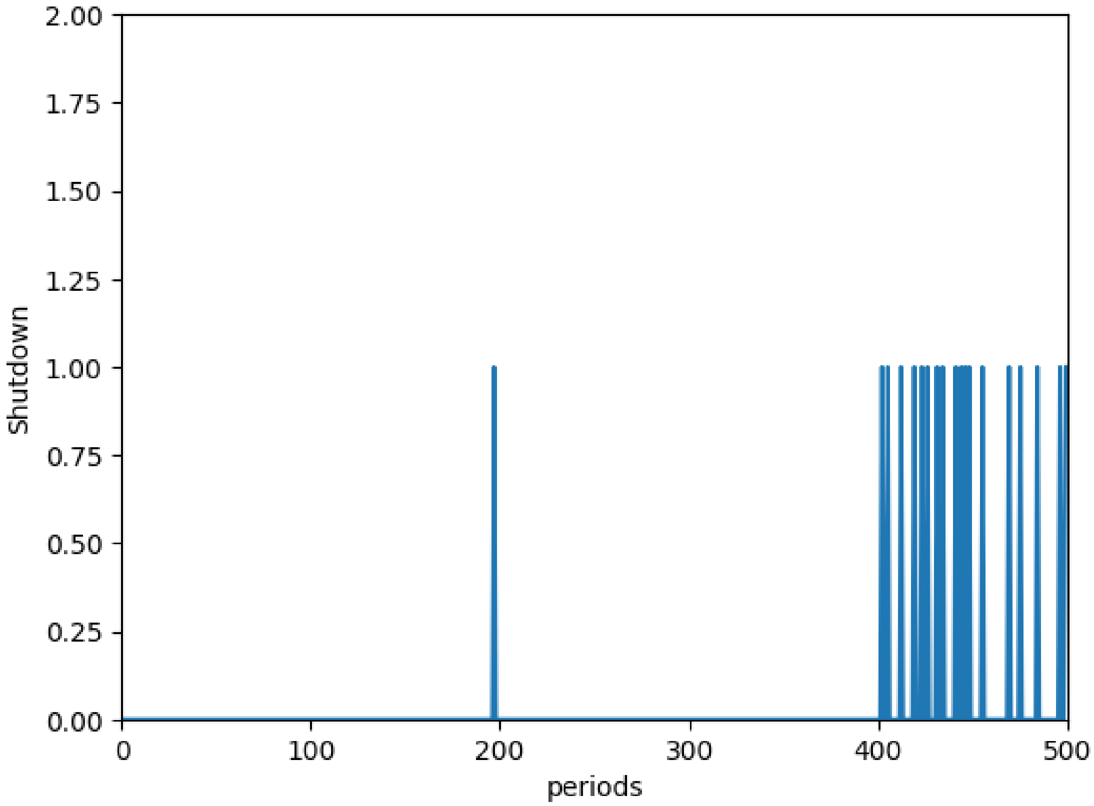
```
import matplotlib.pyplot as plt
plt.plot(Val3)
plt.ylabel('On_off status')
plt.xlabel('periods')
plt.axis([0,K, 0,2])
plt.show()
```



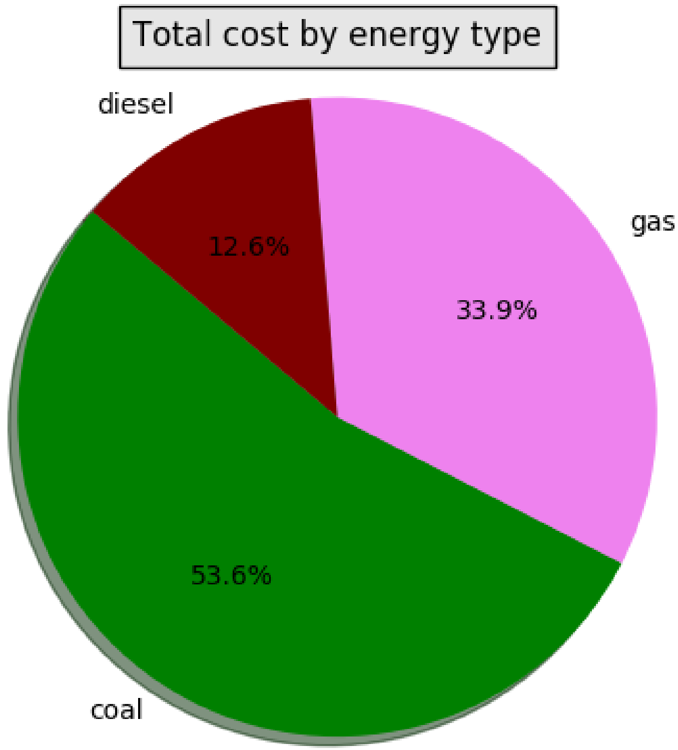
[71]:
```
k=[indic_on_off[10*i] for i in range(10)]
print(sum(k))
```

```
x_2000 + x_2010 + x_2020 + x_2030 + x_2040 + x_2050 + x_2060 + x_2070 + x_2080 +
x_2090
```

[72]:
```
sage.misc.sage_timeit_class.SageTimeit
import matplotlib.pyplot as plt
plt.plot(Val4)
plt.ylabel('Shutdown')
plt.xlabel('periods')
plt.axis([0,K, 0,2])
plt.show()
```



[73]:
```
import matplotlib.pyplot as plt
coal=sum([gencost[i]+startcost[i]+shutcost[i] for i in range(0,2)])
gas =sum([gencost[i]+startcost[i]+shutcost[i] for i in range(2,6)])
diesel =sum([gencost[i]+startcost[i]+shutcost[i] for i in range(6,10)])
cost=[coal,gas,diesel]
# Data to plot
labels = 'coal', 'gas', 'diesel'
cost=[coal,gas,diesel]
#colors = ['gold', 'yellowgreen', 'lightcoral']
explode = ( 0, 0, 0)  # explode 1st slice

# Plot
plt.pie(cost, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Total cost by energy type', bbox={'facecolor':'0.9', 'pad':5})

plt.axis('equal')
plt.show()
show(cost)
```

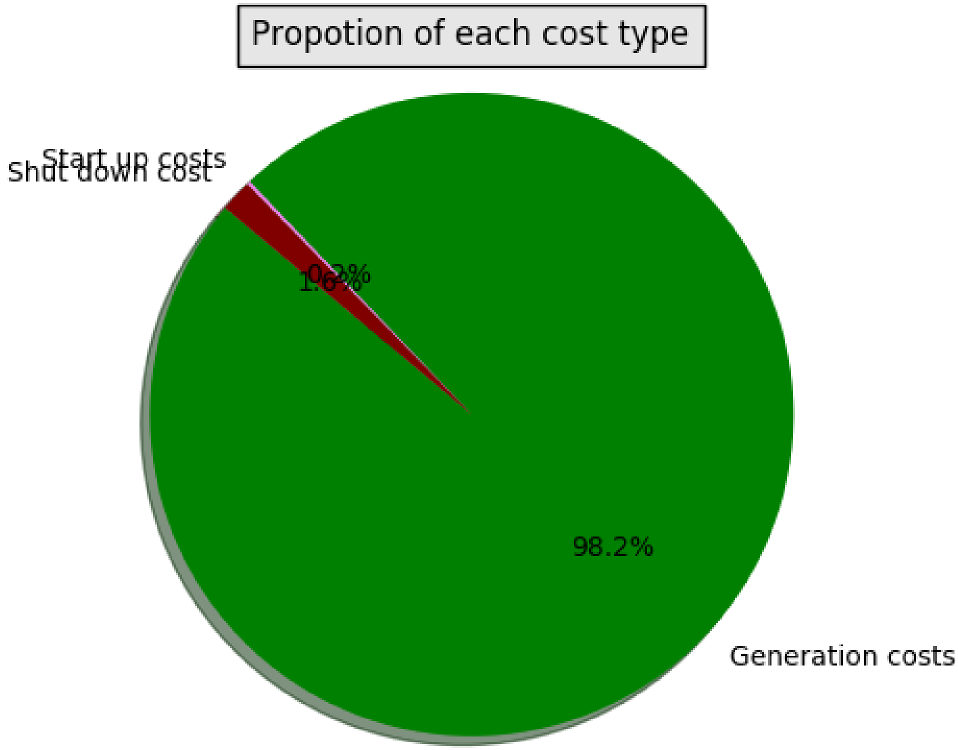



```
[1.71161425660000e6, 1.08328695600000e6, 401228.606400000]
```

[74]:

```
import matplotlib.pyplot as plt
Tgencost= sum(gencost)
Tstartcost= sum(startcost)
Tshutcost= sum(shutcost)
#gas =sum([gencost[i]+startcost[i]+fixedcost[i] for i in range(2,6)])
#diesel =sum([gencost[i]+startcost[i]+fixedcost[i] for i in range(6,10)])
cost=[coal,gas,diesel]
# Data to plot
labels = 'Generation costs', 'Start up costs', 'Shut down cost'
cost=[Tgencost,Tstartcost,Tshutcost]
colors = ['green', 'violet', 'maroon']
explode = ( 0,0 , 0)  # explode 1st slice

# Plot
plt.pie(cost, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Propotion of each cost type', bbox={'facecolor':'0.9', 'pad':5})

plt.axis('equal')
plt.show()
```

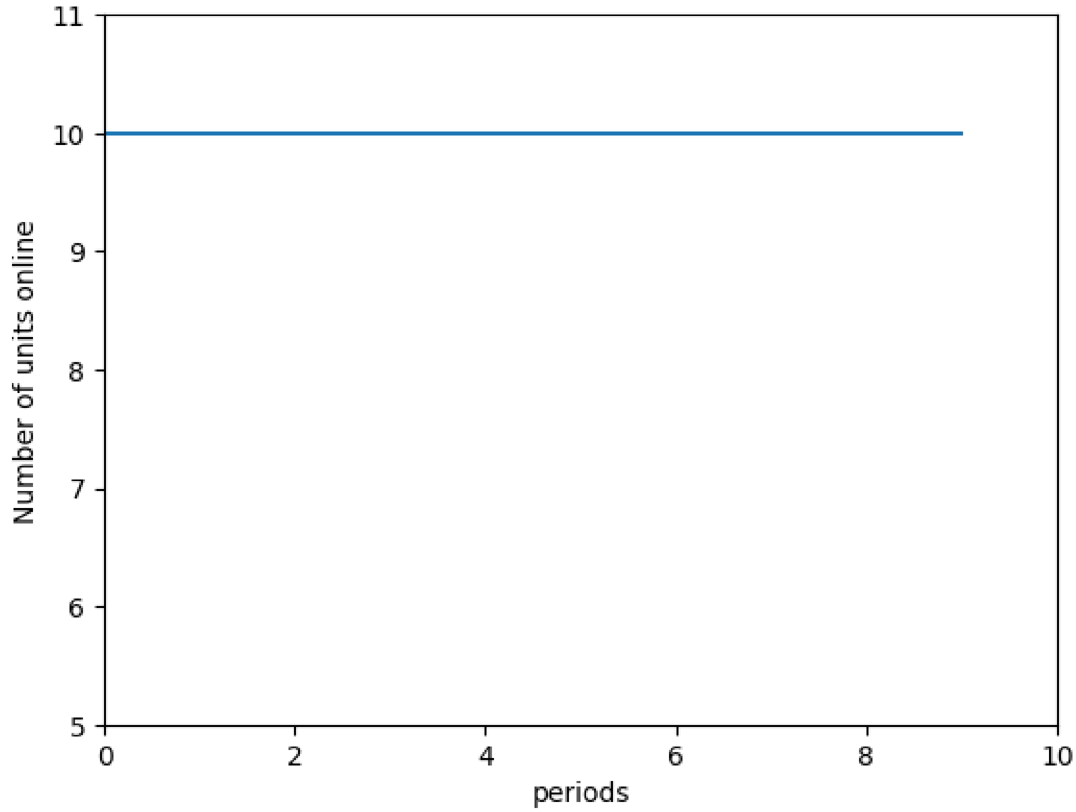
```
show(cost)
```



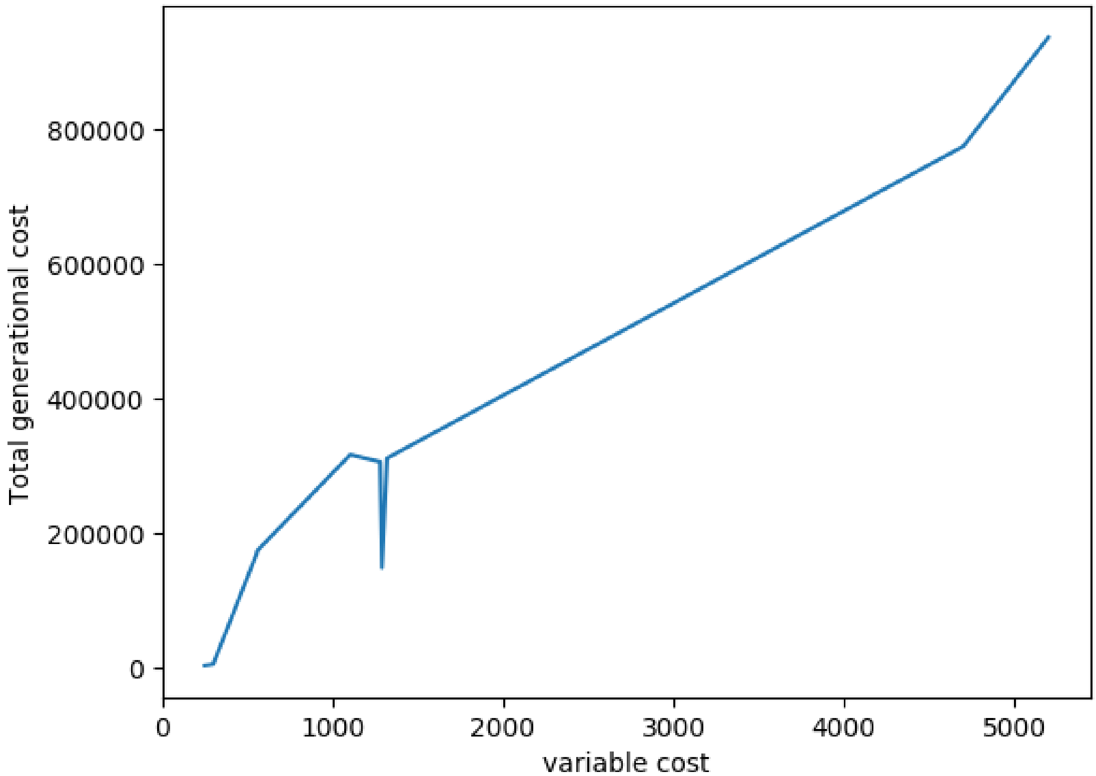
[3.13928781900000e6, 5710.00000000000, 51132.0000000000]

[75]:
```
N_online=[sum([Val3[10*i+j] for i in range(10)]) for j in range(10)]
show(N_online)
import matplotlib.pyplot as plt
plt.plot(N_online)
plt.ylabel('Number of units online')
plt.xlabel('periods')
plt.axis([0,10, 5,11])
plt.show()
```

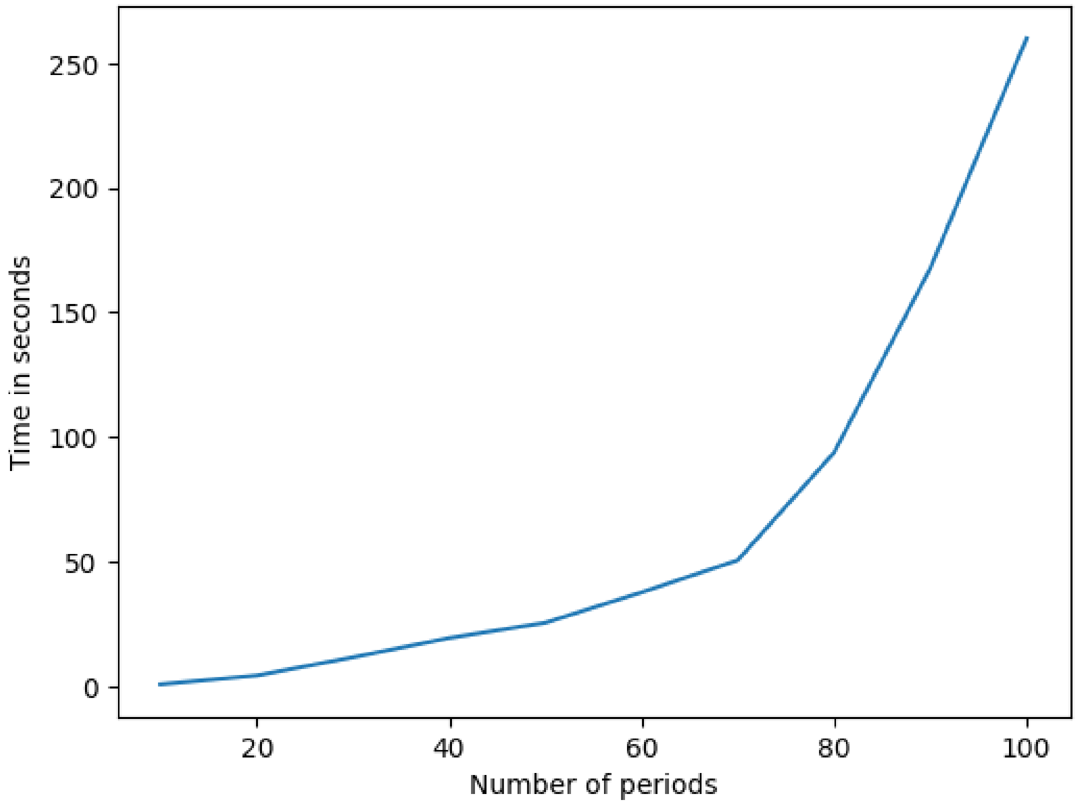
[10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0]



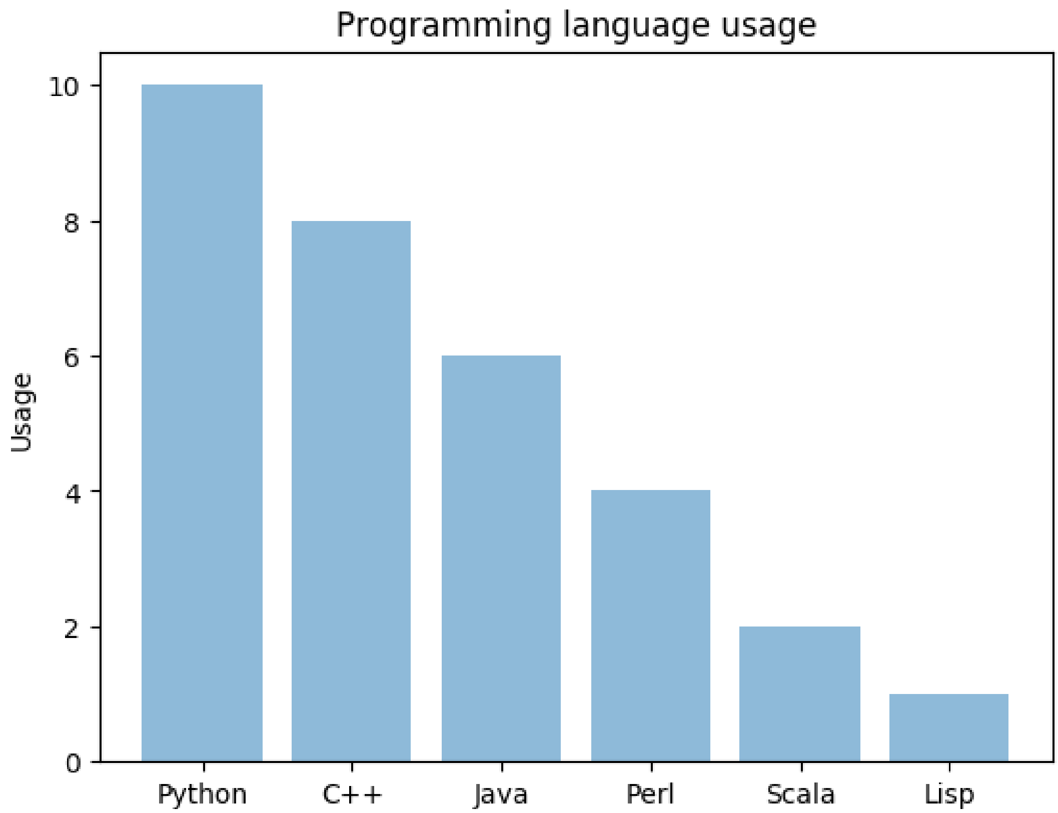
[76]:
```
gencost=[sum([UNITCOMIT[n][10]*Val1[T*n+t] for t in range(T)]) for n in␣
↪range(N)]
startcost=[sum([UNITCOMIT[n][9]*Val2[T*n+t] for t in range(T)]) for n in␣
↪range(N)]
shutcost=[sum([UNITCOMIT[n][0]*Val4[T*n+t] for t in range(T)]) for n in␣
↪range(N)]
p=gencost
m=[UNITCOMIT[n][9] for n in range(N)]
import matplotlib.pyplot as plt
n_periods=[10,20,30,40,50,60,70,80,90,100]
Time =[0.97,4.49,11.77,19.53,25.65,37.83,50.71,93.88,167.73,259.83]
plt.plot(m,p)
plt.ylabel('Total generational cost')
plt.xlabel('variable cost')
plt.show()
```



[77]:
```
import matplotlib.pyplot as plt
n_periods=[10,20,30,40,50,60,70,80,90,100]
Time =[0.97,4.49,11.77,19.53,25.65,37.83,50.71,93.88,167.73,259.83]
plt.plot(n_periods,Time )
plt.ylabel('Time in seconds')
plt.xlabel('Number of periods')
plt.show()
```



[78]:
```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Usage')
plt.title('Programming language usage')
plt.show()
```



[79]:
```
gencostt=[sum([UNITCOMIT[n][10]*Var[T*n+t] for n in range(N)]) for t in␣
 ↪range(T)]
startcostt=[sum([UNITCOMIT[n][9]*indic_su[T*n+t] for n in range(N)]) for t in␣
 ↪range(T)]
shutcostt=[sum([UNITCOMIT[n][0]*indic_sd[T*n+t] for n in range(N)]) for t in␣
 ↪range(T)]
```

0.2 LAGRANGIAN RELAXATION

[80]:
```
N=10
T=10
K=N*T
R = PolynomialRing(RR, 'mu', K)
S = PolynomialRing(RR, 'beta', K)
mu = R.gens()
beta=S.gens()
#tim = S.gens()
mu=[SR(mu[t]) for t in srange(0,T)];
beta=[SR(beta[t]) for t in srange(0,T)]
```

```
R = PolynomialRing(RR, 'mu', T)
S = PolynomialRing(RR, 'beta', T)
mu = R.gens()
beta=S.gens()
mu=[SR(mu[t]) for t in srange(0,T) ];
beta=[SR(beta[t]) for t in srange(0,T) ]
print(beta)
H = PolynomialRing(RR, 'indic_su', K)
S = PolynomialRing(RR, 'indic_on_off', K)
T = PolynomialRing(RR, 'indic_sd', K)
R = PolynomialRing(RR, 'Gen', K)
Gen = R.gens()
tim = S.gens()
Gen=[SR(Gen[k]) for k in srange(0,K)];
indic_su= H.gens()
indic_on_off=S.gens()
indic_sd= T.gens()
tim = S.gens()
indic_su=[SR(indic_su[k]) for k in srange(K)];
indic_on_off=[SR(indic_on_off[k]) for k in srange(K)];
indic_sd=[SR(indic_sd[k]) for k in srange(K)]
```

```
[beta0, beta1, beta2, beta3, beta4, beta5, beta6, beta7, beta8, beta9]
```

[81]:
```
N=10
T=10
gencost=sum([UNITCOMIT[n][10]*Gen[T*n+t] for n in range(N) for t in range(T)])
startcost=[sum([UNITCOMIT[n][9]*indic_su[T*n+t] for n in range(N) for t in␣
 ↪range(T)])]
shutcost=[sum([UNITCOMIT[n][0]*indic_sd[T*n+t] for n in range(N) for t in␣
 ↪range(T)])]
z=gencost
```

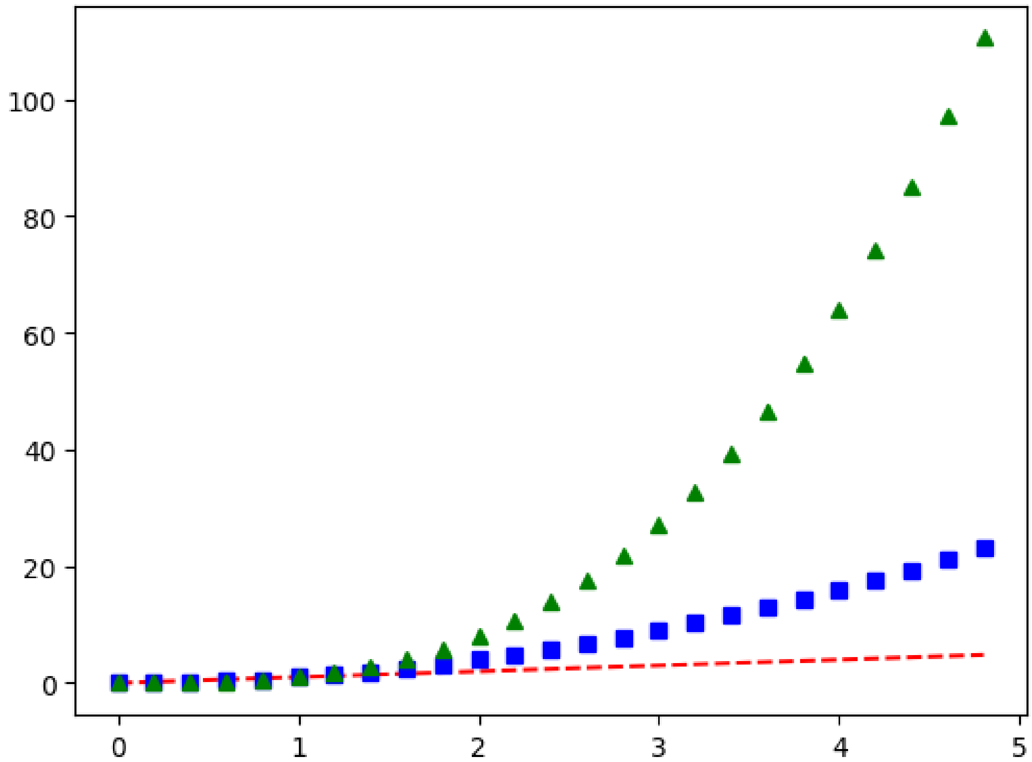
[82]:
```
M=[derivative(z,Gen[k]) for k in range(K) ]
```

[83]:
```
R = PolynomialRing(GF(2), 'indic_su', K)
S = PolynomialRing(GF(2), 'indic_on_off', K)
T = PolynomialRing(GF(2), 'indic_sd', K)
indic_su= R.gens()
indic_on_off=S.gens()
indic_sd= T.gens()
#tim = S.gens()
indic_su=[SR(indic_su[k]) for k in srange(0,K)];
R = PolynomialRing(RR, 'indic_su', K)
indic_su = R.gens()
indic_su=[SR(indic_su[k]) for k in srange(0,K) ];
print(indic_su)
```

```
[indic_su0, indic_su1, indic_su2, indic_su3, indic_su4, indic_su5, indic_su6,
indic_su7, indic_su8, indic_su9, indic_su10, indic_su11, indic_su12, indic_su13,
indic_su14, indic_su15, indic_su16, indic_su17, indic_su18, indic_su19,
indic_su20, indic_su21, indic_su22, indic_su23, indic_su24, indic_su25,
indic_su26, indic_su27, indic_su28, indic_su29, indic_su30, indic_su31,
indic_su32, indic_su33, indic_su34, indic_su35, indic_su36, indic_su37,
indic_su38, indic_su39, indic_su40, indic_su41, indic_su42, indic_su43,
indic_su44, indic_su45, indic_su46, indic_su47, indic_su48, indic_su49,
indic_su50, indic_su51, indic_su52, indic_su53, indic_su54, indic_su55,
indic_su56, indic_su57, indic_su58, indic_su59, indic_su60, indic_su61,
indic_su62, indic_su63, indic_su64, indic_su65, indic_su66, indic_su67,
indic_su68, indic_su69, indic_su70, indic_su71, indic_su72, indic_su73,
indic_su74, indic_su75, indic_su76, indic_su77, indic_su78, indic_su79,
indic_su80, indic_su81, indic_su82, indic_su83, indic_su84, indic_su85,
indic_su86, indic_su87, indic_su88, indic_su89, indic_su90, indic_su91,
indic_su92, indic_su93, indic_su94, indic_su95, indic_su96, indic_su97,
indic_su98, indic_su99]
```

[84]:

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



[85]:
```
import numpy as np
import matplotlib.pyplot as plt
D=[[Val1[T*n+t]for t in range(T)] for n in range(N)]
N = 24
menMeans = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd = (2, 3, 4, 1, 2)
womenStd = (3, 5, 2, 3, 3)
ind = np.arange(N)    # the x locations for the groups
width = 0.5       # the width of the bars: can also be len(x) sequence

p1 = plt.bar(ind, tuple(D[0]), width)
p2 = plt.bar(ind, tuple(D[1]), width)#, yerr=menStd)
p3 = plt.bar(ind, tuple(D[2]), width)
p4 = plt.bar(ind, tuple(D[3]), width)
p5 = plt.bar(ind, tuple(D[4]), width)#, yerr=menStd)
p6 = plt.bar(ind, tuple(D[5]), width)
p7 = plt.bar(ind, tuple(D[6]), width)
p8 = plt.bar(ind, tuple(D[7]), width)#, yerr=menStd)
p9 = plt.bar(ind, tuple(D[8]), width)
p10 = plt.bar(ind,tuple(D[9]), width)#, yerr=womenStd)
```

```
plt.ylabel('Power Output and Demand (MW)')
plt.title('1 day Unit Commitment Problem')
plt.xticks(ind, ('1', '', '', '', '5','', '', '', '', '10','', '', '', '',␣
↪'15','', '', '', '', '20','', '', '', ''))
plt.yticks(np.arange(0, 1500, 100))
plt.legend((p1[0], p2[1],p3[0], p4[0],p5[0], p6[0],p7[0], p8[0], p9[0],␣
↪p10[0]),('coal1', 'coal2','gas1', 'gas2','coal3', 'coal4','diesel1',␣
↪'diesel2','diesel3', 'diesel4'))
plt.show()
show(D[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-85-f6e5a5f4aeed> in <module>()
      1 import numpy as np
      2 import matplotlib.pyplot as plt
----> 3 D=[[Val1[T*n+t]for t in range(T)] for n in range(N)]
      4 N = Integer(24)
      5 menMeans = (Integer(20), Integer(35), Integer(30), Integer(35),␣
↪Integer(27))

TypeError: range() integer end argument expected, got sage.rings.polynomial.
↪multi_polynomial_libsingular.MPolynomialRing_libsingular.
```

[ ]:
```
# libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
#import pandas as pd

# y-axis in bold
rc('font', weight='bold')
# Values of each group
bars1 = D[0]
bars2 = D[1]
bars3 = D[2]
bars4 = D[3]
bars5 = D[4]
bars6 = D[5]
bars7 = D[6]
bars8 = D[7]
bars9 = D[8]
bars10= D[9]


# Heights of bars1 + bars2 (TO DO better)
#sition of the bars on the x-axis
```

```python
r = [i for i in range(24)]

# Names of group and bar width
names = ['','','','','5','','','','','10','','','','','15','','','','','20','','','','']
barWidth = 0.5

# Create brown bars
p1=plt.bar(r, bars1,bottom=bars, color='Maroon', edgecolor='white', width=barWidth)
# Create green bars (middle), on top of the firs ones
p2=plt.bar(r, bars2, bottom=bars1, color='blue', edgecolor='white', width=barWidth)
# Create green bars (top)
p3=plt.bar(r, bars3, bottom=bars2, color='orange', edgecolor='white', width=barWidth)
p4=plt.bar(r, bars4,bottom=bars3, color='red', edgecolor='white', width=barWidth)
# Create green bars (middle), on top of the firs ones
p5=plt.bar(r, bars5, bottom=bars4, color='green', edgecolor='white', width=barWidth)
# Create green bars (top)
p6=plt.bar(r, bars6, bottom=bars5, color='lightblue', edgecolor='white', width=barWidth)
p7=plt.bar(r, bars7,bottom=bars6, color='violet', edgecolor='white', width=barWidth)
# Create green bars (middle), on top of the firs ones
p8=plt.bar(r, bars8, bottom=bars7, color='purple', edgecolor='white', width=barWidth)
# Create green bars (top)
p9=plt.bar(r, bars9, bottom=bars8, color='lightgreen', edgecolor='white', width=barWidth)
p10=plt.bar(r, bars10, color='tomato', edgecolor='white', width=barWidth)

plt.legend((p1, p2,p3, p4,p5, p6,p7, p8, p9, p10),('coal1', 'coal2','gas1', 'gas2','gas3', 'gas4','diesel1', 'diesel2','diesel3', 'diesel4'))

# Custom X axis
plt.xticks(r, names, fontweight='bold')
plt.xlabel("Time")
plt.ylabel('Power Output and Demand (MW)')
plt.title('1 day Unit Commitment Problem')
# Show graphic
plt.show()
```

[ ]:
```
#P=[sum([Val1[J*i+j] for j in range(J)]) for i in range(I)]
N_online=[sum([Val3[T*n+t] for n in range(10)]) for t in range(24)]
P=N_online
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('1', '', '', '', '5', '','', '', '', '10','', '', '', '', '15',␣
↪'','', '', '', '20','', '', '', '')
y_pos = np.arange(len(objects))
performance = P

plt.bar(y_pos, performance, align='center', alpha=0.8, width=0.5,␣
↪color=['blue'])
plt.xticks(y_pos, objects)
plt.ylabel(' Number of units online')
plt.xlabel('periods')
plt.title('')
plt.show()
```

[ ]:
```
A=matrix([[1,2],[3,4]])
A
```

[ ]:
```
A=[i for i in range(101)]
B=[j for j in range(101,202)]
C=[A[i]*B[i] for i in range(101)]
show(sum(C)/102)
```

[ ]:
```
10002*1990000
```

[ ]:
```
A=[i for i in range(101)]
```

[ ]:
```

```