Homework 3-Question 1: Doodle scheduling

Doodle Inc. is looking to interview a candidate for a new software engineer position at their company. It works like this: the interview (10 AM to 3 PM) is divided into a number of 20-minute time slots that may be used for 1-on-1 meetings with the candidate. There is also a one-hour time slot in the middle of the day where 3 employees take the candidate out for lunch. It would be nice for all 15 senior employees to meet with the candidate at some point during the day, but everybody has a busy schedule so it's not clear whether this will be possible. A doodle poll (obviously) was sent to the 15 senior employees to figure out their availability

	10:00	**10:20**	**10:40**	**11:00**	**11:20**	**11:40**	**Lunch**	**1:00**	**1:20**	**1:40**	**2:00**	**2:20**	**2:40**
Manuel	0	0	1	1	0	0	0	1	1	0	0	0	0
Luca	0	1	1	0	0	0	0	0	1	1	0	0	0
Jule	0	0	0	1	1	0	1	1	0	1	1	1	1
Michael	0	0	0	1	1	1	1	1	1	1	1	1	0
Malte	0	0	0	0	0	0	1	1	1	0	0	0	0
Chris	0	1	1	0	0	0	0	0	1	1	0	0	0
Spyros	0	0	0	1	1	1	1	0	0	0	0	0	0
Mirjam	1	1	0	0	0	0	0	0	0	0	1	1	1
Matt	1	1	1	0	0	0	0	0	0	1	1	0	0
Florian	0	0	0	0	0	0	0	1	1	0	0	0	0
Josep	0	0	0	0	0	0	1	1	1	0	0	0	0
Joel	1	1	0	0	0	1	1	1	1	0	0	1	1
Tom	1	1	1	0	1	1	0	0	0	0	0	1	1
Daniel	0	1	1	1	0	0	0	0	0	0	0	0	0
Anne	1	1	0	0	1	1	0	0	0	0	0	0	0

In the table, a 1 means that the employee is available at the indicated time, while a 0 means that they are unavailable. Determine whether a feasible interview schedule exists. If so, print out a calendar for the candidate that lists who they will be meeting at each time slot.

Problem Data

In [1]:
```
using NamedArrays
# Employees
employees = ["Manuel", "Luca", "Jule", "Michael", "Malte", "Chris",
    "Spyros", "Mirjam", "Matt", "Florian", "Josep", "Joel", "Tom", "Daniel", "Anne"]

# Slots
slots = ["10:00","10:20","10:40","11:00","11:20","11:40","Lunch",
    "1:00","1:20","1:40","2:00","2:20","2:40"]

# Slot Capacity
slotCapacity = Dict(zip(slots,[1,1,1,1,1,1,3,1,1,1,1,1,1]))

raw = [ 0 0 1 1 0 0 0 1 1 0 0 0 0;
        0 1 1 0 0 0 0 0 1 1 0 0 0;
        0 0 0 1 1 0 1 1 0 1 1 1 1;
        0 0 0 1 1 1 1 1 1 1 1 1 0;
        0 0 0 0 0 0 1 1 1 0 0 0 0;
        0 1 1 0 0 0 0 0 1 1 0 0 0;
        0 0 0 1 1 1 1 0 0 0 0 0 0;
        1 1 0 0 0 0 0 0 0 0 1 1 1;
        1 1 1 0 0 0 0 0 0 1 1 0 0;
        0 0 0 0 0 0 0 1 1 0 0 0 0;
        0 0 0 0 0 0 1 1 1 0 0 0 0;
        1 1 0 0 0 1 1 1 1 0 0 1 1;
        1 1 1 0 1 1 0 0 0 0 0 1 1;
        0 1 1 1 0 0 0 0 0 0 0 0 0;
        1 1 0 0 1 1 0 0 0 0 0 0 0;]

availability = NamedArray(raw, (employees, slots), ("Employees", "Slots"));
```

Problem Model

In [2]:
```
using JuMP

m = Model()

# decision variables, flow on edges from Layer 1: employees to Layer 2: slots
@variable(m, x[employees,slots] >= 0)

# availability constraint, to assign an employee to slot only when the employee is available.
for e in employees
    for s in slots
        @constraint(m, x[e,s] <= availability[e,s])
    end
end

# constraint to limit only one employee per slot and three employees for lunch.
@constraint(m,attendancePerSlot[s in slots],sum(x[e,s] for e in employees) <= slotCapacity[s])

# constraint for all employees to meet the candidate exactly once.
@constraint(m,attendancePerEmployee[e in employees],sum(x[e,s] for s in slots) == 1)

# Max flow objective along edges
@objective(m, Max, sum(x[e,s] for e in employees for s in slots))
;
```

In [3]:
```
status = solve(m)
println(status)
println(getobjectivevalue(m))
# nicely formatted solution
solution = NamedArray( Int[getvalue(x[i,j]) for i in employees, j in slots], (employees,slots),
    ("Employees","Slots") )
[println(s,"\t:\t" ,e) for s in slots for e in employees if solution[e,s] == 1]
println()
```

```
Optimal
15.0
10:00   :       Mirjam
10:20   :       Joel
10:40   :       Manuel
11:00   :       Daniel
11:20   :       Jule
11:40   :       Anne
Lunch   :       Malte
Lunch   :       Spyros
Lunch   :       Josep
1:00    :       Florian
1:20    :       Luca
1:40    :       Chris
2:00    :       Matt
2:20    :       Michael
2:40    :       Tom
```

The assignment of employees to slots has been shown above.

Homework 3 Question 2: Car Rental

A small car rental company has a fleet of 94 vehicles distributed among its 10 agencies. The location of every agency is given by its geographical coordinates x and y in a grid based on miles. We assume that the road distance between agencies is approximately 1.3 times the Euclidean distance (as the crow flies). The following table indicates the coordinates of all agencies, the number of cars required the next morning, and the stock of cars in the evening preceding this day.

Agency number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
x-coordinate	0	20	18	30	35	33	5	5	11	2
y-coordinate	0	20	10	12	0	25	27	10	0	15
Required cars	10	6	8	11	9	7	15	7	9	12
Cars present	8	13	4	8	12	2	14	11	15	7

Supposing the cost for transporting a car is $0.50 per mile, the movements of cars that allow the company to re-establish the required numbers of cars at all agencies, minimizing the total cost incurred for transport.

Problem Data

In [1]:
```
# Helper Function
function agencyEuclDistance(ax, ay, bx, by)
    return 1.3*(sqrt((ax-bx)^2 + (ay-by)^2))
end;
```

In [2]:
```
using NamedArrays

# agency list
agency = [1,2,3,4,5,6,7,8,9,10]

# required cars per agency
required = Dict(zip(agency, [10 6 8 11 9 7 15 7 9 12]))

# present cars per agency
present =  Dict(zip(agency, [8 13 4 8 12 2 14 11 15 7]))

# location (x,y) tuples per agency
location = Dict(zip(agency,[( 0 ,  0)
                            ( 20,  20)
                            ( 18,  10)
                            ( 30,  12)
                            ( 35,   0)
                            ( 33,  25)
                            (  5,  27)
                            (  5,  10)
                            ( 11,   0)
                            (  2,  15)]))

# cost per mile for transportation
costPerMile = 0.5

# Arrays to store agency to agency cost (agency, agency)
costRaw = reshape([costPerMile*agencyEuclDistance(location[a][1],location[a][2],
            location[b][1],location[b][2])
    for a in agency for b in agency],length(agency), length(agency))
cost = NamedArray(costRaw, (agency,agency), ("Agency", "Agency"))
;
```

Problem Model

In [3]:
```
using JuMP

m = Model()

# Variables for edge cost from agency to agency
@variable(m, x[agency,agency] >= 0)

# Constraint to limit the number of cars an agency can transfer to the number of cars present
@constraint(m, supply[a in agency], sum(x[a,b] for b in agency) <= present[a])

# Constraint to impose the required number of cars at the end at each agency
@constraint(m, demand[b in agency], sum(x[a,b] for a in agency) == required[b])

# Objective of minimizing cost per car transfer between agencies
@objective(m, Min, sum(cost[a,b]*x[a,b] for a in agency for b in agency))
;
```

In [4]:
```
status = solve(m)
println("Minimum cost to reistablish demands: \$",getobjectivevalue(m))

# nicely formatted solution
raw = Int[getvalue(x[i,j]) for i in agency, j in agency]
raw = hcat(raw,sum(raw,2))
raw = vcat(raw,sum(raw,1))

solution = NamedArray(raw,([1,2,3,4,5,6,7,8,9,10,"Required"],
        [1,2,3,4,5,6,7,8,9,10,"Pres"]),("Src","Dest"))
println(solution)
```

```
Minimum cost to reistablish demands: $152.63901632295628
11×11 Named Array{Int64,2}
Src ╲ Dest │     1     2     3     4     5     6     7     8     9    10  Pres
───────────┼──────────────────────────────────────────────────────────────────
1          │     8     0     0     0     0     0     0     0     0     0     8
2          │     0     6     1     0     0     5     1     0     0     0    13
3          │     0     0     4     0     0     0     0     0     0     0     4
4          │     0     0     0     8     0     0     0     0     0     0     8
5          │     0     0     0     3     9     0     0     0     0     0    12
6          │     0     0     0     0     0     2     0     0     0     0     2
7          │     0     0     0     0     0     0    14     0     0     0    14
8          │     0     0     0     0     0     0     0     7     0     4    11
9          │     2     0     3     0     0     0     0     0     9     1    15
10         │     0     0     0     0     0     0     0     0     0     7     7
Required   │    10     6     8    11     9     7    15     7     9    12    94
```

The car transfers from source agency to destination agency is as shown above. The rightmost column shows the cars initially present at all source agencies, while the bottom most row shows the required cars at each destination agency. The values in these rows and columns is as per the requirement of the question.

Homework 3 Question 3: Building a stadium.

A town council wishes to construct a small stadium in order to improve the services provided to the people living in the district. After the invitation to tender, a local construction company is awarded the contract and wishes to complete the task within the shortest possible time. All the major tasks are listed in the following table. Some tasks can only start after the completion of certain other tasks, as indicated by the “Predecessors” column.

Task	Description	Duration (in weeks)	Predecessors	Maximum reduction (in weeks)	Cost of reduction ($1k/wk)
1	Installing the construction site	2	none	0	–
2	Terracing	16	1	3	30
3	Constructing the foundations	9	2	1	26
4	Access roads and other networks	8	2	2	12
5	Erecting the basement	10	3	2	17
6	Main floor	6	4,5	1	15
7	Dividing up the changing rooms	2	4	1	8
8	Electrifying the terraces	2	6	0	–
9	Constructing the roof	9	4,6	2	42
10	Lighting of the stadium	5	4	1	21
11	Installing the terraces	3	6	1	18
12	Sealing the roof	2	9	0	–
13	Finishing the changing rooms	1	7	0	–
14	Constructing the ticket office	7	2	2	22
15	Secondary access roads	4	4,14	2	12
16	Means of signalling	3	8,11,14	1	6
17	Lawn and sport accessories	9	12	3	16
18	Handing over the building	1	17	0	–

a) What is the earliest possible date of completion for the construction? Note that the last two columns of the table are not relevant for this part of the problem.

Problem Data

In [1]:
```
# this array stores the task names (:a, :b, ..., :r)
nTasks=18
tasks = []
for i = 'a':'a'+nTasks-1
    push!(tasks, Symbol(i))
end

# this dictionary stores the project durations
dur = [2 16 9 8 10 6 2 2 9 5 3 2 1 7 4 3 9 1]
duration = Dict(zip(tasks,dur))

red = [0 3 1 2 2 1 1 0 2 1 1 0 0 2 2 1 3 0]
reduction = Dict(zip(tasks,red))

costOfRed = [0 30 26 12 17 15 8 0 42 21 18 0 0 22 12 6 16 0]
costOfReduction = Dict(zip(tasks,costOfRed))

# this dictionary stores the projects that a given project depends on (ancestors)
pre = ([],[tasks[1]],[tasks[2]],[tasks[2]],[tasks[3]],[tasks[4],tasks[5]],[tasks[4]],
    [tasks[6]],[tasks[4],tasks[6]],[tasks[4]],[tasks[6]],[tasks[9]],[tasks[7]],[tasks[2]],
    [tasks[4],tasks[14]],[tasks[8],tasks[11],tasks[14]],[tasks[12]],[tasks[17]])
pred = Dict(zip(tasks,pre));
```

Problem Model

In [2]:
```
using JuMP
m = Model()

# variable to hold start time of all tasks
@variable(m, tstart[tasks] >= 0 )

# constraints imposed on start time of tasks due to its predecessors
start = @constraint(m, [i in tasks, j in pred[i]], tstart[i] >= tstart[j] + duration[j])

# objective to minimize the end time of last task
@objective(m, Min, tstart[tasks[nTasks]] + duration[tasks[nTasks]])

solve(m)
println(getvalue(tstart))
```

```
tstart: 1 dimensions:
[a] = 0.0
[b] = 2.0
[c] = 18.0
[d] = 18.0
[e] = 27.0
[f] = 37.0
[g] = 26.0
[h] = 43.0
[i] = 43.0
[j] = 26.0
[k] = 43.0
[l] = 52.0
[m] = 28.0
[n] = 18.0
[o] = 26.0
[p] = 46.0
[q] = 54.0
[r] = 63.0
```

So the earliest possible completion date is 64 weeks.

b) For some of the tasks, the builder may employ additional workers and rent more equipment to cut down on the total time. The last two columns of the table show the maximum number of weeks that can be saved per task and the associated additional cost per week incurred by the extra work. Plot a trade-off curve that shows extra cost as a function of the number of weeks early we wish the stadium to be completed.

The trick used to solve this part of the question is the following. Reducing a week from any of the task may or may not reduce the overall time of stadium completion. But reducing the time in tasks which have tight constraints will definitely reduce the overall time. To identify those constraints we look at the dual variable for it. If the constraint is slack, its dual variable will be 0, otherwise 1.

In [3]:
```
# Total reduction possible looking at all tight constraints and their maximum reduction weeks
reductionPossible = sum([reduction[j]
        for i in tasks for j in pred[i] if getdual(start[i,j]) == 1])

# If all the reduction possible is implemented, the max cost is stored
maxCostPossible = sum([reduction[j]*costOfReduction[j]
        for i in tasks for j in pred[i] if getdual(start[i,j]) == 1])

println("The reduction possible by seeing the tight bounds is ",
    reductionPossible," weeks at cost \$",maxCostPossible,"k")

[println("Reduce work of ",j," upto maximum of ",reduction[j],
        " weeks at cost \$", costOfReduction[j],"k per week")
    for i in tasks for j in pred[i] if getdual(start[i,j])*reduction[j] > 0];
```

```
The reduction possible by seeing the tight bounds is 12 weeks at cost $297k
Reduce work of b upto maximum of 3 weeks at cost $30k per week
Reduce work of c upto maximum of 1 weeks at cost $26k per week
Reduce work of e upto maximum of 2 weeks at cost $17k per week
Reduce work of f upto maximum of 1 weeks at cost $15k per week
Reduce work of i upto maximum of 2 weeks at cost $42k per week
Reduce work of q upto maximum of 3 weeks at cost $16k per week
```

Figuring out the optimal cost per week reduction from the possible set of tight bounds, turns out to be a linear programming problem. Which is solved next

Problem Data

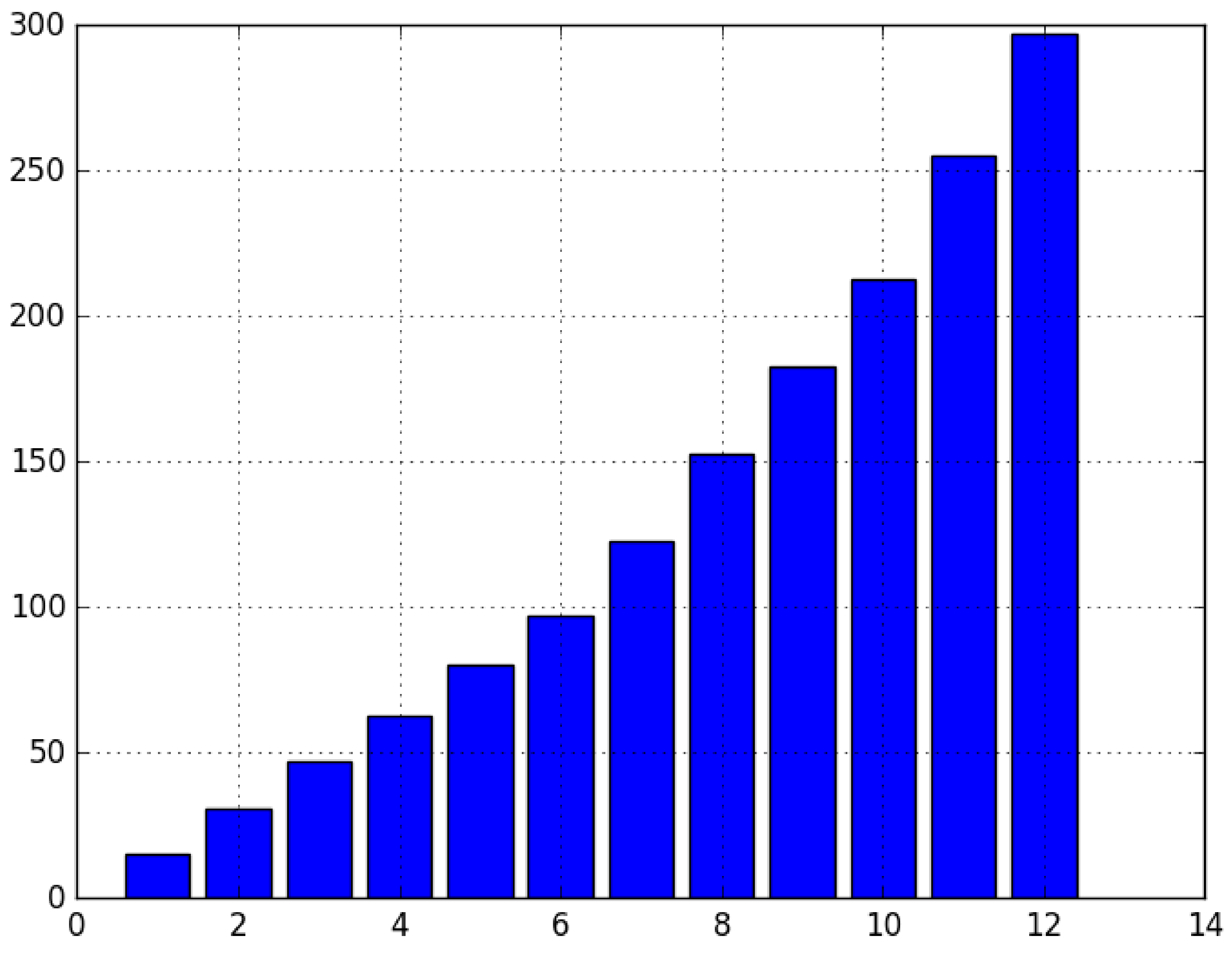
In [4]:
```
# This is the maximum possible reduction possible for tasks which have tight constraints
reductionTight = Dict(zip(tasks, zeros(red)))
[reductionTight[j] = reduction[j]
    for i in tasks for j in pred[i] if  getdual(start[i,j])*reduction[j] > 0];
```

Problem Model

In [5]:
```
md = Model()

# variable for reduction in weeks for each task
@variable(md, x[tasks] >= 0)

# constraint to limit the reduction to reduction possible for each task
@constraint(md, reduce[t in tasks], x[t] <= reductionTight[t])

# objective to minimize the cost for total reduction in weeks
@objective(md, Min, sum(x[t]*costOfReduction[t] for t in tasks));

# Incrementally increasing the lower bound for total reduction in weeks to
# plot the trade off curve.
y = zeros(reductionPossible)
for i in 1:reductionPossible
    @constraint(md, sum(x[t] for t in tasks) >= i)
    status = solve(md)
    y[i] = getobjectivevalue(md)
end
println("Cost per number of weeks reduced: ",y)
```

```
Cost per number of weeks reduced: [15.0,31.0,47.0,63.0,80.0,97.0,123.0,153.0,183.0,213.0,255.0,29
7.0]
```

In [6]:
```
x = linspace(1,reductionPossible,reductionPossible)

using PyPlot
grid("on")
bar(x,y,align="center");
```



c) The town council wants the builder to expedite the project. As an incentive, the council will pay a bonus of $30k/week for each week the work finishes early. When will the project be completed if the builder is acting in a way that maximizes his profit?

In [7]:
```
incentive = 30
[println(i," week reduction gives profit of \$",i*incentive - y[i],"k")
    for i in 1:reductionPossible];
```

```
1 week reduction gives profit of $15.0k
2 week reduction gives profit of $29.0k
3 week reduction gives profit of $43.0k
4 week reduction gives profit of $57.0k
5 week reduction gives profit of $70.0k
6 week reduction gives profit of $83.0k
7 week reduction gives profit of $87.0k
8 week reduction gives profit of $87.0k
9 week reduction gives profit of $87.0k
10 week reduction gives profit of $87.0k
11 week reduction gives profit of $75.0k
12 week reduction gives profit of $63.0k
```

The above code output shows that to maximize profit, the builder needs to finish the project early anywhere between 7-10 weeks. The profit would be $87k

Homework 3 Question 4 Dual interpretation.

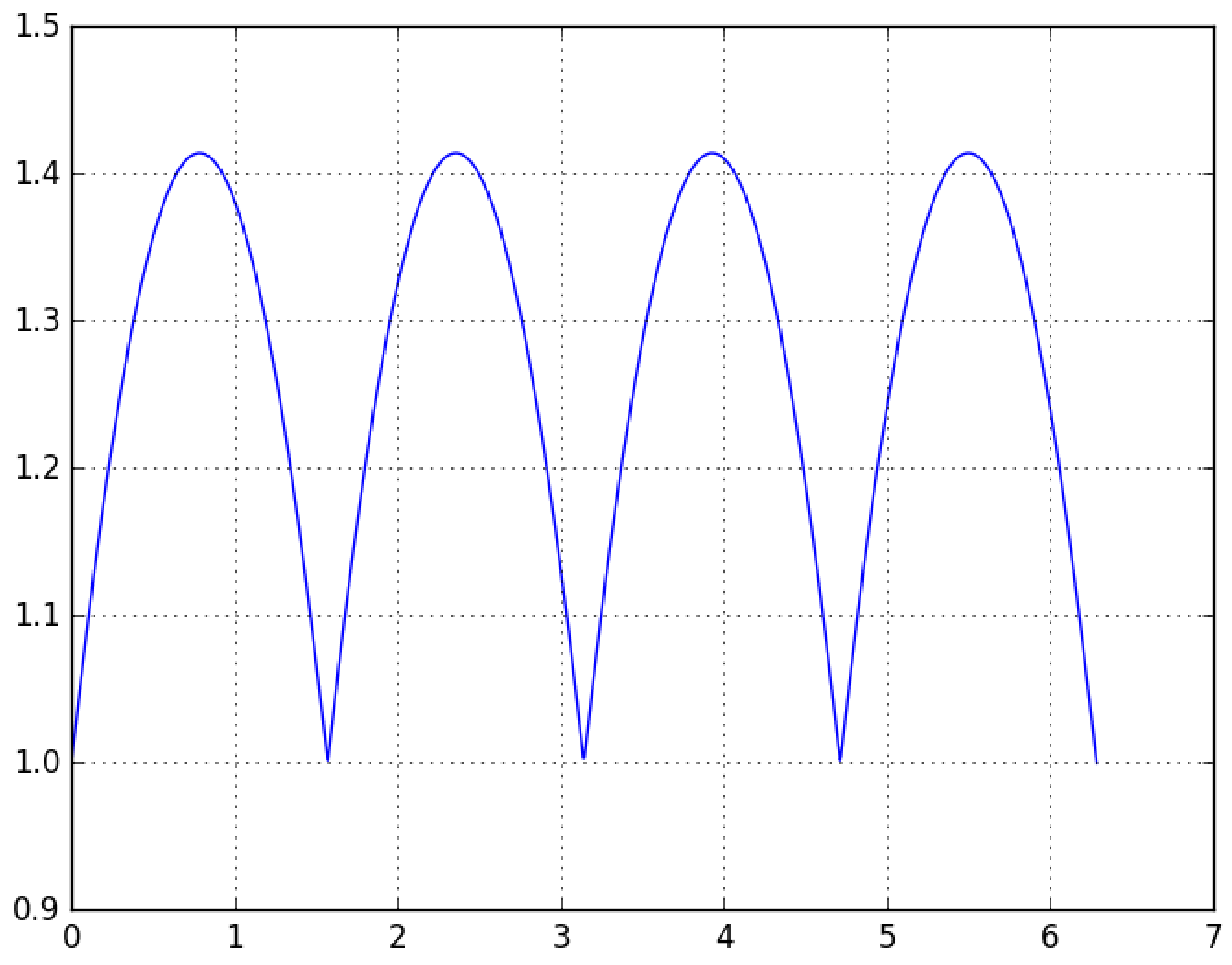
Suppose t ∈ [0, 2π] is a parameter. Consider the following LP:

$$\begin{aligned} \underset{p,q,r,s}{\text{minimize}} \quad & p+q+r+s \\ \text{subject to:} \quad & p-r = cos(t) \\ & q-s = sin(t) \\ & p,q,r,s \geq 0. \end{aligned}$$

a) Plot the optimal objective of this LP as a function of t. Can you explain what you see? Hint: separately consider the cases where cos(t) and sin(t) are positive or negative (four cases).

In [1]:
```
using JuMP

tArg = linspace(0,2*pi,1000)
y = zeros(tArg)
idx = 1
for t in tArg
    m = Model()
    @variable(m, p>=0)
    @variable(m, q>=0)
    @variable(m, r>=0)
    @variable(m, s>=0)
    @constraint(m, p-r == cos(t))
    @constraint(m, q-s == sin(t))
    @objective(m, Min, p+r+q+s )
    solve(m)
    y[idx] = getobjectivevalue(m)
    idx+=1
end
```

In [2]:
```
using PyPlot
grid("on")
plot(tArg, y);
```



The graph above can be explained by considering the four cases of combination of sign of $cos(t)$ and $sin(t)$.

cos(t)	sin(t)	p	q	r	s
+	+	cos(t)	sin(t)	0	0
+	-	cos(t)	0	0	-sin(t)

-	+	0	sin(t)	-cos(t)	0
-	-	0	0	-cos(t)	-sin(t)

In order to minimize the objective of $p + q + r + s$, given constraint $p - r = cos(t)$ and $q - s = sin(t)$, depending on the sign one of the two terms in each equation can be put to zero. More specifically the lower of the two terms in equal to zero in order to minimize the total sum. The other term then becomes equal to $cos(t)$ or $sin(t)$.

So the objective value becomes equal to

$$|cos(t)| + |sin(t)|$$

after considering the signs of all p, q, r and s. Which is also seen through the graph plotted above.

b) Find the dual LP and interpret it geometrically. Does this agree with the solution of part a)?

The pair of primal and dual models that correspond to this model is the free form one. Where primal model

$$\begin{array}{ll} \underset{x}{\text{maximize}} & c^T x \\ \text{subject to:} & Ax \leq b \\ & x\ free. \end{array}$$

corresponds to dual

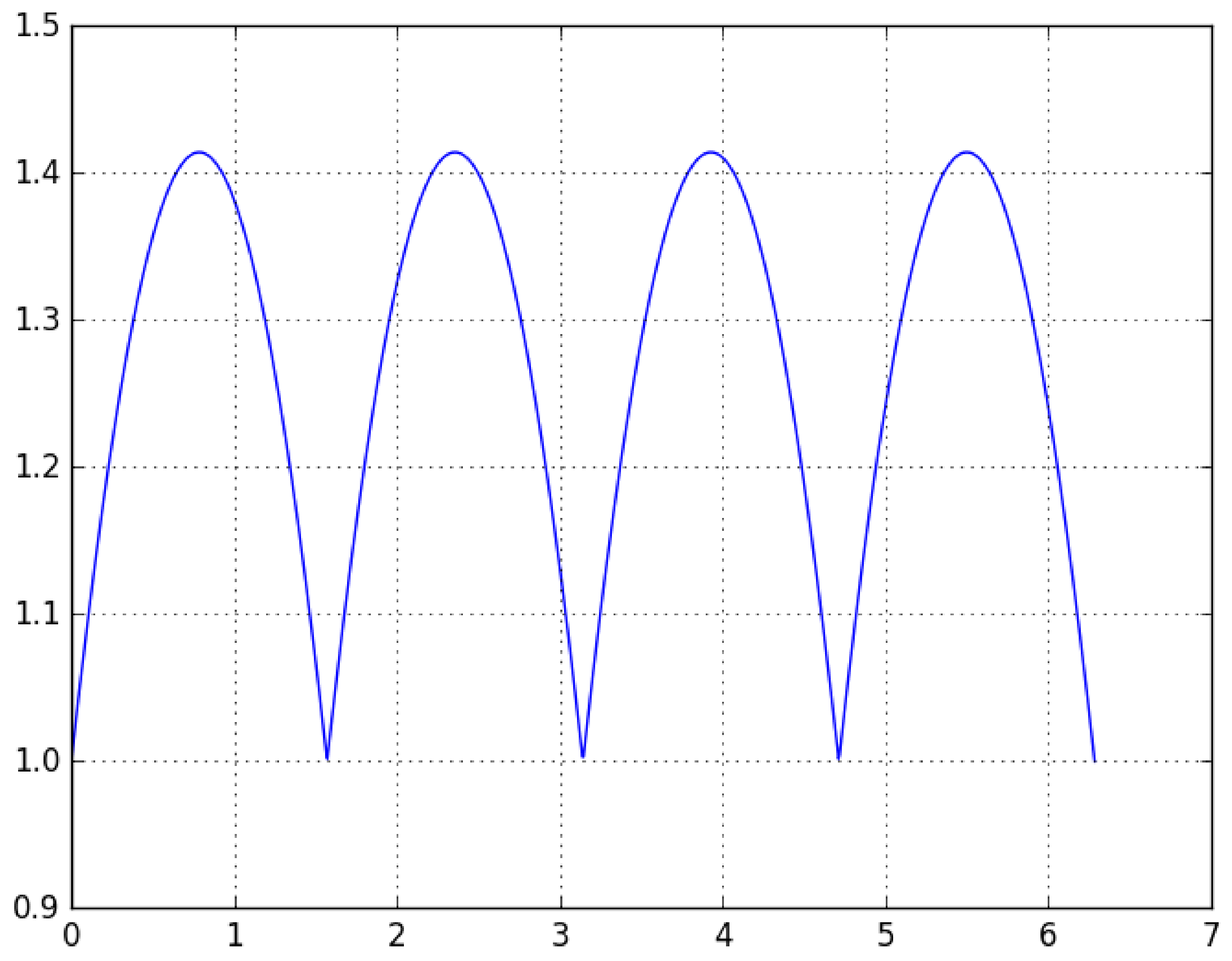
$$\begin{array}{ll} \underset{\lambda}{\text{minimize}} & b^T \lambda \\ \text{subject to:} & A^T \lambda = c \\ & \lambda \geq 0. \end{array}$$

In the case above, the model given in the question is of the dual form. We need to convert the dual form to primal one to answer this question.

Problem Data and Problem Model

In [3]:
```
# λ = [p; q; r; s]
# A, b and c taken from problem given in the question.

b = [1; 1; 1; 1]
A = [1 0 -1 0; 0 1 0 -1]'

tArg = linspace(0,2*pi,1000)
y = zeros(tArg)
idx = 1
for t in tArg
    c = [ cos(t) ; sin(t)]

    m = Model();

    @variable(m, x[1:2])   # variable x
    @constraint(m, A*x .<= b)
    @objective(m, Max, dot(c,x))

    solve(m)
    y[idx] = getobjectivevalue(m)
    idx+=1
end
```

In [4]:
```
grid("on")
plot(tArg, y);
```



The graph is exactly the same which is due to strong duality. Thus the solution to part b) corresponds correctly to part a)