Doodle Scheduling

Determine if is feasible to shedule senior employees with the interviewee subject to the given constraints.

In [277]:
```
JuMP, NamedArrays

vees = [ :Manuel, :Luca, :Jule, :Michael, :Malte, :Chris, :Spyros, :Mirjam, :Matt, :Florian, :Josep, :Joel,
        :Daniel, :Anne ]
periods = [ :am1, :am2, :am3, :am4, :am5, :am6, :Lunch, :pm1, :pm2, :pm3, :pm4, :pm5, :pm6 ]

[ 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 ]

ability = NamedArray( raw, (employees,time_periods), ("employees","time_periods"))

odel()

able(m, 0 <= x[employees,time_periods] <= 1)

 employee should meet interviewee at least once... nothing wrong with extra face time
traint(m, a[i in employees], sum(x[i,j] * availability[i,j] for j in time_periods) >= 1)

east one employee for each time slot. objective should keep this at one
traint(m, b[j in time_periods], sum(x[i,j] * availability[i,j] for i in employees) >= 1)

 employees for lunch. objective should keep this at three
traint(m, sum(x[i,:Lunch] * availability[i,:Lunch] for i in employees) >= 3)

ize the number of meetings of employees with the interviewee
ctive(m, Min, sum(x[i,j] for i in employees, j in time_periods))

m)

on = NamedArray( [ Int(getvalue(x[i,j])) for i in employees, j in time_periods ], (employees, time_periods
OContext(STDOUT, displaysize=(100, 1000)), solution)
```

15×13 Named Array{Int64,2}

name \ time	am1	am2	am3	am4	am5	am6	Lunch	pm1	pm2	pm3	pm4	pm5	pm6
Manuel	0	0	0	1	0	0	0	0	0	0	0	0	0
Luca	0	0	0	0	0	0	0	0	0	1	0	0	0
Jule	0	0	0	0	0	0	0	0	0	0	0	0	1
Michael	0	0	0	0	1	0	0	0	0	0	0	0	0
Malte	0	0	0	0	0	0	0	0	1	0	0	0	0
Chris	0	0	1	0	0	0	0	0	0	0	0	0	0
Spyros	0	0	0	0	0	0	1	0	0	0	0	0	0
Mirjam	0	0	0	0	0	0	0	0	0	0	0	1	0
Matt	0	0	0	0	0	0	0	0	0	0	1	0	0
Florian	0	0	0	0	0	0	0	1	0	0	0	0	0
Josep	0	0	0	0	0	0	1	0	0	0	0	0	0
Joel	0	0	0	0	0	0	1	0	0	0	0	0	0
Tom	0	0	0	0	0	1	0	0	0	0	0	0	0
Daniel	0	1	0	0	0	0	0	0	0	0	0	0	0
Anne	1	0	0	0	0	0	0	0	0	0	0	0	0

Above is a 2-d array, say x, where if x[i,j] = 1, then employee i will meet with interviewee at time period j.

Car Rental

Below we determine the optimal way to move the cars in reguards to minimizing the amount of distance all cars must travel in total.

In [116]:
```
agencies = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
xc       = [0, 20, 18, 30, 35, 33, 5, 5, 11, 2]
yc       = [0, 20, 10, 12, 0, 25, 27, 10, 0, 15]
req      = [10, 6, 8, 11, 9, 7, 15, 7, 9, 12]
cur      = [8, 13, 4, 8, 12, 2, 14, 11, 15, 7]

dist     = Array{Any}(10,10)

m = Model()

for i in 1:10
    for j in 1:10
        dist[i,j] = 1.3*((xc[i] - xc[j])^2 + (yc[i] - yc[j])^2)^(1/2)
    end
end

#x[i,j] := number of cars to move from agency i to agency j
@variable(m, x[i in 1:10,j in 1:10] >= 0)

@expression(m, expr[i=1:10], sum(x[i,j] * dist[i,j] for j=1:10))

#can only send out as many cars as are currently there
@constraint(m, a[i in 1:10], sum(x[i,j] for j in 1:10) <= cur[i])

#must have at least as many cars as are required at each agency
@constraint(m, b[i in 1:10], cur[i] - sum(x[i,j] for j in 1:10) + sum(x[j,i] for j in 1:10) >= req[i])

#minimize total distance traveled
@objective(m, Min, sum(expr[i=1:10]))
solve(m)

cost = 0.5 * getobjectivevalue(m)
solution = NamedArray([Int(getvalue(x[i,j])) for i=1:10, j=1:10], (agencies, agencies), ("Source","Destina
show(IOContext(STDOUT, displaysize=(100, 1000)), solution)
println("\nTotal movement cost: \$", cost)
```

```
10×10 Named Array{Int64,2}
Source ╲ Destination │  1   2   3   4   5   6   7   8   9  10
─────────────────────┼──────────────────────────────────────────
1                    │  0   0   0   0   0   0   0   0   0   0
2                    │  0   0   1   0   0   5   1   0   0   0
3                    │  0   0   0   0   0   0   0   0   0   0
4                    │  0   0   0   0   0   0   0   0   0   0
5                    │  0   0   0   3   0   0   0   0   0   0
6                    │  0   0   0   0   0   0   0   0   0   0
7                    │  0   0   0   0   0   0   0   0   0   0
8                    │  0   0   0   0   0   0   0   0   0   5
9                    │  2   0   3   0   0   0   0   1   0   0
10                   │  0   0   0   0   0   0   0   0   0   0
Total movement cost: $152.63901632295628
```

Above array, say x, represents how to optimally move the cars. x[i,j] := number of cars to move from agency i to agency j. Also above is the total movement cost.

Building a Stadium

Find the optimal schedule which minimizes total construction time. Part (a)

In [ ]:
```
dur  = [ 2 16 9 8 10 6 2 2 9 5 3 2 1 7 4 3 9 1 ]
pred = [ 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 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  1 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0
         0 0  0 1 1  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  1 0 0 0 0 0 0 0 0 0 0 0 0
         0 0  0 1 0  1 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  1 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 1 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 1 0  0 0 0 0 0 0 0 0 1 0 0 0 0
         0 0  0 0 0  0 0 1 0 0 1 0 0 1 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 1 0 ]
```

In [152]:
```
m = Model()

@variable(m, start[i=1:18] >= 0)
@variable(m, t >= 0)

for i = 1:18
    @constraint(m, t >= start[i] + dur[i])
    for j = 1:18
        if(pred[i,j] == 1)
            @constraint(m, start[j] + dur[j] <= start[i])
        end
    end
end

@objective(m, Min, t)
solve(m)

solution = getvalue(start)
for i = 1:18
    println("Start time for task ", i, ": ", solution[i])
end

println("\nConstruction end time: ", getobjectivevalue(m))
```

```
Start time for task 1: 0.0
Start time for task 2: 2.0
Start time for task 3: 18.0
Start time for task 4: 18.0
Start time for task 5: 27.0
Start time for task 6: 37.0
Start time for task 7: 26.0
Start time for task 8: 43.0
Start time for task 9: 43.0
Start time for task 10: 59.0
Start time for task 11: 43.0
Start time for task 12: 52.0
Start time for task 13: 63.0
Start time for task 14: 18.0
Start time for task 15: 60.0
Start time for task 16: 61.0
Start time for task 17: 54.0
Start time for task 18: 63.0

Construction end time: 64.0
```

Building a Stadium

Part (b)

In [278]:
```
max_red  = [ 0 3  1  2  2  1  1 0 2  1  1  0 0 2  2  1 3  0 ]
cost_red = [ 0 30 26 12 17 15 8 0 42 21 18 0 0 22 12 6 16 0 ]

m = Model()

@variable(m, start[i=1:18] >= 0)
@variable(m, t >= 0)
@variable(m, task_red[i=1:18] >= 0);
```

In [275]:
```
for i = 1:18
    @constraint(m, task_red[i] <= max_red[i])
    @constraint(m, t >= start[i] + dur[i] - task_red[i])
    for j = 1:18
        if(pred[i,j] == 1)
            @constraint(m, start[j] + dur[j] - task_red[i] <= start[i])
        end
    end
end

@objective(m, Min, t)
solve(m)

reduction = getvalue(task_red)
solution = getvalue(start)
for i = 1:18
    println("Start time for task ", i, ": ", solution[i], "\t [time reduction: ", reduction[i], "]")
end

println("\nConstruction end time: ", getvalue(t))
```

```
Start time for task 1: 0.0       [time reduction: 0.0]
Start time for task 2: 0.0       [time reduction: 3.0]
Start time for task 3: 15.0      [time reduction: 1.0]
Start time for task 4: 14.0      [time reduction: 2.0]
Start time for task 5: 22.0      [time reduction: 2.0]
Start time for task 6: 31.0      [time reduction: 1.0]
Start time for task 7: 21.0      [time reduction: 1.0]
Start time for task 8: 37.0      [time reduction: 0.0]
Start time for task 9: 35.0      [time reduction: 2.0]
Start time for task 10: 49.0     [time reduction: 1.0]
Start time for task 11: 36.0     [time reduction: 1.0]
Start time for task 12: 44.0     [time reduction: 0.0]
Start time for task 13: 52.0     [time reduction: 0.0]
Start time for task 14: 14.0     [time reduction: 2.0]
Start time for task 15: 51.0     [time reduction: 2.0]
Start time for task 16: 51.0     [time reduction: 1.0]
Start time for task 17: 43.0     [time reduction: 3.0]
Start time for task 18: 52.0     [time reduction: 0.0]

Construction end time: 53.0
```

So now we see the best time we can do is ending 53 weeks after start date and the worst optimal time is 64 weeks. So let's calculate the reduction cost for week reductions in the range of {0,...,11}.

In [244]:
```
m = Model()
@variable(m, start[i=1:18] >= 0)
@variable(m, t >= 0)
@variable(m, task_red[i=1:18] >= 0)


   for i = 1:18
       @constraint(m, task_red[i] <= max_red[i])
       @constraint(m, t >= start[i] + dur[i] - task_red[i])
       for j = 1:18
           if(pred[i,j] == 1)
               @constraint(m, start[j] + dur[j] - task_red[i] <= start[i])
           end
       end
   end


@constraint(m, t == 53)   #here is where we force an endtime value

   @objective(m, Min, sum(task_red[i]*cost_red[i] for i=1:18))
   solve(m)

   reduction = getvalue(task_red)
   solution = getvalue(start)

   println("\nConstruction end time: ", getvalue(t))
   println("Reduction cost: \$", 1000*sum(reduction[i] * cost_red[i] for i = 1:18))
```

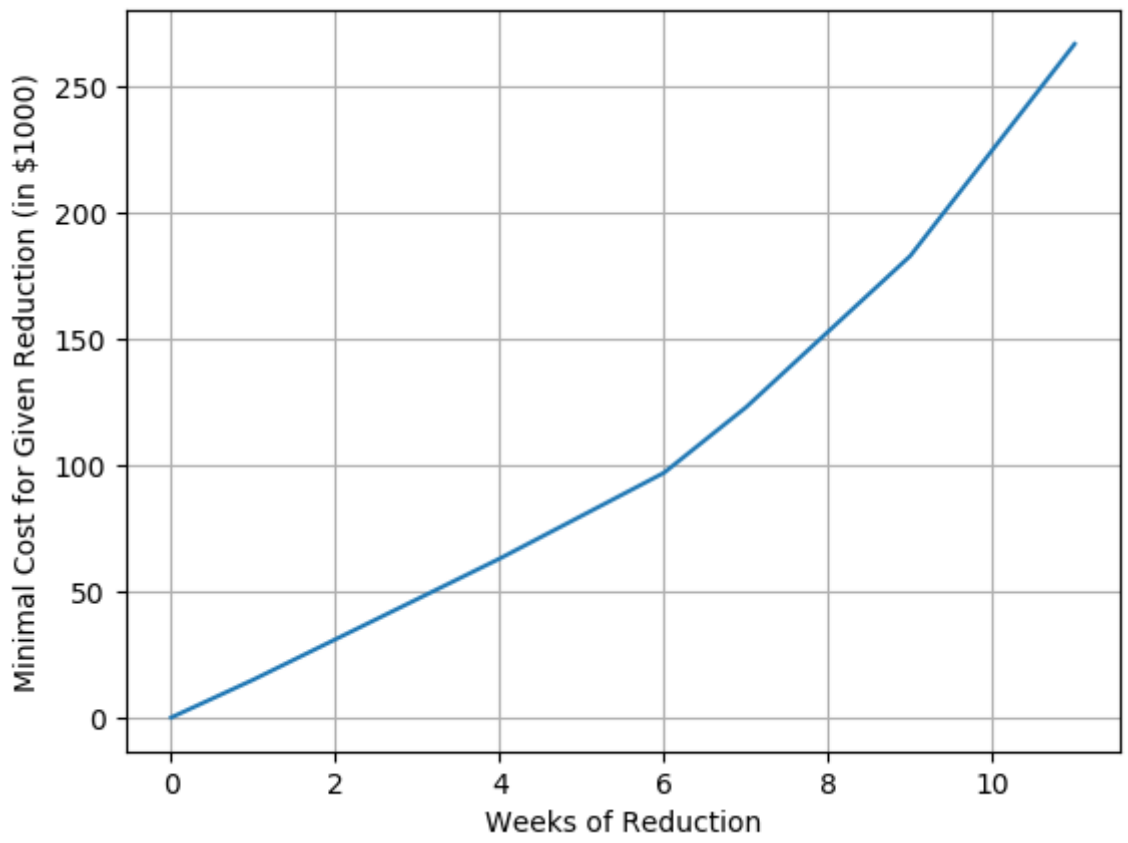
```
Construction end time: 53.0
Reduction cost: $267000.0
```

In [243]:
```
reduction = [ 0, 1,  2,  3,  4,  5,  6,  7,   8,   9,   10,  11, ]
cost      = [ 0, 15, 31, 47, 63, 80, 97, 123, 153, 183, 225, 267]

using PyPlot

grid()
xlabel("Weeks of Reduction")
ylabel("Minimal Cost for Given Reduction (in \$1000)")

plot(reduction, cost)
```



Out[243]:
```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x7fb66d29a7d0>
```

The cost values in the plot above were found by forcing the endtime to a certain value and then calculating the minimal cost to reach the given end date. The code is clumsy, but it worked.

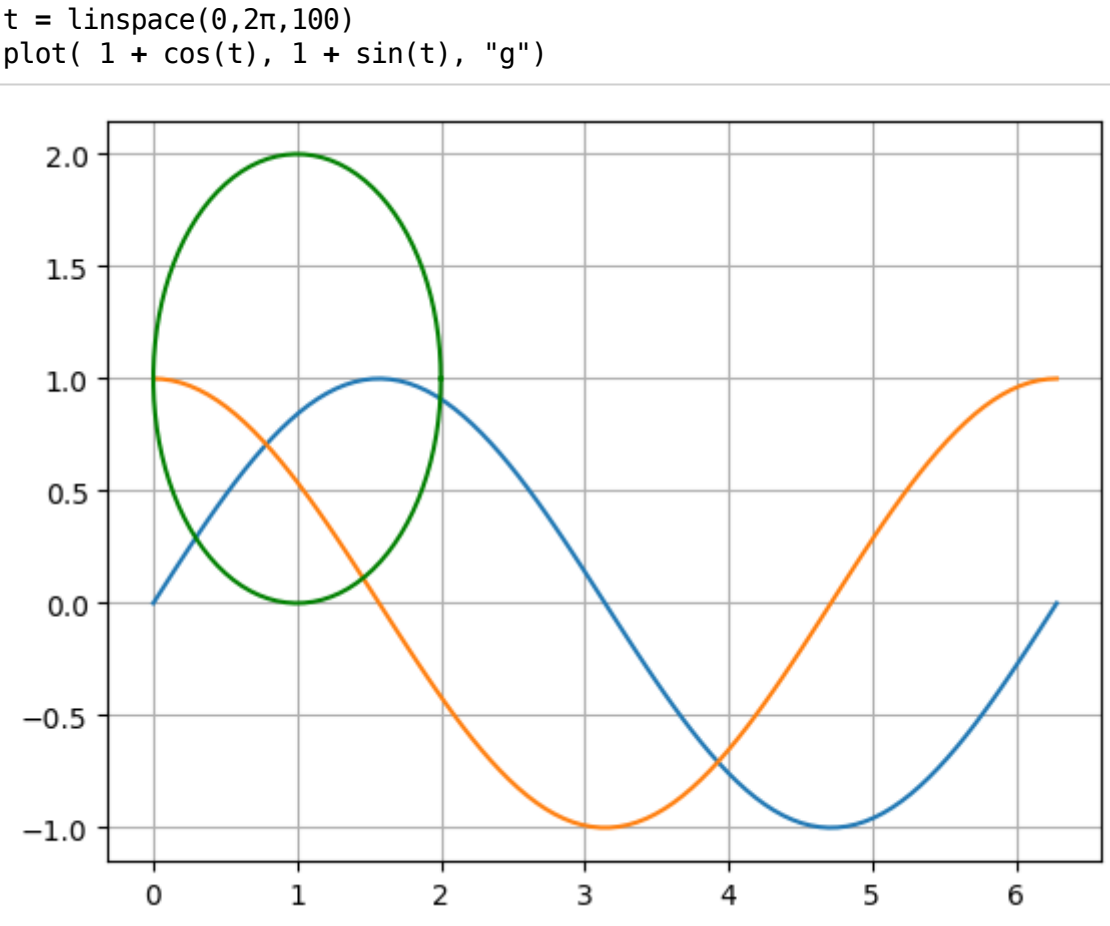
Part (c): Given the minimal reduction costs above, we see that if we try to reduce time more than one week, we incur a cost greater than 30k dollars. Thus we choose to reduce the time by a week, costing us 15k dollars in extra costs, but we recieve a bonus of 30k dollars with a total effect of earning an additional 15k dollars.

Dual Interpretation

In [271]:
```
t = linspace(0,2π,100)
grid()

plot(t, sin(t))
plot(t, cos(t))

t = linspace(0,2π,100)
plot( 1 + cos(t), 1 + sin(t), "g")
```



Out[271]:
```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x7fb66cea6d10>
```

In [269]:
```
A = [ 1  0 -1  0
      0  1  0 -1
     -1  0  1  0
      0 -1  0  1 ]
x = [ p; q; r; s ]
b = [ cost(t); sin(t); cos(t); sin(t) ]
c = [ 1; 1; 1; 1]
```

```
MethodError: objects of type Array{Int64,1} are not callable
Use square brackets [] for indexing an Array.
```

Using the trigonometric identity of:

$$sin^2 + cos^2 = 1$$

We rewrite our contraint as:

$$(p - r)^2 + (q - s)^2 = 1$$

With the additional constraint of p, q, r, and s all being non-negative, this leaves all circles of radius 1 lying completely in the first quadrant. We want to minimize the sum of the coordinates, so choose the circle which is tangent to both the x and y axis. This forces (r,s) to be (1,1). Now we are left with the problem of finding the point on this circle which minimizes the sum of it's components.

The dual is a problem of maximizing the sum four, non-negative constants times cost, sint, cost, and sint respectively. A is symmetric, so it is it's own transpose.

In [270]:
```
A = [ 1  0 -1  0
      0  1  0 -1
     -1  0  1  0
      0 -1  0  1 ]
x = [ l1; l2; l3; l4 ]
b = [ 1; 1; 1; 1 ]
c = [ cost; sint; cost; sint]

#with li >= 0 for i = 1:4
```

```
UndefVarError: l1 not defined
```

Above is the standard form for the dual interpretation. This is very similar to the circle interpretation above, where the difference of two lambdas must be non-negative and the sum lambdas times the trig functions is maximal.