COMP SCI 524

Homework 4

Jingde Wan

Q1

(a)

In [13]:

```
using JuMP

m = Model()

#First layer: store to wands
@variable(m, wands[1:7] >= 00)

#Second layer: wands to owls
@variable(m, 0 <= owls1[1:7] <= 1)
@variable(m, 0 <= owls2[1:7] <= 1)
@variable(m, 0 <= owls3[1:7] <= 1)
@variable(m, 0 <= owls4[1:7] <= 1)
@variable(m, 0 <= owls5[1:7] <= 1)

#Third layer: owls to customer
@variable(m, 0 <= cus1 <= 6)
@variable(m, 0 <= cus2 <= 5)
@variable(m, 0 <= cus3 <= 4)
@variable(m, 0 <= cus4 <= 2)
@variable(m, 0 <= cus5 <= 10)

#add a feedback path
@variable(m, store >= 0)

#first layer blance
for i in 1:7
    @constraint(m, wands[i] == owls1[i]+owls2[i]+owls3[i]+owls4[i]+owls5[i])
end

#second layer balance
@constraint(m, sum(owls1) == cus1)
@constraint(m, sum(owls2) == cus2)
@constraint(m, sum(owls3) == cus3)
@constraint(m, sum(owls4) == cus4)
@constraint(m, sum(owls5) == cus5)

# feedback path balance
@constraint(m, cus1+cus2+cus3+cus4+cus5 == store)
@constraint(m, store == sum(wands))

@objective(m, Max, store)
m
```

...

In [ ]:

```
using Clp
set_optimizer(m, Clp.Optimizer)
optimize!(m)

println("The maximum # of wands that can be delivered: ", objective_value(m))
println("# of wands carried by owls1: ", JuMP.value.(sum(owls1)))
println("# of wands carried by owls2: ", JuMP.value.(sum(owls2)))
println("# of wands carried by owls3: ", JuMP.value.(sum(owls3)))
println("# of wands carried by owls4: ", JuMP.value.(sum(owls4)))
println("# of wands carried by owls5: ", JuMP.value.(sum(owls5)))

println("# of type 1 wands shipped: ", JuMP.value.(wands[1]))
println("# of type 2 wands shipped: ", JuMP.value.(wands[2]))
println("# of type 3 wands shipped: ", JuMP.value.(wands[3]))
println("# of type 4 wands shipped: ", JuMP.value.(wands[4]))
println("# of type 5 wands shipped: ", JuMP.value.(wands[5]))
println("# of type 6 wands shipped: ", JuMP.value.(wands[6]))
println("# of type 7 wands shipped: ", JuMP.value.(wands[7]))
```

(b)

The subset of nodes containing the destination node is {customer node, owl 5 node}. Given that the optimal objective value of primal problem is 24, we know that the optimal objective value of dual is 24. By the above output, we know that the the flow of the arc from owls1,2,3,4 to customer are 6, 5, 4, 2 respectively, so we need extra 7 from wands to owl 5 to acieve 24, which is the optimal objective value.

Q2

(a)

In [14]:

```
# STARTER CODE FOR STIGLER'S DIET PROBLEM
using Pkg
Pkg.add("NamedArrays")
Pkg.add("CSV")
Pkg.add("DataFrames")

using NamedArrays
using CSV
using DataFrames

# import Stigler's data set
raw = CSV.read(joinpath(@__DIR__, "stigler.csv"), DataFrame)
(m,n) = size(raw)

n_nutrients = 2:n       # columns containing nutrients
n_foods = 2:m           # rows containing food names

nutrients = names(raw)[n_nutrients]   # the list of nutrients (convert to 1-D array)
foods = raw[n_foods, 1][:]            # the list of foods (convert to 1-D array)

# lower[i] is the minimum daily requirement of nutrient i.
lower = Dict(zip(nutrients,raw[1, n_nutrients]))

# data[f,i] is the amount of nutrient i contained in food f.
# # Because we cannot construct namedarray directly from data frame
# # we need to convert it into a matrix first
data = convert(Matrix{Float64}, raw[n_foods, n_nutrients])
data = NamedArray(data, (foods, nutrients), ("foods", "nutrients"))
```

...

Let x_i to denote the unit I spend on i-th food. The minimization problem (Primal) can be described as following:

In [15]:

```
using JuMP

model = Model()

@variable(model, x[1:length(n_foods)] >= 0)

@constraint(model, sum(data[i, "Calories (1000)"] * x[i] for i in 1:length(n_foods))
@constraint(model, sum(data[i, "Ascorbic Acid (mg)"] * x[i] for i in 1:length(n_fooc
@constraint(model, sum(data[i, "Calcium (g)"] * x[i] for i in 1:length(n_foods)) >=
@constraint(model, sum(data[i, "Niacin (mg)"] * x[i] for i in 1:length(n_foods)) >=
@constraint(model, sum(data[i, "Thiamine (mg)"] * x[i] for i in 1:length(n_foods)) >
@constraint(model, sum(data[i, "Iron (mg)"] * x[i] for i in 1:length(n_foods)) >= lo
@constraint(model, sum(data[i, "Vitamin A (1000 IU)"] * x[i] for i in 1:length(n_foo
@constraint(model, sum(data[i, "Riboflavin (mg)"] * x[i] for i in 1:length(n_foods))
@constraint(model, sum(data[i, "Protein (g)"] * x[i] for i in 1:length(n_foods)) >=

@objective(model, Min, sum(x[i] for i in 1:length(n_foods)))
model
```

...

In [17]:

```
using Clp
set_optimizer(model, Clp.Optimizer)
optimize!(model)

println("The total cost of cheapest diet: ", objective_value(model)*365)

for i in 1:length(n_foods)
    println(foods[i], ": ", getvalue(x[i]))
end
```

```
The total cost of cheapest diet: 39.66173154546625
Wheat Flour (Enriched): 0.02951906167648827
Macaroni: 0.0
Wheat Cereal (Enriched): 0.0
Corn Flakes: 0.0
Corn Meal: 0.0
Hominy Grits: 0.0
Rice: 0.0
Rolled Oats: 0.0
White Bread (Enriched): 0.0
Whole Wheat Bread: 0.0
Rye Bread: 0.0
Pound Cake: 0.0
Soda Crackers: 0.0
Milk: 0.0
Evaporated Milk (can): 0.0
Butter: 0.0
Oleomargarine: 0.0
Eggs: 0.0
Cheese (Cheddar): 0.0
Cream: 0.0
Peanut Butter: 0.0
Mayonnaise: 0.0
Crisco: 0.0
Lard: 0.0
Sirloin Steak: 0.0
Round Steak: 0.0
Rib Roast: 0.0
Chuck Roast: 0.0
Plate: 0.0
Liver (Beef): 0.0018925572907052643
Leg of Lamb: 0.0
Lamb Chops (Rib): 0.0
Pork Chops: 0.0
Pork Loin Roast: 0.0
Bacon: 0.0
Ham, smoked: 0.0
Salt Pork: 0.0
Roasting Chicken: 0.0
Veal Cutlets: 0.0
Salmon, Pink (can): 0.0
Apples: 0.0
Bananas: 0.0
Lemons: 0.0
Oranges: 0.0
Green Beans: 0.0
Cabbage: 0.011214435246144865
Carrots: 0.0
Celery: 0.0
```

```
Lettuce: 0.0
Onions: 0.0
Potatoes: 0.0
Spinach: 0.005007660466725203
Sweet Potatoes: 0.0
Peaches (can): 0.0
Pears (can): 0.0
Pineapple (can): 0.0
Asparagus (can): 0.0
Green Beans (can): 0.0
Pork and Beans (can): 0.0
Corn (can): 0.0
Peas (can): 0.0
Tomatoes (can): 0.0
Tomato Soup (can): 0.0
Peaches, Dried: 0.0
Prunes, Dried: 0.0
Raisins, Dried: 0.0
Peas, Dried: 0.0
Lima Beans, Dried: 0.0
Navy Beans, Dried: 0.061028563526693246
Coffee: 0.0
Tea: 0.0
Cocoa: 0.0
Chocolate: 0.0
Sugar: 0.0
Corn Syrup: 0.0
Molasses: 0.0
Strawberry Preserves: 0.0
Coin0506I Presolve 9 (0) rows, 76 (-1) columns and 569 (-1) elements
Clp0006I 0  Obj 0 Primal inf 5.1310537 (9)
Clp0006I 6  Obj 0.10866228
Clp0000I Optimal - objective value 0.10866228
Coin0511I After Postsolve, objective 0.10866228, infeasibilities - dua
l 0 (0), primal 0 (0)
Clp0032I Optimal objective 0.1086622782 - 6 iterations time 0.002, Pre
solve 0.00
```

Then we construct the dual problem to the primal:

In [18]:

```
using JuMP

m_dual = Model()

@variable(m_dual, y[1:length(n_nutrients)] >= 0)

for i in 1:length(n_foods)
    @constraint(m_dual, sum(data[i,j] * y[j] for j in 1:length(n_nutrients)) <= 1)
end

@objective(m_dual, Max, sum(y[j] * lower[nutrients[j]] for j in 1:length(n_nutrients
m_dual
```

...

In [19]:

```
using Clp
set_optimizer(m_dual, Clp.Optimizer)
optimize!(m_dual)

println("The total cost of cheapest diet: ", objective_value(m_dual)*365)

for i in 1:length(n_nutrients)
    println(nutrients[i], ": ", getvalue(y[i]))
end
```

```
The total cost of cheapest diet: 39.661731545466274
Calories (1000): 0.008765147298049485
Protein (g): 0.0
Calcium (g): 0.03173771344563715
Iron (mg): 0.0
Vitamin A (1000 IU): 0.00040023272172538176
Thiamine (mg): 0.0
Riboflavin (mg): 0.016358032699276687
Niacin (mg): 0.0
Ascorbic Acid (mg): 0.00014411751545899702
Coin0506I Presolve 32 (-45) rows, 9 (0) columns and 251 (-319) element
s
Clp0006I 0  Obj -0 Dual inf 120.70151 (9)
Clp0006I 12  Obj 0.10866228
Clp0000I Optimal - objective value 0.10866228
Coin0511I After Postsolve, objective 0.10866228, infeasibilities - dua
l 0 (0), primal 0 (0)
Clp0032I Optimal objective 0.1086622782 - 12 iterations time 0.002, Pr
esolve 0.00
```

In [20]:

```
println("What I am willing to pay per pill: ", 0.03173771344563715*(500/1000))
```

```
What I am willing to pay per pill: 0.015868856722818576
```

(2)

In [21]:

```
# STARTER CODE FOR QUESTION 2B

foods2 = [foods; "Calcium Pills"];

data2 = [data; [0 0 0.5/0.01 0 0 0 0 0 0]];
setnames!(data2, foods2, 1);

data2
```

Out[21]:

78×9 Named Array{Float64,2}

A \ nutrients	Calories (1000)	…	Ascorbic Acid (mg)
Wheat Flour (Enriched)	44.7	…	0.0
Macaroni	11.6		0.0
Wheat Cereal (Enriched)	11.8		0.0
Corn Flakes	11.4		0.0
Corn Meal	36.0		0.0
Hominy Grits	28.6		0.0
Rice	21.2		0.0
Rolled Oats	25.3		0.0
White Bread (Enriched)	15.0		0.0
Whole Wheat Bread	12.2		0.0
Rye Bread	12.4		0.0
Pound Cake	8.0		0.0
⋮	⋮	⋱	⋮
Peas, Dried	20.0		0.0
Lima Beans, Dried	17.4		0.0
Navy Beans, Dried	26.9		0.0
Coffee	0.0		0.0
Tea	0.0		0.0
Cocoa	8.7		0.0
Chocolate	8.0		0.0
Sugar	34.9		0.0
Corn Syrup	14.7		0.0
Molasses	9.0		0.0
Strawberry Preserves	6.4		0.0
Calcium Pills	0.0	…	0.0

In [22]:

```
using JuMP

model1 = Model()

@variable(model1, x[1:length(n_foods)+1] >= 0)

@constraint(model1, sum(data2[i, "Calories (1000)"] * x[i] for i in 1:length(n_foods
@constraint(model1, sum(data2[i, "Ascorbic Acid (mg)"] * x[i] for i in 1:length(n_fo
@constraint(model1, sum(data2[i, "Calcium (g)"] * x[i] for i in 1:length(n_foods)+1)
@constraint(model1, sum(data2[i, "Niacin (mg)"] * x[i] for i in 1:length(n_foods)+1)
@constraint(model1, sum(data2[i, "Thiamine (mg)"] * x[i] for i in 1:length(n_foods)+
@constraint(model1, sum(data2[i, "Iron (mg)"] * x[i] for i in 1:length(n_foods)+1) >
@constraint(model1, sum(data2[i, "Vitamin A (1000 IU)"] * x[i] for i in 1:length(n_f
@constraint(model1, sum(data2[i, "Riboflavin (mg)"] * x[i] for i in 1:length(n_foods
@constraint(model1, sum(data2[i, "Protein (g)"] * x[i] for i in 1:length(n_foods)+1)

@objective(model1, Min, sum(x[i] for i in 1:length(n_foods)+1))
model1
```

...

In [23]:

```
using Clp
set_optimizer(model1, Clp.Optimizer)
optimize!(model1)

println("The total cost of cheapest diet: ", objective_value(model1)*365)

for i in 1:length(n_foods)+1
    println(foods2[i], ": ", getvalue(x[i]))
end
```

```
The total cost of cheapest diet: 36.9982473745081
Wheat Flour (Enriched): 0.06598060307911847
Macaroni: 0.0
Wheat Cereal (Enriched): 0.0
Corn Flakes: 0.0
Corn Meal: 0.0
Hominy Grits: 0.0
Rice: 0.0
Rolled Oats: 0.0
White Bread (Enriched): 0.0
Whole Wheat Bread: 0.0
Rye Bread: 0.0
Pound Cake: 0.0
Soda Crackers: 0.0
Milk: 0.0
Evaporated Milk (can): 0.0
Butter: 0.0
Oleomargarine: 0.0
Eggs: 0.0
Cheese (Cheddar): 0.0
Cream: 0.0
Peanut Butter: 0.0
Mayonnaise: 0.0
Crisco: 0.0
Lard: 0.0
Sirloin Steak: 0.0
Round Steak: 0.0
Rib Roast: 0.0
Chuck Roast: 0.0
Plate: 0.0
Liver (Beef): 0.00784433892120114
Leg of Lamb: 0.0
Lamb Chops (Rib): 0.0
Pork Chops: 0.0
Pork Loin Roast: 0.0
Bacon: 0.0
Ham, smoked: 0.0
Salt Pork: 0.0
Roasting Chicken: 0.0
Veal Cutlets: 0.0
Salmon, Pink (can): 0.0
Apples: 0.0
Bananas: 0.0
Lemons: 0.0
Oranges: 0.0
Green Beans: 0.0
Cabbage: 0.011195027632464827
Carrots: 0.0
Celery: 0.0
```

```
Lettuce: 0.0
Onions: 0.0
Potatoes: 0.0
Spinach: 0.003911295356684479
Sweet Potatoes: 0.0
Peaches (can): 0.0
Pears (can): 0.0
Pineapple (can): 0.0
Asparagus (can): 0.0
Green Beans (can): 0.0
Pork and Beans (can): 0.0
Corn (can): 0.0
Peas (can): 0.0
Tomatoes (can): 0.0
Tomato Soup (can): 0.0
Peaches, Dried: 0.0
Prunes, Dried: 0.0
Raisins, Dried: 0.0
Peas, Dried: 0.0
Lima Beans, Dried: 0.0
Navy Beans, Dried: 0.0
Coffee: 0.0
Tea: 0.0
Cocoa: 0.0
Chocolate: 0.0
Sugar: 0.0
Corn Syrup: 0.0
Molasses: 0.0
Strawberry Preserves: 0.0
Calcium Pills: 0.012433796310553268
Coin0506I Presolve 9 (0) rows, 77 (-1) columns and 570 (-1) elements
Clp0006I 0  Obj 0 Primal inf 4.7027054 (9)
Clp0006I 5  Obj 0.10136506
Clp0000I Optimal - objective value 0.10136506
Coin0511I After Postsolve, objective 0.10136506, infeasibilities - du
al 0 (0), primal 0 (0)
Clp0032I Optimal objective 0.1013650613 - 5 iterations time 0.002, Pr
esolve 0.00
```

In [24]:

```
println("money saved on a year basis: ", 39.66173154546625-36.9982473745081)
```

```
money saved on a year basis: 2.6634841709581565
```

So the new diet help us save $2.66 per year.

Q3

(a)

In [25]:

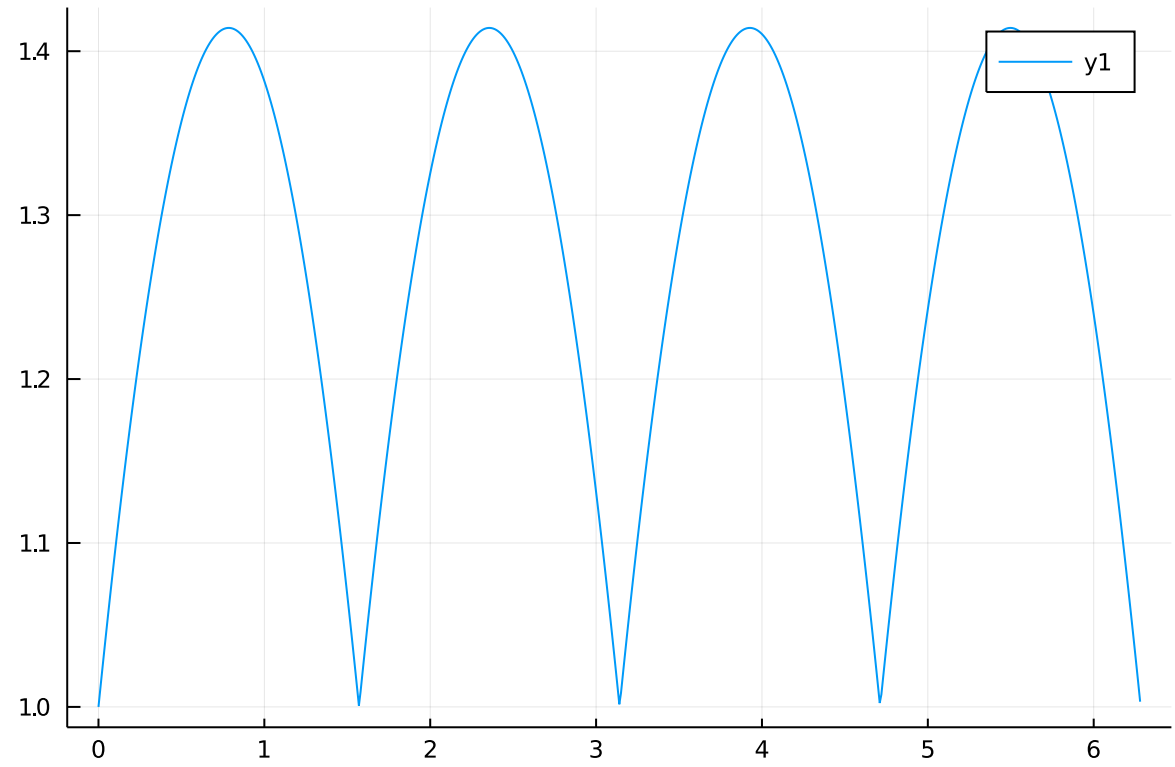
```
using JuMP, Clp

ylist = []

for t in (0:200pi)/100

    m3 = Model(Clp.Optimizer)

    @variable(m3, p >= 0)
    @variable(m3, q >= 0)
    @variable(m3, r >= 0)
    @variable(m3, s >= 0)

    @constraint(m3, p - r == cos(t))
    @constraint(m3, q - s == sin(t))

    @objective(m3, Min, p+q+r+s)

    optimize!(m3)

    push!(ylist, JuMP.objective_value(m3))
end
```

...

In [26]:

```
using Plots
plot((0:200pi)/100, ylist)
```

Out[26]:



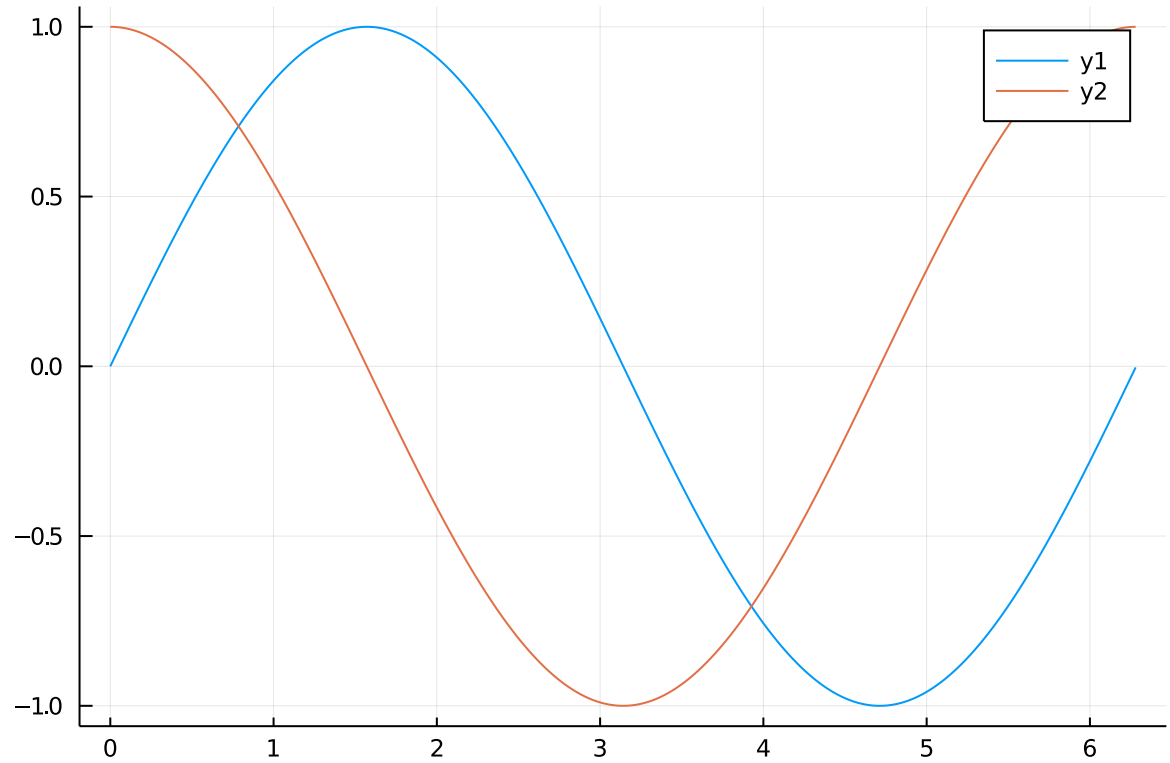
The optimal objective of this LP can be expressed as a function of t with period $\frac{\pi}{2}$. There are totally 4

situations.

In [7]:

```
using Plots
sinlist = []
coslist = []

for x in (0:200pi)/100
    push!(sinlist, sin(x))
    push!(coslist, cos(x))
end


plot((0:200pi)/100, sinlist)
plot!((0:200pi)/100, coslist)
```

Out[7]:



1) When $cos(t) > 0$ and $sin(t) > 0$, $0 < t < \frac{\pi}{2}$

So $p \geq 0$ and $q \geq 0$ since $r \geq 0$ and $s \geq 0$

To minimize p+q+r+s, let r=s=0. So the objective value is $p + q = sin(t) + cos(t)$ which is the same as first $\frac{1}{4}$ of the graph above

2) When $cos(t) < 0$ and $sin(t) > 0$, $\frac{\pi}{2} < t < \pi$

To minimize p+q+r+s, let s=p=0. So the objective value is $r + q = sin(t) - cos(t)$ which is the same as second $\frac{1}{4}$ of the graph above

3) When $cos(t) < 0$ and $sin(t) < 0$, $\pi < t < \frac{3\pi}{2}$

To minimize p+q+r+s, let q=p=0. So the objective value is $r + s = -sin(t) - cos(t)$ which is the same as third $\frac{1}{4}$ of the graph above

4) When $cos(t) > 0$ and $sin(t) < 0$, $\frac{3\pi}{2} < t < 2\pi$

To minimize p+q+r+s, let r=q=0. So the objective value is $p + s = -sin(t) + cos(t)$ which is the same as last $\frac{1}{4}$ of the graph above

(b)

Dual is shown below:

$$\max_{a,b} \quad cos(t) * a + sin(t) * b$$

$$subject\ to: \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

In [10]:

```
ylist_dual = []

for t in (0:200pi)/100

    m3d = Model(Clp.Optimizer)

    @variable(m3d, a)
    @variable(m3d, b)

    @constraint(m3d, a <= 1)
    @constraint(m3d, b <= 1)
    @constraint(m3d, -a <= 1)
    @constraint(m3d, -b <= 1)

    @objective(m3d, Max, a*cos(t)+b*sin(t))

    optimize!(m3d)

    push!(ylist_dual, JuMP.objective_value(m3d))
end
```

...

In [12]:

```
plot((0:200pi)/100, ylist_dual)
```

Out[12]: