Pitt INFORMS `Python` and `Julia` Tutorial

Hamdy Salman, and Wei Wang, and Tomás Lagos

In this tutorial we introduce `Python` and `Julia` for mathematical programming. To carry this out, make sure all the software is properly installed in your favorite platform[1].

- First install the latest Gurobi version (8.1.1 at the time of this tutorial)[2], you may have to register first if you have not already. Then access the license with your account and set it in your machine, using the command line with the **grbgetkey** command.
- Get `Anaconda` for Python 3.7 (long term support is recommended)[3], and follow the instructions for your specific OS.
- Get the lastest (or long term support) version of Julia (here 1.2.0, 1.0.5 for long term support)[4].

Solving Problem by `Python`

Setting up `Python`

To be able to use Python to invoke Gurobi three major steps are required

- Activating Gurobi License
- Install Anaconda
- Install Gurobi into Anaconda

please follow the instructions at http://www.gurobi.com/downloads/get-anaconda to install and register a free academic license.
The main power of using Python is the availability of data packages, simulation packages, design of experiment and statistical packages, and plotting packages. here is a list of packages I use:

- gurobipy: This is the Gurobi package that allows the user to invoke the optimizer from a Python script
- pandas: This package is the equivalent of R in Python, it contains Data structures that can be very helpful to store deal with data
- matplotlib: This package provides powerful potting functions

This Tutorial will be using Spyder IDE to write Python scripts, Spyder will be automatically installed when Anaconda is installed.

[1] Make sure you use the links in the footnotes for an easier installation process
[2] http://www.gurobi.com/downloads/
[3] https://www.gurobi.com/get-anaconda/
[4] https://julialang.org/downloads/

A Small Example

Consider the problem with two variables and two constraints

$$\begin{aligned} \max \quad & 3x + 4y && (1) \\ \text{s.t.} \quad & 5x + 2y \leq 10 && (2) \\ & 3x + 5y \leq 12 && (3) \\ & x \geq 0, y \geq 0 && (4) \end{aligned}$$

It can be solved with the following Python codes

```
"""
Modified on Tue Oct 29 21:34:30 2019
@author: tol28
Created on Mon Nov 13 20:13:09 2017
@author: hamdy
"""


from gurobipy import * #Calls Gurobi Package
import numpy as np #Calls numpy Package


#Is better to create models within functions:
    #This allows the momory of the model to be
    #released once the function returns

def solve_example_1(a1x,a1y,a2x,a2y,b1,b2,cx,cy,debug=0):
    #Model Definition
    m= Model('Example 1')

    #Varriables
    x = m.addVar(vtype=GRB.CONTINUOUS,lb=0, name ='x')
    y = m.addVar(vtype=GRB.CONTINUOUS,lb=0, name ='y')
    m.update()

    #Constraints
#     m.addConstr(5*x+2*y <= 10)
#     m.addConstr(3*x+5*y <= 12)
    con1=m.addConstr(a1x*x+a1y*y <= b1,name='c1')
    con2=m.addConstr(a2x*x+a2y*y <= b2,name='c2')

    #Objective Function
#     m.setObjective(3*x + 4*y, GRB.MAXIMIZE)
    m.setObjective(cx*x + cy*y, GRB.MAXIMIZE)
    m.optimize()

    if m.status == GRB.Status.OPTIMAL:
        if debug: print('Optimal objective: %g' % m.objVal)
        #You can also get the reference of variables or
        # constraints by the name you gave them at its creation
        return (m.objVal, m.getVarByName("x").x, m.getVarByName("y").x,
                m.getConstrByName("c1").pi, m.getConstrByName("c2").pi);
    elif m.status == GRB.Status.INFEASIBLE:
        if debug: print('Model is infeasible')
```

```
        m.computeIIS()
        if debug: print('\nThe following constraint(s) cannot be satisfied:')
        for c in m.getConstrs():
            if c.IISConstr:
                if debug: print('%s' % c.constrName)
                if debug: print('')
        return None
    elif m.status == GRB.Status.UNBOUNDED:
        if debug: print('Model is unbounded')
        return None
#    elif m.status == GRB.Status.INF_OR_UNBD:
#        if debug: print('Model is infeasible or unbounded')
#        return None
    else:
        if debug: print('Optimization ended with status %d' % model.status)
        return None



result=solve_example_1(5,2,3,5,10,12,3,4)
print()
if result != None:
    print("Objective=",result[0])
    print("x=",result[1])
    print("y=",result[2])
    print("d1=",result[3])
    print("d2=",result[4])
```

Variables

When creating variables, we always need to associate them with models. Variables for a model named `m` can be created as follow

```
#a Cont. Variable thats is greater than zero
x = m.addVar(vtype=GRB.CONTINUOUS,lb=0, name ='x')
#a Cont. Variable thats is greater than 2 and less than 5
x = m.addVar(vtype=GRB.CONTINUOUS,lb=2,ub=5, name ='x')
#an Integer. Variable thats is greater than 2 and less than 5
x = m.addVar(vtype=GRB.INTEGER,lb=2,ub=5, name ='x')
x = m.addVar(vtype=GRB.BINARY, name ='x') #a Binary. Variable
x = m.addVar(vtype=GRB.BINARY, name ='x') #a Binary. Variable
# defines a list of 10 varriables
x = [m.addVar(vtype=GRB.BINARY, name ='x%d' % J) for J in range(10)]
# defines a Series of 10 varriables
x = pd.Series([m.addVar(vtype=GRB.BINARY, name ='x%d' % J) for J in range(10)],
                                index=  [i for i in range(10)])
```

You can also create a multi-dimensional variable

```
#Adding $x_{ijk}$ using Dataframes"
Index = [(i,j,k) for i in range (5) for j in range(5) for k in range(7)]
var = [m.addVar(lb=0, vtype=GRB.CONTINUOUS, name = "X"+str(i)) for i in Index]
demand = [np.random.randint(300,700) for i in Index]
x = pd.DataFrame({'x':var, 'demand':demand},
index = pd.MultiIndex.from_tuples(Index, names=['i', 'j','k']))
```

using pandas can make it easier to read excel files, as an example assume that the demand is stored in an excel file.

```
#Reading Excel files"
InputPath = 'C:\\Users\\hamdy\\Desktop\\Station Location v4\\Gurobi_Tutorial.xlsx'
# Python uses \\ instead of \
xl = pd.ExcelFile(InputPath)
demand = xl.parse("Sheet1")
```

After adding variables, constraints, or changing parameters of the model m, the model needs to be updated using m.update(), this is can be time consuming to limit the use of m.update in your model.

Constraints

Constraints also need to be associated with models. Sometimes we may need references for them, but references are not necessary. For a model named `m`, we can created constraints as follow

```
#Different ways to add constraints
m.addConstr(x+y>0)
m.addConstrs((x[i]>=0 for i in range(5)), name='Constraint')
m.addConstrs(x.sum(i, '*') == [0, 2] for i in [1, 2, 4])
m.addConstrs(z[i] == max_(x[i], y[i]) for i in range(5))
#Constraints can also be added using pandas' functions
m.addConstrs((x.loc[[(i,j,k) for i in range(5)],"x"].values).sum()<=10*x.loc[(i,j,k),
                                  'demand'] for j in range(5) for k in range(7))
```

Tuning a model

```
"""
Created on Fri Jun  7 16:37:11 2019

@author: tomas
"""
import time
from gurobipy import *
import os
import numpy as np

def read_sudoku_file(filename):
    f = open(filename)

    grid=[]
    l=f.readlines();
    for (i,ln) in enumerate(l):
        row=ln.replace(" \n","").split(" ");
#         row=ln.split(",");
        grid.append([]);
        for j in range(len(row)):
            grid[i].append(int(row[j]));
    f.close()

    #grid = f.read().split(",")

    s = sum([len(S) for S in grid]);#len(grid[0])
```

```
    n = int(s**0.5);


    # Create our 3-D array of model variables

    #model = Model('sudoku')

    #vars = model.addVars(n,n,n, vtype=GRB.CONTINUOUS, lb=0, ub=1 , name='G')
    #vars = model.addVars(n,n,n, vtype=GRB.BINARY, name='G')


    # Fix variables associated with cells whose values are pre-specified
    A=[];b=[];
    for i in range(n):
        for j in range(n):
            if grid[i][j] != 0:
                v = int(grid[i][j]) - 1
    #            vars[i,j,v].LB = 1
                l=len(A)
                A+=[[0 for k in range( n**3)]];
                A[l][i*n**2 + j*n + v]=1;b+=[1];



    # Each cell must take one value

    for i in range(n):
        for j in range(n):
            l=len(A);
            A+=[[0 for k in range( n**3)]];
            b+=[1]
            for k in range(n):
                A[l][i * n**2 + j * n + k]=1;


    #model.addConstrs((vars.sum(i,j,'*') == 1
    #                  for i in range(n)
    #                  for j in range(n)), name='V')

    # Each value appears once per row

    for i in range(n):
        for j in range(n):
            l=len(A);
            A+=[[0 for k in range( n**3)]];
            b+=[1]
            for k in range(n):
                A[l][j * n**2 + k * n + i]=1;

    #model.addConstrs((vars.sum(i,'*',v) == 1
    #                  for i in range(n)
    #                  for v in range(n)), name='R')

    # Each value appears once per column
```

```
    for i in range(n):
        for j in range(n):
            l=len(A);
            A+=[[0 for k in range( n**3)]];
            b+=[1]
            for k in range(n):
                A[l][k * n**2 + i * n + j]=1;

    #model.addConstrs((vars.sum('*',j,v) == 1
    #                 for j in range(n)
    #                 for v in range(n)), name='C')
    #

    # Each value appears once per subgrid
    n05=int(n**0.5);
    for i1 in range(n05):
        for j1 in range(n05):
            for k in range(n):
                l=len(A);
                A+=[[0 for m in range( n**3)]];
                b+=[1]
                for i2 in range(i1*n05,(i1+1)*n05):
                    for j2 in range(j1*n05,(j1+1)*n05):
                        A[l][i2 * n**2 + j2 * n + k]=1;
    return A,b


def solRELX_Ax_e_b(A,b,c,j,debug):#x is restricted positive, problem is maximization
    m = Model("sudoku"+str(j+1));
    m.setParam( 'OutputFlag', debug)
    # Create variables
    x={};
    C={}
    for i in range(len(A[0])):
#        x[i] = m.addVar(vtype=GRB.CONTINUOUS, lb=0,obj=c[i]);
        x[i] = m.addVar(vtype=GRB.BINARY, lb=0,obj=c[i]);
    for (i,a) in enumerate(A):
        C[i]=m.addConstr(quicksum(a[k]*x[k] for k in range(len(a))) == b[i]);

    m.update();

    t=time.time()
    m.optimize();
    t=time.time()-t;
    if m.status == GRB.Status.OPTIMAL:
        if debug: print('Optimal objective: %g' % m.objVal)
        m.write("models/out"+str(j+1)+".lp")
        return [x[i].x for i in range(len(c))],t;
    elif m.status == GRB.Status.INF_OR_UNBD:
        if debug: print('Model is infeasible or unbounded')
        return None
    elif m.status == GRB.Status.INFEASIBLE:
```

```
        if debug: print('Model is infeasible')
        m.computeIIS()
        if debug: print('\nThe following constraint(s) cannot be satisfied:')
        for c in m.getConstrs():
            if c.IISConstr:
                if debug: print('%s' % c.constrName)
                if debug: print('')
        return None
    elif m.status == GRB.Status.UNBOUNDED:
        if debug: print('Model is unbounded')
        return None
    else:
        if debug: print('Optimization ended with status %d' % model.status)
        return None



tm=[]
debug=0
for i in range(100):
    A,b=read_sudoku_file("instances_sudoku/"+str(i+1)+"-Sudoku.txt")
    resuts=solRELX_Ax_e_b(A,b,[0 for i in range(len(A[0]))],i,debug);
    if resuts==None: continue;
    x,t=resuts;
    tm.append(t);
    if debug:
        n=9;
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if x[i * n**2 + j * n + k]>0.99:
                        if j%3==0 and j!=0:
                            print("|",end="")
                        print("|",k+1,end="")
                if len([x[i * n**2 + j * n + k] for k in range(n) if 0.01<
                x[i * n**2 + j * n + k] and x[i * n**2 + j * n + k]<0.99])>0:
                    if j%3==0 and j!=0:
                        print("|",end="")
                    print("|",0,end="")
            print("|",end=".")
            print("");
            if (i+1)%3==0 and i!=0:
                print("----------------------------------")


ttm=[]
for i in range(100):
    model = read(os.getcwd()+"/models/out"+str(i+1)+".lp");
    model.setParam( 'OutputFlag', debug)
    # Set the TuneResults parameter to 1
    model.Params.tuneResults = 1
    # Tune the model
    model.tune()
    if model.tuneResultCount > 0:
```

```python
        # Load the best tuned parameters into the model
        model.getTuneResult(0)
        # Write tuned parameters to a file
        model.write("tuned/tune"+str(i+1)+".prm")
        # Solve the model using the tuned parameters
        t=time.time()
        model.optimize()
        ttm.append(time.time()-t);

print("Average solution time=",np.mean(tm)/60," minutes.");
print("Average (tuned) solution time=",np.mean(ttm)/60," minutes.");
```

Solving Problem by Julia

Setting up Julia

Start Julia, you should get something similar to Figure 1. To install the packages type "] " in the console, then an interface similar to the second line in Figure 1 should appear.

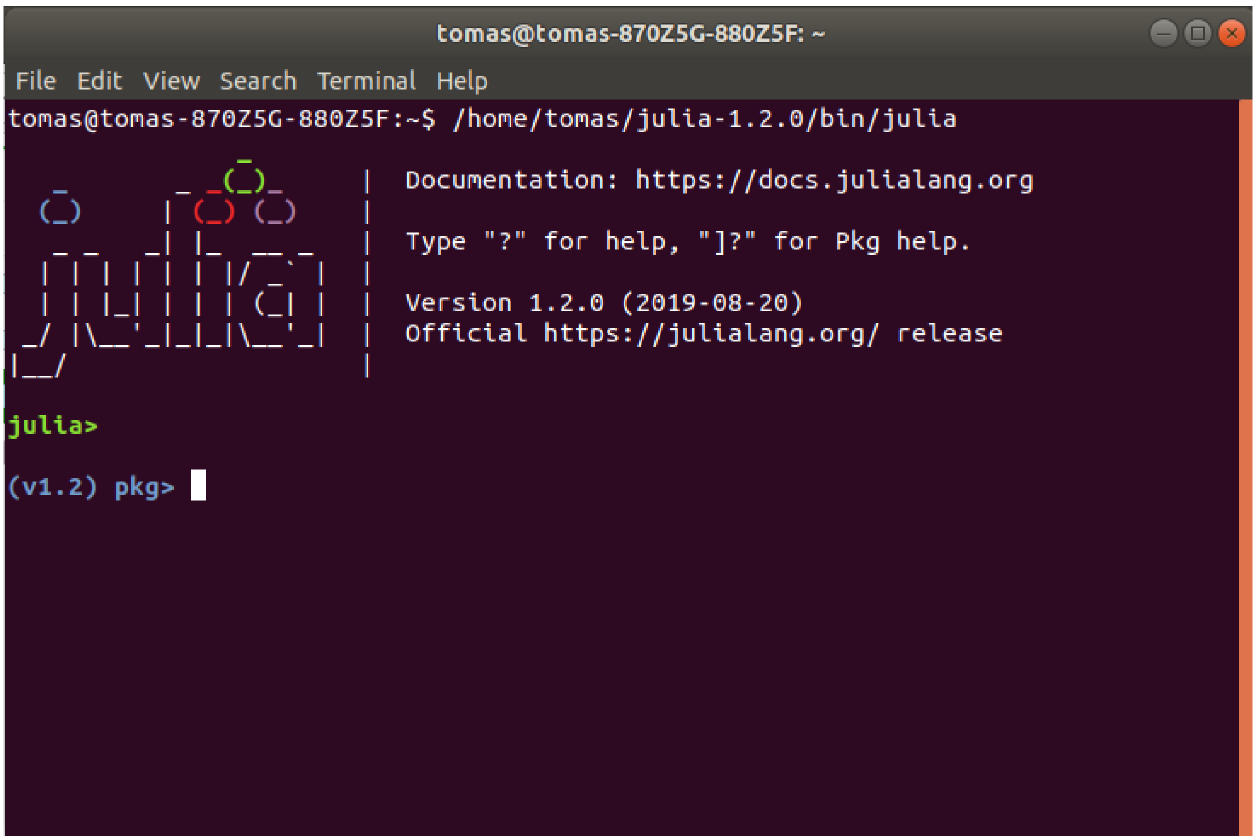

Figure 1: Starting Julia

Once you are in the packages interface, add the packages as follows (you can add all of them in a single line, the reason we did it different here is due to space):

```
    add JuMP Gurobi MathOptInterface HypothesisTests Distributions PyCall
    add Optim Pajarito Alpine Convex  TravelingSalesmanHeuristics MathProgBase
    add LinearAlgebra GLPK Cbc CPLEX
```

A Small Example

Consider the problem with two variables and two constraints

$$\begin{aligned} \max \quad & 3x + 4y && (5)\\ \text{s.t.} \quad & 5x + 2y \leq 10 && (6)\\ & 3x + 5y \leq 12 && (7)\\ & x \geq 0, y \geq 0 && (8) \end{aligned}$$

It can be solved with the following codes

```
using JuMP,Gurobi # we are using packages JuMP and Gurobi
const MOI = JuMP.MathOptInterface
m = Model(with_optimizer(Gurobi.Optimizer))
# create a model named m and we are using Gurobi to solve it

@variable(m,x>=0)
@variable(m,y>=0)
# create two nonnegative variables x,y and associate them with model m

@objective(m,Max,3x+4y)
# create a maximization objective function and associate it with model m

m1=@constraint(m,5x+2y<=10)
m2=@constraint(m,3x+5y<=12)
# create constraints and associate them with model m

print(m) # print out the model

status=optimize!(m) # Solve the model

println("Objective value: ",JuMP.objective_value(m))
println("x = ",JuMP.value(x))
println("y = ",JuMP.value(y))
println("dual m1=",JuMP.dual(m1))
println("dual m2=",JuMP.dual(m2))
# show optimal value and solution
```

Models

Models can be created by

```
model=Model(with_optimizer((solver).Optimizer(Option1=Value1,...),env))
```

`solver` is the name of the solver used to solve this model. In this tutorial we are using `Gurobi` so the solver name is `Gurobi.Optimizer`. All options are solver dependent parameters. Below are some example when using `Gurobi`.

```
ModelName=Model(with_optimizer(Gurobi.Optimizer(TimeLimit=300)))
# set time limit to 5 mins

ModelName=Model(with_optimizer(Gurobi.Optimizer(MIPGap=1e-5,IntFeasTol=1e-6)))
# change MIP gap and integrality tolerance
# MIPgap determines when MIP problems are considered solved to optimal
# IntFeasTol determines when solutions are considered integeral
```

```
ModelName=Model(with_optimizer(Gurobi.Optimizer(Cuts=0,BranchDir=-1)))
# turn off all cuts and do depth first search on branch and bound tree
```

Variables

When creating variables, we always need to associate them with models. Variables for a model named `m` can be created as follow

```
@variable(m,x) # free variable
@variable(m,x>=lb) # variable with lower bound lb
@variable(m,x<=ub) # variable with upper bound ub
@variable(m,lb<=x<=ub) # variable with lower and upper bounds
@variable(m,x[1:M,1:N],Bin) # M by N matrix of binary variables
@variable(m,x[i=1:M]>=2i,Int) # integer variable array with lower bounds
```

You can also create variables in a block

```
@variables m begin
    x>=0
    0<=y<=5
    X[1:10],Bin
    Y[i=1:5,j=1:5]>=i+j,Int
end
```

Constraints

Constraints also need to be associated with models. Sometimes we may need references for them, but references are not necessary. For a model named `m`, we can created constraints as follow

```
@constraint(m,x-y>=0)
@constraint(m,sum(x[i] for i=1:5)==1)
@constraint(m,ConRef[i=1:3],x[i]>=y) # constraints with reference
@constraint(m,ConRef[i=1:5,j=1:5;i>=j],x[i]-y[j]>=0)
# only one condition can be added
# use logical operators && and || for complex conditions
```

Constraints can also be created in a block

```
@constraints(m,begin
  x>=0
  y-z<=3
  ConRef[i=1:3],x[i]>=y
end)
```

Getting Results

Model can be solved by

```
optimize!(ModelName)
if termination_status(ModelName)!=MOI.INFEASIBLE
    println("Model is not Infeasible!")
end
```

1 A more involved example: Python and Cutting Planes for STSP

Symetric Travelling Salesman Problem.
Let $G = (V, E)$ be an undirected graph, where V and E represent the vertex set and edge set, respectively. Each edge has an associated cost $c_e \geq 0$. A tour is a cycle that visits once each of the nodes of the graph. The objective of the problem is to find the tour that minimizes the sum of the costs of the edges used. Let $\delta(S) = \{(i,j) \in E : i \in S,\ j \in V \setminus S\}$ be a *cut* of S, where $S \subseteq V$. Consider the following formulation for the STSP:

$$\begin{array}{lll} & \min \sum_{e\in E} c_e x_e & \\ s.t & & \\ & \sum_{e\in\delta(\{i\})} x_e = 2 & \forall i \in V \\ & \sum_{e\in\delta(S)} x_e \geq 2 & \forall S \subset V,\ |S| \geq 2 \\ & x_e \in \{0,1\} & \forall e \in E \end{array} \tag{9}$$

The decision variable x_e is equal to 1 if the edge is included in the tour, 0 otherwise.

One approach to tackle this problem is to relax the Subtours Elimination Constraints, solve the problem, then add the violated constraints of the original problem to the relaxed problem and solve it again until no restrictions are violated. The following implementation develops in this simple idea using lazy constraints in Python + Gurobi.

```

#!/usr/bin/python

# Copyright 2017, Gurobi Optimization, Inc.

# Solve a traveling salesman problem on a randomly generated set of
# points using lazy constraints.   The base MIP model only includes
# 'degree-2' constraints, requiring each node to have exactly
# two incident edges.  Solutions to this model may contain subtours -
# tours that don't visit every city.  The lazy constraint callback
# adds new constraints to cut them off.

import sys
import math
import random
import itertools
from gurobipy import *

n=100#NUMBER OF CITIES

# Callback - use lazy constraints to eliminate sub-tours

def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        # make a list of edges selected in the solution
        vals = model.cbGetSolution(model._vars)
        selected = tuplelist((i,j) for i,j in model._vars.keys() if vals[i,j] > 0.5)
        # find the shortest cycle in the selected edge list
        tour = subtour(selected)
        if len(tour) < n:
            # add subtour elimination constraint for every pair of cities in tour
            model.cbLazy(quicksum(model._vars[i,j]
                                  for i,j in itertools.combinations(tour, 2))
                             <= len(tour)-1)
```

```


# Given a tuplelist of edges, find the shortest subtour

def subtour(edges):
    unvisited = list(range(n))
    cycle = range(n+1) # initial length has 1 more city
    while unvisited: # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            thiscycle.append(current)
            unvisited.remove(current)
            neighbors = [j for i,j in edges.select(current,’*’) if j in unvisited]
        if len(cycle) > len(thiscycle):
            cycle = thiscycle
    return cycle


# Parse argument

#if len(sys.argv) < 2:
#    print(’Usage: tsp.py npoints’)
#    exit(1)
#n = int(sys.argv[1])

# Create n random points

random.seed(1)#fixes the seed, this guarantees the same output on each run
points = [(random.randint(0,100),random.randint(0,100)) for i in range(n)]

# Dictionary of Euclidean distance between each pair of points

dist = {(i,j) :
    math.sqrt(sum((points[i][k]-points[j][k])**2 for k in range(2)))
    for i in range(n) for j in range(i)}

m = Model()

# Create variables

vars = m.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name=’e’)
for i,j in vars.keys():
    vars[j,i] = vars[i,j] # edge in opposite direction

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead.  The following would be equivalent
# to the preceding m.addVars() call...
#
# vars = tupledict()
# for i,j in dist.keys():
#   vars[i,j] = m.addVar(obj=dist[i,j], vtype=GRB.BINARY,
#                        name=’e[%d,%d]’%(i,j))
```

```python


# Add degree-2 constraint

m.addConstrs(vars.sum(i,'*') == 2 for i in range(n))

# Using Python looping constructs, the preceding would be...
#
# for i in range(n):
#   m.addConstr(sum(vars[i,j] for j in range(n)) == 2)


# Optimize model

m._vars = vars
m.Params.lazyConstraints = 1
m.optimize(subtourelim)

vals = m.getAttr('x', vars)
selected = tuplelist((i,j) for i,j in vals.keys() if vals[i,j] > 0.5)

tour = subtour(selected)
assert len(tour) == n

print('')
print('Optimal tour: %s' % str(tour))
print('Optimal cost: %g' % m.objVal)
print('')
```