Aalto University
School of Business

Optimization software

- *Software for optimization*
- *Example: Python + PuLP*
- *Extra example: Python + Gurobi*

Optimization Software beyond Excel Solver

- Optimization solvers (especially MILP)
  - Commercial: Gurobi, IBM CPLEX Optimizer, FICO Xpress, MOSEK, …
  - Open source: lp_solve, GLPK, Open Solver (http://opensolver.org/), PuLP

- Optimization models are usually build with some "programming language" which then calls the solver
  - E.g. R, Matlab, C++, Java, AMPL, Python
  - Excel interfaces exists for most solvers (At least through Visual Basic)

- Next a demo: Model is written in Python and then solved with PuLP
  - Extra slides: Python+Gurobi implementation of the same model
    - Free academic license for Gurobi available here: https://www.gurobi.com/downloads/free-academic-license/

P&P transportation problem revisited: LP formulation

$$\begin{aligned}\min\ & 53x_{11} + 95x_{12} + 136x_{13} + 160x_{14} \\ & + 60x_{21} + 120x_{22} + 132x_{23} + 140x_{24} \\ & + 210x_{31} + 190x_{32} + 89x_{33} + 71x_{34}\end{aligned}$$

Minimize total transportation costs

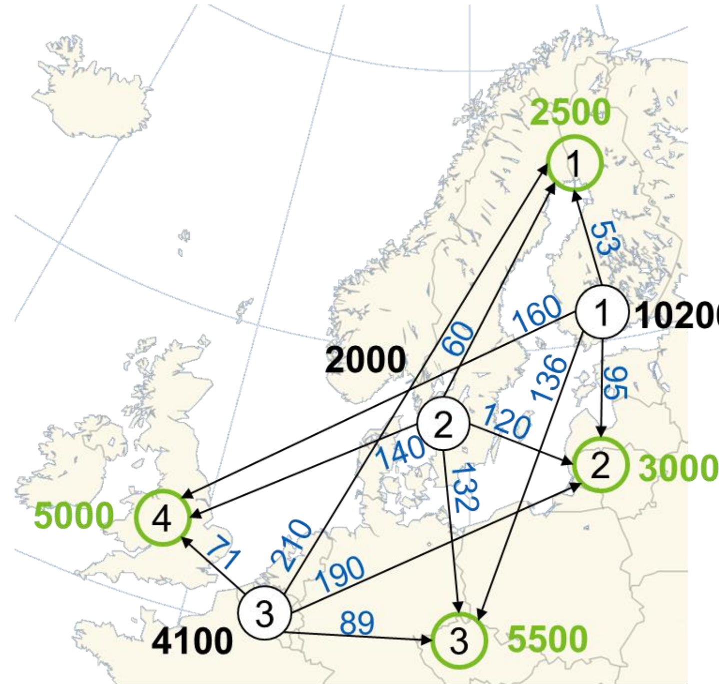
$$\begin{aligned} x_{11} + x_{21} + x_{31} &= 2500 \\ x_{12} + x_{22} + x_{32} &= 3000 \\ x_{13} + x_{23} + x_{33} &= 5500 \\ x_{14} + x_{24} + x_{34} &= 5000 \end{aligned}$$

Satisfy demand

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &\le 10200 \\ x_{21} + x_{22} + x_{23} + x_{24} &\le 2000 \\ x_{31} + x_{32} + x_{33} + x_{34} &\le 4100 \end{aligned}$$

Do not exceed production capacities

$$x_{ij} \ge 0, i = 1, \dots, 3, j = 1, \dots 4$$



P&P transportation problem revisited: Spreadsheet implementation

G9 =SUMPRODUCT($C$5:$F$7;$C$12:$F$14)

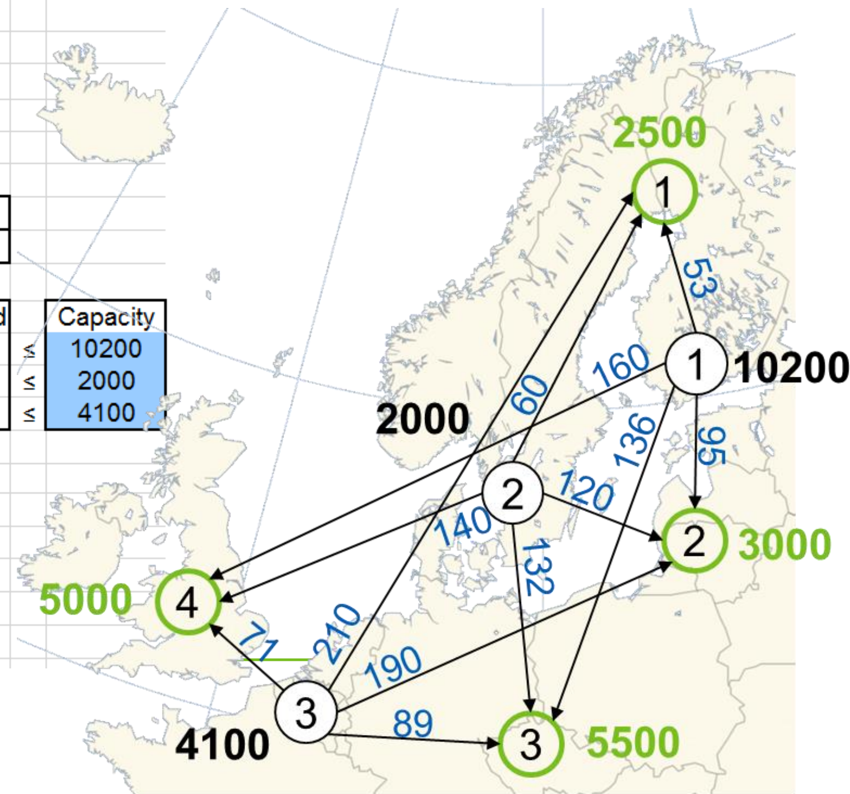
Pulp&Paper Company

Transportation cost per ton		Warehouse			
		1. Finland	2. Lithuania	3. Czech	4. UK
Mill	1. Finland	53 €	95 €	136 €	160 €
	2. Sweden	60 €	120 €	132 €	140 €
	3. Belgium	210 €	190 €	89 €	71 €

Total Cost
1 578 200 €

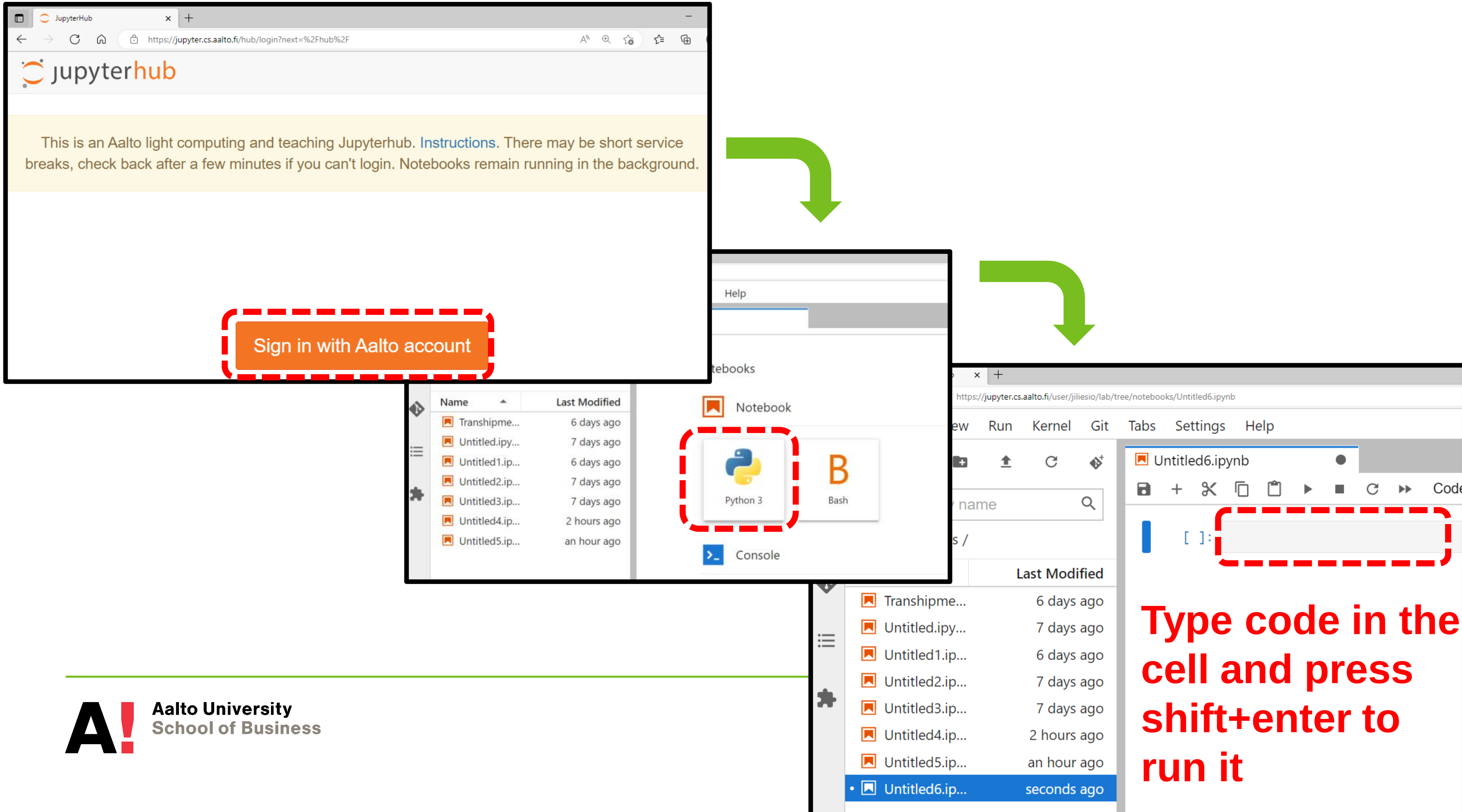
Transported quantity x_{ij} (tons)		Warehouse						
		1. Finland	2. Lithuania	3. Czech	4. UK	Total Shipped		Capacity
Mill	1. Finland	2500	3000	4400	0	9900	≤	10200
	2. Sweden	0	0	1100	900	2000	≤	2000
	3. Belgium	0	0	0	4100	4100	≤	4100
	Total Received	2500	3000	5500	5000			
		=	=	=	=			
	Demand	2500	3000	5500	5000			

Parameters
Decision variables



Aalto University
School of Business

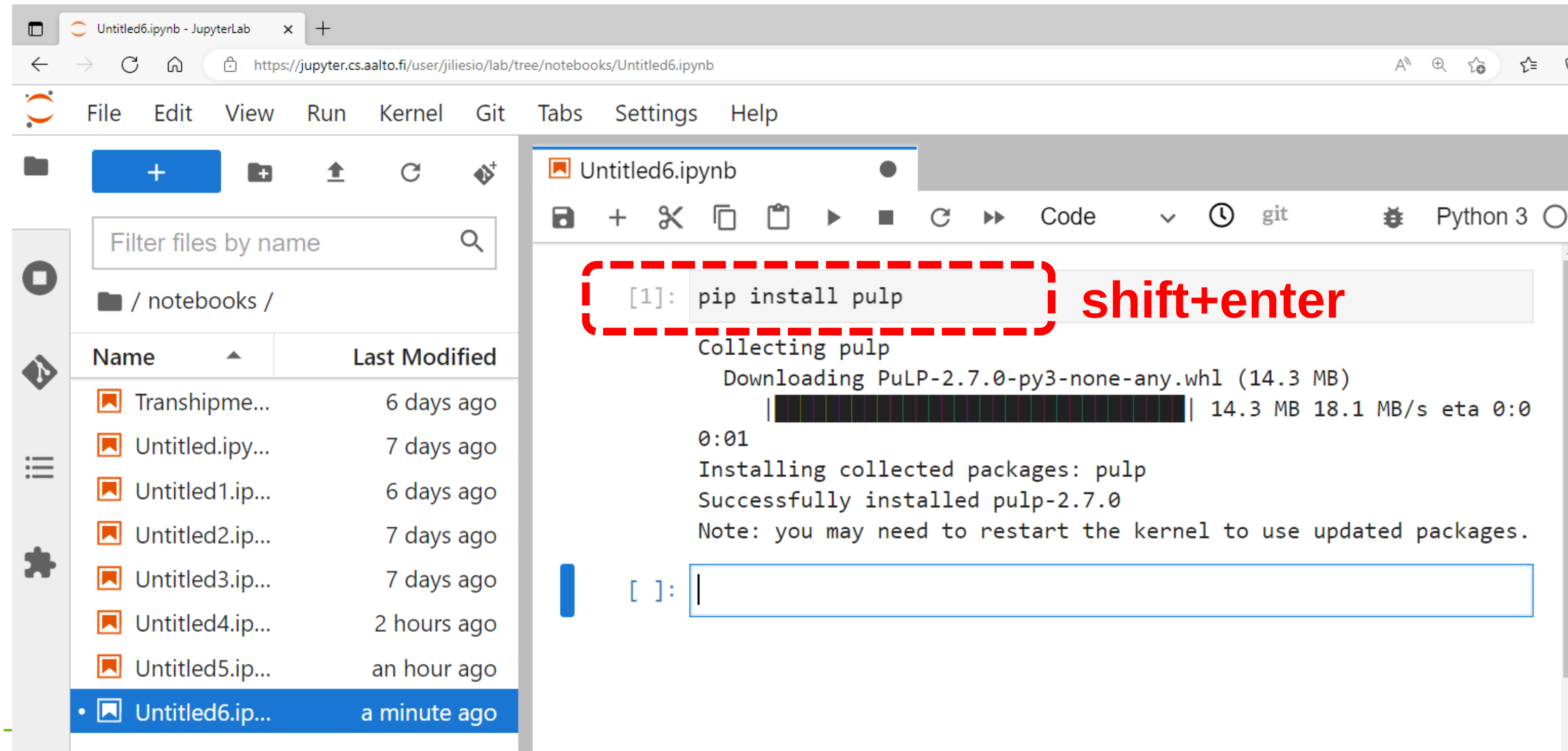
Login to jupyter.cs.aalto.fi



Type code in the cell and press shift+enter to run it

Aalto University
School of Business

Install the PuLP solver for MILP problems



P&P transportation problem revisited: Indices and parameters

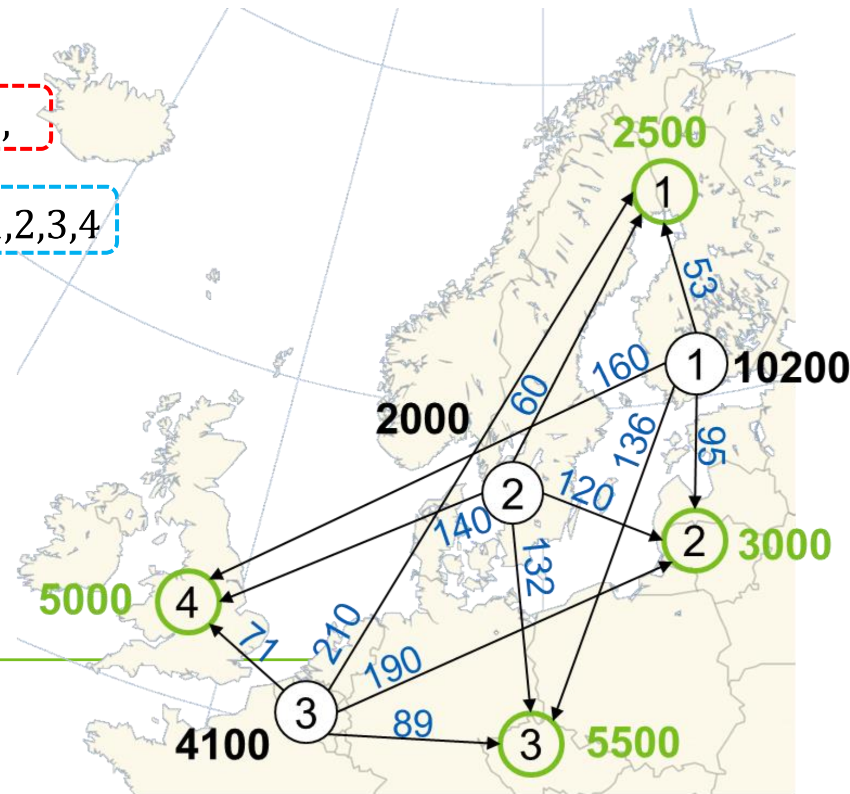
PPcompany.ipynb

Code git

```python
from pulp import *

#Indices
mills = list(range(3))
warehouses =  list(range(4))

#Parameters
costs=[[53,95,136,160],
[60,120,132,140],
[210, 190, 89, 71]]
capacities=[10200, 2000, 4100]
demands = [2500, 3000, 5500, 5000]

#Create a optimization model (minimization)
model = LpProblem("PP_model", LpMinimize)
```

mill $i = 1,2,3,$

warehouse $j = 1,2,3,4$



P&P transportation problem revisited: Decision variables

Decision variables x_{ij}:

Tons of carton transported form mill i

to warehouse j:

$x_{ij} \geq 0,$

$\boxed{i = 1,2,3,}$

$\boxed{j = 1,2,3,4}$

PPcompany.ipynb

Code

git

```
#Create decision variables x_ij
x=[[LpVariable("x_"+str(i+1)+str(j+1),0,None) for j in warehouses] for i in mills]
```

Upper bound

Lower bound

Arbitrary name for the decision variable; here we use x_11,…,x_34

str( ) method returns a string presentation of an integer

P&P transportation problem revisited: Capacity constraints

$$\left.\begin{aligned} x_{11}+x_{12}+x_{13}+x_{14} &\le 10200 \\ x_{21}+x_{22}+x_{23}+x_{24} &\le 2000 \\ x_{31}+x_{32}+x_{33}+x_{34} &\le 4100 \end{aligned}\right\} \Leftrightarrow \boxed{\sum_{j=1}^{4} x_{ij}} \le s_i, \boxed{i = 1,2,3}$$

npany.ipynb

Code git

```python
#Add capacity constraints at the mills:
for i in mills:
    model += (lpSum([x[i][j] for j in warehouses])<=capacities[i], "Capacity_at_mill_"+str(i+1))
```

Constraints are added to the model object

Name for the constraint

P&P transportation problem revisited: Demand constraints

$$\left.\begin{aligned} x_{11} + x_{21} + x_{31} &= 2500 \\ x_{12} + x_{22} + x_{32} &= 3000 \\ x_{13} + x_{23} + x_{33} &= 5500 \\ x_{14} + x_{24} + x_{34} &= 5000 \end{aligned}\right\} \Leftrightarrow \boxed{\sum_{i=1}^{3} x_{ij}} = d_j, \boxed{j = 1,2,3,4}$$

mpany.ipynb

Code git

```python
#Add balance constraints at the warehouses:
for j in warehouses:
    model += (lpSum([x[i][j] for i in mills])==demands[j], "Demand_at_warehouse_"+str(j+1))
```

P&P transportation problem revisited: Objective function

$$\min 53x_{11} + 95x_{12} + 136x_{13} + 160x_{14} + 60x_{21} + 120x_{22} + 132x_{23} + 140x_{24} + 210x_{31} + 190x_{32} + 89x_{33} + 71x_{34}$$

$$\Leftrightarrow \min \boxed{\sum_{i=1}^{3} \sum_{j=1}^{4} c_{ij} x_{ij}}$$

PPcompany.ipynb

Code git

```
#Add objective function:
model += (lpSum([costs[i][j]*x[i][j] for i in mills for j in warehouses]) , "Transportation_costs")
```

P&P transportation problem revisited: Solve the model and print the optimal solution

```
#Print optimal objective function value:
print("Optimal transportation cost "+str(model.objective.value())+" .")


#Print optimal decision variable values:
for i in mills:
    print("------")
    for j in warehouses:
        if (x[i][j].varValue>0):
            print("Transport "+str(x[i][j].varValue)+" tons from mill "+str(i+1)+" to warehouse "+str(j+1)+".")
```

P&P transportation problem revisited: Run the program

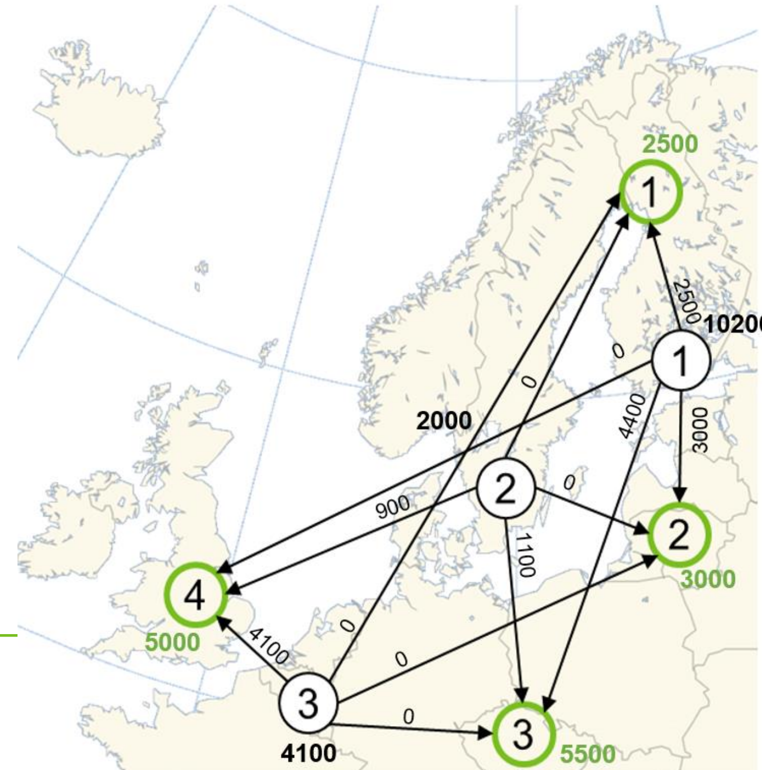
PPcompany.ipynb

```python
from pulp import *

#Indices
mills = list(range(3))
warehouses =  list(range(4))

#Parameters
costs=[[53,95,136,160],
[60,120,132,140],
[210, 190, 89, 71]]
capacities=[10200, 2000, 4100]
demands = [2500, 3000, 5500, 5000]

#Create a optimization model (minimization)
model = LpProblem("PP_model", LpMinimize)

#Create decision variables x_ij
x=[[LpVariable("x_"+str(i+1)+str(j+1),0,None) for j in warehouses] for i in mills]

#Add objective function:
model += (lpSum([costs[i][j]*x[i][j] for i in mills for j in warehouses]) , "Transportation_costs")

#Add capacity constraints at the mills:
for i in mills:
    model += (lpSum([x[i][j] for j in warehouses])<=capacities[i], "Capacity_at_mill_"+str(i+1))

#Add balance constraints at the warehouses:
for j in warehouses:
    model += (lpSum([x[i][j] for i in mills])==demands[j], "Demand_at_warehouse_"+str(j+1))

model.solve() #Solve the optimal solution to model

#Print optimal objective function value:
print("Optimal transportation cost "+str(model.objective.value())+" .")


#Print optimal decision variable values:
for i in mills:
    print("------")
    for j in warehouses:
        if (x[i][j].varValue>0):
            print("Transport "+str(x[i][j].varValue)+" tons from mill "+str(i+1)+" to warehouse "+str(j+1)+".")
```

shift+enter

P&P transportation problem revisited: Program output

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /opt/software/lib/python3.10/site-packages/pulp/solverdir/cbc/linux/
64/cbc /tmp/6a293f80378e420883abbd5fa5fbc186-pulp.mps timeMode elapsed branch print
ingOptions all solution /tmp/6a293f80378e420883abbd5fa5fbc186-pulp.sol (default str
ategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 12 COLUMNS
At line 49 RHS
At line 57 BOUNDS
At line 58 ENDATA
Problem MODEL has 7 rows, 12 columns and 24 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 7 (0) rows, 12 (0) columns and 24 (0) elements
0  Obj 0 Primal inf 16000 (4)
7  Obj 1578200
Optimal - objective value 1578200
Optimal objective 1578200 - 7 iterations time 0.002
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00

Optimal transportation cost 1578200.0 .
------
Transport 2500.0 tons from mill 1 to warehouse 1.
Transport 3000.0 tons from mill 1 to warehouse 2.
Transport 4400.0 tons from mill 1 to warehouse 3.
------
Transport 1100.0 tons from mill 2 to warehouse 3.
Transport 900.0 tons from mill 2 to warehouse 4.
------
Transport 4100.0 tons from mill 3 to warehouse 4.
```



P&P transportation problem revisited: Inspecting the model object

PPcompany.ipynb

Code git

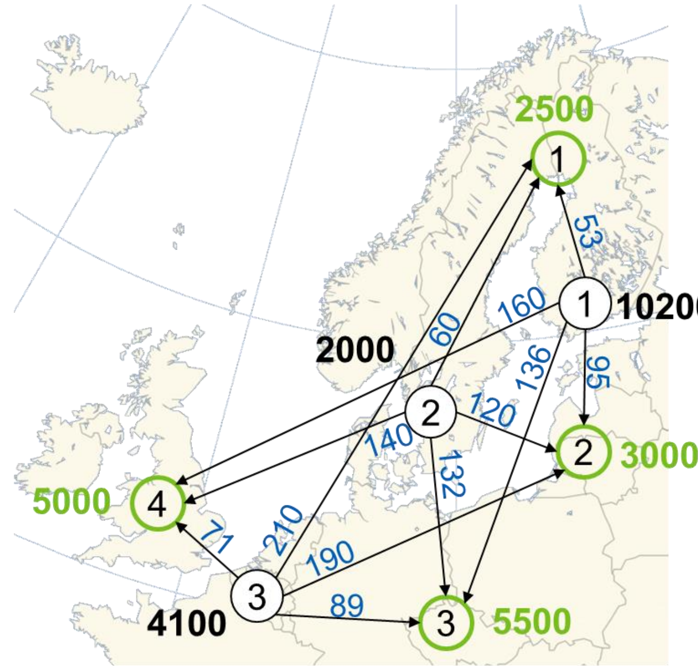
[6]: model shift+enter

```
[6]: PP_model:
MINIMIZE
53*x_11 + 95*x_12 + 136*x_13 + 160*x_14 + 60*x_21 + 120*x_22 + 132*x_23 + 140*x_24 + 210*x_31 + 190*x_32 + 89*x_33 + 71*x_34 + 0
SUBJECT TO
Capacity_at_mill_1: x_11 + x_12 + x_13 + x_14 <= 10200

Capacity_at_mill_2: x_21 + x_22 + x_23 + x_24 <= 2000

Capacity_at_mill_3: x_31 + x_32 + x_33 + x_34 <= 4100

Demand_at_warehouse_1: x_11 + x_21 + x_31 = 2500

Demand_at_warehouse_2: x_12 + x_22 + x_32 = 3000

Demand_at_warehouse_3: x_13 + x_23 + x_33 = 5500

Demand_at_warehouse_4: x_14 + x_24 + x_34 = 5000

VARIABLES
x_11 Continuous
x_12 Continuous
x_13 Continuous
x_14 Continuous
x_21 Continuous
x_22 Continuous
x_23 Continuous
x_24 Continuous
x_31 Continuous
x_32 Continuous
x_33 Continuous
x_34 Continuous
```

Extra example: P&P transportation problem in Python + Gurobi

P&P transportation problem revisited: Python + Gurobi -implementation

```python
#Import Gurobi solver library
from gurobipy import *

#Data
costs=[[53,95,136,160],
[60,120,132,140],
[210, 190, 89, 71]]

capacities=[10200, 2000, 4100]
demands = [2500, 3000, 5500, 5000]
```



P&P transportation problem revisited: Python + Gurobi -implementation

```
#Create the LP model in gurobi
model = Model("PP company transportation")

#Indexes
mills = range(3)
warehouses = range(4)

#Decision variables x_ij
x=[]
for i in mills:
    x.append([])
    for j in warehouses:
        x[i].append(model.addVar(lb=0, name="x%d.%d" % (i, j)))
model.update()
```

Decision variables x_{ij}:

Tons of carton transported form mill i to warehouse j:

$x_{ij} \geq 0$,

$i = 1,2,3,$

$j = 1,2,3,4$

P&P transportation problem revisited: Python + Gurobi -implementation

$$\left.\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &\le 10200 \\ x_{21} + x_{22} + x_{23} + x_{24} &\le 2000 \\ x_{31} + x_{32} + x_{33} + x_{34} &\le 4100 \end{aligned}\right\} \Leftrightarrow \boxed{\sum_{j=1}^{4}} x_{ij} \le s_i, \boxed{i = 1,2,3}$$

```python
#Capacity constraints
for i in mills:
    model.addConstr(quicksum(x[i][j] for j in warehouses)<=capacities[i],"Capacity constraint")
model.update()
```

P&P transportation problem revisited: Python + Gurobi -implementation

$$\left.\begin{aligned} x_{11} + x_{21} + x_{31} &= 2500 \\ x_{12} + x_{22} + x_{32} &= 3000 \\ x_{13} + x_{23} + x_{33} &= 5500 \\ x_{14} + x_{24} + x_{34} &= 5000 \end{aligned}\right\} \Leftrightarrow \boxed{\sum_{i=1}^{3}} x_{ij} = d_j, \boxed{j = 1,2,3,4}$$

```python
#Demand constraints
for j in warehouses:
    model.addConstr(quicksum(x[i][j] for i in mills)==demands[j],"Demand constraint")
model.update()
```

P&P transportation problem revisited: Python + Gurobi -implementation

$$\min 53x_{11} + 95x_{12} + 136x_{13} + 160x_{14} + 60x_{21} + 120x_{22} + 132x_{23} + 140x_{24} + 210x_{31} + 190x_{32} + 89x_{33} + 71x_{34}$$

$$\Leftrightarrow \min \boxed{\sum_{i=1}^{3}} \boxed{\sum_{j=1}^{4}} c_{ij}x_{ij}$$

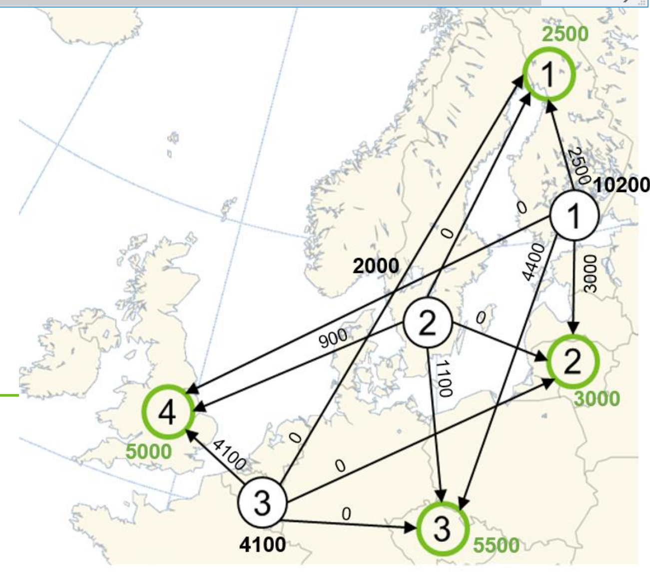
```
#objective function
model.setObjective(quicksum(costs[i][j]*x[i][j] for i in mills for j in warehouses))
model.modelSense = GRB.MINIMIZE
model.update()
```

P&P transportation problem revisited: Python + Gurobi -implementation

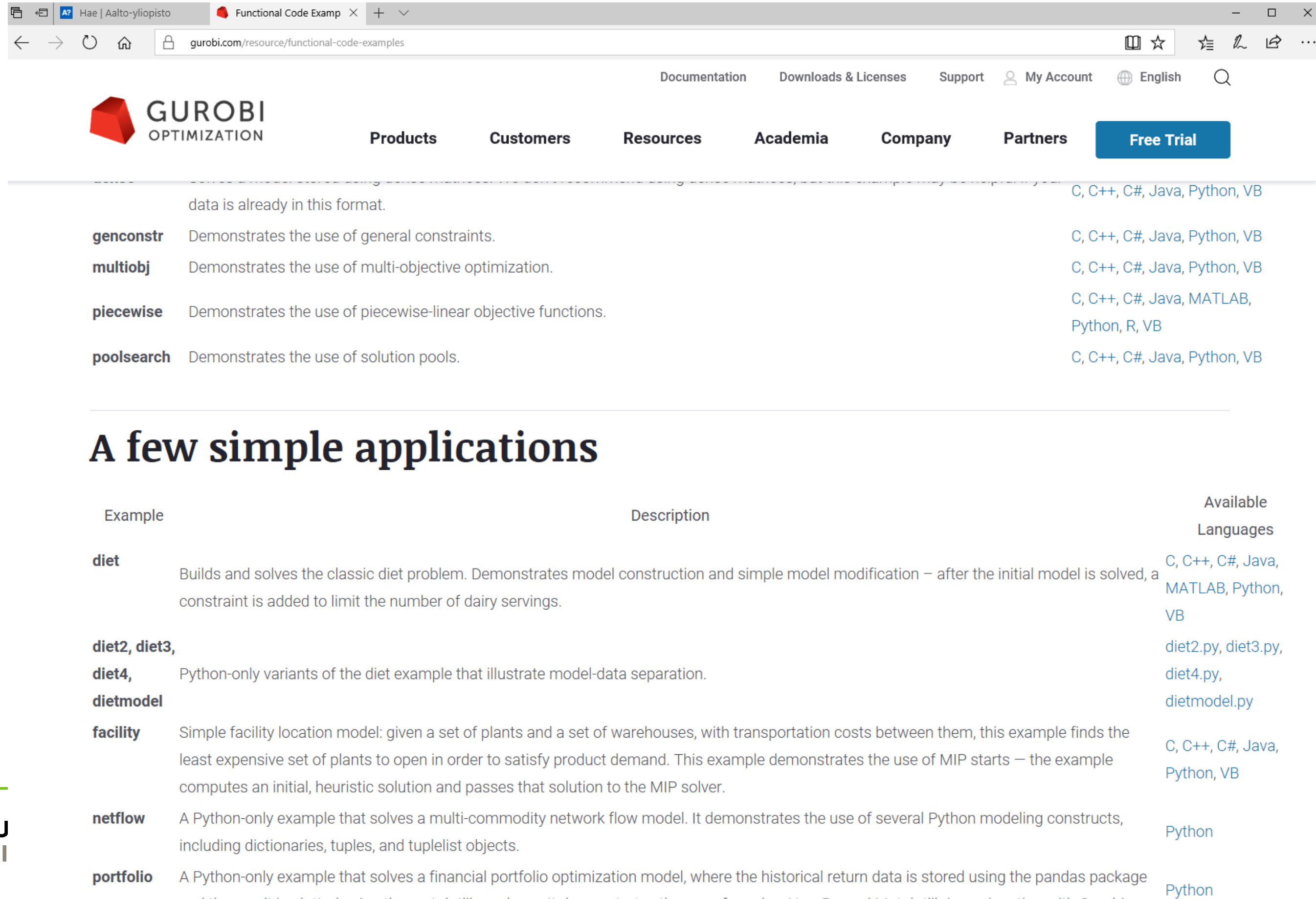
```python
#Find optimal solution
model.optimize()

#Collect optimal decision variable
#values to array x_optimal
x_optimal=[]
for i in mills:
    x_optimal.append([])
    for j in warehouses:
        x_optimal[i].append(x[i][j].x)

#Print optimal transportation
#cost and solution
print("Optimal transportation cost:")
print(model.objVal)
print("Optimal transportation quantities:")
print(x_optimal)
```

Windows PowerShell (5)

```
PS C:\Users\jiliesio\Dropbox (Aalto)\Business Decisions 1\2019\Lectures> gurobi '.\7 - PPcompany.py'
Using license file C:\Users\jiliesio\gurobi.lic
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.0 build v9.0.0rc2 (win64)
Optimize a model with 7 rows, 12 columns and 24 nonzeros
Model fingerprint: 0x215c731e
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [5e+01, 2e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+03, 1e+04]
Presolve time: 0.00s
Presolved: 7 rows, 12 columns, 24 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.2620000e+06   6.400000e+03   0.000000e+00      0s
       3    1.5782000e+06   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.00 seconds
Optimal objective  1.578200000e+06
Optimal transportation cost:
1578200.0
Optimal transportation quantities:
[[2500.0, 3000.0, 4400.0, 0.0], [0.0, 0.0, 1100.0, 900.0], [0.0, 0.0, 0.0, 4100.0]]
PS C:\Users\jiliesio\Dropbox (Aalto)\Business Decisions 1\2019\Lectures>
```



More examples: www.gurobi.com/resource/functional-code-examples/



gurobi.com/resource/functional-code-examples

Documentation | Downloads & Licenses | Support | My Account | English

GUROBI OPTIMIZATION

Products | Customers | Resources | Academia | Company | Partners | Free Trial

	data is already in this format.	C, C++, C#, Java, Python, VB
genconstr	Demonstrates the use of general constraints.	C, C++, C#, Java, Python, VB
multiobj	Demonstrates the use of multi-objective optimization.	C, C++, C#, Java, Python, VB
piecewise	Demonstrates the use of piecewise-linear objective functions.	C, C++, C#, Java, MATLAB, Python, R, VB
poolsearch	Demonstrates the use of solution pools.	C, C++, C#, Java, Python, VB

A few simple applications

Example	Description	Available Languages
diet	Builds and solves the classic diet problem. Demonstrates model construction and simple model modification – after the initial model is solved, a constraint is added to limit the number of dairy servings.	C, C++, C#, Java, MATLAB, Python, VB
diet2, diet3, diet4, dietmodel	Python-only variants of the diet example that illustrate model-data separation.	diet2.py, diet3.py, diet4.py, dietmodel.py
facility	Simple facility location model: given a set of plants and a set of warehouses, with transportation costs between them, this example finds the least expensive set of plants to open in order to satisfy product demand. This example demonstrates the use of MIP starts — the example computes an initial, heuristic solution and passes that solution to the MIP solver.	C, C++, C#, Java, Python, VB
netflow	A Python-only example that solves a multi-commodity network flow model. It demonstrates the use of several Python modeling constructs, including dictionaries, tuples, and tuplelist objects.	Python
portfolio	A Python-only example that solves a financial portfolio optimization model, where the historical return data is stored using the pandas package and the result is plotted using the matplotlib package. It demonstrates the use of pandas, NumPy, and Matplotlib in conjunction with Gurobi.	Python