Python and AMPL:

Build Prescriptive Analytics Applications Quickly with Pandas, Colab, Streamlit, and *amplpy*

Filipe Brandão, Robert Fourer

{fdabrandao,4er}@ampl.com

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Business Analytics Conference

Aurora, Colorado — 17 April 2023
Technology Tutorial

Python and AMPL: Build Prescriptive Analytics Applications Quickly with Pandas, Colab, Streamlit, and *amplpy*

Python and its vast ecosystem are great for data pre-processing, solution analysis, and visualization, but Python's design as a general-purpose programming language makes it less than ideal for expressing the complex optimization problems typical of prescriptive analytics. AMPL is a declarative language that is designed for describing optimization problems and that integrates naturally with Python.

In this presentation, you'll learn how the combination of AMPL modeling with Python environments and tools has made optimization software more natural to use, faster to run, and easier to integrate with enterprise systems. Following a quick introduction to model-based optimization, we will show how AMPL and Python work together in a range of contexts:

- Installing AMPL and solvers as Python packages
- Importing and exporting data naturally from/to Python data structures such as Pandas dataframes
- Developing AMPL model formulations directly in Jupyter notebooks
- Using AMPL and open-source solvers for free on Google Colab, with no arbitrary problem size limits
- Turning Python scripts into prescriptive analytics applications in minutes with Pandas, Streamlit, and amplpy

Mathematical Optimization

In concept,

- Given an objective function of some *decision variables*
- Choose values of the variables to
  make the objective as large or as small as possible
- Subject to constraints on the values of the variables

In practice,

- A paradigm for a very broad variety of *decision problems*
- A valuable approach to making decisions

Optimization in Operations Research and Analytics

Given a recurring need to make many interrelated decisions

- Purchases, production and shipment amounts, assignments, . . .

Consistently make highly desirable choices

By applying ideas from mathematical optimization

- Ways of describing problems *(models)*
- Ways of solving problems *(algorithms)*

Optimization in Practice

Large numbers of decision variables

- Thousands to millions

An objective function

- To be minimized or maximized

Various constraint types

- 10-20 distinct types
- Thousands to millions of each type
- Few variables involved in each constraint

Solved many times

- In development
  - different model formulations & solver strategies
- In deployment
  - different data scenarios

and for Optimization

Python

- ❖ Executable, general-purpose programming language
- \+ Vast ecosystem for
  data pre-processing, solution analysis, and visualization
- – Awkward for defining optimization problems

AMPL

- ❖ Declarative, specialized modeling language
- \+ Designed for defining optimization models
- \+ Integrates with Python's ecosystem

Outline

Example: Network design with redundancy

- Defining the problem in a way that people understand
  - in words — in algebra — in ***AMPL***
- Adding data and solving
  - interactive prototyping environment

Integrating with Python

- Interfacing with Python using ***amplpy***
- Installing AMPL and solvers as ***Python packages***
- Developing AMPL models directly in ***Jupyter notebook***s
- Using AMPL and solvers free on ***Google Colab***
- Importing and exporting data naturally
  from/to Python data structures such as ***Pandas dataframes***
- Turning Python scripts into prescriptive analytics applications
  in minutes with Pandas, amplpy, and ***Streamlit***

Example:

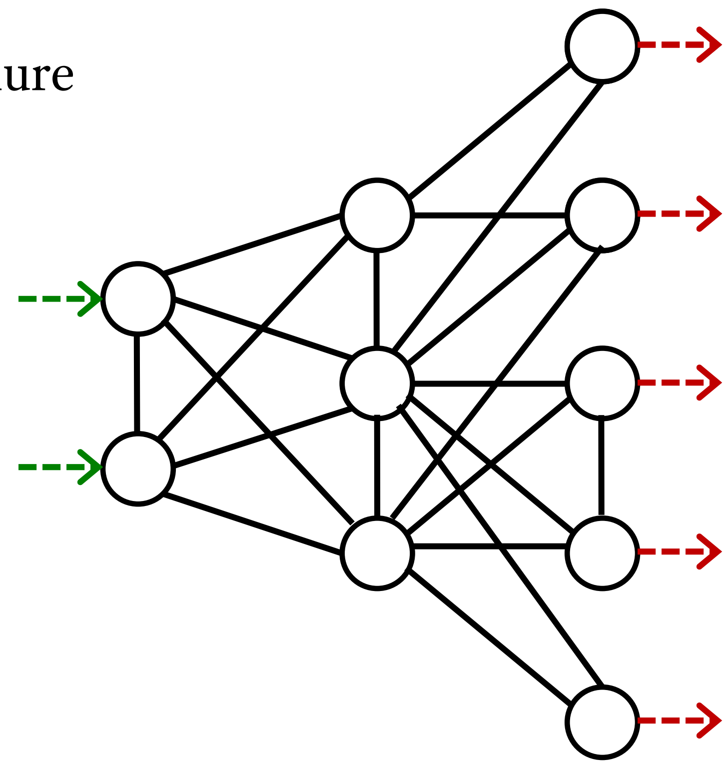
Network Design with Redundancy

Motivation

- Build a least-cost network
  that withstands any single-node failure

Context

- a flow network
  - nodes ○ representing locations
  - links —— connecting nodes



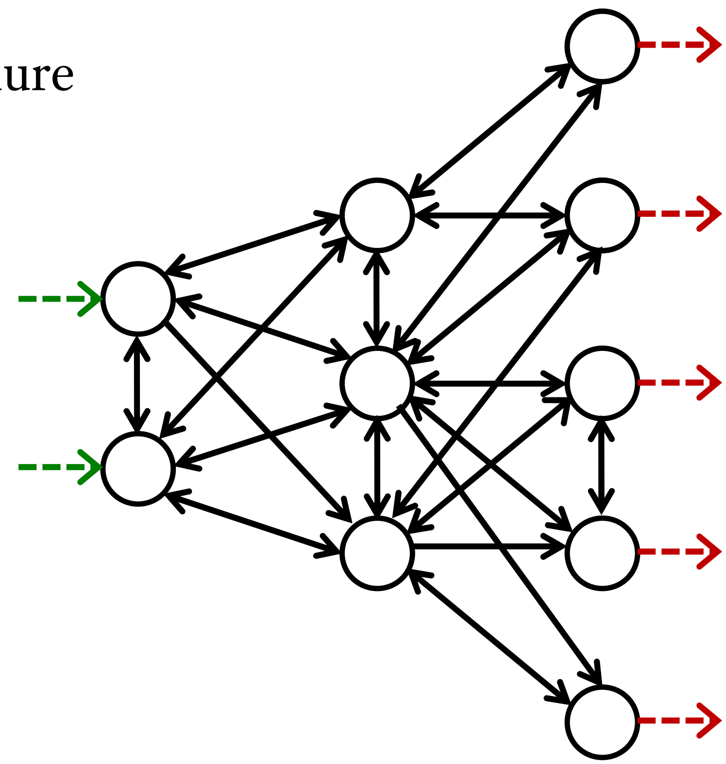
Example:
Network Design with Redundancy

Motivation

- Build a least-cost network
  that withstands any single-node failure

Context

- a flow network
  - nodes ◯ representing cities
  - links —— connecting nodes
  - arcs ⟷ representing flows
- production ---> at nodes
  demands ---> at nodes
- flow capacities on arcs
  costs of building links



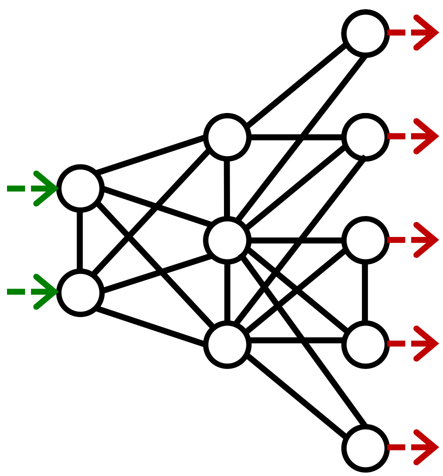
Network Redundancy

Decide

- ❖ which links to build

So that

- ❖ total construction cost is kept low



and

- ❖ flows on each arc respect capacities
- ❖ only built arcs have flows
- ❖ supplies, demands, and shipments are in balance at each node

and

- ❖ ***there is still a feasible flow when any one node fails***

Algebraic Formulation

Given sets

N	network nodes
$L \subseteq N \times N$	(undirected) links connecting nodes
$A = L \cup \{(j, i) \colon (i, j) \in L\}$	(directed) arcs from node to node

```
set N;                                         # nodes
set L within N cross N;                        # links (undirected)
set A = L union setof {(i,j) in L} (j,i);      # arcs (directed)
```

https://colab.research.google.com/github/ampl/amplcolab/blob/master/authors/fdabrandao/military/electric_grid_with_redundancy.ipynb

Network Redundancy

Algebraic Formulation

Given data

b_i demand/production at node i, for each $i \in N$
> 0 implies demand, < 0 implies production

c_{ij} cost to build a link connecting i and j, for each $(i, j) \in L$

u upper limit (capacity) on amount sent along any arc

```
param demsup {N};   # demand (positive) or production (negative)
param cost {L};     # cost to build a link
param capacity;     # capacity of links
```

Algebraic Formulation

Determine

y_{ij} 1 if a link is built from i to j, 0 otherwise
for each $(i, j) \in L$

f_{ij} flow from i to j when there is no failure, for each $(i, j) \in A$

f_{ij}^{r} flow from i to j when node r has failed,
for each $(i, j) \in A$ and $r \in N$

```
var Build {L} binary;  # Build[i,j] = 1 iff link btw i & j is built
var Flow {A} >= 0;     # Flow[i,j] is flow from i to j
var FlRm {A,N} >= 0;   # FlRm[i,j,rm] is flow from i to j
                       # when node rm is removed
```

Algebraic Formulation

Minimize

$\sum_{(i,j)\in L} c_{ij} y_{ij}$

total cost of all links built

```
minimize TotalBuildCost:
    sum {(i,j) in L} cost[i,j] * Build[i,j];
```

Network Redundancy

Algebraic Formulation

Subject to

$\sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij} \geq b_i$, for all $i \in N$

flow in minus flow out must be ≥ demand, or
flow out minus flow in must be ≤ production

$f_{ij} \leq u y_{ij}$, $f_{ji} \leq u y_{ij}$ for all $(i,j) \in L$

when a link is built from i to j, flow may not exceed capacity;
when no link is built from i to j, there can be no flow

```
subject to Balance {i in N}:
  sum {(j,i) in A} Flow[j,i] - sum {(i,j) in A} Flow[i,j] >= demsup[i];

subject to ArcExists1 {(i,j) in L}:
  Flow[i,j] <= capacity * Build[i,j];
subject to ArcExists2 {(i,j) in L}:
  Flow[j,i] <= capacity * Build[i,j];
```

Network Redundancy

Algebraic Formulation

Subject to

$\sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij} \geq b_i,$ for all $i \in N$

flow in minus flow out must be ≥ demand, or
flow out minus flow in must be ≤ production

$f_{ij} \leq u y_{ij},\ f_{ji} \leq u y_{ij}$ for all $(i,j) \in L$

when a link is built from i to j, flow may not exceed capacity;
when no link is built from i to j, there can be no flow

```
subject to Balance {i in N}:
  sum {(j,i) in A} Flow[j,i] - sum {(i,j) in A} Flow[i,j] >= demsup[i];

subject to ArcExists1 {(i,j) in L}:
  Build[i,j] = 0 ==> Flow[i,j] = Flow[j,i] = 0;

subject to ArcExists2 {(i,j) in L}:
  Build[i,j] = 1 ==> Flow[i,j] <= capacity and Flow[j,i] <= capacity;
```

Algebraic Formulation

Subject to

$\sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij} \geq b_i,$ for all $i \in N$

flow in minus flow out must be ≥ demand, or
flow out minus flow in must be ≤ production

$f_{ij} \leq u y_{ij},\ f_{ji} \leq u y_{ij}$ for all $(i,j) \in L$

when a link is built from i to j, flow may not exceed capacity;
when no link is built from i to j, there can be no flow

```
subject to Balance {i in N}:
  sum {(j,i) in A} Flow[j,i] - sum {(i,j) in A} Flow[i,j] >= demsup[i];

subject to ArcExists1 {(i,j) in L}:
  Flow[i,j] <= capacity * Build[i,j];
subject to ArcExists2 {(i,j) in L}:
  Flow[j,i] <= capacity * Build[i,j];
```

Network Redundancy

Algebraic Formulation

Subject to

$\sum_{(j,i)\in A} f_{ij}^r - \sum_{(i,j)\in A} f_{ji}^r \geq b_i,$ for all $r \in N$, $i \in N \setminus \{r\}$

when node r fails,
same flow balance constraints at other nodes

$f_{ij}^r \leq u y_{ij},\ \ f_{ji}^r \leq u y_{ij}$ for all $r \in N$, $(i,j) \in L$

when node r fails,
same capacity/no-flow constraints at links

```
subject to BalanceRm {rm in N, i in N diff {rm}}:
  sum{(j,i)in A}FlRm[j,i,rm]-sum{(i,j)in A}FlRm[i,j,rm]>=demsup[i];

subject to ArcExistsRm1 {(i,j) in L, rm in N}:
  FlRm[i,j,rm] <= capacity * Build[i,j];
subject to ArcExistsRm2 {(i,j) in L, rm in N}:
  FlRm[j,i,rm] <= capacity * Build[i,j];
```

Network Redundancy

Algebraic Formulation

Subject to

$$\sum_{(i,r)\in A} f_{ir}^r + \sum_{(r,j)\in A} f_{rj}^r = 0, \qquad \text{for all } r \in N$$

when node r fails, there is no flow into it or out of it

```
subject to RemoveNode {rm in N}:
    sum {(i,rm) in A} FlRm[i,rm,rm] +
    sum {(rm,j) in A} FlRm[rm,j,rm] = 0;
```

AMPL Environments

Stand-alone interactive

- ❖ Model definition statements
- ❖ Model and solver management commands
- ❖ Scripting facilities

Python integrated

- ❖ Application programming interface (API)
- ❖ AMPL and solvers as Python packages
- ❖ AMPL in Jupyter notebooks
  - ∗ Models in notebook cells
  - ∗ Optimization applications in collaboratories

Network Redundancy

Data Instance

```
param: N: demsup :=
   1  -900    4  -450    7 200    10 250    13 300    16 150    19 100
   2  -500    5  -750    8 300    11 300    14 300    17 250    20 250
   3 -1200    6 -1200    9 200    12 250    15 250    18 300    21 250 ;

param: L: cost :=
    6 18  223.607     5 16  291.548     2 11  200.000   18 19  100.000
    1  2  300.000     7 12  373.363     3 14  353.553    2  3  141.421
    6 15  344.819     4 18  316.228    10 12  330.151    6 17  180.278
   17 20  250.000     6 16  158.114     5 11  223.607    2  9  351.283
   18 20  158.114     8  9  550.091    19 21  254.951    3  9  281.780
    1 11  360.555    16 18  254.951     4 19  360.555    1  7  150.000
   12 16  445.982     9 14  380.000     2  8  200.998    4 14  254.951
   11 13  360.555    18 21  291.548    14 20  360.555    1 10  270.740
   16 17  150.000    17 21  403.113    13 17  320.156   12 15  353.553
   19 20  212.132     6 19  200.000     5 17  250.000   13 15  378.021
   12 13  284.429     9 11  418.808     7 10  121.655    5 18  316.228
    1  8  101.980     3 11  141.421     8 10  372.156    7  8  250.799
    5 13  200.000    17 18  111.803    15 17  432.897    4 20  158.114
   13 16  254.951    20 21  223.607    10 13  466.154    4 21  254.951
   11 14  380.789     7 13  427.200     8 11  297.321   15 16  284.429
    3  8  323.110    17 19  180.278 ;

param capacity := 1000;
```

Stand-Alone Interactive

First Try

```
ampl: model netredun.mod;
ampl: data netredun.dat;

ampl: option solver gurobi;
ampl: solve;

Gurobi 10.0.0: infeasible problem
753 simplex iterations
1 branching nodes
```

Stand-Alone Interactive

Second Try: Solved

```
ampl: model netredun.mod;
ampl: data netredun.dat;

ampl: option solver gurobi;
ampl: solve;

Presolve eliminates 269 constraints and 248 variables.
Adjusted problem:
2542 variables:
        62 binary variables
        2480 linear variables
2921 constraints, all linear; 9920 nonzeros
        2921 inequality constraints
1 linear objective; 62 nonzeros.

Gurobi 10.0.0: optimal solution; objective 4495.012
461159 simplex iterations
1796 branching nodes

ampl: display _ampl_time, _solve_elapsed_time;
_ampl_time = 0.140625
_solve_elapsed_time = 16.859
```

Stand-Alone Interactive

Results: # of Links Built and Used

```
ampl: print count {(i,j) in L} (Build[i,j] > 0);
21

ampl: print count {(i,j) in L}
ampl?    (Build[i,j] > 0 and Flow[i,j] + Flow[j,i] > 0);
17

ampl: display {(i,j) in L: Build[i,j] > 0}
ampl?    diff {(i,j) in L: Flow[i,j] + Flow[j,i] > 0};

(2,11)  (9,14)  (2,8)  (4,20) ;
```

Stand-Alone Interactive

Results: # of Links Used in Each Scenario

```
ampl: option display_1col 0, omit_zero_rows 1;

ampl: display {rm in N} count {(i,j) in L}
ampl?    (Build[i,j] > 0 and FlRm[i,j,rm] + FlRm[j,i,rm] > 0);

 1 18     4 18     7 16    10 16    13 17    16 15    19 17
 2 17     5 19     8 17    11 16    14 15    17 16    20 16
 3 19     6 19     9 16    12 15    15 17    18 15    21 15

ampl: display {(i,j) in L} count {rm in N}
ampl?    (Build[i,j] = 1 and FlRm[i,j,rm] + FlRm[j,i,rm] > 0);

1   7     17        4   20    18      12  15    18
1   8     19        5   13    19      13  16    19
2   8      8        6   15    18      16  17    11
2   11     8        6   17    19      17  18    18
3   9     19        7   10    19      18  19    18
3   11    19        9   14    18      19  21    15
4   14    17       10   12    16      20  21    16 ;
```

Modeling in AMPL vs. Modeling in Python

Pyomo, gurobipy, DOcplex, CVXPY, PuLP, etc.

- Optimization model is *also* defined by Python statements
- Combines modeling and programming

. . . puts everything in one language

AMPL

- Optimization model is written in AMPL statements
- Separates modeling and programming
- Allows a much cleaner statement of the modeling

. . . facilitates development and maintenance

Min-Cost Flow in Pyomo

89 lines (70 sloc) | 3.47 KB

Raw Blame

```
import pyomo
import pandas
import pyomo.opt
import pyomo.environ as pe

class MinCostFlow:
    def __init__(self, nodesfile, arcsfile):
        """Read in the csv data."""
        # Read in the nodes file
        self.node_data = pandas.read_csv('nodes.csv')
        self.node_data.set_index(['Node'], inplace=True)
        self.node_data.sort_index(inplace=True)
        # Read in the arcs file
        self.arc_data = pandas.read_csv('arcs.csv')
        self.arc_data.set_index(['Start','End'], inplace=True)
        self.arc_data.sort_index(inplace=True)

        self.node_set = self.node_data.index.unique()
        self.arc_set = self.arc_data.index.unique()

        self.createModel()

    def createModel(self):
        """Create the pyomo model given the csv data."""
        self.m = pe.ConcreteModel()

        # Create sets
        self.m.node_set = pe.Set( initialize=self.node_set )
        self.m.arc_set = pe.Set( initialize=self.arc_set , dimen=2)
```

Min-Cost Flow in Pyomo *(cont'd)*

```
        # Create variables
        self.m.Y = pe.Var(self.m.arc_set, domain=pe.NonNegativeReals)

        # Create objective
        def obj_rule(m):
            return sum(m.Y[e] * self.arc_data.ix[e,'Cost'] for e in self.arc_set)
        self.m.OBJ = pe.Objective(rule=obj_rule, sense=pe.minimize)

        # Flow Ballance rule
        def flow_bal_rule(m, n):
            arcs = self.arc_data.reset_index()
            preds = arcs[ arcs.End == n ]['Start']
            succs = arcs[ arcs.Start == n ]['End']
            return sum(m.Y[(p,n)] for p in preds) - sum(m.Y[(n,s)] for s in succs) == self.node_data.ix
        self.m.FlowBal = pe.Constraint(self.m.node_set, rule=flow_bal_rule)

        # Upper bounds rule
        def upper_bounds_rule(m, n1, n2):
            e = (n1,n2)
            if self.arc_data.ix[e, 'UpperBound'] < 0:
                return pe.Constraint.Skip
            return m.Y[e] <= self.arc_data.ix[e, 'UpperBound']
        self.m.UpperBound = pe.Constraint(self.m.arc_set, rule=upper_bounds_rule)

        # Lower bounds rule
        def lower_bounds_rule(m, n1, n2):
            e = (n1,n2)
            if self.arc_data.ix[e, 'LowerBound'] < 0:
                return pe.Constraint.Skip
            return m.Y[e] >= self.arc_data.ix[e, 'LowerBound']
        self.m.LowerBound = pe.Constraint(self.m.arc_set, rule=lower_bounds_rule)
```

Min-Cost Flow in Pyomo *(cont'd)*

```python
    def solve(self):
        """Solve the model."""
        solver = pyomo.opt.SolverFactory('gurobi')
        results = solver.solve(self.m, tee=True, keepfiles=False, options_string="mip_tolerances_integr

        if (results.solver.status != pyomo.opt.SolverStatus.ok):
            logging.warning('Check solver not ok?')
        if (results.solver.termination_condition != pyomo.opt.TerminationCondition.optimal):
            logging.warning('Check solver optimality?')


if __name__ == '__main__':
    sp = MinCostFlow('nodes.csv', 'arcs.csv')
    sp.solve()
    print('\n\n---------------------------')
    print('Cost: ', sp.m.OBJ())
```

https://github.com/Pyomo/PyomoGallery/blob/master/pandas_min_cost_flow/min_cost_flow.py

Min-Cost Flow Model in AMPL

```
# Sets

set Nodes;
set Arcs within Nodes cross Nodes;

# Parameters

param cost {Arcs};
param upperBound {Arcs} >= 0 default Infinity;
param lowerBound {Arcs} >= 0 default 0;
param imbalance {Nodes};

# Variables

var Flow {(i,j) in Arcs} >= lowerBound[i,j], <= upperBound[i,j];

# Objective function

minimize TotalCost: sum {(i,j) in Arcs} cost[i,j] * Flow[i,j];

# Flow balance constraints

subject to FlowBal {i in Nodes}:

  sum {(j,i) in Arcs} Flow[j,i] -
  sum {(i,j) in Arcs} Flow[i,j] = imbalance[i];
```

Pyomo Complications for a Harder Case

1 Answer

Sorted by: Reset to default | Date created (oldest first)

0

As you already guessed, these are rather coding issues than solver issues.

Regarding the pyomo model: Both `m.pred` and `m.PAIRS` in your linked MWE are empty. DiGraph's predecessor method returns an iterator and expects the node `n` as argument whose predecessors you'd like to iterate over. That's why

```
predecessors = numpy.array(G.predecessors)
```

is just a numpy array of a method and not an iterator. So it doesn't make sense in your case. Since you can initialize Pyomo Sets from any *Iterable*, you can use a simple list instead of an iterator:

```
predecessors = list(list(G.predecessors(node)) for node in G.nodes())
```

Then, `predecessors[i-1]` gives you all the direct predecessor nodes of the node i as Python uses zero-based indexing and your graph nodes start with 1. Alternatively, you can easily create a dict such that `predecessors[i, j]` gives you the direct predecessor nodes of the nodes i and j, i.e. the edge (i, j):

```
predecessors = {(i,j): list(G.predecessors(i)) + list(G.predecessors(j)) for (i,j) in S}
```

Maximum in gurobipy

TypeError: unsupported operand type(s) for *: 'int' and 'GenExprMax' Answered

Follow 2

Hi

I'm trying to solve a production problem. when the x change, it will cost a different additional cost. I need to compare the (x[i] -x[i-1]) with 0. how can I solve this.

```
production_change_cost = gp.quicksum(3 * gp.max_(0,(x[i] -x[i-1] for i in periods)) \
                       + 0.8 * gp.max_(0,(x[i-1] - x[i] for i in periods)))
```

Maximum in gurobipy *(reply)*

General constraints are meant to be used to define single constraints. It is not possible to use these constructs in other expressions, i.e., it is not possible to use gp.max_ in a more complex constraint other than y = gp.max_.

Moreover, as described in the [documentation of the addGenConstrMax method](), gp.max_ only accepts single variables as inputs. Thus, it is not possible to pass expressions `x[i] -x[i-1]`. To achieve what you want, you have to introduce additional auxiliary variables `aux[i] = x[i] -x[i-1]` and additional equality constraints z1 = gp.max_ and z2 = gp.max_

```
aux1 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar1")
aux2 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar2")
# are you sure that i-1 does not lead to a wrong key access?
m.addConstrs((aux1[i] = x[i]-x[i-1| for i in periods), name = "auxconstr1")
m.addConstrs((aux2[i] = x[i-1]-x[i| for i in periods), name = "auxconstr2")
z1 = m.addVar(lb = -GRB.INFINITY, name="z1")
z2 = m.addVar(lb = -GRB.INFINITY, name="z2")
m.addConstr(z1 = gp.max_(0,aux1),name="maxconstr1")
m.addConstr(z2 = gp.max_(0,aux2),name="maxconstr2")
[...]
production_change_cost = gp.quicksum(3 * z1 + 0.8 * z2)
```

Maximum in AMPL

```
param T > 0;

var x {0..T} >= 0;

var production_change_cost =
   3 * max(0, {i in 1..T} x[i] - x[i-1]) +
   0.8 * max(0, {i in 1..T} x[i-1] - x[i]);
```

gurobipy Efficiency Concerns

> *Does this mean that building the model becomes faster if you use quicksum() or does also solving the model becomes faster when you use quicksum()?*

In principle, one can hope that building the model will become faster with quicksum(). But, as stated in the documentation, there are even quicker ways, such as addTerms or the LinExpr() constructors. You can use time or timeit to check the performance of Python's sum() method, the quicksum() method (as well as the others, if desired) and to assess it against the needs of your application.

I do not believe that using quicksum() will speed up the solution process. This method is only used to construct the model and - to the best of my knowledge - has no impact on the solution process. Perhaps someone from Gurobi team can confirm this?

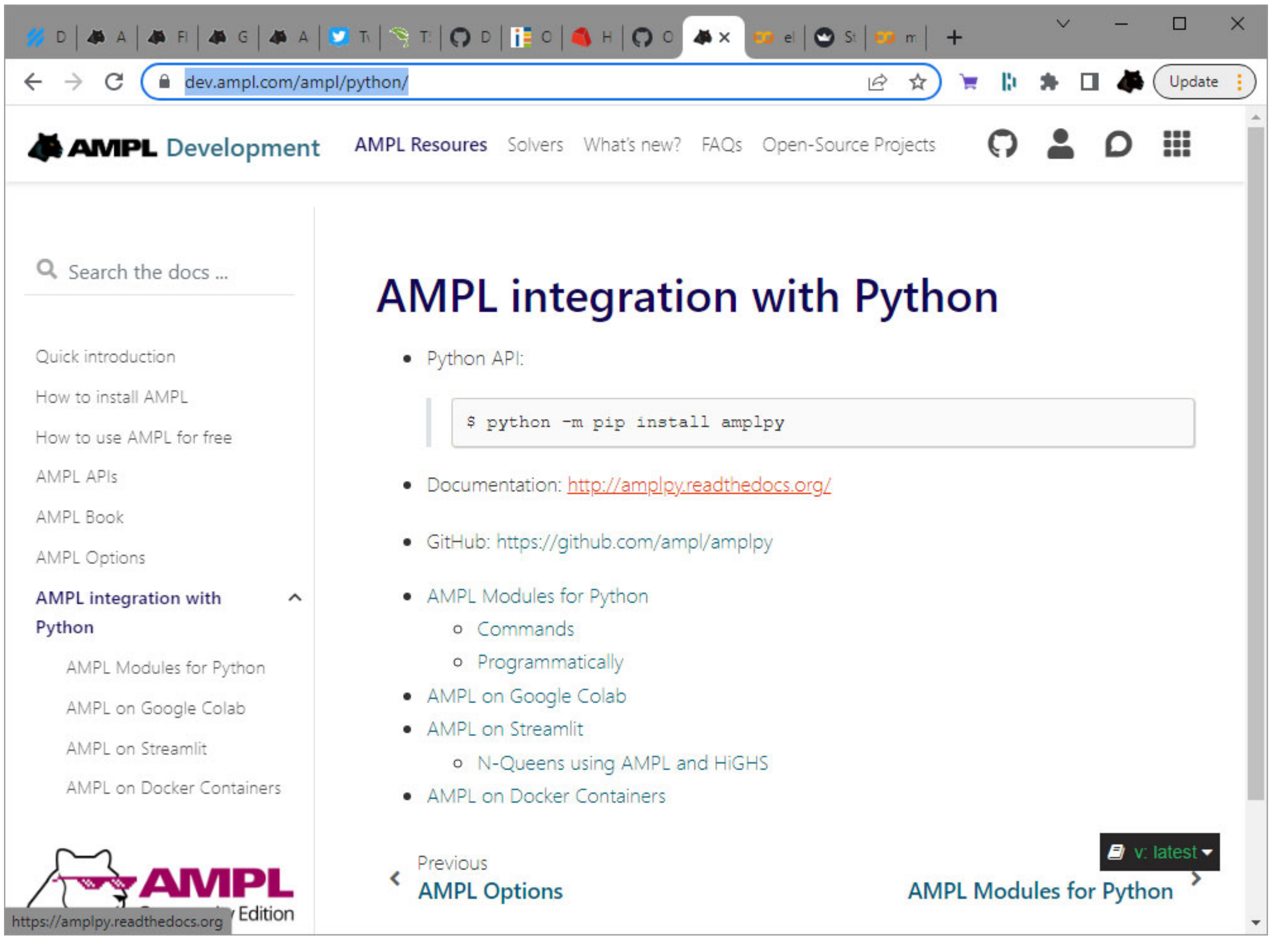
> *Can we use quicksum() in the same way as we would use sum() (in terms of arguments and such)?*

Consult the documentation for instructions on how to use the method.

. . . AMPL has a single fast* `sum` *operator

AMPL Integration with Python

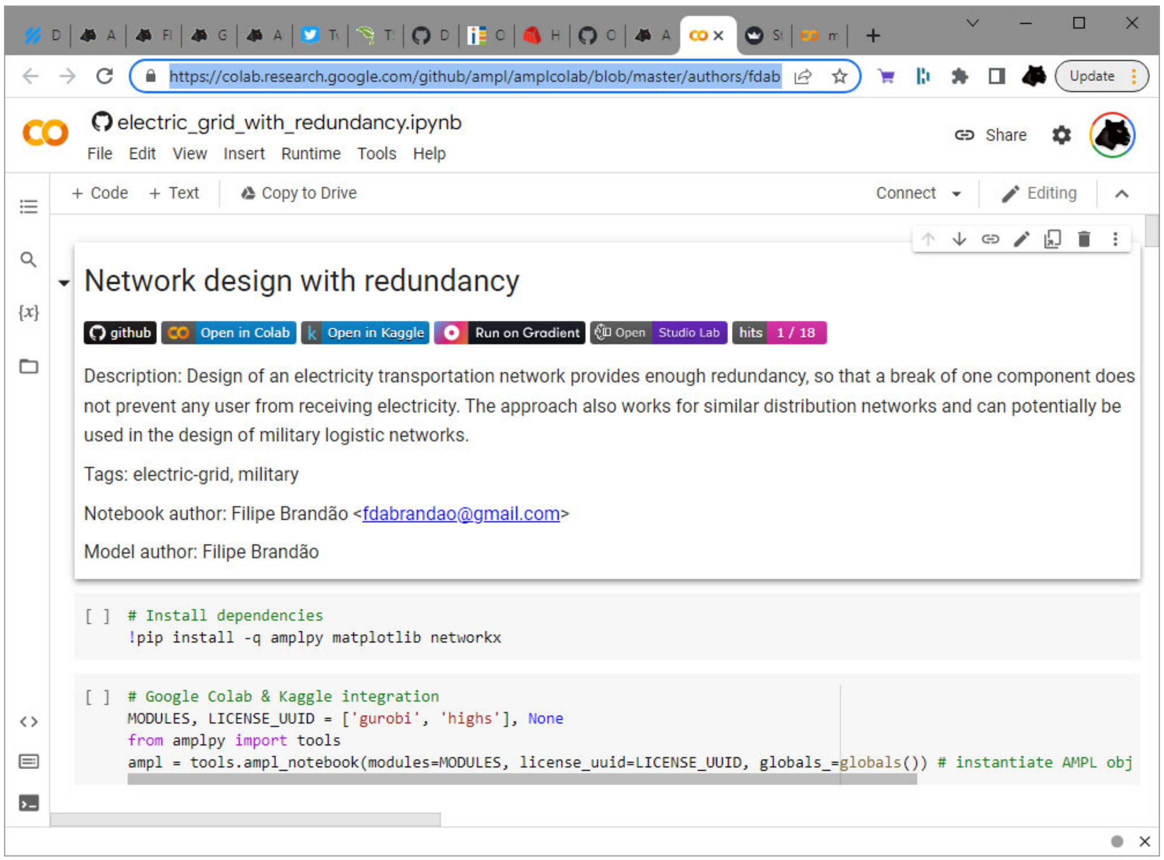
https://dev.ampl.com/ampl/python/



Integration with Python

AMPL Model in a Python Notebook

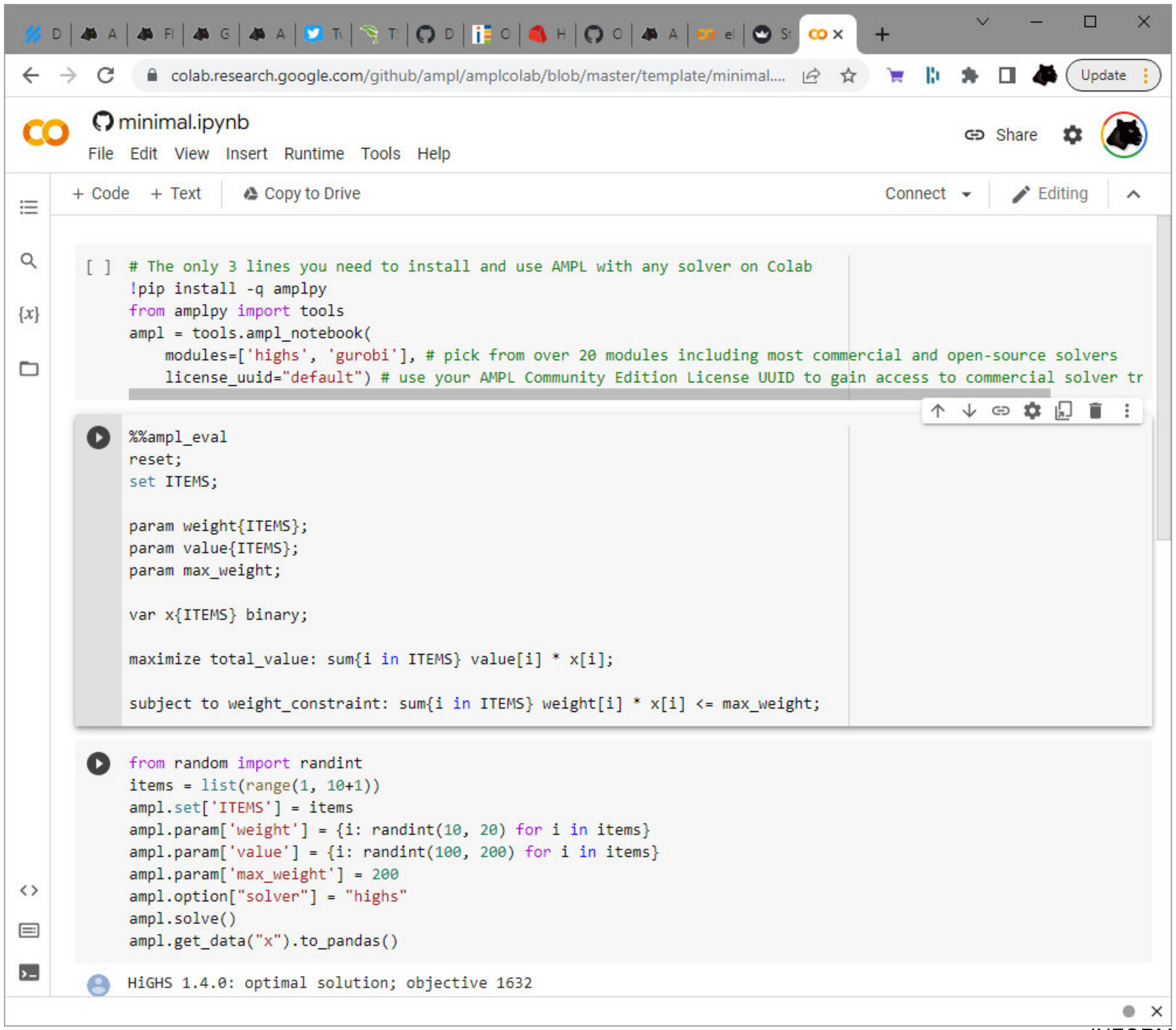
https://colab.research.google.com/github/ampl/amplcolab/blob/master/authors/fdabrandao/military/electric_grid_with_redundancy.ipynb



Integration with Python

Minimal Notebook to Get Started

https://try.ampl.com



Integration with Python

AMPL Model in a Streamlit Application

https://nqueens-with-ampl.streamlit.app/