UCDAVIS
CENTER FOR WATERSHED SCIENCES

CALVIN (Python Version)
Fall 2018 Shortcourse

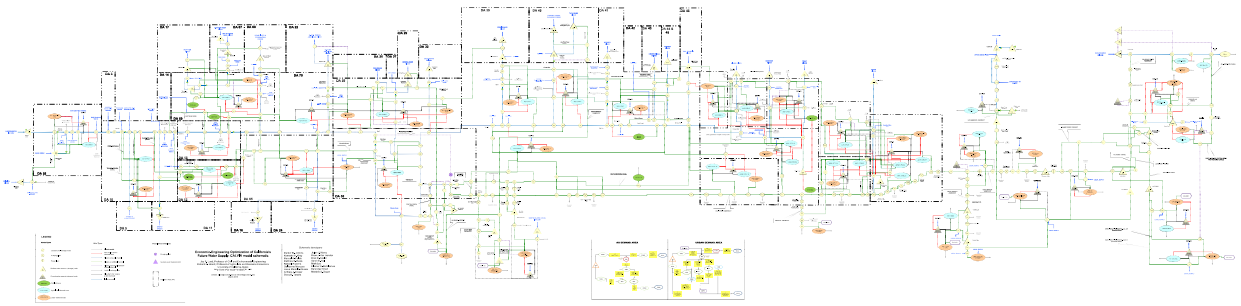
Prepared by

Mustafa S. Dogan

msdogan@ucdavis.edu

Date: October 5, 2018

Location: Center for Watershed Sciences Conference Room



Date: Friday, October 5, 2018

Location: UC Davis, Center for Watershed Sciences Conference Room

Registration: https://goo.gl/forms/6nUVGddb8xhUOSVn1

Shortcourse GitHub Repo: https://github.com/msdogan/CALVIN-shortcourse

Tentative Agenda and Topics

10.00 – 10:05 am	-	Introduction
10:05 – 10:30 am	-	Set-up required software
10:30 – 11:15 am	-	CALVIN Theory
11:15 – 12:00 pm	-	HOBBES overview and exporting network data
12:00 – 01:00 pm	**-**	**Lunch break**
01:00 – 01:15 pm	-	CALVIN Python version updates and Pyomo
01:15 – 02:15 pm	-	“Abstract model”: run and analyses
02:15 – 03:15 pm	-	“Concrete model”: run and analyses
03:15 – 03:30 pm	**-**	**Break**
03:30 – 04:30 pm	-	Postprocessing and analyzing results

Summary

This shortcourse is intended for those who are interested in California's water supply system and large-scale water optimization modeling. Mechanics of the CALVIN model will be covered. This crash course introduces open-source CALVIN version modeled in Python-based Pyomo environment, employing faster solvers and giving an opportunity for better representation of the system. It walks through steps for required software installation process for the CALVIN model, as well as creating a model run and postprocessing results.

Recommended readings

- **Original publication of CALVIN** (Draper et al., 2003):

Draper, A. J., Jenkins, M. W., Kirby, K. W., Lund, J. R., & Howitt, R. E. (2003). Economic-Engineering Optimization for California Water Management. *Journal of Water Resources Planning and Management*, *129*(3), 155–164. https://doi.org/10.1061/(ASCE)0733-9496(2003)129:3(155)

- **Open-source Python version of CALVIN** (Dogan et al., 2018):

Dogan, M. S., Fefer, M. A., Herman, J. D., Hart, Q. J., Merz, J. R., Medellín-Azuara, J., & Lund, J. R. (2018). An open-source Python implementation of California's hydroeconomic optimization model. *Environmental Modelling & Software*, *108*, 8–13. https://doi.org/10.1016/j.envsoft.2018.07.002

Documentation and related theses/publications:

https://calvin.ucdavis.edu/node

https://watershed.ucdavis.edu/shed/lund/CALVIN/

Table of Contents

Required Software

Several different installations are required in order to export network data from HOBBES, and then create and run CALVIN model. CALVIN is an open-source project, so all installations are free of charge! CALVIN can connect to commercial solvers, some of which are free for academic purposes, but there we will use open-source solvers for the purposes of this shortcourse. Required installations are cross-platform, so they (should) run on Windows or Mac OS.

GitHub account

Model data and components are hosted in GitHub, a web-based data and code hosting service with version control. So, having a GitHub account and downloading GitHub desktop is strongly recommended but files can be downloaded from hosted repositories as zip files without an account.

- Sign up for a GitHub account: https://github.com/
- Download GitHub desktop: https://desktop.github.com

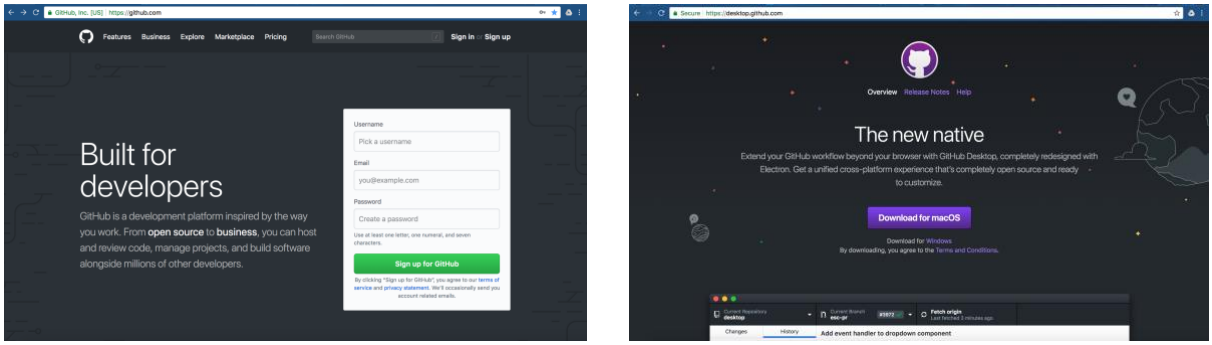


Figure 1. GitHub sign-up page and desktop

Cloning or downloading required GitHub repositories

After signing up and installing GitHub desktop, go to each of these following repositories and clone them, or if you can download as zip file as shown below. Please read "Readme" files by scrolling down in each repository. Required repositories:

- https://github.com/ucd-cws/calvin-network-data
- https://github.com/ucd-cws/calvin-network-tools
- https://github.com/ucd-cws/calvin

Bonus

- https://github.com/msdogan/CALVIN-shortcourse

Cloning steps (repeat this for each repository):

1. Go to repository link (above)
2. Click on green “Clone or download” button.
3. “Open in Desktop” or if you do not have account or GitHub desktop, click on “Download ZIP”

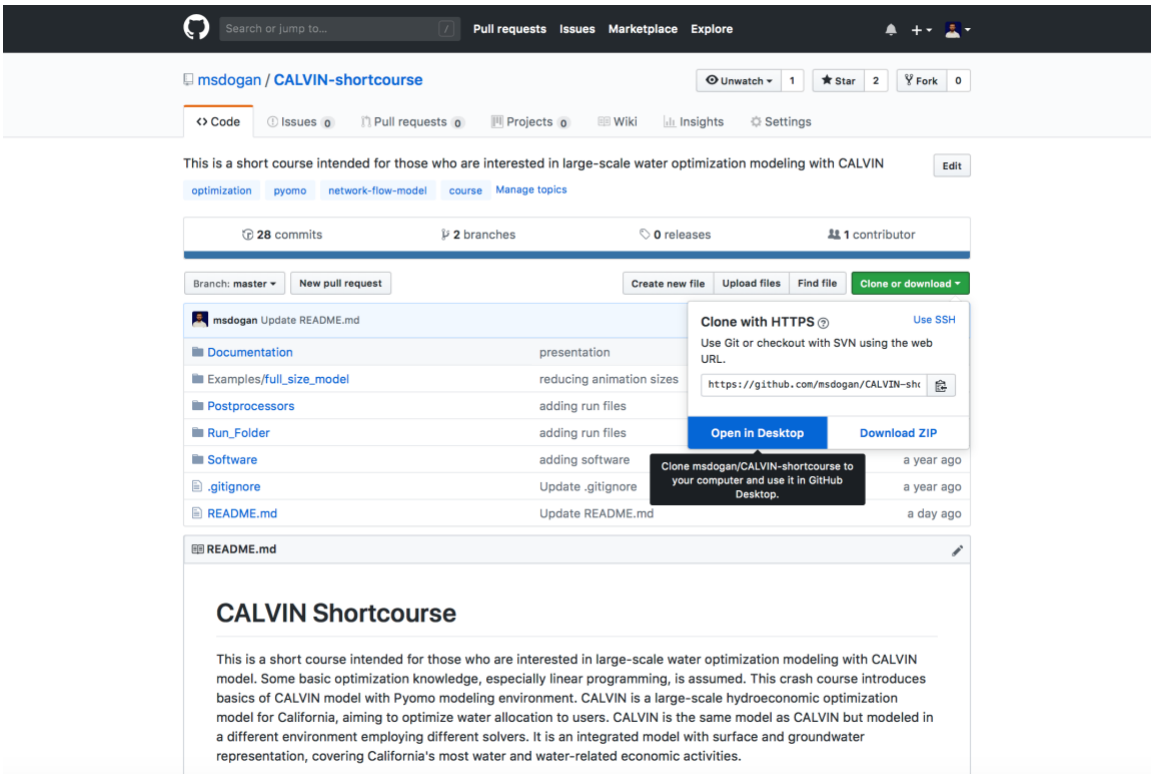


Figure 2. Cloning a repository

Installing Python and other libraries via Anaconda package

We will use Anaconda, a free and open source distribution of the Python programming language for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Anaconda distribution has several packages, including Numpy, Scipy and Pandas. If you already have Anaconda with Python 3.0+, you do not need to install again, you can proceed to Pyomo and solver installations.

Download link: https://www.anaconda.com/download

Installing Pyomo and its solver

Pyomo is a high level Python optimization modeling library. Pyomo is like a interface between your data and solver in a sense that it prepares your parameter data, decision variables, objective function, and constraints in way that solvers can understand and solve. Pyomo then gets raw results from solvers and organizes for us to Postprocess. Pyomo simply communicates between data and solvers. CALVIN can connect to several solvers but we will use GLPK, an open-source linear programming (LP) solver.

After installing anaconda (python v3+), installing pyomo and GLPK solver are straightforward. We will use command line to install required packages.

```
conda install -c conda-forge pyomo pyomo.extras glpk
```

Mac OS Pyomo and solver installation

Open Terminal (you can search for “Terminal” in your Spotlight Search if it is not in your Dock), and **type** the following command and hit **enter** to run.

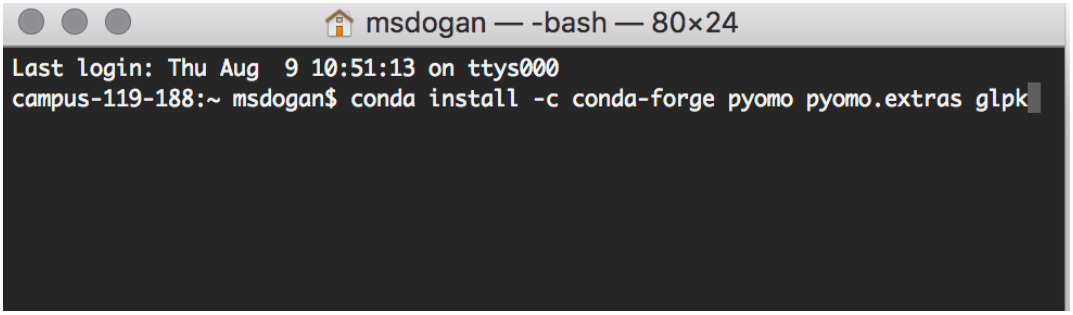


Figure 3. Mac OS Terminal Pyomo and GLPK solver installation

Windows Pyomo and solver installation

You can open command line by searching “Command Prompt” in your Windows search. **Type** command above in your command line console and hit **enter** to run.

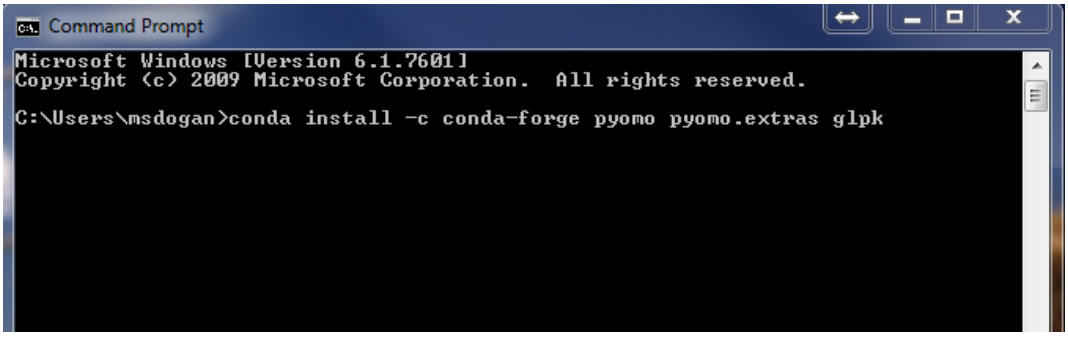


Figure 4. Windows Command Prompt Pyomo and GLPK solver installation

Here is a successful installation of Pyomo and GLPK solver

```
campus-119-188:~ msdogan$ conda install -c conda-forge pyomo
pyomo.extras glpk
Solving environment: done

## Package Plan ##

  environment location: /Users/msdogan/anaconda

  added / updated specs:
    - glpk
    - pyomo
    - pyomo.extras

The following packages will be downloaded:
    package            |  build
    ---------------------------|-----------------
    pyutilib-5.6.3     |  py27_0         333 KB  conda-forge
    glpk-4.65          |  h16a7912_1     1.0 MB  conda-forge
    pyomo.extras-3.3   |  py27_0         3 KB  conda-forge
    pyomo-5.5.0        |  py27_1         2.7 MB  conda-forge
    ------------------------------------------------------------
                           Total:         4.1 MB
The following packages will be UPDATED:

glpk:    4.61-0 conda-forge --> 4.65-h16a7912_1 conda-forge
pyomo:    5.1.1-py27_0 conda-forge --> 5.5.0-py27_1 conda-forge
pyomo.extras: 3.2-py27_0 conda-forge --> 3.3-py27_0 conda-forge
pyutilib: 5.4.1-py27_0 conda-forge --> 5.6.3-py27_0 conda-forge

Proceed ([y]/n)? y

Downloading and Extracting Packages
Pyutilib 5.6.3: ######################################### | 100%
glpk 4.65: ############################################## | 100%
```

```
pyomo.extras 3.3: ##################################### | 100%
pyomo 5.5.0: ######################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Optional: Text editor

If you do not have a text editor, I recommend Sublime Text https://www.sublimetext.com or Notepad++ https://notepad-plus-plus.org. Text editor can be used to open data file. With Sublime Text you can also run Python.

CALVIN Theory and Background

Developed in early 2000s, **CAL**ifornia **V**alue **I**ntegrated **N**etwork model (CALVIN) combines ideas from economics and engineering optimization with advances in software and data to suggest more integrated management of water supplies regionally and throughout California. CALVIN is a hydro-economic optimization model for California's advanced water infrastructure that integrates the operation of water facilities, resources, and demands, and it aims to optimize surface and groundwater deliveries to agricultural and urban water users. It allocates water to minimize water scarcity and operating costs (maximize statewide agricultural and urban economic value), considering physical and policy constraints. It replicates water market operations transferring water from users with lower willingness-to-pay (WTP) to users with higher WTP. CALVIN uses historical hydrology and 2050 water demand projections for its operations.

CALVIN forces quantitative understanding of integrated water and economic sys- tem. Motivation for the CALVIN effort include:

- making better sense of integrated system and operations
- seeking ways to improve system management
- quantifying user willingness to pay for additional water
- finding insights into changes in physical capacities and policies

CALVIN is a network-flow model over a physical network represented by a set of nodes N and links A. Links are defined by $(i,j,k) \in A$, where i is the origin node, j is the terminal node, and k is piecewise component used to represent nonlinear penalty (or cost) curves with a convex piecewise delineation. The component k creates multiple links from origin node i to terminal node j with monotone decreasing unit costs. Each link has the following properties: flow X_{ijk}, which is the decision variable; unit cost c_{ijk};

lower bound l_{ijk}; upper bound u_{ijk}; and amplitude or loss factor a_{ijk}. The objective function and constraints are:

$$\min_X z = \sum_i \sum_j \sum_k c_{ijk} X_{ijk} \qquad \textit{Equation 1}$$

$$X_{ijk} \geq l_{ijk}, \forall (i,j,k) \in A \qquad \textit{Equation 2}$$

$$X_{ijk} \leq u_{ijk}, \forall (i,j,k) \in A \qquad \textit{Equation 3}$$

$$\sum_i \sum_k X_{jik} - \sum_i \sum_k a_{ijk} X_{ijk} = 0\,, \forall j \in N \qquad \textit{Equation 4}$$

The objective function (*Equation 1*) is a summation over all links (i,j,k) and represents the total cost of flow conveyed in the network. *Equation 2*, *Equation 3*, and *Equation 4* represent the lower bound, upper bound, and mass balance constraints, respectively.

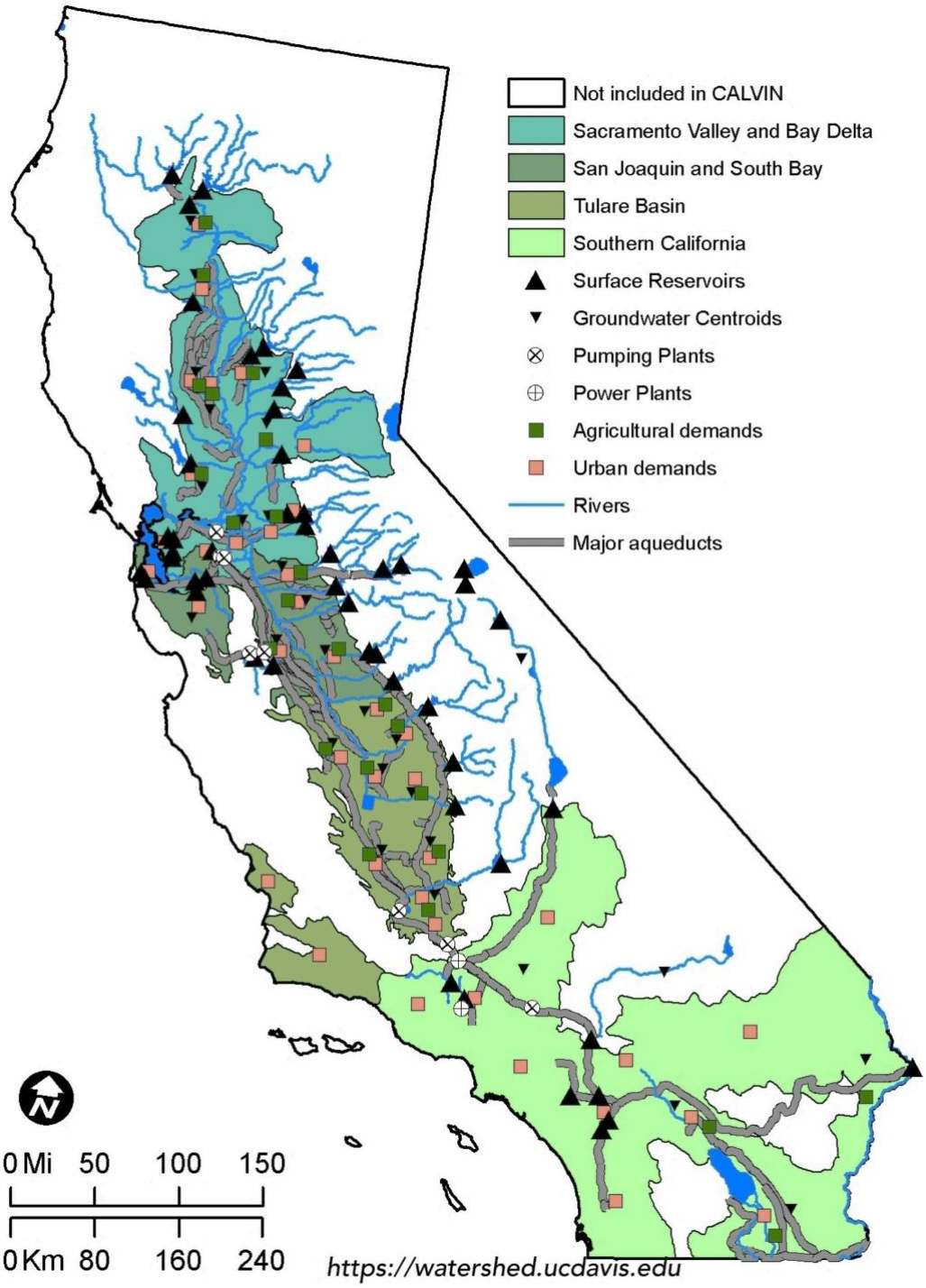


Figure 5. California's water infrastructure and CALVIN coverage

CALVIN online schematic and visualization tool

This tool shows schematic of georeferenced nodes and links on a California map by reading data from HOBBES database. Regions and network features can be filtered through its interface. You can see information about the feature (node or link) by clicking. If there is any time-series or cost data, plots will be shown. Visualization tool also has an animation layer, which shows already optimized flow and storage operations. Animation layer can be active by checking box on the bottom of page.

Link here: https://cwn.casil.ucdavis.edu

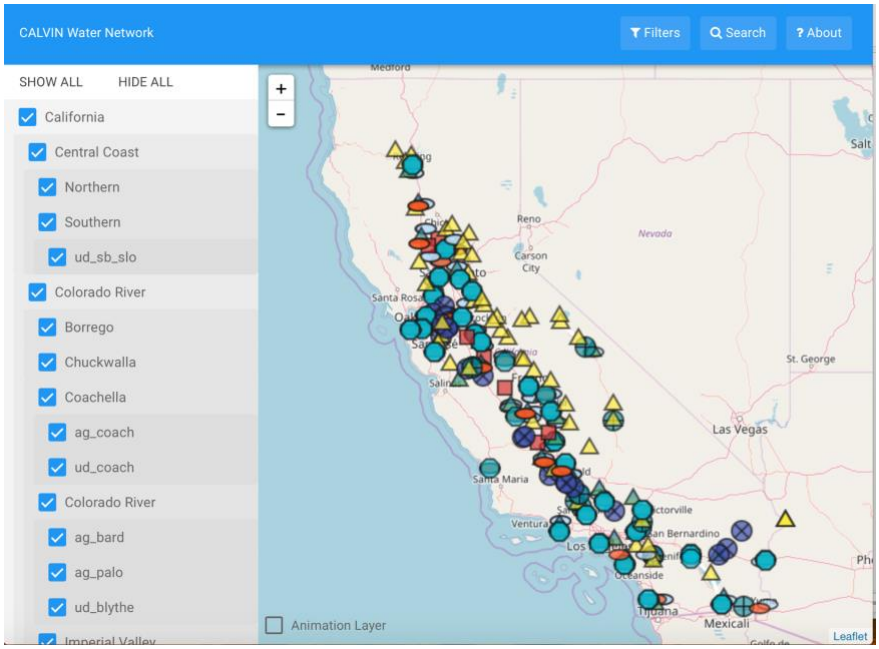


Figure 6. CALVIN visualization tool

Agricultural Demand

CALVIN optimizes reservoir and water supply operations to minimize water scarcity and operating costs. Water scarcity and economic costs occur when a user's total is not met. As a result, agricultural production losses occur. Demand and penalty curves quantifying how much loss occur depending on water delivery differ for each subregion and are obtained from Statewide Agricultural Production (SWAP) model (Howitt et al., 2012).

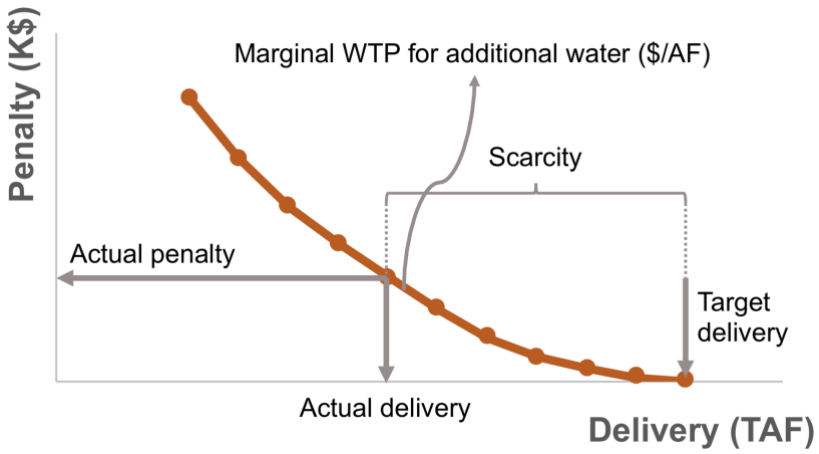


Figure 7. Typical penalty curve for water delivery

In CALVIN, there are two water sources available for agricultural users: groundwater and surface water. Both supplies are aggregated in one node (A###) and after applying reuse multiplier on link (A###-HU###), demand penalties and delivery targets are applied on HU###-CVPMG and CVPMS links. Agricultural demand areas are divided into two parts based on their return flow to either groundwater (CVPMG) or surface water (CVPMS). After that consumptive use ratios (amplitudes) are applied and remaining water goes back to network.

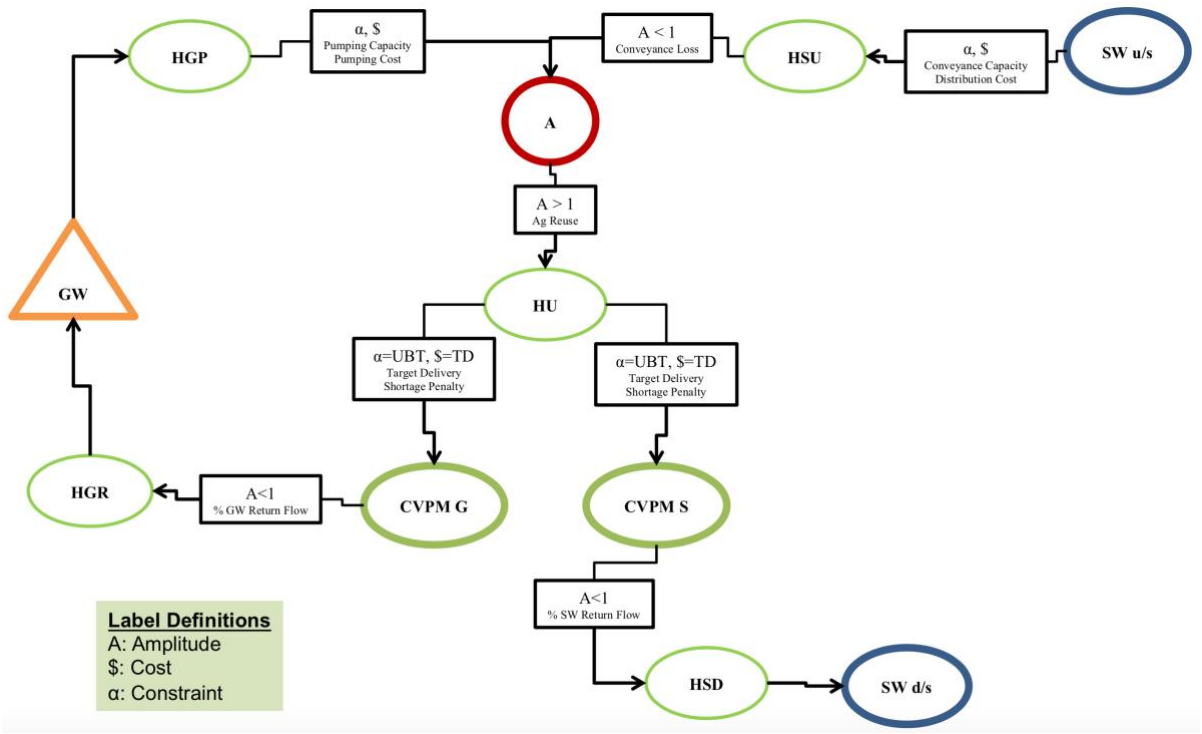


Figure 8. Generalized representation of agricultural demand in CALVIN

Urban Demand

Urban areas have more water supply sources available than agricultural users. In addition to groundwater and surface water, desalination, potable and nonpotable re-cycled wastewater are available for urban users. Surface water deliveries are treated in a water treatment plant node (WTP###) and all sources, except nonpotable recycled wastewater, are aggregated in U### nodes. CALVIN's urban areas split into three uses: exterior, interior, and industrial. While potable recycled wastewater is available for all three uses (HP###)), nonpotable recycled wastewater is available only for exterior and industrial uses (HNP###). After applying consumptive use ratios on return links, industrial and interior return flows are sent to wastewater treatment plant nodes (WWP###) and then returned to surface or groundwater.

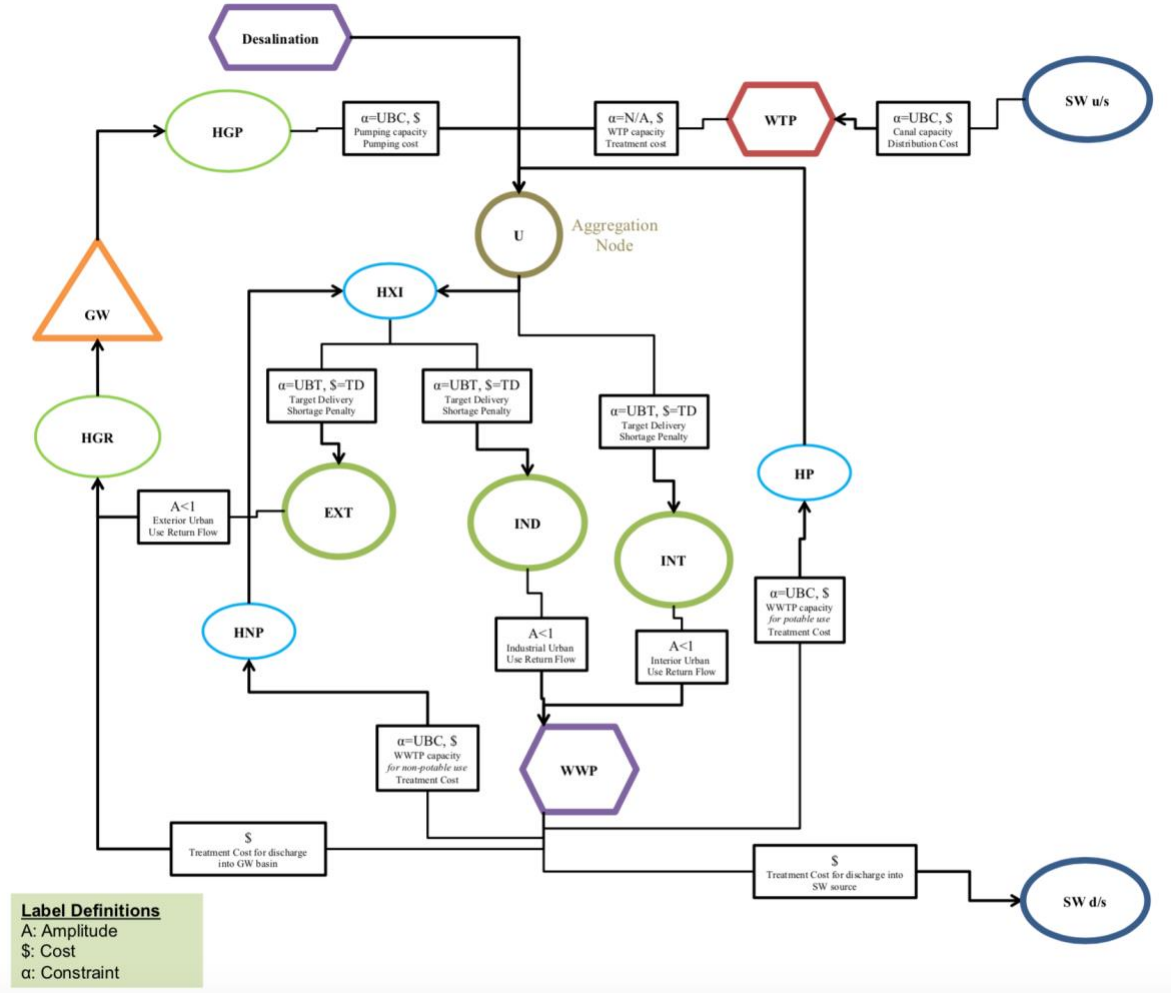


Figure 9. Urban water demand in CALVIN

Environmental Demand

Wildlife refuge areas do not have an economic representation in CALVIN. Deliveries are represented with constrained flows, which assures that environmental deliveries must be met before other deliveries. Groundwater, surface water and agricultural return flow supplies, which are aggregated in R### nodes, are available for wildlife refuge users and all return flows go to surface flow.

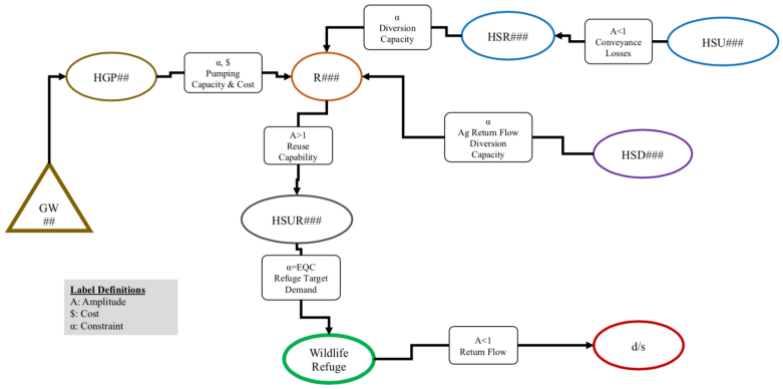


Figure 10.Wildlife refuge demand in CALVIN

In addition to wildlife refuge, CALVIN represents minimum in-stream flow requirements. These requirements are mostly on rivers (usually below reservoir releases or diversion points) and represented as lower bound constraints on the water network.

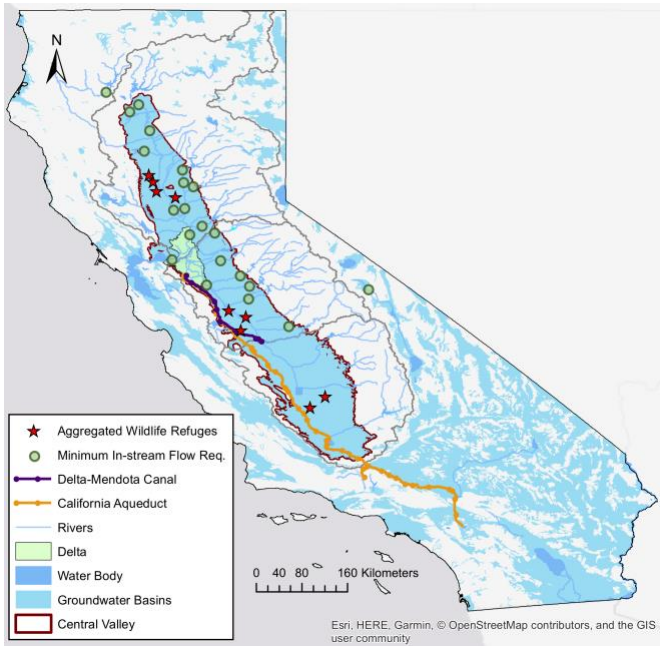


Figure 11. CALVIN's minimum in-stream flow locations and wildlife demands

HOBBES Database

The HOBBES Project is a bottom up approach to improve and organize the data for water modeling efforts in California. This effort is trying to provide a venue for modelers in California and elsewhere to create an open, organized and documented quantitative representation of the state's intertied water resources system. Geocoded elements in this database can be interactively converted into tiered networks able to be solved by multiple modeling platforms depending on user preferences, with the appropriate translators. Many HOBBES tools will be web-based with exporting capabilities to the most common analytical and modeling software. HOBBES serves as a cross- platform for data storage, display and documentation. It is a framework for database that aims to better organize data and makes model integration and communication easier by using common format and metadata. Classical approach in modeling is that first model is built and then required data are collected. But HOBBES reverses this order; it serves as a data hub and models are built on top of this database. HOBBES uses GitHub to keep track of changes and documentation. It also has an animation tool to display data as shown earlier.

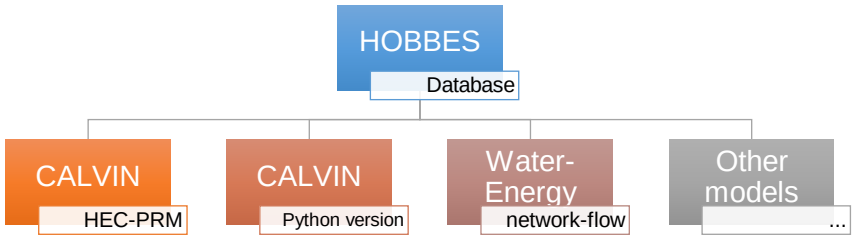


Figure 12. HOBBES database and model integration

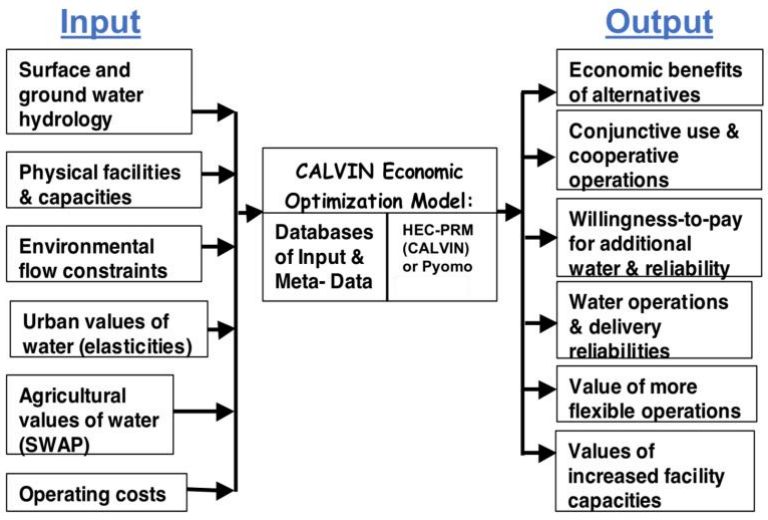


Figure 13. CALVIN data overflow with input and output data

Resolving Infeasibilities: Debug Mode

Debug mode adds two more links, “debugsource” and “debugsink”, to each node in the network to prevent infeasibilities. While “debugsource” injects water, “debugsink” removes water from the system if needed at very high cost, such as $2,000,000 per acre-foot, which is higher than any other cost in the system. Since the objective is to minimize statewide costs, the model does not use these debug links unless it is really needed, such as mass balance violations. Think of a case where there is a minimum in-stream flow requirement downstream, and your inflow is less than the requirement. The model will try to meet the environmental constraint because it is hard coded and the model cannot change it, so the operation will terminate saying the result is infeasible, and modeler will not know where the problem is because there is hundreds of those requirements. So, debug mode will inject water in that case helping the model find a feasible solution. After that the modeler will look at flows at debug links, and if any of these flows are greater than zero, it means there is mass balance problem. The modeler knows the location and magnitude of the problem and will find a solution.

There are two options for running CALVIN in debug mode: If you are using abstract model, network data can be exported in debug mode, where additional links are added during export process. Second, there is a debug mode in concrete model that adds additional links to the network even if network data is exported without debug mode. Concrete model also iterates to eliminate infeasibilities (debug flows) and creates a report and feasible dataset. Both abstract and concrete models will be discussed in following sections.

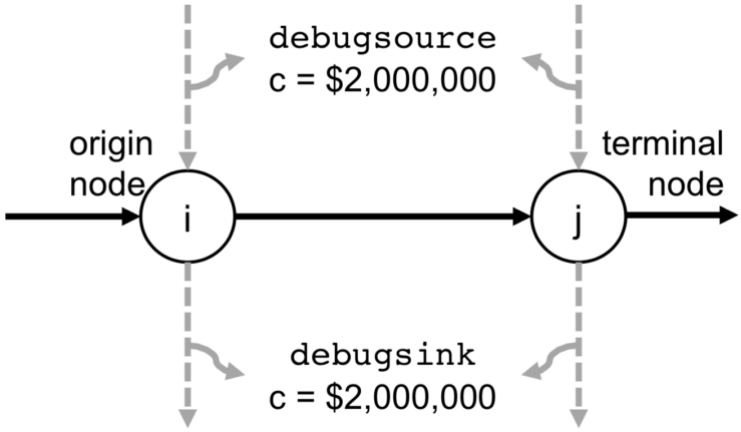


Figure 14. If model is run in debug mode, two additional links are added to each node at very high cost

Updated CALVIN Model (Python Version)

The vast improvements in computing power since CALVIN's inception and its ability to explore potential scenarios in California water supply provide an opportunity to move the network structure and data to a new optimization platform. General design goals:

- *Cross-platform*: Model should run on Windows/OSX/Linux.
- *Open data formats*: Input and output data should use only non-proprietary formats, such as CSV and JSON.
- *Freely available*: Programming language and solvers should be free and open-source. Several solvers (Gurobi, CPLEX) CALVIN can connect to are cost-free only for academic use, but they are not strictly required.
- *Separation of model and data*: The HOBBES project stores the network dataset independent from any particular model, allowing this work to use multiple state-of-the-art solvers or models.

Modeling in Pyomo

Pyomo, a high level optimization modeling language in Python, provides a flexible, extensible modeling framework that supports the central ideas of modern algebraic modeling languages within a widely used programming language. Pyomo supports the formulation and analysis of mathematical models for complex optimization applications. Mathematical concepts of optimization:

- **Variables**: These represent unknown or changing parts of a model
- **Parameters**: These are symbolic representations for real-world data, which might vary for different scenarios.
- **Relations**: These are equations, inequalities, or other mathematical relationships that define how different parts of a model are connected to each other.

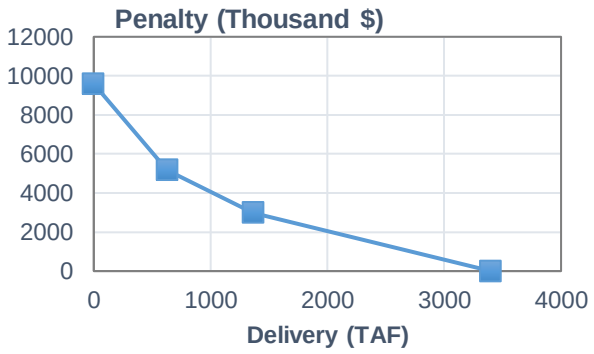
CALVIN Python version is modeled using Pyomo optimization modeling language. There are two types of modeling options in Pyomo: Abstract and Concrete models. Abstract models are easier to understand and implement, while concrete models provide more flexibility, such as they can be called in a loop and do not require command line running. Either abstract or concrete, modeling structure and outputs do not change.

A small network example (Example 1)

Below, there is an abstract and concrete modeling example of a small network (Example 1). Different model sizes, time-steps, or adding more nodes-links to network to not change structure files. Only `data.dat` files or `links.csv` files change and CALVIN is run with defined data file.

Example 1 is a network modeling of 1 reservoir (SR_SHA) with two time-steps (1983-10-31, 1983-11-30). Network parameters are shown in table below.

Table 1. Small network example properties (Example 1)

Initial storage (TAF)	3686.84
Ending period storage (TAF)	2923.297
Reservoir inflow 1983-10-31 (TAF/m)	301.765
Reservoir inflow 1983-11-30 (TAF/m)	650.408
Evaporation loss factor	0.99716
Penalty curve (due to (lost) hydropower benefits)	
Reservoir storage capacity (TAF)	3400 (less due to flood storage)
Reservoir release capacity	None (assumed for simplicity)

Converting Example 1 into network-flow problem

Data file above is visualized on a network schematic below. In CALVIN, all water comes from a super source node called “SOURCE” and goes into super sink node called “SINK”, and only for these two nodes, mass balance constraint is skipped. “INITIAL” and “FINAL” represent initial and ending storage boundary conditions. “INFLOW” nodes deliver water to reservoir at each time-step. There is two time-steps (1983-10-31 and 1983-11-30) in Example 1. Storage is water flow in time and represented in

“SR_SHA.1983-10-31” to “SR_SHA.1983-11-30” link with a piecewise cost curves (k=0,1,2). In order to create a small sample, I used “SINK” nodes, and this represents reservoir releases below example. However, in original model, instead of “SINK”, reservoirs are connected to other components of the network. You can simply think that if more elements are added to Example 1, the network expands horizontally. If more time-steps are added to Example 1, the network expands vertically.

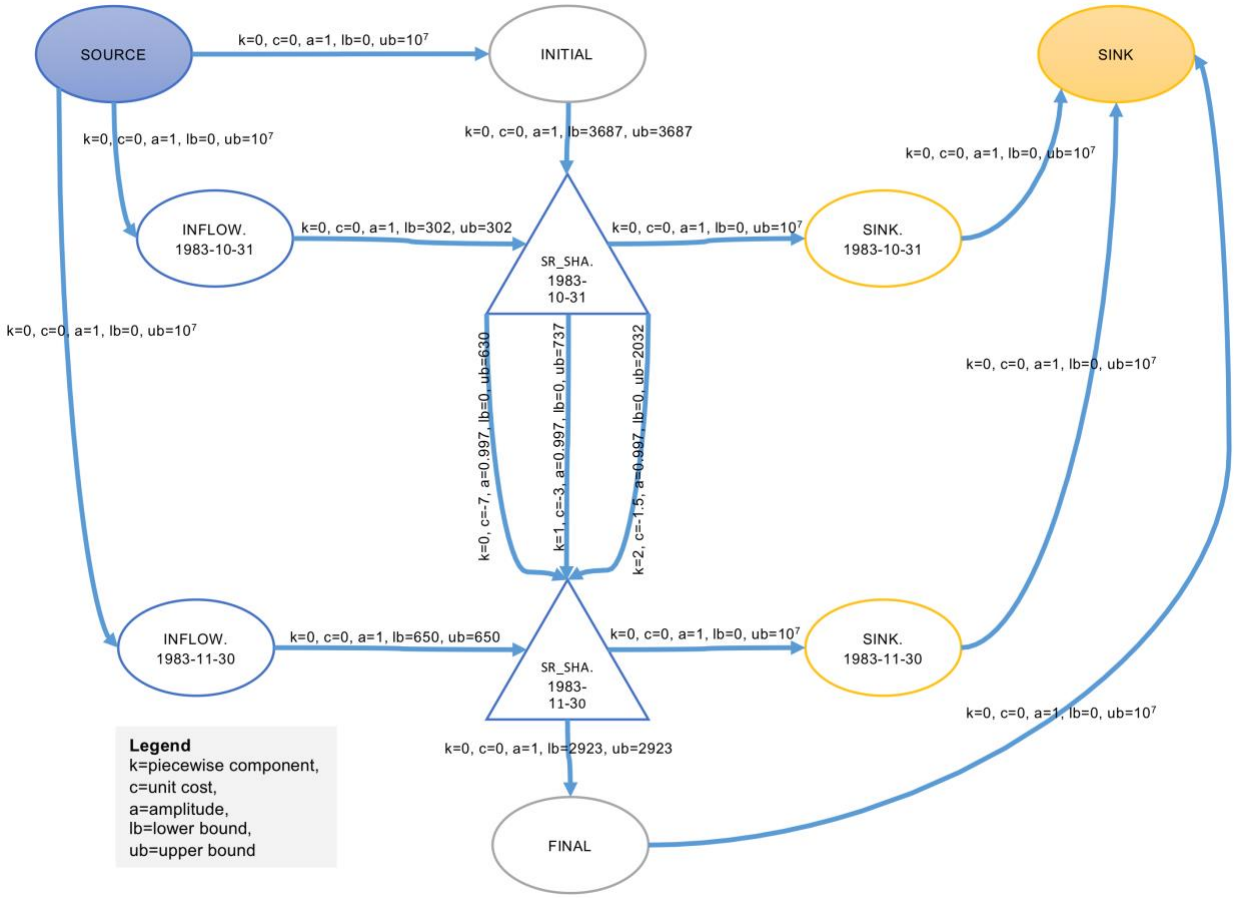


Figure 15. Example 1 network schematic with properties (k, c, a, lb, ub)

Abstract CALVIN Model

Abstract models require two files:

- **Python script** (`calvin.py`) that describes decision variables and has modeling relations: objective function and constraints.
- **Data file** (`data.dat`) that has parameter and properties. This data file is tab separated.

Python Script (Structure file)

```
from __future__ import division
from pyomo.environ import
import itertools

model = AbstractModel()

# Nodes in the network
model.N = Set()

# Network arcs
model.k = Set()
model.A = Set(within=model.N*model.N*model.k)

# Source node
model.source = Param(within=model.N)
# Sink node
model.sink = Param(within=model.N)
# Flow capacity limits
model.u = Param(model.A)
# Flow lower bound
model.l = Param(model.A)
# Link amplitude (gain/loss)
model.a = Param(model.A)
# Link cost
model.c = Param(model.A)

# The flow over each arc
model.X = Var(model.A, within=Reals)

# Minimize total cost
def total_rule(model):
    return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
model.total = Objective(rule=total_rule, sense=minimize)

# Enforce an upper bound limit on the flow across each arc
def limit_rule_upper(model, i, j, k):
    return model.X[i,j,k] <= model.u[i,j,k]
model.limit_upper = Constraint(model.A, rule=limit_rule_upper)

# Enforce a lower bound limit on the flow across each arc
def limit_rule_lower(model, i, j, k):
    return model.X[i,j,k] >= model.l[i,j,k]
model.limit_lower = Constraint(model.A, rule=limit_rule_lower)

# Enforce flow through each node (mass balance)
def flow_rule(model, node):
  if node in [value(model.source), value(model.sink)]:
      return Constraint.Skip
  outflow = sum(model.X[i,j,k]/model.a[i,j,k] for i,j,k in model.A)
  inflow = sum(model.X[i,j,k] for i,j,k in model.A)
  return inflow == outflow
model.flow = Constraint(model.N, rule=flow_rule)
```

Data file (Example 1)

The data file data.dat includes list of nodes, and list of links (i,j,k) with properties. All links have cost c, amplitude a, lower bound l, and upper bound u. Below is an example data file.

```
set N :=
INITIAL   SR_SHA.1983-10-31  SR_SHA.1983-11-30  FINAL     INFLOW.1983-10-31  INFLOW.1983-11-30
          SINK.1983-10-31    SINK.1983-11-30    SOURCE    SINK;

set k := 0 1 2 3 4 5;

param source := SOURCE;
param sink := SINK;

param: A: c a l u :=
SOURCE    INITIAL   0         0         1         0         10000000
INITIAL   SR_SHA.1983-10-31  0         0         1         3686.84   3686.84
SR_SHA.1983-10-31  SR_SHA.1983-11-30  0         -7.00344  0.99716   0         630.403
SR_SHA.1983-10-31  SR_SHA.1983-11-30  1         -2.97378  0.99716   0         737.389
SR_SHA.1983-10-31  SR_SHA.1983-11-30  2         -1.46632  0.99716   0         2032.208
FINAL     SINK      0         0         1         0         10000000
SR_SHA.1983-11-30  FINAL     0         0         1         2923.297  2923.297
SOURCE    INFLOW.1983-10-31  0         0         1         0         10000000
INFLOW.1983-10-31  SR_SHA.1983-10-31  0         0         1         301.765   301.765
SOURCE    INFLOW.1983-11-30  0         0         1         0         10000000
INFLOW.1983-11-30  SR_SHA.1983-11-30  0         0         1         650.408   650.408
SR_SHA.1983-10-31  SINK.1983-10-31    0         0         1         0         10000000
SR_SHA.1983-11-30  SINK.1983-11-30    0         0         1         0         10000000
SINK.1983-10-31    SINK      0         0         1         0         10000000
SINK.1983-11-30    SINK      0         0         1         0         10000000;
```

Running abstract model

First run `compiler.py`, which prepare input data.dat for the abstract model. This script takes matrix data (`links.csv`) exported from HOBBES and creates tab separated input data file for Pyomo.

Following command should be run in the local directory where calvin.py and data.dat files are located.

```
pyomo solve --solver=glpk --solver-suffix=dual
calvin_abstract.py data_example1.dat --stream -solver --json
```

If the model is run successfully, following should be displayed:

```
campus-117-092:base_case msdogan$ pyomo solve --solver=glpk --solver-
suffix=dual calvin_abstract.py data.dat --stream-solver --json
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[    0.02] Applying solver
GLPSOL: GLPK LP/MIP Solver, v4.65
Parameter(s) specified in the command line:
 --write /Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpDSDOmX.glpk.raw
 --wglp /Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpEaoFmb.glpk.glp
 --cpxlp /Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpfL_tVk.pyomo.lp
Reading problem data from '/Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpfL_tVk.pyomo.lp'...
39 rows, 16 columns, 55 non-zeros
198 lines were read
Writing problem data to '/Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpEaoFmb.glpk.glp'...
162 lines were written
GLPK Simplex Optimizer, v4.65
39 rows, 16 columns, 55 non-zeros
Preprocessing...
~     0: obj =  -9.587689481e+03  infeas =  0.000e+00
OPTIMAL SOLUTION FOUND BY LP PREPROCESSOR
Time used:   0.0 secs
Memory used: 0.0 Mb (40557 bytes)
Writing basic solution to '/Users/msdogan/Documents/github/CALVIN-
shortcourse/Run_Folder/base_case/tmpDSDOmX.glpk.raw'...
64 lines were written
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: -9587.6894813
    Solver results file: results.json
[    0.06] Applying Pyomo postprocessing actions
[    0.06] Pyomo Finished
```

Postprocessing abstract model

Abstract model generates `results.json` file and puts all output in one single file. However, often we need outputs in time-series format and separate files for flow, storage, evaporation, and dual values. CALVIN's Python-based postprocessor scripts create those separate files and save them as *.csv which can be easily shared and used to create figures.

Run `postprocess.py` to create time-series data from `results.json`.

Concrete CALVIN Model

The concrete model is a “user friendly version” can be run directly without command line, and the model can be run within a loop, which makes running different simulations or limited foresight simulation easier. Both abstract and concrete models use the same objective function, decision variables and constraints. The concrete model is just another way of modeling in Pyomo. Concrete model uses object-oriented programming, where functions are defined in one file (`calvin.py`) and called in another file (`main.py`). Concrete model still requires exported network matrix from HOBBES, but it functions to organize input data and postprocess outputs defined in concrete model (no need to run `compiler.py` and `postprocess.py`).

Running concrete model

You just need to run `main.py` as a regular Python script. It will call required commands to create and run the model and postprocess outputs.

`main.py` script (Example)

```
from calvin import *

# specify data file
calvin = CALVIN(links_example1.csv')

# create pyomo model from specified data file
calvin.create_pyomo_model(debug_mode=False)

# solve the problem
calvin.solve_pyomo_model(solver='glpk', nproc=1,
debug_mode=False)

# postprocess results to create time-series files
postprocess(calvin.df, calvin.model, resultdir='results')
```

Concrete model in debug mode

Debug mode in concrete model automatically reduces and eliminates infeasibilities in the network. Infeasibilities occur if any constraint is not met (lower bound, upper bound or mass balance), resulting in an infeasible solution. Debug mode here displays where problem is tries to eliminate problem if it encounters an infeasible problem. Below there Is an infeasible network data (links_infeasible.csv) of a 1-year CALVIN model (Oct. 2002-Sep. 2003, monthly). Turn on “debug_mode=True” to run concrete model in debug mode.

```
from calvin import *

# specify data file
calvin = CALVIN(links_infeasible.csv')

# create pyomo model from specified data file
calvin.create_pyomo_model(debug_mode=True)

# solve the problem
calvin.solve_pyomo_model(solver='glpk', nproc=1,
debug_mode=True)

# postprocess results to create time-series files
postprocess(calvin.df, calvin.model, resultdir='results-
infeasible')
```