CALVIN & PyVIN Shortcourse Handout

Mustafa S. Dogan [*],

April 28, 2017

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 also introduces open-source PyVIN model which is the same model as CALVIN but 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 PyVIN model, as well as creating a model run and postprocessing results.

Contents

1 Prerequisites **3**
1.1 Command line 4
1.1.1 Windows 4
1.1.2 Mac OS 7

2 Introduction and background **9**

3 HOBBES database **12**
3.1 Data flow overview 12

4 Updated version: PyVIN **13**
4.1 Model architecture 13
4.2 Debug mode 14
4.3 PyVIN model 14

5 Your first model run **16**
5.1 Postprocessing results 17

6 References and useful links **18**

[*]Graduate Student, Dept. of Civil and Env. Eng., Univ. of California, Davis, 1 Shields Avenue, Davis, CA 95616. E-mail: msdogan@ucdavis.edu.

Tentative Agenda and Topics

10:00 - 10:15 am	–	Introduction and set-up
10:15 - 11:15 am	–	CALVIN theory and model introduction
11:15 - 11:30 am	–	**Break**
11:30 - 11:45 am	–	HOBBES database
11:45 - 12:00 pm	–	Data flow overview
12:00 - 1:00 pm	–	**Break**
1:00 - 1:20 pm	–	PyVIN updates and model architecture
1:20 - 1:35 pm	–	A PyVIN example
1:35 - 1:50 pm	–	**Break**
1:50 - 2:10 pm	–	Required software and installation
2:10 - 2:30 pm	–	Your first PyVIN run
2:30 - 3:00 pm	–	Postprocessing results

1 Prerequisites

Please bring your **laptop** with below software dependencies installed if you want a hands-on PyVIN experience. Install following software in advance since some of them, such as Anaconda, takes long time to download and install. If you have any question, e-mail msdogan@ucdavis.edu

- **Python version 3+ with Anaconda or Miniconda (not Python v3.6!)**
  Anaconda is a freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment (Source: Wikipedia).
  Link: https://repo.continuum.io/miniconda/

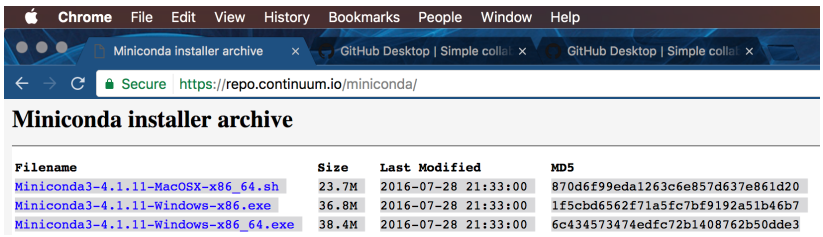


Figure 1: Miniconda3 packages for Windows and Mac OS

Note: Download Python v3.0-3.5 (not v2.7 & v3.6) because Pyomo command line installation works only with Python v3+. Also, Pyomo is not compatible with the latest Python distribution, v3.6, yet. So, download either miniconda3 version 4.1.11 or anaconda3 version 4.1.1, which have Python v3.5.

- **GitHub**
  GitHub is a web-based Git or version control repository and Internet hosting service. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project (Source: Wikipedia).
  Link: https://desktop.github.com

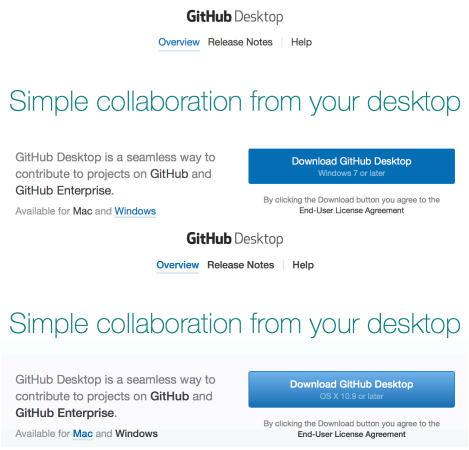


Figure 2: GitHub installation for Windows and Mac OS

- **Pyomo**
  Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities. The easiest way to install Pyomo is to type following commands to your command line tool and press ENTER.

  Windows:
  Command: `conda install -c cachemeorg pyomo`
  Command: `conda install -c cachemeorg pyomo.extras`
  Command: `conda install -c cachemeorg glpk`

  Mac OS:
  Command: `conda install -c conda-forge pyomo`
  Command: `conda install -c conda-forge pyomo.extras`
  Command: `conda install -c conda-forge glpk`

Note: if you don't know how to use command line, please refer to description below and then type the commands.

1.1 Command line

This shortcourse is not intended to explain command line use. But I will walk you through steps enough to install Pyomo and run PyVIN model. If you like to learn basics of command line, here is a nice crash course:
https://learnpythonthehardway.org/book/appendixa.html

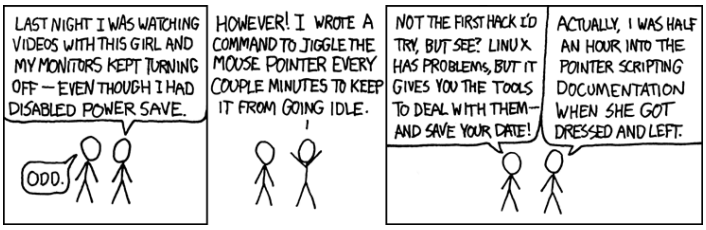


Figure 3: A command line comic from xkcd, source: https://xkcd.com/196/

1.1.1 Windows

After installing GitHub, double click on "Git Shell" and then type the command above to install Pyomo and GLPK solver.

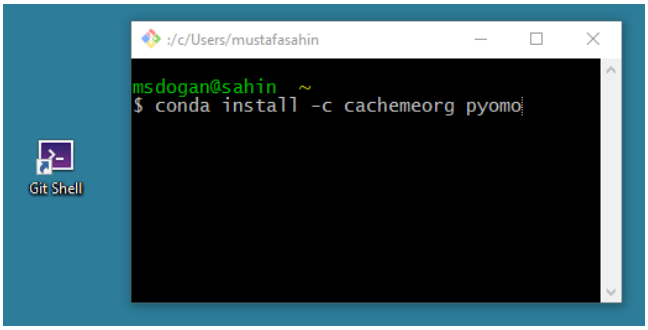


Figure 4: Command line (Git Shell) for Windows

Successful "pyomo" installation on Windows:

```
msdogan@sahin ~
$ conda install -c cachemeorg pyomo
Fetching package metadata .............
Solving package specifications: ..........

Package plan for installation in environment C:\Users\mustafasahin\Miniconda3:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    pyutilib-5.2.3601          |        py35_16         416 KB  cachemeorg
    cryptography-1.7.1         |         py35_0         353 KB
    pyomo-4.2.10784            |        py35_16         1.7 MB  cachemeorg
    pyopenssl-16.2.0           |         py35_0          70 KB
    conda-4.3.16               |         py35_0         553 KB
    ------------------------------------------------------------
                                           Total:         3.1 MB

The following NEW packages will be INSTALLED:

    cffi:          1.9.1-py35_0
    cryptography:  1.7.1-py35_0
    idna:          2.2-py35_0
    nose:          1.3.7-py35_1
    openssl:       1.0.2k-vc14_0                [vc14]
    ply:           3.10-py35_0
    pyasn1:        0.2.3-py35_0
    pycparser:     2.17-py35_0
    pyomo:         4.2.10784-py35_16  cachemeorg
    pyopenssl:     16.2.0-py35_0
    pyro4:         4.41-py35_16       cachemeorg
    pyutilib:      5.2.3601-py35_16   cachemeorg
    serpent:       1.12-py35_16       cachemeorg
    six:           1.10.0-py35_0

The following packages will be UPDATED:

    conda:         4.1.11-py35_0                --> 4.3.16-py35_0
    conda-env:     2.5.2-py35_0                 --> 2.6.0-0
    requests:      2.10.0-py35_0                --> 2.13.0-py35_0

Proceed ([y]/n)? y

Fetching packages ...
pyutilib-5.2.3 100% |############################| Time: 0:00:04 101.74 kB/s
cryptography-1 100% |############################| Time: 0:00:01 186.03 kB/s
pyomo-4.2.1078 100% |############################| Time: 0:00:07 239.44 kB/s
pyopenssl-16.2 100% |############################| Time: 0:00:00 197.60 kB/s
conda-4.3.16-p 100% |############################| Time: 0:00:02 215.64 kB/s
Extracting packages ...
[      COMPLETE      ]|##############################################| 100%
Unlinking packages ...
[      COMPLETE      ]|##############################################| 100%
```

```
Linking packages ...
[      COMPLETE      ]|#################################################| 100%
```

Successful "pyomo.extras" installation on Windows:

```
msdogan@sahin ~
$ conda install -c cachemeorg pyomo.extras
Fetching package metadata .............
Solving package specifications: .

Package plan for installation in environment C:\Users\mustafasahin\
    Miniconda3:

The following NEW packages will be INSTALLED:

    derapproximator: 0.52-py35_16       cachemeorg
    funcdesigner:    0.5620-py35_16     cachemeorg
    mkl:             2017.0.1-0
    numpy:           1.12.1-py35_0
    openopt:         0.5625-py35_16     cachemeorg
    pyomo.extras:    2.0-py35_16        cachemeorg
    scipy:           0.19.0-np112py35_0
    setproctitle:    1.1.9-py35_5       cachemeorg
    suds-jurko:      0.6-py35_5         cachemeorg

Proceed ([y]/n)? y

mkl-2017.0.1-0 100% |############################| Time: 0:15:39 143.79 kB/s
numpy-1.12.1-p 100% |############################| Time: 0:00:16 221.51 kB/s
setproctitle-1 100% |############################| Time: 0:00:00 299.06 kB/s
suds-jurko-0.6 100% |############################| Time: 0:00:03 248.07 kB/s
derapproximato 100% |############################| Time: 0:00:00 234.74 kB/s
openopt-0.5625 100% |############################| Time: 0:00:01 331.42 kB/s
scipy-0.19.0-n 100% |############################| Time: 0:01:06 202.81 kB/s
funcdesigner-0 100% |############################| Time: 0:00:02  81.36 kB/s
pyomo.extras-2 100% |############################| Time: 0:00:00 193.19 kB/s
```

Successful "GLPK" solver installation on Windows:

```
msdogan@sahin ~
$ conda install -c cachemeorg glpk
Fetching package metadata .............
Solving package specifications: .

Package plan for installation in environment C:\Users\mustafasahin\
    Miniconda3:

The following NEW packages will be INSTALLED:

    glpk: 4.57-vc14_5 cachemeorg [vc14]

Proceed ([y]/n)? y

glpk-4.57-vc14 100% |############################| Time: 0:00:08 283.73 kB/s
```

1.1.2 Mac OS

Search "terminal" in Spotlight Search and then double click. Type the command above to install Pyomo and GLPK solver.

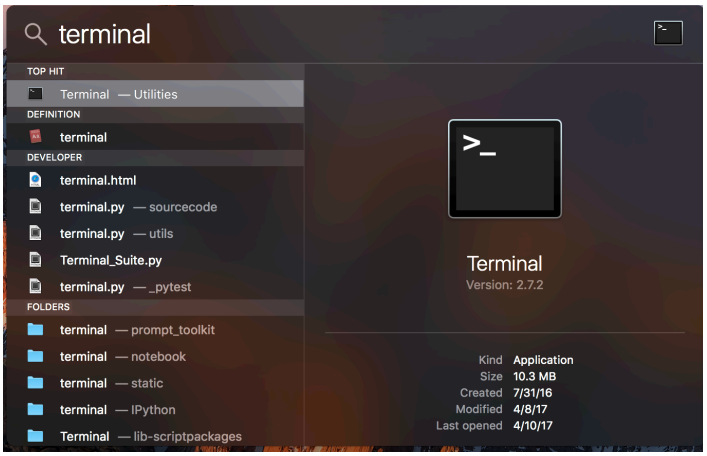


Figure 5: Command line (terminal) for Mac OS

Successful "pyomo" installation on Mac OS:

```
Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge pyomo
Fetching package metadata .............
Solving package specifications: .

Package plan for installation in environment /Users/msdogan/anaconda:

The following NEW packages will be INSTALLED:

    appdirs:    1.4.3-py27_0  conda-forge
    pyomo:      5.1.1-py27_0  conda-forge
    pyutilib:   5.4.1-py27_0  conda-forge

The following packages will be SUPERSEDED by a higher-priority channel:

    conda:      4.3.16-py27_0             --> 4.2.13-py27_0 conda-forge
    conda-env:  2.6.0-0                   --> 2.6.0-0       conda-forge

Proceed ([y]/n)? y

conda-env-2.6. 100% |############################| Time:  0:00:00    1.11 MB/s
appdirs-1.4.3- 100% |############################| Time:  0:00:00   50.42 kB/s
conda-4.2.13-p 100% |############################| Time:  0:00:02  155.17 kB/s
pyutilib-5.4.1 100% |############################| Time:  0:00:01  217.86 kB/s
pyomo-5.1.1-py 100% |############################| Time:  0:00:07  294.78 kB/s
```

Successful "pyomo.extras" installation on Mac OS:

```
Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge pyomo.extras
Fetching package metadata ..........
Solving package specifications: .........

Package plan for installation in environment /Users/msdogan/anaconda:

The following packages will be downloaded:

```

```
    package                    |            build
    ---------------------------|-----------------
    pymysql-0.7.9              |           py27_0         117 KB
    pyodbc-3.0.10              |           py27_1         146 KB
    serpent-1.12               |           py27_0          14 KB  conda-
    forge
    pyro4-4.43                 |           py27_2         109 KB  conda-
    forge
    pyomo.extras-3.2           |           py27_0           1 KB  conda-
    forge
    ------------------------------------------------------------
                                           Total:         387 KB

The following NEW packages will be INSTALLED:

    pymysql:       0.7.9-py27_0
    pyodbc:        3.0.10-py27_1
    pyomo.extras: 3.2-py27_0    conda-forge
    pyro4:         4.43-py27_2  conda-forge
    serpent:       1.12-py27_0  conda-forge

Proceed ([y]/n)? y

Fetching packages ...
pymysql-0.7.9- 100% |############################| Time: 0:00:00 312.08 kB/s
pyodbc-3.0.10- 100% |############################| Time: 0:00:00 288.39 kB/s
serpent-1.12-p 100% |############################| Time: 0:00:00 116.83 kB/s
pyro4-4.43-py2 100% |############################| Time: 0:00:00 183.68 kB/s
pyomo.extras-3 100% |############################| Time: 0:00:00   1.33 MB/s
Extracting packages ...
[      COMPLETE      ]|##################################################| 100%
Linking packages ...
[      COMPLETE      ]|##################################################| 100%
```

Successful "GLPK" solver installation on Mac OS:

```
Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge glpk
Fetching package metadata ...........
Solving package specifications: ..........

Package plan for installation in environment /Users/msdogan/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    gmp-6.1.2                  |                0         716 KB  conda-
    forge
    glpk-4.61                  |                0         1.0 MB  conda-
    forge
    ------------------------------------------------------------
                                           Total:         1.7 MB

The following NEW packages will be INSTALLED:

    glpk: 4.61-0  conda-forge
    gmp:  6.1.2-0 conda-forge

Proceed ([y]/n)? y
```

```
22
23 Fetching packages ...
24 gmp-6.1.2-0.ta 100% |############################| Time: 0:00:02 248.51 kB/s
25 glpk-4.61-0.ta 100% |############################| Time: 0:00:03 290.45 kB/s
26 Extracting packages ...
27 [       COMPLETE       ]|#########################################| 100%
28 Linking packages ...
29 [       COMPLETE       ]|#########################################| 100%
30 Mustafas-MacBook-Pro:~ msdogan$
```

2 Introduction and background

Developed in early 2000s, **CAL**ifornia **V**alue **IN**tegrated 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 an 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 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. Figure 6 regions and coverage of the model.

CALVIN forces quantitative understanding of integrated water and economic system. 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

Figure 7 shows CALVIN's schematic and the water network of California. You can zoom in to see the details if you are using PDF version. In CALVIN, there are two water sources avaiable for agricultural users: groundwater and surface water (Figure 8-a). 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.

Urban areas have more water supply sources available than agricultural users. In addition to groundwater and surface water, desalination, potable and nonpotable recycled wastewater are available for urban users (Figure 8-b). Surface water deliveries are treated in a water treatment plant nodes (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.

Wildlife refuge areas do not have an economic representation in CALVIN (Figure 8-c). 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.

With the recent updates, CALVIN is evolved to open-source PyVIN model, optimizing water resources in a short amount of time with state-of-the-art solvers and modeling platform. PyVIN has the same input data and objective as CALVIN but it is modeled on Pyomo, a Python-based high level algebraic modeling language. PyVIN also uses HOBBES database, which allows better documentation, collaboration and communication between models as well as modelers.

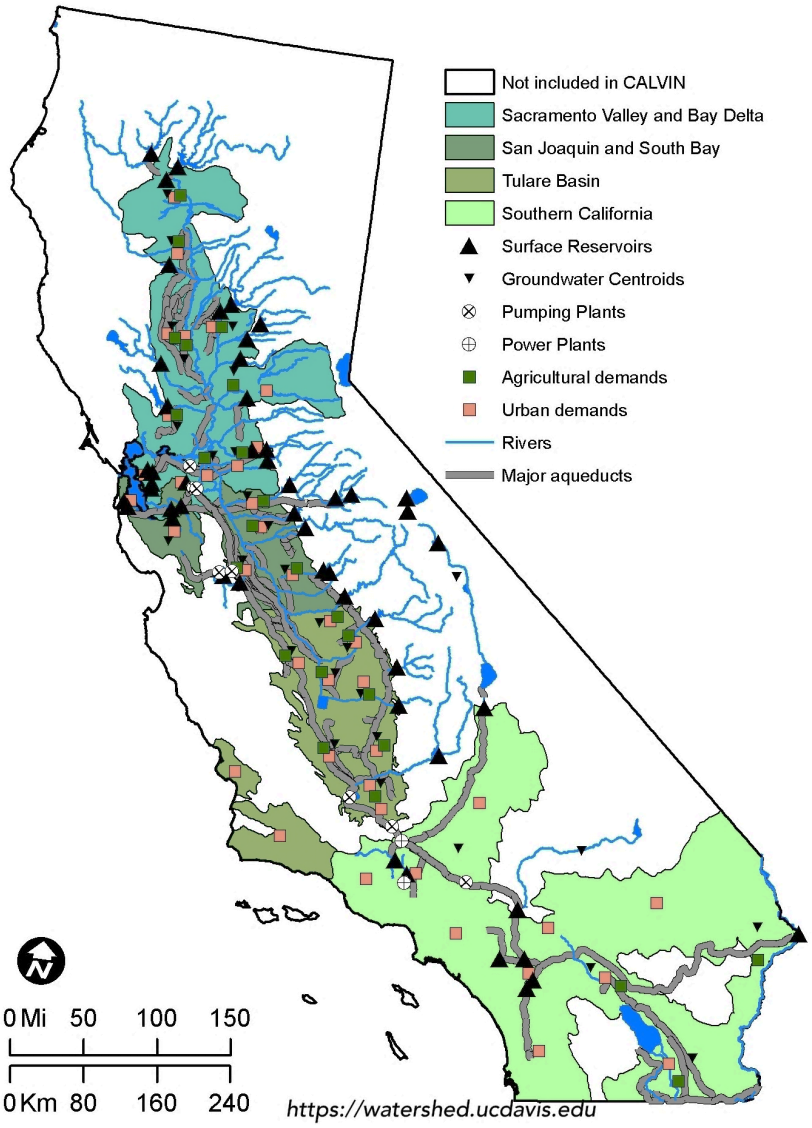


Figure 6: California's water infrastructure and CALVIN's coverage

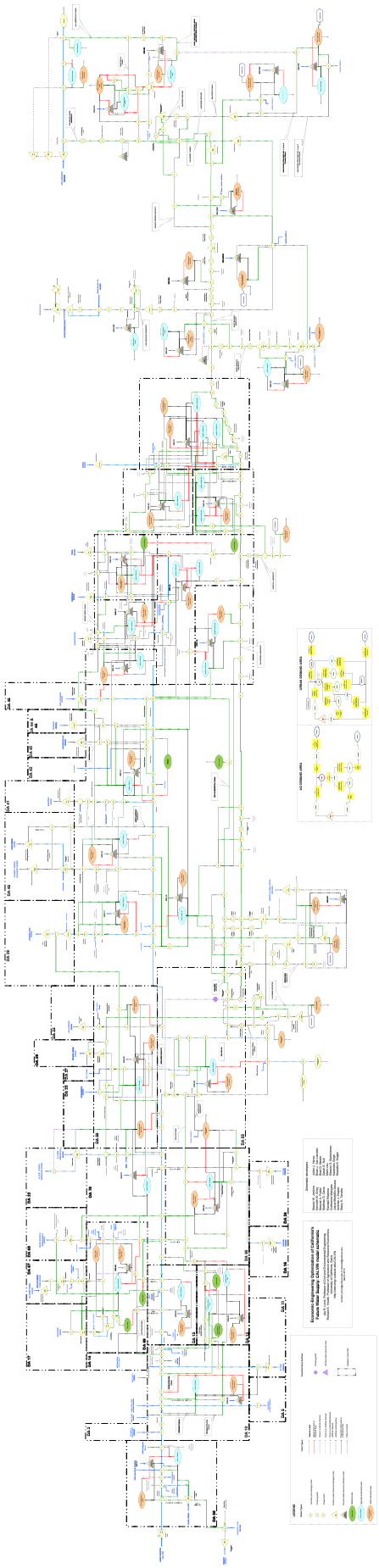
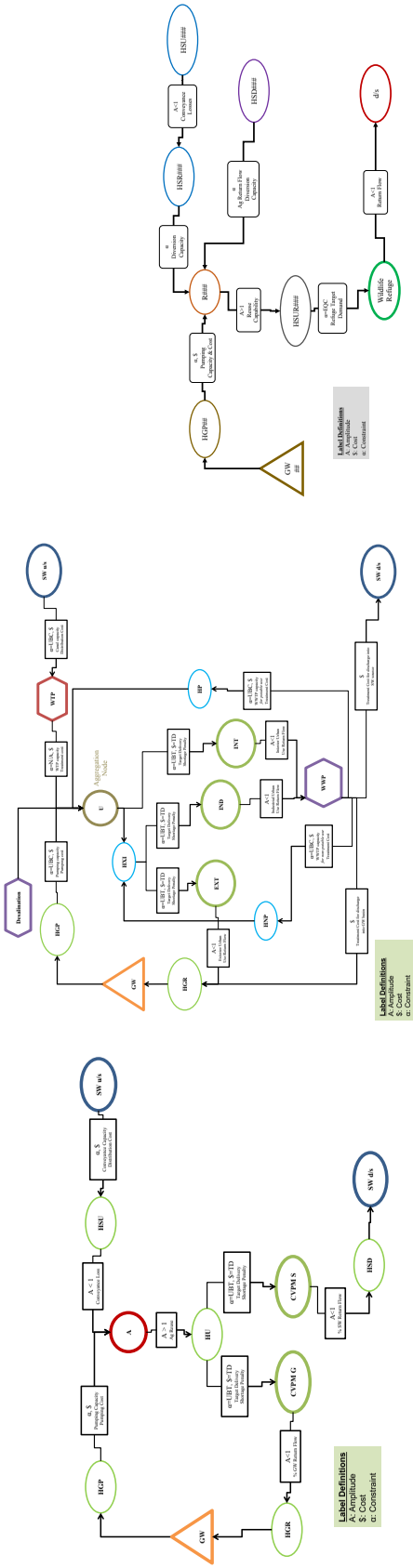


Figure 7: CALVIN Network Schematic



(a) Agricultural demand representation (b) Urban demand representation (c) Wildlife refuge demand representation

Figure 8: CALVIN's demand area schematics

3 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 a animation tool to display data.

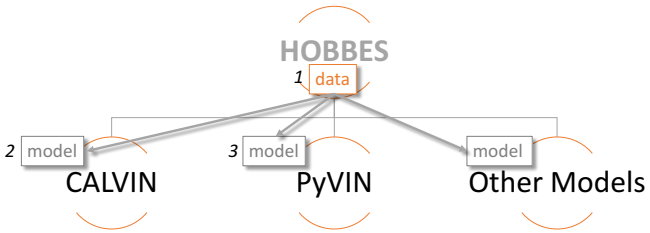


Figure 9: HOBBES database and model integration

3.1 Data flow overview

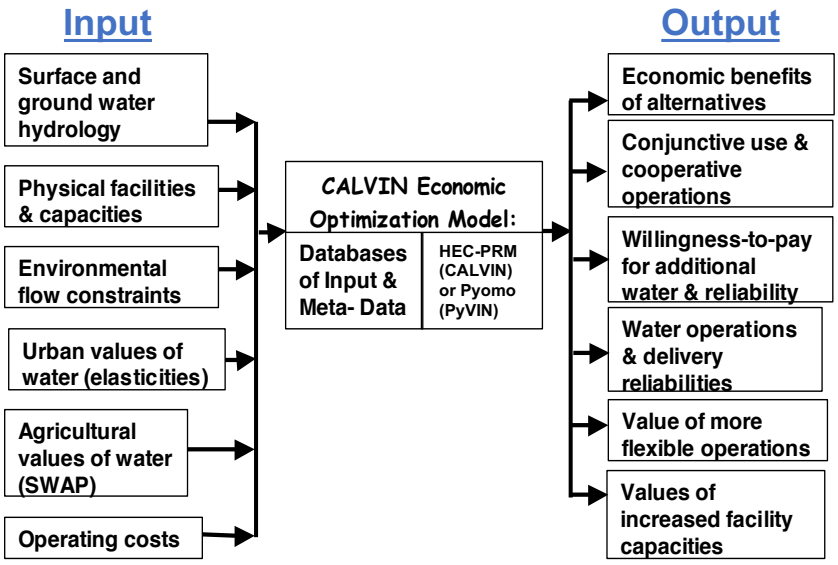


Figure 10: CALVIN (and PyVIN) data overflow with input (left column) and output (right column)

4 Updated version: PyVIN

PyVIN combines CALVIN's knowledge and extensive water infrastructure and hydrology data with a high level algebraic modeling language, Pyomo, and state-of-the-art solvers, such as CPLEX, Gurobi, CBC, and GLPK. Pyomo is an open-source Python based large-scale modeling environment. Its model representation is similar to GAMS and AMPL and it solves the problem with user defined solvers. So, PyVIN is not solver specific and users can choose and install any solver as long as it is compatible with Pyomo.

CALVIN	PyVIN
Large-scale hydroeconomic model	
Optimize water allocation to agricultural and urban users	
Minimize statewide water scarcity and operating costs	
HEC-PRM and VBA based	Pyomo and Python based
Less flexible (limited to HEC-PRM)	More flexible (full LP)
Solver runtime: ~16 hr (depending on initial solution)	Solver runtime: ~1 min (depending on solver)
Requires 32 bit Windows PC	Any computer
HEC-DSS database	Open source: data and source code

Figure 11: CALVIN (and PyVIN) data overflow with input (left column) and output (right column)

4.1 Model architecture

Modeled in different environments, both CALVIN and PyVIN are linear programming network-flow models and solve the same objective function, subject to physical and environmental constraints. The objective is to minimize statewide water scarcity and operating costs (Equation 1). There are three constraints: upper bound (Equation 2), lower bound (Equation 3), and mass balance (Equation 4).

$$max(z) = \sum_i \sum_j \sum_k c_{ijk} X_{ijk} \tag{1}$$

$$X_{ijk} \leq u_{ijk} \tag{2}$$

$$X_{ijk} \geq l_{ijk} \tag{3}$$

$$\sum_j \sum_i X_{ijk} = \sum_i \sum_j a_{ijk} X_{ijk} + b_j \tag{4}$$

where c represents unit cost, X is flow, u is upper bound, l is lower bound, a is amplitude to represent losses, and b is a local inflow to node j.

4.2 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, $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.

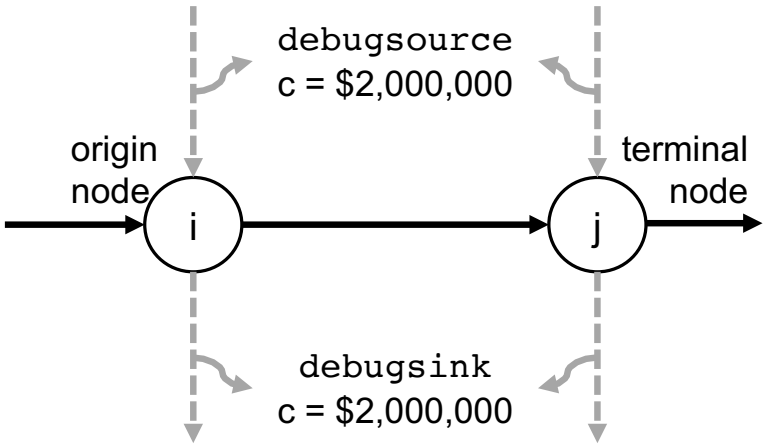


Figure 12: Debug mode links to the system network

4.3 PyVIN model

PyVIN is modeled as an abstract model in Pyomo, separating model structure from data. PyVIN can solve any data size and data do not have to be water resources. As long as it is a network flow problem, such as transportation or transmission, PyVIN can solve it. Also, since it is open-source and has full linear programming features, it has more flexible model representation. Any other constraints can easily be added in addition to three constraint mentioned before. The Pyomo structure code with parameters and equations is shown below.

Command to run Pyomo:

```
pyomo solve --solver=glpk --solver-suffix=dual pyvin.py data.dat --stream
    -solver --report-timing --json
```

structure file (`pyvin.py`)

```
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 (`data.dat`)

The data file `data.dat` includes list of nodes, and list of links 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 ...;

set k := 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15;

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 -7.003  0.997 630.4 630.4
SR_SHA.1983-10-31 SR_SHA.1983-11-30 1 -2.974  0.997 737.4 737.4
SR_SHA.1983-10-31 SR_SHA.1983-11-30 2 -1.466  0.997 632.2 2032.2
SR_SHA.1983-11-30 SR_SHA.1983-12-31 0 -6.972  0.999 609.4 609.4
SR_SHA.1983-11-30 SR_SHA.1983-12-31 1 -3.056  0.999 700.8 700.8
SR_SHA.1983-11-30 SR_SHA.1983-12-31 2 -1.479  0.999 689.7 1941.7
FINAL SINK  0 0 1 0 10000000
SR_SHA.1984-09-30 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
```

5 Your first model run

When you type the run command, the command line output (for the one-year run) should look like, showing run status and objective value, which can also ben found in `results.json`:

```
campus-109-001:Run_Folder msdogan$ pyomo solve --solver=glpk --solver-
    suffix=dual pyvin.py data.dat --json
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
[    0.00] Creating model
[   17.23] Applying solver
[   48.14] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: -495639639.765
    Solver results file: results.json
[   52.84] Applying Pyomo postprocessing actions
[   52.84] Pyomo Finished
```

5.1 Postprocessing results

Pyomo 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. PyVIN's Python-based postprocessor scripts create those separate files and save them as `*.csv` which can be easily shared and used to create figures.

Here is my order to run postprocessing scripts:

- Once you obtain `results.json` file from Pyomo, run `postprocess.py`. This will generate following files:
  - `flow.csv` : monthly flows (TAF\month) on links from October 1921 to September 2003
  - `storage.csv` : monthly surface (TAF) and groundwater storage time-series
  - `evaporation.csv` : evaporation time-series (TAF\month)
  - `shortage_volume.csv` : agricultural and urban water shortage volumes (TAF\month)
  - `shortage_cost.csv` : agricultural and urban water shortage costs ($K\month)
  - `dual_lower.csv` : dual values (Lagrange multipliers) on lower bound constraints ($\AF), showing unit cost of increasing the constraint
  - `dual_upper.csv` : dual values (Lagrange multipliers) on upper bound constraints ($\AF), showing unit cost of increasing the constraint
  - `dual_node.csv` : dual values (Lagrange multipliers) at nodes ($\AF), showing unit cost of injecting additional water (related to mass balance constraint)
- If you have run in "debug mode", then run `debug_flow_finder.py` to see if there is any debug flow.
- Run `aggregate_regions.py` to create regional average results.
- Run `supply_portfolio_hatchedbarplot.py` to plot urban and agricultural water supply portfolios.
- You can always create your own personalized postprocessing scripts based on your needs.

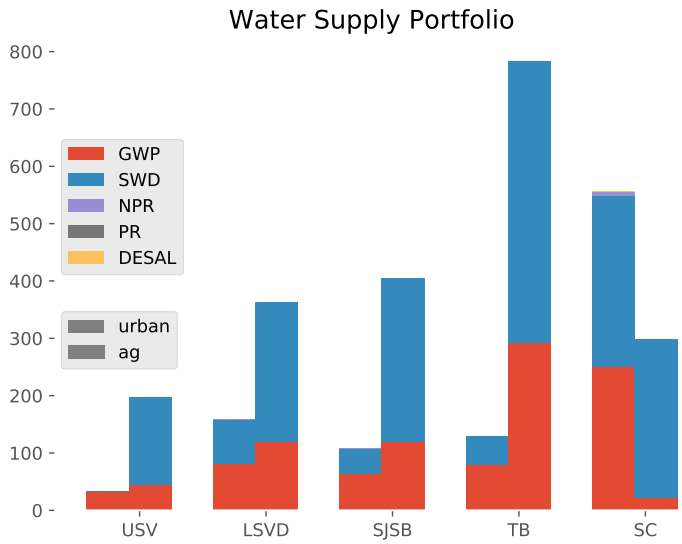


Figure 13: Regional monthly (TAF\month) agricultural and urban water supply portfolios (Upper Sacramento Valley, Lower Sacramento Valley and Delta, San Joaquin and South Bay, Tulare Basin, Southern California)

6 References and useful links

Official CALVIN article that describes the model:

Draper, A. J., Jenkins, M. W., Kirby, K. W., Lund, J. R., & Howitt, R. E. (2003). Economicengineering optimization for California water management. *Journal of water resources planning and management*, 129(3), 155-164.

GitHub repositories:
PyVIN: https://github.com/msdogan/pyvin
CALVIN Data: https://github.com/ucd-cws/calvin-network-data
HOBBES tools: https://github.com/ucd-cws/calvin-network-tools

Web pages:
CALVIN: https://calvin.ucdavis.edu/node
HOBBES: https://hobbes.ucdavis.edu