

CALVIN & PyVIN Shortcourse Handout

Mustafa S. Dogan^{*};

Date: April 28, 2017

Location: The Barn Conference Room (<https://goo.gl/maps/1ZKfUKZQpPw>)

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	5
1.1.1 Windows	5
1.1.2 Mac OS	8
2 Introduction and background	11
3 HOBES database	14
3.1 Data flow overview	14
4 Updated version: PyVIN	15
4.1 Model architecture	15
4.2 Debug mode	16
4.3 PyVIN model	16
5 Your first model run	18
5.1 Historical case one-year	19
5.2 No short-term overdraft on GW_13 subbasin	19
5.3 Postprocessing results	20
5.4 Postprocess your run results	21
6 References and useful links	22

^{*}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 | - HOBES 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 - 3:00 pm | - Your first PyVIN run |
| 3:00 - 4:00 pm | - Postprocessing results |
-

1 Prerequisites

Please bring your **laptop** and try to install below software dependencies in advance (if you can) if you want a hands-on PyVIN experience. Followings are official links to required software but you can also download them from https://github.com/msdogn/CALVIN-PyVIN_shortcourse/tree/master/Software.

If you have any question, e-mail msdogn@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/>

Filename	Size	Last Modified	MD5
Miniconda3-4.1.11-MacOSX-x86_64.sh	23.7M	2016-07-28 21:33:00	870d6f99eda1263c6e857d637e861d20
Miniconda3-4.1.11-Windows-x86.exe	36.8M	2016-07-28 21:33:00	1f5cbd6562f71a5fc7bf9192a51b46b7
Miniconda3-4.1.11-Windows-x86_64.exe	38.4M	2016-07-28 21:33:00	6c34573474edfc72b1408762b50dde3

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.

- **Mac OS miniconda3 installation (*.sh file)**

run the following command and follow instructions prompted

```
1 Mustafas-MacBook-Pro :~ msdogn$ bash Miniconda3-4.1.11-MacOSX-x86_64.sh
```

- **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>

- **GitHub (optional)**

(See instructions below for downloading course materials without installing 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>.

After installing GitHub desktop, go to https://github.com/msdogan/CALVIN-PyVIN_shortcourse and click green box "Clone or download" and "Open in Desktop" and "clone" the shortcourse repository (Figure 2).

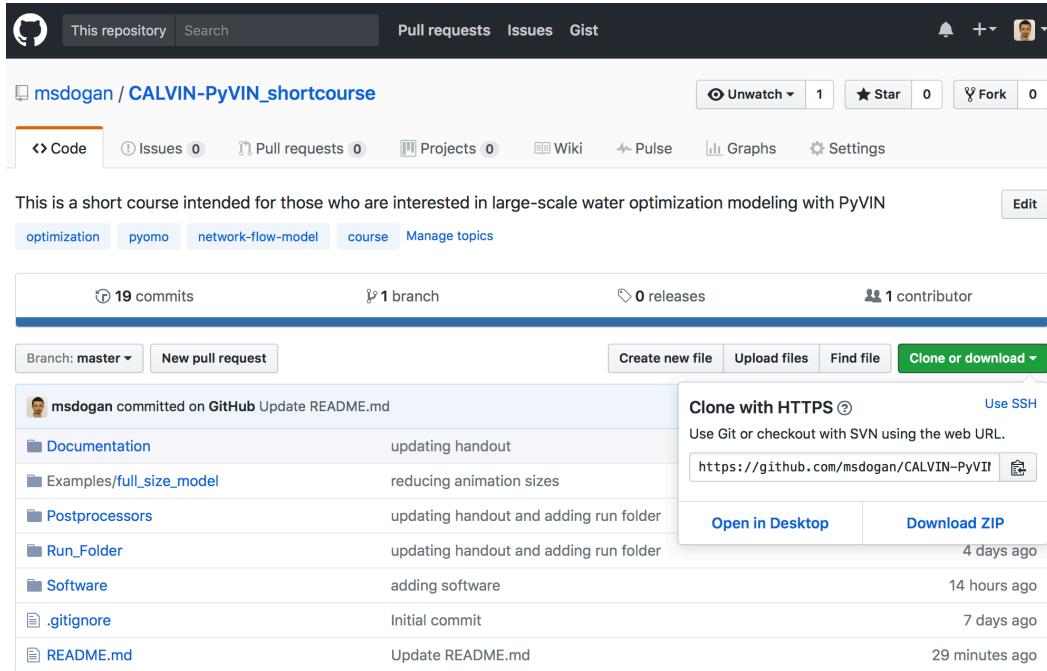


Figure 2: Cloning the shortcourse repository

Downloading shortcourse files without installing GitHub:

If you do not want to install GitHub on your computer, you can download course materials from https://github.com/msdogan/CALVIN-PyVIN_shortcourse.git. Click on green button "Clone or download" and "Download ZIP" (Figure 2). Extract the zip file anywhere you want in your computer.

- **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:

Note: if you are using miniconda3, use `-c cachemeorg` as channel

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>

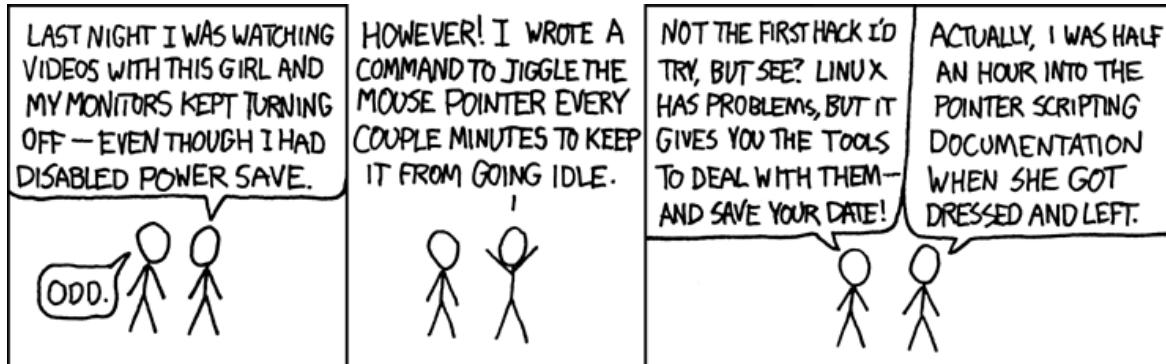


Figure 3: A command line comic from xkcd, source: <https://xkcd.com/196/>

1.1.1 Windows

After installing GitHub, double click on "Git Shell" and then type the command above to install Pyomo and GLPK solver. If you didn't install GitHub, you search for "Command Line" in your computer and type commands there.

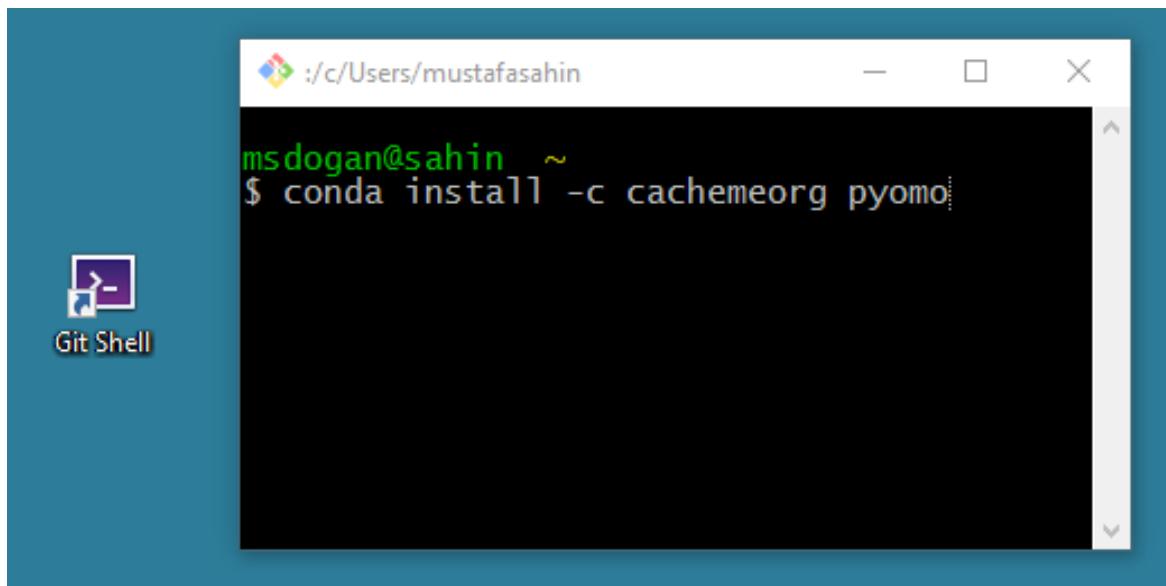


Figure 4: Command line (Git Shell) for Windows

Successful "pyomo" installation on Windows:

```

1 msdogan@sahin ~
2 $ conda install -c cachemeorg pyomo
3 Fetching package metadata .....
4 Solving package specifications: .....
5
6 Package plan for installation in environment C:\Users\mustafasahin\
    Miniconda3:
7
8 The following packages will be downloaded:
9
10
11      package          | build
12      pyutilib-5.2.3601 | py35_16        416 KB
13      cachemeorg
14      cryptography-1.7.1 | py35_0         353 KB
15      pyomo-4.2.10784   | py35_16        1.7 MB
16      cachemeorg
17      pyopenssl-16.2.0  | py35_0         70 KB
18      conda-4.3.16     | py35_0         553 KB
19
20                                     Total:       3.1 MB
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

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:

```

22      cffi:           1.9.1-py35_0
23      cryptography: 1.7.1-py35_0
24      idna:           2.2-py35_0
25      nose:           1.3.7-py35_1
26      openssl:        1.0.2k-vc14_0 [ vc14 ]
27      ply:             3.10-py35_0
28      pyasn1:          0.2.3-py35_0
29      pycparser:       2.17-py35_0
30      pyomo:           4.2.10784-py35_16 cachemeorg
31      pyopenssl:       16.2.0-py35_0
32      pyro4:            4.41-py35_16   cachemeorg
33      pyutilib:         5.2.3601-py35_16 cachemeorg
34      serpent:          1.12-py35_16   cachemeorg
35      six:              1.10.0-py35_0
36
37
38
39      conda:           4.1.11-py35_0      --> 4.3.16-py35_0
40      conda-env:        2.5.2-py35_0      --> 2.6.0-0
41      requests:         2.10.0-py35_0      --> 2.13.0-py35_0
42
43 Proceed ([y]/n)? y
44
45 Fetching packages ...
46 pyutilib-5.2.3 100% ##### Time: 0:00:04 101.74 kB/s
47 cryptography-1 100% ##### Time: 0:00:01 186.03 kB/s
48 pyomo-4.2.1078 100% ##### Time: 0:00:07 239.44 kB/s
49 pyopenssl-16.2 100% ##### Time: 0:00:00 197.60 kB/s
50 conda-4.3.16-p 100% ##### Time: 0:00:02 215.64 kB/s
51 Extracting packages ...
52 [      COMPLETE      ]##### 100%
53 Unlinking packages ...
54 [      COMPLETE      ]##### 100%

```

```
55 Linking packages ...
56 [      COMPLETE      ]||||||||||||||||||||||||||||| 100%
```

Successful "pyomo.extras" installation on Windows:

```
1 msdogan@sahin ~
2 $ conda install -c cachemeorg pyomo.extras
3 Fetching package metadata .....
4 Solving package specifications: .
5
6 Package plan for installation in environment C:\Users\mustafasahin\
   Miniconda3:
7
8 The following NEW packages will be INSTALLED:
9
10    derapproximator: 0.52-py35_16          cachemeorg
11    funcdesigner:    0.5620-py35_16          cachemeorg
12    mkl:           2017.0.1-0
13    numpy:          1.12.1-py35_0
14    openopt:        0.5625-py35_16          cachemeorg
15    pyomo.extras:   2.0-py35_16            cachemeorg
16    scipy:          0.19.0-np112py35_0
17    setproctitle:   1.1.9-py35_5           cachemeorg
18    suds-jurko:    0.6-py35_5            cachemeorg
19
20 Proceed ([y]/n)? y
21
22 mkl-2017.0.1-0 100% ||||||||||||||||||||||| Time: 0:15:39 143.79 kB/s
23 numpy-1.12.1-p 100% ||||||||||||||||||||||| Time: 0:00:16 221.51 kB/s
24 setproctitle-1 100% ||||||||||||||||||||||| Time: 0:00:00 299.06 kB/s
25 suds-jurko-0.6 100% ||||||||||||||||||||||| Time: 0:00:03 248.07 kB/s
26 derapproximat 100% ||||||||||||||||||||||| Time: 0:00:00 234.74 kB/s
27 openopt-0.5625 100% ||||||||||||||||||||||| Time: 0:00:01 331.42 kB/s
28 scipy-0.19.0-n 100% ||||||||||||||||||||||| Time: 0:01:06 202.81 kB/s
29 funcdesigner-0 100% ||||||||||||||||||||||| Time: 0:00:02 81.36 kB/s
30 pyomo.extras-2 100% ||||||||||||||||||||||| Time: 0:00:00 193.19 kB/s
```

Successful "GLPK" solver installation on Windows:

```
1 msdogan@sahin ~
2 $ conda install -c cachemeorg glpk
3 Fetching package metadata .....
4 Solving package specifications: .
5
6 Package plan for installation in environment C:\Users\mustafasahin\
   Miniconda3:
7
8 The following NEW packages will be INSTALLED:
9
10   glpk: 4.57-vc14_5 cachemeorg [vc14]
11
12 Proceed ([y]/n)? y
13
14 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.

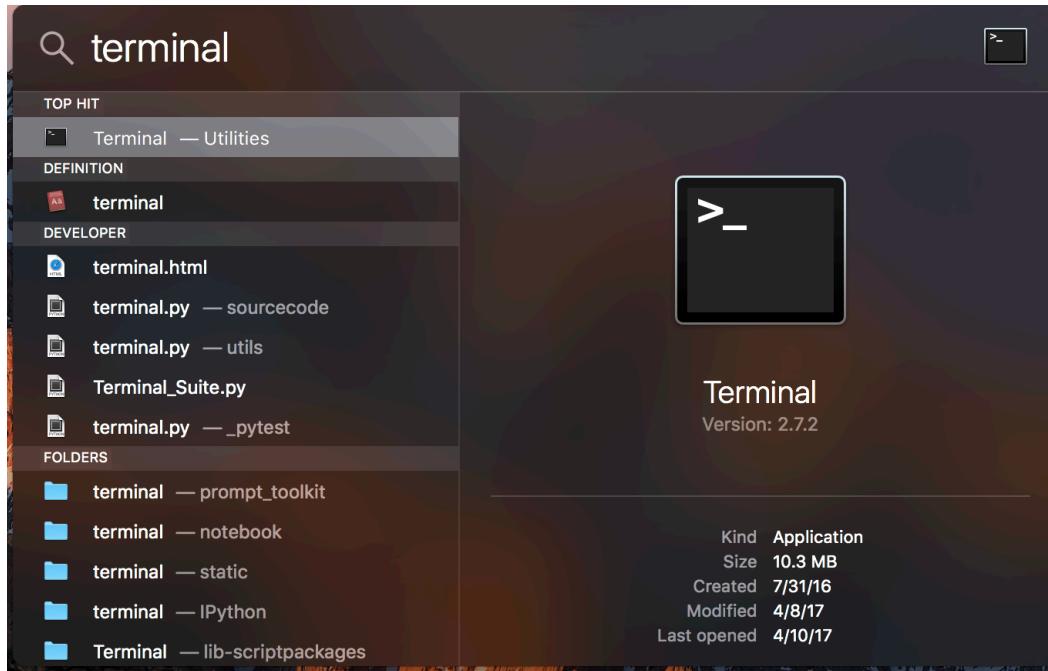


Figure 5: Command line (terminal) for Mac OS

Successful "pyomo" installation on Mac OS:

```

1 Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge pyomo
2 Fetching package metadata .....
3 Solving package specifications: .
4
5 Package plan for installation in environment /Users/msdogan/anaconda:
6
7 The following NEW packages will be INSTALLED:
8
9     appdirs:    1.4.3-py27_0  conda-forge
10    pyomo:      5.1.1-py27_0  conda-forge
11    pyutilib:   5.4.1-py27_0  conda-forge
12
13 The following packages will be SUPERSEDED by a higher-priority channel:
14
15     conda:        4.3.16-py27_0          --> 4.2.13-py27_0  conda-forge
16     conda-env:    2.6.0-0              --> 2.6.0-0          conda-forge
17
18 Proceed ([y]/n)? y
19
20 conda-env-2.6. 100% ##### Time: 0:00:00 1.11 MB/s
21 appdirs-1.4.3- 100% ##### Time: 0:00:00 50.42 kB/s
22 conda-4.2.13-p 100% ##### Time: 0:00:02 155.17 kB/s
23 pyutilib-5.4.1 100% ##### Time: 0:00:01 217.86 kB/s
24 pyomo-5.1.1-py 100% ##### Time: 0:00:07 294.78 kB/s

```

Successful "pyomo.extras" installation on Mac OS:

```

1 Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge pyomo.extras

```

```

2 Fetching package metadata .....
3 Solving package specifications: .....
4
5 Package plan for installation in environment /Users/msdogan/anaconda:
6
7 The following packages will be downloaded:
8
9   package                                | build
10
11   pymysql-0.7.9                         py27_0      117 KB
12   pyodbc-3.0.10                          py27_1      146 KB
13   serpent-1.12                           py27_0      14 KB  conda-
14   pyro4-4.43                            py27_2      109 KB  conda-
15   forge                                 py27_0      1 KB   conda-
16
17                                         Total:      387 KB
18
19 The following NEW packages will be INSTALLED:
20
21   pymysql:      0.7.9-py27_0
22   pyodbc:       3.0.10-py27_1
23   pyomo.extras: 3.2-py27_0    conda-forge
24   pyro4:        4.43-py27_2    conda-forge
25   serpent:      1.12-py27_0    conda-forge
26
27 Proceed ([y]/n)? y
28
29 Fetching packages ...
30 pymysql-0.7.9- 100% ##### Time: 0:00:00 312.08 kB/s
31 pyodbc-3.0.10- 100% ##### Time: 0:00:00 288.39 kB/s
32 serpent-1.12-p 100% ##### Time: 0:00:00 116.83 kB/s
33 pyro4-4.43-py2 100% ##### Time: 0:00:00 183.68 kB/s
34 pyomo.extras-3 100% ##### Time: 0:00:00 1.33 MB/s
35 Extracting packages ...
36 [      COMPLETE      ]##### 100%
37 Linking packages ...
38 [      COMPLETE      ]##### 100%

```

Successful "GLPK" solver installation on Mac OS:

```

1 Mustafas-MacBook-Pro:~ msdogan$ conda install -c conda-forge glpk
2 Fetching package metadata .....
3 Solving package specifications: .....
4
5 Package plan for installation in environment /Users/msdogan/anaconda:
6
7 The following packages will be downloaded:
8
9   package                                | build
10
11   gmp-6.1.2                             0          716 KB  conda-
12   forge                                  0          1.0 MB  conda-
13   glpk-4.61                             0
14
15                                         Total:      1.7 MB

```

```
15  
16 The following NEW packages will be INSTALLED:  
17  
18     glpk: 4.61-0  conda-forge  
19     gmp:   6.1.2-0  conda-forge  
20  
21 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, **CALifornia Value INtegrated** 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 available 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###-CVPMS and CVPMS links. Agricultural demand areas are divided into two parts based on their return flow to either groundwater (CVPMS) 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.

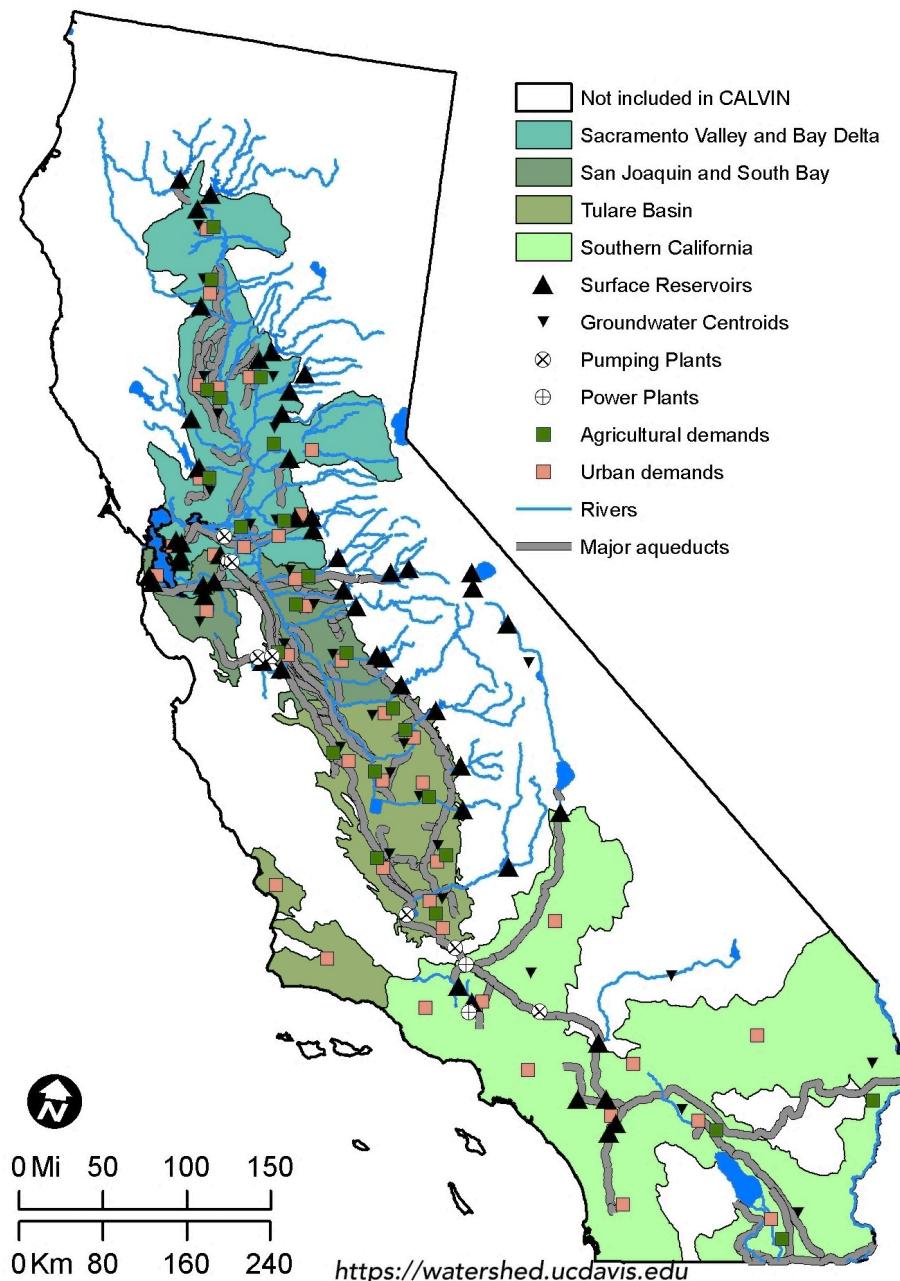


Figure 6: California's water infrastructure and CALVIN's coverage

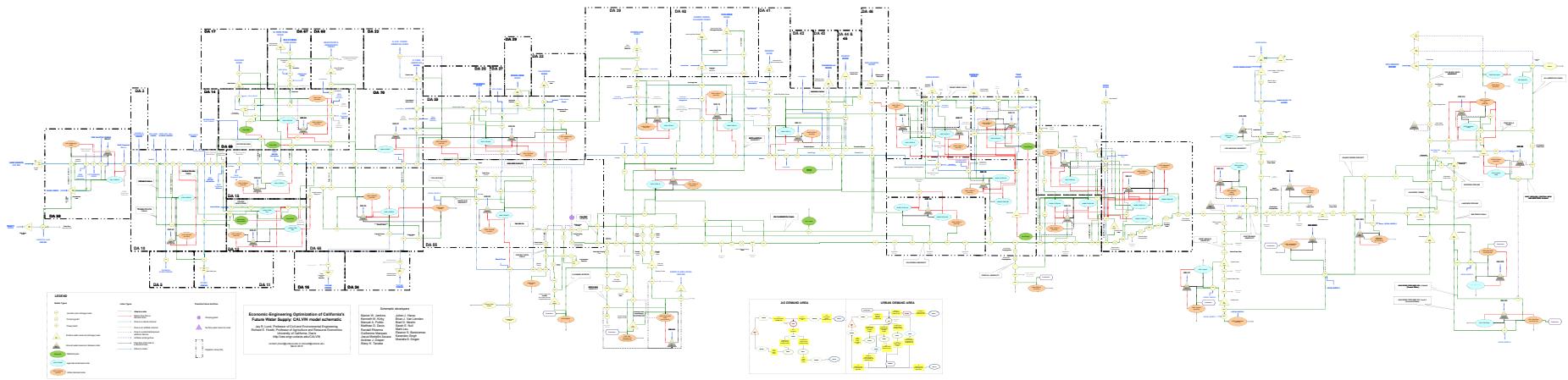


Figure 7: CALVIN Network Schematic

13

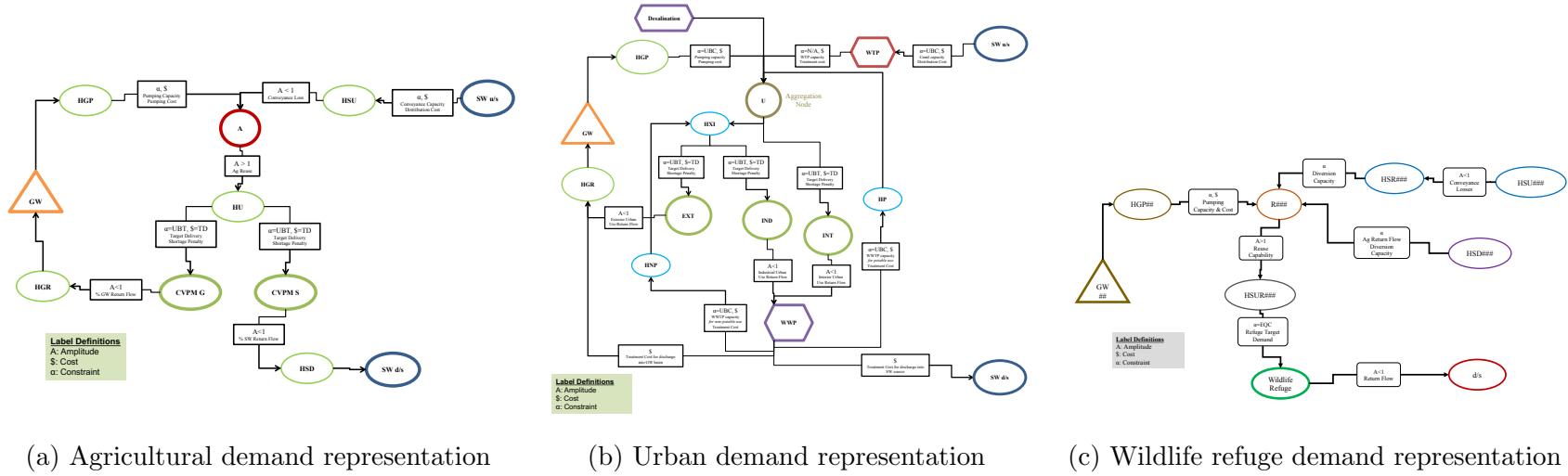


Figure 8: CALVIN's demand area schematics

updated on April 25, 2017

3 HOBES 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.

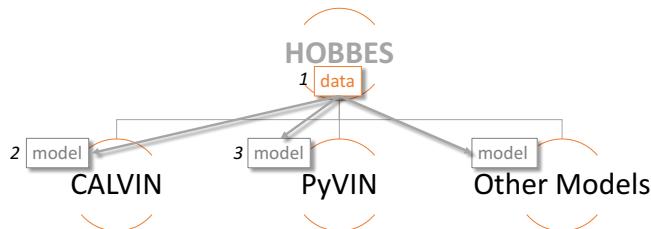


Figure 9: HOBBES database and model integration

3.1 Data flow overview

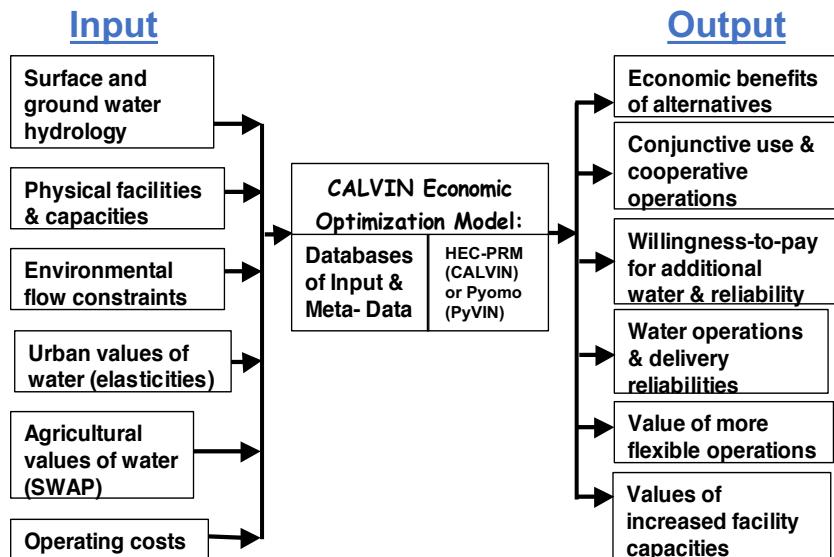


Figure 10: CALVIN (and PyVIN) data overflow with input (left column) and output (right column)

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).

$$\min(z) = \sum_i \sum_j \sum_k c_{ijk} X_{ijk} \quad (1)$$

$$X_{ijk} \leq u_{ijk} \quad (2)$$

$$X_{ijk} \geq l_{ijk} \quad (3)$$

$$\sum_j \sum_i X_{ijk} = \sum_i \sum_j a_{ijk} X_{ijk} + b_j \quad (4)$$

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.

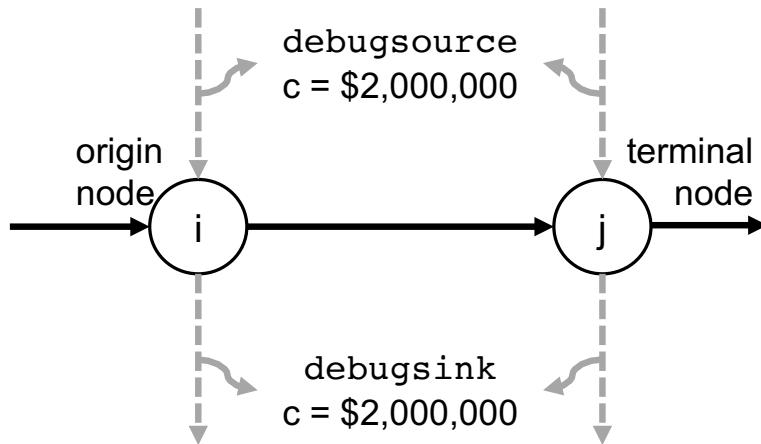


Figure 12: Debug mode links to the system network

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:

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

structure file (pyvin.py)

```

1 from __future__ import division
2 from pyomo.environ import *
3 import itertools
4
5 model = AbstractModel()
6
7 # Nodes in the network
8 model.N = Set()
9
10 # Network arcs
11 model.k = Set()
12
13 model.A = Set(within=model.N*model.N*model.k)
14
15 # Source node
16 model.source = Param(within=model.N)
17 # Sink node
18 model.sink = Param(within=model.N)
19
20 # Flow capacity limits
21 model.u = Param(model.A)
22 # Flow lower bound
23 model.l = Param(model.A)
24 # Link amplitude (gain/loss)
25 model.a = Param(model.A)
26 # Link cost
27 model.c = Param(model.A)
28
29 # The flow over each arc
30 model.X = Var(model.A, within=Reals)
31
32 # Minimize total cost
33 def total_rule(model):
34     return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
35 model.total = Objective(rule=total_rule, sense=minimize)
36
37 # Enforce an upper bound limit on the flow across each arc
38 def limit_rule_upper(model, i, j, k):
39     return model.X[i,j,k] <= model.u[i,j,k]
40 model.limit_upper = Constraint(model.A, rule=limit_rule_upper)
41
42 # Enforce a lower bound limit on the flow across each arc
43 def limit_rule_lower(model, i, j, k):
44     return model.X[i,j,k] >= model.l[i,j,k]
45 model.limit_lower = Constraint(model.A, rule=limit_rule_lower)
46
47 # Enforce flow through each node (mass balance)
48 def flow_rule(model, node):
49     if node in [value(model.source), value(model.sink)]:
50         return Constraint.Skip
51     outflow = sum(model.X[i,j,k]/model.a[i,j,k] for i,j,k in model.A)
52     inflow = sum(model.X[i,j,k] for i,j,k in model.A)
53     return inflow == outflow
54 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.

```

1 set N :=
2 INITIAL SR.SHA.1983-10-31 SR.SHA.1983-11-30      FINAL
3 INFLOW.1983-10-31 INFLOW.1983-11-30 ...;
4
5 set k := 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15;
6
7 param source := SOURCE;
8 param sink := SINK;
9
10 param: A: c a l u :=
11 SOURCE INITIAL 0 0 1 0 10000000
12 INITIAL SR.SHA.1983-10-31 0 0 1 3686.84 3686.84
13 SR.SHA.1983-10-31 SR.SHA.1983-11-30 0 -7.003 0.997 630.4 630.4
14 SR.SHA.1983-10-31 SR.SHA.1983-11-30 1 -2.974 0.997 737.4 737.4
15 SR.SHA.1983-10-31 SR.SHA.1983-11-30 2 -1.466 0.997 632.2 2032.2
16 SR.SHA.1983-11-30 SR.SHA.1983-12-31 0 -6.972 0.999 609.4 609.4
17 SR.SHA.1983-11-30 SR.SHA.1983-12-31 1 -3.056 0.999 700.8 700.8
18 SR.SHA.1983-11-30 SR.SHA.1983-12-31 2 -1.479 0.999 689.7 1941.7
19 FINAL SINK 0 0 1 0 10000000
20 SR.SHA.1984-09-30 FINAL 0 0 1 2923.297 2923.297
21 SOURCE INFLOW.1983-10-31 0 0 1 0 10000000
22 INFLOW.1983-10-31 SR.SHA.1983-10-31 0 0 1 301.765 301.765
23 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 be found in `results.json`:

```

1 campus-109-001:Run_Folder msdogan$ pyomo solve --solver=glpk --solver-
   suffix=dual pyvin.py data.dat --json
2 [    0.00] Setting up Pyomo environment
3 [    0.00] Applying Pyomo preprocessing actions
4 [    0.00] Creating model
5 [  17.23] Applying solver
6 [  48.14] Processing results
7     Number of solutions: 1
8     Solution Information
9         Gap: 0.0
10        Status: feasible
11        Function Value: -495639639.765
12     Solver results file: results.json
13 [  52.84] Applying Pyomo postprocessing actions
14 [  52.84] Pyomo Finished

```

5.1 Historical case one-year

This is a historical case one-year period model run from October 2002 to September 2003, with boundary conditions from full size (82-year) model run. As described in the previous section, you need two files to run PyVIN: `pyvin.py` and `data.dat`.

- Go to run folder `CALVIN-PyVIN_shortcourse/Run_Folder/base_case/` and take a look at `data.dat` file. You can open this file with any text editor.
- Open command line and change working directory to
`~/CALVIN-PyVIN_Shortcourse/Run_Folder/base_case/`
- Run command:

```
1 pyomo solve --solver=glpk --solver-suffix=dual pyvin.py data.dat --
   stream-solver --report-timing --json
2
```

5.2 No short-term overdraft on GW_13 subbasin

This hypothetical no short-term overdraft case limits groundwater pumping from subbasin GW_13 by setting some storage constraints. We will fix the groundwater storage every six months to initial storage value.

- Go to run folder `CALVIN-PyVIN_shortcourse/Run_Folder/no_overdraft/`
- Find groundwater subbasin GW_13 in `data.dat` file (between row 22536 and 22548)
- Change `GW_13.2003-03-31 GW_13.2003-04-30` lower bound l and upper bound u to "128964.721". Making upper and lower bound the same forces model to keep at fixed values.
- Change `GW_13.2003-09-30 FINAL` lower bound l and upper bound u to "128964.721". Your `data.dat` file should look like:

```
1 INITIAL GW_13.2002-10-31 0 0.0 1.0 128964.721 128964.721
2 GW_13.2002-10-31 GW_13.2002-11-30 0 0.0 1.0 0.0 1.00E+12
3 GW_13.2002-11-30 GW_13.2002-12-31 0 0.0 1.0 0.0 1.00E+12
4 GW_13.2002-12-31 GW_13.2003-01-31 0 0.0 1.0 0.0 1.00E+12
5 GW_13.2003-01-31 GW_13.2003-02-28 0 0.0 1.0 0.0 1.00E+12
6 GW_13.2003-02-28 GW_13.2003-03-31 0 0.0 1.0 0.0 1.00E+12
7 GW_13.2003-03-31 GW_13.2003-04-30 0 0.0 1.0 128964.721 128964.721
8 GW_13.2003-04-30 GW_13.2003-05-31 0 0.0 1.0 0.0 1.00E+12
9 GW_13.2003-05-31 GW_13.2003-06-30 0 0.0 1.0 0.0 1.00E+12
10 GW_13.2003-06-30 GW_13.2003-07-31 0 0.0 1.0 0.0 1.00E+12
11 GW_13.2003-07-31 GW_13.2003-08-31 0 0.0 1.0 0.0 1.00E+12
12 GW_13.2003-08-31 GW_13.2003-09-30 0 0.0 1.0 0.0 1.00E+12
13 GW_13.2003-09-30 FINAL 0 0.0 1.0 128964.721 128964.721
```

- Open command line and change working directory to
`~/CALVIN-PyVIN_Shortcourse/Run_Folder/no_overdraft/`
- Run command:

```
1 pyomo solve --solver=glpk --solver-suffix=dual pyvin.py data.dat --
   stream-solver --report-timing --json
2
```

5.3 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.

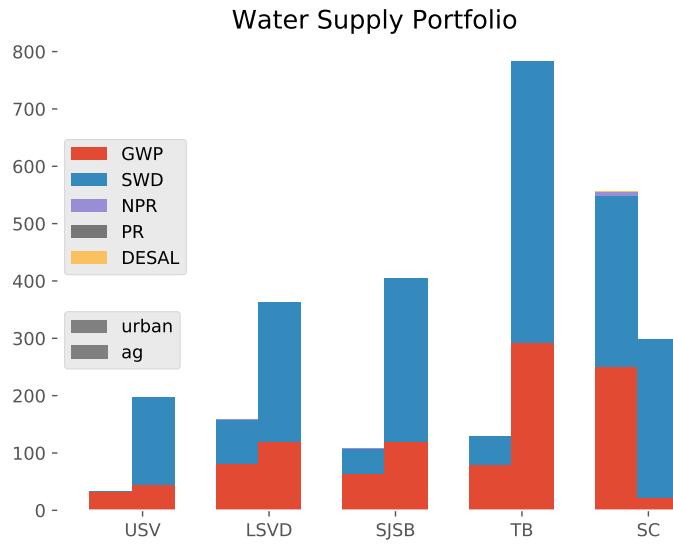


Figure 13: Regional monthly (TAF\month) agricultural and urban water supply portfolios (Upper Sacramento Valley, Lower Sacramento Valley and Delta, San Joaquin and South Bay, Tulare Basin, Southern California)

5.4 Postprocess your run results

Run postprocessors for your historical and no overdraft runs and compare results. Compare, especially, groundwater storage on GW_13 between base and no overdraft cases. Analyze following results:

- Compare objective function value between base and no overdraft cases. Which one is lower? Why?
- Using `storage.csv` files, plot GW_13 one-year flow time-series on the same graph and analyze differences.
- Using `shortage_cost.csv` files, compare agricultural water scarcity costs on links "HU306-CVPM13G" and "HU306-CVPM13S" between two runs. Note: first sum "HU306-CVPM13G" and "HU306-CVPM13S" and then compare.

For following analyses, please use full size model results (82-year) under
`~/CALVIN-PyVIN_Shortcourse/Examples/full_size_model/`

- Using `storage.csv`, calculate monthly average storage and create time-series plots for:
 - Shasta Reservoir (SR_SHA)
 - Lake Oroville (SR_ORO)
 - Folsom Lake (SR_FOL)
- Using `storage.csv`, plot Central Valley groundwater time-series. Central Valley basin is represented with 21 subbasins: GW_01, GW_02, ..., GW_20, GW_21. So, you need to sum them up.

- Using `flow.csv`, calculate monthly and annual average Delta exports, which are water transfers from California Aqueduct (Link: PMP_Banks-D800) and Delta-Mendota Canal (Link: PMP_Tracy-D701) (Figure 14). Visualize your results.

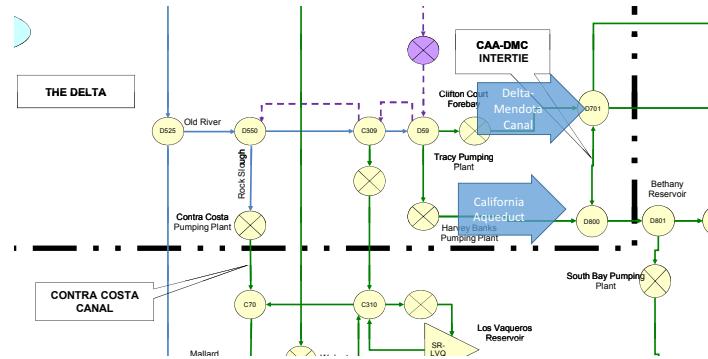


Figure 14: Delta exports from Tracy and Banks pumping plants

- Are you interested in any other analyses?

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>

HOBES tools: <https://github.com/ucd-cws/calvin-network-tools>

Web pages:

CALVIN: <https://calvin.ucdavis.edu/node>

HOBES: <https://hobbes.ucdavis.edu> & <https://cwn.casil.ucdavis.edu>