

DRIP Water Resource Allocation Tool

Lisa Rennels and Nick Depsky

11th May 2018

1 I. Intro

DRIP is a water resource allocation tool, modeled after [WEAP](#) from the [Stockholm Environment Institute's U.S. Center](#). WEAP can be summarized as follows:

*WEAP ("Water Evaluation And Planning" system) is a user-friendly software tool that takes an integrated approach to water resources planning.

Freshwater management challenges are increasingly common. Allocation of limited water resources between agricultural, municipal and environmental uses now requires the full integration of supply, demand, water quality and ecological considerations. The Water Evaluation and Planning system, or WEAP, aims to incorporate these issues into a practical yet robust tool for integrated water resources planning. WEAP is developed by the Stockholm Environment Institute's U.S. Center.*

The source code for DRIP is available on [GitHub](#) and includes several script files as well as input data files to run a small example. In addition to the modeling in Julia, we created a small model in WEAP for testing purposes. The required software is proprietary, so it is not included in Github.

2 II. User Guide and Example

1. Setup In order to use DRIP, you will first want to include the helper functions as follows:

```
using Weave
include("helper_functions.jl")
```

```
| (::#8) (generic function with 1 method)
```

Next, set up some basic date information:

```
const start_year = 1990
const stop_year = 2017
const tstep = 12
const modays = [31, 28, 31, 30 ,31, 30, 31, 31, 30 ,31, 30 , 31];
```

2. Define Demand The `DefineDemand.jl` script allows a user to define specific **demand** nodes of which can be of several categories including agricultural, municipal, industrial, and instream flow requirements (IFRs). Each demand nodes has several metadata requirements including

- months - a list of months (*12 by 1 array of Strings*)
- node_type - type of node (*String*)
- name - name of the node (*String*)
- size - size of the node (*Float64*)
- rate - monthly rate of flow through node (*12 by 1 array of Float64*)
- size_units - units of size variable (*String*)
- demand_units = units of rate variable (*String*)
- priority - demand priority (*Int64*)
- Loc- location (or position) in list of all nodes (*Int64*)

Once all demand nodes are defined, you can create the structures with an `include` statement and convert them to a `DataFrame` structure using the `rdf` function as follows:

```
include("DefineDemand.jl")
demand = ddf(demand_nodes, start_year, stop_year)
head(demand)
```

```
6×5 DataFrame.DataFrame
 Row   Date      Name                Quantity  Units  Loc
 1  1990-01-01  "City: Berkeley"  1.2e9    "MM3"  1
 2  1990-02-01  "City: Berkeley"  1.2e9    "MM3"  1
 3  1990-03-01  "City: Berkeley"  1.2e9    "MM3"  1
 4  1990-04-01  "City: Berkeley"  1.2e9    "MM3"  1
 5  1990-05-01  "City: Berkeley"  1.2e9    "MM3"  1
 6  1990-06-01  "City: Berkeley"  1.2e9    "MM3"  1
```

The *include* call will return a list of dictionaries, one for each node, and the *rdf* call will convert this data structure into a **DataFrame** for use by the main DRIP script.

You can also graphically view the demand time series using `dplot`

```
dplot(demand,1990,1990)
```

```
"no matching spec found for plot.encoding.y, possible causes : no matching
spec found for plot.encoding.y.type, possible causes : 'Quantitative' is no
t one of [ordinal,nominal,quantitative,temporal] in plot.encoding.y.type, '
Quantitative' is not one of [geojson,latitude,longitude] in plot.encoding.y
.type, unexpected param 'field' in plot.encoding.y"
```

3. Define Supply The `DefineSupply.jl` script allows a user to define specific **supply** nodes of which can be of several categories including tributary inflow, catchment inflows, return flows, and groundwater. Each supply nodes has several metadata requirements including

- `filepath` - the file path of the .csv file containing flow information (*String*)
- `name` - name of supply node (*String*)
- `supply_units` - units of rate variable (*String*)
- `Loc` - location (or position) in list of all nodes (*Int64*)

Note that supply information is assumed to be stored in a .csv file in the format of month (Date format) in the first column and Supply (number format) in the second column. Example files can be found in this folder eg. *NYuba_Inflow_Month*.

Once all supply nodes are defined, you can create the structures with an `include` statement and convert them to a `DataFrame` structure using the `rdf` function as follows:

```
include("DefineSupply.jl")
supply = sdf(supply_nodes, start_year, stop_year)
head(supply)
```

```
6×5 DataFrame.DataFrame
 Row   Date      Name      Quantity  Units  Loc
 1  1990-01-01  "Headflow"  11.6035   "CMS"   1
 2  1990-02-01  "Headflow"   9.22421  "CMS"   1
 3  1990-03-01  "Headflow"  22.4477   "CMS"   1
 4  1990-04-01  "Headflow"  30.9198   "CMS"   1
 5  1990-05-01  "Headflow"  22.4766   "CMS"   1
 6  1990-06-01  "Headflow"  17.5219   "CMS"   1
```

The *include* call will return a list of dictionaries, one for each node, and the *sdf* call will convert this data structure into a **DataFrame** for use by the main DRIP script.

You can also graphically view the demand time series using `splot`

```
splot(supply,1990,2017)
```

```
"no matching spec found for plot.encoding.y, possible causes : no matching
spec found for plot.encoding.y.type, possible causes : 'Quantitative' is no
t one of [ordinal,nominal,quantitative,temporal] in plot.encoding.y.type, '
Quantitative' is not one of [geojson,latitude,longitude] in plot.encoding.y
.type, unexpected param 'field' in plot.encoding.y"
```

4. Define Infrastructure The `DefineInfrastructure.jl` script allows a user to define specific **infrastructure** nodes, which as of now are simply reservoirs. Each infrastructure node has several metadata requirements including

- name - name of reservoir node (*String*)
- storage_capacity - capacity of reservoir (*Float64*)
- init_storage - initial storage of reservoir (*Float64*)
- top_of_conservation - (*12 by 1 array of Float64*)
- storage_units - units of storage (*String*)
- Loc - location (or position) in list of all nodes (*Int64*)

Note that supply information is assumed to be stored in a .csv file in the format of month (Date format) in the first column and Supply (number format) in the second column. Example files can be found in this folder eg. *NYuba_Inflow_Month*.

Once all supply nodes are defined, you can create the structures with an `include` statement and convert them to a `DataFrame` structure using the `rdf` function as follows:

```
include("DefineInfrastructure.jl")
reservoirs = rdf(reservoir_nodes, start_year, stop_year)
```

The `include` call will return a list of dictionaries, one for each node, and the `rdf` call will convert this data structure into a `DataFrame` for use by the main DRIP script.

5. Run DRIP Once you have set up the demand, supply, and reservoir nodes, you can run the main DRIP allocation routine! This allocation script will run the allocation algorithm and return `results`, which is a vector of values between 0 and 1 indicating the fraction of demand fulfilled for that specific demand node.

```
include("DRIP_allocation_routine.jl")
results = DRIP_allocation_routine(demand_nodes, supply_nodes, reservoir_nodes,
    (stop_year - start_year + 1))
```

3 III. WEAP Model

4 IV. Future Work

Nick and I are interested in expanding upon this work, and have begun to do so in the `DRIP_allocation_routine_WIP.jl` file. The most significant changes will be to the allocation algorithms, and include an optimisation component that mirrors the one used in WEAP. A small model of the inner portion of this optimisation can be found in `Excel_Optim_Model.xlsx`.