

Point Model Enhancements

Note on Changes to MLP Script and Plugin for Incorporating Dynamic Water

Budget Computation

Prepared By: Asim R P

March 2022

IIT Bombay

Mumbai

Table of Contents

| | |
|--|----|
| 1. Introduction | 2 |
| 1.1. Motivation | 2 |
| 1.2. Background | 2 |
| 1.3. Objectives | 3 |
| 2. About Point Model V1 | 3 |
| 2.1. Use of Point Model V1 for MLP | 3 |
| 2.2. Challenges with V1 | 3 |
| 2.2.1. Water balance computation for all time steps at a point | 3 |
| 2.2.2. Weather data format | 4 |
| 2.2.3. Performance | 4 |
| 2.2.4. Error logging and output identification | 4 |
| 3. Enhancements in Point Model V2 | 5 |
| 3.1. Space-time interchange | 5 |
| 3.2. Simpler hourly weather format | 5 |
| 3.3. Improved usability | 5 |
| 3.4. Improved performance | 7 |
| 3.5. Test automation using a sandbox dataset | 8 |
| 4. Water budget for villages left out in 2020 | 9 |
| 4.1. Obtain the latest source code (step 1) | 9 |
| 4.2. Obtain input data (step 2) | 9 |
| 4.3. Running the Python application (step 3) | 10 |
| 4.4. Examine the generated logs (step 4) | 10 |
| 4.5. Inspect output artefacts (step 5) | 12 |
| 4.5.1. Pointwise water budget | 12 |
| 4.5.2. Zone-wise aggregate output | 12 |
| 4.5.3. SQL query for aggregation | 13 |
| 4.6. Problems with zones data | 13 |
| 5. Discussion | 14 |
| 6. References | 15 |

1. Introduction

1.1. Motivation

Indian Institute of Technology, Bombay (IITB) has been the knowledge partner to Project on Climate Resilient Agriculture (PoCRA), Government of Maharashtra. A key contribution of the IITB team is to provide a scientific framework and tools for water budget computation. The core engine for the IITB-PoCRA water budget computation is a point level soil-water balance model integrated into a GIS environment.

Over the last four years, the software framework for water budget computation is being improved iteratively, based on valuable feedback received from PMU and validation data obtained by the IITB team through field work. This iteration of improvements focuses on dynamic computation of the water budget. It enables use cases such as kharif advisories on dry as well as wet spell impact at village level, real time water budget computation, and so on. The changes made to the software framework to implement dynamic water budget computation are discussed in this note.

1.2. Background

This note explains enhancements to the kharif soil moisture model, popularly known within PoCRA teams as ‘the point model’, scheduled for delivery by the IITB team as part of MoU IV. This delivery was originally scheduled for phase-I, however, considering PMU’s request for immediate deployment of the ‘well beneficiary module’ and the time constraints, it was decided that this item will be delivered in a later phase tentatively by the end of phase-IV. Now, given the PMU’s requirement for deployment and testing of the enhanced code on PMU’s machine well before the upcoming monsoon, it was decided that this item will be delivered in phase-III only¹.

The IT Stack report [1] from MoU-III has details on evolution of the point model since its inception, from being a daily model to where we started using hourly weather data for better accuracy. We refer to the version of the point model currently deployed within PMU [2] as V1 and the enhanced version to be delivered as part of MoU IV [3] as V2. The enhancements are

¹ Further, the delivery of ‘Interim Report on Implementation of Regional Geography and Integration with PMU IT Stack’ earlier scheduled for phase-III has been moved to phase-IV to accommodate and ensure consistent workflow as well as manmonths.

focused on improving the utility of the point model for more real time planning. The point model V1 has already been put to several uses within PMU and this document focuses on the use case for micro level planning (MLP) for V2.

1.3. Objectives

- To document the point model enhancement for dynamic computation of the water budget.
- To describe performance optimizations, testability and usability improvements implemented in point model V2.
- To document usage of enhanced point model (V2) to compute water budget for the villages that were left out during the 2020 water budget run for PoCRA region.

2. About Point Model V1

This section discusses the point model V1 that is currently deployed within PMU.

2.1. Use of Point Model V1 for MLP

The point model is run once per year, to evaluate crop water demand across the PoCRA region for crops that are known to be sown in that year. The input region is sampled into a grid of points and each point is attributed with parameters such as slope, soil type, soil depth, land use class, etc. Water balance is computed at each point using hourly weather data for the given year. The output is then aggregated over time, for the whole monsoon duration and beyond. The output is also aggregated over space using zones, such that one row of water balance parameters (AET, runoff, infiltration, soil moisture, PET, etc.) is emitted for each zone.

2.2. Challenges with V1

We now discuss some key challenges of the currently deployed point model version V1.

2.2.1. Water balance computation for all time steps at a point

Hourly weather data for the desired time frame is used by V1 to compute water balance for one point for all the time frames. The same computation is performed for the next point and the process continues until this is performed for all the points spanning the desired region. This

strategy does not align with how things happen in reality. In reality, the state of the region as a whole advances over time, based on changes in weather parameters and crop growth. A closer approximation of this reality would enable better understanding of the situation on ground and more informed planning.

2.2.2. Weather data format

Weather data format used by V1 involves arrays of each weather parameter having length = $365 * 24$ items. The weather data obtained from skymet is encoded in arrays and loaded into a PostgreSQL table. The V1 code to use weather data in this format must unnest all the array elements. In addition, each array element is assigned a day of year and hour of day. Data for other input parameters is in tabular format, which is read using a standard database cursor interface. The existing weather format proves cumbersome if weather data is to be analysed along with the point model output, say for example, to detect dry and wet spells. A better, more standard format involves one row per hour with all weather parameters as columns.

2.2.3. Performance

As mentioned earlier, each point in the spatially sampled input region needs to be assigned attributes such as slope, soil depth, soil type, nearest weather station, etc. Each of these input parameters is available as a vector or a raster layer from PostGIS. The code in V1 performs spatial join of each point with an input layer within Python code. QGIS and even more so, PostGIS, are designed to perform such computations much more efficiently. A CPU usage profile obtained for a test run of the V1 code using standard Python profiler [4] shows several performance bottlenecks within this area of computation. According to the feedback received from PMU, computation of water balance for one PoCRA district using V1 code requires about 7 to 8 hours for its completion. This runtime is prohibitively large.

2.2.4. Error logging and output identification

The output generated by V1 is a zone-wise csv file, having one row per zone, per crop with the water balance parameters. In this scenario, in case of an error, the villages for which water balance computation failed cannot be identified easily. After fixing the errors, possibly by rectifying input data, the next step is to re-run the point model. It is often needed to distinguish the output generated by the previous run from that of the latest run. This is not easily possible in V1.

3. Enhancements in Point Model V2

The enhancements being worked on by the IITB team are geared towards improving adoption of the point model within PMU for more real time micro level planning. The reader is encouraged to follow the source code links at the end and provide review comments on GitLab if any.

3.1. Space-time interchange

The innermost loop in V2 is flipped such that the point model now computes water balance for a single time step over all points. The state of the region advances further at each time step. Water balance results at each point are stored in a PostGIS table with point geometry at the end of the simulation. This enables the ability to pause and resume the point model at any time step. Given an automatic way to update hourly weather data as and when skymet makes new weather data available, V2 can be run for ongoing kharif season as a cron job, automatically.

3.2. Simpler hourly weather format

The hourly weather data format in V2 is simple viz. one row per hour. The complex logic to compute day of year, based on leap year detection, is no longer needed. The builtin date type offered by PostgreSQL is used for this purpose. The code to process hourly weather data has become simpler.

3.3. Improved usability

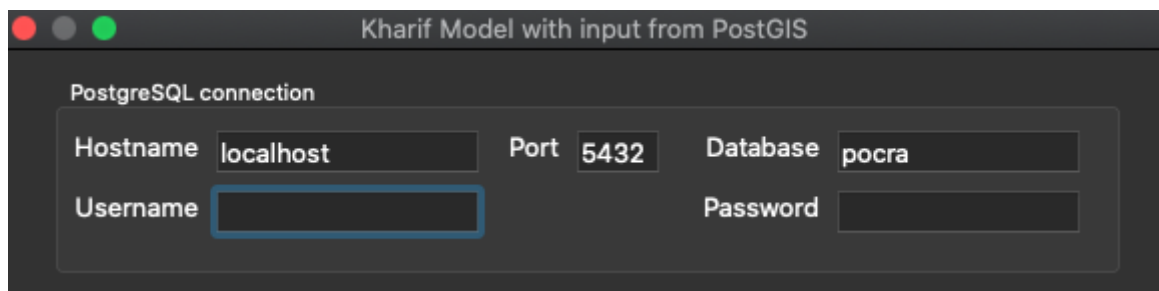
The V2 code runs a standalone Python application. It does not need to be invoked as a QGIS plugin. Inputs can be provided using a UI dialog box and a config file. The command to invoke the application is as shown in Figure 1. The config file argument is mandatory. The config file contains schema specifications and a few other details, accepting all of which from a UI dialog box is not convenient. The vincodes argument is optional, it is used only when a list of villages to compute water balance for is conveniently available.

```
$ kharif_multicrop_plugin_ponding/kharif_model.py --help
usage: kharif_model.py [-h] --config CONFIGFILE [--vincodes VINCODESFILE]

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIGFILE    data set and other parameters
  --vincodes VINCODESFILE
                        file containing vincodes of input villages, one vincode per line
```

Figure 1: Snippet of V2 command line

If a vincodes file is not provided, the villages to compute water balance need to be specified in the UI dialog box by entering one or more of the following parameters: district, taluka, village name or PoCRA cluster ID. A few snippets from the input dialog box are added below. Figure 2 shows the interface to enter database connection parameters. The point model V2 needs a PostgreSQL database with PostGIS extension installed, same as V1.



The screenshot shows a window titled "Kharif Model with input from PostGIS". Inside, there is a section labeled "PostgreSQL connection" containing five input fields: "Hostname" with the value "localhost", "Port" with the value "5432", "Database" with the value "pocra", "Username" (which is currently empty and has a blue border), and "Password" (which is also empty).

Figure 2: PostgreSQL connection parameters from input dialog

The PostgreSQL database is expected to have tables conforming to the certain schema. The schema name, table name and geometry / raster column name can be specified in the input dialog, as shown in Figure 3. The data sets needed to compute water balance, such as land use, soil, slope, etc. are visible in Figure 3.

| | Schema | Table | Geometry column |
|------------------|--------------|---------------------|-----------------|
| Villages table | a_dashboard | Village | geom |
| Regions table | vaterbalance | zones | geom |
| LULC table | a_dashboard | LULC1516 | geom |
| Soil table | a_dashboard | SoilDepth | geom |
| Drainage table | a_dashboard | River | geom |
| Slope (raster) | slope_raster | slope_osmanabad_525 | rast |
| Hourly weather | vaterbalance | hourly_weather | |
| Weather stations | vaterbalance | aws_locations_4326 | geom |
| Model output | vaterbalance | modeloutput | geom |
| Known crops | vaterbalance | crops | |

Figure 3: PostGIS table information from input dialog

This is a significant improvement from a usability standpoint. Such an interface was not available in V1. In order to run the point model, one had to modify certain lines in python files and then run them. Another improvement is that the code does not need to be colocated with the PostgreSQL database server. See the report on MLP script [5] for details on the usage of V1 code.

3.4. Improved performance

Using profilers available in Python, we have profiled the point model for CPU usage. The top CPU consumer is the formulae used to compute water balance at each point. In V2, points having similar slope, identical soil depth, soil texture, land use class and the same nearest weather station are grouped together. Water balance is computed once for each distinct group. This is referred to as grouping of points based on the hydrological response unit (HRU).

The next highest CPU consumer is the code that assigns spatial attributes to a point by evaluating containment of a point within other geometric and raster layers. This is made more efficient in V2 by pushing as much spatial join computation as possible towards the data, that is within PostGIS. When this is not easily possible, we have used QGIS native processing algorithms. With this strategy, spatial join computation within Python code is completely eliminated. Aggregation of point-wise water balance results for reporting purposes has also been pushed to PostGIS.

The optimizations in V2 have shown more than 50% reduction in total run time. E.g. water balance for the year 2020 for 196 villages in Yavatmal district for 48 crops was computed by V2 in about 200 minutes. Similar sized workload with V1 would have taken over 5 hours to finish. Note that V1 code must be run from the pocra_dashboard server deployed within PMU. It is closely coupled with data and does not have well defined, easy to run tests. Therefore, for V1 performance measurement, we have relied on accounts from PMU team members who were tasked with running the V1 code on a yearly basis.

3.5. Test automation using a sandbox dataset

One of the most important properties of high quality software is ability to test it and quickly get feedback on a code change. Several new tests have been added to V2 code, such as mass conservation at the end of water balance computation, automatic comparison of generated point wise output with a known good output / oracle. GNU Make is used to invoke tests. The data set used for tests is generated by extracting a sample of each input table from the PMU database. The test database is initialised with the “test_setup” make target. Some of the sample tests from a test run are shown below.

```
$ make test_setup && make QGIS_PREFIX_PATH=/user test
#3 PostGIS layers having different CRS are rejected ... ok
#5 Test that output from multiple runs can be distinguished ... ok
#6 Execute soil moisture model against PostGIS inputs ... ok
#10 PET should be 0 before sowing and after harvest ... ok
#12 Validate water balance on multiple grid points ... ok
#13 Weather data parsing logic honors start and end dates ... ok
#15 Validate sowing day for Sengaoon rain circle in 2020 ... ok
```

There are many more tests, only a few are included in the above snippet for brevity. Besides being able to run the tests using a simple `make` command, the continuous integration infrastructure offered by GitLab on cloud is leveraged to run the tests automatically [6], whenever the point model code is changed. If a test fails, an email notification with a link to details of the failure and the culprit commit is sent to the team. This infrastructure is set up by defining a simple pipeline in a `.gitlab-ci.yml` file located at the top level in the source repository [3].

4. Water budget for villages left out in 2020

Among over 5000 villages in PoCRA region, 673 villages were left out from water budget computation for the year 2020. This section demonstrates water budget computation for the year 2020 using the enhanced point model (V2) for some of the 673 villages. Unlike V1, the point model V2 can be run on any workstation that meets the prerequisites in [7]. Step by step instructions to run the point model are outlined below.

4.1. Obtain the latest source code (step 1)

Open terminal and run the following commands. Replace the argument of `git clone` with the link target, starting with “`http://`”.

```
$ git clone pocra-iitb/kharif\_multicrop\_plugin\_ponding
$ cd kharif_multicrop_plugin_ponding
```

4.2. Obtain input data (step 2)

The data set required to run the point model V2 includes PostGIS tables for soil, land use, slope, zones, villages, hourly weather data and the MLP crops data containing crop ID and WALMI PET values. This data for the PoCRA region is available within the PostgreSQL

instance running on the PMU dashboard VM in Microsoft Azure cloud, IP address 104.211.153.215, port 5433.

4.3. Running the Python application (step 3)

The slope raster in the PMU dashboard database is partitioned by district. Consequently, the point model needs to be run district-wise. The program **kharif_model.py** is the main entry point. From the 673 remaining villages, 37 villages from Osmanabad are given as input for this demonstration. Census codes of these villages are written to a text file with one code per line. This text file is specified as the **--vincodes** argument. The config file specified as the **--config** argument contains details of each PostGIS table including schema, table and column names. To run the program, type the following commands in the same terminal opened in step 1.

```
$ export QGIS_PREFIX_PATH=/usr
$ QGIS_PROCESSING=/usr/share/qgis/python/plugins
$ export PYTHONPATH=$PWD:$PWD/...$QGIS_PROCESSING:$PYTHONPATH
$ ./kharif_model.py \
    --vincodes=remaining_vincodes_osmanabad.csv \
    --config=config_osmanabad.ini
```

A dialog box shown in figure 4 will pop up. The parameters are populated from the specified config file and may be changed as needed.

4.4. Examine the generated logs (step 4)

The point model writes messages including village-wise progress of water budget computation, the timestamp that identifies output rows being generated by this run, warnings and errors, if any, to the file **kharif_model.log** under the specified output directory. For example, the first few lines emitted by the mode run in step 4 are:

```
INFO Starting to process 37 villages with run id '2022-05-03 17:53:05'
monsoon start 2020-06-01, monsoon end 2020-10-31, model end 2020-11-10
INFO          Progress:          0/37          villages          done
Processing village 561411
```

Kharif Model with input from PostGIS

PostgreSQL connection

Hostname Port Database

Username Password

Data source

District Taluka

Village PoCRA cluster ID

| | Schema | Table | Geometry column |
|------------------|---|--|-----------------------------------|
| Villages table | <input type="text" value="a_dashboard"/> | <input type="text" value="Village"/> | <input type="text" value="geom"/> |
| Regions table | <input type="text" value="waterbalance"/> | <input type="text" value="zones"/> | <input type="text" value="geom"/> |
| LULC table | <input type="text" value="a_dashboard"/> | <input type="text" value="LULC1516"/> | <input type="text" value="geom"/> |
| Soil table | <input type="text" value="a_dashboard"/> | <input type="text" value="SoilDepth"/> | <input type="text" value="geom"/> |
| Drainage table | <input type="text" value="a_dashboard"/> | <input type="text" value="River"/> | <input type="text" value="geom"/> |
| Slope (raster) | <input type="text" value="slope_raster"/> | <input type="text" value="slope_osmanabad_525"/> | <input type="text" value="rast"/> |
| Hourly weather | <input type="text" value="weather"/> | <input type="text" value="hourly_weather"/> | <input type="text"/> |
| Weather stations | <input type="text" value="waterbalance"/> | <input type="text" value="aws_locations_4326"/> | <input type="text" value="geom"/> |
| Model output | <input type="text" value="waterbalance"/> | <input type="text" value="modeloutput"/> | <input type="text" value="geom"/> |
| Known crops | <input type="text" value="waterbalance"/> | <input type="text" value="crops"/> | <input type="text"/> |

Model parameters

Kharif crops Output directory

Monsoon year (yyyy) Sowing threshold

Simulation length (days)

Figure 4: point model input dialog

The highlighted timestamp is used to inspect the point-wise water budget output generated by this run. At the end of the run, the log file contains a summary, as follows:

```
INFO 35 of 37 villages processed in 2819 seconds, timestamp for output
identifier: '2022-05-03 17:53:05'
processing failed for the following villages:
561274: Feature (15885) from "regions" has invalid geometry ...
561273: Feature (15885) from "regions" has invalid geometry ...
```

As seen from the log messages, the model took about 47 minutes to finish. Water budget could not be computed for 2 villages because of invalid geometry in the zones table. The feature ID 15885 can be used to query the zones table as follows.

```
select vrcode, zone_name, st_isvalid(geom), st_astext(geom, 2) from
waterbalance.zones where gid = 15885;
NOTICE: Self-intersection at or near point 589740.04 2042182.05
  vrcode | zone_name | st_isvalid | st_astext
-----+-----+-----+-----
  561273 | Bavi-1    | f          | MULTIPOLYGON(...)
(1 row)
```

GIS personnel within PMU need to analyse this problem and fix the geometry so that water budget for the skipped villages can be computed successfully.

4.5. Inspect output artefacts (step 5)

The three artefacts produced by the point model are the point-wise output in a PostGIS table, zone-wise aggregate water budget in CSV format, and the SQL script used to generate the aggregate output. Each one is described below.

4.5.1. Pointwise water budget

The user-specified PostGIS table is populated with point-wise water budget parameters. In this example, it is `waterbalance.modeloutput`. The table can be queried using SQL. The following query shows the total number of villages, zones and points processed in this run.

```
select count(distinct vrcode) villages, count(distinct region_name) zones,
count(*) points from waterbalance.modeloutput
where run_id = '2022-05-03 17:53:05';
  villages | zones | points
-----+-----+-----
       35 |    94 | 199024
```

4.5.2. Zone-wise aggregate output

A CSV file with one row for each zone is created. This file feeds into the MLP workflow to obtain village level water balance charts. The path of the CSV file is reported in the last few lines of the log file:

```
INFO query used for region-wise aggregation: .../region-wise-agg.sql
INFO Region wise output: .../region-wise-output.csv (4083 lines)
Done
```

A row in the CSV output contains zone name, crop ID and aggregated water budget parameters such as AET, PET, runoff, etc. over the zone. The CSV output conforms to the following key properties.

- Mass conservation at the end of monsoon for non-rabi crops (all quantities in mm):
$$\text{rainfall} = \text{AET} + \text{GW recharge} + \text{runoff} + \text{soil moisture}$$
- PET crop end = PET monsoon end + PET post monsoon
- For crops that are not harvested before the model end date (the last time step), WALMI PET from MLP crops database is used as PET crop end.

4.5.3. SQL query for aggregation

The CSV output is generated by aggregating the point-wise output using SQL. The SQL statement used can be found in **region-wise-agg.sql**. The SQL file may be run again to get the same aggregate data or tweaked to get a different level of aggregation (e.g. village-wise water budget instead of zone-wise). Basic knowledge of SQL is necessary to understand and tweak this SQL query.

4.6. Problems with zones data

As shown in the point model run for 37 villages in Osmanabad, the zones data contains invalid geometries. The zones data also contains duplicate geometries: 159 zones in phase_1_clusters_zoning_updated, 2 zones in phase_2_clusters_zoning_updated and 1 from phase_3_clusters_zoning_updated. The following SQL statement can identify duplicate geometries.

```
select
    vrcode, zone_name, st_area(geom)::numeric(10,2) area,
    st_perimeter(geom)::numeric(10,2) perimeter, count(*)
from
    adminregions.phase_1_clusters_zoning_updated
group by
    vrcode, zone_name, geom
having count(*) > 1 order by vrcode, zone_name;
```

The output of this query indicates as many as 9 copies of the same geometry for some zones, as seen in the following sample from the output rows:

| vincode | zone_name | area | perimeter | count |
|---------|-------------|------------|-----------|-------|
| 528619 | Gadegaon-1 | 617962.14 | 4114.68 | 2 |
| 529853 | Rel-6 | 3132999.83 | 14077.14 | 4 |
| 532074 | Kurankhed-1 | 4353278.85 | 11830.57 | 9 |
| 532074 | Kurankhed-2 | 2010253.80 | 9453.91 | 9 |
| 532075 | Nimkheda-1 | 1417414.40 | 6470.47 | 9 |

Once the reason for duplicates is ascertained by GIS personnel in PMU, the duplicate records can be removed using a simple SQL statement such as the following:

```
delete from adminregions.phase_1_clusters_zoning_updated z1
using adminregions.phase_1_clusters_zoning_updated z2
where
    z1.ctid < z2.ctid and
    z1.vincode = z2.vincode and
    z1.zone_name = z2.zone_name and
    z1.geom = z2.geom;
```

5. Discussion

The enhancements to the point model described above are possible through continuous rigorous testing, field observations and prototyping from the IITB team and valuable feedback received from PMU. We hope to receive more such feedback and continue to use it to improve the point model. Water balance data can be made available in real time if the point model V2 is deployed to be run periodically as and when new weather data is available from skymet. This should broaden the horizons for various stakeholders including PMU and decision makers at village level.

Deployment of the point model V2 and eventual handover to PMU will be much more efficient if the PMU IT team can engage at source code level, by participating in code review, development and testing activities. The interface provided by GitLab [3] can be leveraged for this purpose.

6. References

- [1] <https://drive.google.com/file/d/13DuVAcbPbcgN5uYnb0iV1VFTsodqSXQE/view>
- [2] https://gitlab.com/pocra-iitb/kharif_multicrop_plugin_ponding/-/tree/V1_MoU_III
- [3] https://gitlab.com/pocra-iitb/kharif_multicrop_plugin_ponding
- [4] <https://docs.python.org/3/library/profile.html>
- [5] <https://drive.google.com/file/d/18zI2c3bNzqXsuQ90N9Je1q5yCpofT6iq/view>
- [6] https://gitlab.com/pocra-iitb/kharif_multicrop_plugin_ponding/-/pipelines
- [7] https://gitlab.com/pocra-iitb/kharif_multicrop_plugin_ponding/-/blob/main/README.md