AutoDQM

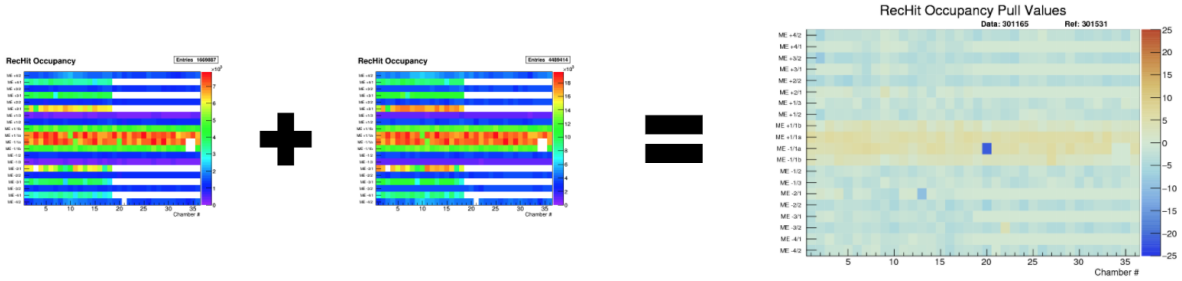
Release 127.0.0.1

CMS Collaboration

Nov 25, 2020

CONTENTS:

AutoDQM is a statistical tool for Data Quality Management (DQM) at the Large Hadron Particle Collider.



Before AutoDQM, DQM shifters were tasked with looking at hundreds of DQM histograms, looking for hard-to-spot issues in data collection. AutoDQM runs long established statistical tests using ROOT on these graphs and outputs outliers on a simple, but effective, GUI that expedites the shifters' task.

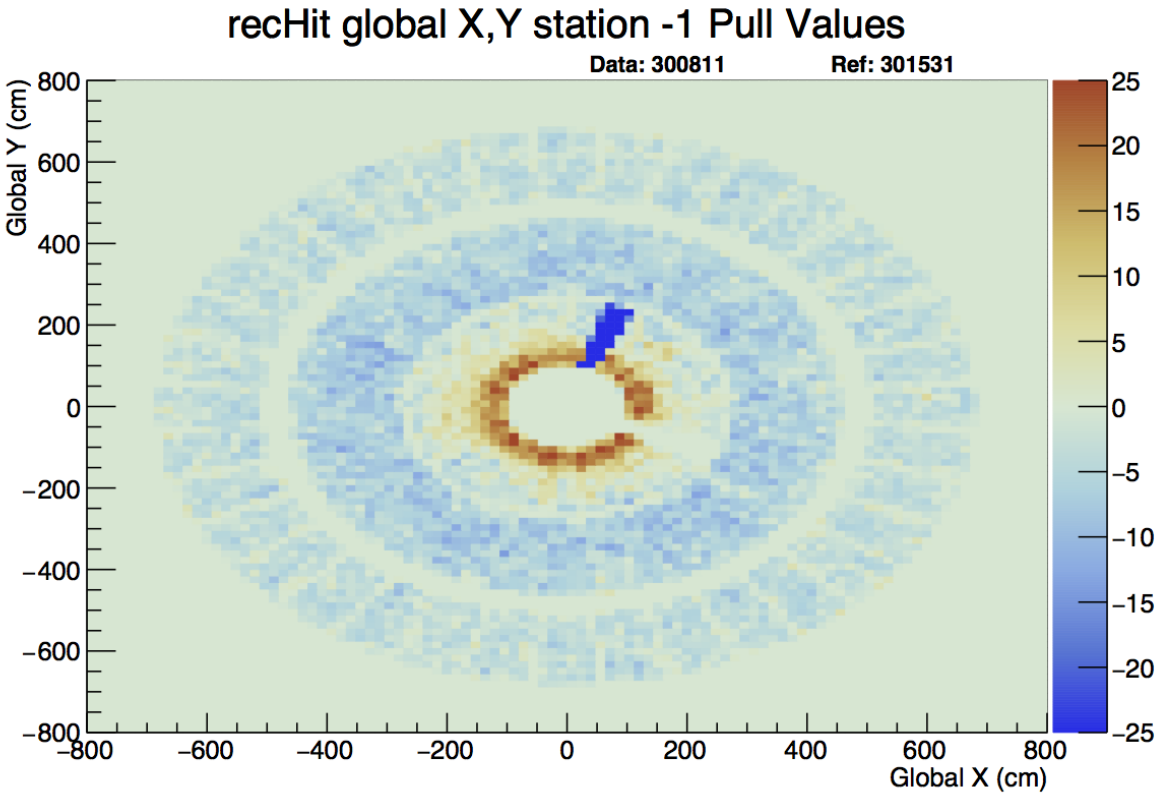
CHAPTER

ONE

THE STATISTICAL ARSENAL

AutoDQM uses a variety of long-established statistical tests to determine how similar a "data" run is to a reference run selected by the DQM shifter.

1.1 Bin-by-Bin Pull Values

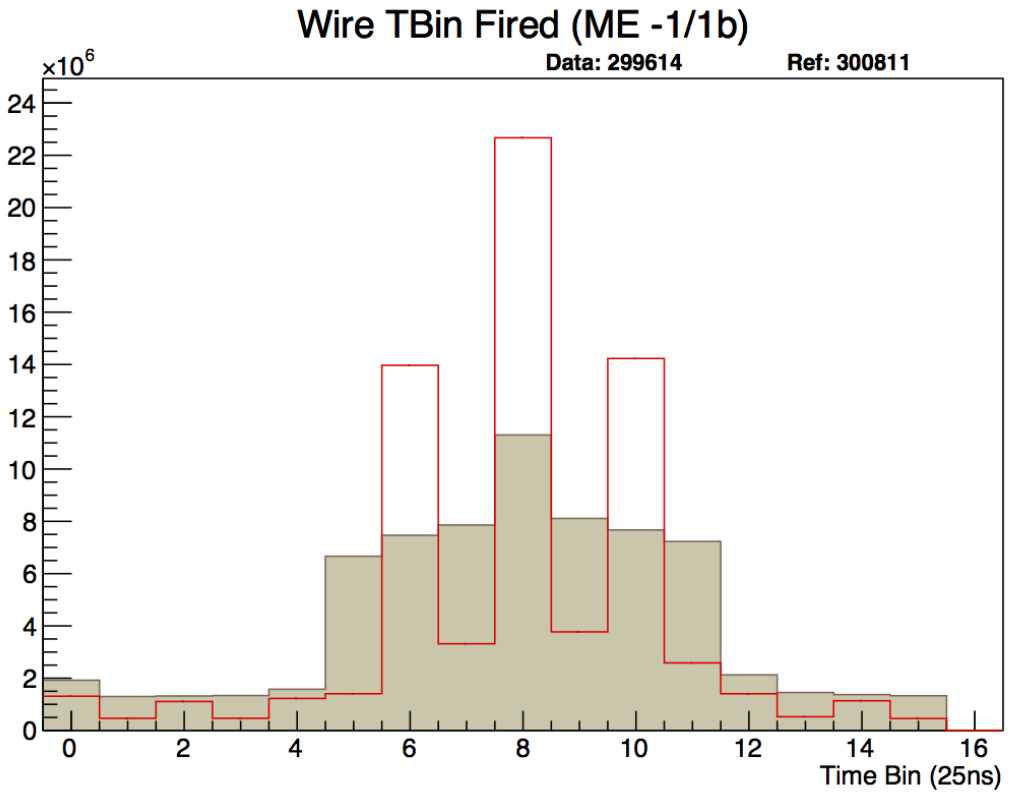


For 2D histograms AutoDQM goes bin-by-bin through both the data and reference histograms, then plots the difference between each corresponding bin onto a new, identical histogram, taking proper Poisson errors into account. The equation for this calculation is fairly simple:

$$\text{pull value} = \frac{(x_1 - x_2)^2}{\epsilon_1^2 + \epsilon_2^2}$$

where "x" is the bin value and epsilon is the error of that bin value. The code for this can be found here.

1.2 Kolmogorov-Smirnov Test

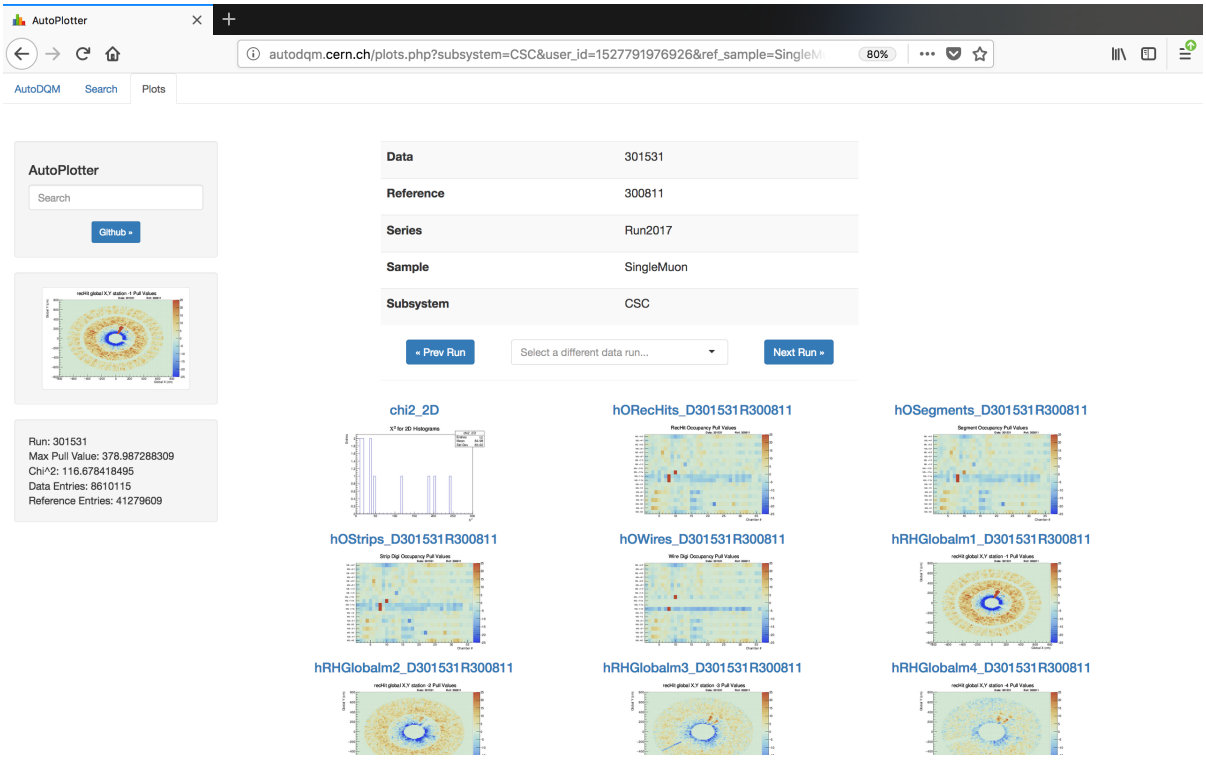


The Kolmogorov-Smirnov test compares the distance between points on a sample distribution and some reference distribution and qualifies how close or far they are. It is too complex to discuss here, but Wikipedia has a good article on it.

CHAPTER

TWO

EXAMPLE OUTPUT:



Here, Run 301531 was taken as the 'data' run with Run 300811 as the reference. AutoDQM outputted more readable graphs with clear indications that there are some dead cells.

2.1 AutoDQM

AutoDQM parses DQM histograms and identifies outliers by various statistical tests for further analysis by the user. Its output can be easily parsed by eye on an AutoPlotter-based html page which is automatically generated when you submit a query from the AutoDQM GUI. Full documentation for AutoDQM can be found on our wiki.

1. *Features*
2. *Setting Up AutoDQM for Development*
3. *Using AutoDQM Offline*
4. *Environment Variables*

2.1.1 Features

AutoDQM.py

- [x] Outputs histograms that clearly highlight outliers
- [x] Creates a .txt file along with each .pdf with relevant information on it
- [x] Allows user to easily change input
- [x] Seeks and accurately finds outliers

index.php

- [x] Previews input in a readable way
- [x] Gives a clear indication of the status of a user's query

plots.php

- [x] Dynamically displays text files below AutoPlotter toolbar
- [x] Unique url's for sharing plots pages with the data and reference data set names

2.1.2 Setting Up AutoDQM for Development

This shows how to set up AutoDQM to be served from your local machine or a machine on CERN OpenStack. This was written based on a fresh CC7 VM on CERN OpenStack.

You'll need a CERN User/Host certificate authorized with the CMS VO. CMS VO authorization can take ~8 hours so bear that in mind. Certificates can be aquired either from https://cern.ch/ca or, on a CC7 machine, by using auto-enrollment https://ca.cern.ch/ca/Help/?kbid=024000.

Install docker according to https://docs.docker.com/install/ and docker-compose through pip because CC7 has an old versions in it's repositories. Enable+start the docker service, and be sure to add your user to the docker group.

```
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install docker-ce -y
sudo yum install python-pip -y
sudo pip install docker-compose
```

```
sudo gpasswd -a [user] docker
sudo systemctl enable --now docker
```

You may need to relog into your account before the group settings take effect.

Store your CERN certificate into docker secrets. You may need to extract your cert from PKCS12 format:

```
openssl pkcs12 -in cern-cert.p12 -out cern-cert.public.pem -clcerts -nokeys
openssl pkcs12 -in cern-cert.p12 -out cern-cert.private.key -nocerts -nodes
```

```
docker swarm init
docker secret create cmsvo-cert.pem cern-cert.public.pem
docker secret create cmsvo-cert.key cern-cert.private.key
```

Then initialize a docker swarm, build the autodqm image with docker-compose, and deploy the image as a docker stack

```
docker-compose build
docker stack deploy --compose-file=./docker-compose.yml autodqm
```

You can now view AutoDQM at `http://127.0.0.1`. If you would like to make your instance of AutoDQM public, oopen port 80 to http traffic on your firewall. For example, on CC7:

```
sudo firewall-cmd --permanent --add-port=80/tcp
sudo firewall-cmd --reload
```

After making changes to configuration or source code, rebuild and redeploy the newly built image:

```
docker-compose build
docker stack rm autodqm
docker stack deploy --compose-file=./docker-compose.yml autodqm
```

If you're using a CC7 image, you may want to disable autoupdate:

```
sudo systemctl stop yum-autoupdate.service
sudo systemctl disable yum-autoupdate.service
```

2.1.3 Using AutoDQM Offline

The `./run-offline.py` script can retrieve run data files and process them without needing a web server. Run `./run-offline.py --help` for all the options.

1. Supply certificate files to the environment variables below. Alternatively, the default uses the files produced when running voms-proxy-init, so that may work instead.
2. Use `./run-offline.py` to process data with AutoDQM

2.1.4 Environment Variables

- `ADQM_CONFIG` location of the configuration file to use
- `ADQM_DB` location to store downloaded root files from offline DQM
- `ADQM_TMP` location to store generated temporary pdfs, pngs, etc
- `ADQM_SSLCERT` location of CMS VO authorized public key certificate to use in querying offline DQM
- `ADQM_SSLKEY` location of CMS VO authorized private ky to use in querying offline DQM
- `ADQM_CACERT` location of a CERN Grid CA certificate chain, if needed

2.2 Online

The AutoDQM GUI is currently hosted on a VM at CERN, which can be accessed in a few simple steps. If you're on the CERN network simply navigate to http://autodqm.cern.ch, otherwise you need to set up a proxy into the CERN network:

1. Setup your browser's SOCKS Proxy settings (Instructions: Firefox (recommended), Chrome)
2. `ssh` into CERN from your terminal

   ```
   >> ssh <username>@lxplus.cern.ch -ND 1080
   ```

3. Navigate to http://autodqm.cern.ch from the browser you set up.

2.3 Offline

The `./run-offline.py` script can retrieve run data files and process them without needing a web server. Run `./run-offline.py --help` for all the options.

1. Supply certificate files to the environment variables below. Alternatively, the default uses the files produced when running voms-proxy-init, so that may work instead.
2. Use `./run-offline.py` to process data with AutoDQM

AutoDQM is an open source project. As such, we are constantly ensuring that AutoDQM is configurable for all subsystems that are interested in using it. With Release v2.1.0, it is now possible to add algorithms to AutoDQM's processing step, making it easy for anyone interested in improving the tool to contribute.

1. The Configuration JSON
2. Adding Your Subsystem
3. Adding an Algorithm

2.4 The Configuration JSON

If AutoDQM is not already configured for your subsystem, you can easily push configurations for it. AutoDQM has a conveniently named `` `configs.json <https://github.com/jkguiang/AutoDQM/blob/release-v2.0.0/configs.json>`_ ``, where each entry in the JSON contains the appropriate instructions for AutoDQM's processing algorithms. Let's break it down:

2.4.1 Example entry:

```
"CSC": {
    "main_gdir":"DQMData/Run {0}/CSC/Run summary/CSCOfflineMonitor/",
    "hists":[
      {
        "ks_cut":0.0,
        "norm_type": "row",
        "path":"Occupancy/hORecHits"
      }

...

      {
        "ks_cut":0.0,
        "path":"recHits/hRHGlobal*"
      },

...

      {
        "always_draw":true,
        "path":"Digis/hWirenGroupsTotal"
      },
```

"CSC": Each entry in the JSON is indexed by the acronym of the subsystem: in this case, "CSC". Each subsystem is, itself, another python dictionary.

`"main_gdir"`: The path to the DQM histograms for any subsystem included in the DQM `.root` files. Generally, each run has a different run number in its directory path, so we put a pythonic `"{0}"` in its place so that the path string may be used dynamically in the processing script.

`"hists"`: This is a list of python dictionaries which each contain the path to specific histograms for AutoDQM to process as well as several tuning parameters.

- `"path"`: This specifies the path to the histogram. If there are many histograms of the same type, you may use an asterisk to tell AutoDQM to find all histograms of a matching path (see line 27 above). `**\ *This is the ONLY required parameter for every histogram*\ **`
- `"ks_cut"`: This is the cut for the Kolmogorov-Smirnov Test that is run on 1D histograms.
- `"norm_type"`: Currently AutoDQM only normalizes histograms by two schemes: row-by-row or as a whole. If you would like AutoDQM to normalize this histogram row-by-row, you would include this option, pointing to `"row"`. Otherwise, you do not need to include this parameter.
- `"always_draw"`: Include this parameter pointing to the boolean value `true` to tell AutoDQM to *always* draw this histogram.

2.5 Adding Your Subsystem

To add your subsystem, simply clone the latest release, make the necessary additions to `configs.json`, then make a pull request.

2.6 Adding an Algorithm

To add an algorithm to AutoDQM's processing step, first contact us at autodqm@gmail.com so that we may collaborate on designing an appropriate addition. Assuming that the decision has been made to add your algorithm, you must first ensure that it is properly formatted:

2.6.1 The Comparator Object

```
def comparators():
    return {
        'new_algo': new_algo
    }
```

This object associates a string to the algorithm function you've written. This allows AutoDQM to find and use your algorithm.

2.6.2 Algorithm Arguments and the Histpair object

```
def new_algo(histpair,
             new_cut=500, min_entries=100000, new_option='new_opt',
             **kwargs):
```

Every algorithm that AutoDQM uses must handle a `histpair <https://github.com/jkguiang/AutoDQM/blob/release-v2.1.0/autodqm/histpair.py>`_object. Put simply, each histpair object contains all of the information passed from the user's input (i.e. the name of the data and reference runs, the series and samples of those runs, etc.). Any other key word arguments should be specified or otherwise passed through **kwargs.

2.6.3 Plugin Results Object

```
from autodqm.plugin_results import PluginResults

def new_algo( ..., **kwargs ):

    """
        New algorithm contents
    """

    return PluginResults(
        c,
        show=is_outlier,
        info=info,
        artifacts=artifacts
```

In order for AutoDQM to understand your algorithm's output (i.e. drawn histograms, text files, etc.), you must pass them in a `PluginResults <https://github.com/jkguiang/AutoDQM/blob/release-v2.1.0/autodqm/plugin_results.py>`_object.

When your algorithm has been properly formatted, you can make a pull request to AutoDQM's development branch, making sure to place it in the plugins directory.

We have written instructions on how to set up a development version of AutoDQM. Unless you have local DQM files on hand, we recommend using a CERN OpenStack VM. Instructions for setting one up can be found here.

2.7 Environment Variables

- ADQM_CONFIG location of the configuration file to use
- ADQM_DB location to store downloaded root files from offline DQM
- ADQM_TMP location to store generated temporary pdfs, pngs, etc
- ADQM_SSLCERT location of CMS VO authorized public key certificate to use in querying offline DQM
- ADQM_SSLKEY location of CMS VO authorized private ky to use in querying offline DQM
- ADQM_CACERT location of a CERN Grid CA certificate chain, if needed

2.8 Join the team!

Interested in contributing to AutoDQM? Just shoot us an email at autodqm@gmail.com. If you're anxious to get started, just clone the latest release and follow the instructions in the README to get a working development environment. Note: you will need Docker on whatever machine or server you clone AutoDQM to in order to run the web browser outside of the dedicated CERN VM.

2.8.1 Current Developers

- Jonathan Guiang
- Alex Aubuchon

2.9 autodqm package

2.9.1 Submodules

autodqm.cfg module

exception autodqm.cfg.**error**
Bases: Exception

autodqm.cfg.**get_subsystem**(*cfg_dir*, *subsystem*)
Return the dict-based configuration of subsystem from cfg_dir.

autodqm.cfg.**list_subsystems**(*cfg_dir*)
Return a list of names of subsystem configs in cfg_dir.

autodqm.compare_hists module

autodqm.compare_hists.**compile_histpairs**(*config_dir*, *subsystem*, *data_series*, *data_sample*, *data_run*, *data_path*, *ref_series*, *ref_sample*, *ref_run*, *ref_path*)
Compiles histogram pairs

exception autodqm.compare_hists.**error**
Bases: Exception

autodqm.compare_hists.**identifier**(*hp*, *comparator_name*)
Return a *hashed* identifier for the histpair

autodqm.compare_hists.**load_comparators**(*plugin_dir*)
Load comparators from each python module in ADQM_PLUGINS.

autodqm.compare_hists.**process**(*config_dir*, *subsystem*, *data_series*, *data_sample*, *data_run*, *data_path*, *ref_series*, *ref_sample*, *ref_run*, *ref_path*, *output_dir='./out/'*, *plugin_dir='./plugins/'*)
Processes histograms

This is a further change, I'm using autodoc_mock_imports = ["root"]

Will now try ROOT instead of root

autodqm.dqm module

class autodqm.dqm.**DQMRow**(*name*, *full_name*, *url*, *size*, *date*)
Bases: tuple

property date
Alias for field number 4

property full_name
Alias for field number 1

property name
Alias for field number 0

property size
Alias for field number 3

property url
Alias for field number 2

class autodqm.dqm.**DQMSession**(*cert*, *db*, *cache=None*, *workers=16*)
Bases: requests_futures.sessions.FuturesSession

Encapsulates an interface to DQM Offline.

fetch_run(*series*, *sample*, *run*)
Fetch and cache a run data file.

Returns the path to the downloaded file.

fetch_run_list(*series*, *sample*)
Return DQMRows corresponding to the runs available under the given series and sample.

fetch_sample_list(*series*)
Return DQMRows corresponding to the samples available under the given series.

fetch_series_list()
Return DQMRows corresponding to the series available on DQM Offline.

stream_run(*series*, *sample*, *run*, *chunk_size=4096*)
Stream and cache a run data file.

Returns a generator that yields StreamProg tuples corresponding to the download progress.

class autodqm.dqm.**StreamProg**(*cur*, *total*, *path*)
Bases: tuple

property cur
Alias for field number 0

property path
Alias for field number 2

property total
Alias for field number 1

exception autodqm.dqm.**error**
Bases: Exception

autodqm.histpair module

class autodqm.histpair.**HistPair**(*config*, *data_series*, *data_sample*, *data_run*, *data_name*, *data_hist*, *ref_series*, *ref_sample*, *ref_run*, *ref_name*, *ref_hist*)

Bases: object

Data class for storing data and ref histograms to be compared by AutoDQM, as well as any relevant configuration parameters.

autodqm.plugin_results module

class autodqm.plugin_results.**PluginResults**(*canvas*, *show=False*, *info={}*, *artifacts=[]*)

Bases: object

Data class for storing the results of a plugin function

self.canvas: the canvas to be saved and displayed self.show: whether the canvas should be shown by default self.info: dictionary of any extra information that should be displayed self.artifacts: root objects that need to be protected from garbage collection

PYTHON MODULE INDEX

a

INDEX