Chapter 1

Running a study

1. Download (or update) epistudy_configs from git repository (https://ndsslgit.vbi.vt.edu/ndssl-software/dicex_epistudy)

```
you@sfxlogin1: cd <git_repo_dir>
you@sfxlogin1: git clone git@ndsslgit.vbi.vt.edu:ndssl-software/dicex_epistudy.git
```

2. Turn on the dicex enviorment

```
you@sfxlogin1:. /home/pgxxc/public/dicex/dicex.sh
```

   The dot in the last command is necessary

3. Create an experiment directory from where you will run a study

```
you@sfxlogin1: mkdir <exp_work_dir>
you@sfxlogin1: cd <exp_work_dir>
```

4. Choose a study config file from epistudy_configs, lets say block study. The config file for this study is in epistudy_configs/block/epistudy_cfg.xml

5. Copy the file to the experiment work directory

```
you@sfxlogin1: cd <exp_work_dir>
you@sfxlogin1: cp <git_repo_dir>/epistudy_configs/block/epistudy_cfg.xml .
```

6. Open the file and modify the paramters. The parameters mentioned at the top (shown below) of the file is mandatory

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE epistudy [
<!ENTITY var_region "Miami">
<!ENTITY var_cfg_dir "/home/sandeep/git_repos/epistudy_configs/">
<!ENTITY var_run_dir "/home/sandeep/experiments/dicex_epistudy/">
<!ENTITY var_intv_type "block">
<!ENTITY var_dicex_base_dir "/home/pgxxc/public/">
...
...
]>
```

   Change the var_cfg_dir, var_run_dir to point to appropriate directory. var_cfg_dir is the path for epistudy_configs directory. var_run_dir is the path for directory where the run output and config files will be stored.

7. Now call epistudy_workflow with the corresponding epistudy_config file. For block study the command would look like

```
you@sfxlogin1: epistudy_workflow.sh epistudy_cfg.xml
```

   This would launch a database, and run all the cells, and the replicates.

Some additional fine points and dicex intracacies/weirdness/gotchas

- **The database runs on compute node on an arbitary chosen port**. To access the database, first find the hostname and port number from the dbsession_<dbsession-tag>.py file. This file would look as follows:

```
server_ip="sfx052"
server_port=5853
work_dir="/home/sandeep/experiments/dicex_epistudy/tmpGn417y"
jobid="1966115.master.cm.cluster"
pgpid="4219"
```

  Now ssh into the compute node as

```
you@sfxlogin1: ssh <server_ip> #or in this example, ssh sfx052
```

  Access the database as follows:

```
you@<some-compute-node>: psql -p <server_port> postgres #or in this example, psql -p 5853 postgres
```

  Alternatively, the tables can be accessed from the shadowfax login node itself:

```
you@sfxlogin1: dpsql epistudy_cfg.xml
```

  This will open the postgres client program and provide a prompt. You would need to know basic sql (`http://www.itl.nist.gov/div897/ctg/dm/sql_examples.htm`).

- **Please remove the job after it is finished**
  This is necessary because we create files in the temporary disk folder and if left unremoved it would cause troubles. lease follow these steps.

```
you@sfxlogin1: epistudy_cleanup.sh epistudy_cfg.xml
```

Chapter 2

Analytics and Visualization on replicate simulation outputs

In addition to defining an experiment design and running it using Shadowfax, Dicex_EpiStudy allows for analysis and visualization of output data across the replicates of the experiment design. The main functionality here is that the configuration file used to define the experiment design is also used for driving the visualization and analysis.

Provided with this git repository is a set of analysis scripts for block intervention study. These are

- replicate_analysis_block_intv.py
- replicate_analysis_block_intv_impl.py

Browse through these files to get a feel how to use the analysis routines.

2.0.1 Running an analysis for each replicate

Running a single type of analysis for each of the replicate is done through a mechanism that is often described as visitor pattern. The analysis is described as function that has the signature described below. Next, this function is handed over to another function called visit_replicates. visit_replicates calls the analysis function for each occurance of the replicate descriptor. The signature of the analysis function

```
def my_analysis_func(cfg_root=None, param_root=None, replicate_desc=None, rep_dir=None, args=None):
```

Here the main arguments are replicate_desc which is the descriptor of the replicate and the args.

Now, to run the analysis for the each of the generated replicate call the visitor with your analysis function as argument as follows:

```
import replicate_visitor as rv
rv.visit_replicates(cfg_root=cfg_root, method=my_analysis_func, args=[session])
```

The last "[session]" argument is necessary. If my_analysis_func needs additional arguments, then they should be added to the args list. For a complete working example see the accompanying example files mentioned above.

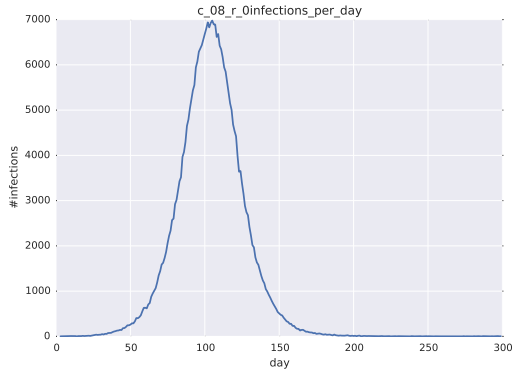
Why use visitor pattern and not simply a list of replicate descriptors? The descriptor names are "opaque" and do not tell what the configurations of the replicates where. The cfg_root is a handle to the XML file the defines the replicate configuration. It can be used, for example, to normalize the output value with respect to input parameter value.

2.0.2 What kind of analysis/plots are possible using the tool

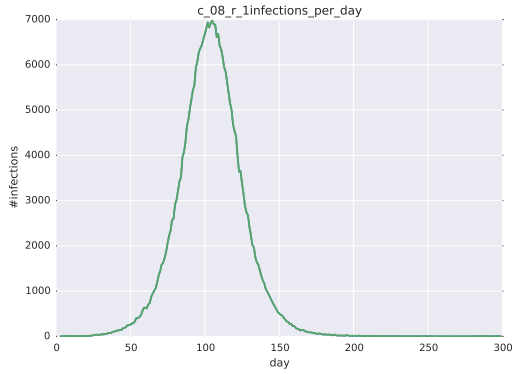
The Dicex_Epistudy uses VERSA data engine for analysis and plotting. The possibilites are in the relam of data joining, filtering, and aggregations (count, sum, mean, max, avg.). Currently, line and bar graphs are supported (a cool DynamicCircos feature should be added soon).

However, lets go over the simplest analysis where we just print the output on to the console, sort of the "Hello World" equivalent.

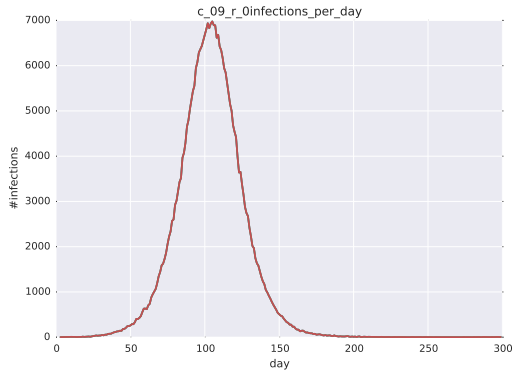
```
def hello_world(cfg_root=None, param_root=None, replicate_desc=None, rep_dir=None, args=None):
    import versa_api as vapi
    session = args[0]
    model_obj = rau.get_replicate_model_obj(replicate_desc, '_block_daily_diag_count')
    print vapi.scan(session, model_obj)
```



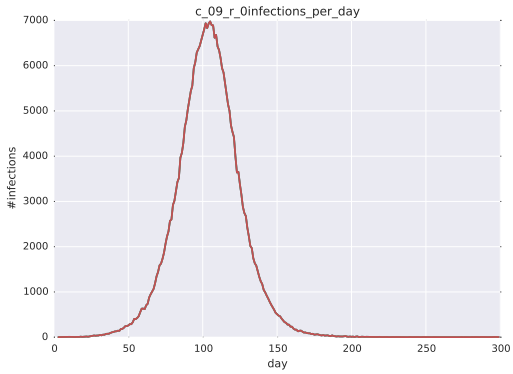
(a) c_08r_0



(b) c_08r_1



(c) c_09r_0



(d) c_09r_1

Figure 2.1: Performing analysis and plots for each replicate

The above will print the records for "block_daily_diag_count" table for all the replicates.

2.0.3 Drawing epicurves for each replicates

The table "block_daily_diag_count" maintains number of infections per day per block. In the next example, we will aggregate over day to compute number of infections per day and plot this generated epicurve.

```
import versa_api as vapi
import versa_api_analysis as vapia
def draw_epicurve(cfg_root=None, param_root=None, replicate_desc=None, rep_dir=None, args=None):

    session = args[0]
    model_obj = rau.get_replicate_model_obj(replicate_desc, '_block_daily_diag_count')
    res = vapia.build_distribution_X_vs_agg_Y(session = session, model =model_obj,  agg_by= 'day', agg_on= 'in
    vapip.plot_X_vs_Y(res, replicate_desc+'infections_per_day', 'day', '#infections')
    print vapi.scan(session, model_obj)
```

The function build_distribution_X_vs_agg_Y computes the aggregate on 'day' by summing across all the block infections. It returns the result dictionary with two keys: xValues and yValues. The function plot_X_vs_Y plots this epicurve.

To reiterate, if we pass the function as argument to visit_replicate as follows:

```
rv.visit_replicates(cfg_root=cfg_root, method=my_analysis_func, args=[session])
```

it would compute and plot epicurve for each of the replicates.

2.0.4 Analysis using data from all replicates