MNcwixsec

A Python tool for drawing well cross sections using Minnesota CWI data

Description

The purpose of this tool is to easily visualize well construction and local geologic layers in cross-section views. The tool is designed for use with the *Minnesota County Well Index* database (CWI) as the data source. The user provides a list of well identifiers and optional drawing directives to the tool, and the tool constructs a cross section view of the well or wells. The tool can be instructed to include only wells having required drawing information, and can be told which well components to include in the drawing.

The tool draws the individual wells in vertical cross section, and scales their height and width for visibility. The cross section line can be either a fence line or a projected line.

A fence line in plan view is a polyline passing through each well. The user may provide an approximate fence line path, or an approximate angle for the fence line to trend along. The program will adjust the fence line to pass exactly through each well, while trying to respect the approximate line or angle provided. If no line or angle is provided, it will try to have the line trend from left to right, or from south to north, while minimizing the total length of the poly-line. The spacing between wells in the cross-section view is proportional to the line segments between the wells, but that is adjusted to prevent the wells from overlapping in the cross-section view.

A projected line in plan view is a straight line, and normals are drawn to it from each well. The user may specify the exact location of the cross-section line by giving two points on the line, and the program will move the end points to the intersections of the first and last projection lines. Or the user may specify only the angle of the cross-section line, and the program will determine its location to pass throug the centroid of the well locations in plan view. If no line or angle is provided, it will try to find the angle most closely approximating the trend direction of the wells in plan view, and will pass through the centroid of the points. The spacing between wells in the cross-section view is proportional to the spacing between the lines of projection where they intersect the cross-section line. The lines of projection lines are generally normal to the cross-section line, except where they are adjusted so that the wells can be drawn in the cross-section view without overlapping.

If only one well is drawn, that is called a *singleton*. There is no line in plan view, and the program ignores any fence line or projected line directives.

The wells are drawn on the cross-section view with diameter and vertical dimensions scaled independently. The diameter is scaled to improve visibility of all the drawing components. The vertical dimension is scaled to fill the drawing area. At the present time, the only way for the user to adjust the scaling is by hard coding in the Python scripts.

The tool can depict any well component permitted by the command line parameters. These include:

- Casing(s)
- Screen(s)
- Drilled hole diameter(s)
- Grout
-

Perforation intervals
- Static water elevation
- First bedrock elevation
- Stratigraphic layers

Stratigraphic layers are drawn inside the well diameter, as if you are seeing a core taken from the well. The program does not attempt to interpolate or draw anything between wells along the cross-section line.

The tool does not currently draw angled holes correctly.

Getting Started

To use this tool, one needs to:

- Set up a Python environment
- Provide a data source (such as MNcwi `https://github.com/panibillo/MNcwi.git`)
- Have a way of selecting well identifiers of interest
- Choose your drawing control directives (command line options)
- Call the tool.

The command line interface is built using Python's argparse module. The argparse help describes how to call the tool and how to set the drawing parameters. A demo file is provided that shows examples of command lines, and demonstrates calling the tool from a Python script. When called from a script, the command line module is used to pre-process command line arguments into the arguments required by the xsec_main module.

It is presumed that the data source will be the Minnesota CWI database itself, or a clone of it having all of the "c4" data tables described on the MWI website:

CWI data tables

To connect to a different data source, see [l]

It is presumed that the method for selecting well identifiers will be GIS based, and that the user may implement methods to convert the graphics for display in a GIS. But the current version has no GIS links or dependencies, and produces only simple demonstration graphics with matplotlib.

Dependencies

- Python >= 3.7

Python libraries used include:

- argparse
- pysqlite
- numpy

- matplotlib
- A data source. The provided data module `xsec_data_MNcwi.py` is written to use a local SQLite clone of the CWI database data tables.
- A legend source. The provided legend module `xsec_legend.py` is written to use a local SQLite databases to define the legends. The provided legends are still quite limited, and are provided primarily for testing and demonstration. The main limitations are a very limited set of stratigraphic codes, and unaesthetic or non-intuitive depictions of some well components.

SQLite databases can be viewed or edited through Python, but a convenient way to view and edit them is with SQLiteStudio, available at

SQLiteStudio

This project has been tested on Windows 10 and with Python 3.7.

Installing

- Download the project files from GitHub. Open a git Bash window and navigate to the parent folder where you want to install it. Then on the Bash line enter:

```
git clone https://github.com/panibillo/MNcwixsec.git
```

That should create a folder named `MNcwixsec` with the project files inside of it.

- Alternatively download the zip file from Github and unzip
- You are reponsible for obtaining a data source. If you are able to get instructions and credentials for downloading the Minnesota County Well Index data files, then a suitable data source can be built in SQLite using project MNcwi:

```
github.com/panibillo/MNcwi.git
```

- Edit the script `xsec_data` to correctly point to your data source.
    - If the data source is an SQLite database created by MNcwi schema version 3, then the scripts should work as is.
    - Otherwise you will have to study to code to see how to connect to a different data source, how to identify the wells, and how to obtain the needed data values.
- Create legends for the drawing components. The demo drawing module uses matplotlib, and the legends for the drawing components are defined in an SQLite database named `xsec_Legend.sqlite`. The legends are designed only to make the components distinct to evaluate the demonstration; They are not particularly

aesthetic or logical. Only a handful of stratigraphic codes have been defined in the legend file.

- If you wish to use a Microsoft Access database, it appears that Python and MS Office must *both* be installed as 32-bit, or must *both* be installed as 64-bit.

Executing program

```
> python <your_path>MNcwixsec\src\xsec_demo.py
> python <your_path>MNcwixsec\src\xsec_main.py -p -i 195748 200828 200830 -a 45 -R M
```

- You can run the demonstration script, `xsec_demo.py` from the command line:

  python <your_path>\MNcwixsec\src\xsec_demo.py

- You can run the demonstration script, `xsec_demo.py` from within an IDE:

  ```
  from xsec_demo import run_demo run_demo()
  ```

- You can learn how to compose command lines by studying the demo and by getting the help from the command line or from the Python prompt. Of course you can also look at `xsec_cl.py`.

  python <your_path>\MNcwixsec\src\xsec_main.py -h

  ```
  from xsec_cl import xsec_parse_args xsec_parse_args('-h')
  xsec_parse_args(['-h',]) # alternative syntax
  ```

- You can use `xsec_demo.py` as a model to develop your own methods for composing a command line, and pass it to `xsec_main.py` from the command line, from within an IDE or from another script.

  python <your_path>MNcwixsec\src\xsec_main.py -p -i 195748 200828 200830 -a 45 -R M

  ```
  from xsec_main import xsec_Main xsec_Main('-p -i 195748
  200828 200830 -a 45 -R M')
  xsec_Main('-p -i 195748 200828 200830 -a 45 -R M'.split())
  # alternative syntax
  ```

Program outline and aspirations

- A data source. E.g. a database such as CWI or a local database.

  - o

Create a module xsec_data_ that inherits from xsec_data_abc. Use xsec_data_MNcwi.py as an example.

- The xsec_data module reads the source data into Python dictionaries whose keys are well identifiers, and whos values are either scalars (numeric or text values), or Python Dataclasses defined in xsec_data_abc. The xsec_data module(s) contains the knowledge of how to map the source data into those dictionaries, and how to fill in missing data if desired.
- In the graphic interface, the elements drawn in the cross section view could have arbitrary attributes attached to them, which could be read from the data source along with the information needed for drawing.
  The programmer can safely add new attibutes to the Dataclasses without breaking things.

- A drawing module.

  - The current graphic output implementation is in xsec_draw.py. It is a very simple application in matplotlib. If one wanted to stick with matplotlib, then one could have the draw module automatically save the plot to a file.
  - The current implementation does not draw anything in the map view other than the section line, dots for the wells, and in the case of a projected cross section line it draws the projection lines from the
    wells to the cross section line.

- Legends.

  - The current implementation reads legend information either hard-coded in the files xsec_legend.py, or from a database xsec_Legends.sqlite.
    The attributes lists in xsec_Legends.sqlite are only sufficient for the examples coded in xsec_test.py. The lithology attributes are
    particularly incomplete when compared to CWI, and the Aquifer attribute codes simply copy the lithology codes.

  - (The current implementation probably confuses lithology with some other concepts. So some name changing might be in order.)
  - The legend definitions have to know attributes that are in the source data, and the drawing instructions that are required by the graphics module that you use. The drawing instructions in matplotlib are some- times inconsistent between lines, points, and polygons, and the legends as implemented are not always perfect models.

-

A way to invoke the program.

- To invoke from the command line, run xsec_main.py with appropriate command line arguments. e.g: C:> python [path to project]\xsec_main.py -s -i 207269
- The command line options are describe in xsec_cl.py. To read the options from the command line, enter the help command -h: C:> python [path to project]\xsec_main.py -h
- To invoke xsec from another script, compose the command line options as a list of string values, pass the options to xsec_parse_args in xsec_cl.py, and then call Xsec_main with the commands object returned
  by xsec_parse_args. There is an example in xsec_testruns.py.

- Adjust the drawing scaling parameters.

  - An attempt was made to implement each drawing task with some arbitrary parameters to control the scaling of different drawing elements. These are typically passed as arguments with default values. In some cases the examples pass them from the calling routines, and in some cases they have been made available to set in the argument list of the receiving module - but they are not yet well explained and there is not yet an example of setting them.
  - Not too much effort has yet been spent on the scaling parameters.
    Optimal values probably depend on the drawing module, on the legends, on which well components are selected to be included, and on the number of wells included in the section. The parameters attempt to adjust for the number of wells included, but whether their design is sufficient for that is an open question.

- Handle special cases and incomplete or inconsistent data

  - In the face of incomplete data, or poorly entered data, the program should attempt to display as much data as possible, and or inform the user about what data are missing or appear to be incorrect. Missing data from one well or one component should not prevent the display of other wells or other components.

Authors

William Olsen

Version History

- Software version 2021-10-22

License

Acknowledgments

Much of the logic for this program was conceived and originally developed in the Dakota County Environmental Resources Department, Dakota County, Minnesota.

The CWI database and the MWI website have been developed and maintained by many remarkable people who are remarkably dedicated to preserving and sharing well data in Minnesota. These are maintained primarily by the Minnesota Geologic Survey and the Minnesota Department of Health, and with the help of numerous collaborators.

MWI map interface

MWI text search

CWI documentation