可視化プログラムについての説明やコードは全て、(<https://github.com/denshisujikken-theme-B/chouryu-keisan>) にある。以下はその中から抜粋した内容である。見やすさの観点からも上記URLにアクセスされたい。

プログラムファイル

* visualize.py (メイン-by宇埜 一平)
* getdata.py (2020process.csv からデータを取得する-by渡邉 健人)
* draw\_graph.py (電圧の時間変化をプロットする-by野々村 祐人)

依存関係

* visualize.py
  + - numpy
    - pygame
    - loadflow
    - getdata
      * + numpy
    - draw\_graph
      * + matplotlib
        + pygame-matplotlib

目次

0. pygame の概要

1. loadflow の動作

2. visualize.py の動作

3. getdata.py の動作

4. draw\_graph.py の動作

**0. pygame** の概要

pygame は python を用いたゲーム制作用のモジュールである。ゲームオブジェクトである`Surface`オブジェクトとその位置を表す`Rect`オブジェクトを用いて、毎フレームごとに画面を描画する。`while` ループ一回で1フレームである。

|  |
| --- |
| pygame の初期化 |
| スクリーンの初期化 |
|  |
| Surface オブジェクトを用意 |
| Rect オブジェクトを設定 |
|  |
| while True: |
| イベントを取得、処理 |
| Surface の描画 |
| 更新 |

骨格

**1. loadflow** の動作

loadflow は潮流計算を効率的に行うために自作したモジュールである。コードはこちらへ。(https://github.com/peibra/loadflow)

ここでは簡単な例を用いて動作を追う。

ノードが0番から2番までの3つであり、0-1-2と直線的に繋がっているとする。使い方は以下の通りである。以下の手順で計算は完了する。レジスタンスなどの行列は自動で対称行列になる。

|  |
| --- |
| from loadflow import PowerSystem, LoadFlow |
|  |
| ps = PowerSystem(3) # ノード3つを初期化 |
|  |
| ps.r[0, 1] = 0.01 # ノード0と1の間のレジスタンス |
| ps.r[1, 2] = 0.04 |
|  |
| ps.x[0, 1] = 0.05 # リアクタンス |
| ps.x[1, 2] = 0.025 |
|  |
| ps.b[0, 1] = 0.04 # サセプタンス |
| ps.b[1, 2] = 0.02 |
|  |
| ps.bc[1] = 0.1 # ノード1につくブランチ以外のサセプタンス |
| ps.bc[2] = 0.1 |
|  |
| ps.P[1] = - 0.6 # ノード1の消費される有効電力 |
| ps.Q[1] = - 0.3 # 無効電力 |
| ps.P[2] = - 0.6 |
| ps.Q[2] = - 0.3 |
|  |
| ps.V[0] = 1. # ノード0の電圧 |
| ps.theta[0] = 0. # 位相 |
|  |
| lf = LoadFlow(ps) # 潮流計算の準備 |
| lf.calculate() # 潮流計算を実行 |

計算結果は `lf.V` などとすることで取得できる。注意が必要なのは `ps.V` などは初期設定であり、`lf.V` などは計算結果であることである。計算に失敗した場合、全てに関して `None` を返す。

|  |
| --- |
| print(lf.V) # 電圧の大きさ |
| >>> [1.0, 0.9651997474478232, 0.9340151566375318] |
|  |
| print(lf.theta) # 電圧の位相 |
| >>> [0.0, -0.05906500726151168, -0.06665066281793608] |
|  |
| print(lf.P) # 有効電力 |
| >>> [[ 0. 1.23602053 0. ]  [-1.21841535 0. 0.61841531]  [ 0. -0.59999996 0. ]] |
|  |
| print(lf.Q) # 無効電力 |
| >>> [[ 0. 0.50246384 0. ]  [-0.37580572 -0. 0.22486348]  [ 0. -0.19531394 -0. ]] |
|  |
| # lf.power は複素電力 P + jQ を返す |

これを用いると、全ての設定をそのままに、`ps.P` などだけを更新して `lf.calculate()` を行うのを繰り返せば、異なる消費電力に対しての計算結果が効率的に、容易に得られる。

以下では実際に内部でどのような動作をしているのかを見ていく。ただし、レジスタンスなどの行列を自動で対称行列にするテクニックなどの技巧的なことはここでは説明しない。そのような説明は(https://github.com/denshisujikken-theme-B/chouryu-keisan/blob/main/LOADFLOW\_DETAILS.md) を参照。

**PowerSystem** クラス

初期化

`ps = PowerSystem(n)` のようにこのクラスのインスタンスを作成すると、`r, x, b, bc, P, Q, V, theta` とノード数 `n` が適切に初期化される。

メソッド

\_ から始まるものは外部からのアクセスを意図しない内部メソッドである。

`\_initialize\_vec` は既知の`ps.P`、`ps.Q` を含むベクトル `\_p` と、未知の `V`、`theta` の値を含むベクトル `\_v` を作成する。また、値だけではどの変数に対する値か分からなくなるため、これらのベクトルが持つ情報を格納した配列 `\_p\_content`、`\_v\_content` も同時に作成する。`\_p`、`\_v` の大きさは等しい。

`\_Y\_diag` と `\_Y\_nondiag` はそれぞれノードアドミタンス行列の対角成分と非対角成分を計算する。

`Y` は、ノードアドミタンス行列を返す。これは上記の内部メソッドを使って計算したものを返す。

**LoadFlow** クラス

初期化

`lf = LoadFlow(ps)` とすると、`lf` が参照する `PowerSystem` が `ps` に初期化される。

メソッド

`calculate` はニュートンラプソン法を用いて潮流計算をするための本体である。働きは以下のようになる。

1. `PowerSystem.Y` を呼び出す。

2. `V, theta` をニュートンラプソン法の初期値として初期化する。

3. `p, v` を初期化。

4. `p, v` と同じ大きさの、`fnc\_v` を初期化。これは p = f(v) を満たすような f(v) を表すベクトルの候補である。

5. `p - fnc\_v` の無限大ノルムが 0.001 以下になるまで次の処理を繰り返す。ただし、ループ回数が50回に達すれば失敗と見なして計算結果を `None` とする。

1. `V, theta` を使って `fnc\_v` の成分を計算

2. `V, theta, Y` からヤコビアン `∂f/∂v` の成分を計算

3. ヤコビアンを作成

4. `fnc\_v` を作成

5. `v = v + (∂f/∂v)^(-1) \* (p-fnc\_v)` より `v` を更新。ただし、 \* は行列の積である。

6. `v` の値を `V, theta` に代入

6. 計算に成功した場合、`V` と `PowerSystem` のパラメータを使い、電力 `power, P, Q` を計算する。計算に失敗した場合、電力は `None` を返す。

**2. visualize.py** の動作

visualize.py は pygame を用いた潮流計算の可視化シミュレータのメインのプログラムである。この項では実装上の細かい点には触れずにプログラムの全体像を俯瞰する。詳細は実際のコードを参照されたい。十分なコメントを載せてある。

プログラムの制御には基本的にはフラグを使っている。「0. pygame の概要」に書いたような骨格になっており、発生するイベントによりフラグを更新し、その下の描画部分ではフラグを元に何を描画するかを決めている。

コントロール機能

* 日付の設定
* 時間の設定
* ２つのノード間の電力需要の割合を設定
* 時間変化のコントロール
  + - 再生と停止
    - ループ
    - 1日分の再生時間の設定
* 電圧のグラフの表示

日付の設定

本プログラムでは2020年度の沖縄電力の電力需給データから8日を選んで使用している。その8つには0から7までのidが振ってあり、プログラム中では一貫してそれらのidで日付を指定、参照している。現在の日付は左上に表示される。そこをクリックするとカレンダーが表示され、クリックした日付が選択される。左上のボタン、カレンダー内の各日付はそれぞれ pygame の Surface オブジェクトになっており、それぞれとマウスの当たり判定により、イベントが発生する。

時間の設定

本プログラムにおける時間の変数 time はプログラム内で、あらゆる動作の基準になるパラメータである。例えば、時間を表すバーをクリックすることによって時間を変更できるが、クリック位置から直接指針の位置を決めるのではなく、クリック位置から時間を決定し、時間から指針の位置を決定する。これは矛盾が起きないようにするためである。プログラム内で、分の単位まで時刻が表示されるようになっているが、使用しているデータは１時間ごとのものである。これは時刻をさまざまなパラメータとして用いているために、１時間ごとの時間変化では視覚的に満足の行く効果が得られないからである。例えば、電力の流れを表す矢印の脈動は時刻の分の成分を使っている。また、空の色も時刻から算出している。しかし、毎フレーム潮流計算をするのは計算資源の無駄だから、時刻の時間成分が変化したときにのみ潮流計算されるようになっている。

２つのノード間の電力需要の割合を設定

沖縄電力のデータからは総需要しか得られないが、それを2つのノードに割り振る時の割合によって計算結果がどう変化するのかを知ることができるようにした。中央上部のスライドを変化させることで、設定することができる。各ノードが消費している電力をゲージに示してわかりやすくした。

時間変化のコントロール

中央下部のコントロールパネルで時間変化の設定ができる。

* 再生と停止

時間の更新を開始、停止することができる。

* ループ

時間が24時以上になると、０時に戻るが、右側のボタンでそれと同時に再生するか、停止するかを選択できる。

* **1**日分の再生時間**(**周期**)**の設定

左側のスライドで1日分のシミュレーションを何秒で再生するかを選択することができる。つまり、時間の変数の更新は、`time += 1/period` となる。ただし、`period` は1日分の周期である。これは本プログラムのフレームレートが24Hzだからである。

電圧のグラフの表示

`while` ループの中で１日の間に24回の潮流計算が行われるのであるが、その24回分の各ノードの電圧が計算のたびに配列に保存される(4x24の２次元配列)。そして、23時になると、データがプロットされ、Surface として配列に保存される。ただし、最新の4つのプロット以外は消去される。また、データを保存する際に手動での時間変化または日付の変化を行った場合はグラフはプロットされない。一度プロットした後に日付や需要の割合を触らずに二度目の実行をしてもプロットされない。つまり、過去4回の異なる条件でのシミュレーションにおける各ノードの電圧の時間変化を表示するというわけである。

**while** 文の内容

プログラムのメインの部分である `while` 文の内容についての流れを見ていく。後に描画したものは先に描画したものの上に被さることに注意。

|  |
| --- |
| while True |
|  |
| イベントループ |
|  |
| 背景描画 |
| ノード間のブランチとして線を描画 |
|  |
| 日付、時刻から該当データを取得し、潮流計算をする |
| 計算を元に電圧の指針を設定 |
| 計算を元にブランチの有効電力を示す棒を設定 |
|  |
| ブランチの有効電力を示す棒を描画 |
| 各ノードを描画 |
| 電圧の指針を描画 |
|  |
| グラフ表示ボタンを描画 |
| 時刻を示すバーを描画 |
| 時刻の数値を描画 |
|  |
| 負荷ノードの有効電力を示すゲージを設定、描画 |
|  |
| 日付の表示を描画 |
| if カレンダーが表示されている: |
| カレンダーを描画 |
|  |
| コントロールパネルを描画 |
| 再生または停止ボタンを描画 |
| 周期設定用のスライド描画 |
| ループがアクティブかに応じて描画 |
|  |
| 電力需要の割合変化スライドの描画 |
|  |
| if グラフが表示: |
| グラフ背景描画 |
| グラフ描画 |
|  |
| if 再生中: |
| 時刻　+= 1/周期 |
|  |
| if 時刻 >= 24: |
| 時刻 = 0 |
| if ループが非アクティブ: |
| 再生を停止 |
|  |
| if このフレームで時刻の時間成分または日付が変更された: |
| 計算済み = False |
|  |
| フレームを更新 |

イベントループの内容は以下である。

|  |
| --- |
| for イベント: |
| if マウスが押された: |
| if グラフが非表示: |
|  |
| if 日付表示が押された: |
| カレンダーの表示/非表示を切り替える |
|  |
| if カレンダーが表示されている: |
| if 日付が押された: |
| 日付 = 押された日付 |
| 時刻 = 0 |
| 電力需要の割合 = デフォルト値 |
| 周期 = デフォルト値 |
| グラフ描画済み = False |
|  |
| if グラフ表示ボタンが押された: |
| グラフを表示する |
| 再生を停止する |
|  |
| if 時刻バーが押された: |
| 時刻の値を更新する |
| 時刻をドラッグ開始する |
| グラフデータの取得を取り消しする |
|  |
| if 停止中: |
| if 再生ボタンが押された: |
| 再生を開始する |
| if 計算失敗: |
| 時刻 = 0 にしてやり直す |
| elif 停止ボタンが押された: |
| 再生を停止する |
| 時刻の分をその時間の30分に設定する |
| if ループボタンが押された: |
| ループのアクティブ/非アクティブを切り替える |
|  |
| if 電力需要の割合変化スライドが押された: |
| 割合変化を開始する |
| グラフ描画済み = False |
|  |
| if 周期の変化スライドが押された: |
| 周期変化を開始する |
|  |
| else: # グラフが表示されている |
| グラフを非表示にする |
|  |
| if マウスが動いた: |
|  |
| if 時刻のドラッグが開始されている: |
| 時刻の値を更新する |
|  |
| if 需要の割合変化が開始されている: |
| 割合の値を更新する |
|  |
| if 周期変化が開始されている: |
| 周期の値を更新する |
|  |
| if マウスが放された: |
|  |
| 時刻の値の更新を停止する |
| 割合の値の更新を停止する |
| 周期の値の更新を停止する |

**3. getdata.py** の動作

あらかじめ整型してある 2020process.csv の行と列の添字と、日付、時刻の情報を対応づけて、`[電力需要, 太陽光発電量]`返す関数 `getdata` を含む。

**4. draw\_graph.py** の動作

24時間分の電圧のデータを4つのノードに対してプロットし、グラフを作成する関数 `draw\_graph` を含む。ただし、`matplotlib` を用いており、描画したグラフを `pygame.Surface` オブジェクトとして出力するバックエンド `pygame-matplotlib` を用いている。関数は `pygame.Surface` を返す。