FIPA Algorithm (Python Code)

FIPA is executing regarding SPV penetration fluctuation rate for every point of 1-minute temporal resolution and varies with load demand. In this investigation, LSSPV is connected to the IEEE 39 bus system via bus 32 for initial condition. In conclusion FIPA used to move the whole network including load demand regarding to intermittency LSSPV. With this proposed technique, the adaptable and resilient infrastructure can be improved especially in high variability of RES. Besides, this approach will enable utility providers, distributed generator teams, and financial departments to assess and address issues related to electricity prices and the quality of supply in the solar photovoltaic (SPV) sector. The integration of renewable energy into the generation and distribution system can achieve sustainability goals and address climate change.

For detailed information please email me: mashitah1116@gmail.com

Please cite any of this paper for credit:

1. Title: An innovative FIPA computerization for intermittency LSSPV generation reconfiguration

   Author: mashitah mohd hussain

2. Title: Investigation of Critical Time Analysis Considering Shunt Compensation Interconnecting WECC SPV Model

   Author: mashitah mohd hussain

3. Title: Short Term Forecasting of Electrical Consumption Using A NeuralNetwork: Joint Approximate Diagonal Eigenvalue

   Author: mashitah mohd hussain

```
################################################################################
# Locate PSSE Directory
#--------------------------------------------------------------------------------------------------
def latest_psse_location():
  import _winreg
  ptiloc = r'SOFTWARE\PTI'
  lver = 'PSS\xaeE 32'
  lverloc = ptiloc + '\\' + lver + '\\' + 'Product Paths'
  lverkey = _winreg.OpenKey(_winreg.HKEY_LOCAL_MACHINE, lverloc, 0,
_winreg.KEY_READ)
  lverdir, stype = _winreg.QueryValueEx(lverkey, 'PsseInstallPath')
  _winreg.CloseKey(lverkey)
```

```
  return lverdir

#--------------------------------------------------------------------------------------------------
# Array Initialisation
#--------------------------------------------------------------------------------------------------
solv = [None,
       'Iteration limit exceeded',
       'Blown up (only when non-divergent option disabled)',
       'Terminated by non-divergent option',
       'Terminated by console interrupt',
       'Singular Jacobian matrix or voltage of 0.0 detected',
       'Inertial power flow dispatch error (INLF)'
       'OPF solution met convergence tolerance (NOPF)',
       'Solution not attempted']

#--------------------------------------------------------------------------------------------------
# PSSE Initialization
#--------------------------------------------------------------------------------------------------
from math import sqrt
import os,sys
sys.path.append(p)
os.environ['PATH'] = p + ';' + os.environ['PATH']
if p.find('32') < 0:
  q = psp  + r'\PSSPY27'
  sys.path.append(q)
  os.environ['PATH'] = q + ';' + os.environ['PATH']
import psspy, redirect
_i,_f=psspy.getbatdefaults()
_s=psspy.getdefaultchar()
redirect.psse2py()
ierr = psspy.psseinit(12000)

#--------------------------------------------------------------------------------------------------
# Get Generation from PSSE
#--------------------------------------------------------------------------------------------------
ierr, xgnbn = psspy.amachint(-1, 4, 'NUMBER')
ierr, xgnmw = psspy.amachreal(-1, 4, 'PGEN')
gnbw = [[],[]]
for i in range(len(xgnbn[0])):
  gnbw[0].append(xgnbn[0][i])
for i in range(len(xgnmw[0])):
  gnbw[1].append(xgnmw[0][i])
del(xgnbn)
del(xgnmw)
```

```
ierr = psspy.close_powerflow()

# PSSE Execution
#----------------------------------------------------------------------------------------------
print 'Get Location Start - ' + datetime.datetime.now().strftime('%d %b %Y %H:%M:%S')
extnam = '_' + datetime.datetime.now().strftime('%Y%m%d%H%M%S')
prgf = xlp + '\\TSGetLocByBusProgress.log'
repf = xlp + '\\TSGetLocByBusReport.log'
alrf = xlp + '\\TSGetLocByBusAlert.log'
prmf = xlp + '\\TSGetLocByBusPrompt.log'
if os.path.exists(prgf): os.remove(prgf)
if os.path.exists(repf): os.remove(repf)
if os.path.exists(alrf): os.remove(alrf)
if os.path.exists(prmf): os.remove(prmf)
ierr = psspy.lines_per_page_one_device(1,100000)
ierr = psspy.progress_output(2,prgf,[0,0])
ierr = psspy.report_output(2,repf,[0,0])
ierr = psspy.alert_output(2,alrf,[0,0])
ierr = psspy.prompt_output(2,prmf,[0,0])

# ----------------------------------------------------------------------------------------------
   # Branch Data
   # Branch Data - Integer
   istrings =
['fromnumber','tonumber','status','nmeternumber','owners','own1','own2','own3','own4']
   ierr, idata = psspy.abrnint(sid, owner_brn, ties_brn, flag_brn,1, istrings)
   if ierr:
      print 'psspy.abrnint error = %d' % ierr
      psspy.pssehalt_2()
   ierr, tmpdat = psspy.atrnint(sid, owner_brn, ties_brn, flag_brn,1, istrings)
   if ierr:
      print 'psspy.atrnint error = %d' % ierr
      psspy.pssehalt_2()
   for ii in range(len(idata)):
    for jj in range(len(tmpdat[ii])):
      idata[ii].append(tmpdat[ii][jj])
   ibranch = array2dict(istrings, idata)
   # Branch Data - Real
   rstrings = ['amps','pucur','pctrate','pctratea','pctrateb','pctratec','pctmvarate',
            'pctmvaratea','pctmvarateb',#'pctmvaratec',
            'fract1','fract2','fract3','fract4','rate','ratea','rateb','ratec',
            'p','q','mva','ploss','qloss',
            'o_p','o_q','o_mva','o_ploss','o_qloss'
            ]
```

```
ierr, rdata = psspy.abrnreal(sid, owner_brn, ties_brn, flag_brn,1, rstrings)
if ierr:
   print 'psspy.abrnreal error = %d' % ierr
   psspy.pssehalt_2()
ierr, tmpdat = psspy.atrnreal(sid, owner_brn, ties_brn, flag_brn,1, rstrings)
if ierr:
   print 'psspy.atrnreal error = %d' % ierr
   psspy.pssehalt_2()
for ii in range(len(rdata)):
 for jj in range(len(tmpdat[ii])):
   rdata[ii].append(tmpdat[ii][jj])
rbranch = array2dict(rstrings, rdata)
# Branch Data - Complex
xstrings = ['rx','pq','pqloss','o_pq','o_pqloss']
ierr, xdata = psspy.abrncplx(sid, owner_brn, ties_brn, flag_brn,1, xstrings)
if ierr:
   print 'psspy.abrncplx error = %d' % ierr
   psspy.pssehalt_2()
tstrings = ['rxact','pq','pqloss','o_pq','o_pqloss']
ierr, tmpdat = psspy.atrncplx(sid, owner_brn, ties_brn, flag_brn,1, tstrings)
if ierr:
   print 'psspy.atrncplx error = %d' % ierr
   psspy.pssehalt_2()
for ii in range(len(xdata)):
 for jj in range(len(tmpdat[ii])):
   xdata[ii].append(tmpdat[ii][jj])
xbranch = array2dict(xstrings, xdata)
```