Intro

The purpose of this document is to record changes of note made to PST over the course of the SETO work that may be worth not forgetting about. It should be noted that PST SETO is based on PST version 3.

PSS model

There was a correction to the washout time constant in the PSS model between PST version 2.x and 3. To accommodate for this, the SETO version has two pss files named `pss2` and `pss3` which mimic the computation of each PST version respectively. The idea is to enable a user to specify which model the pss settings use in a particular case. The current usage is similar to:

```
copyfile([PSTpath 'pss2.m'],[PSTpath 'pss.m']); % use version 2 model of PSS
```

Alternatively, a `pssGainFix` variable may be set to 1, or true, which will adjust the version 2 data from a `d_` file to work the same way with the version 3 model. This is accomplished by executing: `pss_con(:,3) = pss_con(:,3)./pss_con(:,4);` While this works, it's kind of confusing and may be removed.

Sub-transient Machine model

There are three versions of the `mac_sub` model available. The `_ORIG` model is the standard PST model based on the R. P. Schulz, "Synchronous machine modeling" algorithm. The `_NEW` model is based on the PSLF 'genrou' model by John Undrill. The `_NEW2` model is the same as the `_NEW` model with alternative calculations for .... Any model may be copied over the `mac_sub` file for use.

exc_dc12

In 2015 there were 'errors' corrected in the saturation block that create differences between version 2 and 3 of this model. Effects are noticeable, but a solution hasn't been investigated yet.

exc_st3

Corrected theta index to `n_bus` from `n` per Ryan Elliott. Corrected simple `*` to `.*` int the `if ~isempty(nst3_sub)` section.

mac_tra

Commented out code that prevented the setting equal of the transient reactances.

lmod and rlmod

Fixed state limiting. While if over-limit, the derivative was set to zero, the state was not set to the maximum/minimum value correctly.

pwrmod

This is the power or current injection model Dan created for version 2.3. It's meant to model the 'grid following' type of converters. It is included in both the non-linear and linear simulation modes of PST SETO.

ivmmod

This is the voltage behind an impedance model Dan created. It's meant to model a 'grid forming' converter where voltage and angle are manipulatable. While there are questions about the reality of such operations, the model exists and appears to work in the non-linear simulation of PST SETO.

livePlot

Function that creates plot during non-linear simulation. While this operation existed in previous versions of pst, it is now functionalized to allow users to more easily define what is displayed (if anything) during simulations. The `livePlot` function is designed to be overwritten in a similar fashion as pss or machine models previously described.

lmon

Coded to actually monitor line current and power flow during non-linear simulation. The `lmon_indx` function creates the required data structures and indices.

area

Created area functionality via the `area_def`. The `area_indx` function creates the required data structures and indices.

calcAveF

Calculates system and area weighted average frequency. System values stored in `g.sys.aveF` and area values stored in `g.area.area(x).aveF`.

calcAreaVals

Calculates total area generation and interchange. Intentions were to also sum total load, but there were complications with that and work in that direction seemed not entirely useful.

agc

Automatic generation control model that calculates RACE and distributes signal to defined controlled generators according to participation factor. The ACE signal is filtered through a PI before being distributed to each generator through a unique low pass filter that adds the negative of the value to the governor `tg_sig` value. (NOTE: the `tg_sig` value is equivalent to an addition to the governors P_{ref} value) The `agc_indx` function creates the required data structures and indices.

cleanZeros

Function that cleans all zero variables from the global g. Executed at the end of `s_simu_Batch`. Stores cleared variable names in the `clearedVars` cell that is stored in `g.sys`

Global Variables

To enable easier manipulation of PST - it was decided to create a global structure that contains all system globals. While this may or may not have been a good idea - it happened. Initial results show a speed up of over 2 times. In other words, it could be assumed previous versions of PST spend half of their computation time loading globals...

Inside the global variable `g` are fields that corresponds to models, or groups, of other globals. For example, the `g.mac.mac_spd` global contains a all machine speeds while the `g.bus.bus_v` contains all bus voltages, etc. As of this writing, compiled on July 23, 2020, the following subparagraphs describe the globals contained in each field of the global `g`.

lmod

```
global lmod_con % defined by user
global n_lmod lmod_idx % initialized and created in lm_indx
global lmod_sig lmod_st dlmod_st % initialized in s_simu
global lmod_pot  % created/initialized in lmod.m
global lmod_data % added by Trudnowski - doesn't appear to be used?
```

tg

```
%% turbine-governor variables
global  tg_con tg_pot
global  tg1 tg2 tg3 tg4 tg5 dtg1 dtg2 dtg3 dtg4 dtg5
global  tg_idx  n_tg tg_sig tgh_idx n_tgh
```

It should be noted that the hydro governor model `tgh` has not been modified as no examples seemed to use it.

rlmod

```
global  rlmod_con n_rlmod rlmod_idx
global  rlmod_pot rlmod_st drlmod_st
global  rlmod_sig
```

exc

```
global  exc_con exc_pot n_exc
global  Efd V_R V_A V_As R_f V_FB V_TR V_B
global  dEfd dV_R dV_As dR_f dV_TR
global  exc_sig % pm_sig n_pm % not related to exciters?
global smp_idx n_smp dc_idx n_dc  dc2_idx n_dc2 st3_idx n_st3;
global smppi_idx n_smppi smppi_TR smppi_TR_idx smppi_no_TR_idx ;
```

```
global smp_TA smp_TA_idx smp_noTA_idx smp_TB smp_TB_idx smp_noTB_idx;
global smp_TR smp_TR_idx smp_no_TR_idx ;
global dc_TA dc_TA_idx dc_noTR_idx dc_TB dc_TB_idx dc_noTB_idx;
global dc_TE  dc_TE_idx dc_noTE_idx;
global dc_TF dc_TF_idx dc_TR dc_TR_idx
global st3_TA st3_TA_idx st3_noTA_idx st3_TB st3_TB_idx st3_noTB_idx;
global st3_TR st3_TR_idx st3_noTR_idx;
```

mac

```
global  mac_con mac_pot mac_int ibus_con
global  mac_ang mac_spd eqprime edprime psikd psikq
global  curd curq curdg curqg fldcur
global  psidpp psiqpp vex eterm ed eq
global  pmech pelect qelect
global  dmac_ang dmac_spd deqprime dedprime dpsikd dpsikq
global  n_mac n_em n_tra n_sub n_ib
global  mac_em_idx mac_tra_idx mac_sub_idx mac_ib_idx not_ib_idx
global  mac_ib_em mac_ib_tra mac_ib_sub n_ib_em n_ib_tra n_ib_sub
global pm_sig n_pm
global  psi_re psi_im cur_re cur_im

% added
global mac_trip_flags
global mac_trip_states
```

pss

```
global  pss_con pss_pot pss_mb_idx pss_exc_idx
global  pss1 pss2 pss3 dpss1 dpss2 dpss3 pss_out
global  pss_idx n_pss pss_sp_idx pss_p_idx;
global  pss_T  pss_T2 pss_T4 pss_T4_idx
global pss_noT4_idx % misspelled in pss_indx as pss_noT4
```

Despite the renaming of the `pss_noT4_idx`, it doesn't seem to actually be used anywhere.

pwr

```
global  pwrmod_con n_pwrmod pwrmod_idx
global  pwrmod_p_st dpwrmod_p_st
global  pwrmod_q_st dpwrmod_q_st
```

```
global  pwrmod_p_sig pwrmod_q_sig
global  pwrmod_data
```

There are some cells that contain user defined derivatives that aren't included yet.

ncl

```
global  load_con load_pot nload
```

sys

```
global  basmva basrad syn_ref mach_ref sys_freq
```

svc

```
global  svc_con n_svc svc_idx svc_pot svcll_idx
global  svc_sig
% svc user defined damping controls
global n_dcud dcud_idx svc_dsig
global svc_dc % user damping controls?
global dxsvc_dc xsvc_dc
%states
global B_cv B_con
%dstates
global dB_cv dB_con
```

There seems to be some code related to user defined damping control of SVC, but it is not described in the user manual. (Added by Graham around 98/99)

tcsc

```
global  tcsc_con n_tcsc tcsvf_idx tcsct_idx
global  B_tcsc dB_tcsc
global  tcsc_sig tcsc_dsig
global  n_tcscud dtcscud_idx  %user defined damping controls
% previous non-globals added as they seem to relavant
global xtcsc_dc dxtcsc_dc td_sig tcscf_idx
global tcsc_dc
```

Similar to the SVC, there seems to be some added functionality for controlled damping, but no examples exist? (Added by Graham around 98/99)

igen

```
%% induction genertaor variables - 19
global  tmig  pig qig vdig vqig  idig iqig igen_con igen_pot
global  igen_int igbus n_ig
%states
global  vdpig vqpig slig
%dstates
global dvdpig dvqpig dslig
% added globals
global s_igen
```

ind

```
%% induction motor variables - 21
global  tload t_init p_mot q_mot vdmot vqmot  idmot iqmot ind_con ind_pot
global  motbus ind_int mld_con n_mot t_mot
% states
global  vdp vqp slip
% dstates
global dvdp dvqp dslip
% added globals
global s_mot
global sat_idx dbc_idx db_idx % has to do with version 2 of mac_ind
% changed all pmot to p_mot (mac_ind1 only)
```

Two models of this are included as `mac_ind1` (a basic version from 2.3), and `mac_ind2` which is an updated induction motor model. Default behavior is to use the newer model (`mac_ind2`).

dc

```
%% HVDC link variables
global  dcsp_con  dcl_con  dcc_con
global  r_idx  i_idx n_dcl  n_conv  ac_bus rec_ac_bus  inv_ac_bus
global  inv_ac_line  rec_ac_line ac_line dcli_idx
global  tap tapr tapi tmax tmin tstep tmaxr tmaxi tminr tmini tstepr tstepi
global  Vdc  i_dc P_dc i_dcinj dc_pot alpha gamma
global  VHT dc_sig  cur_ord dcr_dsig dci_dsig
global  ric_idx  rpc_idx Vdc_ref dcc_pot
global  no_cap_idx  cap_idx  no_ind_idx  l_no_cap  l_cap
global  ndcr_ud ndci_ud dcrud_idx dciud_idx dcrd_sig dcid_sig
```

```
% States
%line
global i_dcr i_dci  v_dcc
global di_dcr  di_dci  dv_dcc
global dc_dsig % added 07/13/20 -thad
%rectifier
global v_conr dv_conr
%inverter
global v_coni dv_coni

% added to global dc
global xdcr_dc dxdcr_dc xdci_dc dxdci_dc angdcr angdci t_dc
global dcr_dc dci_dc % damping control
global  ldc_idx
global rec_par inv_par line_par
```

Some DC related functions reused global variable names for local values but avoided conflict by not importing the specific globals. During global conversion this caused some issues with accidental casting to global and overwriting issues. While the non-linear and linear simulations run, there may be issues with this problem yet to be discovered.

bus

Contains `bus` and all altered bus arrays associated with faults created in `y_switch`. Also contains the `bus_v` and `theta` information related to buses.

line

Contains `line` and all altered line arrays associated with faults created in `y_switch`.

int

Contains reduced Y matrices, voltage recovery matrcies, and bus order variables created in `y_switch` associated with faults. Example variables are shown below.

```
>> g.int
ans =
      Y_gprf: [4x4 double]
    Y_gncprf: [4x3 double]
    Y_ncgprf: [3x4 double]
     Y_ncprf: [3x3 double]
     V_rgprf: [10x4 double]
```

```
    V_rncprf: [10x3 double]
       boprf: [13x1 double]
        Y_gf: [4x4 double]
      Y_gncf: [4x3 double]
      Y_ncgf: [3x4 double]
       Y_ncf: [3x3 double]
       V_rgf: [10x4 double]
      V_rncf: [10x3 double]
         bof: [13x1 double]
      Y_gpf1: [4x4 double]
    Y_gncpf1: [4x3 double]
    Y_ncgpf1: [3x4 double]
     Y_ncpf1: [3x3 double]
     V_rgpf1: [10x4 double]
    V_rncpf1: [10x3 double]
       bopf1: [13x1 double]
      Y_gpf2: [4x4 double]
    Y_gncpf2: [4x3 double]
    Y_ncgpf2: [3x4 double]
     Y_ncpf2: [3x3 double]
     V_rgpf2: [10x4 double]
    V_rncpf2: [10x3 double]
       bopf2: [13x1 double]
```

lmon

This contains variables for line monitoring not previously collected during simulation. An example of lmon contents is shown below.

```
>> g.lmon
ans =
     lmon_con: [3 10]
       n_lmon: 2
    busFromTo: [2x2 double]
         line: [1x2 struct]
 >> g.lmon.line(1)
 ans =
     busArrayNdx: 3
         FromBus: 3
           ToBus: 4
           iFrom: [1x8751 double]
```

```
  iTo: [1x8751 double]
sFrom: [1x8751 double]
  sTo: [1x8751 double]
```

area

This contains variables for area calculation and operation. An example of area variables are shown below.

```
>> g.area
ans =
    area_def: [13x2 double]
      n_area: 2
        area: [1x2 struct]
>> g.area.area(1)
ans =
           number: 1
        areaBuses: [7x1 double]
           macBus: [2x1 double]
        macBusNdx: [1 2]
      maxCapacity: 1800
          loadBus: [4x1 double]
       loadBusNdx: [3 4 5 10]
           genBus: [3x1 double]
        genBusNdx: [1 2 11]
             totH: [1x8751 double]
             aveF: [1x8751 double]
           totGen: [1x8751 double]
          totLoad: []
              icA: [1x8751 double]
              icS: [1x8751 double]
    exportLineNdx: []
    importLineNdx: [11 12]
         n_export: []
         n_import: 2
```

agc

This contains variables for agc calculation and operation. An example of AGC variables are shown below

```
>> g.agc
ans =
      agc: [1x2 struct]
    n_agc: 2
>> g.agc.agc(1)
ans =
              area: 1
         startTime: 25
        actionTime: 15
              gain: 2
             Kiace: 3
             Btype: 1
                 B: 1
               kBv: []
           condAce: []
                Kp: 0.0400
                 a: 1.0000e-03
       ctrlGen_con: [2x3 double]
         n_ctrlGen: 2
         macBusNdx: [1 2]
             tgNdx: [1 2]
           ctrlGen: [1x2 struct]
              race: [1x8751 double]
            d_sace: [1x8751 double]
              sace: [1x8751 double]
          ace2dist: [1x8751 double]
              iace: []
            aceSig: -0.0144
      iaceStartNdx: 8439
             iaceN: 624
             Bcalc: 18
    nextActionTime: 130
```

k

To allow for functionalized running, various index values were placed into the global

```
global k_inc h ks h_sol
golbal k_incdc h_dc
```