function [matData, data] = fx\_getMats(FileNames)

% fx\_getMats(FileNames)

% Written by J. Jonkman, NREL

% 19-July-2016: Updated by B. Jonkman (NREL) to convert FAST v8.16

% linearization files into format expected by mbc3.m

% 22-Jan-2018: Updated by B. Jonkman, (Envision Energy) for BeamDyn linearization

% converted to a function with data types.

% 7-Feb-2019: Updated by B. Jonkman (Envision) & Nick Johnson (NREL) for

% first-order states

% This m-file is used to read in the data written to multiple FAST linear output

% (.lin) files, compute the state matrix, [A], at each of the equally-spaced

% azimuth steps and their azimuth-average, along with their eigenvalues and

% eigenvectors.

%

% inputs:

% FileNames - cell string containing names of FAST linear output files

% outputs:

% matData - structure containing computations of FAST linear data.

% data - raw data read from the FAST linearization files.

%

% ASSUMPTIONS:

% - all files in FileNames contain the same data structures (i.e., they

% have the same numbers of inputs, outputs, and states; if a state matrix

% exists in one, it exists in all)

% - all files in FileNames have the same rotor speed, but have different

% azimuth angles

% - BeamDyn blade nodes are discretized in the same way for each blade.

% - Modules with second-order states store them as all the x values

% followed by all the x\_dot values (in the same order)

% - descriptions of inputs, outputs, and (non-ElastoDyn) states are triplets

% if they match in all characters except the blade number. (see

% findBladeTriplets.m for details)

if nargin < 1 || isempty(FileNames)

% FileNames = {'5MW\_Land\_BD\_Linear.1.lin','5MW\_Land\_BD\_Linear.2.lin'};

d=dir('\*.lin');

for ix=1:length(d)

FileNames{ix}=d(ix).name;

end

elseif ~iscell(FileNames)

FileNames = {FileNames}; % convert (hopefully a) single string to cell;

end

% Input data from linearization files:

matData.NAzimStep = length(FileNames);

data(matData.NAzimStep) = ReadFASTLinear(FileNames{matData.NAzimStep}); %we'll read this twice so we can allocate space first; putting it at matData.NAzimStep saves some reallocation later

matData.NumStates = data(matData.NAzimStep).n\_x;

matData.NumStates2 = data(matData.NAzimStep).n\_x2;

matData.ndof1 = matData.NumStates - matData.NumStates2; % number of first-order states = number of first-order DOFs

matData.ndof2 = data(matData.NAzimStep).n\_x2 / 2; % half the number of second-order states = number of second-order DOFs

% matData.ndof = matData.ndof2 + matData.ndof1; %half the number of second-order states plus the number of first-order states (i.e., states that aren't derivatives)

matData.NumInputs = data(matData.NAzimStep).n\_u;

matData.NumOutputs = data(matData.NAzimStep).n\_y;

%% .................................

% allocate space for these variables

% ..................................

matData.Azimuth = zeros(matData.NAzimStep, 1);

matData.Omega = zeros(matData.NAzimStep, 1);

matData.OmegaDot = zeros(matData.NAzimStep, 1);

matData.WindSpeed = zeros(matData.NAzimStep, 1);

if ( matData.NumStates > 0 )

matData.DescStates = data(matData.NAzimStep).x\_desc;

matData.StateDerivOrder = data(matData.NAzimStep).x\_DerivOrder;

matData.xdop = zeros(matData.NumStates, matData.NAzimStep);

matData.xop = zeros(matData.NumStates, matData.NAzimStep);

matData.A = zeros(matData.NumStates, matData.NumStates, matData.NAzimStep);

end

if ( matData.NumInputs > 0 )

matData.DescCntrlInpt = data(matData.NAzimStep).u\_desc;

if (matData.NumStates>0)

matData.B = zeros(matData.NumStates, matData.NumInputs, matData.NAzimStep);

end

end

if ( matData.NumOutputs > 0 )

matData.DescOutput = data(matData.NAzimStep).y\_desc;

if ( matData.NumStates > 0 )

matData.C = zeros(matData.NumOutputs, matData.NumStates, matData.NAzimStep);

end

if ( matData.NumInputs > 0 )

matData.D = zeros(matData.NumOutputs, matData.NumInputs, matData.NAzimStep);

end

end

%% Reorder state matrices so that they follow the {q2, q2\_dot, q1}

% format that is assumed in the MBC3 equations.

if ( matData.NumStates > 0 )

% keep StateOrderingIndx for applying inverse of MBC3 later

% (to visualize mode shapes)

[matData.StateOrderingIndx, checkEDstates] = getStateOrderingIndx(matData);

x\_rotFrame(matData.StateOrderingIndx) = data(matData.NAzimStep).x\_rotFrame;

matData.DescStates(matData.StateOrderingIndx) = data(matData.NAzimStep).x\_desc;

matData.StateDerivOrder(matData.StateOrderingIndx) = data(matData.NAzimStep).x\_DerivOrder;

end

%% .................................

% get data into variables expected by GetMats (concatenate data from

% different azimuths into matrices)

% ..................................

for iFile = 1:matData.NAzimStep

data(iFile) = ReadFASTLinear(FileNames{iFile});

matData.Omega(iFile) = data(iFile).RotSpeed;

matData.Azimuth(iFile) = data(iFile).Azimuth\*180/pi;

if isfield(matData,'WindSpeed')

if isfield(data(iFile),'WindSpeed')

matData.WindSpeed(iFile) = data(iFile).WindSpeed;

else

matData = rmfield(matData,'WindSpeed');

end

end

if isfield(data(iFile), 'A')

matData.A(matData.StateOrderingIndx,matData.StateOrderingIndx,iFile) = data(iFile).A;

end

if isfield(data(iFile), 'B')

matData.B(matData.StateOrderingIndx,:,iFile) = data(iFile).B;

end

if isfield(data(iFile), 'C')

matData.C(:,matData.StateOrderingIndx,iFile) = data(iFile).C;

end

if isfield(data(iFile), 'D')

matData.D(:,:,iFile) = data(iFile).D;

end

if isfield(data(iFile), 'x\_op')

matData.xop(matData.StateOrderingIndx,iFile) = cell2mat(data(iFile).x\_op);

end

if isfield(data(iFile), 'xdot\_op')

matData.xdop(matData.StateOrderingIndx,iFile) = cell2mat(data(iFile).xdot\_op);

end

end

%% Find the azimuth-averaged linearized 1st order state matrices:

if isfield(matData,'A')

matData.Avgxdop = mean(matData.xdop,2);

matData.Avgxop = mean(matData.xop, 2);

matData.AvgA = mean(matData.A,3);

end

if isfield(matData,'B')

matData.AvgB = mean(matData.B,3);

end

if isfield(matData,'C')

matData.AvgC = mean(matData.C,3);

end

if isfield(matData,'D')

matData.AvgD = mean(matData.D,3);

end

%%

foundED = true;

for i=1:matData.ndof2

col = strfind(matData.DescStates{i},'DOF\_GeAz'); % find the starting index of the string 'DOF\_GeAz'

if ( ~isempty(col) ) % true if the matData.DescStates contains the string 'DOF\_GeAz'

matData.Omega(:) = matData.xdop(i,:)';

matData.OmegaDot(:) = matData.xdop(i+matData.ndof2,:)';

foundED = true;

break;

end

end

for i=1:matData.ndof2

col = strfind(matData.DescStates{i},'DOF\_DrTr'); % find the starting index of the string 'DOF\_DrTr'

if ( ~isempty(col) ) % true if the matData.DescStates contains the string 'DOF\_GeAz'

matData.Omega(:) = matData.Omega(:) + matData.xdop(i,:)'; %This always comes after DOF\_GeAz so let's just add it here (it won't get written over later).

matData.OmegaDot(:) = matData.OmegaDot(:) + matData.xdop(i+matData.ndof2,:)';

foundED = true;

break;

end

end

if ~foundED

for i=1:matData.ndof2

col = strfind(matData.DescStates{i},'MBD Gearbox\_Rot'); % find the starting index of the string 'Gearbox\_Rot'

if ( ~isempty(col) ) % true if the matData.DescStates contains the string 'MBD Gearbox\_Rot'

matData.Omega(:) = matData.xdop(i,:)';

matData.OmegaDot(:) = matData.xdop(i+matData.ndof2,:)';

break;

end

end

end

% ----------- Find multi-blade coordinate (MBC) transformation indices ----

%% Find the indices for, state triplets in the rotating frame

% (note that we avoid the "first time derivative" states)

if matData.ndof2 > 0

if (checkEDstates)

[matData.RotTripletIndicesStates2, matData.n\_RotTripletStates2] = findBladeTriplets\_EDstate(x\_rotFrame(1:matData.ndof2),matData.DescStates(1:matData.ndof2) );

else

[matData.RotTripletIndicesStates2, matData.n\_RotTripletStates2] = findBladeTriplets( x\_rotFrame(1:matData.ndof2),matData.DescStates(1:matData.ndof2) );

end

else

matData.RotTripletIndicesStates2 = [];

matData.n\_RotTripletStates2 = 0;

end

if matData.ndof1 > 0

[matData.RotTripletIndicesStates1, matData.n\_RotTripletStates1] = findBladeTriplets( x\_rotFrame( (matData.NumStates2+1):end) ,matData.DescStates((matData.NumStates2+1):end) );

else

matData.RotTripletIndicesStates1 = [];

matData.n\_RotTripletStates1 = 0;

end

%% Find the indices for control input triplets in the rotating frame:

if (matData.NumInputs > 0)

[matData.RotTripletIndicesCntrlInpt, matData.n\_RotTripletInputs] = findBladeTriplets(data(1).u\_rotFrame,matData.DescCntrlInpt );

else

matData.RotTripletIndicesCntrlInpt = [];

matData.n\_RotTripletInputs = 0;

end

%% Find the indices for output measurement triplets in the rotating frame:

if (matData.NumOutputs > 0 )

[matData.RotTripletIndicesOutput, matData.n\_RotTripletOutputs] = findBladeTriplets(data(1).y\_rotFrame,matData.DescOutput );

else

matData.RotTripletIndicesOutput = [];

matData.n\_RotTripletOutputs = 0;

end

return;

end

%% Reorder state matrices so that all the second-order module's displacements

% are first, followed by all the modules' velocities, followed by all of

% the first-order states.

function [StateOrderingIndx, checkEDstates] = getStateOrderingIndx(matData)

StateOrderingIndx = (1:matData.NumStates)';

lastModName = '';

checkEDstates = true;

lastModOrd = 0;

mod\_nDOFs = 0; % number of DOFs in each module

sum\_nDOFs2 = 0; % running total of second-order DOFs

sum\_nDOFs1 = 0; % running total of first-order DOFs

indx\_start = 1; % starting index of the modules

for i=1:matData.NumStates

modName = strtok(matData.DescStates{i}); % name of the module whose states we are looking at

% ED has the blade number in the state description twice, so we

% have to check the strings differently. We'll check here if a

% different module is used for the blade DOFs:

if strncmpi(modName,'BD',2) || strncmpi(modName,'MBD',3)

checkEDstates = false;

end

if ~strcmp(lastModName,modName)

% this is the start of a new set of DOFs, so we'll set the

% indices for the last matrix

if lastModOrd == 2

mod\_nDOFs = mod\_nDOFs / 2;

StateOrderingIndx( indx\_start :(indx\_start+mod\_nDOFs-1)) = sum\_nDOFs2 + (1:mod\_nDOFs); % q2 starts at 1

StateOrderingIndx( (indx\_start+mod\_nDOFs):(i - 1)) = sum\_nDOFs2 + matData.ndof2 + (1:mod\_nDOFs); % q2\_dot starts at matData.ndof2 + 1

sum\_nDOFs2 = sum\_nDOFs2 + mod\_nDOFs;

else

StateOrderingIndx( indx\_start :(indx\_start+mod\_nDOFs-1)) = sum\_nDOFs1 + matData.NumStates2 + (1:mod\_nDOFs); % q1 starts at matData.NumStates2 + 1

sum\_nDOFs1 = sum\_nDOFs1 + mod\_nDOFs;

end

% reset for a new module

mod\_nDOFs = 0;

indx\_start = i; % start of this module

lastModName = modName;

lastModOrd = matData.StateDerivOrder(i);

end

mod\_nDOFs = mod\_nDOFs+1;

end

% repeat for the last module found:

if lastModOrd == 2

mod\_nDOFs = mod\_nDOFs / 2;

StateOrderingIndx( indx\_start :(indx\_start+mod\_nDOFs-1)) = sum\_nDOFs2 + (1:mod\_nDOFs); % q2 starts at 1

StateOrderingIndx( (indx\_start+mod\_nDOFs):matData.NumStates) = sum\_nDOFs2 + matData.ndof2 + (1:mod\_nDOFs); % q2\_dot starts at matData.ndof2 + 1

else

StateOrderingIndx( indx\_start :(indx\_start+mod\_nDOFs-1)) = sum\_nDOFs1 + matData.NumStates2 + (1:mod\_nDOFs); % q1 starts at matData.NumStates2 + 1

end

end