% Script to generate domain "terrain-mapped" cross section from WRF output

% % Erik M Neemann

% 7 April 2014

clear all;

close all;

clc;

% General Notes:

% - This script plots the zonal wind component for a west-east cross section

% in the Uintah Basin, UT

% - Data is time-averaged from wrfout files at 6-hour intervals that

% contain 6 times each (hourly data). Total data in this example spans 145

% hours from 0000 UTC 1 Feb to 0000 UTC 7 Feb 2013 (inclusive)

% - Data is also spatially averaged along a customizable number of

% gridpoints perpendicular to the centerline of the cross section in either

% direction (originally 10 gridpoints on each side; width = 10).

% - Depth of the cross section is also customizable (originally 25 model

% levels). If deeper than 5000 m, code will need to be modified

% - ps2pdf function needed in matlab directory to create multi-page pdf

% download from http://www.mathworks.com/matlabcentral/fileexchange/19516-ps2pdf

%% Average U Wind Cross Sections

% path to location of wrfout files

% FULL Simulation - Full Basin snow, modified Thompson microphysics

path1 = '/uufs/chpc.utah.edu/common/home/horel-group2/eneemann/Uintah\_Basin\_Runs/WRFv3.5/Feb\_2013\_snow\_TIN12IAU0T/wrfout\_d03\_2013-02';

% Define an array with the times to include in the time average.

% The format is ['-dd\_hh:mm:ss';'-dd\_hh:mm:ss';'-dd\_hh:mm:ss']. When the loop below

% concatenates path and times it must result in a string that creates the full

% path and filename of each wrfout file.

% The loop below will read and manipulate a variable from the wrfout netcdf

% at each time included in the array.

times = ['-01\_00:00:00'; '-01\_06:00:00'; '-01\_12:00:00';...

'-01\_18:00:00';...

'-02\_00:00:00'; '-02\_06:00:00'; '-02\_12:00:00';...

'-02\_18:00:00';...

'-03\_00:00:00'; '-03\_06:00:00'; '-03\_12:00:00';...

'-03\_18:00:00';...

'-04\_00:00:00'; '-04\_06:00:00'; '-04\_12:00:00';...

'-04\_18:00:00';...

'-05\_00:00:00'; '-05\_06:00:00'; '-05\_12:00:00';...

'-05\_18:00:00';...

'-06\_00:00:00'; '-06\_06:00:00'; '-06\_12:00:00';...

'-06\_18:00:00'; '-07\_00:00:00'];

% calculate # of wrfout files to loop through

numtimes = length(times(:,1));

% Loop through each time to load U wind data and create an array with

% hourly data for all 145 hours

% For my example, the last file only contains 1 hour of data. So I

% leavethat file out from this loop and simply place it in the final

% position in the array after the loop.

for i = 1:numtimes-1;

% Generate complete data file path/filename for each wrfout file

string1 = [path1 times(i,:)];

% ncread uses the strings from above to read a wrfout 3d array

% my wrfouts are set up to have 6 hours per file

% --> this adds a dimension to the array making it 4d

% since U is actually 4d it reads in (i\_pts, j\_pts, k\_pts, hours)

U = double(ncread(string1,'U'));

% U is now 298x321x41x6 (i\_pts, j\_pts, k\_pts, hours).

% We want to merge data into one large array that will be 298x321x41x145

% with a value for each grid point x 145 hours

% calculate 4th dimension indicies for each file to be slotted into the

% correct position in the array with hourly data

z1 = (i-1)\*6+1; %first index

z2 = (i-1)\*6+6; %last index

% add the files' six 1-hour times into appropriate position, based on

% indicies calculated above

U\_hrly\_all(:,:,:,z1:z2) = U;

end

% now load last single hour to tack on end of array in last position

string1b = [path1 times(25,:)]; %string for last hour

Ub = double(ncread(string1b,'U'));

U\_hrly\_all(:,:,:,145) = Ub;

% load PH & PHB: needed to calculate geopotential height later on...

PH = double(ncread(string1,'PH')); % perturbation geopotential

PHB = double(ncread(string1,'PHB')); % base state geopotential

% take mean along 4th dimension to get time average for all 145 hours

% data will be reduced to i\_pts by j\_pts by k\_pts (298x321x41)

U\_tavg\_all = mean(U\_hrly\_all,4);

% load elevation to plot on domain in figure 1

HGT = double(ncread(string1,'HGT'));

HGT = HGT(:,:,1); % only use 1st hour

HGT\_conts = [1000:250:3500]; % create contour intervals for plotting

%% Create Cross Section - edit this block to change equation, start, end, depth, and width

% cut cross section at angle using following equation:

x = 1:length(U(:,1,1,1)); % x is # of gridpoints in i direction

slope = -.1411

yintercept = 184.44

y = slope\*x + yintercept; % centerline of cross section

% equation above derived from i,j gridpoints in wrf output (use ncview,

% RIP, etc. to help you pick points)

% insert slope & yintercept from your line equation

% Select cross section start/end in i direction gridpoints, depth in model

% layers, and width in gridpoints

crsa = 74; % i gridpoint at start of xsection (point "a")

crsb = 237; % i gridpoint at end of xsection (point "b")

depth = 25; % in model levels

width = 10; % # of grid points on each side of centerline (perpendicular to xsection)

% NOTE: to do a xsection that's only 1 gridpoint wide, set width = 0

ystart = slope\*x + yintercept - width; %bottom of rectangle to be plotted on map as a check for location

yend = slope\*x + yintercept + width; %top of rectangle to be plotted on map as a check for location

%% Cut Cross Section through 3d domain

% get PH & PHB data from 1st hour - this will be used for all hours

PH\_1hr = PH(:,:,:,1);

PHB\_1hr = PHB(:,:,:,1);

% preallocate arrays of NaNs for data that will be taken along xsection

% 2nd dimension is total width of xsection, 1 is added to include centerline

U\_tavg\_angled = NaN(length(U(:,1,1,1)-1),(2\*width+1),length(U(1,1,:,1)));

PH\_tavg\_angled = NaN(length(PH\_1hr(:,1,1,1)-1),length(PH\_1hr(1,1,:,1)));

PHB\_tavg\_angled = NaN(length(PHB\_1hr(:,1,1,1)-1),length(PHB\_1hr(1,1,:,1)));

% pull data along xsection using equation from previous block

% U data has width, so resulting array is 3d rectangle along xsection

for X = 1:length(U(:,1,1,1))-1;

for Z = 1:length(U(1,1,:,1));

U\_tavg\_angled(X,:,Z) = U\_tavg\_all(X,(int64(slope\*X + yintercept-width)):(int64(slope\*X + yintercept+width)),Z);

% only centerline is used for PH/PHB in geopotential calculation,

% so resulting array is 2d rectangle

PH\_tavg\_angled(X,Z) = PH\_1hr(X,(int64(slope\*X + yintercept)),Z);

PHB\_tavg\_angled(X,Z) = PHB\_1hr(X,(int64(slope\*X + yintercept)),Z);

end

end

% take spatial average along the width of the xsection (2nd dimension)

U\_tavg\_xavg\_angled = mean(U\_tavg\_angled,2); % result is 2d array

% trim down array to specified depth and gridpoints (crsa:crsb)

% result is 2d array along xsection (i\_pts by depth)

U\_tavg\_trim\_depth = U\_tavg\_xavg\_angled(crsa:crsb,1:depth);

%% Check that cross section is correct and build colormap

% plot elevation data, xsection centerline and bounds used for gridpoint

% averaging to make sure they are in the right location

figure(1)

contourf(HGT,HGT\_conts); % plot terrain in grayscale

colormap(flipud(gray));

hold on;

plot(y(crsa:crsb),x(crsa:crsb),'r','LineWidth',3) %plot xsection centerline

plot(ystart(crsa:crsb),x(crsa:crsb),'r','LineWidth',1) %plot bottom bound of spatial avg

plot(yend(crsa:crsb),x(crsa:crsb),'r','LineWidth',1) %plot top bound of spatial avg

colorbar;

set(gca,'view',[90 -90]) %rotate to view from correct orientation

hold off;

% make all text in the figure to size 12 and bold

set(gca,'FontSize',12,'fontWeight','bold')

set(findall(gcf,'type','text'),'FontSize',12,'fontWeight','bold')

% set orientation of .ps output

h=gcf;

set(h,'PaperOrientation','landscape');

set(h,'PaperUnits','normalized');

set(h,'PaperPosition', [0 0 1 1]);

print -dpsc2 -r600 input1; %create .ps file

%create blue to red colormap (only 19 divisions) for shading U winds

blue\_to\_red = zeros(19,3);

blue\_to\_red(1,:) = [0 0 1]; %blue

blue\_to\_red(1:10,3) = 1;

B1to49 = linspace(0,1,10)'; %fade to white (0 to 1)

blue\_to\_red(1:10,1) = B1to49; %apply to "red"

blue\_to\_red(1:10,2) = B1to49; %apply to "green"

blue\_to\_red(10,:) = [1 1 1]; %white

G51to99 = linspace(1,0,10)'; %fade from white (1 to 0)

blue\_to\_red(10:19,2) = G51to99; %apply to "green"

blue\_to\_red(10:19,3) = G51to99; %apply to "blue"

blue\_to\_red(10:19,1) = 1;

blue\_to\_red(19,:) = [1 0 0]; %red

%end colormap creation

%% Calculate correct geopotential height along xsection

GEO\_angled = PH\_tavg\_angled + PHB\_tavg\_angled; % sum for total geopotential (at bottom of gridbox)

Z = GEO\_angled./9.81; % height is geopotential divided by gravity (height at bottom of gridbox)

% loop through data to calculate mid-model heights (where U variable lives) in meters

Z\_cor = NaN(size(Z));

for n = 1:length(Z(:,1));

for z = 1:length(Z(1,:))-1;

Z\_cor(n,z) = (Z(n,z) + Z(n,z+1))/2;

end

end

% trim down array to specified depth and gridpoints (crsa:crsb)

Z\_fix\_trim\_depth\_m = Z\_cor(crsa:crsb,1:depth);

%% Map data to Terrain and Interpolate between model data

% This is where things start to get complicated. In order to map the data

% to the terrain, we have to find the height at each gridpoint (X) and

% each model level (Z). Then we will plot this data to a very large array where

% each vertical level represents 1 m. The model data will only be ploted on

% the level closest to it's actual height. Example: If the height of point

% (100,7) in the xsection X-Z plane is 1651.3 m, it will only be plotted at

% the point (100,1651) in our new array. Then we loop thru the data and

% linearly interpolate between these data points for each vertical column.

% allocate array to map data to terrain, and fill it with NaNs

terrain\_mapped\_tavg = NaN(crsb-crsa,5000); % capping xsection at 5000 m since I'm interested in lower atmosphere

% create array with only data filled at model level heights and NaNs everywhere else

for r = 1:length(U\_tavg\_trim\_depth(:,1)); % step along xsection direction

for s = 1:length(U\_tavg\_trim\_depth(1,:)); % step along vertical direction

% apply U value at (r,s) to (r,height(r,s)) in new array

terrain\_mapped\_tavg(r,int64((Z\_fix\_trim\_depth\_m(r,s)))) = U\_tavg\_trim\_depth(r,s);

end

end

% create another array to hold "fixed" data after interpolation

terrain\_mapped\_fix\_tavg = terrain\_mapped\_tavg;

%% Build placeholder array to aid interpolation

% create "placeholder" array that will be used to help interpolation, fill

% it with 0s and make depth 5000 meters

placeholder = zeros(crsb-crsa,5000);

placelow = 0;

% loop for tavg

for a = 1:length(terrain\_mapped\_tavg(:,1)); %step thru horizontal

for b = 1:length(terrain\_mapped\_tavg(1,:)); %step thru vertical

% 1st pass:

% if slot on model level filled, placelow becomes that model level (placeholder)

if isnan(terrain\_mapped\_tavg(a,b)) == 0 && placelow == 0;

placelow = b;

placeholder(a,b) = b;

continue % go to next step in for loop

else

end

% all subsequent passes:

% if model level is above placeholder and unfilled, nothing happens

if placelow ~= 0 && b > placelow && isnan(terrain\_mapped\_tavg(a,b)) == 1;

% if model level is filled (& above old placeholder), it becomes new placholder

else

placelow = b;

placeholder(a,b) = b;

end

end

end

% array is now filled with model height only at model height, rest is 0s

placeholder(:,1) = 0; % set lowest level to 0

placeholder((crsb-crsa)+1,:) = 0; % make last column 0

% collapse placeholder array by removing 0s from each column

placeholder\_fix = zeros(crsb-crsa,depth);

for a = 1:crsb-crsa; %step horizontally

place\_a = placeholder(a,:); %grab column at point a

place\_a(place\_a==0)=[]; %remove 0s in column (make them blanks)

placeholder\_fix(a,:) = place\_a; %place collapsed column into the "fixed" array

end

%check to see if zeros removed from placeholder...they are!

%% Linearly Interpolate in the vertical

% given placeholder matrix exists without zeros, use to interpolate

for a = 1:length(terrain\_mapped\_tavg(:,1))-1; %step thru horizontal

l = 1;

for b = 1:length(terrain\_mapped\_tavg(1,:)); %step thru vertical

if l > (depth-1);

placehigh = 5000;

else

placelow = placeholder\_fix(a,l);

placehigh = placeholder\_fix(a,l+1);

end

if b > 1 && isnan(terrain\_mapped\_tavg(a,b)) == 1 && b > placelow && b < placehigh;

% this code replaces the NaNs with interpolated values in between the

% model levels (when b is between placelow and placehigh)

terrain\_mapped\_fix\_tavg(a,b) = ((b-placelow)/(placehigh-placelow))\*(terrain\_mapped\_tavg(a,placehigh)) +...

(1 - ((b-placelow)/(placehigh-placelow)))\*(terrain\_mapped\_tavg(a,placelow));

else if b > 1 && isnan(terrain\_mapped\_tavg(a,b)) == 0 && b > placelow;

l = l+1; % raise placelow for next vertical step if b > placelow

else %retain values on model levels from original array

terrain\_mapped\_fix\_tavg(a,b) = terrain\_mapped\_tavg(a,b);

end

end

end

end

% loop back thru entire array...if 0s exist, replace with avg of cells above/below

for a = 1:length(terrain\_mapped\_tavg(:,1))-1; %step thru horizontal

for b = 1:length(terrain\_mapped\_tavg(1,:))-1; %step thru vertical

if b > placeholder\_fix(a,1) && terrain\_mapped\_fix\_tavg(a,b) == 0; %find 0s

%replace with avg of cells above/below

terrain\_mapped\_fix\_tavg(a,b) = (terrain\_mapped\_fix\_tavg(a,b-1)+terrain\_mapped\_fix\_tavg(a,b+1))/2;

end

end

end

%NaNs remain for all cells in the array that are below the model terrain

%% Not sure if this is still needed with interpolation method...

%Uncomment the loop below, if you are getting errors at the top of your

%plotted cross section

% Pass through data array final time to place NaNs above last model level.

% This prevents last model level data from getting stretched to top of

% plot. Shouldn't be needed, now that interpolation is used in place of

% the original scheme which filled the last model level data upward to the

% top of the array.

% c = 0;

% % loop for tavg

% for a = 1:length(terrain\_mapped\_tavg(:,1)); %step thru horizontal

% for b = 1:length(terrain\_mapped\_tavg(1,:)); %step thru vertical

% if isnan(terrain\_mapped\_tavg(a,b)) == 0

% c = b; % c ends up being highest model level data

% else

% end

% end

% for b = 1:length(terrain\_mapped\_tavg(1,:)); %step thru vertical

% if b > (c+40); %more than 40 m above last model level data

% terrain\_mapped\_fix\_tavg(a,b) = 0; %NaN

% else

% end

% end

% end

%% Create Terrain Outline

% create new array that will be used to outline terrain

terrain = terrain\_mapped\_fix\_tavg;

%Loop through array, find NaNs below the the model terrain, and replace

%them with -9999. We'll later plot a contour at -50 that will outline the

%terrain since all cells with real data are > -50 and all cells below model

%terrain are < -50 because they will be -9999

for i = 1:length(terrain(:,1)); %step thru horizontal

for j = 1:length(terrain(1,:)); %step thru vertical

if isnan(terrain(i,j)) == 1 %if cell is a NaN, replace with -9999

terrain(i,j) = -9999;

else

continue

end

end

end

tercont = [-50 -50];

%% Plot Final Cross Section

windconts = [-2 -1 -0.5 0 2 4 6 8]; % set contours for U wind speed

xwidth = crsb-crsa;

figure(2)

colormap(blue\_to\_red);

contourf(terrain\_mapped\_fix\_tavg,100,'LineStyle','none');

hold on;

contour(terrain\_mapped\_fix\_tavg,windconts,'k','LineWidth',1);

axis([1400 3000 0 xwidth]); % set vertical/horizontal data ranges

colorbar;

caxis([-5 5]); % set constant colorbar range (must center on 0)

contour(terrain,tercont,'k','LineWidth',2); % thick contour of terrain outline

set(gca,'view',[90 -90]) %rotate to view correct orientation

title({'Time Averaged U-Wind Cross Section (m/s)','Full Snow'},'FontSize',12,'FontWeight','bold');

xlabel('Height (m)','FontSize',12,'FontWeight','bold');

ylabel('East-West Gridpoints','FontSize',12,'FontWeight','bold');

% make all text in the figure to size 12 and bold

set(gca,'FontSize',12,'fontWeight','bold')

set(findall(gcf,'type','text'),'FontSize',12,'fontWeight','bold')

% set orientation of .ps output

h=gcf;

set(h,'PaperOrientation','landscape');

set(h,'PaperUnits','normalized');

set(h,'PaperPosition', [0 0 1 1]);

print -dpsc2 -r600 input1 -append; %append to .ps file

% use ps2pdf function to create multi-page pdf file from .ps file

ps2pdf('psfile', 'input1.ps', 'pdffile', 'xsect\_wind.pdf', 'gspapersize', 'letter', 'deletepsfile', 1);