MATLAB Scripts for Processing Track Files (including script definitions)

August, 2023

Note: See MATLAB Event Set Variables document for a description of the variables contained in each MATLAB event set

- We recommend using the *m_map* routines, which is the default behavior as determined by the variable *MapOverride* in the *params.m* file.
- The behavior of the scripts is controlled using the file *params.m*. **It is a good idea to become familiar with this script.**
- Please see **Scripts** section below for detailed description of individual scripts.
- The User functions section describes some

What's new: As of August, 2023: An error was discovered in the rain processing scripts; basically, a factor of two in the Coriolis parameter as it is used to calculate the angular momentum gradient. This has been fixed, but it was necessary to recalibrate the precipitation efficiency and it is now set to 0.5. Using the old value of *eprecip* will result in too much rainfall.

Preparation:

Unzip the zip file containing the MATLAB scripts (scripts_verN.M.zip) into a single initially empty directory.

Add this new directory (**including its subdirectories**) to your MATLAB path.

If you have and wish to use the MATLAB mapping toolbox, (note recommended), go to http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html and download the shapefile version of the GSHHG data into the (initially empty) shapefiles directory created in the first step above. Unzip the GSHHG zip file. (After you have done this, you may want to discard the GSHHG zip file.) You should now have two subdirectories in the shapefiles subdirectory.

Place the MATLAB track file <trackname>.zip that you acquired from WindRiskTech into the subdirectory *event_sets*. DO NOT UNZIP!

Run the script *initialize.m*. This may take a few minutes, but you only need to do this once. All this script does is read in some map background .jpeg files, convert them to MATLAB binary format, and write them out so that they can be quickly read into certain scripts. Note that the output files can be quite large. No need to re-run *initialize.m* for new event sets.

There are options to create graphical output in *Google Earth* (the free version is fine). To use these, you must of course have *Google Earth*, and you must also install the free MATLAB *KML toolbox* available at

MATLAB central file exchange: http://www.mathworks.de/matlabcentral/fileexchange/34694-kml-toolbox-v2-1 . Note that all the available scripts in *Google Earth* have equivalents that use mapping routines provided as part of the set of scripts.

In each plot, the variable on the x-axis is always called x, that on the y axis is called y, and for contour plots, the contoured variable is called z. For x-y plots with more than one y, the second variable is called y2, the third y3, etc. A few graphs have multiple x values: x, x2, x3, etc, and some best track contour plots have z2 as the contoured variable.

Before you can use most of the scripts**, you must first run *prep.m***. When prompted, enter the event set file name (without any directory name and without the ".zip"). Note that running *prep.m* produces no plots, but just produces some temporary files (*temp.mat* and *sorted.mat*) that are in turn read by the various plotting scripts. It need never be run again until/unless a new event set is used. There is an alternative script, *prepfilter.m*, that prepares subsets of the specified event set that include events that pass through a circle of a specified radius centered at a point of interest, or through any of a set of line segments specified in a '*poly.in*' file, or (in the case of a global data set) that is confined to a particular ocean basin, or that includes only a subset of years and/or months. *Poly.in* is just an ascii file consisting of 4 columns containing, respectively, the longitude and latitude of the beginning point of each line segment, and the longitude and latitude of the end point of each line segment. The segments can be, but need not be, contiguous. If they together form a closed polygon, the filter will also include storms whose entire tracks are within the polygon. An easy way to create a poly filter is to do so using Google Earth; see the entry for the script *ge_to_poly.m* under *Main utilities* below.

It is important to note that running *prep.m* produces several new variables that can be quite valuable. These include *vnet.mat*, the maximum ground-relative wind in each event, including the effects of background flow, and *vmax.mat*, the maximum ground relative wind at the point of interest (if there is one) or along the line segments specified in *poly.in*.

Some of the scripts below require best track data in MATLAB binary format. These can be obtained from ftp://texmex.mit.edu/pub/emanuel/HURR/tracks/ and should be placed in the *best_tracks* subdirectory and updated annually.

Almost all of the parameter values used by the scripts are set in the single MATLAB script *params.m*. You may find it convenient to keep this script open in its own window so that the parameters can easily be modified. The default values of the parameters are also indicated here.

A few of the scripts (e.g. *pointplot.m*) require radial profiles of wind. The user has a choice of several published profiles, but recent research shows that the outer dimensions of tropical cyclones are lognormally distributed in nature. The scripts provide an option (the parameter *randfac*) to draw randomly from an empirically determined log-normal distribution; if that option is activated, then the results will differ each time the script is invoked, corresponding to a different overall storm diameter or randomly drawn set of storm diameters, for scripts that use more than one event. A warning message to this effect appears in a MATLAB dialog box.

Scripts:

[**Convention:** In the descriptions below, the index *n* refers to the event number, while the index *m* refers to the 2-hour records of each event. For example, *latstore (n,m)* is an array containing *n* tracks each of which has *m* two-hour observations. (The arrays are padded with zeros between the end of each track and the end of the file.) A full description of the arrays is provided at the end of this document.]

a. Parameter script

params.m: This file simply sets the parameter values used by the various scripts; this is the place to adjust them. The parameters are briefly described, and their default values are listed.

b. Main utilities:

prep.m: Reads in the main track file and calculates the full wind speed, including the effect of storm translation. It also calculates the lifetime maximum wind speed of each event, the maximum wind speed at a point of interest, if the tracks have been filtered to pass within a certain distance of that point, or the maximum wind speed in the storm as it passes over any of the line segments specified in 'poly.in', of the tracks have been filtered to pass through such line segments. The "point of interest" is read from the master binary file, but can be specified independently if desired. This script need only be run once, until/unless a new event set is used. It produced temporary files, *temp.mat* and *sorted.mat*, that are read by the other scripts.

prepfilter.m: This is the same as *prep.m* except that it prepares a subset of the main track file that consists of tracks that pass through any of a set of line segments in a user-specified file of the same structure as '*poly.in*', or that pass within a specified radius of a given point, or that occur in a specified ocean basin, or that occur within a given set of years and/or months. This is useful for examining subsets of the main track file, for example, those tracks that make landfall in the U.S.

c. General graphics control menu

gmenu.m: This opens up a menu window from which one can summon most of the graphics packages described below. Be aware that some of the packages ask for input from MATLAB dialog boxes. Also note that many of the parameters controlling the graphs must be set within the script *params.m*.

d. Plotting and other scripts (in alphabetical order)

Note: The designation *(mt)* in some of the scripts denotes the presence of an equivalent script that uses the MATLAB Mapping Toolbox.

Note: For these scripts, the variable on the x-axis is always called x, that on the y axis is called y, and for contour plots, the contoured variable is called z. For x-y plots with more than one y, the second variable is called y2, the third y3, etc. Some plots have more than one x or z as well.

The documentation below is also accessible by clicking on the link 'doc' at the lower left side of each plot (except for Google Earth plots).

alltrge.m: Plots multiple storm tracks in Google Earth. The tracks can be specified in the order they were produced in, ordered by their lifetime maximum wind speed, or ordered by the maximum wind speed at a point of interest or at the intersection points of a specified set of line segments in a poly file.

alltrmap.m: This plots multiple storm tracks on a map background provided by any of a limited set of supplied image files. (One can provide one's own image, but it must be an equirectangular projection.) The tracks can be specified in the order they were produced in, ordered by their lifetime maximum wind speed, or ordered by the maximum wind speed at a point of interest or at the intersection points of a specified set of line segments.

alltrbestmap.m: Same idea as *alltrmap*, but plots historical tracks whose source is described in the introduction.

allbestovermap.m: Overlays historical tracks onto track maps created using *alltrmap*, which must be run directly before this script is executed. The historical tracks must be as described in the introduction.

alltrplot(mt).m: Same *as alltrmap*, but the map background is supplied using the MATLAB Mapping Toolbox, or if absent, the *m_map* routines, which have to be installed for this script to work (see introduction).

alltrbestplot.m: Same idea as *alltrplot*, but plots historical tracks whose source is described in the introduction. (Note: This script does not have an equivalent that uses the MATLAB mapping toolbox.)

allbestoverplot(mt).m: Overlays historical tracks onto track maps created using *alltrplot*, which must be run directly before this script is executed. The historical tracks must be as described in the introduction.

annual.m: Makes a bar chart of the number of storms in each month, with the first track point determining the month of formation.

annualbesterr.m: Compares the frequency of storms from synthetic sets to those of best track data. The former can be normalized to the best track annual frequency. The script *annual* must be run immediately before this, and the script must have access to best track data (see introduction). There is an option to add 90% confidence limits based on an assumption that the best track storm count in each month has a Poisson distribution.

bestplot(mt).m: Plots tracks and intensities of historical storms, which must be available in MATLAB binary format, as described in the introduction.

bestproc.m: Reads in a processes best track data, interpolating them from 6 to 2 hour intervals. It also applies the same spatial filters as used in *prep.m*, which must be run first.

genpoints(mt).m: Scatter plot of genesis points for the whole track set, with genesis defined as the first point of each event with maximum winds exceeding a specified threshold. The events may be filtered to include only those whose maximum lifetime wind speed, or maximum wind speed at a point of interest, exceeds a specified threshold. The MATLAB Mapping Toolbox must be installed, or the *m_map* routines are required for this (see introduction).

genpointsbest(mt).m: Overlays a scatter plot of best track genesis points on the map created by *genpoints.m*, which must be run first. The same filters are used as in *genpoints*.

gendensity(mt).m: Contour plots a map of the spatial density of genesis points in the whole track set, with genesis defined as the first point of each event with maximum winds exceeding a specified threshold. The events may be filtered to include only those whose maximum lifetime wind speed, or maximum wind speed at a point of interest, exceeds a specified threshold. Either the MATLAB Mapping Toolbox or the *m_map* routines are required for this (see introduction).

gendensitybesttrack(mt).m: Contour plots a map of the spatial density of best track genesis points as in *gendensity.m*, which must be run first. The same filters are used as in *gendensity*.

p_to_v.m: This function estimates maximum 1-minute (circular) winds (m/s) at 10-m altitude from surface central pressure, using equations from an idealized zero saturation potential vorticity model with assumed constant surface relative humidity from the environment inward to the radius of maximum winds.

pdimap(mt).m: Creates maps of the power dissipation index. The parameter *mres* specifies the resolution of the maps. Either the MATLAB Mapping Toolbox or the *m_map* routines are required for this (see introduction).

pdimapbest(mt).m: Creates maps of the power dissipation index from best track data, as in *pdimap*, which must be run first.

pointplot.m: Plots time series of wind speed and direction at a given point of interest, specified by maximum lifetime wind, by maximum wind at the point of interest, or by track number.

pressurefield.m: Contours the surface pressure (hPa) associated with a specified event at a given time. The background surface pressure is included, but it is assumed that this does not vary over the map. Note that the pressure field may not be completely consistent with the surface wind field in regions of high winds. One may also plot the track of the event up to the specified time. There is no MATLAB mapping toolbox equivalent for this script.

rainfield(mt).m: Contours the rain rate associated with a specified event at a specified time. The radial wind profile used in the calculation is calculated in *windprofilem.m*. One may also plot the track of the event up to the specified time.

rainpdf.m: Return period of rain rate and accumulated rainfall at the default or specified point of interest.

rainseries.m: Time series of rain rate and accumulated rainfall at the default or specified point of interest.

rainswath(mt).m: Contours the accumulated rainfall associated with a specific event.

rdoubletime.m: Plots the radii of maximum wind, including those of secondary maxima (if present), and an outer radius, all associated with a specific event. The outer radius is defined in terms of a threshold value of the circular component of surface wind; this value can be set at the beginning of the script.

return_period.m: Uses normalized cumulative distribution functions to plot return periods from synthetic tracks, and fits a theoretical function to the results. These are return periods of maximum winds within *city_radius* of the point of interest, if *shape* ='circ', or of maximum winds along the intersection of line segments, if *shape*='poly'.

return_best.m: Overlays the return periods calculated from historical tracks onto the results of *return_period*, which must be run first. Historical data must be in matlab binary format, as described in the introduction. (These are defined such that, if the best track data are drawn from the same distribution as the synthetic track data, 5% of the samples should lie to the left of the left error bar, and 5% to the right of the right error bar.)

sestats.m: Plots six key statistics of secondary eyewalls (SEs). The first plot just shows the probability of storms with the number of SEs given on the x axis. The second plot shows the mean and standard deviation of lifetime maximum wind speed among the events with the number of SEs given on the x axis. The third plot shows the mean and standard deviation of wind speed integrated over the life of each event, among the events with the number of SEs given on the x axis. The final three plots are identical to the first three except that they show statistics of "primary eyewall replacements" (PERs), which occur when a new eyewall forms inside an existing eyewall and eventually replaces it. This phenomenon happens during the decay phase of some simulated tropical cyclones.

swath(mt).m: This contours the maximum winds (knots) experienced at each point in a specified area associated with a particular storm. Either the MATLAB Mapping Toolbox or the *m_map* routines are required for this (see introduction).

time_series.m: For data sets that contain multiple years, this script plots time series of various quantities such as annual frequency of all events, frequency of hurricanes of various categories, power dissipation, etc. This script accesses best track data.

trackdensity(mt).m: Contours the number of tracks per unit area in the whole event set. Units are number per year per one degree latitude square.

trackdensitybest(mt).m: Contours number of best track per degree latitude square per year, as in *trackdensity*, which must be run first.

trackstats.m: Calculates histograms of 6-hour west-east and north-south displacements along all the tracks in an event set and compares them to similar displacements in best-track data sets. The script must have access to best track data in MATLAB binary format (see "Preparation" section).

trmap.m: Plots a single tropical cyclone track on a map background provided by an image file; the image must be a map in equirectangular projection. The track can be specified by maximum lifetime

intensity, by maximum wind speed at a point of interest or the intersection points of tracks with a specified set of line segments, or by track number. The intensity of the storm can be shown on the map, together with daily *00 GMT* positions. Other graphs show the evolution over the life of the storm of several quantities of interest.

trplot(mt).m: Same *as trmap*, but the map background is supplied using either the MATLAB Mapping Toolbox or the *m_map* routines, which have to be installed for this script to work (see introduction).

trplotge.m: Same *as trmap*, but includes options to plot tracks in *Google Earth* or with map backgrounds as in *trplot* (see introduction).

v_to_p.m: This function estimates surface central pressure from maximum 1-minute (circular) winds (m/s) at 10-m altitude, using equations from an idealized zero saturation potential vorticity model with assumed constant surface relative humidity from the environment inward to the radius of maximum winds.

vdoubletime.m: Plots a time series of maximum ground-relative wind, including that of secondary wind maxima if present, associated with a specific event.

vhisto.m: This makes an annual exceedence frequency chart for lifetime maximum wind speeds or maximum wind speeds within *city_radius* of a point of interest or at intersection points of supplied line segment set using all the events in a track set.

vhistobest.m: Compares annual exceedence frequencies of synthetic track data to those of best-track data. The script *vhisto* must be run directly before this, and the script must have access to best track data in MATLAB binary format (see "Preparation" section). There is an option to add 90% confidence limits based on an assumed Poisson distribution. (These are defined such that, if the best track data are drawn from the same distribution as the synthetic track data, 5% of the samples should lie above the upper error bar, and 5% below the lower error bar.)

windfield(mt).m: Contours the wind speed associated with a specified event at a specified time. The radial wind profile is calculated in *windprofilem.m*. One may also plot the track of the event up to the specified time.

windfield_quiver.m: Same as windfield.m but displays surface wind as vectors. Note that there is no MATLAB mapping toolbox equivalent for this script.

windpdf.m: Plots return periods of storm peak wind speed at the default or user-specified point-of-interest (POI) using the entire event set.

e. User functions *(not in any particular order)*

Function [vs2,dir,dayk,v,f,rp] = ***windpdfx****(POI,cal)*: This function creates time series of tropical cyclone wind speed and direction at a specified set of geographic points for each event in the set and also creates annual exceedance frequencies for peak winds at the specified points. Because it operates on all the events, it can be slow when many geographic points are specified.

Please type *doc windpdfx* at the MATLAB prompt for detailed instructions on using this function.

Function [rainrate,rain,dayk,rx,f,rp] = ***rainpdfx****(POI,cal,rmin,rmax)*: Like *windpdfx* but creates time series of rainfall rate at each of a specified set of geographic points as well as annual exceedance frequencies of storm total rainfall. Type *doc rainpdfx* at the MATLAB prompt for detailed instructions on using this function.

Function [x,y,windspeed] = ***windfieldx****(nt,monthplot,dayplot,hourplot)*: This function creates a gridded set of wind speeds for a particular storm at a given date and time. The grid can be specified or automatically generated by the function using parameters from the *params.m* file. Type *doc windfieldx* at the MATLAB prompt for detailed instructions on using this function.

Function [x,y,rainrate] = ***rainfieldx****(nt,monthplot,dayplot,hourplot)*: Like *windfieldx*, but creates a gridded set of rainfall rates for a particular storm at a given date and time. Type *doc rainfieldx* at the MATLAB prompt for detailed instructions on using this function.

Function [x,y,maxwind] = ***windswathx****(nt)*: Creates a gridded set of maximum surface winds speeds experienced at each of a gridded set of points during the passage of a particular tropical cyclone. The grid can be specified or automatically generated by the function using parameters from the *params.m* file. Type *doc windswathx* at the MATLAB prompt for detailed instructions on using this function.

Function [x,y,netrain] = ***rainswathx****(nt)*: Like *windswathx* but creates a gridded set of storm total rainfalls for a particular tropical cyclone. Type *doc rainswathx* at the MATLAB prompt for detailed instructions on using this function.

Function [] = ***ge_to_poly*** *(infilename,outfilename)*: Creates a poly file from .kml format files created using *Google Earth*. To use this, first open *Google Earth* and use its *add path* tool to create a series of line segments. You should create a set of separate .kml files each containing a ***contiguous*** set of line segments and you must name each file according to the convention <name>1.kml, <name>2.kml….., where <name> is any name you choose. For example, path1.kml, path2.kml, path3.kml. Note that the line segments in, say, path2.kml need not be contiguous with those in path1.kml or path3.kml. In creating the paths, use discrete mouse clicks to create a series of straight line segments; holding down and dragging the mouse will create a series with a potentially large number of nodes, slowing down the filtering scripts. Once created, find the files in the left-hand column in the *Google Earth* "Places" pane, right-click, and save each to the main directory where the WindRiskTech scripts are stored. Then run

ge_to_poly (infilename, outfilename), where *infilename* is the name you gave the path files (e.g. *path* in the example above) and *outfilename* is the name you wish to call the resulting poly file. (Both names should be in single quotes.) Note that this poly file is a single file that contains the data in all the .kml files used. In the special case of a single closed polygon, which is treated differently by the filtering scripts, you should manually edit the poly file to insure that the very last lat-long point is identical to the first.

f. ***Auxiliary function****s (Not normally used directly)*

*Function [ ] =****geoback*** *(xmin,xmax,ymin,ymax)*: Reads in various shapefiles that provide map backgrounds for maps produced using the MATLAB Mapping Toolbox. The parameters give the longitude and latitudes of the bounding box of the map.

Function [pass,xint,yint,jint,kint,kfrac] = ***seg*** *(xtrack,ytrack,xa,ya,xb,yb)*: This function determines whether a set of *n t*racks (given by coordinates *xtrack*,*ytrack*) intersects a series of *m-1* line segments whose end points are given by *xa(m), ya(m), xb(m), yb(m)*. *Pass*=0 means no intersection; *pass*=1 means intersection.

The quantities *xint* and *yint* constitute the intersection points, along the line segment numbered *jint* and the *kint*'th point along the track. *kfrac* is the fraction of the distance between track points *i* and *i+1* that the intersection occurs.

Note that *pass* is a vector of length *n*, where *n* is the number of tracks, and the other quantities are matrices of dimension (*n,maxtimes*), where *maxtimes* is the specified upper limit on the number of times a particular track may intersect the set of line segments.

Function [ut,vt,jmax] = ***utrans*** *(lat,long)*: Given 2-hour latitude and longitude positions along a track, this returns the west-eat and south-north components of the storm translation velocity. The function assumes that the lats and longs are two-D arrays whose first index is the event number and whose second index is the 2-hour position for the event. It also returns *jmax(n)*, where *n* is the number of events. This marks the last 2-hour position of the track; the translation speeds are padded with zeros from the end of each track to the end of the file. Some smoothing is done to the translation speeds; the mount of smoothing is adjustable using the parameter *smfac*.

Function [ut,vt,uinc,vinc] = ***utransfull*** *(lat,long,vstore,u850store,v850store)*: Returns translations speeds as in *utrans.m* but also returns zonal and meridional increments (knots) to maximum circular wind speed based on storm translation and an additional baroclinic correction. The additional inputs are the arrays of maximum circular storm wind (knots) and environmental zonal and meridional flow (knots) at the storm positions.

Function [ut,vt,jmax] = ***utransbest*** *(latstore,longstore,monthstore,daystore,hourstore)*: Similar to utrans, but requires month, day and hour information. This is ideal for historical tracks whose data points may be unevenly spaced in time.

Function [transfactor] = ***transfunction****(latitude)*: This function calculates the fraction, *transfactor*, of the translation speed to add to the circular storm wind speed. This fraction may be a function of *latitude*.

Function [nint,xint,yint,jint,kint] = ***boxm*** *(long,lat,xa,ya,xb,yb)*: Given 2-D latitude and longitude positions along a track (as in *utrans*), and *j-1* line segments whose end points are given by *xa(j), ya(j), xb(j),* and *yb(j)*, this function returns the intersection longitude and latitude (*xint(n)* and *yint(n)*) of each track with the set of segments. Nint is 0 or 1 denoting no intersection or an intersection of the track with the set of line segments. Note that in the event that a track intersects the array of line segments more than once, only the first intersection point is calculated. The last point of the set *xa(j), ya(j), xb(j),* and *yb(j),* may equal the first, in which case one has a closed polygon. In this case, the routine also checks for tracks that lie entirely within the polygon. For this reason, the segments must be specified going CLOCKWISE around the polygon. The quantities *xint* and *yint* constitute the first intersection point, along the line segment numbered *jint* and the *kint'th* point along the track. In the case of a closed polygon, if the track is entirely within the polygon, *jint* is set equal to *-1* and *xint=yint=0.0.*

Function [nint,xint,yint,jint,kint,vint,kfrac] = ***boxmv****(xtrack,ytrack,vtrack,xa,ya,xb,yb*): Same as *boxm.m*, but also interpolates input wind speed to the intersection point of the track with the relevant line segment. Interpolated wind speed given by *vint*.

Function [vs, dir, dayk] = ***pointseriesn*** *(latstore, longstore, vstore, rmstore, vsestore, rmsestore, monthstore, daystore, hourstore, uinc, vinc, jmax, plat, plong, timeres)*: This function takes the n X m matrices for the latitudes, longitudes, radii of maximum winds, dates, times, and translation speeds and, using the latitude (*plat*) and longitude (*plong*) vectors of a set of points of interest, calculates the time series of wind speed and direction at the points of interest. A wind profile, calculated in *windprofilem.m*, is fitted to the maximum wind speeds and radii of maximum winds (including any secondary maxima present) to calculate the wind speeds and directions at the points of interest, and the translation speed contribution to the net wind is accounted for. The routine *utransfull.m* should be run before this one to provide the velocity increments *uinc* and *vinc*. Note that when this routine is called by prep, the point of interest is taken to be the center of the circle, if circular filtering has been used to generate the track set.

Function [vs, dir] = ***pointshortn****(latstore, longstore, vstore, rmstore, vsestore, rmsestore, uinc, vinc, jmax, plat, plong)*: Same as function *pointseries* above, but omits calculation of date/time and is thus a little faster.

Function [w] = ***pointwfield****(latstore,longstore,vstore,rmstore,ut,vt,us,vs,plat,plong,h,hx,hy)*: This function takes the nn X 400 'store' matrices and, using the latitude (plat) and longitude (plong) vectors of points of interest, calculates the spatial distribution of vertical velocity centered at *plong, plat*. The routine *utrans.m* should be run before this one to give the velocity increments *ut* and *vt*. The arrays *us* and *vs* are the vertical shears (m/s) used to estimate the baroclinic components of the vertical motion. The matrices *h, hx, and hy* contain topographic heights and their derivatives in *x* and *y*.

Function [w] = pointwshortn(latstore, longstore, vstore, rmstore, vsestore, rmsestore, ut, vt, us, vs, plat, plong, h, hx, hy, timeres): This function take the nn X 400 'store' matrices and, using the latitude (plat) and longitude (plong) vectors of points of interest, calculates time series of vertical velocity at the points of interest. The routine *utrans.m* should be run before this one to

give the velocity increments *ut* and *vt*. The arrays *us* and *vs* are the vertical shears (m/s) used to estimate the baroclinic components of the vertical motion. The matrices *h, hx, and hy* contain topographic heights and their derivatives in *x* and *y*.

Function [w,dayk] = ***pointwn****(latstore, longstore, vstore, rmstore, vsestore, rmsestore, monthstore, daystore, hourstore, ut, vt, us, vs, plat, plong, h, hx, hy, timeres)*: This function take the nn X 400 'store' matrices and, using the latitude (plat) and longitude (plong) vectors of points of interest, calculates time series of vertical velocity at the points of interest, storing the date and time in the date number array *dayk*. The routine *utrans.m* should be run before this one to give the velocity increments *ut* and *vt*. The arrays *us* and *vs* are the vertical shears (m/s) used to estimate the baroclinic components of the vertical motion. The matrices *h, hx, and hy* contain topographic heights and their derivatives in *x* and *y*.

Function [wq] = ***pointwshortnq****(latstore, longstore, dq, vstore, rmstore, vsestore, rmsestore, ut, vt, us, vs, plat, plong, h, hx, hy, timeres)*: This function take the nn X 400 'store' matrices and, using the latitude (plat) and longitude (plong) vectors of points of interest, calculates time series of vertical velocity multiplied by saturation specific humidity at the points of interest. The routine *utrans.m* should be run before this one to give the velocity increments *ut* and *vt*. The arrays *us* and *vs* are the vertical shears (m/s) used to estimate the baroclinic components of the vertical motion. The matrices *h, hx, and hy* contain topographic heights and their derivatives in *x* and *y*. The matrix *dq* contains the saturation specific humidity at 900 hPa.

Function [wq,dayk] = ***pointwshortnqdx****(latstore,longstore,dq,vstore,rmstore, datestore, vsestore,rmsestore,ut,vt,us,vs,plat,plong,h,hx,hy,timeres,wrad)*: Same as *pointshortnq* but also returns *dayk*, a matlab datenumber array corresponding to the time series of *wq*. In this case, *plat* and *plong* can be either scalars or 1-D vectors. Note that *datestore* is the input matlab datenumber array corresponding to the dates and times of the other variables.

Function [wq,dayk] = ***pointwq****(latstore, longstore, dq, vstore, rmstore, vsestore, rmsestore, ut, vt, us, vs, plat, plong, h, hx, hy, timeres)*: This function take the nn X 400 'store' matrices and, using the latitude (plat) and longitude (plong) vectors of points of interest, calculates time series of vertical velocity multiplied by saturation specific humidity at the points of interest, storing the date and time in the datenumber array *dayk*. The routine *utrans.m* should be run before this one to give the velocity increments *ut* and *vt*. The arrays *us* and *vs* are the vertical shears (m/s) used to estimate the baroclinic components of the vertical motion. The matrices *h, hx, and hy* contain topographic heights and their derivatives in *x* and *y*. The matrix *dq* contains the saturation specific humidity at 900 hPa.

Function [rain,rainrate,dayk] = ***raingen****(plat,plong)*: This function acts on the active event set (as contained in *temp.m*) and for specified latitudes *plat* and longitudes *plong* returns the storm total rainfall (*rain*, in mm) and times series of rain rate (*rainrate*, in mm/hr). The matrix *dayk* contains the matlab datenumbers corresponding to the *rainrate* matrix. *plat* and *plong* can be scalars or 1-D vectors containing the latitudes and longitudes of the point(s) of interest. Note that this function can take a while to run if *plat* and *plong* are long vectors.

Function [V] = ***windprofilem****(vm, rm, vm2, rm2, r, wp):* This function return radial profiles of azimuthal wind, *V*, given matrices containing the maximum circular wind speed, *vm*, the radii of maximum

wind, *rm*, the maximum circular wind speed of any secondary eyewalls present (0 if not present), *vm2*, the radii of maximum winds of any secondary eyewalls present (0 if not present), *rm2*, the distances *r* of each event from the points of interest, and the wind profile *wp* (see below; must be equal to 1, 2 or 3).

Function [V] = ***windprofiles****(vm, rm, r, wp):* This function return radial profiles of azimuthal wind, *V*, given matrices containing the maximum circular wind speed, *vm*, the radii of maximum wind, *rm*, the distances *r* of each event from the points of interest, and the wind profile *wp* (see below; must be equal to 1, 2 or 3). This function does not account for secondary eyewalls (see *windprofilem* above).

Function [pass vmax] = ***best_filter*** *(vsm, latm, longm, city_radius, clat, clong)*: Thus function finds those best tracks that pass within city_radius of [clat, clong] and also finds the maximum wind speed (vmax) within that circle. Pass= 1 if track meets the criterion, 0 if it does not. Use *best_filter_poly* if line segment filtering is desired.

Function [pass vmax]= ***best_filter_poly*** *(latm,longm,vsm)* Filters tracks that pass through line segments specified in the file 'poly.in', and also finds the maximum wind speed of storms at the nearest observation time prior to passing over the line segment.

Function [pass vmax]= ***best_filter_poly_us*** *(latm,longm,vsm)* Filters tracks that pass through U.S. Gulf and east coastal line segments specified in the file 'polynet.in', and also finds the maximum wind speed of storms at the nearest observation time prior to passing over the line segment.

Function [pdiland, jland] = ***landfallpdi****(lat, long, v, jmax)*: Determines power dissipation summed on first land fall points, using bathymetry data set.

Function [landtemp] = ***landcalc****(x,y)*: Calculates whether a point (x,y) is over land or ocean based on a bathymetry/topography data set.

Function [polytype passi passe xint yint jint kint kfrac] = ***polyfilter****(lat,long)*: This script finds whether storm tracks cross line segments given in *'poly.in'*. *Passe*= 1 if track meets the criterion, 0 if it does not. Also finds whether each point on a track lies entirely within a set of line segments if they constitute a closed polygon. If so, *passi* is set to unity, otherwise to zero. (Note that *passi* = 1 by default if line segments are not a closed polygon. Also note that *passi* is a 2-D matrix and that *passe* is a 1-D vector.)
Polytype is '*closed*' for closed polygons, '*open*' otherwise.

The quantities *xint* and *yint* constitute the intersection points, along the line segment numbered *jint* and the *kint*'th point along the track. *kfrac* is the fraction of the distance between track points i and i+1 that the intersection occurs.

Note that *passe* is a vector of length *n*, where *n* is the number of tracks, and the other quantities are matrices of dimension (*n,maxtimes*), where *maxtimes* is the specified upper limit on the number of times a particular track may intersect the set of line segments.

Function [rfac] = ***outer****(n,m)*: This uses the Chavas and Emanuel (2010) work to make a random draw from a suitable log-normal distribution to rescale the outer radius. The output non-dimensional factor *rfac* is the outer radius scale factor, which has a mean of 1. Scale all outer radii by this random number, and all radii of maximum winds by this number squared (according to Emanuel and Rotunno, 2011, but first power seems more reasonable). Note that parameters are specified here rather than in *params.m*.

Function [q900 q600] = ***qs900****(T600)*: Calculates saturation specific humidity at 600 hPa and saturation specific humidity at 950 hPa given 600 hPa T (K) and assuming a moist adiabatic lapse rate. (Note: In spite of the variable and function name, we currently calculate saturation specific humidity at 950 hPa.