Iterative Rain Cell Tracking Software Manual

Christopher Moseley, Jan Haerter

February 9, 2020

1 License

The Iterative Rain cell Tracking (IRT) software can be distributed and modified freely for non-commercial use. Reference for the tracking method:

- C. Moseley, O. Henneberg, and J. O. Haerter (2019): A statistical model for isolated convective precipitation events. Journal of Advances in Modeling Earth Systems, **11**, 360-375, doi:10.1029/2018MS001383

2 Software Requirements

- It is recommended to use the IRT code on a **Linux or UNIX system**.
- **A standard Fortan 90 compiler:** The IRT code is written completely in Fortran 90. It can be compiled by any standard fortran compiler, e.g. the GNU compiler "gfortran". The code is currently not paralellized.
- **Climate Data Operators (CDO):** The IRT software uses the SERVICE format (file ending ".srv") for input and output data sets, since this format is easy to handle in Fortan 90 and no additional libraries have to be included. As data sets are usually given in more wide spread formats like NetCDF, they have to be converted to SERVICE first. This is conveniently possible with the CDO software. It is freely available through https://code.mpimet.mpg.de/projects/cdo/

3 Files included in the IRT package

irt_objects_release.f90
The Fortran 90 code of the program that identifies spatially connected objects for every time step independently, and establishes links between objects of subsequent time steps. Input data in SERVICE format: The data file of the tracked field, additional field which are to be averaged over the track life cycles (optionally), and the field of the advection velocity on a coarse grid (optionally). Output: Object data (ASCII format), mask of object IDs (SERVICE format).

irt_advection_field_release.f90
The Fortran 90 code of the program that generated the field of the advection velocity on a coarser spatial and temporal grid that is used for the next iteration of irt_objects_release.f90. Input data: The object data output file of by irt_objects_release.f90. Output data: The coarse velocity field (SERVICE format).

irt_tracks_release.f90
The Fortran 90 code of the program that forms tracks out of the identified objects. Input data: The

object data output file of by irt_objects_release.f90. Output data in ASCII format: Track data with all available information and a header line of every track, and track data with reduced information which is used as input for the program irt_trackmask_release.f90.

irt_trackmask_release.f90
The Fortran 90 code of the program that generates a mask with track IDs in SERVICE format. Input data: The (sorted) output of reduced information track data from irt_tracks_release.f90, and the field of object IDs generated by irt_objects_release.f90.

irt_tracklinks_release.f90
The Fortran 90 code of a program that finds the "parent" tracks of a track that was initiated by merging or splitting, and the "child" tracks of a track that was terminated by merging or splitting. Input data: Output of irt_tracks_release.f90. Output: Track data with extender header.

compile.sh
A shell script to compile the code.

iterate.sh
A shell script to perform one IRT iteration automatically.

irt_objects_input_00.srv.gz
A tutorial dataset for testing the IRT code.

copy_srv.f90
Example program to demonstrate how to handle SRV files in Fortran. It simply reads in an input file in SRV format and copies it to an output file.

4 Setting the parameters

The parameters for the tracking have to be written hard-coded into the Fortran 90 file irt_parameters.f90. This file has to be modified first, before the code can be compiled. The following parameters have to be set:

Domain size:
The domain size longitudinal and latitudinal direction is given by the parameters *domainsize_x* and *domainsize_y*, respectively. The domain size of an input field can be found e.g. by the calling the CDO command
cdo griddes example_dataset.nc .
and the domainsize must be set according to the values of xsize and ysize in the output of the CDO command, i.e.
INTEGER, PARAMETER :: domainsize_x = 960
INTEGER, PARAMETER :: domainsize_y = 960
in the case of the tutorial dataset.

Grid information:
This is important for the calculation of the object areas in the first place. The switch
LOGICAL, PARAMETER :: llonlatgrid = .TRUE.
states that the grid is given in geographical longitudes and latitudes. In this case the grid information has to provided by three parameters given in degrees, containing the latitude of the southern edge of

the domain, and the grid spacing in zonal and meridional direction:
REAL, PARAMETER :: lat_first = -45.
REAL, PARAMETER :: lat_inc = 0.1
REAL, PARAMETER :: lon_inc = 0.1
In this case, the scale parameter
REAL, PARAMETER :: unit_area = 12345.
defines the area of a square measuring 1° in zonal and meridional direction (i.e. a value 12345 as in the example would measure the areas in km^2).

If *llonlatgrid* is set to *.FALSE.* all grid boxes have an equal area given by *unit_area* (e.g. in an idealized model output with periodic boundary conditions). in this case, the parameters *lat_first, lat_inc, lon_inc* have to be set to arbitrary values, but have no effect on the tracking output.

Switch for periodic/non-periodic boundary conditions:
If periodic boundary conditions should be applied in x-direction, the switch *lperiodic_x* has to be set to
LOGICAL, PARAMETER :: lperiodic_x = .TRUE.
Otherwise, it has to be set to
LOGICAL, PARAMETER :: lperiodic_x = .FALSE.
Likewise, the switch *lperiodic_y* has to be set for periodicity in y-direction.

Cutoff value for the tracking field:
A minimal *threshold* value has to be defined that decides if a grid point should be counted as belonging to an object or not. E.g. in the case of rain intensity, a grid point must have at least the intensity given by *threshold* to be counted as a precipitating patch. If precipitation intensity given in mm/h is tracked,
REAL, PARAMETER :: threshold=1.0
could be a reasonable choice. The value is required be the program *irt_objects_release.f90.*

Minimum size of a patch:
Patches which are smaller than *minimum_size* grid points will be neglected. A reasonable choice may be: *REAL, PARAMETER :: minimum_size=4.*

Grid description of the coarse velocity field:
The coarse advection velocity field is not needed for the first run of *irt_objects_release.f90* (i.e. the first iteration of IRT), but for all subsequent iterations. It is given in grid points per time interval. To define the spatial and temporal grid of the velocity field, the following parameters have to be set:

- The number of time steps in the input dataset can be found e.g. by running the CDO command *cdo ntime example_dataset.nc* ,
  and the parameter *time_steps* has to be set accordingly:
  INTEGER, PARAMETER :: time_steps = 120

- The number of time steps of the velocity field is *nt_bins*, and the number of grid boxes in x- and y-direction are *nx_bins* and *ny_bins*, e.g.
  INTEGER, PARAMETER :: nt_bins = 10
  INTEGER, PARAMETER :: nx_bins = 2
  INTEGER, PARAMETER :: ny_bins = 2
  In this case, the domain will be divided into 2×2 equally spaced squares. The whole time series will be divided into *nt_bins*= 10 equally long time intervals, therefore the temporal resolution of the velocity dataset is one hour (the input dataset has 120 time steps with 5 minutes interval,

i.e. is in total 10 hours long).

Maximum velocity:
The parameter *max_velocity* is a threshold with the intent to filter out events with unrealisticly high diagnosed velocities. This filters out artifacts that could e.g. happen when patched which do not actually belong to the same event accidently overlap. A reasonable value could be
REAL, PARAMETER :: max_velocity = 10.

Minimum sample size for advection velocity:
If the sample size for the diagnostics of advection velocity is too small, the mans value could be too noisy. The parameter *min_cells* gives the minimum sample size of cells that are required to assign a mean value to a velocity grid box. If the sample size is smaller, then the velocity of the grid box will be interpolated from the neighbouring grid boxes. A reasonable value could be
INTEGER, PARAMETER :: min_cells = 10

Additional averaging fields:
Besides reading in only the tracking field (i.e. rain intensity in case of the tutorial dataset), there is the option to read in additional fields that will be averaged over the track life cycles. This is controlled by the parameter *n_fields*. If the tracking field should be the only input file, then it should be set to 0:
INTEGER, PARAMETER :: n_fields = 0
If we want to study e.g. the life cycles of the convergence field inside the tutorial file *example_dataset.nc*, we can set n_fields= 1. **Controlling the sensitivity of tracks to merging and splitting events:** The parameter *threshold_ratio* controls if a merging/splitting event terminates tracks or not, and must have a value between 0 and 1. If *threshold_ratio*= 0, every such event leads to termination of all involved track. If *threshold_ratio*= 1, at every merging/spitting event the largest involved track is not terminated, while all others terminate. A more detailed descrition is given in Moseley et al. (2019). Example:
REAL, PARAMETER :: threshold_ratio=0.5

Buffer sizes:
The following parameters are just buffer sizes and do not affect the tracking results. They have to be increased if necessary.

- The maximum number of objects in a single time step: *max_no_of_cells*
- The maximum length of a track: *max_length_of_track*
- The maximum number of tracks at a single time step: *max_no_of_tracks*

Missing value:
The parameter *miss* defines the missing value. If a value is less or equal than *miss* will be regarded as missing value. By default we use:
REAL, PARAMETER :: miss = -9999.

5 Compiling the code

After all parameters have to be adjusted in the Fotran 90 files, the code can be compiled. All code files can be compiled independently. The shell script *compile.sh* compiles all four Fortran 90 files with the GNU compiler "gfortran". If you use another compiler, you can adjust the variable *COMPILE_COMMAND* in the script. The executables will be generated with the file ending ".x".

6 Running the tracking

In the following, we apply the tracking on the tutorial dataset *example_dataset.nc*.

Preparing input datasets:
The NetCDF file *example_dataset.nc* contains the variables *r_int* (the rain intensity given in $[mm/h]$) and *conv_h_low* (the moisture convergence below 2000 m given in $[mm/s]$). It is given on a 960×960 grid with periodic boundary conditions and contains 120 time steps.

We use the intensity field as tracking field and the convergence field as additional averaging field. Using CDO, the fields should first be separated and converted into SERVICE format. There are *n_fields*+1 input datsets, and they must have the file names *irt_objects_input_00.srv* for the tracking field (here: rain intensity) and *irt_objects_input_xx.srv* where xx= $01, 02, \ldots$ for the additional fields:
cdo -f srv selvar,r_int example_dataset.nc irt_objects_input_00.srv
cdo -f srv selvar,conv_h_low example_dataset.nc irt_objects_input_01.srv
The tutorial dataset does not contain missing values. If a dataset contains missing values, they must be set to *miss* (-9999 by default) before the conversion. This is possible with the CDO command *setmisstoc*, e.g.:
cdo -f srv setmisstoc,-9999 <filename>.nc irt_objects_input.srv

Note: Some CDO versions will convert the data to 64 bit double precision variables in the SRV file. If you compile the code for 32 bit single precision floating point values, this will cause the code to crash or produce garbage values. In this case, you have to use the option "-b F32" in the CDO commands.

Setting parameters:
Before compiling, the following parameters have to be set as described in section 4.

Running first iteration:

- **Step 1:** The objects have to be identified by calling:
  ./irt_objects_release.x 1
  The argument "1" means that no advection velocity field will be read in. The program reads in the files *irt_objects_input.srv* and *irt_objects_input_conv.srv*, and generates the output files *irt_objects_output.txt*, and *irt_objects_mask.srv*

- **Step 2:** To generate the coarse advection velocity field, call:
  ./irt_advection_field_release.x
  The program reads in the file *irt_objects_output.txt*, and writes the advection velocity field into the file *irt_advection_field.srv* in SERVICE format.

- **Step 3:** Combining the identified objects to tracks is done by calling the program:
  ./irt_tracks_release.x
  It reads in the file *irt_objects_output.txt* and writes two output files: *irt_tracks_output.txt* containing the full tracks information, and the auxiliary file *irt_tracks_nohead_output.txt* with reduced information. Then, add track links by calling:
  ./irt_tracklinks_release.x
  It reads in the file *irt_tracks_output.txt*, and writes out the file *irt_tracklinks_output.txt* with extended track headers.

- **Step 4:** The final step is to generate the SERVICE file with the mask of track IDs. Therefore, two steps have to be done: First, the auxiliary file *irt_tracks_nohead_output.txt* has to be sorted with respect to time step (second column). With a Linux system, this can by done e.g by using the "sort" command:
  sort -n -k2 irt_tracks_nohead_output.txt > irt_tracks_sorted.txt
  In the second step, call the program:
  ./irt_trackmask_release.x
  It reads in the sorted file *irt_tracks_sorted.txt* and the mask of object IDs *irt_objects_mask.srv* and writes out the SERVICE file *irt_tracks_mask.srv*.

Running second and subsequent iterations:
After the first iteration has been run, the file *irt_advection_field.srv* exists which is needed to perform the second iteration. This can be done by calling: *./irt_objects_release.x 2*
Here, the argument "2" means that the velocity field will be read in and applied for the identification of overlaps between objects of consecutive time steps. Steps 2 to 4 are the same as for the first iteration. The procedure can be iterated until the velocity field converges. Note: In principle, it is possible to perform the second and all further iteration already after step 2, steps 3 and 4 do not have to be run every time necessarily.

Shell script for the iteration process:
The shell script *iterate.sh* performs all the above mentioned steps automatically, and saves the tracking result of every iteration in separate files. Call the script with: *./iterate.sh*

7 Format of the output files

This section explains the output files which are necessary for the analysis and visualization of the tracking results.

Format of the objects data output file *irt_objects_output.txt*
The data file is in text format. Each line contains information of one identified object (connected patch). The number of colums is $23+2\times n\_fields$, where *n_fields* is the parameter defining the number of additional input fields as described in section 4. The columns contain the following information:

1. Time step
2. Object ID1: ID of the object for the given time step (counting begins with 1 at every new time step). If 0, it means that no object was found at the given time step, and all values in the other columns will be set to 0.
3. Object ID2: ID of the object for the whole time series (counting is continued from one time step to the next)
4. Age1: Age of the cell (in time steps). A cell merger inherits the age of the *oldest* cell involved in the merging.
5. Age2: Age of the cell (in time steps). A cell merger inherits the age of the *largest* cell involved in the merging. Note that Age2$\leq$Age1.
6. Area of the object in the units given in the grid information

7. Mean value of the tracked field, averaged over the object's area. If *n_fields*> 0, then the next *n_fields* columns contain area-averages over the other input fields.

8. Minimum value of the tracked field within the object's area. If *n_fields*> 0, then the next *n_fields* columns contain minimum values over the other input fields within the object.

9. Maximum value of the tracked field within the object's area. If *n_fields*> 0, then the next *n_fields* columns contain maximum values over the other input fields within the object.

10. x coordinate (in grid boxes, starting with 1) of the object's most western point (i.e. smallest x-coordinate)

11. x coordinate of the object's most eastern point (i.e. largest x-coordinate)

12. y coordinate of the object's most southern point (i.e. smallest y-coordinate)

13. y coordinate of the object's most northern point (i.e. largest y-coordinate)

14. x coordinate of the weighted center of mass of the object

15. y coordinate of the weighted center of mass of the object

16. x component of the object's estimated velocity given in grid points per time step. Missing value if not detectable.

17. y component of the object's estimated velocity given in grid points per time step. Missing value if not detectable.

18. ID2 of the largest object of the next time step that overlaps with this object. If 0, no such object exists.

19. Area of the object given in the column before

20. ID2 of the second largest object of the next time step that overlaps with this object. If 0, no such object exists.

21. Area of the object given in the column before

22. ID2 of the largest object of the previous time step that overlaps with this object. If 0, no such object exists.

23. Area of the object given in the column before

24. ID2 of the second largest object of the previous time step that overlaps with this object. If 0, no such object exists.

25. Area of the object given in the column before

Format of the tracks data output file *irt_tracks_output.txt* and *irt_tracklinks_output.txt*
The data file is in text format. Each track consists of a header line, followed by a track body. Tracks are separated by a dummy line containing the character *. A **track header line** consists of 5 or 9 columns (in *irt_tracks_output.txt* or *irt_tracklinks_output.txt*, respectively) containing the following information:

1. Track ID

2. Time step when track begins

3. Duration of the track in time steps

4. Code labeling the situation at the initiation of the track:
   0: Track emerges by itself
   1: Track is fragment of a splitting event
   2: Track is result of merging event
   -1: Track initiated by contact with missing values

5. Code labeling the situation at the termination of the track:
   0: Track dissipated
   1: Track terminated by merging event
   2: Track terminated by splitting event
   -1: Track terminated by contact with missing values

6. (Only in *irt_tracklinks_output.txt*) ID of largest "parent track" in case the track was initiated by merging or splitting. Zero if non-existent.

7. (Only in *irt_tracklinks_output.txt*) ID of second largest "parent track". Zero if non-existent.

8. (Only in *irt_tracklinks_output.txt*) ID of largest "child track" in case the track was terminated by merging or splitting. Zero if non-existent.

9. (Only in *irt_tracklinks_output.txt*) ID of second largest "child track". Zero if non-existent.

The *track body* has one line for every time step of the track life time, with $12+2\times$*n_fields* columns containing the following information:

1. Track ID

2. Time step

3. Object ID1

4. Object ID2

5. Object Age1 (in time steps)

6. Object Age2 (in time steps)

7. Area of the object in grid points

8. Mean value of the tracked field, averaged over the object's area. If *n_fields*> 0, then the next *n_fields* columns contain area-averages over the other input fields.

9. Minimum value of the tracked field within the object's area. If *n_fields*> 0, then the next *n_fields* columns contain minimum values over the other input fields within the object.

10. Maximum value of the tracked field within the object's area. If *n_fields*> 0, then the next *n_fields* columns contain maximum values over the other input fields within the object.

11. x coordinate (in grid boxes, starting with 1) of the object's most western point (i.e. smallest x-coordinate)

12. x coordinate of the object's most eastern point (i.e. largest x-coordinate)

13. y coordinate of the object's most southern point (i.e. smallest y-coordinate)

14. y coordinate of the object's most northern point (i.e. largest y-coordinate)

15. x coordinate of the weighted center of mass of the object
16. y coordinate of the weighted center of mass of the object
17. x component of the object's estimated velocity given in grid points per time step. Missing value if not detectable (usually at the end of the track).
18. y component of the object's estimated velocity given in grid points per time step. Missing value if not detectable (usually at the end of the track).
19. ID2 of the largest object of the next time step that overlaps with this object. If 0, no such object exists.
20. Area of the object given in the column before
21. ID2 of the largest object of the next time step that overlaps with this object. If 0, no such object exists.
22. ID2 of the second largest object of the next time step that overlaps with this object. If 0, no such object exists.
23. ID2 of the largest object of the previous time step that overlaps with this object. If 0, no such object exists.
24. ID2 of the second largest object of the previous time step that overlaps with this object. If 0, no such object exists.

Mask files in SERVICE format

- The file *irt_objects_mask.srv* contains a mask of the identified objects and marks them with the object ID1. Objects which are neglected because they are too small, or because they touch missing values, are indicated by the value -1.
- The file *irt_tracks_mask.srv* constains a mask of all objects and marks them with the track ID. Objects which belong to tracks that are neglected because they are only 1 time step long, or because they touch missing values, are indicated by the value -1. Modify the program *irt_trackmask_release.f90* if you wish to label the objects differently (e.g. by object age, or track type).
- The masks can be converted to NetCDF e.g. by the CDO command
  cdo -f nc copy irt_objects_mask.srv irt_objects_mask.nc
  cdo -f nc copy irt_tracks_mask.srv irt_tracks_mask.nc