# Mini-manual on How to use coherent structures tracking data and codes

Version 0.2

Adrián Lozano Durán

May 21, 2014

## Introduction

This document describes very briefly the codes needed to track in time coherent structures and the format of the data generated. The codes are located at `/wamba15/adrian/trabajo/codes/strucs/graphs/`. This manual also includes the main input for each code (i.e., the most important files the codes need to read) although there are more input parameters that have to be provided and are not described here. The starting point to use the codes is the compressed list of pixels of the objects or the velocity and discriminant fields in physical space. The steps are:

1. Compute coherent structures.
2. Compute coherent structures' properties.
3. Compute connections between structures.
4. Compute graphs and branches.
5. Compute statistics for mergers and splits.

All the input and output files are written in binary and big endian.

## Compute coherent structures

- *Codes*: in the folder `clustering/`. For uvsters `uvstering.f90` and for vortex clusters `clustering.f90` using the list of compressed structures and `uvstering_oldstyle.f90` and `clustering_oldstyle.f90` using the physical velocity or discriminant files. they do the labeling process and compute the structures for each flow field.

- *Input*: For `uvstering.f90` and `clustering.f90` the compressed list of pixels. For `uvstering_oldstyle.f90` and `clustering_oldstyle.f90` the streamwise and wall-normal velocity fields in physical space for uvsters or discriminant of the velocity gradient tensor for vortex clusters.

- *Output*: file with the list of pixels of the structures. There is one file for every flow field. All the data is written with integers 16 bits using Fortran direct access. We denote `N_m` as the number of pixels of the object `m`, `i_{n,m},k_{n,m},j_{n,m}` are the n-th streamwise, spanwise and wall-normal pixel coordinates for the object `m`. The variables `nx` and `nz` are the maximum number of points in streamwise and spanwise directions of the flow field. The file is written as follows:

```
nx        nz        0
N_1       0         0         ! object 1
i_{1,1}   k_{1,1}   j_{1,1}
i_{2,1}   k_{2,1}   j_{2,1}
...
i_{N_1,1} k_{N_1,1} j_{N_1,1}
N_2       0         0         ! object 2
i_{1,2}   k_{1,2}   j_{1,2}
```

```
i_{2,2}   k_{2,2}   j_{2,2}
...
i_{N_2,2} k_{N_2,2} j_{N_2,2}
...
...
N_m       0         0         ! object m
i_{1,m}   k_{1,m}   j_{1,m}
i_{2,m}   k_{2,m}   j_{2,m}
...
i_{N_m,m} k_{N_m,m} j_{N_m,m}
...
```

Warning: `N_m` is incorrect for those cases in which the number of pixels of the object cannot be represented with an integer 16 bits.

## Compute coherent structures' properties

- *Code*: in the folder `list/`. `lister.f90` for objects created with the physical fields and `lister_compressed.f90` for the ones computed with the compressed list.
- *Input*: the list of pixels from the previous code.
- *Output*: file with the properties of each object. There is one file for every flow field. For a given object,
  - `jmin`: minimum wall-normal position of the object.
  - `jmax`: maximum wall-normal position.
  - `im`: streamwise position of the center of its circumscribed box.
  - `km`: spanwise position of the center of its circumscribed box.
  - `lx`: streamwise length.
  - `lz`: spanwise length.
  - `u`: mean streamwise velocity fluctuations.
  - `v`: mean wall-normal velocity fluctuations.
  - `uv`: mean Reynolds stress.
  - `w`: mean spanwise velocity fluctuations.
  - `u2`: mean squared streamwise velocity fluctuations.
  - `v2`: mean squared wall-normal velocity fluctuations.
  - `uv2`: mean squared Reynolds stress.
  - `w2`: mean squared spanwise velocity fluctuations.
  - `V`: volume of the object.
  - `Vb`: volume of the circumscribed box.
  - `nc`: number of pixels.
  - `nrec`: pointer to the position of the beginning of the object in the list of pixels.

The variables `jmin`, `jmax`, `im`, `km`, `lx` and `lz` are given in pixels and the rest in outer units. All of them are reals 32 bits except for `nc` and `nrec` which are integers 64 bits. The file is written in Fortran direct access (record length = 21). `N` is the total number of objects. The structure of the file is as follows:

```
jmin1 jmax1 im1 km1 lx1 lz1 u1 v1 uv1 w1 u21 v21 uv21 w21 V1 Vb1 D1 nc1 nrec1
jmin2 jmax2 im2 km2 lx2 lz2 u2 v2 uv2 w2 u22 v22 uv22 w22 V2 Vb2 D2 nc2 nrec2
             ...
jminN jmaxN imN kmN lxN lzN uN vN uvN wN u2N v2N uv2N w2N VN VbN DN ncN nrecN
```

The number at the end of the variable represents the number of the object.

## Compute connections between structures

- *Code*: folder `interclus/version5/`. Compile with `make interclus`. Computes the geometrical intersection of objects at time $t_n$ with those at time $t_{n+1}$, i.e., forward connections. The backwards connections can be easily deduced.
- *Input*: ASCII list with the paths to the files containing the list of pixels of the objects at a given time $t_i$ and in chronological order.
- *Output*: One file containing the connections of the objects at time $t_n$ with those at time $t_{n+1}$ with $n = 1, .., Nt - 1$ where $Nt$ is the total number of files in the temporal series. All numbers are reals 32 bits written in Fortran stream access. Every object can be uniquely identified by the time it belongs to, $t$, and the number of object at that time, $ib$. Nomenclature: for a given time `m`, `ib_i` is the object's number with `i=1,...,nib` and `nib` the total number of objects at time `m`. `n_i` is its total number of forward connections, `ia_{i,j}` with `j=1,...,n_i` are the numbers of the objects at time `m+1` that are connected with `ib_i` and `w_{i,j}` the intersected volume in outer units. `nd` is the total number of objects with no forwards connections. `nf` is the number of the last object at a given time which has forward connections. In general `nf`$\neq$ `nib` The file is written as follows:

```
!---------time 1----------!
ib_1  n_1                           ! object 1
ia_{1,1}  ...  ia_{1,n_1}
w_{1,1}   ...  w_{1,n_1}
ib_2  n_2                           ! object 2
ia_{2,1}  ...  ia_{2,n_2}
w_{2,1}   ...  w_{2,n_2}
...
ib_nf  n_nf                         ! last object
ia_{nf,1}  ...  ia_{nf,n_nf}
w_{nf,1}   ...  w_{nf,n_nf}
nd  0
nib 0
!---------time 2----------!
```

```
...
...
!--------last time---------!
ib_1  n_1
ia_{1,1}  ...  ia_{1,n_1}
w_{1,1}   ...  w_{1,n_1}
ib_2  n_2
ia_{2,1}  ...  ia_{2,n_2}
w_{2,1}   ...  w_{2,n_2}
...
nd  0
nib 0
-100 -100
```

The last two numbers indicate the end of the file.

## Compute graphs and branches

- *Code*: in the folder `evolutions/version2/`, file `evolutions_version2.f90`. It computes branches and graphs.
- *Input*: file with the forwards connections of the structures produced by the previous code and the lists of the properties of the objects at each time.
- *Output*: There are three output files with extension *.branches, *.graphs, *.links.
  - The file *.branches contains information about the branches. The number of a branch is given by the order in which it appears in the file. The data is written in Fortran stream access as follows:

    ```
    ! header
    ne
    lifetimes(1:ne)
    valid(1:ne)
    ! branch 1
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 1
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 2
    ...
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! last time
    ...
    ! branch i
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 1
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 2
    ...
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! last time
    ...
    ! last  branch
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 1
    t  lx  ly  lz  x  y  z  vol  stat i u  v  ! time 2
    ```

```
...
t  lx  ly  lz  x  y  z  vol  stat i u  v  ! last time
```

`ne` is the total number of branches. `lifetimes` is a vector with `ne` elements which contains integers 32 bits with the number of times of each branch. `valid` is a logical vector with 8 bits per element and `ne` elements, one per branch. Its value is false for invalid branches, for instance, when they are very small or spurious. For the branches all the variables are real 32 bits. Every variable within the branch should have a subindex `{i,j}` that it was omitted for clarity where `i` is the number of the branch and `j` the temporal position within it. The branches are formed by only one object at each time and the data above show the information stored at a given time for each object of the branch. The variables are:

∗ t: integer representing the number of the file that identifies the temporal position the object.

∗ lx: streamwise length of the object.

∗ ly: wall-normal length of the object.

∗ lz: spanwise length of the object.

∗ x: streamwise position of the center of its circumscribed box.

∗ y: wall-normal position of the center of its circumscribed box.

∗ z: spanwise position of the center of its circumscribed box.

∗ vol: volume of the object.

∗ status: its value is 0 when there is no merging or splitting happening, i.e., there is one backward and one forward connection. In the first point of the branch its value is -1 when there are no backward connections and in the last point is 1 if there are not forwards connections. Primary branches are those whose edges are -1 and 1 respectively.

∗ i: number of the object.

∗ u: mean streamwise velocity fluctuations.

∗ v: mean wall-normal velocity fluctuations.

All the variables are in outer units. The variable ”t” can be converted to physical time using the time stored in the flow fields or in the compressed lists. The variables (t,i) can be understood as the ID of the objects since they identify it unambiguously.

– The file *.graphs contains the branches that form each graph. Every graph is formed by `ni` branches with numbers `ibranch_1` to `ibranch_ni` (the order in which they appear in the file *.branches). There are in total `ng` graphs. The variables are integers 32 bits and are written in Fortran stream access. The file is organized as follows:

```
! header
ng
n_branches(1:ng)
! graph 1
ibranch_1 ibranch_2 ... ibranch_n1
...
```

```
! graph i
ibranch_1 ibranch_2 ... ibranch_ni
...
! graph ng
ibranch_1 ibranch_2 ... ibranch_nng
```

where `n_branches` is a vector of integers 32 bits with `ng` elements that contains the number of branches per graph.

– The file *.links contains how the branches are connected. The file is written with sequential unformatted Fortran as follows:

```
! number of branches
X
ne
X
! links for branch 1
X ibranch_1  ibranch_2 ... X ! backward links
X ibranch_1  ibranch_2 ... X ! forward links
...
! links for branch ne
X ibranch_1  ibranch_2 ... X ! backward links
X ibranch_1  ibranch_2 ... X ! forward links
```

`X` denotes que integers written by fortran when using sequential access and `ibranch_i` the id of the branches connected to a given branch backwards and forwards.

## Compute statistics for mergers and splits

- *Code*: in the folder `stats/mergesplits/`, file `mergesplitstats.f90`. It computes statistics for mergers and splits.
- *Input*: Files *.branches and *.links generated by the previous code.
- *Output*: File with extension *.ms which is divided in three parts. The first two parts have information about the three objects involved in a merger or split. They will be denoted with the subindices s (for the small fragment), m (for the medium fragment) and l (for the large one) (see paper for more details). The first part has information about the splits and the second one about the mergers. The last part has information related to mergers and splits from the point of view of the branch. The number of splits is `nsplits`, the number of mergers `nmergers` and the number of branches `ne`. The output is written using Fortran sequential access as follows:

```
!------------------part I-------------------!
ns typeB(1:nsplits) ns        ! integer 32 bits
ns Dt(1:nsplits)    ns        ! real 32 bits
ns vol_b(1:nsplits) ns        ! real 32 bits
ns vol_m(1:nsplits) ns        ! real 32 bits
ns vol_s(1:nsplits) ns        ! real 32 bits
ns typeQ(1:nsplits) ns        ! real 32 bits
ns coord_b(1:2,1:nsplits) ns  ! integer 32 bits
ns coord_m(1:2,1:nsplits) ns  ! integer 32 bits
ns coord_s(1:2,1:nsplits) ns  ! integer 32 bits
ns y_b(1:nsplits) ns          ! real 32 bits
ns y_m(1:nsplits) ns          ! real 32 bits
ns y_s(1:nsplits) ns          ! real 32 bits
ns l_b(1:nsplits) ns          ! real 32 bits
ns l_m(1:nsplits) ns          ! real 32 bits
ns l_s(1:nsplits) ns          ! real 32 bits
ns Dx(1:nsplits)  ns          ! real 32 bits
ns Dz(1:nsplits)  ns          ! real 32 bits
ns Dy(1:nsplits)  ns          ! real 32 bits
ns ymin_b(1:nsplits) ns       ! real 32 bits
ns ymin_m(1:nsplits) ns       ! real 32 bits
ns ymin_s(1:nsplits) ns       ! real 32 bits
ns ymax_b(1:nsplits) ns       ! real 32 bits
ns ymax_m(1:nsplits) ns       ! real 32 bits
ns ymax_s(1:nsplits) ns       ! real 32 bits
4ns_l T_l(1:ns_l) 4ns_l       ! real 32 bits
4ns_v T_v(1:ns_v) 4ns_v       ! real 32 bits
4ns_l l_l(1:ns_l) 4ns_l       ! real 32 bits
4ns_v l_v(1:ns_v) 4ns_v       ! real 32 bits

!------------------part II------------------!
nm typeB(1:nmergers) nm       ! integer 32 bits
nm Dt(1:nmergers)    nm       ! real 32 bits
nm vol_b(1:nmergers) nm       ! real 32 bits
nm vol_m(1:nmergers) nm       ! real 32 bits
nm vol_s(1:nmergers) nm       ! real 32 bits
nm typeQ(1:nmergers) nm       ! real 32 bits
nm coord_b(1:2,1:nmergers) nm ! integer 32 bits
nm coord_m(1:2,1:nmergers) nm ! integer 32 bits
nm coord_s(1:2,1:nmergers) nm ! integer 32 bits
nm y_b(1:nmergers) nm         ! real 32 bits
nm y_m(1:nmergers) nm         ! real 32 bits
nm y_s(1:nmergers) nm         ! real 32 bits
nm l_b(1:nmergers) nm         ! real 32 bits
nm l_m(1:nmergers) nm         ! real 32 bits
nm l_s(1:nmergers) nm         ! real 32 bits
nm Dx(1:nmergers)  nm         ! real 32 bits
nm Dz(1:nmergers)  nm         ! real 32 bits
nm Dy(1:nmergers)  nm         ! real 32 bits
nm ymin_b(1:nmergers) nm      ! real 32 bits
nm ymin_m(1:nmergers) nm      ! real 32 bits
nm ymin_s(1:nmergers) nm      ! real 32 bits
nm ymax_b(1:nmergers) nm      ! real 32 bits
nm ymax_m(1:nmergers) nm      ! real 32 bits
nm ymax_s(1:nmergers) nm      ! real 32 bits
4nm_l T_l(1:nm_l) 4nm_l       ! real 32 bits
4nm_v T_v(1:nm_v) 4nm_v       ! real 32 bits
```

```
4nm_l l_l(1:nm_l) 4nm_l       ! real 32 bits
4nm_v l_v(1:nm_v) 4nm_v       ! real 32 bits

!-----------------part III-----------------!
nb typeB   nb                 ! integer 32 bits
nb DT      nb                 ! real 32 bits
nb vol_s   nb                 ! real 32 bits
nb IT      nb                 ! real 32 bits
nb vol_m   nb                 ! real 32 bits
nb lx      nb                 ! real 32 bits
nb ly      nb                 ! real 32 bits
nb lz      nb                 ! real 32 bits
nb vol     nb                 ! real 32 bits
nb lxmax   nb                 ! real 32 bits
nb lymax   nb                 ! real 32 bits
nb lzmax   nb                 ! real 32 bits
nb volmax  nb                 ! real 32 bits
nb num_mer nb                 ! integer 32 bits
nb num_spl nb                 ! integer 32 bits
nb yc      nb                 ! real 32 bits
nb typeY   nb                 ! integer 32 bits
```

where `ns`, `nm` and `nb` are integers 32 bits with values 4*`nsplits`, 4*`nmergers` and 4*`ne` respectively and are the consequence of the Fortran sequential access. Next we summarize the meaning of each variable for both part I and II:

- `typeB`: type of branch the merger or the split belongs to. Its value is 1 for primary branches and 0 otherwise.
- `Dt`: times elapsed from the beginning of the branch to the times of the merging or splitting.
- `vol_b`: volume of objects `b`.
- `vol_m`: volume of objects `m`.
- `vol_s`: volume of objects `s`.
- `typeQ`: type of Q-branch. Its value is 1 for Q1, 2 for Q2, 3 for Q3 and 4 for Q4.
- `coord_b`: ID of the object `b`, $(t, i)$, where $t$ is the number of its flow field and $i$ the objects' number in within it.
- `coord_m`: same as before for `m` objects.
- `coord_s`: same as before for `s` objects.
- `y_b`: wall-normal position of the center of gravity of the circumscribed box for objects `b`.
- `y_m`: same as before for `m` objects.
- `y_s`: same as before for `s` objects.
- `l_b`: length of the diagonal of the circumscribed box for objects `b`.
- `l_m`: same as before for `m` objects.

- `l_s`: same as before for `s` objects.
- `Dx`: relative distance in streamwise direction between the center of the boxes of objects `m` and `s`.
- `Dz`: relative distance in spanwise direction.
- `Dy`: relative distance in wall-normal direction.
- `ymin_b`: minimum wall-normal height of objects `b`.
- `ymin_m`: same as before for `m` objects.
- `ymin_s`: same as before for `s` objects.
- `ymax_b`: maximum wall-normal height of objects `b`.
- `ymax_m`: same as before for `m` objects.
- `ymax_s`: same as before for `s` objects.
- `T_l`: time elapsed between inertial mergers or splits.
- `T_v`: time elapsed between viscous mergers or splits.
- `l_l`: size of the objects in an inertial merge or split (see paper for definition).
- `l_v`: size of the objects in a viscous merge or split.

For part III:

- `typeB`: type of the branch. Its value is 1 for primary branches, -1 for not valid branches and 0 otherwise.
- `DT`: not used.
- `vol_s`: total volume lost in the branch by the splits.
- `IT`: not used.
- `vol_m`: total volume gain in the branch by the mergers.
- `lx`: average streamwise length of the branch.
- `ly`: average wall-normal length of the branch.
- `lz`: average spanwise length of the branch.
- `vol`: average volume of the branch.
- `lxmax`: maximum streamwise length of the branch.
- `lymax`: maximum wall-normal length of the branch.
- `lzmax`: maximum spanwise length of the branch.
- `volmax`: maximum volume of the branch.
- `num_mer`: number of mergers in the branch.
- `num_spl`: number of splits in the branch.
- `yc`: average center of gravity of the branch.
- `typeY`: type of branch with respect to the wall. Its value is 3 for tall attached, 2 for detached and 1 for buffer layer.