# SCMAGNET: Input file user guide

Daniele Placido

August 2020

## Contents

# 1 Introduction

New Python version of 4C code input file are .xlsx file that user should compile before run the simulation.

There are five kinds of input file: the global ones used to set the simulation features common to all the conductors (file Transitory_Input.xlsx, section 2); input file useful to build code objects, both conductors and their elementary components such as fluid channels, strands and jackets, like conductor_definition.xlsx, conductor_1_grid.xlsx, conductor_1_input.xlsx and conductor_1_operation.xlsx (addressed in section 3); file that describes the interfaces between channels in conductors and interfaces between conductors both topologically and physically (contact perimeters and heat transfer coefficients), file conductor_1_coupling.xlsx has all this info about conductors channels and it is described in section 4; file with tables to perform interpolation in both time and space or only in time to evaluate some useful parameters like magnetic field or external heat (file dedicated to this task are alphab_dummy.xlsx, bfield.xlsx, flow_dummy.xlsx, I_file.xlsx, Q_file_dummy.xlsx, strain_dummy.xlsx, which are the topic of section 5); finally conductors_diagnostic.xlsx (see section 6) belongs to the last class of input file, output settings, that allows user to store the simulation status at a fixed instant of time and save the time evolution of selected variables at a given spatial coordinates chosen by the user.

In this document, we will go through all these kinds of input file describing their main features, were and how they are used, providing also a short user guide to describe how this input file work and how to correctly compile them.

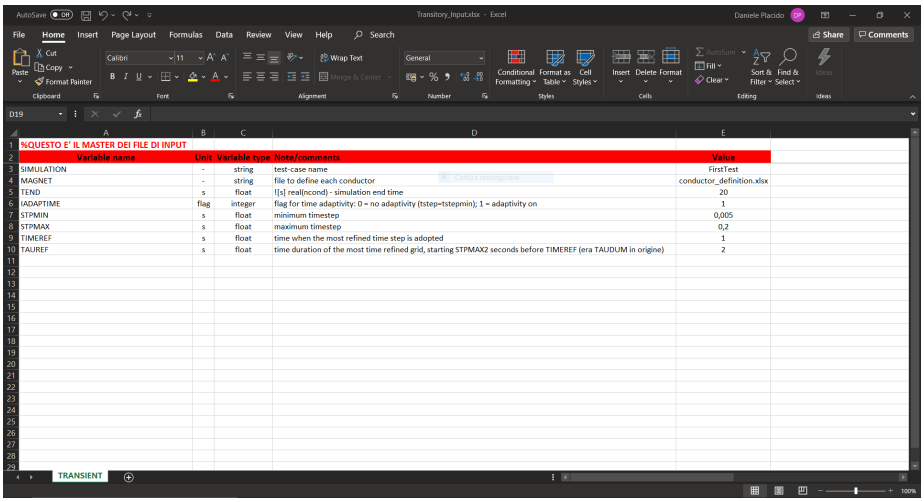
# 2 Global simulation setup: Transitory_Input.xlsx

This kind of input file is needed to set simulation parameters and flags common to all the conductors used in the simulation. An example of this file is shown in figure 1.

## 2.1 Main features

This workbook has a single sheet named **TRANSIENT** with five columns and a ten rows (at the time being). The second row shows the column headings:

- *Variable name* collects variables name;
- *Unit* shows the corresponding SI units, if any;
- *Variable type* identifies the type of the variable, in this file three types of variable coexists: float, integer and string;
- *Note/comments* where user comments related on that variable can be written;
- *Value* is the column devoted to input values and it should be carefully compiled by the user.



%QUESTO E' IL MASTER DEI FILE DI INPUT

| Variable name | Unit | Variable type | Note/comments | Value |
|---|---|---|---|---|
| SIMULATION | - | string | test-case name | FirstTest |
| MAGNET | - | string | file to define each conductor | conductor_definition.xlsx |
| TEND | s | float | [s] real(ncond) - simulation end time | 20 |
| IADAPTIME | flag | integer | flag for time adaptivity: 0 = no adaptivity (tstep=tstepmin); 1 = adaptivity on | 1 |
| STPMIN | s | float | minimum timestep | 0,005 |
| STPMAX | s | float | maximum timestep | 0,2 |
| TIMEREF | s | float | time when the most refined time step is adopted | 1 |
| TAUREF | s | float | time duration of the most time refined grid, starting STPMAX2 seconds before TIMEREF (era TAUDUM in origine) | 2 |

Figure 1: Transitory_Input.xlsx input file formatting example. There is only one sheet called **TRANSIENT**. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required.

Cells filled in red have a specific meaning in the input file which is that those cells are locked, i.e. they can not be modified directly by the user. This is an important point and its utility will come out in the next section 2.3.

## 2.2 Where and how

Input file Transitory_Input.xlsx is read in file ConductorStarter.py and loaded in a dictionaty called transitory_input exploiting function InitializationFunctions.py/Read_input_file, as shown by the following line of code.

```
transitoryinputfile = load_workbook('Description_of_Components/
Transitory_Input.xlsx', data_only = True)
sheetInputTransitory = transitoryinputfile['TRANSIENT']

# transient_input dictionary Initialization (cdp, 06/2020)
transient_input = Read_input_file(sheetInputTransitory, init_file = \
"Transitory_Input.xlsx")
...
```

Dictionary keys are read from column *Variable name* while dictionary values came from *Value* and are converted to the python variable type according to column *Variable type*; if a not valid variable type is given an error is raised.

It was mentioned that there are three types of variables, now lets give more details about them. Float variables are used to set the simulation end time, minimum and maximum time steps, the time at which a most refined mesh

is needed and how long it should be used. Integer variable is a flag to set the adaptivity in time. Finally string values are used to give a name to the simulation, which is exploited by the code to organize the Output directory, and to tell the code which is the next input file to be read.

### 2.3 Compile input file

Since the first four headings are used also in other input file heading (see sections 3.1.1, 3.2.1, 3.3.1 and 6.1), those file are linked with Transitory_Input.xlsx, in this way the column heading can be written only once in this file and it is automatically available to all the linked file. User can not edit this cells because they are locked. To modify their content, user should remove the sheet protection ad to do this a password may be required. As said in section 2.1, all the red filled cells are locked and this should minimize both typos and accidental changes by the user.

User can add or delete rows, except for the second one. He may add also columns, but at the time being the code works fine with the current format.

It is crucial that user does not write comments outside the prescribed cells, otherwise errors may arise due to the way in which python gets the maximum and minimum columns and rows values of the sheet. For the same reason, if new rows (or columns) are edited and than removed, it is not sufficient to clean cells content (canc): those cells must be deleted.

As far as simulation name is concerned, it is important to give different names to simulations with different input data to keep Output directory well organized and avoid the risk to overwrite old saved solutions.

## 3 Input file for objects construction

There are several input file dedicated to the objects constructions, their number is given by $2(n+1)$ where $n$ is the number of conductors. Two of them are necessary to build conductors, namely concuctor_definition.xlsx and conductor_grid.xlsx; the other $2n$ contain information for making conductor components (both fluid and solid components) and are called conductor_i_input.xlsx and conductor_i_operation.xlsx for the i-th conductor. In the following these input file are described in detail in dedicated sections.

### 3.1 File conductor_definition.xslx

This file is read after file Transitory_Input.xlsx and it is the starting point to the conductor construction by the code; an example is shown in figure 2 below.

#### 3.1.1 Main features

The first thing that catches the eye is the presence of some cells filled in red, already encountered in section 2.1, which cannot be directly modified by the user.

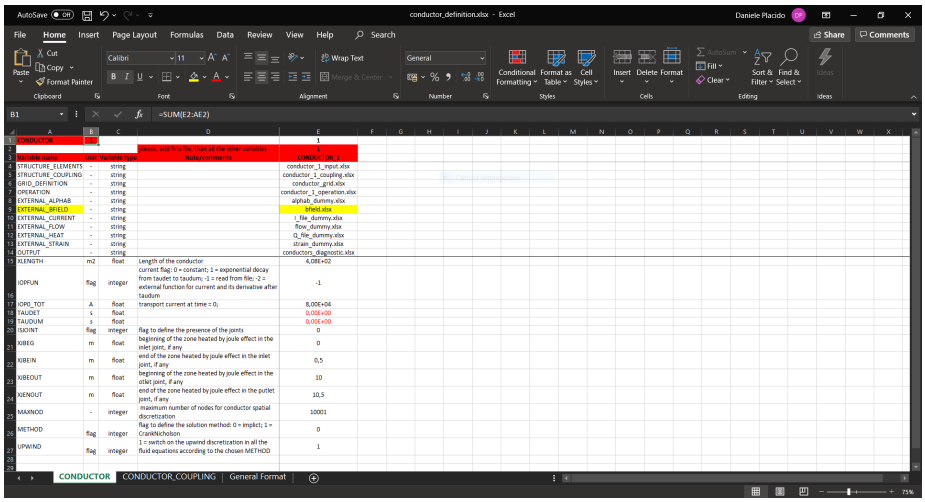



Figure 2: conductor_definition.xslx input file formatting example. There are three sheets but here the focus is on sheet **CONDUCTOR**. Content of some red filled cells (the first four columns of the third row) came form input file Transitory_Input.xlsx, in order to speed up file compilation. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required. The sheet is divided into two parts: the first collects all the file names that should be read and loaded by the code to complete the conductor definition, the second defines conductor geometry, the transported current, wheter or not there are joints, the length of heated zone and flags for numeric.

Cell A1 shows the kind of object, CONDUCTOR in this case, and it is a common feature with the other file mentioned above as it will be cleared in the next sections. Cell B1 shows the total number of conductor objects considered in the simulation and are evaluated by exploiting pairs of cells for each conductor starting form the fifth column. This will be explained in section 3.1.3.

The first four columns of the third row, namely cells A3, B3, C3 and D3 are linked with the cells A2, B2, C2, D2 of sheet **TRANSIENT** of file Transitory_Input.xlsx and represents the column headings. Their meaning is discussed in section 2.1. All the other columns of the third row are constructed combining cell A1 with cells E1, F1 and so on according to the number of conductors to be initialized, invoking excel function textjoin.

As well as in sheet **TRANSIENT** of file Transitoy_Input.xlsx, also in this sheet three types of variables coexist, namely float integer and string. This last kind of variables is used to tell the code what file should be read to complete conductor initialization. Some of them are read by default like conductor_grid.xlsx, conductor_i_input.xlsx and conductor_i_operation.xlsx, others are needed only if some flags are negative as deeply argued in section 5.

Float values are used to initialize the conductors, while integers are flags except for MAXNOD which defines the upper bound to the number of node used in the conductor spatial discretization. There are four flags: one to impose the current trend, one to introduce the presence of joints and two flags to set the numeric, probably the most important ones in this input file. Flag METHOD defines the solution method used to solve the linear system of equations, flag UPWIND. if 1, switch on the upwind discretization in all fluid equations according to the chosen solution method.

It is important to notice that all this variables can be set for each conductor and there is no need to have exactly the same values for different conductors; in other words, different conductors may have different values of input data.

### 3.1.2 Where and how

Sheet **CONDUCTOR** is read by class Conductors method __init__ exploiting function Read_input_file from UtilityFunctions/Auxiliary_functions.py and it is subdivided by this function into two dictionaries. The __init__ lines of code are shown below.

```
...
def __init__(self, basePath, sheetConductorsList, ICOND):

self.ICOND = ICOND
self.NAME = sheetConductorsList.cell(row = 1, column = 1).value
self.ID = f"{self.NAME}_{self.ICOND}"

[self.file_input, self.dict_input] = Read_input_file(
sheetConductorsList, self.ICOND,
init_file = "conductor_definition.xlsx")
...
```

The first dictionaty is called file_input and collects in its key-values pair all the file names to be eventually read and loaded later on by the code. The second one is the dictionary of conductor variables and flags and is called dict_input. Both dictionaries are Conductors class attributes, which means code build these dictionaries for each conductor without overwriting the keys. Dictionaries keys are read from column *Variable name* while dictionary values came from column *CONDUCTOR_I* for the i-th conductor and are converted to the python variable type according to column *Variable type*; if a not valid variable type is given an error is raised.

### 3.1.3 Compile input file

As said in section 2.3, red filled cells can not be edited by the user; moreover the first four column headings came from sheet **TRANSIENT** of file Transitory_Input.xlsx, so if they should be changed user should act on this sheet.

User can add or delete rows except for the third one. If new rows should be added, user is invited to add first file (i.e. string type variables) and than all

the other variables. In this way sheet subdivision into two dictionaries will be preserved.

User can also add new columns if more than one conductor is needed. It is now time to explain how the total number of conductors (cell B1) is computed. For a single conductor, user writes in cell E1 the conductor number which must start from one. In cell E2 there is an excel formula that set cell value to one if the value of cell E1 is larger than zero, and to zero otherwise. If there is a second conductor it is sufficient to select cells E1 to E3 and drag in cells F1 to F3, than change F1 value from one to two and in cell F2 another one will appear. Note also that the new column heading *CONDUCTOR_2* will automatically appear in cell F3 and so on if other conductors should be added. The value in cell B1 is the sum of the values in cells from E2 to AE2, so that at the time being 26 conductors can be defined. This number can be further increased by unprotecting the sheet and changes the second index in the sum function of cell B2.

There are some advantages of this strategy. If user has a lot of conductors for which he has data available but he needs only a fraction of them, he should not delete the ones that are not useful for his simulation; indeed it is sufficient to set values on the first rows to 0 in correspondence to the not needed conductors. Automatically the total number of conductors will be updated and conductors with 0 in the column header will be ignored. In this way there is no need to clear the file, user should just chose the conductors that he needs. Moreover there is no need to have the conductors ID sorted.

Column headings can be achieved simply dragging the content in already existing cells. It is up to user to fill in red the new heading cells and lock them protecting the sheets. It is not mandatory to add a password. While protecting the sheet, user is encouraged to select all the tips in the window that will pop-up to decide which actions are still available in the sheet.

It is crucial that user does not write comments outside the prescribed cells, otherwise errors may arise due to the way in which python gets the maximum and minimum columns and rows values of the sheet. For the same reason, if new rows (or columns) are edited and than removed, it is not sufficient to clean cells content (canc): those cells must be deleted.

## 3.2 File conductor_grid.xslx

This file completes the conductors initialization providing data to perform the spatial discretization. Its structure is analogous to the one of sheet **CONDUCTOR** of file conductor_definition.xlsx, as can be seen from figure 3.

### 3.2.1 Main features

Workbook is made by a single sheet called **GRID**. Red filled not editable cells came from both sheet **CONDUCTOR** of file conductor_definition.xlsx (cells A1 and cells from E on) and sheet **TRANSIENT** of file Transitory_Input.xlsx

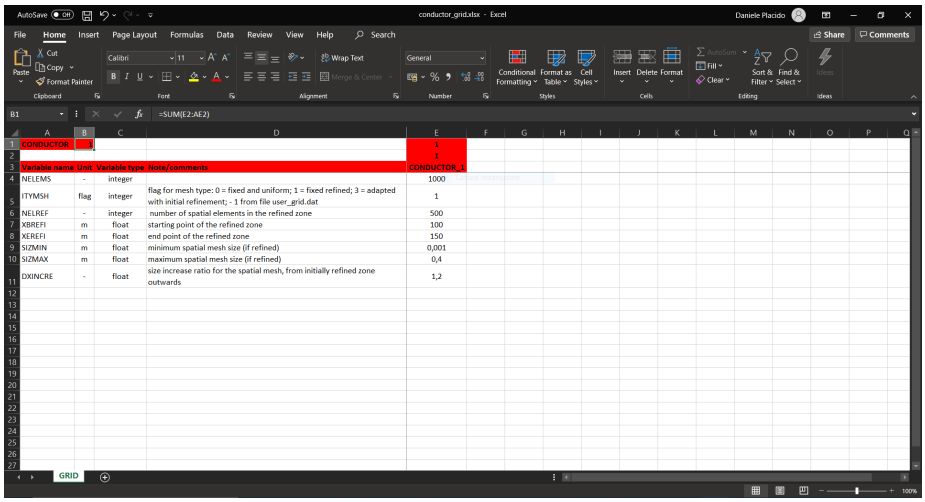



Figure 3: conductor_grid.xslx input file formatting example. There is only one sheet called **GRID**. Content of some red filled cells (the first four columns of the third row) came form input file Transitory_Input.xlsx, while others from sheet **CONDUCTOR** of file conductor_definition.xslx, in order to speed up file compilation. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required.

(cells A3, B3, C3 and D3). Headings meaning is explained in section 2.1. Cell B1 is evaluated with the same procedure described in section 3.1.3.

There are only two types of variables, float and integer, both used to set grid features. Different conductors may have different values of those input parameters, so that user can construct different spatial discretization and optimize computational cost.

### 3.2.2 Where and how

This file is converted into a dictionary by the class Conductors method __init__ exploiting the already mentioned function Read_input_file; dictionary name is dict_discretization and it is a class Conductors attribute, so that each conductor has its dict_discretization dictionary with all the information about the spatial discretization. Here is the code that calls function Read_input_file:

```
...
# CREATE grid for the i-th conductor
self.dict_discretization = {}
self.dict_discretization["Grid_input"] = Read_input_file(gridCond['GRID'],
self.ICOND, init_file = f"conductor_{self.ICOND}_grid.xlsx")
...
```

Dictionary construction process is the same described in sections 2.2 and 3.1.2.

### 3.2.3 Compile input file

As said in section 2.3, red filled cells can not be directly edited by the user; moreover the first four column headings came from sheet **TRANSIENT** of file Transitory_Input.xlsx, so if they should be changed user should act on this sheet.

User can add or delete rows except for the third one. User can also add new columns if more than one conductor is needed, adopting the same procedure described in section 3.1.3. Unfortunately, there is no obvious way to automatically add a new column in a file which is linked to another one, when in this second file a new column is edited; in other words the compilation of this file is not fully automated. Column headings can be build simply dragging the content of already existing cells. It is up to user fill in red the new heading cells and lock them protecting the sheets. It is not mandatory to add a password. While protecting the sheet, user is encouraged to select all the tips in the window that will pop-up to decide which actions are still available in the sheet.

Code raises an error message if the number of conductors defined in file conductor_definition.xslx is different from the number of conductors defined in file conductor_1_grid.xslx.

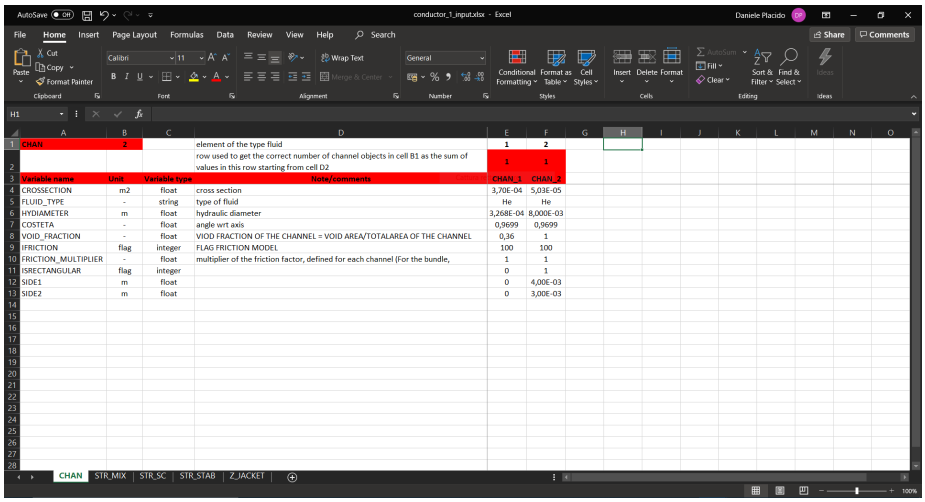
It is crucial that user does not write comments outside the prescribed cells, otherwise errors may arise due to the way in which python gets the maximum and minimum columns and rows values of the sheet. For the same reason, if new rows (or columns) are edited and than removed, it is not sufficient to clean cells content (canc): those cells must be deleted.

## 3.3 Files conductor_i_input.xlsx and conductor_i_operation.xlsx

This files are needed to initialize fluid and solid components objects that constitute conductors. For each conductor, a pair of conductor_i_input.xlsx and conductor_i_operation.xlsx file is required. File format is shown in figures 4 and 5.

### 3.3.1 Main features

This files have some common features with files conductor_definition.xlsx and conductor_1_grid.xlsx described in previous sections 3.1.1 and 3.2.1. Both workbooks consists of five sheets, one for each basic conductor components which are channels, mixed strand, superconductor strands, stabilizer strands and jackets. Sheets should be sorted alphabetically and since jacket must be the last sheet, sheets names are **CHAN**, **STR_MIX**, **STR_SC**, **STR_STAB** and, **Z_JACKET**.



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | CHAN | 2 | | element of the type fluid | 1 | 2 |
| 2 | | | | row used to get the correct number of channel objects in cell B1 as the sum of values in this row starting from cell D2 | 1 | 1 |
| 3 | Variable name | Unit | Variable type | Note/comments | CHAN_1 | CHAN_2 |
| 4 | CROSSECTION | m2 | float | cross section | 3,70E-04 | 5,03E-05 |
| 5 | FLUID_TYPE | - | string | type of fluid | He | He |
| 6 | HYDIAMETER | m | float | hydraulic diameter | 3,268E-04 | 8,000E-03 |
| 7 | COSTETA | - | float | angle wrt axis | 0,9699 | 0,9699 |
| 8 | VOID_FRACTION | - | float | VIOD FRACTION OF THE CHANNEL = VOID AREA/TOTALAREA OF THE CHANNEL | 0,36 | 1 |
| 9 | IFRICTION | flag | integer | FLAG FRICTION MODEL | 100 | 100 |
| 10 | FRICTION_MULTIPLIER | - | float | multiplier of the friction factor, defined for each channel (For the bundle, | 1 | 1 |
| 11 | ISRECTANGULAR | flag | integer | | 0 | 1 |
| 12 | SIDE1 | m | float | | 0 | 4,00E-03 |
| 13 | SIDE2 | m | float | | 0 | 3,00E-03 |

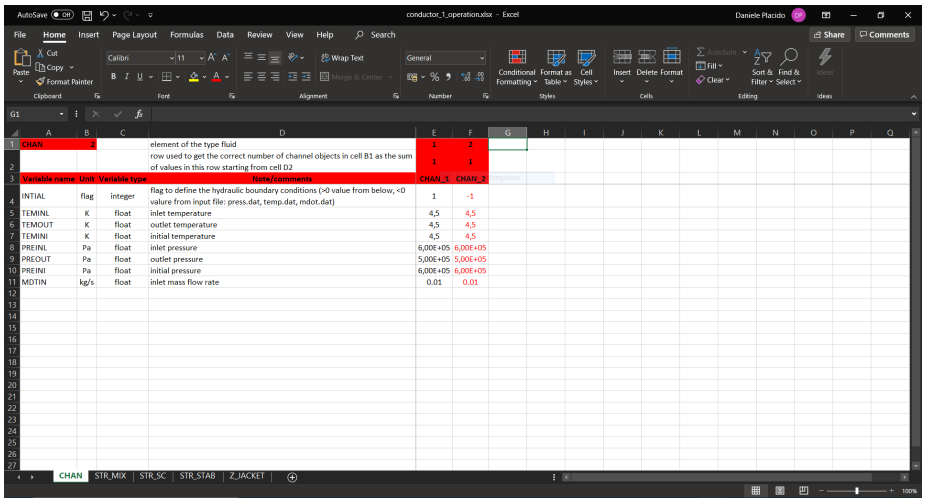
Figure 4: conductor_i_input.xlsx input file formatting example. There are five sheets that share the same format, so the focus is on the first sheet called **CHAN**; each sheet initializes a specific kind of objects. Please notice that sheets are sorted alphabetically and since jacket must be the last component, the sheet that initializes them is called **Z_JACKET**. Not all the sheets may be used by the code to initialize the i-th conductor. Content of some red filled cells (the first four columns of the third row) came form input file Transitory_Input.xlsx, in order to speed up file compilation. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required.

Since they share the same format, let focus on sheet **CHAN**. As for file conductor_definition.xlsx cell A1 shows the object name and cell B1 the total number of objects to be defined, evaluated with the strategy described in section 3.1.3. First four columns of the third row came from sheet **TRANSIENT** of file Transitory_Input.xlsx; the other headings of the third row are constructed exploiting the excel function textjoin as explained in section 3.1.1. As usual, this red filled cells can not be directly edited by the user because they are locked.

The same holds for sheets of file conductor_i_operation.xlsx, with the only difference that cell A1 of this sheets is linked with the cell A1 of sheets in file conductor_i_input.xlsx. This allows user to un-protect just this last file, modify cells A1 if needed and automatically changes propagates also to sheets of file conductor_i_operation.xlsx.

Variables types can be string to define fluid types or materials types in general, integer to set flags and float write input data.

Each sheet may have more than one object of the same kind with different values, as shown in figures 4 and 5 where two different channels are defined in sheet **CHAN**.



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | CHAN | 2 | | element of the type fluid | 1 | 2 |
| 2 | | | | row used to get the correct number of channel objects in cell B1 as the sum of values in this row starting from cell D2 | 1 | 1 |
| 3 | Variable name | Unit | Variable type | Note/comments | CHAN_1 | CHAN_2 |
| 4 | INTIAL | flag | integer | flag to define the hydraulic boundary conditions (>0 value from below, <0 value from input file: press.dat, temp.dat, mdot.dat) | 1 | -1 |
| 5 | TEMINL | K | float | inlet temperature | 4,5 | 4,5 |
| 6 | TEMOUT | K | float | outlet temperature | 4,5 | 4,5 |
| 7 | TEMINI | K | float | initial temperature | 4,5 | 4,5 |
| 8 | PREINL | Pa | float | inlet pressure | 6,00E+05 | 6,00E+05 |
| 9 | PREOUT | Pa | float | outlet pressure | 5,00E+05 | 5,00E+05 |
| 10 | PREINI | Pa | float | initial pressure | 6,00E+05 | 6,00E+05 |
| 11 | MDTIN | kg/s | float | inlet mass flow rate | 0.01 | 0.01 |

Figure 5: conductor_i_operation.xlsx input file formatting example. There are five sheets that share the same format, so the focus is on the first sheet called **CHAN**; each sheet initializes a specific kind of objects. Please notice that sheets are sorted alphabetically and since jacket must be the last component, the sheet that initializes them is called **Z_JACKET**. Not all the sheets may be used by the code to initialize the i-th conductor. Content of some red filled cells (the first four columns of the third row) came form input file Transitory_Input.xlsx, others came from the corresponding sheets of input file conductor_i_input.xlsx, in order to speed up file compilation. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required.

Not all the sheets should be compiled by the user as it will be better explained in section 3.3.3.

### 3.3.2 Where and how

This pair of input files is converted into two separate dictionaries by function Read_input_file, called respectively dict_input and dict_operation, which are both attributes of FluidComponents and SolidComponents classes. These dictionaries are created by the __init__ method of classes FluidComponents, MixSC-Stabilizer, Stabilizer, SuperConductors and Jacket; the last four classes inherits the attributes from the father class SuperConductors.

The lines of code are the same for all the conductor components and are shown below.

```
...
def __init__(self, sheet, sheetOpar, icomp, name, icond):
self.NAME = name
```

```
self.ID = f"{self.NAME}_{icomp}"

# dict_input dictionary initialization (cdp, 06/2020)
self.dict_input = Read_input_file(sheet, icond, icomp,
init_file = f"conductor_{icond}_input.xlsx")
# dict_operation dictionary initialization (cdp, 06/2020)
self.dict_operation = Read_input_file(sheetOpar, icond, icomp,
init_file = f"conductor_{icond}_operation.xlsx")
...
```

The dictionaries are loaded only once during conductors initialization and they are widely used in the code. Dictionaries construction process is analogous to that described in section 3.1.2, with the only difference that the values came from columns *CHAN_I*, *STR_MIX_I* and similar.

### 3.3.3 Compile input file

Even though this file are somehow coupled, user is invited to compile first input file conductor_i_input.xlsx, since it is the "source file" for input file conductor_i_operation.xlsx.

As said in section 2.3, red filled cells can not be directly edited by the user; moreover the first four column headings came from sheet **TRANSIENT** of file Transitory_Input.xlsx, so if they should be changed user should act on this sheet.

User can add or delete rows except for the third one. User can also add new columns if other conductors components are needed, adopting the same procedure described in section 3.1.3. It is important to add conductors components in the correct sheet: to avoid this error, user is strongly recommended to use the above recalled strategy.

Unfortunately, there is no obvious way to automatically add a new column in a file which is linked to another one, when in this second file a new column is edited; in other words the compilation of this files is not fully automated. Therefore user must remember to add the same columns also in the corresponding sheets of file conductor_i_operation.xlsx, after that file conductor_i_input.xlsx was edited. Column headings can be achieved simply dragging the content of already existing cells. It is up to user fill in red the new heading cells and lock them protecting the sheets. It is not mandatory to add a password. While protecting the sheet, user is encouraged to select all the tips in the window that will pop-up to decide which actions are still available in the sheet.

Code raises an error message if the number of components defined in each sheet of file conductor_i_input.xlsx is different from the number of components defined in each sheet of file conductor_i_operation.xlsx.

It is crucial that user does not write comments outside the prescribed cells, otherwise errors may arise due to the way in which python gets the maximum and minimum columns and rows values of the sheet. For the same reason, if new rows (or columns) are edited and than removed, it is not sufficient to clean cells content (canc): those cells must be deleted.

# 4 Input file for interfaces: conductor_i_coupling.xlsx

This file stores information about the interface between fluid and solid components which constitute the conductor; to each conductor corresponds a file with this name. File format is shown in figures 6, 7 and 8.

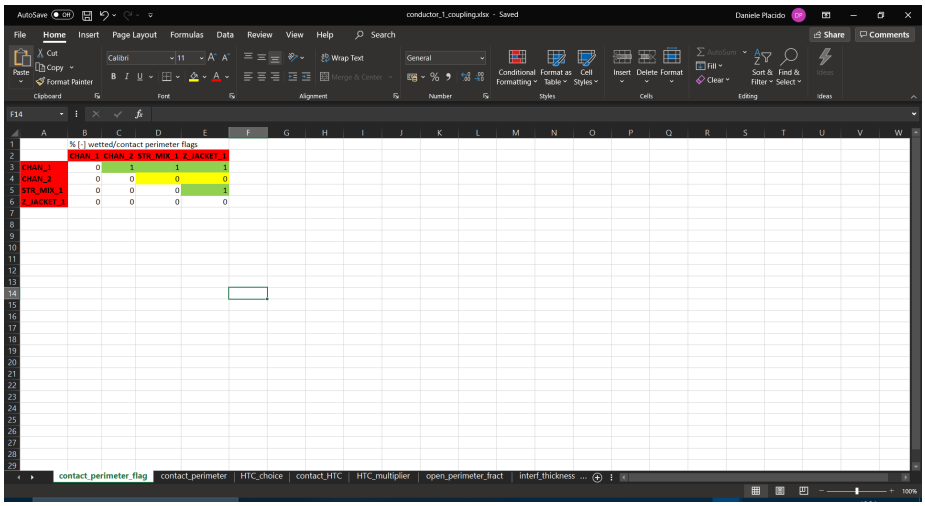



Figure 6: conductor_i_coupling.xlsx input file formatting example. There are eight sheets, here the first sheet called **contact_perimeter_flag** is shown, which contains the values of the flags to establish whether or not there is contact between the components of the conductor. Content of red filled cells in the second rows came form the sheets of input file conductor_i_input.xlsx; red filled cells in the first column are obtained equating the values of the cells in second row, in order to speed up the compilation of the file. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required. User should compile only the upper triangular matrix, where a conditional formatting is introduced: if flag value is 1 the cell is filled in green, if it is 0 the cell is filled in yellow. All the other cells should have 0 value.

## 4.1 Main features

This group of file has the largest number of sheets, as a matter of fact workbook is made by eight sheets.

The first row contains a comment with an explanation of the sheet content and the unit of measurement; second row and first column of the sheet shows the square and symmetric matrix columns and rows headings respectively, which is the way in which information is provided in this file. Each row and each column

corresponds to the i-th conductor components, headings help to easily visualize and read matrix content. They are made by the components IDs: in sheet **contact_perimeter_flag** second row is constructed exploiting column headings from sheets of file conductor_i_input.xlsx, while the first column is built from the second row. For all the other sheets, these headings came form a link with sheet **contact_perimeter_flag**.

In figure 6 matrix is four times four since conductor is composed of two channels, one strand of kind mixed and one jacket. It should be noted that only actually used components appear in the headings.

Even though matrix has no empty cells, only the yellow filled ones are actually used by the code, corresponding to the upper triangular matrix thanks to the symmetry.

Excel conditional formatting is introduced in all sheets to help user visualize the cells that he should edit and how, as explained in secion 4.3.

All these sheets combined together allows to compute the heat transfer coefficients between fluid components, between fluid and solid components and between solid components.

Let detail each sheet in separate sections.

#### 4.1.1 Sheet contact_perimeter_flag

Shows flags for wet (or contact) perimeter between conductor components. If flag is equal to one there is an interface between components, otherwise flag is equal to zero. The main diagonal elements are always zero since the components can not be in contact with themselves. In the code those values are forced to be integers.

#### 4.1.2 Sheet contact_perimeter

Shows the contact perimeter value between components that are in contact. It corresponds to an area per unit length. The only non zero cells are the ones corresponding to a non zero flag in sheet **contact_perimeter_flag**. Also in this sheet main diagonal elements are always zero. Code treats this values as float.

#### 4.1.3 Sheet HTC_choice

It is the most difficult sheet to interpret and it is shown in figure 7 for the sake of clarity. It shows flags to define the model to compute the heat transfer coefficients. First of all, user should notice that, as opposed to the previous two sheets, in this case also cells belonging to the main diagonal are yellow filled. Moreover this is the only sheet with this feature.

Let focus on main diagonal. The red integer values on the first $n_{ch}$ elements of the main diagonal, if $n_{ch}$ is the total number of channels in the i-th conductor, are used by the code to select the correlation to evaluate the steady heat transfer coefficient for channel objects. At the time being possible red flag values are:

- 1 fluid channels, Dittus-Boelter correlation;

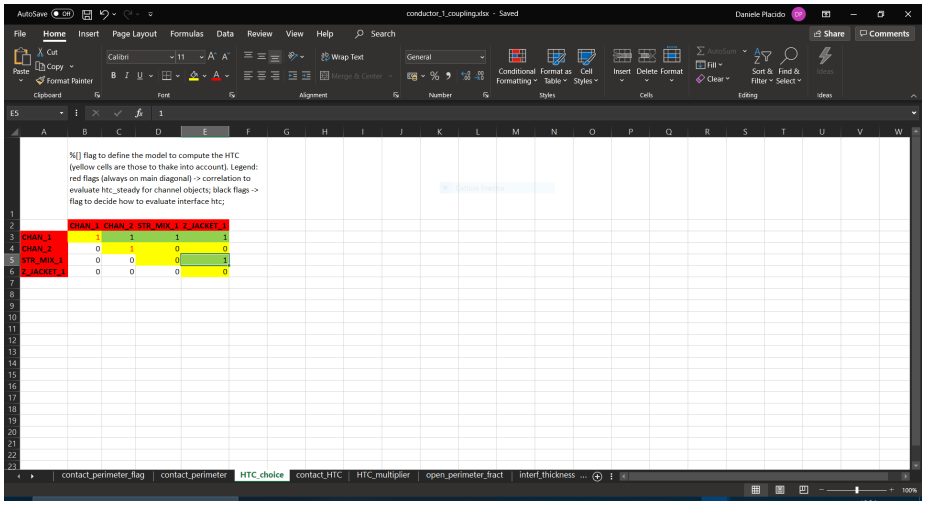



Figure 7: conductor_i_coupling.xlsx input file formatting example. There are eight sheets, this figure shows sheet called **HTC_choice**, which contains the flags to establish how to evaluate the heat transfer coefficient. Content of red filled cells in the second rows came form sheet **contact_perimeter_flag**, used as "source" sheet; red filled cells in the first column are obtained equating the values of the cells in second row, in order to speed up the compilation of the file. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required. This is the only sheet with non zero values on the main diagonal. This flags, highlighted in red, are used to correctly evaluate the steady heat transfer coefficient for channels, regardless of whether or not they are in contact, according to tokamak kind and geometry. Together with main diagonal, user should compile only the upper triangular matrix, where a conditional formatting is introduced based on the flag values of sheet **contact_perimeter_flag**: if flag value is 1 the corresponding cell in sheet **HTC_choice** is filled in green, if it is 0 the corresponding cell in sheet **HTC_choice** is filled in yellow. This is to help the user locate the cells he need to fill in with non-zero values. All the other cells should have 0 value.

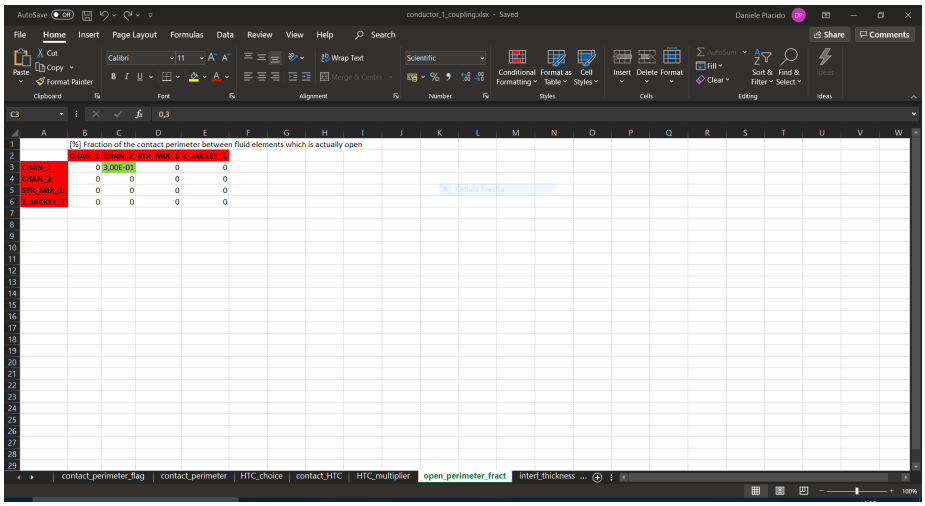



Figure 8: conductor_i_coupling.xlsx input file formatting example. There are eight sheets, this figure shows sheet called **open_perimeter_fract**, which contains the fraction of the contact perimeter between channels that should be considered open. Content of red filled cells in the second rows came form sheet **contact_perimeter_flag**, used as "source" sheet; red filled cells in the first column are obtained equating the values of the cells in second row, in order to speed up the compilation of the file. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required. User should compile only the top left part of the matrix, the square matrix describing channels contact. A conditional formatting is introduced based on the flag values of sheet **contact_perimeter_flag**: if flag value is 1 the corresponding cell in sheet **open_perimeter_fract** is filled in green, if it is 0 the corresponding cell in sheet **open_perimeter_fract** is filled in yellow. This is to help the user locate the cells he need to fill in with non-zero values. All the other cells should have 0 value.

- 119 fluid channels, rectangular ducts from DEMO common memo;
- 120 fluid channels, triangular ducts from DEMO common memo;
- 121 fluid channels, any other duct shape from DEMO common memo;
- 211 fluid channels, HTS CL.

The others cells on the main diagonal corresponding to SolidComponents objects have zero values, since for them the steady heat transfer coefficient has no physical meaning.

Black flags outside the main diagonal state how to evaluate the interface heat transfer coefficient as follows:

- if flag equals -1, the value of the heat transfer coefficient comes form the upper triangular matrix in the sheet **contact_HTC** (see section 4.1.4);
- if flag value is larger than zero, heat transfer coefficient is evaluated with suitable correlations by the code.

It should be noted that, regardless of the black flag values, the heat transfer coefficients between solid components is constant and set to $500\,\mathrm{W\,m^{-2}\,K^{-1}}$, due to lack of experimental or numerical information.

#### 4.1.4 Sheet contact_HTC

The upper triangular matrix has the values of the interface heat transfer coefficients if outside main diagonal flag of sheet **HTC_choice** is equal to $-1$. In this case the heat transfer coefficient is assumed constant along the conductor length.

#### 4.1.5 Sheet HTC_multiplier

This sheets holds the values for the multiplier for the heat transfer coefficients, used when needed by the code to perform the calculation. Non zero values are only in correspondence of non zeros flag in sheet **contact_perimeter_flag** and they are converted to float by the code.

#### 4.1.6 Sheet open_perimeter_fract

Here is shown the fraction of the contact perimeter between channels which is actually open, used to evaluate the open heat transfer coefficient; therefore the only non zeros are in the upper left matrix which describes interfaces between channels. If there is contact between channels, concatc_perimeter_flag will be equal one and in the corresponding cell of this sheet a value lower than one will be found. In the example proposed in figure 8 there is only one value larger than zero since there are only two channels and therefore only one possible interface. This value must be larger than zero but smaller than one and it is treated like a float by the code.

#### 4.1.7 Sheet interf_thickness

This sheet is similar to the previous one with the difference that here is shown the thicness of the interface between fluid components, used to evaluate the thermal resistence. Considerations analogous to that made in section 4.1.6 are valid also for this sheet.

#### 4.1.8 Sheet trans_transp_multiplier

To complete the evaluation of the open heat transfer coefficient, a multiplier is needed which is found in this last sheet. The same considerations made in section 4.1.6 can be still applied also to this sheet.

## 4.2 Where and how

This sheets are loaded as matrix during the conductor instance by the constructor of class Conductors, method __init__. Those matrices have the same name of workbook sheets and are Conductors attributes, as shown by the following lines of code.

```
...
# Filling matrix for interfaces between conductor object components
# with values from conductor_i_coupling.xlsx sheets (cdp 06/2020)
for row in range(self.numComponents):
        for col in range(self.numComponents):
                self.contact_perimeter_flags[row, col] =
                sheet_CP_flags.cell(row = 3 + row, column = 2 + col).value
                self.contact_perimeter[row, col] = sheet_CP.cell(row = 3
                + row,  column = 2 + col).value
                self.HTC_choice[row, col] = sheet_HTC_choice.cell(row =
                3 + row, column = 2 + col).value
                self.Contact_HTC[row, col] = sheet_HTC.cell(row = 3 +
                row, column = 2 + col).value
                self.HTC_multipl[row, col] = sheet_HTC_multipl.cell(row
                = 3 + row, column = 2 + col).value
                self.open_perimeter_fract[row, col] =
                sheet_open_perimeter_fract.cell(row = 3 + row,
                column = 2 + col).value
                self.interf_thickness[row, col] =
                sheet_interf_thickness.cell(row = 3 + row,
                column = 2 + col).value
                self.trans_transp_multiplier[row, col] =
                sheet_trans_transp_multiplier.cell(row = 3 + row,
                column = 2 + col).value
# End reading and loading conductor_i_coupling.xlsx sheets content (cdp 06/2020)
...
```

They are used by Conductors method Eval_transp_coeff to evaluate the heat transfer coefficients and in the construction of the coefficient matrix SMAT by the function STEP that solves each step of the transient.

## 4.3 Compile input file

The user must be careful in filling out these sheets. He should remember that only for the second row of the first sheet (**contact_perimeter_flag**) there is the link whit input file conductor_i_input.xlsx, to correctly write column headings; the first column of this sheet should be filled equating the cells to the values of the second row; in this way typing errors are minimized. The second row of the other sheets can be obtained in two different ways: the first is by repeating the same procedure done for the first sheet but it is preferable that user, once

compiled sheet **contact_perimeter_flag**, links the second row of the other sheets with the second row of this sheet and drags the cells to get the columns heading. The first column will be obtained equating the cells values to the one of the second row as said above. This procedure must be done for each sheet. Once the headers are obtained, the user must fill the cells with red and proceed to protect the sheet. While protecting the sheet, user is encouraged to select all the tips in the window that will pop-up to decide which actions are still available in the sheet.

After that, user should write the matrix elements, to help him in this delicate procedure, excel conditional formatting is exploited as follows. At the beginning all the cells of the upper triangular matrix are filled in yellow, so that user can easily locate where to eventually change the values, by default set to zero everywhere. User must start with the first sheet and set flags to one where interfaces should raise, cells filling colour will shift to green to state that contacts are introduced. Automatically in all the others sheets, cells that corresponds to the ones with the value of contact_perimeter_flag set to one, change their fill color from the default yellow to green. Only the green cells of those sheets must be edited by the user with suitable non zero values. It is up to the user to expand this conditional formatting to all the new needed cells and sheets, if any.

The main diagonal, with the exception of the sheet **HTC_choice**, and the lower triangular block always have zeros.

User should choose properly the flag value on the main diagonal in sheet **HTC_choice** according to channel geometry and the kind of tokamak considered in order to correctly evaluate channels steady state heat transfer coefficients.

# 5 Input file with tables to perform interpolation

This kind of input files are needed to compute parameters from the interpolation of the values in the provided tables, and they are in common with all the conductors. These files can be grouped according to two criteria: by objects to which they apply or by the way in which interpolation is performed. As far as this last criteria is concerned, file flow_dummy.xlsx allows interpolation only in time; all the others files may require a double interpolation in time and space to get the parameters value, the structure is based on file bfield.xlsx. According to the first grouping criteria, it is possible to distinguish three subgroups: file related to fluid components objects (flow_dummy.xlsx), file related to strand objects (strain_dummy.xlsx) and file related to both strand and jacket objects (i.e. solid components objects) (alphab_dummy.xlsx, bfield.xlsx, I_file.xlsx, Q_file_dummy.xlsx). Despite the differences in the type of objects managed by the file, they share almost always the same structure as it will be better explained later on. An example of this file is shown in figures 9 and 10.

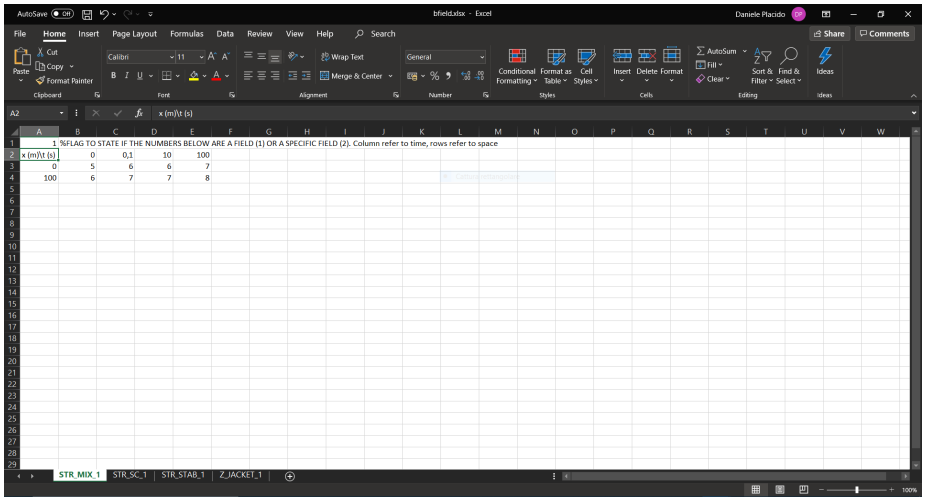



Figure 9: bfield.xlsx input file formatting example. The number of sheets depends on the number of solid components that constitute the conductor, however they share the same format so figure shows the first sheet called **STR_MIX_1**. This sheet contains values of the magnetic field for strand of type mixed; if the value in cell A1 is 1 units are T, else if the value in cell A1 is 2 units are $\mathrm{T\,A^{-1}}$. Second rows has the times on which perform interpolation in time, first column has the spatial coordinates on which perform interpolation in space. If time is lower than minimum time in the second row or is large than the maximum time in the same row, it is assumed that magnetic field is constant and it is eventually evaluated with a simple interpolation in space; an analogous rationale holds for the spatial coordinates.

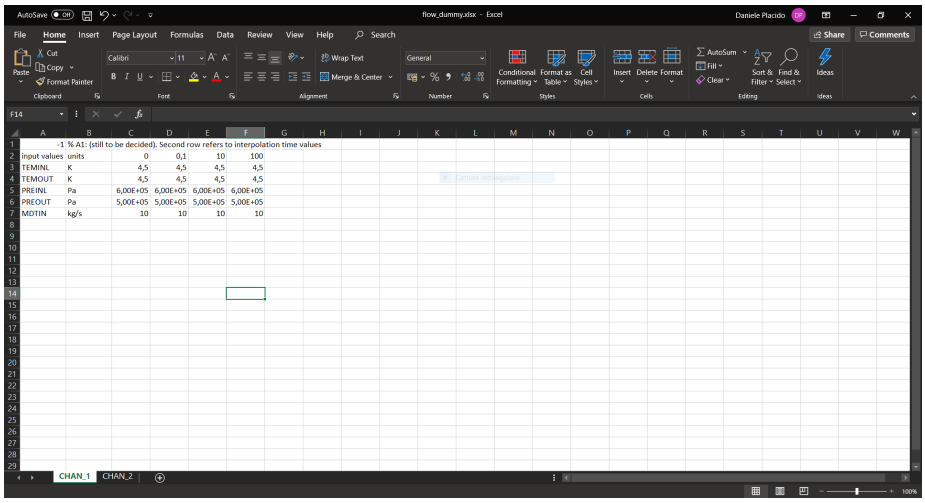



Figure 10: flow_dummy.xlsx input file formatting example. The number of sheets depends on the number of channels that constitute the conductor, however they share the same format so figure shows the first sheet called **CHAN_1**. This sheet contains values of channel inlet and outlet temperature and pressure, together with inlet mass flow rate to set flow initial conditions. Second rows has the times on which perform interpolation in time, first column has the name of the variables. Cell A1 is left free to set a flag, but it is not defined yet. If time is lower than minimum time in the second row or is large than the maximum time in the same row, it is assumed that the property is constant and equal to the value of the property at the first or last available time.

## 5.1 Main features

Workbook has as many sheets as the number of objects that should be managed by the file; for instance in the above figure there are four sheets, one for each kind of solid components objects. This means that the conductors may be composed of an arbitrary combination of the three strands type and a single jacket, the maximum number of solid components constituting a conductor should be lower or equal than four. If a different kind of mixed strand is used a new sheet called **STR_MIX_2** should be added to the file.

Let focus on two sheets from different workbooks, like **STR_MIX_1** from file bfiled.xlsx and **CHAN_1** from file flow_dummy.xlsx. First cell is in general used to set some flags value: if flag is equal to one, tables values are T; if flag is equal to two tables values are given per unit of current, or in other words $\mathrm{T\,A^{-1}}$. The same applies for file alphab_dummy.xlsx. In other files like flow_dummy.xlsx or strain_dummy.xlsx the flag meaning is not yet decided at the time being. In cell B1, a comment that explains the flag meaning an how to read the table is always present. Cell (A2) in sheets like **STR_MIX_1** is used to clarify that in the second row there are the times while in the column A there are spatial

coordinates at which values are available; to get the needed value a double interpolation should be done.

The table of sheet **CHAN_1** looks different from the one described above. First of all, it should be noted that there are only the time values because in this case only the time interpolation is required. On column A starting from the third row there are parameters name and column B shows their SI units. Starting from column C parameters values at given time steps are exploited to perform interpolation in time.

## 5.2 Where and how

This files are read and loaded if a certain flag has a negative value, which means that the value should be taken from the file by an interpolation. This procedure is carried out by a set of three functions belonging to file Utility-Functions/Auxiliary_functions.py. Function Get_from_xlsx calls the other two functions in different ways according to the kind of interpolation needed; the table is decomposed into a matrix and two vectors if the double interpolation is needed, or a matrix and the time vector only if the interpolation in time is required, by function Read_interp_file. Finally the interpolation is performed by the function Interpolation. Call of function Get_from_xlsx to load file bfield.xlsx is reported below.

```
def Get_B_field(self, conductor, time, **options):
        if options.get("Where") == "nodal":
                # compute B_field in each node (cdp, 07/2020)
                if self.dict_operation["IBIFUN"] < 0:
                    # cza to enable other negative (read from file)
                    #flags -> ibifun.eq.-3, see below (August 29, 2018)
                    path = conductor.basepath +
                    conductor.file_input["EXTERNAL_BFIELD"]
                    # call Get_from_xlsx on the component
                    [self.dict_node_pt["B_field"], flagSpecfield] =
                    Get_from_xlsx(conductor, path, self, time)
                    if flagSpecfield == 2: # BFIELD is per unit of current
                        self.dict_node_pt["B_field"] =
                        self.dict_node_pt["B_field"]*conductor.IOP_TOT
```

File flow_dummy.xlsx is read if flag INTIAL is negative in the function Gen_flow; files bfield.xlsx, I_file_dummy.xlsx, Q_file_dummy.xlsx are read in class SolidComponents by functions Get_B_field, Get_I and Get_Q if flags IOPFUN, IBFUN and IQFUN are negative respectively; file alphab_dummy.xlsx is read in class Strands by function Get_alphaB if flag IALPHAB is negative; finally file strain_dummy.xlsx is read by function Get_EPS if flag IEPS is negative.

### 5.3 Compile input file

Flag value should be an integer, so to avoid user error the code force it to an integer. At the time being it can be a positive or negative value since for many files the flag meaning is not yet decided and it does not have any effect on properties evaluations. In files bfiedl.xslx and alphab_dummy.xlsx flag meaning is decided so a guide line can be given: flag value must be 1 or 2 according that values are pure or per unit of current as pointed out in section 5.1.

Tables values are converted by the code in float values. If time and space interpolation is needed, user should be aware of how to interpret the table. The spatial coordinates may not cover the whole conductor length, in this case the minimum and maximum spatial coordinate are the starting and ending points on which the interpolation is performed. This means that before the minimum spatial coordinate and after the maximum one the property value is constant and equal to the value obtained by the time interpolation at the minimum and at the maximum spatial coordinates in the tables. A similar reasoning holds for the time, common also to file for which only the time interpolation is needed like flow_dummy.xlsx: the minimum and maximum time in the table may not correspond to the starting and ending time of the simulation. In this case, if time is lower than the minimum or larger than the maximum, property value is evaluated by a space interpolation at the minimum or at the maximum time (if the double interpolation is needed), or set equal to the value corresponding to the minimum or maximum time value if only the time interpolation is required.

## 6 Output settings: conductors_diagnostic.xlsx

This file should be compiled by the user if he wants to store some data that can be used to make plots. Its structure is shown in figure 11 and described it detail in next section 6.1. Files where data are saved have extension .txt.

### 6.1 Main features

This file is at conductors level, which means that user can set when (time step) and where (spatial coordinate) save information provided by the code for each considered conductor, analogously to what is done in file conductor_definition.xslx or conductor_grid.xlsx (see sections 3.1.1 and 3.2.1).

The workbook is composed of two sheets, sheet **Space** is used to store the status of the simulation at given time, while sheet **Time** used to save the time evolution of the solution, fluid density in channels, channels inlet and outlet mass flow rate, magnetic field, strands current sharing temperature at user defined spatial coordinates. Sheets organization is the same so it is possible to focus only on sheet **Space**, since it is the one shown in figure 11; moreover the red filled cells of sheet **Time** are linked to the ones of sheet **Space**.

Cell A1 shows the name of the object referred to, conductor, while cell B1 shows the total number of conductors considered in the code. This number is evaluated in the same way explained in section 3.3.3. Indeed, the red filled cells

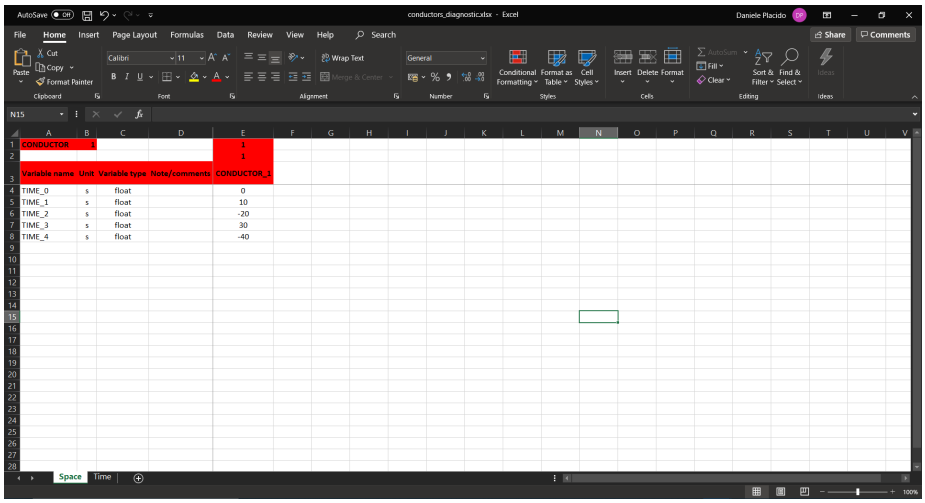



Figure 11: conductor_definition.xslx input file formatting example. There are three sheets but here the focus is on sheet **CONDUCTOR**. Content of some red filled cells (the first four columns of the third row) came form input file Transitory_Input.xlsx, in order to speed up file compilation. To minimize typos, red filled cells can not be directly edited by the user, unless he removes the sheet protection. In this case, a password may be required.

came from sheet **CONDUCTOR** of file conductor_definition.xlsx, so those files are linked and the red cells can not be directly modified by the user as described in section 2.1, please refer to this section to recall the meaning of cells E1, E2, F1, F2 and so on.

First four columns of the third row are came from sheet **TRANSIENT** of file Transitory_Input.xlsx and their meaning is already explained in 2.1; starting from the fifth column, the column heading is the conductor ID, constructed exploiting cell A1 and the values of the first row starting from fifth column as already pointed out in section 3.1.1.

## 6.2 Where and how

Despite the very similar structures, those sheets are loaded by the Conductors class method __init__ in two different ways to keep into account that solution values are stored in different ways: sheet **Space** is loaded as a dictionary exploiting the already mentioned function Read_input_file, while sheet **Time** is loaded as a vector, this is shown by the following lines of code.

```
...
# Dictionary to save solution spatial discretization initialization:
# (cdp, 08/2020)
sheet_Space = output["Space"]
```

```
self.dict_Space_save = Read_input_file(sheet_Space, self.ICOND,
init_file = "conductors_diagnostic.xlsx")
# Array to save solution and parameters in time to exploit array smart
# notation (cdp, 08/2020)
sheet_Time = output["Time"]
r_first = sheet_Time.min_row + 3 # first row to be read (cdp, 08/2020)
r_last = sheet_Time.max_row # last row to be read (cdp, 08/2020)
c_current = 3 + self.ICOND # column to be read
self.Time_save = np.zeros(r_last - r_first)
for rr in sheet_Time.iter_cols(min_row = r_first, max_row = r_last,
min_col = c_current, max_col = c_current, values_only = True):
    self.Time_save = np.array(rr, dtype=float)
```

In the first case, dictionary keys are read from column *Variable name* while dictionary values came from *CONDUCTOR_I* and are converted to the python variable type according to column *Variable type*; if a not valid variable type is given an error is raised.

Spatial shape of the solution is saved by default each 100 iteration or when time is equal to the time indicated by the user, invoking function Save_simulation_space in UtilityFunction/Auxiliary_functions.py. Files keep saved the last three status of the transient, overwriting the oldest one. At each time step, code calls function Save_simulation_time in UtilityFunction/Auxiliary_functions.py to save solutions, fluid density in channels, channels inlet and outlet mass flow rate, magnetic field and strands current sharing temperature in chosen spatial coordinates.

## 6.3 Compile input file

As already said in section 2.3, red filled cells can not be directly edited by the user; moreover the first four column headings came from sheet **TRANSIENT** of file Transitory_Input.xlsx, so if they should be changed user should act on this sheet.

User can add or delete rows except for the third one. User can also add new columns adopting the same procedure described in section 3.1.3 if there are more than one conductor. Unfortunately, there is no obvious way to automatically add a new column in a file which is linked to another one, when in this second file a new column is edited; in other words the compilation of this file is not fully automated. Column headings can be build simply dragging the content of already existing cells. It is up to user fill in red the new heading cells and lock them protecting the sheets. It is not mandatory to add a password. While protecting the sheet, user is encouraged to select all the tips in the window that will pop-up to decide which actions are still available in the sheet.

It is crucial that user does not write comments outside the prescribed cells, otherwise errors may arise due to the way in which python gets the maximum and minimum columns and rows values of the sheet. For the same reason, if new rows (or columns) are edited and than removed, it is not sufficient to clean

cells content (canc): those cells must be deleted.

Each conductor can have an arbitrary number of time steps and spatial coordinates at which the solution and the other quantities can be stored. Since there is a single file, if for the i-th conductor the user is interested to store the solution at $n$ time step values and for the j-th conductor at $m$ time step values with $m \neq n$, the extra time step in sheet **Space** should be negative since only positive values are considered by the code. The same rationale applies to sheet **Time**.

To better explain how the user should compile this file, let consider an example with two conductors called CONDUCTOR_1 and CONDUCTOR_2. Let say that for the first one user needs to see the spatial distribution of the solution at 3 different time step 10 s, 50 s and 100 s while for the second he needs to see the solution only at 10 s and 100 s. He should set three values for each conductor: CONDUCTOR_1 will have all positive values while CONDUCTOR_2 will have two positive and one negative values. Code does not matter about the location of the negative values, provided they are on the correct column, and there is no need to sort the values, but it is strongly recommended to keep a logical order in the input file.