pansys Documentation

Release 0.1a

Najeem Muhammed

Nov 28, 2018

User's Guide:

The pansys module helps you interact with ansys through python. Starting a panys session is as easy as setting a variable.

```
from pansys import Ansys
ans = Ansys()
```

Now you're ready to send commands to your newly created ansys session.

```
ans.send("/prep7")
ans.send("""n,,1
    n,,2""")
```

As you must have guessed, `ans.send` command will let you send commands from python to ansys in string format.

You can get data out of Ansys as well.

```
nmax = ans.get("node","","num","max")
ncount = ans.get("node","","count","")
```

Using `get_list()` function, you can get any ansys list item as well.

```
nodes = ans.get_list("nlist")
```

You can also start an ansys session in a remote machine. You will have to set up your ssh keys for this to work.

```
ans = Ansys(host="remotesystem")
```

Look into the documentation to get to know the API better.

CHAPTER 1

API

1.1 Ansys object

class pansys.**Ansys**(*startcommand=None*, *startfolder=None*, *cleanup=False*, *host=None*)

Ansys session class

Ansys class to create an interactive ansys session. You can interact with an ansys session with the help of this class.

Prerequisites:

- Ansys should be installed in your computer.
- Ansys should have a interactive commandline interface.

To initiate, create an instance of the class as shown below:

```
>>> ans = Ansys()
```

This will create an instance of ansys v.15 in a newly created folder. You can change the start command of ansys by setting the `startcommand` as shown below:

```
>>> ans = Ansys(startcommand="ansys130")
```

If you end up changing the `startcommand` everytime, you can as well set an environment variable, `PANSYS_STARTCOMMAND` to the command of your choice. If this environment variable is found, the value of this environment variable will be used for starting `Ansys` session. However, if you start the `Ansys` session with the `startcommand` specified, then the specified command will take precedence.

You can also change the folder where you want Ansys to start by setting the `startfolder` parameter.

Parameters

- **startcommand** (*str*) – Ansys start command. The linux command corresponding to the version of ansys you want to open. You can give the license type as well along with the ansys command.

- **startfolder** (*str*) – The folder in which you want to start ansys. The folder should be existing already. If left blank, a new folder of the format `YYYYMMDDHHSS` will be started in the location where python is running and ansys will be started inside that.
- **cleanup** (*bool*) – If true will delete the ansys working directory after the ansys has exited. Will not delete if an existing start folder was given.
- **host** (*str*) – The system in which you want to start the Ansys session. You can pass in the format `user@system` where user is the username you want to use to connect to the system and system is the network name of the system or the ip-address of the system to which you want to connect to. You can omit the user part if you want to connect to the remote machine with the current login itself. It is expected that you have set up ssh-keys in the remote system for this to work.

__del__()
Destructor function for ansys exiting

__repr__()
Representation of the object

get(*entity*, *entnum*, *item1*, *it1num=''*, *item2=''*, *it2num=''*)
Wrapper for ansys `*GET` command

Function to execute the `*get` command of ansys. Sequence of input data is same as in ansys.

Parameters

- **entity** (*str*) –
- **entnum** (*str*) –
- **item1** (*str*) –
- **it1num** (*str*) –
- **item2** (*str*) –
- **it2num** (*str*) –

Note: All arguments are in the same order as per ansys `*get` documentation.

Returns Output of `*get`. Can be int, float, exponential or string.

get_list(*command_string*, ***kwargs*)
Extract any list from ansys

Function to get any ansys list as a `pandas.DataFrame`.

Example

```
>>> a.get_list("nlist")
```

Note: The function assumes that the `command_string` will return a column data.

For example, instead of using `a.get_list("elist")` you should use `a.get_list("elist,,,,1")` For getting element list.

This function passess all keyword arguments directly to `pandas.read_table()` function. The default values of `delim_whitespace` is `True` and `skiprows` is 2.

Parameters

- **command_string** (*str*) – The Ansys command which will output a column data.
- ****kwargs** – All keyword arguments for the read_table function in pandas is applicable for this function as well.

Returns

A `pandas.DataFrame` with the data that ansys returned when `command_string` was passed.

Return type pandas.Dataframe

get_output(*command_string*, *persist=False*)

Function to get ansys output as a file

The function uses `/output` command in ansys to redirect the output to a file. The path to this file will be returned from the command.

Parameters

- **command_string** (*str*) – The command(s) to be executed for which the output is sought.
- **persist** (*bool*) – If True, will create a unique file which will not be overwritten because of subsequent call of this function. Default is False which will write the output to a file `out.out` in the ansys working directory.

Returns

Path to a file which contains the output of the ansys command call.

Return type str

get_queue()

Returns a generator with the commands in the current queue, submitted using the *pansys.Ansys.queue()* method.

Returns A file object pointing to the command list

Return type object

output

The output of the last executed Ansys command

plot(*command_string*)

Plot anything in ansys

Function to return an image with a plot from Anys.

Example

```
>>> ans.plot("eplot")
```

Parameters **command_string** (*str*) – The command for plotting in ansys

Returns The path to the image file.

Return type str

queue(*command_string*)
Queue commands for delayed execution

When there is a large number of ansys commands that you want to pass, use this function to queue up them for execution. To execute the queue, use the *pansys.Ansys.run_queue()* method.

Parameters **command_string** (*str*) – Required. The command that you want to add to the queue.

Returns None

run_queue(***kwargs*)
Runs all the commands in the queue

This method writes all the commands that are queued using the *pansys.Ansys.queue()* method to a file and execute them in one go by using the `/input` command of Ansys. This will be sent using the *pansys.Ansys.send()* method and hence will accept all keyword arguments of the same method.

Parameters **kwargs** – Optional. See keyword args for *pansys.Ansys.send()*

Returns None

send(*command_string*, ***kwargs*)
Sending a command to ansys

Function to send commands to interactive ansys session. Commands can be single line or multiline.

Example

```
>>> ans.send('/post1')
>>> ans.send('''
...     file,results,rst
...     set,last
...     esel,s,mat,,1
...     ''')
```

You can process the output from any ansys command using the `output_function` argument.

Example

```
>>> def parseout(line):
...     if "WARNING" in line:
...         print("Found a warning")
...     else:
...         pass
...
>>> ans.send("set, last", silent=False, output_function=parseout)
```

In the above scenario, "Found a warning" string will be printed for every occurance of a warning for the ansys command `set,last`. For other lines, no action will be taken.

Parameters

- **command_string** (*str*) – Required. The string containing ansys command silent (bool): Optional. Boolean value which when set true will print the output from ansys after executing `command_string`

- **output_function** (*function*) – Optional. A function which will process the output from ansys. The output will be passed line by line to this function. silent option should be set to False for this to work.

Returns None

version
The version of ansys for the current active session.

wd
Current working directory where Ansys is running.

CHAPTER 2

Installing pansys

For pansys to work, you need ANSYS installed on a linux machine.

```
pip install pansys
```

The pansys module relies on a command based interface of ANSYS. By default pansys assumes that you have a command called `ansys150` mapped to the ansys executable. If that does not work for you, use the `startcommand` keyword argument when initiating `Ansys` object and point it to the ANSYS executable.

CHAPTER 3

Examples

It's always easy to learn from examples. Below are some useful examples of how to use the pansys python module.

If you are working interactively with pansys, its recommended to use a `jupyter-notebook` or a `jupyter-console`. The interactivity provided by these interfaces is much better than using a python interpretter.

All the examples shown have been created using `jupyter-notebook`

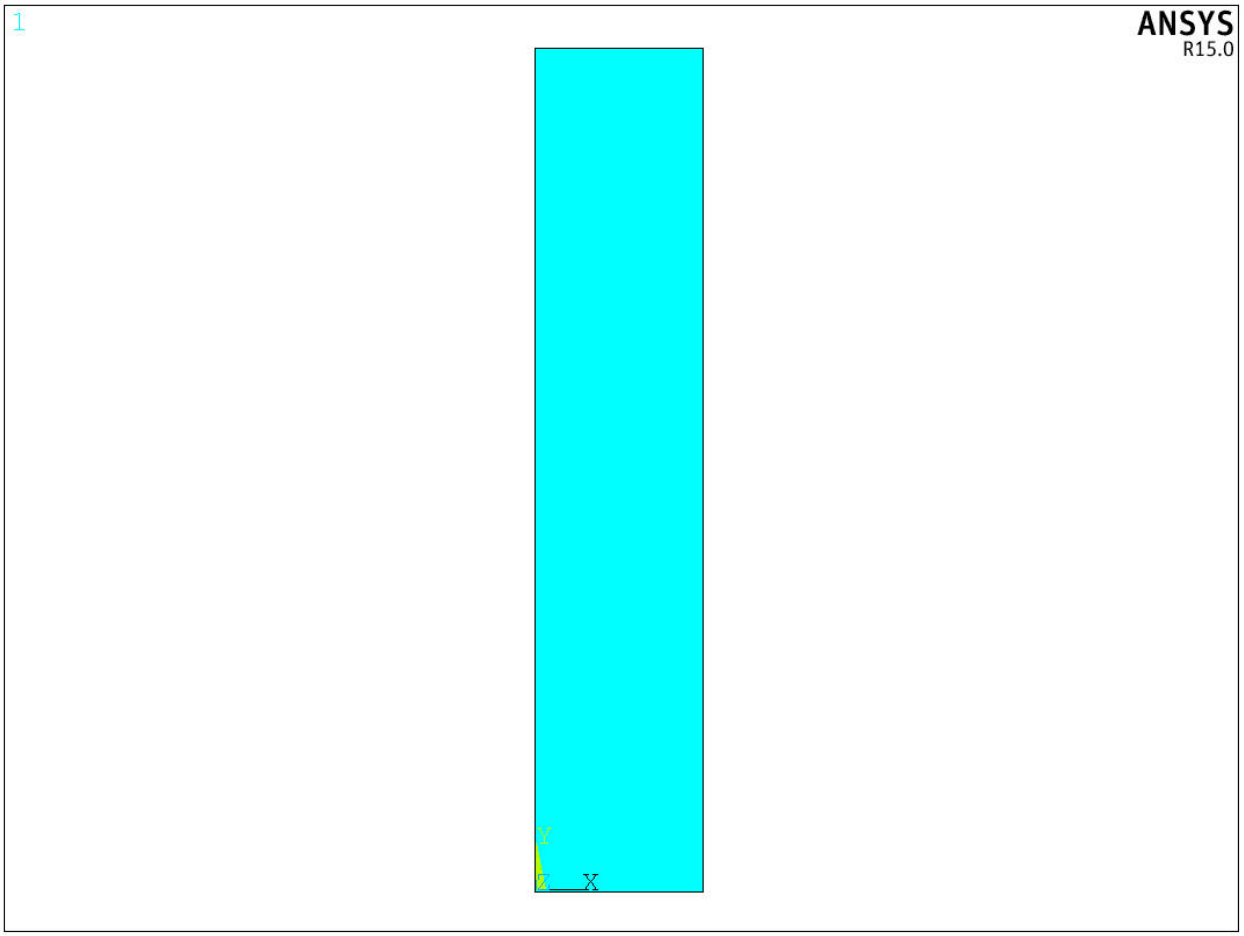
3.1 Ansys solution example

This example shows how you can create a simple geometry directly using pansys, apply boundary conditions, loads, materials etc and run the solution.

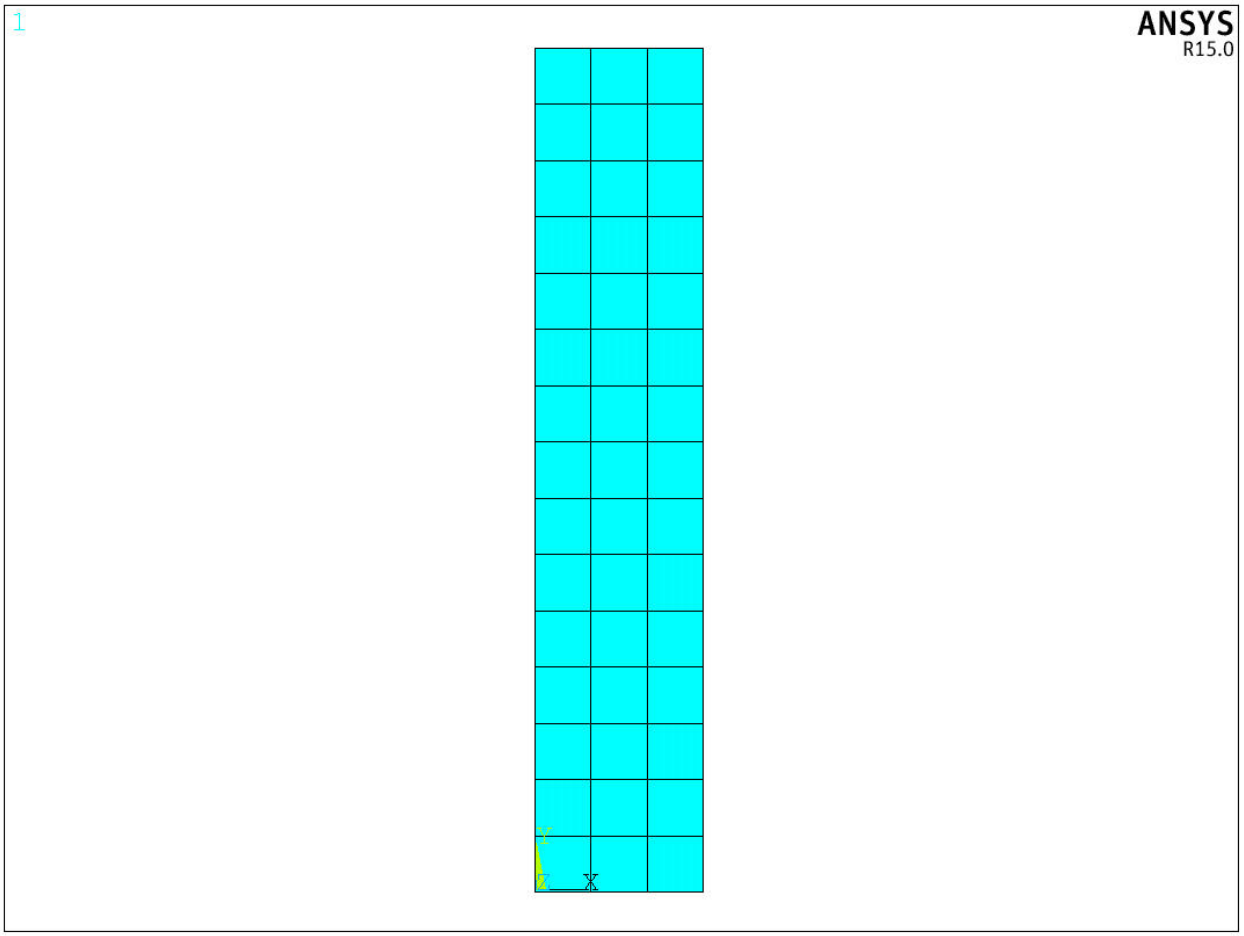
```
In [1]: from pansys import Ansys
In [2]: ans = Ansys(startcommand='ansys150 -p ansys', cleanup=True)
In [3]: ans.send('/prep7')
```

3.1.1 Creating the FE model

```
In [4]: ans.send('block,0,1,0,5,0,10,')
In [5]: img = ans.plot('aplot')
In [6]: from IPython.display import Image
In [7]: Image(img)
```




```
In [8]: ans.send("et,1,SOLID185")
In [9]: ans.send("vsweep, all")
In [10]: Image(ans.plot('eplot'))
```



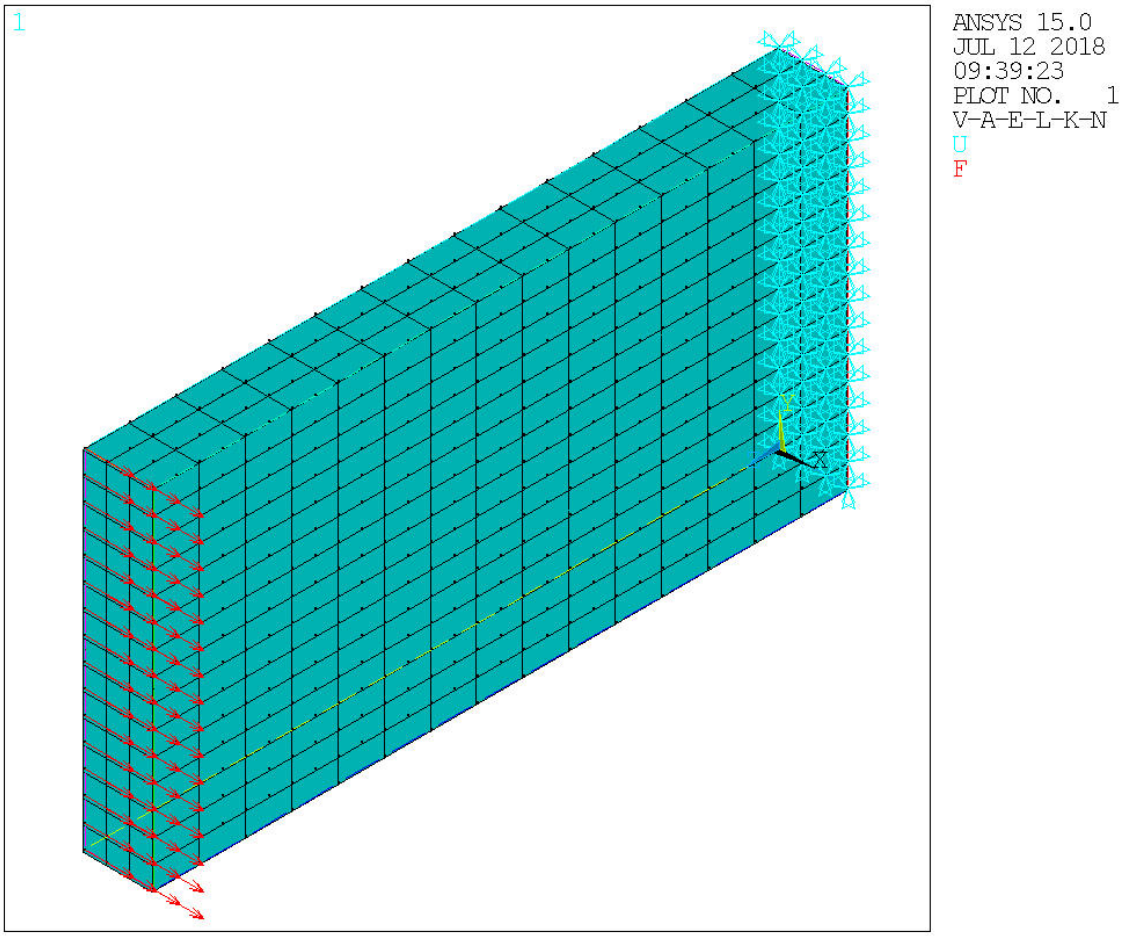
3.1.2 Creating components boundary conditions and loads

```
In [11]: ans.send('nsel,r,loc,z,-0.01,0.01')

In [12]: ans.send('cm,fixity,node')

In [13]: ans.send('nsel,s,loc,z,10-0.01,10+0.01')

In [14]: ans.send('cm,load,node')
```

3.1.3 Applying loads and boundary conditions

```
In [15]: nnodes = ans.get('node','','count')

In [16]: nnodes

Out[16]: 64

In [17]: load_val = 1000

In [18]: ans.send("""
         finish
         /solu
         """)

In [19]: ans.send("""
         cmsel,s,fixity
```

```
         d,all,all
         """)

In [20]: ans.send("""
         cmsel,s,load
         f,all,fx,{}
         """.format(load_val/nnodes))

In [21]: ans.send("allsel")

In [22]: ans.send("""
         /view,1,1,1,1
         /ang,1
         /auto,1
         """)

In [23]: ans.send('/pbc,all,,1')

In [24]: Image(ans.plot('gplot'))
```



3.1.4 Applying material properties

```
In [25]: ans.send('mp,ex,1,1e11')

In [26]: ans.send('mp,mu,1,0.3')
```

3.1.5 Start solution

After issueing the solve command, Ansys may show some warnings and ask whether it you want to proceed or not. pansys always asks ansys to proceed. The assumption here is that pansys is not really a replacement for interactive ANSYS, instead its a means to perform complicated automations with ANSYS. Hence, the user already know what she/he is going to do.

Setting the `silent` key of `send` command to `False` will make pansys show the output of the command in real time. If you want to process the output yourselves, you can pass a function to `send` command with `output_func` keyword argument.

```
In [27]: ans.send('solve', silent=False)
```

```
solve

*****  ANSYS SOLVE    COMMAND  *****
*** NOTE ***                            CP =       0.838   TIME= 09:39:23
There is no title defined for this analysis.
*** SELECTION OF ELEMENT TECHNOLOGIES FOR APPLICABLE ELEMENTS ***
                ---GIVE SUGGESTIONS ONLY---

 ELEMENT TYPE    1 IS SOLID185. IT IS ASSOCIATED WITH LINEAR MATERIALS ONLY
 AND POISSON'S RATIO IS NOT GREATER THAN 0.49. KEYOPT(2)=3 IS SUGGESTED.



                      S O L U T I O N   O P T I O N S

   PROBLEM DIMENSIONALITY. . . . . . . . . . . . .3-D
   DEGREES OF FREEDOM. . . . . . UX   UY   UZ
   ANALYSIS TYPE . . . . . . . . . . . . . . . . .STATIC (STEADY-STATE)
   GLOBALLY ASSEMBLED MATRIX . . . . . . . . . . .SYMMETRIC

 *** WARNING ***                        CP =       0.844   TIME= 09:39:23
 Element 1 (SOLID185) uses material 1 for which Poisson's ratio is
 undefined.  A default value of 0.3 will be used.  Input MP,PRXY, 1,0.3
 to eliminate this warning.

 *** NOTE ***                           CP =       0.846   TIME= 09:39:23
 Present time 0 is less than or equal to the previous time.  Time will
default to 1.
*** NOTE ***                            CP =       0.846   TIME= 09:39:23
 The step data was checked and warning messages were found.
  Please review output or errors file (
 /home/yy53393/github/pansys/docs/_src/examples/pansys_20180712093919/fi
 ile.err ) for these warning messages.

 *** NOTE ***                           CP =       0.846   TIME= 09:39:23
 Results printout suppressed for interactive execute.

 *** NOTE ***                           CP =       0.846   TIME= 09:39:23
 The conditions for direct assembly have been met.  No .emat or .erot
 files will be produced.

 *** WARNING ***                        CP =       0.846   TIME= 09:39:23
 A check of your load data produced 1 warnings.
Should the SOLV command be executed? [y/n]
y
L O A D   S T E P   O P T I O N S
```

```
   LOAD STEP NUMBER. . . . . . . . . . . . . . . .     1
   TIME AT END OF THE LOAD STEP. . . . . . . . . .  1.0000
   NUMBER OF SUBSTEPS. . . . . . . . . . . . . . .     1
   STEP CHANGE BOUNDARY CONDITIONS . . . . . . . .    NO
   PRINT OUTPUT CONTROLS . . . . . . . . . . . . .NO PRINTOUT
   DATABASE OUTPUT CONTROLS. . . . . . . . . . . .ALL DATA WRITTEN
FOR THE LAST SUBSTEP

SOLUTION MONITORING INFO IS WRITTEN TO FILE= file.mntr

Range of element maximum matrix coefficients in global coordinates
 Maximum = 1.083214625E+10 at element 398.
 Minimum = 1.083214625E+10 at element 595.

   *** ELEMENT MATRIX FORMULATION TIMES
  TYPE    NUMBER   ENAME      TOTAL CP  AVE CP

     1       675  SOLID185      0.119   0.000176
Time at end of element matrix formulation CP = 1.12382901.
SPARSE MATRIX DIRECT SOLVER.
Number of equations =        2880,    Maximum wavefront =     60
Memory allocated for solver =                 15.259 MB
  Memory required for in-core =                  6.777 MB
  Optimal memory required for out-of-core =      2.017 MB
  Minimum memory required for out-of-core =      1.608 MB

 *** NOTE ***                            CP =       1.306   TIME= 09:39:26
 The Sparse Matrix solver is currently running in the in-core memory
 mode.  This memory mode uses the most amount of memory in order to
 avoid using the hard drive as much as possible, which most often
 results in the fastest solution time.  This mode is recommended if
 enough physical memory is present to accommodate all of the solver
data.
curEqn=   2881  totEqn=   2880 Job CP sec=      1.150
Factor Done= 100% Factor Wall sec=      0.053 rate=    1553.7 Mflops
Sparse solver maximum pivot= 8.359709376E+10 at node 739 UY.
 Sparse solver minimum pivot= 5.27830482E+09 at node 564 UX.
 Sparse solver minimum pivot in absolute value= 5.27830482E+09 at node
564 UX.
*** ELEMENT RESULT CALCULATION TIMES
  TYPE    NUMBER   ENAME      TOTAL CP  AVE CP

     1       675  SOLID185      0.135   0.000200

   *** NODAL LOAD CALCULATION TIMES
  TYPE    NUMBER   ENAME      TOTAL CP  AVE CP

     1       675  SOLID185      0.012   0.000018
 *** LOAD STEP     1   SUBSTEP     1  COMPLETED.    CUM ITER =      1
*** TIME =   1.00000         TIME INC =   1.00000      NEW TRIANG MATRIX


*** ANSYS BINARY FILE STATISTICS
BUFFER SIZE USED= 16384
1.000 MB WRITTEN ON ASSEMBLED MATRIX FILE: file.full
1.500 MB WRITTEN ON RESULTS FILE: file.rst

SOLU_LS2:
```

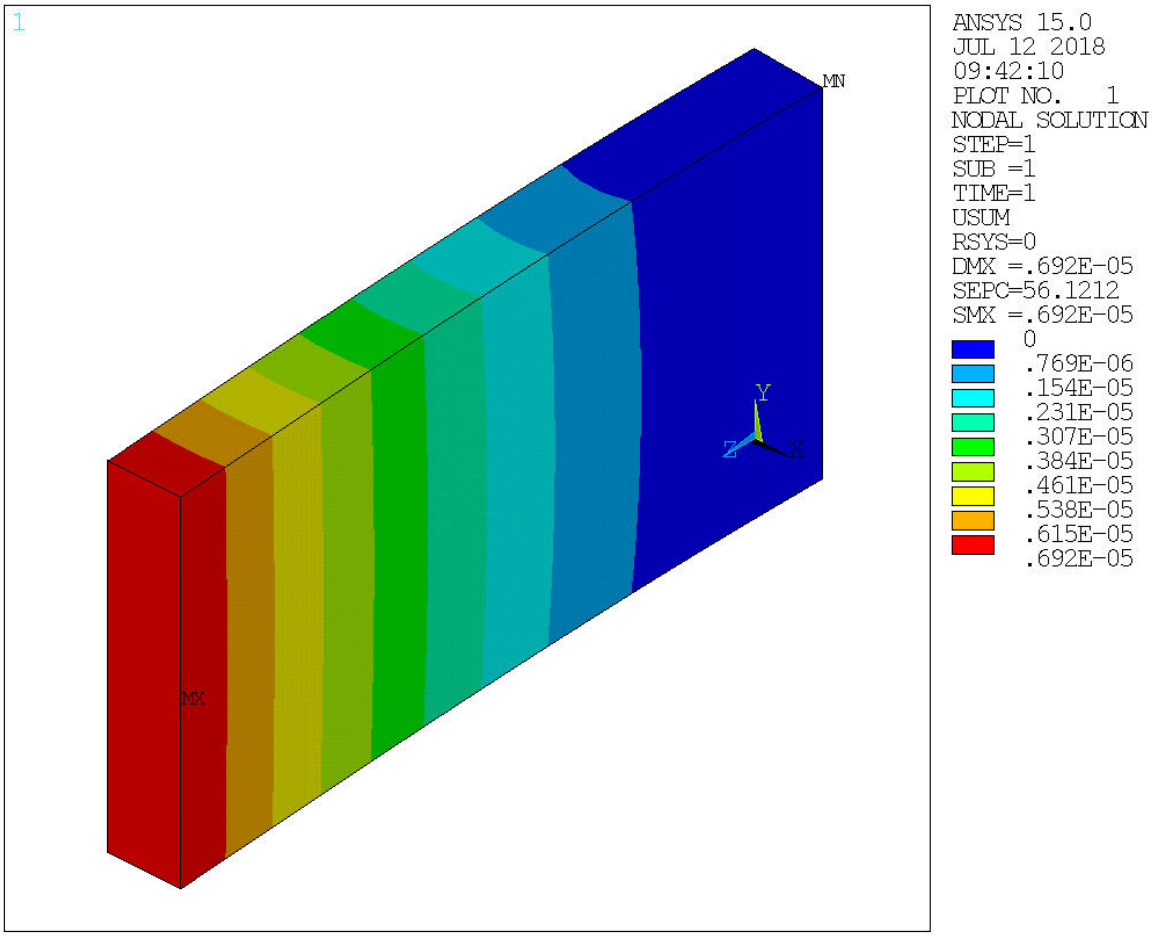
```
In [29]: ans.send("""
         finish
         /post1""")
```

3.1.6 Post processing

You can directly make plots from a Jupyter-notebook using the `Image` method of `IPython.display` module and `plot` method of `Ansys` object.

Any plotting command passed to `plot` function will produce a new image file and return the path to that image file.

```
In [30]: Image(ans.plot('plnsol,u,sum'))
```



Extracting lists

Using the `get_list` method, you can extract any list which will return data in a column format. The function internally uses `pandas.read_table` and passes all keyword arguments from `get_list` to `pandas.read_table`. Hence, you can use your own settings in case need arises.

In the following section, the displacement values from result is extracted and added it to the original coordinates of the nodes to get the updated positions. You can pass this back to the model to by writing out this dataframe and reading it back into model using `nread` command in ANSYS.

Extract displacements

```
In [31]: disp = ans.get_list('prnsol,u,sum')
```

Setting node numbers as the index of the dataframe.

```
In [35]: disp = disp.set_index('NODE')

In [41]: disp.head()
```

Out[41]:

NODE	UX	UY	UZ	USUM
1	4.175461e-08	1.795474e-08	5.657464e-08	7.257073e-08
2	1.556826e-07	2.465593e-08	1.200923e-07	1.981594e-07
3	3.574027e-07	2.815715e-08	1.785558e-07	4.005142e-07
4	6.338732e-07	2.988743e-08	2.333957e-07	6.761376e-07
5	9.814102e-07	2.976412e-08	2.831793e-07	1.021882e-06

Getting the nodal locations

```
In [33]: nlocs = ans.get_list('nlist')

In [38]: nlocs = nlocs.set_index("NODE")

In [42]: nlocs.head()
```

Out[42]:

NODE	X	Y	Z	THXY	THYZ	THZX
1	0.0	0.33333	0.66667	0.0	0.0	0.0
2	0.0	0.33333	1.33330	0.0	0.0	0.0
3	0.0	0.33333	2.00000	0.0	0.0	0.0
4	0.0	0.33333	2.66670	0.0	0.0	0.0
5	0.0	0.33333	3.33330	0.0	0.0	0.0

Creating a copy of nodal location dataframe and modifiying its X,Y and Z values to reflect the displacement results.

```
In [43]: newloc = nlocs.copy()

In [47]: newloc.X = newloc.X + disp.UX
         newloc.Y = newloc.X + disp.UY
         newloc.Z = newloc.X + disp.UZ

In [48]: newloc.head()
```

Out[48]:

NODE	X	Y	Z	THXY	THYZ	THZX
1	1.670184e-07	1.849732e-07	2.235931e-07	0.0	0.0	0.0
2	6.227303e-07	6.473862e-07	7.428226e-07	0.0	0.0	0.0
3	1.429611e-06	1.457768e-06	1.608166e-06	0.0	0.0	0.0
4	2.535493e-06	2.565380e-06	2.768889e-06	0.0	0.0	0.0
5	3.925641e-06	3.955405e-06	4.208820e-06	0.0	0.0	0.0

Extracting stress results

`prnsol,s,prin` will output the principal stresses, stress intensity and equivalent stress as a column data. This when passed to `get_list` command will create a `pandas` DataFrame object which you can do operations on.

```
In [49]: stress = ans.get_list('prnsol,s,prin')

In [57]: stress_comp = ans.get_list('prnsol,s,comp')

In [51]: stress.describe()
```

Out[51]:

	NODE	S1	S2	S3	SINT	\
count	1024.00000	1024.000000	1.024000e+03	1024.000000	1024.000000	
mean	512.50000	1759.415595	3.906240e-10	-1759.415595	3518.831191	
std	295.74764	2360.955818	4.121431e+02	2360.955818	2755.911176	

```
min       1.00000    -67.431017 -1.516078e+03 -10182.646000    249.533180
25%     256.75000    195.068432 -2.051532e+02  -2523.917175   1414.674325
50%     512.50000    555.387185  0.000000e+00   -555.387185   2718.112550
75%     768.25000   2523.917175  2.051532e+02   -195.068432   5313.088775
max    1024.00000  10182.646000  1.516078e+03     67.431017  10396.160000

              SEQV
count  1024.000000
mean   3373.138197
std    2638.761378
min     232.421030
25%    1362.514750
50%    2668.931550
75%    5205.215950
max    9738.712700
```

In [58]:
```
stress_comp.describe()
```

Out[58]:
```
             NODE            SX            SY            SZ            SXY  \
count  1024.00000  1.024000e+03  1.024000e+03  1.024000e+03  1.024000e+03
mean    512.50000 -9.768519e-12 -4.440892e-16 -1.012523e-13 -4.462516e-11
std     295.74764  3.877772e+02  4.235339e+02  4.016076e+03  7.717880e+01
min       1.00000 -1.197029e+03 -1.516337e+03 -9.184036e+03 -5.706479e+02
25%     256.75000 -9.239690e+01 -2.432290e+02 -2.510376e+03 -8.206400e+00
50%     512.50000  0.000000e+00  0.000000e+00  0.000000e+00  6.068757e-14
75%     768.25000  9.239690e+01  2.432290e+02  2.510376e+03  8.206400e+00
max    1024.00000  1.197029e+03  1.516337e+03  9.184036e+03  5.706479e+02

                SYZ          SXZ
count  1.024000e+03  1024.000000
mean  -9.960925e-10   261.900917
std    2.005821e+02   643.002914
min   -1.523964e+03  -766.467980
25%   -5.648059e+01    48.175390
50%   -5.000000e-09   112.944940
75%    5.648059e+01   166.714970
max    1.523964e+03  3476.179700
```

In [56]:
```
stress[stress.SEQV == stress.SEQV.max()]
```

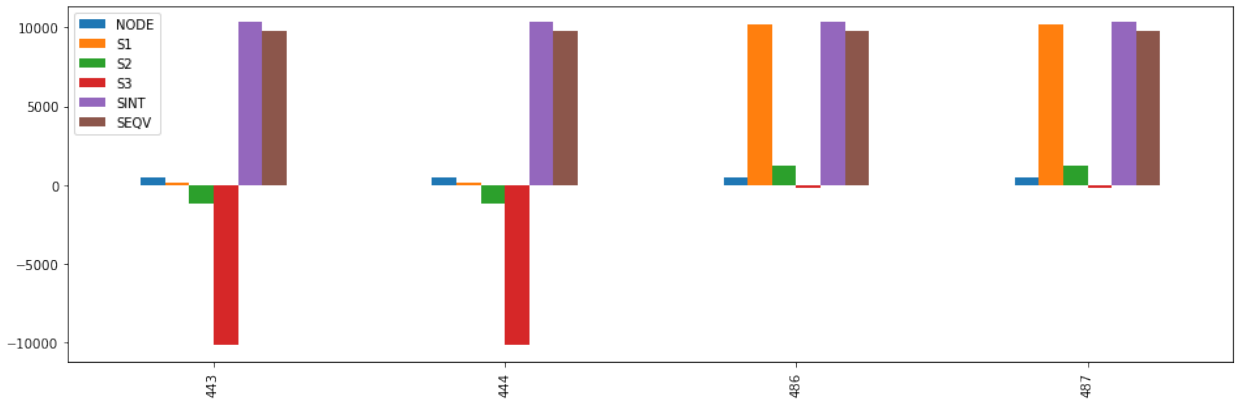
Out[56]:
```
     NODE           S1        S2           S3       SINT       SEQV
443   444    157.91667 -1185.212 -10182.64600  10340.563  9738.7127
444   445    157.91667 -1185.212 -10182.64600  10340.563  9738.7127
486   487  10182.64600  1185.212   -157.91667  10340.563  9738.7127
487   488  10182.64600  1185.212   -157.91667  10340.563  9738.7127
```

Plotting stress results

In [66]:
```
%matplotlib inline
```

In [68]:
```
stress[stress.SEQV == stress.SEQV.max()].plot(kind='bar', figsize=(16,5))
```

Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7cc6d2080>
```



Finding the stress components of nodes with high equivalent stress

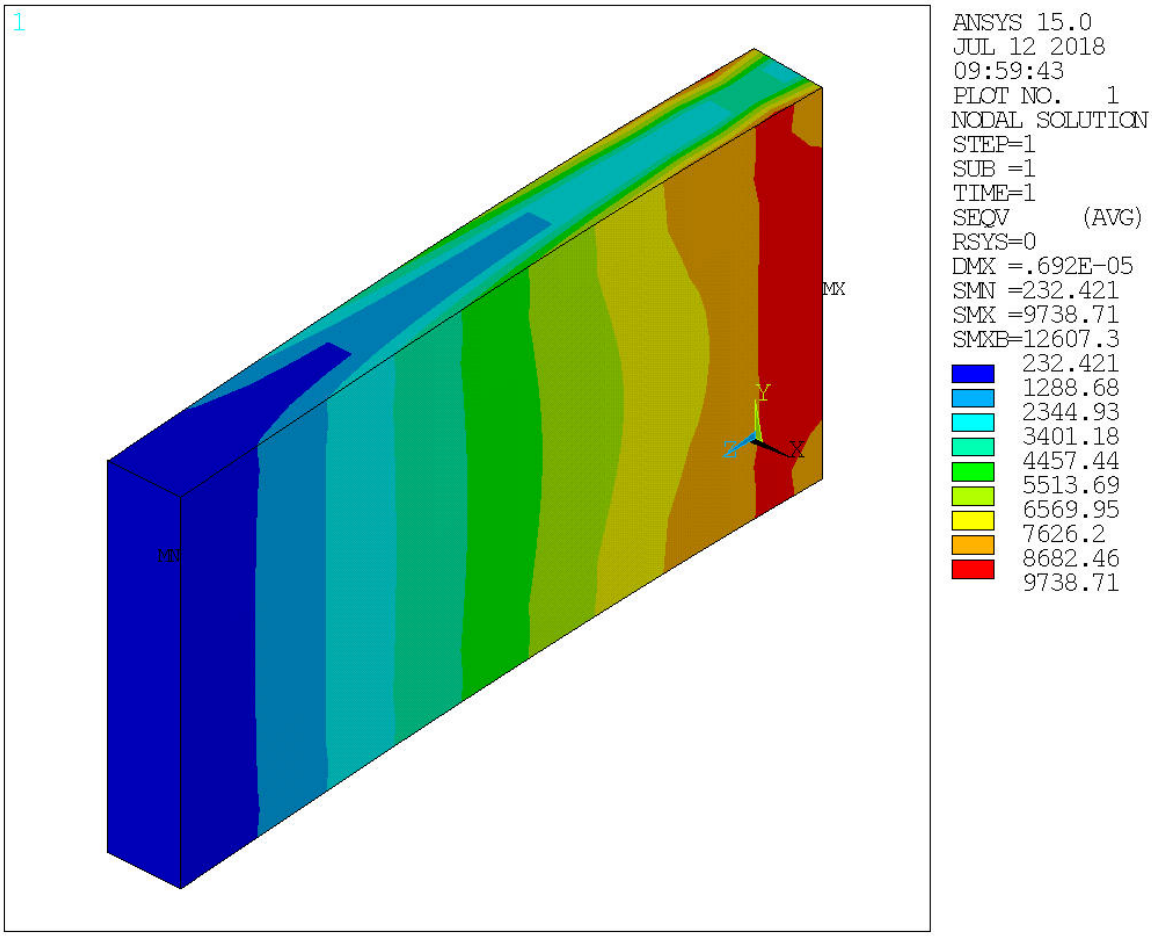
The following commands will extract all nodes where stress is equal to or greater than 95% of max equivalent stress and show their component stresses.

```
In [70]: eqv_nodes = stress[stress.SEQV >= stress.SEQV.max()].NODE.values
         stress_comp[stress_comp.NODE.apply(lambda x: x in eqv_nodes)]
```

Out[70]:

	NODE	SX	SY	SZ	SXY	SYZ	SXZ
443	444	-1185.212	-1185.212	-8839.5173	-2.768785e-12	-29.764415	3476.1797
444	445	-1185.212	-1185.212	-8839.5173	3.013895e-12	29.764415	3476.1797
486	487	1185.212	1185.212	8839.5173	2.440367e-12	-29.764415	3476.1797
487	488	1185.212	1185.212	8839.5173	-2.427766e-12	29.764415	3476.1797

```
In [53]: Image(ans.plot('plnsol,s,eqv'))
```



In [ ]:

3.2 Basic Example

This example shows the basic features of the pansys module. In this example, an ansys session is started and a basic FE model is created.

```
In [1]: from pansys import Ansys
```

To start a new session of ansys, just initialize the class `Ansys()` as shown below.

```
In [2]: a = Ansys()
```

Now we are ready to send commands to the ansys session.

```
In [3]: a.send("/prep7")
```

Let's add a new `BEAM188` element type and some section and material properties.

```
In [4]: a.send("""et,1,188
            sectype,1,beam,csolid
            secdata,0.1
            mp,ex,1,1e12
            mp,prxy,1,0.3
            """)
```

Now let's create our model.

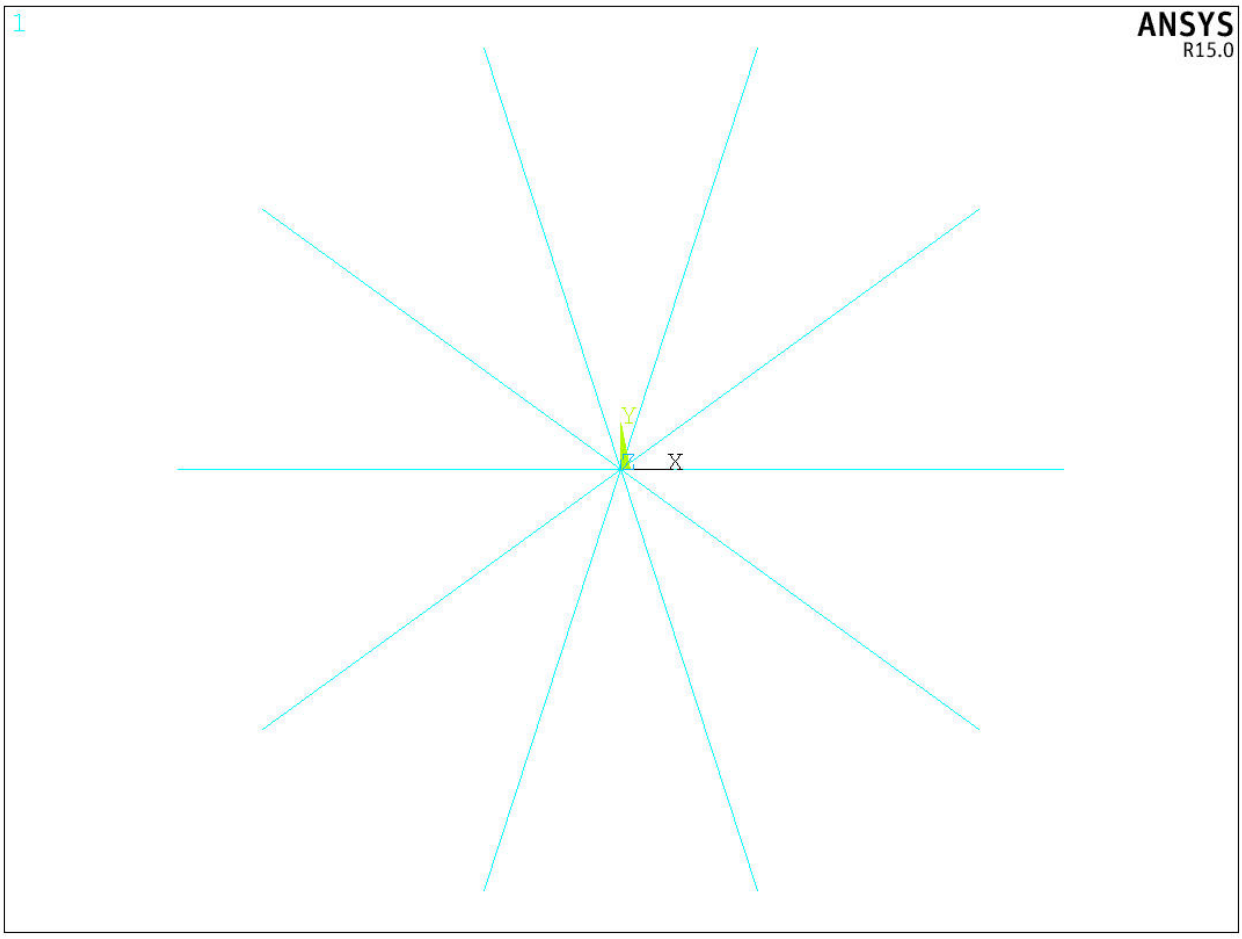
```
In [5]: a.send("csys,1")

In [6]: a.send("n")
        for i in range(10):
            a.send("n,,1,{}".format(360/10*i))
            a.send("e,1,{}".format(i+2))
```

We can take out an ansys plot directly from python. To do that, use the *pansys.Ansys.plot()* function.

```
In [7]: img = a.plot("eplot")
```

Now the `img` variable contains the path to the image file. You can render that in a jupyter notebook using the IPython.display.Image method as shown below.

```
In [8]: from IPython.display import Image
        Image(img)
```



3.3 Basic Example (with SSH)

This example shows the basic features of the pansys module. In this example, an ansys session is started and a basic FE model is created.

```
In [1]: from pansys import Ansys
```

To start a new session of ansys, just initialize the class `Ansys()` as shown below.

```
In [2]: a = Ansys(host='aerox33798', cleanup=True)
```

Now we are ready to send commands to the ansys session.

```
In [3]: a.send("/prep7")
```

Let's add a new `BEAM188` element type and some section and material properties.

```
In [4]: a.send("""et,1,188
            sectype,1,beam,csolid
            secdata,0.1
            mp,ex,1,1e12
            mp,prxy,1,0.3
            """)
```

Now let's create our model.

```
In [5]: a.send("csys,1")
```

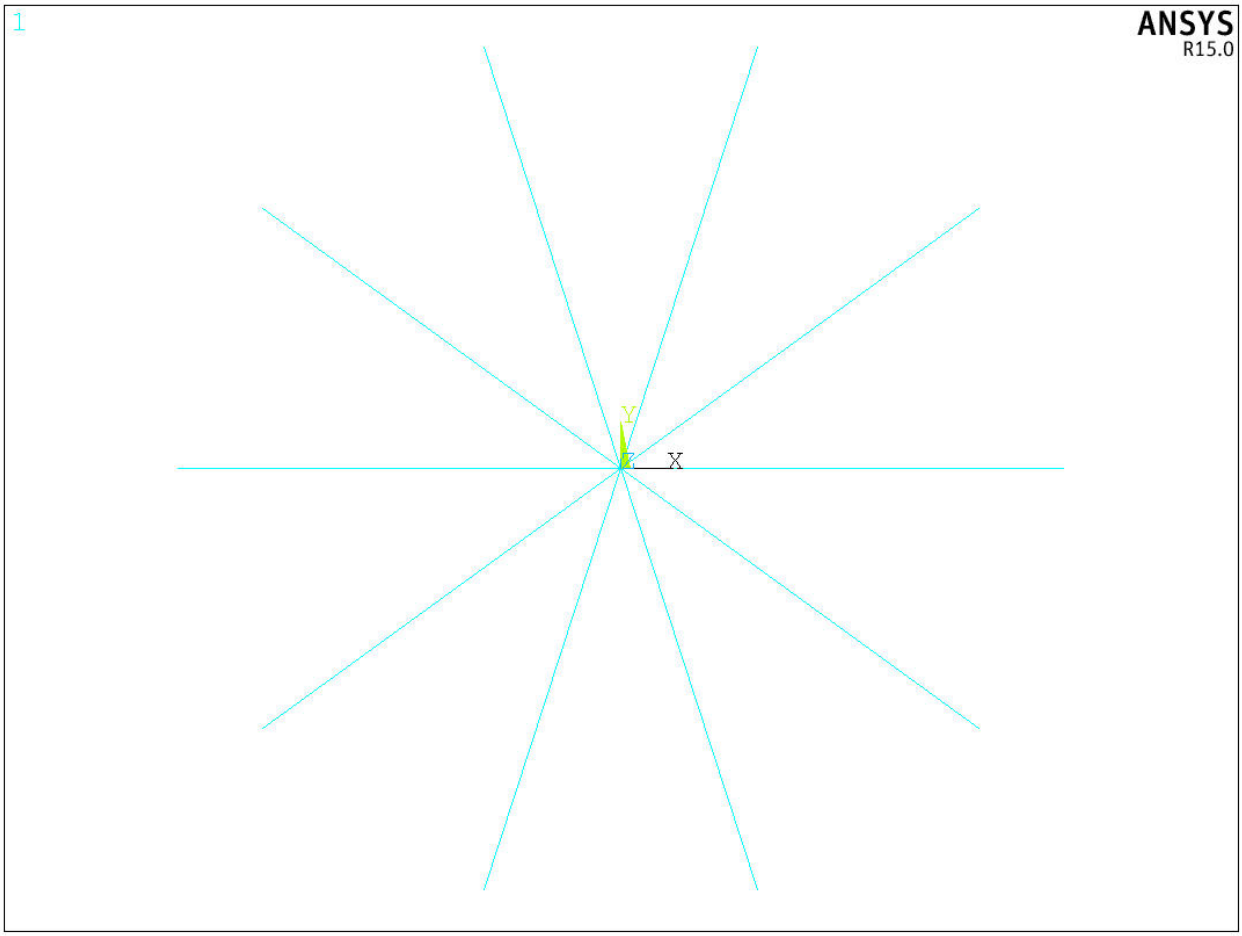
```
In [6]: a.send("n")
        for i in range(10):
            a.send("n,,1,{}".format(360/10*i))
            a.send("e,1,{}".format(i+2))
```

We can take out an ansys plot directly from python. To do that, use the *`pansys.Ansys.plot()`* function.

```
In [7]: img = a.plot("eplot")
```

Now the `img` variable contains the path to the image file. You can render that in a jupyter notebook using the IPython.display.Image method as shown below.

```
In [8]: from IPython.display import Image
        Image(img)
```




```
In [9]: del a
```

3.4 Ansys command output processing

You can extract the output of `Ansys.send` command in multiple ways. This example demonstrates these various methods.

```
In [1]: from pansys import Ansys

In [2]: a = Ansys(cleanup=True)

In [3]: a.send('/prep7')

In [6]: a.send("""
        et,,185
        et,,186
        et,,187
        """)
```

3.4.1 The output property

After any send command is executed, if you access the `output` property of the `Ansys` object, you will get the output of the last command.

Example shown below.

```
In [41]: a.send("etlist")
         print(a.output)
```

```
etlist

 LIST ELEMENT TYPES FROM      1 TO      3 BY      1

 ELEMENT TYPE      1 IS SOLID185     3-D 8-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      2 IS SOLID186     3-D 20-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      3 IS SOLID187     3-D 10-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 CURRENT NODAL DOF SET IS  UX    UY    UZ
  THREE-DIMENSIONAL MODEL

 PREP7:
```

3.4.2 Live output

The output property is accessible only after the command is executed. So it becomes un-usable for commands which take a long time to execute and you want to see the progress live; eg: solution.

In such cases, you can use the `silent` keyword argument of the `send` command as shown below.

```
In [8]: a.send('etlist', silent=False)
```

```
etlist

 LIST ELEMENT TYPES FROM      1 TO      3 BY      1

 ELEMENT TYPE      1 IS SOLID185     3-D 8-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      2 IS SOLID186     3-D 20-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      3 IS SOLID187     3-D 10-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

CURRENT NODAL DOF SET IS  UX    UY    UZ
THREE-DIMENSIONAL MODEL
```

```
PREP7:
```

3.4.3 The output_function

The send command also accepts another keyword arguement called the `output_function` which accepts a function. The output of the send command is passed to this function for further processing. By default, the `output_function` is set to the `print` funtion and hence, the output is printed out.

In the following example, a custom `output_function` is defined which will convert all the text into lower case.

```
In [9]: def output_proc(l):
            print(l.lower())

In [12]: a.send('etlist', silent=False, output_function=output_proc)

etlist

 list element types from      1 to      3 by      1

 element type      1 is solid185     3-d 8-node structural solid
  keyopt( 1- 6)=        0      0      0        0      0      0
  keyopt( 7-12)=        0      0      0        0      0      0
  keyopt(13-18)=        0      0      0        0      0      0

 element type      2 is solid186     3-d 20-node structural solid
  keyopt( 1- 6)=        0      0      0        0      0      0
  keyopt( 7-12)=        0      0      0        0      0      0
  keyopt(13-18)=        0      0      0        0      0      0

 element type      3 is solid187     3-d 10-node structural solid
  keyopt( 1- 6)=        0      0      0        0      0      0
  keyopt( 7-12)=        0      0      0        0      0      0
  keyopt(13-18)=        0      0      0        0      0      0

current nodal dof set is  ux    uy    uz
three-dimensional model

prep7:
```

You can notice that all the output was converted to lowercase letters.

3.4.4 Caveats of the output_function

The data transferred to the output function is not line by line. It depends on the refresh rate of ANSYS output and how quickly python is reading it in.

The below example demonstrates this issue.

```
In [25]: def output_proc2(l):
             print(l)
             print('----------------------')

In [38]: a.send('etlist', silent=False, output_function=output_proc2)

etlist

 LIST ELEMENT TYPES FROM      1 TO      3 BY      1

 ELEMENT TYPE      1 IS SOLID185     3-D 8-NODE STRUCTURAL SOLID
```

```
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      2 IS SOLID186     3-D 20-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      3 IS SOLID187     3-D 10-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0
----------------------

----------------------
CURRENT NODAL DOF SET IS  UX    UY    UZ
----------------------
THREE-DIMENSIONAL MODEL
----------------------

----------------------
PREP7:
----------------------
```

If the input to the `output_function` was line by line, you would have seen `----------` after every line. But that is not the case. This is because the output was read-in as chunks of multiple lines. So, if you want to process line by line, then you will have to split it yourselves by `\r\n`.

```
In [43]: def output_proc3(l):
             for line in l.split('\r\n'):
                 print(line)
                 print('----------')
```

```
In [44]: a.send('etlist', silent=False, output_function=output_proc3)
```

```
etlist
----------

----------
 LIST ELEMENT TYPES FROM      1 TO      3 BY      1
----------

----------
 ELEMENT TYPE      1 IS SOLID185     3-D 8-NODE STRUCTURAL SOLID
----------
  KEYOPT( 1- 6)=        0      0      0        0      0      0
----------
  KEYOPT( 7-12)=        0      0      0        0      0      0
----------
  KEYOPT(13-18)=        0      0      0        0      0      0
----------

----------
 ELEMENT TYPE      2 IS SOLID186     3-D 20-NODE STRUCTURAL SOLID
----------
  KEYOPT( 1- 6)=        0      0      0        0      0      0
----------
  KEYOPT( 7-12)=        0      0      0        0      0      0
----------
```

```
  KEYOPT(13-18)=        0      0      0        0      0      0
---------------------

---------------------
 ELEMENT TYPE      3 IS SOLID187     3-D 10-NODE STRUCTURAL SOLID
---------------------
  KEYOPT( 1- 6)=        0      0      0        0      0      0
---------------------
  KEYOPT( 7-12)=        0      0      0        0      0      0
---------------------
  KEYOPT(13-18)=        0      0      0        0      0      0
---------------------

---------------------
CURRENT NODAL DOF SET IS  UX    UY    UZ
---------------------
THREE-DIMENSIONAL MODEL
---------------------

---------------------
PREP7:
---------------------
```

If you dont want live updates, you can always use the `output` variable.

```
In [36]: for line in a.output.split('\r\n'):
             if 'element type ' in line.lower():
                 temp = [x.strip() for x in line.split()]
                 print(temp[2], temp[4], temp[5], temp[6])
```

```
1 SOLID185 3-D 8-NODE
2 SOLID186 3-D 20-NODE
3 SOLID187 3-D 10-NODE
```

3.4.5 Large outputs

If the output of your command is large and you dont want to keep the entire thing in memory, then there is another method called `get_output` which will write the output of any command passed to it to a file.

```
In [45]: etypes = a.get_output('etlist')

In [46]: etypes

Out[46]: '/home/yy53393/github/pansys/docs/_src/examples/pansys_20180712125145/out.out'

In [48]: !cat $etypes
```

```
 LIST ELEMENT TYPES FROM      1 TO      3 BY      1

 ELEMENT TYPE      1 IS SOLID185     3-D 8-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      2 IS SOLID186     3-D 20-NODE STRUCTURAL SOLID
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 ELEMENT TYPE      3 IS SOLID187     3-D 10-NODE STRUCTURAL SOLID
```

```
  KEYOPT( 1- 6)=        0      0      0        0      0      0
  KEYOPT( 7-12)=        0      0      0        0      0      0
  KEYOPT(13-18)=        0      0      0        0      0      0

 CURRENT NODAL DOF SET IS  UX    UY    UZ
  THREE-DIMENSIONAL MODEL
```

As you can see, the `get_output` command wrote the output to a file called `out.out` in your current working directory. This file will get overwritten every time you use `get_output`. If you want to keep the output file, you should set the `persist` keyword argument to true which will produce a unique filename every time.

```
In [49]: etypes = a.get_output('etlist', persist=True)

In [50]: etypes

Out[50]: '/home/yy53393/github/pansys/docs/_src/examples/pansys_20180712125145/e562762c-155f-466c-897
```

- genindex

Python Module Index

p

Index

Symbols

A

G

O

P

Q

R

S

V

W