Dynare Time Series & Reporting

Houtan Bastani
houtan@dynare.org

CEPREMAP

11 June 2015

Outline

1. Time Series
   - Overview
   - A Programming Note
   - Syntax
     - `dates` Syntax
     - `dseries` Syntax
2. Reporting
   - Overview
   - Syntax
3. Putting it All Together

Outline

1. Time Series
   - Overview
   - A Programming Note
   - Syntax
2. Reporting
3. Putting it All Together

Overview

- Provide support for time series (`dseries`)
  - Based on an implementation for handling dates (`dates`)
- Beta version in Dynare 4.4. Mature version ready in Dynare 4.5
- Currently only used for reporting, though use will increase with time (*e.g.*, to be included in new estimation code)
- Compatible with all setups that are supported by Dynare
  - Windows, Mac OS X, Linux
  - MATLAB 7.5 (R2007b) or later, Octave
- Must run `dynare` or `dynare_config` at least once in the current MATLAB/Octave session before use
- More complete information is included in the Dynare manual

A Programming Note (1/3)

- Time series and dates (and reporting) are implemented as MATLAB/Octave classes
- Inplace modification of instantiated objects not supported. Let me explain . . .
  - A class is a template for defining objects, defining their member variables and methods.
    - *e.g.*, The `dates` class defines 2 member variables–`freq` and `time`–and many methods (analogous to functions)
  - An object is an instance of a specific class. For exemplary purposes, imagine an object `X`, which is an instantiation of the class "Integer". The object has value 1:

```
>> X

X =

     1
```

  - You can call any method defined for the integer class on integer object `X`. Imagine such a method is called `multiplyByTwo()`.
  - In most object-oriented languages, writing `X.multiplyByTwo();` will change the value contained in `X` to 2

A Programming Note (2/3)

- But! For MATLAB/Octave's implementation of classes this is not the case as it does not support inplace modification

```
>> X.multiplyByTwo()

ans =

     2

>> X

X =

     1
```

A Programming Note (3/3)

- To get the desired change, you must overwrite `X`

```
>> X = X.multiplyByTwo()

X =

     2
```

- Keep this in mind when using Dynare dates, time series, and reporting

Syntax

- Two components of a time series:
  - A time component. In Dynare: `dates`
  - A data component mapped to time. In Dynare: `dseries`

`dates` Syntax

- The `dates` command creates an object that represents zero or more dates at a given frequency
- A `dates` object contains 3 members (fields):
  - `freq`: 1, 'y' (Annual); 4, 'q' (Quarterly); 12, 'm' (Monthly); 52, 'w' (Weekly)
  - `time`: A `<<No of dates>>`$\times$2 matrix; the 1st col is the year and the 2nd col is the period
- `dates` members cannot be modified. Thus, this is not allowed

```
>> dd.freq = 12;
```

Creating a new dates object in MATLAB/Octave

- A single date

```
>> t = dates('1999y');
```

- Multiple dates

```
>> t = dates('1999q1', '2020q2', '-190q3');
```

  Notice that noncontiguous and negative dates are possible

- Can also create dates programatically

```
>> t = dates(4, [1990; 1990; 1978], [1; 2; 3])
t = <dates: 1990Q1, 1990Q2, 1978Q3>
```

- Can specify an empty dates...

```
>> qq = dates('Q');
>> qq = dates(4);
```

- ...and use it to instantiate new dates

```
>> t = qq(1990, 1);
t = <dates: 1990Q1>
```

```
>> t = qq([1990; 1990; 1978], [1; 2; 3])
t = <dates: 1990Q1, 1990Q2, 1978Q3>
```

Creating a new dates object in a .mod file

- The preprocessor allows for syntax simplification when dates are in .mod files
- A single date `t = 1999y;`

  ⇒ `t = dates('1999y');`
- Multiple dates `t = [1999q1 2020q2 1960q3];`

  ⇒ `t = [dates('1999q1') dates('2020q2') dates('1960q3')];`
- NB: This can cause problems when dates are included in strings. *e.g.*,

  `disp('In 1999q1, ...')`

  would be transformed into

  `disp('In dates('1999q1'), ...')`
- To fix this, simply prefix any date that you don't want transformed by the preprocessor with a '$': `disp('In $1999q1, ...')`

  ⇒ `disp('In 1999q1, ...')`

Collections and Ranges of dates in a .mod file

- A collection of dates

```
a = 1990Q1; b = 1957Q1; c = -52Q1;
d = [a b c];
⇒ d = <dates: 1990Q1, 1957Q1, -52Q1>
```

- A Range of dates

```
a = 1990Q3:1991Q2
⇒ a = <dates: 1990Q3, 1990Q4, 1991Q1, 1991Q2>
```

- A set of regularly-spaced dates

```
a = 1990Q3:2:1991Q2
⇒ a = <dates: 1990Q3, 1991Q1, 1991Q3>
```

Comparing dates

- The following comparison operators are supported: $==$, $\backsim=$, $<$, $\leq$, $>$, and $\geq$
- Compared objects must have the same frequency
- Compared objects must have the same number of elements (except for singletons)
- Returns the result of an element-wise comparison
- Let `a=[1999W1 2020W3]`, `b=1999W1`, and `c=[1888w1 2020w3]`. Then

```
>> a == b

ans =

     1
     0
```

```
>> a > b

ans =

     0
     1
```

```
>> c < a

ans =

     1
     0
```

Arithmetic Operations on dates

- The unary + and - operators

```
>> a = dates('1999q4');
>> +a ⇒ ans = 2000q1
>> -a ⇒ ans = 1999q3
>> ++-+a ⇒ ans = 2000q2
```

- The binary + and - operators
  - Objects must have the same frequency
  - Objects must have the same number of elements (except for singletons)
  - `1999q4 + 2 ⇒ ans = 2000q2`
  - `1999w4 + 2 ⇒ ans = 1999w6`
  - `1999m4 - 2 ⇒ ans = 1999m2`
  - `1999m4 - 1999m2 ⇒ ans = 2`
  - `1999m4 + 1999m2 ⇒ ans = <dates: 1999M4, 1999M2>`
- The binary * operator

```
>> a*3
ans = <dates: 1999Q4, 1999Q4, 1999Q4>
```

Set Operations on dates

- Let `a = [1990Q1:1991Q1 1990Q1]; b = [1990Q3:1991Q3];`
- `intersect`: returns the intersection of the arguments

```
>> intersect(a, b)
ans = <dates: 1990Q3, 1990Q4, 1991Q1>
```

- `setdiff`: returns dates present in first arg but not in second

```
>> setdiff(a, b)
ans = <dates: 1990Q1, 1990Q2>
```

- `union`: returns the union of two sets (repititions removed)

```
union(a, b)
ans = <dates: 1990Q1, 1990Q2,  ..., 1991Q2, 1991Q3>
```

- `unique()`: removes repititions from set (keeps last unique value)

```
>> a.unique()
ans = <dates: 1990Q2, 1990Q3, 1990Q4, 1991Q1, 1990Q1>
```

Misc dates operations

- Can index a dates object

```
>> a = dates('2000y'):dates('2009y');
>> a(1)
ans = <dates: 2000Y>
>> a(1:3)
ans = <dates: 2000Y, 2001Y, 2002Y>
>> a([1,4,5])
ans = <dates: 2000Y, 2003Y, 2004Y>
```

- pop(): Remove last element

```
>> a.pop()
ans = <dates: 2000Y, 2001Y,  ..., 2007Y, 2008Y>
```

- char(): Return string representation

```
>> a(1).char()
ans = 2000Y
```

- More in the Dynare manual

dseries Syntax

- A dseries is composed of zero or more individual time series
- All time series in a dseries must have the same frequency
- A dseries runs from the earliest date to the latest date, with NaN's inserted to pad the shorter series
- A dseries object contains 4 members:
  - data: The data points
  - dates: The dates of the sample
  - name: Names of the variables
  - tex: LaTeX names of the variables
- dseries members cannot be modified

Creating a new dseries object (1/2)

Load data directly

- Syntax:

```
ts = dseries(DATA, INITIAL_PERIOD, NAMES, TEX_NAMES)
```

- *e.g.*, 2 variables, 'MyVar1' and 'MyVar2', with 3 annual observations starting in 1999:

```
>> ts = dseries([1 2;3 4;5 6], '1999y', ...
                {'MyVar1', 'MyVar2'}, {'MyVar_1', 'MyVar_2'});

ts is a dseries object:

      | MyVar1 | MyVar2
1999Y | 1      | 2
2000Y | 3      | 4
2001Y | 5      | 6
```

Creating a new dseries object (2/2)

Load series from CSV/spreadsheet (`.csv`, `.xls`) or MATLAB file (`.m`, `.mat`)

- Syntax:

```
>> ts = dseries(FILENAME);
```

- File format (`.csv`, `.xls`): dates (optional) in first column (using the standard format: 1990Q1 for quarterly data, 1990Y for annual data, ...) and variable names (optional) in the first row
- File format (`.m`, `.mat`): variables `INIT__`, `NAMES__`, and `TEX__` are optional. More info in the manual. Data are vectors.

```
INIT__ = '1999q1';
NAMES__ = {'cons'};
cons = randn(100,1);
```

Create an empty time series. Useful for renaming dseries.

- `tseries = dseries();`
- `ts = tseries(randn(3,2));`

Creating subsamples from a dseries

- Let ts be a dseries with 3 variables and 5 observations from 2000Y to 2004Y

```
ts=dseries(randn(5,3), '2000y')
```

- To obtain a subsample from 2001Y to 2003Y

```
ts(2001Y:2003Y)
```

- Can also use integer indices (in a roundabout way)

```
start = find(ts.dates==2001Y);
ts(ts.dates(start:end));
```

Extracting variables from a dseries

- Let

```
>> ts = dseries(randn(5,6), '2000q1', ...
                {'GDP_US', 'GDP_FR', 'GDP_JA', ...
                 'CPI_US', 'CPI_FR', 'CPI_JA'});
```

- We can extract one variable using syntax like

```
>> ts.GDP_US
```

- To get all the GDP variables

```
>> ts{'GDP_US', 'GDP_FR', 'GDP_JA'}
```

- A shorter way to do the same thing

```
>> ts{'GDP_@US,FR,JA@'}
```

- To get GDP & CPI (NB: max 2 implicit loops)

```
>> ts{'@GDP,CPI@_@US,FR,JA@'}
```

Applying methods to dseries

- Suppose ts is as above. Then, to apply a method (*e.g.*, log()) to GDP:

```
ts{'GDP_@US,FR,JA@'}=ts{'GDP_@US,FR,JA@'}.log()
```

- To apply a method to a subsample of all the variables:

```
ts(2000Q2:2000Q4) = ts(2000Q2:2000Q4).log()
```

- To apply a method to a subsample of some of the variables:

```
ts(2000Q2:2000Q4){'GDP_@US,FR,JA@'} = ...
  ts(2000Q2:2000Q4){'GDP_@US,FR,JA@'}.log()
```

Merging dseries

- Suppose that ts and ds are two dseries objects with the same variables observed on different time ranges. These dseries objects can be merged using the following syntax:

```
vs = [ts; ds];
```

- Suppose that ts and ds are two dseries objects with different variables observed on the same or different time ranges. These dseries objects can be merged using the following syntax:

```
vs = [ts, ds];
```

If ts and ds are not defined over the same time range, the time range of vs will be the union of ts.dates and ds.dates, NaNs will be added for the missing observations.

Arithmetic Operations on dseries

- The binary operators +, -, *, /, and ^ perform element-wise arithmetic operations on dseries
- They can be used with two dseries or one dseries and a real number
- Two dseries can have different date ranges. Non overlapping dates will be filled with NaNs
- Take addition for example
  - Let

```
>> ts0 = dseries(ones(2,2)*2, '2000W1', {'MyVar1', 'MyVar2'});
>> ts1 = dseries(ones(3,2)*3, '2000W2', {'YrVar1', 'YrVar2'});
>> ds = ts0.MyVar1;
```

  - ts0+3 will add 3 to every element in ts0
  - ts0+[3 4] will add 3 to every element in ts0.MyVar1 and 4 to every element in ts0.MyVar2
  - ts0+ts1 will add ts0.MyVar1 to ts1.YrVar1 and ts0.MyVar2 to ts1.YrVar2. Only 2000W2 will contain the value 5 for both variables. All other points will contain NaN
  - ts0+ds will add ds to ts0.MyVar1 and ts0.MyVar2. All values will contain 4

Leads and Lags with dseries (1/2)

- Let

```
>> ts = dseries([1:4]');
```

- Then

```
>> ts.lead()

ans is a dseries object:

   | lead(Variable_1,1)
1Y | 2
2Y | 3
3Y | 4
4Y | NaN
```

```
>> ts.lag()

ans is a dseries object:

   | lag(Variable_1,1)
1Y | NaN
2Y | 1
3Y | 2
4Y | 3
```

Leads and Lags with dseries (2/2)

- You can lead/lag a `dseries` by more than one period
  - `ts.lead(k)` where $k \in \mathbb{Z}$
  - `ts.lag(k)` where $k \in \mathbb{Z}$
- A shorthand syntax is available as well
  - Lead: `ts(k)` where $k \in \mathbb{Z}$
  - Lag: `ts(-k)` where $k \in \mathbb{Z}$

Outline

1 Time Series

2 Reporting
- Overview
- Syntax

3 Putting it All Together

Overview

- Beta version in Dynare 4.4. Mature version ready in Dynare 4.5
- Introduce reporting functionality to Dynare
  - Input: `dseries`
  - Output: LaTeX report & compiled `.pdf`
- Graphs and Tables are modular

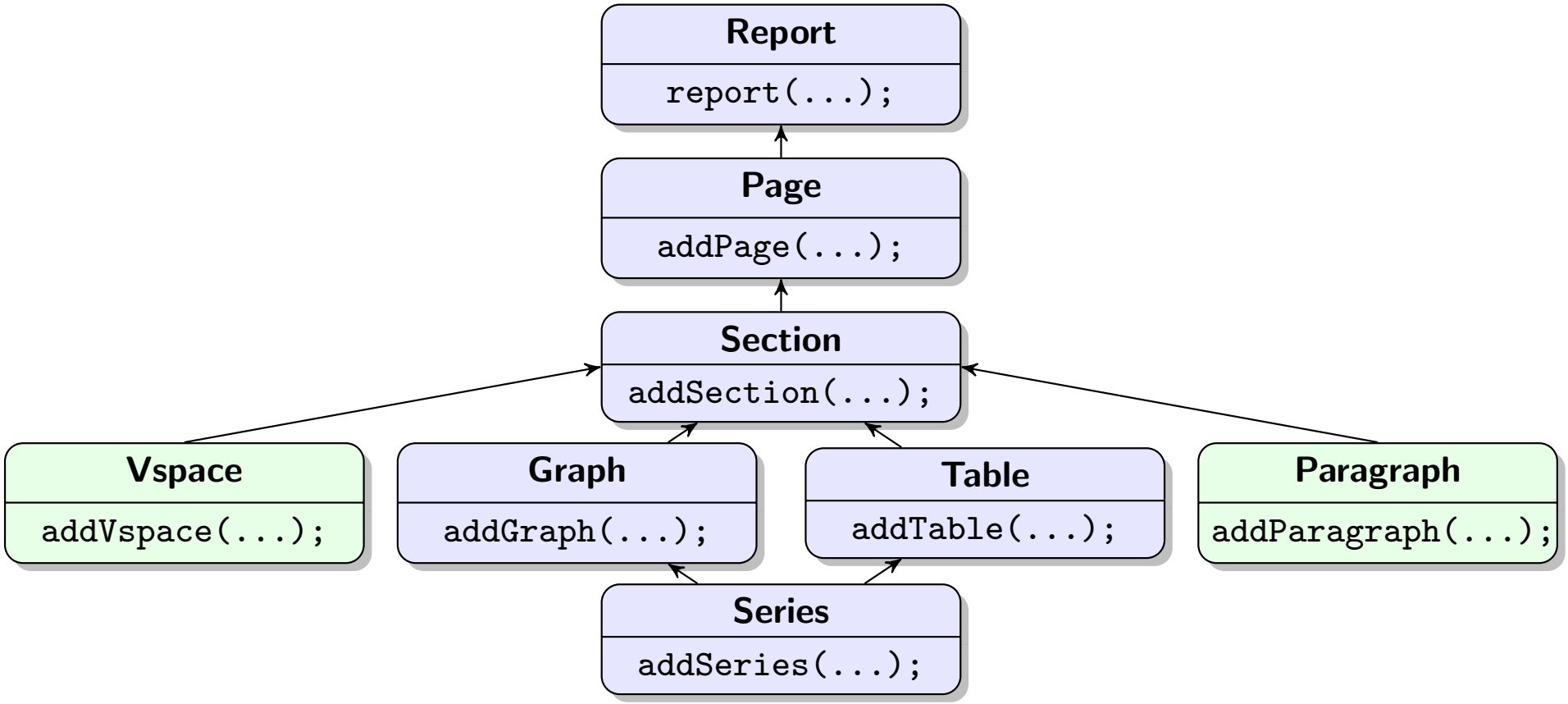
  $\Rightarrow$ Can easily be included in another document
- Graphs are produced in Ti*k*Z/PGFPlots (standard in a TeX distribution)
  - Scales well
  - Formating follows that of enclosing document
- Dynare provides a subset of the many Ti*k*Z options
  - You can easily modify the Ti*k*Z graph if the option you want is not in Dynare
- Works with MATLAB & Octave
- Works much faster than similar softawre
- NB: Must install a LaTeX distribution to compile reports
  - On Windows use MiKTeX: http://miktex.org
  - On Mac OS X use MacTeX: http://tug.org/mactex
  - On Linux use TeX Live: available from your package manager

How Reporting Works

- Reports are created command by command
  - Hence the order of commands matters
- All reporting commands act on the previously added object until an object of greater or equal hierarchy is added (see next slide)
  - *e.g.*, Once you add a `Page` to your report with the `addPage()` command, every `Section` you add via the `addSection()` command will be placed on this page. Only when you add another `Page` will items go on a new page.
  - This will become more clear with an example
- Options to reporting commands are passed in option name/value pairs
  - *e.g.*, `addPage('title', {'Page Title', 'Page Subtitle'})`

Reporting Class Hierarchy

- Class names on the top half of the box, constructor names on the bottom
- Arrows represent what the new object can be added to; objects in green are treated a bit differently (explained below)



Reporting Syntax (1/3)

- `report(...)`: Create a report
  - **Options**: `compiler`, `showDate`, `fileName`, `margin`, `marginUnit`, ...

```
>> rep = report('title', 'Dynare Summer School 2014', ...
                'fileName', 'myDynareReport.tex');
```

- `addPage(...)`: Add a page to the Report
  - **Options**: `footnote`, `orientation`, `paper`, `title`, `titleFormat`

```
>> rep = rep.addPage('title', {'Page Title', 'Page Subtitle'}, ...
                     'titleFormat', {'\large\bfseries', '\large'});
```

- `addSection(...)`: Add a section to the current Page
  - You can think of a section as a matrix. As graphs and/or tables are added to a section, it fills up from left to right. Once you have added `cols` objects, a new row is started.
  - **Options**: `cols`, `height`

```
>> rep = rep.addSection('cols', 3);
```

Reporting Syntax (2/3)

- `addVspace(...)`: Add a vertical space to the current `Section`.
  - If the row has been completely filled in, this adds space between this row and the next row. If not, this adds space at the end of the `Section` and closes it; in other words, in this situation if you want to add more graphs, you'd have to create a new `Section` first.
  - **Options**: `hline`, `number`

```
>> rep = rep.addVspace('hline', 2, 'number', 3);
```

- `addParagraph(...)`: Add text to the current `Section`
  - To add anything other than a paragraph (or multiple paragraphs) to a `Section`, you must add a new `Section` to the page
  - **Options**: `balancedCols`, `cols`, `heading`, `index`, `text`

```
>> rep = rep.addParagraph('text', 'Lorem ipsum\ldots\newline');
```

Reporting Syntax (3/3)

- addGraph(...): Add a graph to the current Section
  - **Options**: data, graphDirName, graphName, graphSize, height, ...

```
>> rep = rep.addGraph('title', 'Headline Inflation (y/y)', ...
                      'xrange', dates('2007q1'):dates('2013q4'), ...
                      'shade', dates('2010q1'):dates('2013q4'), ...
                      'showZeroline', true);
```

- addTable(...): Add a table to the current Section
  - **Options**: data, showHlines, precision, range, seriesToUse, ...

```
>> rep = rep.addTable('title', {'Real GDP Growth','subtitle 1'}, ...
                      'range', dates('2007y'):dates('2014y'), ...
                      'vlineAfter', dates('2011y'));
```

- addSeries(...): Add a series to the current Graph or Table
  - **Options**: data, graphHline, graphLegendName, graphLineColor, ...

```
>> rep = rep.addSeries('data', db_q.LRPOIL_WORLD, ...
                       'graphLineColor', 'blue', ...
                       'graphLineWidth', 1.5, ...
                       'graphMarker', 'triangle*');
```

Output

To create a report:

- `write()`: Writes the report to a LaTeX file
- `compile(...)`: Compiles the report, creating a `.pdf` file
  - **Options**: `compiler`

Report Output

- Unless you pass the `fileName` option to `report(...)`, the report will be located in your working directory with the name `report.tex`. The compiled version will be called `report.pdf`.
- Unless you pass the `graphDirName` or `graphName` options to `addGraph(...)`, your graphs will be in a subdirectory of your working directory called `tmpRepDir`. The default name will take the form `graph_pg9_sec1_row1_col5.tex`
- The same holds for the tables (substituting 'table' for 'graph' above).
- Thus you can easily modify these files and include them in another report.

Outline

1 Time Series

2 Reporting

3 Putting it All Together

Create Report of IRFs from example1.mod (1/3)

- example1.mod is located in the Dynare examples directory
- The lines below can be added at the end of that file.

Create dseries from IRFs

```
shocke = dseries();
shocku = dseries();
@#define endovars=["y", "c", "k", "a", "h", "b"]
@#for var in endovars
  shocke = [shocke dseries(@{var}_e, 2014q3, '@{var}')];
  shocku = [shocku dseries(@{var}_u, 2014q3, '@{var}')];
@#endfor
```

Create Report of IRFs from example1.mod (2/3)

Populate Report

```
r = report();
@#for shock in ["e", "u"]
    r = r.addPage('title',{'Dseries/Report Example','Shock @{shock}'},...
                  'titleFormat', {'\Large\bfseries', '\large\bfseries'});
    r = r.addSection('cols', 2);
@#  for var in endovars
      r = r.addGraph('data', shock@{shock}.@{var}, 'title', '@{var}', ...
                     'showGrid', false, 'yTickLabelPrecision', 2, ...
                     'yTickLabelZeroFill', false, ...
                     'showZeroLine', true, 'zeroLineColor', 'red');
@#  endfor
    r = r.addVspace('number', 2);
    r = r.addSection('cols', 1);
    r = r.addTable('range', 2022q1:2024q1, 'precision', 5);
@#  for var in endovars
      r = r.addSeries('data', shock@{shock}.@{var});
@#  endfor
@#endfor
```

Create Report of IRFs from example1.mod (3/3)

Compile Report

```
r.write();
r.compile();
```

Output Files

```
>> ls report.*
report.aux  report.log  report.pdf  report.synctex.gz  report.tex

>> ls tmpRepDir/
graph_pg1_sec1_row1_col1.tex  graph_pg2_sec1_row1_col1.tex
graph_pg1_sec1_row1_col2.tex  graph_pg2_sec1_row1_col2.tex
graph_pg1_sec1_row2_col1.tex  graph_pg2_sec1_row2_col1.tex
graph_pg1_sec1_row2_col2.tex  graph_pg2_sec1_row2_col2.tex
graph_pg1_sec1_row3_col1.tex  graph_pg2_sec1_row3_col1.tex
graph_pg1_sec1_row3_col2.tex  graph_pg2_sec1_row3_col2.tex
table_pg1_sec2_row1_col1.tex  table_pg2_sec2_row1_col1.tex
```