Chapter 3: Timeseries

Timeseries (in short ts), series associated with dates, can be manipulated very easily in GROCER. Before viewing how to create and manipulate such objects (part 3) and before a description of each function (part 4), it can be useful to know how they are implemented in GROCER (part 1) and how a user can define her own frequencies along with their representation (part 2).

1. How are timeseries built in GROCER?

The type « timeseries » does not exist as such in Scilab. Scilab however allows one to create such a type, by using the « typed list » (tlist) object. A tlist is a peculiar list. Its first element is a string vector. The first co-ordinate of this vector is its type name and the following ones are the names of all the tlist fields. A ts is a tlist T with T(1) = ['ts';'freq';'dates';'series']. That means that the type of the tlist is 'ts' and that its fields are 'freq', 'dates' and 'series'. The ts contains 3 other elements: T(2), T(3) and T(4). Each of these elements can be manipulated as T(i) or T('name') where 'name' is the name recorded as T(1)(i). For example, if you want to recover the frequency of a timeseries ts, you can use ts(2) or ts('freq').

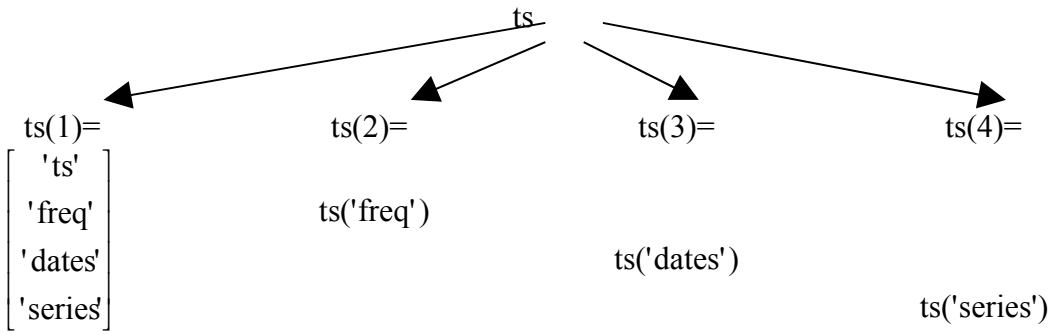
So:

- ts('freq') is the frequency of the timeseries: 1 for annual series, 4 for quarterly ones, 12 for monthly ones,…
- ts('dates') is a real vector (of 'constant' type in scilab) containing the dates covered by the timeseries, as they are coded in GROCER;
- ts('series') is a real vector, corresponding to the values of the timeseries.

<u>Remarks</u>:

- ts('dates') and ts('series') are column vectors with the same size;
- ts('series') is a real vector, with potentially NA values, coded %na in scilab;
- contrary to some other software, the dates of the ts are embodied in the ts and are not associated with the database.

Figure 1: representation of a ts



The use of the typed list tool has one great advantage: it allows the use of the « overloading » capability in Scilab. By writing specific functions (usually very simple), most mathematical (if not all) functions that exist in Scilab, can be applied as such to timeseries. For example, if ts1 and ts2 are

two timeseries, then you can create the new timeseries ts1+ts2, ts1*ts2, log(ts1),… The software will use the GROCER functions %ts_a_ts, %ts_m_ts, %ts_log, … in the library ts, but you will only have to write ts1+ts2, ts1*ts2, log(ts1)… The type list framework has one drawback: manipulation of timeseries is somehow slower than the manipulation of vectors[1]. I have also used the overloading capability to adapt the presentation of a ts: each time you create a ts on screen, GROCER calls the function %ts_p and executes it. So, if the display of ts does not appeal to you, you can modify this function %ts_p according to your wishes.

2. Defining the available frequencies and the corresponding representation of dates (*)

The user can define her own frequencies and the corresponding representation of the dates with the function def_basets. This function has two arguments: a (nx2) matrix of integers and a n vector of strings. Each row of the first argument is the representation of a frequency: [k 1] for frequencies more then annual and [1 k] for frequencies less than annual. A quarterly frequency is then coded: [4 1] and a decennary one [1 10]. The corresponding row of the second argument represents the way it will be coded: 'q' for instance for a quarterly frequencies, but you can choose any other letter ('t' for instance for 'trimestriel' in French).

Grocer contains 5 such default frequencies: annual (symbol 'a'); half_yearly (symbol 'h'); quarterly (symbol 'q'); monthly (symbol 'm'); decennary (symbol 'd'). They have been created by the following grocer command:

```
-->def_basets([1 1 ; 2 1 ; 4 1 ; 12 1 ; 1 10], ['a' ; 'h' ; 'q' ; 'm'  ; 'd'])
```

If you want to rename quarterly dates and add a half-decennary frequency (used for instance in some regressions in the growth empirics field), then enter the following command:

```
-->def_basets([1 1 ; 2 1 ; 4 1 ; 12 1 ; 1 10 ; 1 5], ['a' ; 'h' ; 't' ; 'm'  ;
'd' ; 'g' ])
```

Grocer will then recognize than all dates that contain the letter 'a' is an annual time series, 't' that it is a quarterly ts and that all dates containing the letter 'g' are half-decennary ones…

Note that the code of frequency must be a letter and that you cannot use the letters `'f'` and `'z'`, which are used to denote any frequency previously undefined and that the user wants to use temporarily (higher than and lower than annual respectively).

3. How to create a timeseries?

There are 3 ways for creating timeseries: by using the function reshape, already encountered in the previous example; by importing data from a csv file; as a result of the manipulation of other timeseries.

[1] I have made some tests and among the solutions I have tested, the one which has been adopted is the best… All the stuff needed to implement ts could perhaps have been done in fortran or c, and the result should have been a little bit more efficient… but it would have cost much more time and is left for future developments.

The **function reshape**[2] (see the description p. 13) allows one to transform a vector (row or column) into a timeseries starting at a date defined by the user. The syntax is very simple:
--> ts=reshape(vec,date);
where vec is a (row or column) vector and date is a string like '15a' (annual ts starting at year 15), '1985q1' (quarterly ts starting at the first quarter of year 1985), '1998m5' (monthly ts starting at the fifth month of year 1998) or '42f2d1' (series of periodicity 2 in the year and starting at first period of year 42). Note that only annual, quarterly and monthly series are explicitly implemented in GROCER, but that you can create and use ts of any frequency (by specifying the frequency after the character f in the date and the period after the character d; in the example, the series is a half-yearly one). Note also that if you want to create explicitly new frequency types, you have to adapt the following functions: car2freq, date2fq, date2num&fq, date2num_m, reshape.

The **function impexc2bd** (see for more details chapter 4) allows one to import ts from Excel. There are a few conditions to verify: the series must have the same frequency and the same time span (but #N/A are allowed); the series can be in row or in column, but the first one must have the name "dates" on the first cell and GROCER type dates thereafter (see for instance the file datacons.xls in the library bdexamples); the original Excel file must have been recorded has a csv file (datacons.csv as regards the file datacons.xls).

Last you can create ts by **manipulating existing ts**: this is what the rest of this chapter is devoted to. The existing functions are the following:

- arithmetical functions: +, -, *, /, ^. These functions work in accordance with the intuition. For instance, if ts1 and ts2 are two ts, then "ts1 + ts2" is the sum of the ts ts1 and ts2, "ts1 – ts2" their difference, "ts1 * ts2" their product, "ts1/ts2" the division of ts1 by ts2. Note that the ts must have the same frequency, they must have an overlapping time span and the result exists only on this overlapping time span. If num is a constant then "ts1+num" is the ts where num has been added to every value of ts1, "ts1-num" is the ts where num has been substracted to every value of ts1, "num*ts1" is the ts where every value of ts1 has been multiplied by num, "ts1/num" is the ts where every value of ts1 has been divided by num

Examples:

```
-->ts1=reshape([1:24],'1a'); ts2=reshape(2*[1:10],'18a');tsr=ts1+ts2
```

dates tsr
18a 20
19a 23
20a 26
21a 29
22a 32
23a 35
24a 38

*
* *

-->tsr4=ts1/ts2
dates tsr4
18a 9
19a 4.75
20a 3.3333333
21a 2.625

[2] This function looks very much like the Troll one.

22a 2.2
23a 1.9166667
24a 1.7142857

*
* *

Note that the resulting ts are defined over the period “18a”-“24a”.

Examples:
-->tsr8=ts2/2
dates tsr8
18a 1
19a 2
20a 3
21a 4
22a 5
23a 6
24a 7
25a 8
26a 9
27a 10

*
* *

-->tsr9=ts2^0.5
dates tsr9
18a 1.4142136
19a 2
20a 2.4494897
21a 2.8284271
22a 3.1622777
23a 3.4641016
24a 3.7416574
25a 4
26a 4.2426407
27a 4.472136

*
* *

- comparisons: ts1=ts2, ts1 ≠ ts2 , ts1<ts2, ts1 ≤ ts2 ts1>ts2, ts1 ≥ ts2 , the result being a ts with 0 values when the condition is fulfilled et 1 if not.
- mathematical functions: ceil, cos, exp, floor, log, log10, round, sin, sqrt, tan;
- retrieving parts of the ts tlist: datets, freqts, listna, prtts, series, ts2vec, values;
- elementary transformations of a ts: del, growthr, lagts, subper;

- extending a ts by one or more ts: extrap, overlay;
- changing the frequency of a ts by averaging or taking value from a specified period: m2q, q2a;
- transforming dates or frequencies in numbers or the reverse: car2freq, date2fq, date2num&fq, date2num, date2num_m, freq2car, num2date;
- miscellaneous: mean, sum, cumprod, cumsum, reshape.

3. The ts functions

This section gives the extensive description of the functions shortly described above, presented in alphanumeric order.

car2freq_____transforms a car into a frequency

CALLING SEQUENCE

f=car2freq(car)

PARAMETERS

INPUT:
car = a character (a, q or m)

OUTPUT:
f = an integer

DESCRIPTION
Transforms a character into its corresponding frequency (very basic!).

Example:
f=car2freq('q')

date2fq_____finds the frequency associated to a date

CALLING SEQUENCE
fq=date2fq(dat)

PARAMETERS

INPUT:
dat = a date string

OUTPUT:
fq = an integer (the frequency of the date)

DESCRIPTION
Returns the frequency of a date. Must be extended if one wants to introduce a new default frequency.

Examples:
1) fq=date2fq('1985q4')
2) fq=date2fq('2002a')
3) fq=date2fq('97m12')
4) fq=date2fq('1985f52q4')

These examples return the frequencies of annual, quarterly, monthly and weekly dates (note how the weekly date has been represented).

date2num_fq_____transforms a date into number and frequency

CALLING SEQUENCE
[num,fq]=date2num_fq(dat)

PARAMETERS

INPUT:
dat = a date string

OUTPUT:
* num = the numerical representation of the dates
* fq = its frequency

DESCRIPTION
Returns the numerical representation and the frequency of a date. Must be extended if one wants to introduce a new default frequency

Examples:
1) [num,fq]=date2num_fq('1985q4')
2) [num,fq]= date2num_fq ('2002a')
3) [num,fq]= date2num_fq ('97m12')
4) [num,fq]= date2num_fq ('1985f52q4')

These examples return the frequencies of annual, quarterly, monthly and weekly dates.

date2num_____transforms a date into a number

CALLING SEQUENCE
num=date2num(dat)

PARAMETERS

INPUT:
dat = a date string

OUTPUT:
num = the numerical representation of the date

DESCRIPTION

Returns the numerical representation of a date. Must be extended if one wants to introduce a new default frequency

Examples:
1) d=date2num(‘1985q4’)
2) d=date2num (‘2002a’)
3) d=date2num (‘97m12’)
4) d=date2num (‘1985f52q4’)

These examples return the numerical representation of an annual, a quarterly, a monthly and a weekly date.

date2num_m_____transforms a matrix of dates into numbers

CALLING SEQUENCE
num=date2num_m(dat)

PARAMETERS

INPUT:
dat = a matrix of dates strings

OUTPUT:
* num = the numerical representation of the dates

DESCRIPTION
Returns the numerical representation of a matrix of dates. Must be extended if one wants to introduce a new default frequency.

Examples:
1) v = date2num_m([‘1a’ ’2a’ ’3a’ ’4a’])
2) v = date2num_m(string([1:4])+’a’)
3) v= date2num_m([‘1975q1’; ‘1985q2’])

The first 2 examples give the same results (note that: 1] the transformation of the constant vector [1:4] into the corresponding string 2] the fact that in scilab you can add a (1x1) matrix–here the car ‘a’- to any vector or matrix, the (1x1) is then added to every entry of this matrix): the (1x4) vector [1:4] = [1 2 3 4]. The second example gives the (2x1) vector [7901; 7942]

datelf2hf_____ transformation of a date into a higher frequency

CALLING SEQUENCE
datnew=datelf2hf(dat,divfq,ind)

PARAMETERS

INPUT:
* dat = a string representing a grocer date

* fq = the destination frequency
* ind = a scalar indicating

OUTPUT:
datenew = the corresponding high frequency date

DESCRIPTION
For a given date, gives the corresponding date at a higher frequency, with the corresponding subdivision given by ind. The same purpose as datelf2hf0 but with some tests on the input.

Examples:
1) datelf2hf('1985q2',3,2)
2) datelf2hf('1970a',4,3)

Example 1 gives :
1985m5

And example 2:
1970q3

datelf2hf_____ transformation of a date into a higher frequency

CALLING SEQUENCE
datnew=datelf2hf0(dat,divfq,ind)

PARAMETERS

INPUT:
* dat = a string representing a grocer date
* fq = the destination frequency
* ind = a scalar indicating

OUTPUT:
datenew = the corresponding high frequency date

DESCRIPTION
For a given date, gives the corresponding date at a higher frequency, with the corresponding subdivision given by ind. The same purpose as datelf2hf but without tests on the input.

Examples:
1) datelf2hf0('1985q2',3,2)
2) datelf2hf0('1970a',4,3)

Example 1 gives :
1985m5

And example 2:
1970q3

datets_____returns the time span of a ts

CALLING SEQUENCE
d=datets(ts)

PARAMETERS

INPUT:
ts = a time series

OUTPUT:
d = the numerical representation of ts dates

DESCRIPTION
Returns the dates of a time series.

def_basets_____definition of the frequencies database

CALLING SEQUENCE
def_basets(basefqnum, basefqlit)

PARAMETERS

INPUT:
* basefqnum = a (nr x nc) matrix of frequencies
--> if nc == 1 then all frequencies are at most annual (monthly, quarterly, weekly, ... as well as annual)
--> if nc == 2 then frequencies can be less than annual
in that case each row has a 1 in the first or second cell:
- a 1 in the first cell means that the frequency is less than annual (decades, centuries,...) and the second cell is the frequency (10 for decades, 100 for centuries,...)
- a 1 in the second cell means that the frequency is more than annual (monthly, quarterly, weekly, ... as well as annual) and the first cell is the frequency (12 for monthly, 4 for quarterly, ... 1 for annual)
* basefqlit = a string matrix associating a symbol to each frequency in basefqnum

---
OUTPUT:
nothing: data are saved into the file SCI\macros\grocer\db\basets.dat

DESCRIPTION
Creates the data base of the frequencies and their literal representation.

Examples:
1) def_basets([1 1 ; 2 1 ; 4 1 ; 12 1 ; 1 10], ['a' ; 'h' ; 'q' ; 'm' ; 'd'])
2) def_basets([1 1 ; 2 1 ; 4 1 ; 12 1 ; 1 10; 1 100], ['a' ; 's' ; 't' ; 'm' ; 'd'; 'c'])

Example 1 is taken from function def0_basets. It creates the default grocer ts: annual (symbol 'a'); half_yearly (symbol 'h'); quarterly (symbol 'q'); monthly (symbol 'm'); decennary (symbol 'd').

Example 2 creates the same ts as in example 1, but gives the symbol 't' to the quarterly frequency and adds the centenary frequency, with symbol 'c'.

def0_basets_____definition of the default frequencies database

CALLING SEQUENCE
def0_basets()

PARAMETERS

INPUT:
nothing

OUTPUT:
nothing: data are saved into the file SCI\macros\grocer\db\basets.dat

DESCRIPTION
Creates the default data base of the frequencies and their literal representation. Allows the user to reinstall the default frequencies, for instance if it has been erroneously altered.

delts_____differentiates a ts

CALLING SEQUENCE
tsout=delts(arg1,…,argn)

PARAMETERS

INPUT:
* arg1 = the order of differentiation (optional, set to 1 if not provided, can be >0, =0 or <0) or a ts
* arg2 = a time series x if arg1 is the order of differentiation; omitted if not
---
OUTPUT:
* tsout = the differentiated time series

DESCRIPTION
Computes x(t)-x(t-n) for a timeseries x. Equivalent, but a little bit quicker than x-lagts(n,x)

Examples:
if x is a ts:
1) y = delts(x)
2) y = delts(1,x)
3) y = delts(4,x)
4) y = delts(-3,x)

The first 2 examples are equivalent: y is the result of the differenciation of x. The result of the third one is the ts y = x - x(-4) and the result of the fourth one is the ts = x – x(+3). Note that del(2,x) is not the same as del(del(x)).

diff_date_____returns the difference between 2 dates

CALLING SEQUENCE
num=diff_date(lastdate,firstdate)

PARAMETERS

INPUT:
* lastdate = last date
* firstdate = first date

OUTPUT:
* num = the number of observations between the 2 dates

DESCRIPTION
Computes the number of observations between 2 dates

Examples:
1) d = diff_date('1995a','1985a')
2) d = diff_date('1987q4','1985q2')
3) d = diff_date('72m1','83m12')

The first example gives 10, the second 10 also and the third -143.

extrap_____extrapolates a ts by another ts

CALLING SEQUENCE
tsout=extrap(arg1,…,argn)

PARAMETERS

INPUT:
* arg1,…,argn = n time series

OUTPUT:
* tsout = a time series

DESCRIPTION
Extrapolates the first time series by the growth rate of the following ones over the future of each preceding time series. Can probably be made more efficient.

Examples:

if x is a ts defined over the time span ‘10a’-‘20a’, y a ts defined over the time span ‘19a-28a’ and z a ts defined over the time span ‘1a’-‘35a’:
1) a = overlay(x,y)
2) b= overlay(x,y,z)
3) c= overaly(x,z)

a is a ts defined over the time span ‘10a’-‘28a’, equal to x over the time span ‘10a’-‘20a’, y*value(x/y,’20a’) over the time span ‘21a’-‘28a’. b is a ts defined over the time span ‘10a’-‘28a’, equal to x over the time span ‘10a’-‘20a’, y*value(x/y,’20a’) over the time span ‘21a’-‘28a’, z*value(y/z,’25a’)* value(x/y,’20a’) over the time span ‘26a’-‘28a’.

freq2car_____transforms a frequency into a string

CALLING SEQUENCE
car=freq2car(f)

PARAMETERS

INPUT:
* f= a frequency (1,4 or12)

OUTPUT:
* car = a string

DESCRIPTION
Transforms a frequency in the corresponding string. Must be extended if one wants to introduce a new default frequency

Examples:
1) c = freq2car(1)
2) c = freq2car(4)
3) c=freq2car(12)
Give: ‘a’, ‘q’, ‘m’.

freqts_____ returns the frequency of a ts

CALLING SEQUENCE
f=freqts(ts)

PARAMETERS

INPUT:
ts = a time series

OUTPUT:
f = the frequency of the timeseries

DESCRIPTION

Returns the frequency of a time series. This one line function is equivalent to the instruction: ts('freq').

growthr_____growth rate of a ts

CALLING SEQUENCE
ts=growthr(ts,lag)

PARAMETERS

INPUT:
* ts = a timeseries
* lag = an integer (optional, set to 1 if not provided)

OUTPUT:
* ts = the growthr of the time series

DESCRIPTION
Computes ts(t)/ts(t-lag)-1. If a value for ts is 0, then the corresponding value is set to 'NA'

Examples:
1) gx = growthr(x)
2) ga = growthr(x,12)
gx is simply the growth rate of ts x; if ts is a monthly ts, ga is mom annual growth rate.

lagts_____lag a ts

CALLING SEQUENCE
ts=lagts()

PARAMETERS

INPUT:
* first argument = the oder of differenciation (optional, set to 1 if not provided)
* last argument (which can be the first !) = a time series

OUTPUT:
* ts = the lagged time series

DESCRIPTION
Computes x(t-n) for a timeseries x(t) (very simple indeed !).

Examples:
1) y = lagts(x)
2) y = lagts(4,x)
3) y = lagts(-1,x)

In example 1, y is x lagged once; in example 2, y is x lagged four times; in example 3, y is x forwarded once.

m2q_____transforms monthly data into quarterly ones

CALLING SEQUENCE
tsout=m2q(tsin,ind)

PARAMETERS

INPUT:
* tsin = a monthly timeseries
* ind =
 - -1 if the quarterly data is to be the sum of the corresponding months
 - 0 if the quarterly data is to be the mean of the corresponding months
 - 1 if the quarterly data is to be the value of the first month
 - 2 if the quarterly data is to be the value of the second month
 - 3 if the quarterly data is to be the value of the third month
ind is optional; if not provided, it is set to -1

OUTPUT:
* tsout = the corresponding quarterly time series

DESCRIPTION
Transforms monthly data into quarterly ones.

Examples:
if x is a monthly ts (such as reshape([1:24],'1978m2') then
1) y = m2q(x)
2) y = m2q(x,-1)
3) y = m2q(x,3)

Examples 1 and 2 give exactly the same result: a quarterly ts (from 1978q2 to 1979q4 with values 12, 21, 30, 39, 48, 57, 66 with the ts x above). Example 3 gives a quarterly ts whose values come from the third month of each quarter (from 1978q1 to 1979q4 with values 2, 5, 8, 11, 14, 17, 20, 23 with the ts x above).

num2date_____transforms a number into a date

CALLING SEQUENCE
dat=num2date(num,f)

PARAMETERS

INPUT:
* num = a date in numerical presentation
* f = dates frequency (in constant format)

OUTPUT:
* dat = a date in string format

DESCRIPTION
Transforms a date in numerical presentation into its string format.

Examples:
1) num2date(200,1)
2) num2date(200,4)
3) num2date(200,52)

Example 1 gives "200a", example 2 "49q4", example 3 3f52d44.

overlay_____overlay several ts

CALLING SEQUENCE
tsout=overlay(arg1,…,argn)

PARAMETERS

INPUT:
* arg1,…,argn = n time series

OUTPUT:
* tsout = a timeseries whose periodicity is the periodicity of the first argument (other series with another periodicity are ignored) and whose date range runs from the earliest startdate of any of the input series to the latest enddate of any of the series. For each date in the output series, overlay searches the argument list from left to right looking for a non-NA value in the corresponding position of an input series, or an input argument which is a non-NA constant. The first such value found is placed in the output series in that position. If none is found, then an NA is placed in the output series.

DESCRIPTION
The same as the portable troll function overlay. Creates a timeseries by overlaying several timeseries.

Examples:
if x is a ts defined over the time span '10a'-'20a', y a ts defined over the time span '19a-28a' and z a ts defined over the time span '1a'-'35a':
1) u = overlay(x,y)
2) u = overlay(x,z)
3) u = overlay(x,z,y)
4) u = overlay(z,y)

Example 1 gives the ts defined over the time span '10a'-'28a', equal to x over the time span '10a'-'20a', equal to y over the time span '21a-28a'. Example 2 gives the ts defined over the time span '1a'-'35a', equal to z over the time span '1a-9a', equal to x over the time span '10a'-'20a', equal to z over the time span '21a-35a'. Example 3 gives the same result as example2. Example 4 gives z.

prtts_____print ts

CALLING SEQUENCE
prtts(arg1,…,argn)

PARAMETERS

INPUT:
* arg1,…,argn = n timeseries or names of timeseries (between quotes)

OUTPUT:
* nothing

DESCRIPTION
Prints time series. Timeseries are printed four by four; if they are supplied between quotes, then they are printed under their name; if not, they are printed under the name 'series i'; if there are NA's, a NA is printed. Provided that the final object is a ts, formulas are authorized in any input argument

Examples:
1) prtts(‘x’)
2) prtts(‘x’,’lagts(x)+y’,’y’,z)

Example 1 prints the series ts. Example 2 prints the ts x, lagts(x)+y, y and z, the first 3 are printed with their names, the fourth is printed as “series 4”.

q2a_____transforms quarterly data into annual ones

CALLING SEQUENCE
tsout=q2a(tsin,ind)

PARAMETERS

INPUT:
* tsin = a quarterly timeseries
* ind =
 - -1 if the annual data is to be the sum of the corresponding quarters
 - 0 if the annual data is to be the mean of the corresponding quarters
 - 1 if the annual data is to be the value of the first quarter
 - 2 if the annual data is to be the value of the second quarter
 - 3 if the annual data is to be the value of the third quarter
 - 4 if the annual data is to be the value of the fourth quarter
ind is optional; if not provided, it is set to -1

OUTPUT:
* tsout = the corresponding quarterly time series

DESCRIPTION
Transforms quarterly data into annual ones.

Examples:

if x is a quarterly ts (such as reshape([1:13],'1978q2') then
1) y = m2q(x)
2) y = m2q(x,-1)
3) y = m2q(x,2)

Examples 1 and 2 give exactly the same result: a yearly ts (from 1979a to 1980a with values 22 and 38 with the ts x above). Example 3 gives a yearly ts whose values come from the second quarter of each year (from 1978a to 1981q4 with values 1, 5, 9, 13 with the ts x above).

reshape_____creation of a ts from a vector

CALLING SEQUENCE
ts=reshape(mat,datin)

PARAMETERS

INPUT:
* mat = a (1 x nobs) or (nobs x 1) vector
* datin = a date string

OUTPUT:
* ts = the corresponding time series

DESCRIPTION
Creates a time series from a vector of values and a beginning date.

Examples:
1) x = reshape([21:44],'1a')
2) x = reshape([1:24]','1985q1')
3) x = reshape(y,'1985f2d2')

Example 1 creates an annual ts, from the year 1 to the year 24, with values 21 to 44. Example 2 creates a quarterly ts, from the first quarter of 1985 to the last quarter of 2000, with values 1 to 24. Note that the data are given as a row vector in example 1 and a column vector in example 2: it doesn't matter, provided that you don't give a matrix with more than one dimension. Example 3 creates a half-yearly ts, from the second half-year of 1985 until a date determined by the length of the –supposed-vector y and with values equal to that of vector y.

series_____returns the values of a ts

CALLING SEQUENCE
s=series(ts)

PARAMETERS

INPUT:
ts = a time series

OUTPUT:
s = the vector of the time series values

DESCRIPTION
Returns the vector of a time series values. This one line function is equivalent to the instruction: ts('dates')

st_deviation_____standard deviation of a vector or a ts

CALLING SEQUENCE
sd=st_deviation(x,cr)

PARAMETERS

INPUT:
ts = a time series

OUTPUT:
s = the vector of the time series values

DESCRIPTION
Calculates the standard deviation of a vector or a timeseries.

Example : y=st_deviation(x)

subper_____restrict a ts to a subperiod

CALLING SEQUENCE
ts=subper(ts,arg1,..,argn)

PARAMETERS

INPUT:
* ts = a timeseries
* arg1,…,argn =
 - either a (1 x 2) vector of dates strings
 - or 2 dates strings, representing the begining and the end of the subperiod
---
OUTPUT:
ts = the timeseries over the subperiod

DESCRIPTION
Takes a timeseries over a subperiod.

Example:
y = subper(x,'1985q1','1990q4')

Assuming that the ts x is defined over a period including the interval ['1985q1';'1990q4'], y is x restricted to the period ['1985q1';'1990q4'].

ts2vec_____transforms a ts into a vector

CALLING SEQUENCE
vecout=ts2vec(grocer_ts,arg1,…,argn)

PARAMETERS

INPUT:
* ts = a time series
* arg1,…,argn =
  - an even number of date strings corresponding to the bounds over which the values are taken
  - a (1 x 2*n) vector of date strings corresponding to the bounds over which the values are taken (optional : if not provided, the function operates over the whole time range of the series)

OUTPUT:
* vecout = (N x 1) vector

DESCRIPTION
Transforms the values of a timeseries into the vector of its value.

Examples:
Assume that x is a ts defined over the period '1954m1' – '2002m5' then
1) y = ts2vec(x,'1960m1','1989m12')
2) y = ts2vec(x,['1960m1' '1989m12'])
3) y = ts2vec(x,'1960m1','1989m12','2000m1','2002m5')
4) y = ts2vec(x,['1960m1' '1989m12' '2000m1' '2002m5'])

Examples 1 and 2 do the same thing: take the values of ts y over the period '1960m1' to '1989m12'. Examples 3 and 4 do the same thing: take the values of ts y over the period '1960m1' to '1989m12' and then '2000m1' to '2002m5'.

values_____value of a ts at some date

CALLING SEQUENCE
val=values(ts,dat)

PARAMETERS

INPUT:
* ts = a timeseries
* dat = a date string

OUTPUT:
* val = the value of a timeseries for the given date

DESCRIPTION
Gives the value of a timeseries for a given date.

Example:
y=values(x,'1985Q4')

var_t_____variances of n ts at a given date

CALLING SEQUENCE
[ts]=var_t(arg1,…,argn)

PARAMETERS

INPUT:
* arg1,…,argn = n timeseries

OUTPUT:
* ts = the time series whose values are, for each date the variance at this date of the values of the n time series

DESCRIPTION
Computes the timeseries whose values are, for each date, the variance at this date of the values of n time series (doubtful usefulness, but exists!).

Example:
y = var(ts1,ts2,ts3,ts4)
for each date y is the variance of the values of ts1, ts2, ts3 and ts4 at this date.