finreturns — Generate financial returns[+]

[+]This command is part of StataNow.

Description

`finreturns` takes time-series variables containing the prices of financial assets and computes various types of asset-return series, including differences, percentage differences, log differences, annualized differences, and annualized log differences. All may be adjusted with custom multiplicative and additive factors.

Quick start

Generate simple returns using asset price data in variables `p1`, `p2`, and `p3`

```
finreturns p1 p2 p3
```

Same as above, but use `r` as the stub for generated variables instead of the default stub `_simple_`

```
finreturns p1 p2 p3, simple(r)
```

Same as above, but use `s` as the stub for simple return variables and stub `log` for log return variables

```
finreturns p1 p2 p3, simple(s) log(log)
```

Generate simple returns and multiply all returns by 100 to obtain percentages

```
finreturns p1 p2 p3, simple(r) multiply(100)
```

Menu

Statistics > Financial statistics > Generate financial returns

Syntax

```
finreturns varlist [ if ] [ in ] [ , options ]
```

options	Description
Main	
`simple`[(*newvars* | *stub*[, *retopts*])]	simple returns; the default
`ratio`[(*newvars* | *stub*[, *retopts*])]	ratio returns
`fdiff`[(*newvars* | *stub*[, *retopts*])]	first-difference returns
`log`[(*newvars* | *stub*[, *retopts*])]	log returns
`annual`[(*newvars* | *stub*[, *retopts*])]	annualized ratio returns
`logannual`[(*newvars* | *stub*[, *retopts*])]	annualized log returns
`replace`	overwrite any existing variables that match *newvars* or *stub*
Options	
`multiply(`*varname* | #`)`	multiply generated returns by *varname* or #
`add(`*varname* | #`)`	adjust generated returns additively by *varname* or #
`annualfreq(`#`)`	annualize `annual()` and `logannual()` returns by #
`ceiling`	round default annualization frequency with function `ceil()` instead of with function `floor()`
`preview(`*listspec*`)`	customize preview of generated returns
`nopreview`	suppress preview of generated returns
`numbered`	index new variable names with numbers instead of asset price variable names
`stlabel`	use verbose Stata variable labels for generated variables
`float`	set type of generated variables to `float`

You must `tsset` your data before using `finreturns`; see [TS] **tsset**.

retopts	Description
`multiply(`*varname* | #`)`	multiply generated returns by *varname* or #
`add(`*varname* | #`)`	adjust generated returns additively by *varname* or #
`numbered`	index new variable names with numbers instead of asset price variable names

listspec is

```
[ varlist ] [ in ] [ , listopts ]
```

where *listopts* may be any option for `list`, except for the options listed as summary options and advanced options in [D] **list**.

Options

Main

`simple`[(*newvars* | *stub*[, *retopts*])] computes simple returns, which are the proportion change in prices from the previous period. Thus, simple returns are calculated as the difference between the current and lagged price, divided by lagged price. These are often called net returns. You may specify `simple` with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with `_simple_`. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the `numbered` option is specified. When returns are computed for a single variable, such as `finreturns p1`, the name provided in `simple()` is used as the variable name. When returns are computed for multiple variables, such as `finreturns p1 p2 p3`, the name provided in `simple()` is assumed to be a stub.

retopts can be `multiply()` or `add()` and `numbered`; see their descriptions below.

`ratio`[(*newvars* | *stub*[, *retopts*])] computes the ratio of the current price to the lagged price. These are often called gross returns. You may specify `ratio` with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with `_ratio_`. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the `numbered` option is specified. When returns are computed for a single variable, such as `finreturns p1`, the name provided in `ratio()` is used as the variable name. When returns are computed for multiple variables, such as `finreturns p1 p2 p3`, the name provided in `ratio()` is assumed to be a stub.

retopts can be `multiply()` or `add()` and `numbered`; see their descriptions below.

`fdiff`[(*newvars* | *stub*[, *retopts*])] computes the first difference of prices. You may specify `fdiff` with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with `_fdiff_`. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the `numbered` option is specified. When returns are computed for a single variable, such as `finreturns p1`, the name provided in `fdiff()` is used as the variable name. When returns are computed for multiple variables, such as `finreturns p1 p2 p3`, the name provided in `fdiff()` is assumed to be a stub.

retopts can be `multiply()` or `add()` and `numbered`; see their descriptions below.

`log`[(*newvars* | *stub*[, *retopts*])] computes log returns, which are the logarithm of the price in the current period minus the logarithm of the price in the previous period. When returns are small, log returns approximate simple returns. You may specify `log` with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with `_log_`. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the `numbered` option is specified. When returns are computed for a single variable, such

as finreturns p1, the name provided in log() is used as the variable name. When returns are computed for multiple variables, such as finreturns p1 p2 p3, the name provided in log() is assumed to be a stub.

retopts can be multiply() or add() and numbered; see their descriptions below.

annual[(*newvars* | *stub*[, *retopts*])] computes the annual return, which is the ratio return at an annual rate. You may specify annual with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with _annual_. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the numbered option is specified. When returns are computed for a single variable, such as finreturns p1, the name provided in annual() is used as the variable name. When returns are computed for multiple variables, such as finreturns p1 p2 p3, the name provided in annual() is assumed to be a stub.

retopts can be multiply() or add() and numbered; see their descriptions below.

logannual[(*newvars* | *stub*[, *retopts*])] computes the log return at an annual rate. You may specify logannual with no arguments to assign default Stata variable names to the generated returns. The default names are the names of the specified asset price variables prefixed with _logannual_. The argument may be a list of new variable names for the generated returns. Or the argument may be a stub, in which case the generated returns will be named *stubvar1*, *stubvar2*, etc., where *var1*, *var2*, and so on are the names of the specified asset price variables, or *stub*1, *stub*2, and so on if the numbered option is specified. When returns are computed for a single variable, such as finreturns p1, the name provided in logannual() is used as the variable name. When returns are computed for multiple variables, such as finreturns p1 p2 p3, the name provided in logannual() is assumed to be a stub.

retopts can be multiply() or add() and numbered; see their descriptions below.

replace specifies that any existing variables having the same names specified in *newvars* or *stub* may be overwritten.

Options

multiply(*varname* | #) multiplies all generated asset returns by the variable *varname* or by the constant #. Adjusting by *varname* allows observation-by-observation adjustment rather than the adjustment applied by a constant, multiply(#). Most usefully, multiply(100) transforms the generated asset returns from proportions (such as 0.015) to percentages (such as 1.5). Only one of multiply() or add() may be specified.

When multiply() is specified as part of *retopts*, it affects only the returns generated within the corresponding option such as simple() or log(). You cannot specify multiply() both globally and within the options.

add(*varname* | #) adds *varname* or # to all generated asset returns. add(*varname*) adds the values in *varname* to the generated asset returns; this allows observation-by-observation adjustment. add(#) adds the constant # to the generated asset returns. For example, with this option, a variable such as inflation can be added to or subtracted from all generated returns, so that the generated returns are real returns. Only one of add() or multiply() may be specified.

When add() is specified as part of *retopts*, it affects only the returns generated within the corresponding option such as simple() or log(). You cannot specify add() both globally and within the options.

annualfreq(#) specifies the integer factor used to annualize returns created by annual and logannual. By default, the scaling implied by the frequency of the data is used: 365 for daily data, 12 for monthly data, and 4 for quarterly data. If the data are tsset with a delta other than 1, these default frequencies are 365/delta, 12/delta, and 4/delta, respectively, rounded with the floor() function by default or with the ceil() function if the ceiling option is specified.

ceiling specifies that the default annualization frequency be rounded up with the ceil() function rather than rounded down with the floor() function. Custom annualization frequencies specified with annualfreq() are not rounded, so ceiling is ignored when annualfreq() is specified.

preview(*listspec*) customizes the preview, specifying which variables and observations to list and controlling the format of lists.

nopreview specifies that the preview of return variables should be suppressed from the output.

numbered specifies that new return variable names created by default or by specifying a stub be numbered. The default is to use the asset price variable name in the new return variable name. For example, if the asset price variables are aaa and bbb, then the default simple returns are named _simple_aaa and _simple_bbb. When numbered is specified, the return variables are named _simple_1 and _simple_2. Similarly, if simple(s_) is specified to indicate that s_ be used as a stub, then new variables are named s_aaa and s_bbb by default but named s_1 and s_2 when numbered is specified.

When numbered is specified as part of *retopts*, it affects only the returns generated within the corresponding option such as simple() or log().

stlabel labels the generated returns with verbose labels. By default, the variable label is the capitalized form of the specified asset price variable.

float specifies that the type of the generated variables be float. By default, all return variables are generated as double regardless of the storage type set by users.

Remarks and examples

finreturns generates growth rate variables representing returns from an existing list of variables representing prices. Several growth rates are available, with different properties that may be more or less appropriate for different tasks.

The first difference, sometimes called the dollar return, is just the change in price between two periods,

$$P_t - P_{t-1}$$

which provides a measure of absolute change. More commonly used is a relative or proportional change. This can be in gross terms, known as the ratio return,

$$\frac{P_t}{P_{t-1}}$$

or in net terms, which is known as the simple return:

$$\frac{P_t}{P_{t-1}} - 1$$

Unlike first differences, these are scale-free measures.

However, relative returns are not additive. That is, a 1% increase in a stock's price, followed by a 1% decrease, does not yield back the original price. This is because $(1+0.01)(1-0.01) \neq 1$. For example, if a stock's price falls by 50%, then it must rise by 100% to return to the original value. In some statistical analyses of asset returns, it is important to consider sums of returns. In these situations, the log return is often used:

$$\log P_t - \log P_{t-1} = \log\left(\frac{P_t}{P_{t-1}}\right)$$

For small returns, the log return is close to the simple return, but log returns are additive; a log return of 0.01 in P_t, followed by a log return of -0.01, corresponds to a zero net change in P_t.

Simple and log returns are measured in proportions, so a value of, say, 0.01 corresponds to a 0.01 proportional increase in the price of an asset. Analysts sometimes wish to scale their returns to percentages,

$$100 \times \left(\frac{P_t}{P_{t-1}} - 1\right)$$
$$100 \times (\log P_t - \log P_{t-1})$$

Other times, we may wish to adjust for a time-varying variable. For example, if inflation from time $t-1$ to time t is denoted π_t, then real returns may be expressed as

$$r_t^{\text{gross,real}} = \frac{P_t}{P_{t-1}} \times \frac{1}{1+\pi_t}$$
$$r_t^{\text{log,real}} = \log P_t - \log P_{t-1} - \pi_t$$

where we use the fact that $\log(1+\pi_t)$ is approximately π_t for small values of π_t. The adjustment factors here vary over time, rather than being a constant. `finreturns` allows either multiplicative or additive adjustments, though not both at the same time.

The returns we have discussed so far are one-period returns. But sometimes we wish to express returns at an annual rate. This corresponds to the situation that a monthly return continues for 12 months, a quarterly return for 4 quarters, and so on. `finreturns` can compute annualized ratio returns,

$$\left(\frac{P_t}{P_{t-1}}\right)^m - 1$$

where m is the compounding factor or annualization frequency equal to 12 for monthly data, 4 for quarterly data, etc., and log annual-returns,

$$\log\left(\frac{P_t}{P_{t-1}}\right) \times m$$

Example 1: Simple returns

We use a dataset with stock prices on 25 fictional companies and describe the first few variables.

```
. use https://www.stata-press.com/data/r19/finex
(Fictional stock price data)
. describe datestr-wgt
```

Variable name	Storage type	Display format	Value label	Variable label
datestr	str11	%11s		String date
datem	int	%tm		Monthly date
sp500	double	%10.0g		S&P 500
vol	float	%9.0g		Volatility index
fedfunds	float	%9.0g		Federal funds rate
acme	float	%9.0g		Aciron Medical, Inc.
bat	float	%9.0g		Boron Advanced Technologies
iron	float	%9.0g		Industrial Operations Network
dune	float	%9.0g		Digital Urban Network Enterprise
tyr	float	%9.0g		Tyndale Resources Group
glo	float	%9.0g		Green Logistics, Inc.
spa	float	%9.0g		Space Rocket MFG
wgt	float	%9.0g		Widget Gadgets

We first create the simple return on stock `acme`.

```
. finreturns acme
Simple returns generated for variable acme:
```

	datem	acme	_simple_acme
1.	1955m1	10.00609	.
2.	1955m2	10.24331	.02370763
3.	1955m3	10.38311	.01364765
4.	1955m4	10.49098	.01038863
5.	1955m5	10.59686	.0100931
6.	1955m6	10.75985	.01538048
7.	1955m7	10.67278	-.00809163
8.	1955m8	10.63304	-.00372363
9.	1955m9	10.59028	-.00402204
10.	1955m10	10.48927	-.00953793

By default, the simple return is computed. Because no information about the new variable name was provided, `finreturns` generated a variable with the name `_simple_acme`. The value of 0.0237 in observation 2 of the new variable corresponds to the simple return from period 1 to period 2, an increase of 2.37%.

The output consists of a listing of the first 10 observations of the original variable (`acme`) and all variables generated from that original variable.

When multiple stocks are specified, `finreturns` produces returns for each of them.

```
. finreturns bat iron
Simple returns generated for variable bat:
```

	datem	bat	_simple_bat
1.	1955m1	10.00766	.
2.	1955m2	10.63947	.06313206
3.	1955m3	10.37636	-.02472954
4.	1955m4	10.87464	.04802038
5.	1955m5	10.76794	-.00981138
6.	1955m6	11.72315	.08870823
7.	1955m7	13.10283	.11768852
8.	1955m8	12.848	-.01944843
9.	1955m9	13.79885	.07400814
10.	1955m10	12.84272	-.06929062

```
Simple returns generated for variable iron:
```

	datem	iron	_simple_iron
1.	1955m1	10.00591	.
2.	1955m2	10.45016	.0443978
3.	1955m3	10.29193	-.01514066
4.	1955m4	10.83185	.05246058
5.	1955m5	10.77944	-.00483852
6.	1955m6	11.84796	.09912496
7.	1955m7	13.68152	.15475758
8.	1955m8	13.35943	-.02354157
9.	1955m9	14.41827	.07925792
10.	1955m10	13.08083	-.09276043

Using the `preview()` option, we could customize the look of this output.

If we `describe` the returns generated so far, we will find that `finreturns` has given the generated variables some labels.

```
. describe _simple*
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
_simple_acme    double  %10.0g                ACME
_simple_bat     double  %10.0g                BAT
_simple_iron    double  %10.0g                IRON
```

The default label is the capitalized version of the original variable name. If the original variable names are stock tickers, as will often be the case in financial applications, then the labels on the returns will also be the tickers. The `stlabel` option provides a more verbose label.

Often, you will wish to create many returns series at once and will not wish to type all the price series' variable names. The `finreturns` command understands Stata's varlist conventions; see **[U] 11.4 varname and varlists**. In particular, the notation *varname1-varname2* means all variables between the two specified variable names, inclusive. This feature can be combined with the ability to specify a stub for the returns variable names in `simple()`. We type the `finreturns` command and then use `describe` to see the new variables:

```
. finreturns acme-wgt, simple(r_)
  (output omitted)
. describe r*
```

Variable name	Storage type	Display format	Value label	Variable label
r_acme	double	%10.0g		ACME
r_bat	double	%10.0g		BAT
r_iron	double	%10.0g		IRON
r_dune	double	%10.0g		DUNE
r_tyr	double	%10.0g		TYR
r_glo	double	%10.0g		GLO
r_spa	double	%10.0g		SPA
r_wgt	double	%10.0g		WGT

Eight new returns series—`r_acme`, `r_bat`, and so on—were created, each corresponding to one variable in the original list. The labels on the returns inform you to which series the return corresponds.

◁

▷ Example 2: Different types of returns

So far, we have looked only at simple returns. But `finreturns` can produce multiple types of returns simultaneously. Let's use one stock price `epg` and the `ratio()`, `simple()`, and `log()` options, which each create one new variable, to illustrate generating multiple return types. We also use the `stlabel` option to use more descriptive variable labels. We then use `describe` on the generated variables to see that each one is labeled with more verbose variable labels.

```
. finreturns epg, ratio(ratio_epg) simple(net_epg) log(log_epg) stlabel
Ratio, simple, and log returns generated for variable epg:
```

	datem	epg	ratio_epg	net_epg	log_epg
1.	1955m1	10.00759	.	.	.
2.	1955m2	10.24591	1.023814	.02381396	.02353483
3.	1955m3	10.17058	.99264819	-.00735181	-.00737897
4.	1955m4	10.2799	1.0107486	.01074861	.01069126
5.	1955m5	10.39898	1.011583	.01158298	.01151642
6.	1955m6	10.9156	1.0496805	.04968051	.04848584
7.	1955m7	11.94555	1.0943553	.09435535	.09016546
8.	1955m8	11.92886	.99860297	-.00139703	-.00139801
9.	1955m9	12.75635	1.0693691	.06936906	.06706881
10.	1955m10	12.04943	.94458278	-.05541722	-.05701195

```
. describe *_epg
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
ratio_epg       double  %10.0g                Ratio returns for epg
net_epg         double  %10.0g                Simple returns for epg
log_epg         double  %10.0g                Log returns for epg
```

The preview displays the first 10 observations on each variable we created, as well as the original variable. The ratio (gross) return is one plus the simple (net) return. The log return is the log difference; because the log differences are fairly small, the log returns are close to the simple returns.

◁

▷ Example 3: Adjusting the returns

The returns are stored as proportions, log differences, annualized percentages, or annualized log differences. Some analysts find it easier to interpret percentages, such as 10 rather than 0.1. The `multiply()` option makes converting to percentages easy.

```
. finreturns das, simple(r_das) multiply(100)
(all returns multiplied by 100)
Simple returns generated for variable das:
```

	datem	das	r_das
1.	1955m1	10.00926	.
2.	1955m2	10.31771	3.0816631
3.	1955m3	10.30525	-.12077009
4.	1955m4	10.35527	.48538577
5.	1955m5	10.31676	-.37190946
6.	1955m6	10.37442	.55891598
7.	1955m7	10.77979	3.9074243
8.	1955m8	10.8401	.55941417
9.	1955m9	11.06466	2.0715751
10.	1955m10	10.59127	-4.278394

The returns are now in percentages, still at a monthly rate because the underlying data are monthly.

Sometimes, the variable we would like to adjust by is not a constant. For example, consider the notion of excess returns, which is the return on one asset minus the return on a reference asset. The `add()` option makes computation of such objects easy. Let's find the return of stock `khc` in excess of the return of a reference asset, the nominal interest rate in the `fedfunds` variable.

We first create the variable we wish to adjust by. It is the negative of the interest rate, in monthly percentage changes:

```
. generate double ffr = -fedfunds/1200
```

We divide the interest rate by 100 to get proportions and divide by 12 to get a monthly rate. We also invert the sign, making it negative, because `add()` is additive, not subtractive. Then we can call `finreturns`.

```
. finreturns khc, simple(khc1) add(ffr) preview(in 1/5)
(all returns adjusted additively by ffr)
Simple returns generated for variable khc:
```

	datem	khc	khc1
1.	1955m1	10.00526	.
2.	1955m2	10.34669	.03305012
3.	1955m3	10.2937	-.00624616
4.	1955m4	10.44079	.01309755
5.	1955m5	10.66776	.02054699

In observation 2, stock `khc` returned 3.3% over the monthly interest rate.

◁

▷ Example 4: Annualization of daily returns

To annualize means to calculate the annual equivalent of returns that were measured at a higher frequency, m. Monthly returns, for instance, are measured at a frequency of $m = 12$ times a year. Annual returns are calculated as

$$r_t^{\text{annual}} = \left(1 + r_t^{\text{simple}}\right)^m - 1$$

where $r_t^{\text{simple}} = (P_t/P_{t-1}) - 1$.

But what is the correct m to annualize a daily return? This depends on the number of days a return is actually generated each year, which in turn depends on whether the asset is traded every day of the year and on whether a year is common or leap. In the context of stocks, a popular approach is to assume 20 trading days per month (that is, approximately 4 weeks minus their weekends), which translates to an m of 20 days $\times$ 12 months $= 240$.

To annualize the returns of the `afh` stock using this rule of thumb, we type

```
. finreturns afh, annual(afh_yr240) annualfreq(240)
(annualization frequency m = 240)
Annual returns generated for variable afh:
```

	datem	afh	afh_yr240
1.	1955m1	10.0088	.
2.	1955m2	10.03054	.68326427
3.	1955m3	10.24619	163.85376
4.	1955m4	10.1483	-.90012114
5.	1955m5	10.49291	3022.9268
6.	1955m6	10.62643	19.796663
7.	1955m7	11.34264	6287254.9
8.	1955m8	11.20041	-.95161606
9.	1955m9	11.60558	5055.8637
10.	1955m10	11.05977	-.99999047

Note that, if you do not specify a frequency with the annualfreq() option, finreturns assumes 365 trading days per year, which may or may not be appropriate for the assets you want to study.

◁

Stored results

finreturns stores the following in r():

Scalars

r(annualfreq)	frequency for annualization
r(ceiling)	1 if ceiling specified, 0 otherwise

Macros

r(varlist)	variable names in *varlist*
r(newvars)	names of all variables created by finreturns
r(simplevars)	new variables storing simple returns, if specified
r(ratiovars)	new variables storing ratio returns, if specified
r(fdiffvars)	new variables storing first-difference returns, if specified
r(logvars)	new variables storing log returns, if specified
r(annualvars)	new variables storing annualized ratio returns, if specified
r(logannualvars)	new variables storing annualized log returns, if specified
r(multiply)	constant or variable name used for multiplicative adjustment
r(add)	constant or variable name used for additive adjustment
r(simple_multiply)	multiplicative adjustment used for simple returns
r(ratio_multiply)	multiplicative adjustment used for ratio returns
r(fdiff_multiply)	multiplicative adjustment used for first-difference returns
r(log_multiply)	multiplicative adjustment used for log returns
r(annual_multiply)	multiplicative adjustment used for annualized ratio returns
r(logannual_multiply)	multiplicative adjustment used for annualized log returns
r(simple_add)	additive adjustment used for simple returns
r(ratio_add)	additive adjustment used for ratio returns
r(fdiff_add)	additive adjustment used for first-difference returns
r(log_add)	additive adjustment used for log returns
r(annual_add)	additive adjustment used for annualized ratio returns
r(logannual_add)	additive adjustment used for annualized log returns

Methods and formulas

Methods and formulas are presented under the following headings:

Basic formulas

Let P_t be any time series, though we will usually think of it as a price series. Then the returns generated by the ratio() option are calculated as

$$r_t^{\text{ratio}} = \frac{P_t}{P_{t-1}}$$

The returns generated by the fdiff() option are calculated as

$$r_t^{\text{fdiff}} = P_t - P_{t-1}$$

The returns generated by the `simple()` option are calculated as

$$r_t^{\text{simple}} = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1 = r_t^{\text{ratio}} - 1$$

The returns generated by the `log()` option are calculated as

$$r_t^{\log} = \log\left(\frac{P_t}{P_{t-1}}\right) = \log P_t - \log P_{t-1}$$

The returns generated by the `annual()` option are calculated as

$$r_t^{\text{annual}} = \left(r_t^{\text{ratio}}\right)^m - 1$$

where m depends on the frequency of the data. For quarterly data, $m = 4$; for monthly data, $m = 12$; for weekly data, $m = 52$; and for daily data, $m = 365$.

The returns generated by the `logannual()` option are calculated as

$$\log\left(\frac{P_t}{P_{t-1}}\right) \times m$$

Options for adjusting returns

The `multiply()` and `add()` options adjust the generated returns.

`multiply(#)` multiplies all generated returns by a constant. For instance, `multiply(100)` multiplies all returns by 100, thus converting proportions into percentages.

`multiply(`*varname*`)` multiplies each observation in the return series by the corresponding value in the specified *varname*. If that variable takes on value c_t at time t, then this option computes

$$r_t^{\text{adj}} = r_t c_t$$

The `add()` option makes the adjustment additively rather than multiplicatively and can also be specified with either a constant or a *varname*. Hence,

$$r_t^{\text{adj}} = r_t + c_t$$

For example, if the returns were nominal, and the variable c_t held the negative of the inflation rate, then `add()` would provide a one-line solution for obtaining real returns.

References

Campbell, J. Y. 2018. *Financial Decisions and Markets: A Course in Asset Pricing*. Princeton, NJ: Princeton University Press.

Hurn, S., V. L. Martin, P. C. B. Phillips, and J. Yu. 2020. *Financial Econometric Modeling*. Oxford: Oxford University Press.

Also see

[FIN] **finportfolio** — Financial portfolio selection+

[TS] **tsset** — Declare data to be time-series data