Operators

# Detailed Operator Descriptions

## Table of Contents

- hump(x, hump = 0.01)
- hump_decay
- jump_decay
- ts_regression(y, x, d, lag = 0, rettype = 0)
- ts_co_skewness(y, x, d)
- ts_co_kurtosis(y, x, d)
- ts_corr(x, y, d)
- ts_triple_corr(x, y, z, d)
- ts_partial_corr(x, y, z, d)
- ts_moment(x, d, k)
- ts_skewness(x,d)
- ts_kurtosis(x, d)
- ts_decay_exp_window(x, d, factor = 1.0, nan = True)
- ts_percentage(x, d, percentage=0.5)
- ts_weighted_decay(x, k=0.5)
- ts_arg_max(x, d)
- ts_av_diff(x, d)
- ts_returns (x, d, mode = 1)
- ts_scale(x, d, constant = 0)
- ts_entropy(x,d, buckets=10)
- ts_rank_gmean_amean_diff(input1, input2, input3,...,d)

4. Cross Sectional Operators
   - rank(x, rate=2):
   - rank_by_side (x, rate=2, scale=1)
   - generalized_rank(open, m=1)
   - regression_neut (Y, X)
   - regression_proj (Y, X)
   - scale (x, scale=1, longscale=1, shortscale=1)
   - scale_down(x,constant=0)
   - truncate(x,maxPercent=0.01)
   - vector_neut(x,y)
   - quantile(x,driver="gaussian",sigma=1.0)

5. Group Operators
   - group_backfill(x, group, d, std = 4.0)
   - group_coalesce(original_group, group2, group3,...)
   - group_neutralize(x, group)
   - group_normalize(x, group, constantCheck=False, tolerance=0.01, scale=1)
6. Logical Operators
   - if_else(event_condition, Alpha_expression_1, Alpha_expression_2)

# Arithmetic Operators

## log_diff(x)

The operator calculates the log of the data field value today and subtracts from it the log of the data field values of yesterday; Ensure your data does not contain NaNs or if it does, process the data so as to remove the NaNs

**Example**:

```
-log_diff(close)
```

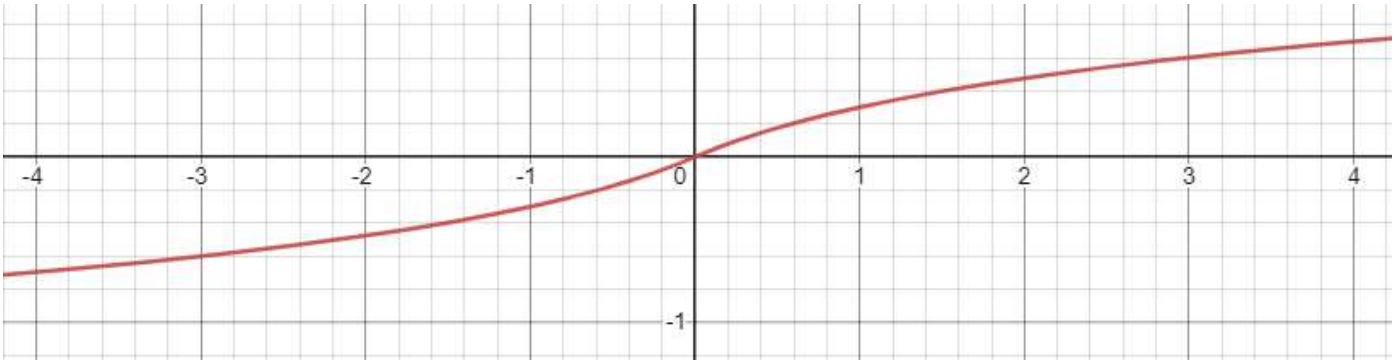
Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 2 | 1 | 0.01 | Industry | On | Off | Verify |

## s_log_1p(x)

sign(x) * log(1 + abs(x))
Confine function to a shorter range using logarithm such that higher input remains higher and negative input remains negative as an output of resulting function and -1 or 1 is an asymptotic




**Example:**
If x = 9 ⇒ abs(x) = 9 ⇒ 1+abs(x) = 10 ⇒ log(1+abs(x)) = 1 and sign(x) = 1 ⇒ sign(x) * log(1+abs(x)) = 1
If x = -9 ⇒ abs(x) = 9 ⇒ 1+abs(x) = 10 ⇒ log(1+abs(x)) = 1 and sign(x) = -1 ⇒ sign(x) * log(1+abs(x)) = -1

---

## densify(x)

This operator converts a grouping field of many buckets into lesser number of only available buckets so as to make working with grouping fields computationally efficient. The below example will make the implementation more clear.

**Example:**
Say a grouping field is provided as integer (industry: tech -> 0, airspace -> 1, ...) and for certain date, we have instruments having grouping field values among {0, 1, 2, 99}, so, instead of creating 100 buckets and keeping 96 of them empty it is better to just create 4 buckets with values {0,1,2,3}. So, if number of unique values in x is n, densify maps values those values between 0 to (n-1). Magnitude order need not be preserved.

---

## max(x, y..)

**Example**:

```
1  max (close, vwap)
```

Open example alpha in Simulate

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 2 | 1 | 0.01 | Industry | On | Off | Verify |

## min (x, y..)

**Example:**

```
min(close, vwap)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## multiply (x, y,.., filter=true)

**Examples**:

```
multiply(rank(-returns), rank(volume/adv20), filter=true)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## nan_mask (x, y)

**Example:**

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## replace (x, target="v1 v2 ..vn", dest="d1,d2,..dn")

The operator replaces the target values in input with the destination values; Possible combinations of target value numbers and destination value numbers are (N target -> N destination; 1->1 and N->1 replacement); The target and destination values are entered within quotes and are space separated.

**Examples**:

replace (returns, target="0.1 0.2", dest="0 0"); replace (returns, target="0.1 0.2", dest="0");

replace (returns, target=0.1, dest=0)

```
1  replace (returns, target=0.1, dest=0)
```

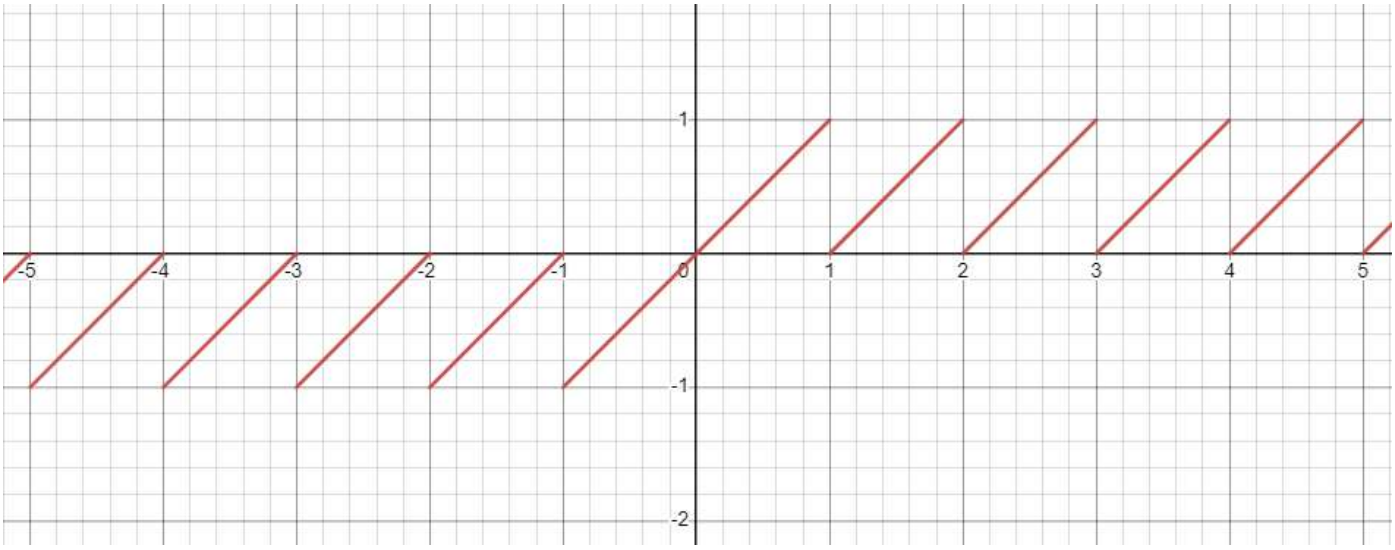
Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## fraction(x)

**sign(x) * (abs(x) – floor(abs(x)))**
This operator removes the whole number part and returns the remaining fraction part with sign. Range is between -1 to 1 and it is an odd function

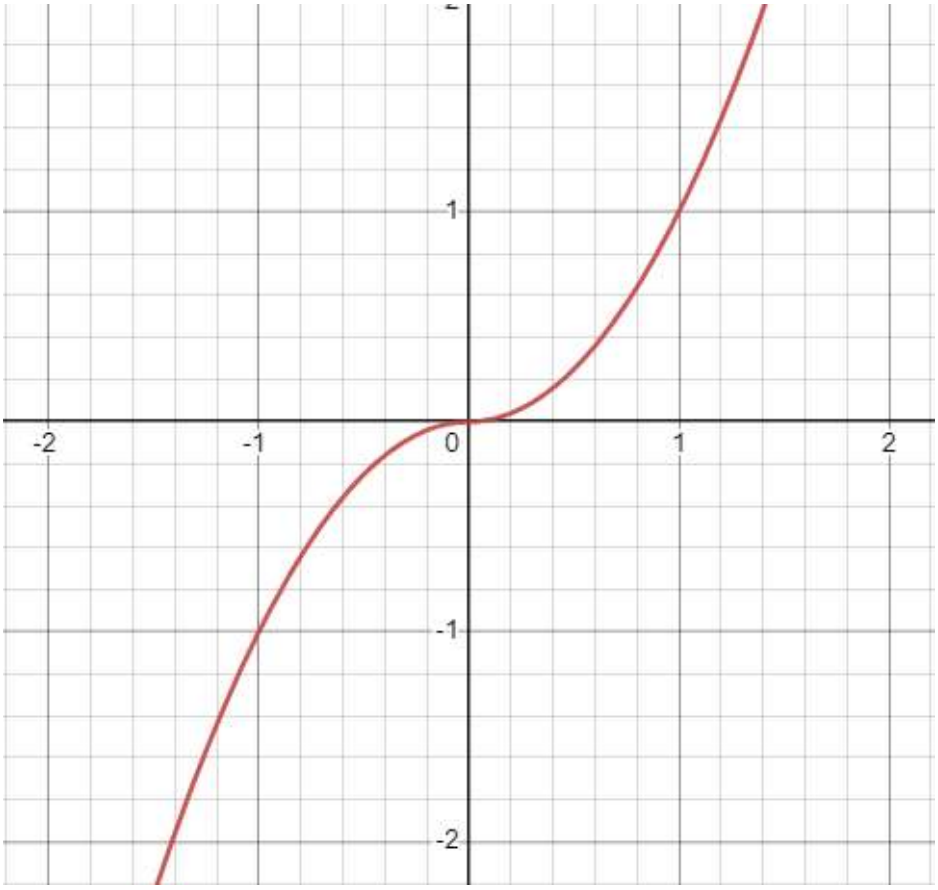



Example:
If x = 5.63 ⇒ abs(x) = 5.63 ⇒ floor(abs(x)) = 5 ⇒ (abs(x) – floor(abs(x))) = (5.63-5) = 0.63 and sign(x) = +1 => sign(x) * (abs(x) – floor(abs(x))) = fraction(x) = 0.63
If x = -4.59 ⇒ abs(x) = 4.59 ⇒ floor(abs(x)) = 4 ⇒ (abs(x) – floor(abs(x))) = (4.59-4) = 0.59 and sign(x) = -1 => sign(x) * (abs(x) – floor(abs(x))) = fraction(x) = -0.59

---

## signed_power(x, y)

**sign(x) * (abs(x) ^ y)**
x raised to the power of y such that final result preserves sign of x. For power of 2, x ^ y will be a parabola but signed_power(x, y) will be odd and one-to-one function (unique value of x for certain value of signed_power(x, y)) unlike parabola.




**Example:**
If x = 3, y = 2 ⇒ abs(x) = 3 ⇒ abs(x) ^ y = 9 and sign(x) = +1 ⇒ sign(x) * (abs(x) ^ y) = signed_power(x, y) = 9
If x = -9, y = 0.5 ⇒ abs(x) = 9 ⇒ abs(x) ^ y = 3 and sign(x) = -1 ⇒ sign(x) * (abs(x) ^ y) = signed_power(x, y)

---

## nan_out (x, lower=0, upper=0)

**Example**:

```
nan_out (returns, lower=-0.1, upper=0.1)
```

Open example alpha in Simulate

Simulation Settings

Region Universe Language Decay Delay Truncation Neutralization Pasteurization NaN Handling Unit Handling

## power (x, y)

**Examples**:

```
power (returns, volume/adv20); power (returns, volume/adv20, precise=true)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## purify(x)

**Example**:

```
purify(returns)
```

Open example alpha in Simulate

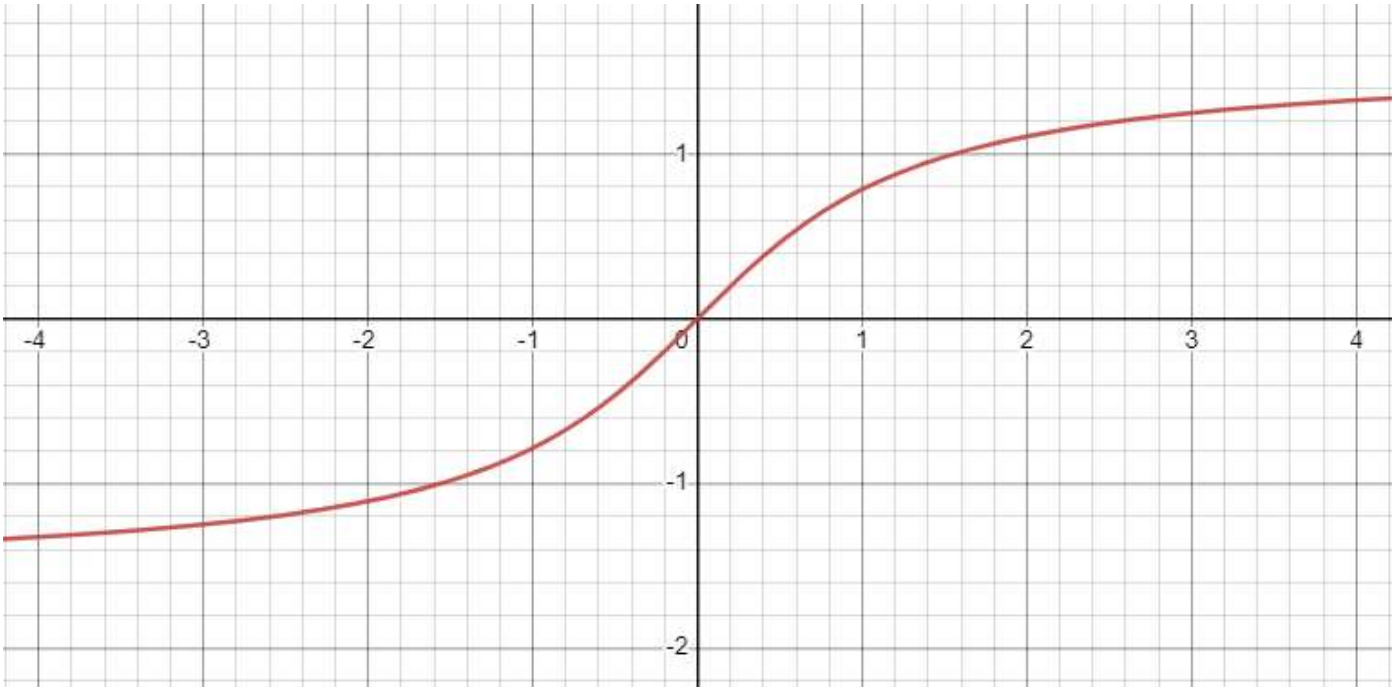
Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

# Transformational Operators

## arc_tan(x)

This operator does inverse tangent of input. Hence, constraints input within –pi/2 to pi/2 where ends. For very small values of x, arc_tan(x) = x. This is an odd and one-to-one function.




Example:
If x = 0 ⇒ arc_tan(x) = 0
If x = 4 ⇒ arc_tan(x) = 1.32
If x = -4 ⇒ arc_tan(x) = -1.32

---

## keep(x, f, period = 5)

This operator outputs value x when f changes and continues to do that for "period" days after f stopped changing. After "period" days since last change of f, NaN is output. This can be expressed as the below code:

D = days_from_last_change(f); u = trade_when(D < period, x, D > period); u

**Example:**

```
keep(sales / assets, sign(ts_delta(close, 252)), period = 63)
```

Open example alpha in Simulate

Simulation Settings

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 5 | Market | On | Off | Verify |

## clamp(x, lower = 0, upper = 0, inverse = False, mask = "")

This operator limits input value between lower and upper bound in inverse = false mode (which is default). Alternatively, when inverse = true, values between bounds are replaced with mask, while values outside bounds are left as is.

This can be expressed as the below code:

q = if_else(x < lower, lower, x); u = if_else(q > upper, upper, q);

v = if_else(x > lower && x < upper, mask, x);

if_else(inverse, v, u)

where mask is one of: 'nearest_bound', 'mean', 'NAN' or any floating point number.

**Example:**

clamp(-ts_returns(close, 5), lower = -0.05, upper = 0.05)

clamp(1 – ts_returns(close, 5), lower = .95, upper = 1.05, inverse = true, mask = 1.01)

```
1  clamp(-ts_returns(close, 5), lower = -0.05, upper = 0.05)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## filter

This operator is used to filter the value. It is calculated as follows:

Thus it can be divided into two parts: one that acts like weighted moving average (h) and recursive part (t). This allows to create different kinds of filters like decay linear or exponential decay.

**Example:**

```
filter(sales/assets, h="1,2,3,4",t="0.5,0.05.0.005")
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## bucket

Convert float values into indexes for user-specified buckets. Bucket is useful for creating group values, which can be passed to group operators as input.

If **buckets** are specified as "num_1, num_2, ..., num_N", it is converted into brackets consisting of [(num_1, num_2, idx_1), (num_2, num_3, idx_2), ..., (num_N-1, num_N, idx_N-1)]

Thus with buckets="2, 5, 6, 7, 10", the vector "-1, 3, 6, 8, 12" becomes "0, 1, 2, 4, 5"

If **range** if specified as "start, end, step", it is converted into brackets consisting of [(start, start+step, idx_1), (start+step, start+2*step, idx_2), ..., (start+N*step, end, idx_N)].

Thus with range="0.1, 1, 0.1", the vector "0.05, 0.5, 0.9" becomes "0, 4, 8"

Note that two hidden buckets corresponding to (-inf, start] and [end, +inf) are added by default. Use the **skipBegin**, **skipEnd** parameters to remove these buckets. Use **skipBoth** to set both **skipEnd** and **skipBegin** to true.

If you want to assign all NAN values into a separate group of their own, use **NANGroup**. The index value will be one after the last bucket

**Examples:**

my_group = bucket(rank(volume), buckets ="0.2,0.5,0.7", skipBoth=True, NANGroup=True);

group_neutralize(sales/assets, my_group)

---

## kth_element

Returns k-th value of input by looking through lookback days while ignoring space separated scalars in ignore list. This operator is also known as **backfill** operator as it can be used to backfill missing data.

**ignore** parameter is used to provide list of separated scalars to ignore from counting

**Example of backfill:**

```
kth_element(sales/assets,252,k="1",ignore="NAN 0")
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

---

## trade_when

This operator can be used in order to change Alpha values only under a specified condition and to hold Alpha values in other cases. It also allows to close Alpha positions (assign NaN values) under a specified condition:

Trade_When (x=triggerTradeExp, y=AlphaExp, z=triggerExitExp)

If triggerExitExp > 0, Alpha = nan;

Else if triggerTradeExp > 0, Alpha = AlphaExp;

else, Alpha = previousAlpha

price volume alphas.

**Examples:**

Trade_When (volume >= ts_sum(volume,5)/5, rank(-returns), -1)

If (volume >= ts_sum(volume,5)/5), Alpha = rank(-returns);

else trade previous Alpha;

exit condition is always false.

Trade_When (volume >= ts_sum(volume,5)/5, rank(-returns), abs(returns) > 0.1)

If abs(returns) > 0.1, Alpha = nan;

else if volume >= ts_sum(volume,5)/5, Alpha = rank(-returns);

else trade previous Alpha.

---

## left_tail

**Example:**

left_tail(rank(close), maximum = 0.02)

returns NaN, if rank(close)> 0.02; else returns rank(close).

---

## right_tail

**Example:**

right_tail(rank(volume), minimum = 0.5)

returns NaN, if rank(volume) < 0.5; else returns rank(volume).

---

## tail

**Example:**

## tanh(x)

Explanation: Hyperbolic tangent of x. Output will be between -1 and 1. For very small values of x, tanh(x) = x. This is an odd and one-to-one function.

**Example:**

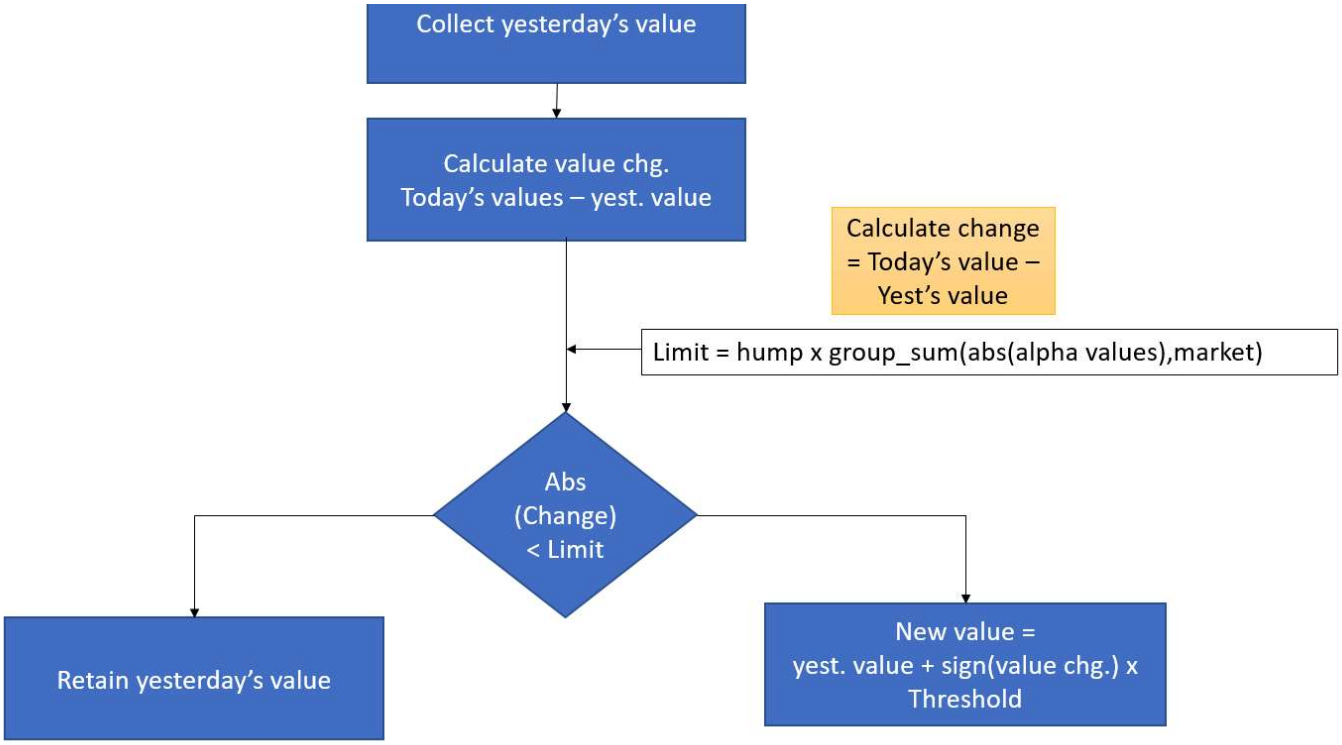
- If x = 0, tanh(x) = 0
- If x = 2, tanh(x) = 0.964
- If x = -2, tanh(x) = -0.964

# Time Series Operators

## hump(x, hump = 0.01)

This operator limits the frequency and magnitude of changes in the Alpha (thus reducing turnover). If today's values show only a minor change (not exceeding the Threshold) from yesterday's value, the output of the hump operator stays the same as yesterday. If the change is bigger than the limit, the output is yesterday's value plus the limit in the direction of the change.

Flowchart of the Hump operator:




```
hump(-ts_delta(close, 5), hump = 0.00001)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 5 | Market | On | Off | Verify |

## hump_decay

This operator helps to ignore the values that changed too little corresponding to previous ones. It can be used in two modes.

The **default** mode:

if relative == False:

if **abs**(x[t] - x[t-1])> p, return x[t], else x[t-1]

if **abs**(x[t] - x[t-1])> p * **abs**(x[t] + x[t-1]), return x[t], else x[t-1]

**Example:**

```
hump_decay(sales/assets, p=0.1, relative=True)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## jump_decay

If there is a huge jump in current data compare to previous one, apply force:

jump_decay(x) = abs(x-ts_delay(x, 1)) > sensitivity * ts_stddev(x,d) ? ts_delay(x,1) + ts_delta(x, 1) * force: x

If **stddev** enabled, jump threshold will be calculated as **sensitivity * stddev** otherwise it is **sensitivity**

**Example:**

```
jump_decay(sales/assets,252,stddev=True,sensitivity=0.5,force=0.1)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_regression(y, x, d, lag = 0, rettype = 0)

errors on this set assumes minimal value:

$$(x_i, y_i), \quad i \in 1 \dots d, \quad \Rightarrow \tilde{y}_i = \beta x_i + \alpha$$

$$\beta, \alpha = \arg\min \Sigma_i (y_i - \tilde{y}_i)^2 = \arg\min_{\alpha,\beta} \Sigma_i (y_i - \beta x_i - \alpha)^2.$$

Beta and Alpha in second line are OLS Linear Regression coefficients.

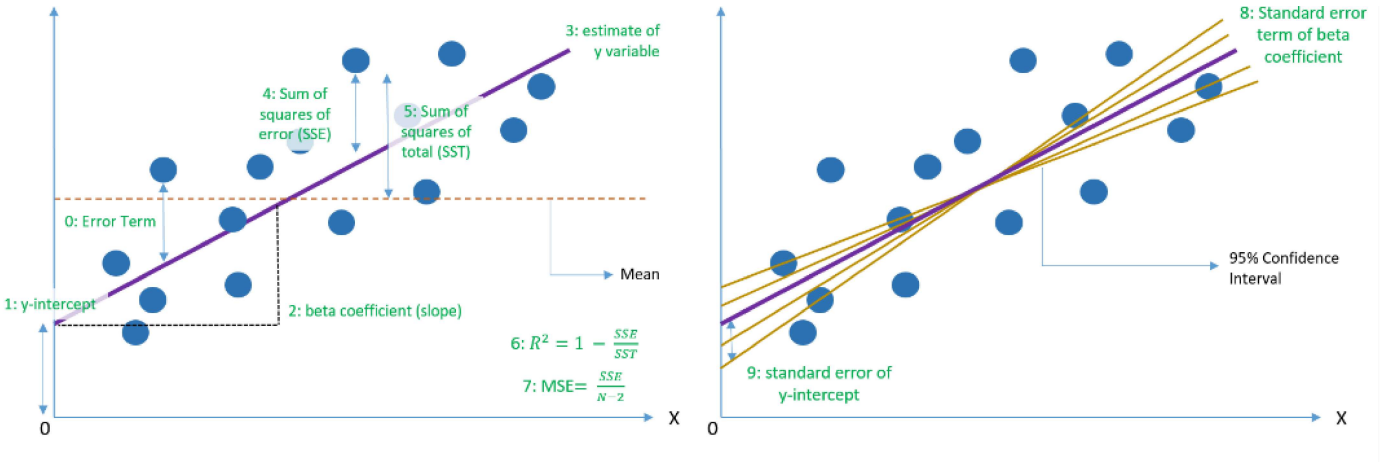
ts_regression operator returns various parameters related to said regression. This is governed by "rettype" keyword argument, which has a default value of 0. Other "rettype" argument values correspond to:

$$0: y_{di} - \tilde{y}_{di}, \quad 1: \alpha, \quad 2: \beta, \quad 3: \tilde{y}_{di},$$

$$4: \mathrm{SSE} = \Sigma_i (y_i - \beta x_i - \alpha)^2, \quad 5: \mathrm{SST} = \Sigma_i \left(y_i^2 - y_i \cdot n^{-1} \Sigma_j y_j\right), \quad 6: \mathrm{R}^2 = 1 - \frac{\mathrm{SSE}}{\mathrm{SST}}$$

$$7: \mathrm{s}^2 = \frac{\mathrm{SSE}}{\mathrm{N} - 2}, \quad 8: \mathrm{s}^2 \cdot \sqrt{n^{-1} + \frac{n^{-1}(n-1) \cdot \Sigma_j x_j}{\Sigma_i \left(x_i - n^{-1} \Sigma_j x_j\right)^2}}, \quad 9: \sqrt{\frac{n^{-1}(n-1) \cdot \mathrm{s}^2}{\Sigma_i \left(x_i - n^{-1} \Sigma_j x_j\right)^2}}$$

| rettype argument | return value |
|---|---|
| 0 | Error Term |
| 1 | y-intercept (α) |
| 2 | slope (β) |
| 3 | y-estimate |
| 4 | Sum of Squares of Error (SSE) |
| 5 | Sum of Squares of Total (SST) |
| 6 | R-Square |
| 7 | Mean Square Error (MSE) |
| 8 | Standard Error of β |
| 9 | Standard Error of α |




Here, "di" is current day index, "n"(may differ from d) is a number of valid (x, y) tuples used for calculation. All summations are over day index, using only valid tuples.

"lag" keyword argument may be optionally specified (default value is zero) to calculate lagged regression parameters instead:

$$\tilde{y}_i = \beta x_{i-lag} + \alpha$$

- Examples:
  - ts_regression(est_netprofit, est_netdebt, 252, lag = 0, rettype = 2)
    - Taking the data from the past 252 trading days (1 year), return the β coefficient from the equation when estimating the est_netprofit using the est_netdebt

```
ts_regression(ts_mean(volume, 2), ts_returns(close, 2), 252)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 5 | Market | On | Off | Verify |

## ts_co_skewness(y, x, d)

$$CoSkewness(Y,X) = \frac{\mathbb{E}[(Y - \mathbb{E}Y)(X - \mathbb{E}X)^2]}{\sigma_Y \sigma_X^2}$$

**Example:**

```
ts_co_skewness(volume, returns,20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_co_kurtosis(y, x, d)

Cokurtosis is the fourth standardized cross central moment. It shows how two time-series vectors change together.

$$CoKurtosis(Y,X) = \frac{\mathbb{E}[(Y - \mathbb{E}Y)(X - \mathbb{E}X)^3]}{\sigma_Y \sigma_X^3}$$

**Example:**

```
ts_co_kurtosis(vwap, volume,20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

Pearson correlation measures the linear relationship between two variables. It's most effective when the variables are normally distributed and the relationship is linear.

$$Correlation(x, y) = \frac{\sum_{i=t-d+1}^{t}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=t-d+1}^{t}(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

**Example:**

```
ts_corr(vwap, close, 20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_triple_corr(x, y, z, d)

$$TripleCorrelation(X, Y, Z) = \frac{\mathbb{E}[(X - \mathbb{E}X)(Y - \mathbb{E}Y)(Z - \mathbb{E}Z)]}{\sigma_X \sigma_Y \sigma_Z}$$

Triple correlation is the third standardized cross central moment of three time-series vectors.

**Example:**

```
ts_triple_corr(vwap,close,low, 20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|

Partial correlation is a measure of the correlation between two time-series vectors with removed effects of third time-series vector.

$$PartialCorrelation(X,Y,Z) = \frac{\rho_{XY} - \rho_{XZ}\rho_{YZ}}{\sqrt{1-(\rho_{XZ})^2}\sqrt{1-(\rho_{YZ})^2}}$$

where ρXY is correlation between x and y

For example, there are three vectors X, Y, Z (see figure below). Pearson correlation between X and Y greater than 0.9. However, we can see that X and Y vectors are in antiphase. Both vectors have strong positive correlation with vector Z. To remove effect of vector Z we can use partial correlation. PartialCorrelation(X, Y, Z) = -1

**Example:**

```
ts_partial_corr(vwap, close, volume, 20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_moment(x, d, k)

ts_moment return kth central moment of time-series vector X.

$$\mu_k = \mathbb{E}[(X - \mathbb{E}X)^k]$$

Setting k =2 gives variance of x over last d days; setting k = 3 gives 3rd central moment of x, etc.

**Example:**

```
ts_moment(returns, 60, k =3)
```

Simulation Settings

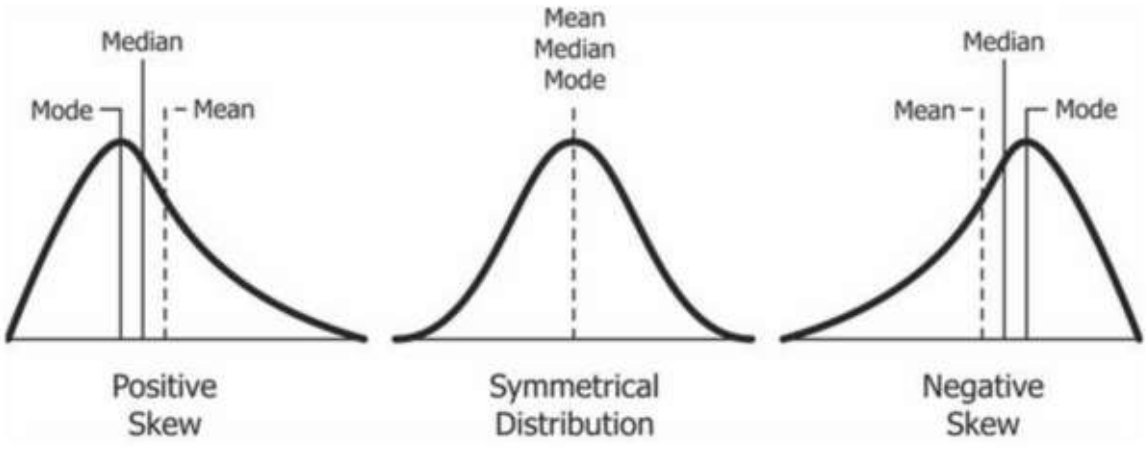
| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_skewness(x,d)

Skewness is the third central standardized moment of time-series vector X. Skewness could be treated as measure of the asymmetry of distribution of values in time-series vector X.

$$S(X) = \frac{\mathbb{E}[(X - \mathbb{E}X)^3}{(\mathbb{E}[(X - \mathbb{E}X)^2)^{3/2}}$$

If the skewness is greater than zero, the distribution is positively skewed; if it's less than zero, it's negatively skewed; and if equal to zero, it's symmetric. For interpretation and analysis, focus on downside risk. Negatively skewed distributions have what statisticians call a long left tail, which for investors can mean a greater chance of extremely negative outcomes. Positive skew would mean frequent small negative outcomes, and extremely bad scenarios are not as likely.




A nonsymmetrical or skewed distribution occurs when one side of the distribution does not mirror the other. Applied to investment returns, nonsymmetrical distributions are generally described as being either positively skewed (meaning frequent small losses and a few extreme gains) or negatively skewed (meaning frequent small gains and a few extreme losses).

For positive skew: Mean > Median > Mode

```
ts_skewness(returns, 60)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_kurtosis(x, d)

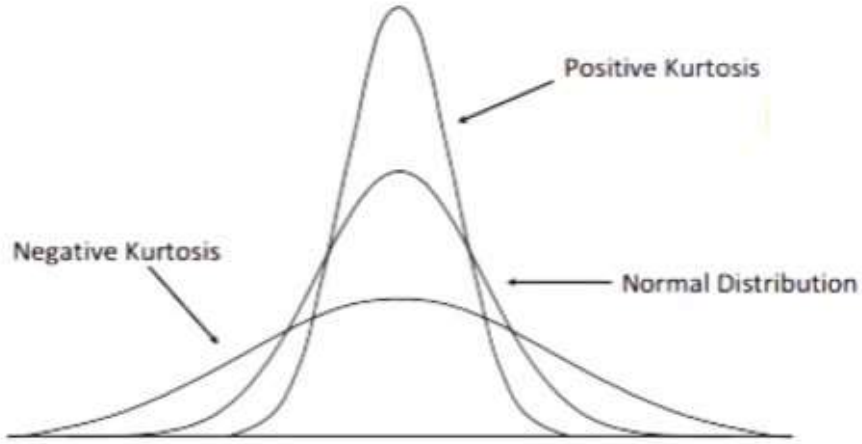
Kurtosis is the fourth central standardized moment of time-series vector X. Kurtosis could be treated as measure of how heavy or how thin tails of distribution of values in time-series vector X.

$$K(X) = \frac{\mathbb{E}[(X - \mathbb{E}X)^4}{(\mathbb{E}[(X - \mathbb{E}X)^2)^2} - 3$$

Kurtosis refers to the degree of peak in a distribution. More peak than normal (leptokurtic) means that a distribution also has fatter tails and that there are lower chances of extreme outcomes compared to a normal distribution.

The kurtosis formula measures the degree of peak. Kurtosis equals three for a normal distribution; excess kurtosis calculates and expresses kurtosis above or below 3. There are 3 types of kurtosis, categorized according to their degree of “peakedness”:

- Mesokurtic: A distribution with the same kurtosis as the normal distribution is called mesokurtic.
- Leptokurtic: Distributions with relatively large tails (that is, more peaked than normal) are called leptokurtic. If a return distribution has more returns clustered closely around the mean, it is leptokurtic.
- Platykurtic: Distributions with small tails (that is, less peaked than normal) are called platykurtic. If a return distribution has many returns with large deviations from the mean, it is platykurtic.




In the above graphs, K = excess kurtosis (defined below).

A kurtosis number (average deviations to the fourth power divided by the standard deviation to the fourth power) is evaluated in relation to the normal distribution, for which kurtosis = 3. Since excess kurtosis = kurtosis - 3, any positive number for excess kurtosis means the distribution is leptokurtic (meaning fatter tails and lower risk of extreme outcomes), and any negative number for excess kurtosis means the distribution is platykurtic.

**Example:**

```
ts_kurtosis(returns, 20)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_decay_exp_window(x, d, factor = 1.0, nan = True)

Exponentially decays time-series vector X over the past d days, where factor is the smoothing factor:

be assigned to older values than the newer ones.

If you decrease factor (bring it closer to 0) then the function would decay faster, assigning much less weight to older values. If you increase factor (bring it closer to 1) then the function would decay more slowly.

Parameter nan = True means that nan will be returned in case if values of X is nan. If you set parameter nan = False and all values of X is nan, the operator will return 0.

**Example:**

```
ts_decay_exp_window(close, 5, factor = 0.5)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_percentage(x, d, percentage=0.5)

Ts_percentage returns percentile of time-series X for the past d days.

For example, parameter percentile = 0.3 means that operator will return value of X that greater than 30% of values for the past d days.

Parameter percentile = 0.5 means that operator will return value of X that greater than 50% of values for the past d days, so it will be equal to median of X for the past d days.

**Example:**

```
ts_percentage(returns,20, percentage = 0.1)
```

Open example alpha in Simulate

Simulation Settings

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## ts_weighted_decay(x, k=0.5)

Instead of replacing today's value with yesterday's as in ts_delay(x,1), it assigns weighted average of today's and yesterday's values with weight on today's value being k and yesterday's being (1-k)

**Example:**
If today's value of x is 6, yesterday's is 4, then ts_weighted_decay(x,k=0.25) = 6*0.25+4*0.75 = 4.5

## ts_arg_max(x, d)

It returns the relative index of the max value in the time series for the past d days. If the current day has the max value for the past d days, it returns 0. If previous day has the max value for the past d days, it returns 1.

**Example:**
If d = 6 and values for past 6 days are [6,2,8,5,9,4] with first element being today's value then max value is 9 and it is present 4 days before today. Hence, ts_arg_max(x, d) = 4

## ts_av_diff(x, d)

This operator returns x – ts_mean(x, d), but it deals with NaNs carefully

**Example:**
If d = 6 and values for past 6 days are [6,2,8,5,9,NaN] then ts_mean(x,d) = 6 since NaN are ignored from mean computation. Hence, ts_av_diff(x,d) = 6-6 = 0

## ts_returns (x, d, mode = 1)

If mode = 1, it returns (x – ts_delay(x, d )) / ts_delay(x, d)

Then ts_returns(x,d,mode = 1) = relatively how much is 9 greater than 4 = (9-4)/4 = 1.25

Then ts_returns(x,d,mode = 2) = relatively how much is 9 greater than average of 4 and 9 = (9-4)/((9+4)/2)=10/13

---

## ts_scale(x, d, constant = 0)

This operator returns (x – ts_min(x, d)) / (ts_max(x, d) – ts_min(x, d)) + constant
This operator is similar to scale down operator but acts in time series space

**Example:**
If d = 6 and values for last 6 days are [6,2,8,5,9,4] with first element being today's value,
ts_min(x,d) = 2, ts_max(x,d) = 9
ts_scale(x,d,constant = 1) = 1 + (6-2)/(9-2) = 1.57

---

## ts_entropy(x,d, buckets=10)

For each instrument, we collect values of **input** in the past **d** days and calculate the probability distribution then the information entropy via a histogram as a result. You can control the number of buckets using 'buckets' parameter, the default value of which is 10. To know more about entropy, refer https://en.wikipedia.org/wiki/Entropy_(information_theory)

Hence, output for each instrument is: log(d) - 1/d * sum_over_buckets( H[i] * log(H[i]) ), where H is histogram with number of buckets equal to parameter 'buckets'.

---

## ts_rank_gmean_amean_diff(input1, input2, input3,...,d)

This operator returns **Geometric Mean** of ts_rank(input,d) of all input - Arithmetic Mean of ts_rank(input,d) of all input. This is similar to rank_gmean_amean_diff operator but in time-series space.

**Example:**

Geometric mean - Arithmetic mean = ts_rank_gmean_amean_diff(input1, input2, input3,...)

0.43-0.49 = -0.06

If any of the input has value NaN for a certain instrument, ts_rank_gmean_amean_diff(input1, input2, input3,...) = NaN

---

# Cross Sectional Operators

## rank(x, rate=2):

The Rank operator ranks the value of the input data x for the given stock among all instruments, and returns float numbers equally distributed between 0.0 and 1.0. When rate is set to 0, the sorting is done precisely. The default value of rate is 2.

**Example:**

Rank(close); Rank (close, rate=0) # Sorts precisely

X = (4,3,6,10,2) => Rank(x) = (0.5, 0.25, 0.75, 1, 0)

---

## rank_by_side (x, rate=2, scale=1)

The operator ranks the positive part and negative parts of input, separately; The ranking fashion is similar to the rank operator, however, the final output is scaled to the book size. The booksize can be mentioned as an optional parameter, **the default value of scale is 0 which essentially means no scaling**; When rate is set to 0, the sorting is done precisely; the default value of rate is set to 2.

**Example**:

```
rank_by_side (close, rate=1, scale=10)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## generalized_rank(open, m=1)

$$rank_i = \frac{\sum_{j=1}^{N} |x_i - x_j|^m \times sign(x_i - x_j)}{N}, \text{where } i \in 1..N, sign(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases}$$

In case when m=0 since any number raised to the zero power is 1, the result is the number of instruments with lesser value minus the number of instruments with bigger value divided by total count of instruments.

In case of m=1 the result is simply

$$x - \bar{x}$$

Note: It takes very long time when M != 0 or M != 1.

## regression_neut (Y, X)

The operator conducts the cross-sectional regression on the stocks with Y as target and X as the independent variable; Parameters 'a' and 'b' are calculated and then final output is computed for each of the stocks as Y-(a+b*X)

**Example**:

```
regression_neut (-returns, close)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

## regression_proj (Y, X)

independent variable, Parameters 'a' and 'b' are calculated and then final output is computed for each of the stocks as (a+ (b*X))

**Example**:

"-returns-regression_proj (-returns, ts_stddev(returns,20))" is equivalent to regression_neut (-returns, close)

---

## scale (x, scale=1, longscale=1, shortscale=1)

The operators scale the input to the booksize; We can optionally tune the booksize to scale by mentioning additional parameter "scale=booksize value"; We can also scale the long positions and short positions to separate scales by mentioning additional parameters to the operator: longscale= long booksize and shortscale=short booksize; default value of each leg of scale would be 0, which would mean no scaling, if any one of the leg is specified. Scale the alpha so that the sum of abs(x) over all instruments equals 1. To scale to a different book size, use Scale(x)*booksize.

Please check examples for the application of the same

**Examples**:

scale(returns, scale=4); scale (returns, scale= 1) + scale (close, scale=20); scale (returns, longscale=4, shortscale=3)

---

## scale_down(x,constant=0)

Scales all values in each day proportionately between 0 and 1 such that minimum value maps to 0 and maximum value maps to 1. constant is the offset by which final result is subtracted

**Example:**
If for a certain date, instrument values of certain input x is [15,7,0,20], max = 20 and min = 0
scale_down(x,constant=0) = [(15-0)/20,(7-0)/20,(0-0)/'20,(20-0)/20] = [0.75,0.35,0,1]
scale_down(x,constant=1) = [0.75-1,0.35-1,0-1,1-1] = [-0.25,-0.65,-1,0]

---

## truncate(x,maxPercent=0.01)

**Example:**
If for a certain date, instrument value of certain input x is [3,7,20,6] and say maxPercent = 0.5. Hence, one instrument should not have value more than 50% of sum of all values. Sum of values = 3 + 7 + 20 + 6 = 36. 50% of sum = 18. Hence, all values more than 18 should be capped to 18. Hence, output array = [3,7,18,6]

---

## vector_neut(x,y)

input1 neutralize to input2

For given vector A (i.e., input1) and B (i.e., input2), it finds a new vector A' (i.e., output) such that A' is orthogonal to B. It calculates projection of A onto B, and then subtracts projection vector from A to find the rejection vector (i.e., A') which is perpendicular to the B.

**Example**:

```
vector_neut(open,close)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |

---

## quantile(x, driver = gaussian, sigma = 1.0)

1. Rank the input raw Alpha vector
   The ranked Alpha value would be within [0, 1]
2. Shift the ranked Alpha vector
   For every Alpha value in the ranked Alpha vector, it is shifted as: Alpha_value = 1/N + Alpha_value * (1 - 2/N), here assume there are N instruments with value in the Alpha vector. The shifted Alpha value would be within [1/N, 1-1/N]
3. Apply distribution for each Alpha value in the ranked Alpha vector using the specified driver. Driver can be one of "gaussian", "uniform", "cauchy"

```
quantile(close, driver = gaussian, sigma = 0.5 )
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Market | On | Off | Verify |

## normalize(x, useStd = false, limit = 0.0)

This operator calculates the mean value of all valid alpha values for a certain date, then subtracts that mean from each element. If useStd= true, the operator calculates the standard deviation of the resulting values and divides each normalized element by it. If limit is not equal to 0.0, operator puts the limit of the resulting alpha values (between -limit to + limit).
Example:
If for a certain date, instrument value of certain input x is [3,5,6,2], mean = 4 and standard deviation = 1.82
normalize(x, useStd = false, limit = 0.0) = [3-4,5-4,6-4,2-4] = [-1,1,2,-2]
normalize(x, useStd = true, limit = 0.0) = [-1/1.82,1/1.82,2/1.82,-2/1.82] = [-0.55,0.55,1.1,-1.1]

## rank_gmean_amean_diff(input1, input2, input3,...)

This operator returns difference of geometric mean and arithmetic mean of cross sectional rank of inputs.

$$Geometric\ Mean = \left(\prod_{i=1}^{n} rank(input_i)\right)^{\frac{1}{n}} = \sqrt[n]{rank(input_1), rank(input_2), .., rank(input_n)}$$

$$Arithmetic\ Mean = \frac{1}{n}\sum_{i=1}^{n} rank(input_i) = \frac{rank(input_1) + rank(input_2) + .. + rank(input_n)}{n}$$

**Example:**
If for a certain date, 5 input ranks for a certain instrument is 0.6, 0.8, 0.2, 0.25, 0.6 then

If any of the input has value NaN for a certain instrument, rank_gmean_amean_diff(input1, input2, input3,...) = NaN

---

# Group Operators

## group_backfill(x, group, d, std = 4.0)

If a certain value for a certain date and instrument is NaN, from the set of same group instruments, calculate winsorized mean of all non-NaN values over last d days. Winsorized mean means inputs are truncated by std * stddev where stddev is the standard deviation of inputs.

**Example:**
If d = 4 and there are 3 instruments(i1, i2, i3) in a group whose values for past 4 days are x[i1] = [4,2,5,5], x[i2] = [7,NaN,2,9], x[i3] = [NaN,-4,2,NaN] where first element is most recent, then if we want to backfill x, we will only have to backfill x[i3]'s first element because every other instrument's first element is non-NaN.

The non-NaN values of other groups are [4,2,5,5,7,2,9,-4,2]. Mean = 3.56, Standard deviation is 3.71 and none of the item is outside the range of 3.56 – 4 * 3.71 and 3.56 + 4 * 3.71. Hence, we don't need to clip elements to those limits. Hence, Winsorized mean = backfilled value = 3.56.

For three instruments, group_backfill(x, group, d, std = 4.0) = [4,7,3.56]

---

## group_coalesce(original_group, group2, group3,...)

Operator returns first among inputs non-nan - that is, defined - value for each (date, instrument). Operator may be used to fill missing values in one grouping field with other grouping fields, while ensuring that groups produced by different fields do not overlap - even if field *values* do.

For example, group_coalesce(subindustry, industry, 1) would return value of subindustry field only if subindustry is known for given instrument on given date , otherwise it will return value of industry field if that is defined; Finally, it will return a special value if both subindustry and industry are undefined, distinct from any possible value coming from two previous fields.

Note: Internally, all grouping fields have integer values. You can build your own integer fields for use as groups in group operators! The bucket operator is a great tool which can be used to build

It could be that industry = "Oil" for "Oil" stocks, while subindustry = "Oil" for "Oil Services" (values are only examples).
Operator helps to distinguish the two groups if two fields are combined, as in:
subindustry_if_not_nan_else_industry = group_coalesce(subindustry, industry)
subindustry_if_not_nan_else_industry will be equal to 1 if it is known that stock is "Oil Services", but will be equal to 2 if subindustry is undefined for stock and we only know that industry is "Oil".

For some combination of fields, further examples:
group_coalesce(1, 2, 1) = 1
group_coalesce(nan, 2, 1) = 2
group_coalesce(nan, 1, 1) = 4 # second input being equal to 1 is not same group as first input being equal to 1
group_coalesce(1, 1, 1) = 1 # first input is valid for instrument so 1 is returned
group_coalesce(nan, nan, 1) = 5 # third input being equal to 1 is not same as any of previous cases

Output of operator is best utilized as "group" argument of group operator:
group_rank(ts_zscore(sales / assets, 252), group_coalesce(subindustry, industry))

---

## group_neutralize(x, group)

Neutralize alpha against groups. Difference between normalize and group_neutralize is in normalize, every element is subtracted by mean of all values of all instruments on that day whereas in group_neutralize, element is subtracted by mean of all values of the group of instruments that it belongs on that day.

**Example:**
If values of field x on a certain date for 10 instruments is [3,2,6,5,8,9,1,4,8,0] and first 5 instruments belong to one group, last 5 belong to other, then mean of first group = (3+2+6+5+8)/5 = 4.8 and mean of second group = (9+1+4+8+0)/5 = 4.4. Subtracting means from instruments of respective groups gives [3-4.8, 2-4.8, 6-4.8, 5-4.8, 8-4.8, 9-4.4, 1-4.4, 4-4.4, 8-4.4, 0-4.4] = [-1.8, -2.8, 1.2, 0.2, 3.2, 4.6, -3.4, -0.4, 3.6, -4.4]

---

## group_normalize(x, group, constantCheck=False, tolerance=0.01, scale=1)

constantCheck. If sanity check fails it assigns NaN to output

Currently two sanity checks are done
- First one is to check whether all numbers in a group is the same.
- Second one is to check all numbers in a group almost same (uses tolerance). The latter one checks constant-ness as an absolute difference. Hence, if (group max - group min) < abs(group min) * tolerance; then it is constant
- scale: output scaling factor
- output = input / absolute_sum * scale for each group

**Example:**
If for a certain date, values of x over certain 6 instruments are [4,-5,6,-8,1,0]. First 3 instruments are of group1 and last 3 of group2. Sum of absolute values of group1 and group2 are 15 and 9
group_normalize(x, group, constantCheck=False, tolerance=0.01, scale=2) = [2*4/15,- 2*5/15, 2*6/15,- 2*8/9, 2*1/9, 2*0/9]
group_normalize(x, group, constantCheck=True, tolerance=0.01, scale=2) = [NaN, NaN, NaN, NaN, NaN, NaN]
If x = [5,5,5,6,6.05,6.03] then sum of abs value of group1 and group2 are 15 and 18.08.
group_normalize(x, group, constantCheck=True, tolerance=0.01, scale=2) = [2*5/15, 2*5/15, 2*5/15, 2*6/18.08, 2*6.05/18.08, 2*6.03/18.08] because First sanity check is passed for group1 and second sanity check is passed for group2

---

# Logical Operators

## if_else(event_condition, Alpha_expression_1, Alpha_expression_2)

If the event condition provided is true, Alpha_expression_1 will be returned. If the event condition provided is false, Alpha_expression_2 will be returned.

**Example:**

We are interested in testing our hypothesis that if the stock price of a company has increased over the last 2 days, it may decrease in the future. Also, if the number of stocks bought and sold today is higher than the monthly average, then the reversion effect may be observed more profoundly.

We will implement this hypothesis by taking positions according to the difference of close price today and 3 days ago with alpha_2 using the ts_delta operator. When current volume is higher

```
1  event = volume > adv20;
2  alpha_1 = 2 * (-ts_delta(close, 3));
3  alpha_2 = (-ts_delta(close, 3));
4  if_else(event, alpha_1, alpha_2)
```

Open example alpha in Simulate

Simulation Settings

| Region | Universe | Language | Decay | Delay | Truncation | Neutralization | Pasteurization | NaN Handling | Unit Handling |
|---|---|---|---|---|---|---|---|---|---|
| USA | TOP3000 | Fast Expression | 3 | 1 | 0.01 | Industry | On | Off | Verify |