// This source code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/MPL/2.0/

// © TradingView

//@version=5

library("ZigZag", overlay = true)

// ZigZag Library

// v7, 2023.10.26

// This code was written using the recommendations from the Pine Script™ User Manual's Style Guide:

// https://www.tradingview.com/pine-script-docs/en/v5/writing/Style\_guide.html

//#region ———————————————————— Library types and functions

// @type Provides calculation and display properties to `ZigZag` objects.

// @field devThreshold The minimum percentage deviation from a point before the `ZigZag` changes direction.

// @field depth The number of bars required for pivot detection.

// @field lineColor The color of each line drawn by the `ZigZag`.

// @field extendLast A condition allowing a line to connect the most recent pivot with the current close.

// @field displayReversalPrice A condition to display the pivot price in the pivot label.

// @field displayCumulativeVolume A condition to display the cumulative volume for the pivot segment in the pivot label.

// @field displayReversalPriceChange A condition to display the change in price or percent from the previous pivot in each pivot label.

// @field differencePriceMode The reversal change display mode. Options are "Absolute" or "Percent".

// @field draw A condition to determine whether the `ZigZag` displays lines and labels.

// @field allowZigZagOnOneBar A condition to allow double pivots i.e., when a large bar makes both a pivot high and a pivot low.

export type Settings

float devThreshold = 5.0

int depth = 10

color lineColor = #2962FF

bool extendLast = true

bool displayReversalPrice = true

bool displayCumulativeVolume = true

bool displayReversalPriceChange = true

string differencePriceMode = "Absolute"

bool draw = true

bool allowZigZagOnOneBar = true

// @type Represents a significant level that indicates directional movement or potential support and resistance.

// @field ln A `line` object connecting the `start` and `end` chart points.

// @field lb A `label` object to display pivot values.

// @field isHigh A condition to determine whether the pivot is a pivot high.

// @field vol The cumulative volume for the pivot segment.

// @field start A `chart.point` object representing the coordinates of the previous point.

// @field end A `chart.point` object representing the coordinate of the current point.

export type Pivot

line ln

label lb

bool isHigh

float vol

chart.point start

chart.point end

// @type An object to maintain a Zig Zag's settings, pivots, and cumulative volume.

// @field settings A `Settings` object to provide calculation and display properties.

// @field pivots An array of `Pivot` objects.

// @field sumVol The volume sum for the current `Pivot` object's line segment.

// @field extend A `Pivot` object used to project a line from the last pivot point to the current bar.

export type ZigZag

Settings settings

array<Pivot> pivots

float sumVol = 0

Pivot extend = na

// @function Identifies a pivot point when the `src` has not reached beyond its value

// from `length` bars ago. Finds pivot highs when `isHigh` is `true`, and

// finds pivot lows otherwise.

// @param src (series float) The data series to calculate the pivot value from.

// @param length (series float) The length in bars required for pivot confirmation.

// @param isHigh (simple bool) Determines whether the pivot is a pivot high or pivot low.

// @returns (chart.point) A `chart.point` object when a pivot is found, `na` otherwise.

findPivotPoint(series float src, series float length, simple bool isHigh) =>

float pivotPrice = nz(src[length])

if length == 0

chart.point.new(time, bar\_index, pivotPrice)

else if length \* 2 <= bar\_index

bool isFound = true

for i = 0 to math.abs(length - 1)

if (isHigh and src[i] > pivotPrice) or (not isHigh and src[i] < pivotPrice)

isFound := false

break

for i = length + 1 to 2 \* length

if (isHigh and src[i] >= pivotPrice) or (not isHigh and src[i] <= pivotPrice)

isFound := false

break

if isFound

chart.point.new(time[length], bar\_index[length], pivotPrice)

// @function Calculates the deviation percentage between the `price` and the `basePrice`.

// @param basePrice (series float) The start price.

// @param price (series float) The end price.

// @returns (float) The signed deviation percentage.

calcDev(series float basePrice, series float price) =>

float result = 100 \* (price - basePrice) / math.abs(basePrice)

// @function Calculates the difference between the `start` and `end` point as a price or

// percentage difference and converts its value to a "string".

// @param start (series float) The start price.

// @param end (series float) The end price.

// @param settings (series Settings) A `Settings` object.

// @returns (string) A "string" representation of the difference between points.

priceRotationDiff(series float start, series float end, Settings settings) =>

float diff = end - start

string sign = math.sign(diff) > 0 ? "+" : ""

string diffStr = switch settings.differencePriceMode

"Absolute" => str.tostring(diff, format.mintick)

=> str.tostring(diff \* 100 / start, format.percent)

string result = str.format("({0}{1})", sign, diffStr)

// @function Creates a "string" containing the price, cumulative volume, and change in price

// for the pivot.

// @param start (series float) The start price.

// @param end (series float) The end price.

// @param vol (series float) The pivot's cumulative volume.

// @param settings (series Settings) A `Settings` object.

// @returns (string) A "string" to display in pivot labels.

priceRotationAggregate(series float start, series float end, series float vol, Settings settings) =>

string str = ""

if settings.displayReversalPrice

str += str.tostring(end, format.mintick) + " "

if settings.displayReversalPriceChange

str += priceRotationDiff(start, end, settings) + " "

if settings.displayCumulativeVolume

str += "\n" + str.tostring(vol, format.volume)

str

// @function Creates a label with coordinates from the `point` if the `settings` display

// properties allow it.

// @param isHigh (series bool) The condition to determine the label's color and location.

// @param point (series chart.point) A `chart.point` object.

// @param settings (series Settings) A `Settings` object.

// @returns (void) The function does not return a value.

makePivotLabel(series bool isHigh, chart.point point, Settings settings) =>

if settings.displayReversalPrice or settings.displayReversalPriceChange or settings.displayCumulativeVolume

[yloc, txtColor] = switch

isHigh => [yloc.abovebar, color.green]

=> [yloc.belowbar, color.red]

label.new(point, style = label.style\_none, xloc = xloc.bar\_time, yloc = yloc, textcolor = txtColor)

// @function Updates a `Pivot` object's properties, including its `end` point,

// cumulative volume, label text, and label and line drawing locations.

// Can be used as a function or method.

// @param this (series Pivot) The `Pivot` object to update.

// @param end (series chart.point) A new `chart.point` for the `end` field of the `Pivot`.

// @param vol (series float) The cumulative volume of the `Pivot`.

// @param settings (series Settings) A `Settings` object.

// @returns (void) The function does not return a value.

method updatePivot(Pivot this, chart.point end, float vol, Settings settings) =>

this.end := end

this.vol := vol

if not na(this.lb)

this.lb.set\_point(this.end)

this.lb.set\_text(priceRotationAggregate(this.start.price, this.end.price, this.vol, settings))

this.ln.set\_second\_point(this.end)

// @function Creates a new `Pivot` object, and assigns a line and label if the `draw` field

// of the `settings` allows it.

// @param start (series chart.point) A `chart.point` for the `start` of the `Pivot`.

// @param end (series chart.point) A `chart.point` for the `end` of the `Pivot`.

// @param vol (series float) The cumulative volume of the `Pivot`.

// @param isHigh (series bool) Specifies whether the `Pivot` represents a pivot high or pivot low.

// @param settings (series settings) A `Settings` object.

// @returns (Pivot) The new `Pivot` object.

newPivot(

series chart.point start, series chart.point end, series float vol, series bool isHigh, series Settings settings

) =>

Pivot p = Pivot.new(na, na, isHigh, vol, start, end)

if settings.draw

p.ln := line.new(start, end, xloc = xloc.bar\_time, color = settings.lineColor, width = 2)

p.lb := makePivotLabel(isHigh, end, settings)

p.updatePivot(end, vol, settings)

p

// @function Deletes the `line` and `label` objects assigned to the `ln` and `lb` fields in

// a `Pivot` object.

// Can be used as a function or method.

// @param this (series Pivot) The `Pivot` object to modify.

// @returns (void) The function does not return a value.

method delete(series Pivot this) =>

if not na(this.ln)

this.ln.delete()

if not na(this.lb)

this.lb.delete()

// @function Determines whether the `price` of the `point` reaches past the `price` of the

// `end` chart point of a `Pivot` object.

// Can be used as a function or method.

// @param this (series Pivot) A `Pivot` object.

// @param point (series chart.point) A `chart.point` object.

// @returns (bool) `true` if the `price` of the `point` reaches past that of the `end`

// in the `Pivot` object, `false` otherwise.

method isMorePrice(series Pivot this, series chart.point point) =>

int m = this.isHigh ? 1 : -1

bool result = point.price \* m > this.end.price \* m

// @function Returns the last `Pivot` object from a `ZigZag` instance if it contains at

// least one `Pivot`, and `na` otherwise.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @returns (Pivot) The last `Pivot` object in the `ZigZag`.

export method lastPivot(series ZigZag this) =>

int numberOfPivots = this.pivots.size()

Pivot result = numberOfPivots > 0 ? this.pivots.get(numberOfPivots - 1) : na

// @function Updates the fields of the last `Pivot` in a `ZigZag` object and sets the

// `sumVol` of the `ZigZag` to 0.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @param point (series chart.point) The `chart.point` for the `start` of the last `Pivot`.

// @returns (void) The function does not return a value.

method updateLastPivot(series ZigZag this, series chart.point point) =>

Pivot lastPivot = this.lastPivot()

if this.pivots.size() == 1

lastPivot.start := point

if this.settings.draw

lastPivot.ln.set\_first\_point(point)

lastPivot.updatePivot(point, lastPivot.vol + this.sumVol, this.settings)

this.sumVol := 0

// @function Pushes a new `Pivot` object into the `pivots` array of a `ZigZag` instance.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @param new (series Pivot) The new `Pivot` to add to the ZigZag.

// @returns (void) The function does not return a value.

method newPivotFound(series ZigZag this, series Pivot new) =>

this.pivots.push(new)

this.sumVol := 0

// @function Determines if a new pivot point is detected or if the properties of the

// last `Pivot` in the `ZigZag` need to be updated by comparing the `end` of the

// last `Pivot` to a new `point`. Updates the `ZigZag` and returns `true` if

// either condition occurs.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @param isHigh (series bool) Determines whether it checks for a pivot high or pivot low.

// @param point (chart.point) A `chart.point` to compare to the `end` of the last

// `Pivot` in the `ZigZag`.

// @returns (bool) `true` if it updates the last `Pivot` or adds a new `Pivot` to

// the `ZigZag`, `false` otherwise.

method newPivotPointFound(series ZigZag this, simple bool isHigh, series chart.point point) =>

bool result = false

Pivot lastPivot = this.lastPivot()

if not na(lastPivot)

if lastPivot.isHigh == isHigh

if lastPivot.isMorePrice(point)

this.updateLastPivot(point)

result := true

else

float dev = calcDev(lastPivot.end.price, point.price)

if (not lastPivot.isHigh and dev >= this.settings.devThreshold) or

(lastPivot.isHigh and dev <= -1 \* this.settings.devThreshold)

newPivotFound(this, newPivot(lastPivot.end, point, this.sumVol, isHigh, this.settings))

result := true

else

this.newPivotFound(newPivot(point, point, this.sumVol, isHigh, this.settings))

result := true

result

// @function Tries to find a new pivot point for the `ZigZag` instance. Updates the

// `ZigZag` and returns `true` when it registers a detected pivot.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @param src (series float) The data series to calculate the pivot value from.

// @param isHigh (simple bool) Determines whether it checks for a pivot high or pivot low.

// @param depth (series int) The number of bars to search for new pivots.

// @param registerPivot (series bool) A condition that determines whether or not to register a pivot.

// @returns (bool) `true` when a new pivot point is registered and the `ZigZag` is updated,

// `false` otherwise.

method tryFindPivot(

series ZigZag this, series float src, simple bool isHigh, series int depth, series bool registerPivot = true

) =>

chart.point point = findPivotPoint(src, depth, isHigh)

bool result = not na(point) and registerPivot ? this.newPivotPointFound(isHigh, point) : false

// @function Updates a `ZigZag` objects with new pivots, volume, lines, and labels.

// NOTE: This function must be called on every bar for accurate calculations.

// Can be used as a function or method.

// @param this (series ZigZag) A `ZigZag` object.

// @returns (bool) `true` when a new pivot point is registered and the `ZigZag` is updated,

// `false` otherwise.

export method update(series ZigZag this) =>

int depth = math.max(2, math.floor(this.settings.depth / 2))

this.sumVol += nz(volume[depth])

bool somethingChanged = this.tryFindPivot(high, true, depth)

somethingChanged := this.tryFindPivot(

low, false, depth, this.settings.allowZigZagOnOneBar or not somethingChanged

) or somethingChanged

Pivot lastPivot = this.lastPivot()

float remVol = math.sum(volume, math.max(depth, 1))

if this.settings.extendLast and barstate.islast and not na(lastPivot)

bool isHigh = not lastPivot.isHigh

float curSeries = isHigh ? high : low

chart.point end = chart.point.new(time, bar\_index, curSeries)

if na(this.extend) or somethingChanged

if not na(this.extend)

this.extend.delete()

this.extend := newPivot(lastPivot.end, end, this.sumVol, isHigh, this.settings)

this.extend.updatePivot(end, this.sumVol + remVol, this.settings)

somethingChanged

// @function Instantiates a new `ZigZag` object with optional `settings`.

// If no `settings` are provided, creates a `ZigZag` object with default settings.

// @param settings (series Settings) A `Settings` object.

// @returns (ZigZag) A new `ZigZag` instance.

export newInstance(series Settings settings = na) =>

ZigZag result = ZigZag.new(na(settings) ? Settings.new() : settings, array.new<Pivot>())

//#endregion

//#region ———————————————————— Example Code

// @variable The deviation percentage from the last local high or low required to form a new Zig Zag point.

float deviationInput = input.float(5.0, "Deviation (%)", minval = 0.00001, maxval = 100.0)

// @variable The number of bars in the pivot calculation.

int depthInput = input.int(10, "Depth", minval = 1)

// @variable The color of the Zig Zag's lines.

color lineColorInput = input.color(#2962FF, "Line Color")

// @variable If `true`, the Zig Zag will also display a line connecting the last known pivot to the current `close`.

bool extendInput = input.bool(true, "Extend to Last Bar")

// @variable If `true`, the pivot labels will display their price values.

bool showPriceInput = input.bool(true, "Display Reversal Price")

// @variable If `true`, each pivot label will display the volume accumulated since the previous pivot.

bool showVolInput = input.bool(true, "Display Cumulative Volume")

// @variable If `true`, each pivot label will display the change in price from the previous pivot.

bool showChgInput = input.bool(true, "Display Reversal Price Change", inline = "Price Rev")

// @variable Controls whether the labels show price changes as raw values or percentages when `showChgInput` is `true`.

string priceDiffInput = input.string("Absolute", "", options = ["Absolute", "Percent"], inline = "Price Rev")

// @variable A `Settings` instance for `ZigZag` creation.

var Settings settings =

Settings.new(

deviationInput, depthInput,

lineColorInput, extendInput,

showPriceInput, showVolInput,

showChgInput, priceDiffInput

)

// @variable A `ZigZag` object created using the `settings`.

var ZigZag zigZag = newInstance(settings)

// Update the `zigZag` on every bar.

zigZag.update()

//#endregion