**Overview**

The "Dual Polynomial Projection" script is designed to project price trends using two polynomial regressions with different degrees and periods. A polynomial regression fits a curve to a set of data points, helping to predict future values. This script plots two polynomial lines on a chart to show potential price trends and forecasts based on historical data.

**1. Indicator Declaration**

pinescript

Copy code

//@version=5

//@Yaseen Khalil

indicator("Dual Polynomial Projection", overlay=true, max\_lines\_count=500)

* **//@version=5**: Specifies the use of Pine Script version 5, which allows for more advanced features.
* **indicator("Dual Polynomial Projection", overlay=true, max\_lines\_count=500)**: This line defines the name of the indicator and sets overlay=true, meaning the lines will be drawn on the main price chart. The max\_lines\_count=500 parameter limits the number of lines drawn to avoid performance issues.

**2. Configurations**

pinescript

Copy code

periodLength1 = input.int(100, title="Period Length 1", minval=0)

forecastLength1 = input.int(10, title="Forecast Length 1", minval=0)

polyDegree1 = input.int(3, title="Polynomial Degree 1", minval=0, maxval=8)

periodLength2 = math.round(periodLength1 \* 0.95) // 5% less than periodLength1

forecastLength2 = forecastLength1

polyDegree2 = polyDegree1 + 1 // 1 degree greater than polyDegree1

priceSource = input(close)

lockForecast = input(false, title="Lock Projection")

* **periodLength1**: Defines the lookback period for the first polynomial regression. A longer period captures more historical data.
* **forecastLength1**: Sets the number of future bars to project using the polynomial regression.
* **polyDegree1**: Specifies the degree of the polynomial (how complex the curve is). A higher degree allows more curve flexibility but risks overfitting.
* **periodLength2**: Automatically set to 95% of periodLength1, making the second regression slightly shorter for comparison.
* **forecastLength2**: Set equal to forecastLength1, meaning both projections extend the same number of bars into the future.
* **polyDegree2**: One degree higher than polyDegree1 to create a slightly more complex second regression.
* **priceSource**: Specifies the data series used (usually the closing price).
* **lockForecast**: A toggle to lock the projection and stop updating it on new bars.

**3. Styling Options**

pinescript

Copy code

colorUp = input.color(#1a85ff, title="Uptrend Color", group="Visualization")

colorDown = input.color(#e600e6, title="Downtrend Color", group="Visualization")

colorForecast1 = input.color(#ffa500, title="Forecast Color 1", group="Visualization")

colorForecast2 = input.color(#ff5733, title="Forecast Color 2", group="Visualization") // Different color for second polynomial projection

lineThickness = input(1, title="Line Thickness", group="Visualization")

* **Purpose**: Defines colors for uptrends, downtrends, and forecasted lines to visually differentiate between trends and projections. lineThickness controls how thick the lines appear on the chart.

**4. Initialize Line Arrays**

pinescript

Copy code

var lineArray1 = array.new\_line(0)

var lineArray2 = array.new\_line(0)

if (barstate.isfirst)

for i = -forecastLength1 to periodLength1 - 1

array.push(lineArray1, line.new(na, na, na, na, style=line.style\_arrow\_left))

for i = -forecastLength2 to periodLength2 - 1

array.push(lineArray2, line.new(na, na, na, na, style=line.style\_dashed))

* **lineArray1 and lineArray2**: Create arrays to store lines drawn by the indicator.
* **if (barstate.isfirst)**: Runs only once at the beginning to initialize lines in the arrays, using line.new to create blank lines for each projection.

**5. Design Matrix & Partial System Solution**

pinescript

Copy code

timeIndex = bar\_index

var designMatrix1 = matrix.new<float>(0, 0)

var responseMatrix1 = matrix.new<float>(0, 0)

var designMatrix2 = matrix.new<float>(0, 0)

var responseMatrix2 = matrix.new<float>(0, 0)

if (barstate.isfirst)

for degreeIndex = 0 to polyDegree1

tempColumn = array.new\_float(0)

for lenIndex = 0 to periodLength1 - 1

array.push(tempColumn, math.pow(lenIndex, degreeIndex))

matrix.add\_col(designMatrix1, degreeIndex, tempColumn)

for degreeIndex = 0 to polyDegree2

tempColumn = array.new\_float(0)

for lenIndex = 0 to periodLength2 - 1

array.push(tempColumn, math.pow(lenIndex, degreeIndex))

matrix.add\_col(designMatrix2, degreeIndex, tempColumn)

var invDesign1 = matrix.inv(matrix.mult(matrix.transpose(designMatrix1), designMatrix1))

var designTransposed1 = matrix.mult(invDesign1, matrix.transpose(designMatrix1))

var invDesign2 = matrix.inv(matrix.mult(matrix.transpose(designMatrix2), designMatrix2))

var designTransposed2 = matrix.mult(invDesign2, matrix.transpose(designMatrix2))

* **designMatrix1 and designMatrix2**: These matrices are used to set up the polynomial regression equations. They contain values raised to powers that match the polynomial degree, which are then used to calculate the curve's coefficients.
* **matrix.inv and matrix.mult**: These functions handle matrix algebra to solve the regression equations. The process finds the best-fit polynomial coefficients by using matrix inversion and multiplication.

**6. Compute Polynomial Regressions**

pinescript

Copy code

var hasRun = 1

var matrix<float> coefMatrix1 = na

var matrix<float> coefMatrix2 = na

var iterX1 = -forecastLength1

var iterX2 = -forecastLength2

var float projValue1 = na

var float projValue2 = na

if (barstate.islast)

if (hasRun)

priceArray1 = array.new\_float(0)

for i = 0 to periodLength1 - 1

array.push(priceArray1, priceSource[i])

matrix.add\_col(responseMatrix1, 0, priceArray1)

coefMatrix1 := matrix.mult(designTransposed1, responseMatrix1)

priceArray2 = array.new\_float(0)

for i = 0 to periodLength2 - 1

array.push(priceArray2, priceSource[i])

matrix.add\_col(responseMatrix2, 0, priceArray2)

coefMatrix2 := matrix.mult(designTransposed2, responseMatrix2)

* **Purpose**: These lines calculate the polynomial coefficients by multiplying the transposed design matrix with the response matrix (price values). The coefficients describe the polynomial curve that best fits the price data.

**7. Plot the Polynomial Projections**

pinescript

Copy code

float previousY1 = na

float previousY2 = na

lineIndex1 = 0

lineIndex2 = 0

for i = -forecastLength1 to periodLength1 - 1

calcY1 = 0.

for j = 0 to polyDegree1

calcY1 += math.pow(i, j) \* matrix.get(coefMatrix1, j, 0)

if lineIndex1 == 0

projValue1 := calcY1

trendColor1 = calcY1 < previousY1 ? colorUp : colorDown

currLine1 = array.get(lineArray1, lineIndex1)

line.set\_xy1(currLine1, timeIndex - i + 1, previousY1)

line.set\_xy2(currLine1, timeIndex - i, calcY1)

line.set\_color(currLine1, i <= 0 ? colorForecast1 : trendColor1)

line.set\_width(currLine1, lineThickness)

previousY1 := calcY1

lineIndex1 += 1

* **Purpose**: This block plots the first polynomial projection on the chart. It calculates the y-values (calcY1) using the polynomial coefficients and draws lines between points. The line color changes based on whether the trend is moving up or down.

**8. Second Polynomial Projection**

pinescript

Copy code

for i = -forecastLength2 to periodLength2 - 1

calcY2 = 0.

for j = 0 to polyDegree2

calcY2 += math.pow(i, j) \* matrix.get(coefMatrix2, j, 0)

if lineIndex2 == 0

projValue2 := calcY2

trendColor2 = calcY2 < previousY2 ? colorUp : colorDown

currLine2 = array.get(lineArray2, lineIndex2)

line.set\_xy1(currLine2, timeIndex - i + 1, previousY2)

line.set\_xy2(currLine2, timeIndex - i, calcY2)

line.set\_color(currLine2, i <= 0 ? colorForecast2 : trendColor2)

line.set\_width(currLine2, 1)

previousY2 := calcY2

lineIndex2 += 1

* **Purpose**: This part plots the second polynomial projection in a similar manner but with slightly different settings (period length and degree). It provides an additional view of potential price movement using a more complex curve.