**PaySym: Technical Documentation**

**Table of Contents**

1. [Introduction](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#introduction)
2. [Application Overview](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#application-overview)
3. [Calculation Formulas](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#calculation-formulas)
4. [Scenario Management](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#scenario-management)
5. [Visualization and Analysis](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#visualization-and-analysis)
6. [Data Export Functionality](https://claude.ai/chat/c0420084-81aa-4059-b1bf-50f4190bc94a#data-export-functionality)

**Introduction**

The Advanced Payout Elasticity Simulator is a comprehensive tool designed for variable compensation specialists to model, analyze, and optimize sales incentive plans. This documentation provides a detailed explanation of the underlying calculations, formulas, and features to help compensation teams understand how different payout structures affect overall compensation costs, performance incentives, and financial risks.

**Application Overview**

The simulator models three primary components of variable compensation:

1. **Monthly Commissions**: Tiered commission structures with optional 3-month rolling averages
2. **Quarterly Bonuses**: Achievement-based payments tied to quarterly performance
3. **Continuity Bonuses**: Additional rewards for maintaining performance across consecutive quarters

The application provides four main analytical views:

* Results Dashboard: Key performance indicators and payout breakdown
* Scenario Comparison: Side-by-side analysis of different payout structures
* ROI Analysis: Return on investment metrics for compensation spending
* Risk Analysis: Financial risk assessment at different performance levels

**Calculation Formulas**

**Monthly Commission**

The commission formula evaluates each month's sales against tiered thresholds, with an option to use a 3-month rolling average:

function calculateCommission(monthlySales, fte, useRollingAverage, month, previousMonths, allMonthlySales) {

if (fte <= 0.7) return 0;

let salesValue = monthlySales;

// Apply 3-month rolling average if enabled

if (useRollingAverage) {

if (month === 0) {

// January: Average with previous Nov and Dec

salesValue = (monthlySales + previousMonths[0] + previousMonths[1]) / 3;

} else if (month === 1) {

// February: Average with previous Dec and current Jan

salesValue = (monthlySales + previousMonths[1] + allMonthlySales[0]) / 3;

} else {

// All other months: Average with previous 2 months

salesValue = (monthlySales + allMonthlySales[month-1] + allMonthlySales[month-2]) / 3;

}

}

// Calculate commission based on thresholds

let commission = 0;

for (let i = 0; i < thresholds.length; i++) {

const tier = thresholds[i];

if (salesValue > tier.threshold) {

const tierAmount = Math.min(salesValue, tier.upTo) - tier.threshold;

commission += tierAmount \* tier.percentage;

}

}

return commission;

}

**Key Points:**

* **3-Month Rolling Average**: When enabled, calculates each month's commission based on the average of current month plus previous two months, which stabilizes monthly commissions and reduces sales manipulation.
* **Tiered Commission Structure**: Applies different commission percentages to portions of sales that fall within each defined tier.
* **FTE Adjustment**: Full-time equivalent factor (0.7-1.0) affects commission calculations, representing partial time commitments.

**Quarterly Bonus**

Quarterly bonuses are calculated based on achievement percentage thresholds:

function calculateQuarterlyBonus(achievementPercentage, fte) {

// Find applicable bonus based on achievement percentage

let bonus = 0;

for (const tier of thresholds) {

if (achievementPercentage >= tier.threshold && achievementPercentage <= tier.upTo) {

bonus = tier.bonus;

break;

}

}

return bonus \* fte;

}

**Key Points:**

* **Achievement-Based**: Bonuses are tied to specific achievement percentage ranges.
* **Threshold Structure**: Defined tiers with fixed bonus amounts that increase with higher achievement levels.
* **FTE Adjustment**: Bonus is adjusted based on FTE value.

**Continuity Bonus**

Continuity bonuses reward sustained performance across quarters:

function calculateContinuityBonus(prevAchievement, currentAchievement, fte) {

const continuityThreshold = parseFloat(document.getElementById('continuityThreshold').value);

if (prevAchievement < continuityThreshold || currentAchievement < continuityThreshold) return 0;

// Find applicable bonus based on current achievement level

let bonus = 0;

for (const tier of thresholds) {

if (currentAchievement >= tier.threshold && currentAchievement <= tier.upTo) {

bonus = tier.bonus;

break;

}

}

return bonus \* fte;

}

**Key Points:**

* **Consecutive Achievement**: Requires two consecutive quarters above a defined threshold (typically 100%).
* **Current Quarter Level**: Bonus amount is determined by the current quarter's achievement level.
* **FTE Adjustment**: Bonus is adjusted based on FTE value.

**Total Payout Calculation**

The total payout combines all compensation components:

function calculateTotalPayout(quarterlies, monthlySales, fte) {

// Calculate quarterly bonuses

const quarterlyBonuses = quarterlies.map(q => calculateQuarterlyBonus(q, fte));

// Calculate continuity bonuses

const continuityBonuses = [0];

for (let i = 1; i < quarterlies.length; i++) {

continuityBonuses.push(calculateContinuityBonus(quarterlies[i-1], quarterlies[i], fte));

}

// Calculate monthly commissions

const commissions = monthlySales.map((m, index) =>

calculateCommission(m, fte, useRollingAverage, index, previousMonths, monthlySales));

// Total quarterly bonuses

const totalQuarterlyBonus = quarterlyBonuses.reduce((sum, q) => sum + q, 0);

// Total continuity bonuses

const totalContinuityBonus = continuityBonuses.reduce((sum, c) => sum + c, 0);

// Total commissions

const totalCommission = commissions.reduce((sum, c) => sum + c, 0);

// Grand total

const totalPayout = totalQuarterlyBonus + totalContinuityBonus + totalCommission;

// Average yearly achievement

const avgAchievement = quarterlies.reduce((sum, q) => sum + q, 0) / quarterlies.length;

// Calculate yearly revenue (sum of monthly sales)

const yearlyRevenue = monthlySales.reduce((sum, m) => sum + m, 0);

return {

quarterlyBonuses,

continuityBonuses,

commissions,

totalQuarterlyBonus,

totalContinuityBonus,

totalCommission,

totalPayout,

avgAchievement,

yearlyRevenue

};

}

**Key Points:**

* **Component Integration**: Combines monthly commissions, quarterly bonuses, and continuity bonuses.
* **First Quarter Exception**: No continuity bonus is possible for Q1 (represented by initial 0 value).
* **Return Object**: Returns detailed breakdown of all payout components for analysis.

**Elasticity Analysis**

Elasticity analysis simulates payout changes across achievement levels:

function simulateElasticity() {

const fte = parseFloat(document.getElementById('fte').value);

const results = [];

// Get base monthly sales

const baseMonthlySales = [];

for (let j = 1; j <= 12; j++) {

const salesInput = document.getElementById(`m${j}Sales`);

baseMonthlySales.push(parseFloat(salesInput.value));

}

// Simulate from 0% to 200% in 5% increments

for (let i = 0; i <= 40; i++) {

const achievement = i \* 5;

// Scale monthly sales by achievement percentage

const scaledMonthlySales = baseMonthlySales.map(sales => sales \* achievement / 100);

// Calculate quarterly bonus for this achievement level

const quarterlyBonus = calculateQuarterlyBonus(achievement, fte) \* 4; // For 4 quarters

// Calculate commission for the scaled sales

const commissionsForElasticity = scaledMonthlySales.map((m, index) =>

calculateCommission(m, fte, useRollingAverage, index, previousMonths, scaledMonthlySales));

const totalCommission = commissionsForElasticity.reduce((sum, c) => sum + c, 0);

// Total excluding continuity bonus

const totalExcludingContinuity = quarterlyBonus + totalCommission;

results.push({

achievement,

commission: totalCommission,

quarterlyBonus: quarterlyBonus,

totalExcludingContinuity

});

}

return results;

}

**Key Points:**

* **Achievement Range**: Simulates achievement levels from 0% to 200% in 5% increments.
* **Sales Scaling**: Proportionally scales monthly sales based on achievement percentage.
* **Continuity Exclusion**: Excludes continuity bonuses from elasticity analysis since they depend on sequential performance.
* **Uniform Achievement**: Assumes same achievement level across all four quarters for simulation purposes.

**ROI Analysis**

ROI analysis evaluates compensation efficiency at different achievement levels:

function simulateROI() {

const yearlyTarget = parseFloat(document.getElementById('yearlyTarget').value);

const fte = parseFloat(document.getElementById('fte').value);

const results = [];

// Simulate from 0% to 200% in 10% increments

for (let i = 0; i <= 20; i++) {

const achievement = i \* 10;

// Calculate revenue based on achievement

const revenue = yearlyTarget \* (achievement / 100);

// Calculate payout for this achievement level

const quarterlyBonus = calculateQuarterlyBonus(achievement, fte) \* 4; // For 4 quarters

// Scale monthly sales by achievement percentage

const scaledMonthlySales = baseMonthlySales.map(sales => sales \* achievement / 100);

// Calculate commission

const commissionsForElasticity = scaledMonthlySales.map((m, index) =>

calculateCommission(m, fte, useRollingAverage, index, previousMonths, scaledMonthlySales));

const totalCommission = commissionsForElasticity.reduce((sum, c) => sum + c, 0);

// Total excluding continuity bonus

const totalPayout = quarterlyBonus + totalCommission;

// Calculate ROI

const roi = totalPayout > 0 ? revenue / totalPayout : 0;

results.push({

achievement,

revenue,

payout: totalPayout,

roi

});

}

return results;

}

**Key Points:**

* **ROI Definition**: Calculated as revenue divided by total payout (revenue per euro spent on compensation).
* **10% Increments**: Uses larger increments than elasticity analysis for clearer trend visualization.
* **Linear Revenue Assumption**: Revenue is assumed to scale linearly with achievement percentage.

**Elasticity Per Range**

Detailed elasticity analysis by achievement ranges:

function calculateElasticityPerRange() {

const elasticityData = simulateElasticity();

const rangeResults = {};

// Define ranges

const ranges = [

{ name: '0to40', start: 0, end: 40 },

{ name: '41to70', start: 41, end: 70 },

{ name: '71to89', start: 71, end: 89 },

{ name: '90to99', start: 90, end: 99 },

{ name: '100to104', start: 100, end: 104 },

{ name: '105to114', start: 105, end: 114 },

{ name: '115to129', start: 115, end: 129 },

{ name: '130plus', start: 130, end: 200 }

];

ranges.forEach(range => {

// Find data points in this range

const pointsInRange = elasticityData.filter(d =>

d.achievement >= range.start && d.achievement <= range.end);

if (pointsInRange.length >= 2) {

const firstPoint = pointsInRange[0];

const lastPoint = pointsInRange[pointsInRange.length - 1];

const achievementDiff = lastPoint.achievement - firstPoint.achievement;

const payoutDiff = lastPoint.totalExcludingContinuity - firstPoint.totalExcludingContinuity;

const elasticity = achievementDiff > 0 ? payoutDiff / achievementDiff : 0;

// Calculate revenue impact per percentage point

const yearlyTarget = parseFloat(document.getElementById('yearlyTarget').value);

const revenuePerPoint = yearlyTarget / 100;

rangeResults[range.name] = {

elasticity: elasticity,

revenuePerPoint: revenuePerPoint,

roi: revenuePerPoint > 0 ? revenuePerPoint / elasticity : 0

};

} else {

rangeResults[range.name] = {

elasticity: 0,

revenuePerPoint: 0,

roi: 0

};

}

});

return rangeResults;

}

**Key Points:**

* **Range Definition**: Pre-defined achievement ranges that align with common payout thresholds.
* **Slope Calculation**: Calculates the payout increase per 1% achievement increase within each range.
* **End-to-End Comparison**: Measures from the first to last point in each range to smooth out small variations.
* **Revenue Per Point**: Assumes 1% of yearly target as the revenue impact per percentage point.

**Risk Assessment**

Financial risk analysis at different performance levels:

function generateRiskAssessment() {

const yearlyTarget = parseFloat(document.getElementById('yearlyTarget').value);

const fte = parseFloat(document.getElementById('fte').value);

// Calculate payouts at different risk levels

const lowRiskScenario = calculateTotalPayout([80, 80, 80, 80], Array(12).fill(20000 \* 0.8), fte);

const targetScenario = calculateTotalPayout([100, 100, 100, 100], Array(12).fill(20000), fte);

const highRiskScenario = calculateTotalPayout([150, 150, 150, 150], Array(12).fill(20000 \* 1.5), fte);

// Calculate revenue at different levels

const lowRiskRevenue = yearlyTarget \* 0.8;

const targetRevenue = yearlyTarget;

const highRiskRevenue = yearlyTarget \* 1.5;

// Calculate profit margins (assuming 30% margin)

const profitMargin = 0.3;

const lowRiskProfit = lowRiskRevenue \* profitMargin;

const targetProfit = targetRevenue \* profitMargin;

const highRiskProfit = highRiskRevenue \* profitMargin;

// Calculate compensation as percentage of profit

const lowRiskCompRatio = lowRiskScenario.totalPayout / lowRiskProfit \* 100;

const targetCompRatio = targetScenario.totalPayout / targetProfit \* 100;

const highRiskCompRatio = highRiskScenario.totalPayout / highRiskProfit \* 100;

let riskRating;

let recommendation;

if (highRiskCompRatio > 30) {

riskRating = "High";

recommendation = "The current compensation structure presents significant financial risk at high achievement levels. Consider capping bonuses or implementing a declining rate structure for achievements above 130%.";

} else if (targetCompRatio > 20) {

riskRating = "Medium";

recommendation = "The compensation structure is moderately risky at target achievement. Consider optimizing the threshold levels to better align with business margins.";

} else {

riskRating = "Low";

recommendation = "The compensation structure is well balanced with good alignment between performance and payout. The risk to company profitability is minimal even at high achievement levels.";

}

return {

lowRiskPayout: lowRiskScenario.totalPayout,

targetPayout: targetScenario.totalPayout,

highRiskPayout: highRiskScenario.totalPayout,

riskRating,

recommendation,

payoutPercentages: {

lowRisk: lowRiskCompRatio,

target: targetCompRatio,

highRisk: highRiskCompRatio

}

};

}

**Key Points:**

* **Standard Scenarios**: Evaluates three key risk points - 80% (low), 100% (target), and 150% (high) achievement.
* **Profit Margin Assumption**: Uses a fixed 30% profit margin calculation for risk assessment.
* **Risk Thresholds**: Classifies risk based on compensation-to-profit ratios:
  + High Risk: >30% of profit going to compensation at high achievement
  + Medium Risk: >20% of profit going to compensation at target achievement
  + Low Risk: <20% of profit going to compensation at target achievement

**Scenario Management**

The application supports saving and comparing different payout structures and performance profiles:

**Payout Structure**

A payout structure encompasses all configuration related to how compensation is calculated:

const structure = {

name,

useRollingAverage,

previousMonths,

commissionThresholds,

quarterlyThresholds,

continuityThreshold: parseFloat(document.getElementById('continuityThreshold').value),

continuityThresholds,

quarterlyWeights

};

**Performance Profile**

A performance profile captures achievement levels and monthly sales patterns:

const profile = {

name,

fte,

quarterlyAchievements,

monthlySales,

yearlyTarget

};

**Preset Models**

The application includes predefined payout structure presets:

1. **Conservative Model**:
   * Lower commission percentages (1.5%, 3%, 4.5%)
   * Higher thresholds (€15,000, €30,000, €45,000)
   * Lower bonus amounts
   * Uses 3-month rolling average for commissions
2. **Balanced Model**:
   * Medium commission percentages (2%, 4%, 6%)
   * Standard thresholds (€10,000, €25,000, €40,000)
   * Moderate bonus amounts
   * Uses 3-month rolling average for commissions
3. **Aggressive Model**:
   * Higher commission percentages (3%, 5%, 8%)
   * Lower thresholds (€8,000, €20,000, €35,000)
   * Higher bonus amounts
   * Monthly commission calculation (no rolling average)

**Performance Presets**

The application includes performance profile presets:

1. **Low Performer**: ~70-85% achievement
2. **Average Performer**: ~90-110% achievement
3. **Top Performer**: ~120-140% achievement

**Visualization and Analysis**

**Key Performance Indicators**

The application calculates and displays the following key metrics:

1. **Revenue vs Target**: Actual revenue as a percentage of yearly target
2. **Payout % of Revenue**: Total compensation as a percentage of total revenue
3. **Next Threshold**: Next achievement level that would trigger a higher bonus
4. **Year-End Projection**: Estimated total annual payout

**Charts and Visualizations**

The application provides several visual analyses:

1. **Annual Payout Composition**: Breakdown of commission, quarterly bonus, and continuity bonus
2. **Quarterly Performance Comparison**: Achievement vs. payout by quarter
3. **Monthly Payout Breakdown**: Monthly distribution of all compensation components
4. **Payout Elasticity**: Visualization of how compensation changes with achievement levels (0-200%)
5. **ROI Analysis**: Revenue, payout, and ROI ratio across achievement levels
6. **Risk Analysis**: Payout distribution at different performance scenarios

**Data Export Functionality**

The application supports exporting results in two formats:

**PDF Export**

Generates a structured PDF report containing:

* Summary data
* Quarterly breakdown table
* Monthly breakdown table
* Elasticity analysis table

**CSV Export**

Exports detailed data in CSV format including:

* Commission calculation method
* Summary metrics
* Quarterly breakdown
* Monthly breakdown
* Elasticity analysis by range

**Technical Notes for Implementation**

1. **FTE Limitations**: The application enforces a minimum FTE value of 0.7, as employees below this threshold typically do not qualify for the full variable compensation plan.
2. **Quarterly Weights**: The system supports weighted quarters (e.g., higher importance for Q4), with validation to ensure weights sum to 100%.
3. **Rolling Average Implementation**: The 3-month rolling average includes special handling for January (uses previous November and December) and February (uses previous December and current January).
4. **Browser Storage**: Saved scenarios are stored in the browser's localStorage, allowing persistence between sessions without server-side storage.
5. **Elasticity Slope Calculation**: Slopes are calculated between the first and last points in each range to provide a stable measurement of elasticity.
6. **Chart Rendering**: The application uses Chart.js for interactive visualizations with custom tooltip formatting for monetary values.
7. **Profit Margin Assumption**: The risk analysis assumes a standard 30% profit margin, which may need adjustment based on specific industry benchmarks.