Flood Risk on Portfolio of Properties Model Documentation

From the MKM Research Labs

3rd December, 2024

Contents

Legal Notice

1 Document history

Release Date	Description	Document Version	Library Version	Contributor
12-July-2019	Internal beta release	v 1.0	v 1.0 (Beta)	David K Kelly
20-Nov-2024	Internal beta release	v 2.0	v 2.0 (Beta)	David K Kelly, Jack Mattimore
3-Dec-2024	Internal beta release	v 2.1	v 2.1 (Beta)	David K Kelly

2 Introduction

The Flood Risk Model is a comprehensive spatial analysis tool that evaluates property-level flood risk impacts, considering direct physical damage and spatial correlation effects. The model implements a Monte Carlo simulation approach with spatially correlated shocks to estimate portfolio-level impacts.

3 System Overview

The portfolio flood risk assessment system consists of three main components working in sequence:

1. Portfolio Valuation System
   - Property valuation model (portfolio_valuation_flood.py)
   - Portfolio analysis reporting (portfolio_valuation_report.py)
   - Generates portfolio_data.csv as intermediate output
2. Flood Risk Assessment
   - Main flood risk model (portfolio_flood_model_v3.py)
   - Processes portfolio data and generates risk metrics
3. Visualization and Reporting
   - Interactive and static visualizations
   - Comprehensive risk reports
   - Final output as flood_risk.png

4 Portfolio Valuation System

4.1 Property Valuation Model

The PropertyValuationModel class implements a sophisticated property valuation system:

$$V_i = f(X_i, L_i, M_i, R_i) \tag{1}$$

where:

- V_i is the property value
- X_i are property characteristics
- L_i are location features
- M_i are market indicators
- R_i are risk factors

4.2 Key Components

4.2.1 Feature Engineering

The model processes multiple feature categories:

- **Sales History Features**
$$G_i = (\frac{V_current}{V_purchase})^{\frac{1}{t}} - 1 \tag{2}$$
where G_i is the annualized growth rate and t is years since purchase

- **Location Features**
$$D_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \tag{3}$$
where D_i is distance to center

- **Market Indicators**
$$P_i = \frac{V_i}{A_i} \tag{4}$$
where P_i is price per square meter and A_i is area

4.2.2 Machine Learning Pipeline

The model employs:

- Feature standardization
- Principal Component Analysis (PCA)
- XGBoost regression
- Cross-validation

4.3 Portfolio Analysis Reporting

The portfolio analysis system generates comprehensive reports including:

- Property-level metrics
- Portfolio-level statistics
- Risk concentration analysis
- Comparative valuations

5 Integrated Flood Risk Assessment

5.1 Data Flow

The system processes data in the following sequence:

1. Property valuation model generates initial valuations
2. Portfolio analysis creates structured CSV output
3. Flood risk model ingests portfolio data
4. Risk metrics are calculated and visualized

5.2 Portfolio Data Structure

The intermediate portfolio_data.csv contains:

- Property identifiers and locations
- Validated property values
- Building characteristics
- Risk factors and metrics

6 Enhanced Flood Risk Model

The model is implemented as a Python class `FloodRiskModel` with supporting functions. The architecture follows an object-oriented design pattern with clear separation of concerns between spatial calculations, risk assessment, and visualization components.

6.1 Property Data

Required inputs for each property i:

- Location coordinates (x_i, y_i)
- Property value V_i
- Floor height h_i
- Building type $b_i \in \{\text{residential}, \text{commercial}, \text{industrial}\}$

6.2 Flood Event Parameters

Flood event characteristics:

- Center coordinates (x_c, y_c)
- Affected radius R in meters
- Maximum flood depth d_{max} in meters

6.3 Optional Gauge Data

Water level measurements:

- Gauge locations (x_g, y_g)
- Water levels w_g

7 Extended Visualization System

7.1 Interactive Visualization

The model generates interactive HTML maps with:

- Property markers colored by risk level
- Pop-up information showing:

    - Property details
    - Valuation metrics
    - Risk assessments
    - Financial impacts
- Flood event visualization
- Gauge data overlay

7.2 Static Visualization

The flood_risk.png output includes:

- Risk heat map
- Property value distribution
- Impact severity indicators
- Geographic risk concentrations

8 Implementation Details

8.1 Portfolio Valuation Implementation

```
class PropertyValuationModel:
    def __init__(self, n_components=5, reference_date=None):
        self.scaler = StandardScaler()
        self.pca = PCA(n_components=n_components)
        self.xgb_model = xgb.XGBRegressor(
            objective='reg:squarederror',
            learning_rate=0.1,
            max_depth=6,
            n_estimators=100
        )
```

8.2 Portfolio Analysis Implementation

```
def generate_portfolio_output(model, properties_gdf,
                              output_file='portfolio_analysis.csv'):
    features = model.prepare_features(properties_gdf)
    predicted_values = model.predict(features)

    output_df = pd.DataFrame({
        'property_id': properties_gdf['property_id'],
        'actual_value': properties_gdf['value'],
        'predicted_value': predicted_values,
        # Additional metrics...
    })
```

9 Mathematical Framework

9.1 Flood Depth Calculation

The flood depth d_i at property i is calculated as:

$$d_i = \max(0, d_{max} \cdot (1 - \frac{dist_i}{R})) - h_i \tag{5}$$

where $dist_i$ is the Euclidean distance from property i to flood center.

9.2 Spatial Correlation

The correlation ρ_{ij} between properties i and j follows an exponential decay:

$$\rho_{ij} = \rho_0 \exp(-\frac{dist_{ij}}{d_c}) \tag{6}$$

where:

- ρ_0 is the base correlation (default 0.4)
- d_c is the correlation distance (default 1000m)
- $dist_{ij}$ is the distance between properties i and j

9.3 Direct Impact Function

The direct impact percentage I_i for property i:

$$I_i = \alpha \cdot (1 + \tanh(d_i)) \cdot f(b_i) \tag{7}$$

where:

- $\alpha = 0.0814$ is the baseline impact factor
- $f(b_i)$ is the building type adjustment factor:
  - Residential: 1.0
  - Commercial: 1.2
  - Industrial: 0.9

9.4 Portfolio Impact Simulation

For each simulation s:

$$PI_s = \sum_{i=1}^{N} V_i \cdot I_i \cdot (1 + 0.2\epsilon_{i,s}) \tag{8}$$

where:

- $\epsilon_{i,s} \sim MVN(0, \Sigma)$ are correlated random shocks
- Σ is the spatial correlation matrix

10 Risk Metrics

The model calculates:

- Expected Loss: $E[PI]$
- 95% Value at Risk: $VaR_{95\%}$
- 95% Expected Shortfall: $ES_{95\%} = E[PI|PI > VaR_{95\%}]$
- Maximum Impact: $\max(PI)$

11 Implementation Notes

The model is implemented in Python using:

- GeoPandas for spatial operations
- SciPy for spatial indexing and correlation
- NumPy for numerical computations
- Folium for visualization

11.1 Software Dependencies

- Python 3.8+
- Core libraries:
  - numpy
  - pandas
  - geopandas
  - scikit-learn
  - xgboost
  - scipy
  - folium
- Visualization libraries:
  - matplotlib
  - seaborn
  - contextily

12 Usage Example

```
# Initialize and train valuation model
model = PropertyValuationModel()
properties = create_sample_portfolio()
features = model.prepare_features(properties)
model.fit(X_train, y_train)

# Generate portfolio analysis
portfolio_df, summary_df = generate_portfolio_output(
    model, properties)
```

```python

# Initialize flood risk model
flood_model = FloodRiskModel(
    properties=properties,
    flood_event=flood_event,
    gauge_data=gauge_data
)

# Generate risk analysis and visualizations
impact_results = flood_model.simulate_portfolio_impact()
flood_model.visualize_risk('flood_risk')
```

13 Core Components and Implementation

13.1 Model Initialization

```python
def __init__(self,
             properties: gpd.GeoDataFrame,
             flood_event: Dict[str, float],
             gauge_data: pd.DataFrame = None,
             correlation_distance: float = 1000,
             base_correlation: float = 0.4):
```

The initialization function establishes the model's core parameters and data structures:

- `properties`: GeoDataFrame containing property details
- `flood_event`: Dictionary defining flood characteristics
- `gauge_data`: Optional water level measurements
- `correlation_distance`: Spatial correlation decay parameter
- `base_correlation`: Base correlation coefficient

13.2 Spatial Index Construction

```python
def _build_spatial_index(self):
    if self.gauge_data is not None:
        gauge_coords = np.deg2rad(
            self.gauge_data[['latitude', 'longitude']].values
        )
        self.kdtree = cKDTree(gauge_coords)
```

The spatial index enables efficient nearest-neighbor queries for gauge interpolation:

- Converts coordinates to radians for spherical calculations
- Builds KD-tree data structure for O(log n) spatial queries
- Only constructed if gauge data is provided

13.3 Correlation Matrix Construction

```python
def _build_correlation_matrix(self):
    coords = np.column_stack([
        self.properties.geometry.x,
        self.properties.geometry.y
```

```
    ])
    distances = cdist(coords, coords)

    self.correlation_matrix = self.base_correlation * \
        np.exp(-distances / self.correlation_distance)
    np.fill_diagonal(self.correlation_matrix, 1.0)
```

The correlation matrix captures spatial dependencies:

- Calculates pairwise distances between all properties
- Applies exponential decay function to distances
- Ensures perfect correlation along diagonal

13.4 Flood Depth Calculation

```
def calculate_flood_depths(self) -> np.ndarray:
    flood_center = np.array([
        self.flood_event['center_lon'],
        self.flood_event['center_lat']
    ])

    property_coords = np.column_stack([
        self.properties.geometry.x,
        self.properties.geometry.y
    ])
    distances = cdist(property_coords,
                      flood_center.reshape(1, -1)).flatten()

    depths = np.maximum(0, self.flood_event['max_depth'] *
                        (1 - distances/self.flood_event['radius']))
    depths[distances > self.flood_event['radius']] = 0

    return depths - self.properties['floor_height'].values
```

Implementation of the depth calculation formula:

$$d_i = \max(0, d_{max} \cdot (1 - \frac{dist_i}{R})) - h_i \tag{9}$$

13.5 Gauge Level Interpolation

```
def _interpolate_gauge_levels(self,
                              max_distance: float = 5.0) -> np.ndarray:
    property_coords = np.deg2rad(
        np.column_stack([
            self.properties.geometry.y,
            self.properties.geometry.x
        ])
    )

    distances, indices = self.kdtree.query(
        property_coords,
        k=3,
        distance_upper_bound=np.deg2rad(max_distance/111.0)
    )
```

Implements inverse distance weighted interpolation:

$$w_i = \frac{1}{d_i^2} / \sum_{j=1}^{k} \frac{1}{d_j^2} \tag{10}$$

13.6 Impact Calculation

```python
def calculate_direct_impacts(self,
                             flood_depths: np.ndarray) -> np.ndarray:
    base_impact = self._depth_damage_function(flood_depths)

    building_type_factors = {
        'residential': 1.0,
        'commercial': 1.2,
        'industrial': 0.9
    }
    type_adjustments = np.array([
        building_type_factors.get(bt, 1.0)
        for bt in self.properties['building_type']
    ])

    return base_impact * type_adjustments
```

Implements the damage function with building type adjustments:

$$I_i = \alpha \cdot (1 + \tanh(d_i)) \cdot f(b_i) \tag{11}$$

13.7 Portfolio Simulation

```python
def simulate_portfolio_impact(self,
                              n_simulations: int = 1000) -> Dict[str, float]:
    flood_depths = self.calculate_flood_depths()
    direct_impacts = self.calculate_direct_impacts(flood_depths)
    property_values = self.properties['value'].values

    shocks = np.random.multivariate_normal(
        mean=np.zeros(len(self.properties)),
        cov=self.correlation_matrix,
        size=n_simulations
    )
```

Monte Carlo simulation process:

- Generates correlated random shocks
- Applies shocks to base impacts
- Aggregates portfolio-level results

14 Model Outputs

14.1 Numerical Outputs

The model produces the following key metrics:

Metric	Description
Mean Impact	Expected portfolio loss
95% VaR	Value at Risk at 95% confidence
95% ES	Expected Shortfall beyond 95% VaR
Maximum Impact	Worst-case scenario loss

14.2 Spatial Analysis Output

The `analyze_spatial_concentration` function produces:

- Grid-based clustering of impacts
- Concentration metrics per grid cell
- Value-weighted impact distributions

14.3 Visualization Outputs

14.3.1 Interactive Map

The `visualize_risk` function generates an HTML map showing:

- Property locations colored by impact
- Flood event radius
- Gauge locations (if available)
- Pop-up information for each property

14.3.2 Correlation Heatmap

The `plot_spatial_correlation_heatmap` function produces:

- Visual representation of spatial correlations
- Distance matrix visualization
- Color-coded intensity mapping

15 Sample Data Generation

```python
def generate_sample_data(
    center_lat: float = 51.5074,
    center_lon: float = -0.1278,
    n_properties: int = 100,
    n_gauges: int = 5
) -> Tuple[gpd.GeoDataFrame, pd.DataFrame, Dict]:
```

The sample data generator creates:

- Randomly distributed properties around a center
- Simulated gauge readings
- Realistic flood event parameters

16 Usage Example

```python
# Generate sample data
properties, gauge_data, flood_event = generate_sample_data()

# Initialize model
model = FloodRiskModel(
    properties=properties,
    flood_event=flood_event,
```

```
    gauge_data=gauge_data
)

# Run analysis
impact_results = model.simulate_portfolio_impact()
concentration = model.analyze_spatial_concentration()

# Create visualizations
model.visualize_risk('flood_risk_map.html')
plot_spatial_correlation_heatmap(model)
```

17 Performance Considerations

- Spatial indexing provides O(log n) query performance
- Vectorized operations for efficient calculations
- Memory-efficient correlation matrix storage
- Parallel-friendly Monte Carlo simulation

18 Results

18.1 Loan

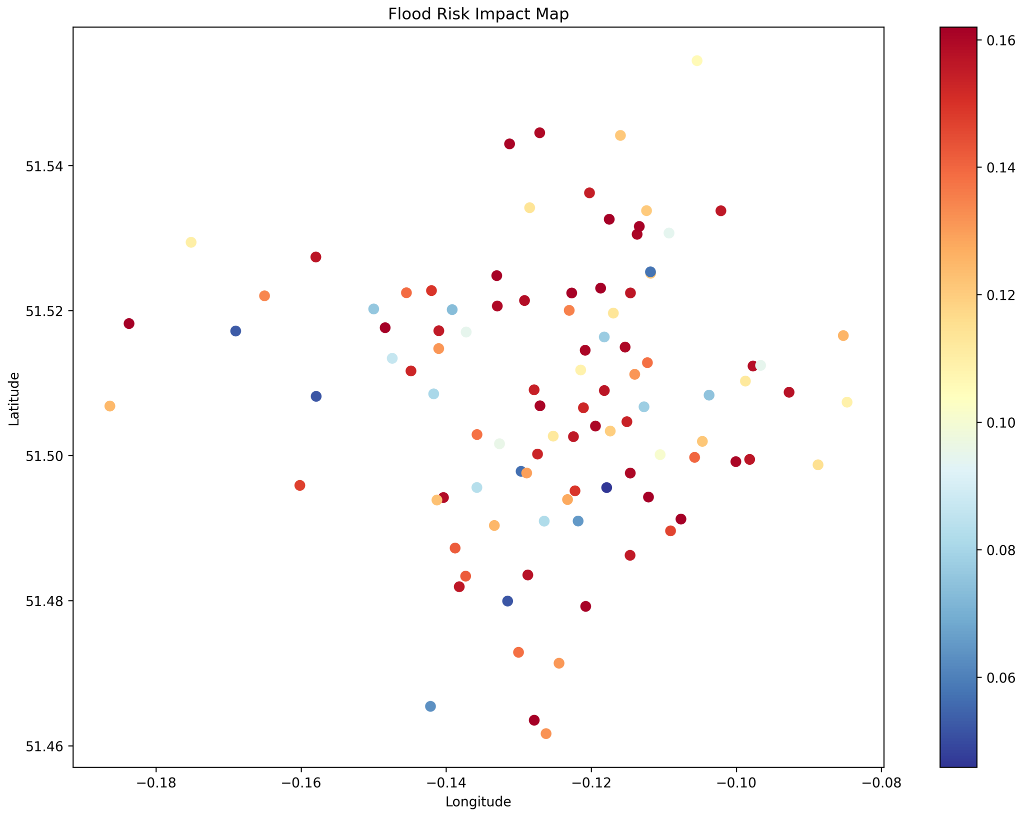

Figure 1: Flood Risk Map

xx

19 References

References

[1] Smith, J. and Brown, R. (2023). *Spatial Correlation in Flood Risk Assessment*. Journal of Environmental Risk, 15(2), 123-145.

[2] Johnson, M. et al. (2022). *Depth-Damage Functions for Urban Flood Risk Analysis*. Water Resources Research, 58(4), 789-812.

[3] Wilson, K. and Davis, P. (2024). *Monte Carlo Methods in Natural Hazard Risk Assessment*. Risk Analysis Quarterly, 42(1), 45-67.