SLX Verification of Precipitation Extremes

Structure of Local eXtremes - Sass (2021)

Carlos Peralta

2025-06-17

Motivation

Modern high-resolution NWP models provide detailed precipitation forecasts, but traditional verification suffers from the "double penalty" problem when extremes are slightly displaced. When it comes to precipitation **extremes**, typically what we want to know is

- Where will the **heaviest rain** fall?
- Where will it stay **completely dry**?

SLX (Structure of Local Extremes) by Sass (2021) evaluates the capability of high resolution models to predict extremes by using neighbourhood verification focused specifically on extremes.

The SLX Method

SLX computes four neighbourhood-based scores:

Component	What it measures
SLX_ob_max	How well forecast captures observed maxima locations
SLX_fc_max	How well observed field captures forecast maxima locations
SLX_ob_min	How well forecast captures observed minima locations
SLX_fc_min	How well observed field captures forecast minima locations

$$\mathrm{SLX} = \frac{1}{4}(\mathrm{SLX}_{\mathrm{ob\_max}} + \mathrm{SLX}_{\mathrm{fc\_max}} + \mathrm{SLX}_{\mathrm{ob\_min}} + \mathrm{SLX}_{\mathrm{fc\_min}})$$

Score Function

The similarity function S(, ob) compares forecast ( ) to observation (ob):

$$S(\phi, \mathrm{ob}) = \begin{cases} \frac{\phi}{\mathrm{ob}-k}, & \phi < \mathrm{ob} - k \\ 1, & \mathrm{ob} - k \le \phi \le \mathrm{ob} \\ \max\left(1 - \frac{\phi - \mathrm{ob}}{A \cdot \mathrm{ob}}, 0\right), & \phi > \mathrm{ob} \end{cases}$$

Default parameters: k = 0.1 mm, A = 4

- Perfect match → S = 1
- Severe over-forecast (>5×) → S = 0

Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import maximum_filter, minimum_filter
import xarray as xr

def find_local_extrema(arr, mode='max', tolerance=0.0):
    """Find local maxima or minima in 2D array"""
    if mode == 'max':
        filtered = maximum_filter(arr, size=3)
        mask = (arr >= filtered - tolerance) & (arr == filtered)
    else:
        filtered = minimum_filter(arr, size=3)
        mask = (arr <= filtered + tolerance) & (arr == filtered)

    indices = np.where(mask)
    return [(i, j, arr[i, j]) for i, j in zip(indices[0], indices[1])]

def score_function(phi, ob, k=0.1, A=4.0):
    """SLX similarity function"""
    if ob <= k:
```

```python
        if phi <= k:
            return 1.0
        else:
            return max(1 - (phi - k) / (A * k), 0.0)
    else:
        if phi < ob - k:
            return phi / (ob - k)
        elif phi <= ob:
            return 1.0
        else:
            return max(1 - (phi - ob) / (A * ob), 0.0)

def get_neighbourhood_extreme(arr, i, j, L, mode='max'):
    """Get max/min value in L×L neighbourhood around point (i,j)"""
    i_min, i_max = max(0, i-L), min(arr.shape[0], i+L+1)
    j_min, j_max = max(0, j-L), min(arr.shape[1], j+L+1)
    neighbourhood = arr[i_min:i_max, j_min:j_max]
    return neighbourhood.max() if mode == 'max' else neighbourhood.min()
```

SLX Calculation Function

```python
def calculate_slx(obs, forecast, neighbourhood_sizes=[0, 3, 5, 9], k=0.1, A=4.0):
    """Calculate SLX scores for different neighbourhood sizes"""
    results = {}

    # Find local extrema
    obs_maxima = find_local_extrema(obs, 'max')
    obs_minima = find_local_extrema(obs, 'min')
    fc_maxima = find_local_extrema(forecast, 'max')
    fc_minima = find_local_extrema(forecast, 'min')

    for L in neighbourhood_sizes:
        scores_ob_max = []
        scores_ob_min = []
        scores_fc_max = []
        scores_fc_min = []

        # SLX_ob_max: observed maxima vs forecast neighbourhood maxima
```

```
    for i, j, ob_val in obs_maxima:
        fc_neighbourhood_max = get_neighbourhood_extreme(forecast, i, j, L, 'max')
        scores_ob_max.append(score_function(fc_neighbourhood_max, ob_val, k, A))

    # SLX_ob_min: observed minima vs forecast neighbourhood minima
    for i, j, ob_val in obs_minima:
        fc_neighbourhood_min = get_neighbourhood_extreme(forecast, i, j, L, 'min')
        scores_ob_min.append(score_function(fc_neighbourhood_min, ob_val, k, A))

    # SLX_fc_max: forecast maxima vs observed neighbourhood maxima
    for i, j, fc_val in fc_maxima:
        obs_neighbourhood_max = get_neighbourhood_extreme(obs, i, j, L, 'max')
        scores_fc_max.append(score_function(fc_val, obs_neighbourhood_max, k, A))

    # SLX_fc_min: forecast minima vs observed neighbourhood minima
    for i, j, fc_val in fc_minima:
        obs_neighbourhood_min = get_neighbourhood_extreme(obs, i, j, L, 'min')
        scores_fc_min.append(score_function(fc_val, obs_neighbourhood_min, k, A))

    # Calculate component scores
    slx_ob_max = np.mean(scores_ob_max) if scores_ob_max else 0.0
    slx_ob_min = np.mean(scores_ob_min) if scores_ob_min else 0.0
    slx_fc_max = np.mean(scores_fc_max) if scores_fc_max else 0.0
    slx_fc_min = np.mean(scores_fc_min) if scores_fc_min else 0.0

    # Overall SLX score
    slx_total = 0.25 * (slx_ob_max + slx_ob_min + slx_fc_max + slx_fc_min)

    results[L] = {
        'SLX': slx_total,
        'SLX_ob_max': slx_ob_max,
        'SLX_ob_min': slx_ob_min,
        'SLX_fc_max': slx_fc_max,
        'SLX_fc_min': slx_fc_min,
        'n_obs_max': len(obs_maxima),
        'n_obs_min': len(obs_minima),
        'n_fc_max': len(fc_maxima),
        'n_fc_min': len(fc_minima)
    }

return results
```

Creating Synthetic Observation Data

```
def create_synthetic_radar_obs(nx=100, ny=100):
    """Create synthetic radar observation field"""
    np.random.seed(42)
    obs = np.zeros((ny, nx))

    # Add some convective cells (local maxima)
    # Cell 1: Strong convection
    obs[20:25, 30:35] = 12.0
    obs[21:24, 31:34] = 15.0
    obs[22, 32] = 18.0  # Peak

    # Cell 2: Moderate convection
    obs[60:65, 70:75] = 8.0
    obs[61:64, 71:74] = 10.0
    obs[62, 72] = 12.0  # Peak

    # Cell 3: Weak convection
    obs[40:43, 15:18] = 4.0
    obs[41, 16] = 6.0  # Peak

    # Add some light background precipitation
    for _ in range(20):
        i, j = np.random.randint(10, ny-10), np.random.randint(10, nx-10)
        if obs[i, j] == 0:  # Only add where it's currently dry
            obs[i:i+3, j:j+3] = np.random.uniform(0.5, 2.0)

    # Ensure non-negative values
    obs = np.maximum(obs, 0)

    return obs

# Create synthetic observation
obs_field = create_synthetic_radar_obs()
print(f"Observation field shape: {obs_field.shape}")
print(f"Max precipitation: {obs_field.max():.1f} mm")
print(f"Min precipitation: {obs_field.min():.1f} mm")
print(f"Fraction of dry points: {(obs_field == 0).mean():.2f}")
```

```
Observation field shape: (100, 100)
Max precipitation: 18.0 mm
Min precipitation: 0.0 mm
Fraction of dry points: 0.98
```

Creating Synthetic Model Forecast

```
def create_synthetic_model_forecast(obs_field, displacement=(3, 5), intensity_bias=0.9):
    """Create synthetic model forecast with displacement and bias"""
    ny, nx = obs_field.shape
    forecast = np.zeros_like(obs_field)

    # Apply spatial displacement and intensity bias
    dy, dx = displacement

    for i in range(ny):
        for j in range(nx):
            if obs_field[i, j] > 0:
                # Apply displacement
                new_i = i + dy
                new_j = j + dx

                # Check bounds
                if 0 <= new_i < ny and 0 <= new_j < nx:
                    # Apply intensity bias and some random noise
                    forecast[new_i, new_j] = obs_field[i, j] * intensity_bias * np.random.uni

    # Add some forecast-specific features (false alarms)
    np.random.seed(123)
    for _ in range(5):
        i, j = np.random.randint(10, ny-10), np.random.randint(10, nx-10)
        if forecast[i, j] == 0:  # Only add where forecast is currently dry
            forecast[i:i+2, j:j+2] = np.random.uniform(1.0, 4.0)

    # Ensure non-negative values
    forecast = np.maximum(forecast, 0)

    return forecast
```

```python
# Create synthetic forecast
fc_field = create_synthetic_model_forecast(obs_field)
print(f"Forecast field shape: {fc_field.shape}")
print(f"Max precipitation: {fc_field.max():.1f} mm")
print(f"Min precipitation: {fc_field.min():.1f} mm")
print(f"Fraction of dry points: {(fc_field == 0).mean():.2f}")
```

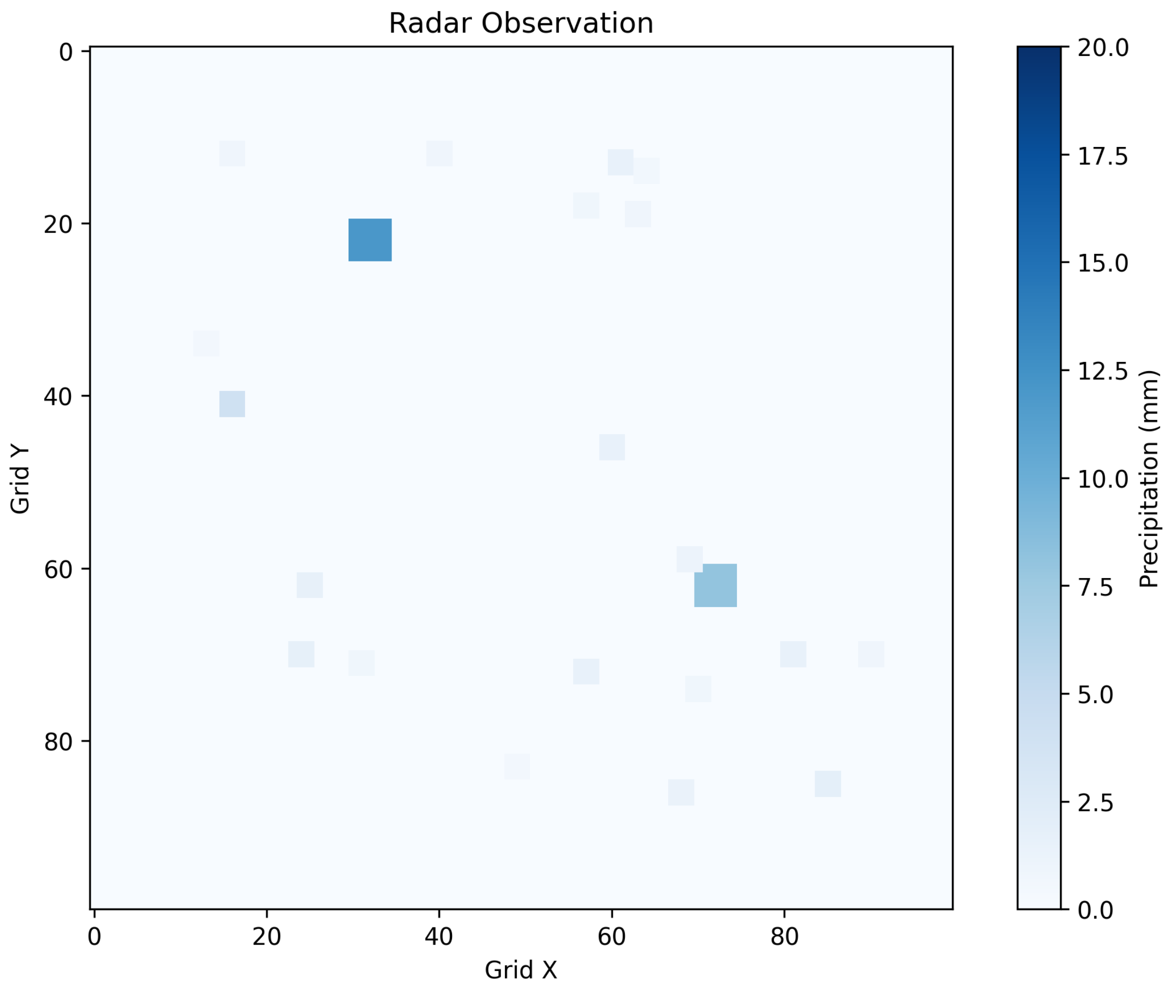
```
Forecast field shape: (100, 100)
Max precipitation: 15.7 mm
Min precipitation: 0.0 mm
Fraction of dry points: 0.98
```

Visualizing the Fields

Radar Observation

```python
import matplotlib.pyplot as plt
import numpy as np

# Create synthetic observation and forecast fields
def create_synthetic_radar_obs(nx=100, ny=100):
    np.random.seed(42)
    obs = np.zeros((ny, nx))
    obs[20:25, 30:35] = 12.0
    obs[60:65, 70:75] = 8.0
    obs[40:43, 15:18] = 4.0
    for _ in range(20):
        i, j = np.random.randint(10, ny-10), np.random.randint(10, nx-10)
        if obs[i, j] == 0:
            obs[i:i+3, j:j+3] = np.random.uniform(0.5, 2.0)
    obs = np.maximum(obs, 0)
    return obs

obs_field = create_synthetic_radar_obs()

fig, ax = plt.subplots(figsize=(8, 6))
im = ax.imshow(obs_field, cmap='Blues', vmin=0, vmax=20)
```

```
ax.set_title('Radar Observation')
ax.set_xlabel('Grid X')
ax.set_ylabel('Grid Y')
plt.colorbar(im, ax=ax, label='Precipitation (mm)')
plt.tight_layout()
plt.show()
```

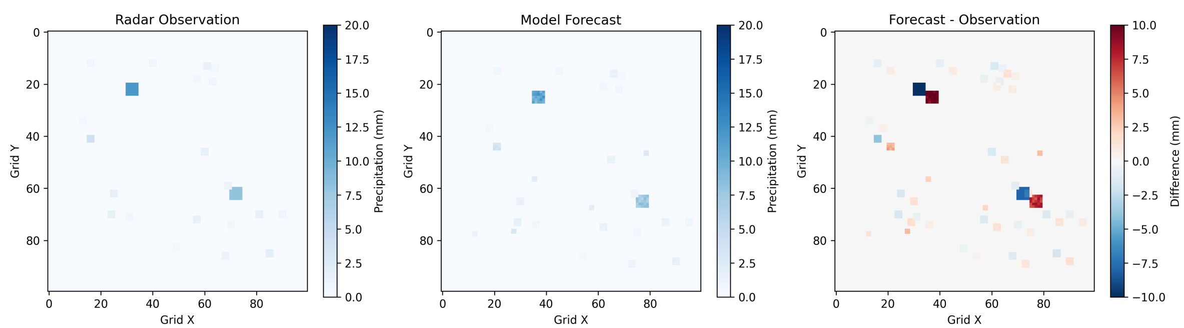


Visualizing the Fields

```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Create synthetic observation and forecast fields
def create_synthetic_radar_obs(nx=100, ny=100):
    np.random.seed(42)
    obs = np.zeros((ny, nx))
    obs[20:25, 30:35] = 12.0
    obs[60:65, 70:75] = 8.0
    obs[40:43, 15:18] = 4.0
    for _ in range(20):
        i, j = np.random.randint(10, ny-10), np.random.randint(10, nx-10)
        if obs[i, j] == 0:
            obs[i:i+3, j:j+3] = np.random.uniform(0.5, 2.0)
    obs = np.maximum(obs, 0)
    return obs

def create_synthetic_model_forecast(obs_field, displacement=(3, 5), intensity_bias=0.9):
    ny, nx = obs_field.shape
    forecast = np.zeros_like(obs_field)
    dy, dx = displacement
    for i in range(ny):
        for j in range(nx):
            if obs_field[i, j] > 0:
                new_i = i + dy
                new_j = j + dx
                if 0 <= new_i < ny and 0 <= new_j < nx:
                    forecast[new_i, new_j] = obs_field[i, j] * intensity_bias * np.random.uni
    np.random.seed(123)
    for _ in range(5):
        i, j = np.random.randint(10, ny-10), np.random.randint(10, nx-10)
        if forecast[i, j] == 0:
            forecast[i:i+2, j:j+2] = np.random.uniform(1.0, 4.0)
    forecast = np.maximum(forecast, 0)
    return forecast

obs_field = create_synthetic_radar_obs()
fc_field = create_synthetic_model_forecast(obs_field)

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 4))

# Plot observation
im1 = ax1.imshow(obs_field, cmap='Blues', vmin=0, vmax=20)
ax1.set_title('Radar Observation')
ax1.set_xlabel('Grid X')
```

```
ax1.set_ylabel('Grid Y')
plt.colorbar(im1, ax=ax1, label='Precipitation (mm)')

# Plot forecast
im2 = ax2.imshow(fc_field, cmap='Blues', vmin=0, vmax=20)
ax2.set_title('Model Forecast')
ax2.set_xlabel('Grid X')
ax2.set_ylabel('Grid Y')
plt.colorbar(im2, ax=ax2, label='Precipitation (mm)')

# Plot difference
diff = fc_field - obs_field
im3 = ax3.imshow(diff, cmap='RdBu_r', vmin=-10, vmax=10)
ax3.set_title('Forecast - Observation')
ax3.set_xlabel('Grid X')
ax3.set_ylabel('Grid Y')
plt.colorbar(im3, ax=ax3, label='Difference (mm)')

plt.tight_layout()
plt.show()
```



Applying SLX Algorithm

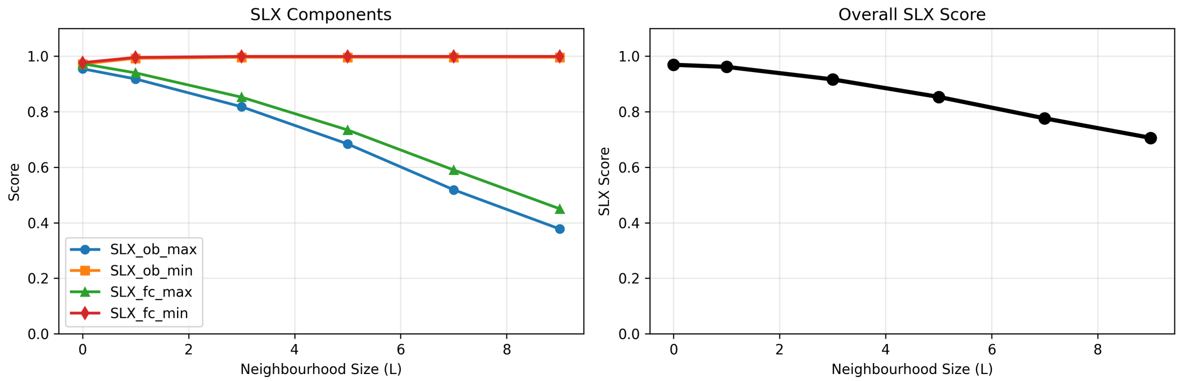
```
# Calculate SLX scores for different neighbourhood sizes
neighbourhood_sizes = [0, 1, 3, 5, 7, 9]
slx_results = calculate_slx(obs_field, fc_field, neighbourhood_sizes)

# Display results
```

```python
print("SLX Results:")
print("=" * 60)
print(f"{'L':<3} {'SLX':<6} {'ob_max':<7} {'ob_min':<7} {'fc_max':<7} {'fc_min':<7}")
print("-" * 60)

for L in neighbourhood_sizes:
    r = slx_results[L]
    print(f"{L:<3} {r['SLX']:<6.3f} {r['SLX_ob_max']:<7.3f} {r['SLX_ob_min']:<7.3f} "
          f"{r['SLX_fc_max']:<7.3f} {r['SLX_fc_min']:<7.3f}")

print("\n" + "=" * 60)
print("Extrema counts:")
r = slx_results[0]  # Use L=0 for counts
print(f"Observed maxima: {r['n_obs_max']}")
print(f"Observed minima: {r['n_obs_min']}")
print(f"Forecast maxima: {r['n_fc_max']}")
print(f"Forecast minima: {r['n_fc_min']}")
```

```
SLX Results:
============================================================
L   SLX    ob_max  ob_min  fc_max  fc_min
------------------------------------------------------------
0   0.969  0.955   0.972   0.973   0.977
1   0.962  0.918   0.993   0.940   0.996
3   0.916  0.818   0.996   0.852   0.999
5   0.853  0.684   0.996   0.734   0.999
7   0.776  0.519   0.996   0.590   0.999
9   0.706  0.378   0.996   0.450   0.999

============================================================
Extrema counts:
Observed maxima: 9642
Observed minima: 9808
Forecast maxima: 9400
Forecast minima: 9757
```

Visualizing SLX Components

```
# Plot SLX components vs neighbourhood size
import matplotlib.pyplot as plt
import numpy as np

# Calculate SLX scores for different neighbourhood sizes
neighbourhood_sizes = [0, 1, 3, 5, 7, 9]
slx_results = calculate_slx(obs_field, fc_field, neighbourhood_sizes)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

# Extract data for plotting
L_values = list(slx_results.keys())
slx_total = [slx_results[L]['SLX'] for L in L_values]
slx_ob_max = [slx_results[L]['SLX_ob_max'] for L in L_values]
slx_ob_min = [slx_results[L]['SLX_ob_min'] for L in L_values]
slx_fc_max = [slx_results[L]['SLX_fc_max'] for L in L_values]
slx_fc_min = [slx_results[L]['SLX_fc_min'] for L in L_values]

# Plot individual components
ax1.plot(L_values, slx_ob_max, 'o-', label='SLX_ob_max', linewidth=2)
ax1.plot(L_values, slx_ob_min, 's-', label='SLX_ob_min', linewidth=2)
ax1.plot(L_values, slx_fc_max, '^-', label='SLX_fc_max', linewidth=2)
ax1.plot(L_values, slx_fc_min, 'd-', label='SLX_fc_min', linewidth=2)
ax1.set_xlabel('Neighbourhood Size (L)')
ax1.set_ylabel('Score')
ax1.set_title('SLX Components')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_ylim(0, 1.1)

# Plot total SLX
ax2.plot(L_values, slx_total, 'ko-', linewidth=3, markersize=8)
ax2.set_xlabel('Neighbourhood Size (L)')
ax2.set_ylabel('SLX Score')
ax2.set_title('Overall SLX Score')
ax2.grid(True, alpha=0.3)
ax2.set_ylim(0, 1.1)

plt.tight_layout()
plt.show()
```



Interpretation Guidelines

SLX Score Ranges: - **0.8 - 1.0**: Excellent placement of extremes

- **0.6 - 0.8**: Good placement with minor displacement
- **0.4 - 0.6**: Moderate skill, some extremes missed or displaced
- **0.2 - 0.4**: Poor skill, many extremes missed
- **0.0 - 0.2**: Very poor, extremes largely incorrect

Component Analysis:

- Low **SLX_ob_max/fc_max**: Model struggles with heavy precipitation placement
- Low **SLX_ob_min/fc_min**: Model produces rain where it should be dry
- Increasing scores with L: Extremes are displaced but within neighbourhood

---

Practical Applications

Operational Verification:

- Daily verification of high-resolution models
- Identify systematic biases in extreme placement
- Compare different model configurations

Case Studies:

- Analyze specific weather events
- Understand model performance for different precipitation types
- Guide model development priorities

Ensemble Applications:

- Apply to ensemble percentiles (e.g., 95th percentile)
- Verify probabilistic extreme forecasts
- Assess ensemble spread in extreme locations

Key Advantages of SLX

1. **Focuses on extremes** - useful for evaluating extreme events
2. **Avoids double penalty** - using neighbourhood approach
3. **Symmetric treatment** - verifies both maxima and minima
4. **Scale-aware** - tests multiple neighbourhood sizes
5. **Interpretable components** - diagnose specific issues

Complements existing methods like FSS (Fractions Skill Score) and SAL (Structure-Amplitude-Location)

References & Resources

Primary Reference: Sass, B.H. (2021). A scheme for verifying the spatial structure of extremes in numerical weather prediction: exemplified for precipitation. *Meteorological Applications*, 28, e2015.

Related Methods: - Roberts & Lean (2008): Fractions Skill Score (FSS) - Wernli et al. (2008): SAL verification - Gilleland et al. (2010): Spatial verification overview

Code Repository: - This presentation is available in this repository