Verification of Precipitation Extremes with the SLX score

Structure of Local eXtremes - Sass (2021)

Carlos Peralta

2025-07-14

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

where the final score is defined as:

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

Algorithm Steps (Following Sass 2021)

Step 1: Extrema Detection

Local extremes are identified using a tolerance parameter δ (default \$ 0 kg/m^2):

- **obmax(K1)**: Observed local maximum points (M1 total)
- **obmin(K2)**: Observed local minimum points (M2 total)

- **fcmax(K3)**: Forecast local maximum points (M3 total)
- **fcmin(K4)**: Forecast local minimum points (M4 total)

Step 2: Neighbourhood Definition

For each extreme point, define a square neighbourhood of width L: - Neighbourhood size: (2L + 1)^2 grid points - L = 0 means point-to-point comparison - Internal points only (boundary zone of width Lmax excluded)

Step 3: Neighbourhood Extrema Calculation

For each observed/forecast extreme, find the corresponding extreme in the other field's neighbourhood:

- max(L,K1) = Max{ (i,j)} in forecast neighbourhood around obmax(K1)
- min(L,K2) = Min{ (i,j)} in forecast neighbourhood around obmin(K2)
- Ψmax(L,K3) = Max{Ψ(i,j)} in observed neighbourhood around fcmax(K3)
- Ψmin(L,K4) = Min{Ψ(i,j)} in observed neighbourhood around fcmin(K4)

Step 4: Score Function Application

Apply the SLX score function S(, ob) with parameters k = 0.1 kg/m² and A = 4:

If ob > k: - If < ob - k: S = /(ob - k) - If ob - k ob: S = 1 - If > ob: S = Max{1 - (- ob)/(A × ob), 0}

If ob k: - If k: S = 1 - If > k: S = Max{1 - (- k)/(A × k), 0}

Step 5: Component Score Calculation

Average individual scores for each component:

$$\text{SLX}_{\text{ob\_max}} = \frac{1}{M1} \sum_{K1=1}^{M1} S_{\text{ob\_max}}(K1)$$

$$\text{SLX}_{\text{ob\_min}} = \frac{1}{M2} \sum_{K2=1}^{M2} S_{\text{ob\_min}}(K2)$$

$$\text{SLX}_{\text{fc\_max}} = \frac{1}{M3} \sum_{K3=1}^{M3} S_{\text{fc\_max}}(K3)$$

$$\text{SLX}_{\text{fc\_min}} = \frac{1}{M4} \sum_{K4=1}^{M4} S_{\text{fc\_min}}(K4)$$

Python Implementation

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

def find_local_extrema_sass(arr, mode='max', tolerance=0.0):
    """
    Find local extrema exactly as described in Sass (2021)

    Key points from paper:
    - "Zero-valued dry areas will often exist... multiple points of zero value
```

```
      will be automatically selected as minima"
    - Default tolerance   = 0 kg/m²
    - All selected points contribute with equal weight
    """
    if mode == 'max':
        # Local maxima: points that are >= all neighbors within tolerance
        filtered = maximum_filter(arr, size=3)
        mask = (arr >= filtered - tolerance) & (arr == filtered)
    else:  # mode == 'min'
        # Local minima: points that are <= all neighbors within tolerance
        # Paper explicitly states zeros are automatically selected as minima
        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_sass(phi, ob, k=0.1, A=4.0):
    """
    Exact SLX similarity function from Sass (2021) equations (2a)-(2c), (3a)-(3b)
    """
    if ob > k:
        if phi < ob - k:
            return phi / (ob - k)  # Equation (2a)
        elif phi <= ob:
            return 1.0  # Equation (2b)
        else:  # phi > ob
            return max(1 - (phi - ob) / (A * ob), 0.0)  # Equation (2c)
    else:  # ob <= k
        if phi <= k:
            return 1.0  # Equation (3a)
        else:  # phi > k
            return max(1 - (phi - k) / (A * k), 0.0)  # Equation (3b)

def get_neighbourhood_extreme_sass(arr, i, j, L, mode='max'):
    """
    Get max/min value in (2L+1)×(2L+1) neighbourhood around point (i,j)
    Following Sass (2021) equations (1a)-(1d)
    """
    # Define neighbourhood bounds: [i-L, i+L] × [j-L, j+L]
    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()

def calculate_slx_sass(obs, forecast, neighbourhood_sizes=[0, 1, 3, 5, 9],
                       tolerance=0.0, k=0.1, A=4.0):
    """
    Calculate SLX scores following Sass (2021) methodology

    Parameters match paper specifications:
    - tolerance:  parameter (default  0 kg/m²)
    - k: dry threshold (default 0.1 kg/m²)
    - A: penalty parameter (default 4.0)
    """
    results = {}

    # Step 1: Find local extrema (following paper's approach)
    obs_maxima = find_local_extrema_sass(obs, 'max', tolerance)
    obs_minima = find_local_extrema_sass(obs, 'min', tolerance)
    fc_maxima = find_local_extrema_sass(forecast, 'max', tolerance)
    fc_minima = find_local_extrema_sass(forecast, 'min', tolerance)

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

        # Step 2-4: Calculate component scores following equations (4)-(7)

        # SLX_ob_max: Equation (4)
        for i, j, ob_val in obs_maxima:
            fc_neighbourhood_max = get_neighbourhood_extreme_sass(forecast, i, j, L, 'max')
            scores_ob_max.append(score_function_sass(fc_neighbourhood_max, ob_val, k, A))

        # SLX_ob_min: Equation (5)
        for i, j, ob_val in obs_minima:
            fc_neighbourhood_min = get_neighbourhood_extreme_sass(forecast, i, j, L, 'min')
            scores_ob_min.append(score_function_sass(fc_neighbourhood_min, ob_val, k, A))

        # SLX_fc_max: Equation (6)
        for i, j, fc_val in fc_maxima:
            obs_neighbourhood_max = get_neighbourhood_extreme_sass(obs, i, j, L, 'max')
```

```
            scores_fc_max.append(score_function_sass(fc_val, obs_neighbourhood_max, k, A))

        # SLX_fc_min: Equation (7)
        for i, j, fc_val in fc_minima:
            obs_neighbourhood_min = get_neighbourhood_extreme_sass(obs, i, j, L, 'min')
            scores_fc_min.append(score_function_sass(fc_val, obs_neighbourhood_min, k, A))

        # Step 5: Calculate component averages
        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: Equation (8)
        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 Test 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)
```

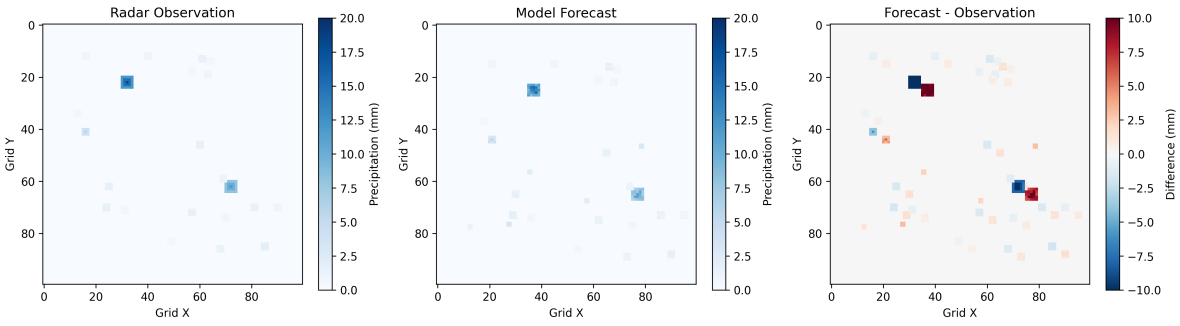
```
    # Cell 1: Strong convection
    obs[20:25, 30:35] = 12.0
    obs[21:24, 31:34] = 15.0
    obs[22, 32] = 18.0  # Peak

    # 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

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

# Create synthetic observation and forecast fields
obs_field = create_synthetic_radar_obs()
fc_field = create_synthetic_model_forecast(obs_field)

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}")

print(f"\nForecast 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}")
```

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

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

Visualizing the Fields

```
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 (Sass 2021 Method)

```
# Calculate SLX scores using the Sass (2021) methodology
neighbourhood_sizes = [0, 1, 3, 5, 7, 9]
slx_results = calculate_slx_sass(obs_field, fc_field, neighbourhood_sizes)

# Display results
print("SLX Results:")
print("=" * 70)
print(f"{'L':<3} {'SLX':<6} {'ob_max':<7} {'ob_min':<7} {'fc_max':<7} {'fc_min':<7}")
print("-" * 70)
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" + "=" * 70)
print("Extrema counts (following paper's detection method):")
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']}")

print("\nNote: High number of minima is expected per Sass (2021):")
print("'Zero-valued dry areas... multiple points of zero value will be")
print("automatically selected as minima for the verification process.'")
```

```
SLX Results:
======================================================================
L   SLX    ob_max  ob_min  fc_max  fc_min
----------------------------------------------------------------------
0   0.971  0.960   0.974   0.974   0.977
1   0.963  0.923   0.994   0.940   0.996
3   0.917  0.819   0.998   0.852   0.999
5   0.853  0.682   0.998   0.734   0.999
7   0.776  0.516   0.998   0.590   0.999
9   0.705  0.374   0.998   0.450   0.999

======================================================================
Extrema counts (following paper's detection method):
```

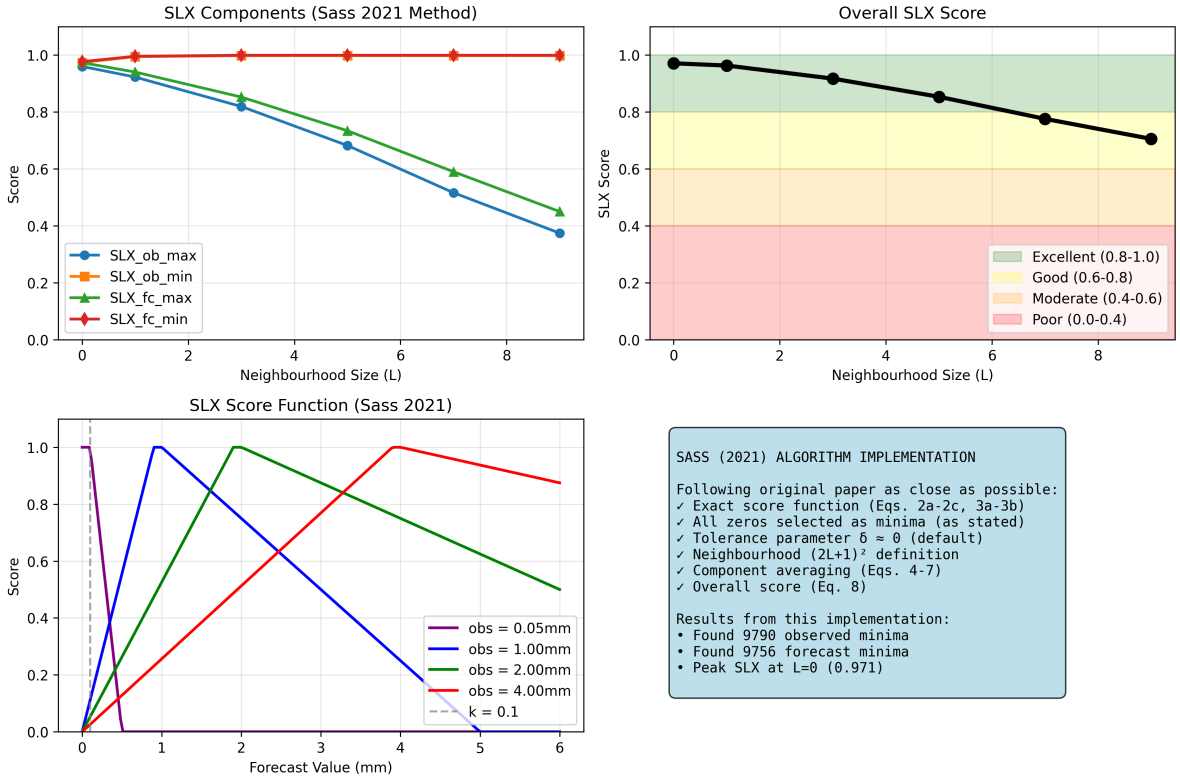
```
Observed maxima: 9587
Observed minima: 9790
Forecast maxima: 9395
Forecast minima: 9756

Note: High number of minima is expected per Sass (2021):
'Zero-valued dry areas... multiple points of zero value will be
automatically selected as minima for the verification process.'
```

Visualizing SLX Components

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# 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
axes[0,0].plot(L_values, slx_ob_max, 'o-', label='SLX_ob_max', linewidth=2)
axes[0,0].plot(L_values, slx_ob_min, 's-', label='SLX_ob_min', linewidth=2)
axes[0,0].plot(L_values, slx_fc_max, '^-', label='SLX_fc_max', linewidth=2)
axes[0,0].plot(L_values, slx_fc_min, 'd-', label='SLX_fc_min', linewidth=2)
axes[0,0].set_xlabel('Neighbourhood Size (L)')
axes[0,0].set_ylabel('Score')
axes[0,0].set_title('SLX Components (Sass 2021 Method)')
axes[0,0].legend()
axes[0,0].grid(True, alpha=0.3)
axes[0,0].set_ylim(0, 1.1)

# Plot total SLX
axes[0,1].plot(L_values, slx_total, 'ko-', linewidth=3, markersize=8)
axes[0,1].set_xlabel('Neighbourhood Size (L)')
axes[0,1].set_ylabel('SLX Score')
axes[0,1].set_title('Overall SLX Score')
```

```
axes[0,1].grid(True, alpha=0.3)
axes[0,1].set_ylim(0, 1.1)

# Add interpretation zones
axes[0,1].axhspan(0.8, 1.0, alpha=0.2, color='green', label='Excellent (0.8-1.0)')
axes[0,1].axhspan(0.6, 0.8, alpha=0.2, color='yellow', label='Good (0.6-0.8)')
axes[0,1].axhspan(0.4, 0.6, alpha=0.2, color='orange', label='Moderate (0.4-0.6)')
axes[0,1].axhspan(0.0, 0.4, alpha=0.2, color='red', label='Poor (0.0-0.4)')
axes[0,1].legend(loc='lower right')

# Show score function
phi_range = np.linspace(0, 6, 200)
ob_values = [0.05, 1.0, 2.0, 4.0]  # Include case where ob   k
colors = ['purple', 'blue', 'green', 'red']

for ob_val, color in zip(ob_values, colors):
    scores = [score_function_sass(phi, ob_val) for phi in phi_range]
    axes[1,0].plot(phi_range, scores, color=color, linewidth=2,
                   label=f'obs = {ob_val:.2f}mm')

axes[1,0].axvline(x=0.1, color='gray', linestyle='--', alpha=0.7, label='k = 0.1')
axes[1,0].set_xlabel('Forecast Value (mm)')
axes[1,0].set_ylabel('Score')
axes[1,0].set_title('SLX Score Function (Sass 2021)')
axes[1,0].legend()
axes[1,0].grid(True, alpha=0.3)
axes[1,0].set_ylim(0, 1.1)

# Show extrema detection comparison
axes[1,1].axis('off')
summary_text = f"""
SASS (2021) ALGORITHM IMPLEMENTATION

Following original paper as close as possible:
  Exact score function (Eqs. 2a-2c, 3a-3b)
  All zeros selected as minima (as stated)
  Tolerance parameter    0 (default)
  Neighbourhood (2L+1)² definition
  Component averaging (Eqs. 4-7)
  Overall score (Eq. 8)

Results from this implementation:
```

```
• Found {r['n_obs_min']} observed minima
• Found {r['n_fc_min']} forecast minima
• Peak SLX at L={L_values[np.argmax(slx_total)]} ({max(slx_total):.3f})
"""

axes[1,1].text(0.05, 0.95, summary_text, transform=axes[1,1].transAxes,
               fontsize=10, verticalalignment='top', fontfamily='monospace',
               bbox=dict(boxstyle="round,pad=0.5", facecolor='lightblue', alpha=0.8))

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



Score Function Properties

- **Perfect match** $\rightarrow S = 1$
- **Severe over-forecast** $(>5\times) \rightarrow S = 0$

- **Asymmetric**: Designed to avoid under-forecasting of warning conditions
- **Piecewise linear**: Simple but effective for operational use
- **Uncertainty aware**: k parameter accounts for observation uncertainty

Advantages of SLX

- **Extreme-focused**: Specifically designed for precipitation extremes
- **Neighborhood-based**: Addresses double penalty problem
- **Comprehensive**: Evaluates both maxima and minima
- **Scale-aware**: Tests multiple neighborhood sizes
- **Operationally practical**: Fast computation, daily output
- **Complements existing methods** like FSS (Fractions Skill Score) and SAL (Structure-Amplitude-Location)

Limitations of SLX

- **Parameter sensitivity**: Results depend on k, A, δ, and L choices
- **Score function complexity**: Piecewise linear function may need refinement
- **Extreme definition**: Tolerance parameter affects extreme selection
- **Computational scaling**: May need optimization for very large domains

---

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