Verification of Precipitation Extremes with the SLX score

Validation example

Carlos Peralta

2025-07-22

Validation test case

Section 4.3 of Sass (2021) discusses how the SLX (Structure, Location, and eXtreme) responds to different types of forecast errors: displacement, amplitude, and structure errors, using synthetic fields.

Key points:

- SLX is sensitive to spatial structure and location of extremes.
- Synthetic fields are used to illustrate how the score responds to controlled errors.
- The score should be lowest (best) when the forecast matches the analysis and degrade as errors are introduced.

Example implementation in python

The following python code attempts to reproduce the fields in Fig 7 an obtain the degradation shown in Figure as the analysis and forecast get farther from each other.

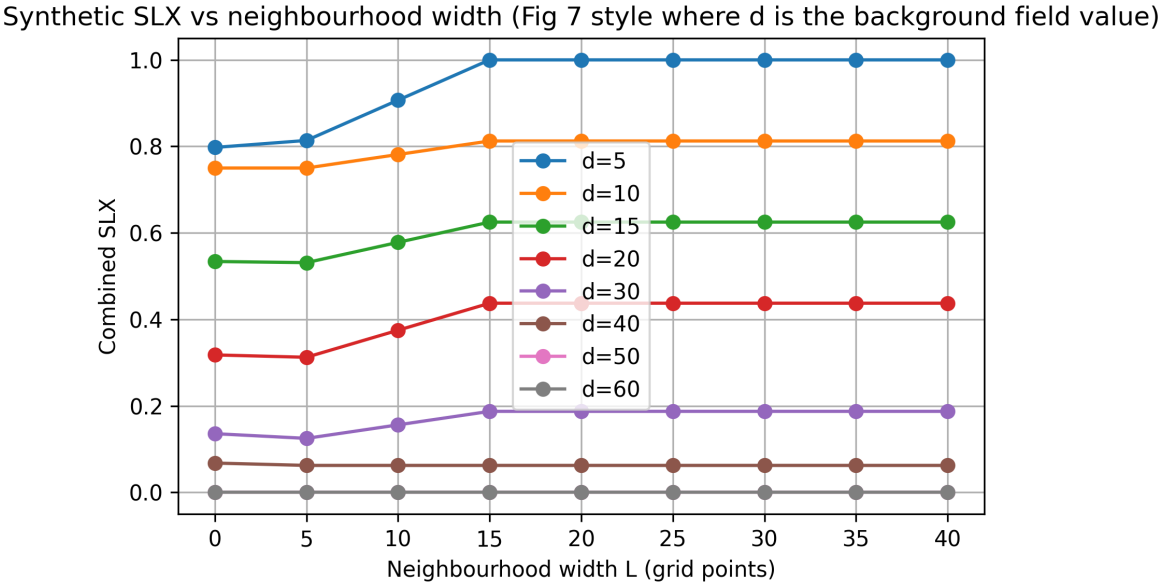
```
import numpy as np
import matplotlib.pyplot as plt

# ---- SLX score implementation ----

def S_score(ob, phi, k=0.1, A=4):
    """Piece-wise linear score function from Sass (2021)"""
    if ob > k:
        if phi < ob - k:
            return phi / (ob - k)
```

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


def local_extreme_indices(field, mode='max', delta=0.0):
    """Return list of (i,j) where the value is a local extreme (global for simplicity)."""
    if mode == 'max':
        target = field.max()
        mask = np.abs(field - target) <= delta
    else:
        target = field.min()
        mask = np.abs(field - target) <= delta
    return list(zip(*np.where(mask)))


def neighbourhood_view(field, i, j, L, mode='max'):
    n, m = field.shape
    i0, i1 = max(0, i - L), min(n, i + L + 1)
    j0, j1 = max(0, j - L), min(m, j + L + 1)
    neigh = field[i0:i1, j0:j1]
    return neigh.max() if mode == 'max' else neigh.min()


def SLX_components(analysis, forecast, L, delta=0.0):
    # identify extreme points
    ob_max_pts = local_extreme_indices(analysis, 'max', delta)
    ob_min_pts = local_extreme_indices(analysis, 'min', delta)
    fc_max_pts = local_extreme_indices(forecast, 'max', delta)
    fc_min_pts = local_extreme_indices(forecast, 'min', delta)

    # helper to compute average score over a list of points
    def avg_score(pts, ob_field, fc_field, mode):
        if not pts:
            return np.nan
        scores = []
```

```
        for (i, j) in pts:
            if mode == 'ob_max':
                ob = ob_field[i, j]
                phi = neighbourhood_view(fc_field, i, j, L, 'max')
            elif mode == 'ob_min':
                ob = ob_field[i, j]
                phi = neighbourhood_view(fc_field, i, j, L, 'min')
            elif mode == 'fc_max':
                ob = neighbourhood_view(ob_field, i, j, L, 'max')
                phi = fc_field[i, j]
            elif mode == 'fc_min':
                ob = neighbourhood_view(ob_field, i, j, L, 'min')
                phi = fc_field[i, j]
            else:
                raise ValueError
            scores.append(S_score(ob, phi))
        return np.mean(scores)

    # need access to both fields inside nested func
    global ob_field  # use globals for quick hack
    ob_field = analysis

    s_ob_max = avg_score(ob_max_pts, analysis, forecast, 'ob_max')
    s_ob_min = avg_score(ob_min_pts, analysis, forecast, 'ob_min')
    s_fc_max = avg_score(fc_max_pts, analysis, forecast, 'fc_max')
    s_fc_min = avg_score(fc_min_pts, analysis, forecast, 'fc_min')
    combined = np.nanmean([s_ob_max, s_ob_min, s_fc_max, s_fc_min])
    return combined, s_ob_max, s_ob_min, s_fc_max, s_fc_min

# ---- Synthetic case (Fig 7) ----


def generate_fields(N=50, block_size=10, b=5, d=15, c=5):
    analysis = np.full((N, N), b, dtype=float)
    forecast = np.full((N, N), d, dtype=float)
    # Observed field: two blocks at the top
    a1 = (15, 15+block_size, 10, 10+block_size)
    a2 = (15, 15+block_size, 30, 30+block_size)
    analysis[a1[0]:a1[1], a1[2]:a1[3]] = b + c
    analysis[a2[0]:a2[1], a2[2]:a2[3]] = b + c
    # Forecast field: two blocks at the bottom
    f1 = (30, 30+block_size, 10, 10+block_size)
```

```
    f2 = (30, 30+block_size, 30, 30+block_size)
    forecast[f1[0]:f1[1], f1[2]:f1[3]] = d + c
    forecast[f2[0]:f2[1], f2[2]:f2[3]] = d + c
    return analysis, forecast


#plot the fields
#forecast, analysis = generate_fields(d=5)


def plot_fields(forecast, analysis):
    fig, axs = plt.subplots(1, 2, figsize=(10, 4))
    im0 = axs[0].imshow(analysis, cmap='viridis', origin='lower')
    axs[0].set_title('Analysis Field')
    plt.colorbar(im0, ax=axs[0], fraction=0.046, pad=0.04)

    im1 = axs[1].imshow(forecast, cmap='viridis', origin='lower')
    axs[1].set_title('Forecast Field')
    plt.colorbar(im1, ax=axs[1], fraction=0.046, pad=0.04)

    plt.suptitle('Synthetic Analysis and Forecast Fields')
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()

# compute SLX for several d values
Ds = [5, 10, 15, 20, 30, 40,50,60]
#L_values = list(range(0, 21))
L_values = list(range(0,41,5))

results = {}
for d in Ds:
    analysis, forecast = generate_fields(d=d)
    #plot_fields(forecast, analysis)
    rows = []
    for L in L_values:
        combined, *_ = SLX_components(analysis, forecast, L)
        rows.append(combined)
    results[d] = rows

# Plotting similar to Figure 8
plt.figure(figsize=(6,4))
for d, rows in results.items():
```

```
    plt.plot(L_values, rows, marker='o', label=f'd={d}')
plt.title('Synthetic SLX vs neighbourhood width (Fig 7 style where d is the background field
plt.xlabel('Neighbourhood width L (grid points)')
plt.ylabel('Combined SLX')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Implementation in R

As we can see in the synthetic example, the SLX follows the general trend of getting worse as the analysis and forecast are farter from each other. We used the same approach to test the R/harp implementation of the SLX score. Instead of generating the data analytically we read a section of the CLAEF domain and created a flat field perturbed following the same pattern described in the code above and in the paper. The domain size is a bit bigger and the squares are displaced a bit more but the general shape is the same:

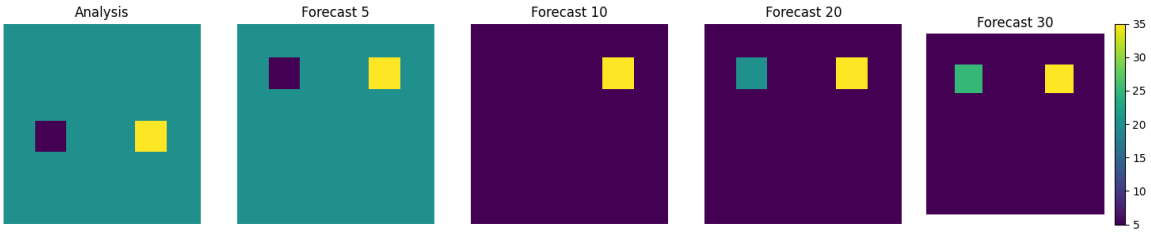


Figure 1: synthetic fields with different perturbations to the background foecast field (the analalysis remains the same)

The variation of SLX with L for different perturbations is shown in the plot below.

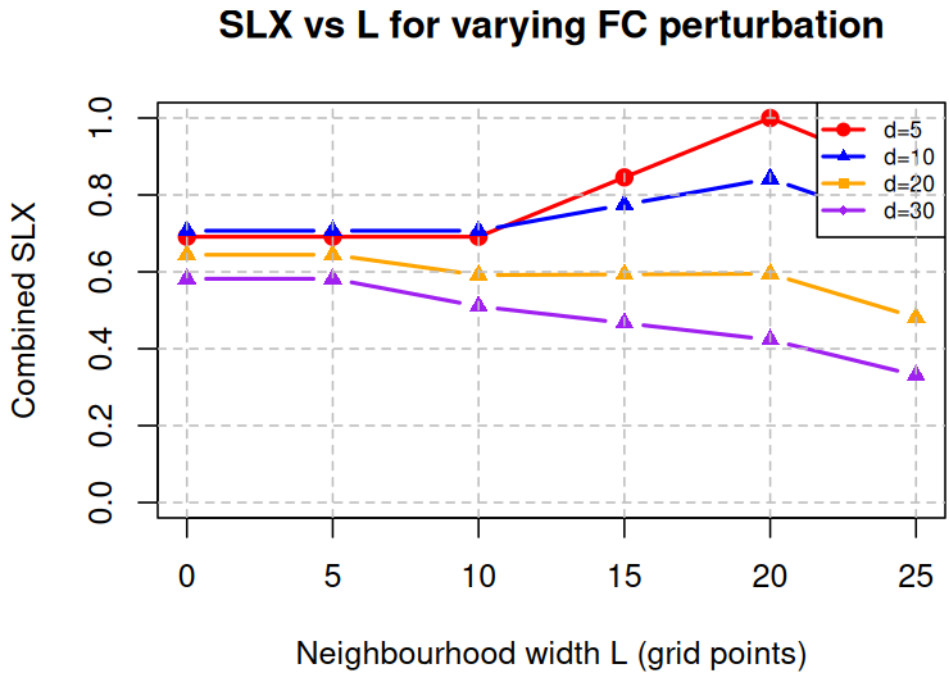


Figure 2: SLX results with synthetic data

Additionally, an analytic field similar to the one in the python code above was also produced and generated a similar trend. Results below.

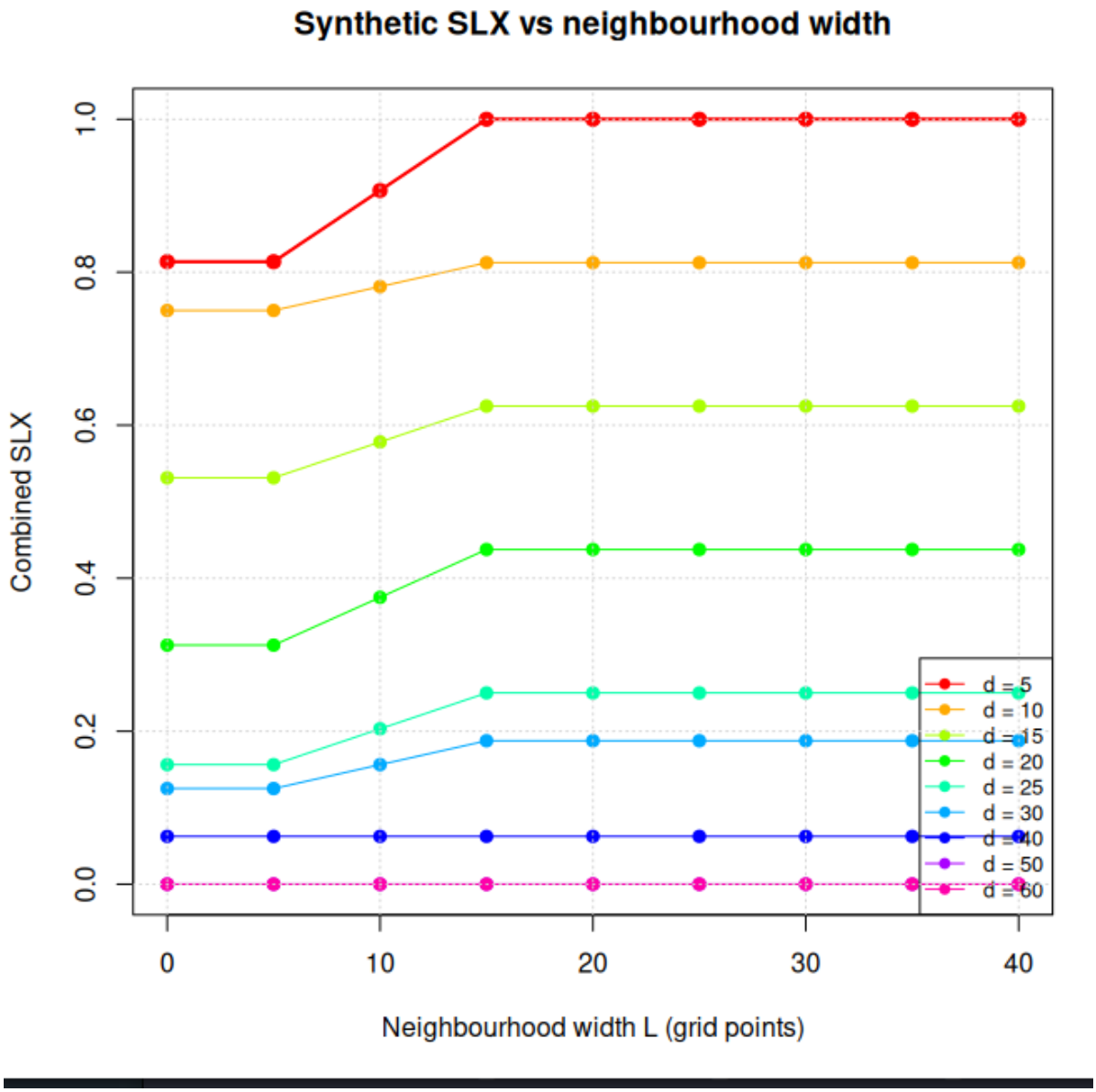


Figure 3: SLX results with analytic data

The code is available in this repository under `ACCORD_VS_202507/slx_validation_case.R`.

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 corrected presentation is available in this repository