

Supplementary Material for “Advanced Micronutrient Imaging in Plants: The POLYX Beamline at the NSRC SOLARIS JU”

M. PYPKA^a, A. BARABASZ^a, K. SOWA^b, P. WRÓBEL^{b,c},
T. KOŁODZIEJ^b, P. KORECKI^d AND O. SIEMIANOWSKI^{a,*}

^a*University of Warsaw, Faculty of Biology, Institute of Experimental Plant Biology and Biotechnology, Department of Plant Metal Homeostasis, Miecznikowa 1, 02-096 Warsaw, Poland*

^b*National Synchrotron Radiation Centre SOLARIS, Jagiellonian University, Czerwone Maki 98, 30-392 Kraków, Poland*

^c*AGH University of Krakow, Faculty of Physics and Applied Computer Science, Mickiewicza 30, 30-059 Kraków, Poland*

^d*Jagiellonian University, Institute of Physics, Łojasiewicza 11, 30-348 Kraków, Poland*

Received: 30.09.2025 & Accepted: 02.12.2025

Doi: [10.12693/APhysPolA.149.218.SS1](https://doi.org/10.12693/APhysPolA.149.218.SS1)

*e-mail: o.siemianowski@uw.edu.pl

Supplementary material provides additional details that complement and support the results presented in the main text of the article [1]. Included here are quantitative estimates of fluorescence self-absorption in hydrated plant tissue (Table S1), full-resolution elemental maps acquired using polycapillary and monocapillary optics (Fig. S1), anatomical reference images of tobacco roots (Fig. S2), and resolution-dependent co-localization

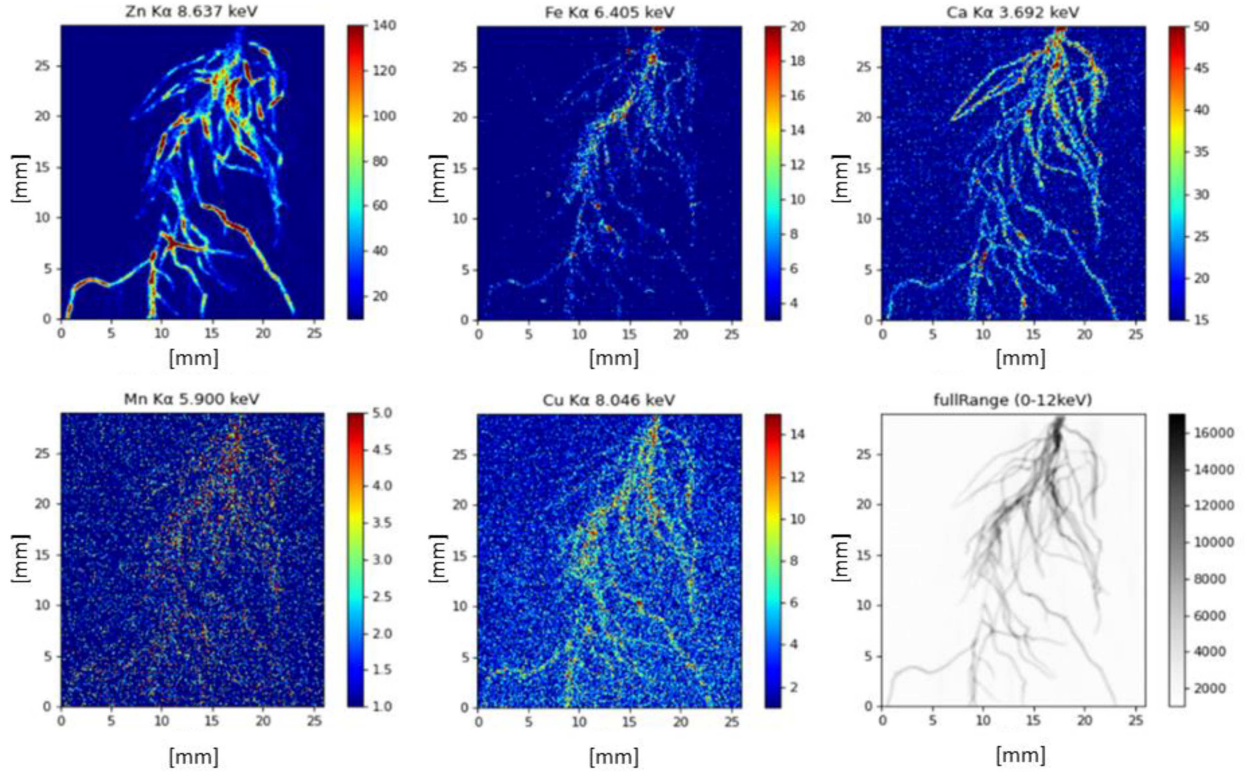
matrices (Fig. S3). The authors also provide the complete Python code used for image normalization, thresholding, co-localization analysis, and visualization. These materials are intended to facilitate reproducibility, enable independent verification of the analytical workflow, and provide deeper insight into the micronutrient distribution patterns obtained at the POLYX beamline.

TABLE SI

Approximate fluorescence self-absorption in hydrated plant tissue ($\sim 90\%$ water, $\rho \approx 1 \text{ g/cm}^3$) for Ca, Mn, Fe, Cu, and Zn $K\alpha$ lines at effective path lengths of 100, 200, and 400 μm , and at 45° detection geometry (path lengths scaled by $1/\cos(45^\circ) \approx 1.414$ to 141, 283, 566 μm). Values computed using the Beer–Lambert law, $A = 1 - \exp(-\frac{\mu}{\rho} \rho d)$, where: A — absorbance; μ — mass attenuation coefficient; ρ — density of the material; d — sample thickness. Representative water mass attenuation coefficients at the stated energies: Ca 3.69 keV, Mn 5.90 keV, Fe 6.40 keV, Cu 8.04 keV, Zn 8.64 keV (NIST X-Ray Mass Attenuation Coefficients database). Assumptions are as follows: homogeneous tissue, straight-line escape paths, no window/filter losses; numbers are interpretation guides, not calibration constants.

Element $K\alpha$ energy	100 μm absorption	200 μm absorption	400 μm absorption
Ca (3.69 keV)	$\sim 10\%$	$\sim 20\%$	$\sim 36\%$
Mn (5.90 keV)	$\sim 2.5\text{--}3\%$	$\sim 5\%$	$\sim 10\%$
Fe (6.40 keV)	$\sim 2\%$	$\sim 4\%$	$\sim 8\%$
Cu (8.04 keV)	$\sim 1\text{--}1.5\%$	$\sim 2\text{--}3\%$	$\sim 5\%$
Zn (8.64 keV)	$\sim 1\%$	$\sim 2\%$	$\sim 4\%$

Polycapillary optics



Monocapillary optics

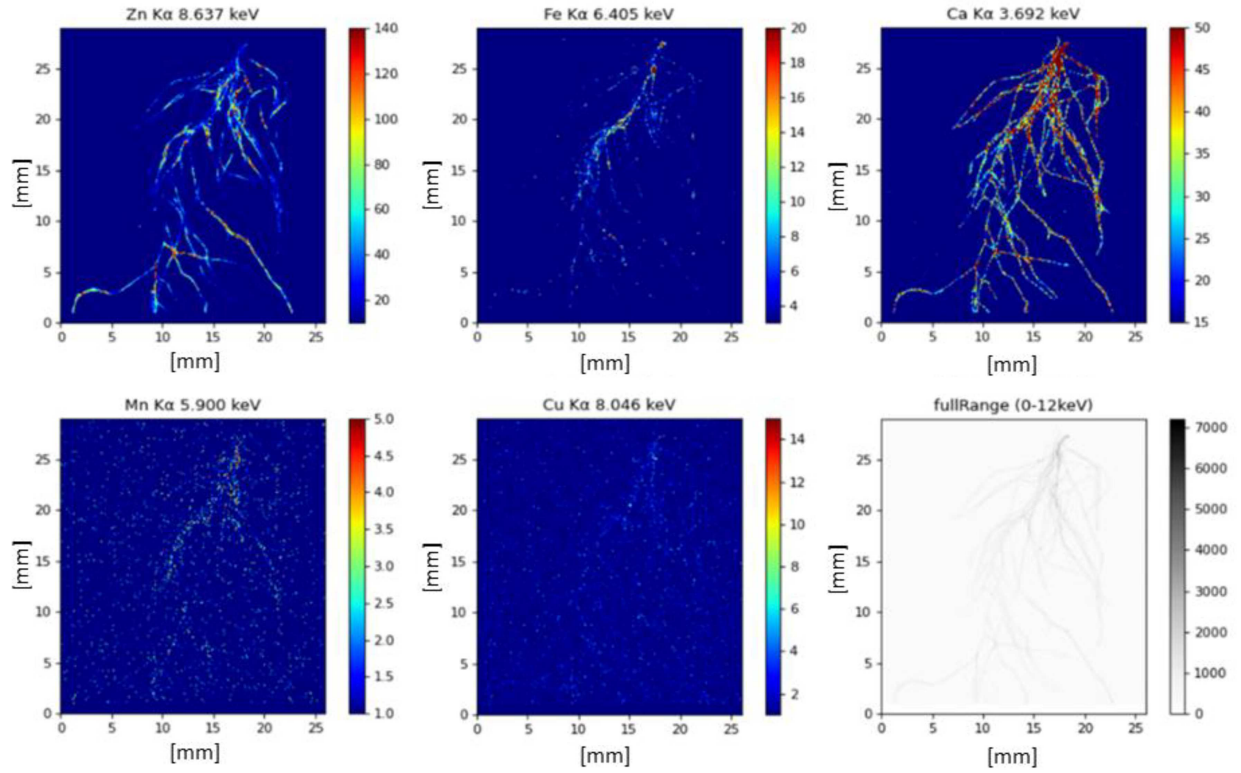


Fig. S1. Full map of the root system made using polycapillary and monocapillary optics.

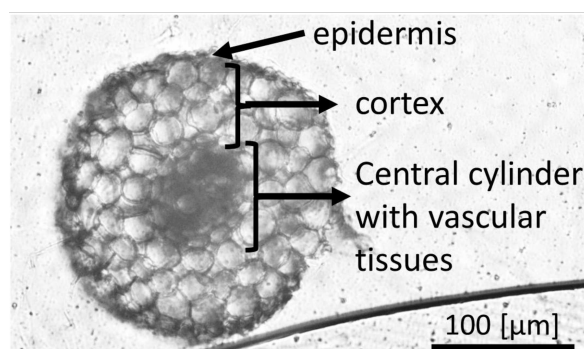


Fig. S2. Transverse section of a 5-week-old tobacco lateral root. Epidermis, cortex, and central cylinder are indicated.

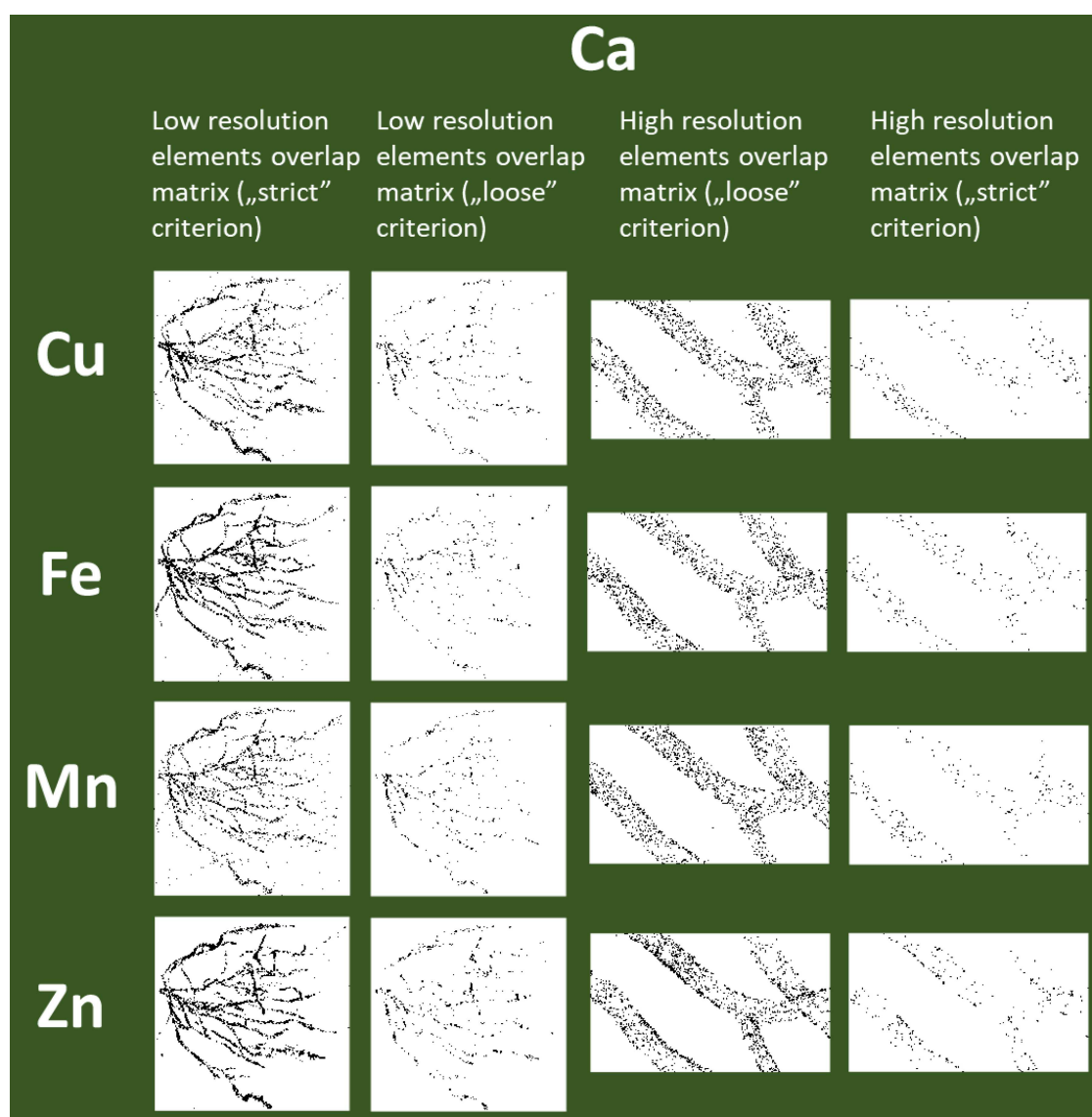


Fig. S3. Resolution-dependent co-localization matrices of Ca with other elements. Matrix overviews of binary overlap masks for Ca paired with Cu, Fe, Mn, and Zn under two imaging resolutions: (a) low (100 μm) and (b) high (5 μm). Each matrix includes masks generated using strict (97%) and loose (92%) pixel-wise overlap thresholds, as indicated. Individual cells show the spatial overlap between Ca and the element specified in the corresponding row, under the resolution and threshold condition defined by the column.

Codes used in the study (all need path and filename adjustment to be used)

Code 1

```

1: #CODE main to analyse co-localization and
  make composite .tiff images with overlaps
2: from pathlib import Path
3: import itertools
4: import numpy as np
5: import tifffile as tiff
6: import pandas as pd
7: from scipy.stats import pearsonr, spearmanr

8: # ===== SETTINGS =====

9: input_dir = Path(r"C:\...") #Please input
  full path to your images
10: output_dir = input_dir / "coloc_results"
11: output_dir.mkdir(exist_ok = True)

12: elements_to_use = ["Zn-Ka", "Fe-Kb",
  "Cu-Ka", "Mn-Ka", "Ca-Ka"] #depending on
  number of elements to be analysed and if given
  phrase is present in the file name — please
  adjust.

13: low_cut_percent = 5
14: high_cut_percent = 1
15: strict_percentile = 97
16: loose_percentile = 92

17: # ===== HELPERS =====

18: def load_tiff(path):
19:     return tiff.imread(path).astype(np.float32)

20: def normalize_per_channel(img, low_cut = 5,
  high_cut = 1):
21:     low = np.percentile(img, low_cut)
22:     high = np.percentile(img, 100 - high_cut)
23:     x = np.clip(img, low, high) - low
24:     scale = max(high - low, 1e-9)
25:     return (x / scale).astype(np.float32)

26: def to_uint8(mask):
27:     return (mask.astype(np.uint8) * 255)

28: def save_rgb_tiff(path, r, g, b):
29:     rgb_cyx = np.stack([to_uint8(r),
  to_uint8(g), to_uint8(b)], axis = 0)
30:     tiff.imwrite(path, rgb_cyx, imagej = True,
  metadata = {'axes': 'CYX'})

31: def manders(a, b, ta, tb):
32:     mask_a = a >= ta
33:     mask_b = b >= tb
34:     M1 = float(a[mask_b].sum() / a.sum())
  if a.sum() > 0 else 0.0

35:     M2 = float(b[mask_a].sum() / b.sum())
  if b.sum() > 0 else 0.0
36:     return M1, M2

37: # ===== FIND FILES =====

38: element_files = { }
39: for f in input_dir.glob("*.tif*"):
  #depending on name used this may be different
  — our files had _ROI_ — if other phrase is
  used, rplease adjust
40:     name = f.stem
41:     if "_ROI_" in name:
42:         element = name.split("_ROI_")[-1]
43:         if element in elements_to_use:
44:             element_files[element] = f

45: elements = sorted(element_files.keys())
46: print("Found elements:", elements)

47: # ===== LOOP OVER PAIRS =====

48: results = []
49: for el1, el2
  in itertools.combinations(elements, 2):
50:     print(f"Processing: {el1} vs {el2}")

51: img1 = normalize_per_channel
  (load_tiff(element_files[el1]),
  low_cut_percent, high_cut_percent)
52: img2 = normalize_per_channel
  (load_tiff(element_files[el2]),
  low_cut_percent, high_cut_percent)

53: # Thresholds

54: thr1_strict =
  np.percentile(img1, strict_percentile)
55: thr2_strict =
  np.percentile(img2, strict_percentile)
56: thr1_loose =
  np.percentile(img1, loose_percentile)
57: thr2_loose =
  np.percentile(img2, loose_percentile)

58: # Binary masks

59: bin1_strict = (img1 >= thr1_strict)
60: bin2_strict = (img2 >= thr2_strict)
61: overlap_strict = bin1_strict & bin2_strict
62: bin1_loose = (img1 >= thr1_loose)
63: bin2_loose = (img2 >= thr2_loose)
64: overlap_loose = bin1_loose & bin2_loose

65: # Save RGB TIFFs

66: save_rgb_tiff(output_dir /
  f"{el1}_{el2}_strict_mask_rgb.tiff",
  bin1_strict, bin2_strict, overlap_strict)

```

```

67: save_rgb_tiff(output_dir /
    f"{el1}_{el2}_loose_mask_rgb.tiff",
    bin1_loose, bin2_loose, overlap_loose)

68: # Metrics

69: flat1 = img1.flatten()
70: flat2 = img2.flatten()
71: pearson = pearsonr(flat1, flat2)[0]
72: spearman = spearmanr(flat1, flat2)[0]
73: M1_strict, M2_strict = manders(img1, img2,
    thr1_strict, thr2_strict)
74: M1_loose, M2_loose = manders(img1, img2,
    thr1_loose, thr2_loose)

75: results.append({
76:     "Element1": el1, "Element2": el2,
77:     "Pearson": pearson,
78:     "Spearman": spearman,
79:     "Manders_M1_strict": M1_strict,
80:     "Manders_M2_strict": M2_strict,
81:     "Manders_M1_loose": M1_loose,
82:     "Manders_M2_loose": M2_loose,
83:     "Thr1_strict": float(thr1_strict),
84:     "Thr2_strict": float(thr2_strict),
85:     "Thr1_loose": float(thr1_loose),
86:     "Thr2_loose": float(thr2_loose),
87:     "Pixels_used": img1.size
88: })

89: # ===== SAVE CSV =====

90: df = pd.DataFrame(results)
91: csv_path =
    output_dir / "co-localization_metrics.csv"
92: df.to_csv(csv_path, index = False)
93: print(f"Saved metrics table: {csv_path}")

Code 2

1: #CODE that makes heatmaps and graphs
    to summarize quantification from dataset
    extracted

2: from co-localization file
3: import numpy as np
4: import pandas as pd
5: import matplotlib.pyplot as plt
6: import seaborn as sns

7: # ===== Combined dataset (here past the data
    from co-localization_metrics.csv generated in
    #CODE 1) =====
8: # Each tuple is:
9: # (Element1, Element2,
10: # M5_loose, M100_loose,
11: # M5_strict, M100_strict,
12: # Pearson_5, Pearson_100,
13: # Spearman_5, Spearman_100)

14: pairs = [
15:     ("Ca","Cu", Manders for 5uM strict,
        Manders for 5uM loose,
        Manders for 100uM strict,
        Manders for 100uM loose,
        Pearson for 5uM,
        Pearson for 100uM,
        Spearman for 5uM,
        Spearman for 100uM),
16: ]
17: elements = ["Ca","Cu","Fe","Mn","Zn"]

18: # === Build matrices ===

19: def empty_matrix():
    return pd.DataFrame(np.nan,
        index = elements,
        columns = elements)

20: m5_loose, m100_loose = empty_matrix(),
    empty_matrix()
21: m5_strict, m100_strict = empty_matrix(),
    empty_matrix()

22: p5, p100 = empty_matrix(), empty_matrix()
23: s5, s100 = empty_matrix(), empty_matrix()
24: manders_loose_robust,
    manders_strict_robust, pearson_robust,
    spearman_robust, combined_stability =
    {}, {}, {}, {}

25: for e1,e2,m5l,m100l,m5s,m100s,p5v,p100v,s5v,
    s100v in pairs:
26:     # Fill symmetric matrices
27:     for M,val in [(m5_loose,m5l),
        (m100_loose,m100l),(m5_strict,m5s),
        (m100_strict,m100s),(p5,p5v),(p100,p100v),
        (s5,s5v),(s100,s100v)]:
28:         M.loc[e1,e2] = M.loc[e2,e1] = val
29:         key = f"{e1}-{e2}"
30:         manders_loose_robust[key] =
            m100l/m5l if m5l != 0 else np.nan
31:         manders_strict_robust[key] =
            m100s/m5s if m5s != 0 else np.nan
32:         pearson_robust[key] =
            p100v/p5v if p5v != 0 else np.nan
33:         spearman_robust[key] =
            s100v/s5v if s5v != 0 else np.nan
34:         combined_stability[key] =
            1 / (abs(manders_loose_robust[key]-1) +
                abs(manders_strict_robust[key]-1) +
                abs(pearson_robust[key]-1) +
                abs(spearman_robust[key]-1))

35: # Differences

36: mdiff_loose = m100_loose - m5_loose
37: mdiff_strict = m100_strict - m5_strict
38: pdiff = p100 - p5
39: sdiff = s100 - s5

```

```

40: # === Combined figure ===

41: sns.set(style="white")
42: fig = plt.figure(figsize = (28,18))
43: gs = fig.add_gridspec(4,4, width_ratios =
    [1,1,1,0.7])

44: heatmaps = [
45:     (m5_loose, "Manders Loose (5µm)",
    "magma", None, (0,0)),
46:     (m100_loose, "Manders Loose (100µm)",
    "magma", None, (0,1)),
47:     (mdiff_loose, "Manders Loose Δ",
    "RdBu_r", 0, (0,2)),
48:     (m5_strict, "Manders Strict (5µm)",
    "magma", None, (1,0)),
49:     (m100_strict, "Manders Strict (100µm)",
    "magma", None, (1,1)),
50:     (mdiff_strict, "Manders Strict Δ",
    "RdBu_r", 0, (1,2)),
51:     (p5, "Pearson (5µm)",
    "coolwarm", 0, (2,0)),
52:     (p100, "Pearson (100µm)",
    "coolwarm", 0, (2,1)),
53:     (pdiff, "Pearson Δ",
    "RdBu_r", 0, (2,2)),
54:     (s5, "Spearman (5µm)",
    "coolwarm", 0, (3,0)),
55:     (s100, "Spearman (100µm)",
    "coolwarm", 0, (3,1)),
56:     (sdiff, "Spearman Δ",
    "RdBu_r", 0, (3,2)),
57: ]

58: for data, title, cmap, center, (r,c) in heatmaps:
59:     ax = fig.add_subplot(gs[r,c])
60:     sns.heatmap(data, annot = True, fmt = ".2f",
    cmap = cmap, center = center,
    xticklabels = elements, yticklabels = elements,
    square = True,
    cbar_kws = {'label': 'Value'}, ax = ax)
61:     ax.set_title(title)

62: # Bar charts

63: ax1 = fig.add_subplot(gs[0,3])
64: sns.barplot(
    x = list(manders_loose_robust.values()),
    y = list(manders_loose_robust.keys()),
    color = "darkorange", ax = ax1)
65: ax1.axvline(1.0, color = "gray",
    linestyle = "-");

    ax1.set_title("Manders Loose Robustness")

66: ax2 = fig.add_subplot(gs[1,3])
67: sns.barplot(
    x = list(manders_strict_robust.values()),
    y = list(manders_strict_robust.keys()),
    color = "purple", ax = ax2)

68: ax2.axvline(1.0, color = "gray",
    linestyle = "-");
    ax2.set_title("Manders Strict Robustness")

69: ax3 = fig.add_subplot(gs[2,3])
70: sns.barplot(
    x = list(pearson_robust.values()),
    y = list(pearson_robust.keys()),
    color = "steelblue", ax = ax3)
71: ax3.axvline(1.0, color = "gray",
    linestyle = "-");
    ax3.set_title("Pearson Robustness")

72: ax4 = fig.add_subplot(gs[3,3])
73: sns.barplot(
    x = list(spearman_robust.values()),
    y = list(spearman_robust.keys()),
    color = "teal", ax = ax4)
74: ax4.axvline(1.0, color = "gray",
    linestyle = "-");
    ax4.set_title("Spearman Robustness")

75: plt.tight_layout()
76: plt.savefig("coloc_summary_combined1.png",
    dpi = 300, bbox_inches = "tight")
77: plt.savefig("coloc_summary_combined1.tiff",
    dpi = 300, bbox_inches = "tight")
78: plt.show()

79: # === Combined stability bar chart ===

80: plt.figure(figsize = (8,5))
81: stability_sorted =
    dict(sorted(combined_stability.items(),
    key = lambda x:x[1], reverse = True))
82: sns.barplot(x = list(stability_sorted.values()),
    y = list(stability_sorted.keys()),
    color = "seagreen")
83: plt.title("Combined Stability Score")
84: plt.tight_layout()
85: plt.savefig("coloc_combined.png", dpi = 300)
86: plt.savefig("coloc_combined.tiff", dpi = 300)
87: plt.show()

```

Code 3

```

1: #CODE that takes composite image and
    extracts overlap channel (third) making overlap
    matrix but inverted (W background B signal)

2: import os
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import tiff as tiff

6: # === SETTINGS ===

7: input_dir = r"C:\Users\...\\"
8: #Please input full path to your images

```

```

9: elements = ["Ca-Ka", "Cu-Ka", "Fe-Kb",
  "Mn-Ka", "Zn-Ka"] #depending on number
  of elements to be analysed and if given phrase
  is present in the file name – please adjust.
10: def create_overlap_matrix(threshold =
  "strict"):
11:     n = len(elements)
12:     fig, axs = plt.subplots(n, n,
  figsize = (1.6 * n, 1.6 * n))
13:     for i, e1 in enumerate(elements):
14:         for j, e2 in enumerate(elements):
15:             ax = axs[i, j]
16:             ax.axis("off")
17:             if i == j:
18:                 ax.text(0.5, 0.5, e1, ha = "center",
  va = "center", fontsize = 9)
19:             else:
20:                 el1, el2 = sorted([e1, e2])
21:                 fname =
  f"{el1}_{el2}_{threshold}_mask_rgb.tiff"
22:                 #depending on file name!
23:                 fpath = os.path.join(input_dir, fname)
24:                 if os.path.exists(fpath):
25:                     rgb = tiff.imread(fpath)
26:                     # Extract overlap channel
  (channel index 2)
27:                     if rgb.ndim == 3 and rgb.shape[0] == 3:
  # CYX
28:                         overlap = rgb[2] > 0
29:                     elif rgb.ndim == 3 and
  rgb.shape[-1] == 3: # HWC
30:                         overlap = rgb[..., 2] > 0
31:                     else:
32:                         continue
33:                     # Invert the grayscale image
34:                     inverted = np.logical_not(overlap)
35:                     ax.imshow(inverted, cmap = "gray")
36:                 else:
37:                     ax.text(0.5, 0.5, "N/A", ha = "center",
  va = "center", fontsize = 8,
  color = "gray")

38: fig.suptitle(f"Overlap Matrix ({threshold})",
  y = 0.995)
39: fig.tight_layout()
40: out_png = os.path.join(input_dir,
  f"overlap_matrix_{threshold}_overlap.png")
41: out_tif = os.path.join(input_dir,
  f"overlap_matrix_{threshold}_overlap.tiff")
42: fig.savefig(out_png, dpi = 300,
  bbox_inches = "tight")
43: fig.savefig(out_tif, dpi = 300,
  bbox_inches = "tight")
44: plt.show()

45: # === CREATE BOTH STRICT & LOOSE
  MATRICES ===

46: create_overlap_matrix("strict")
47: create_overlap_matrix("loose")

```

References

- [1] M. Pypka, A. Barabasz, K. Sowa, P. Wróbel, T. Kołodziej, P. Korecki, O. Siemi-anowski, *Acta Phys. Pol. A* **149**, 218 (2026).