Example use of Chromosight: Loops during yeast metaphase

August 27, 2021

In this notebook, we demonstrate how `chromosight quantify` can be used to compare chromatin loops between *S. cerevisiae* cultures arrested in G1 phase vs metaphase. In this notebook, we re-analyse Hi-C data from Garcia-Luis, J., Lazar-Stefanita, L., Gutierrez-Escribano, P. et al., 2019.

Input data:

Files used in this analysis are the output from `chromosight quantify`. Loop scores were computed on all 2-way combinations from a set of high confidence RAD21 binding sites separated by 10 to 50kb, on two Hi-C datasets at 2kb resolution: One with G1-arrested cells and the other with metaphase-arrested cells.

- `scer_w303_g1_2kb_SRR8769554.cool`: Hi-C matrix of cells stopped in G1 phase, at 2kb resolution. From Dauban et al. 2020
- `scer_w303_mitotic_2kb_merged.cool`: Hi-C matrix of metaphasic cells, at 2kb resolution. From Garcia-Luis et al. 2019
- `rad21.bed2d`: bed file containing all pairs of positions of RAD21 (cohesin) peaks in metaphasic *S. cerevisiae* separated by 10-50kb.

  Note: see the end of this notebook for an explanation on how to generate a bed2d file from a ChIP-seq bed file.

Getting loop scores

Loop scores at all pairs of positions can be computed using `chromosight quantify`. However, to ensure scores are comparable, the number of contacts should be similar between matrices. When using cool files, cooler can be used for this operation:

```
$ cooler info input/scer_w303_mitotic_2kb_merged.cool | grep sum
    "sum": 44048750

$ cooler info input/scer_w303_g1_2kb_SRR8769554.cool | grep sum
    "sum": 5862820
```

The G1 matrix has around 5.8M contacts whereas the metaphase matrix has 44M. Fortunately, chromosight has a `--subsample` option, which can be used to bring both matrices to the same coverage before computing scores:

```
chromosight quantify --pattern loops \
                     --subsample 5862820 \
                     --win-fmt npy \
                     scer_cohesin_peaks.bed2d \
                     input/scer_w303_g1_2kb_SRR8769554.cool \
                     quantify/rad21_g1

chromosight quantify --pattern loops \
                     --subsample 5862820 \
                     --win-fmt npy \
                     input/scer_cohesin_peaks.bed2d \
                     input/scer_w303_mitotic_2kb_merged.cool \
                     quantify/rad21_metaphase
```

For each condition, `chromosight quantify` generates 2 files:

- A table containing the coordinates and pattern matching scores of all input coordinates.
- A numpy binary file containing a stack of images around the input coordinates. Those images are stored in the same order as the coordinates from the table.

```
quantify/
├── rad21_g1.npy
├── rad21_g1.tsv
├── rad21_metaphase.npy
└── rad21_metaphase.tsv
```

Analysing loop scores

We can now use python to load and compare results from chromosight quantify. Below are a series of analyses showing some examples of downstream processing that can be performed on chromosight results.

[203]:
```
%config InlineBackend.figure_format = 'svg'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import chromosight.kernels as ck
import scipy.stats as st
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

res = 2000
```

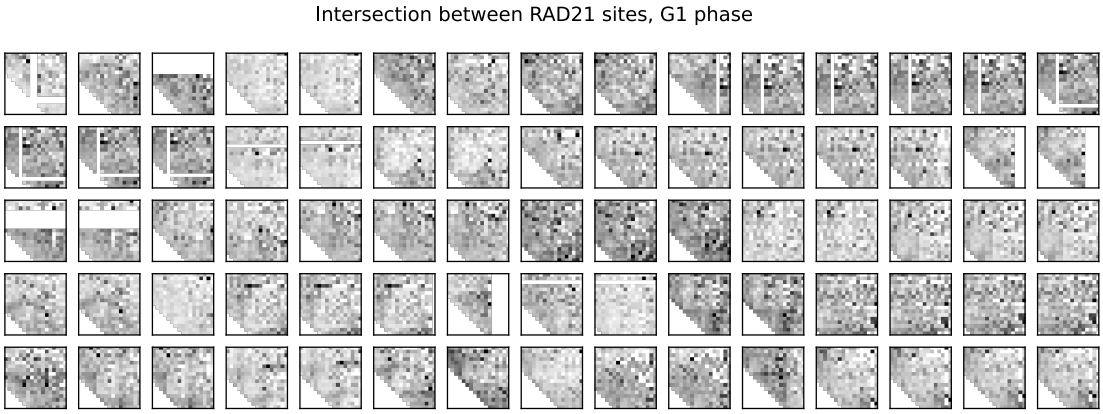
[204]:
```
# Load images (vignettes) around RAD21 interactions coordinates
images_g = np.load('quantify/rad21_g1.npy')
images_m = np.load('quantify/rad21_metaphase.npy')

# Load lists of RAD21 interactions coordinates with their loop scores
# Compute loop size (i.e. anchor distance) for each RAD21 combination
get_sizes = lambda df: np.abs(df.start2 - df.start1)
loops_g = pd.read_csv('quantify/rad21_g1.tsv', sep='\t')
loops_g['loop_size'] = get_sizes(loops_g)
loops_m = pd.read_csv('quantify/rad21_metaphase.tsv', sep='\t')
loops_m['loop_size'] = get_sizes(loops_m)

# Merge data from both conditions into a single table
loops_g['condition'] = 'g1'
loops_m['condition'] = 'metaphase'
loops_df = pd.concat([loops_g, loops_m]).reset_index(drop=True)
images = np.concatenate([images_g, images_m])

# Remove NaN scores (e.g. in repeated regions or overlap the matrix edge)
nan_mask = ~np.isnan(loops_df['score'])
loops_df = loops_df.loc[nan_mask, :]
images = images = images[nan_mask, :, :]

# The loop kernel can be loaded using chromosight.kernels.loops
kernel = np.array(ck.loops['kernels'][0])
pileup_kw = {'vmin': -1, 'vmax': 1, 'cmap': 'seismic'}
```

Peeking at the input coordinates

Images around RAD21 sites 2-way combinations extracted by chromosight can be viewed using numpy and matplotlib. Note there are series off overlapping and slightly shifted images. This is because of adjacent RAD21 sites which are closer in the genome than the size of the vignettes.

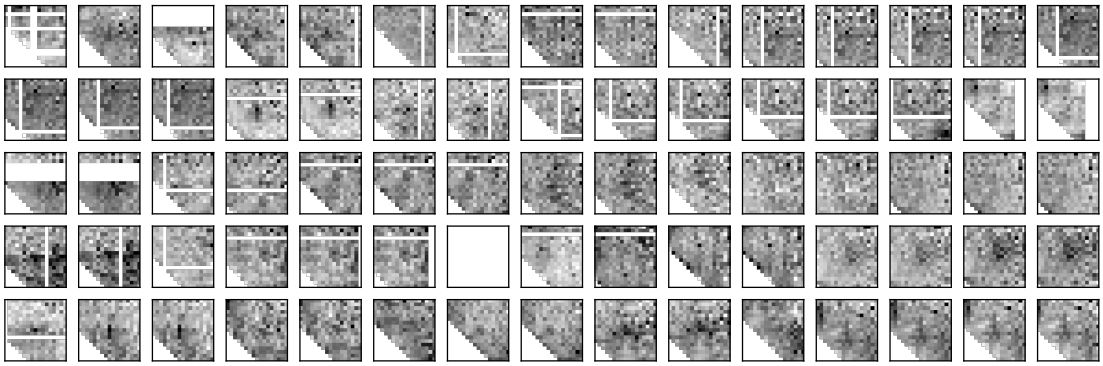
[193]:
```
# Decide how many rows and columns of images to show
r, c = 5, 15
valid_imgs = np.where(~loops_g.score.isnull() & ~loops_g.score.isnull())[0]
fig, axes = plt.subplots(r, c, figsize=(12, 4), subplot_kw={'xticks':[],␣
 ↪'yticks':[]})
# Show each image as a greyscale vignette
for i, ax in zip(valid_imgs, axes.flat):
    img = images_g[i, :, :] # Showing examples from the end of the image stack␣
 ↪(M phase)
    ax.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
plt.suptitle("Intersection between RAD21 sites, G1 phase")
```

[193]: Text(0.5, 0.98, 'Intersection between RAD21 sites, G1 phase')



[194]:
```
fig, axes = plt.subplots(r, c, figsize=(12, 4), subplot_kw={'xticks':[],
 ↪'yticks':[]})
first_m = np.where(loops_df.condition == 'metaphase')[0][0]
# Show each image as a greyscale vignette
for i, ax in zip(valid_imgs, axes.flat):
    img = images_m[i, :, :] # Showing examples from the end of the image stack
 ↪(M phase)
    ax.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
plt.suptitle("Intersection between RAD21 sites, Metaphase")
```

[194]: Text(0.5, 0.98, 'Intersection between RAD21 sites, Metaphase')

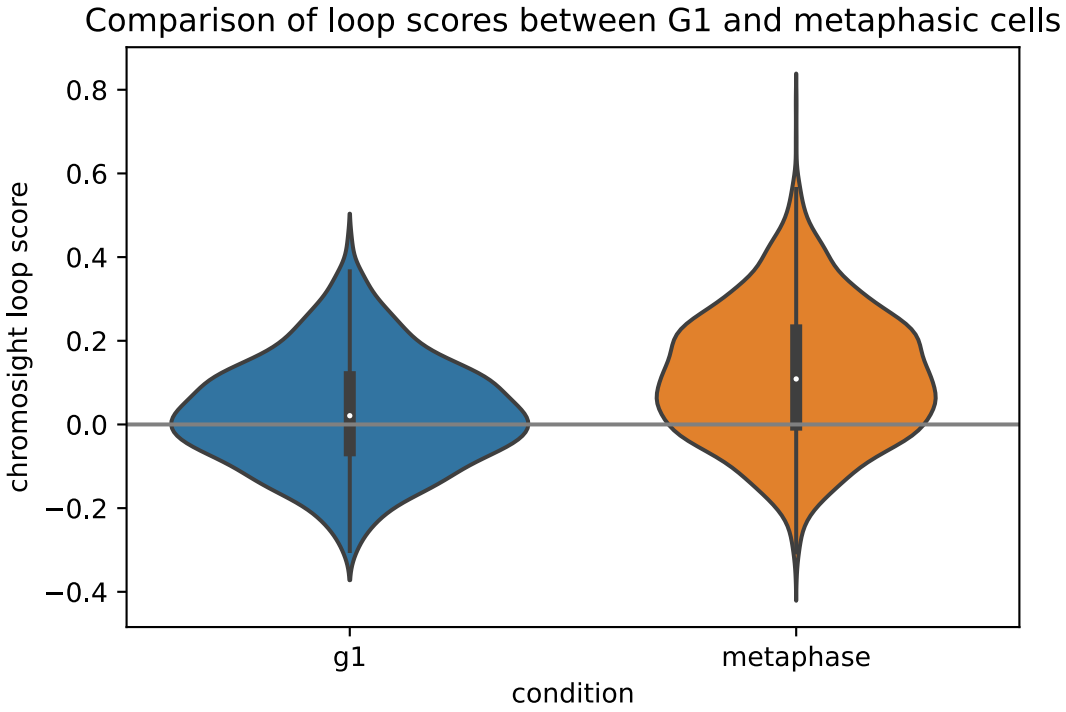


Comparing the distribution of scores

The distribution of chromosight scores (i.e. correlation coefficients with the loop kernel) can be compared between the 2 conditions, revealing that metaphasic cells tend to have stronger loops.

[196]:
```
sns.violinplot(data=loops_df, x='condition', y='score')
plt.ylabel('chromosight loop score')
plt.title('Comparison of loop scores between G1 and metaphasic cells')
plt.axhline(0, c='grey')
```

[196]: <matplotlib.lines.Line2D at 0x7f53f4892a50>

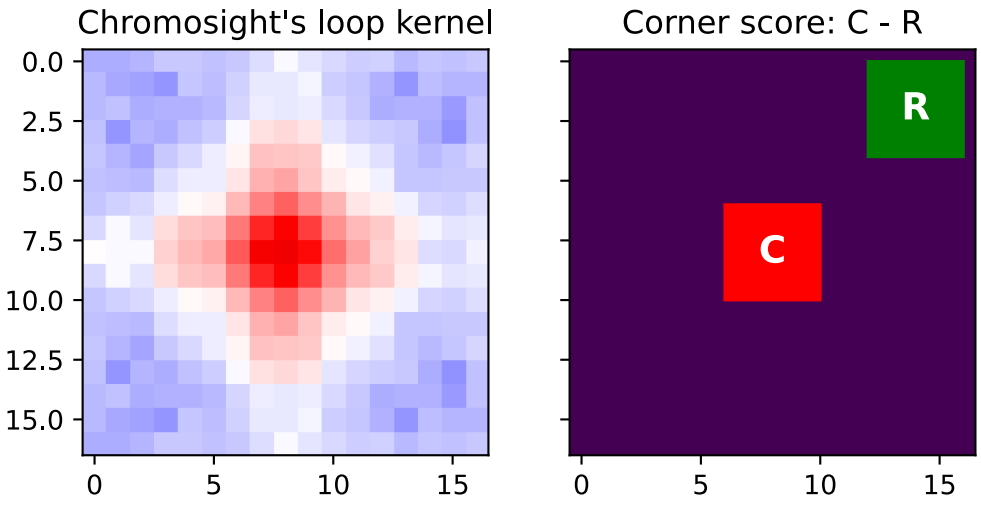


Using different metrics

Chromosight scores loops using their pearson correlation with a "loop kernel" (see below). However, one might want to use another metric than chromosight's score to rank loops. One such metric commonly used in the litterature is the "corner score", which uses the contrast between the center of the image (C) and the corner (R).

[197]:
```
import matplotlib.patches as patches
fig, axes = plt.subplots(1, 2, sharex=True, sharey=True)
axes[0].imshow(np.log(kernel), **pileup_kw)
axes[0].set_title("Chromosight's loop kernel")
axes[1].imshow(np.zeros((17, 17)))
center_rect = patches.Rectangle(
    (8-2, 8-2), 4, 4, linewidth=1, edgecolor='r', facecolor='r'
)
corner_rect = patches.Rectangle(
    (17-5, 0), 4, 4, linewidth=1, edgecolor='g', facecolor='g'
)
axes[1].annotate('C', (8, 8), color='w', weight='bold', fontsize=14,␣
 ↪ha='center', va='center')
axes[1].annotate('R', (14, 2), color='w', weight='bold', fontsize=14,␣
 ↪ha='center', va='center')
axes[1].add_patch(center_rect)
axes[1].add_patch(corner_rect)
axes[1].set_title("Corner score: C - R")
```

[197]:
```
Text(0.5, 1.0, 'Corner score: C - R')
```



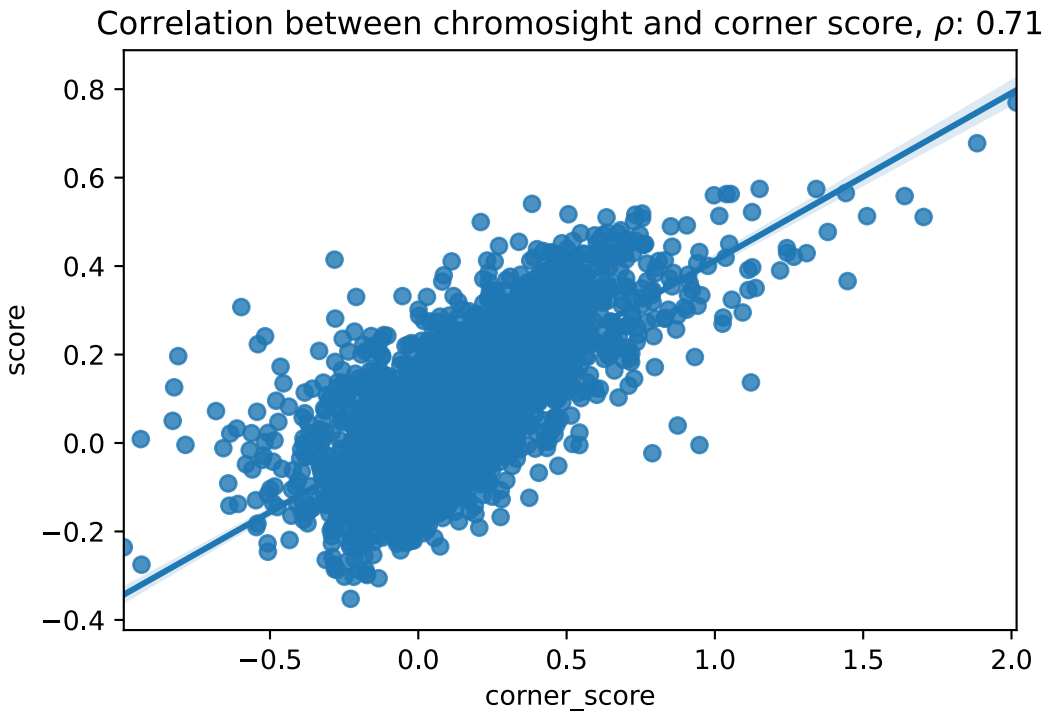
The function defined below could be used to compute the corner score. It computes the difference between the average of contacts in the center and top right corner. Using the rop right corner is better to avoid contacts enrichments for due to the diagonal. This is a pretty intuitive metric tailored based on expectations we have about loops. Here, we define center and corner radii as 10% of the image radius. For our 17x17 images, this means both regions will be 2+1 = 3x3 pixels.

[198]:
```
def corner_score(image, prop_radius=0.1):
    """
    Compute a loop intensity score from a pileup

    Parameters
    ----------
    image : numpy.array of floats
        2D array representing the window around a pattern.
    prop_radius : float
        Proportion of image radius used when selecting
        center and corner contacts.

    Returns
    -------
    float :
        Corner score, defined as mean(center) - mean(corner).
    """
    n, m = image.shape
    center = int(prop_radius * n)
    half_h = n // 2
    half_w = m // 2
    le = half_h - center
    ri = half_h + center + 1
    hi = half_w - center
    lo = half_w + center + 1
    center_mean = np.nanmean(image[hi:lo, le:ri])
    top_right_mean = np.nanmean(image[:hi, ri:])
    return center_mean - top_right_mean
```

This homemade corner score correlates well with chromosight's pearson score:

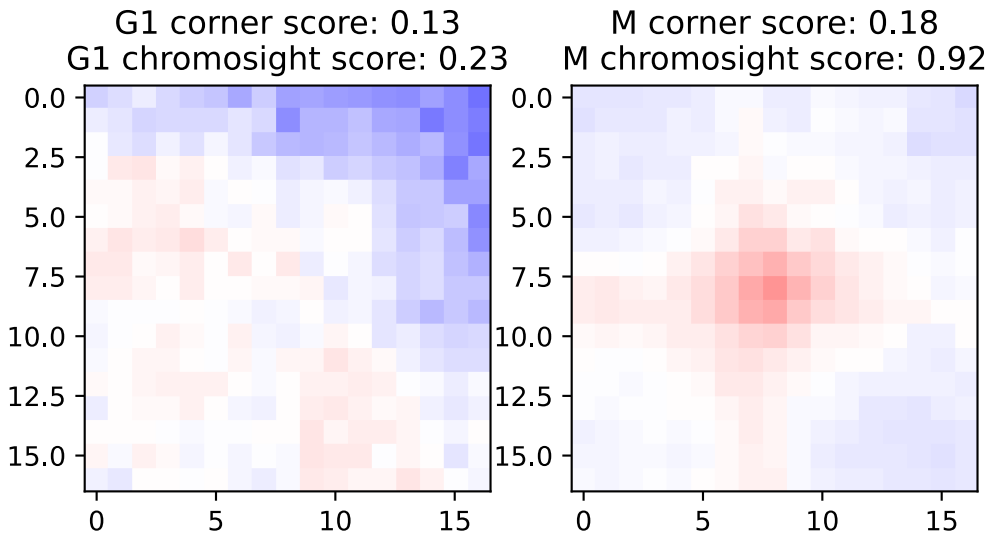
[199]:
```
import scipy.stats as st
loops_df['corner_score'] = [corner_score(m) for m in images]
comp_df = loops_df.loc[
    ~np.isnan(loops_df.corner_score) & ~np.isnan(loops_df.score), :
]
sns.regplot(data=comp_df, x='corner_score', y='score')
plt.title(
    r'Correlation between chromosight and corner score, $\rho$: '
    f'{np.round(st.pearsonr(comp_df.corner_score, comp_df.score)[0], 2)}')
```

[199]:
```
Text(0.5, 1.0, 'Correlation between chromosight and corner score, $\\rho$:
0.71')
```



By computing the pileup (average) of all patterns separately for G1 and M conditions, we can visually appreciate the stronger loop signal in metaphasic cells (M) compared to G1. Computing the chromosight and corner score directly on those pileups shows that the chromosight score makes it easier to discriminate the two conditions. The [-1,1] range is also convenient to interpret results. Note that the chromosight score below is just the pearson coefficient of the pileup with the loop kernel.

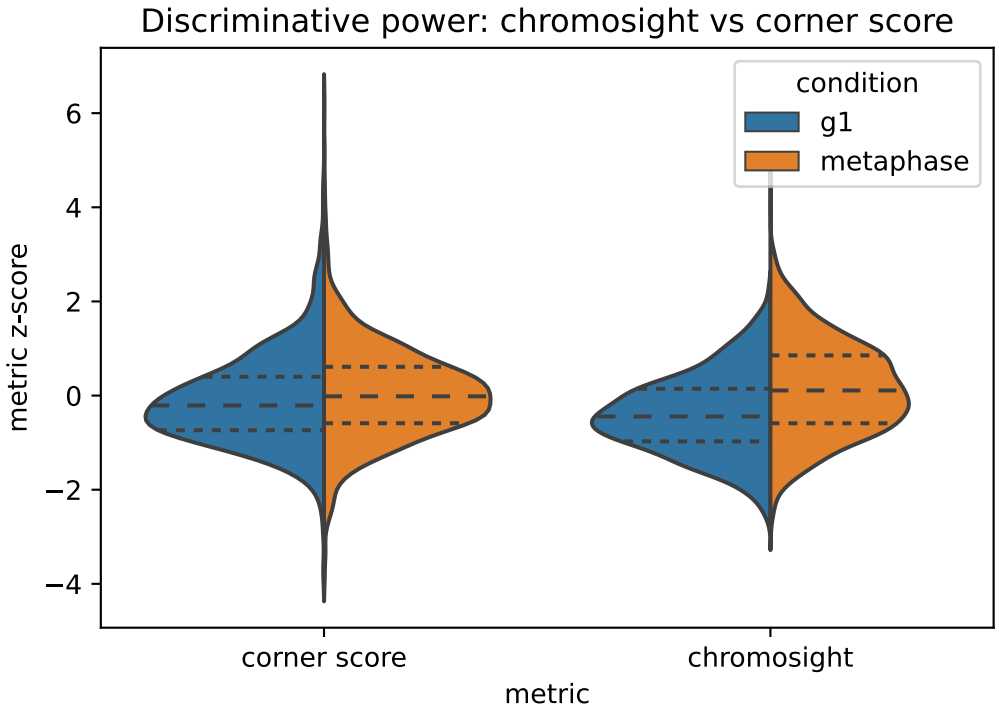
[200]:
```
centroid_g1 = np.apply_along_axis(np.nanmean, 0, images[loops_df.condition ==␣
 ↪'g1'])
centroid_m = np.apply_along_axis(np.nanmean, 0, images[loops_df.condition ==␣
 ↪'metaphase'])

fig, ax = plt.subplots(1, 2)
ax[0].imshow(np.log(centroid_g1), **pileup_kw)
ax[0].set_title(
    f'G1 corner score: {corner_score(centroid_g1):.2f}\n'
    f'G1 chromosight score: {np.round(st.pearsonr(centroid_g1.flat, kernel.
 ↪flat)[0], 2)}'
)
ax[1].imshow(np.log(centroid_m), **pileup_kw)
ax[1].set_title(
    f'M corner score: {corner_score(centroid_m):.2f}\n'
    f'M chromosight score: {st.pearsonr(centroid_m.flat, kernel.flat)[0]:.2f}'
)
plt.show()
```



Instead of summarizing the 2 conditions using only pileups, we can compare the ability of both score to separate the G1 and metaphasic cells based on the distribution of all patterns. Note that both scores are z-transformed to make their ranges comparable.

[201]:
```
corner = comp_df.drop('score', axis=1).rename(columns={'corner_score': 'score'})
corner['metric'] = 'corner score'
corner['score'] = st.zscore(corner['score'])
chromo = comp_df.drop('corner_score', axis=1)
chromo['metric'] = 'chromosight'
chromo['score'] = st.zscore(chromo['score'])
comp_scores = pd.concat([corner, chromo]).reset_index(drop=True)
sns.violinplot(data=comp_scores, x='metric', y='score', split=True,␣
 ↪hue='condition', inner='quartile')
plt.ylabel('metric z-score')
plt.title('Discriminative power: chromosight vs corner score')
```

[201]:
```
Text(0.5, 1.0, 'Discriminative power: chromosight vs corner score')
```



Comparison of loop footprints

For vizualisation purposes, each window can be summarized to a 1D band representing the sum of columns or rows. Here, we compute both the average of rows and columns, and use the element-wise average of both 1D vectors. This gives a good approximation of a 'loop footprint' and is convenient for visualisation.

Each image is centered to its mean to homogenize the overall contact counts in windows. This avoid having globally darker or lighter images and emphasizes relative contrasts within the im-

ages.

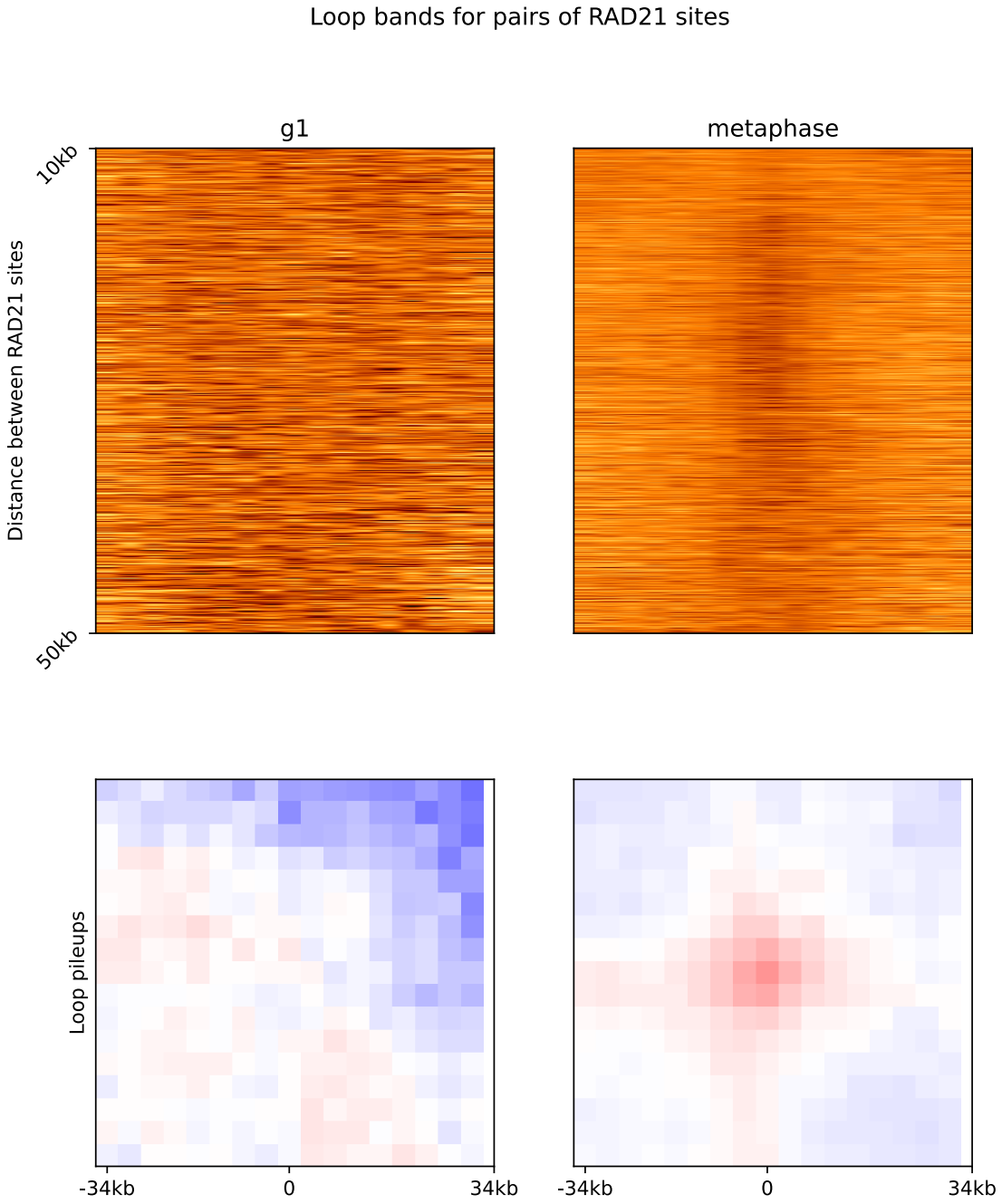
Bands are then sorted by loop size (i.e. distance between anchors) and plotted as a stack from shortest to longest distance interactions.

[ ]:
```
# Center images by subtracting their mean
centered = images.copy()
for img in range(centered.shape[0]):
    centered[img] -= np.nanmean(centered[img])

# Summarise each image by taking the average of its row and col sums.
bands = (np.nansum(centered, axis=1) + np.nansum(centered, axis=2)) / 2


# Reorder bands by distance between anchors
sort_var = 'loop_size'
sorted_bands = bands[np.argsort(loops_df[sort_var]), :]
sorted_cond = loops_df.condition.iloc[np.argsort(loops_df[sort_var])]
sorted_centered = centered[np.argsort(loops_df[sort_var])]

# Define a subset to visualise (too many images so see them all at once)
#smallest_group = np.min(np.unique(sorted_cond, return_counts=True)[1])-1
#smallest_group = 500

# Define saturation threshold for the colormaps
vmax_bands = np.percentile(bands, 99.9)
vmax_img = np.percentile(centered, 99)
```

[202]:
```
fig, axes = plt.subplots(2, 2, figsize=(8, 10))
for i, cond in enumerate(['g1', 'metaphase']):
    axes[0, i].imshow(
        sorted_bands[sorted_cond == cond, :],
        cmap='afmhot_r',
        vmax=vmax_bands,
    )
    axes[0, i].set_title(cond)
    # Compute pileup by averaging all windows for each condition
    centroid = np.apply_along_axis(
        np.nanmean,
        0,
        images[loops_df.condition == cond],
    )
    axes[1, i].imshow(np.log(centroid), **pileup_kw)
    axes[0,i].set_aspect('auto')
    # The rest is just to improve figure aesthetics
    axes[0, i].set_xticks([])
    axes[1, i].set_yticks([])
    if i > 0:
```

```
        axes[0, i].set_yticks([])
    else:
        #axes[0, i].set_yticklabels([], ["10kb", "25kb", "50kb"])
        axes[0, i].set_yticks(
            [0, sorted_bands[sorted_cond == cond, :].shape[0]]
        )
        axes[0, i].set_yticklabels(
            ['10kb', '50kb'],
            minor=False,
            rotation=45
        )

    axes[1, i].set_xticks([0, centroid.shape[0] // 2, centroid.shape[0]])
    half_w = int((res * centroid.shape[0] // 2) / 1000)
    half_w_bp = int(half_w * res / 1000)
    axes[1, i].set_xticklabels([f"{-half_w_bp}kb", "0", f"{half_w_bp}kb"])
    #axes[1, i].set_title(f"corner score: {np.round(corner_score(centroid), 2)}")

axes[0, 0].set_ylabel('Distance between RAD21 sites')
axes[1, 0].set_ylabel('Loop pileups')
plt.suptitle(f'Loop bands for pairs of RAD21 sites')
#plt.savefig('figs/bands_pileup_prots.svg')
```

[202]: Text(0.5, 0.98, 'Loop bands for pairs of RAD21 sites')



Note: Generating a BED2D file

ChIP-seq peaks are often stored as BED files, containing genomic intervals where DNA-binding proteins are enriched. Such files can be used to generate a BED2D file for `chromosight quantify`. This is done by generating all possible 2-ways combinations of peaks that follow desired criteria. In the example below, we use bedtools and awk to generate all intrachromosomal combinations

where peaks are separated by more than 10kb and less than 50kb.

```
MINDIST=10000
MAXDIST=50000
bedtools window -a input/scer_cohesin_peaks.bed \
                -b input/scer_cohesin_peaks.bed \
                -w $MAXDIST \
    | awk -vmd=$MINDIST '$1 == $4 && ($5 - $2) >= md {print}' \
    | sort -k1,1 -k2,2n -k4,4 -k5,5n \
    > input/scer_cohesin_peaks.bed2d
```