Change detection in pattern intensities

September 9, 2021

This notebook details the inner working of Pareidolia and walks through each step with code and visualization. We show the intermediate steps, parameters involved and use a small region of mouse chromosome 14 as an example to visualize the results.

Throughout the notebook, we call internal functions of pareidolia and chromosight to show the transformations going on inside the program. In a normal use case, however, this whole process is not neccessary and the user can simply execute the program as explained in the documentation, either through the command line interface or the python API.

Background

We compare several samples issued from 2 different timepoints (t_0, t_1). Multiple samples (replicates) $(r_1, r_2, ..., r_R)$ can share the same timepoint. Each sample has a matrix $M_{r,t}$ where $M_{r,t}[i, j]$ is a Pearson correlation coefficient with a kernel K representing the pattern of interest. If the kernel was of size $K_m x K_n$, the correlation coefficient was computed as:

$$M_{r,t}[i, j] = Corr(K, H_{r,t}[i - \frac{K_m}{2} : i + \frac{K_m}{2}, j - \frac{K_n}{2} : j + \frac{K_n}{2}])$$

Where $H_{r,t}$ is the Hi-C matrix of the sample.

The method is inspired by median filtering-based background formation. We start by generating a background matrix for each condition (timepoint), whose values are defined as the median of all replicates in that condition:

$$B_t[i, j] = median(M_{1,t}, M_{2,t}, ..., M_{R,t})$$

Next, we compute global background matrix, defined as:

$$B = median(B_1, B_2, ..., B_T)$$

Finally we extract the change as the difference between each condition 's background.

$$D = B_{t_1} - B_{t_0}$$

Changes can then filtered on various criteria to extract patches of strong differences

[486]:
```
from typing import Iterable, Optional, Tuple, Iterator, Set
from skimage.filters import threshold_otsu
import matplotlib.pyplot as plt
import scipy.sparse as sp
import chromosight.kernels as ck
import chromosight.utils.preprocessing as cup
import chromosight.utils.detection as cud
import pareidolia.detection as pad
import pareidolia.preprocess as pap
import pareidolia.hic_utils as pah
import numpy as np
import pandas as pd
import cooler
RES = 40000
region = 'chr14:12900000-14000000'
```

Example data

In this case, we are using mice bone macrophage infected by a bacterium at different timepoints. Each timepoint includes several replicates. We will be comparing infected samples to non-infected samples.

[487]:
```
def get_cool(sample: str) -> cooler.Cooler:
    """Given a sample name, load the corresponding cool file."""
    cool_path = f'./data/output/cool/{sample}.mcool::/resolutions/{RES}'
    try:
        cool = cooler.Cooler(cool_path)
    except OSError:
        cool = None
    return cool


# Load references to cool files from each sample into the dataframe
samples = pd.read_csv('samples.tsv', sep='\t', usecols=[0, 4])
samples["cool"] = samples.library.apply(get_cool)
samples['cond'] = 'control'
samples.loc[samples.infection_time > 0, 'cond'] = 'treat'
# Remove samples without Hi-C data
samples = samples.loc[~samples.cool.isnull(), :]
samples = samples.set_index('library')
samples.head(20)
```

[487]:

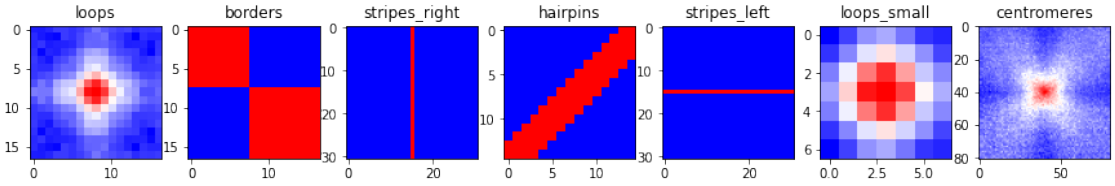
library	infection_time	cool	cond
PM51	0	<Cooler "PM51.mcool::/resolutions/40000">	control
PM52	20	<Cooler "PM52.mcool::/resolutions/40000">	treat
PM53	20	<Cooler "PM53.mcool::/resolutions/40000">	treat
PM54	2	<Cooler "PM54.mcool::/resolutions/40000">	treat
PM55	2	<Cooler "PM55.mcool::/resolutions/40000">	treat
PM121	0	<Cooler "PM121.mcool::/resolutions/40000">	control
PM122	20	<Cooler "PM122.mcool::/resolutions/40000">	treat
PM123	20	<Cooler "PM123.mcool::/resolutions/40000">	treat
PM124	2	<Cooler "PM124.mcool::/resolutions/40000">	treat
PM125	2	<Cooler "PM125.mcool::/resolutions/40000">	treat
PM126	20	<Cooler "PM126.mcool::/resolutions/40000">	treat
PM127	20	<Cooler "PM127.mcool::/resolutions/40000">	treat

Preprocessing matrices We first need to convert Hi-C matrices into convolution maps for a pattern of interest (e.g. loops). We also need to make sure we are comparing the same positions between maps by keeping the same nonzero values in sparse matrices. We also work on a subregion of the matrices to make computations faster.

Below, we perform the steps which are done internally in Pareidolia to explore the intermediates used to compute and filter changes. The examples are visualized on a small region to make visualizations more clear.

Pareidolia measures changes relative to the a pattern of interest, represented by the kernel K. Default chromosight kernels shown below can be used by providing the kernel name. Alternatively, a user defined matrix can be provided.

[488]:
```
%matplotlib inline

fig, ax = plt.subplots(1, len(ck.kernel_names), figsize=(15, 20))
for a, name in zip(ax, ck.kernel_names):
    kernel_mat = np.array(getattr(ck, name)['kernels'][0])
    a.imshow(np.log1p(kernel_mat), cmap='bwr')
    a.set_title(name)
```

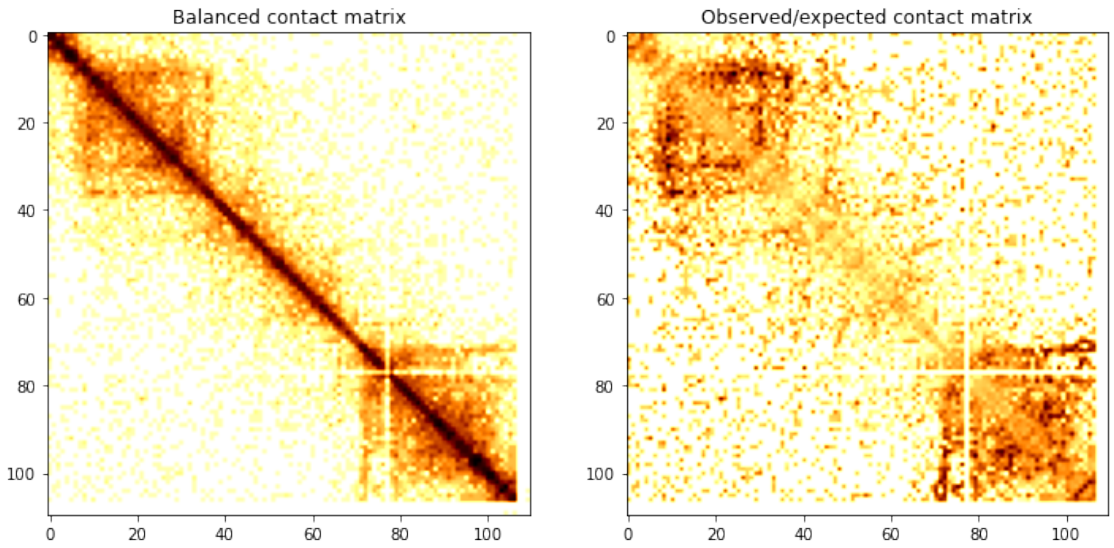


The first step is to preprocess Hi-C matrices. We work on sparse matrices to reduce memory usage. In order for samples to be comparable, they must have the same sparsity structure (the positions of explicitely stored values).

First, we subsample contacts in all matrices to ensure they have the same coverage (i.e. the coverage of the lowest sample). We also detrend the matrix for the distance-contact decay gradient. The resulting matrix (also called observed/expected) is computed by dividing each value by the average of its diagonal.

[489]:
```
%matplotlib inline
region = 'chr14:21600000-26000000'
# Compute lowest amount of contacts among all matrices
min_contacts = pah.get_min_contacts(samples.cool, region=region)

# Subsample, preprocess and subset matrices
samples['mat'] = samples.cool.apply(
    lambda c: pah.preprocess_hic(c, min_contacts=min_contacts, region=region)
)
fig, ax = plt.subplots(1, 2, figsize=(12, 8))
ax[0].imshow(np.log1p(samples.cool['PM51'].matrix(sparse=False, balance=False).
 ↪fetch(region)), cmap='afmhot_r')
ax[0].set_title('Balanced contact matrix')
ax[1].set_title('Observed/expected contact matrix')
ax[1].imshow(np.log1p(samples.mat['PM51'].toarray()), cmap='afmhot_r')
```

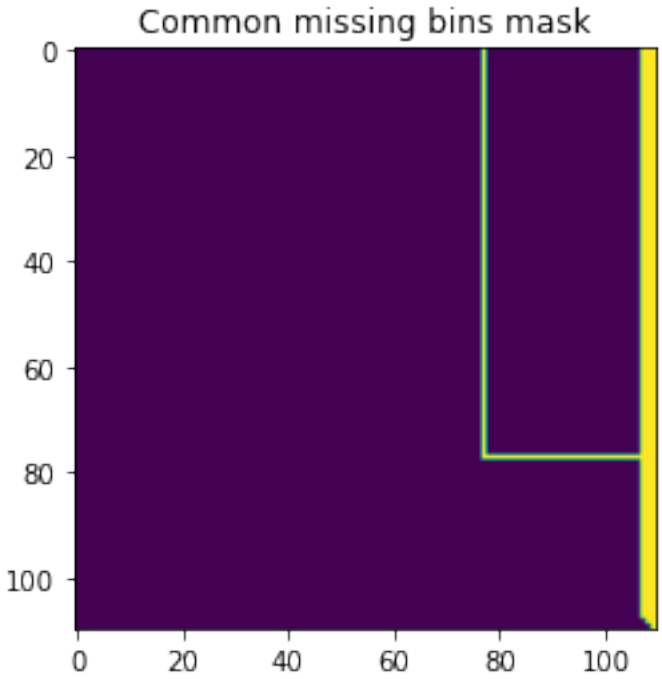
[489]:
```
<matplotlib.image.AxesImage at 0x7f86369de890>
```



Then, we compute the missing bins in each sample (genomic bins of low coverage) and take the union of those bins across samples. The resulting mask is applied on all samples.

[490]:
```
%matplotlib inline

# Get bins valid in all matrices
common_valid = pap.get_common_valid_bins(samples.mat)

# Generate mask to remove all bins missing in any matrix
common_mask = cup.make_missing_mask(
    samples.mat[0].shape,
    common_valid,
    common_valid,
    max_dist=None,
    sym_upper=True
)

# Make sure missing bins are set to 0 in all matrices and discard lower triangle
samples.mat = samples.mat.apply(lambda m: cup.erase_missing(sp.triu(m),␣
 ↪common_valid, common_valid))


plt.imshow(common_mask.toarray())
plt.title("Common missing bins mask")
```

[490]:
```
Text(0.5, 1.0, 'Common missing bins mask')
```



Each sample's preprocessed matrix is fed to Chromosight's convolution engine to return a matrix

of identical dimension whose values are the correlation coefficient with the kernel matrix at each position.

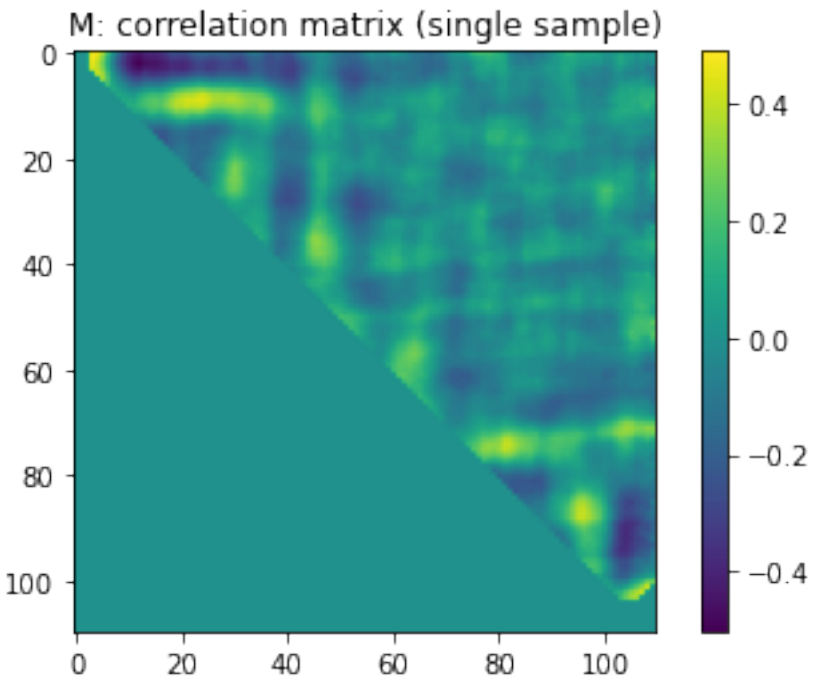
[491]:
```
# Compute correlation maps, using the same mask for all samples
# Note normxcorr2 returns a map of correlations, and one of p-values
# (None by default). Hence the slicing of the output
samples['corr'] = samples.mat.apply(lambda m: cud.normxcorr2(
    m,
    ck.loops['kernels'][0],
    full=True,
    missing_mask=common_mask,
    sym_upper=True
)[0])
```

Then, we take the union of nonzero positions across all samples' correlation matrices and enforce explicit storage of those positions across samples. This will ensure all samples have the same sparsity, so that we can perform operations directly on the aligned nonzero values.

[492]:
```
# First option: Keep values that are present in any matrix to zero
total_nnz_set = pap.get_nnz_union(samples['corr'])
samples['corr_union'] = samples['corr'].apply(lambda c: pap.fill_nnz(c,␣
 ↪total_nnz_set))
```

[493]:
```
%matplotlib inline
plt.imshow(samples.corr_union['PM51'].toarray())
plt.title("M: correlation matrix (single sample)")
plt.colorbar()
```

[493]:
```
<matplotlib.colorbar.Colorbar at 0x7f863684c350>
```



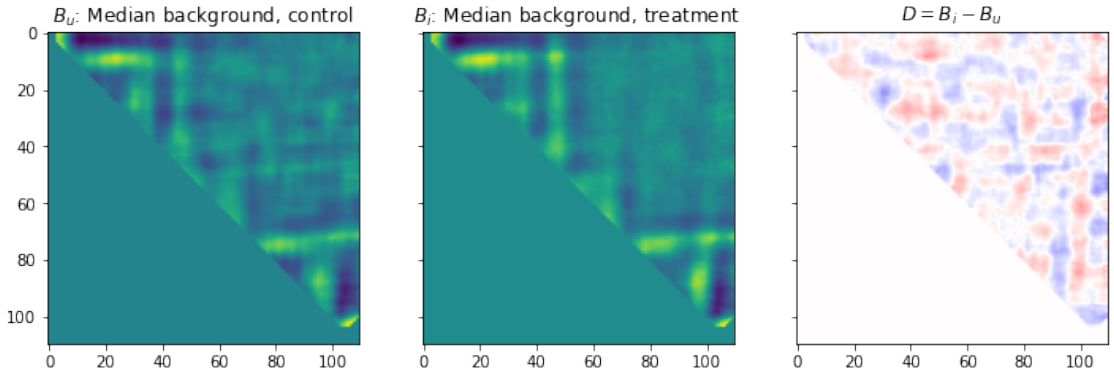
[494]:
```
# Take one infected (t=20h) and one uninfected matrix. Visualise the␣
 ↪median-filter
# based background accumulation
u_union = samples['corr_union'].loc[samples.cond == 'control'].values
i_union = samples['corr_union'].loc[samples.cond == 'treat'].values
bg_uni = pad.median_bg(u_union)
bg_inf = pad.median_bg(i_union)
```

[495]:
```
bg_inf
```

[495]:
```
<110x110 sparse matrix of type '<class 'numpy.float64'>'
        with 6067 stored elements in Compressed Sparse Row format>
```

[496]:
```
%matplotlib inline
diff = bg_inf.toarray() - bg_uni.toarray()

f, ax = plt.subplots(1, 3, sharex=True, sharey=True, figsize=(12, 4))
ax[0].imshow(bg_uni.toarray(), cmap='viridis')
ax[0].set_title("$B_u$: Median background, control")
ax[1].imshow(bg_inf.toarray(), cmap='viridis')
ax[1].set_title("$B_i$: Median background, treatment")
ax[2].imshow(diff, cmap='seismic', vmin=-1, vmax=1)
ax[2].set_title("$D= B_i - B_u$")
```

[496]:
```
Text(0.5, 1.0, '$D= B_i - B_u$')
```



Filtering changes

We can clearly identify regions that change between conditions, however this is only based on the median and does not account for variability. It would be better to penalize regions with technical variability.

Pareidolia uses a combination of 3 filters to select region with differential pattern intensity.

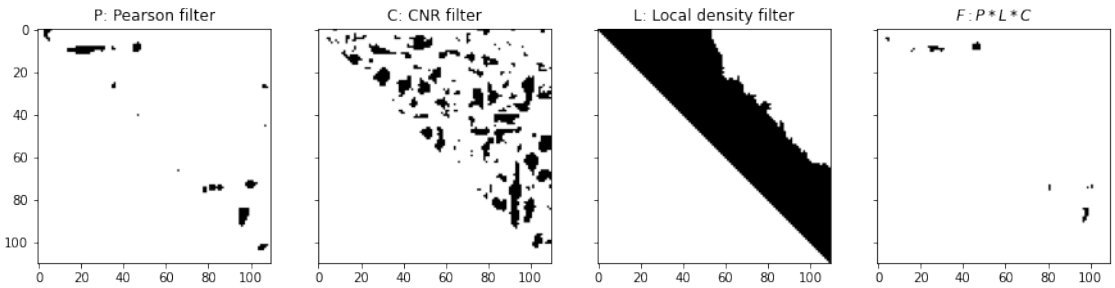
- Pearson score: At least 1 sample must have a pearson score (pattern similarity) above T_p.
- Local contact density: All samples must have a local contact density (nonzero contacts in surrounding window) above T_d
- Contrast-to-noise ratio: The contrast-to-noise-ratio $\frac{D}{\sigma}$ with sigma being within-condition position-wise standard errors must be above T_c

[497]:
```
Td, Tp, Tc = 0.2, 0.35, 0.2

pearson_fail = [
    (m.data < Tp).astype(bool) for m in samples["corr_union"]
]
pearson_fail = np.bitwise_and.reduce(pearson_fail)


# Get thresholded pearson matrix
ex = samples['corr_union']['PM51'].copy().tocoo()
P = sp.coo_matrix(
    ([True for _ in range(sum(~pearson_fail))], (ex.row[~pearson_fail], ex.
 ↪col[~pearson_fail])),
    shape=ex.shape,
    dtype=bool)

# Get thresholded CNR matrix
```

```
_, C = pah._median_bg_subtraction(
    pd.DataFrame(
        {
            'cond': ['ctrl' if c else 'treat' for c in samples.cond.values ==␣
 ↪'control'],
            'mat': samples.corr_union
        },
    ),
    control='ctrl',
    cnr_thresh=Tc
)
C.data[C.data < Tc] = 0.0
C.data[C.data > 0] = 1

# Get thresholded density matrix
L = pah.make_density_filter(samples.mat, Td, win_size=17, sym_upper=True)

# Convert everything to dense boolean array for visualization
filt = lambda m: m.toarray().astype(bool)
P, C, L = filt(P), filt(C), filt(L)
```

[504]:
```
%matplotlib inline
fig, ax = plt.subplots(1, 4, figsize=(15, 4), sharex=True, sharey=True)
ax[0].imshow(P, cmap='Greys')
ax[0].set_title('P: Pearson filter')
ax[1].imshow(C, cmap='Greys')
ax[1].set_title('C: CNR filter')
ax[2].imshow(L, cmap='Greys')
ax[2].set_title('L: Local density filter')
ax[3].imshow(P * C * L, cmap='Greys')
ax[3].set_title('$F: P * L * C$')
```

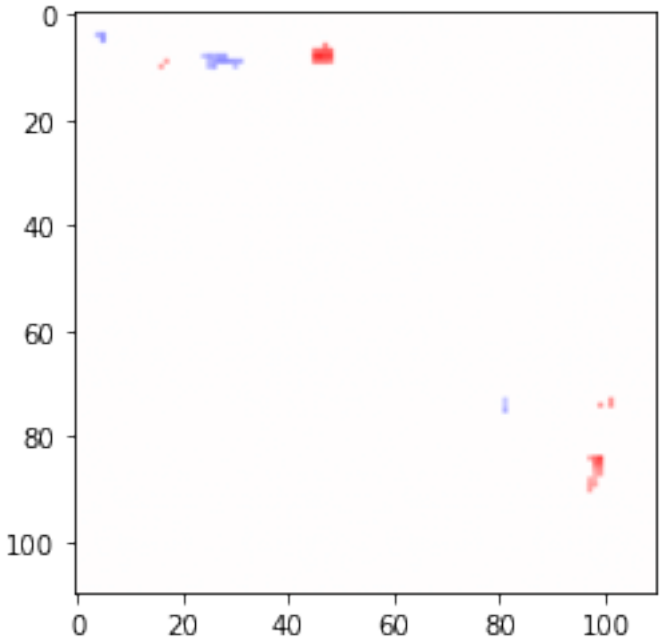
[504]:
```
Text(0.5, 1.0, '$F: P * L * C$')
```



We can then use the resulting filter F to mask the pattern chang matrix.

[505]:
```
%matplotlib inline
diff_f = diff * P * C * L
plt.imshow(diff_f, cmap='seismic', vmin=-.5, vmax=.5)
```

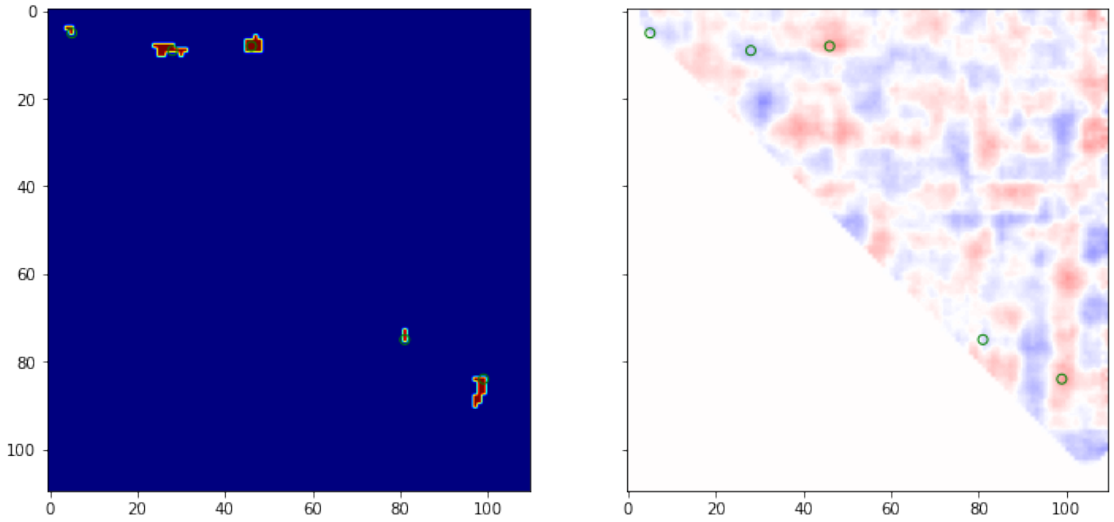
[505]:
```
<matplotlib.image.AxesImage at 0x7f8635e01bd0>
```



Finally, discrete patches of change (or foci) are extracted from the matrix, and the coordinate of the local maximum of change value in each patch is returned.

[506]:
```
%matplotlib inline
foci, foci_mat = cud.pick_foci(sp.csr_matrix(np.abs(diff_f)), pearson=0,␣
 ↪min_size=3)
foci_mat= foci_mat.toarray().astype(float)
foci_mat[foci_mat != 0] = 10
fig, ax = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(12, 8))
ax[0].imshow(foci_mat, cmap='jet')
ax[0].scatter(foci[:, 1], foci[:, 0], edgecolors='green', facecolors='none')
ax[1].imshow(diff, cmap='seismic', vmax=1, vmin=-1)
ax[1].scatter(foci[:, 1], foci[:, 0], edgecolors='green', facecolors='none')
```

[506]:
```
<matplotlib.collections.PathCollection at 0x7f8635cd4ed0>
```

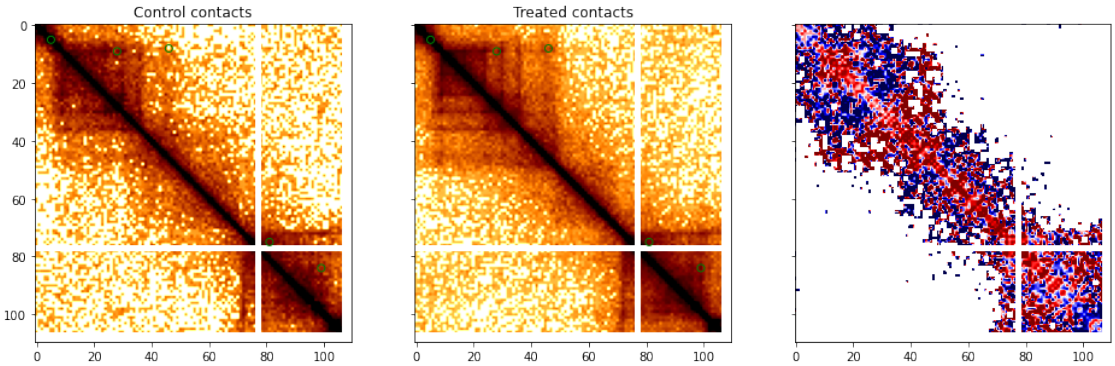


We can now visualize the original contact maps (averaged by condition) with the overlay of detected differential loop positions:

[507]:
```
control_contacts = np.dstack(
    [clr.matrix(balance=True, sparse=False).fetch(region) for clr in samples.
 ↪cool[samples.cond=='control']]
).mean(axis=2)
treatment_contacts = np.dstack(
    [clr.matrix(balance=True, sparse=False).fetch(region) for clr in samples.
 ↪cool[samples.cond=='treat'][:2]]
).mean(axis=2)
```

[508]:
```
%matplotlib inline
fig, ax = plt.subplots(1, 3, sharex=True, sharey=True, figsize=(16, 8))
vmax = np.nanpercentile(control_contacts**0.2, 98)
ax[0].imshow(control_contacts**0.2, cmap='afmhot_r', vmax=vmax)
ax[0].scatter(foci[:, 1], foci[:, 0], edgecolors='green', facecolors='none')
ax[0].set_title('Control contacts')
ax[1].imshow(treatment_contacts**0.2, cmap='afmhot_r', vmax=vmax)
ax[1].set_title('Treated contacts')
ax[1].scatter(foci[:, 1], foci[:, 0], edgecolors='green', facecolors='none')
ax[2].imshow(np.log2(treatment_contacts/control_contacts), cmap='seismic',␣
 ↪vmin=-0.5, vmax=0.5)
```

[508]:
```
<matplotlib.image.AxesImage at 0x7f8635ba4f90>
```


Control contacts
Treated contacts
0
20
40
60
80
100