LatticeData

August 5, 2018

1 DataBase for some commonly used cluster

1.1 unit-cell information

1.1.1 1D chain

- points
  - $p_0 = (0,)$
- translation-vectors
  - $\mathbf{a}_0 = (1,)$
  - $\mathbf{b}_0 = (2\pi,)$

1.1.2 square

- points
  - $p_0 = (0, 0)$
- translation-vectors
  - $\mathbf{a}_0 = (1, 0)$
  - $\mathbf{a}_1 = (0, 1)$
  - $\mathbf{b}_0 = (2\pi, 0)$
  - $\mathbf{b}_1 = (0, 2\pi)$

1.1.3 triangle

- points
  - $p_0 = (0, 0)$
- translation-vectors
  - $\mathbf{a}_0 = (1, 0)$
  - $\mathbf{a}_1 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (2\pi, -\frac{2\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (0, \frac{4\pi}{\sqrt{3}})$

1.1.4 honeycomb

- points
  - $p_0 = (0, 0)$
  - $p_1 = (0, \frac{1}{\sqrt{3}})$
- translation-vectors
  - $\mathbf{a}_0 = (1, 0)$
  - $\mathbf{a}_1 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (2\pi, -\frac{2\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (0, \frac{4\pi}{\sqrt{3}})$

1.1.5 kagome

- points
  - $p_0 = (0, 0)$
  - $p_1 = (\frac{1}{4}, \frac{\sqrt{3}}{4})$
  - $p_2 = (\frac{1}{2}, 0)$
- translation-vectors
  - $\mathbf{a}_0 = (1, 0)$
  - $\mathbf{a}_1 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (2\pi, -\frac{2\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (0, \frac{4\pi}{\sqrt{3}})$

1.1.6 cubic

- points
  - $p_0 = (0, 0, 0)$
- translation-vectors
  - $\mathbf{a}_0 = (1, 0, 0)$
  - $\mathbf{a}_1 = (0, 1, 0)$
  - $\mathbf{a}_2 = (0, 0, 1)$
  - $\mathbf{b}_0 = (2\pi, 0, 0)$
  - $\mathbf{b}_1 = (0, 2\pi, 0)$
  - $\mathbf{b}_2 = (0, 0, 2\pi)$

1.2 special cluster information

1.2.1 square-cross

- points
  - $p_0 = (0,0)$
  - $p_1 = (0,1)$
  - $p_2 = (1,1)$
  - $p_3 = (1,2)$
  - $p_4 = (2,2)$
  - $p_5 = (2,1)$
  - $p_6 = (3,1)$
  - $p_7 = (3,0)$
  - $p_8 = (2,0)$
  - $p_9 = (2,-1)$
  - $p_{10} = (1,-1)$
  - $p_{11} = (1,0)$
- translation-vectors
  - $\mathbf{a}_0 = (3,2)$
  - $\mathbf{a}_1 = (3,-2)$
  - $\mathbf{b}_0 = (\frac{\pi}{3}, \frac{\pi}{2})$
  - $\mathbf{b}_0 = (\frac{\pi}{3}, -\frac{\pi}{2})$

1.2.2 square-z

- points
  - $p_0 = (0,0)$
  - $p_1 = (0,1)$
  - $p_2 = (0,2)$
  - $p_3 = (1,2)$
  - $p_4 = (1,3)$
  - $p_5 = (2,3)$
  - $p_6 = (2,2)$
  - $p_7 = (2,1)$
  - $p_8 = (1,1)$
  - $p_9 = (1,0)$
- translation-vectors
  - $\mathbf{a}_0 = (3,1)$
  - $\mathbf{a}_1 = (1,-3)$
  - $\mathbf{b}_0 = (\frac{3\pi}{5}, \frac{\pi}{5})$
  - $\mathbf{b}_0 = (\frac{\pi}{5}, -\frac{3\pi}{5})$

1.2.3 triangle-star

- points
  - $p_0 = (0, 0)$
  - $p_1 = (0, -\sqrt{3})$
  - $p_2 = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$
  - $p_3 = (-\frac{3}{2}, -\frac{\sqrt{3}}{2})$
  - $p_4 = (-1, 0)$
  - $p_5 = (-\frac{3}{2}, \frac{\sqrt{3}}{2})$
  - $p_6 = (-\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $p_7 = (0, \sqrt{3})$
  - $p_8 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $p_9 = (\frac{3}{2}, \frac{\sqrt{3}}{2})$
  - $p_{10} = (1, 0)$
  - $p_{11} = (\frac{3}{2}, -\frac{\sqrt{3}}{2})$
  - $p_{12} = (\frac{1}{2}, -\frac{\sqrt{3}}{2})$
- translation-vectors
  - $\mathbf{a}_0 = (\frac{7}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{a}_1 = (\frac{5}{2}, -\frac{3\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (\frac{6\pi}{13}, \frac{10\pi}{13\sqrt{3}})$
  - $\mathbf{b}_0 = (\frac{2\pi}{13}, -\frac{14\pi}{13\sqrt{3}})$

1.2.4 honeycomb-benene

- points
  - $p_0 = (0, 0)$
  - $p_1 = (0, \frac{1}{\sqrt{3}})$
  - $p_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $p_3 = (1, \frac{1}{\sqrt{3}})$
  - $p_4 = (1, 0)$
  - $p_5 = (\frac{1}{2}, -\frac{1}{2\sqrt{3}})$
- translation-vectors
  - $\mathbf{a}_0 = (\frac{3}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{a}_1 = (\frac{3}{2}, -\frac{\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (\frac{2\pi}{3}, \frac{2\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (\frac{2\pi}{3}, -\frac{2\pi}{\sqrt{3}})$

1.2.5 honeycomb-diphenyl

- points
  - $p_0 = (0, 0)$
  - $p_1 = (0, \frac{1}{\sqrt{3}})$
  - $p_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $p_3 = (1, \frac{1}{\sqrt{3}})$
  - $p_4 = (\frac{3}{2}, \frac{\sqrt{3}}{2})$
  - $p_5 = (2, \frac{1}{\sqrt{3}})$
  - $p_6 = (2, 0)$
  - $p_7 = (\frac{3}{2}, -\frac{1}{2\sqrt{3}})$
  - $p_8 = (1, 0)$
  - $p_9 = (\frac{1}{2}, -\frac{1}{2\sqrt{3}})$
- translation-vectors
  - $\mathbf{a}_0 = (\frac{5}{2}, \frac{\sqrt{3}}{2})$
  - $\mathbf{a}_1 = (\frac{5}{2}, -\frac{\sqrt{3}}{2})$
  - $\mathbf{b}_0 = (\frac{2\pi}{5}, \frac{2\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (\frac{2\pi}{5}, -\frac{2\pi}{\sqrt{3}})$

1.2.6 honeycomb-gear

- points
  - $p_0 = (0, 0)$
  - $p_1 = (0, \frac{1}{\sqrt{3}})$
  - $p_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$
  - $p_3 = (\frac{1}{2}, \frac{5}{2\sqrt{3}})$
  - $p_4 = (1, \sqrt{3})$
  - $p_5 = (\frac{3}{2}, \frac{5}{2\sqrt{3}})$
  - $p_6 = (2, \sqrt{3})$
  - $p_7 = (\frac{5}{2}, \frac{5}{2\sqrt{3}})$
  - $p_8 = (\frac{5}{2}, \frac{\sqrt{3}}{2})$
  - $p_9 = (3, \frac{1}{\sqrt{3}})$
  - $p_{10} = (3, 0)$
  - $p_{11} = (\frac{5}{2}, -\frac{1}{2\sqrt{3}})$
  - $p_{12} = (\frac{5}{2}, -\frac{\sqrt{3}}{2})$
  - $p_{13} = (2, -\frac{2}{\sqrt{3}})$
  - $p_{14} = (\frac{3}{2}, -\frac{\sqrt{3}}{2})$
  - $p_{15} = (1, -\frac{2}{\sqrt{3}})$
  - $p_{16} = (\frac{1}{2}, -\frac{\sqrt{3}}{2})$
  - $p_{17} = (\frac{1}{2}, -\frac{1}{2\sqrt{3}})$

- $p_{18} = (1, 0)$
- $p_{19} = (1, \frac{1}{\sqrt{3}})$
- $p_{20} = (\frac{3}{2}, \frac{\sqrt{3}}{2})$
- $p_{21} = (2, \frac{1}{\sqrt{3}})$
- $p_{22} = (2, 0)$
- $p_{23} = (\frac{3}{2}, -\frac{1}{2\sqrt{3}})$

- translation-vectors
  - $\mathbf{a}_0 = (3, \sqrt{3})$
  - $\mathbf{a}_1 = (3, -\sqrt{3})$
  - $\mathbf{b}_0 = (\frac{\pi}{3}, \frac{\pi}{\sqrt{3}})$
  - $\mathbf{b}_1 = (\frac{\pi}{3}, -\frac{\pi}{\sqrt{3}})$

In [1]:
```
from itertools import product

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

In [2]:
```
# database for some commonly used unit-cells

dtype = np.float64

chain_cell_info = {
    "points": np.array([[0.0]], dtype=dtype),
    "vectors": np.array([[1.0]], dtype=dtype)
}

square_cell_info = {
    "points": np.array([[0.0, 0.0]], dtype=dtype),
    "vectors": np.array([[1.0, 0.0], [0.0, 1.0]], dtype=dtype)
}

triangle_cell_info = {
    "points": np.array([[0.0, 0.0]], dtype=dtype),
    "vectors": np.array([[1.0, 0.0], [0.5, np.sqrt(3)/2]], dtype=dtype)
}

honeycomb_cell_info = {
    "points": np.array([[0.0, 0.0], [0.0, 1/np.sqrt(3)]], dtype=dtype),
    "vectors": np.array([[1.0, 0.0], [0.5, np.sqrt(3)/2]], dtype=dtype)
}

kagome_cell_info = {
    "points": np.array(
        [[0, 0], [0.25, np.sqrt(3)/4], [0.5, 0.0]], dtype=dtype
    ),
```

```
    "vectors": np.array([[1, 0], [0.5, np.sqrt(3)/2]], dtype=dtype)
}

cubic_cell_info = {
    "points": np.array([[0.0, 0.0, 0.0]], dtype=dtype),
    "vectors": np.array(
        [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]], dtype=dtype
    )
}

common_cells_info = {
    "chain": chain_cell_info,
    "square": square_cell_info,
    "triangle": triangle_cell_info,
    "honeycomb": honeycomb_cell_info,
    "kagome": kagome_cell_info,
    "cubic": cubic_cell_info,
}
```

```
In [3]: # database for some commonly used clusters
square_cross_info = {
    "points": np.array(
        [[0.0, 0.0], [0.0, 1.0], [1.0, 1.0], [1.0, 2.0],
         [2.0, 2.0], [2.0, 1.0], [3.0, 1.0], [3.0, 0.0],
         [2.0, 0.0], [2.0, -1.0], [1.0, -1.0], [1.0, 0.0]],
        dtype=dtype
    ),
    "vectors": np.array([[3.0, 2.0], [3.0, -2.0]], dtype=dtype)
}

square_z_info = {
    "points": np.array(
        [[0.0, 0.0], [0.0, 1.0], [0.0, 2.0], [1.0, 2.0], [1.0, 3.0],
         [2.0, 3.0], [2.0, 2.0], [2.0, 1.0], [1.0, 1.0], [1.0, 0.0]],
        dtype=dtype
    ),
    "vectors": np.array([[3.0, 1.0], [1.0, -3.0]], dtype=dtype)
}

triangle_star_info = {
    "points": np.array(
        [[0.0, 0.0],
         [0.0, -np.sqrt(3)], [-0.5, -np.sqrt(3)/2], [-1.5, -np.sqrt(3)/2],
         [-1.0, 0.0], [-1.5, np.sqrt(3)/2], [-0.5, np.sqrt(3)/2],
         [0.0, np.sqrt(3)], [0.5, np.sqrt(3)/2], [1.5, np.sqrt(3)/2],
         [1.0, 0.0], [1.5, -np.sqrt(3)/2], [0.5, -np.sqrt(3)/2]],
        dtype=dtype
    ),
```

```
    "vectors": np.array(
        [[3.5, np.sqrt(3)/2], [2.5, -1.5*np.sqrt(3)]], dtype=dtype
    )
}

honeycomb_benzene_info = {
    "points": np.array(
        [[0.0, 0.0], [0.0, 1/np.sqrt(3)], [0.5, np.sqrt(3)/2],
         [1.0, 1/np.sqrt(3)], [1.0, 0.0], [0.5, -0.5/np.sqrt(3)]],
        dtype=dtype
    ),
    "vectors": np.array(
        [[1.5, np.sqrt(3)/2], [1.5, -np.sqrt(3)/2]], dtype=dtype
    )
}

honeycomb_diphenyl_info = {
    "points": np.array(
        [[0.0, 0.0], [0.0, 1/np.sqrt(3)], [0.5, np.sqrt(3)/2],
         [1.0, 1/np.sqrt(3)], [1.5, np.sqrt(3)/2], [2.0, 1/np.sqrt(3)],
         [2.0, 0.0], [1.5, -0.5/np.sqrt(3)], [1.0, 0.0],
         [0.5, -0.5/np.sqrt(3)]],
        dtype=dtype
    ),
    "vectors": np.array(
        [[2.5, np.sqrt(3)/2], [2.5, -np.sqrt(3)/2]], dtype=dtype
    )
}

honeycomb_gear_info = {
    "points": np.array(
        [[0.0, 0.0], [0.0, 1/np.sqrt(3)], [0.5, np.sqrt(3)/2],
         [0.5, 2.5/np.sqrt(3)], [1.0, np.sqrt(3)], [1.5, 2.5/np.sqrt(3)],
         [2.0, np.sqrt(3)], [2.5, 2.5/np.sqrt(3)], [2.5, np.sqrt(3)/2],
         [3.0, 1/np.sqrt(3)], [3.0, 0.0], [2.5, -0.5/np.sqrt(3)],
         [2.5, -np.sqrt(3)/2], [2.0, -2/np.sqrt(3)], [1.5, -np.sqrt(3)/2],
         [1.0, -2/np.sqrt(3)], [0.5, -np.sqrt(3)/2], [0.5, -0.5/np.sqrt(3)],
         [1.0, 0.0], [1.0, 1/np.sqrt(3)], [1.5, np.sqrt(3)/2],
         [2.0, 1/np.sqrt(3)], [2.0, 0.0], [1.5, -0.5/np.sqrt(3)]],
        dtype=dtype
    ),
    "vectors": np.array([[3, np.sqrt(3)], [3, -np.sqrt(3)]], dtype=dtype)
}

special_clusters_info = {
    "square_cross": square_cross_info,
    "square_12": square_cross_info,
    "cross": square_cross_info,
```

```
        "square_z": square_z_info,
        "square_10": square_z_info,
        "z": square_z_info,

        "triangle_star": triangle_star_info,
        "triangle_13": triangle_star_info,
        "star": triangle_star_info,

        "honeycomb_benzene": honeycomb_benzene_info,
        "honeycomb_6": honeycomb_benzene_info,
        "benzene": honeycomb_benzene_info,

        "honeycomb_diphenyl": honeycomb_diphenyl_info,
        "honeycomb_10": honeycomb_diphenyl_info,
        "diphenyl": honeycomb_diphenyl_info,

        "honeycomb_gear": honeycomb_gear_info,
        "honeycomb_24": honeycomb_gear_info,
        "gear": honeycomb_gear_info,
    }
```

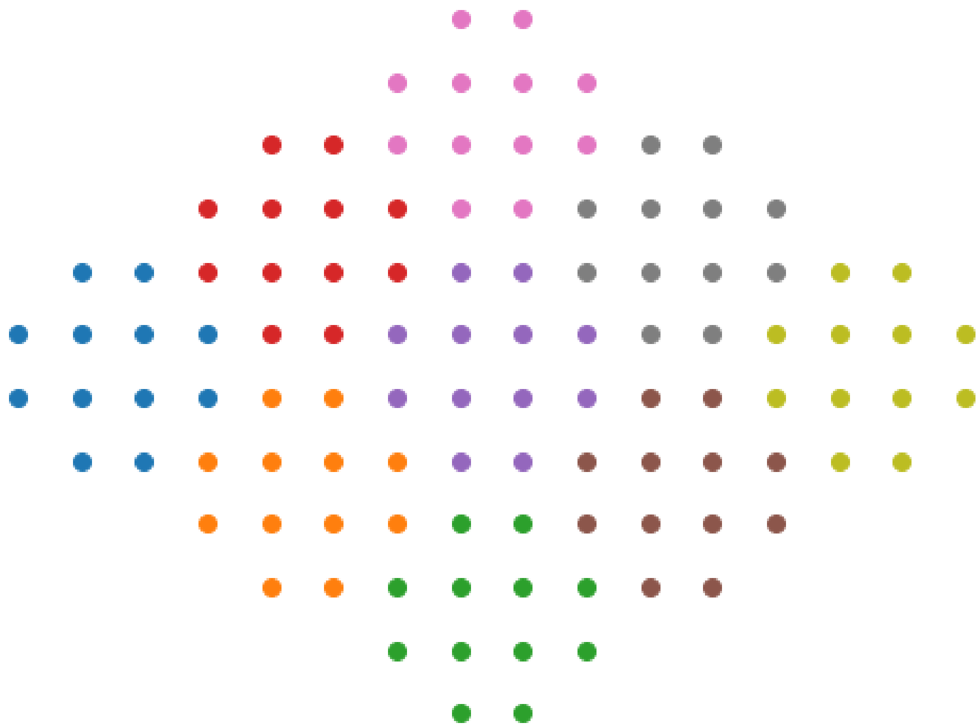
```
In [4]: def special_cluster(which):
            """
            Generating some special cluster

            Parameters
            ----------
            which : str
                Which special lattice to generate
                Currently supported special lattice:
                    "square_cross" | "square_z" | "triangle_star" |
                    "honeycomb_benzene" | "honeycomb_diphenyl" | "honeycomb_gear"
                Alias:
                    "square_cross" | "square_12" | "cross";
                    "square_z" | "square_10" | "z";
                    "triangle_star" | "triangle_13" | "star";
                    "honeycomb_benzene" | "honeycomb_6"| "benzene";
                    "honeycomb_diphenyl" | "honeycomb_10" | "diphenyl";
                    "honeycomb_gear" | "honeycomb_24" | "gear"

            Returns
            -------
            points : ndarray
                The coordinates of the points in the cluster
            vectors : ndarray
                The translation vectors of the cluster
            """
```

```python
    try:
        cluster_info = special_clusters_info[which]
        return cluster_info["points"], cluster_info["vectors"]
    except KeyError:
        raise KeyError("Unrecognized special lattice name!")


def lattice_generator(which, num0=1, num1=1, num2=1):
    """
    Generating a common cluster with translation symmetry

    Parameters
    ----------
    which : str
        Which  type of lattice to generate.
        Legal value:
            "chain" | "square" | "triangle" | "honeycomb" | "kagome" | "cubic"
    num0 : int, optional
        The number of unit cell along the first translation vector
        default: 1
    num1 : int, optional
        The number of unit cell along the second translation vector. It only
        takes effect for 2D and 3D lattice.
        default: 1
    num2 : int, optional
        The number of unit cell along the second translation vector. It only
        takes effect for 3D lattice.
        default : 1

    Returns
    -------
    points : ndarray
        The coordinates of the points in the cluster
    vectors : ndarray
        The translation vectors of the cluster
    """

    assert isinstance(num0, int) and num0 >= 1
    assert isinstance(num1, int) and num1 >= 1
    assert isinstance(num2, int) and num2 >= 1

    try:
        cell_info = common_cells_info[which]
        cell_points = cell_info["points"]
        cell_vectors = cell_info["vectors"]
    except KeyError:
        raise KeyError("Unrecognized lattice type!")
```

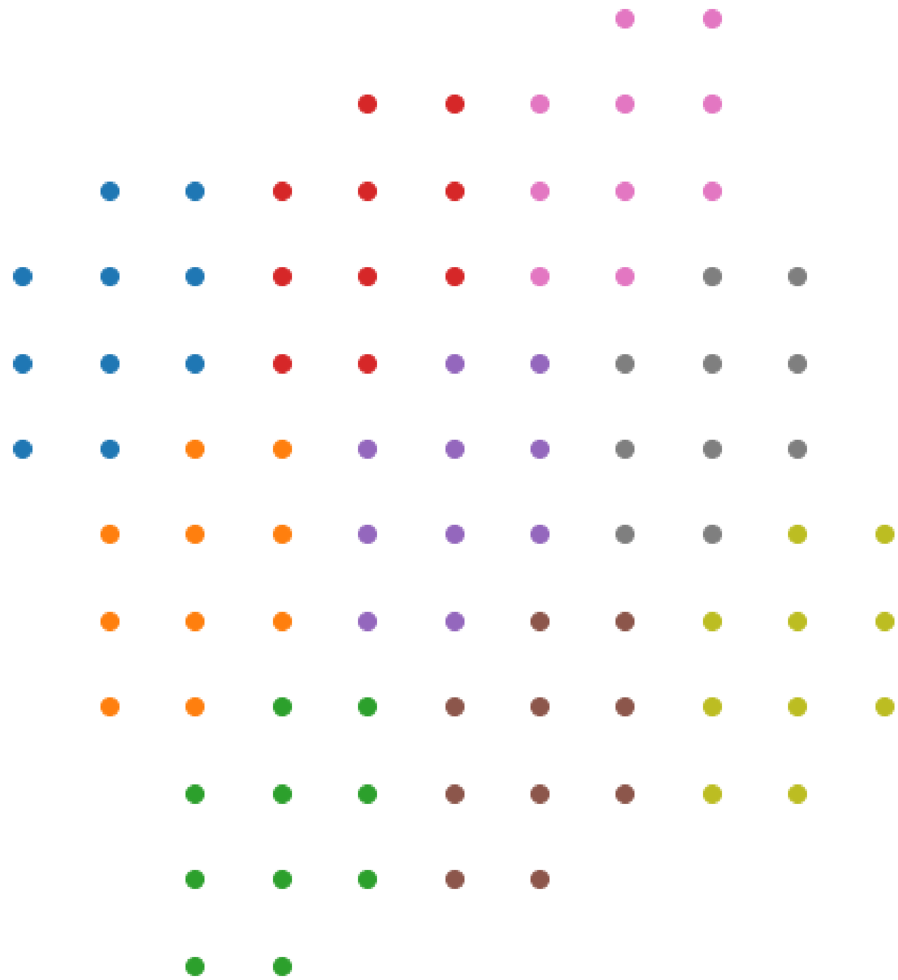
```
    if which == "chain":
        if num0 == 1:
            return cell_points, cell_vectors
        else:
            vectors = cell_vectors * np.array([[num0]])
            mesh = product(range(num0))
    elif which == "cubic":
        if num0 == 1 and num1 == 1 and num2 == 1:
            return cell_points, cell_vectors
        else:
            vectors = cell_vectors * np.array([[num0], [num1], [num2]])
            mesh = product(range(num0), range(num1), range(num2))
    else:
        if num0 == 1 and num1 == 1:
            return cell_points, cell_vectors
        else:
            vectors = cell_vectors * np.array([[num0], [num1]])
            mesh = product(range(num0), range(num1))

    dim = cell_points.shape[1]
    dRs = np.matmul(list(mesh), cell_vectors)
    points = np.reshape(dRs[:, np.newaxis, :] + cell_points, newshape=(-1, dim))

    return points, vectors

def show(points, vectors, scope=0):
    """
    Plot the given `points`

    Parameter
    ---------
    points : ndarray
        The coordinates of the points
    vectors : ndarray
        The trnaslation vectors of the cluster
    scope : int, optional
        Determine the number of cluster to draw
        default: 0
    """

    assert isinstance(points, np.ndarray) and points.ndim == 2
    assert isinstance(vectors, np.ndarray) and vectors.ndim == 2
    assert isinstance(scope, int) and scope >= 0

    point_num, space_dim = points.shape
    trans_dim, tmp = vectors.shape
    if space_dim > 2:
```

```python
                raise ValueError("Not supported space dimension!")

            if trans_dim > space_dim:
                raise ValueError("The number of translation vectors should be no more than the

            if tmp != space_dim:
                raise ValueError("The translation vectors should have the same space dimension

            clusters = [
                points + np.matmul(tmp, vectors)
                for tmp in product(range(-scope, scope+1), repeat=trans_dim)
            ]

            fig, ax = plt.subplots()
            fig.set_size_inches((16, 9))
            ax.set_axis_off()
            ax.set_aspect("equal")

            if space_dim == 1:
                ys = np.zeros(shape=points.shape)
                for cluster in clusters:
                    ax.plot(cluster, ys, marker="o", ls="None", ms=8)
            else:
                for cluster in clusters:
                    ax.plot(cluster[:, 0], cluster[:, 1], marker="o", ls="None", ms=8)

            left, right = ax.get_xlim()
            bottom, top = ax.get_ylim()
            half = max(right - left, top - bottom) / 2
            x_center = (right + left) / 2
            y_center = (top + bottom) / 2
            ax.set_xlim(left=x_center-half, right=x_center+half)
            ax.set_ylim(bottom=y_center-half, top=y_center+half)
            plt.show()
```

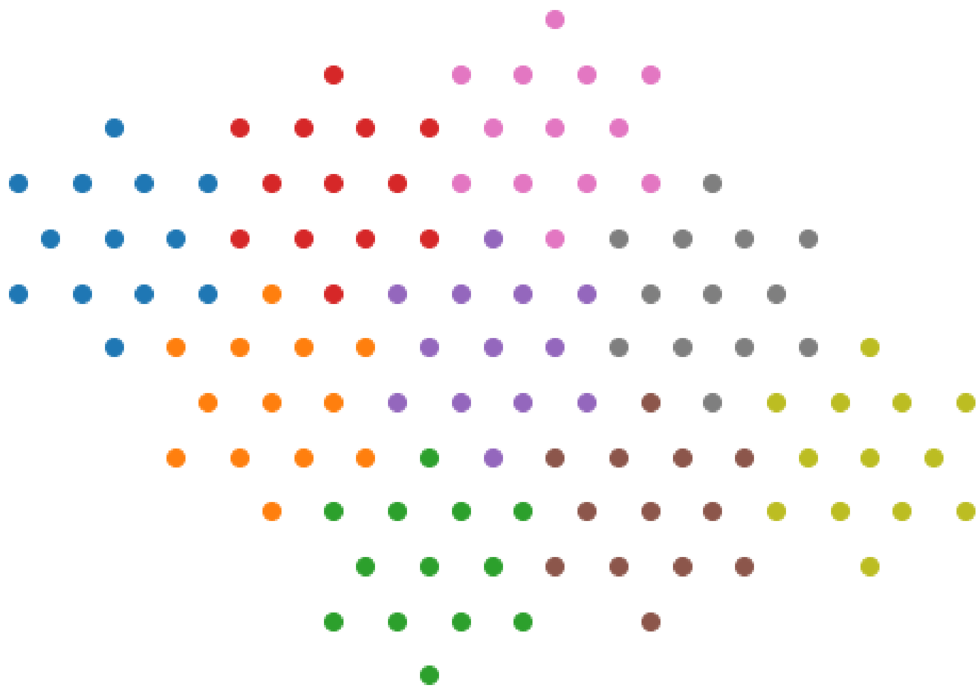
```
In [5]: show(*special_cluster("square_cross"), scope=1)
```



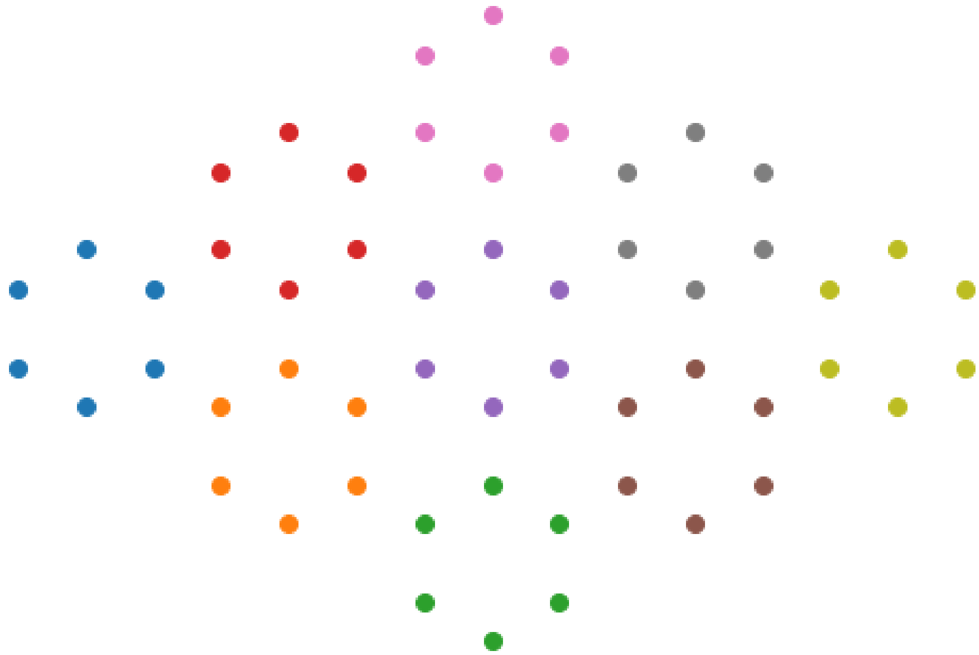
```
In [6]: show(*special_cluster("square_z"), scope=1)
```



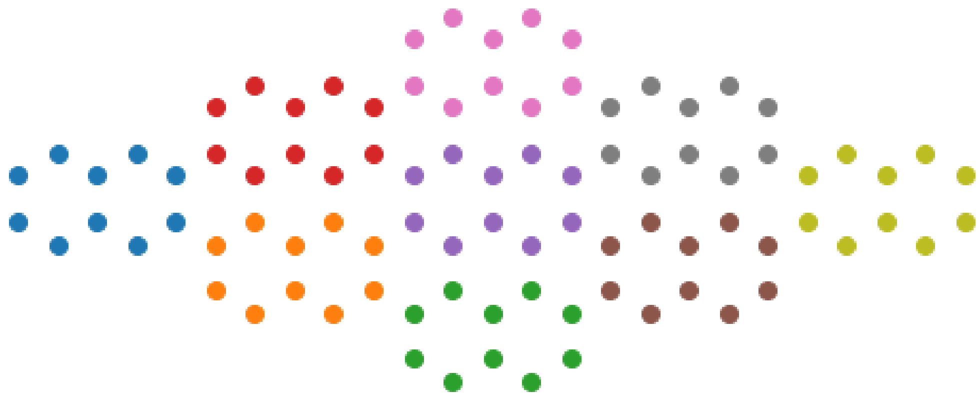
```
In [7]: show(*special_cluster("triangle_star"), scope=1)
```



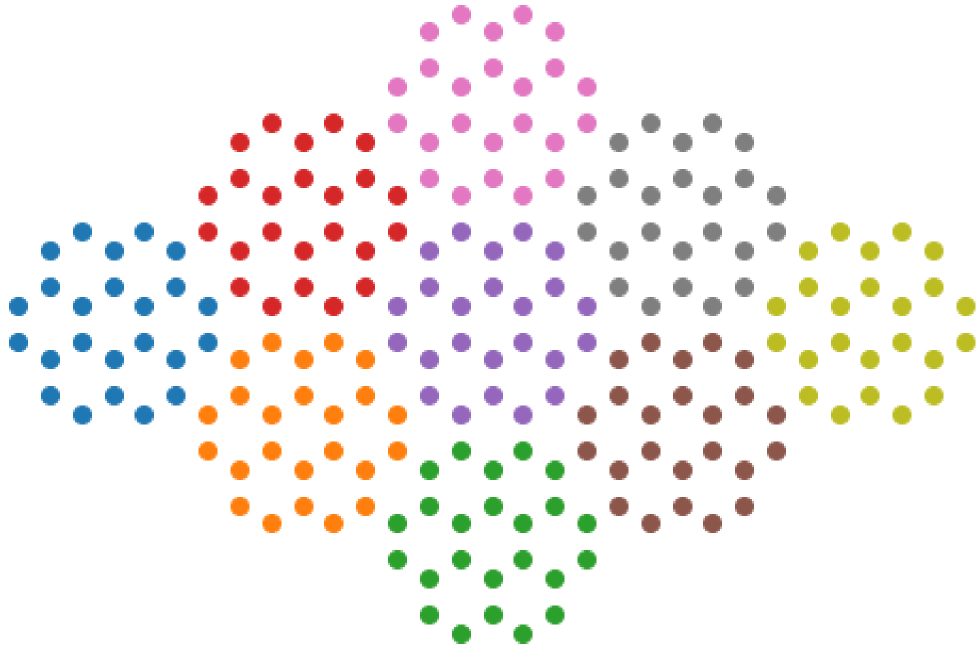
```
In [8]: show(*special_cluster("honeycomb_benzene"), scope=1)
```



```
In [9]: show(*special_cluster("honeycomb_diphenyl"), scope=1)
```

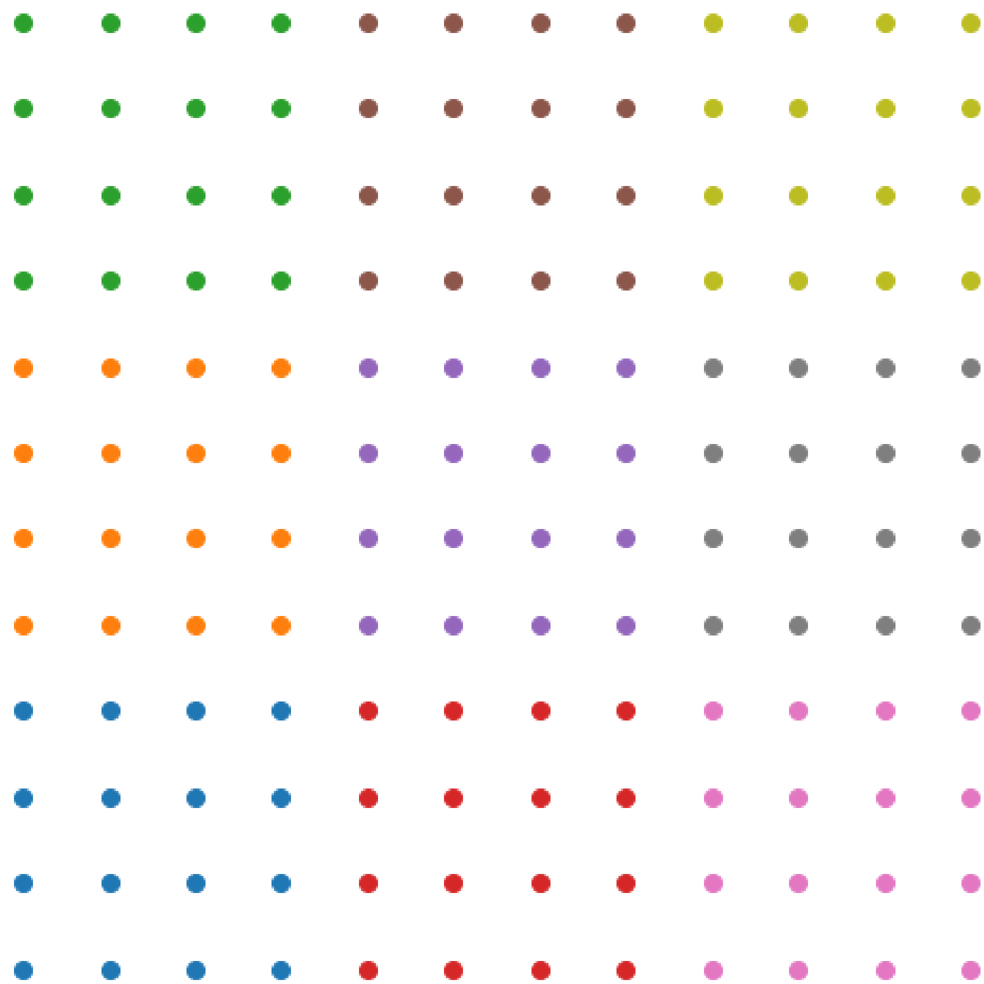


```
In [10]: show(*special_cluster("honeycomb_gear"), scope=1)
```

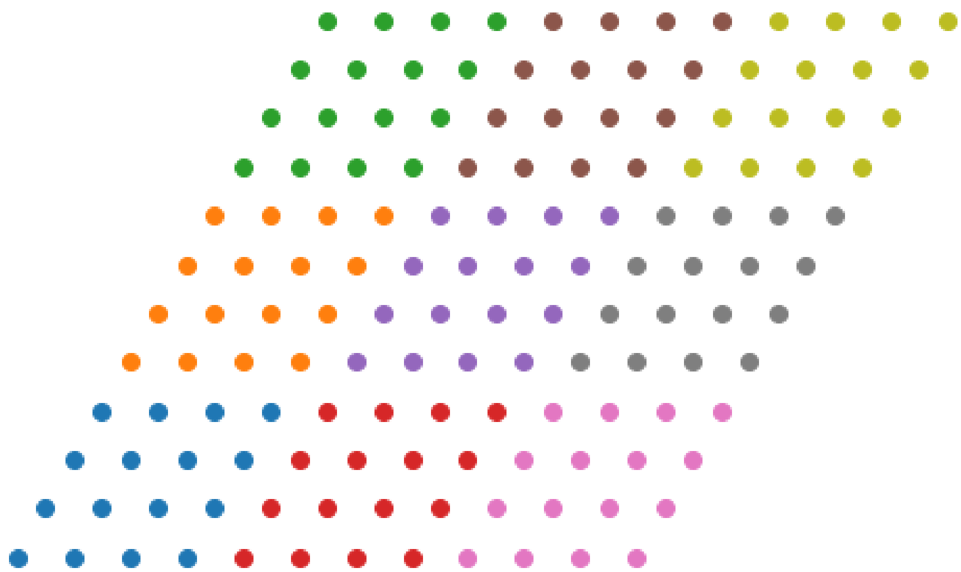


```
In [11]: num = 4
         show(*lattice_generator("chain", num0=num), scope=1)
```

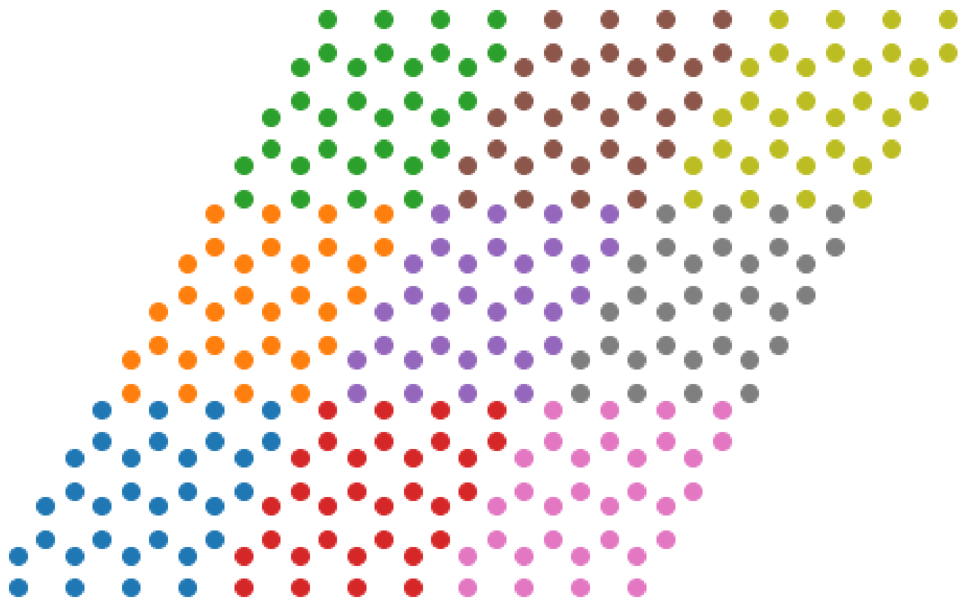
```
In [12]: show(*lattice_generator("square", num0=num, num1=num), scope=1)
```



```
In [13]: show(*lattice_generator("triangle", num0=num, num1=num), scope=1)
```



```
In [14]: show(*lattice_generator("honeycomb", num0=num, num1=num), scope=1)
```



```
In [15]: show(*lattice_generator("kagome", num0=num, num1=num), scope=1)
```