Base

Geometry

Class representing geometry of an atom arrangement.

Attributes:

Name	Type	Description
`sites`	`List[Tuple[float, float]]`	Atom site arrangement
`filling`	`List[int]`	Which sites are filled
`parallel_decoder`	`Optional[ParallelDecoder]`	Decoder object for decoding Geometry object

LocalTask

Bases: `Task`

`Task` to use for local executions for simulation purposes..

RemoteTask

Bases: `Task`

`Task` to use for remote executions to run the program on Quera Quantum Computers.

Report

```
Report(data, metas, geos, name='')
```

Report is a helper class for organizing and analysing data

ANALYZING RESULTS

When you've retrieved your results from either emulation or hardware you can generate a `.report()`:

```
report = results.report()
```

For the examples below we analyze the results of a two atom program.

The report contains useful information such as:

- The raw bitstrings measured per each execution of the program

  ```
  >>> report.bitstrings()
  [array([[1, 1],
          [1, 1],
          [1, 1],
          ...,
          [1, 1],
          [1, 1],
  ```

- The number of times each unique bitstring occurred:

  ```
  >>> report.counts()

  [OrderedDict([('11', 892), ('10', 59), ('01', 49)])]
  ```

- The Rydberg Density for each atom

  ```
  >>> report.rydberg_densities()

                   0      1
  task_number
  0             0.053  0.054
  ```

Source code in `src\bloqade\task\base.py`

```
def __init__(self, data, metas, geos, name="") -> None:
    self.dataframe = data  # df
    self._bitstrings = None  # bitstring cache
    self._counts = None  # counts cache
    self.metas = metas
    self.geos = geos
    self.name = name + " " + str(datetime.datetime.now())
```

markdown `property`

```
markdown
```

Get the markdown representation of the dataframe

bitstrings

```
bitstrings(filter_perfect_filling=True, clusters=[])
```

Get the bitstrings from the data.

Parameters:

Name	Type	Description	Default
`filter_perfect_filling`	`bool`	whether return will only contain perfect filling shots. Defaults to True.	`True`
`clusters`	`Union[tuple[int, int], List[tuple[int, int]]]`	(tuple[int, int], Sequence[Tuple[int, int]]): cluster index to filter shots from. If none are provided all clusters are used, defaults to [].	`[]`

Returns:

Name	Type	Description
`bitstrings`	`list of ndarray`	list corresponding to each task in the report. Each element is an ndarray of shape (nshots, nsites) where nshots is the number of shots for the task and nsites is the number of sites in the task. For example:

```
[array([[1, 1],
        [1, 1],
```

Name	Type	Description

		`[1, 1],`
		`...,`
		`[1, 1],`
		`[1, 1],`
		`[1, 0]], dtype=int8)]`

Note

Note that nshots may vary between tasks if filter_perfect_filling is set to True.

Source code in `src\bloqade\task\base.py`

~~~python
@beartype
def bitstrings(
    self,
    filter_perfect_filling: bool = True,
    clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
) -> List[NDArray]:
    """Get the bitstrings from the data.

    Args:
        filter_perfect_filling (bool): whether return will
            only contain perfect filling shots. Defaults to True.
        clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
            cluster index to filter shots from. If none are provided
            all clusters are used, defaults to [].

    Returns:
        bitstrings (list of ndarray): list corresponding to each
            task in the report. Each element is an ndarray of shape
            (nshots, nsites) where nshots is the number of shots for
            the task and nsites is the number of sites in the task.
            For example:
            ```python3
            [array([[1, 1],
                    [1, 1],
                    [1, 1],
                    ...,
                    [1, 1],
                    [1, 1],
                    [1, 0]], dtype=int8)]
            ~~~

    Note:
        Note that nshots may vary between tasks if filter_perfect_filling
        is set to True.

    """

    task_numbers =
self.dataframe.index.get_level_values("task_number").unique()

    bitstrings = []
    for task_number in task_numbers:
        mask = self._filter(
            task_number=task_number,
            filter_perfect_filling=filter_perfect_filling,
            clusters=clusters,
)
        if np.any(mask):
            bitstrings.append(self.dataframe.loc[mask].to_numpy())
        else:
            bitstrings.append(
                np.zeros((0, self.dataframe.shape[1]), dtype=np.uint8)
)
```
```

```
    return bitstrings
```

counts

```
counts(filter_perfect_filling=True, clusters=[])
```

Get the counts of unique bit strings.

Parameters:

Name	Type	Description	Default
`filter_perfect_filling`	`bool`	whether return will only contain perfect filling shots. Defaults to True.	`True`
`clusters`	`Union[tuple[int, int], List[tuple[int, int]]]`	(tuple[int, int], Sequence[Tuple[int, int]]): cluster index to filter shots from. If none are provided all clusters are used, defaults to [].	`[]`

Returns:

Name	Type	Description
`counts`	`list of OrderedDict[str, int]`	list corresponding to each task in the report. Each element is an ndarray of shape (nshots, nsites) where nshots is the number of shots for the task and nsites is the number of sites in the task. For example: `[OrderedDict([('11', 892), ('10', 59), ('01', 49)])]`

Note

Note that nshots may vary between tasks if filter_perfect_filling is set to True.

Source code in `src\bloqade\task\base.py`

~~~python
def counts(
    self,
    filter_perfect_filling: bool = True,
    clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
) -> List[OrderedDict[str, int]]:
    """Get the counts of unique bit strings.

    Args:
        filter_perfect_filling (bool): whether return will
            only contain perfect filling shots. Defaults to True.
        clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
            cluster index to filter shots from. If none are provided
            all clusters are used, defaults to [].

    Returns:
        counts (list of OrderedDict[str, int]): list corresponding to each
            task in the report. Each element is an ndarray of shape
            (nshots, nsites) where nshots is the number of shots for
            the task and nsites is the number of sites in the task.
            For example:
            ```python
                [OrderedDict([('11', 892), ('10', 59), ('01', 49)])]
            ~~~

    Note:
        Note that nshots may vary between tasks if filter_perfect_filling
        is set to True.

    """

    def _generate_counts(bitstring):
        output = np.unique(bitstring, axis=0, return_counts=True)

        count_list = [
            ("".join(map(str, bitstring)), int(count))
            for bitstring, count in zip(*output)
        ]
        count_list.sort(key=lambda x: x[1], reverse=True)
        count = OrderedDict(count_list)

        return count

    return list(
        map(_generate_counts, self.bitstrings(filter_perfect_filling,
clusters))
)
```

## list_param
```

```
list_param(field_name)
```

List the parameters associate with the given variable field_name for each tasks.

Parameters:

Name	Type	Description	Default
`field_name`	`str`	variable name	*required*

Source code in `src\bloqade\task\base.py`

```
def list_param(self, field_name: str) -> List[Union[Number, None]]:
    """
    List the parameters associate with the given variable field_name
    for each tasks.

    Args:
        field_name (str): variable name

    """

    def cast(x):
        if x is None:
            return None
        elif isinstance(x, (list, tuple, np.ndarray)):
            return list(map(cast, x))
        else:
            return float(x)

    return list(map(cast, (meta.get(field_name) for meta in self.metas)))
```

rydberg_densities

```
rydberg_densities(filter_perfect_filling=True, clusters=[])
```

Get rydberg density for each task.

Parameters:

Name	Type	Description	Default
`filter_perfect_filling`	`bool`	whether return will only contain perfect filling shots. Defaults to True.	`True`
`clusters`	`Union[tuple[int, int], List[tuple[int, int]]]`	(tuple[int, int], Sequence[Tuple[int, int]]): cluster index to filter shots from. If none are provided all clusters are used, defaults to [].	`[]`

Returns:

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td><code>rydberg_densities</code></td><td><code>Union[Series, DataFrame]</code></td><td>per-site rydberg density for each task as a pandas DataFrame or Series. For example:
<pre>
0   1
task_number
0   0.053 0.054
</pre></td></tr>
</table>

Source code in `src\bloqade\task\base.py`

~~~python
@beartype
def rydberg_densities(
    self,
    filter_perfect_filling: bool = True,
    clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
) -> Union[pd.Series, pd.DataFrame]:
    """Get rydberg density for each task.

    Args:
        filter_perfect_filling (bool, optional): whether return will
            only contain perfect filling shots. Defaults to True.
        clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
            cluster index to filter shots from. If none are provided
            all clusters are used, defaults to [].

    Returns:
        rydberg_densities (Union[pd.Series, pd.DataFrame]):
            per-site rydberg density for each task as a pandas DataFrame or Series.
            For example:
            ```python
            0      1
            task_number
            0            0.053  0.054
            ~~~
    """
    mask = self._filter(
        filter_perfect_filling=filter_perfect_filling, clusters=clusters
)
    df = self.dataframe[mask]
    return 1 - (df.groupby("task_number").mean())
```

show

```
show()
```

Interactive Visualization of the Report
```

Source code in `src\bloqade\task\base.py`

```
def show(self):
    """
    Interactive Visualization of the Report

    """
    display_report(self)
```