benji_socfb_investigate

March 6, 2023

[ ]:
```
import networkit as nk
```

[ ]:
```
import networkit as nk
import matplotlib.pyplot as plt
import numpy as np

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

graph_folder = 'data/socfb/'
path = graph_folder + 'socfb-Brown11.SpaceOne'
g = nk.readGraph(path, nk.Format.EdgeListSpaceOne)
```

[ ]:
```
path = graph_folder + 'socfb-UCLA.SpaceOne'
g2 = nk.readGraph(path, nk.Format.EdgeListSpaceOne)
```

[ ]:
```
nk.overview(g)
```

```
Network Properties:
nodes, edges                    8600, 384526
directed?                       False
weighted?                       False
isolated nodes                  0
self-loops                      0
density                         0.010399
clustering coefficient          0.217382
min/max/avg degree              1, 1075, 89.424651
degree assortativity            0.069552
number of connected components  8
size of largest component       8586 (99.84 %)
```

[ ]:
```
foo= nk.generators.ChungLuGenerator([1,2])
temp = foo.fit(g)
```

[ ]:
```
g2 = temp.generate()
```

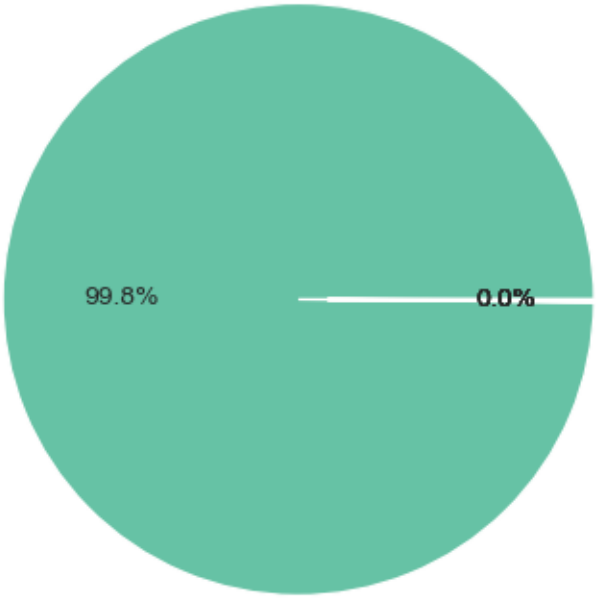
[ ]:
```
nk.overview(g2)
```

```
Network Properties:
nodes, edges                    8600, 383923
directed?                       False
weighted?                       False
isolated nodes                  92
self-loops                      0
density                         0.010383
clustering coefficient          0.032640
min/max/avg degree              0, 1088, 89.284419
degree assortativity            0.398573
number of connected components  93
size of largest component       8508 (98.93 %)
```

[ ]:
```
g.degree(40)
```

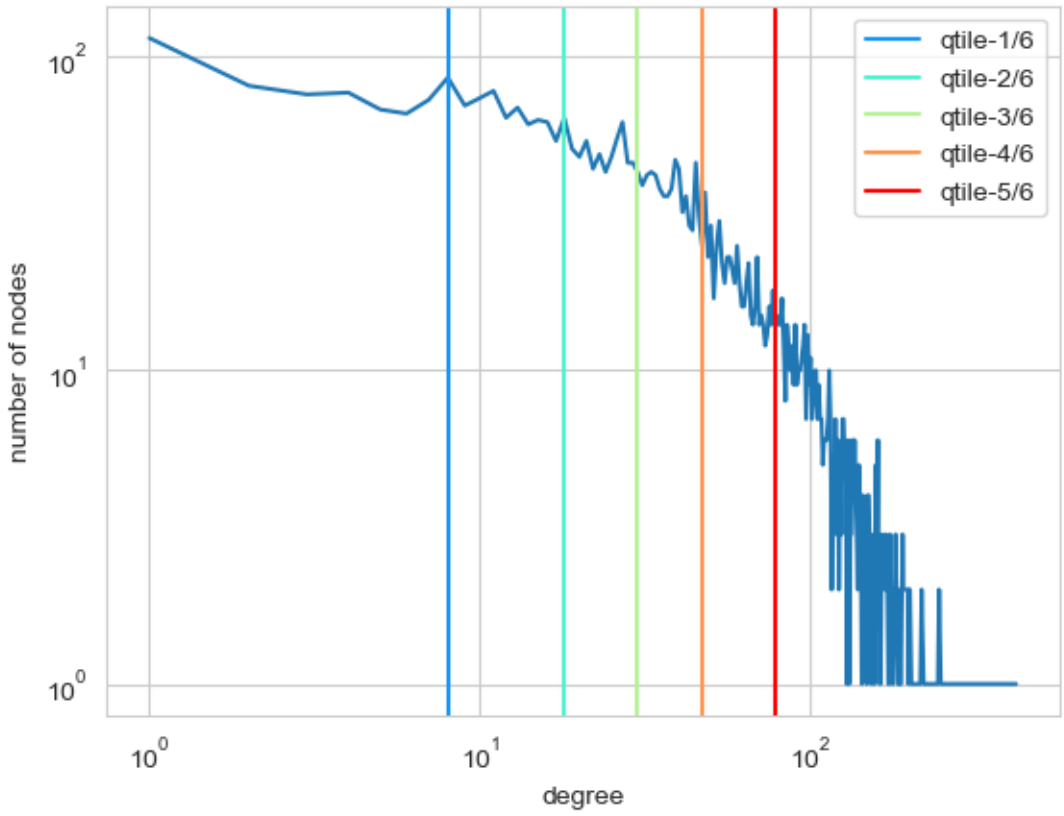
[ ]: 46

[ ]:
```
nk.plot.connectedComponentsSizes(g)
```



[ ]:
```
def plot_degree_dist(g, pl_fit=False, vlines=0):
    dd = sorted(nk.centrality.DegreeCentrality(g).run().scores(), reverse=True)
    degrees, numberOfNodes = np.unique(dd, return_counts=True)
    plt.xscale("log")
```

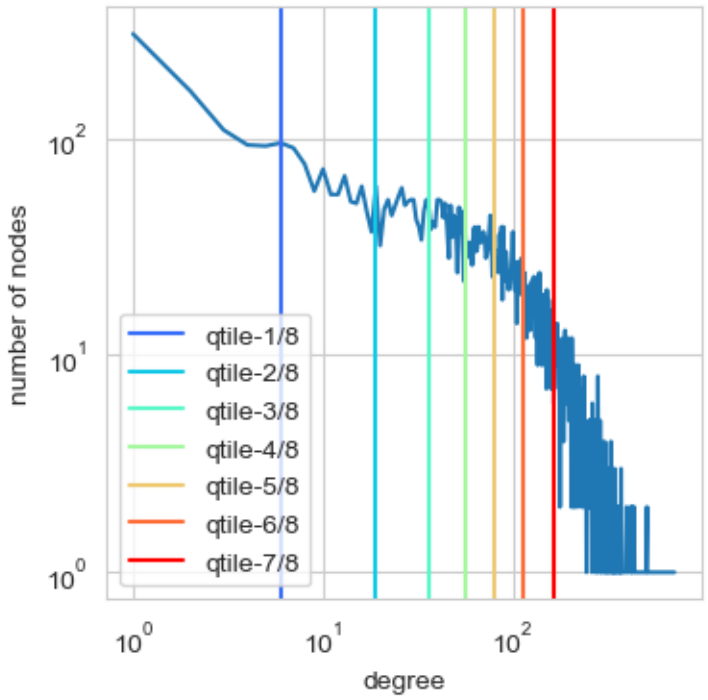
```
    plt.xlabel("degree")
    plt.yscale("log")
    plt.ylabel("number of nodes")
    # plt.scatter(degrees, numberOfNodes, s=1.1, marker='x')
    plt.plot(degrees, numberOfNodes)
    if pl_fit:
        fit = powerlaw.Fit(dd)
        xmin, xmax = fit.xmin, fit.xmax
        fit_bin_edges, fit_bin_proportions = fit.pdf()
        fit_bin_middles = [np.mean((fit_bin_edges[i], fit_bin_edges[i+1]))
                    for i in range(len(fit_bin_edges)-1)]
        plt.plot(fit_bin_middles, fit_pdf * len(fit.data), 'r')
        # plt.axvline(xmin, color='r')
    if vlines > 0:  # plot like quartile lines for number of nodes.
        # rough q-tiles
        q = vlines
        colors = plt.cm.rainbow(np.linspace(0, 1, q))
        rev_dd = list(reversed(dd))
        for i in range(1, q):
            plt.axvline(rev_dd[i * len(dd)//q], label=f'qtile-{i}/{q}',␣
↪c=colors[i])
        plt.legend()
    plt.show()
    # plt.figure()
    # plt.xscale("log")
    # plt.hist(dd, bins=20)

plot_degree_dist(g, vlines=6)
```



[ ]:
```
import glob
paths = np.random.choice(glob.glob(graph_folder + 'socfb-*'), 8)
for path in paths:
    g = nk.readGraph(path, nk.Format.EdgeListSpaceOne)
    print(path)
    print(f'nodes: {g.numberOfNodes()}, avg degree: {2*g.numberOfEdges()/g.
 ↪numberOfNodes():.4f}')
    # dd = sorted(nk.centrality.DegreeCentrality(g).run().scores(),␣
 ↪reverse=True)
    # degrees, numberOfNodes = np.unique(dd, return_counts=True)
    # fit = powerlaw.Fit(dd)
    # print(f'percent nodes that look powerlawy: {len(fit.data)/len(fit.
 ↪data_original):.4f}')
    plt.figure(figsize=(4, 4))
    plot_degree_dist(g, vlines=8)
```

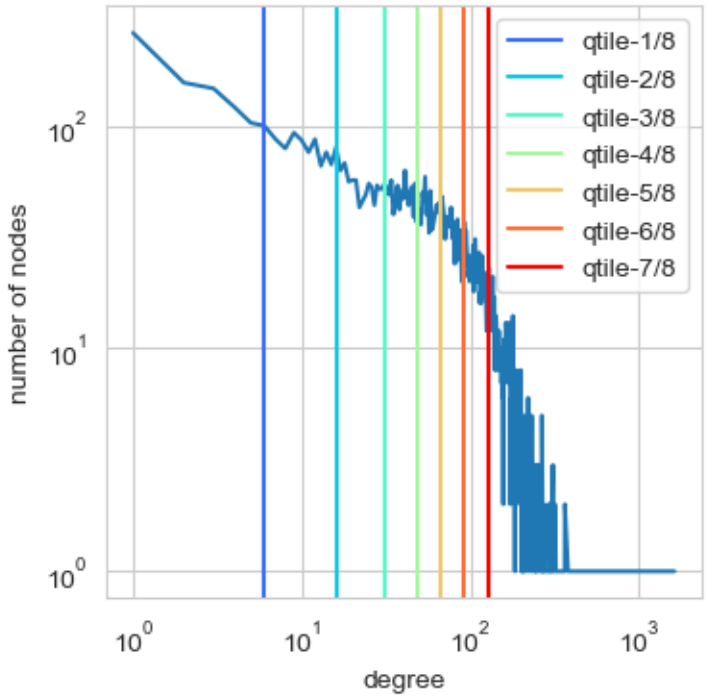
```
data/socfb/socfb-MIT8.SpaceOne
nodes: 6440, avg degree: 78.0286
```

[ ]:
```
<Figure size 400x400 with 0 Axes>
```



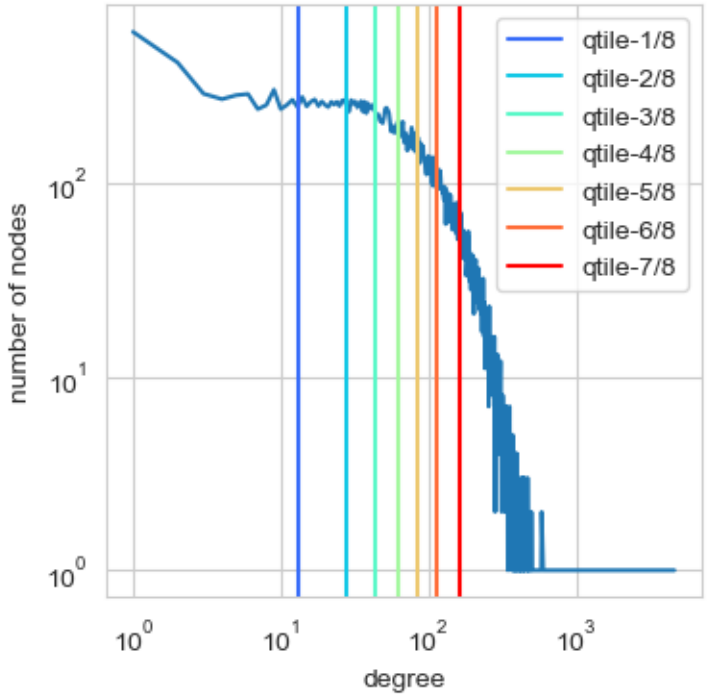

```
data/socfb/socfb-UChicago30.SpaceOne
nodes: 6591, avg degree: 63.1476
```

[ ]: <Figure size 400x400 with 0 Axes>



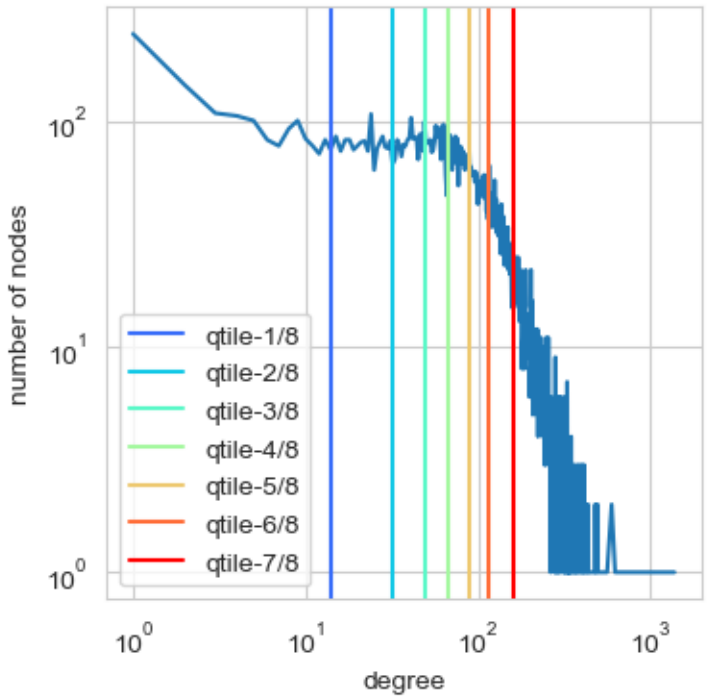
```
data/socfb/socfb-UIllinois20.SpaceOne
nodes: 30809, avg degree: 82.0817
```

[ ]: <Figure size 400x400 with 0 Axes>



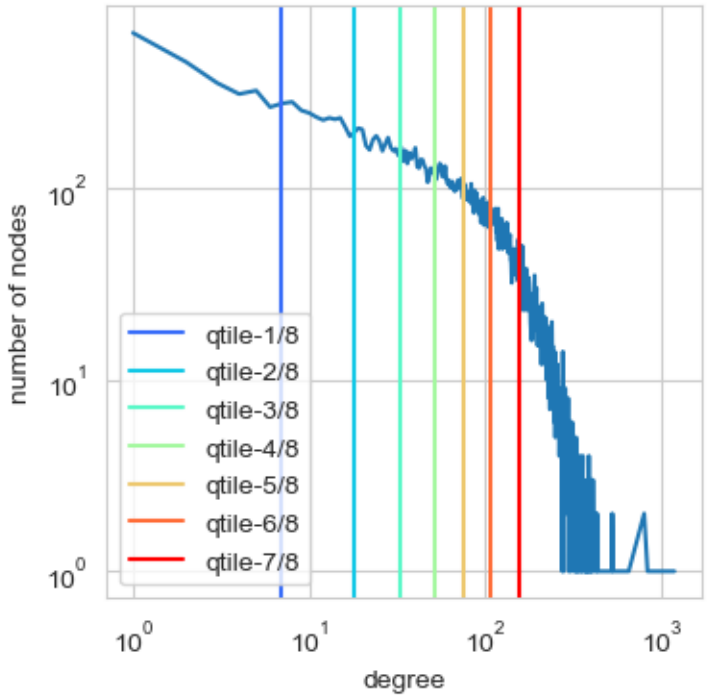

```
data/socfb/socfb-BC17.SpaceOne
nodes: 11509, avg degree: 84.6237
```

[ ]: <Figure size 400x400 with 0 Axes>



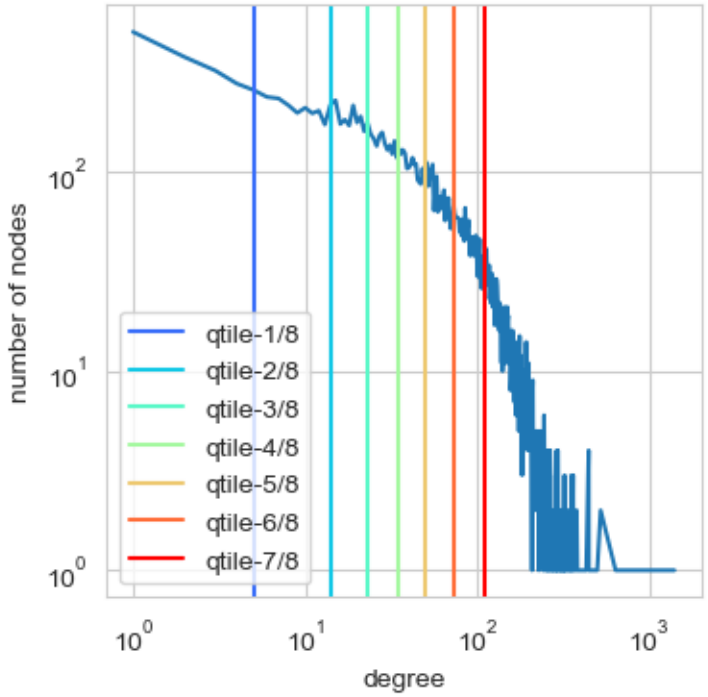

```
data/socfb/socfb-UCLA26.SpaceOne
nodes: 20467, avg degree: 73.0555
```

[ ]: <Figure size 400x400 with 0 Axes>



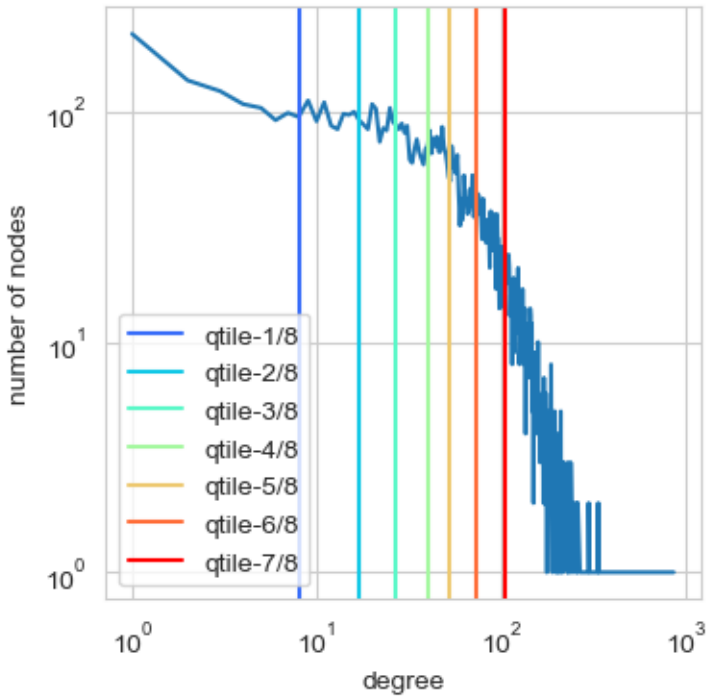
```
data/socfb/socfb-Temple83.SpaceOne
nodes: 13686, avg degree: 52.7247
```

[ ]: <Figure size 400x400 with 0 Axes>



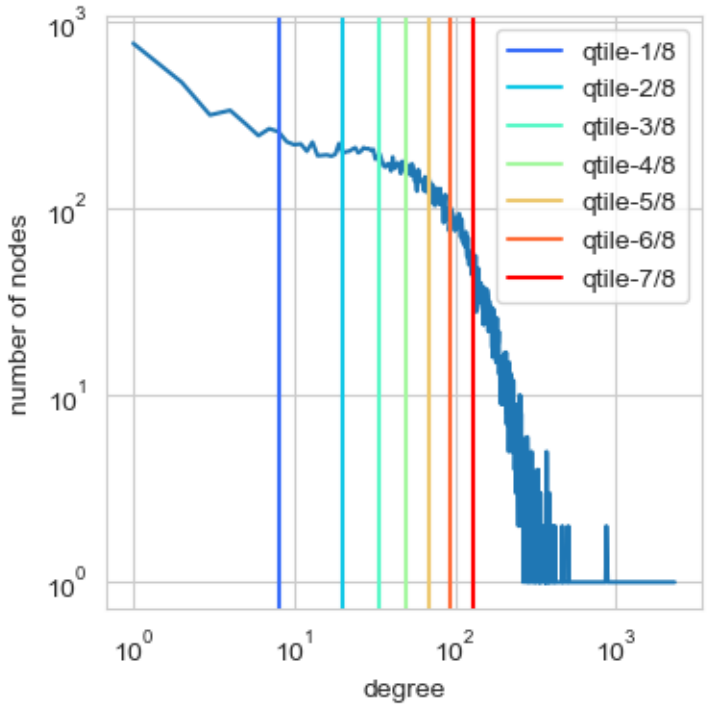

```
data/socfb/socfb-Vermont70.SpaceOne
nodes: 7324, avg degree: 52.2176
```

[ ]: <Figure size 400x400 with 0 Axes>




```
data/socfb/socfb-NYU9.SpaceOne
nodes: 21679, avg degree: 66.0284
```

[ ]: <Figure size 400x400 with 0 Axes>



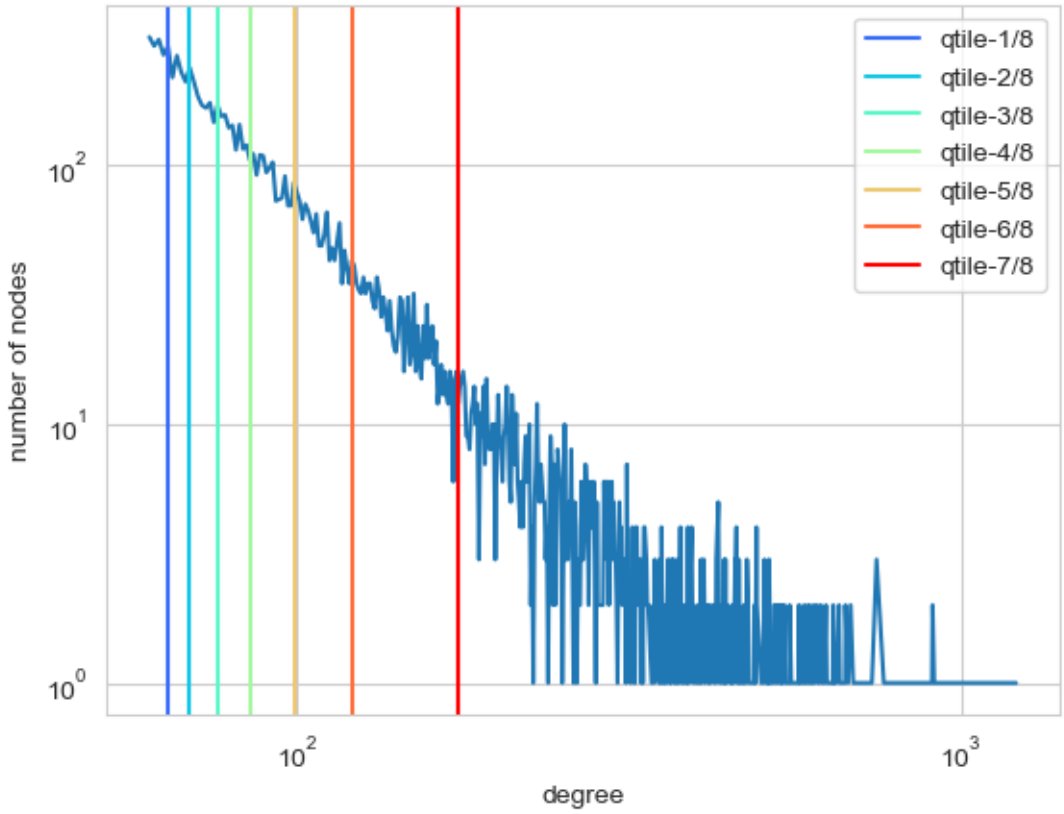
[ ]:
```
nk.overview(g)
```

```
Network Properties:
nodes, edges                    21679, 715715
directed?                       False
weighted?                       False
isolated nodes                  0
self-loops                      0
density                         0.003046
clustering coefficient          0.194039
min/max/avg degree              1, 2315, 66.028415
degree assortativity            0.011059
number of connected components  24
size of largest component       21623 (99.74 %)
```

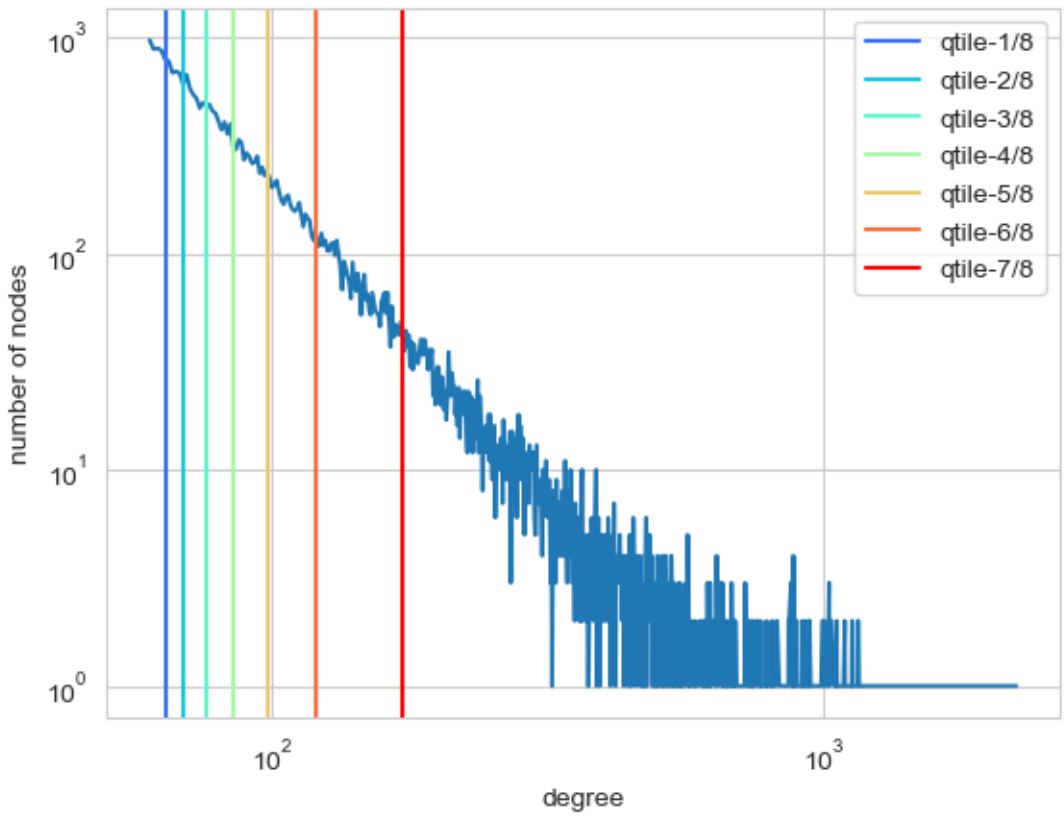
[ ]:
```
prefg = nk.generators.BarabasiAlbertGenerator(k=60, nMax=10000).generate()
nk.overview(prefg)
plot_degree_dist(prefg, vlines=8)
```

```
Network Properties:
nodes, edges                    10000, 596460
directed?                       False
weighted?                       False
isolated nodes                  0
```

```
self-loops                        0
density                           0.011930
clustering coefficient            0.039202
min/max/avg degree                60, 1210, 119.292000
degree assortativity              0.365811
number of connected components    1
size of largest component         10000 (100.00 %)
```



[ ]:
```
prefg = nk.generators.BarabasiAlbertGenerator(k=60, nMax=30000).generate()
plot_degree_dist(prefg, vlines=8)
```



[ ]:
```
dd = sorted(nk.centrality.DegreeCentrality(g).run().scores(), reverse=True)
import powerlaw
fit = powerlaw.Fit(dd)
fit.alpha
```

```
Calculating best minimal value for power law fit
xmin progress: 99%
```

[ ]:
```
4.882104969179315
```

[ ]:
```
fit_bin_edges, fit_bin_proportions = fit.pdf()
fit_bin_middles = [np.mean((fit_bin_edges[i], fit_bin_edges[i+1])) for i in␣
 ↪range(len(fit_bin_edges)-1)]
```

[ ]:
```
np.mean(123, 123)
```

```
---------------------------------------------------------------------------
AxisError                                 Traceback (most recent call last)
/Users/benjidayan/My Drive/eth_courses/GIRG/nemo-eva/benji_socfb_investigate.
 ↪ipynb Cell 12 in <cell line: 1>()
```

```
----> <a href='vscode-notebook-cell:/Users/benjidayan/My%20Drive/eth_courses/
 ↪GIRG/nemo-eva/benji_socfb_investigate.ipynb#X20sZmlsZQ%3D%3D?line=0'>1</a> np
 ↪mean(123, 123)

File <__array_function__ internals>:5, in mean(*args, **kwargs)

File /opt/homebrew/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric
 ↪py:3440, in mean(a, axis, dtype, out, keepdims, where)
   3345 @array_function_dispatch(_mean_dispatcher)
   3346 def mean(a, axis=None, dtype=None, out=None, keepdims=np._NoValue, *,
   3347          where=np._NoValue):
   3348     """
   3349     Compute the arithmetic mean along the specified axis.
   3350
   3351     Returns the average of the array elements.  The average is taken ov r
   3352     the flattened array by default, otherwise over the specified axis.
   3353     `float64` intermediate and return values are used for integer input .
   3354
   3355     Parameters
   3356     ----------
   3357     a : array_like
   3358         Array containing numbers whose mean is desired. If `a` is not a
   3359         array, a conversion is attempted.
   3360     axis : None or int or tuple of ints, optional
   3361         Axis or axes along which the means are computed. The default is
 ↪to
   3362         compute the mean of the flattened array.
   3363
   3364         .. versionadded:: 1.7.0
   3365
   3366         If this is a tuple of ints, a mean is performed over multiple␣
 ↪axes,
   3367         instead of a single axis or all the axes as before.
   3368     dtype : data-type, optional
   3369         Type to use in computing the mean.  For integer inputs, the␣
 ↪default
   3370         is `float64`; for floating point inputs, it is the same as the
   3371         input dtype.
   3372     out : ndarray, optional
   3373         Alternate output array in which to place the result.  The defau t
   3374         is ``None``; if provided, it must have the same shape as the
   3375         expected output, but the type will be cast if necessary.
   3376         See :ref:`ufuncs-output-type` for more details.
   3377
   3378     keepdims : bool, optional
   3379         If this is set to True, the axes which are reduced are left
   3380         in the result as dimensions with size one. With this option,
   3381         the result will broadcast correctly against the input array.
```

```
3382
3383         If the default value is passed, then `keepdims` will not be
3384         passed through to the `mean` method of sub-classes of
3385         `ndarray`, however any non-default value will be.  If the
3386         sub-class' method does not implement `keepdims` any
3387         exceptions will be raised.
3388
3389     where : array_like of bool, optional
3390         Elements to include in the mean. See `~numpy.ufunc.reduce` for␣
↪details.
3391
3392         .. versionadded:: 1.20.0
3393
3394     Returns
3395     -------
3396     m : ndarray, see dtype parameter above
3397         If `out=None`, returns a new array containing the mean values,
3398         otherwise a reference to the output array is returned.
3399
3400     See Also
3401     --------
3402     average : Weighted average
3403     std, var, nanmean, nanstd, nanvar
3404
3405     Notes
3406     -----
3407     The arithmetic mean is the sum of the elements along the axis divid d
3408     by the number of elements.
3409
3410     Note that for floating-point input, the mean is computed using the
3411     same precision the input has.  Depending on the input data, this ca
3412     cause the results to be inaccurate, especially for `float32` (see
3413     example below).  Specifying a higher-precision accumulator using th
3414     `dtype` keyword can alleviate this issue.
3415
3416     By default, `float16` results are computed using `float32`␣
↪intermediates
3417     for extra precision.
3418
3419     Examples
3420     --------
3421     >>> a = np.array([[1, 2], [3, 4]])
3422     >>> np.mean(a)
3423     2.5
3424     >>> np.mean(a, axis=0)
3425     array([2., 3.])
3426     >>> np.mean(a, axis=1)
3427     array([1.5, 3.5])
```

```
   3428
   3429     In single precision, `mean` can be inaccurate:
   3430
   3431     >>> a = np.zeros((2, 512*512), dtype=np.float32)
   3432     >>> a[0, :] = 1.0
   3433     >>> a[1, :] = 0.1
   3434     >>> np.mean(a)
   3435     0.54999924
   3436
   3437     Computing the mean in float64 is more accurate:
   3438
   3439     >>> np.mean(a, dtype=np.float64)
-> 3440     0.55000000074505806 # may vary
   3441
   3442     Specifying a where argument:
   3443
   3444     >>> a = np.array([[5, 9, 13], [14, 10, 12], [11, 15, 19]])
   3445     >>> np.mean(a)
   3446     12.0
   3447     >>> np.mean(a, where=[[True], [False], [False]])
   3448     9.0
   3449
   3450     """
   3451     kwargs = {}
   3452     if keepdims is not np._NoValue:

File /opt/homebrew/anaconda3/lib/python3.9/site-packages/numpy/core/_methods.py
 ↪167, in _mean(a, axis, dtype, out, keepdims, where)
    164 def _mean(a, axis=None, dtype=None, out=None, keepdims=False, *,␣
 ↪where=True):
    165     arr = asanyarray(a)
--> 167     is_float16_result = False
    169     rcount = _count_reduce_items(arr, axis, keepdims=keepdims,␣
 ↪where=where)
    170     if rcount == 0 if where is True else umr_any(rcount == 0, axis=None :

File /opt/homebrew/anaconda3/lib/python3.9/site-packages/numpy/core/_methods.py
 ↪76, in _count_reduce_items(arr, axis, keepdims, where)
     74     axis = (axis,)
     75 items = 1
---> 76 for ax in axis:
     77     items *= arr.shape[mu.normalize_axis_index(ax, arr.ndim)]
     78 items = nt.intp(items)

AxisError: axis 123 is out of bounds for array of dimension 0
```

[ ]:
```
fit_pdf
```

[ ]:
```
array([9.20451277e-03, 2.92894238e-03, 7.95891683e-04, 1.46666276e-04,
       5.66488789e-05, 2.62203382e-05])
```

[ ]:
```
help(nk.centrality.DegreeCentrality)
```

```
Help on class DegreeCentrality in module networkit.centrality:

class DegreeCentrality(Centrality)
 |  DegreeCentrality(G, normalized=False, outDeg=True, ignoreSelfLoops=True)
 |
 |  Node centrality index which ranks nodes by their degree.
 |  Optional normalization by maximum degree.  run() runs in O(n) time if
ignoreSelfLoops is false or the graph
 |  has no self-loops; otherwise it runs in O(m).
 |
 |  Constructs the DegreeCentrality class for the given Graph `G`. If the scores
should be normalized,
 |  then set `normalized` to True.
 |
 |  Parameters
 |  ----------
 |  G : networkit.Graph
 |          The input graph.
 |  normalized : bool, optional
 |          Normalize centrality values in the interval [0,1]. Default: False
 |  outdeg : bool, optional
 |          If set to true, computes the centrality based on out-degrees,
otherwise based on the in-degrees. Default: True
 |  ignoreSelfLoops : bool, optional
 |          If set to true, self loops will not be taken into account. Default:
True
 |
 |  Method resolution order:
 |      DegreeCentrality
 |      Centrality
 |      networkit.base.Algorithm
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__ = __reduce_cython__(…)
 |
 |  __setstate__ = __setstate_cython__(…)
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
```

```
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Centrality:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  centralization(…)
 |      centralization()
 |
 |      Compute the centralization of a network with respect to some centrality
measure.
 |      The centralization of any network is a measure of how central its most
central
 |      node is in relation to how central all the other nodes are.
 |      Centralization measures then (a) calculate the sum in differences
 |      in centrality between the most central node in a network and all other
nodes;
 |      and (b) divide this quantity by the theoretically largest such sum of
 |      differences in any network of the same size.
 |
 |      Returns
 |      -------
 |      float
 |              Centralization value.
 |
 |  maximum(…)
 |      maximum()
 |
 |      Return the maximum theoretical centrality score.
 |
 |      Returns
 |      -------
 |      float
 |              The maximum theoretical centrality score for the given graph.
 |
 |  ranking(…)
 |      ranking()
 |
 |      Returns the ranking of nodes according to their score.
 |
 |      Returns
 |      -------
 |      dict(tuple(int, float))
 |              A vector of pairs sorted into descending order. Each pair
contains a node and the corresponding score
```

```
|
|  score(…)
|      score(v)
|
|      Returns the score of node v for the centrality algorithm.
|
|      Parameters
|      ----------
|      v : int
|              Node index.
|
|      Returns
|      -------
|      float
|              The score of node v.
|
|  scores(…)
|      scores()
|
|      Returns the scores of all nodes for the centrality algorithm.
|
|      Returns
|      -------
|      list(float)
|              The list of all scores.
|
|  ------------------------------------------------------------------------
|  Methods inherited from networkit.base.Algorithm:
|
|  hasFinished(…)
|      hasFinished()
|
|      States whether an algorithm has already run.
|
|      Returns
|      -------
|      bool
|              True if Algorithm has finished.
|
|  run(…)
|      run()
|
|      Executes the algorithm.
|
|      Returns
|      -------
|      networkit.base.Algorithm
|              self
```

[ ]:
```
dir(g)
```

[ ]:
```
['__class__',
 '__copy__',
 '__deepcopy__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__ne__',
 '__new__',
 '__pyx_vtable__',
 '__reduce__',
 '__reduce_cython__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__setstate_cython__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'addEdge',
 'addNode',
 'addNodes',
 'attachNodeAttribute',
 'checkConsistency',
 'compactEdges',
 'degree',
 'degreeIn',
 'degreeOut',
 'detachNodeAttribute',
 'edgeId',
 'forEdges',
 'forEdgesOf',
```

```
 'forInEdgesOf',
 'forNodePairs',
 'forNodes',
 'forNodesInRandomOrder',
 'hasEdge',
 'hasEdgeIds',
 'hasNode',
 'increaseWeight',
 'indexEdges',
 'isDirected',
 'isIsolated',
 'isWeighted',
 'iterEdges',
 'iterEdgesWeights',
 'iterInNeighbors',
 'iterInNeighborsWeights',
 'iterNeighbors',
 'iterNeighborsWeights',
 'iterNodes',
 'numberOfEdges',
 'numberOfNodes',
 'numberOfSelfLoops',
 'removeAllEdges',
 'removeEdge',
 'removeMultiEdges',
 'removeNode',
 'removeSelfLoops',
 'restoreNode',
 'setWeight',
 'sortEdges',
 'swapEdge',
 'totalEdgeWeight',
 'upperEdgeIdBound',
 'upperNodeIdBound',
 'weight',
 'weightedDegree',
 'weightedDegreeIn']
```

[ ]: